Einführung in die Graphische Datenverarbeitung

Liebe Fernstudentin, lieber Fernstudent,

wir begrüßen Sie zum Kurs “Einführung in die Graphische Datenverarbeitung”. Sicher sind Sie schon durch das Fernsehen, Ihren persönlichen Rechner oder durch Fachzeitschriften mit der Graphischen Datenverarbeitung in Berührung gekommen und haben sich gefragt, wie die photorealistischen Bilder erzeugt werden oder wie die Interaktion über Maus oder Tablett am Bildschirm realisiert wird. Der vorliegende Kurs soll Sie in diese “Geheimnisse” einführen.
Auch bei diesem Kurs bitten wir Sie um Ihre Mitarbeit in Form von Fehlerhinweisen und sachlicher Kritik, damit spätere Semester von einer verbesserten Fassung profitieren können. Wir wünschen Ihnen viel Freude beim Studieren dieses Kurses und einen erfolgreichen Abschluß.

Einordnung des Kurses in den Studienplan

Der Kurs “Einführung in die Graphische Datenverarbeitung” ist insbesondere für Studenten gedacht, die im Bachelor-Studiengang die Vertiefungsrichtung ”Graphik und Algorithmen” gewählt haben. Er entstand im wesentlichen aus dem Kurs “Graphische Datenverarbeitung I” und einer Einführung in Themen des Kurses “Graphische Datenverarbeitung II” des Diplomstudiengangs. Weitere Gebiete der Graphischen Datenverarbeitung werden im Kurs “Graphische Datenverarbeitung II” behandelt.

Der Autor

Wolfgang Straßer

- Studium der Nachrichtentechnik an der TU Berlin (1962–1969)
- Entwicklungsingenieur bei Nixdorf (1969–1971)
- Wissenschaftlicher Assistent an der TU Berlin (1971–1973)
- Promotion zum Dr.– Ing. (1974)
- Heinrich–Hertz–Institut Berlin (1973–1978)
- Professor für Informatik an der TH Darmstadt (1978–1986)
- Professor für Informatik an der Universität Tübingen, Fachgebiet Graphisch–Interaktive Systeme (seit Oktober 1986)

Inhaltsverzeichnis

Lehrziele

Nach dem Durcharbeiten sollten Sie verstehen

- wie die wichtigsten graphischen Peripheriegeräte funktionieren und im Prinzip aufgebaut sind,
- welche Komponenten zu einem Rastersichtgerät gehören,
- wie die Komponenten eines Rastersichtgerätes zusammenarbeiten und wo die leistungsbestimmenden Stellen sind,
- was Herstellerangaben über die Leistungsdaten der Displays bedeuten,
- welche Entwicklung in naher Zukunft aufgrund der fortschreitenden Halbleitertechnik zu erwarten ist,
- wie bei vorgegebener Anwendung ein hinreichendes System zu konfigurieren ist,
- wie die wichtigsten Algorithmen zum Zeichnen von Geraden, Kreisen, gefüllten Dreiecken usw. auf Rasterbildschirmen arbeiten.

Kapitel 1

Einführung

Die Graphische Datenverarbeitung (GDV) hat in der Informationstechnologie einen sehr hohen Stellenwert errungen. Mitverantwortlich dafür sind ihre spektakulären Merkmale, die sie von anderen Disziplinen der Informatik abheben. Sie bietet Verfahren und Techniken, um Bilder zu produzieren und zu manipulieren — wobei die Bilder eine besondere Art der Ausgabe von rechnerinternen Darstellungen sind.

Die Bedeutung des Kommunikationsmediums *Bild* ist seit Jahren bekannt. Die vormals noch hohen Kosten für graphische Geräte und außergewöhnliche Rechnerbelastungen haben die Verbreitung graphischer Anwendungssysteme verzögert. Erst das fallende Preis–/Leistungsverhältnis für die Hardware und die Verwendung standardisierter Software ermöglichen einen wirtschaftlich vertretbaren Einsatz der Computer-Graphik. Das Gebiet der Graphischen Datenverarbeitung wird in der Gerätetechnik insbesondere von den USA dominiert. Im Text werden deshalb gebräuchliche englische Fachausdrücke verwendet.

1.1 Klassifizierung der Graphischen Datenverarbeitung

AUSGABE ——— EINGABE	Bild	Beschreibung
Bild	Bildverarbeitung	Bildanalyse
Beschreibung	generative Computer–Graphik	“andere DV–Disziplinen”

Abbildung 1.1: Klassifizierung nach Rosenfeld

Die ISO–Definition beschreibt die Graphische Datenverarbeitung (computer graphics) als Methode zur Datenkonvertierung zwischen Rechner und graphischen Endgeräten: “ Methods and techniques for converting data to and from graphics

displays via computer ” [ISO82]. Anhand der in dieser Begriffserklärung enthaltenen Komponenten *Daten, Sichtgerät* und *Rechner* wurde die Graphische Datenverarbeitung bereits 10 Jahre zuvor von Rosenfeld [NR72] untergliedert (Bild 1.1), und zwar in die Teilgebiete *generative Computergraphik* (interactive computer graphics), *Bildverarbeitung* (image processing) und *Bildanalyse* (picture analysis).

1.1.1 Die generative Computer–Graphik

Die generative Computer–Graphik behandelt künstlich erzeugte Bilder, die in Form von Bildbeschreibungen (Datenstrukturen) vorliegen und durch den Rechner (d.h. vom Programm) erzeugt werden. Die Daten selbst können

- vom Benutzer eingegeben sein,
- vom Rechner erzeugt bzw. erfaßt worden sein,
- aus Kommandos bzw. Aktionen des Benutzers an einem graphischen Arbeitsplatz resultieren.

Die künstlich erzeugten Bilder können durch Untergliederung in Segmente strukturiert werden. Die einzelnen Objekte (Segmente) können transformiert werden, d.h. vergrößert, verkleinert, auf der Ausgabefläche gedreht und verschoben werden, unsichtbar geschaltet sein bzw. blinkend dargestellt werden. Zur Vervollständigung der Mensch–Maschine–Schnittstelle sind in der generativen Computer–Graphik Methoden und Techniken enthalten, die graphische Eingaben (z.B. Positionseingabe, Objektidentifikation) ermöglichen. In diesem Punkt setzt ein weiteres Kriterium für die Untergliederung des Bereiches Computer–Graphik an: man unterscheidet prinzipiell nach passiven und interaktiven Systemen, wobei passive Systeme oft als Untermenge der interaktiven Systeme angesehen werden.

1.1.1.1 Passive Systeme

Die passive generative Computer-Graphik konzentriert sich auf Probleme, die die Ausgabe graphischer Daten stellt. Sie umfaßt Ausgabegeräte sowie Ausgabe– und Darstellungstechniken, die eine permanente Ausgabe auf Papier oder Film ermöglichen (z.B. off-line–Plotsysteme). Nach einmaliger Bilderstellung ist keine Abänderung des Bildes mehr möglich. Dies kann nur durch eine vorherige Umdefinition der Daten und anschließende, erneute vollständige Bilderzeugung erfolgen (Bild 1.2).

1.1.1.2 Interaktive graphische Systeme

Als ein interaktives graphisches System bezeichnet man die aus einer Gerätekonfiguration und Programmsystemen bestehenden Systeme, die sich zum einen zur

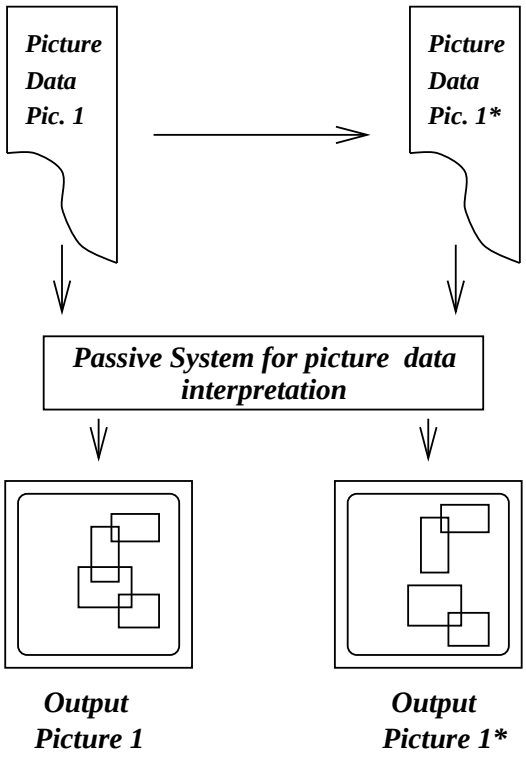


Abbildung 1.2: Bildmodifikation bei passiven Systemen

Erzeugung und zur Darstellung graphischer Ausgabe (ähnlich dem passiven System) eignen und darüber hinaus über Hardware– und Softwarekomponenten verfügen, die eine interaktive Arbeitsweise unterstützen. Durch interaktive graphische Systeme wird im Gegensatz zu der passiven Graphik dem Benutzer die Möglichkeit eingeräumt, Bilder bzw. Bildobjekte dynamisch zu beeinflussen (Bild 1.3). Bei solchen Anwendungen stehen Mensch und Maschine in einem Regelkreis, der oft mehrfach durchlaufen wird, bis die gewünschten Resultate erreicht sind.

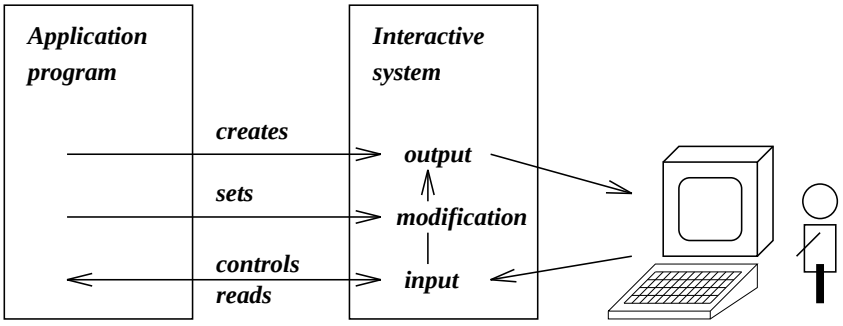


Abbildung 1.3: Interaktive graphische Systeme

Die interaktive Graphische Datenverarbeitung umfaßt

– Verfahren zur Beschreibung und Speicherung graphischer Daten,

– den Transfer von Daten, insbesondere die Eingabe und die Ausgabe graphischer Daten,

– die Verarbeitung der Daten, insbesondere aber die Synthese graphischer Daten,

– den Entwurf und die Implementierung interaktiver graphischer Systeme und

– den Entwurf und die Integration graphischer Datentypen in Programmierumgebungen.

1.1.2 Die Bildverarbeitung

Die Bildverarbeitung stellt Methoden und Techniken zur Verfügung, die Darstellung eines Bildes so zu verändern, daß z.B. die menschliche Wahrnehmung den Informationsgehalt eines Bildes leichter erkennt. Bilder existieren hier a priori und werden durch Digitalisierung von Photographien, TV-Bildern oder Satellitenbildern gewonnen. Das Ausgabemedium der Bildverarbeitung bestand zunächst aus Fernsehmonitoren, später aus höher auflösenden Rasterbildschirmen. Durch die Weiterentwicklung der Rastertechnologie und die Preisreduktionen bei Halbleiterspeichern verstärkte sich der Trend, Rasterdisplays auch für generative Computer-Graphik einzusetzen. Eine stärkere Annäherung der Gebiete Bildverarbeitung, generative Graphik und Bildanalyse erfolgte über dieses Darstellungsmedium.

Die Verfahren der Bildverarbeitung arbeiten auf unstrukturierten Bildern. Dies ist gleichzeitig das wesentliche Unterscheidungsmerkmal zwischen Graphiksystemen und Bildverarbeitungssystemen (Bild 1.4). Die verwalteten Daten sind in

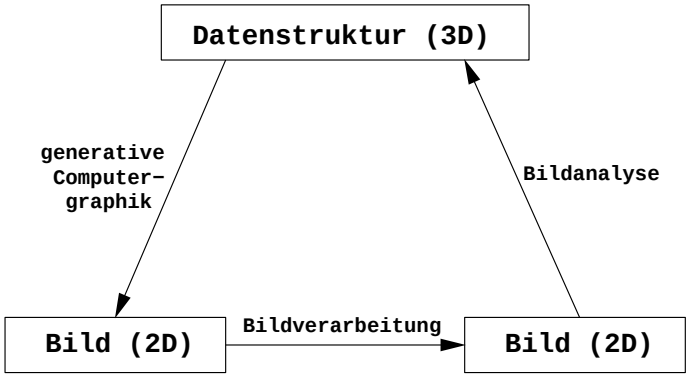


Abbildung 1.4: Zusammenhang zwischen den Teilgebieten der GDV

der generativen Graphik eine strukturierte Menge von Ausgabeelementen. In der Bildverarbeitung wird ein Bild als eine Menge von $n \times m$ Bildpunkten (Pixel) angesehen, wobei jedem Bildpunkt ein Grauwert bzw. eine Farbe zugeordnet ist. Ein Bild besitzt ansonsten keine Struktur. Die Bildverarbeitung wird aus diesem Grund auch oft als die Anwendung von Algorithmen auf Zahlenfelder betrachtet. Die Hauptaufgaben der Bildverarbeitung sind:

– **Bildverbesserung**, z.B. durch Kontrastvergrößerung oder Unterdrückung von Hintergrundstörungen,

– **Bildauswertung**, d.h. Größenbestimmung, Konturerfassung, Erkennen typischer Bildeigenschaften,

– **Zeichenerkennung**, d.h. Herausziehen und Klassifizieren von Bildeigenschaften.

1.1.3 Die Bildanalyse

Die Bildanalyse beschäftigt sich mit der Zerlegung eines Bildes in *Urbilder*, d.h. in bekannte graphische Objekte (wie Dreiecke, Kreise etc.), so daß bekannte Datenstrukturen aufgebaut werden können. Durch anschließenden Vergleich mit bekannten Zerlegungen können Objekte in Bildern erkannt werden.

Die Bildanalyse nutzt dabei Methoden der Bildverarbeitung zur Zerlegung sowie Techniken der generativen Computer–Graphik zur Komposition der Urbilder. Diese Wechselwirkung verdeutlicht das Strukturbild in Bild 1.5.

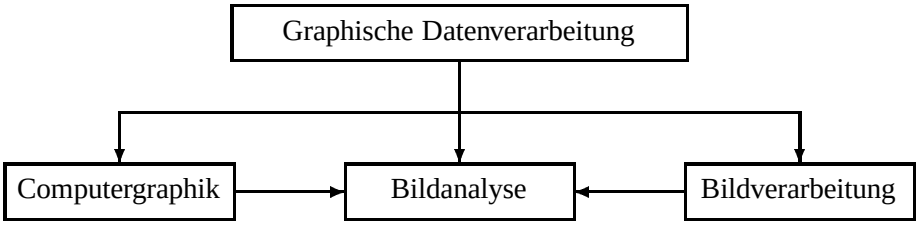


Abbildung 1.5: Wechselwirkung zwischen den Teilgebieten der GDV

1.2 Die Entwicklung der Graphischen Datenverarbeitung

Die Entwicklung der Graphischen Datenverarbeitung kann sehr gut anhand ihrer Meilensteine aufgezeigt werden. Dabei ist klar zu erkennen, daß die Gerätetechnik hauptverantwortlich für den Bau und die Verbreitung graphischer Systeme ist. Seit Beginn der Informationstechnologie wurden Zeichnungen (Plots) mit Druckern erstellt. Um 1950 wurde am MIT [Massachusetts Institute of Technology] zum ersten Mal eine *Kathodenstrahlröhre* (CRT) ausschließlich für die Bildausgabe unter Computersteuerung (Whirlwind) eingesetzt [Eve52]. In den folgenden Jahren wurden keine nennenswerten Neuerungen in der Computer–Graphik erzielt, da die damaligen Rechner eher auf das “number crunching” als auf interaktive Anwendungen ausgerichtet waren. Ende der fünfziger Jahre setzte man CRTs in Verbindung mit Lichtgriffeln (lightpens) ein, um Punkte auf dem Bildschirm anzusprechen. Die interaktive Computer–Graphik beginnt eigentlich erst mit den

Arbeiten von Sutherland [Sut63], die das *Sketchpad–System* beschreiben. In diesem System wurden erstmals viele heute noch übliche Methoden und Techniken zusammengefaßt und demonstriert. Datenstrukturen für Bildhierarchien oder die Bildkomposition aus graphischen Standardelementen sind heute ebenso gebräuchlich wie Interaktionstechniken für Tastatur und Lichtgriffel zur Benutzung von Menüs, die hier erstmals realisiert wurden.

Anforderungen aus den technischen Entwurfs– und Konstruktionsbereichen der Automobil– und der Flugzeugindustrie (General Motors, Lockhead) führten Mitte der 60er Jahre zu verstärkten Forschungsanstrengungen, aus denen heraus die ersten kommerziellen CAD-/CAM-Systeme der 70er Jahre entstanden. Diese Systeme benötigten teure graphische Geräte und bedingten infolge der großen Datenbestände und interaktiven Bildmanipulationstechniken hohe Rechnerbelastungen. Zudem war die Erstellung und der Einsatz großer interaktiver Programmsysteme unüblich für “Fortran-Batch”–Programmierer und ihr Einsatz dadurch erschwert. Hinzu kam ferner, daß graphische Software infolge der jeweils unterschiedlichen Geräte *nicht portabel* war und daß bei dem Betrieb neuer Geräte die Anwendungssoftware jeweils in Teilen neu erstellt werden mußte. Deshalb existieren nur wenige Installationen.

Die Programmierung von Bildern erfolgte mit graphischen Unterprogrammpaketen, die zunächst als firmeninterne “Standards” (Plot10, CalComp) eingesetzt waren, bevor sie einen breiten Anwenderkreis eroberten [Sch78]. Diese Pakete boten dem Programmierer funktional eine einheitliche Schnittstelle, waren aber in sehr hohem Maße geräteabhängig. Geräteunabhängigkeit zeichnet die Softwareentwicklung und Standardisierung der 80er Jahre aus [Gra79], [DIN82], [ISO84].

Die Hardwareentwicklungen gingen in Richtung hochauflösender, billiger *Rastergeräte* (z.B. 1280×1024 Pixel), die über eine Vielzahl von Farbstufungen sowie über höhere graphische Funktionen (Transformationen (Kap. 3), Clipping (Kap. 5)) verfügen. Im Bereich der graphischen Eingabegeräte verliert der Lichtgriffel zunehmend an Bedeutung. Er wird verstärkt durch kostengünstigere und ergonomischere Geräte (z.B. Maus, Tablett) ersetzt.

1.3 Typische Anwendungen der Graphischen Datenverarbeitung

In weiten Bereichen der Industrie, des Managements, in öffentlichen Bereichen wie Verwaltung, Lehre und sogar zu Unterhaltungszwecken werden die Techniken der Graphischen Datenverarbeitung eingesetzt. Die Anzahl der Anwendungen und der Anwendungsklassen steigt ständig, nicht zuletzt, weil sich graphische Ausgabegeräte zu preiswerten Massenprodukten entwickeln. Im folgenden werden beispielhaft einige typische Einsatzgebiete vorgestellt.

1.3.1 Plotsysteme zur Zeichnungserstellung

Im Geschäftsbereich, in Wissenschaft und Technik werden 2*D*– und 3*D*–Graphiksysteme verwendet, um z.B. mathematische, physikalische oder wirtschaftliche Funktionen darzustellen (z.B. Bild 1.6). Die Ergebnisse werden in Form von Histogrammen, Balkendiagrammen, Funktionengebirgen, Kuchendiagrammen als Ablaufpläne etc. vorgestellt und dadurch in einer klar strukturierten und übersichtlichen Weise präsentiert. Die Computer–Graphik wird in diesen Bereichen benutzt, um komplexe Systeme und deren Zusammenhänge zu veranschaulichen und eine Entscheidungsfindung zu unterstützen.

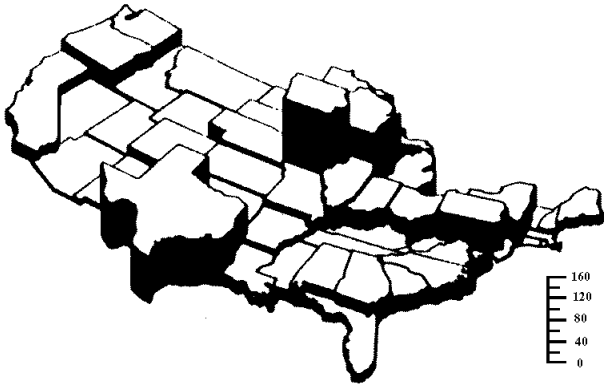


Abbildung 1.6: Präsentation digital ermittelter Ergebnisse

1.3.2 Kartographie

Graphiksysteme werden zur Erstellung sehr genauer Darstellungen von geologischen und geographischen Daten eingesetzt. Die Ausgabe erfolgt auf Papier bzw. Filmen. Als Vorteil wird gesehen, daß aus einer einzigen Datenstruktur mehrfach Zeichnungen erstellt werden können und die graphische Grundstruktur entsprechend unterschiedlicher, “nichtgraphischer” Größen zu unterschiedlichen Darstellungen (z.B. in verschiedenen Farben) führen kann. Beispiele dafür sind Landkarten, Liegenschaftskarten, Reliefkarten, Wetterkarten etc. (Bild 1.7 zeigt einen Ausschnitt aus einer Liegenschaftskarte).

1.3.3 Computerunterstützter Entwurf

Im *Computer Aided Drafting and Design*–Bereich werden interaktive graphische Systeme eingesetzt, um mechanische oder elektrische Geräte zu entwerfen. In CAD-Systemen, die als Anwendung eines graphischen Systems zu betrachten sind, werden Strukturen aufgebaut und verwaltet, wie z.B. Häuser, Automobile, Produktionsanlagen. CAD-Systeme wurden zunächst eingesetzt, um Entwürfe zu erstellen und zu zeichnen. In zunehmendem Maße werden Methoden in CAD-Systeme eingebracht, die eine Bewertung der Entwürfe ermöglichen, d.h.

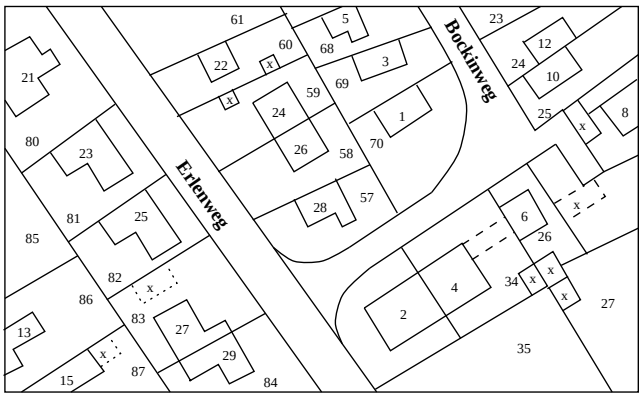


Abbildung 1.7: Kartographie, Liegenschaftskarte

ihre Korrektheit kann per Simulations- und Testverfahren bereits während der Entwurfsphase überprüft werden. Die Testabläufe und -ergebnisse lassen sich ebenfalls graphisch darstellen. Aus dem Entwurf lassen sich direkt Stücklisten usw. erstellen.

1.3.4 Prozeßüberwachung

In der Prozeßsteuerung erfassen Sensoren eine große Anzahl von Meßwerten, die einer Überwachungszentrale zugestellt werden. Dort werden sie in die Darstellung des aktuellen Prozeßablaufs eingebracht und über Kontrollbildschirme ausgegeben. Kritische Werte bzw. Situationen werden dem Bedienpersonal durch besondere optische Zeichen (Farben, Blinken etc.) angezeigt. Technische Prozesse aus Kraftwerken und chemischen Fabriken, dem Straßenverkehr (Ampelsteuerung, Fahrpläne) und militärischen Einsatzbereichen u.v.m. werden auf diese Weise kontrolliert und gesteuert.

1.3.5 Simulation, Computer–Animation und Virtuelle Realität

Prozesse mit einem schnellen Zeitverlauf wie z.B. Deformierungen, chemische Reaktionen etc. lassen sich simulieren und in einer Folge von berechneten Bildern auf einem Film ausgeben. Der zeitliche Ablauf einer Deformierung kann dann durch Aufführung des Filmes gezeigt werden. Mit Supercomputern können solche Darstellungen bereits schon in Echtzeit gewonnen werden.

Flugsimulatoren sind ein Beispiel für den Einsatz interaktiver graphischer Systeme, in denen ein Bild erzeugt und in Abhängigkeit von den Bedienerreaktionen ein “neues”, d.h. ein sich anschließendes Bild in Echtzeit erzeugt wird.

Die Computer–Animation ist die moderne Variante des Zeichentrickfilms. In ihr fließt das technische Wissen über photorealistische Bildsynthese, Beschreibung und Berechnung von Bewegungsabläufen sowie künstliche Gestaltungsfähigkeit

zur Produktion synthetischer Filme zusammen. Der heute schon erreichte Leistungsstand wird durch Filme wie ANTZ, DAS GROSSE FRESSEN, TOY STORY oder dem Science-fiction-Film STAR WARS, EPISODE 1, eindrucksvoll dokumentiert. Doch auch aus vielen anderen Filmen wie z.B. TITANIC oder selbst Fernsehserien ('Helicops') sind durch Computer-Animation erzeugte Szenen nicht mehr wegzudenken.

Der Begriff "Virtuelle Realität" (virtual reality) kennzeichnet Graphiksysteme höchster Leistungsfähigkeit, mit denen in Echtzeit eine physikalische Umgebung, z.B. ein Molekülverband oder eine Hotelhalle, mit allen wichtigen Effekten berechnet, photorealistisch dargestellt und vom Anwender erlebt werden kann. Hierzu wurden neuartige Ein–/Ausgabegeräte wie Datenhandschuh, Datenanzug oder Sichtgeräte (headmounted display) entwickelt. Folgendes Szenario soll das Konzept der virtuellen Realität verdeutlichen:

Ein Hotel wurde mit einem CAD-System in allen Einzelheiten entworfen. Der Architekt legt die speziellen Ein-/Ausgabegeräte an und kann nun virtuell durch das Hotel gehen. Seine aktuelle Position und Blickrichtung wird durch einen Sender dem System permanent mitgeteilt. Die beiden Minifarbsichtgeräte im Helm vor seinen Augen vermitteln ihm einen korrekten visuellen Eindruck, er kann Lichtverhältnisse überprüfen, den Schall seiner Schritte auf dem Marmor hören und sich, gesteuert durch die Rückstellkräfte im Datenhandschuh, nicht nur den schönsten, sondern auch den weichsten Apfel am Büfett aussuchen.

Das breite Spektrum der Anwendungsmöglichkeiten kann wirtschaftlich nicht durch ein Universalsystem abgedeckt werden. Deshalb werden graphische Systeme entsprechend der Anwendung konfiguriert, d.h. aus einer Menge von Hardware- und Software-Komponenten wählt man geeignete aus.

1.4 Modellkonfiguration graphischer Systeme

Die schnelle Hardware-Entwicklung und die Standardisierungsarbeiten im Softwarebereich trugen dazu bei, einheitliche Terminologien und Modelle zu finden und anzuwenden. So wurde auch ein einheitliches Modell (Grundkonfiguration, *Kernsystem*) für graphische Hardware-Software-Systeme entwickelt.

Basierend auf den Grundkonzepten *Eingabe* (Deuten, Zeigen, Wählen) und *Ausgabe* (Visualisierung graphischer Datenstrukturen) einigte man sich auf die in Bild 1.8 dargestellte Basiskonfiguration eines graphischen Systems. Die einzelnen Komponenten wie Anwendungssysteme (z.B. Modellierungssysteme, CAD-Systeme etc.), Graphiksystem (z.B. Java3D, OpenGL, GKS, PHIGS) und graphische Peripherie (Sichtgeräte, Eingabegeräte) werden in den folgenden Kapiteln näher erörtert, ebenso wie die Konzepte zur graphischen Ein- und Ausgabe sowie zur Dialogführung.

Entsprechend den spezifischen Anwendungsanforderungen (z.B. Bürographik, Simulatoren, CAD) werden graphische Systeme konfiguriert.
Auswahlkriterien sind:

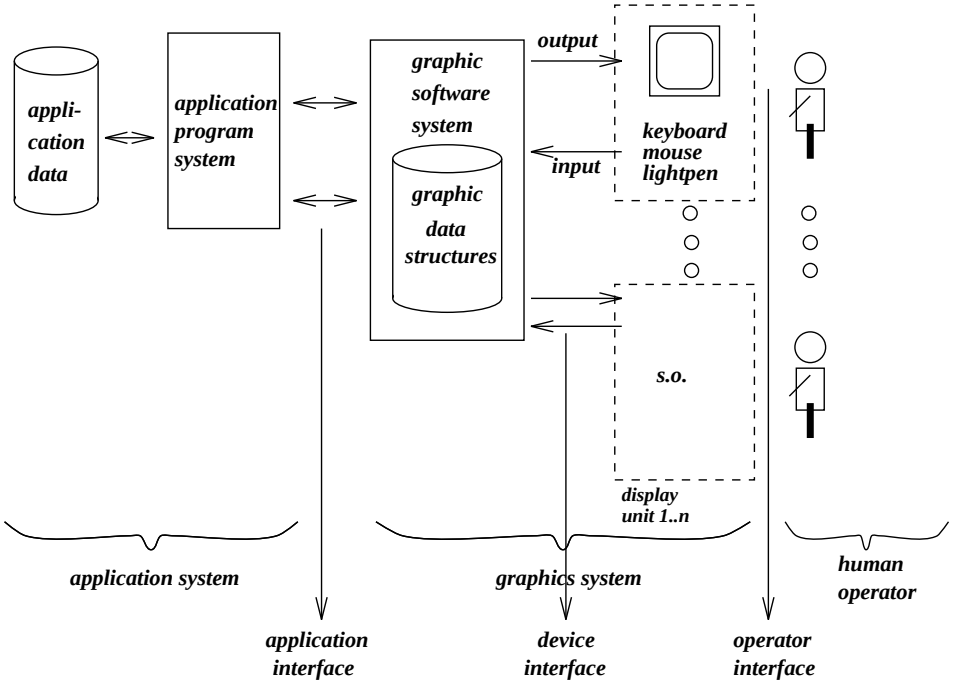


Abbildung 1.8: Basiskonfiguration graphischer Systeme

– für das Anwendungssystem die Anwendungsproblematik, z.B. Darstellen, Zeichnen, Entwerfen;

– für das graphische System eher technische Aspekte wie z.B. 2*D*–, 3*D*–Darstellungsverfahren, Interaktion, Sprachanbindung, graphische Darstellungselemente etc.;

– für die graphischen Geräte die Auflösung, die Darstellungsgeschwindigkeit und –qualität, die Farbgebung, die Größe des Ausgabemediums, logische und physikalische Schnittstellen.

1.5 Graphische Datenstrukturen und Datentypen

Die Daten innerhalb von Anwendungssystemen, die eine Problemlösung graphisch ausgeben sollen, sind oft Größen und Maße (reelle Zahlen), die im Kontext mit dem Anwendungsprozeß zu interpretieren und darzustellen sind. Beispiele dafür sind technische Regelsysteme, in denen Flüsse, Temperaturen und Druck etc. ständig gemessen und auf Bildschirmen dargestellt werden, oder auch die Darstellung statistisch erfaßter Daten mittels Diagrammen etc. Anwendungssysteme im Konstruktions- und Entwurfsbereich benötigen zusätzlich zur Sichtbarmachung von Daten die Zusammenfassung von Bildelementen zu Teilbildern und die Identifikation dieser Teilbilder für die interaktive Manipulation.

Datenstrukturen können hinsichtlich der Funktionalität (passiv, interaktiv, 2*D*, 3*D* etc.) der graphischen Systeme und der Anforderungen der Anwendungen unter-

schieden werden; z.B. benötigen reine Plotsysteme (passiv) weder Strukturinformationen noch Identifikationsmerkmale für Bilder.

Die Datenstrukturen jeder Systemkomponente (Anwendungssystem, graphisches System, graphische Peripherie) werden im Zuge der Visualisierung auf Datenstrukturen des jeweils darunter befindlichen Systems abgebildet (Bild 1.9).

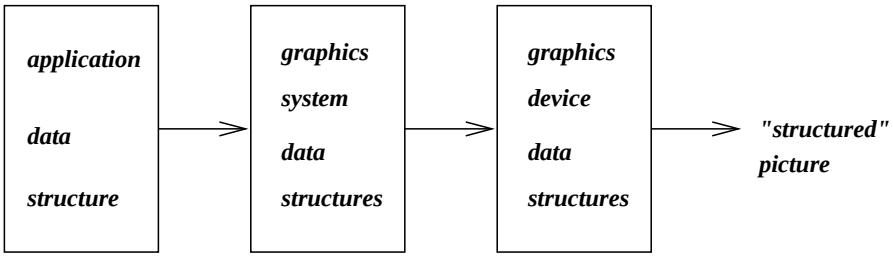


Abbildung 1.9: Abbildung von graphischen Datenstrukturen

1.5.1 Die Daten des Anwendungssystems — die Modelldaten

Die Datenstrukturen des Anwendungssystems werden als Anwendungsmodell bezeichnet. Sie werden i.allg. nicht nur zur graphischen Darstellung, sondern in weitaus größerem Maße für Simulationen und Tests verwendet, und zwar mit der Absicht, Abläufe zu verstehen, sie sichtbar zu machen, zu lehren und mit ihnen zu experimentieren. Das Anwendungsmodell enthält nicht notwendigerweise graphische Informationen. Diese lassen sich aus den Modelldaten herleiten, wenn man z.B. die im folgenden aufgeführten Modellkomponenten mit einbezieht:

- Basisdatenstrukturen,
- Regeln für die Konstruktion strukturierter Daten,
- Beziehungen innerhalb von Datenstrukturen, Verbindungen zwischen Datenstrukturen,
- Namen und Identifikationsmerkmale.

Aus den Daten des Anwendungsmodells lassen sich zum einen geometrische Daten bzw. graphische Elemente ableiten, zum anderen ermöglichen die Modelldaten eine Attributierung der graphischen Elemente. Nichtgeometrische Attribute (Linientyp, Farbe) ergeben sich oft aus dem Modellkontext (worauf bei der Ausgabe Wert gelegt werden soll). Die geometrischen Daten umfassen:

- die Objektart,
- die Objektposition,
- die Objektgröße,
- die Objektdarstellung (Attribute).

Topologische Daten sind ebenfalls im Anwendungsmodell enthalten. Sie beschreiben die Zusammenhänge zwischen den Objekten eines Bildes: “ist Teilbild von” oder “ist verbunden mit”.

Sie werden beispielsweise in einer Verbindungsmatrix gehalten, in Präzedenzgraphen, Ableitungsbäumen etc. Die topologischen Daten können sehr unterschiedlich interpretiert werden, z.B. Nachbarschaften bezeichnen, Bildhierarchien kennzeichnen oder aber die einzelnen Daten eines Objektes in Relation zueinander setzen, z.B. Indizierung von Attributen.

Weiterhin ist üblicherweise der Abstraktionsgrad der Modelldaten des Anwendungssystems höher als der des graphischen Systems. Jede Datenstruktur des Anwendungssystems (z.B. die Front eines Hauses) muß sich auf niedrigere Strukturen (graphische Objekte) und diese wiederum auf elementare graphische Ausgabeeinheiten abbilden lassen (Bild 1.9).

Die Anwendungsmodelle sind häufig hierarchisch aufgebaut. Ausgehend von einer Grundmenge von (Standard–) Komponenten werden Objekte höherer Ordnung definiert. Nur die wenigen Komponenten der Grundmenge sind monolithisch. Eine Abänderung eines derartigen Grundelementes verändert seine Darstellung in allen Bildern des Anwendungssystems, die es verwenden. Neben den hierarchischen Datenstrukturen werden in den Anwendungen häufig Listen– und Ringstrukturen verwendet, die den Objektzusammenhang beschreiben und Bilder als eine lineare Verkettung von Grundelementen ausweisen.

1.5.2 Die Datenstrukturen des graphischen Systems

Graphische Systeme erzeugen Bilder aus *graphischen Objekten*. Diese Bildkomponenten (Teilbilder) sind mit bestimmten Eigenschaften versehen. Auf diesen Objekten ist eine durch das graphische System definierte Menge graphischer Operationen ausführbar, die diese Objekte in ihrer Darstellung verändern können.

Eigenschaften sind:

- Sichtbarkeit,
- Identifizierbarkeit,
- Farbe/Grauwerte,
- Prioritäten (z.B. um dicht beieinanderliegende Objekte eindeutig ansprechen zu können).

Operationen werden im allgemeinen auf das gesamte Objekt, d.h. auf alle in ihm enthaltenen Unterstrukturen, in gleicher Weise ausgeführt. Sie können wie folgt festgelegt sein:

- Erzeugen von Objekten,
- Löschen von Objekten,

– Namensgebung, –änderung,

– Kopieren, Einsetzen von Objekten,

– Transformieren von Objekten,

– Ändern der Eigenschaften.

Ein graphisches Objekt kann aus weiteren, weniger komplizierten Objekten zusammengesetzt sein. Die Zerlegung eines Objektes in immer kleinere graphische Objekte kann jedoch nicht beliebig fortgesetzt werden. Dieser Prozeß ist endlich, es werden irgendwann einmal graphische Objekte erreicht, die sich nicht weiter zerlegen lassen. Diese kleinsten, unteilbaren graphischen Objekte, *graphische Primitiva* genannt, werden je nach der Art der graphischen Daten verschieden dargestellt. In Strichzeichnungen technischer Natur z.B. sind es Geraden– und Kurvenabschnitte. Entsprechend den zu verarbeitenden Primitiva kann man die generative Computergraphik weiter unterscheiden, und zwar nach:

– **Vektorgraphik:**
  In diesem Zweig werden ausschließlich Geraden bzw. Geradenabschnitte als Primitiva für die graphische Darstellung herangezogen.

– **Strichgraphik:**
  Primitiva sind sowohl Geradenabschnitte wie auch Kurvenabschnitte.

– **Flächengraphik:**
  Grundelemente sind Flächen bzw. Teilflächen mit unterschiedlichen Graustufen oder Farbwerten.

– **2D-Graphik:**
  (2-dimensional) befaßt sich mit der Darstellung von 2D–Objekten.

– **3D-Graphik:**
  beschäftigt sich mit der 2D–Darstellung von 3D–Objekten.

– **Rastergraphik:**
  Primitivum ist der Bildpunkt = picture element = pixel.

Aus gerätetechnischer Sicht gibt es heute nur noch Rastergraphik. Man unterscheidet dann noch zwischen 2D– und 3D–Graphik.

Die Objekterzeugung aus Primitiva ist zunächst einstufig hierarchisch [DIN82], sie kann aber mehrstufig hierarchisch werden durch stufenweise Objekterstellung aus bereits erstellten Objekten [ISO84].

Allen graphischen Systemen ist eine gewisse Menge von Primitiva gleich. Sobald ein System geräteunabhängig ist, muß es sowohl Elemente der Vektorgraphik als auch der Rastergraphik in sich vereinen.

1.5.2.1 Graphische Datentypen

Basisdatentyp für alle graphischen Primitiva ist der `PUNKT` in einem Koordinatensystem bzw. eine Position, eine Lage etc. Je nach Dimension des Koordinatensystems unterscheidet man $2D$– und $3D$–Punkte (Bild 1.10). Ein Punkt P wird durch seine x–, y– und z–Komponenten festgelegt, die jeweils den Abstand des Punktes vom Ursprung des Koordinatensystems angeben. Ein $2D$–Punkt kann als Sonderfall eines $3D$–Punktes angesehen werden, bei dem die z–Koordinate konstant $(= 0)$ gewählt ist, d.h. die Ebene $z = 0$ bezeichnet üblicherweise ein $2D$–System.

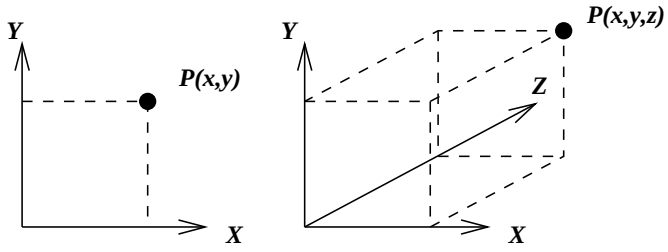


Abbildung 1.10: Der Punkt als graphischer Basisdatentyp

Der Datentyp `PUNKT` wird in einer Programmiersprache entsprechend seinen Koordinaten aus 2 bzw. 3 Real– oder Integer-Werten dargestellt (sofern keine mächtigeren Sprachmittel zur Verfügung stehen):

```
type POINT-3:  real X      type POINT-2: real X
               real Y                    real Y;
               real Z;
```

oder auch $P(X,Y,Z)$ und $P(X,Y)$. Weitere graphische Datentypen basieren auf Punkten, um die Position (Lage) eines graphischen Primitivums im definierten Koordinatensystem und seine Größe festzulegen. Beispiele dafür sind (siehe auch Bild 1.11):

- **Linien** oder **Vektoren**
  als kürzeste Entfernung zweier Punkte P_1 und P_2.
- **Texte**
  werden durch eine Anfangsposition und eine Zeichenfolge beschrieben, zusätzlich sind oft geometrische Attribute definierbar wie z.B. Schreibrichtung, Zeichengröße etc.
- **Polygone**
  stellen eine Linienfolge aus $(n-1)$ aneinanderhängenden Linien zwischen n Punkten $(P_1 \ldots P_n)$ dar.
- **Rechtecke**
  können als **geschlossene Polygone** aus jeweils 4 Punkten erzeugt werden.

Liegt das Rechteck parallel zu den Koordinatenachsen, so genügt die Angabe der Diagonalen (2 Punkte).

– **Dreiecke**
lassen sich durch ihre Eckpunkte definieren.

– **Kreise**
lassen sich durch Mittelpunkt und Kreispunkt, durch Mittelpunkt und Radius oder durch 3 Kreispunkte definieren.

– **Flächen**
lassen sich als zu füllende Rechtecke, Dreiecke, geschlossene Polygone etc. definieren.

Prinzipiell lassen sich alle Objekte der Computergraphik, beginnend mit einfachen Primitiva (z.B. Punkte, Linien) bis hin zu Körpern (z.B. Würfel) durch Punkte beschreiben, deren Relationen dem erzeugenden System bekannt sind (wie bei Primitiva) oder ihm mitgeteilt werden (z.B. durch eine Kantenliste).

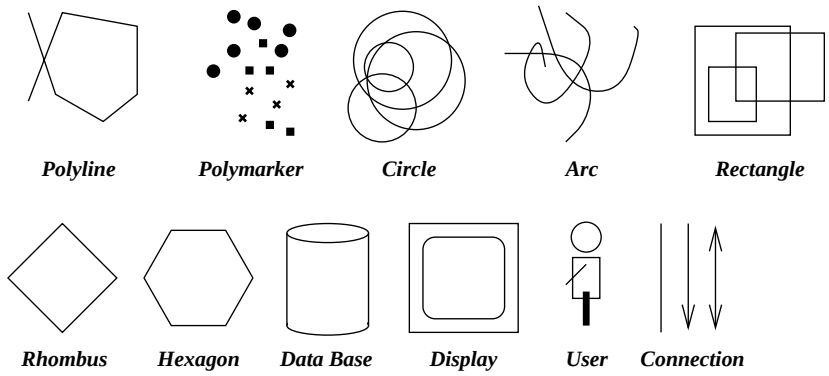


Abbildung 1.11: Graphische Primitiva und höhere Objekte

Die Algorithmen eines graphischen Systems, das unterschiedliche Darstellungen eines graphischen Objektes zuläßt, müssen sicherstellen, daß semantisch gleiche Objektdefinitionen auch zu gleichen Bildern führen.

Man hat sich im Zuge der Standardisierung graphischer Systeme auf einen Standardsatz (Minimalsatz) graphischer Primitiva geeinigt, mit dem es möglich ist, 2*D*–Bilder zu programmieren bzw. 3*D*–Szenen graphisch darzustellen (d.h. auf 2*D*–Bildschirmgeräten sichtbar zu machen):

– Punkte bzw. Markierungen,

– Linien, Linienzüge (Polygone),

– Flächen,

– Text,

– Punktmengen (meist rechteckige Felder von Bildpunkten).

1.5.3 Datenstrukturen und Datentypen graphischer Geräte

Bei der graphischen Ausgabe werden die im Anwendungsmodell enthaltenen graphischen Objekte zunächst auf Daten des graphischen Systems abgebildet (Bild 1.12), bevor sie dann in der nächsten Umsetzungsphase auf Daten des angesteuerten graphischen Gerätes abgebildet werden. Bei diesem Abbildungsprozeß können jeweils höchstens gleichmächtige Datenstrukturen der Zielsysteme verwendet werden. In den meisten Fällen wird jedoch eine Abbildung auf niedrigere Strukturen erfolgen (z.B. Kreis auf eine Folge von Linien des Kreisumfangs) und damit auch Strukturinformation der Modelldaten verloren gehen. Ein Ausnutzen höherer Datenstrukturen des Zielsystems ist nur bedingt möglich (z.B. Adressierung spezieller Geräteeigenschaften wie Splinegeneratoren).

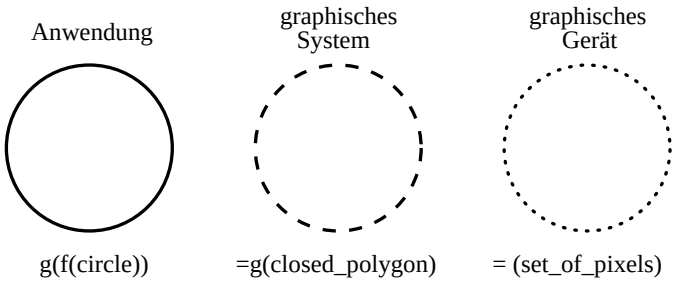


Abbildung 1.12: Verlust von Strukturinformation bei der Abbildung von Modelldaten auf Gerätedaten

Elementare Datentypen von Geräten sind dabei ein zweidimensionales Feld von Rasterpunkten bzw. eine sequentielle Liste, bestehend aus sogenannten “move and draw”–Befehlsfolgen. Ein Rasterpunkt bzw. eine “move-to”-Koordinate ist dabei eine Festpunktzahl aus einem endlichen Zahlenbereich, der der Geräteauflösung entspricht (z.B. 0 . . . 1023).

Basierend auf diesen beiden elementaren Bilddatenstrukturen verfügen heutige Geräte über weitaus höhere *logische* Schnittstellen. Diese ermöglichen oft ein Ansteuern der Geräte auf der Ebene der vorgestellten graphischen Primitiva. Die Geräte führen oft selbst die zur Darstellung notwendigen Transformationen und Attributanbindungen durch, so daß — aus Systemprogrammierersicht — die Datenstrukturen und Datentypen von graphischen Geräten sich an die in “Kernsystemen” (Standards) festgelegten Typen annähern.

1.5.4 Datentypen der graphischen Eingabe

Bei der Darstellung der im Anwendungssystem definierten Modelle bedient man sich graphischer Objekte und Primitiva. Für die Programmierung von Dialogen, d.h. Aktionen (Ausgaben) und Reaktionen (Benutzereingaben) werden ebenso Datentypen benötigt, aus denen unterschiedliche Eingaben (wie z.B. Deuten, Zeigen, Wählen, Benennen etc.) herzuleiten sind.

Die Eingabegeräte selbst liefern physikalische Eingabewerte, die auf der Geräteebe-

ne zunächst noch nicht unmittelbar mit der erzeugten Ausgabe in Zusammenhang zu setzen sind, da die erzeugte Ausgabe und die Eingabe physikalisch häufig voneinander entkoppelt sind. Die Geräte liefern beispielsweise zwei Festpunktzahlen (in einem bestimmten Bereich), eine Festpunktzahl, eine Speicheradresse oder eine Zeichenfolge. Diese Daten sind zunächst einmal in einen lokalen Kontext (Bild 1.13) zu bringen, der z.B. eine Position auf dem Bildschirm bezeichnet, Cursor-, Text- oder Kommandoeingabe bedeutet. Durch Interpretation und Zuhilfenahme von Ausgabedatenstrukturen kann man die physikalischen Eingaben unterschiedlichen Kontexten zuordnen (“was wollte der Benutzer haben”) und somit logische Eingabedaten erzeugen (Position im kartesischen Anwendungskoordinatensystem, eine Auswahl, einen Objektnamen, einen String etc.). Das Anwendungssystem entscheidet daraufhin, in welchen Anwendungskontext die erfolgten logischen Eingaben zu bringen sind. Ein Text oder eine Auswahl kann beispielsweise auf der Ebene der Anwendung als ein Kommando interpretiert werden.

user	physical input	physical out/in	graphics system	application system
construct delete select	2–integer 1–integer characters	command address value position string	object name position text selection	command parameter

Abbildung 1.13: Kontextabhängige Interpretation der Eingabe

1.6 Graphische Programmiersprachen

Eine graphische Programmiersprache (auch Graphiksprache) ist eine Programmiersprache, die die Bearbeitung, Verarbeitung, Speicherung, die Ein– und die Ausgabe graphischer Daten ermöglicht. Programmiersprachen unterscheidet man üblicherweise nach verschiedenen Sprachebenen:

- Maschinensprachen,
- Programmiersprachen, und zwar
  - maschinenorientiert (Assemblierersprachen),
  - problemorientiert (höhere Sprachen),
  - benutzerorientiert (Dialogsprachen),
- Kommandosprachen.

Graphische Sprachen lassen sich ebenfalls in diese Ebenen untergliedern: Die Maschinensprache eines graphischen Systems bzw. eines graphischen Gerätes umfaßt

den Befehlsvorrat, der durch den Entwurf festgelegt ist (bei Geräten oft 'escape'-Sequenzen). Das installierte Graphiksystem kann nicht die Eigenheiten eines jeden Endgerätes unmittelbar nutzen, eine Anpassung der Endgeräte an eine gemeinsame Schnittstelle (Metasprache) ist notwendig.

Problemorientierte graphische Sprachen können realisiert werden durch:

- ein Unterprogrammpaket einer algorithmischen Sprache,
- eine eigenständige Programmiersprache,
- Erweiterung einer algorithmischen Sprache mit Compilererweiterung,
- Erweiterung einer algorithmischen Sprache durch Einsatz eines Precompilers oder
- Einsatz einer erweiterbaren Programmiersprache.

Um graphische Daten auf der Ebene einer problemorientierten Sprache verarbeiten zu können, ist es naheliegend, eine Graphiksprache in Form eines Unterprogrammpaketes (Graphikbibliothek) mit Hilfe einer existierenden Programmiersprache zu implementieren. Fest umrissene Aufgaben wie Eingabe, Ausgabe und graphische Operationen werden einer oder mehreren Prozeduren zugeordnet. Die graphische Sprache selbst ist in diesem Fall nur indirekt definiert. Ein Leitfaden beschreibt die im System gegebenen Unterprogramme mit Namen, Art, Funktion und deren zugeordneten Parameterlisten und gibt Anwendungsbeispiele (z.B. [DIN82],[EKP84]).

Die Nachteile einer derartigen graphischen Sprachdefinition liegen in der Komplexität des Unterprogrammpaketes begründet. Es ist nicht ganz einfach, sich für die Programmierung die Prozeduren mit Parameteranzahl und –typen, Datentransferrichtung usw. zu merken und alle Möglichkeiten auszuschöpfen.

Eigenständige graphische Programmiersprachen umgehen die oben angeführten Nachteile, sie müssen allerdings alle erforderlichen algorithmischen Eigenschaften einer Sprache einschließen. Eine solche Sprache muß direkt definiert sein, d.h. graphische Zuweisungen, graphische Ein-/Ausgabe und sonstige graphische Operationen müssen explizit vorhanden sein. Die Aufgabe ist hier die Definition einer graphischen Sprache durch Syntax und Semantik, die den Bau eines speziellen Compilers ermöglichen. Da zusätzlich zu den graphischen Elementen die algorithmischen Elemente miteinzubeziehen sind, ist eine derartige Sprachdefinition eine recht komplexe Aufgabe.

Die Erweiterung einer bestehenden algorithmischen Sprache ist ein bislang häufig praktizierter Vorschlag zur Implementierung einer graphischen Sprache. Die Erweiterung einer Wirtssprache kann auf zwei Weisen erfolgen: entweder durch Erweiterung des vorhandenen Sprachcompilers oder durch Einsatz eines Precompilers, der die Spracherweiterungen auf die Konstrukte der Wirtssprache abbildet. Der erste Weg setzt gute Kenntnisse des zu verwendenden Compilers voraus, der zweite bedingt eine zweiphasige Verarbeitung der erstellten Programme, bestehend aus der Precompilierung mit anschließender Compilierung. Erweiterbare

Sprachen erlauben die Definition neuer Datentypen durch Spezifikation der internen Darstellung von Daten und ihrer Operationen. Es ist somit möglich, eine einheitliche graphische Sprache aus einem erweiterbaren Sprachkern zu definieren.

Zur Erstellung von dreidimensionalen Graphik-Anwendungen werden häufig Graphik-Softwarebibliotheken benützt. Diese bestehen aus Prozeduren zur Darstellung von dreidimensionalen virtuellen Objekten.
Zur Zeit ist die Schnittstellendefinition OpenGL [SIL93] ein de facto Standard zur Erstellung von Graphik-Anwendungen. OpenGL definiert nur die Schnittstelle zu einer Graphik-Softwarebibliothek, das heißt die Prozedurdeklarationen, jedoch nicht deren Implementierung. Zu OpenGL sind von verschiedenen Firmen und Organisationen Implementierungen für Unix/Linux-, Windows-, MacOS- und andere Systeme erhältlich. Mit Hilfe von Prozeduraufrufen einer OpenGL-Implementierung lassen sich Graphik-Anwendungen effizient erstellen. Für OpenGL ist ein Gremium aus Vertretern von Soft- und Hardware-Firmen verantwortlich.
Eine OpenGL-Implementierung stellt Prozeduren zur Darstellung von Linien und Polygonen zur Verfügung. Will man beispielsweise mit einer OpenGL-Implementierung eine Kugel definieren und anschließend von der OpenGL-Implementierung darstellen lassen, muß man zuvor die Kugeloberfläche aus Dreiecken modellieren. Alle Dreiecke zusammen müssen eine annähernd kugelförmige Fläche ergeben. Erst die einzelnen Dreiecke lassen sich dann durch den Aufruf entsprechender Prozeduren einer OpenGL-Implementierung auf dem Bildschirm darstellen.
Um solche Körper einfacher definieren zu können, werden deshalb in der Regel weitere Graphik-Softwarebibliotheken benutzt, die auf einer OpenGL-Implementierung aufsetzen. Mit ihnen kann eine Kugel durch ihren Mittelpunkt und Radius definiert werden. Ein solche Softwarebibliothek zerlegt selbständig die Kugeloberfläche in Dreiecke. Anschließend ruft die Softwarebibliothek Funktionen der OpenGL-Implementierung auf, welche die Dreiecke auf dem Bildschirm darstellen. Eine leicht zu bedienende Graphik-Softwarebibliothek ist Java3D. Für Java3D ist die Firma Sun verantwortlich.

1.7 Ausblick

Die Graphische Datenverarbeitung wird auch in den kommenden Jahren von den Leistungssteigerungen der Mikroelektronik neue Entwicklungsimpulse erhalten. Wo früher Leistungsangaben mit M(10^6) wie z.B. Megabyte begannen, steht heute fast überall ein G(10^9) für Giga... . Damit sind die Voraussetzungen geschaffen, auch volumenorientierte Graphik gerätetechnisch zu unterstützen. Speicher mit einem Volumen von 24×1024^3 sind kostengünstig zu realisieren. Der Weg von der Vektor– über die Flächen– zur Volumengraphik ist vorgezeichnet. Wichtige Anwendungen der Medizin (siehe Bild 1.14) und des wissenschaftlichen Rechnens nutzen die Volumengraphik seit geraumer Zeit [Str91].

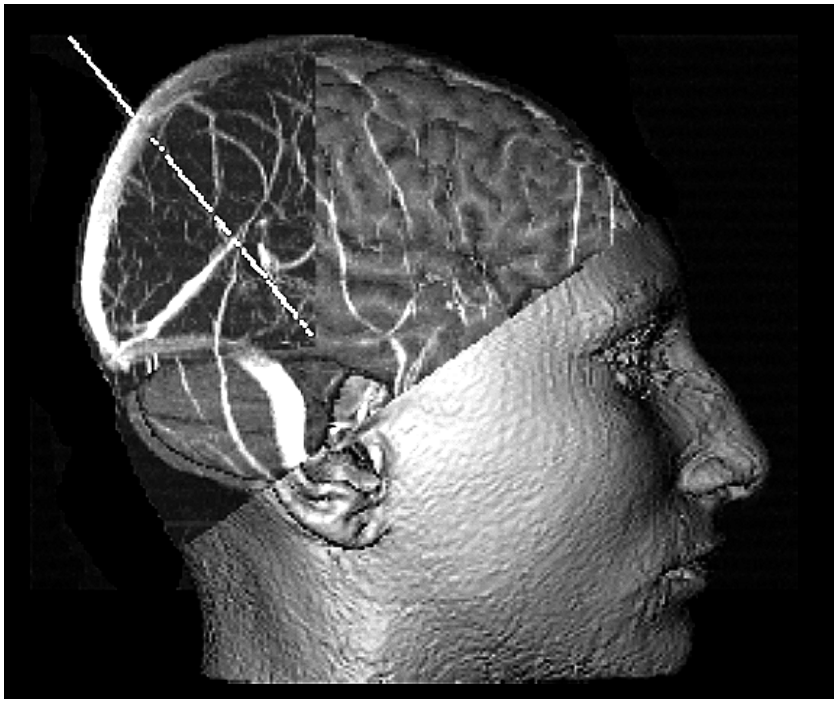


Abbildung 1.14: Stereotaxieplanung mit Volumengraphik

Kapitel 2

Rastertechnik

Graphiksysteme zeichnen sich gegenüber “normalen” Computersystemen durch spezielle Peripherie zur graphischen Ein–/Ausgabe, durch Spezialrechner (Display–Prozessor) zur Berechnung spezieller Graphikfunktionen und durch ihren Echtzeitcharakter mit hohen Anforderungen an das Interaktionsvermögen aus. In dieser Lehreinheit werden, ausgehend von den Peripheriegeräten, alle Hardwarekomponenten eines Graphischen Systems betrachtet. Dabei konzentrieren wir uns auf Rastergeräte und schließen die Behandlung von Rasteralgorithmen zum Zeichnen der graphischen Primitiva mit ein.

2.1 Das Sichtgerät (Display)

Das Sichtgerät mit dem Bildschirm ist die dominierende Komponente des graphischen Arbeitsplatzes, über die der graphische Dialog zwischen Benutzer und Anwendungsprogramm größtenteils abgewickelt wird. Die Kathodenstrahlröhre (CRT = Cathode Ray Tube) hat sich seit den Anfängen der Graphischen Datenverarbeitung trotz vieler neuer Technologien als meist angewandtes Ausgabegerät behauptet. Die Gründe hierfür liegen u.a. in folgenden Vorteilen:

Kathodenstrahlröhre = CRT

Vorteile

– hohe Auflösung,
– einfache Adressierung,
– volle Farbtüchtigkeit,
– niedriger Preis bei hoher Zuverlässigkeit.

2.1.1 Die Kathodenstrahlröhre

Die Kathodenstrahlröhre (CRT) wird sowohl in Vektor– oder kalligraphischen Sichtgeräten (Oszilloskop–Prinzip mit wahlfreier x-y–Positionierung) als auch in Rastersichtgeräten (z.B. nach dem Fernsehrasterprinzip) eingesetzt. Bild 2.1 zeigt

stark schematisiert den Aufbau einer Kathodenstrahlröhre. Sie besteht aus einer Hochvakuumröhre mit Strahlerzeugung, Fokussierung, Beschleunigung, Ablenkung und Phosphorschicht.

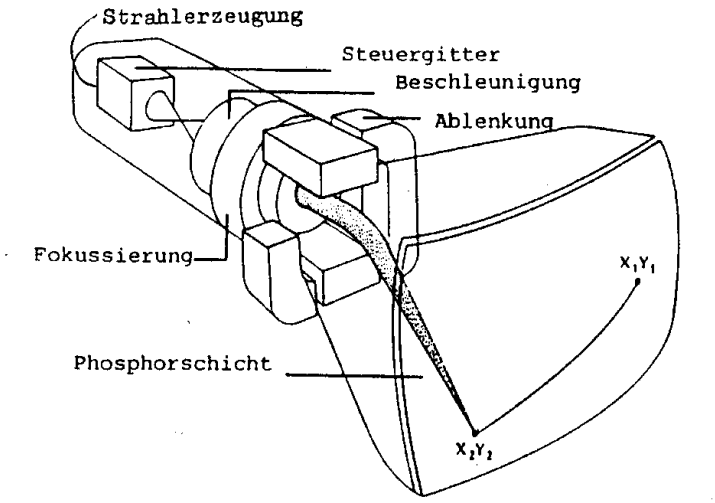


Abbildung 2.1: Aufbau einer Kathodenstrahlröhre

Prinzipielle Wirkungsweise

Die Strahlerzeugung wird durch eine beheizte Kathode mit Steuergitter (Wehneltzylinder) gebildet.

Das Steuergitter erlaubt eine Strahlmodulation, mit der die Helligkeit des Bildes eingestellt werden kann. Die Fokussierung ist ein magnetisches oder elektrostatisches elektronenoptisches System und wirkt auf das aus der Kathode austretende Elektronenbündel wie ein Linsensystem. Es bündelt die Elektronen und formt aus ihnen einen feinen Elektronenstrahl. Die Beschleunigung entsteht in einem elektrischen Feld und wird durch eine oder mehrere Anoden meist im Bereich der Fokussierung gebildet. Der Elektronenstrahl trifft jedoch nicht auf die Anode, sondern tritt durch ein Loch in ihr in das Ablenkungssystem ein. Dies kann elektrostatisch durch Ablenkplatten oder über Ablenkspulen elektromagnetisch ausgeführt sein. In jedem Fall wird der Elektronenstrahl entsprechend den angelegten Ablenkspannungen bzw. entsprechend den fließenden Ablenkströmen abgelenkt, damit er an der gewünschten Stelle auf den Bildschirm trifft und die dort befindliche Bildphosphorschicht zum Leuchten bringt. Die Ablenkung erfolgt unabhängig voneinander in zwei orthogonalen Richtungen. Es können also Intensität, horizontale und vertikale Position gesteuert werden.

Die meisten Sichtgeräte, z.B. auch die Fernsehmonitore, arbeiten mit elektromagnetischer Ablenkung, da die Magnetspulen außerhalb des Glaskolbens angeordnet werden können.

2.1.1.1 Phosphor und Persistenz (Nachleuchtdauer)

Bildwiederholung

Der Bildschirm ist mit einem Phosphor belegt, der beim Auftreffen des Elektronenstrahls Licht emittiert (Fluoreszenz). Nach dem Abschalten des Elektronenstrahls wird mit abnehmender Helligkeit einige Zeit weiter Licht emittiert (Phosphoreszenz). Die Nachleuchtdauer (Persistenz) ist bei der Auswahl einer Bildröhre neben der Farbe und der Helligkeit von größter Wichtigkeit. Sie bestimmt, wie oft das Bild regeneriert, d.h. wiederholt werden muß, damit der Benutzer den Eindruck eines stehenden, flimmerfreien Bildes erhält.

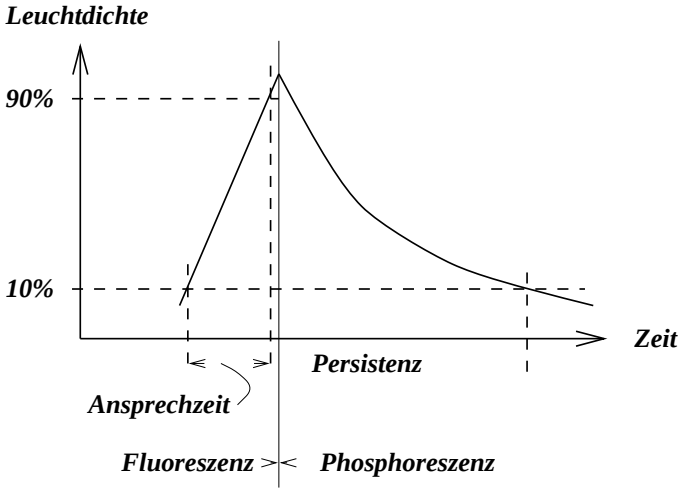


Abbildung 2.2: Zeitverlauf der Lichtemission des Phosphors

Wahl des Phosphors

Die Bildwiederholfrequenz liegt bei Sichtgeräten zwischen 30 und 80 Bildern pro sec. (beim Kino 24 Bilder pro sec.). Sie ist u.a. vom verwendeten Phosphor abhängig. Große Bildwiederholfrequenz und kurze Nachleuchtdauer bedeuten, daß für den Aufbau jedes einzelnen Bildes wenig Zeit zur Verfügung steht. Die darstellbare Information ist dann gering. Eine kleine Bildwiederholfrequenz und große Nachleuchtdauer haben Schlierenbildung bei bewegten Gegenständen zur Folge, man sieht “Geisterbilder”.

Ebenfalls großen Einfluß darauf, ob ein Beobachter ein Bild als flimmerfrei empfindet, haben Helligkeit, Struktur und Farbe des Bildes sowie die umgebende Raumbeleuchtung. Zudem ist die “Flimmerfrei”–Frequenz sehr von der Person des Beobachters abhängig. Deshalb werden heute meist Bildwiederholfrequenzen oberhalb von 60 Hz verwendet.

Der Zwang zur Bildwiederholung hat die Entwicklung der GDV wesentlich beeinflußt. Einerseits ist die Bildwiederholung mit einem Zentralrechner sehr unwirtschaftlich, weshalb immer neue Varianten von Displayprozessoren zur Lösung dieses Problems entwickelt wurden [MS68], andererseits ist sie der Schlüssel zu dynamischen Darstellungen und hoher Interaktionsrate.

2.1.1.2 Gammakorrektur

Die Helligkeit des Leuchtpunktes wird bei CRTs durch den Strahlstrom (Stärke des auf dem Phosphor auftreffenden Elektronenstrahls) bestimmt. Da die Beschleunigungsspannung konstant ist, kann von einem linearen Zusammenhang zwischen Strahlstrom und emittierter Lichtleistung ausgegangen werden. Trotzdem ist die Ansteuerkennlinie fast aller erhältlichen Monitore nicht linear. Das liegt an einer nichtlinearen Strahlstrom-/Steuerspannungs-Kennlinie, die für die Fernsehübertragung gewollt und durch die Norm festgelegt ist. Der Zusammenhang zwischen Steuerspannung U_G am Steuergitter und Strahlstrom I (Helligkeit) kann durch folgende Gleichung ausgedrückt werden:

$$I = I_{max} * (U_G / U_{G_{max}})^{\gamma},$$

wobei I_{max} eine Gerätekonstante ist.

Nichtlineare Ansteuerkennlinie des Bildschirms

Bild 2.3 zeigt die Kennlinie eines Monitors mit $\gamma = 2.2$. Wird die Röhre mit 60% der maximalen Eingangsspannung angesteuert, so werden nur 33% der maximalen Helligkeit erreicht ($0.6^{2.2} = 0.33$).

Um Linearität zu erreichen, muß mit einer korrigierten Steuerspannung

$$\left(\frac{U_G}{U_{G_{max}}}\right)^{\frac{1}{\gamma}}$$

gearbeitet werden. Dies leistet die Gammakorrektur, deren Wirkungsweise in Bild 2.3 verdeutlicht wird.

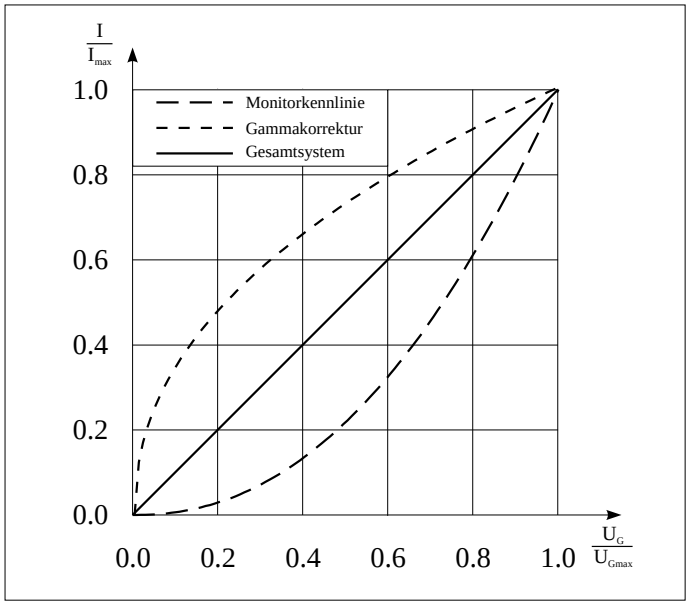


Abbildung 2.3: Gammakorrektur bei einem Fernsehmonitor, Helligkeit I in Abhängigkeit von der Ansteuerspannung U_G

Beim Fernsehen wird die Korrektur dieses Effekts auf der Aufnahmeseite vorgenommen. Da die Helligkeit vom menschlichen Sehsystem in etwa logarithmisch

erfaßt wird (d.h. eine exponentielle Helligkeitssteigerung wird als linear empfunden), ist durch dieses Vorgehen gewährleistet, daß die Bereiche kleinerer Helligkeit gegen Übertragungsfehler nicht empfindlicher sind als die Bereiche größerer Helligkeit.

In der GDV gibt es für die Durchführung der Gamma-Korrektur mehrere Möglichkeiten:

1. Die Helligkeitswerte werden gleich bei der Berechnung korrigiert.
2. Die unkorrigierten Werte werden durch eine vorberechnete Tafel (Lookup table) korrigiert.
3. Im Monitor wird eine analoge Gammakorrektur durchgeführt.

Die erste Methode erlaubt eine individuelle Anpassung an Monitor und Betrachter. Graphiksysteme bieten meistens die ersten beiden Möglichkeiten als Optionen an. Bei einem Farbbildschirm muß die Korrektur für jede Farbkomponente durchgeführt werden.

2.1.2 Die Lochmaskenröhre

Die Lochmaskenröhre in Bild 2.4 ist der am häufigsten verwendete Röhrentyp für alle Arten von Farbdarstellungen. Drei Elektronenstrahlsysteme werden verwendet, um die Phosphorleuchtpunkte oder –streifen für die drei Grundfarben anzusprechen. Die Punkte sind eng genug angeordnet, um dem Auge als nur einer zu erscheinen. Die Farbe ergibt sich aus der entsprechenden additiven Mischung der einzelnen Punkte. Eine Lochmaske wird verwendet, um sicherzustellen, daß jeder Strahl nur den für ihn vorgesehenen Farbpunkt anspricht. Die Strahlen für die roten, grünen und blauen Punkte müssen im richtigen Winkel durch die Maskenöffnung passieren, um ihre zugehörigen Phosphorpunkte zu treffen (Konvergenz).

Lochmaske

Zwei verschiedene Anordnungen sind für die drei Phosphorfarbpunkte und die drei Strahlerzeugungssysteme gebräuchlich. Der Phosphor kann in Punkten angeordnet werden (Bild 2.4) oder in Streifen (Bild 2.5).

Wenn ein Maximum an Auflösungsvermögen verlangt wird, benutzt man die Punktanordnung des Phosphors, weil das Punktmuster kleinere horizontale Abstände zwischen den Triaden (d.h. Abstände, bis sich eine gegebene Farbe wiederholt) zuläßt. Die Strahlerzeugungssysteme sind meistens in einer “Delta”–Form, passend zu dem Punktmuster, oder in einer “Inline”–Position, passend zu den Streifen, angeordnet.

Die Inline–Anordnung der Strahlerzeugungssysteme hat den Vorteil, daß die Schaltung, um die drei Strahlen an der Maske konvergent zu halten, weniger aufwendig ist. Die Röhren mit höchster Auflösung sind nach dem “Delta”–Prinzip aufgebaut. Ihre Konvergenz in den Randzonen der Bildschirmfläche ist oft unbefriedigend.

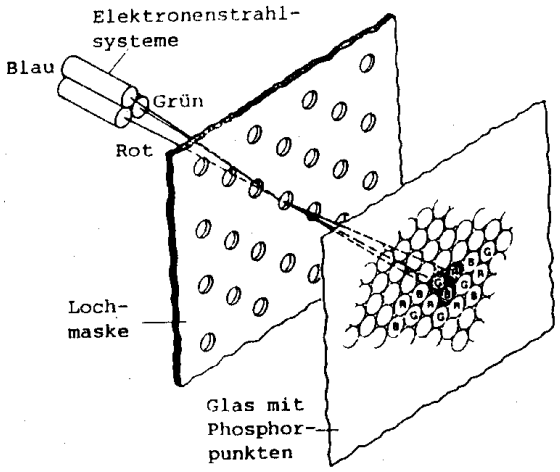


Abbildung 2.4: Lochmaskenröhre mit Delta–Anordnung des Elektronenstrahlsystems und punktförmigem Phosphor

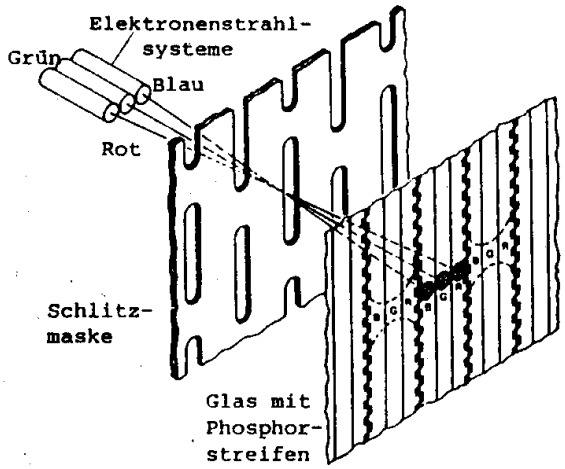


Abbildung 2.5: Inline–Anordnung des Elektronenstrahlsystems mit streifenförmiger Anordnung des Phosphors

2.1.3 Flüssigkristallanzeigen

Flüssigkristallanzeigen [Jac92] (LCD = Liquid Cristal Display) werden seit etwa 1970 in Bildanzeigesystemen eingesetzt. Zu den besonderen Vorteilen dieser Anzeigen zählen vor allem ihr minimaler Stromverbrauch und ihre sehr niedrige Betriebsspannung.

Ein wesentlicher Nachteil der Flüssigkristallanzeigen ist ihre passive Arbeitsweise, bei der ihre Anzeigeelemente auftreffendes Licht durchlassen oder reflektieren. Dadurch, daß diese Anzeigen kein Licht emittieren, sind zusätzliche Lichtquellen erforderlich, um die Anzeige hinreichend beleuchten zu können.

2.1.3.1 Grundlagen

Flüssigkristalle wurden bereits im Jahre 1888 von dem Physiker F. Reinitzer entdeckt. Sie besitzen die Fließeigenschaften gewöhnlicher Flüssigkeiten. Im Unterschied zu diesen weisen jedoch ihre organischen Moleküle eine Orientierungsordnung auf, wie sie für Kristalle typisch ist. Die Form dieser Moleküle ist langgestreckt oder scheibenförmig, wobei ihre Achsen einheitlich ausgerichtet sind. Eine Ausrichtung der Moleküle auf eine der üblichen kristallinen Gitterstrukturen besteht jedoch nicht.

Abhängig von der Art der Ausrichtung ihrer Moleküle teilt man Flüssigkristalle in nematische, smekmatische und cholesterinische Kristalltypen ein. *Nematische Flüssigkristalle* sind durch eine fadenförmige Ausrichtung der Moleküle gekennzeichnet. Die Molekülanordnung der *smekmatischen Flüssigkristalle*, die häufig in Seifen zu finden sind, ist schichtenförmig. Die *cholesterinischen Flüssigkristalle*, die auch in Cholesterinen nachweisbar sind, weisen hingegen eine Molekülorientierung auf, die sich wendelförmig verändert. Auf eine vertiefte Darstellung ihrer Unterschiede und ihrer technischen Einsetzbarkeit wird hier verzichtet und auf die Spezialliteratur verwiesen [Hug89], [Bil83], [Cha77].

Flüssigkristallzellen werden mit zwei parallelen Glasplatten aufgebaut, die sich im Abstand von $5\mu m$ bis $10\mu m$ voneinander entfernt befinden und die den Flüssigkristall einschließen. Die Flüssigkristallmoleküle einer Zelle müssen, ohne daß ein elektrisches Feld einwirkt, mit einer bestimmten Orientierung zu den Oberflächen der Glasplatten ausgerichtet sein. Diese Ausrichtung wird beispielsweise durch eine mikroskopisch feine Riffelung der Glasplattenoberflächen oder durch Auftragen einer dünnen, dielektrischen Schicht von Siliziummonoxid bewirkt, vgl. [Pro87].

Flüssigkristallanzeigen lassen sich in reflektive, transmissive und transflexive Anzeigen unterteilen. Während sich reflektive Anzeigen gut für den Tageslichtbetrieb eignen, werden transmissive Anzeigen, die in der Regel mit einer Hintergrundbeleuchtung ausgestattet sind, vorwiegend für Arbeiten in dunklen Räumen verwendet. Die transflexiven Anzeigen enthalten eine Reflektorfolie, die sowohl das Licht einer Hintergrundbeleuchtung durchläßt als auch den Lichtanteil reflektiert, der auf die Frontseite der Anzeige auftrifft. Transflexive Anzeigen sind daher im Tageslichtbetrieb wie auch im Dunkeln einsetzbar.

2.1.3.2 Aufbau und Funktion

Im folgenden wird speziell auf die Funktionsweise der Anzeigen eingegangen, die auf dem verdrillt-nematischen Flüssigkristalltyp (twisted nematic cells) basieren, da dieser derzeit die größte Verbreitung und Bedeutung besitzt.

Die Innenseiten der beiden oben erwähnten Glasplatten, die den Flüssigkristall einschließen, werden mit mikroskopisch feinen Längsriffelungen versehen. Die Glasplatten sind mit einem Elektrodenmaterial bedampft, das sowohl durchsichtig als auch leitend ist. Als Material wird vielfach Indiumzinnoxid (ITO = **i**ndium **t**in

oxide) verwendet. Die Außenseiten der Glasplatten sind mit Polarisationsschichten belegt, die nur das Licht mit der Wellenebene des Polarisators durchlassen. Die Wellenebene und damit die maximale Filterwirkung des einen Polarisators ist zu der des gegenüberliegenden Polarisationsfilters, der als Analysator bezeichnet wird, um 90° verdreht. Infolge der Riffelungen in beiden Platten, die rechtwinklig zueinander ausgerichtet sind, werden die Achsen der Flüssigkristallmoleküle, die die Plattenoberflächen berühren, so beeinflußt, daß sich diese gleichfalls im rechten Winkel zueinander einstellen.

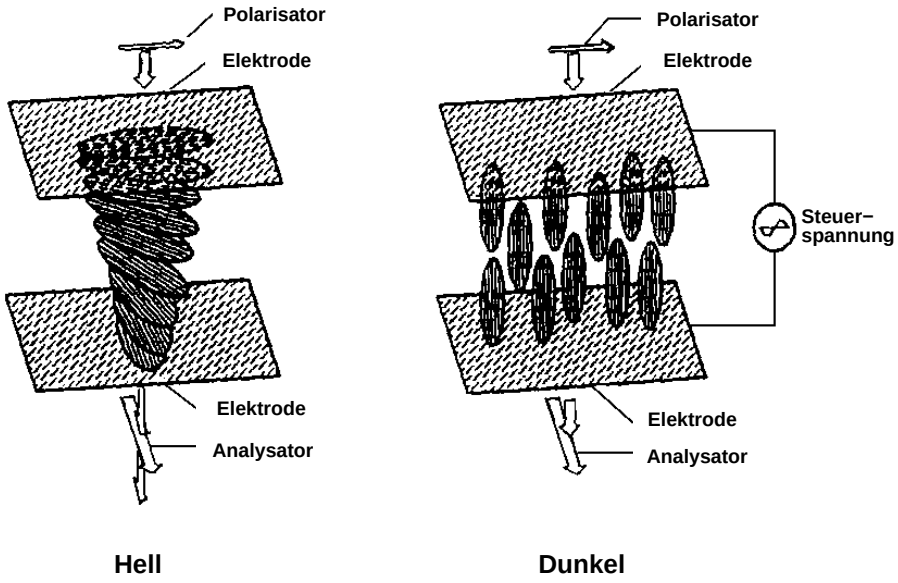


Abbildung 2.6: Funktion einer verdrillten nematischen Flüssigkristallzelle

Hierdurch wird, wie in Bild 2.6 (in Anlehnung an [Hug89]) dargestellt, ein um 90° verdrehter Molekülfaden erzeugt, der die Polarisationsebene des Lichtes so verändert, daß diese mit der Polarisationsrichtung des Analysators übereinstimmt. Die Lichtabschwächung ist in diesem Fall minimal.

Legt man ein elektrisches Feld an die Elektroden, so werden die Molekülachsen aus ihrer Ruhelage gedreht. Dies hat zur Folge, daß sich auch die Polarisationsebenen der durch den Kristall hindurchtretenden Lichtwellen verändern. Da der Analysator, der nur das Licht mit der ursprünglichen Wellenebene vorzugsweise durchläßt, in diesem Fall stark absorbierend wirkt, erscheint die Flüssigkristallzelle als lichtundurchlässig. Das maximal erreichbare Kontrastverhältnis zwischen beiden Schaltzuständen beträgt nach [Pro87] etwa 50:1.

Um die elektrolytische Zersetzung der Elektroden zu vermeiden, wird die Flüssigkristallzelle vielfach mit den Wechselspannungsimpulsen ohne Gleichspannungsanteil betrieben. Für den Fall, daß ein Gleichspannungsbetrieb erforderlich ist, sind die Elektroden mit einer lichtdurchlässigen Isolierschicht aus Indiumzinnoxid (ITO) geschützt.

2.1.3.3 Anzeigenmatrix

Zur Ansteuerung der einzelnen Zellen innerhalb einer Anzeigenmatrix werden vielfach Dünnfilmtransistoren (TFT = Thin-Film-Transistor) verwendet. Diese Feldeffekttransistoren werden in einer matrixförmigen Anordnung auf einem sehr dünnen amorphen Siliziumsubstrat (a-Si) aufgebaut und, wie in Bild 2.7 dargestellt, mit Zeilen- und Spaltenleitungen verbunden.

Ein Problem dieser Anordnung ist, daß die TFT-Schaltmatrix als ein integrierter Schaltkreis zu betrachten ist [Hug89], deren Transistorzellen sich auf einer einzigen Scheibe aus durchsichtigem Siliziumsubstrat befinden. Infolge der Größe dieser Siliziumscheibe, die durch die Bilddiagonale bestimmt wird, treten erhebliche fertigungstechnische Schwierigkeiten auf. Jedem Schalttransistor ist eine Flüssigkristallzelle (LC-Zelle) zugeordnet, die von einer separaten Elektrode angesteuert wird. Eine auf einem Glassubstrat aufgetragene Gegenelektrode ist allen LC-Zellen gemeinsam. Die Elektrodenflächen sind gegenüber dem Flüssigkristall isoliert und wirken somit wie Zellenkondensatoren.

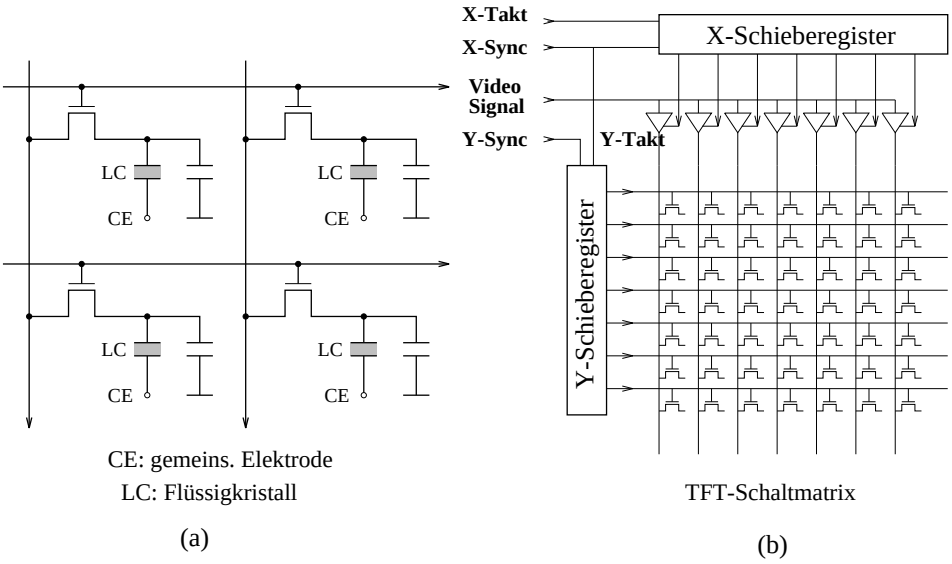


Abbildung 2.7: Steuerung einer TFT-Matrix: a) TFT-Matrix, b) Matrixansteuerung

Im Normalfall sind die LC-Zellen lichtdurchlässig, da infolge der gesperrten Transistoren keine Spannung an den LC-Elektroden anliegt. Das Ausschalten einer LC-Zelle erfolgt durch Anlegen eines hinreichend hohen Spannungspotentials an die entsprechenden Spalten- und Zeilenleitungen. Hierdurch wird der am Kreuzungspunkt befindliche Transistor geöffnet und sein Drain-Potential U_D zur LC-Elektrode durchgeschaltet.

Die Steuerung der TFT-Schaltmatrix erfolgt mit Hilfe zweier Schieberegister. Das X-Schieberegister wird mit den Taktimpulsen X-Takt weitergeschaltet. Zur Fortschaltung des Y-Schieberegisters dient das Synchronisationssignal X-Sync, das vor jeder neuen Zeile ausgelöst wird. Es wird direkt über die geöffnete Source-

Drain-Strecke des Schalttransistors zur Elektrode der Flüssigkristalle geführt, deren Lichtdurchlässigkeit von der durchgeschalteten Videosignalspannung abhängt. Die Bildsynchronisierung erfolgt mit Hilfe des Signals Y-Sync, das nach der letzten Bildzeile ausgelöst wird.

2.1.3.4 Farbdarstellung

Um Farbdarstellungen zu erreichen, sind jeweils drei LC-Zellen zu einem RGB-Farbtripel zusammengefaßt. Wie in Bild 2.8 dargestellt, sind die einzelnen Zellen mit Mikrofiltern für die Farben Rot, Grün und Blau ausgestattet.

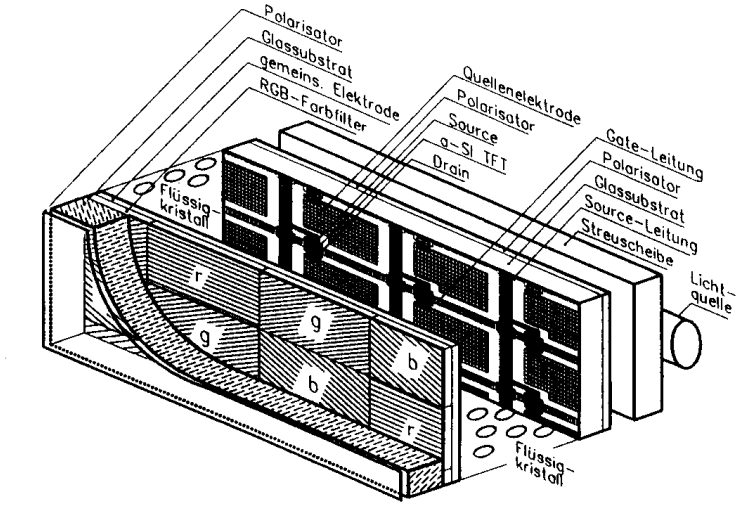


Abbildung 2.8: Prinzipieller Aufbau der LC-Anzeige

Diese LC-Zellen, die wie Bild 2.9 zeigt, entweder orthogonal oder deltaförmig angeordnet sind, dienen zur Farbfilterung der Weißlichtanteile, die von den einzelnen LC-Zellen eines Farbtripels durchgelassen werden.

2.2 Drucker

Graphische Arbeitsplätze beinhalten in der Regel eine Möglichkeit zur Ausgabe des Bildschirminhalts auf Papier. Im folgenden werden kurz die Arbeitsweisen von Laserdruckern und Tintenstrahldruckern erläutert.

2.2.1 Laserdrucker

Xerographie

Der Laserdrucker nutzt das xerographische (elektrographische) Aufzeichnungs-

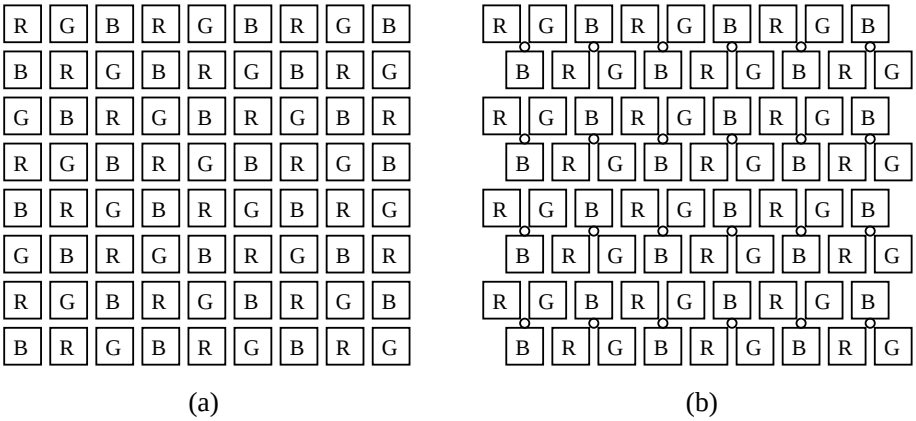


Abbildung 2.9: Zellenanordnungen: a) orthogonal, b) deltaförmig

verfahren [TK82], das auch in den meisten leistungsfähigen Kopierern angewandt wird.

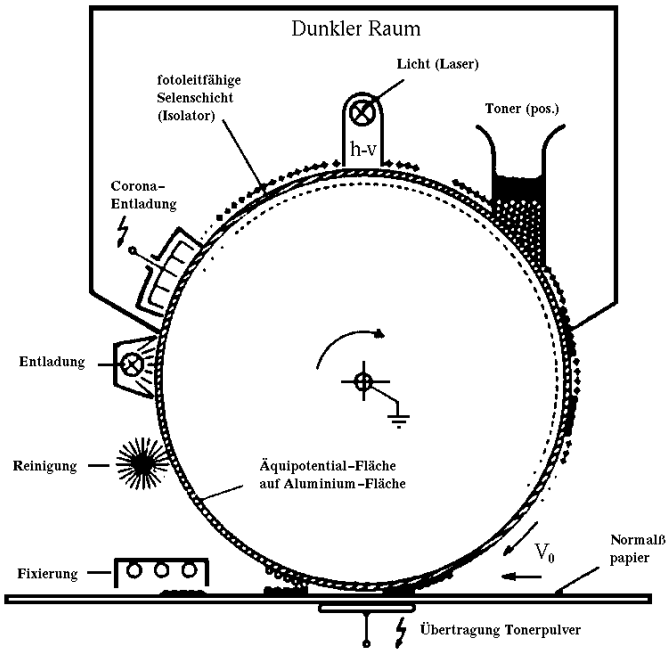


Abbildung 2.10: Prinzip des xerographischen Druckverfahrens

In einem Dunkelraum (Bild 2.10) wird mittels einer Coronaentladung die photoleitfähige Selenschicht auf einer sich kontinuierlich drehenden Aluminiumtrommel mit einer homogenen, positiven Flächenladung versehen. Dieser photoleitfähigen Schicht, die sich im Dunkeln wie ein Isolator und bei Licht wie ein Halbleiter verhält, wird die zu druckende Information durch entsprechende Belichtung als latentes Entladungsbild aufgeprägt, das anschließend in einer Tonerstation mit einem positiven Toner sichtbar gemacht wird.

Hinter dem Toner wird dieses Bild mit dem synchron laufenden Normalpapier in Kontakt gebracht. Dabei werden die Tonerpartikel durch Anlegen eines elektrostatischen Feldes auf das Papier übertragen und anschließend thermisch fixiert. Vor einer erneuten Aufladung wird die photoleitfähige Schicht von restlichen Tonerpartikeln gereinigt und entladen. Das Bild wird von einem Laserstrahl als Punktmuster auf die photoleitfähige Schicht geschrieben. Die Auflösung beträgt ca. 10 Punkte pro *mm*, womit eine sehr gute Bildqualität für Text als auch für beliebige Graphik (z.B. japanische Schriftzeichen, begrenzte Grauwertdarstellungen) erzielt wird. Farbtauglichkeit läßt sich durch Koppelung mehrerer Stationen mit verschiedenfarbigen Tonern erzielen.

Laserdrucker sind Seitendrucker, denn der Druckvorgang läßt sich jeweils nur zum Seitenende stoppen. Die maximale Druckleistung liegt bei ca. 150 Seiten pro Minute.

2.2.2 Tintenstrahldrucker

Beim Tintenstrahldrucker entsteht das Bild durch gezieltes Spritzen von Tintentröpfchen auf Normalpapier (Bild 2.11).

Diese Technik wird inzwischen so gut beherrscht, daß die Druckqualität fast an die des Laserdruckers herankommt. Die auf einem beweglichen Druckkopf angebrachten Düsen können mit unterschiedlichen Tinten, z.B. Cyan, Gelb und Magenta (häufig zusätzlich schwarz) versorgt werden, so daß gute Farbbilder erzeugt werden können. Weitere Vorzüge des Tintenstrahldruckers sind Geräusch– und Verschleißarmut sowie geringer Energieverbrauch.

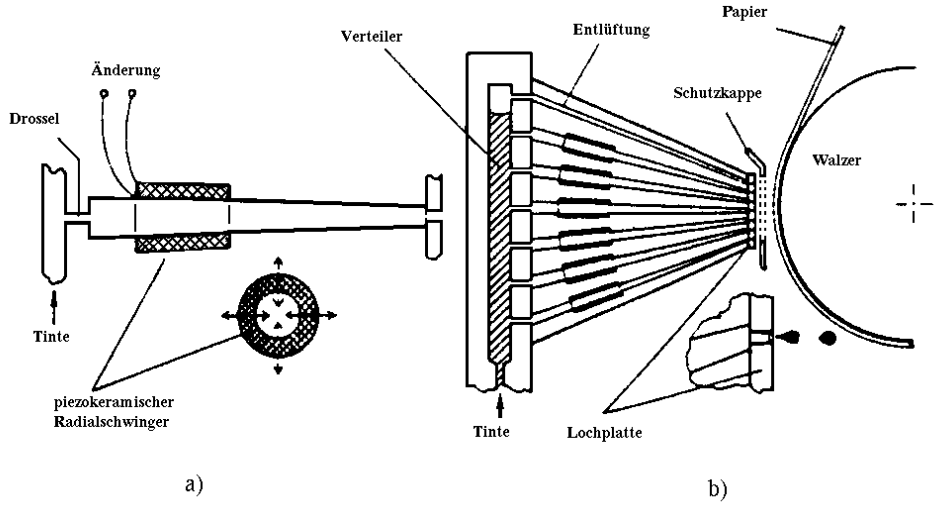


Abbildung 2.11: Tintenschreibwerk nach dem Unterdruckverfahren
a) Einzeldüse, b) Kopf mit 2×6 Düsen

Die Erzeugung der Tintentröpfchen mit einem Durchmesser von ca. $50\mu m$ bis $100\mu m$ kann auf die unterschiedlichsten Weisen erfolgen. Verbreitet sind außer

Verfahren, die piezoelektrische Schwinger benutzen, vor allem das sogenannte bubble-jet-Verfahren. Dabei wird in unmittelbarer Nähe der Düse durch ein kleines Heizelement Tinte zum Verdampfen gebracht, was den Ausstoß eines Tröpfchens zur Folge hat. Die Herstellung von Druckköpfen, die solche Düsen enthalten, ist inzwischen so billig geworden, daß bei manchen Druckern Druckkopf und Tintenpatrone eine Wegwerfeinheit bilden.

2.3 Eingabegeräte

Für die GDV werden selbstverständlich alle Eingabegeräte benutzt, die auch für die alphanumerische Datenverarbeitung eingesetzt werden. Darüberhinaus sind verschiedene Eingabegeräte entwickelt worden, die den Möglichkeiten der Graphik und dem Interaktionsbedürfnis des Benutzers besser gerecht werden.

2.3.1 Der Lichtgriffel (Lightpen)

Der Lichtgriffel erlaubt dem Benutzer, direkt mit dem Bild auf dem Sichtgerät zu kommunizieren. Wesentliches Element des Lichtgriffels ist ein Lichtsensor (Photodiode, –transistor), der eintreffende Lichtimpulse verstärkt und an die Displayhardware zur Verarbeitung weitergibt (Bild 2.12).

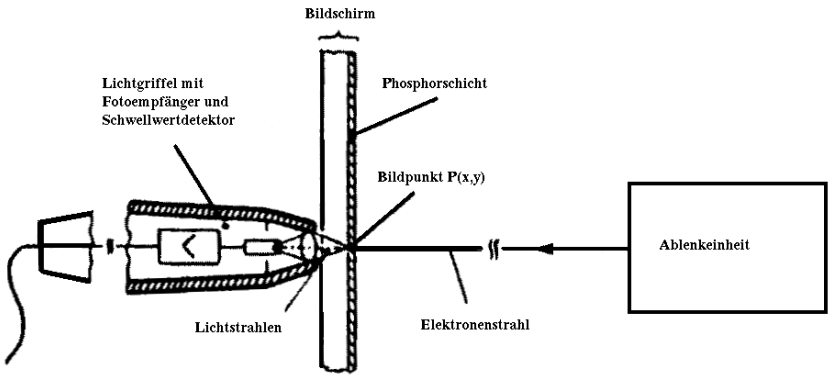


Abbildung 2.12: Prinzipieller Aufbau des Lightpen

Zum Funktionieren ist also ein Lichtimpuls erforderlich. Dieser wird bei allen Ausgabeverfahren mit zyklischer Bildwiederholung durch den vorüberlaufenden Schreibstrahl geliefert. Es leuchtet ein, daß ein Lichtgriffel nur dort ansprechen kann, wo auf dem Bildschirm eine Lichterscheinung auftritt.

Wegen der Unschärfe seiner Optik und der Schwierigkeit der exakten Positionie- Lichtgriffel erfaßt Bildelemente

rung auf dem Bildschirm wird der Lichtgriffel meist zum Picken größerer Bildelemente verwandt. Das Halten des Lichtgriffels vor einem senkrecht stehenden Bildschirm ist recht anstrengend. Deshalb wird der Lichtgriffel heute nur noch selten eingesetzt.

Ein Comeback des totgesagten Griffels kündigt sich allerdings mit dem Erscheinen von Notebook Computern an, die als Eingabeeinheit einen speziellen Stift besitzen, mit dem direkt auf den Schirm “geschrieben” werden kann. Der Rechner ist mit Software ausgestattet, die handschriftliche Eingaben in ASCII übersetzt [Car91]. Speziell entwickelte graphische Benutzeroberflächen erlauben auch Computerlaien ein intuitives Arbeiten mit solchen Geräten.

2.3.2 Das Tablett

Eines der wichtigsten graphischen Eingabegeräte ist das graphische Tablett. Der Benutzer kann mit ihm graphische Information in gewohnter Weise (wie bei Verwendung von Papier und Bleistift) zum Rechner übertragen.

Tablett erfaßt Positionen

Die *Position* eines Stiftes wird innerhalb einer rechteckigen Fläche zweihundert–bis fünfhundertmal pro Sekunde bestimmt. Damit ist sichergestellt, daß schnelle Bewegungen des Stiftes ebenfalls exakt erfaßt werden können. Die bei so hohen Meßraten anfallenden Datenmengen werden allerdings in den meisten Fällen gleich weiterverarbeitet, um eine Datenreduktion zu erreichen. Das am häufigsten angewandte Verfahren überprüft, ob sich ein neues Koordinatenpaar gegenüber dem zuletzt gemessenen um einen bestimmten Mindestbetrag verändert hat. Erst wenn das der Fall ist, wird dieses Wertepaar berücksichtigt.

Mit fallenden Hardwarekosten werden auch online–Zeichenerkenner für diese Aufgabe attraktiv. Sie sind in der Lage, den Entstehungsvorgang eines Zeichens zur Optimierung des Erkennungsalgorithmus zu nutzen. Ihre Bedeutung liegt allerdings mehr in der Verbesserung der Mensch-Maschine–Kommunikation.

Es gibt mehrere Verfahren zur Bestimmung der Position eines Stiftes auf einem Tablett. Hier soll nur das magnetostriktive Tablett besprochen werden.

2.3.2.1 Das magnetostriktive Tablett

Digitalisierer mit magnetostriktiver Kopplung benutzen ferromagnetische Drähte als Signalträger, die unter dem Einfluß eines äußeren magnetischen Feldes ihre Form geringfügig verändern (Magnetostriktion, Joule–Effekt). Im Tablett ist ein Kreuzleitersystem aus Stahldrähten eingebettet (Bild 2.13).

Ein zur Richtung der Stahldrähte senkrecht angelegtes Magnetfeld, das durch Sendespulen am Rande des Tabletts hervorgerufen wird, erzeugt eine Längenänderung der Stahldrähte. Diese Längenänderung pflanzt sich als mechanische Spannungswelle mit einer Geschwindigkeit von ca. 5000 m/sec längs des Drahtes fort. Passiert die Welle die im Stift befindliche Empfängerspule, so erzeugt die mit der

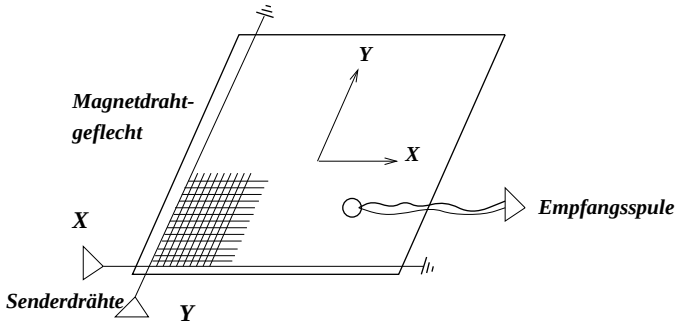


Abbildung 2.13: Digitalisierer mit magnetostriktiver Kopplung

Welle verknüpfte Flußänderung einen Spannungsimpuls im Stift. Die Laufzeit der Welle ist proportional zum Abstand des Stifts von der Sendespule am Rande des Tabletts (Bild 2.14).

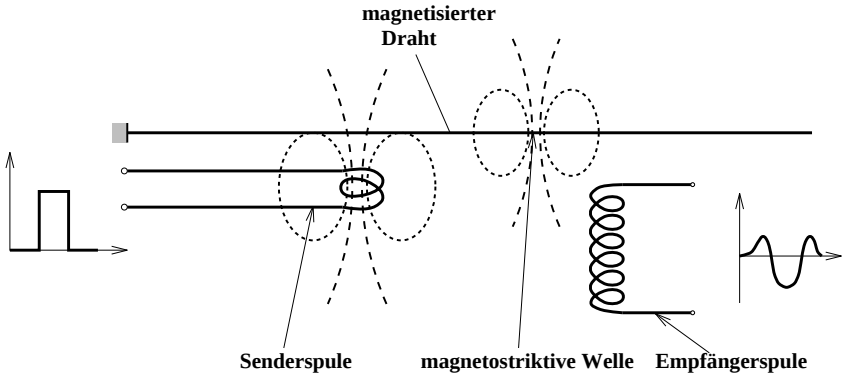


Abbildung 2.14: Prinzip der magnetostriktiven Übertragung

Da die Entfernung immer längs der Drähte gemessen wird, ist es nicht notwendig, daß die Drähte absolut parallel verlaufen. Ebensowenig müssen sie so dicht gespannt sein, wie dies aus der Auflösung des Gerätes zu schließen wäre. Es genügt vielmehr, sie in einem Abstand von 2 bis 3 mm zu spannen, damit überall auf dem Tablett eine ausreichende Flußänderung gewährleistet ist. Dieses Prinzip hat sich bezüglich Genauigkeit (0,01 mm) und Robustheit im Betrieb sehr bewährt und wird deshalb z.Zt. in den meisten Tabletts angewandt.

2.3.3 Der Scanner

Scanner dienen der direkten Eingabe von Strichzeichnungen oder auch Halbtonbildern in den Computer. Für niedrige Auflösungen kann eine gewöhnliche Fernsehkamera mit nachgeschaltetem Analog-Digital-Wandler verwendet werden. Für höhere Auflösungen (z.B. 4800 x 7200 Pixel) werden CCD-Scanner (Charged Coupled Device) eingesetzt, die eine CCD-Zeile enthalten, auf die ein Streifen

des abzutastenden Bildes optisch abgebildet wird. Nach dem Auslesen wird die CCD–Zeile oder die Vorlage mechanisch weiterbewegt, bis das ganze Bild eingelesen ist. Mehrfarbiges Einlesen ist dadurch möglich, daß verschiedenfarbige Lichtquellen zur Beleuchtung der Vorlage verwendet werden. Die Technologie ist durch den Einsatz in Faxgeräten sehr verbreitet und deshalb auch preisgünstig erhältlich.

2.3.4 Indirekt graphische Eingabegeräte

Indirekt graphische Eingabegeräte bedienen sich meist des Cursors, eines speziellen Zeichens auf dem Bildschirm, um dem Benutzer die Kontrolle über seine Aktionen zu gestatten. Man kann sie demnach auch als Cursor–Positionierungsgeräte bezeichnen.

Es gibt drei verschiedene Methoden der Cursorpositionierung: Die statisch absolute, die statisch relative und die dynamische Positionierung. Im ersten Fall liefert das Eingabegerät die Koordinaten des Punktes, der durch die Stellung des Eingabegerätes festgelegt wird, im zweiten Fall ein Koordinateninkrement entsprechend einer Stellungsänderung des Eingabegerätes, und im dritten Fall sorgt eine dynamische Folge von Punktkoordinaten für eine Bewegung des Cursors, deren Richtung und Geschwindigkeit durch die Stellung des Eingabegerätes gegeben ist.

2.3.4.1 Joystick

Der Joystick ist ein senkrecht stehender Hebel, der am unteren Ende kardanisch gelagert ist und von Federn in der Mitten–Ruhestellung gehalten wird. Da sich durch diese Lagerung sein oberes Ende nach links, rechts, vorn und hinten um wählbare Beträge bewegen läßt, ist er ein idealer “Steuerknüppel” für den Cursor. Die wichtigste Betriebsart ist dynamisch, jedoch läßt sich für sehr schnelle Positionierung unter Verzicht auf Genauigkeit auch die statisch absolute Betriebsart verwenden. Die Bewegung des Joysticks wird auf zwei Potentiometer, die der x– und y–Richtung entsprechen, übertragen und steht dann als Spannung für jede Koordinate zur Verfügung. Eine sehr einfache und billige Ausführung verwendet einen fest eingespannten Stab, an dem Dehnungsmeßstreifen angebracht sind.

2.3.4.2 Maus

Die Maus ist das wohl am häufigsten eingesetzte *2D*-Eingabegerät. Die Relativbewegung der Maus gegenüber der Arbeitsfläche ist die entscheidende Meßgröße. Diese Bewegung kann entweder mechanisch oder auch optisch aufgenommen werden. Mechanische Mäuse besitzen auf der Unterseite eine Rollkugel, durch die die Bewegung an optische Sensoren weitergegeben wird. Optische Mäuse benötigen eine speziell gemusterte Unterlage, die die von Leuchtdioden auf der Unterseite der Maus ausgesandten Strahlen reflektiert. Bei Bewegungen der Maus

ergeben sich typische Impulsfolgen, die von der Elektronik ausgewertet werden.

Da die Maus aufgenommen und an anderer Stelle wieder abgesetzt werden kann, kommt nur die relative Positionierung als Betriebsart in Frage. Um Platz zu sparen, wird meist eine dynamisch angepaßte Auflösung benutzt (hat nichts mit dynamischer Positionierung zu tun), d.h. eine gleich große Bewegung der Maus wird in eine größere Cursorbewegung umgesetzt, je schneller sie erfolgt. Dadurch kann, ohne den Unterarm vom Tisch zu nehmen oder gar die Maus abzusetzen, der Cursor sehr präzise geführt und trotzdem der ganze Bildschirm überstrichen werden.

2.3.4.3 Rollkugel

Die Rollkugel kann als eine “auf dem Rücken liegende Maus” betrachtet werden. Das Gerät steht auf dem Tisch und der Benutzer bewegt die Kugel mit seiner Handfläche. Um eine ruhige Bewegung zu erreichen, ist die Masse der Kugel relativ hoch, was große und schnelle Positionsänderungen erschwert. Gegenüber der Maus sind bei der Rollkugel alle drei Betriebsarten sinnvoll.

2.3.4.4 Touchpad, MousePoint

Ein Touchpad ist ein stationäres Zeigergerät, welches hauptsächlich für Laptops genutzt wird. Touchpads bieten eine kleine, flache Oberfläche, über die man seinen Finger gleiten läßt, indem man die gleiche Bewegung ausführt, als würde man eine Maus bedienen. Sie wurden ursprünglich entwickelt, um eine natürlichere und intuitivere Verbindung zum Computer zu bieten, als dies mit der Maus möglich ist.

Touchpads nutzen das Prinzip der Kopplungsladung und benötigen einen führenden Aufdruckspunkt, wie z.B. einen Finger. Sie beinhalten ein zweischichtiges Elektrodengitter, welches mit einem integrierten Kreislauf unterhalb des Pads montiert ist. Die obere Ebene beinhaltet vertikale Elektrodenbahnen, während die untere aus horizontalen Elektrodenbahnen zusammengesetzt ist. Die Ladung von jeder der horizontalen Elektroden zu jeder der vertikalen Elektroden wird über den integrierten Kreislauf gemessen. Ein Finger nahe dem Schnittpunkt zweier Elektroden ändert die Ladung zwischen diesen, da ein Finger andere leitende Eigenschaften besitzt als die Luft. Auf der Basis solcher Änderungen an verschiedenen Orten kann die Position des Fingers präzise festgelegt werden.

Ein MousePoint ist ein ’Mini-Joystick’, der sich zwischen den Tasten der Laptop-Tastatur befindet. Er wird mittels eines Fingers gesteuert.

2.3.4.5 Potentiometer

Potentiometer mit Analog-/Digital–Wandler erlauben eine schnelle Eingabe genauer Zahlenwerte, wobei eine Kontrolle über eine Wertanzeige auf dem Bild- Eingabe von Zahlenwerten

schirm geschieht. Drehpotentiometer und Schiebepotentiometer werden gleichermaßen eingesetzt. Aus Benutzersicht eignen sich zur Eingabe von Winkelwerten besonders die Drehpotentiometer.

2.3.5 Mehrdimensionale Eingabe

Für die Manipulation von räumlichen Objekten oder auch die Steuerung von Simulationen im Raum werden drei oder mehrdimensionale Eingabegeräte benötigt.

2.3.5.1 Spaceball

Der Spaceball von Spatial Systems (Bild 2.15) besteht aus einer fest montierten Kugel, die bequem mit der Hand erfaßt werden kann. Gemessen werden Drehmomente und Kräfte (in je drei Dimensionen), die auftreten, wenn versucht wird, die Kugel zu verschieben oder zu verdrehen. Durch die Kraftausübung bewegt sich die Kugel allerdings praktisch nicht, so daß eine dynamische Positionierung angewendet werden muß.

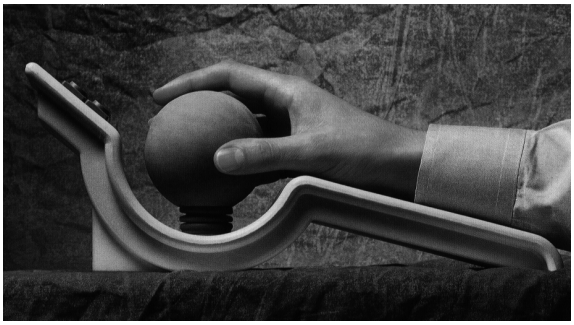


Abbildung 2.15: Spaceball

2.3.5.2 Polhemus 3Space Tracker

Der Polhemus 3Space Tracker [Egl90] erlaubt, Position und Lage im Raum zu bestimmen. Dies geschieht durch elektromagnetische Wellen, die von drei senkrecht zueinander stehenden Senderspulen (Bild 2.16) ausgesandt werden. Drei entsprechend aufgebaute Empfängerspulen empfangen die ausgestrahlten Impulse, deren Intensität ein Maß für die Entfernung von den Senderspulen ist. Das Verhältnis der Intensitäten dient dazu, die Ausrichtung zu bestimmen. Wird ein solcher Detektor in einen Stift eingebaut, kann er als “dreidimensionaler” Bleistift zum 3D-Zeichnen oder Abtasten von 3D-Objekten dienen.

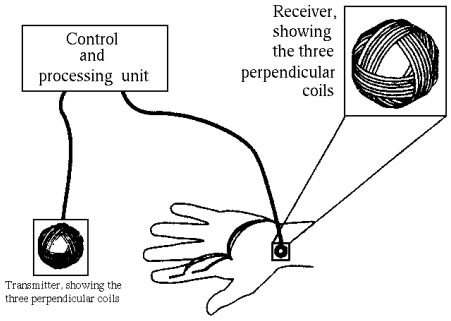


Abbildung 2.16: Der Polhemus 3Space Tracker

2.3.5.3 DataGlove

Der DataGlove von VPL Research [Egl90] dient dazu, die Stellung der Hand im Raum sowie die Fingerbewegungen aufzunehmen. Es handelt sich dabei um einen Handschuh, der an den Stellen, an denen eine Veränderung der Gelenkstellung festgestellt werden soll, mit dünnen Glasfasern beklebt ist, aus denen je nach Stärke der Richtungsänderung Licht austritt (Bild 2.17).

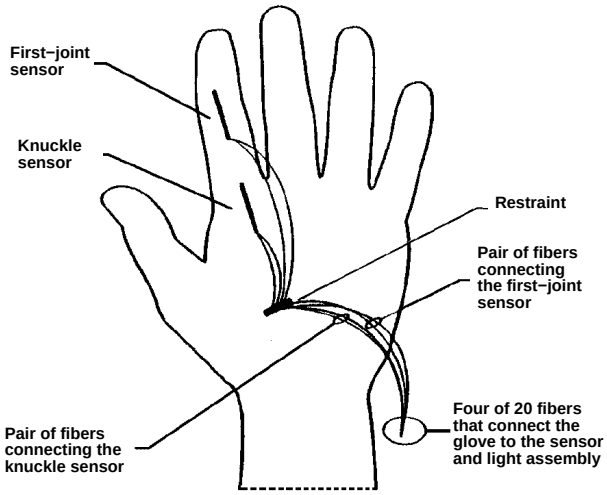


Abbildung 2.17: Der DataGlove von VPL Research

Der Lichtverlust in den Fasern wird durch Phototransistoren gemessen und von der Elektronik ausgewertet. Die Stellung der Hand im Raum wird mit einem Polhemus 3Space Tracker festgestellt. Der DataGlove wird z.B. als Eingabeeinheit für sogenannte virtuelle Realitäten eingesetzt.

2.3.6 Spracheingabe

Die automatische Spracherkennung hat sich in den letzten Jahren, besonders unter dem Einfluß fallender Hardwarekosten, schnell weiterentwickelt. Zuverlässige und preiswerte Systeme sind jetzt erhältlich und können am interaktiven graphischen Arbeitsplatz zur Verbesserung der Mensch-Maschine–Kommunikation eingesetzt werden. Dabei handelt es sich um Geräte, die bei begrenztem Vokabular und abhängig vom Sprecher sicher Einzelwörter erkennen können. Am graphischen Arbeitsplatz kann ein solches System den Benutzer von der Kommandoeingabe über Funktionstastatur oder Menü befreien. Dies ist insbesondere bei Anwendungen mit dem graphischen Tablett interessant.

2.4 Bildrechner (Graphics Display System)

Im folgenden werden wir unter Bildrechner den Teil eines Rechners verstehen, der für die GDV zuständig ist und aus einem “Allzweckrechner” einen leistungsfähigen Rechner für komplexe Aufgaben der GDV macht. Da graphische Benutzungsoberflächen inzwischen zum Standard eines jeden Rechners gehören, enthält jeder “Allzweckrechner” auch einen Bildrechner in Form von Graphikkarten, die sich jedoch in ihrer Leistungsfähigkeit erheblich unterscheiden.

Definition der DPU

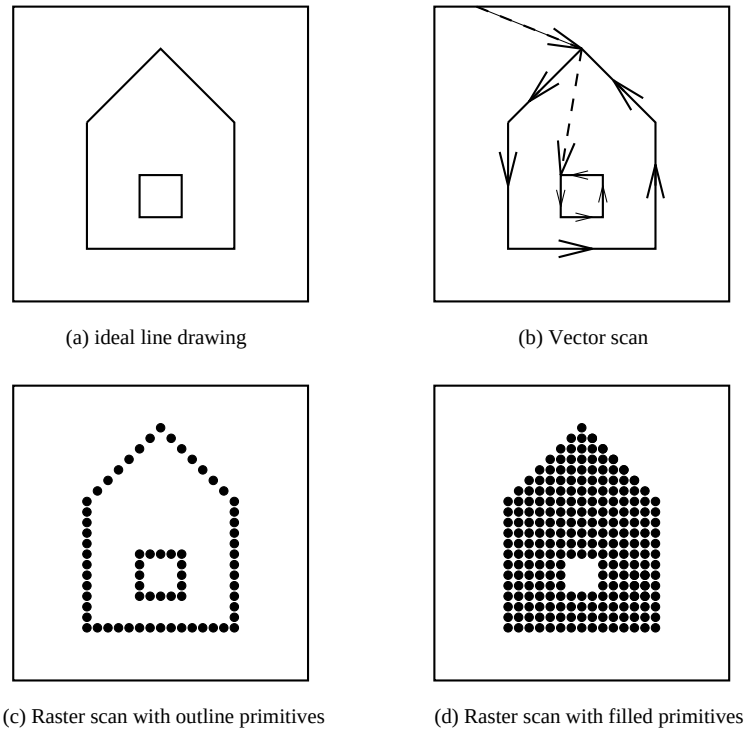
Der wichtigste und komplizierteste Teil eines Bildrechners ist der sogenannte Displayprozessor (DPU=Display Processing Unit). Seine Aufgabe besteht darin, die Bilddefinition des Anwendungsprogramms so aufzubereiten, daß auf dem nachgeschalteten Ausgabegerät (Display) das gewünschte Bild erscheint. Diese Aufgabe ist sehr rechenintensiv und speziell und würde den normalen Betrieb der CPU stark beeinträchtigen.

In Abhängigkeit von den Forderungen der Anwendung und den eingesetzten Ausgabegeräten wird der Displayprozessor DPU unterschiedlich komplex sein und nach verschiedenen Prinzipien arbeiten. Das Prinzip des Bildaufbaus läßt zwei Klassen von DPUs unterscheiden (Bild 2.18), Vektorgeräte und Rastergeräte.

Unterschied zwischen Vektor- und Rastergerät

Aus Bild 2.18 ist zu erkennen, daß bei Vektorgeräten die graphischen Objekte (Linien, Kreise etc.) aus der Bilddefinition direkt auf den Bildschirm gezeichnet werden. Bei Rastergeräten müssen die Objekte dagegen zuerst gerastert und in einem Bildpunktspeicher abgelegt werden (s.u.). Beim Vektorgerät geschieht die zur Aufrechterhaltung eines Bildes notwendige Bildwiederholung durch periodisches Auswerten der Bilddefinition, was die Möglichkeit zu dynamischen Darstellungen eröffnet. Beim Rastergerät hingegen wird die Bildwiederholung durch periodisches Auslesen des Bildpunktspeichers realisiert [FvDFH90].

Vektorgeräte wurden seit Mitte der sechziger Jahre gebaut und waren in den siebziger Jahren der dominierende Gerätetyp. Heute spielen sie nur noch in Simulatoren eine Rolle, bei denen Echtzeit das wichtigste Kriterium ist und vektorgraphische Darstellungen akzeptabel sind. In allen anderen Bereichen werden Rastergraphikgeräte eingesetzt. Ihre hervorstechendsten Eigenschaften sind:



(a) ideal line drawing (b) Vector scan

(c) Raster scan with outline primitives (d) Raster scan with filled primitives

Abbildung 2.18: Vektor– und Rasterverfahren

– Das Ausgabegerät erfüllt höchste Qualitätsanforderungen bei niedrigen Kosten.

– Die vorhandene volle Farbtüchtigkeit und die Fähigkeit zur Flächendarstellung ist für viele Anwendungen erforderlich.

– Die mögliche Fernsehkompatibilität erlaubt das Mischen synthetischer und natürlicher Bilder, die Nutzung öffentlicher Kommunikationsdienste (Bildschirmtext, Videotext, HDTV etc.) und Multi–Media–Anwendungen.

– Interaktive Computergraphik und Bildverarbeitung können mit einem System durchgeführt werden.

– Die Komplexität flimmerfrei darstellbarer Bilder ist praktisch unbegrenzt.

– Das Fernsehgerät ist jedermann als Gebrauchsgegenstand vertraut.

Bild 2.19 zeigt die Hauptkomponenten eines Rasterdisplay(system)s in einem funktionalen Modell. Eine Realisierung erfordert je nach gewünschter Systemleistung unterschiedliche Architekturen.

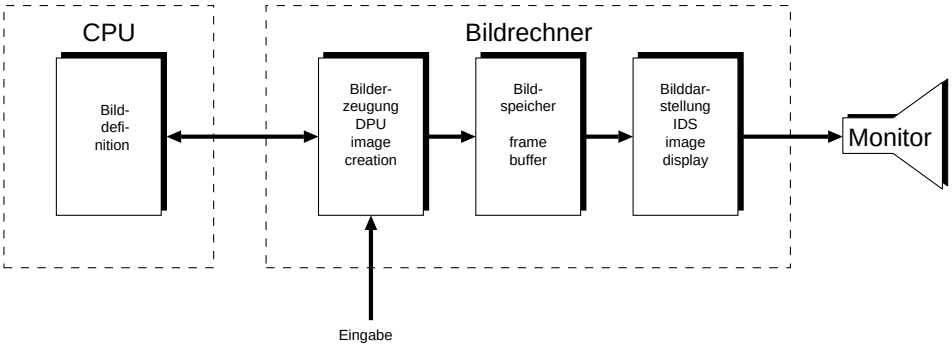


Abbildung 2.19: Hauptkomponenten eines Rasterdisplay(system)s

2.4.1 Rasterdisplaysysteme

2.4.1.1 Das einfachste Rasterdisplaysystem

Graphikfunktionen in Software realisiert

Werden die Funktionen der DPU durch Software in der CPU realisiert, so erhält man das gerätetechnisch einfachste Rasterdisplaysystem (siehe Bild 2.20).

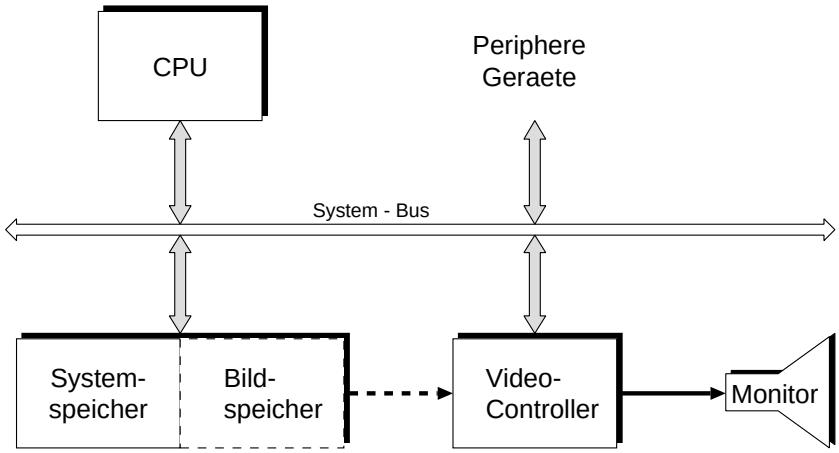


Abbildung 2.20: Das einfachste Rasterdisplaysystem

Der Bildspeicher (frame buffer) ist Teil des Arbeitsspeichers (system memory). Ist hierfür ein fester Adreßbereich mit zweitem Zugriffspfad reserviert, so kann der Videocontroller (IDS=Image Display System), ohne den Systembus zu belasten, die Bildinformation direkt auslesen.

Nur ein großer Adreßraum im Speicher

Ein großer Vorteil dieser Architektur ist der gemeinsame Adreßraum für Anwendungsprogramme, Displayprogramme und Bildspeicher. Sie erlaubt dem Programmierer größte Flexibilität bei minimalen Hardwarekosten. Dies wird erkauft durch lange Bildaufbauzeiten, weil die Bildrasterung per Software relativ langsam ist. Zusätzlich wird ein großer Adreßbereich (20 bit) für den Bildspeicher beansprucht, was z.B. mit einer 64 bit CPU gelöst werden kann. Schnellere Systeme

sind nur durch zusätzliche Hardware zu erreichen.

2.4.1.2 Rasterdisplaysystem mit integrierter DPU

Displayprozessor für graphische Algorithmen

Zur Beschleunigung graphikspezifischer Operationen kann in die Architektur von Bild 2.20 ein Displayprozessor (DPU) ergänzt werden. Die Vorteile des einheitlichen Adreßraumes bleiben erhalten: CPU, DPU und IDS sind am Systembus und haben so Zugriff zum Systemspeicher (Bild 2.21)

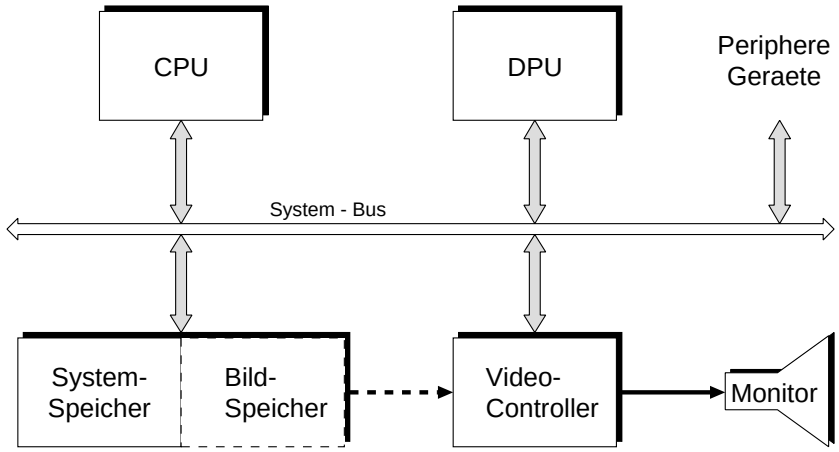


Abbildung 2.21: Rasterdisplaysystem mit integrierter DPU

Konkurrenz um den Speicherzugriff

Der einheitliche Adreßraum erlaubt elegante Anwendungsprogramme und ist einfach zu überschauen. Bei Echtzeitanwendungen können jedoch Probleme mit der Konkurrenz der einzelnen Komponenten um den Speicherzugriff auftreten. Dies wird z.T. durch feste Widmung eines Speicherbereichs als Bildspeicher gelöst (gestrichelte Variante in Bild 2.21).

2.4.1.3 Rasterdisplaysystem mit peripherer DPU

Bild 2.22 zeigt die typische Anordnung eines Rasterdisplays mit peripherem Displayprozessor. Hier hat die DPU einen eigenen Arbeitsspeicher mit separatem Bildspeicher.

Damit existieren drei Speicherbereiche: Im Hauptspeicher (system memory) liegen die Anwendungsdaten, das Anwendungsprogramm, die graphische Software und das Betriebssystem. Im Speicher der DPU liegen dementsprechend graphische Daten und Programme, z.B. für die rechenintensive Rasterung des Bildes. Im Bildspeicher (frame buffer) ist das Rasterbild punktweise gespeichert. Dieser Speicher wird mit 60 Hz oder mehr zur Bildwiederholung ausgelesen.

Gegenüber den anderen Architekturen ist hier die funktionale Trennung der Kom-

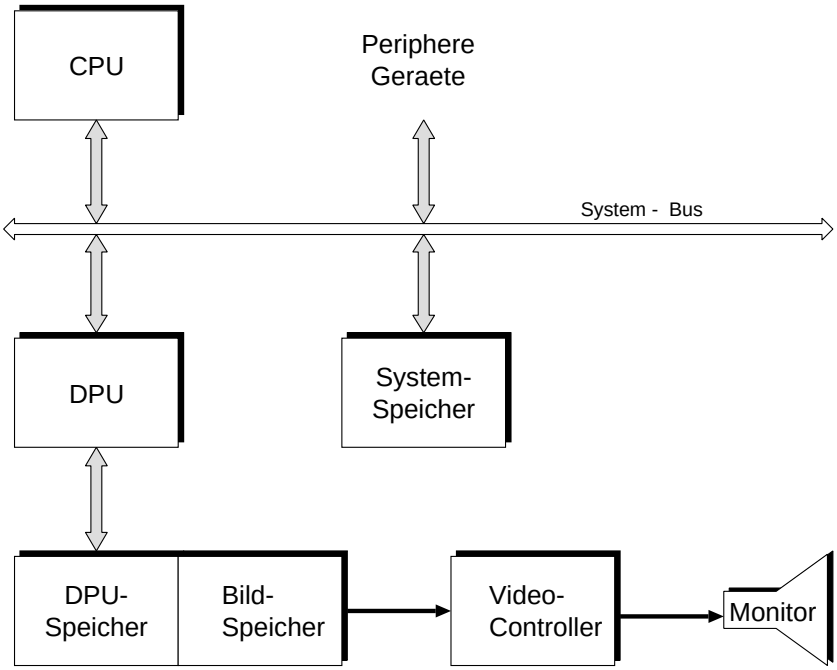


Abbildung 2.22: Rasterdisplaysystem mit peripherer DPU

ponenten entsprechend Bild 2.19 am konsequentesten realisiert. Das System läßt sich deshalb am leichtesten entsprechend den Anforderungen einer Anwendung maßschneidern. Nachteilig ist der für Echtzeitanwendungen notwendige, aber langsame Datentransfer zwischen Hauptspeicher und Bildspeicher. Viele Systeme für Anwendungen mit mittleren Anforderungen haben diese Architektur.

Nach dieser Darstellung der wichtigsten Architekturen werden die einzelnen Komponenten betrachtet:

2.4.2 Bilddarstellung (image display)

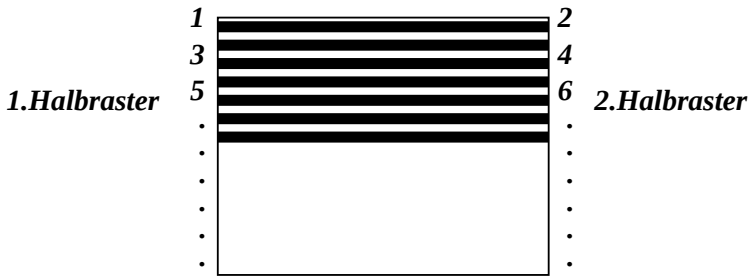


Abbildung 2.23: Zeilensprungverfahren des Fernsehens

Bildausgabe ist an exaktes Zeitraster gebunden

Der Bilddarstellungsteil (IDS), oft auch image display, Video– oder CRT–Controller genannt, erzeugt durch zeilenweises Schreiben auf dem Videomonitor ein

stehendes, flimmerfreies Bild. Dazu ist ein exakter Zeitablauf entsprechend der Fernsehnorm oder der vorgegebenen Daten (Videobandbreite, Zeilen– und Bildfrequenz usw.) des eingesetzten Monitors einzuhalten. Manche Systeme verwenden, wie das Fernsehen, zur Reduzierung der erforderlichen Bandbreiten das sogenannte Zeilensprungverfahren (Interlacing), bei dem zwei Bilder mit halbierter Vertikalauflösung nacheinander und kammartig ineinander greifend geschrieben werden (Bild 2.23). Die Vollbildfrequenz wird auf diese Weise halbiert. Dieses Verfahren arbeitet unter der Annahme, daß der Bildinhalt benachbarter Zeilen kaum Unterschiede aufweist, zufriedenstellend (Fernsehen). Bei synthetisch erzeugten Bildern in graphischen Systemen ist diese Voraussetzung aber oft nicht erfüllt, was sich dann störend als Flimmern bemerkbar macht (z.B. Vektor genau auf einer Zeile). Dies gilt insbesondere dann, wenn am graphischen Arbeitsplatz der Betrachtungsabstand als Funktion der Bildauflösung nicht eingehalten werden kann (Bild 2.24).

Zusammenhang zwischen erforderlicher Zeilenzahl und Betrachtungsgeometrie

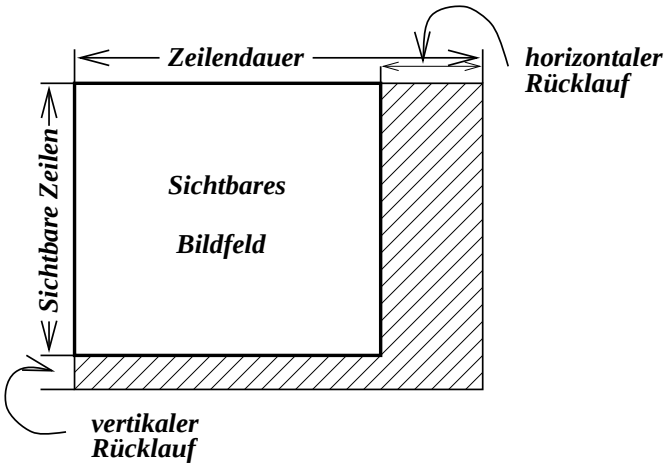


Abbildung 2.24: Zur Geometrie der Betrachtung eines Fernsehbildes

Optimale Betrachtungsverhältnisse sind dann gegeben, wenn der Blickwinkel für eine Zeilenbreite kleiner oder gleich der Sehschärfe des Auges ist, die bei Fernsehrandbedingungen ca. 1,5 Bogenminuten beträgt [The73]. Aus der Betrachtungsgeometrie (Bild 2.24) erhält man

$$z = \frac{v}{e}\frac{1}{\gamma} \quad \text{mit} \quad \gamma = 4 \cdot 10^{-4}\text{rad}$$

als Zusammenhang zwischen Zeilenanzahl z, Bildhöhe v und Betrachtungsabstand e.

Die meisten Rastergraphiksysteme arbeiten zur Verbesserung der Bildqualität und, damit zusammenhängend, zur Vermeidung schneller Ermüdung mit Vollbild. Deshalb werden im folgenden die speziellen Probleme der Halbbildtechnik nicht behandelt.

Die nutzbare Zeit zur Darstellung des Bildes auf dem Monitor wird durch Bild– und Zeilendauer (bzw. Bild– und Zeilenfrequenz) sowie die Zeit für horizontalen und vertikalen Strahlrücklauf bestimmt (Bild 2.25).

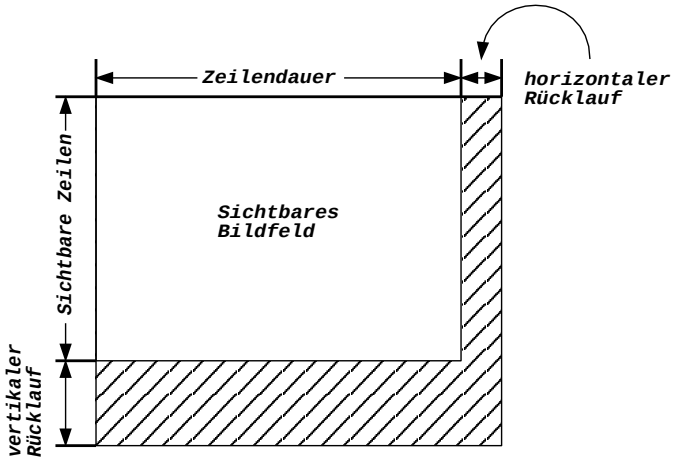


Abbildung 2.25: Verluste durch endliche Rücklaufzeiten

Beispielsweise ist bei der 625 Zeilennorm des Fernsehens die Zeilendauer 64 μs, wovon ca. 20 % für den horizontalen Rücklauf verlorengehen. Von der Bilddauer muß die Zeit für den vertikalen Rücklauf, in diesem Fall die Dauer von 25 Zeilen, abgezogen werden. Bild 2.26 gibt die nutzbare Zeit für verschiedene Displayformate an.

Visible Area Pixel x Lines	Refresh Rate in Hz	Inter-Laced?	Vertical Retrace Time in μs	Horizontal Retrace Time in μs	Total Line Time in μs	Pixel Time in ns
512 x 485	30	YES	1271	10.9	63.56	102.8
640 x 485	30	YES	1271	10.9	63.56	82.3
512 x 512	30	YES	1203	10.9	60.4	96.7
1024 x 768	30	YES	1250[a]	7[a]	40.1	32.37
1024 x 1024	30	YES	1250	7	30.11	22.57
1280 x 960	30	YES	1250	7	32.12	19.62
1280 x 1024	30	YES	1250	7	30.11	18.06
512 x 485	60	NO	1250	7	31.79	18.06
640 x 485	60	NO	1250	7	31.79	38.73
512 x 512	60	NO	1250	7	30.11	45.14
1024 x 768	60	NO	600[b]	4[b]	20.92	16.52
1024 x 1024	60	NO	600	4	15.69	11.42
1280 x 960	60	NO	600	4	16.74	9.95
1280 x 1024	60	NO	600	4	15.69	9.13

(a) Nominal RS-343-A specifications

(b) Typical high-preformance monitor specifications

Abbildung 2.26: Zeitverhältnisse bei verschiedenen Displayformaten

Bildpunktdauer als Funktion des Bildformats

Eine entscheidende Einflußgröße für den Entwurf nicht nur des IDS, sondern des gesamten Rastersystems ist die Bildpunktdauer t. Sie läßt sich berechnen aus

$$t = \frac{\left[\dfrac{\dfrac{1}{\text{Bildfrequenz}} - \text{vertik. Rücklaufzeit}}{\text{Anz. der sichtbaren Zeilen pro Bild}}\right] - \text{horizont. Rücklaufzeit}}{\text{Anz. der sichtbaren Bildpunkte pro Zeile}}$$

Displayformate

Viele Sichtgeräte haben eine rechteckige Darstellungsfläche. Beim Fernsehen z.B. ist das Seitenverhältnis $4:3$. Ein Bild mit 512×512 Bildpunkten wird dann bei voller Ausnutzung der Darstellungsfläche mit einem Seitenverhältnis von $4:3$ dargestellt, d.h. die Bildpunkte sind rechteckig. Dieser Effekt ist häufig unerwünscht. Zwei Möglichkeiten der Abhilfe werden z.Zt. angewandt:

– Das Verhältnis der adressierbaren Punkte entspricht dem Seitenverhältnis, z.B. 680×480 oder 1280×960. Damit wird die Bildschirmfläche ganz ausgenutzt, und der Maßstab ist in beiden Koordinatenrichtungen gleich.

– Die horizontale Darstellungskapazität wird nicht voll ausgenutzt. Bildpunkte und Gesamtbildfläche sind quadratisch.

Nach diesen allgemeinen Darlegungen zum Rasterformat sind folgende Aufgaben des IDS zu erkennen:

1) Erzeugen der horizontalen (HSYNC) und vertikalen (VSYNC) Synchronimpulse für das entsprechende Bildformat (Die Ablenkspannungen werden im Monitor selbst erzeugt),

2) Adressierung und Auslesen des Bildspeichers,

3) Ansteuern des Monitors mit den entsprechenden Intensitäts– oder Farbwerten und dem Dunkelsignal für die horizontale und vertikale Austastlücke sowie Digital–Analog–Wandlung (DAC).

Hieraus ergibt sich das Blockschaltbild des IDS in Bild 2.27.

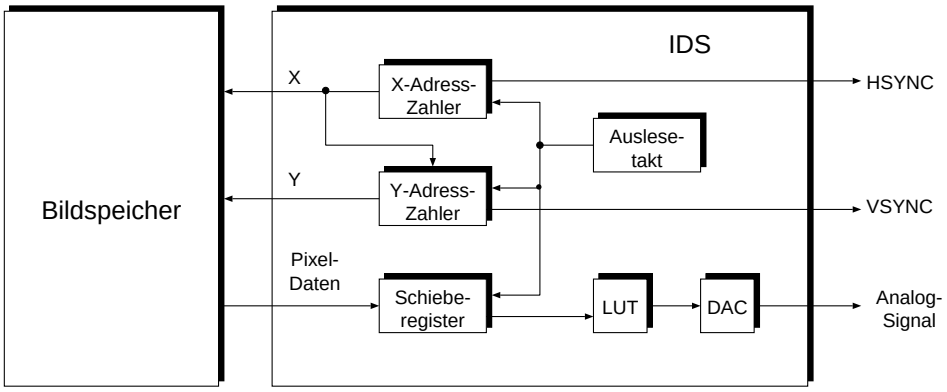


Abbildung 2.27: Blockschaltbild des einfachen IDS

Je nach Verhältnis zwischen Bildpunktdauer, Zykluszeit und Organisation des Bildspeichers muß mehr als ein Bildpunkt gleichzeitig ausgelesen und im IDS

in einem Schieberegister, das vom Bildpunkttakt gesteuert die Parallel–Serien–Wandlung vornimmt, zwischengespeichert werden (im folgenden wird manchmal statt Bildpunkt oder Bildelement auch picture element, pixel oder pel verwendet).

Bitmap

Das einfachste und gebräuchlichste Rasterdisplay besteht aus einem Pixelspeicher mit einer Auslesesteuerung wie in Bild 2.27 und realisiert eine 1 : 1 Zuordnung zwischen Speicher und Bildschirmadresse. Deshalb spricht man auch von Map–Display (Bild 2.28). Stehen 3 und mehr Bits pro Speicheradresse zur Verfügung, so können auch Farben dargestellt werden.

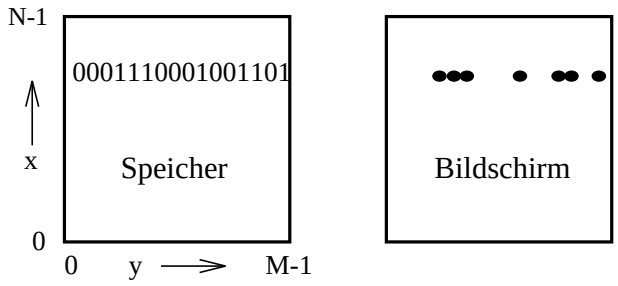


Abbildung 2.28: Einfachstes Rasterdisplay

Die meisten IDS steuern mit dem Pixelwert nicht direkt die Intensität oder Farbe, sondern interpretieren ihn als Adresse für eine Farbtafel (LUT = Look–up–table).

Farbtafel LUT

Die Tafeleinträge definieren dann die auszugebende Farbe (Bild 2.29). Die Vorteile der LUT sind:

1) Die Färbung des Bildes kann ohne Änderung des Bildwiederholspeicherinhaltes variiert werden, z.B. durch Umschalten zwischen verschiedenen LUT oder Neuladen einer beschreibbaren LUT. Wird die LUT mit schnellen Schreib–/Lesespeichern realisiert, so können Bewegtbildeffekte erzielt werden.

2) Die Anzahl der verfügbaren Farben kann erhöht werden, wenn man die Wortlänge in der Tafel gegenüber der Pixelwortlänge (Bits/Pixel) erhöht. Selbstverständlich wird aber die maximale Anzahl verschiedener, gleichzeitig darstellbarer Farben von der Pixelwortlänge bestimmt. Man spricht deshalb auch von der wählbaren Farbpalette aus der Gesamtzahl der verfügbaren Farben.

3) Filterfunktionen für Bildverarbeitungsaufgaben können realisiert werden. Hierzu gehört auch die Gamma-Korrektur oder die Interpretation der einzelnen Speicherebenen und/oder ihrer Kombinationen als Bilder.

Die Möglichkeiten zur unterschiedlichen Interpretation der Bildspeicherebenen und zur Erzeugung von Bewegtbildeffekten sollen etwas genauer betrachtet werden:

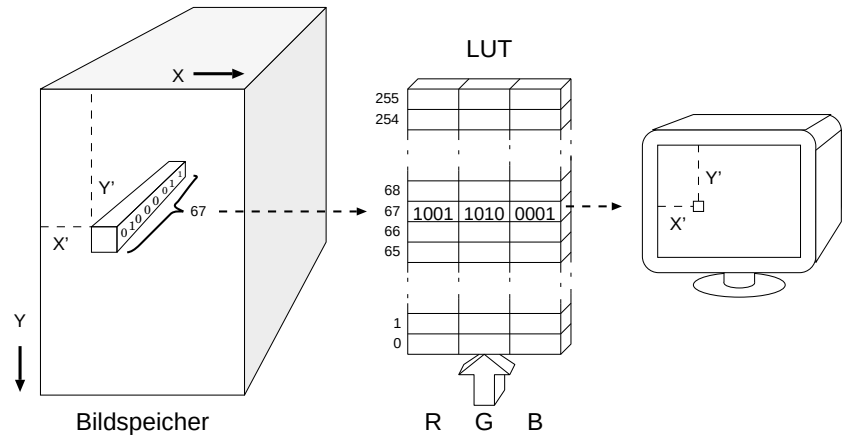


Abbildung 2.29: Farbdefinition über Farbtafel (R=rot, G=grün, B=blau)

2.4.2.1 Bitebenenextraktion

Sind *N* Bits pro Pixel vorhanden, so können z.B. *N* Binärbilder, N/3–RGB–Bilder etc. abgespeichert werden. Jedes dieser Bilder kann durch geeignetes Laden der LUT (wenn für die Anwendung erforderlich, auch mit dem Bildtakt nacheinander!) dargestellt werden. Für den Fall der Binärbilder muß die LUT unter all den Adressen, die an der entsprechenden Stelle eine 1 führen, “Weiß”, sonst “Schwarz” beinhalten. Auf ähnliche Weise kann man auch bei Grau– oder Farbbildern verfahren.

2.4.2.2 Bildspeicherebenen mit Prioritätszuordnung

Pixelwert			**Inhalt der LUT**	**Inhalt der LUT**
0	0	0	Hintergrundfarbe	weiß
0	0	1	Farbe von Bild C	blau
0	1	0	Farbe von Bild B	grün
0	1	1	Farbe von Bild B	dunkles blaugrün
1	0	0	Farbe von Bild A	rot
1	0	1	Farbe von Bild A	purpur
1	1	0	Farbe von Bild A	braun
1	1	1	Farbe von Bild A	schwarz
Bild:	A	B C	Priorität A > B > C	Transparenz

Abbildung 2.30: Verdeckungspriorität und Transparenz mittels LUT

Bei vielen Anwendungen kommen Bilder vor, deren Objekte sich in der Tiefe (Z–Koordinate) nicht durchdringen und denen deshalb bezüglich der gegenseitigen Verdeckungsmöglichkeiten eine feste Reihenfolge zugeordnet werden kann (z.B. Mehrebenenlayout für eine integrierte Schaltung, die Fenster eines “Window–

Managers" etc.). Diese Reihenfolge bzw. Priorität wird auf die Bitebenen übertragen und durch entsprechende Farben realisiert (z.B. höchste Priorität für das höchstwertige Bit etc.). Da es sich nicht um ein echtes räumliches (3D) Problem handelt, spricht man oft von $2\frac{1}{2}D$. Will man anstelle des Verdeckungseffektes Transparenz erzielen, muß man bei vorgegebener Priorität die entsprechenden Mischfarben in der LUT eintragen (Bild 2.30). Im Beispiel ist ABC=RGB.

$2\frac{1}{2}$D

2.4.2.3 Bewegtbildeffekte

Dynamik mit Farbtafeln

Die Zuordnung zwischen Pixelwert und Farbe kann bei jedem Bildwechsel geändert werden. Soll z.B. ein Punkt entlang einer Kurve wandern, so lädt man linear ansteigende Werte in die aufeinanderfolgenden Pixel dieser Kurve. Dem Pixelwert des Startpunkts ordnet man z.B. "weiß", den restlichen "schwarz" zu. Diese Zuordnung wird dann bei jedem Bildwechsel um eine Stelle verschoben.

Natürlich kann man auf diese Weise auch kompliziertere, farbige Objekte auf gestaltetem Hintergrund bewegen. Am einfachsten sind zyklische Abläufe realisierbar.

2.4.2.4 Bildtransformationen

Im IDS können neben den Farbtransformationen auch geometrische Transformationen in Echtzeit realisiert sein, ebenfalls ohne den Bildspeicherinhalt zu verändern. Dabei wird die starre 1 : 1 Zuordnung zwischen Bildpunktadressen im Bildspeicher und Bildpunktkoordinaten auf dem Bildschirm aufgehoben und eine Window–Viewport–Transformation durchgeführt (Bild 2.31), die im allgemeinen eine Skalierung, Translation und eventuelle Rotation beinhaltet. Für beliebige Rotationen sind allerdings aufwendige Filteroperationen zur Vermeidung von störenden Diskretisierungseffekten (aliasing) nötig, weshalb normalerweise nur um Vielfache von 90^0 gedreht werden kann. Die Skalierung beschränkt sich in der Regel auf ganzzahliges Vergrößern (Zooming) und wird durch entsprechende Pixelwiederholung in X– und Y–Richtung erzeugt. Zur Translation benötigt man Koordinatenregister und schnelle Komparatoren, die die aktuelle Strahlposition mit der gewünschten neuen Objektposition vergleichen und die Adressierung für den Bildspeicher steuern. Diese Art Adreßsteuerung wird auch für die 90^0–Rotation sowie die Ausschnittsbildung eingesetzt. Sollen diese Bildtransformationen auch individuell für die Bitebenen durchführbar sein, dann müssen für jede Ebene Koordinatenregister und Komparatoren vorhanden sein.

Transformationen ohne Bildspeicheränderung

Transformationen ohne Bildspeicheränderung (" on the fly ")

2.4.3 Bilderzeugung (image creation)

Der Bilderzeugungsteil, oft auch "image creation system" (ICS) genannt, enthält als wichtigste Komponente den Displayprozessor (DPU). Um seine Aufgaben und verschiedenen Realisierungsmöglichkeiten diskutieren zu können, betrachten wir

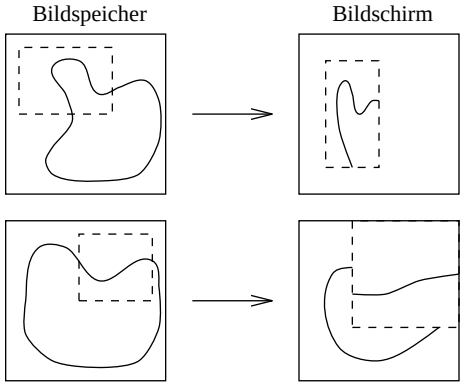


Abbildung 2.31: Transformation bei der Bildausgabe

im Vorgriff auf spätere Kapitel den gesamten Informationsfluß von der Bilddefinition im Anwendungsprogramm bis zum gerasterten Bild im Bildspeicher (Bild 2.32).

Die hierbei durchlaufene Folge von Bearbeitungsstufen wird häufig Ausgabe– oder Bildgenerierungspipeline genannt. In Analogie zum Photographieren spricht man auch von der synthetischen Kamera. Liegt die Szenenbeschreibung vor, werden zunächst – wie beim Photographieren – Kameraposition und Blickrichtung festgelegt und dann ein Bildausschnitt gewählt (clippen). Die perspektivische Transformation sorgt für die Erzeugung eines Tiefeneindrucks im zweidimensionalen Bild.

Mit der Verdeckungsrechnung werden an dieser Stelle des Bildes die Rückseiten der Objekte aus der Szenenbeschreibung entfernt. Die folgende Beleuchtungsrechnung nutzt Beleuchtungsparameter und Oberflächeneigenschaften zur Berechnung einer möglichst realistischen Farbgebung. Diese Verarbeitungsschritte werden als Geometrieverarbeitung bezeichnet und können in einem Geometrieprozessor realisiert sein, der dann als Teil des Displayprozessors betrachtet wird.

Geometrieverarbeitung

Geometrieprozessor

Die nachfolgende Stufe, der Displayprozessor, führt die Bildrasterung durch. Neben der eigentlichen Zerlegung der graphischen Primitiva in Rasterpunkte werden die Verdeckungsrechnung, die Beleuchtungsrechnung und die Bildglättung (anti–aliasing) durchgeführt und das Ergebnis im Bildspeicher abgelegt. Dabei konkurriert das Bilderzeugungssystem mit dem Darstellungssystem um den Zugriff zum Bildspeicher.

Die Architekturbetrachtungen am Beginn dieses Kapitels haben gezeigt (Bilder 2.21 bis 2.23), daß das Spektrum der Realisierungsmöglichkeiten für den Bilderzeugungsteil von der reinen Softwarelösung über den Spezialprozessor bis zum eigenständigen Rechner reicht.

Die Forderungen der Anwendung werden letztlich bestimmen, welche Variante gewählt wird. Hierzu ist es erforderlich, sich den zahlenmäßigen Zusammenhang zwischen gewünschter Leistungsfähigkeit und hierzu notwendiger Rechenkapazi-

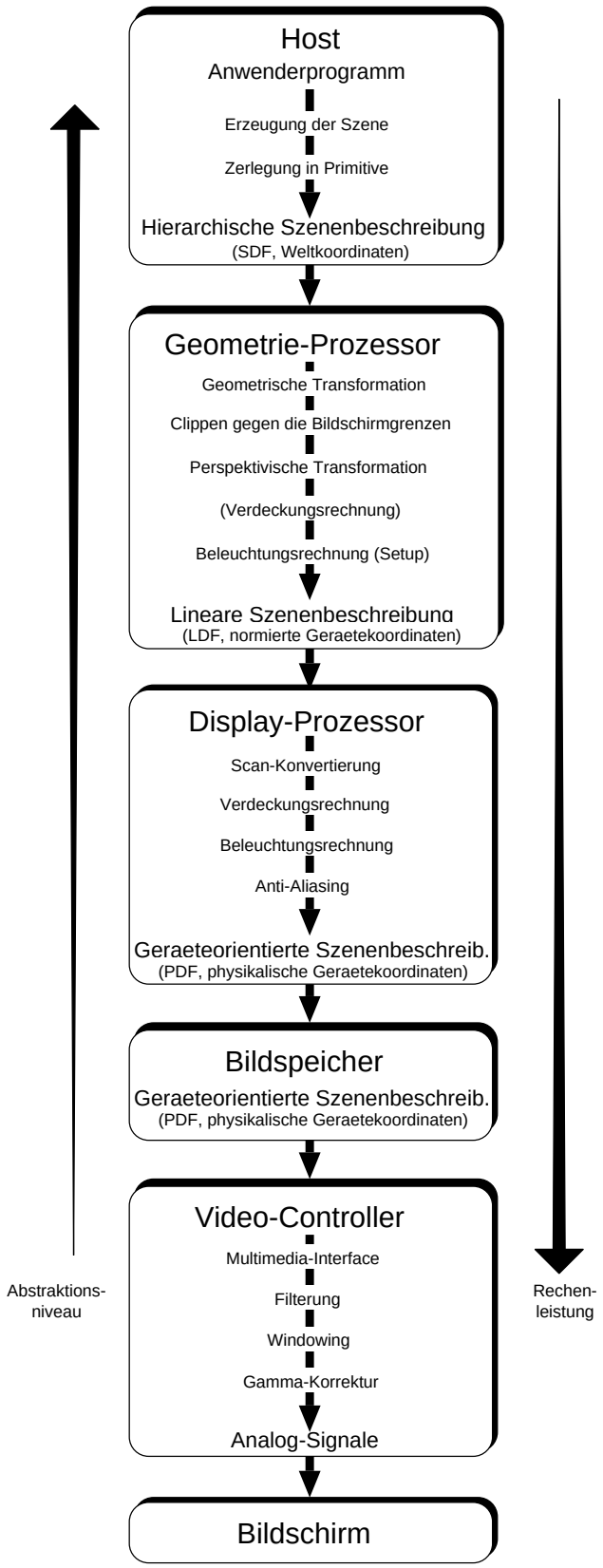


Abbildung 2.32: Die Bildgenerierungspipeline

tät und Übertragungsbandbreite klarzumachen.

Folgendes sehr vereinfachtes Beispiel kann das leisten; die Zahlenwerte sind als grobe Abschätzung zu verstehen. Noch nicht eingeführte Begriffe werden im Vorgriff auf das spätere Kapitel 7 über “Beleuchtungsmodelle” verwendet.

2.4.3.1 Graphiksystem geringerer Leistungsfähigkeit

- 100.000 Dreiecke pro Bild
- 1 Lichtquelle
- Interpolation von Farbwerten zwischen Eckpunkten (GOURAUD–Shading)
- Berechnung von 10 Bildern pro Sekunde
- Kein Anti–Aliasing

Folgende Abkürzungen werden verwendet:

FLOPS = Floating point operations per second
FXOPS = Fixed point operations per second
PDF = Physikalisches Display File
SDF = Strukturiertes Display File = Displayliste
LDF = Lineares Display File
Wort = 3 Bytes

2.4.3.2 Anforderungen an den Host-Rechner

Die CPU berechnet aus der Anwendung den Aufbau der Szene. Die dafür notwendige Rechenleistung ist stark von der Anwendung abhängig. Deshalb kann sie hier nicht abgeschätzt werden.

Neben der reinen Rechenleistung muß der Host auch eine genügend hohe Bandbreite aufweisen, um die anfallenden Daten an das Graphiksystem zur Aufrechterhaltung der Bildrate übertragen bzw. das SDF anlegen zu können. Die Spezifikation von 100.000 Dreiecken pro Bild erfordert die Übertragung von 300.000 Eckpunktdaten. Jeder Eckpunkt ist durch je drei Gleitkommazahlen für seine Koordinaten und den zugehörigen Normalenvektor (3 Komponenten) beschrieben. Der Normalenvektor wird u.a. für die Beleuchtungsrechnung benötigt. Zusätzlich müssen für jedes Dreieck noch die in der Beleuchtungsrechnung verwendeten Oberflächenparameter k_a, k_d, k_s und m übergeben werden (vgl. das spätere Kapitel 7 über “Beleuchtungsmodelle”).

2.4.3.3 Anforderungen an den Geometrie-Prozessor

Die Berechnungen im Geometrie-Prozessor werden in Gleitkommadarstellung durchgeführt. Im einzelnen sind die Aufgaben des Geometrie-Prozessors folgen-

de:

Transformation der Eckpunkte

Die Transformation erfolgt durch Multiplikation der homogenen Punktkoordinaten (Vektor mit vier Komponenten) mit einer $(4 \times 4)-$Transformationsmatrix (vgl. Kapitel 3 'Affine und perspektivische Abbildungen'). Dies erfordert für jeden Punkt 16 Multiplikationen und 12 Additionen von Gleitkommazahlen, insgesamt also 28 Gleitkommaoperationen (FLOP).

Transformation der Normalenvektoren

Auch die Transformation von Normalenvektoren geschieht durch die Multiplikation des Normalenvektors mit einer (4×4)-Matrix, der Transponierten der inversen Transformationsmatrix für die Eckpunkte. Dies erfordert wiederum 16 Multiplikationen und 12 Additionen (28 FLOP). Der transformierte Normalenvektor muß anschließend noch normiert werden, was mittels Division jeder Komponente durch die Länge des Vektors erfolgt. Die Berechnung der reziproken Quadratwurzel läßt sich mit 16 Gleitkommaoperationen abschätzen. Zusammen mit den 3 Multiplikationen und den 2 Additionen für die Summe der Quadrate sowie den 3 Multiplikationen mit den Komponenten des Vektors ergeben sich schließlich 24 Gleitkommaoperationen für die Normierung. Insgesamt sind zur Transformation eines Normalenvektors also 52 FLOP erforderlich.

Beleuchtungsrechnung

Die Berechnung des Beleuchtungsmodells gemäß dem Phongmodell (vgl. das spätere Kapitel 7 über "Beleuchtungsmodelle") erfolgt für jede der Farbkomponenten RGB. Dabei müssen jedoch die Skalarprodukte nur einmal berechnet werden. Für ein Skalarprodukt sind 3 Multiplikationen und 2 Additionen nötig. Die Potenzierung findet durch einen Tabellenzugriff statt, der ebenfalls mit einer Gleitkommaoperation veranschlagt wird. Somit sind für jede Lichtquelle 11 FLOP für die Skalarprodukte erforderlich. Weiterhin müssen für jede Farbkomponente pro Lichtquelle 4 Multiplikationen und 1 Addition durchgeführt werden. Dazu kommen noch 1 Multiplikation und 1 Addition zur Berücksichtigung der ambienten Beleuchtung. Insgesamt sind also $(2 + \mathrm{L} \times 26)$ FLOP pro Punkt notwendig, wobei L die Anzahl der Lichtquellen angibt.

Clippen

Der Aufwand beim Clippen mittels des Algorithmus nach Sutherland und Hodgman besteht bei sogenannten kanonischen Clipvolumen hauptsächlich in einfachen Tests jeden Punktes gegen die sechs Clip-Ebenen. Folglich sind pro Eckpunkt 6 Vergleiche (= 6 FLOP) auszuführen.

Projektion der Eckpunkte

Die Umwandlung von homogenen Koordinaten (vgl. Abschnitt 3.3 'Affine und perspektivische Abbildungen in Matrizenschreibweise') in 3D-Raumkoordinaten erfolgt durch Multiplikation der Komponenten *x*, *y* und *z* mit dem Kehrwert der homogenen Koordinate *w*. Für die Kehrwertbildung werde 1 FLOP veranschlagt, so daß insgesamt 4 Gleitkommaoperationen für die Projektion einer Ecke nötig sind.

Umwandlung in Gerätekoordinaten

Die Überführung der Szene von normierten Gerätekoordinaten (NDC) in

physikalische Gerätekoordinaten (PDC) erfordert für jede der drei Komponenten einer Punktkoordinate eine Multiplikation zur Skalierung, eine Addition für eine Verschiebung sowie die Umwandlung in das Festpunktformat (=1 FLOP). Es sind also 9 FLOP zur Transformation eines Punktes in die Gerätekoordinaten erforderlich.

Initialisierung des Display-Prozessors

Die Berechnung der Parameter der HESSE-Form einer Geraden aus den Eckpunkten verlangt folgende Operationen:

Die Ermittlung der reziproken Länge einer Dreiecksseite geschieht durch 3 Additionen, 2 Multiplikationen (zur Quadrierung) und die Bestimmung des Kehrwerts einer Quadratwurzel (16 FLOP). Dies beläuft sich auf insgesamt 21 FLOP. Zur Berechnung der drei gesuchten Parameter sind dann weitere 8 FLOP pro Gerade erforderlich.

Zur Berechnung des Kehrwerts des gemeinsamen Nenners sind 12 FLOP nötig. Für jede der Komponenten *RGB* und *z* braucht man 33 FLOP, um die Parameter endgültig zu ermitteln. Die gesamte Initialisierung erfordert folglich pro Dreieck:

$$[3 \cdot (21+8) + 12 + 4 \cdot 33] \text{ FLOP} = 231 \text{ FLOP}$$

Laden des Display-Prozessors

Die so berechneten Initialisierungsparameter müssen zum Display-Prozessor übertragen bzw. in das LDF geschrieben werden. Die dazu erforderliche Bandbreite berechnet sich aus der Anzahl der pro Dreieck gesandten Datenworte. Wegen der Darstellung der Dreiecke durch die Randgeraden (in HESSE-Form) werden für jede Dreiecksseite 3 Worte übertragen. Je 3 Worte werden auch für den z-Wert und jede Komponente der Farbe bzw. des Normalenvektors übertragen. Insgesamt werden also 21 Worte pro Dreieck übertragen. Soll die Beleuchtungsrechnung im Display-Prozessor geschehen, müssen zusätzlich noch die 4 Worte zur Beschreibung der Oberflächeneigenschaften eines Dreiecks übertragen werden. Damit erhöht sich die Zahl der nötigen Datenworte auf 25.

2.4.3.4 Anforderungen an den Display-Prozessor

Der Display–Prozessor führt seine Berechnungen mit Größen durch, die als Festkomma-Zahlen vorliegen. Er muß folgende Aufgaben bearbeiten:

Scan-Konvertierung

Die Ermittlung der bedeckten Rasterpunkte erfordert für jedes Pixel in einem Dreieck 7 Festkomma-Additionen. Diese Operationen schließen bereits die notwendigen Interpolationen von Tiefenwert (*z*) und Farbkomponenten (*RGB*) bzw. Komponenten des Normalenvektors ein.

Verdeckungsrechnung (vgl. Kapitel 5 'Visibilität)

Wird ein Pixel von einem Dreieck überdeckt, so muß ermittelt werden, ob es näher zum Betrachter liegt als das bereits berechnete und gespeicherte (z–Buffer). Dies erfordert einen Vergleich, der einer Festkomma-Operation (FXOP) entspricht.

Beleuchtungsrechnung

Falls die Beleuchtungsrechnung vom Display-Prozessor durchgeführt wird, erfordert dies pro Pixel $2 + \mathrm{L} \cdot 26$ FLOP.

Anti-Aliasing (vgl. Abschnitt 2.5.4 'Glätten von Polygonkanten')

Die Berechnung der Subpixel-Maske erfordert pro Dreiecksseite 31 Additionen und 16 logische Operationen, für die insgesamt eine Festkomma-Operation angesetzt wird. Dazu kommt pro Dreieck ein Vergleich der Tiefenwerte. Zusammen braucht man also 95 Festkomma-Operationen für die Berechnung der Subpixel-Maske eines Dreiecks.

Die Zusammenfassung der 16 Subpixelfarben bei 4×4 Subpixel zur endgültigen Pixelfarbe geschieht für jede Farbkomponente durch 15 Additionen und eine Shiftoperation. Es sind also 48 Festkomma-Operationen zur Errechnung der Pixelfarbe nötig.

Wenn $\overline{p}$ die Anzahl der Dreiecke pro Pixel ist, dann schlägt das gesamte Anti-Aliasing also mit $\overline{p} \cdot 95 + 48$ Festkomma-Operationen pro Pixel zu Buche.

Schreiben des Bildspeichers (PDF)

Wenn ein Dreieck in einem Pixel sichtbar ist, werden zwei Werte in den Bildspeicher geschrieben: Farbwert (*RGB*) und Tiefenwert (z). Für den Vergleich der z-Werte während der Verdeckungsrechnung muß der z-Wert des im Bildspeicher stehenden Pixels gelesen werden. Zusammen sind also maximal 3 Speicherzugriffe pro Pixel notwendig.

Damit resultieren folgende Anforderungen an den Geometrie–Prozessor eines Graphiksystems geringerer Leistungsfähigkeit:

Transformation der Eckpunkte:

$$300\,000 \frac{\text{Ecken}}{\text{Bild}} \cdot 28 \frac{\text{FLOP}}{\text{Ecke}} \cdot 10 \frac{\text{Bilder}}{\text{sec}} = 84 \text{ MFLOPS}$$

Transformation der Normalenvektoren:

$$300\,000 \frac{\text{Ecken}}{\text{Bild}} \cdot 52 \frac{\text{FLOP}}{\text{Ecke}} \cdot 10 \frac{\text{Bilder}}{\text{sec}} = 156 \text{ MFLOPS}$$

Beleuchtungsrechnung für jeden Eckpunkt:

$$300\,000 \frac{\text{Ecken}}{\text{Bild}} \cdot 28 \frac{\text{FLOP}}{\text{Ecke}} \cdot 10 \frac{\text{Bilder}}{\text{sec}} = 84 \text{ MFLOPS}$$

Clippen:

$$300\,000 \frac{\text{Ecken}}{\text{Bild}} \cdot 6 \frac{\text{FLOP}}{\text{Ecke}} \cdot 10 \frac{\text{Bilder}}{\text{sec}} = 18 \text{ MFLOPS}$$

Projektion der Eckpunkte:

$$300\,000 \frac{\text{Ecken}}{\text{Bild}} \cdot 4 \frac{\text{FLOP}}{\text{Ecke}} \cdot 10 \frac{\text{Bilder}}{\text{sec}} = 12 \text{ MFLOPS}$$

Umwandlung in Gerätekoordinaten:

$$300\,000\frac{\text{Ecken}}{\text{Bild}}\cdot 9\frac{\text{FLOP}}{\text{Ecke}}\cdot 10\frac{\text{Bilder}}{\text{sec}} = 27\ \text{MFLOPS}$$

Initialisierung des Display–Prozessors:

$$100\,000\frac{\text{Dreiecke}}{\text{Bild}}\cdot 231\frac{\text{FLOP}}{\text{Dreiecke}}\cdot 10\frac{\text{Bilder}}{\text{sec}} = 231\ \text{MFLOPS}$$

Laden des Display–Prozessors:

$$100\,000\frac{\text{Dreiecke}}{\text{Bild}}\cdot 21\frac{\text{Worte}}{\text{Dreieck}}\cdot 10\frac{\text{Bilder}}{\text{sec}} = 21\ \frac{\text{MWorte}}{\text{sec}}$$

Für dieses Graphiksystem muß der Geometrie–Prozessor also mindestens **612 MFLOPS** leisten. Seine Übertragungsbandbreite muß mindestens **21 MWorte/sec** betragen.

2.4.3.5 Anforderungen an den Display–Prozessor eines Grafiksystems geringerer Leistungsfähigkeit

Scan–Konvertierung:

$$7\frac{\text{FXOPS}}{\text{Pixel}}\cdot 100\frac{\text{Pixel}}{\text{Dreieck}}\cdot 100\,000\frac{\text{Dreiecke}}{\text{Bild}}\cdot 10\frac{\text{Bilder}}{\text{sec}} = 700\ \text{MFXOPS}$$

Verdeckungsrechnung:

$$1\frac{\text{FXOPS}}{\text{Dreieck}}\cdot 3\frac{\text{Dreiecke}}{\text{Pixel}}\cdot 2^{20}\frac{\text{Pixel}}{\text{Bild}}\cdot 10\frac{\text{Bilder}}{\text{sec}} = 30\ \text{MFXOPS}$$

Beleuchtungsrechnung nicht vorhanden.

Anti–Aliasing nicht vorhanden.

Schreiben des PDF:

$$3\frac{\text{Worte}}{\text{Dreieck}}\cdot 3\frac{\text{Dreiecke}}{\text{Pixel}}\cdot 2^{20}\frac{\text{Pixel}}{\text{Bild}}\cdot 10\frac{\text{Bilder}}{\text{sec}} = 90\ \frac{\text{MWorte}}{\text{sec}}$$

Für das Graphiksystem mittlerer Leistung muß der Display–Prozessor also mindestens **730 MFXOPS** leisten.

2.4.3.6 Bandbreite des Host–Rechners eines Graphiksystems geringerer Leistungsfähigkeit

$$\left(300\,000\frac{\text{Ecken}}{\text{Bild}}\cdot 6\frac{\text{Worte}}{\text{Ecke}} + 100\,000\frac{\text{Oberflächen}}{\text{Bild}}\cdot 4\frac{\text{Parameter}}{\text{Oberfläche}}\cdot 1\frac{\text{Wort}}{\text{Parameter}}\right)$$

$$\cdot 10 \frac{\text{Bilder}}{\text{sec}} = 22 \frac{\text{MWorte}}{\text{sec}}$$

2.4.3.7 Graphiksystem höherer Leistungsfähigkeit

- 100.000 Dreiecke pro Bild
- 8 Lichtquellen
- Interpolation von Normalenvektoren zwischen Eckpunkten. Die Farbwerte werden für jedes Pixel gesondert errechnet (PHONG–Shading, vgl. Kapitel 7 über 'Beleuchtungsmodelle').
- Berechnung von 30 Bildern pro Sekunde
- Anti–Aliasing mittels Subpixelmasken. Die mittlere Anzahl von Dreiecken in einem Pixel sei 3. Jedes Pixel sei in 4×4 Subpixel unterteilt.

Das Berechnen im einzelnen soll hier unterbleiben. Bei der folgenden Zusammenfassung der Leistungsanforderungen ist zu beachten, daß die Beleuchtungsrechnung beim Hochleistungssystem nicht im Geometrie–, sondern im Displayprozessor durchgeführt wird.

erforderliche	Graphiksystem	
Rechenleistung	geringerer Leistung	höherer Leistung
Host	ca. 50 MFLOPS	ca. 150 MFLOPS
Geometrie–Prozessor	612 MFLOPS	1.584 MFLOPS
Display–Prozessor	730 MFXOPS	73.000 MFXOPS

2.4.3.8 Gegenüberstellung der Bandbreiten von Host, Geometrie–Prozessor und Display–Prozessor für zwei exemplarische Graphiksysteme

erforderliche	Graphiksystem	
Bandbreite	geringerer Leistung	höherer Leistung
Host	22MWorte/sec	66MWorte/sec
Geometrie–Prozessor	21MWorte/sec	75 MWorte/sec
Display–Prozessor	90 MWorte/sec	283 MWorte/sec

2.4.3.9 Pixeloperationen

Rasteroperationen

Zum Abschluß dieses Abschnitts soll noch eine sehr allgemeine und vielseitig nutzbare Funktion innerhalb des Bilderzeugungssystems, die sogenannte Raster Op [NS79] oder BitBlt (Block Transfer of Bits), besprochen werden. Mit ihr kann ein Pixelblock als Quelle (Source) mit dem Inhalt eines anderen Blocks als Senke (Destination) logisch oder arithmetisch verknüpft und an dessen Stelle abgespeichert werden:

Destination := *Source* op *Destination*

Theoretisch können alle 16 Boole'schen Funktionen zweier Variablen realisiert werden. Praktisch beschränkt man sich aber auf folgende Funktionen bei binären Displays: (*D*=Destination, *S*=Source)

Funktion			**Operation**	**Anwendung**
Clear	*D*	:=	constant	Löschen oder Initialisieren
Constant	*D*	:=	feste Muster	Hintergrund
Copy	*D*	:=	*S*	Text, Scrolling
Invert	*D*	:=	NOT *D*	Optisches Hervorheben
AND	*D*	:=	*S* AND *D*	Kombinationsfunktion
OR	*D*	:=	*S* OR *D*	Kombinationsfunktion
XOR	*D*	:=	*S* XOR *D*	Informationserhaltende Funktion

Bild 2.33 zeigt die unterschiedliche Wirkung von OR und XOR. Das Exklusive ODER = XOR ist wegen seiner informationserhaltenden Wirkung besonders interessant. Wird ein Pixelwert *S* zweimal nacheinander mit einem Pixelwert *D* über XOR verknüpft, dann wird der ursprüngliche Wert wieder hergestellt (XOR = $\oplus$):

XOR–Funktion

$$(S \oplus D) \oplus D = S \oplus (D \oplus D) = S \oplus 0 = S$$

Deshalb bieten die meisten Geräte für das Beschreiben des Speichers alternativ den "Copy"– oder "XOR"–Modus an.

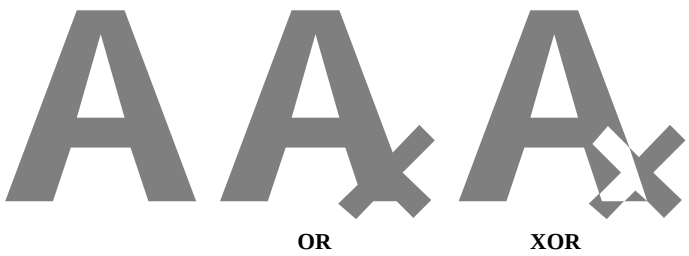


Abbildung 2.33: OR und XOR Verknüpfungen

Die Anwendung der Raster Op auf Grauwertdisplays (mehrere Pixelebenen) bedarf einer Erläuterung. Die in der folgenden Aufstellung eingeführten Funktionen MIN und MAX sind äquivalent zu den Boole'schen AND und OR. Sei 1 der größte Grauwert, dann ist (1 – Grauwert) äquivalent zur Boole'schen Inversion.

Funktion			Operation	Anwendungen
Minimum	D	:=	MIN (S, D)	AND bei Grauwerten
Maximum	D	:=	MAX (S, D)	OR bei Grauwerten
Komplement	D	:=	1 - D	Invert bei Grauwerten
Summe	D	:=	$D+S$	XOR bei Grauwerten
Differenz	D	:=	$D-S$	XOR bei Grauwerten
Neg. Differenz	D	:=	$S-D$	XOR bei Grauwerten
Skalierung	D	:=	$k \cdot D$	Kontraständerung
Helligkeit	D	:=	$D+k$	Helligkeitsänderung

Wenn Sie die PDF-Version des Kurses bearbeiten, können Sie jetzt das Interaktionselement 'pixelblocks' anwählen.

2.4.4 Bildspeicher

Konkurrenz um den Zugriff zum Bildspeicher

Der Bildspeicher (Bildwiederholungsspeicher, image storage, frame buffer) ist das Bindeglied zwischen Bilderzeugung (ICS) und –darstellung (IDS) und muß deren Anforderungen erfüllen. Beide Systemteile wollen mit hoher Bandbreite auf den Speicher zugreifen. Das IDS ist dabei zur Erzielung eines flimmerfreien Bildes an den starren Ablauf der Rasterablenkung gebunden, während das ICS möglichst wahlfrei viele Bildpunkte pro Zeiteinheit ändern will, um hohe Interaktionsraten oder Bewegtbildeffekte zu erzielen. Beim Entwurf des Bildspeichers ist darauf zu achten, daß ICS und IDS die erforderlichen Speicherzyklen konfliktfrei erhalten können. Zur Vereinfachung der Darstellungen, Berechnungen und Vergleiche werden folgende Voraussetzungen getroffen: Es wird ein binäres Display (1 bit/Pixel) betrachtet. Höhere Bitzahlen pro Pixel sind dann durch entsprechende Vervielfachung (Parallelschaltung von Speicherchips) zu realisieren.

Alle Abschätzungen mit 100 ns Zykluszeit der Speicherbausteine

Die Bildpunkte werden nach der Reihenfolge ihrer Ausgabe auf dem Bildschirm unter sequentiellen Adressen abgespeichert. Unabhängig von tatsächlich verfügbaren Speicherbausteinen wird eine Zykluszeit von 100*ns* zugrunde gelegt. Aus Kostengründen werden dynamische Speicher (DRAMs) verwendet.

Aus Bild 2.36 ist zu entnehmen, daß die Bildpunktdauer im allgemeinen kürzer als ein Speicherzyklus ist. Die Bildpunktdauer für ein 1024×1024 Display beträgt z.B. bei 60 Hz nur 11.42*ns*. Demnach muß der Speicher so organisiert sein, daß mit einem Zugriff mehrere Pixel gleichzeitig ausgelesen werden können. Andererseits soll natürlich vermieden werden, daß die Anzahl der Speicherbausteine, die zur Realisierung der geforderten Speicherbandbreite nötig ist, sehr viel mehr Speicherkapazität aufweist als für das gewählte Bildformat erforderlich ist.

Im folgenden werden drei Varianten zur Lösung des Zugriffsproblems diskutiert. Dabei wird folgendes Zahlenbeispiel zugrunde gelegt (siehe Bild 2.36):

Bildschirm	1024×1024
Bildwiederholfrequenz	60 Hz
Horizontaler Strahlrücklauf	4 μs
Vertikaler Strahlrücklauf	600 μs
Zeilendauer	$15.69\mu s \approx 16\mu s$
Punktdauer	$11.42ns \approx 10ns$
Speicherzykluszeit	$100ns$

2.4.4.1 Bildspeicher aus mehreren Speicherbänken

Aufteilung der Speicherzugriffe auf ICS und IDS

Wird der Bildspeicher z.B. aus 16 gleichzeitig adressierbaren Speicherbänken aufgebaut, so können 16 Pixel parallel ausgelesen werden. Sie werden in einem Schieberegister zwischengespeichert und dann mit dem Bildpunkttakt seriell ausgelesen. Da mit einem Zugriff von $100ns$ 16 Pixels bzw $160ns$ Bilddarstellungszeit gewonnen werden (spezielle Zugriffsmodi wie Page–Mode werden nicht betrachtet), verbraucht das IDS nur 100/160 der insgesamt während einer Bildschirmzeile zur Verfügung stehenden Zugriffszeit. Der Rest (60/160) zuzüglich der horizontalen Strahlrücklaufzeit von $4\mu s$ kann vom ICS genutzt werden. Pro Bild hat das ICS dann folgende Zugriffszeit:

$$\begin{aligned} \text{Zeilenzahl} \cdot \left[\left(\frac{60}{160} \cdot 12\mu s\right) + 4\mu s\right] + \text{Vertikale Strahlrücklaufzeit}(600\mu s) &= \\ 1024 \cdot \left[\left(\frac{60}{160} \cdot 12\mu s\right) + 4\mu s\right] + 600\mu s &= \\ 1024 \cdot 8,5\mu s + 600\mu s \approx 9300\mu s & \end{aligned}$$

Das ICS greift im Gegensatz zum IDS wahlfrei auf den Speicher zu und kann demnach pro Bildzyklus $[9300\mu s/100ns]$ Pixel = 93000 Pixel neu schreiben. Ein völlig neues Bild ist dann nach $2^{20}/93000 \approx 11$ Bildzyklen oder $180ms$ aufgebaut. Diese Rechnung ist sehr optimistisch! Sie setzt voraus, daß das ICS die angebotenen Schreibzeiten mit 10 MHz $= \frac{1}{100\ ns}$ Schreibfrequenz auch nutzen kann. Dies wird in der Regel nur möglich sein, wenn zwischen ICS und Bildspeicher ein Pufferspeicher geschaltet wird, um dem ICS ein möglichst gleichmäßiges Schreiben mit niedriger Taktrate zu ermöglichen. Setzt man diese Überlegung konsequent fort, kommt man zum sogenannten Wechselspeicher (double buffer).

2.4.4.2 Wechselspeicher (double buffer)

Der Wechselspeicher besteht aus zwei kompletten Bildspeichern (Bild 2.34). Sie werden im Wechsel betrieben.

Wird z.B. aus Bildspeicher 1 vom IDS der Pixelstrom zum Bildaufbau ausgelesen, so wird gleichzeitig vom ICS Bildspeicher 2 mit neuer Bildinformation geladen. Sobald das neue Bild aufgebaut ist, frühestens jedoch nach Auslesen des vollständigen Bildes, werden mit den Multiplexern die Funktionen vertauscht. Beide

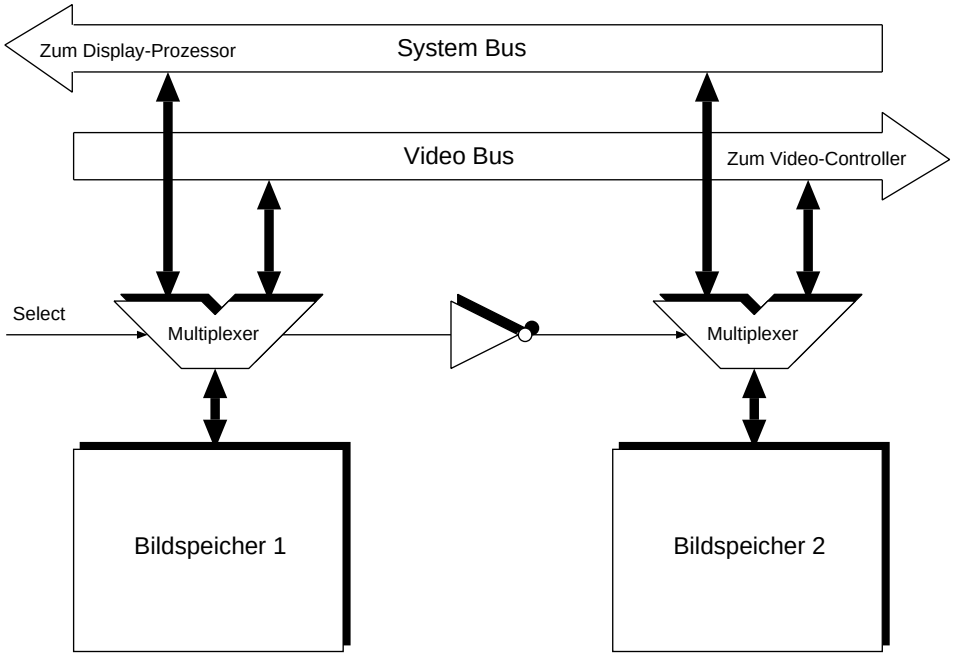


Abbildung 2.34: Bildspeicher realisiert als Wechselspeicher

Prozesse, d.h. Laden und Auslesen, sind auf diese Weise völlig entkoppelt. Jeder Prozeß hat die maximal mögliche Zugriffszeit. Der Preis hierfür ist die doppelte Realisierung des Bildspeichers. Trotzdem ist diese Lösung bei den meisten Systemen höherer Leistung zu finden.

Aus den besonderen Anforderungen an den Bildspeicher wurde das sogenannte Video–RAM (VRAM) entwickelt. Durch seine über das DRAM hinausgehenden Eigenschaften ist es möglich, den Speicherzugriff zur Bildwiederholung auf wenige Speicherzyklen zu reduzieren.

2.4.4.3 Bildspeicher mit VRAM

Beim VRAM wird die Tatsache ausgenutzt, daß in einem DRAM–Speicher der Zugriff zweistufig erfolgt. Zuerst wird die Zeile der Speichermatrix ausgewählt und anschließend die gewünschte Spalte (siehe Bild 2.35).

Wird nach der Zeilenauswahl die gesamte Information dieser Zeile in ein zusätzliches Schieberegister geladen, so steht ein serieller Ein–/Ausgang zusätzlich zum bereits vorhandenen parallelen Ein–/Ausgang zur Verfügung. Kann das Schieberegister wahlweise durch einen Bypass umgangen werden, so entsteht das VRAM (Bild 2.36).

Bei dieser Konstruktion stehen also mit einem Speicherzyklus alle 512 Pixel der

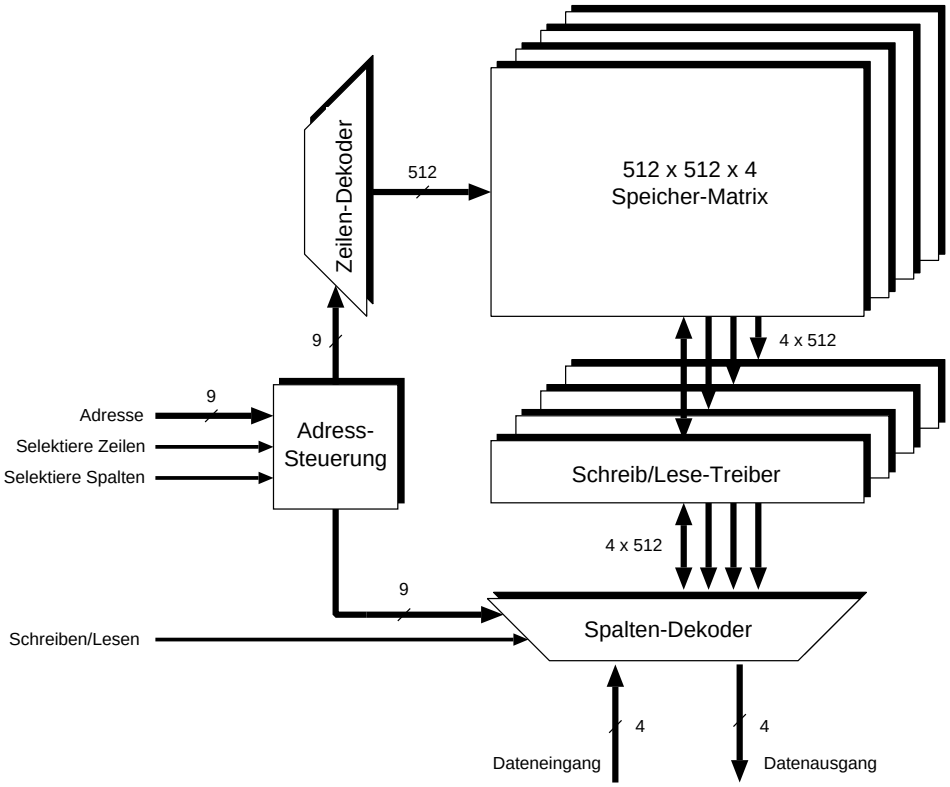


Abbildung 2.35: 1 Mbit DRAM in 256 k×4 Anordnung

Speicherzeile zur Verfügung. Das bedeutet, daß für das ICS 511 Zyklen für den wahlfreien Zugriff frei sind.

Bild 2.37 zeigt das VRAM symbolisch.

Nehmen wir als Beispiel einen VRAM mit einer seriellen Datenrate von 30 MHz. Ein 512×512×60 Hz Display könnte mit diesem Baustein gut realisiert werden. Unser obiges Zahlenbeispiel mit 1024×1024 Pixel benötigt für die erforderliche Bandbreite von 100MHz (10*ns* Bildpunktdauer) eine Realisierung mit 4 Speicherbänken (siehe Bild 2.38).

Die 4 Speicherbänke arbeiten auf ein externes Schieberegister (als Multiplexer realisiert), das üblicherweise mit den Digital–Analog–Convertern (DAC) und der LUT integriert ist. In diesem Fall steht mit einem Speicherzugriff die Information von (4×512) Pixel zur Verfügung! Das bedeutet, daß das ICS nur in jeder zweiten Zeile einen Speicherzyklus an das IDS ”verliert”. Damit ist mit VRAMs die Konkurrenz zwischen ICS und IDS um Speicherzugriffe praktisch beseitigt. In vielen Systemen kann wahlweise mit oder ohne Wechselspeicher gearbeitet werden. Dann ist natürlich eine Realisierung mit VRAMs erforderlich.

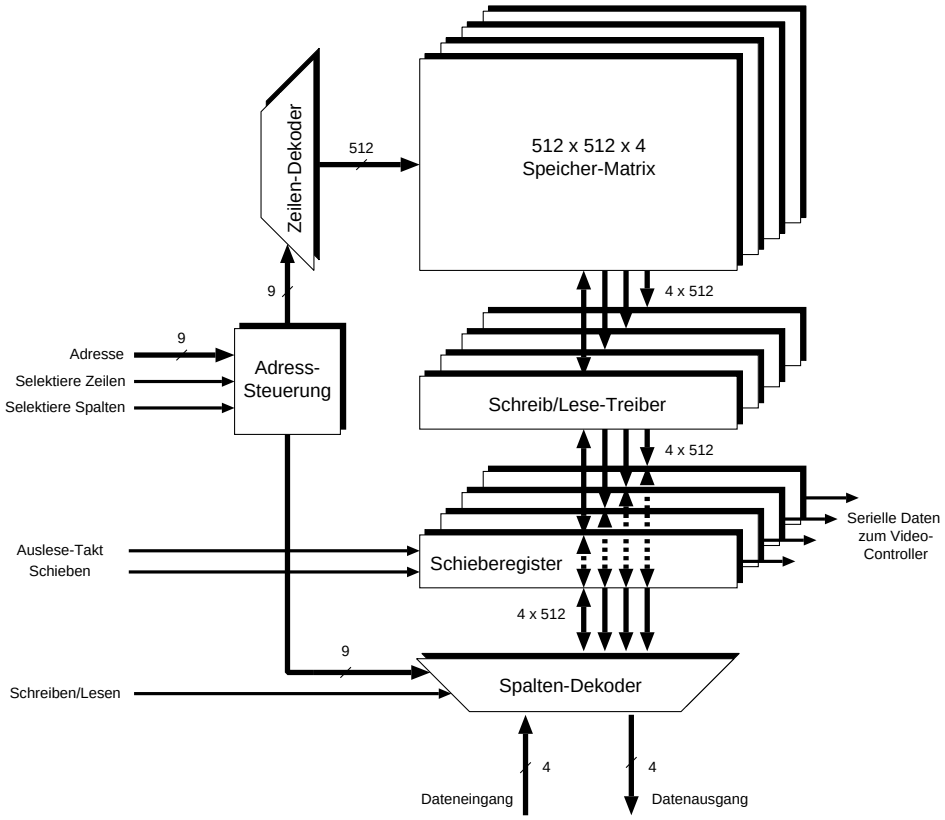


Abbildung 2.36: 1–Mbit VRAM in 256 k×4 Anordnung
(Die gestrichelten Linien im Schieberegister symbolisieren einen Bypass)

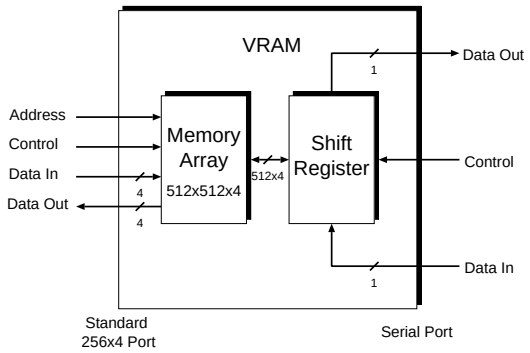


Abbildung 2.37: Symbolische Darstellung des VRAM 512 × 512 × 4

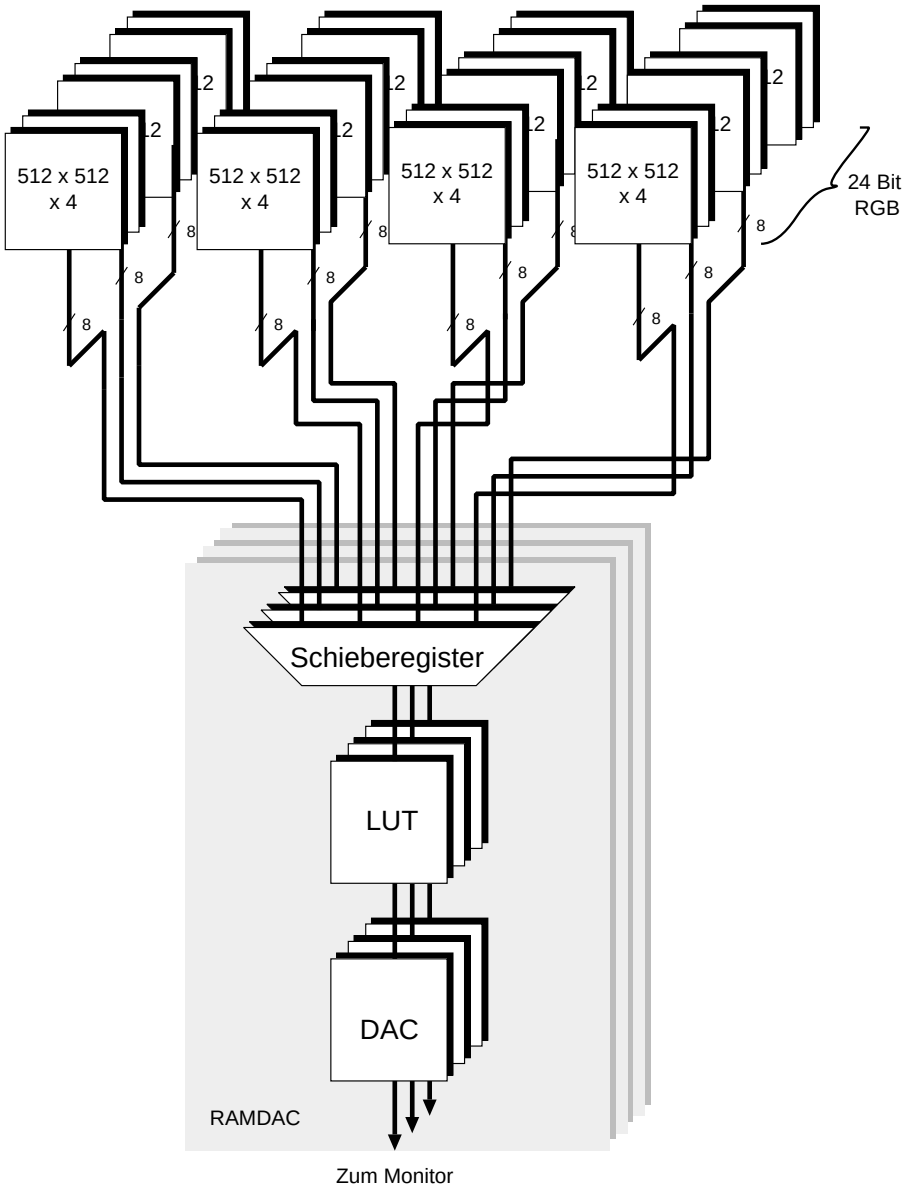


Abbildung 2.38: Bildspeicher für ein 1024×1024×24 Display aus 1 Mbit VRAMs und RAMDACs

2.5 Rasteralgorithmen

Im Bilderzeugungsteil (ICS) des Rastergraphiksystems werden die graphischen Primitiva, aus denen die Szene zusammengesetzt ist, in Rasterpunkte (Pixel) zerlegt. Die Geschwindigkeit, mit der diese Rasterung durchgeführt wird, bestimmt das Echtzeitverhalten des Systems wesentlich. Deshalb ist die Bereitstellung effizienter Rasteralgorithmen von zentraler Bedeutung.

2.5.1 Rasterung von Strecken

Die geradlinige Verbindung zweier Punkte, in der Fachsprache oft Vektor genannt, ist das wichtigste graphische Element (Primitivum). Der beste bisher bekannte Algorithmus zur Rasterung von Strecken stammt von Bresenham [Bre65]. Er beruht auf dem einfachen Prinzip der Rundung (siehe Bild 2.39). Die folgende Herleitung des Algorithmus betrachtet Vektoren, deren Anfangs– und Endpunkte auf dem Raster liegen und eine Steigung zwischen 0° und 45° haben. Alle anderen Vektoren können hieraus durch Vertauschen der X– und Y–Koordinaten unter Beachtung der Vorzeichen gewonnen werden. Seien $P_1(X_1, Y_1)$ und $P_2(X_2, Y_2)$ die Anfangs– und Endpunkte des Vektors und

$$\begin{aligned} \Delta X &= X_2 - X_1 \geq 0 \\ \Delta Y &= Y_2 - Y_1 \geq 0 \\ \Delta X &\geq \Delta Y. \end{aligned}$$

Um die Rundung durchführen zu können, wird eine Entscheidungsgröße E eingeführt, die die (vertikale) Entfernung zwischen dem exakten Punkt und der Mitte zwischen den beiden möglichen Rasterpunkten angibt und deren Vorzeichen als Kriterium für die Rundung auf den nächstliegenden Rasterpunkt dient (Bild 2.39).

Ist $E \leq 0$, so wird der exakte y–Wert auf den nächst kleineren, ganzzahligen Y–Wert abgerundet, d.h. die Y–Koordinate wird gegenüber dem vorhergehenden Bildpunkt nicht verändert. Andernfalls wird aufgerundet und Y um 1 erhöht.

Zur Berechnung von E wird ohne Einschränkung der Allgemeinheit vereinfachend angenommen, der betrachtete Vektor beginne im Koordinatenursprung (vgl. Bild 2.39).

Große Buchstaben geben Rasterpunkte, kleine geben Punkte des Vektors an:

$$E_1 = y_1 - \frac{1}{2} = \frac{\Delta Y}{\Delta X} - \frac{1}{2}$$

Fallunterscheidung:

Fall 1: $E_1 > 0$:

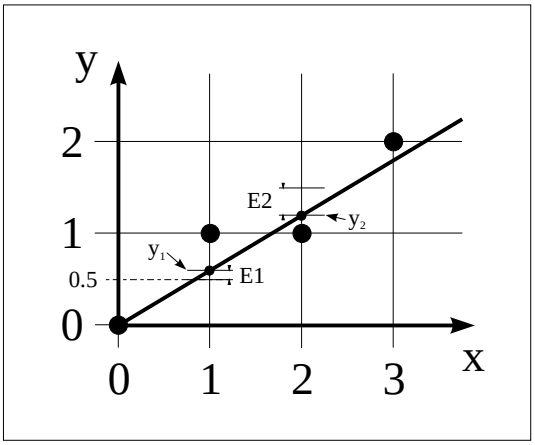


Abbildung 2.39: Erzeugung der Rasterpunkte des Vektors durch Rundung

Nächster Punkt wird:

$$\begin{aligned} X_1 &= X_0 + 1 = 1 \\ Y_1 &= Y_0 + 1 = 1 \end{aligned}$$

Für die nächste Entfernung E_2 gilt dann:

$$\begin{aligned} E_2 &= y_2 - Y_1 - \frac{1}{2} \\ &= y_1 + \frac{\Delta Y}{\Delta X} - Y_0 - 1 - \frac{1}{2} \\ &= E_1 + \frac{\Delta Y}{\Delta X} - 1 \end{aligned}$$

d.h.

$$E := E + \frac{\Delta Y}{\Delta X} - 1$$

Fall 2: $E_1 \leq 0$:

Nächster Punkt wird:

$$\begin{aligned} X_1 &= X_0 + 1 = 1 \\ Y_1 &= Y_0 = 0 \end{aligned}$$

Für die nächste Entfernung E_2 gilt dann:

$$\begin{aligned} E_2 &= y_2 - Y_1 - \frac{1}{2} \\ &= y_2 - Y_0 - \frac{1}{2} \\ &= y_1 + \frac{\Delta Y}{\Delta X} - \frac{1}{2} \\ &= E_1 + \frac{\Delta Y}{\Delta X} \end{aligned}$$

d.h.

$$E := E + \frac{\Delta Y}{\Delta X}$$

Da von E nur das Vorzeichen interessiert, können die störenden Quotienten durch Multiplikation mit $2 \cdot \Delta X$ beseitigt werden:

$$E_1' = 2\Delta Y - \Delta X$$

$$E_1' > 0 \quad : \quad E' := E' + 2(\Delta Y - \Delta X)$$

$$E_1' \leq 0 \quad : \quad E' := E' + 2\Delta Y$$

Damit läßt sich ein Algorithmus angeben, der nur mit ganzen Zahlen sowie Addition, Subtraktion und Shift arbeitet [Bre65]:

```
ΔY   :=  Y2 - Y1;
ΔX   :=  X2 - X1;
X    :=  X1;  ← Anfangspunkt des Vektors
Y    :=  Y1;
E'   :=  2ΔY - ΔX;

FOR i := 1 TO ΔX DO
BEGIN
     schreibe (X, Y);
     X := X + 1;
     IF E' > 0 THEN
     BEGIN
          Y := Y + 1;
          E' := E' + 2(ΔY - ΔX)
     END
     ELSE E' := E' + 2ΔY
END;
schreibe (X, Y);  ← Endpunkt des Vektors
```

Das Aussehen der so erzeugten Vektoren ist wegen der geringen Auflösung des Rasterbildschirms unbefriedigend (vgl. Bild 2.40). Statt einer glatten Linie sieht man unregelmäßige Treppenstufen. Dieser Effekt, oft “aliasing” genannt, läßt sich mindern, wenn ein Sichtgerät mit voller Grauwert– bzw. Farbauflösung zur Verfügung steht. Dabei wird dann die integrierende Wirkung des Auges ausgenutzt.

Wenn Sie die PDF-Version des Kurses bearbeiten, können Sie jetzt das Interaktionselement ’bresenham’ anwählen.

2.5.2 Glätten von gerasterten Strecken

Zur Darstellung von geglätteten Strecken bietet sich ein einfacher Algorithmus an, der sich, wie schon der Bresenham–Algorithmus, ebenfalls gut für eine Hardwareimplementierung eignet. Der Algorithmus wird wieder am Beispiel eines Vektors mit einer Steigung zwischen 0 und 1 dargestellt (Bild 2.40).

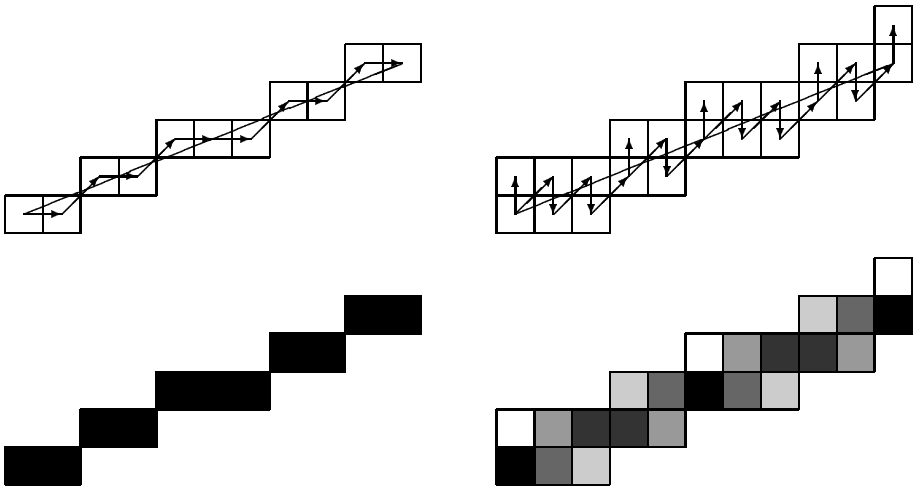


Abbildung 2.40: Gegenüberstellung von ungeglättetem und geglättetem Vektor

Zur Verdeutlichung ist im Bild 2.40 eine übertrieben grobe Rasterung gewählt worden! Es ist zu erkennen, daß bei dem hier vorgestellten einfachen Glättungsverfahren für jeden X–Wert (Spalten) zwei übereinanderliegende Pixel gesetzt werden. Die Gesamthelligkeit ist in jeder Spalte gleich und wird den vertikalen Entfernungen E der beiden Pixel von der zu untersuchenden Strecke entsprechend verteilt. Dabei nimmt die Pixelhelligkeit (Grauwert) linear mit steigender Entfernung ab.

$$\text{Normierter Grauwert} \quad = \quad \begin{cases} 1 & \text{für} \quad E = 0 \\ 0 & \text{für} \quad E = 1 \end{cases} \qquad \text{beim Bildschirm;}$$

Auf dem Papier sind schwarz und weiß vertauscht. Aus Bild 2.40 rechts ist der Ablauf des Glättungsverfahrens zu erkennen: Nach dem Bearbeiten einer Spalte wird immer nach rechts oben weitergegangen (X und Y werden um 1 erhöht). Dann wird die vertikale Entfernung E des erreichten Pixel von dem zu zeichnenden Vektor bestimmt. Das Vorzeichen gibt an, ob das zweite Pixel in dieser Spalte oberhalb oder unterhalb des bereits erreichten Pixel liegt. Folgende Pascalprozedur ist eine mögliche Realisierung dieses Algorithmus:

```
PROCEDURE antialiased-line (X1,Y1,X2,Y2: integer);

    VAR i:integer;
    VAR X,Y,dX,dY:integer;
    VAR incrE, E: real;
    BEGIN
        X := X1;
        Y := Y1;
        dX := X2 - X1;
        dY := Y2 - Y1;
        {Anfangswerte setzen}
```

```
IF dX <> 0 THEN {sicherstellen, daß dX ≠ 0}
incrE := 1 - dY/dX
{Änderung von E, wenn man nach rechts oben zum nächsten Pixel geht}
E:=0;
{Startwert für ganzzahligen Vektoranfangspunkt }
schreibe (X,Y,1);
{Startpixel mit max. Grauwert}
FOR i := 1 TO dX DO
BEGIN
    X := X + 1;
    {nach rechts }
    Y := Y + 1;
    {nach oben }
    E := E + incrE;
    {E für das neue Pixel}
    schreibe (X,Y,(1 - abs(E)));
    {Pixel mit Grauwert 1 - |E|) }
    IF E ≤ 0 THEN
    {Pixel ist unterhalb des Vektors, deshalb ist das zweite Pixel dieser Spalte oberhalb des ersten}
    BEGIN
        Y := Y + 1;
        {nach oben}
        E := E + 1;
        schreibe (X,Y,(1-abs(E)));
        {oberes Pixel mit Grauwert 1 - |E|}
        Y := Y - 1;
        {zurück nach unten, um anschließend wieder nach rechts oben gehen zu können}
        E := E - 1;
    END
    ELSE E > 0
    {Pixel ist oberhalb des Vektors, deshalb ist das zweite Pixel unterhalb davon}
    BEGIN
        Y := Y - 1;
        {nach unten}
        E := E - 1;
        schreibe (X,Y,(1-abs(E)));
        {unteres Pixel mit Grauwert 1-|E|}
    END
END
END;
```

Sind Anfangs– und/oder Endpunkt reelle Zahlen, dann müssen diese Fälle durch besondere Initialisierungen behandelt werden.
Um mit Glättungsverfahren die gewünschte Bildverbesserung zu erreichen, ist noch folgendes zu beachten:

- Eine Gammakorrektur muß vorgenommen werden.

- Wenn Vektoren in geringem Abstand voneinander zu zeichnen sind, so muß verhindert werden, daß hell gesetzte Pixel von einem benachbarten Pixel wieder dunkel gesetzt werden. Dieser Effekt tritt vor allem bei Eckpunkten auf, in denen mehrere Vektoren zusammentreffen. Die einfachste Lösung für dieses Problem besteht darin, einen neuen Helligkeitswert nur dann zu speichern, wenn er größer als der bereits gespeicherte ist.

- Bei jedem Algorithmus zur Streckenrasterung, bei dem die Anzahl der Pixel pro Spalte bzw. Zeile konstant ist, muß berücksichtigt werden, daß Vektoren mit einer Neigung α von $0 < \alpha \leq 45°$ gegenüber den Koordinatenachsen länger als achsenparallele Vektoren mit gleicher Pixelanzahl sind (Bild 2.41). Um den Eindruck gleicher Helligkeit zu erreichen, muß deshalb der Grauwert mit dem Faktor $\frac{1}{\cos\alpha}$ skaliert werden.

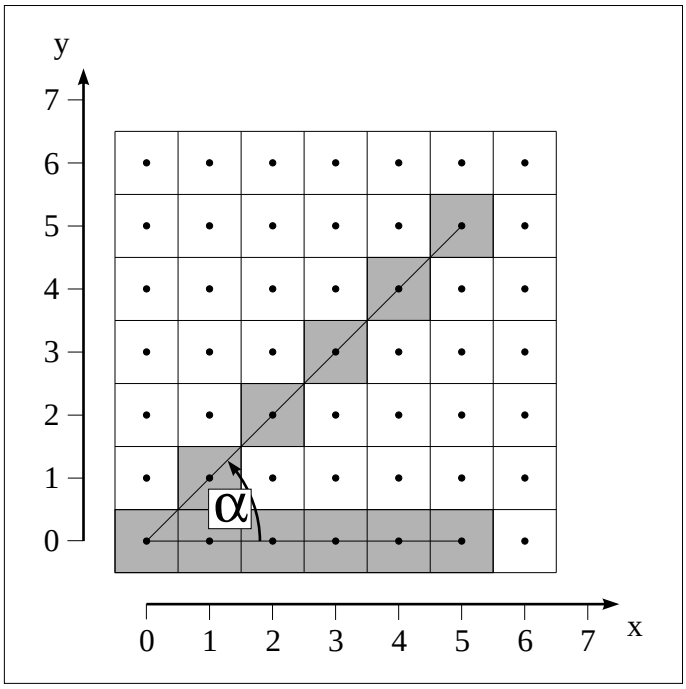


Abbildung 2.41: Unterschiedliche Länge als Funktion von α

Aus Bild 2.41 ist zu erkennen, daß hier ein quadratisches Pixelmodell angenommen wird, das lückenlos und ohne Überlappung die Bildschirmfläche bedeckt. Dies entspricht natürlich nicht den physikalischen Gegebenheiten, jedoch vereinfachen sich die Pixelberechnungen so erheblich. Darüber hinaus ist die so erzielte Bildqualität im Vergleich zu Kreismodellen nicht schlechter. Rastergraphik ist ihrem Wesen nach Flächengraphik. Die Strecke hat deshalb als Begrenzung zwischen zwei Flächen eine viel größere Bedeutung als ein einzelner Vektor.

Aufwendigere und genauere Glättungsverfahren sollen deshalb im Zusammenhang mit der Rasterung von Polygonen besprochen werden.

Wenn Sie die PDF-Version des Kurses bearbeiten, können Sie jetzt das Interaktionselement 'fineline' anwählen.

2.5.3 Rasterung von Polygonen

Werden Polygone als eine Folge von Strecken betrachtet, handelt es sich also um Vektorgraphik, dann kann die Rasterung durch vektorweise Anwendung des Bresenham–Algorithmus durchgeführt werden. Das Entscheidungskriterium bei diesem Algorithmus ist der kleinste Abstand zum gewünschten Vektor. Deshalb liegen die ausgewählten Pixel in der Regel auch links und rechts des idealen Verlaufs (siehe Bild 2.42). Beschreiben Polygone jedoch die Begrenzung von Flächen, dann ist dieser Algorithmus natürlich untauglich, weil das angewandte Entscheidungskriterium irrelevant ist. Bei der Darstellung von Flächen muß nämlich entschieden werden, ob ein Pixel

außerhalb
innerhalb oder
auf

der Begrenzung (Polygonkante) liegt. **Innerhalb** liegende Pixel gehören zur Polygonfläche und müssen gezeichnet werden.

Für Pixel, die **auf** einer Polygonkante liegen, muß eine Vereinbarung getroffen werden, damit auch gemeinsame Kanten zwischen zwei Flächen eindeutig behandelt werden können. Üblicherweise wird vereinbart, daß Pixel **auf** linken und unteren Kanten gezeichnet werden, **auf** rechten und oberen Kanten aber nicht! Das bedeutet, daß in diesen Fällen die oberste Rasterzeile und das rechte Pixel der entsprechenden Zeile fehlt!

Vorteile der Dreieckszerlegung

In der folgenden Beschreibung zur Rasterung von polygonal begrenzten Flächen werden statt allgemeiner Polygone nur Dreiecke betrachtet. (Der beschriebene Algorithmus funktioniert aber auch mit allgemeinen Polygonen). Dreiecke sind per Definition konvex und tragen deshalb zu einer bestimmten Rasterzeile nur ein Segment bei (siehe Bild 2.42), wodurch die Entscheidung auf **innerhalb** oder **außerhalb** vereinfacht wird. Darüber hinaus haben die für die Verdeckungs– und Beleuchtungsrechnung wichtigen Größen bei Dreiecken einen linearen Zusammenhang und können inkrementell berechnet werden (siehe Kapitel 7 über Beleuchtungsmodelle). Dies ist die Voraussetzung für optimale Algorithmen mit der Eignung zur Hardwareimplementierung.

Der Algorithmus besteht aus drei Schritten (siehe Bild 2.42):

1) Berechne die Schnittpunkte zwischen Dreieckskanten und Rasterzeilen.

2) Sortiere die Schnittpunkte nach aufsteigendem X–Wert für jede Rasterzeile.

3) Fülle die Segmente zwischen den entsprechenden Paaren von Schnittpunkten.

Die Berechnung von Schnittpunkten von Rasterzeile zu Rasterzeile wird inkrementell durchgeführt nach der Vorschrift:

$$\begin{aligned} X_{i+1} &= X_i + \frac{1}{m}, \\ \text{wobei } m &= \frac{Y_2 - Y_1}{X_2 - X_1} \text{ die Steigung} \end{aligned}$$

der betrachteten Polygonkante ist. Je nachdem, ob X_{i+1} ganzzahlig oder reell ist und ob es sich um eine linke oder rechte Kante (ein Segment geht immer von links nach rechts) handelt, ist zu entscheiden, welches Pixel als zum Dreieck gehörender Start– bzw. Endpunkt eines Zeilensegments gewählt wird. Dabei ist die oben genannte Vereinbarung für den Fall **auf** zu beachten.

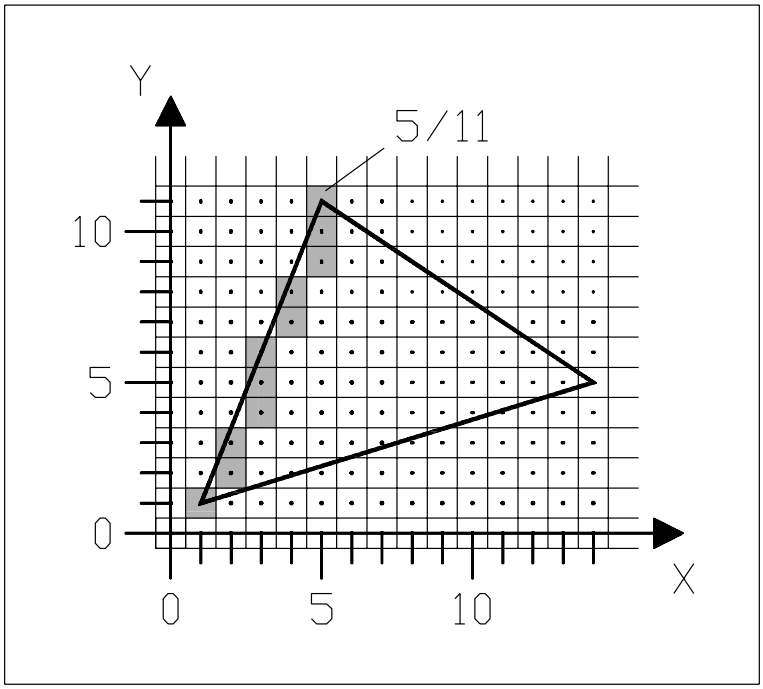


Abbildung 2.42: Rasterung eines Dreiecks

Am Beispiel einer linken Kante wird die Vorgehensweise erklärt (siehe Bild 2.42). Die Rasterung der linken Kante beginnt beim Startpunkt $X_{min} = 1$ und $Y_{min} = 1$. Laut Definition gehört dieser Punkt zum Dreieck. Für die nächste Zeile ($Y = 2$) wird der Kantenschnittpunkt

$$\begin{aligned} X_{i+1} &= X_i + \frac{X_{max} - X_{min}}{Y_{max} - Y_{min}} = X_i + \frac{\Delta X}{\Delta Y}, \\ \text{d.h.} \quad X_2 &= 1 + \frac{2}{5} \end{aligned}$$

inkrementell berechnet. Man erkennt, daß das zum Dreieck gehörende Startpixel für die nächste Zeile durch die zu X_{i+1} nächst größere ganze Zahl bestimmt ist, sofern der gebrochenrationale Teil von X_{i+1} von Null verschieden ist. Ist X_{i+1} ganzzahlig, so ist das Pixel **auf** der linken Kante definitionsgemäß ein Punkt des Dreiecks.

Offenbar ist der Betrag von $1/m$ gar nicht so entscheidend. Vielmehr interessiert, wann die Akkumulation des gebrochenrationalen Anteils von $1/m$ zum Überlauf führt! Hierzu wird keine Division benötigt. Addition und Subtraktion reichen als Operationen aus. Ähnlich der Vorgehensweise beim Bresenham–Algorithmus wird

$$\frac{1}{m} = \frac{\Delta X}{\Delta Y}$$

nicht berechnet, sondern nur ΔX akkumuliert und das Ergebnis mit ΔY verglichen.

Algorithmus mit Integerarithmetik ohne Division

Damit läßt sich für die linke Kante des Dreiecks folgender Algorithmus zur Rasterung angeben:

```
PROCEDURE linkeKante (Xmin, Ymin, Xmax, Ymax : integer);
BEGIN
    X := Xmin; {Startpixel}
    Y := Ymin;
    ΔX := Xmax - Xmin;
    ΔY := Ymax - Ymin;
    accu := 0; {Initialisieren des Akkumulators}
    FOR Y := Ymin TO Ymax DO
    BEGIN
        schreibe(X,Y);
        accu := accu + ΔX;
        WHILE accu > 0 DO
        {Überlauf zum nächsten Pixel!}
        {Accu um ΔY verkleinern}
        BEGIN
            X := X + 1;
            accu := accu - ΔY
        END;
    END
END;
```

Der gleiche Algorithmus wird auch für die rechten Kanten benötigt, was hier aber nicht ausgeführt wird.

Aus Bild 2.42 ist zu entnehmen, daß natürlich auch bei flächigen Objekten die Ränder ästhetisch unbefriedigende Rasterfehler aufweisen. Eine mögliche Lösung ist die exakte Ermittlung der vom Dreieck überdeckten Pixelfläche.

2.5.4 Glätten von Polygonkanten

Ein häufig angewandtes Verfahren zur Glättung (anti–aliasing) sind die sogenannten Subpixelmasken [A+85], [Sch91], [Car89], [Coo86], [F+83], [F+85]. Hierbei wird die Fläche der kritischen Randpixel in eine quadratische Anzahl von Subpixel (z.B. 4×4) unterteilt (siehe Bild 2.43). Statt eines Pixels werden nun 16 Subpixel (die Maske) mit entsprechend hoher Auflösung berechnet. Deshalb spricht

man auch von Überabtastung, over–sampling oder supersampling. Der Grau– oder Farbwert des auszugebenden Pixel wird dann durch Addition der Subpixelbeiträge bestimmt.

Das einfachste und deshalb gebräuchlichste Verfahren ist die Bestimmung der Masken aus der Lage der Mittelpunkte der Subpixel. Ein Subpixel wird gezählt, wenn es **innerhalb** des Polygons liegt. Mit dieser einfachen Methode wird allerdings die überdeckte Pixelfläche – im Vergleich zum Aufwand – äußerst ungenau bestimmt (Bilder 2.43, 2.44).

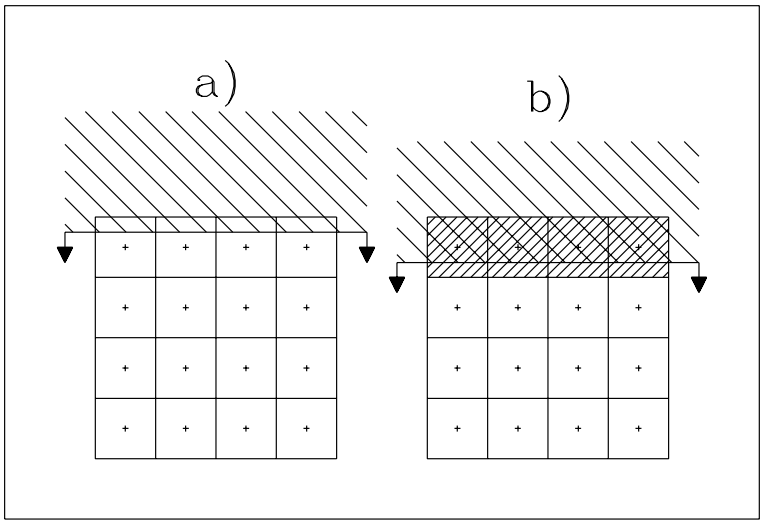


Abbildung 2.43: Horizontale Polygonkante über der in (4×4) Subpixel aufgeteilten Pixelfläche

Wird z.B. eine horizontale untere Polygonkante, wie in Bild 2.43 dargestellt, über das Pixel geschoben, so wird die teilweise Überdeckung der Pixelfläche erst festgestellt, wenn die obere Zeile von Subpixelmittelpunkten erreicht ist. Dann erst werden die 4 oberen Subpixel als **innerhalb** gesetzt usw. Dies bewirkt, daß die Helligkeit des Pixels beim Weiterschieben der Kante sich in 4 Helligkeitsstufen ändert (Bild 2.44a). Bei dem Aufwand von 16 berechneten Subpixeln möchte man aber auch eine Grauwertauflösung von 16 Stufen erreichen, wie Bild 2.44c) zeigt. Der gleiche unbefriedigende Effekt tritt bei anderen Kantenneigungen auf. Bild 2.44b) zeigt noch den Fall einer 45°–Kante.

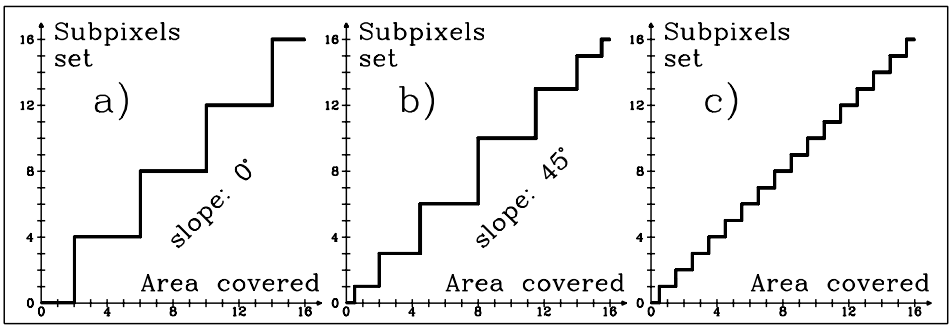


Abbildung 2.44: Zahl der gesetzten Subpixel als Funktion der überdeckten Pixelfläche beim Supersampling:
a) horizontale und vertikale Kanten, b) diagonale Kanten, c) erwünschtes Verhalten

Sind Objektdetails kleiner als ein Subpixel, dann treten beim konventionellen Supersampling besonders unangenehme Fehler auf. Bei dynamischen Darstellungen erscheinen und verschwinden diese Objekte bei ihrer Bewegung über den Bildschirm. Sehr dünne Objekte oder das Ende eines spitzen Dreiecks erscheinen als gestrichelte Linie (Bild 2.45).

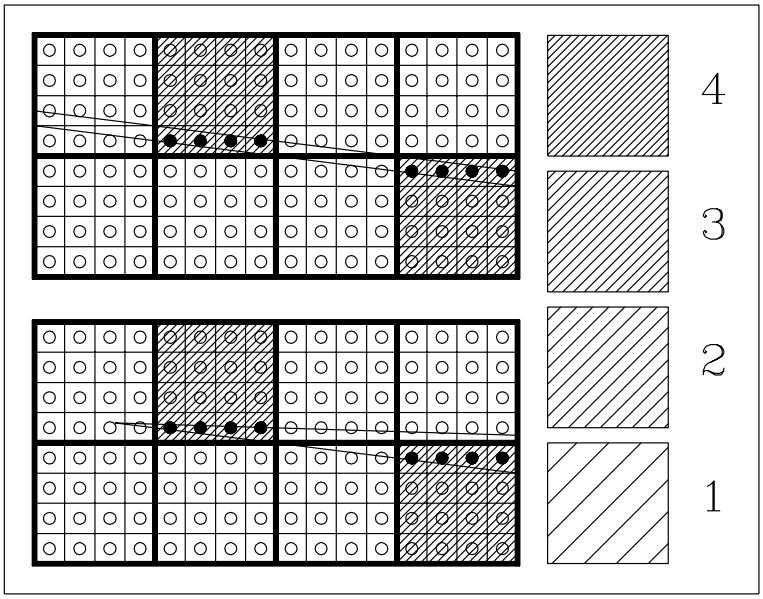


Abbildung 2.45: Probleme beim Supersampling:
gesetzte Subpixel sind schwarz; die Schraffur zeigt den Grauwert abhängig von der Anzahl gesetzter Subpixel.

Eine Möglichkeit zur Milderung dieser Effekte ist stochastisches Sampling [A+85], [Coo86], [D+85]. Allerdings kann das Blinken sich bewegender kleiner Objekte auch hiermit nicht beseitigt werden.

Gute Resultate, die auch in vernünftigem Zusammenhang zum Rechenaufwand stehen, werden durch die exakte Ermittlung der überdeckten Pixelfläche erzielt [Sch91]. Hierbei wird nicht im Mittelpunkt der Subpixel abgetastet, sondern die überdeckte Pixelfläche exakt berechnet und durch die Anzahl der gesetzten Subpixel ausgedrückt. Die Genauigkeit des Resultats entspricht also der Subpixelauflösung der Pixelfläche und demnach dem geleisteten Rechenaufwand.

Im folgenden wird der Algorithmus für (4×4) Subpixel erklärt. Eine Erweiterung auf $n \times n$ ergibt sich einfach.

Der Algorithmus löst die Aufgabe in zwei Schritten:

Zuerst wird die überdeckte Pixelfläche (auf 1/16 genau) berechnet und durch die Anzahl der gesetzten Subpixel angegeben.

Dann wird die Lage der gesetzten Subpixel festgestellt. Hierfür nutzt man die Beobachtung, daß für eine gegebene Subpixelmaske (z.B. 4×4) nur eine geringe Zahl (z.B. 32) unterscheidbarer Reihenfolgen der Subpixelüberdeckung entsprechend unterscheidbarer Kantenneigungen existiert (Bild 2.46).

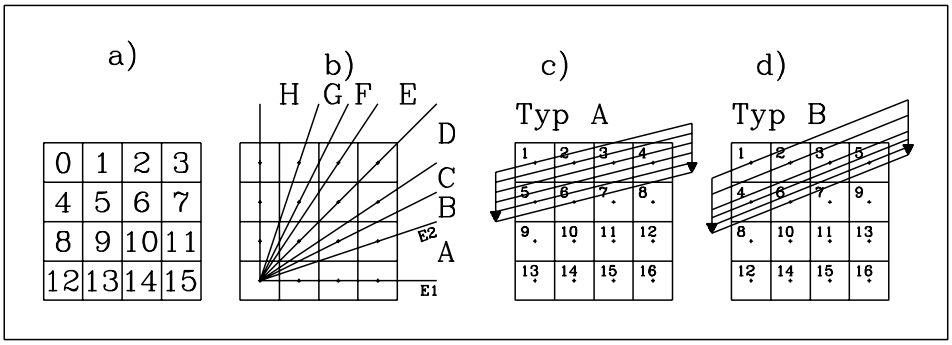


Abbildung 2.46: Numerierungsschema der Subpixel und mögliche Reihenfolge der Subpixelüberdeckung

Beispielsweise sieht man aus Bild 2.46, daß alle Kanten zwischen E_1 und E_2 die Subpixel in der gleichen Reihenfolge überdecken, nämlich wie Typ A.

Man benötigt also Kantenneigung und Abstand zum Pixelmittelpunkt als Eingangsparameter und die zugehörige Pixelmaske als Ausgangsparameter. Dieser Zusammenhang ist nur einmal zu berechnen und kann in einer Tabelle (LUT) abgelegt werden (siehe Bild 2.47).

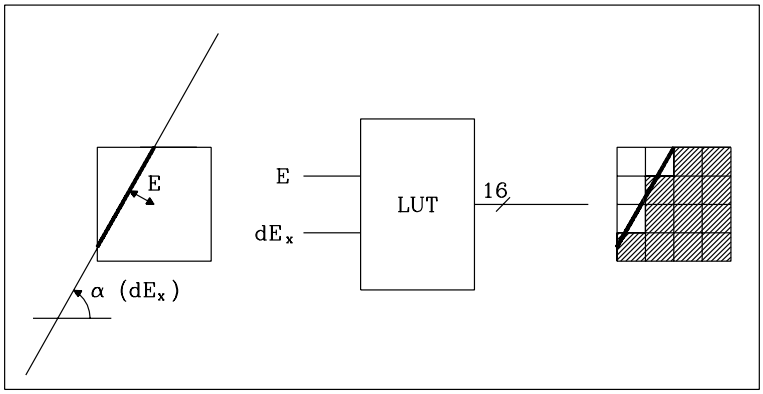


Abbildung 2.47: Tabelle für Subpixelmasken

Zum Aufbau dieser Tabelle wird eine Darstellung der Polygonkante benötigt, die bei vorgegebener Neigung den Abstand eines Pixels zur Polygonkante unmittelbar angibt und eine inkrementelle Berechnung für das Nachbarpixel erlaubt. Dies leistet die Hesse'sche Normalform der Geraden. Dabei ist zu berücksichtigen, daß jetzt auch Pixel **außerhalb** des Polygons betrachtet werden müssen (siehe Bild 2.43).

Sei X,Y das Startpixel der Kante und x,y ein beliebiges Pixel, so kann man als (lineare) Kantenfunktion $E(x,y)$ (Entfernungsfunktion) schreiben (Bild 2.48):

$$E(x,y) = (x-X)\,dE_x + (y-Y)\,dE_y \quad \text{mit} \quad dE_x \sim \frac{\partial E(x,y)}{\partial_x},\ dE_y \sim \frac{\partial E(x,y)}{\partial_y}$$

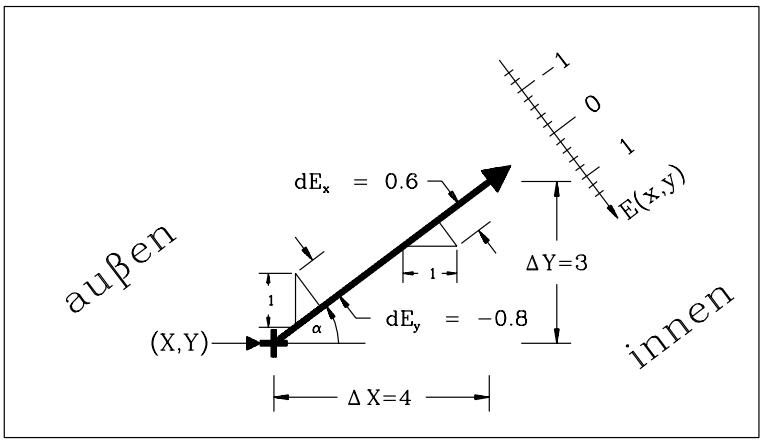


Abbildung 2.48: Definition der Kante

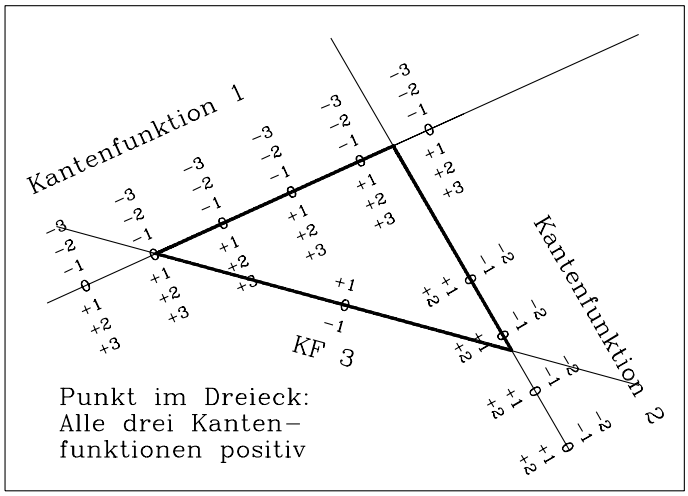


Abbildung 2.49: Feststellung von **innerhalb** und **außerhalb** durch Kantenfunktionen

mit der Bedingung für die Kante:

$$\Delta X \cdot dE_x + \Delta Y \cdot dE_y = 0$$

In Bild 2.48 wurde $dE_x = sin\alpha$ und $dE_y = -cos\alpha$ gesetzt, wodurch diese Bedingung erfüllt ist.

Ist der Wert der Kantenfunktion $E(x,y)$ für ein Pixel (x,y) bekannt, so kann der Wert für ein Nachbarpixel $(x+1,y)$ inkrementell gewonnen werden:

$$E(x+1,y) = E(x,y) + dE_x$$

Mit Hilfe des Vorzeichens von $E(x,y)$ kann entschieden werden, auf welcher Seite der gerichteten Kante ein Pixel liegt. Hierzu genügt es,

$$\begin{array}{lcll} dE_x & = & \Delta Y & \text{bzw.} \\ dE_y & = & -\Delta X & \end{array}$$

zu wählen [Pin88]. Soll $E(x,y)$ den Abstand angeben, dann müssen dE_x und dE_y noch mit der Kantenlänge normiert werden (L_2 – Norm):

$$dE_x = \frac{\Delta Y}{\sqrt{\Delta X^2 + \Delta Y^2}} = \sin\alpha \qquad \text{und}$$
$$dE_y = -\frac{\Delta X}{\sqrt{\Delta X^2 + \Delta Y^2}} = \cos\alpha$$

(Diese Werte in $E(x,y)$ eingesetzt, liefert die Hesse'sche Normalform der Geraden.) Die so berechnete Entfernung $E(x,y)$ gibt allerdings nur für ein kreisförmiges Pixelmodell an, ob das Pixel geschnitten wird (Bild 2.50).

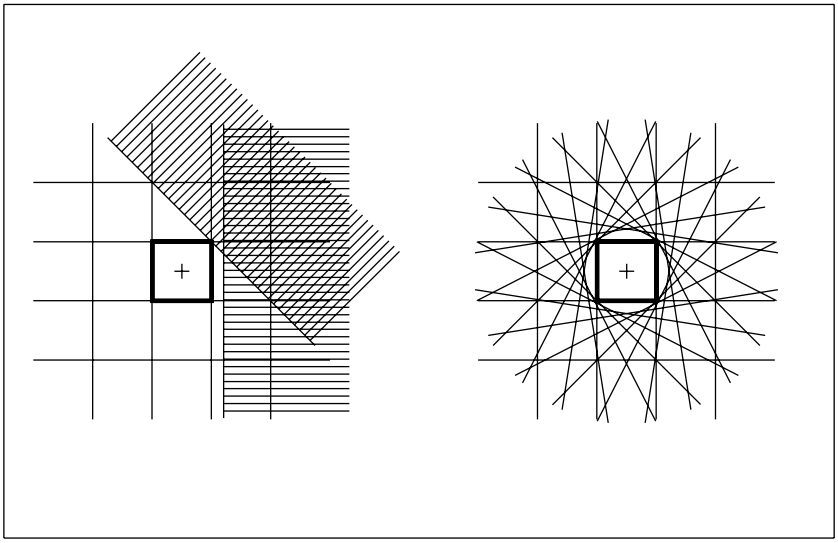


Abbildung 2.50: Alle Kanten mit $E(x,y) = (1/2)^{1/2}$ vom Pixelmittelpunkt (L_2-Norm)

Hier wird aber ein quadratisches Pixelmodell vorausgesetzt. Ein dem Problem angemessenes, winkelabhängiges Verhalten der Entfernungsfunktion $E(x,y)$ liefert die Wahl der sogenannten Manhattan distance oder L_1–Norm zur Normierung:

$$dE_x = \frac{\Delta Y}{|\Delta X| + |\Delta Y|} \qquad \text{und} \qquad dE_y = -\frac{\Delta X}{|\Delta X| + |\Delta Y|}$$

Auf diese Weise wird auch die aufwendige Auswertung der Quadratwurzel vermieden. Bild 2.51 zeigt das Verhalten von $E(x,y) = 1/2$. Man erkennt, daß $E(x,y)$ das gewünschte Entfernungsmaß für das quadratische Pixelmodell liefert.

Zum Abschluß wird die Leistungsfähigkeit des Verfahrens noch mit einigen Ergebnissen belegt. Bild 2.52 zeigt, daß die in Bild 2.45 gezeigten Probleme bei sehr kleinen Objekten gelöst werden können.

Bild 2.53 bestätigt die Leistungsfähigkeit des Verfahrens im Vergleich zu Supersampling bei gleichem Aufwand eindrucksvoll.
Das dritte Bild zeigt das Verfahren von Schilling [Sch91].

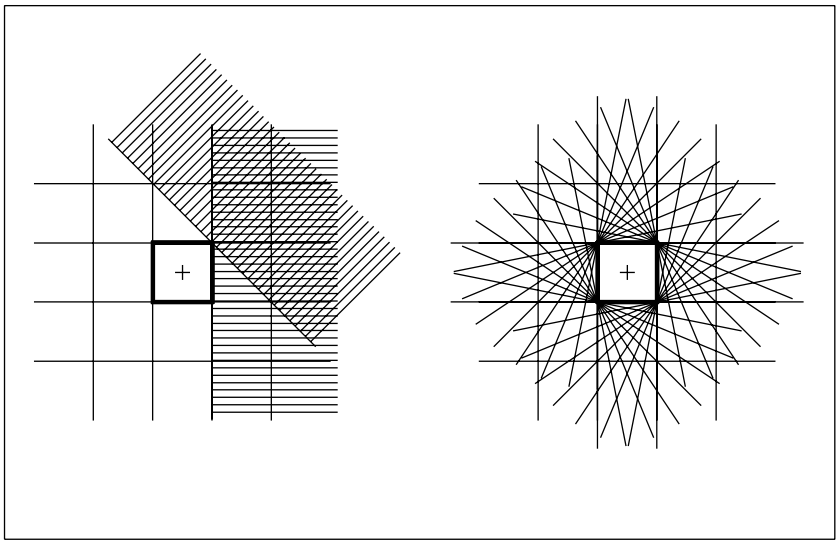


Abbildung 2.51: Alle Kanten mit $E(x,y) = \frac{1}{2}$ vom Pixelmittelpunkt (L_1-Norm)

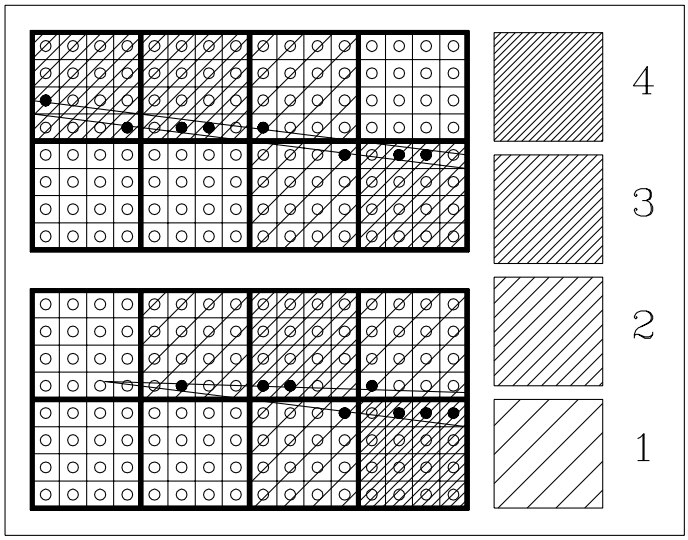


Abbildung 2.52: Probleme aus Bild 2.50 mit $4 \times 4-$ Maske korrekt behandelt

Subpixelmasken können nicht nur zur Glättung eingesetzt werden. In Kapitel 5 'Visibilität' werden sie zur Visibilitätsberechnung verwendet. Dort wird noch gezeigt, daß Anti–aliasing, Visibilität und Transparenz auf der Ebene des einzelnen Pixels gemeinsam betrachtet werden müssen.

Wenn Sie die PDF-Version des Kurses bearbeiten, können Sie jetzt das Interaktionselement 'exact' anwählen.

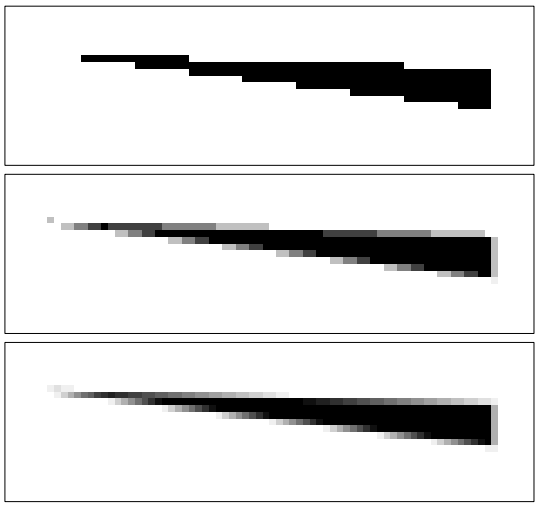


Abbildung 2.53: Leistungsfähigkeit des vorgestellten Verfahrens:
Oben: Spitzes Dreieck ohne Glättung
Mitte: Spitzes Dreieck mit Supersampling
Unten: Spitzes Dreieck mit dem Verfahren von Schilling

2.6 Übungsaufgaben

Aufgabe 1

Welche der Eingabegeräte

a) Tastatur / Keyboard

b) Lichtgriffel / Lightpen

c) Tablett / Tablet

d) Potentiometer / Dials

e) Steuerknüppel / Joystick

f) Maus / Mouse

g) Rollkugel / Trackball

h) Funktionstasten / Function Keys

sind geeignet, um damit die folgenden Aufgaben beim interaktiven Arbeiten mit einem Graphiksystem mit genügender Präzision, jedoch einfach, effizient und elegant zu erfüllen:

– Eingabe eines Punktes

– Eingabe eines Linienzugs / Polyline

– Eingabe einer Kurve

– Auswählen eines Bildobjekts

– Eingabe eines Textes

– Auswahl aus einem Menü von Funktionen

– Hilfe–Anforderung

– Eingabe eines reellen Zahlenwertes

– Bewegen eines Objekts auf dem Schirm

– Eingreifen in eine Simulation

– Eingabe eines dreidimensionalen Objekts

Tragen Sie für jede Aufgabe zu jedem Eingabegerät die Eignung in die Matrix ein: “−” = ungeeignet, “0” = bedingt geeignet, “+” = gut geeignet.

	a	b	c	d	e	f	g	h
Punkteingabe								
Polylineeingabe								
Kurveneingabe								
Objektauswahl								
Texteingabe								
Funktionsmenü								
Hilfe anfordern								
Realwerteingabe								
Objekt bewegen								
Eingriff Simulation								
3D Eingabe								

Aufgabe 2

Gegeben sei die Gerade $y = \frac{5}{9}x + 2$.

a) Tragen Sie den Geradenverlauf im Bereich $x = [0,9]$ unter Benutzung des Bresenham–Algorithmus in ein Pixel–Raster ein.

b) Nennen Sie die Vorteile des Bresenham–Algorithmus.

Aufgabe 3

Ein Kreis ist durch seinen Mittelpunkt **c** und seinen Radius r definiert. Die Aufgabe besteht darin, einen solchen Kreis auf einem Rasterbildschirm zu zeichnen (approximieren). Wir wollen dabei folgenden Algorithmus (Kreisalgorithmus von Bresenham [Bre77]) verwenden:
Sei zunächst einmal $\mathbf{c} = \mathbf{0}$. Jeder Punkt (x,y) hat den euklidischen Abstand

$$d(x,y) = \sqrt{x^2 + y^2} - r$$

vom Kreis $x^2 + y^2 = r^2$. Da diese Funktion aufgrund der Wurzel für schnelle inkrementelle Distanzberechnungen nicht geeignet ist, verwendet Bresenham alternativ die Funktion

$$z = \Delta(x,y) = x^2 + y^2 - r^2.$$

Die Funktion $\Delta(x,y)$ beschreibt ein hyperbolisches Paraboloid. Ein Schnitt mit der Ebene $z = 0$ stellt genau den Ursprungskreis mit Radius r dar. In Abb. 2.54 ist das hyperbolische Paraboloid in der Schnittebene $y = 0$ skizziert. Der Kreis geht in dieser Abbildung durch die Punkte $(r,0)$ und $(-r,0)$, der euklidische Abstand $d(x_i,0)$ entspricht der Länge $|x_i - r|$, und der von Bresenham alternativ verwendete "Abstand" $\Delta(x_i,y_i)$ entspricht der Länge der eingezeichneten gestrichelten Linien.

Kommen wir nun zum eigentlichen Kreisalgorithmus. Gehen wir von der Annahme aus, daß wir uns im ersten Quadranten befinden und die Pixel im Uhrzeigersinn berechnet werden sollen. War (x,y) das zuletzt gesetzte Pixel, dann gibt es

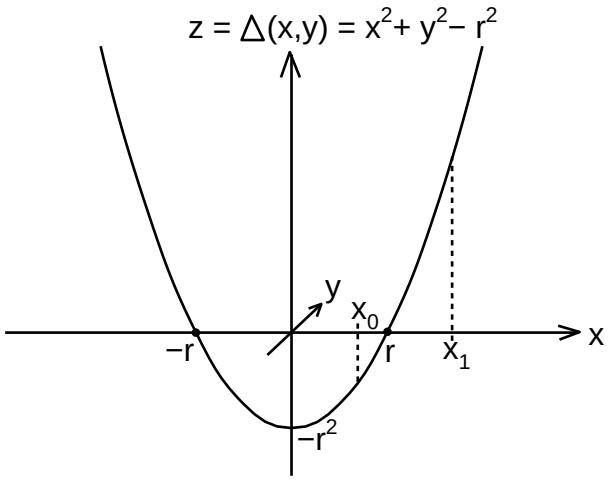


Abbildung 2.54: Schnitt eines hyperbolisches Paraboloids mit der Ebene $y = 0$

für das nächste Pixel nur drei Möglichkeiten: $(x+1,y)$, $(x,y-1)$, $(x+1,y-1)$. Die Abstände dieser Pixel vom Kreis können wie folgt inkrementell berechnet werden:

$$\begin{aligned} \Delta(x+1,y) &= \Delta(x,y)+2x+1 \\ \Delta(x,y-1) &= \Delta(x,y)-2y+1 \\ \Delta(x+1,y-1) &= \Delta(x,y)+2x-2y+2 \end{aligned}$$

Um den Kreis mit Mittelpunkt **c** zu erhalten, wird das Pixel mit dem kleinsten Absolutbetrag von Δ um **c** transliert und gezeichnet.

Aufgabe 4

Ein Rastergerät hat eine Auflösung von 1024×1024 Bildpunkten und ermöglicht farbige Darstellung durch Ansteuerung der drei Farbkathoden (R,G,B). Jede Kathode kann mit 8 Helligkeitsstufen betrieben werden.

a) Wieviele unterschiedliche Farben sind darstellbar?

b) Wie groß muß der Bildwiederholspeicher sein?

c) Wie groß ist die maximal zulässige Zugriffszeit t_{acc} für ein Pixel bei einer Bildwiederholfrequenz von 60 Hz? (Alle Bits eines Pixels werden parallel gelesen; pro Speicherzugriff wird nur 1 Pixel gelesen; die horizontale Rücklaufzeit t_{AH} und die vertikale Rücklaufzeit t_{AV} seien bekannt.)

Aufgabe 5

Gegeben ist ein geschlossenes Polygon, das bereits in den Bildspeicher eingetragen wurde. Das Polygon hat eine gegebene Randfarbe. Kein Pixel im Polygoninneren habe diese Randfarbe. Startet man nun mit einem beliebigen Pixel (x,y)

im Polygoninneren (dem Saatkorn, seed), so läßt sich jedes andere Pixel (x', y') durch eine Kombination von Schritten in östlicher $(x+1)$, westlicher $(x-1)$, nördlicher $(y+1)$ und südlicher $(y-1)$ Richtung vom Pixel (x,y) aus erreichen. Implementieren Sie aufbauend auf dieser Idee einen rekursiven Füllalgorithmus, der sämtliche Pixel im Inneren des Polygons auf eine Füllfarbe setzt.

2.7 Lösungen zu den Übungsaufgaben

Lösung 1

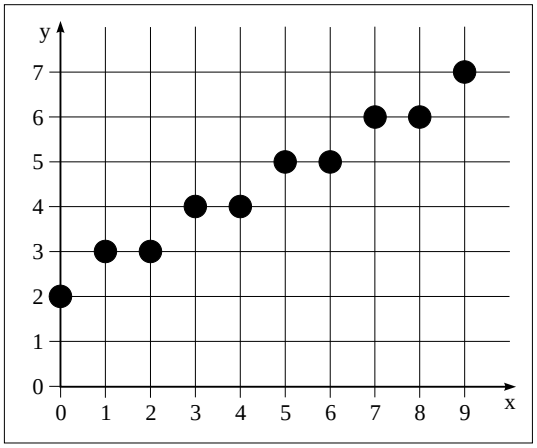
	a	b	c	d	e	f	g	h
Punkteingabe	0	+	+	0	+	+	+	0
Polylineeingabe	0	+	+	0	+	+	+	0
Kurveneingabe	-	0	+	-	0	+	0	-
Objektauswahl	0	+	+	0	0	+	+	0
Texteingabe	+	-	-	-	-	-	-	-
Funktionsmenü	+	+	+	0	+	+	+	+
Hilfe anfordern	+	+	+	0	+	+	+	+
Realwerteingabe	+	0	0	+	0	0	0	0
Objekt bewegen	0	+	+	0	+	+	+	0
Eingriff Simulation	0	+	0	0	+	0	0	+
3D–Eingabe	0	-	-	0	-	-	-	0

Lösung 2

a)

$$
\begin{array}{lcllcl}
y & = & \frac{5}{9}x+2 & & & \\
X_a & = & 0 & Y_a & = & 2 \\
X_e & = & 9 & Y_e & = & 7 \\
\Delta X & = & 9 & \Delta Y & = & 5 \\
2(\Delta Y-\Delta X) & = & -8 & 2\Delta Y & = & 10 \\
E_0' & = & 2\Delta Y-\Delta X=1 & & &
\end{array}
$$

Index	Ausgabe X	Ausgabe Y	neues X	neues Y	neues E'
0	0	2	1	3	-7
1	1	3	2	3	3
2	2	3	3	4	-5
3	3	4	4	4	5
4	4	4	5	5	-3
5	5	5	6	5	7
6	6	5	7	6	-1
7	7	6	8	6	9
8	8	6	9	7	1
9	9	7			



b)
- Nur Integer–Arithmetik erforderlich;
- nur Addition, Subtraktion, Shift benötigt (keine Multiplikation, Division);
- in jedem Iterationsschritt wird genau ein neuer Rasterpunkt generiert (kein Punkt mehrfach);
- die erzeugte Gerade verläuft genau durch den Anfangs– und Endpunkt;
- es werden symmetrische Punktfolgen bzgl. Anfangs– und Endpunkt erzeugt, es gibt Zyklen von Punktfolgen;
- der Abstand zwischen berechneten Rasterpunkten und exakter Gerade ist minimal.

Lösung 3

(**********)

PROCEDURE *k r e i s*

Parameter:

(centerx,centery) *Kreis-Mittelpunkt*

radius *Kreis-Radius*

color *Kreis-Farbe*

Aufgabe:

'kreis' berechnet und zeichnet einen Kreis mit Hilfe des BRESENHAM-Algorithmus.
Dieser arbeitet rein inkrementell, d.h., er kommt völlig ohne Multiplikationen aus.
Wirklich berechnet wird nur ein Oktant des Kreises.
Die 7 anderen Oktanten entstehen durch Spiegelung des berechneten Oktanten an den Haupt- und Diagonalachsen.

BRESENHAM-Algorithmus:

Jeder Punkt (x,y) *der Ebene hat den Abstand*

$$delta = f(x,y) = sqr(x) + sqr(y) - sqr(radius)$$

vom Kreis.
War (x,y) *der zuletzt gezeichnete Punkt, dann gibt es für den nächsten Punkt nur 3 Möglichkeiten:* $(x+1,y)$, $(x,y-1)$, $(x+1,y-1)$.
Die Abstände dieser Punkte vom Kreis berechnen sich wie folgt:

1. *delta := f(x+1,y) = delta + 2 * x + 1*
2. *delta := f(x,y-1) = delta - 2 * y + 1*
3. *delta := f(x+1,y-1) = delta + 2 * x - 2 * y + 2*

Der Punkt mit dem geringsten Abstand wird gezeichnet.

(**********)

PROCEDURE kreis (*centerx, centery, radius, color* : integer);

VAR *x, y, delta, limit*,
deltahorizontal, deltavertical, deltadiagonal,
absdeltahorizontal, absdeltavertical, absdeltadiagonal : integer;

```
(*** SUB-PROCEDURE p l o t 8 ***)
(*** Nur der Punkt im 2. Oktanten wird wirklich berechnet.
Alle übrigen Punkte entstehen aus Spiegelung an der
x-Achse, an der y-Achse und an den beiden Hauptdiagonalen. ***)

PROCEDURE plot8 ( x, y : integer );
BEGIN
    PutPixel ( centerx+y, centery+x, color );
    (*** 1. Oktant *** )
    PutPixel ( centerx+x, centery+y, color );
    (*** 2. Oktant ***)
    PutPixel ( centerx-x, centery+y, color );
    (*** 3. Oktant ***)
    PutPixel ( centerx-y, centery+x, color );
    (*** 4. Oktant ***)
    PutPixel ( centerx-y, centery-x, color );
    (*** 5. Oktant ***)
    PutPixel ( centerx-x, centery-y, color );
    (*** 6. Oktant ***)
    PutPixel ( centerx+x, centery-y, color );
    (*** 7. Oktant ***)
    PutPixel ( centerx+y, centery-x, color );
    (*** 8. Oktant ***)
END;

BEGIN
(*** Initialisierung ***)
    x := 0;
    y := radius;
    delta := 0;
    limit := round(0.5*sqrt(2)*radius);
    plot8 (x,y);

    (*** Schleife über den 2. Oktanten im Uhrzeigersinn,
    d.h. vom Punkt (0,radius) bis zum Punkt: (1/sqrt(2)*radius,1/sqrt(2)*
    radius) ***)

    WHILE x < limit DO
    BEGIN

        deltahorizontal   := delta+2*x+1;
        deltavertical     := delta-2*y+1;
        deltadiagonal     := delta+2*x-2*y+2;

        absdeltahorizontal := abs(deltahorizontal);
        absdeltavertical   := abs(deltavertical);
        absdeltadiagonal   := abs(deltadiagonal);
```

```
            IF absdeltahorizontal <= absdeltavertical THEN
            BEGIN
                IF absdeltahorizontal <= absdeltadiagonal THEN
                BEGIN
                                    x  :=  x+1;
                                delta  :=  deltahorizontal

                END
                ELSE
                (*** absdeltahorizontal > absdeltadiagonal ***)
                BEGIN
                                    x  :=  x+1;
                                    y  :=  y-1;
                                delta  :=  deltadiagonal

                END
            END
            ELSE
            (*** absdeltahorizontal > absdeltavertical ***)
            BEGIN
                IF absdeltavertical <= absdeltadiagonal THEN
                BEGIN
                                    y  :=  y-1;
                                delta  :=  deltavertical

                END
                ELSE
                (*** absdeltavertical > absdeltadiagonal ***)
                BEGIN
                                    x := x+1;
                                    y := y-1;
                                delta := deltadiagonal

                END

            END;
            plot8 (x,y)

        END
    END;
```

Lösung 4

a) Anzahl der Farben $= 8^3 = 512 = 2^{3 \cdot 3} = 2^9$

b) $1024 \times 1024 \times 9 = 9$ MBit

c) $t_{acc} = \dfrac{\frac{\text{sec}}{60} - t_{AV}}{1024} - t_{AH}$

Lösung 5

```
PROCEDURE SeedFill (x,y :integer);
BEGIN
    IF GetPixel(x,y) <> RandFarbe AND GetPixel(x,y) <> Füllfarbe
    THEN
    BEGIN
        PutPixel(x, y, Füllfarbe);
        SeedFill(x + 1, y );
        SeedFill(x - 1, y );
        SeedFill(x , y + 1 );
        SeedFill(x , y - 1 )
    END
END;
```

Literaturverzeichnis

[A+85] G. Abraham et al., *Efficient alias–free rendering using bit–masks and look–up tables*, Computer Graphics 19, 1985, 53–59.

[Bil83] L. M. Bilinov, *Electro–optical and magneto–optical properties of liquid crystals*, Wiley, 1983.

[Bre65] J. E. Bresenham, *Algorithm for Computer Control of a Digital Plotter*, IBM System J, Vol. 4, 1965, Nr. 1, 25–30.

[Bre77] J. E. Bresenham, *A Linear Algorithm for Incremental Digital Display of Circular Arcs*, CACM, Vol. 20, 1977, 100–106.

[Car89] L. Carpenter, *The a–buffer, an antialiasing hidden surface method*, Computer Graphics, Vol. 18, 1989, Nr. 3, 103–108.

[Car91] R. M. Carr, *The point of the pen*, BYTE, Vol. 2, 1991, 211–221.

[Cha77] S. Chandrasekar, *Liquid crystals*, Cambridge University Press, 1977.

[Coo86] R. L. Cook, *Stochastic sampling in computer graphics*, ACM Transactions on Graphics, Vol. 18, 1986, Nr. 3, 51–72.

[D+85] M. A. Z. Dippé et al., *Antialiasing through stochastic sampling*, Computer Graphics, Vol. 19, 1985, Nr. 3, 69–78.

[DIN82] DIN–ISO 7942, *Information Processing – Graphical Kernel System (GKS), Functional Description*, Beuth Verlag GmbH, Berlin, 1982.

[Egl90] H. Eglowstein, *Reach out and touch your data*, BYTE, Vol. 7, 1990, 283–290.

[EKP84] G. Enderle, K. Kansy , G. Pfaff, *Computer Graphics Programming, GKS — The Graphics Standard*, Springer, Berlin, Heidelberg, New York, Tokyo, 1984.

[Eve52] R. R. Everett, *The Whirlwind I Computer*, Rev. Electr. Digital Computing, 1952, 70.

[F+83] E. Fiume et al., *A parallel scan conversion algorithm with anti-aliasing for general purpose Supercomputers*, Computer Graphics, Vol. 17, 1983, Nr. 3, 141–150.

[F+85] H. Fuchs et al., *Fast spheres, shadows, textures, transparencies, and image enhancements in Pixel-planes*, Computer Graphics, Vol. 19, 85, Nr. 3, 111–120.

[FvDFH90] J. D. Foley, A. van Dam, S. T. Feiner , J. F. Hughes, *Computer Graphics - Principles and Practice*, Addison–Wesley, 1990.

[Gra79] Graphics Standard Committee, *Status Report of the Graphics Standard Committee*, Computer Graphics, Vol. 13, 1979, Nr. 3.

[Hug89] A. J. Huges, *Liquid crystal displays in display engineering*, North–Holland, 1989.

[ISO82] ISO DIS 2382/13, *Data Processing Vocabulary — Computer Graphics*, 1982.

[ISO84] ISO/TC97/SC5/WG2/N305, *PHIGS — Programmers Hierarchical Interface to Graphics Systems*, 1984.

[Jac92] D. Jackél, *Grafik Computer Grundlagen, Architekturen und Konzepte computergrafischer Sichtsysteme*, First Ed., Springer Verlag, 1992.

[MS68] T. H. Myer, I. E. Sutherland, *On the Design of Display Processors*, CACM, Vol. 11, 1968, Nr. 6, 410–414.

[NR72] F. Nake, A. Rosenfeld (Eds.), *Graphics Languages*, North Holland Pub. Co, Amsterdam, 1972.

[NS79] W. M. Newman, R. F. Sproull, *Principles of Interactive Computer Graphics*, McGraw-Hill, New York, NY, 1979.

[Pin88] J. Pineda, *A Parallel Algorithm for Polygon Rasterization*, Computer Graphics, Vol. 22, 1988, Nr. 4, 17–20.

[Pro87] W. E. Proebster, *Peripherie von Informationssystemen*, Springer–Verlag, 1987.

[Sch78] G. Schrak, *Graphische Datenverarbeitung*, BI-Wissenschaftsverlag, Mannheim, Wien, Zürich, 1978.

[Sch91] A. Schilling, *A new simple and efficient antialiasing with Subpixel masks*, Computer Graphics, Vol. 25, 1991, Nr. 4, 133–141.

[SIL93] SILICON GRAPHICS INC., *The Open GL Programming Guide*, Addison Wesley, 1993.

[Str91] W. Straßer (Ed.), *Visualisierung* it 2, Oldenbourg–Verlag, München-Wien, April 1991.

[Sut63] I. E. Sutherland, *Sketchpad — A Man-Machine Graphical Communication System*, Spring Joint Comp. Conf. (Baltimore), Spartan Books, Baltimore, 1963.

[The73] R. Theile, *Fernsehtechnik*, Vol. 1, Springer Verlag, 1973.

[TK82] H. J. Tafel, A. Kohl, *Ein– und Ausgabegeräte der Datentechnik*, Hanser–Verlag, München, Wien, 1982.

Index

Glossar

Bildrechner (graphics display system) (2.4, S.40)

Rechner zur Darstellung komplexer graphischer Objekte. Während bei Vektorgeräten die graphischen Objekte direkt aus der Bilddefinition erzeugt und auf dem Bildschirm dargestellt werden, werden bei **Rastergeräten** die darzustellenden Objekte in Rasterpunkte (Pixel) zerlegt. Die zweidimensionale Pixelmatrix wird im Bildspeicher abgelegt. Während bei Vektorgeräten die zur Aufrechterhaltung eines Bildes notwendige Bildwiederholung durch periodisches Auswerten der Bilddefinition erfolgt, wird bei Rastergeräten die Bildwiederholung durch periodisches Auslesen des Bildspeichers realisiert.

Rastergerät (raster display processor) (2.4, S.40)

Bildrechner zur Erzeugung und Darstellung graphischer Objekte
Neben der zentralen CPU, in der die Bilddefinition erzeugt wird, und dem zentralen Speicher enthält ein Rastergerät die folgenden Hauptkomponenten:

- Die Bilderzeugungskomponente (display processor) erzeugt anhand der Bilddefinition die gerasterte Darstellung des Bildes im Bildspeicher.
- Die Bilddarstellungskomponente (Bilddarstellung) erzeugt durch zeilenweises Schreiben auf dem Videomonitor ein stehendes, flimmerfreies Bild.
- Der Bildspeicher ist das Bindeglied zwischen Bilderzeugungs- und Bilddarstellungskomponente.

Für sämtliche Komponenten gibt es eine Vielzahl unterschiedlicher Realisierungs- und Implementierungsmöglichkeiten.

Bresenham-Algorithmus (2.5.1, S.66)

Effizienter Algorithmus zur Rasterung von Strecken. Der Algorithmus arbeitet ausschließlich mit ganzen Zahlen und verwendet lediglich die Operationen Addition, Subtraktion und Shift.

Glätten von Strecken, Glätten von Polygonkanten (2.5.4, S.74)

Die Schwarz-weiß-Darstellung von glatten Strecken führt, abhängig von der Steigung der Strecke, zu mehr oder weniger unregelmäßigen Treppenstufen. Dieser Effekt (aliasing genannt) kann durch die Verwendung von Graustufen gemildert werden. Entsprechende Verfahren werden als Verfahren zum Glätten (anti-aliasing) bezeichnet. Man unterscheidet Verfahren, die auf Pixelebene arbeiten und Verfahren, die auf Subpixelebene arbeiten (over-sampling).

Lehrziele

Nach dem Durcharbeiten sollten Sie verstehen

- warum geometrische Transformationen und ihre effiziente Beschreibung für die Graphische Datenverarbeitung wichtig sind,
- was homogene Koordinaten sind und warum sie in der Graphischen Datenverarbeitung angewandt werden,
- wie die einzelnen Transformationen innerhalb der (4x4)– Matrix definiert werden,
- welcher Zusammenhang zwischen Blickpunkt und Fluchtpunkt in einer perspektivischen Darstellung besteht,
- wie die Transformationsmatrix zu einer gewünschten Transformation bestimmt wird,
- wie Normalen transformiert werden,
- wie eine Kameraposition im Raum definiert wird.

Kapitel 3

Affine und perspektivische Abbildungen

Die Darstellung eines dreidimensionalen Objekts auf dem Bildschirm setzt voraus, daß – dem Photographieren vergleichbar – die Aufnahmebedingungen eingestellt werden können. Das bedeutet z.B. Plazierung und Orientierung des Objekts, Auswahl der Blickrichtung und Aufnahmeentfernung.

Zur Realisierung dieser Funktionen sind Translation, Rotation, Skalierung sowie perspektivische Abbildung und Parallelprojektion als Grundoperationen notwendig. Ebenso notwendig sind diese Funktionen für die Komposition komplexer Szenen aus einfacheren Teilen oder für die Darstellung dynamischer Vorgänge.

Bevor wir jedoch solche Transformationen eingehender untersuchen, wollen wir die mathematischen Grundlagen kurz darstellen.

3.1 Affine Räume

Die Zeichenebene und der uns umgebende Raum können nur dann mit dem $\mathbb{R}^2$ bzw. dem $\mathbb{R}^3$ beschrieben werden, wenn zuvor ein Koordinatensystem festgelegt wird. Üblicherweise haben Objekte und Räume kein “eingebautes” Koordinatensystem; alle Punkte sind gleichberechtigt. Erst bei Vorliegen einer geometrischen Fragestellung werden Koordinatenursprung und -achsen problemspezifisch eingeführt, z.B. rechtwinklige kartesische Koordinaten für eine Polyederwelt. Das hierzu passende mathematische Konzept ist der affine Raum.

3.1.1 Der affine Raum

Eine Menge A^n heißt *n-dimensionaler affiner Raum*, falls ein n-dimensionaler reeller Vektorraum V^n existiert, so daß folgende drei Bedingungen erfüllt sind:

(i) Zu jedem geordneten Paar $(p,q),\ p,q \in A^n$, gehört ein Vektor $v \in V^n$. Man schreibt $v = (\vec{pq})$.

(ii) Zu jedem $p \in A^n$ und jedem $v \in V^n$ existiert ein eindeutig bestimmtes $q \in A^n$, so daß $v = (\vec{pq})$.

(iii) Ist $v = (\vec{pq})$ und $w = (\vec{qr})$, dann gilt $v + w = (\vec{pr})$.

Die Elemente des affinen Raumes A^n heißen *Punkte.*
V^n heißt der zu A^n *assoziierte Vektorraum.*
Die Elemente aus V^n heißen *Vektoren.*
Aus (iii) folgt, daß $(\vec{pp}) = 0$ und $(\vec{pq}) = -(\vec{qp})$.

3.1.1.1 Koordinatensysteme

Eine Menge $(o; e_1, \ldots, e_n)$, bestehend aus einem Punkt $o \in A^n$ und einer Basis $(e_1, \ldots, e_n)$ von V^n heißt *Koordinatensystem* von A^n.
Ist ein Koordinatensystem $(o; e_1, \ldots, e_n)$ gegeben, dann heißt für jeden Punkt $p \in A^n$ der Vektor $v = (\vec{op})$ *Ortsvektor* von p.
Die Komponenten von v bzgl. $(e_1, \ldots, e_n)$ heißen *Koordinaten* von p, d.h. p besitzt die Koordinaten $(x_1, \ldots, x_n)$ genau dann, wenn

$$(\vec{op}) = v = x_1 e_1 + x_2 e_2 + \cdots + x_n e_n. \tag{3.1}$$

Der Punkt o besitzt die Koordinaten $(0, \ldots, 0)$ und heißt *Ursprung* des Koordinatensystems.
Die Punkte p_i mit $(\vec{op_i}) = e_i$ heißen *Einheitspunkte.*
Die Menge der Punkte $(o, p_1, \ldots, p_n)$ eines Koordinatensystems wird als *affine Basis* bezeichnet.

3.1.2 Affine Unterräume

Eine nichtleere Teilmenge $B \subset A^n$ heißt *r-dimensionaler Unterraum* von A^n, falls ein r-dimensionaler Untervektorraum W von V^n existiert, so daß die folgenden beiden Bedingungen erfüllt sind:

(i) $p, q \in B \Rightarrow w = (\vec{pq}) \in W$

(ii) $p \in B, w \in W \Rightarrow$ Es gibt ein $q \in B$ mit $(\vec{pq}) = w$

Dann ist B selbst ein affiner Raum.
Ein $(n-1)$ dimensionaler Unterraum von A^n heißt *Hyperebene.*

Ist ein Punkt $p_0 \in B$ gegeben, so besteht B aus der Menge aller Punkte p, für die $(\vec{p_0 p}) \in W$ ist.

Ist eine Basis $(d_1, \ldots, d_r)$ von W gegeben, so gilt für alle Punkte $p \in B$:

$$(\vec{p_0 p}) = \lambda_1 d_1 + \cdots + \lambda_r d_r \quad (\lambda_i \in \mathbb{R})$$

Bezeichnen wir $(o\vec{p}_0)$ mit v_0 und $(o\vec{p})$ mit v, so gilt

$$v = v_0 + \lambda_1 d_1 + \cdots + \lambda_r d_r.$$

Seien p_i die Punkte mit $(p_0\vec{p}_i) = d_i$, $i = 1,\ldots,r$ und bezeichnen wir $(o\vec{p}_i)$ mit v_i, so gilt:

$$d_i = v_i - v_0, \quad i = 1,\ldots,r$$

und damit

$$\begin{aligned} v &= v_0 + \lambda_1(v_1 - v_0) + \cdots + \lambda_r(v_r - v_0) \\ &= (1 - \sum_{i=1}^{r} \lambda_i)v_0 + \lambda_1 v_1 + \cdots + \lambda_r v_r. \end{aligned} \tag{3.2}$$

Setzen wir

$$\lambda_0 = 1 - \sum_{i=1}^{r} \lambda_i, \tag{3.3}$$

so erhalten wir

$$v = \sum_{i=0}^{r} \lambda_i v_i, \tag{3.4}$$

vgl. dazu Abb. 3.1.

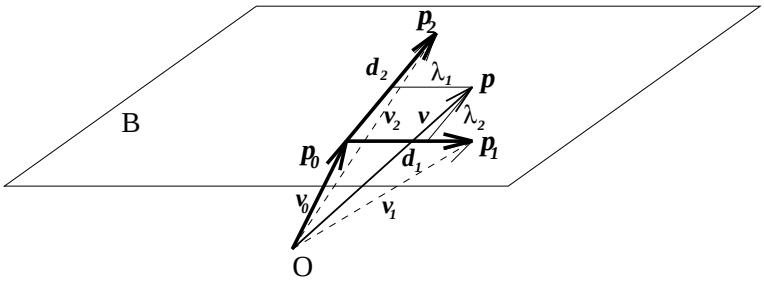


Abbildung 3.1: Affiner Unterraum B

Eine Teilmenge X eines Vektorraumes V heißt *affiner Unterraum* von V, falls es ein $v \in V$ und einen Untervektorraum $W \subset V$ gibt, so daß
$X = v + W = \{u \in V |$ es gibt ein $w \in W$ mit $u = v + w\}$.

Die eindimensionalen Unterräume sind Geraden. Sind zwei verschiedene Punkte p_0 und p_1 gegeben, so gibt es genau eine Gerade g durch die Punkte p_0 und p_1. Ist p ein Punkt von g und seien

$$(o\vec{p}_0) = v_0, (o\vec{p}_1) = v_1 \text{ und } (o\vec{p}) = v, \tag{3.5}$$

so gilt

$$(o\vec{p}) = v = \sum_{i=0}^{1} \lambda_i v_i = \lambda_0 v_0 + \lambda_1 v_1 \tag{3.6}$$

und mit (3.3):

$$v = (1 - \lambda_1)v_0 + \lambda_1 v_1 \tag{3.7}$$

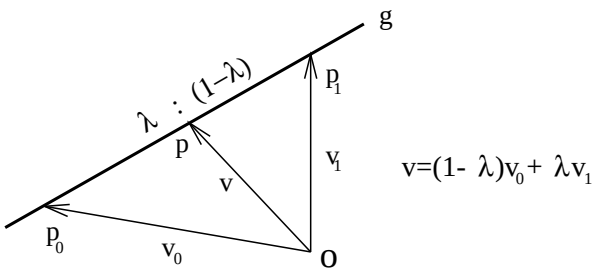


Abbildung 3.2: Eine Affinkombination v der Vektoren v_1 und v_2 definiert einen Punkt p auf der Geraden durch p_0 und p_1

(vgl. Abb. 3.2). Das Geradensegment zwischen p_0 und p_1 wird durch die Menge

$$\{p|(\vec{op}) = v = (1-\lambda)v_0 + \lambda v_1; 0 \leq \lambda \leq 1\} \tag{3.8}$$

beschrieben.

Gilt

$$v = \sum_{i=0}^{n} \lambda_i v_i \quad \text{mit} \quad \sum_{i=0}^{n} \lambda_i = 1 \quad \text{und} \quad v_i = (\vec{op_i}), \tag{3.9}$$

so ist v eine *Affinkombination* der Vektoren v_i bzw. p eine Affinkombination der Punkte p_i.

Liegt das Koordinatensystem fest, so identifiziert man häufig Punkte durch Ortsvektoren und verwendet die Bezeichnungen p_i und v_i synonym. Dies geschieht auch in diesem Text. Operationen auf Punkten des affinen Raumes A^n werden mit Hilfe der entsprechenden Operationen des zugehörigen Vektorraumes V^n definiert.

3.1.2.1 Baryzentrische Koordinaten

Ist in einem affinen Raum A^n ein Koordinatensystem $\mathcal{B} = \{p_0; (\vec{p_0p_1}), \cdots, (\vec{p_0p_n})\}$ gegeben und ist ein Punkt

$$p = \sum_{i=0}^{n} \lambda_i \cdot p_i \quad \text{mit} \quad \sum_{i=0}^{n} \lambda_i = 1$$

als Affinkombination bezüglich dieses Koordinatensystems gegeben, so heißen die Koeffizienten λ_i *baryzentrische Koordinaten* oder *Schwerpunktskoordinaten* von p.

Interpretiert man physikalisch die $n+1$ Punkte p_i als Massepunkte mit den Massen m_i, $\sum_{i=0}^{n} m_i \neq 0$, und die λ_i als normierte Massen

$$\lambda_i = \frac{m_i}{\sum_{j=0}^{n} m_j}, \tag{3.10}$$

so läßt sich der Punkt p als physikalischer Schwerpunkt dieser Massepunkte deuten (vgl. dazu Bild 3.3).

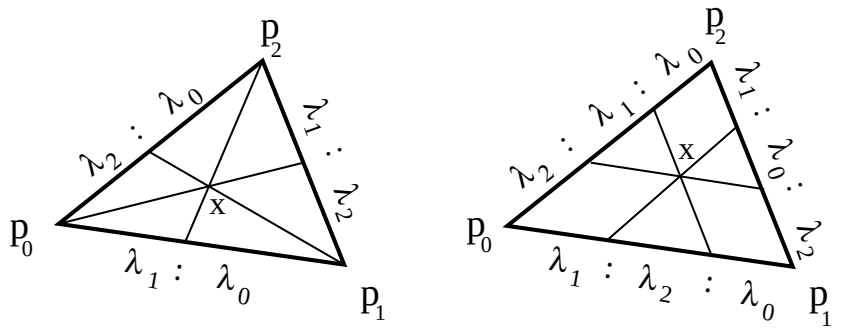


Abbildung 3.3: Baryzentrische Koordinaten eines Punktes $x = \lambda_0 p_0 + \lambda_1 p_1 + \lambda_2 p_2$ im A^2: Die Figuren zeigen die Teilverhältnisse, welche von Geraden durch den Punkt x erzeugt werden. Rechts sind die Geraden mit $\lambda_i = const.$ eingezeichnet.

3.1.2.2 Konvexe Hüllen

Mittels baryzentrischer Koordinaten läßt sich auch die konvexe Hülle von Punkten p_i, $i = 0, \cdots, n$ in einem affinen Raum einfach darstellen.
Die *konvexe Hülle* von Punkten ist die kleinstmögliche Menge von Punkten, bei der mit je zwei Punkten auch die Verbindungsstrecke in der Menge liegt.
Ein einfaches Beispiel ist die eingangs erwähnte Gerade g durch zwei Punkte p_1 und p_2. Ist $0 \leq \lambda \leq 1$ und $p = \lambda \cdot p_1 + (1-\lambda) \cdot p_2$, so liegt p auf der Verbindungsstrecke zwischen p_1 und p_2.
Ein weiteres Beispiel liefern drei verschiedene Punkte p_0, p_1, p_2 in einer affinen Ebene. Die konvexe Hülle dieser Punkte ist das Dreieck mit den Eckpunkten p_0, p_1, p_2.
Allgemein gilt für die konvexe Hülle $co\{p_0, \cdots, p_n\}$ der Punkte $p_0, \cdots, p_n$:

$$co\{p_0, \cdots, p_n\} = \{p \,|\, p = \sum_{i=0}^{n} \lambda_i p_i, \ \sum_{i=0}^{n} \lambda_i = 1 \quad \text{und} \quad \lambda_i \geq 0, i = 0, \cdots, n\}.$$

3.1.2.3 Euklidischer Raum

Alle bisher eingeführten Konzepte des affinen Raumes sind unabhängig von einer Metrik in V^n.
Ist jetzt V^n ein n-dimensionaler Raum mit *Skalarprodukt,* so heißt A^n *euklidischer Raum.* (Für V^n gibt es dann eine orthonormale Basis $(e_1, \ldots, e_n)$.)

3.1.3 Affine Abbildungen

Zu den affinen Räumen gehören affine Abbildungen. Eine Abbildung Φ zwischen zwei affinen Räumen A_1 und A_2 ist *affin,* wenn sie Affinkombinationen erhält. In diesem Sinn ist eine Abbildung

$$\Phi \colon A_1 \to A_2 \tag{3.11}$$

genau dann affin, wenn

$$\Phi\left(\sum_{i=0}^{n}\lambda_i\cdot p_i\right)=\sum_{i=0}^{n}\lambda_i\cdot\Phi(p_i) \tag{3.12}$$

für jede endliche Folge $\lambda_0,\cdots,\lambda_n\in\mathbb{R}$ mit $\sum_{i=0}^{n}\lambda_i=1$ gilt.

Aus der Definition folgt, daß eine affine Abbildung eindeutig durch die Abbildung der affinen Basis festgelegt ist (siehe Bild 3.4).

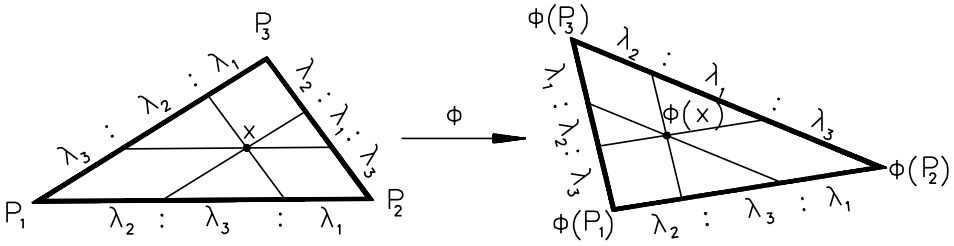


Abbildung 3.4: Affine Koordinatensysteme $\{P_1,P_2,P_3\}$, $\{\Phi(P_1),\Phi(P_2),\Phi(P_3)\}$ und eine affine Abbildung

Sind zwei affine Unterräume $A_1=w_1+V_1$ und $A_2=w_2+V_2$ eines Vektorraumes gegeben, so läßt sich jede affine Abbildung $\Phi\colon A_1\to A_2$ in der Form

$$\Phi(w_1+v)=\Phi(w_1)+\Psi(v) \tag{3.13}$$

schreiben, wobei $\Psi\colon V_1\to V_2$ eine lineare Abbildung ist. Daher läßt sich jede affine Abbildung als Zusammensetzung einer linearen Abbildung und einer Translation schreiben.

Da affine Abbildungen also im allgemeinen aus einem nichtverschwindenden Translationsanteil und einem linearen Anteil bestehen, lassen sich affine Abbildungen des 3-dimensionalen Raumes normalerweise nicht durch 3×3-Matrizen darstellen. Eine Darstellung als 3×3-Matrix ist nur dann möglich, wenn der Translationsanteil verschwindet, d.h. wenn der Ursprung fest bleibt. Es gibt jedoch die Möglichkeit, affine Abbildungen als spezielle projektive Abbildungen, die wir mit 4×4-Matrizen beschreiben können, aufzufassen.

3.2 Projektive Räume

Eine geometrische Grundaufgabe ist die Berechnung von Zentralprojektionen. Mathematisch gesehen sind Zentralprojektionen sogenannte projektive Abbildungen. Sie können daher nur im Kontext projektiver Räume richtig verstanden werden.

Als Beispiel betrachten wir eine Zentralprojektion in der Ebene. Dazu sei ein Beobachtungspunkt q (Punkt des Beobachters), eine Gerade g, auf die projiziert wird, und eine dazu nichtparallele Gerade g' als Objekt gegeben. Der Beobachtungspunkt q soll dabei auf keiner der beiden Geraden g bzw. g' liegen. Nun wird

einem Punkt $p' \in g'$ (Objektpunkt) der Schnittpunkt $p \in g$ der Geraden durch p' und q zugeordnet (siehe Bild 3.5). Die Punkte auf der Objektgeraden g' werden dabei auf die Gerade g projiziert.

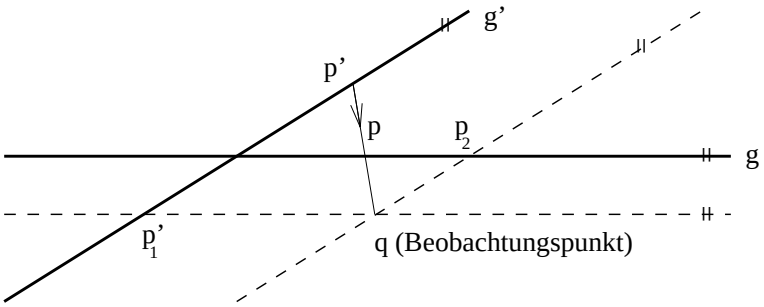


Abbildung 3.5: Zentralprojektion mit Zentrum q (Beobachtungspunkt) von der Geraden g' auf die Gerade g
Dabei besitzt der Punkt p'_1 keinen Bildpunkt und der Punkt p_2 keinen Urbildpunkt.

Bei dieser Konstruktion ist der Bildpunkt p nicht definiert, wenn die Gerade durch p' und q parallel zu g ist.
Umgekehrt kommt der Punkt $p \in g$ nicht als Bildpunkt vor, für den die Gerade durch p und q parallel zu g' ist.
Solche Situationen können in allen Systemen auftreten, in denen mit Projektionen gearbeitet wird. Im projektiven Raum umgeht man diese Ausnahmesituationen.

Bevor wir jedoch analog zu den affinen Räumen eine Definition des projektiven Raumes geben, erläutern wir die Grundkonstruktion für die projektive Ebene.

3.2.1 Die Konstruktion der projektiven Ebene

Sind zwei nichtparallele Geraden g_1, g_2 in der affinen Ebene A^2 gegeben, so haben diese Geraden einen gemeinsamen Schnittpunkt. Parallele Geraden schneiden sich in der affinen Ebene nicht, sie besitzen jedoch eine gemeinsame Richtung. Natürlich sind Richtungen keine affinen Punkte. In einer affinen Ebene A^2 können wir jedoch die Menge aller Richtungen R von Geraden in A^2 definieren. Die Menge aller affinen Punkte $P = A^2$ und die Menge R aller Richtungen von Geraden in dieser Ebene vereinigen wir und erhalten die Menge $P' = P \cup R$. Diese neue Menge P' ist eine Obermenge der ursprünglichen affinen Ebene A^2. Die Elemente dieser Menge nennen wir auch wieder Punkte. Zur Unterscheidung nennen wir Punkte aus P *affine Punkte* und die Punkte aus R *uneigentliche* Punkte oder *Fernpunkte*.

Eine Gerade in P' ist entweder eine affine Gerade, vergrößert um den uneigentlichen Punkt ihrer Richtung, oder die Menge R. Die Menge R wird auch *uneigentliche* Gerade oder *Ferngerade* genannt.

Für die Inzidenzbeziehung von Punkt und Gerade (Wann liegt ein Punkt auf einer Geraden?) bezüglich der Menge P' trifft man folgende Vereinbarungen:

1. Ist sowohl der Punkt als auch die Gerade affin, so gelten dieselben Inzidenz-

beziehungen wie in der affinen Ebene.

2. Sind sowohl der Punkt als auch die Gerade uneigentlich, so liegt der uneigentliche Punkt auf der uneigentlichen Geraden.

3. Ist die Gerade affin und der Punkt uneigentlich, so liegt der Punkt genau dann auf der Geraden, wenn die Richtung der Geraden mit dem gegebenen uneigentlichen Punkt übereinstimmt.

4. Ist umgekehrt der Punkt affin und die Gerade uneigentlich, so liegt keine Inzidenz vor.

Aus diesen Eigenschaften folgt, daß je zwei Geraden genau einen Schnittpunkt besitzen und je zwei Punkte auf genau einer Geraden liegen.

3.2.2 Der projektive Raum

Nach diesen Vorbetrachtungen kommen wir nun zum Begriff des projektiven Raumes.
Ist ein *reeller affiner Raum A*, z.B. der $\mathbb{R}^3$, gegeben, so nennen wir die Menge aller Geraden durch den Ursprung den *reellen projektiven Raum* $P(A)$.
Die Dimension $dimP(A)$ des projektiven Raumes $P(A)$ wird

$$dimP(A) = dim(A) - 1$$

gesetzt.
Ein projektiver Raum der Dimension eins bzw. zwei heißt projektive Gerade bzw. projektive Ebene.
Projektive lineare Teilräume des projektiven Raumes $P(A)$ werden ausgehend von einem linearen Teilraum des zu A korrespondierenden Vektorraumes von A analog definiert. Wir beschränken uns hier auf den Fall $V = \mathbb{R}^4$.

3.2.2.1 Homogene Koordinaten

Die Elemente des dreidimensionalen reellen projektiven Raums $P(\mathbb{R}^4)$ sind Geraden durch den Ursprung in dem zugrundeliegenden affinen Raum $\mathbb{R}^4$. Wir werden sie im folgenden als Punkte des $P(\mathbb{R}^4)$ bzw. $P(A^4)$ bezeichnen. Um für diese Punkte eine Darstellung zu finden, führen wir *homogene Koordinaten* ein.
Ist $v \in \mathbb{R}^4$ ein von Null verschiedener Vektor, so definiert v eine Gerade g durch den Ursprung des $\mathbb{R}^4$ mit der Parametrisierung $g(\lambda) = \lambda \cdot v,\ \lambda \in \mathbb{R}$, die wir zur Abkürzung mit $\mathbb{R} \cdot v$ (projektiver Punkt) bezeichnen. Auf diese Weise erhält man eine Abbildung von dem Vektorraum $\mathbb{R}^4$ in den projektiven Raum $P(\mathbb{R}^4)$, genauer

$$\mathbb{R}^4 - \{0\} \to P(\mathbb{R}^4),\ v \longmapsto \mathbb{R} \cdot v.$$

Zwei von Null verschiedene Vektoren v, v' bestimmen genau dann denselben projektiven Punkt, d.h. dieselbe Gerade durch den Ursprung, wenn es ein $\lambda \in \mathbb{R} - \{0\}$ gibt mit $v' = \lambda v$.

Ist $v = (x,y,z,w) \neq 0 \in \mathbb{R}^4$ gegeben, so bezeichnen wir mit

$$[x,y,z,w] = \mathbb{R} \cdot (x,y,z,w) \tag{3.14}$$

die *homogenen Koordinaten* des projektiven Punktes $\mathbb{R} \cdot v$.
Zur Unterscheidung schreiben wir sie in eckigen Klammern. Dabei ist zu beachten, daß immer mindestens eine der Koordinaten x,y,z,w von Null verschieden ist und daß $[x,y,z,w] = [x',y',z',w']$ genau dann gilt, wenn es ein $\lambda \neq 0$ gibt, mit $x' = \lambda x, y' = \lambda y, z' = \lambda z, w' = \lambda w$. (Bild 3.6).

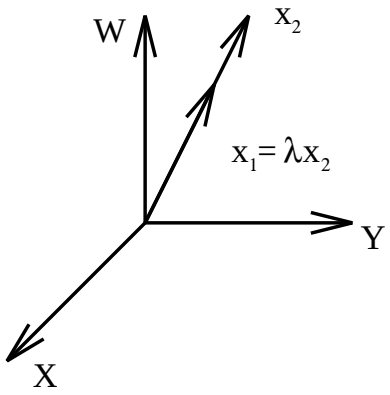


Abbildung 3.6: Die beiden Vektoren $\mathbf{x_1}$ und $\mathbf{x_2}$ definieren dieselbe Gerade im $\mathbb{R}^3$, also denselben projektiven Punkt im $P(\mathbb{R}^3)$

3.2.2.2 Der Zusammenhang zwischen affinem und projektivem Raum

Der affine Raum läßt sich in den projektiven Raum einbetten. Die Abbildung

$$i : \mathbb{R}^3 \to P(\mathbb{R}^4), \quad (x,y,z) \mapsto [x,y,z,1], \tag{3.15}$$

ordnet jedem Punkt des dreidimensionalen affinen reellen Raums einen Punkt im dreidimensionalen projektiven Raum zu. Dieser entspricht einer Geraden im vierdimensionalen affinen reellen Raum. Der Ursprung $(0,0,0)$ des $\mathbb{R}^3$ wird dabei auf den projektiven Punkt mit den homogenen Koordinaten $[0,0,0,1]$ abgebildet.

Bei dieser Abbildung treten projektive Punkte, deren vierte homogene Koordinate Null ist, nicht als Bilder auf. Die Menge

$$H := \{[x,y,z,w] \in P(\mathbb{R}^4) : \ w = 0\}$$

dieser Punkte definiert eine Hyperebene im projektiven Raum, die wir als uneigentliche Hyperebene auffassen können.

Umgekehrt kann man jedem nicht in H enthaltenen projektiven Punkt $p = [x,y,z,w]$ mit $w \neq 0$ einen dreidimensionalen Punkt $p' = (x/w, y/w, z/w)$ zuordnen. Das entspricht einer Zentralprojektion mit dem Ursprung als Zentrum auf den Teilraum $w = 1$. Den reellen dreidimensionalen Raum kann man sich daher bei dieser Konstruktion als die Ebene $w = 1$ im $\mathbb{R}^4$ vorstellen. Die so erhaltene Einbettung des affinen Raumes in den projektiven Raum ist eine Dimension kleiner für den $\mathbb{R}^2$ bzw. $P(\mathbb{R}^3)$ in Bild 3.7 skizziert.

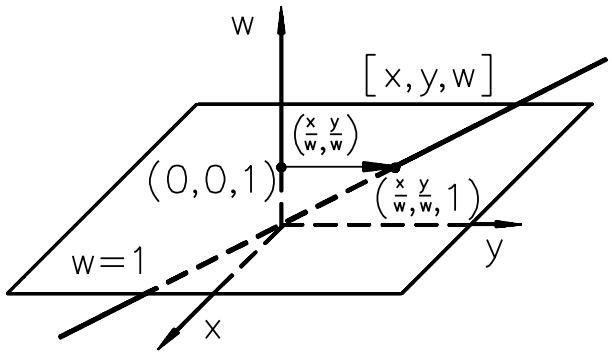


Abbildung 3.7: Einbettung des affinen Raumes $\mathbb{R}^2$ in den projektiven Raum $P(\mathbb{R}^3)$
Den affinen Raum $\mathbb{R}^2$ kann man sich als das Bild des projektiven Raumes $P(\mathbb{R}^3)\backslash\{[x,y,0]\}$ ohne $w=0$ unter einer Zentralprojektion mit dem Ursprung als Zentrum auf die Ebene $w=1$ im $\mathbb{R}^3$ vorstellen.

Die Punkte der Hyperebene H, d.h. Nullpunktsgeraden in der Ebene $w=0$, können als die Bilder der unendlich fernen Punkte des affinen $\mathbb{R}^3$ interpretiert werden. Dieser Sachverhalt läßt sich ebenfalls geometrisch erklären. Für einen beliebigen, aber festen projektiven Punkt $p_0=[x_0,y_0,z_0,0]$ betrachten wir eine affine Gerade g in der Ebene $w=1$, welche durch den Punkt $(0,0,0,1)$ geht und die Richtung des homogenen Koordinatenvektors $[x_0,y_0,z_0,0]$ besitzt (vgl. Bild 3.8). Die Punkte von g können durch $g(t)=(x_0\cdot t, y_0\cdot t, z_0\cdot t, 1)$, $t\in\mathbb{R}$ dargestellt werden. Im projektiven Raum $P(\mathbb{R}^4)$ besitzen die Punkte $g(t)$ von g die homogenen Koordinaten $[x_0\cdot t, y_0\cdot t, z_0\cdot t, 1]=[x_0,y_0,z_0,\frac{1}{t}]$. Daher ist $p_0=[x_0,y_0,z_0,0]$ der Grenzwert für $t\to\infty$ der projektiven Punkte $[x_0,y_0,z_0,\frac{1}{t}]$, d.h. die Nullpunktsgerade p_0 im affinen $\mathbb{R}^4$ liegt in der Ebene $w=0$.

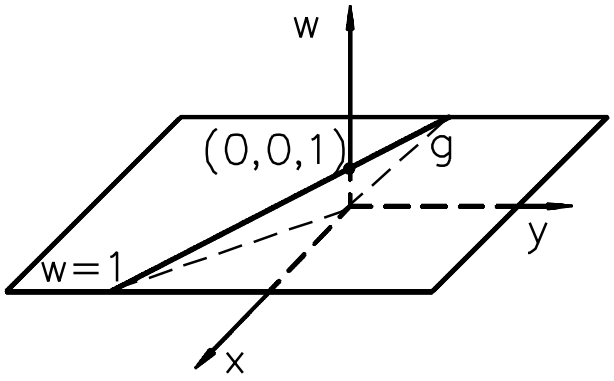


Abbildung 3.8: Eine Gerade g in der Ebene $w=1$ durch den 'Ursprung' $(0,0,1)$

3.2.2.3 Projektive Basis

Ein $(r+1)$-Tupel $(p_0,\cdots,p_r)$ im projektiven Raum $P(\mathbb{R}^4)$ heißt *projektiv unabhängig*, wenn es linear unabhängige Vektoren $v_0,\cdots,v_r\in\mathbb{R}^4$ gibt mit

$$p_i=\mathbb{R}\cdot v_i,\ i=0,\cdots,r.$$

Ein Fünftupel $(p_0,\cdots,p_4)$ von Punkten aus $P(\mathbb{R}^4)$ heißt *projektive Basis* von $P(\mathbb{R}^4)$, wenn je vier davon projektiv unabhängig sind.
Im dreidimensionalen projektiven Raum $P(\mathbb{R}^4)$ ist eine *projektive Basis* durch die von den Vektoren

$$\begin{aligned} v_0 &= [1,0,0,0],\ v_1=[0,1,0,0],\ v_2=[0,0,1,0],\ v_3=[0,0,0,1], \\ v_4 &= [1,1,1,1], \end{aligned}$$

definierten Punkte $p_0,\cdots,p_4$ gegeben, also durch fünf projektive Punkte festgelegt.
Die Notwendigkeit des fünften Basispunktes kann man sich wie folgt veranschaulichen. Jeder der Basisvektoren $v_0,\cdots,v_3\in\mathbb{R}^4$ legt genau einen der projektiven Punkte $p_0,\cdots,p_3\in P(\mathbb{R}^4)$ fest. Diese Punkte sind linear unabhängig und spannen den $P(\mathbb{R}^4)$ auf, doch ist durch die Wahl von vier linear unabhängigen Punkten wegen der homogenen Darstellung der Punkte die Basis $v_0,\cdots,v_3$ nicht eindeutig. Mit v_i stellt auch $v_i'=\rho_i v_i$ denselben Punkt p_i dar. Sucht man nun eine Koordinatendarstellung q eines Punktes $q\in P(\mathbb{R}^4)$,

$$q=\alpha_0 v_0'+\cdots+\alpha_3 v_3'=\alpha_0\rho_0 v_0+\cdots+\alpha_3\rho_3 v_3,$$

so sind die Koordinaten bezüglich $\{v_0',\cdots,v_3'\}$ durch $(\alpha_0,\alpha_1,\alpha_2,\alpha_3)$ gegeben, bezüglich $\{v_0,\cdots,v_3\}$ jedoch durch $(\alpha_0\rho_0,\cdots,\alpha_3\rho_3)$.
Sind die Koeffizienten $\rho_0,\cdots,\rho_3$ verschieden, so unterscheiden sich diese beiden Darstellungen nicht nur durch einen gemeinsamen Faktor. Sie stellen also nicht mehr denselben projektiven Punkt dar (vgl. 3.11). Wird jedoch der fünfte Punkt p_4 mit den Koordinaten

$$v_4=[v_0+v_1+v_2+v_3]=[1,1,1,1]$$

zur Basis hinzugenommen, so müssen in der obigen Darstellung alle ρ_i gleich groß sein, wie es die homogene Darstellung verlangt.

3.2.2.4 Rechenregeln im projektiven Raum

Weil zwei projektive Punkte zwei Geraden im affinen $\mathbb{R}^4$ entsprechen, kann man im projektiven Raum zwei gegebenen Punkten keinen Differenzvektor zuordnen, so wie wir das im affinen Raum gemacht haben. Die Grundoperationen skalare Multiplikation und Vektoraddition lassen sich deshalb auch nach Auszeichnung eines geeigneten Koordinatensystems nicht auf die Elemente projektiver Räume übertragen. Im projektiven Raum kann man die Koordinatenvektoren von Punkten nicht addieren. Aufgrund der homogenen Darstellung der Punkte als Koordinatenvektoren besitzt auch die Skalarmultiplikation im projektiven Raum nicht die Bedeutung der Streckung eines Vektors, wie man sie aus der linearen Algebra kennt. Projektive Punkte können jedoch, der affinen Abbildung affiner Punkte entsprechend, mittels sogenannter projektiver Abbildungen im projektiven Raum bewegt werden.

3.2.3 Projektive Abbildungen

Eine Abbildung f zwischen zwei projektiven Räumen $P(V)$ und $P(W)$ heißt *projektiv,* wenn es eine injektive lineare Abbildung $F : V \to W$ gibt mit

$$f(\mathbb{R}\cdot v) = \mathbb{R}\cdot F(v)$$

für jedes vom Nullvektor verschiedene $v \in V$.
Man schreibt dafür kurz $f = P(F)$.
Zwei lineare Abbildungen $F, F' : V \to W$ definieren dieselbe projektive Abbildung genau dann, wenn es ein $\lambda \neq 0$ gibt, mit $F' = \lambda \cdot F$.
Wie bei linearen und affinen Abbildungen sind auch projektive Abbildungen durch ihre Wirkung auf die Basis eines projektiven Raumes festgelegt. Da der reelle projektive dreidimensionale Raum $P(\mathbb{R}^4)$ fünf Basiselemente besitzt, ist eine projektive Abbildung vom $P(\mathbb{R}^4)$ in den $P(\mathbb{R}^4)$ durch die Abbildung von fünf geeigneten Punkten eindeutig festgelegt.

3.2.3.1 Beschreibung projektiver Abbildungen durch Matrizen

Ist eine projektive Abbildung $f : P(\mathbb{R}^4) \to P(\mathbb{R}^4)$ gegeben, so betrachten wir die zugehörige lineare bijektive Abbildung $F : \mathbb{R}^4 \to \mathbb{R}^4$. Diese Abbildung läßt sich als 4×4-Matrix A beschreiben, die bis auf einen Skalar $\lambda \neq 0$ festgelegt ist. Solche Matrizen heißen *homogen.*
Wird f durch die Matrix

$$A = \begin{pmatrix} a_{00} & \cdots & a_{03} \\ \vdots & & \vdots \\ a_{30} & \cdots & a_{33} \end{pmatrix}$$

dargestellt, so bildet f den Punkt p mit den homogenen Koordinaten $[x,y,z,w]$ auf den Punkt

$$[x,y,z,w]\cdot A = [a_{00}x + \cdots + a_{30}w, \cdots, a_{03}x + \cdots + a_{33}w] \tag{3.16}$$

ab. Diese Abbildung läßt sich vom affinen Standpunkt aus folgendermaßen interpretieren. Der affine Raum $\mathbb{R}^3$ ist in den projektiven Raum $P(\mathbb{R}^4)$ eingebettet, dessen Komplement die Hyperebene

$$H = \{[x,y,z,w] \in P(\mathbb{R}^4) : w = 0\}$$

ist. Für einen projektiven Punkt $p = [x,y,z,w] \in P(\mathbb{R}^4)\backslash H$ erhält man den inhomogenen Koordinatenvektor $(x/w, y/w, z/w)$. Obwohl f nicht notwendig die projektive Hyperebene H in sich überführt, kann man formal die inhomogenen Koordinaten $\frac{x'}{w'}$, $\frac{y'}{w'}$, $\frac{z'}{w'}$ von $f(p) = [x',y',z',w']$ ausrechnen. Es ergibt sich

$$\begin{aligned} \frac{x'}{w'} &= \frac{a_{00}x + a_{10}y + a_{20}z + a_{30}w}{a_{03}x + a_{13}y + a_{23}z + a_{33}w} \\ &\vdots \quad\quad \vdots \\ \frac{z'}{w'} &= \frac{a_{02}x + a_{12}y + a_{22}z + a_{32}w}{a_{03}x + a_{13}y + a_{23}z + a_{33}w}. \end{aligned}$$

Dies ist im allgemeinen eine rationale Transformation des $\mathbb{R}^3$. Die Abbildung f ist genau dann eine affine Abbildung des $\mathbb{R}^3$, wenn alle affinen Punkte auf affine Punkte und alle uneigentlichen Punkte auf uneigentliche Punkte abgebildet werden, d.h. wenn $f(H) = H$ gilt. Wegen

$$f(H) = \{[x,y,z,w] \in P(\mathbb{R}^4) \,|\, a_{03}x + a_{13}y + a_{23}z + a_{33}w = 0\} \tag{3.17}$$

folgt $a_{03} = a_{13} = a_{23} = 0$. Eine affine Abbildung wird daher durch eine homogene Matrix A der Form

$$A = \begin{pmatrix} a_{00} & a_{01} & a_{02} & 0 \\ a_{10} & a_{11} & a_{12} & 0 \\ a_{20} & a_{21} & a_{22} & 0 \\ a_{30} & a_{31} & a_{32} & a_{33} \end{pmatrix}$$

dargestellt.

3.2.3.2 Invariante Eigenschaften affiner und projektiver Abbildungen

Zum besseren Verständnis der affinen und projektiven Abbildungen stellen wir hier noch die wichtigsten charakteristischen Eigenschaften dieser Abbildungen zusammen.
Affine Abbildungen besitzen folgende wichtige Eigenschaften:

1. Beschränkte Objekte bleiben beschränkt.
2. Parallele Objekte bleiben parallel.
3. Die Verhältnisse von Längen, Ebenen und Volumina bleiben unverändert. Liegen drei Punkte a, b, c auf einer Geraden, so bleibt insbesondere ihr Teilverhältnis $TV(a,b,c) = d(a,c)/d(b,c)$ erhalten. Dabei bezeichne $d(x,y)$ den Abstand der beiden Punkte x und y.

Die allgemeineren projektiven Abbildungen besitzen weniger Invarianzeigenschaften:

1. Projektive Geraden werden auf projektive Geraden, projektive Ebenen auf projektive Ebenen abgebildet.
2. Die Reihenfolge von Punkten auf einer projektiven Geraden bleibt erhalten.

Literatur zur Theorie von affinen und projektiven Räumen findet man in Lehrbüchern der linearen Algebra, wie z.B. [Fis79] oder [Sch76]. Das Buch von Penna und Patterson [PP86] behandelt das Thema projektiver Räume speziell unter dem Gesichtspunkt der Computergraphik, ebenso der Artikel von Herman [Her89]. Der Artikel von De Rose [DeR89] zeigt, daß dieselben algebraischen, koordinatenbezogenen Operationen je nach Kontext verschieden interpretiert werden können. Um diese Mehrdeutigkeiten zu vermeiden, wird ein koordinatenfreier Zugang, basierend auf affiner Geometrie, zur Beschreibung geometrischer Operationen entwickelt.

3.3 Affine und projektive Abbildungen in Matrizenschreibweise

Im folgenden werden wir alle Abbildungen, sowohl die affinen als auch die projektiven, durch homogene 4×4-Matrizen beschreiben. Die Gründe sind folgende:

1. Die einheitliche Darstellung erleichtert die Implementierung.

2. Die Hintereinanderausführung verschiedener Transformationen erfordert nur die Multiplikation der Transformationsmatrizen. Denn

$$\begin{aligned} p' &= p \cdot A_1 \\ p'' &= p' \cdot A_2 \\ &\vdots \\ p^n &= p^{n-1} \cdot A_n \end{aligned}$$

ergibt durch Einsetzen

$$p^n = p \cdot A_1 \cdot A_2 \cdot \ldots \cdot A_n.$$

Soll nicht nur ein einzelner Punkt transformiert werden, so ist der Rechenaufwand meist erheblich geringer, wenn erst eine einzige kombinierte Transformationsmatrix als Produkt der einzelnen Transformationen berechnet wird. Dies geschieht einmal, sodann werden alle Punkte entsprechend der resultierenden Matrix transformiert.

3.3.1 Translationen

Die Gleichung $p' = T(p) = T(0) + p$ beschreibt eine Translation T im reellen dreidimensionalen affinen Raum. Mit $T(0) = (x_0, y_0, z_0)$ ergibt sich

$$(x', y', z') = T((x, y, z)) = (x, y, z) + (x_0, y_0, z_0) = (x + x_0, y + y_0, z + z_0), \quad (3.18)$$

d.h. die zu T gehörende lineare Abbildung (siehe 3.13) ist die Identität. Die zur Translation gehörende homogene Matrix sieht dann wie folgt aus:

$$[x', y', z', w'] = [x, y, z, 1] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ x_0 & y_0 & z_0 & 1 \end{bmatrix} \text{ oder} \quad (3.19)$$

$$p' = p \cdot A. \quad (3.20)$$

Dabei wurde $w = w_0 = 1$ gewählt.
Wollen wir die Matrix für beliebige homogene Koordinatenvektoren $[x, y, z, w]$, $[x_0, y_0, z_0, w_0]$ des Punktes p und des Translationsvektors $T(0)$ mit von Null ver-

schiedenen Koeffizienten w und w_0 berechnen, so können wir dieses Resultat verwenden:

$$[x',y',z',w'] = \left[\frac{x}{w},\frac{y}{w},\frac{z}{w},1\right]\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \frac{x_0}{w_0} & \frac{y_0}{w_0} & \frac{z_0}{w_0} & 1 \end{bmatrix} \tag{3.21}$$

Multiplizieren wir den Punkt p mit w und die Matrix A mit w_0, so ändert sich wegen der homogenen Darstellung der Punkte und Abbildungen nichts.

Wir erhalten

$$[x',y',z',w'] = [x,y,z,w]\begin{bmatrix} w_0 & 0 & 0 & 0 \\ 0 & w_0 & 0 & 0 \\ 0 & 0 & w_0 & 0 \\ x_0 & y_0 & z_0 & w_0 \end{bmatrix}. \tag{3.22}$$

3.3.2 Skalierung, Scherung und Rotation

Diese affinen Abbildungen lassen den Ursprung invariant. Sie besitzen daher keinen Translationsanteil. Es ist möglich, diese Abbildungen als 3×3-Matrizen darzustellen. Die zugehörigen homogenen Matrizen dieser Abbildungen sind daher von der Form

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & 0 \\ a_{21} & a_{22} & a_{23} & 0 \\ a_{31} & a_{32} & a_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{3.23}$$

Dabei bestimmen die Bilder der affinen Basisvektoren $(1,0,0),(0,1,0),(0,0,1)$ eine lineare Abbildung $A:\mathbb{R}^3\to\mathbb{R}^3$ vollständig. Multipliziert man die Basisvektoren mit A, so stellen deren Bilder die Zeilen der A beschreibenden Matrix dar. Aufgrund dieses Zusammenhanges lassen sich die Matrizen zur Beschreibung von Skalierung, Scherung und Rotation unmittelbar aus den zugehörigen Bildern ablesen.

3.3.2.1 Skalierung

Bei einer Skalierung S ergibt sich für die affinen Basisvektoren folgende Beziehung (vgl. Bild 3.9).

$$\begin{aligned} S((1,0,0)) &= (s_1,0,0) \\ S((0,1,0)) &= (0,s_2,0) \\ S((0,0,1)) &= (0,0,s_3). \end{aligned} \tag{3.24}$$

Die zugehörige 3×3 Matrix ergibt sich daher zu

$$\begin{pmatrix} s_1 & 0 & 0 \\ 0 & s_2 & 0 \\ 0 & 0 & s_3 \end{pmatrix} \tag{3.25}$$

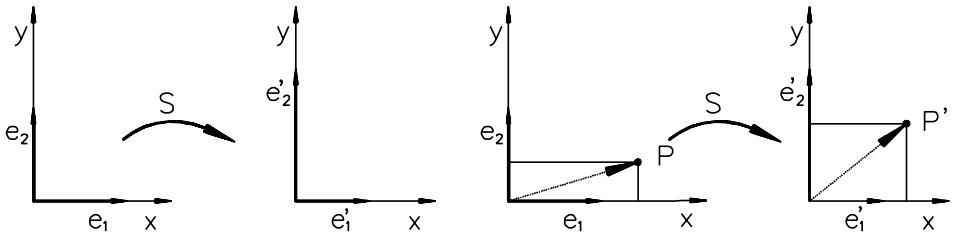


Abbildung 3.9: Skalierung: $e'_1 = s_1 \cdot e_1,\ e'_2 = s_2 \cdot e_2$

und in homogenen Koordinaten

$$[x',y',z',w'] = [x,y,z,w] \begin{bmatrix} s_1 & 0 & 0 & 0 \\ 0 & s_2 & 0 & 0 \\ 0 & 0 & s_3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{3.26}$$

Der Sonderfall $s_1 = s_2 = s_3 = s$ bedeutet die gleiche Skalierung für alle Koordinaten. Die zugehörige homogene Matrix hat dann die Form

$$\begin{bmatrix} s & 0 & 0 & 0 \\ 0 & s & 0 & 0 \\ 0 & 0 & s & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & \frac{1}{s} \end{bmatrix}. \tag{3.27}$$

3.3.2.2 Scherungen

Bei einer Scherung *SH* ergibt sich für die affinen Basisvektoren folgende Beziehung (vgl. Bild 3.10).

$$\begin{array}{lcl} SH((1,0,0)) & = & (1,s_1,s_3) \\ SH((0,1,0)) & = & (s_2,1,s_4) \\ SH((0,0,1)) & = & (s_5,s_6,1) \end{array}. \tag{3.28}$$

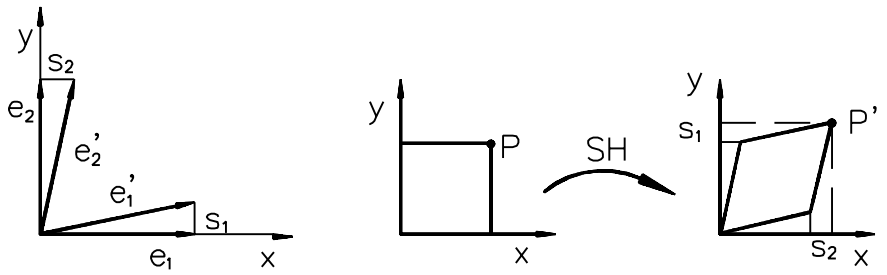


Abbildung 3.10: Scherung in der Ebene: $e'_1 = e_1 + s_1 \cdot e_2, e'_2 = s_2 \cdot e_1 + e_2$

Die zugehörige 3×3 Matrix wird daher zu

$$\begin{pmatrix} 1 & s_1 & s_3 \\ s_2 & 1 & s_4 \\ s_5 & s_6 & 1 \end{pmatrix}. \tag{3.29}$$

In homogenen Koordinaten folgt

$$[x',y',z',w'] = [x,y,z,w] \begin{bmatrix} 1 & s_1 & s_3 & 0 \\ s_2 & 1 & s_4 & 0 \\ s_5 & s_6 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{3.30}$$

3.3.2.3 Rotationen

Bevor wir die Matrix für Rotation um eine beliebige Achse angeben, bestimmen wir die Matrizen für die Rotationen um die Koordinatenachsen.
Bei einer Rotation R_α um die z-Achse ergibt sich für die affinen Basisvektoren folgende Beziehung (vgl. Bild 3.11):

$$\begin{aligned} R_\alpha((1,0,0)) &= (\cos\alpha, \sin\alpha, 0) \\ R_\alpha((0,1,0)) &= (-\sin\alpha, \cos\alpha, 0) \\ R_\alpha((0,0,1)) &= (0,0,1). \end{aligned} \tag{3.31}$$

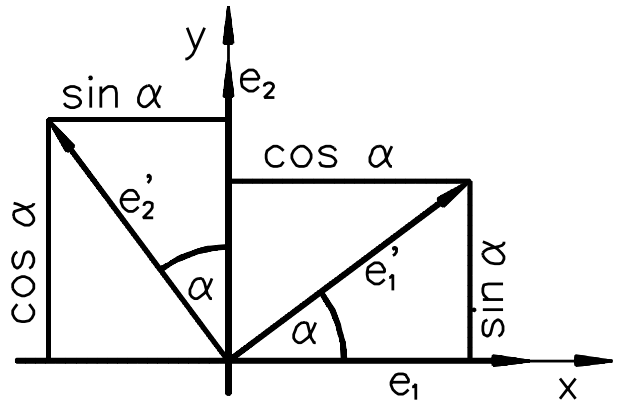


Abbildung 3.11: Rotation mit dem Winkel α um die z-Achse:
$e'_1 = \cos\alpha \cdot e_1 + \sin\alpha \cdot e_2,\ e'_2 = -\sin\alpha \cdot e_1 + \cos\alpha e_2,\ e'_3 = e_3$

Die zugehörige 3×3 Matrix ergibt sich daher zu

$$\begin{pmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}, \tag{3.32}$$

und in homogenen Koordinaten folgt für die Rotation R_α um die z-Achse

$$[x',y',z',w'] = [x,y,z,w] \begin{bmatrix} \cos\alpha & \sin\alpha & 0 & 0 \\ -\sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{3.33}$$

Bei Rotation R_α um die x- bzw. y-Achse ergeben sich analog folgende homogene Darstellungen:

Drehung mit dem Winkel α um die x-Achse:

$$[x',y',z',w'] = [x,y,z,w] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha & 0 \\ 0 & -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.34}$$

Drehung mit dem Winkel α um die y-Achse:

$$[x',y',z',w'] = [x,y,z,w] \begin{bmatrix} \cos\alpha & 0 & -\sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ \sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.35}$$

Die Drehwinkel sind in dem Rechtssystem x,y,z immer positiv (vgl. Bild 3.12).

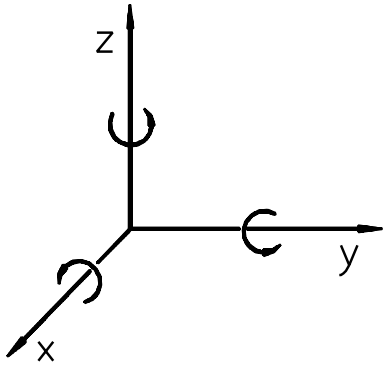


Abbildung 3.12: Festlegung der Drehwinkel

Eine Drehung um eine beliebige Achse in Richtung des *normierten* Vektors (x,y,z) mit dem Winkel α kann auf Drehungen um die Koordinatenachsen zurückgeführt werden. Verknüpft man diese Drehungen, d.h. multipliziert die zugehörigen Rotationsmatrizen, so ergibt sich folgende homogene Darstellung:

$$\begin{bmatrix} tx^2+c & txy+sz & txz-sy & 0 \\ txy-sz & ty^2+c & tyz+sx & 0 \\ txz+sy & tyz-sx & tz^2+c & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{3.36}$$

wobei $s = \sin\alpha$, $c = \cos\alpha$, $t = 1-\cos\alpha$.
Für kleine Winkel α ($\alpha < 1°$) kann man die Bogenlängen $\sin\alpha$ durch α und $\cos\alpha$ durch 1 approximieren:

$$[x',y',z',w'] = [x,y,z,w] \begin{bmatrix} 1 & z\alpha & -y\alpha & 0 \\ -z\alpha & 1 & x\alpha & 0 \\ y\alpha & -x\alpha & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{3.37}$$

Die hier diskutierten Rotationen lassen den Ursprung fest, d.h. es sind Rotationen um Achsen durch den Ursprung. Soll die Rotationsachse durch einen anderen Punkt verlaufen, so kann man die gesuchte Transformationsmatrix durch Verschiebung des Rotationszentrums in den Ursprung, anschließende Rotation und

Zurückverschiebung in das Rotationszentrum konstruieren:

$$p' = p \cdot T^{-1} \cdot R \cdot T. \tag{3.38}$$

Beispiel 3.1:

Gesucht ist eine Rotation in positiver Richtung um eine Achse durch den Punkt (x_0, y_0, z_0) mit dem Winkel α. Die Richtung der Rotationsachse sei die z-Richtung.

$$\begin{aligned} p' &= p \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -x_0 & -y_0 & -z_0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos\alpha & \sin\alpha & 0 & 0 \\ -\sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \\ &\quad \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ x_0 & y_0 & z_0 & 1 \end{bmatrix} \\ &= p \cdot \begin{bmatrix} \cos\alpha & \sin\alpha & 0 & 0 \\ -\sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ x_0 - x_0\cos\alpha + y_0\sin\alpha & y_0 - x_0\sin\alpha - y_0\cos\alpha & 0 & 1 \end{bmatrix}. \end{aligned}$$

Führt man die hier in den Abbildungen diskutierten Schritte nacheinander aus, so ist zu beachten, daß die Matrizenmultiplikation im allgemeinen nicht kommutativ ist und eine andere Reihenfolge auch ein anderes Ergebnis liefert. Alle besprochenen Rotationen lassen sich mit Hilfe von Quaternionen, den Elementen eines über $\mathbb{R}$ vierdimensionalen, nicht kommutativen Körpers, auf besonders schöne Weise darstellen (siehe dazu [Sei90]). Insbesondere lassen sich in dieser Darstellung Rotationen um beliebige Achsen einfacher handhaben.

3.3.3 Transformation von Normalen

Wird z.B. eine affine Ebene E im $\mathbb{R}^4$ durch die Gleichung $x \cdot n^t + d = 0$ mit Normalenvektor $n = (n_x, n_y, n_z, n_w)$, $x = (x_x, x_y, x_z, x_w)$ und $d \in \mathbb{R}$ beschrieben, so ist zu beachten, daß sich Normalenvektoren nicht mit derselben Matrix A wie die Ortsvektoren transformieren. Bezeichnen wir mit I die Einheitsmatrix, so gilt:

$$\begin{aligned} x \cdot n^t &= x \cdot I \cdot n^t && (3.39) \\ &= x \cdot (A \cdot A^{-1}) \cdot n^t && (3.40) \\ &= \underbrace{(x \cdot A)}_{x'} \cdot \underbrace{A^{-1} n^t}_{\left(n \cdot (A^{-1})^t\right)^t}. && (3.41) \end{aligned}$$

Daher müssen Normalen mit der Transponierten der Inversen der Transformationsmatrix A transformiert werden.

3.3.4 Wechsel des Koordinatensystems

In der Praxis ist es oft wesentlich einfacher, mit mehreren Koordinatensystemen zu arbeiten, die den jeweiligen Problemstellungen angepaßt sind.
Sind nun zwei Koordinatensysteme Ψ und Ψ' mit den Basisvektoren e_1, e_2, e_3, e_4 und e'_1, e'_2, e'_3, e'_4 gegeben und beschreibt die Matrix A die Abbildung, welche die Basisvektoren e_i auf die Basisvektoren e'_i, $i = 1, \cdots, 4$ abbildet, so berechnen sich die Koordinaten $[x'_1, x'_2, x'_3, x'_4]$ eines Punktes im Koordinatensystem Ψ' aus den Koordinaten $[x_1, x_2, x_3, x_4]$ desselben Punktes im Koordinatensystem Ψ durch

$$[x'_1, x'_2, x'_3, x'_4] = [x_1, x_2, x_3, x_4] \cdot A^{-1}. \tag{3.42}$$

Als Beispiel dazu betrachten wir eine Drehung in der Ebene um 90°, die wir durch die Matrix $A = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$ beschreiben (vgl. Bild 3.13).

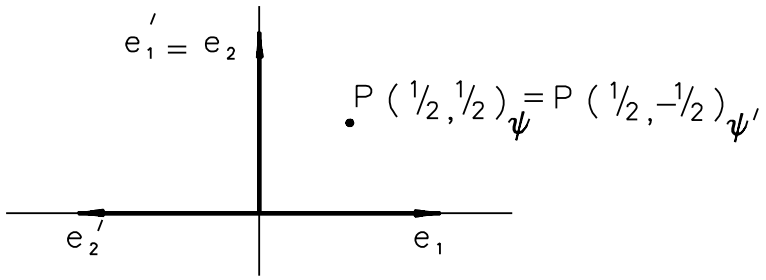


Abbildung 3.13: Die Koordinaten $(x'_1, x'_2)_{\Psi'}$ des Punktes p im Koordinatensystem Ψ' berechnen sich gemäß Gleichung (3.42) aus den Koordinaten $(x_1, x_2)_{\Psi}$ im Koordinatensystem Ψ.

Die Koordinaten der beiden Basisvektoren e'_1 und e'_2 im Koordinatensystem Ψ sind durch

$$(1,0) \cdot \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} = (0,1) \qquad \text{und} \qquad (0,1) \cdot \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} = (-1,0) \tag{3.43}$$

gegeben. Die Koordinaten z.B. des Punktes p mit den Koordinaten $(1,1)$ bezüglich des Koordinatensystems Ψ sind im Koordinatensystem Ψ' durch

$$(1,1) \cdot \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} = (1,-1) \tag{3.44}$$

gegeben.

Abschließend sei darauf hingewiesen, daß sich bei einem Basiswechsel im allgemeinen Längen und Winkel ändern, obwohl sich die Lage von Punkten im Raum nicht verändert hat. Das ist nicht der Fall bei Drehungen und Translationen, wohl aber bei Scherungen, Skalierungen und perspektivischen Abbildungen.

3.3.5 Ebene geometrische Projektionen

Obwohl auch andere Projektionen von Interesse sind (gekrümmte Projektionsstrahlen z.B. in der Relativitätstheorie, Projektionen auf nicht notwendig ebene

Flächen), beschränken wir uns auf ebene geometrische Projektionen. Die ebenen geometrischen Projektionen sind dadurch charakterisiert, daß mit Projektionsstrahlen konstanter Richtung, d.h. entlang von Geraden, auf Ebenen projiziert wird (vgl. Bild 3.14).

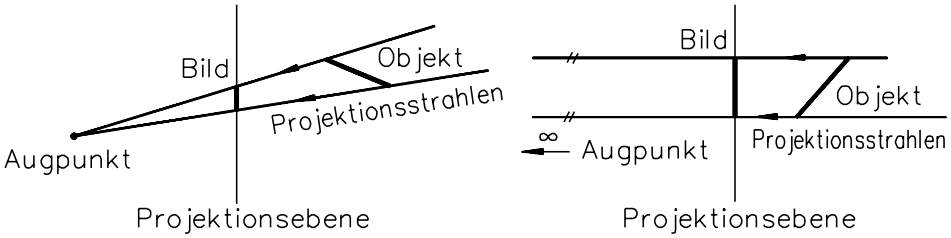


Abbildung 3.14: Perspektivische und parallele Projektionen
Bei Parallelprojektionen sind die Projektionsstrahlen parallel.

Eine sehr schöne Darstellung der ebenen geometrischen Projektionen, die auch die geschichtliche Entwicklung des Verständnisses der Perspektive aufzeigt, ist in [CP78] zu finden. Von dort stammt auch die in Bild 3.15 gezeigte Zusammenstellung ebener Projektionen nebst Beispielen.

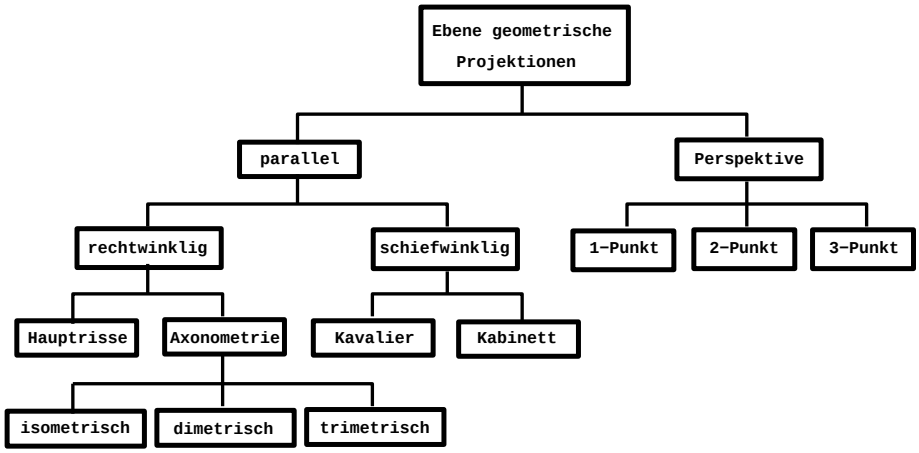


Abbildung 3.15: Einteilung ebener geometrischer Projektionen [Car78]

Die wichtigsten dieser Projektionen werden wir im folgenden kurz darstellen. Wir gehen davon aus, daß die drei Hauptrichtungen des darzustellenden Objekts mit den Richtungen der Koordinatenachsen des zugrundeliegenden Koordinatensystems übereinstimmen (vgl. Bild 3.16).

Im folgenden werden alle Projektionen durch eine invertierbare Transformation und anschließende Parallelprojektion entlang einer Koordinatenachse dargestellt. Das hat folgende Gründe: Erstens bleibt eine Tiefeninformation bis zur Parallelprojektion entlang einer Koordinatenachse erhalten, was für viele Algorithmen wichtig ist, zweitens wird das Auffinden des zu einem zweidimensionalen Bildpunkt gehörenden Urbildpunktes in einer dreidimensionalen Szene dadurch wesentlich erleichtert.

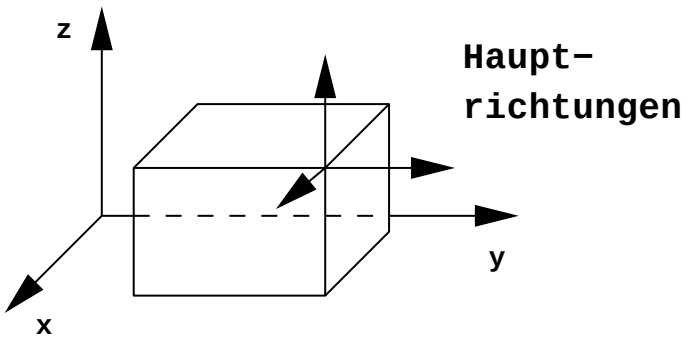


Abbildung 3.16: Die Hauptrichtungen des darzustellenden Objektes stimmen mit den Koordinatenachsen überein

3.3.5.1 Parallelprojektionen

Bei Parallelprojektionen unterscheidet man zwischen rechtwinkligen und schiefwinkligen Projektionen. Bei rechtwinkligen Projektionen steht die Projektionsrichtung senkrecht auf der Projektionsebene, d.h. sie ist parallel zur Normalen der Projektionsebene. Dementsprechend steht bei schiefwinkligen Projektionen die Projektionsrichtung nicht senkrecht auf der Projektionsebene und bildet mit der Normalen der Projektionsebene einen Winkel. Bei Parallelprojektionen bleiben Längen auf Geraden parallel zur Projektionsebene erhalten.

3.3.5.2 Rechtwinklige Projektionen

Ist die Projektionsrichtung durch $p = (p_x, p_y, p_z)$ gegeben (vgl. Bild 3.17), so sind dadurch die Verkürzungsverhältnisse $v_x : v_y : v_z$ auf den Hauptachsen durch die Projektion bestimmt.

Sie entsprechen den Längenverhältnissen der projizierten Einheitsvektoren e'_x, e'_y, e'_z:

$$v_x : v_y : v_z = ||e'_x|| : ||e'_y|| : ||e'_z||.$$

Stimmt die Projektionsrichtung mit einer der Koordinaten-Richtungen überein, so erhalten wir je nach Wahl der Ebene und des Vorzeichens der Projektionsrichtung einen der sechs Hauptrisse eines Objekts (vgl. Bild 3.18).

In Matrizenschreibweise beispielsweise ist die Projektion auf die Ebene $z = z_0$ gegeben durch

$$[x', y', z', w'] = [x, y, z, w] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & z_0 & 1 \end{bmatrix}. \tag{3.45}$$

In den meisten Fällen wird man für x_0, y_0 oder z_0 den Wert Null wählen.

Die Lage des Bildschirmkoordinatensystems wird meist so gewählt, daß sein Ursprung mit dem Ursprung des 3D-Koordinatensystems zusammenfällt und die z-

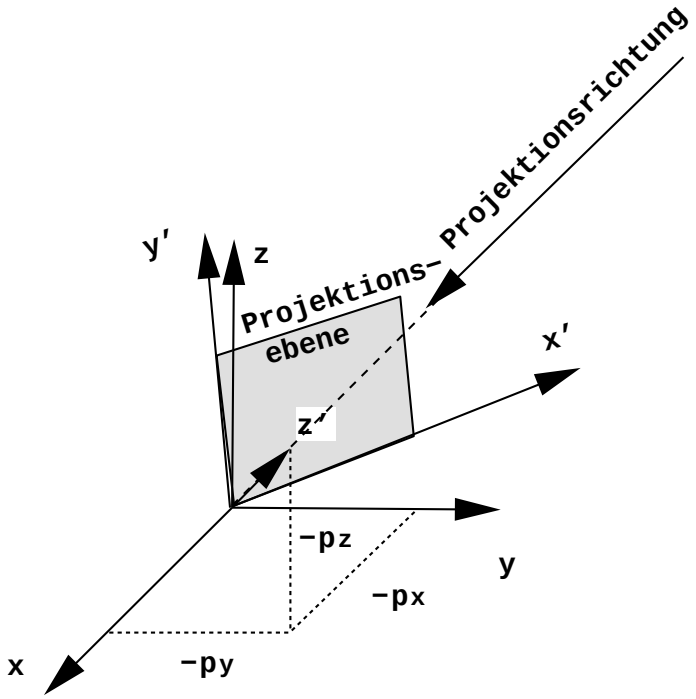


Abbildung 3.17: Rechtwinklige Projektion
Die Oben-Richtung im Bild stimmt mit der $z-$Richtung überein. Die Projektionsrichtung bestimmt die Verkürzungsverhältnisse entlang der Achsen.

Richtung die Oben-Richtung (ViewUp) definiert. Die Projektion der Oben–Richtung auf die Projektionsebene definiert y'.
Seien $p = (p_x, p_y, p_z)$ und $o = (o_x, o_y, o_z)$ die normierte Projektions- bzw. Oben-richtung. Dann läßt sich das rechtshändige Bildschirmkoordinatensystem x', y', z' wie folgt berechnen:

1) $e_{z'} = -p$

2) Da $e_{x'}$ zu der von p und o aufgespannten Ebene senkrecht ist, gilt

$$e_{x'} = \frac{o \times e'_z}{||o \times e'_z||}$$

3) $e_{y'} = e_{z'} \times e_{x'}$.

Dann wird die zu projizierende Szene in diesem Koordinatensystem dargestellt (vgl. Abschnitt 3.3.4 'Wechsel des Koordinatensystems'). Anschließend wird die Parallelprojektion entlang der z'–Achse durchgeführt.

Bildet die Normale der Projektionsebene mit allen drei Koordinaten-Achsen denselben Winkel, so erhält man eine *isometrische* Projektion (vgl. nachfolgendes Beispiel), sind zwei dieser Winkel identisch, so heißt die Projektion *dimetrisch*, sind schließlich alle drei Winkel verschieden, so erhält man die *trimetrische* Projektion (vgl. Bilder 3.19-3.21). Dadurch ergeben sich für die Strecken parallel zu

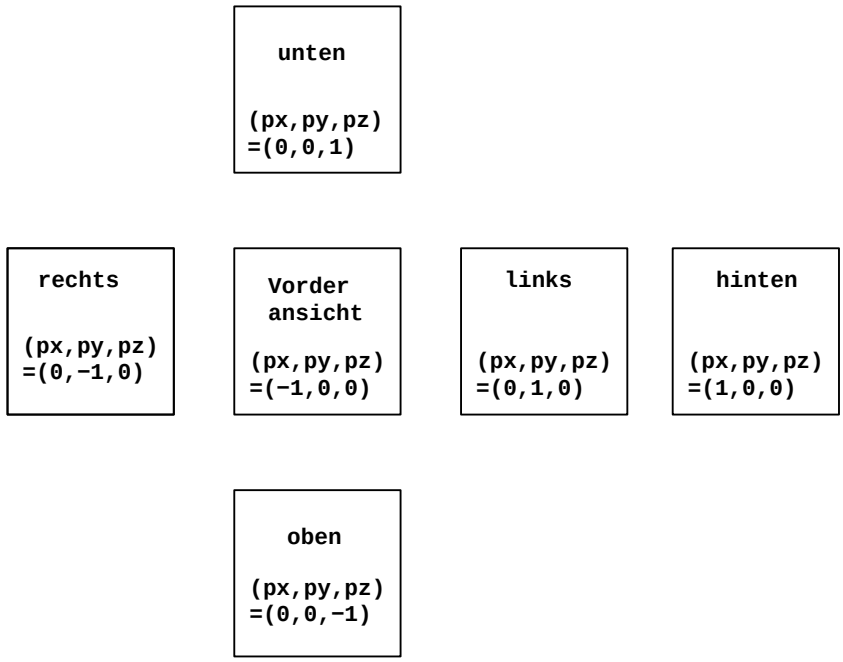


Abbildung 3.18: Die Hauptrisse eines Quaders. Enthält die Bildschirmebene den Ursprung, so ist $p = (p_x, p_y, p_z)$ wie im Bild angegeben zu wählen, um die entsprechenden Ansichten zu erhalten.

den Koordinatenachsen die in den Abbildungen eingezeichneten Längenverhältnisse.

Beispiel 3.2:

Als Beispiel betrachten wir den Fall $p = \frac{1}{\sqrt{3}}(-1,-1,-1),\ o = (0,0,1)$.
Zunächst wird das rechtshändige Bildschirmkoordinatensystem x',y',z' berechnet:

$$\begin{aligned} e_{z'} &= -p = \frac{1}{\sqrt{3}}(1,1,1) \\ e_{x'} &= \frac{o \times e_{z'}}{||o \times e_{z'}||} = \frac{(0,0,1)\times(1,1,1)}{||(0,0,1)\times(1,1,1)||} = \frac{1}{\sqrt{2}}(-1,1,0) \\ e_{y'} &= e_{z'} \times e_{x'} = \frac{1}{\sqrt{6}}(1,1,1)\times(-1,1,0) = \frac{1}{\sqrt{6}}(-1,-1,2) \end{aligned}$$

Tragen wir diese Einheitsvektoren in die Zeilen einer 3×3-Matrix ein, so erhalten wir die Matrix

$$\begin{bmatrix} -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ -\frac{1}{\sqrt{6}} & -\frac{1}{\sqrt{6}} & \frac{2}{\sqrt{6}} \\ \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} \end{bmatrix},$$

die die Einheitsvektoren (1,0,0), (0,1,0) und (0,0,1) des x,y,z-Koordinatensystems auf die Vektoren

$$e_{x'} = \frac{1}{\sqrt{2}}(-1,1,0), e_{y'} = \frac{1}{\sqrt{6}}(-1,-1,2) \text{ und } e_{z'} = \frac{1}{\sqrt{3}}(1,1,1)$$

des Bildschirmkoordinatenystems x',y',z' abbildet.

Zum Wechsel in das Bildschirmkoordinatensystem wird die Inverse dieser Matrix benötigt. Da die Vektoren $e_{x'}, e_{y'}$ und $e_{z'}$ orthonormal sind, stimmt die Inverse dieser Matrix mit ihrer Transponierten überein. Gemäß Formel 3.23 wird die transponierte 3×3-Matrix in eine 4×4-Matrix eingebettet.

Eine anschließende Parallelprojektion entlang der z'-Achse auf die x',y'-Ebene (Projektionsebene) liefert die Gesamttransformation:

$$P = R\cdot P_z = \begin{bmatrix} -\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{6}} & \frac{1}{\sqrt{3}} & 0 \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{6}} & \frac{1}{\sqrt{3}} & 0 \\ 0 & \frac{2}{\sqrt{6}} & \frac{1}{\sqrt{3}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} -\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{6}} & 0 & 0 \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{6}} & 0 & 0 \\ 0 & \frac{2}{\sqrt{6}} & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Diese Projektion bildet die 3D-Einheitsvektoren e_x, e_y, e_z, welche die Hauptrichtungen definieren, auf die 2D-Vektoren

$$(-\frac{1}{\sqrt{2}},-\frac{1}{\sqrt{6}}),\ (\frac{1}{\sqrt{2}},-\frac{1}{\sqrt{6}}) \quad \text{und} \quad (0,\frac{2}{\sqrt{(6)}})$$

ab. In der Projektion sind die Winkel zwischen den Hauptrichtungen daher 120° (vgl. Bild 3.19a). Da die Längen der Bild-Vektoren gleich sind, stimmen in der Projektion die Verkürzungsverhältnisse entlang der Hauptrichtungen überein. Man erhält in diesem Fall eine Isometrie.

Mittels dieser Vektoren können auch die Winkel zwischen den Hauptrichtungen und der x'-Achse des Bildschirmkoordinatensystems (Horizontale) berechnet werden: Zwischen der projizierten x-Hauptrichtung und der x'-Achse des Bildschirmkoordinatensystems ergibt sich ein Winkel von

$$\alpha = \arccos\left(\frac{(-\frac{1}{\sqrt{2}},\frac{1}{\sqrt{6}})\cdot(1,0)}{||(-\frac{1}{\sqrt{2}},\frac{1}{\sqrt{6}})||}\right) = 150°.$$

Auf die gleiche Art ergibt sich ein Winkel zwischen der projizierten y-Hauptrichtung und der x'-Achse von $\beta = 30°$. In der Praxis werden jedoch meist die Winkel α' und β' eingezeichnet (vgl. Bild 3.19b).

Sind Verkürzungsverhältnisse $v_x : v_y : v_z$ vorgegeben, so kann man daraus umgekehrt eine normierte Projektionsrichtung $p = (p_x, p_y, p_z)$ bestimmen, die jedoch nicht eindeutig festgelegt ist:

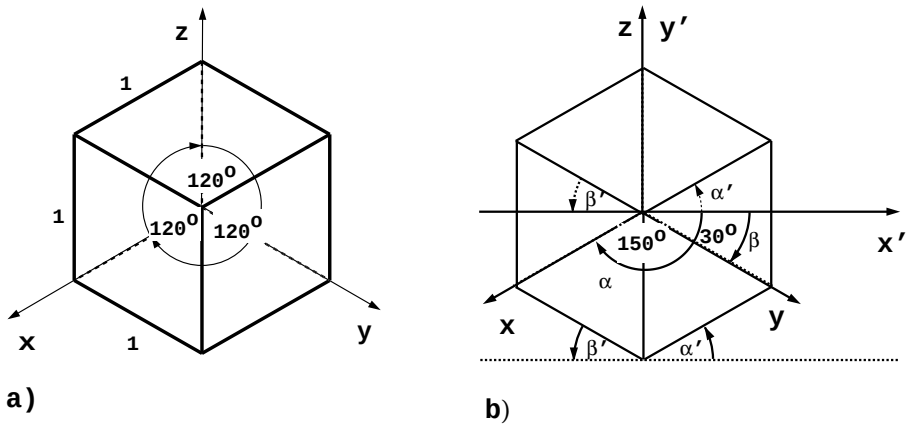


Abbildung 3.19: Isometrie
Die Normale der Projektionsebene bildet mit allen Koordinatenachsen denselben Winkel, woraus gleiche Verkürzungsverhältnisse auf allen drei Achsen folgen.

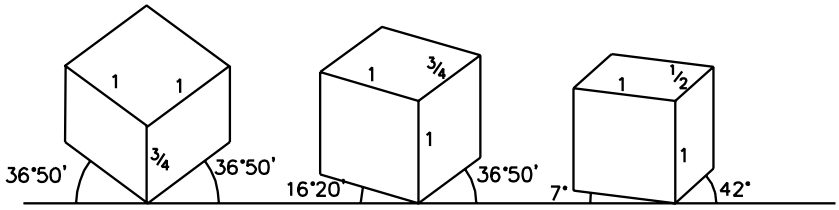


Abbildung 3.20: Dimetrie
Die Normale der Projektionsebene bildet mit zwei Koordinatenachsen denselben Winkel, woraus gleiche Verkürzungsverhältnisse auf zwei Achsen folgen.

Sei z.B. $p = \frac{1}{\sqrt{3}}((\pm)1, (\pm)1, (\pm)1)$. Jede der acht Projektionsrichtungen liefert eine Isometrie.

Wenn Sie die PDF-Version des Kurses bearbeiten, können Sie jetzt das Interaktionselement 'orthograhicprojections' anwählen.

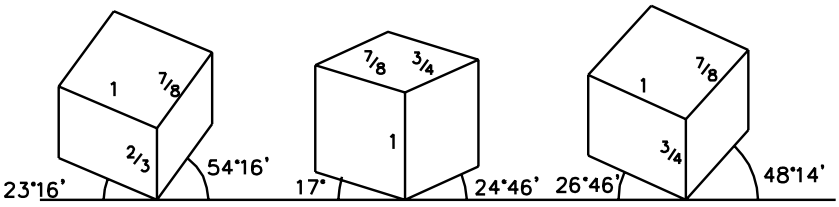


Abbildung 3.21: Trimetrie
Die Normale der Projektionsebene bildet verschiedene Winkel mit den Koordinatenachsen, wodurch sich beliebige Verkürzungsverhältnisse entlang der Achsen realisieren lassen.

3.3.5.3 Schiefwinklige Projektionen

Bei den schiefwinkligen Projektionen ist die Projektionsebene meistens parallel zu zwei Koordinatenachsen, die Projektionsrichtung jedoch nicht senkrecht zu der Projektionsebene (vgl. Bild 3.22a).

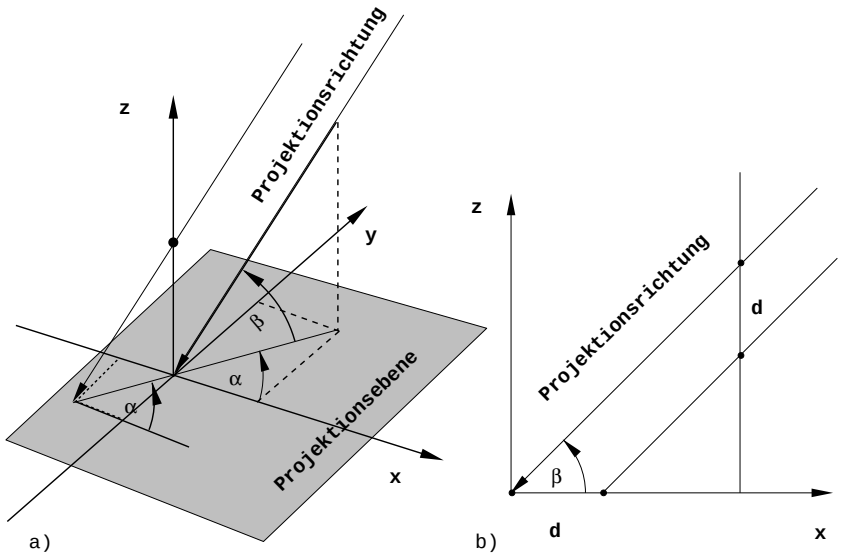


Abbildung 3.22: a): Bei schiefwinkligen Projektionen stimmt die Projektionsrichtung nicht mit der Normalen der Projektionsebene überein. Die Angabe der Projektionsrichtung erfolgt meist mittels der beiden Winkel α und β.
b) Bei der Kavalierprojektion ist $\beta = 45°$. Dadurch bleiben Längen auf Geraden senkrecht zur Projektionsebene unverändert. Zur Vereinfachung wurde im Bild $\alpha = 0°$ gewählt.

Bei der *Kavalierprojektion* bildet die Projektionsrichtung mit der Normalen der Projektionsebene einen Winkel von 45° (vgl. Bild 3.22 b).
Daher bleiben die Längen auf Geraden senkrecht zur Projektionsebene bei dieser Projektion unverändert (vgl. Bild 3.23).

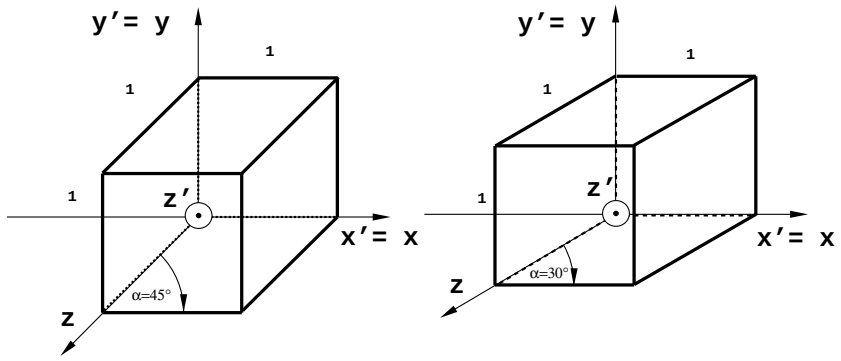


Abbildung 3.23: Kavalierprojektion
Die Längenverhältnisse auf den Achsen bleiben erhalten.

Bei der *Kabinettprojektion* (vgl. Bild 3.24) werden Längen auf Geraden senkrecht zur Projektionsebene um den Faktor $\frac{1}{2}$ verkürzt. Der Winkel β zwischen Projektionsebene und Projektionsrichtung ist daher $\arctan(2) = 63°$.

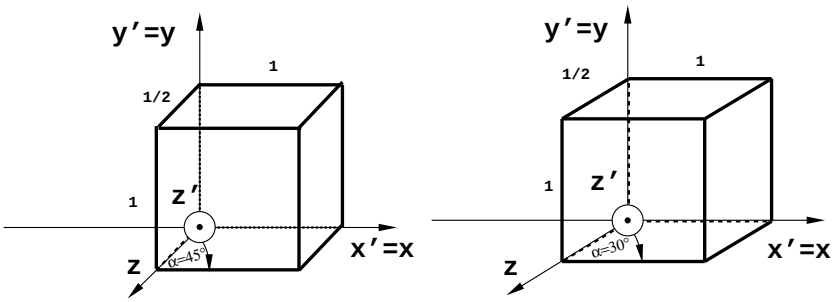


Abbildung 3.24: Kabinettprojektion
Längen auf Geraden senkrecht zur Projektionsebene werden um den Faktor $\frac{1}{2}$ verkürzt.

Abhängig von α und β ergibt sich die normierte Projektionsrichtung

$$p = (-\cos\alpha \cdot \cos\beta, -\sin\alpha \cdot \cos\beta, -\sin\beta).$$

Stimmt die Projektionsebene mit der x,y-Ebene überein (vgl. Bild 3.22), so lassen sich die Matrizendarstellungen der Kavalier- und Kabinettprojektion wie folgt bestimmen:
Wie bei den rechtwinkligen Projektionen wird nach einer Transformation eine Parallelprojektion entlang der z'-Achse durchgeführt. Bei der Transformation werden die Einheitsvektoren e_x und e_y auf sich selbst (die Punkte in der Projektionsebene bleiben fest) und die normierte Projektionsrichtung p auf die z'-Achse abgebildet, d.h. auf ein Vielfaches λ von $e_{z'}$. Damit die Tiefeninformation bei dieser Transformation erhalten bleibt, muß $\lambda > 0$ sein, kann sonst aber beliebig gewählt werden. Wählt man $\lambda = 1$ und trägt die drei Bildvektoren wieder in die Zeilen einer 3×3-Matrix ein, so erhält man die Matrix

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -\cos\alpha\cos\beta & -\sin\alpha\cos\beta & -\sin\beta \end{bmatrix}. \tag{3.46}$$

Zum Wechsel in das Bildschirmkoordinatensystem wird die Inverse dieser Matrix benötigt. Gemäß Formel 3.23 wird die inverse 3×3-Matrix in eine 4×4-Matrix eingebettet, und es ergibt sich folgende Transformations-Matrix:

$$[x',y',z',w'] = [x,y,z,w] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -\frac{\cos\alpha}{\tan\beta} & \frac{-\sin\alpha}{\tan\beta} & -\frac{1}{\sin\beta} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{3.47}$$

Die Komposition mit der anschließenden Parallelprojektion entlang der z'-Achse liefert die Matrix

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -\frac{\cos\alpha}{\tan\beta} & \frac{-\sin\alpha}{\tan\beta} & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{3.48}$$

Dieses Ergebnis erhält man auch ohne unser zweistufiges Berechnungsschema direkt aus Bild 3.22:
Offenbar ist

$$\begin{aligned} (x',y',z') &= (x,y,z)+\lambda(p_x,p_y,p_z) \\ &= (x,y,z)+\lambda(-\cos\alpha\cdot\cos\beta,-\sin\alpha\cdot\cos\beta,-\sin\beta). \end{aligned}$$

Mit $z'=0$ wird $\lambda=-\frac{z}{p_z}=-\frac{z}{-\sin\beta}$ und

$$\begin{aligned} (x',y',z') &= (x,y,z)-\frac{z}{-\sin\beta}(-\cos\alpha\cdot\cos\beta,-\sin\alpha\cdot\cos\beta,-\sin\beta) \\ &= \left(x-z\cdot\frac{\cos\alpha}{\tan\beta},y-z\cdot\frac{\sin\alpha}{\tan\beta},0\right). \end{aligned}$$

3.3.5.4 Perspektivische Projektionen

Perspektivische Projektionen sind keine affinen Abbildungen, da sie z.B. Längenverhältnisse nicht invariant lassen: Vom Blickpunkt weit entfernte Objekte werden kleiner dargestellt als Objekte mit kleinem Abstand zum Blickpunkt (Augpunkt, Beobachtungspunkt). Diese Abbildungen sind projektive Abbildungen und lassen sich daher nur im projektiven Raum als Matrizen beschreiben.
Zum besseren Verständnis von perspektivischen Abbildungen beschränken wir uns zunächst auf den zweidimensionalen Fall. Bild 3.25 zeigt eine perspektivische Projektion eines affinen Punktes $P=(x,y)$ auf einen Bildpunkt auf der affinen Bildgeraden $x=0$.

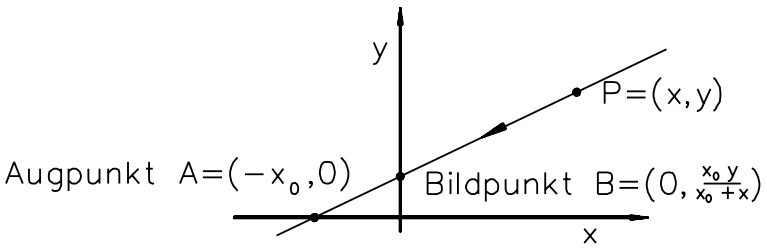


Abbildung 3.25: Perspektivische Projektion eines Punktes $P=(x,y)$ mit Augpunkt $A=(-x_0,0)$ und Projektionsgerade $x=0$

Diese Projektion läßt sich folgendermaßen berechnen. Ist der zu projizierende Punkt $P=(x,y)$ und der Augpunkt $A=(-x_0,0)$ gegeben, so gilt nach dem Strahlensatz für die Koordinaten des Punktes $B=(0,y_0)$

$$\frac{y_0}{y}=\frac{x_0}{x+x_0}.$$

Allgemein kann daher die Abbildung angegeben werden:

$$(x,y) \mapsto (0, \frac{x_0 y}{x_0 + x}) \tag{3.49}$$

und in homogenen Koordinaten

$$[x,y,1] \mapsto [0, \frac{x_0 y}{x_0 + x}, 1] = [0, x_0 y, x_0 + x]. \tag{3.50}$$

Daraus ergibt sich die 3×3-Matrix zur Beschreibung dieser Abbildung.

$$[x',y',1] = [x,y,1] \begin{bmatrix} 0 & 0 & 1 \\ 0 & x_0 & 0 \\ 0 & 0 & x_0 \end{bmatrix} = [x,y,1] \begin{bmatrix} 0 & 0 & \frac{1}{x_0} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{3.51}$$

Wie schon bei den Parallelprojektionen zerlegen wir die Projektion in zwei Abbildungen

$$\begin{bmatrix} 0 & 0 & \frac{1}{x_0} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & \frac{1}{x_0} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{bzw.} \tag{3.52}$$

$$P = T_p \cdot P_p. \tag{3.53}$$

Die Matrix P_p beschreibt die Projektion auf die Gerade $x = 0$. Die Matrix T_p beschreibt die *perspektivische* Transformation, welche im folgenden ausführlicher diskutiert wird.

3.3.5.5 Perspektivische Transformation

Die durch T_p festgelegte perspektivische Transformation

$$[x,y,w] \begin{bmatrix} 1 & 0 & \frac{1}{x_0} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = [x, y, \frac{x}{x_0} + w] \tag{3.54}$$

hat folgende Eigenschaften (vgl. Bild 3.26):

1. Für $w = 1$ und $x = -x_0$ sind die Bildpunkte dieser Abbildung alle von der Form $[-x_0, y, 0]$, d.h. alle Punkte auf der affinen Geraden $x = -x_0$ werden auf unendlich ferne Punkte abgebildet. Um Unstetigkeiten in der perspektivischen Transformation zu vermeiden, bildet man nur Punkte einer der beiden durch diese Gerade bestimmten Halbebenen ab. Man nennt dann diese Halbebene *Sichtfeld* und die Gerade $x = -x_0$ *Sichtgerade.*

2. Die Punkte der projektiven Geraden $x = 0$, das ist die y-Achse vereinigt mit dem unendlich fernen Punkt $[0,1,0]$, sind Fixpunkte dieser Abbildung, d.h. sie bleiben unverändert. Die y-Achse bleibt daher bei dieser Abbildung fest. (Sie bildet die *Projektionsgerade*.) Da auch der unendlich ferne Punkt $[0,1,0]$ ein Fixpunkt ist, bleiben zur y-Achse parallele Geraden parallel zur y-Achse.

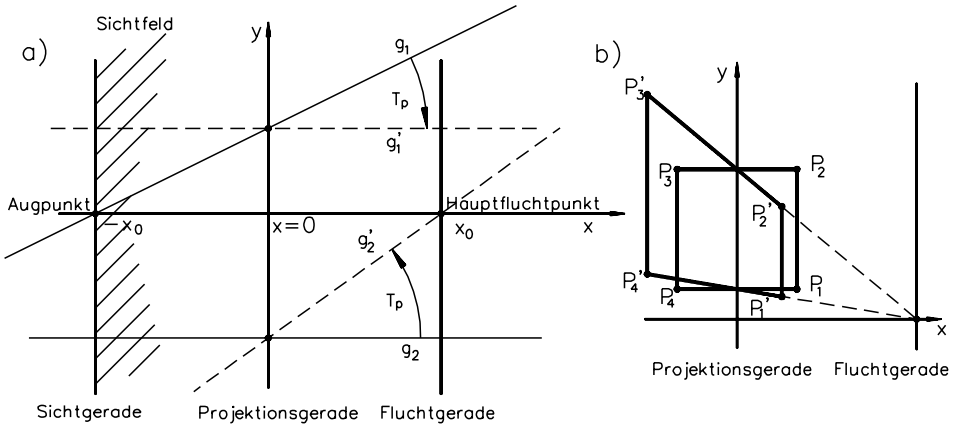


Abbildung 3.26: a) zeigt eine Einpunktperspektive mit zugehörigem Augpunkt, der Fluchtgeraden und dem endlichen Hauptfluchtpunkt.
b) Perspektivische Transformation eines Quadrats
Die zu der x-Achse parallelen Kanten des Quadrats liegen nach der Transformation auf Geraden durch den Fluchtpunkt.

3. Aus 1. folgt zunächst, daß der affine Punkt $[-x_0,0,1]$ auf den unendlich fernen Punkt $[-x_0,0,0]=[1,0,0]$, d.h. auf die Richtung der x-Achse, abgebildet wird. Nach 2. bleiben die Punkte der affinen y-Achse bei der Transformation fest. Auch im projektiven Raum sind Geraden durch zwei Punkte festgelegt. Da bei projektiven Abbildungen Geraden auf Geraden abgebildet werden, folgt daraus, daß eine affine Gerade durch den Augpunkt $[-x_0,0,1]$, welche die affine y-Achse im Punkt $[0,y_0,1]$ schneidet, auf eine affine Gerade parallel zur x-Achse durch den Punkt $[0,y_0,1]$ abgebildet wird.

4. Der Punkt $[x,y,0]$ wird auf den Punkt $[x,y,\frac{x}{x_0}]=[x_0,x_0\frac{y}{x},1]$ abgebildet. Daher enthalten die Bildgeraden von Parallelen zur affinen Geraden mit der Richtung $[x,y,0]$ alle den Punkt $[x_0,x_0\frac{y}{x},1]$, d.h. sie schneiden sich in diesem Punkt. Variiert man x und y, so sieht man, daß die Gesamtheit der Bildpunkte aller Richtungen $[x,y,0]$ auf der Geraden $x=x_0$ liegt. Diese Gerade heißt *Fluchtgerade.*

5. Die Schnittpunkte der Bildgeraden von Parallelen zu den Koordinatenachsen werden *Hauptfluchtpunkte* oder einfach *Fluchtpunkte* genannt. Nach 4. schneiden sich die Bildgeraden von Parallelen zur x-Achse alle in dem affinen Punkt $[x_0,0,1]$. Nach 2. bleiben Parallelen zur y-Achse parallel, d.h. sie schneiden sich nur im unendlich fernen Punkt $[0,1,0]$ ihrer Richtung. Daher besitzt die perspektivische Abbildung, die durch T_p dargestellt wird, nur einen Hauptfluchtpunkt im Affinen. Man nennt diese spezielle Perspektive daher *Einpunktperspektive.*

Durch

$$[x,y,w]\begin{bmatrix}1&0&\frac{1}{x_0}\\0&1&\frac{1}{y_0}\\0&0&1\end{bmatrix}=[x,y,\frac{x}{x_0}+\frac{y}{y_0}+w] \tag{3.55}$$

wird eine *Zweipunktperspektive* beschrieben. Sie besitzt folgende Eigenschaften (vgl. dazu Bild 3.27):

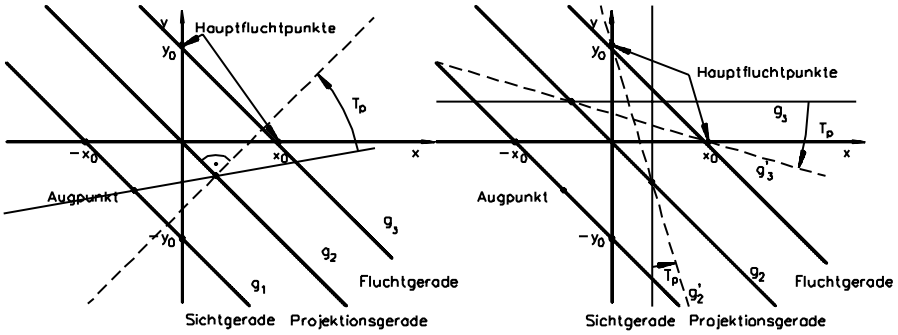


Abbildung 3.27: Zweipunktperspektive mit zugehörigem Augpunkt, der Fluchtgeraden und den Hauptfluchtpunkten
Parallelen zur x- oder y-Achse werden auf Geraden durch die jeweiligen Fluchtpunkte abgebildet.

1. Die Punkte der affinen Geraden $g_1 : \frac{1}{x_0}x + \frac{1}{y_0}y = -1$ werden auf unendlich ferne Punkte abgebildet, d.h. g_1 bildet die Sichtgerade.

2. Die Punkte der affinen Geraden $g_2 : \frac{1}{x_0}x + \frac{1}{y_0}y = 0$ bleiben fest, d.h. g_2 bildet die Projektionsgerade.

3. Der Augpunkt liegt zum einen auf der Sichtgeraden g_1, zum anderen werden affine Geraden durch den Augpunkt auf affine Geraden senkrecht zur Projektionsgeraden abgebildet. Aus diesen beiden Bedingungen folgt für den Augpunkt
$$A = [\frac{-x_0y_0^2}{x_0^2+y_0^2}, \frac{-x_0^2y_0}{x_0^2+y_0^2}, 1].$$

4. Punkte mit den Koordinaten $[x,y,0]$ werden auf Punkte mit Koordinaten
$$[x,y,\frac{1}{x_0}x+\frac{1}{y_0}y] = [\frac{x_0y_0x}{x_0y+y_0x}, \frac{x_0y_0y}{x_0y+y_0x}, 1]$$
abgebildet.
Diese liegen alle auf der affinen Geraden g_3 mit der Gleichung
$$\frac{1}{x_0}x + \frac{1}{y_0}y = 1.$$
Das ist die Fluchtgerade der Perspektive.

5. Die Richtungen von Parallelen zu den Koordinatenachsen, d.h. die Punkte $[1,0,0]$ bzw. $[0,1,0]$, werden auf die beiden Fluchtpunkte $[x_0,0,1]$ bzw. $[0,y_0,1]$ abgebildet.

Die Einpunktperspektive ist ein Spezialfall der Zweipunktperspektive. Liegt einer der beiden Fluchtpunkte $[x_0,0,1] = [1,0,\frac{1}{x_0}]$ oder $[0,y_0,1] = [0,1,\frac{1}{y_0}]$ im Unendlichen, so folgt z.B. für den zweiten Fall

$$\lim_{y_0\to\infty} \begin{bmatrix} 1 & 0 & \frac{1}{x_0} \\ 0 & 1 & \frac{1}{y_0} \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & \frac{1}{x_0} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{3.56}$$

Diese Matrix beschreibt die Einpunktperspektive nach Gleichung (3.54). Die Fluchtgerade, die Projektionsgerade und den Augpunkt für diesen Fall erhält man aus den obigen Formeln ebenfalls durch den Grenzübergang $y_0 \to \infty$. Die Ein- und Zweipunktperspektive unterscheiden sich also dadurch, daß bei der Zweipunktperspektive beide Fluchtpunkte im Endlichen liegen, wogegen bei der Einpunktperspektive einer der beiden Fluchtpunkte im Unendlichen liegt. Umgekehrt gilt die Beziehung

$$\begin{bmatrix} 1 & 0 & \frac{1}{x_0} \\ 0 & 1 & \frac{1}{y_0} \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos\mu & \sin\mu & 0 \\ -\sin\mu & \cos\mu & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & \frac{1}{d} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos\mu & -\sin\mu & 0 \\ \sin\mu & \cos\mu & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.57}$$

mit

$$\mu = \arctan\frac{x_0}{y_0}$$

und

$$d = \frac{x_0 y_0}{\sqrt{x_0^2 + y_0^2}},$$

d.h. man kann die Zweipunktperspektive durch zwei Drehungen aus einer geeigneten Einpunktperspektive gewinnen.

3.3.5.6 Perspektivische Transformation im dreidimensionalen Raum

Eine allgemeine perspektivische Transformation ist durch

$$[x,y,z,w]\begin{bmatrix} 1 & 0 & 0 & \frac{1}{x_0} \\ 0 & 1 & 0 & \frac{1}{y_0} \\ 0 & 0 & 1 & \frac{1}{z_0} \\ 0 & 0 & 0 & 1 \end{bmatrix} = [x,y,z,\frac{x}{x_0}+\frac{y}{y_0}+\frac{z}{z_0}+w] \tag{3.58}$$

gegeben.
Sie besitzt, analog zum zweidimensionalen Fall, folgende Eigenschaften (vgl. dazu Bild 3.28):

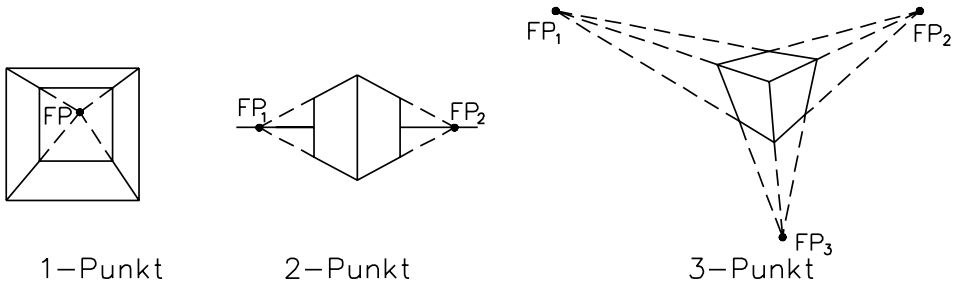


Abbildung 3.28: Beispiele für Ein,- Zwei- und Drei-Punktperspektive

1. Die Punkte der affinen Ebene $E_1 : \frac{1}{x_0}x + \frac{1}{y_0}y + \frac{1}{z_0}z = -1$ werden auf unendlich ferne Punkte abgebildet, d.h. E_1 bildet die *Sichtebene*.

2. Die Punkte der affinen Ebene $E_2 : \frac{1}{x_0}x + \frac{1}{y_0}y + \frac{1}{z_0}z = 0$ bleiben fest, d.h. E_2 bildet die *Projektionsebene*.

3. Der *Augpunkt* liegt zum einen auf der Sichtebene E_1, zum anderen werden affine Geraden durch den Augpunkt auf affine Geraden senkrecht zur Projektionsebene abgebildet. Aus diesen beiden Bedingungen folgt für den Augpunkt

$$A = [\frac{-x_0y_0^2z_0^2}{x_0^2y_0^2 + x_0^2z_0^2 + y_0^2z_0^2}, \frac{-x_0^2y_0z_0^2}{x_0^2y_0^2 + x_0^2z_0^2 + y_0^2z_0^2}, \frac{-x_0^2y_0^2z_0}{x_0^2y_0^2 + x_0^2z_0^2 + y_0^2z_0^2}, 1].$$

4. Punkte mit den Koordinaten $[x,y,z,0]$ werden auf Punkte mit Koordinaten

$$\left[x,y,z,\frac{x}{x_0} + \frac{y}{y_0} + \frac{z}{z_0}\right]$$

   abgebildet.
   Diese liegen auf der *Fluchtebene* mit der Gleichung

$$\frac{1}{x_0}x + \frac{1}{y_0}y + \frac{1}{z_0}z = 1.$$

5. Die Richtungen von Parallelen zu den Koordinatenachsen, d.h. die Punkte $[1,0,0,0]$, $[0,1,0,0]$ bzw. $[0,0,1,0]$ werden auf die drei *Fluchtpunkte* $[x_0,0,0,1]$, $[0,y_0,0,1]$ bzw. $[0,0,z_0,1]$ abgebildet.

Die Spezialfälle der Ein- und Zweipunktperspektive erhält man, wie im zweidimensionalen Fall, auch wieder durch die entsprechenden Grenzübergänge.

Wenn Sie die PDF-Version des Kurses bearbeiten, können Sie jetzt das Interaktionselement 'vanpoints' anwählen.

3.3.5.7 Definition einer allgemeinen perspektivischen Transformation

Bei der praktischen Anwendung von Projektionen liegt der Standort des Betrachters beliebig im Objektraum. Eine allgemeine perspektivische Transformation wird durch die Festlegung eines Augpunktes (*Eye*), eines Blickbezugspunktes (*VRef*) und der Angabe einer Oben-Richtung (*ViewUp*) festgelegt (Bild 3.29).

Die Berechnung der Transformation wird in vier Schritten durchgeführt:

1. Berechnung des Bildschirmkoordinatensystems mit Ursprung *VRef* und den Basisvektoren v'_x, v'_y und v'_z.

   v'_z wird durch *VRef* und *Eye* definiert.
   v'_x steht senkrecht auf den Vektoren *ViewUp* und v'_z.

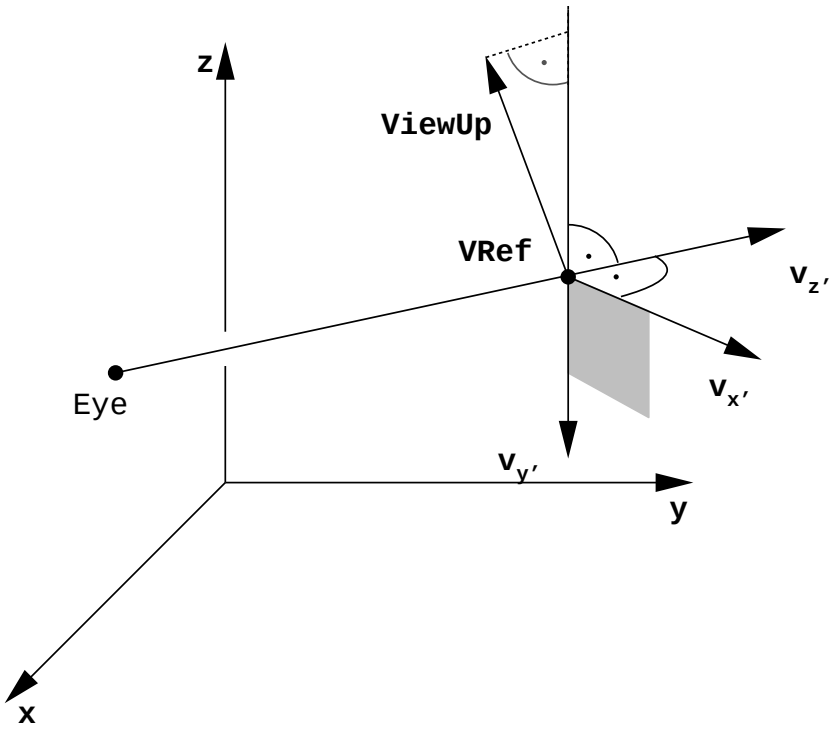


Abbildung 3.29: Festlegung eines Blickkoordinatensystems

Da die $y-$Achse des Bildschirmkoordinatensystems in die entgegengesetzte Richtung des *ViewUp*-Vektors zeigt, bilden die drei Vektoren v'_x, $ViewUp$ und v'_z ein linkshändiges, noch nicht notwendig orthogonales, Koordinatensystem.

Der Vektor *ViewUp* wird daher durch den Vektor v'_y senkrecht zu v'_z und v'_x ersetzt, so daß v'_x, v'_y und v'_z ein rechtshändiges orthogonales Koordinatensystem bilden.

Die Vektoren v'_x, v'_y und v'_z müssen normiert werden!

2. Translation des Bildschirmkoordinatensystems in den Ursprung.

3. Rotation des Bildschirmkoordinatensystems auf das Welt- bzw. Referenzkoordinatensystem:

$$v'_x \to x; \quad v'_y \to y; \quad v'_z \to z$$

4. Perspektivische Transformation mit Abstand $|VRef - Eye|$, wobei der Augpunkt (*Eye*) auf der negativen z-Achse liegt.

Beispiel 3.3:

Gesucht ist eine Transformation T, die das Geradenstück von $P_1(0,0)$ nach $P_2(1,0)$ in das Geradenstück $P'_1P'_2$ transformiert (Bild 3.30).

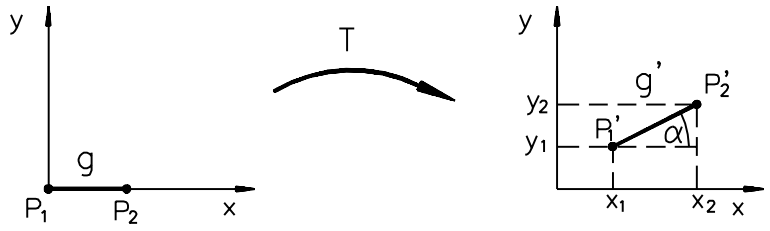


Abbildung 3.30: Beispiel einer affinen Transformation

Die Aufgabe wird so gelöst, daß zuerst die Rotation, dann die Skalierung und zuletzt die Translation ausgeführt wird. Mit

$$\cos\alpha = \frac{x_2 - x_1}{l}$$

und

$$\sin\alpha = \frac{y_2 - y_1}{l}$$

wird

$$T = \begin{bmatrix} \frac{x_2-x_1}{l} & \frac{y_2-y_1}{l} & 0 \\ -\frac{y_2-y_1}{l} & \frac{x_2-x_1}{l} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} l & 0 & 0 \\ 0 & l & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x_1 & y_1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} x_2 - x_1 & y_2 - y_1 & 0 \\ -(y_2 - y_1) & x_2 - x_1 & 0 \\ x_1 & y_1 & 1 \end{bmatrix}.$$

Nicht in allen Fällen läßt sich die gesuchte Transformation so offensichtlich wie im ersten Beispiel aus Grundtransformationen zusammensetzen. In diesen Fällen müssen die gegebenen Koordinaten in die Gleichung eingesetzt und Koeffizientenvergleiche durchgeführt werden.

Wenn Sie die PDF-Version des Kurses bearbeiten, können Sie jetzt das Interaktionselement 'transformation' anwählen.

Beispiel 3.4:

Gesucht ist die Transformationsmatrix T, die eine perspektivische Darstellung entsprechend Bild 3.31 liefert. Die Lage der Punkte kann aus denen der Parallelprojektion in die $x-z-$ bzw. $y-z-$Ebene abgelesen werden.

Da wir eine perspektivische Transformation suchen, müssen wir im projektiven Raum $P(\mathbb{R}^4)$ mit homogenen Koordinaten rechnen. Wie im Kapitel über affine und projektive Räume gezeigt wird, sind die homogenen Koordinaten im $P(\mathbb{R}^4)$ erst durch fünf projektiv unabhängige Punkte festgelegt. Dasselbe gilt für bijektive projektive Abbildungen im $P(\mathbb{R}^4)$. Sie sind durch fünf projektiv unabhängige Bild- und Urbildpunkte festgelegt.

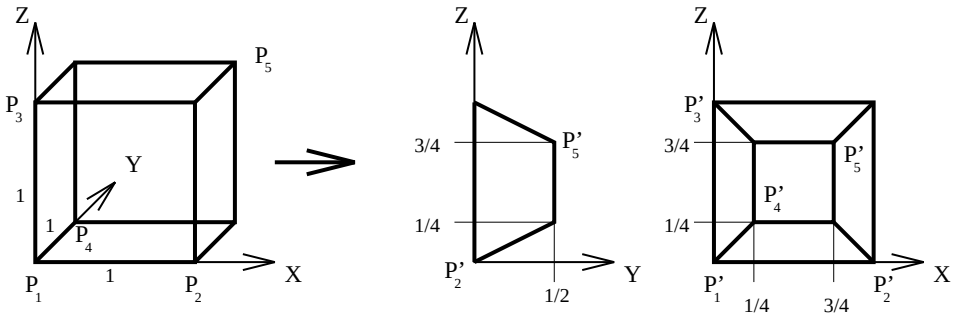


Abbildung 3.31: Beispiel einer perspektivischen Transformation

Um die homogene Matrix T zu finden, durch welche die gesuchte Abbildung beschrieben wird, gehen wir wie folgt vor:
Zunächst bestimmen wir homogene Matrizen A_1 und A_2, welche die Punkte $p_1 = [1,0,0,0],\ p_2 = [0,1,0,0],\ p_3 = [0,0,1,0],\ p_4 = [0,0,0,1],\ p_5 = [1,1,1,1]$ der projektiven Basis des $P(\mathbb{R}^4)$ auf die Punkte $P_1,\cdots,P_5$ bzw. $P_1'\cdots,P_5'$ abbilden. Hat man diese Matrizen bestimmt, so gilt für die gesuchte homogene Matrix T

$$A_1 \cdot T = A_2 \Longleftrightarrow T = A_1^{-1} \cdot A_2.$$

Im zweiten Schritt wird die Matrix A_1 invertiert und mit der Matrix A_2 multipliziert.

Schritt 1: Aus Bild 3.31 entnehmen wir zunächst die homogenen Koordinaten der Urbild- und Bildpunkte

$P_1 = [0,0,0,1],\ P_2 = [1,0,0,1],\ P_3 = [0,0,1,1],\ P_4 = [0,1,0,1],$
$P_5 = [1,1,1,1]$

sowie

$P_1' = [0,0,0,1],\ P_2' = [1,0,0,1],\ P_3' = [0,0,1,1],\ P_4' = [\frac{1}{4},\frac{1}{2},\frac{1}{4},1],$
$P_5' = [\frac{3}{4},\frac{1}{2},\frac{3}{4},1].$

Die Bild- und Urbildpunkte sind projektiv unabhängig, also existieren homogene Matrizen A_1 und A_2 mit der oben beschriebenen Eigenschaft. Schreibt man skalare Vielfache der Koordinaten der jeweils ersten vier Punkte zeilenweise in die Matrizen A_1 und A_2, so ist

$$A_1 = \begin{bmatrix} 0 & 0 & 0 & \lambda_1 \\ \lambda_2 & 0 & 0 & \lambda_2 \\ 0 & 0 & \lambda_3 & \lambda_3 \\ 0 & \lambda_4 & 0 & \lambda_4 \end{bmatrix} = \begin{bmatrix} \lambda_1 & 0 & 0 & 0 \\ 0 & \lambda_2 & 0 & 0 \\ 0 & 0 & \lambda_3 & 0 \\ 0 & 0 & 0 & \lambda_4 \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix}$$

und

$$A_2 = \begin{bmatrix} 0 & 0 & 0 & \lambda_1' \\ \lambda_2' & 0 & 0 & \lambda_2' \\ 0 & 0 & \lambda_3' & \lambda_3' \\ \frac{\lambda_4'}{4} & \frac{\lambda_4'}{2} & \frac{\lambda_4'}{4} & \lambda_4' \end{bmatrix} = \begin{bmatrix} \lambda_1' & 0 & 0 & 0 \\ 0 & \lambda_2' & 0 & 0 \\ 0 & 0 & \lambda_3' & 0 \\ 0 & 0 & 0 & \lambda_4' \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & 1 \end{bmatrix}.$$

Die Matrix A_1 bildet die homogenen Koordinaten der Punkte $p_1, \cdots, p_4$ auf die homogenen Koordinaten der Punkte $P_1, \cdots, P_4$ ab. Entsprechendes gilt für die Matrix A_2. Setzt man die homogenen Koordinaten der Punkte p_5, P_5 und P_5' ein, so erhält man

$$[1,1,1,1] \cdot \begin{bmatrix} \lambda_1 & 0 & 0 & 0 \\ 0 & \lambda_2 & 0 & 0 \\ 0 & 0 & \lambda_3 & 0 \\ 0 & 0 & 0 & \lambda_4 \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix} = \lambda_5 \cdot [1,1,1,1]$$

$$[1,1,1,1] \cdot \begin{bmatrix} \lambda_1' & 0 & 0 & 0 \\ 0 & \lambda_2' & 0 & 0 \\ 0 & 0 & \lambda_3' & 0 \\ 0 & 0 & 0 & \lambda_4' \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & 1 \end{bmatrix} = \lambda_5' \cdot \left[\frac{3}{4}, \frac{1}{2}, \frac{3}{4}, 1\right]$$

und daraus

$$[\lambda_1, \lambda_2, \lambda_3, \lambda_4] = [-2, 1, 1, 1]$$

sowie

$$[\lambda_1', \lambda_2', \lambda_3', \lambda_4'] = [-1, \frac{1}{2}, \frac{1}{2}, 1].$$

Die Koeffizienten λ_i bzw. λ_i', $i = 1, \cdots, 4$, sind dabei bis auf die gemeinsamen Faktoren λ_5 bzw. λ_5' festgelegt, da A_1 und A_2 homogene Matrizen sind.

Schritt 2: Für die gesuchte homogene Matrix T gilt die Beziehung

$$T = A_1^{-1} \cdot A_2,$$

und daraus folgt

$$T = \begin{bmatrix} 0 & 0 & 0 & -2 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix}^{-1} \cdot \begin{bmatrix} 0 & 0 & 0 & -1 \\ \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & \frac{1}{2} \\ 0 & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & \frac{1}{2} \end{bmatrix}.$$

Da die Matrizen A_1 und A_2 bis auf die Faktoren λ_5 und λ_5' festgelegt sind, ist auch T nur bis auf einen Faktor λ_T festgelegt und daher eine homogene Matrix. Weitere Literatur zur Theorie der projektiven und perspektivischen Abbildungen in der Computergraphik findet man z.B. in [Par90], [Kor90], [Her89] und [PP86].

3.4 Übungsaufgaben

Aufgabe 1:

Gegeben seien die Punkte $P_1 = (3,1); P_2 = (1,2); P_1' = (4,2)$ und $P_2' = (2,3)$ in der Ebene (vgl. Bild 3.32).

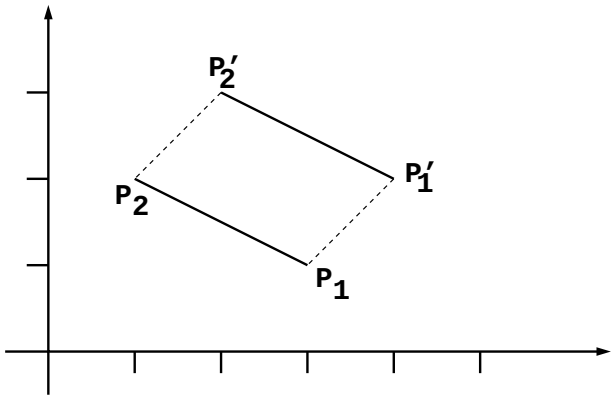


Abbildung 3.32: Punkte P_1, P_2, P_1' und P_2'

a) Bestimmen Sie die Matrix der linearen Transformation T, welche P_1 auf P_1' und P_2 auf P_2' abbildet.

b) Schreiben Sie T als Hintereinanderausführung einer Skalierung S und einer Scherung SH. Geben Sie die zugehörigen Matrizen an.

Aufgabe 2:

Gegeben sei ein Rechteck (Bild 3.33a), das in die Darstellung in Bild 3.33b gebracht werden soll.

a) Bestimmen Sie die Transformationsmatrix, indem Sie vier projektiv unabhängige Punkte vor und nach der Transformation betrachten.

b) Stellen Sie die in a) bestimmte Matrix als Produkt elementarer affiner Transformationen dar (vgl. Bild 3.36).

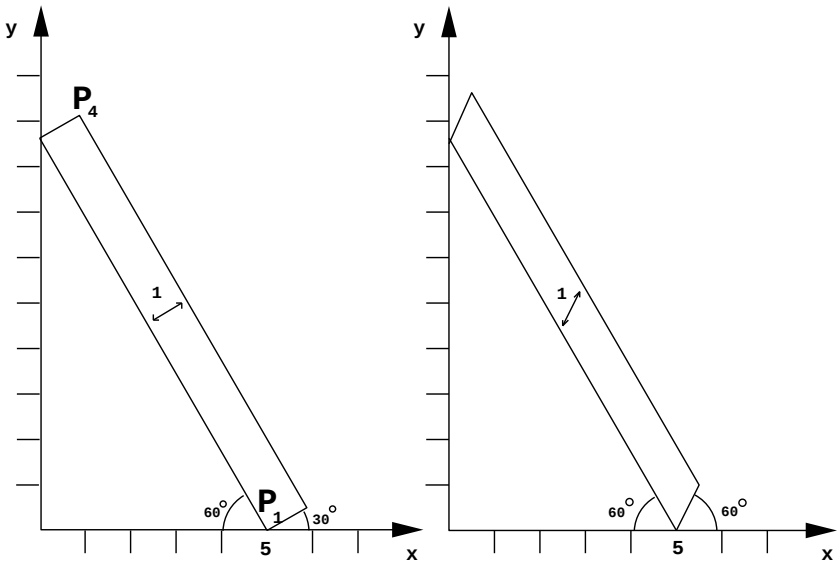


Abbildung 3.33: a) Ausgangslage b) Gesuchte Lage

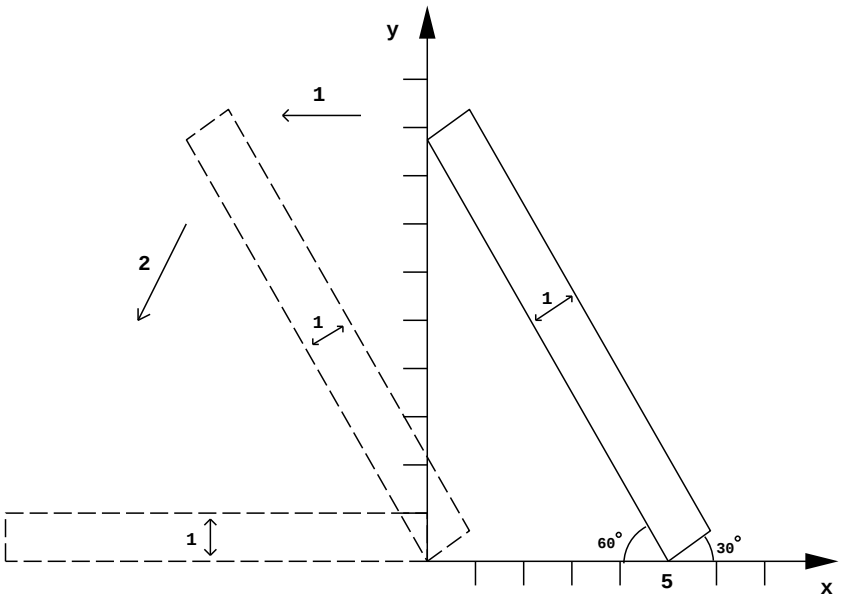


Abbildung 3.34: Folge von Transformationen

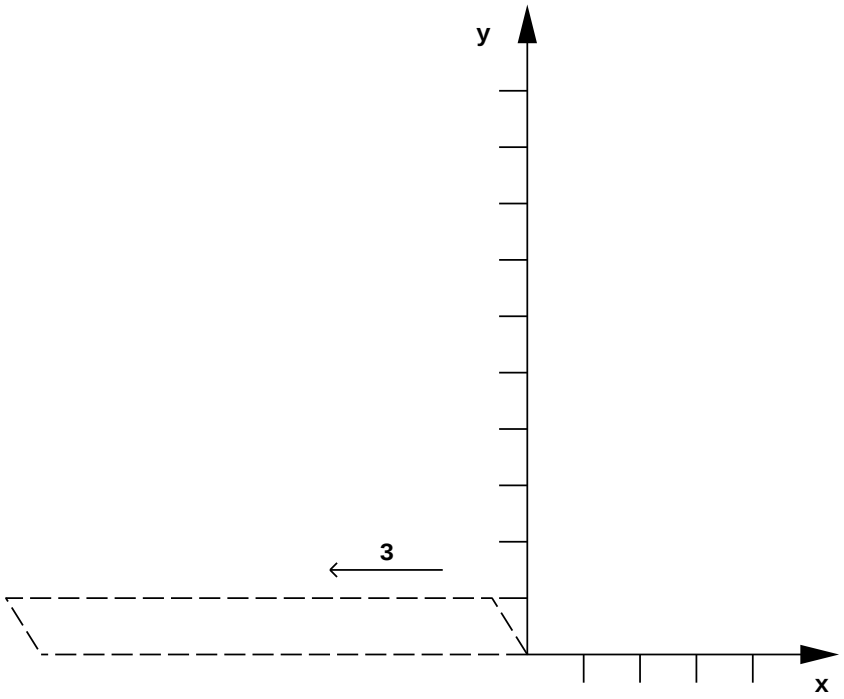


Abbildung 3.35: Folge von Transformationen

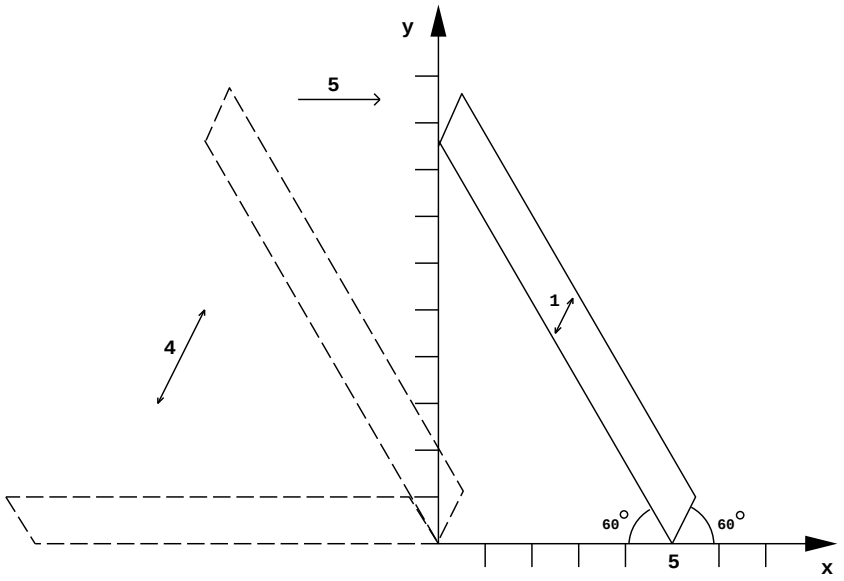


Abbildung 3.36: Folge von Transformationen

Aufgabe 3:

Gegeben sei ein Einheitsprojektionsvektor $p = (p_x, p_y, p_z)$. Bestimmen Sie die Transformationsmatrix M einer rechtwinkligen Parallelprojektion auf eine Bildschirmebene im Ursprung des Koordinatensystems entlang p. Die Oben-Richtung entspreche dabei der positiven z–Achse.

Aufgabe 4:

In der Ebene sei eine perspektivische Transformation durch Augpunkt $A = (-x_0, 0)$, Projektionsgerade $y = 0$ und Fluchtpunkt $F = (x_0, 0)$ sowie ein Quadrat mit Eckpunkten P_1, P_2, P_3, P_4 gegeben (vgl. Bild 3.37).

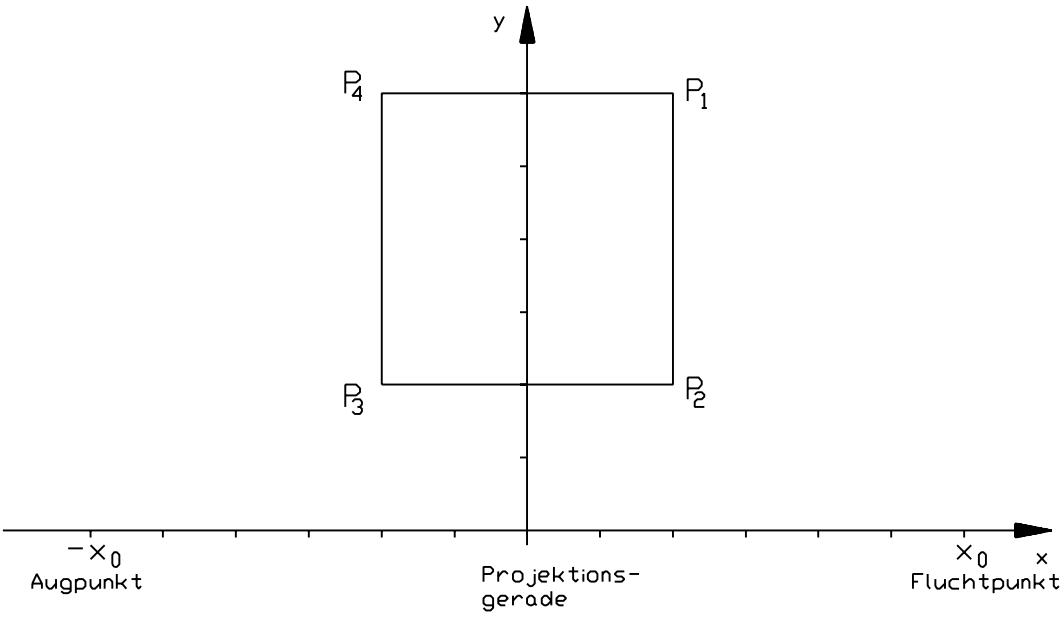


Abbildung 3.37: Quadrat mit Eckpunkten P_1, P_2, P_3 und P_4

a) Konstruieren Sie zeichnerisch die transformierten Punkte P_1', P_2', P_3', P_4'. Überprüfen Sie das Ergebnis durch Rechnung.

b) Eine homogene Geradengleichung ist gegeben durch

$$\vec{x} \cdot \vec{n} = [x, y, w] \cdot \begin{bmatrix} n_x \\ n_y \\ n_w \end{bmatrix} = x n_x + y n_y + w n_w = 0.$$

Sind zwei Punkte $p = (p_x, p_y, p_w)$ und $q = (q_x, q_y, q_w)$ gegeben, so ist die Gleichung der Geraden durch p und q durch

$$\vec{x} \cdot \vec{n} = [x, y, w] \cdot \left[\begin{pmatrix} p_x \\ p_y \\ p_w \end{pmatrix} \times \begin{pmatrix} q_x \\ q_y \\ q_w \end{pmatrix} \right] = 0 \qquad \text{gegeben.}$$

Ermitteln Sie die Gleichungen des Quadrats und transformieren Sie dieselben mit der gegebenen perspektivischen Transformation. Überprüfen Sie das Ergebnis anhand der Konstruktion.

3.5 Lösungen zu den Übungsaufgaben

Lösung 1:

a) Wir bestimmen Matrizen A_1 und A_2, welche die Basisvektoren (1,0) und (0,1) auf die Vektoren (3,1) und (1,2) bzw (4,2) und (2,3) abbilden.

$$A_1 = \begin{pmatrix} 3 & 1 \\ 1 & 2 \end{pmatrix} \qquad A_2 = \begin{pmatrix} 4 & 2 \\ 2 & 3 \end{pmatrix}$$

Die Matrix T der gesuchten Transformation ergibt sich dann zu $M = A_1^{-1} \cdot A_2$

$$\frac{1}{5}\begin{pmatrix} 2 & -1 \\ -1 & 3 \end{pmatrix}\begin{pmatrix} 4 & 2 \\ 2 & 3 \end{pmatrix} = \frac{1}{5}\begin{pmatrix} 6 & 1 \\ 2 & 7 \end{pmatrix}$$

b)

$$T = \begin{pmatrix} a & b \\ c & d \end{pmatrix} = \underbrace{\begin{pmatrix} S_1 & 0 \\ 0 & S_2 \end{pmatrix}}_{\textit{Skalierung}} \underbrace{\begin{pmatrix} 1 & Sh_1 \\ Sh_2 & 1 \end{pmatrix}}_{\textit{Scherung}} = \begin{pmatrix} S_1 & S_1 Sh_1 \\ S_2 Sh_2 & S_2 \end{pmatrix}.$$

Für $a \neq 0$, $d \neq 0$ ergibt sich daraus

$$S_1 = a, \quad S_2 = d, \quad Sh_1 = b/a, \quad Sh_2 = c/d.$$

In unserem Fall ergibt sich

$$T = \frac{1}{5}\begin{pmatrix} 6 & 1 \\ 2 & 7 \end{pmatrix} = \begin{pmatrix} 6/5 & 0 \\ 0 & 7/5 \end{pmatrix}\begin{pmatrix} 1 & 1/6 \\ 2/7 & 1 \end{pmatrix}.$$

Lösung 2:

a) Als vier projektiv unabhängige Punkte wählen wir

$$\begin{array}{lllllll} P_1 & = & [5,0,1] & & P_3 & = & \left[\sqrt{3},1,0\right] \\ P_2 & = & \left[-1,\sqrt{3},0\right] & & P_4 & = & \left[\frac{1}{2}\sqrt{3},\frac{1}{2}+5\sqrt{3},1\right]. \end{array}$$

Diese Punkte werden durch die gesuchte Transformationsmatrix auf die Punkte

$$\begin{array}{lllllll} P_1' & = & [5,0,1], & & P_3' & = & \left[1,\sqrt{3},0\right] \\ P_2' & = & \left[-1,\sqrt{3},0\right] & & P_4' & = & \left[\frac{1}{3}\sqrt{3},1+5\sqrt{3},1\right] \end{array}$$

abgebildet. Nun bestimmen wir zwei homogene Matrizen A_1 und A_2 ,welche die Punkte $p_1=[1,0,0], p_2=[0,1,0], p_3=[0,0,1]$ und $p_4=[1,1,1]$ der projektiven Basis des $\mathbb{R}^3$ auf die Punkte $P_1,...P_4$ bzw. $P_1',...P_4'$ abbilden. Dazu schreiben wir die skalare Vielfachheit der Koordinaten der jeweils 3 ersten Punkte zeilenweise in die Matrizen A_1 und A_2.

$$A_1=\begin{pmatrix} 5\lambda_1 & 0 & \lambda_1 \\ -\lambda_2 & \sqrt{3}\lambda_2 & 0 \\ \sqrt{3}\lambda_3 & \lambda_3 & 0 \end{pmatrix} = \begin{pmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{pmatrix} \cdot \begin{pmatrix} 5 & 0 & 1 \\ -1 & \sqrt{3} & 0 \\ \sqrt{3} & 1 & 0 \end{pmatrix}$$

$$A_2=\begin{pmatrix} 5\lambda_1' & 0 & \lambda_1' \\ -\lambda_2' & \sqrt{3}\lambda_2' & 0 \\ \lambda_3' & \sqrt{3}\lambda_3' & 0 \end{pmatrix} = \begin{pmatrix} \lambda_1' & 0 & 0 \\ 0 & \lambda_2' & 0 \\ 0 & 0 & \lambda_3' \end{pmatrix} \cdot \begin{pmatrix} 5 & 0 & 1 \\ -1 & \sqrt{3} & 0 \\ 1 & \sqrt{3} & 0 \end{pmatrix}$$

Aus den Bedingungen für den vierten Punkt

$$(1,1,1)\cdot\begin{pmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{pmatrix}\cdot\begin{pmatrix} 5 & 0 & 1 \\ -1 & \sqrt{3} & 0 \\ \sqrt{3} & 1 & 0 \end{pmatrix} = \left(\frac{1}{2}\sqrt{3},\frac{1}{2}+5\sqrt{3},1\right)$$

und

$$(1,1,1)\cdot\begin{pmatrix} \lambda_1' & 0 & 0 \\ 0 & \lambda_2' & 0 \\ 0 & 0 & \lambda_3' \end{pmatrix}\cdot\begin{pmatrix} 5 & 0 & 1 \\ -1 & \sqrt{3} & 0 \\ 1 & \sqrt{3} & 0 \end{pmatrix} = \left(\frac{1}{3}\sqrt{3},1+5\sqrt{3},1\right)$$

folgt

$$(\lambda_1,\lambda_2,\lambda_3) = \left(1,5,\frac{1}{2}\right)$$

und

$$\left(\lambda_1',\lambda_2',\lambda_3'\right) = \left(1,5,\frac{1}{3}\sqrt{3}\right).$$

Damit sind die Matrizen A_1 und A_2 vollständig bestimmt.
Die Transformationsmatrix zwischen P_1,P_2,P_3,P_4 und P_1',P_2',P_3',P_4' ergibt

sich nun gemäß

$$\begin{aligned} T &= A_1^{-1} \cdot A_2 \\ &= \begin{pmatrix} 0 & -\frac{1}{20} & \frac{1}{2}\sqrt{3} \\ 0 & \frac{1}{20}\sqrt{3} & \frac{1}{2} \\ 1 & \frac{1}{4} & -\frac{5}{2}\sqrt{3} \end{pmatrix} \begin{pmatrix} 5 & 0 & 1 \\ -5 & 5\sqrt{3} & 0 \\ \frac{1}{3}\sqrt{3} & 1 & 0 \end{pmatrix} \\ &= \begin{pmatrix} \frac{3}{4} & \frac{1}{4}\sqrt{3} & 0 \\ -\frac{1}{12}\sqrt{3} & \frac{5}{4} & 0 \\ \frac{5}{4} & -\frac{5}{4}\sqrt{3} & 1 \end{pmatrix}. \end{aligned}$$

b) Die Transformation wird in fünf Schritten durchgeführt (vgl. Bild 3.34):

1) Translation um den Vektor $(-5,0)$:

$$T_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -5 & 0 & 1 \end{pmatrix}$$

2) Rotation um den Winkel $60°$ um den Punkt $(0,0)$:

$$R_1 = \begin{pmatrix} \frac{1}{2} & \frac{\sqrt{3}}{2} & 0 \\ -\frac{\sqrt{3}}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

3) Scherung mit dem Parameter $s_1 = -\tan 30° = -\frac{1}{\sqrt{3}}$ $\quad (s_2 = 0)$:

$$S_1 = \begin{pmatrix} 1 & 0 & 0 \\ -\frac{1}{\sqrt{3}} & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

4) Rotation um den Winkel $-60°$:

$$R_2 = \begin{pmatrix} \frac{1}{2} & -\frac{\sqrt{3}}{2} & 0 \\ \frac{\sqrt{3}}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

5) Translation um den Vektor $(5,0)$:

$$T_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 5 & 0 & 1 \end{pmatrix}$$

Die gesamte Matrix erhalten wir durch Multiplikation:

$$T_1 R_1 S_1 R_2 T_2 = \begin{pmatrix} \frac{3}{4} & \frac{\sqrt{3}}{4} & 0 \\ -\frac{1}{12}\sqrt{3} & \frac{5}{4} & 0 \\ \frac{5}{4} & -\frac{5}{4}\sqrt{3} & 1 \end{pmatrix}$$

Lösung 3:

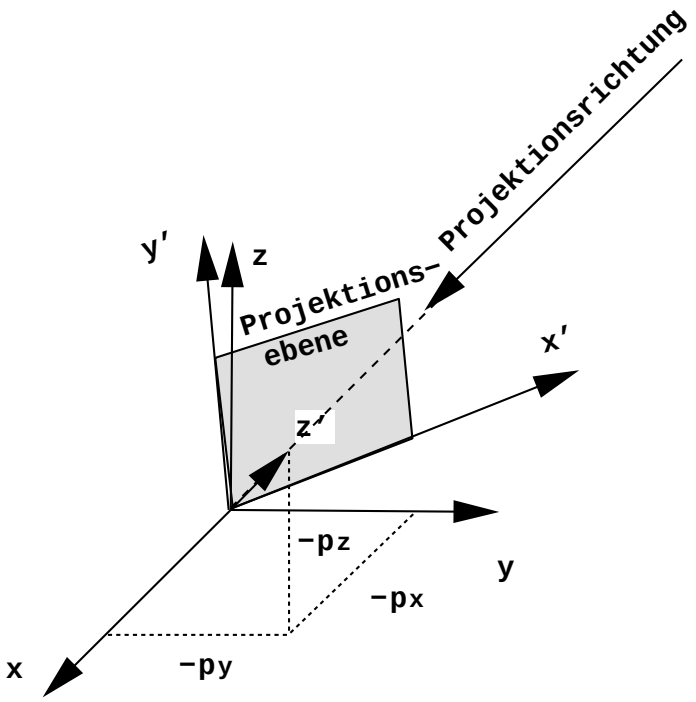


Abbildung 3.38: orthogonales Bildschirmkoordinatensystem

Zunächst bestimmen wir ein orthogonales Bildschirmkoordinatensystem x',y' in der zu $p=(p_x,p_y,p_z)$ senkrechten Ebene durch den Ursprung und setzen

$$z' = -p.$$

Dabei ist darauf zu achten, daß x',y',z' ein rechtshändiges Koordinatensystem definieren.
In einem zweiten Schritt ermitteln wir die Transformationsmatrix, um Punkte im Bildschirmkoordinatensystem darzustellen.
Hierzu nutzen wir die Korrespondenzen

$$\begin{aligned} e_{x'} &\rightarrow e_x = (1,0,0) \\ e_{y'} &\rightarrow e_y = (0,1,0) \\ e_{z'} &\rightarrow e_z = (0,0,1). \end{aligned}$$

Nach dieser Transformation ist eine Parallelprojektion entlang der z'–Achse durchzuführen.

1) Da die Oben-Richtung mit der positiven z-Achse übereinstimmt, ergibt sich:

$$\begin{aligned} e_{z'} &= -p = (-p_x,-p_y,-p_z); \\ e_{x'} &= \frac{e_z \times e_{z'}}{||e_z \times e_{z'}||} = \frac{1}{\sqrt{1-p_z^2}}(p_y,-p_x,0) \\ e_{y'} &= e_{z'} \times e_{x'} = \frac{1}{\sqrt{1-p_z^2}}(-p_x p_z,-p_y p_z,1-p_z^2). \end{aligned}$$

2) Da es sich beim zweiten Schritt um eine lineare Transformation handelt, ist diese durch die folgende Bedingung vollständig bestimmt.

$$\begin{aligned}(-p_x,-p_y,-p_z) &\rightarrow (0,0,1)\\ \tfrac{1}{\sqrt{1-p_z^2}}\left(-p_xp_z,-p_yp_z,1-p_z^2\right) &\rightarrow (0,1,0)\\ \tfrac{1}{\sqrt{1-p_z^2}}\left(p_y,-p_x,0\right) &\rightarrow (1,0,0).\end{aligned}$$

Setzen wir

$$A_1 := \begin{pmatrix} -p_x & -p_y & -p_z \\ -\frac{p_xp_z}{\sqrt{1-p_z^2}} & -\frac{p_yp_z}{\sqrt{1-p_z^2}} & \frac{1-p_z^2}{\sqrt{1-p_z^2}} \\ \frac{p_y}{\sqrt{1-p_z^2}} & -\frac{p_x}{\sqrt{1-p_z^2}} & 0 \end{pmatrix}, \quad A_2 := \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix},$$

so erhalten wir $M_{\vec{p}} = A_1^{-1} \cdot A_2$.
Für $A_1^{-1} = A_1^t$ ergibt sich

$$A_1^{-1} = \begin{pmatrix} -p_x & -\frac{p_xp_z}{\sqrt{1-p_z^2}} & \frac{p_y}{\sqrt{1-p_z^2}} \\ -p_y & -\frac{p_yp_z}{\sqrt{1-p_z^2}} & -\frac{p_x}{\sqrt{1-p_z^2}} \\ -p_z & \frac{1-p_z^2}{\sqrt{1-p_z^2}} & 0 \end{pmatrix}.$$

Daraus folgt

$$M_{\vec{p}} = \begin{pmatrix} \frac{p_y}{\sqrt{1-p_z^2}} & -\frac{p_xp_z}{\sqrt{1-p_z^2}} & -p_x \\ -\frac{p_x}{\sqrt{1-p_z^2}} & -\frac{p_yp_z}{\sqrt{1-p_z^2}} & -p_y \\ 0 & \sqrt{1-p_z^2} & -p_z \end{pmatrix}.$$

3) Führt man anschließend eine Parallelprojektion entlang der z-Achse durch, so erhalten wir $M = M_{\vec{p}} \cdot M_p$,wobei M_p die Matrix der Parallelprojektion entlang der z-Achse ist:

$$\begin{aligned} M &= \begin{pmatrix} \frac{p_y}{\sqrt{1-p_z^2}} & -\frac{p_xp_z}{\sqrt{1-p_z^2}} & -p_x \\ -\frac{p_x}{\sqrt{1-p_z^2}} & -\frac{p_yp_z}{\sqrt{1-p_z^2}} & -p_y \\ 0 & \sqrt{1-p_z^2} & -p_z \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \\ &= \begin{pmatrix} \frac{p_y}{\sqrt{1-p_z^2}} & -\frac{p_xp_z}{\sqrt{1-p_z^2}} & 0 \\ -\frac{p_x}{\sqrt{1-p_z^2}} & -\frac{p_yp_z}{\sqrt{1-p_z^2}} & 0 \\ 0 & \sqrt{1-p_z^2} & 0 \end{pmatrix} \end{aligned}$$

Lösung 4:

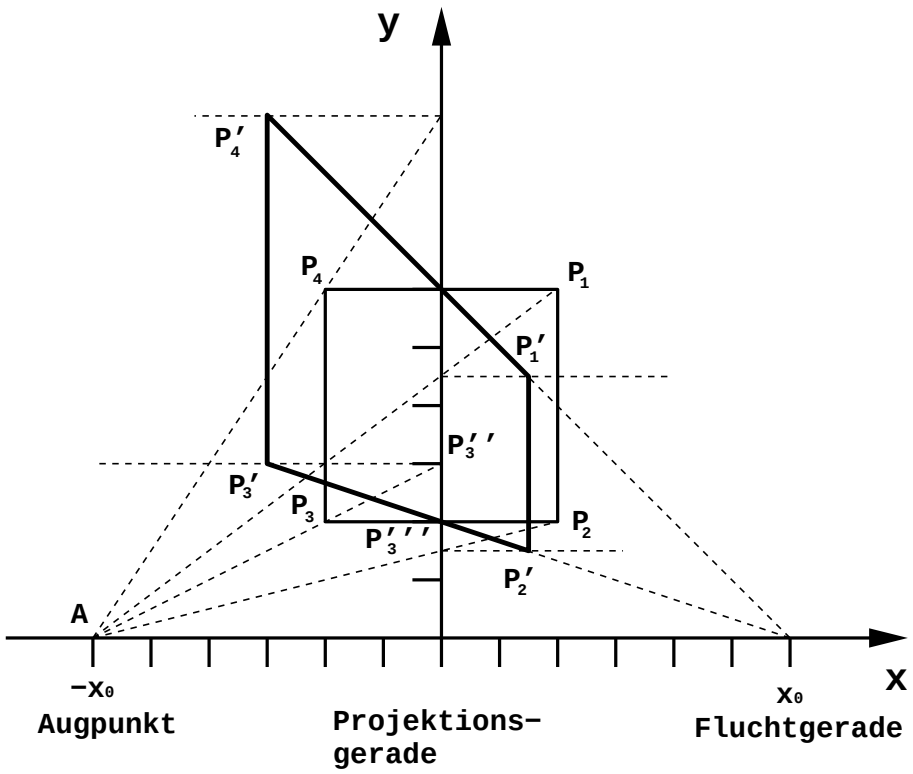


Abbildung 3.39: Konstruktion der transformierten Punkte

a) Alle Geraden durch den Augpunkt A werden auf Parallelen zur x–Achse abgebildet. Dabei bleiben die Punkte auf der y–Achse fest.
Parallelen zur x–Achse werden auf Geraden durch den Fluchtpunkt F abgebildet. Dabei bleiben ebenfalls die Punkte auf der y–Achse fest.
Um z.B. Bedingungen für den Punkt P_3' zu erhalten, betrachten wir die Hilfspunkte P_3'' und P_3''' (vgl. Bild 3.39).
Der Punkt P_3 liegt auf der Geraden $\overline{AP_3''}$. Diese wird bei der perspektivischen Transformation auf die Gerade parallel zur x–Achse durch P_3'' abgebildet. Also liegt P_3' auf dieser Geraden. Der Punkt P_3 liegt auch auf der Geraden $\overline{P_3P_2}$, die durch die perspektivische Transformation auf die Gerade $\overline{P_3'''F}$ abgebildet wird, und daher muß P_3' auch auf dieser Geraden liegen; d.h. aber, P_3' ist der Schnittpunkt der Geraden $\overline{P_3'''F}$ mit der Geraden, die parallel zur x–Achse durch den Punkt P_3'' verläuft.
Die anderen Punkte konstruiert man analog.
Die Matrix der angegebenen perspektivischen Transformation lautet

$$T_p = \begin{pmatrix} 1 & 0 & \frac{1}{6} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Daraus ergibt sich

$$P_1 T_p = [2,6,1] \cdot \begin{pmatrix} 1 & 0 & \frac{1}{6} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = [\tfrac{3}{2}, \tfrac{9}{2}, 1] = P_1'$$

$$P_2 T_p = [2,2,1] \cdot \begin{pmatrix} 1 & 0 & \frac{1}{6} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = [\tfrac{3}{2}, \tfrac{3}{2}, 1] = P_2'$$

$$P_3 T_p = [-2,2,1] \cdot \begin{pmatrix} 1 & 0 & \frac{1}{6} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = [-3,3,1] = P_3'$$

$$P_4 T_p = [-2,6,1] \cdot \begin{pmatrix} 1 & 0 & \frac{1}{6} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = [-3,9,1] = P_4'$$

b) Für die Geradengleichungen ergibt sich

$$g_1 : [x,y,w] \cdot \begin{bmatrix} 1 \\ 0 \\ -2 \end{bmatrix} = 0$$

$$g_2 : [x,y,w] \cdot \begin{bmatrix} 0 \\ -1 \\ 2 \end{bmatrix} = 0$$

$$g_3 : [x,y,w] \cdot \begin{bmatrix} -1 \\ 0 \\ -2 \end{bmatrix} = 0$$

$$g_4 : [x,y,w] \cdot \begin{bmatrix} 0 \\ 1 \\ -6 \end{bmatrix} = 0$$

Die homogenen Koordinaten $\vec{n} = [n_x, n_y, n_w]$ müssen mit der Transponierten der Inversen von T_p transformiert werden:

$$T_{\vec{n}} = \left(T_p^{-1}\right)^T = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -\frac{1}{6} & 0 & 1 \end{pmatrix}$$

Es ergibt sich dann:

$$g_1' : [x,y,w] \cdot \begin{bmatrix} \frac{8}{6} \\ 0 \\ -2 \end{bmatrix} = 0 \Leftrightarrow \tfrac{8}{6}x - 2 = 0 \Leftrightarrow x = \tfrac{3}{2}$$

$$g_2' : [x,y,w] \cdot \begin{bmatrix} -\frac{2}{6} \\ -1 \\ 2 \end{bmatrix} = 0 \Leftrightarrow -\tfrac{2}{6}x - y + 2 = 0 \Leftrightarrow y = 2 - \tfrac{1}{3}x$$

$$g_3' : [x,y,w] \cdot \begin{bmatrix} -\frac{4}{6} \\ 0 \\ -2 \end{bmatrix} = 0 \Leftrightarrow -\tfrac{4}{6}x - 2 = 0 \Leftrightarrow x = -3$$

$$g_4' : [x,y,w] \cdot \begin{bmatrix} 1 \\ 1 \\ -6 \end{bmatrix} = 0 \Leftrightarrow x + y - 6 = 0 \Leftrightarrow y = 6 - x$$

Die Gleichungen der transformierten Geraden können aus der Zeichnung abgelesen werden.

Literaturverzeichnis

[CP78] I. Carlbom, J. Paciorek, *Geometric Projection and Viewing Transformations*, Computing Surveys, Vol. 1, 1978, Nr. 4, 465–502.

[DeR89] T. D. DeRose, *A coordinate-free approach to geometric programming*, Theory and Practice of Geometric Modeling (W. Strasser, H.-P. Seidel, Eds.), Springer Verlag, 1989, 291–305.

[Fis79] G. Fischer, *Analytische Geometrie*, Friedrich Vieweg & Sohn Verlagsgesellschaft mbH, 1979.

[Her89] I. Herman, *Projective geometry and computer graphics*, Advances in Computer Graphics IV (Berlin–Heidelberg–NewYork–Tokyo) (M. Grave, M. Roch, Eds.), Springer Verlag, Berlin–Heidelberg–NewYork–Tokyo, 1989.

[Kor90] K. Kornel, *2d and 3d perspective transforms*, Computer & Graphics, Vol. 14, 1990, Nr. 1, 117–124.

[Par90] B. Pareigis, *Analytische Geometrie und projektive Geometrie für die Computer–Graphik*, Teubner, Stuttgart, 1990.

[PP86] M. Penna, R. Patterson, *Projective Geometry and its Application to Computer Graphics*, Prentice–Hall, 1986.

[Sch76] H. Schaal, *Lineare Algebra und Analytische Geometrie*, Vieweg, Braunschweig, 1976.

[Sei90] H.-P. Seidel, *Quaternionen in Computergraphik und Robotik*, Informationstechnik **it**, Vol. 32, 1990, Nr. 4, 266–275.

Index

Glossar

affiner Raum (3.1.1, S.101)

Eine Menge A^n heißt *n-dimensionaler affiner Raum*, falls ein n-dimensionaler reeller Vektorraum V^n existiert, so daß folgende drei Bedingungen erfüllt sind:

(i) Zu jedem geordneten Paar $(p,q),\ p,q \in A^n$, gehört ein Vektor $v \in V^n$. Man schreibt $v = (\vec{pq})$.

(ii) Zu jedem $p \in A^n$ und jedem $v \in V^n$ existiert ein eindeutig bestimmtes $q \in A^n$, so daß $v = (\vec{pq})$.

(iii) Ist $v = (\vec{pq})$ und $w = (\vec{qr})$, dann gilt $v + w = (\vec{pr})$.

Die Elemente des affinen Raumes A^n heißen *Punkte*;
V^n heißt der zu A^n *assoziierte Vektorraum*.
Die Elemente aus V^n heißen Vektoren. Aus (iii) folgt, daß $(\vec{pp}) = 0$ und $(\vec{pq}) = -(\vec{qp})$.

Koordinatensystem des affinen Raumes (3.1.1.1, S.102)

Ist ein Koordinatensystem $(o; e_1, \ldots, e_n)$ gegeben, dann heißt für jeden Punkt $p \in A^n$ der Vektor $v = (\vec{op})$ *Ortsvektor* von p.
Die Komponenten von v bzgl. $(e_1, \ldots, e_n)$ heißen *Koordinaten* von p, d.h. p besitzt die Koordinaten $(x_1, \ldots, x_n)$ genau dann, wenn

$$(\vec{op}) = v = x_1 e_1 + x_2 e_2 + \cdots + x_n e_n.$$

Der Punkt o besitzt die Koordinaten $(0, \ldots, 0)$ und heißt *Ursprung* des Koordinatensystems.
Die Punkte p_i mit $(\vec{op_i}) = e_i$ heißen *Einheitspunkte*.
Die Menge der Punkte $(o, p_1, \ldots, p_n)$ eines Koordinatensystems wird als *affine Basis* bezeichnet.

Unterraum des affinen Raumes (3.1.2, S.102)

Eine nichtleere Teilmenge $B \subset A^n$ heißt *r-dimensionaler Unterraum* von A^n, falls ein r-dimensionaler Untervektorraum W von V^n existiert, so daß die folgenden beiden Bedingungen erfüllt sind:

(i) $p,q \in B \Rightarrow w = (\vec{pq}) \in W$

(ii) $p \in B, w \in W \Rightarrow$ Es gibt ein $q \in B$ mit $(\vec{pq}) = w$.

Dann ist B selbst ein affiner Raum.
Ein $(n-1)$-dimensionaler Unterraum von A^n heißt *Hyperebene*.

affiner Unterraum eines Vektorraumes (3.1.2, S.103)

Eine Teilmenge X eines Vektorraumes V heißt *affiner Unterraum* von V, falls es ein $v \in V$ und einen Untervektorraum $W \subset V$ gibt, so daß $X = v + W = \{u \in V |$ es gibt ein $w \in W$ mit $u = v + w\}$ gilt.

Beispiel:
Die eindimensionalen Unterräume sind Geraden.

Affinkombination (3.1.2, S.104)

Der Vektor v ist eine *Affinkombination* der Vektoren v_i, wenn

$$v = \sum_{i=0}^{n} \lambda_i v_i \quad \text{mit} \quad \sum_{i=0}^{n} \lambda_i = 1$$

gilt.

baryzentrische Koordinaten (3.1.2.1, S.104)

Sei $\mathcal{B} = \{p_0; (p_0\vec{p}_1), \cdots, (p_0\vec{p}_n)\}$ ein Koordinatensystem in einem affinen Raum A^n und sei ein Punkt

$$p = \sum_{i=0}^{n} \lambda_i \cdot p_i \quad \text{mit} \quad \sum_{i=0}^{n} \lambda_i = 1$$

als Affinkombination bezüglich dieses Koordinatensystems gegeben.
Dann heißen die Koeffizienten λ_i *baryzentrische Koordinaten* oder *Schwerpunktskoordinaten* von p.

konvexe Hülle (3.1.2.2, S.105)

Die *konvexe Hülle* von Punkten ist die kleinstmögliche Menge von Punkten, bei der mit je zwei Punkten auch die Verbindungsstrecke in der Menge liegt:

$$co\{p_0, \cdots, p_n\} = \{p \,|\, p = \sum_{i=0}^{n} \lambda_i p_i, \ \sum_{i=0}^{n} \lambda_i = 1 \text{ und } \ \lambda_i \geq 0, i = 0, \cdots, n\}.$$

Beispiele:
Strecke mit den Begrenzungspunkten p_1 und p_2 :
$co\{p_1, p_2\} = \{p | p = \lambda \cdot p_1 + (1-\lambda) \cdot p_2, 0 \leq \lambda \leq 1\}$
Dreieck mit Eckpunkten p_0, p_1, p_2, wobei die p_0, p_1, p_2 drei verschiedene Punkte in der affinen Ebene sind.

euklidischer Raum (3.1.2.3, S.105)

Wenn der zum affinen Raum A^n assoziierte Vektorraum V^n ein n-dimensionaler Raum mit Skalarprodukt ist, so heißt A^n *euklidischer Raum.*
(Für V^n gibt es dann eine orthonormale Basis $(e_1, \ldots, e_n)$.)

affine Abbildung (3.1.3, S.105)

Eine Abbildung Φ: $A_1 \to A_2$ zwischen zwei affinen Räumen A_1 und A_2 ist affin, wenn

$$\Phi\left(\sum_{i=0}^{n} \lambda_i \cdot p_i\right) = \sum_{i=0}^{n} \lambda_i \cdot \Phi(p_i)$$

für jede endliche Folge $\lambda_0, \cdots, \lambda_n \in \mathbb{R}$ mit $\sum_{i=0}^{n} \lambda_i = 1$ gilt.
Aus der Definition folgt, daß eine affine Abbildung eindeutig durch die Abbildung der affinen Basis festgelegt ist.

Zentralprojektion (3.2, S.106)

Gegeben sei ein Beobachtungspunkt q (Punkt des Beobachters), eine Gerade g, auf die projiziert wird, und eine dazu nichtparallele Gerade g' als Objekt. Der Beobachtungspunkt q soll dabei auf keiner der beiden Geraden g bzw. g' liegen. Nun wird einem Punkt $p' \in g'$ (Objektpunkt) der Schnittpunkt $p \in g$ der Geraden durch p' und q zugeordnet. Die Punkte auf der Objektgeraden g' werden dabei auf die Gerade g projiziert.

projektiver Raum (3.2.2, S.108)

Ist ein *reeller affiner Raum A* gegeben, so nennen wir die Menge aller Geraden durch den Ursprung den *reellen projektiven Raum* $P(A)$.
Die Dimension $dim\, P(A)$ des projektiven Raumes $P(A)$ wird $dim\, P(A) = dim(A) - 1$ gesetzt.
Ein projektiver Raum der Dimension eins bzw. zwei heißt *projektive Gerade* bzw. *projektive Ebene.*
Projektive lineare Teilräume des projektiven Raumes $P(A)$ werden ausgehend von einem linearen Teilraum des zu A korrespondierenden Vektorraumes von A analog definiert.

homogene Koordinaten (3.2.2.1, S.108)

Ist $v \in \mathbb{R}^4$ ein von Null verschiedener Vektor, so definiert v eine Gerade g durch den Ursprung des $\mathbb{R}^4$ mit der Parametrisierung $g(\lambda) = \lambda \cdot v,\ \lambda \in \mathbb{R}$, die wir zur Abkürzung mit $\mathbb{R} \cdot v$ (*projektiver Punkt*) bezeichnen.
Auf diese Weise erhält man eine Abbildung von dem Vektorraum $\mathbb{R}^4$ in den projektiven Raum $P(\mathbb{R}^4)$, genauer

$$\mathbb{R}^4 - \{0\} \to P(\mathbb{R}^4),\ v \longmapsto \mathbb{R} \cdot v.$$

Zwei von Null verschiedene Vektoren v, v' bestimmen genau dann denselben projektiven Punkt, d.h. dieselbe Gerade durch den Ursprung, wenn es ein $\lambda \in \mathbb{R} - \{0\}$ gibt mit $v' = \lambda v$.
Ist $v = (x, y, z, w) \neq 0 \in \mathbb{R}^4$ gegeben, so bezeichnen wir mit

$$[x, y, z, w] = \mathbb{R} \cdot (x, y, z, w)$$

die *homogenen Koordinaten* des projektiven Punktes $\mathbb{R} \cdot v$.
Zur Unterscheidung schreiben wir sie in eckigen Klammern.
Dabei ist zu beachten, daß immer mindestens eine der Koordinaten x, y, z, w von Null verschieden ist und daß $[x, y, z, w] = [x', y', z', w']$ genau dann gilt, wenn es ein $\lambda \neq 0$ gibt mit $x' = \lambda x, y' = \lambda y, z' = \lambda z, w' = \lambda w$.

projektive Basis (3.2.2.3, S.110)

Ein $(r+1)$-Tupel $(p_0, \cdots, p_r)$ im projektiven Raum $P(\mathbb{R}^4)$ heißt *projektiv unabhängig,* wenn es linear unabhängige Vektoren $v_0, \cdots, v_r \in \mathbb{R}^4$ gibt mit $p_i = \mathbb{R} \cdot v_i,\ i = 0, \cdots, r$.
Ein Fünftupel $(p_0, \cdots, p_4)$ von Punkten aus $P(\mathbb{R}^4)$ heißt *projektive Basis* von $P(\mathbb{R}^4)$, wenn je vier davon projektiv unabhängig sind.
Im dreidimensionalen projektiven Raum $P(\mathbb{R}^4)$ ist eine *projektive Basis* durch die von den Vektoren

$$v_0 = [1, 0, 0, 0],\ v_1 = [0, 1, 0, 0],\ v_2 = [0, 0, 1, 0],\ v_3 = [0, 0, 0, 1],$$

$$v_4 = [1, 1, 1, 1],$$

definierten Punkte $p_0, \cdots, p_4$ gegeben, also durch fünf projektive Punkte festgelegt.

projektive Abbildung (3.2.3, S.112)

Eine Abbildung f zwischen zwei projektiven Räumen $P(V)$ und $P(W)$ heißt *projektiv,* wenn es eine injektive lineare Abbildung $F : V \to W$ gibt mit $f(\mathbb{R} \cdot v) = \mathbb{R} \cdot F(v)$ für jedes vom Nullvektor verschiedene $v \in V$.

Man schreibt dafür kurz $f = P(F)$.
Zwei lineare Abbildungen $F, F' : V \to W$ definieren dieselbe projektive Abbildung genau dann, wenn es ein $\lambda \neq 0$ gibt mit $F' = \lambda \cdot F$.
Wie bei linearen und affinen Abbildungen sind auch projektive Abbildungen durch ihre Wirkung auf die Basis eines projektiven Raumes festgelegt. Da der reelle projektive dreidimensionale Raum $P(\mathbb{R}^4)$ fünf Basiselemente besitzt, ist eine projektive Abbildung vom $P(\mathbb{R}^4)$ in den $P(\mathbb{R}^4)$ durch die Abbildung von fünf geeigneten Punkten eindeutig festgelegt.

Matrizendarstellung für projektive Abbildungen (3.2.3.1, S.112)

Ist eine projektive Abbildung $f : P(\mathbb{R}^4) \to P(\mathbb{R}^4)$ gegeben, so betrachten wir die zugehörige lineare bijektive Abbildung $F : \mathbb{R}^4 \to \mathbb{R}^4$. Diese Abbildung läßt sich als 4×4-Matrix A beschreiben, die bis auf einen Skalar $\lambda \neq 0$ festgelegt ist. Solche Matrizen heißen *homogen.*
Wird f durch die Matrix

$$A = \begin{pmatrix} a_{00} & \cdots & a_{03} \\ \vdots & & \vdots \\ a_{30} & \cdots & a_{33} \end{pmatrix}$$

dargestellt, so bildet f den Punkt p mit den homogenen Koordinaten $[x,y,z,w]$ auf den Punkt

$$[x,y,z,w] \cdot A = [a_{00}x + \cdots + a_{30}w, \cdots, a_{03}x + \cdots + a_{33}w]$$

ab.

Translationsmatrix (3.3.1, S.114)

Die Gleichung $p' = T(p) = T(0) + p = (x_0, y_0, z_0) + (x, y, z)$ beschreibt eine Translation T im reellen dreidimensionalen affinen Raum. Die zur Translation gehörende homogene Matrix hat dann die folgende Gestalt:

$$\begin{aligned} [x', y', z', w'] &= [x, y, z, 1] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ x_0 & y_0 & z_0 & 1 \end{bmatrix} \text{ oder} \\ p' &= p \cdot A. \end{aligned}$$

Dabei wurde $w = w_0 = 1$ gewählt.

Skalierungsmatrix (3.3.2.1, S.116)

Bei einer Skalierung S ergibt sich für die affinen Basisvektoren folgende Beziehung:

$$\begin{aligned} S((1,0,0)) &= (s_1, 0, 0) \\ S((0,1,0)) &= (0, s_2, 0) \\ S((0,0,1)) &= (0, 0, s_3). \end{aligned}$$

Man erhält als Skalierungsmatrix

$$[x', y', z', w'] = [x, y, z, w] \begin{bmatrix} s_1 & 0 & 0 & 0 \\ 0 & s_2 & 0 & 0 \\ 0 & 0 & s_3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Der Sonderfall $s_1 = s_2 = s_3 = s$ bedeutet die gleiche Skalierung für alle Koordinaten.

Scherungsmatrix (3.3.2.2, S.117)

Bei einer Scherung *SH* ergibt sich für die affinen Basisvektoren folgende Beziehung:

$$\begin{aligned} SH((1,0,0)) &= (1,s_1,s_3) \\ SH((0,1,0)) &= (s_2,1,s_4) \\ SH((0,0,1)) &= (s_5,s_6,1) \end{aligned} .$$

Man erhält als Scherungsmatrix

$$[x',y',z',w'] = [x,y,z,w] \begin{bmatrix} 1 & s_1 & s_3 & 0 \\ s_2 & 1 & s_4 & 0 \\ s_5 & s_6 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} .$$

Rotationsmatrizen (3.3.2.3, S.117)

Rotation um die Koordinatenachsen:

1. Rotation um die *z*-Achse mit dem Winkel α :

$$[x',y',z',w'] = [x,y,z,w] \begin{bmatrix} \cos\alpha & \sin\alpha & 0 & 0 \\ -\sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} .$$

2. Rotation um die *x*-Achse mit dem Winkel α :

$$[x',y',z',w'] = [x,y,z,w] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha & 0 \\ 0 & -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} .$$

3. Rotation um die *y*-Achse mit dem Winkel α :

$$[x',y',z',w'] = [x,y,z,w] \begin{bmatrix} \cos\alpha & 0 & -\sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ \sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} .$$

Die Drehwinkel sind in dem Rechtssystem x,y,z immer positiv.

Rotation um eine beliebige Achse durch den Ursprung:

Eine Drehung um eine beliebige Achse in Richtung des *normierten* Vektors (x,y,z) mit dem Winkel α kann auf Drehungen um die Koordinatenachsen zurückgeführt werden. Verknüpft man diese Drehungen, d.h. multipliziert die zugehörigen Rotationsmatrizen, so ergibt sich folgende homogene Darstellung:

$$\begin{bmatrix} tx^2+c & txy+sz & txz-sy & 0 \\ txy-sz & ty^2+c & tyz+sx & 0 \\ txz+sy & tyz-sx & tz^2+c & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} ,$$

wobei $s = \sin\alpha$, $c = \cos\alpha$, $t = 1-\cos\alpha$.

Für kleine Winkel α ($\alpha < 1°$) kann man die Bogenlängen $\sin\alpha$ durch α und $\cos\alpha$ durch 1 approximieren:

$$[x',y',z',w'] = [x,y,z,w] \begin{bmatrix} 1 & z\alpha & -y\alpha & 0 \\ -z\alpha & 1 & x\alpha & 0 \\ y\alpha & -x\alpha & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} .$$

Rotation um eine Achse, die nicht durch den Ursprung verläuft:
Man verschiebe das Rotationszentrum in den Ursprung, führe die Rotation durch und verschiebe das Rotationszentrum zurück:

$$p' = p \cdot T^{-1} \cdot R \cdot T.$$

Transformation von Normalen (3.3.3, S.119)
Wird z.B. eine affine Ebene E im $\mathbb{R}^4$ durch die Gleichung

$$x \cdot n^t + d = 0$$

mit Normalenvektor $n = (n_x, n_y, n_z, n_w)$, $x = (x_x, x_y, x_z, x_w)$ und $d \in \mathbb{R}$ beschrieben, so ist zu beachten, daß sich Normalenvektoren nicht mit derselben Matrix A wie die Ortsvektoren transformieren:

$$x \cdot n^t = \underbrace{(x \cdot A)}_{x'} \cdot \underbrace{A^{-1} n^t}_{\left(n \cdot (A^{-1})^t\right)^t}.$$

Wechsel des Koordinatensystems (3.3.4, S.120)
Sind zwei Koordinatensysteme Ψ und Ψ' mit den Basisvektoren e_1, e_2, e_3, e_4 und e'_1, e'_2, e'_3, e'_4 gegeben und beschreibt die Matrix A die Abbildung, welche die Basisvektoren e_i auf die Basisvektoren e'_i, $i = 1, \cdots, 4$ abbildet, so berechnen sich die Koordinaten $[x'_1, x'_2, x'_3, x'_4]$ eines Punktes im Koordinatensystem Ψ' aus den Koordinaten $[x_1, x_2, x_3, x_4]$ desselben Punktes im Koordinatensystem Ψ durch

$$[x'_1, x'_2, x'_3, x'_4] = [x_1, x_2, x_3, x_4] \cdot A^{-1}.$$

ebene geometrische Projektion (3.3.5, S.120)
Die *ebenen geometrischen Projektionen* sind dadurch charakterisiert, daß mit Projektionsstrahlen konstanter Richtung, d.h. entlang von Geraden, auf Ebenen projiziert wird.
Diese Projektionen werden in diesem Skript durch eine invertierbare Transformation und anschließende Parallelprojektion entlang einer Koordinatenachse dargestellt.
Zu diesen Projektionen gehören die *Parallelprojektion (rechtwinklige Projektion, schiefwinklige Projektion)* und die *perspektivischen Projektionen*.

Parallelprojektion (3.3.5.1, S.122)
Bei den Parallelprojektionen unterscheidet man zwischen rechtwinkligen und schiefwinkligen Projektionen.
Bei rechtwinkligen Projektionen steht die Projektionsrichtung senkrecht auf der Projektionsebene, d.h. sie ist parallel zur Normalen der Projektionsebene.
Bei schiefwinkligen Projektionen steht die Projektionsrichtung nicht senkrecht auf der Projektionsebene und bildet mit der Normalen der Projektionsebene einen Winkel.
Bei Parallelprojektionen bleiben Längen auf Geraden parallel zur Projektionsebene erhalten.

rechtwinklige Projektion (3.3.5.2, S.122)

Parallelprojektion entlang der z-Achse auf die Ebene $z = z_0$:

$$[x', y', z', w'] = [x, y, z, w] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & z_0 & 1 \end{bmatrix}.$$

In den meisten Fällen wird man für x_0, y_0 oder z_0 den Wert Null wählen.

Transformation:
Die Lage des Bildschirmkoordinatensystems wird meist so gewählt, daß sein Ursprung mit dem Ursprung des 3D-Koordinatensystems zusammenfällt und die z-Richtung die Oben-Richtung (ViewUp) definiert. Die Projektion der Oben–Richtung auf die Projektionsebene definiert y'.
Seien $p = (p_x, p_y, p_z)$ und $o = (o_x, o_y, o_z)$ die normierte Projektions- bzw. Obenrichtung. Dann läßt sich das rechtshändige Bildschirmkoordinatensystem x', y', z' wie folgt berechnen:

1) $e_{z'} = -p$.
2) Da $e_{x'}$ zu der von p und o aufgespannten Ebene senkrecht ist, gilt

$$e_{x'} = \frac{o \times e'_z}{||o \times e'_z||}.$$

3) $e_{y'} = e_{z'} \times e_{x'}$.

Dann wird die zu projizierende Szene in diesem Koordinatensystem dargestellt (vgl. Abschnitt 3.3.4). Anschließend wird die obige Parallelprojektion entlang der z'–Achse durchgeführt.

schiefwinklige Projektion (3.3.5.3, S.127)

Bei den schiefwinkligen Projektionen ist die Projektionsebene meistens parallel zu zwei Koordinatenachsen, die Projektionsrichtung jedoch nicht senkrecht zu der Projektionsebene.
Bei der *Kavalierprojektion* bildet die Projektionsrichtung mit der Normalen der Projektionsebene einen Winkel von 45°.
Daher bleiben die Längen auf Geraden senkrecht zur Projektionsebene bei dieser Projektion unverändert.
Bei der *Kabinettprojektion* werden Längen auf Geraden senkrecht zur Projektionsebene um den Faktor $\frac{1}{2}$ verkürzt. Der Winkel β zwischen Projektionsebene und Projektionsrichtung ist daher $\arctan(2) = 63°$.
Abhängig von α und β ergibt sich die normierte Projektionsrichtung

$$p = (-\cos\alpha \cdot \cos\beta, -\sin\alpha \cdot \cos\beta, -\sin\beta).$$

Stimmt die Projektionsebene mit der x,y-Ebene überein (vgl. Bild 3.22), so lassen sich die Matrizendarstellungen der Kavalier- und Kabinettprojektion wie folgt bestimmen:
Wie bei den rechtwinkligen Projektionen wird nach einer Transformation eine Parallelprojektion entlang der z'-Achse durchgeführt.
Bei der Transformation werden die Einheitsvektoren e_x und e_y auf sich

selbst (die Punkte in der Projektionsebene bleiben fest) und die normierte Projektionsrichtung p auf die z'-Achse abgebildet, d.h. auf ein Vielfaches λ von $e_{z'}$. Damit die Tiefeninformation bei dieser Transformation erhalten bleibt, muß $\lambda > 0$ sein, kann sonst aber beliebig gewählt werden. Wählt man $\lambda = 1$ und trägt die drei Bildvektoren wieder in die Zeilen einer 3×3-Matrix ein, so erhält man die Matrix

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -\cos\alpha\cos\beta & -\sin\alpha\cos\beta & -\sin\beta \end{bmatrix}.$$

Zum Wechsel in das Bildschirmkoordinatensystem wird die Inverse dieser Matrix benötigt. Gemäß Formel 3.23 wird die inverse 3×3-Matrix in eine 4×4-Matrix eingebettet, und es ergibt sich folgende Transformations-Matrix:

$$[x', y', z', w'] = [x, y, z, w] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -\frac{\cos\alpha}{\tan\beta} & \frac{-\sin\alpha}{\tan\beta} & -\frac{1}{\sin\beta} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Die Komposition mit der anschließenden Parallelprojektion entlang der z'-Achse liefert die Matrix

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -\frac{\cos\alpha}{\tan\beta} & \frac{-\sin\alpha}{\tan\beta} & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

perspektivische Projektion (3.3.5.4, S.129)

Perspektivische Projektionen sind keine affinen Abbildungen, da sie z.B. Längenverhältnisse nicht invariant lassen: Vom Blickpunkt weit entfernte Objekte werden kleiner dargestellt als Objekte mit kleinem Abstand zum Blickpunkt (Augpunkt, Beobachtungspunkt). Diese Abbildungen sind projektive Abbildungen und lassen sich daher nur im projektiven Raum als Matrizen beschreiben.

Beispiel:
Zweidimensionaler Fall: Perspektivische Projektion eines affinen Punktes $P = (x, y)$ mit Augpunkt $A = (-x_0, 0)$ und Projektionsgerade $x = 0$:
Es ergibt sich die folgende 3×3-Matrix zur Beschreibung dieser Abbildung.

$$[x', y', 1] = [x, y, 1] \begin{bmatrix} 0 & 0 & 1 \\ 0 & x_0 & 0 \\ 0 & 0 & x_0 \end{bmatrix} = [x, y, 1] \begin{bmatrix} 0 & 0 & \frac{1}{x_0} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Wie schon bei den Parallelprojektionen zerlegen wir die Projektion in zwei Abbildungen

$$\begin{bmatrix} 0 & 0 & \frac{1}{x_0} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & \frac{1}{x_0} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{bzw.}$$

$$P = T_p \cdot P_p.$$

Die Matrix P_p beschreibt die Projektion auf die Gerade $x = 0$. Die Matrix T_p beschreibt die *perspektivische Transformation.*

perspektivische Transformation im zweidimensionalen Raum (3.3.5.5, S.130)

Einpunktperspektive:

Perspektivische Projektion eines affinen Punktes $P = (x,y)$ auf einen Bildpunkt auf der affinen Bildgeraden $x = 0$:

$$[x,y,w]\begin{bmatrix} 1 & 0 & \frac{1}{x_0} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = [x,y,\frac{x}{x_0}+w]$$

Zweipunktperspektive:

$$[x,y,w]\begin{bmatrix} 1 & 0 & \frac{1}{x_0} \\ 0 & 1 & \frac{1}{y_0} \\ 0 & 0 & 1 \end{bmatrix} = [x,y,\frac{x}{x_0}+\frac{y}{y_0}+w]$$

perspektivische Transformation im dreidimensionalen Raum (3.3.5.6, S.133)

$$[x,y,z,w]\begin{bmatrix} 1 & 0 & 0 & \frac{1}{x_0} \\ 0 & 1 & 0 & \frac{1}{y_0} \\ 0 & 0 & 1 & \frac{1}{z_0} \\ 0 & 0 & 0 & 1 \end{bmatrix} = [x,y,z,\frac{x}{x_0}+\frac{y}{y_0}+\frac{z}{z_0}+w]$$

allgemeine perspektivische Transformation (3.3.5.7, S.134)

Bei der praktischen Anwendung von Projektionen liegt der Standort des Betrachters beliebig im Objektraum. Eine allgemeine perspektivische Transformation wird durch die Festlegung eines Augpunktes (*Eye*), eines Blickbezugspunktes (*VRef*) und der Angabe einer Oben-Richtung (*ViewUp*) festgelegt.
Die Berechnung der Transformation wird in vier Schritten durchgeführt:

1. Berechnung des Bildschirmkoordinatensystems mit Ursprung *VRef* und den Basisvektoren v'_x, v'_y und v'_z.
   v'_z wird durch *VRef* und *Eye* definiert.
   v'_x steht senkrecht auf den Vektoren *ViewUp* und v'_z.
   Da die $y-$Achse des Bildschirmkoordinatensystems in die entgegengesetzte Richtung des *ViewUp*-Vektors zeigt, bilden die drei Vektoren v'_x, $ViewUp$ und v'_z ein linkshändiges, noch nicht notwendig orthogonales, Koordinatensystem.
   Der Vektor *ViewUp* wird daher durch den Vektor v'_y senkrecht zu v'_z und v'_x ersetzt, so daß v'_x, v'_y und v'_z ein rechtshändiges orthogonales Koordinatensystem bilden.
   Die Vektoren v'_x, v'_y und v'_z müssen normiert werden!

2. Translation des Bildschirmkoordinatensystems in den Ursprung.

3. Rotation des Bildschirmkoordinatensystems auf das Welt- bzw. Referenzkoordinatensystem:

$$v'_x \rightarrow x; \quad v'_y \rightarrow y; \quad v'_z \rightarrow z$$

4. Perspektivische Transformation mit Abstand $|VRef - Eye|$, wobei der Augpunkt (*Eye*) auf der negativen z-Achse liegt.

Lehrziele

Nach dem Durcharbeiten dieser Kurseinheit sollten Sie verstehen,

- warum die Parameterdarstellung eine herausragende Bedeutung für den Kurven- und Flächenverlauf hat,
- welche Eigenschaften die wichtigsten Polynombasen haben und für welche Aufgaben sie vorteilhaft eingesetzt werden können,
- wie parametrische und geometrische Stetigkeit zusammenhängen,
- wie ein Spline für eine Approximationsaufgabe zu ermitteln ist,
- wie der de Casteljau- und der de Boor-Algorithmus für Bézier- und B-Spline-Kurven funktionieren,
- wie eine Tensorprodukt-Fläche konstruiert wird,
- wie Spline-Flächen-Interpolation und Spline-Flächen-Approximation funktionieren,
- wie Bézierflächen über Dreiecken erzeugt werden,
- wie man Coons-Pflaster und Gordon-Flächen aufbaut.

Kapitel 4

Kurven und Flächen

In den folgenden Abschnitten beschreiben wir die wichtigsten mathematischen Methoden zur Darstellung von Kurven und Flächen. Wir beschränken uns dabei auf die Parameterdarstellung. Im Kurs werden in der Regel Kurven und Flächen im $\mathbb{R}^3$ behandelt. Abbildungen und Aufgaben beziehen sich der Einfachheit halber oft auf den $\mathbb{R}^2$.

4.1 Parameterdarstellungen von Kurven

Um die Theorie der parametrisierten Kurven und Flächen aus mathematischer Sicht besser verstehen zu können, rufen wir zunächst die wichtigsten mathematischen Begriffe wie parametrisierte Kurven, Funktionenräume und Basen ins Gedächtnis.

4.1.1 Mathematische Grundlagen

4.1.1.1 Parametrisierte Kurven

Eine Abbildung $q : \mathbb{R} \supset I = [a,b] \longrightarrow \mathbb{R}^3$ mit $a < b$ heißt *parametrisierte Kurve*.
Eine Kurve heißt *n–mal stetig differenzierbar*, wenn die Abbildung q mindestens n-mal stetig differenzierbar (kurz: $C^n - stetig$) ist, d.h. $q \in C^n$.
Anschaulich wird durch die Differenzierbarkeit der Kurve ihre Glattheit ausgedrückt.
$u \in [a,b]$ bezeichnet den Parameterbereich der Kurve. Oft beziehen wir uns auf das Intervall $[0,1]$.
Eine Kurve heißt *regulär*, wenn q einmal stetig differenzierbar ist und $q'(u) \neq 0$ für alle $u \in [a,b]$.
Der Gradient $q'(u)$ von q an der Stelle u ist ein Vektor im $\mathbb{R}^3$, der die Tangentenrichtung der Kurve an der Stelle u angibt.
Die Regularität einer Kurve besagt gerade, daß zu jedem Parameterwert ein von

Null verschiedener Tangentenvektor existiert.

Als erstes Beispiel einer parametrisierten Kurve betrachten wir eine Gerade durch zwei Punkte P_1, P_2 mit den Koordinaten (x_1, y_1, z_1) bzw. (x_2, y_2, z_2) des $\mathbb{R}^3$:
Als Parameterbereich I wählen wir ganz $\mathbb{R}$ und definieren die Gerade durch

$$q(u) = (1-u)P_1 + uP_2.$$

Für $u = 0$ ist $q(u) = P_1$, für $u = 1$ ist $q(u) = P_2$, für $u \in (0,1)$ liegt $q(u)$ auf der Strecke zwischen den Punkten P_1 und P_2, für alle anderen Werte von u außerhalb der Verbindungsstrecke. Da q in jeder Koordinate eine lineare Funktion ist, kann q beliebig oft differenziert werden. Weil $q'(u) = P_2 - P_1 \neq 0$ ist, falls $P_1 \neq P_2$, definiert q außer im Ausnahmefall $P_1 = P_2$ eine reguläre Kurve.

Als weiteres Beispiel einer parametrisierten Kurve betrachten wir eine nicht auf ihrem ganzen Parameterbereich reguläre Kurve. Als Parameterbereich wählen wir das Intervall $[-4,4]$ und definieren

$$q : [-4,4] \longrightarrow \mathbb{R}^3 \quad \text{durch} \quad q(u) = (u^3, u^2, 0).$$

Die Ableitung im Punkt $u = 0$ ist der Vektor $(0,0,0)$, d.h. die Kurve ist in diesem Punkt nicht regulär.

Beispiele von parametrisierten Kurven sind in Bild 4.1 dargestellt.

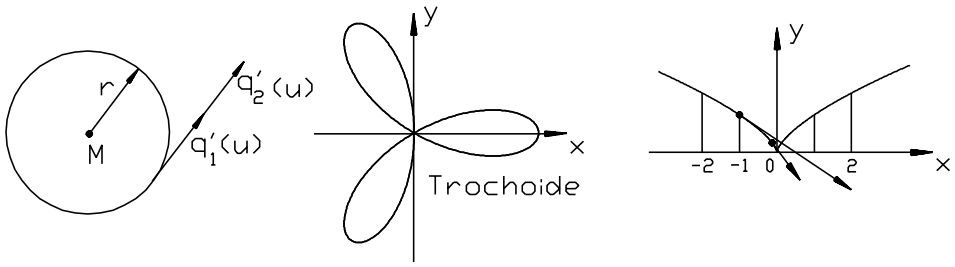


Abbildung 4.1: Parametrisierungen im $\mathbb{R}^2$
Der Kreis wurde auf zwei Arten durch $q_1(u) = (r\cos u, r\sin u),\ u \in [0, 2\pi)$, und $q_2(u) = (r\cos 2u, r\sin 2u),\ u \in [0,\pi)$, parametrisiert.
Der Tangentenvektor der Kurve q_2 ist doppelt so lang wie der der Kurve q_1.
Die Trochoide kann durch $q(u) = (\cos 3u \cos u, \cos 3u \sin u),\ u \in [0, 2\pi)$, parametrisiert werden.
Die dritte dargestellte Kurve ist durch $q(u) = (u^3, u^2)$, $u \in [-4,4]$, parametrisiert. Diese Kurve ist im Punkt $(0,0)$ nicht regulär.

In diesen Beispielen hätte man auch eine andere Parametrisierung wählen und damit dieselben Raumpunkte über einem anderen Parameterbereich beschreiben können. Das führt auf den Begriff der *Parametertransformation.*

Zwei reguläre Kurven $q_1 : [a,b] \to \mathbb{R}^3$, $q_2 : [c,d] \to \mathbb{R}^3$ heißen *äquivalent,* wenn es eine bijektive differenzierbare Abbildung $\varphi : [a,b] \to [c,d]$ mit $\varphi'(u) > 0$ gibt, so daß $q_1 = q_2 \circ \varphi$ gilt.

Wir sagen in diesem Fall auch, daß q_2 durch φ reparametrisiert wurde und nennen φ einen richtungserhaltenden Parameterwechsel oder eine *Reparametrisierung* von q_2.

Als Beispiel für eine Reparametrisierung betrachten wir die beiden Kurven q_1 und q_2 mit

$$q_1 : [0,1] \to \mathbb{R}^2, \quad u \mapsto (2u,u) \quad \text{und} \quad q_2 : [0,\frac{1}{2}] \to \mathbb{R}^2 : \quad u \mapsto (4u,2u)$$

und die Parametertransformation

$$\varphi : [0,1] \to [0,\frac{1}{2}] : \quad u \mapsto u/2.$$

Dann gilt $q_1 = q_2 \circ \varphi$, d.h. q_2 ist durch φ reparametrisiert.

Häufig veranschaulicht man die erste Ableitung einer Kurve in einem Punkt als Geschwindigkeitsvektor eines die Kurve durchlaufenden Punktes. Dieser ist bei der Kurve q_2 genau doppelt so groß, wie bei der Kurve q_1. Die Kurve q_2 durchläuft denselben Wertebereich doppelt so schnell wie die Kurve q_1.
Als weiteres Beispiel ist in Bild 4.1 ein Kreis mit zwei verschiedenen Parametrisierungen dargestellt.

Die *Bogenlänge* $s(u)$ einer parametrisierten regulären Kurve $q : [a,b] \to \mathbb{R}^3$ läßt sich gemäß folgender Formel berechnen:

$$s : [a,b] \to [0,s(b)], \quad u \mapsto s(u) := \int_a^u \|q'(t)\| dt. \tag{4.1}$$

Dabei bezeichnet $\|.\|$ die euklidische Norm im $\mathbb{R}^3$.

Entspricht für $u \in [a,b]$ der Wert der Bogenlänge $s(u)$ der zugrundeliegenden Intervallänge $u-a$, so nennt man *q nach der Bogenlänge parametrisiert.*
Das ist äquivalent zu der Bedingung $\|q'(u)\| = 1, \quad u \in [a,b]$.
Jede reguläre Kurve läßt sich nach der Bogenlänge parametrisieren.

4.1.1.2 Funktionenräume

Die Menge $C[a,b]$ bezeichne die Menge aller stetigen, reellen Funktionen auf dem Intervall $[a,b] \subset \mathbb{R}$. Definiert man für Elemente $f,g \in C[a,b]$ eine Addition und Skalarmultiplikation durch

$$(\alpha f + \beta g)(t) = \alpha f(t) + \beta g(t) \tag{4.2}$$

mit $\alpha,\ \beta \in \mathbb{R}$, so ist $C[a,b]$ ein reeller Vektorraum.
In diesem Vektorraum heißen n Funktionen $f_1, \cdots, f_n \in C[a,b]$ *linear unabhängig*, falls aus $\sum c_i f_i = 0$ auf $[a,b]$ immer $c_1 = c_2 = \cdots = c_n = 0$ folgt.
Dabei ist $f = 0$ auf $[a,b]$, falls $f(t) = 0$ für alle $t \in [a,b]$.

4.1.1.3 Polynomräume

Die Menge aller Polynome $\mathbb{P}^n[a,b]$ vom Grad n bildet einen Unterraum von $C[a,b]$ der Dimension $n+1$. Die Monome $1,t,t^2\cdots t^n$ bilden eine Basis, die sogenannte *Taylorbasis* (Monombasis) des Polynomraums. In dieser Basis ist ein reelles Polynom p durch

$$p(t) = c_n t^n + c_{n-1} t^{n-1} + \cdots + c_1 t + c_0 \tag{4.3}$$

mit Taylorkoeffizienten $c_n,\cdots,c_0 \in \mathbb{R}$ gegeben.
Wählt man statt reeller Koeffizienten c_i Vektoren aus dem $\mathbb{R}^3$, so erhält man Polynomkurven im $\mathbb{R}^3$.

Für den interaktiven Entwurf von Kurven ist diese Darstellungsart der Kurven allerdings nicht besonders geeignet, da sich die Taylorkoeffizienten einer unmittelbaren geometrischen Deutung entziehen. Im folgenden werden wir nun einige Basen des Polynomraumes $\mathbb{P}^n$ diskutieren, die speziellen Aufgabenstellungen angepaßt sind.

4.2 Polynombasen zur Interpolation

4.2.1 Die Interpolationsaufgabe

Die im weiteren interessierende Interpolationsaufgabe läßt sich folgendermaßen formulieren:
Gesucht ist ein Polynom q, so daß folgende Bedingungen erfüllt sind:

$$P_i = q(u_i),\ u_i \in \mathbb{R},\ P_i \in \mathbb{R}^3,\ i = 0,\cdots,n, \tag{4.4}$$

wobei die u_i die Parameterwerte der zugehörigen Punkte P_i darstellen.

Die Punkte P_i heißen *Stützpunkte*, die u_i *Stützstellen* oder Parameterwerte, der Vektor $(u_0,\cdots,u_n)$ heißt *Stützstellenvektor*. Die Paare (u_i,P_i) legen die Knoten des Interpolationspolynoms fest (Bild 4.2).

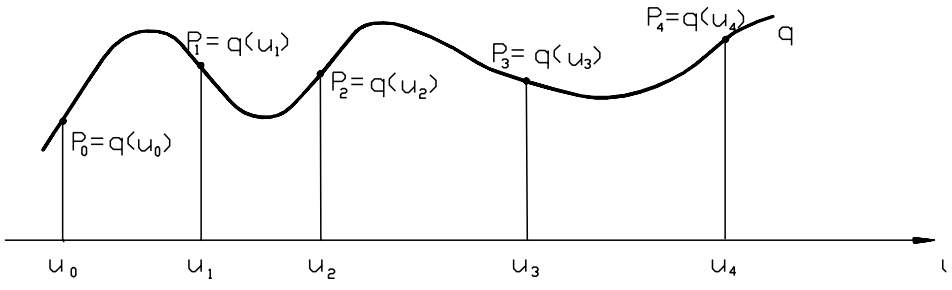


Abbildung 4.2: Prinzip der Interpolation
Dargestellt ist eine Kurve mit Stützstellen $u_i,\ i = 0,\cdots,4$.

Invarianz

Eine wichtige Eigenschaft von Polynomdarstellungen ist die Invarianz bei affinen Abbildungen. Gilt für die Basispolynome $f_0, \cdots, f_n$ einer Polynombasis von $\mathbb{P}^n[a,b],\ a<b,\ a,b\in\mathbb{R}$, und für alle $u\in[a,b]$ die Bedingung

$$f(u)=\sum_{i=0}^{n} f_i(u)=1, \tag{4.5}$$

so gilt für eine affine Transformation Φ und Stützpunkte $P_i,\ i=1,\cdots,n$:

$$\Phi\left(\sum_{i=0}^{n} P_i f_i(u)\right)=\sum_{i=0}^{n}\Phi(P_i) f_i(u). \tag{4.6}$$

Anschaulich gesprochen erhält man dieselbe Interpolationskurve, unabhängig davon, ob man zuerst die Stützpunkte affin verschiebt und dann die Kurve berechnet oder ob man zuerst die Kurve berechnet und dann die einzelnen Kurvenpunkte affin transformiert.

4.2.2 Interpolation mit Monomen

Sind $n+1$ Knoten (u_i,P_i) mit paarweise verschiedenen Stützstellen $u_i, (i=0,\cdots,n)$ gegeben, so suchen wir ein Polynom

$$q(u)=\sum_{j=0}^{n} c_j u^j, \quad c_j\in\mathbb{R}^3, \tag{4.7}$$

mit

$$P_i=q(u_i)=\sum_{j=0}^{n} c_j (u_i)^j, \quad i=0,\cdots,n. \tag{4.8}$$

Dazu sind die Koeffizienten c_i aus folgendem Gleichungssystem zu bestimmen:

$$\begin{pmatrix} 1 & u_0 & \cdots & u_0^n \\ \cdot & & & \cdot \\ \cdot & & & \cdot \\ \cdot & & & \cdot \\ 1 & u_n & \cdots & u_n^n \end{pmatrix} \begin{pmatrix} c_0 \\ \cdot \\ \cdot \\ \cdot \\ c_n \end{pmatrix} = \begin{pmatrix} P_0 \\ \cdot \\ \cdot \\ \cdot \\ P_n \end{pmatrix}. \tag{4.9}$$

Da die Stützstellen als paarweise verschieden vorausgesetzt sind, ist die zu dem obigen Gleichungssystem gehörende Matrix (*Vandermondsche Matrix*) invertierbar und die Koeffizienten c_i lassen sich eindeutig bestimmen. Diese Interpolationsmethode hat aber folgende Nachteile:

1. Die Lösung des obigen Gleichungssystems kann für großes n zu numerischen Problemen führen.

2. Wird nur ein Knoten geändert, so muß im allgemeinen das ganze System neu gelöst werden.

3. Kurven in Taylorbasis sind nicht affin invariant.

4. Durch Monominterpolation entstehende Kurvensegmente können in der Regel nur mit C^0-Stetigkeit aneinandergefügt werden.

5. Die Koeffizienten c_j sind nicht unmittelbar geometrisch interpretierbar.

4.2.3 Interpolation mit Lagrange-Polynomen

Bei dieser Interpolationsform wird anstelle der Taylorbasis eine andere Basis verwendet, die sogenannte *Lagrangebasis* (vgl. Bild 4.3).

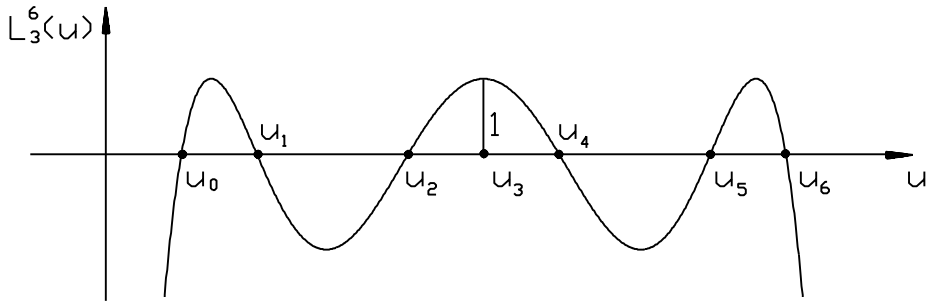


Abbildung 4.3: Lagrange-Polynom L_3^6: Über dem Stützstellenvektor $(u_0,\cdots,u_6)$ gilt $L_3^6(u_i)=\delta_{i3}$.

Sind $n+1$ Knoten $(u_i,P_i),\quad i=0,\cdots,n$, mit $u_0<u_1<\cdots<u_n$ gegeben, so sind die Basisfunktionen wie folgt definiert:

$$L_i^n(u)=\frac{(u-u_0)(u-u_1)\cdots(u-u_{i-1})(u-u_{i+1})\cdots(u-u_n)}{(u_i-u_0)(u_i-u_1)\cdots(u_i-u_{i-1})(u_i-u_{i+1})\cdots(u_i-u_n)}. \tag{4.10}$$

Die Basispolynome haben folgende Eigenschaft:

$$L_i^n(u_k)=\delta_{ik}=\left\{\begin{array}{lll} 1 & \text{für} & i=k \\ 0 & \text{für} & i\neq k \end{array}\right\}, \tag{4.11}$$

wobei δ_{ik} das Kronecker-Symbol ist.
Das Interpolationspolynom hat in dieser Basis die Form

$$q(u)=\sum_{i=0}^{n}P_iL_i^n(u)=\sum_{i=0}^{n}P_i\prod_{\substack{j=0\\ j\neq i}}^{n}\frac{(u-u_j)}{(u_i-u_j)} \tag{4.12}$$

Die Summe der Lagrangeschen Basisfunktionen besitzt folgende wichtige Eigenschaft:

$$\sum_{i=0}^{n}L_i^n(u)=1,\quad u\in[0,1]. \tag{4.13}$$

Daraus folgt die affine Invarianz der Lagrangeschen Interpolationskurve (vgl. Abschnitt 4.2.1 'Die Interpolationsaufgabe').

Beispiel (quadratische Interpolation im $\mathbb{R}^2$ mit Lagrange–Polynomen):

Vorgegeben seien die drei Knoten $(u_0,P_0),(u_1,P_1),(u_2,P_2)$ mit

$$u_0 = 0,\ u_1 = 1,\ u_2 = 2 \quad \text{und} \quad P_0 = (0,0), P_1 = (1,0) \quad \text{und} \quad P_2 = (2,1).$$

Das Interpolationspolynom hat dann folgende Form:

$$p(u) = (0,0)L_0^2(u) + (1,0)L_1^2(u) + (2,1)L_2^2(u). \tag{4.14}$$

Ein Vorteil der Lagrangeinterpolation ist, daß keine Polynomkoeffizienten berechnet werden müssen, da diese mit den zu interpolierenden Punkten übereinstimmen.
Nachteile sind:

1. Der Grad der Interpolationsfunktion hängt von der Anzahl der Bestimmungsstücke ab. Entsprechendes gilt für den Rechenaufwand zur numerischen Auswertung.

2. Lokale Änderungen sind nicht möglich.

3. Bei höheren Polynomgraden zeigt die Lagrange-Interpolation eine unerwünschte Welligkeit.

4. Wie bei den Monomen können Kurvensegmente nur mit C^0-Stetigkeit aneinandergefügt werden.

Wenn Sie die PDF-Version des Kurses bearbeiten, können Sie jetzt das Interaktionselement 'lagrange' anwählen.

4.2.4 Interpolation mit kubischen Hermite-Polynomen

Polynominterpolation ist nicht darauf beschränkt, Punkte zu interpolieren, sondern man kann auch Ableitungen an den Stützstellen interpolieren. Das führt zu einer weiteren nützlichen Polynombasis, der Hermitebasis. Im kubischen Fall (Polynome vom Grade drei) betrachten wir die folgenden vier Basisfunktionen, wobei wir uns auf das Intervall $[0,1]$ beschränken:

$$\begin{aligned} H_0^3(u) &= (1-u)^2(1+2u) \\ H_1^3(u) &= u(1-u)^2 \\ H_2^3(u) &= -u^2(1-u) \\ H_3^3(u) &= (3-2u)u^2 \end{aligned} \tag{4.15}$$

Die Basisfunktionen $H_i^3,\ i = 0,\cdots,3,$ heißen *kubische Hermite-Polynome.* Sie sind in Bild 4.4 dargestellt.

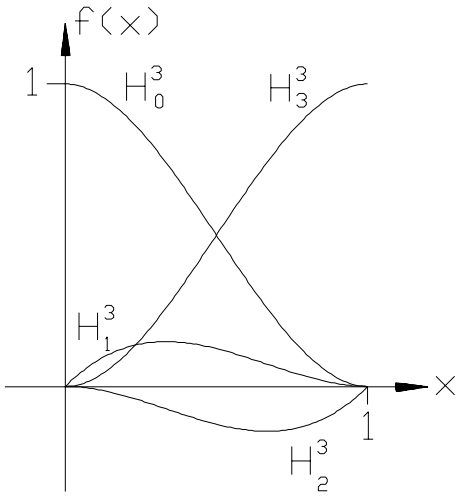


Abbildung 4.4: Hermite-Polynome über dem Intervall $[0,1]$

In der Hermitebasis gelten für die Koeffizienten $c_0,\cdots,c_3$ eines Polynoms

$$q(u) = c_0H_0^3(u) + c_1H_1^3(u) + c_2H_2^3(u) + c_3H_3^3(u)$$

dritten Grades

$$c_0 = q(0), \quad c_1 = q'(0), \quad c_2 = q'(1), \quad c_3 = q(1). \tag{4.16}$$

Daher können die Koeffizienten geometrisch gedeutet werden:
Sind zwei Punkte P_0, P_1 und zwei Tangentenvektoren m_0 und m_1 der Kurve in diesen Punkten zu interpolieren, so löst das Polynom

$$p(u) = P_0H_0^3(u) + m_0H_1^3(u) + m_1H_2^3(u) + P_1H_3^3(u) \tag{4.17}$$

die Interpolationsaufgabe. Beispiele sind in Bild 4.5 dargestellt.

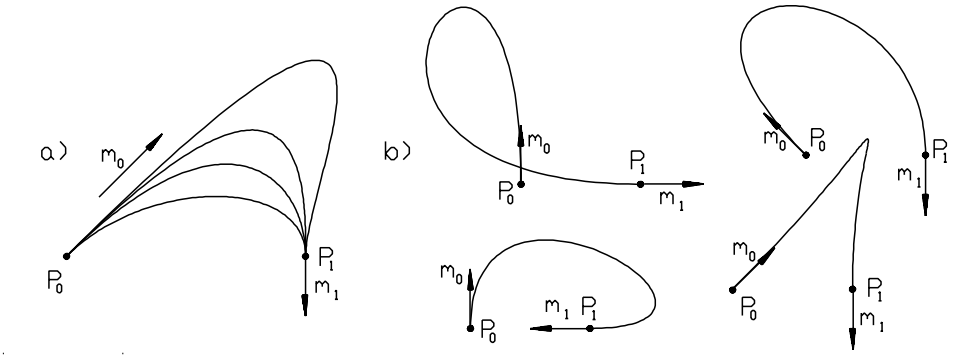


Abbildung 4.5: Die Koeffizienten eines Polynoms in Hermiteform sind Anfangs- und Endpunkt sowie Tangente im Anfangs- und Endpunkt.
a) Variation der Länge des Tangentenvektors im Anfangspunkt verändert die Form der Kurve.
b) Die Richtung der Tangenten wird variiert.

Bestand die Interpolationsaufgabe darin, ein Polygon zu bestimmen, das durch eine vorgegebene Folge von Knoten verläuft, so werden wir im folgenden Polygone

bzw. Polygonsegmente betrachten, die ein durch eine Folge von Knoten vorgegebenes Polygon zwar approximieren, aber nicht notwendig durch diese Knoten verlaufen.

4.3 Die Bernsteinbasis und Bézier -Kurven

4.3.1 Bernsteinpolynome

Die *Bernsteinpolynome* über dem Intervall $[s,t]$ lassen sich aus den binomischen Formeln

$$(t-s)^n = ((t-u)+(u-s))^n = \sum_{i=0}^{n} \left(\begin{array}{c} n \\ i \end{array} \right) (u-s)^i (t-u)^{n-i} \tag{4.18}$$

entwickeln. Die durch $(t-s)^n$ dividierten Summanden

$${}_s^t B_i^n(u) = \frac{1}{(t-s)^n} \left(\begin{array}{c} n \\ i \end{array} \right) (u-s)^i (t-u)^{n-i}, \quad i = 0,1,\cdots,n \tag{4.19}$$

sind Polynome vom Grad n und heißen Bernsteinpolynome vom Grad n.
Sie bilden eine Basis des Polynomraumes $\mathbb{P}^n$ und besitzen folgende Eigenschaf-

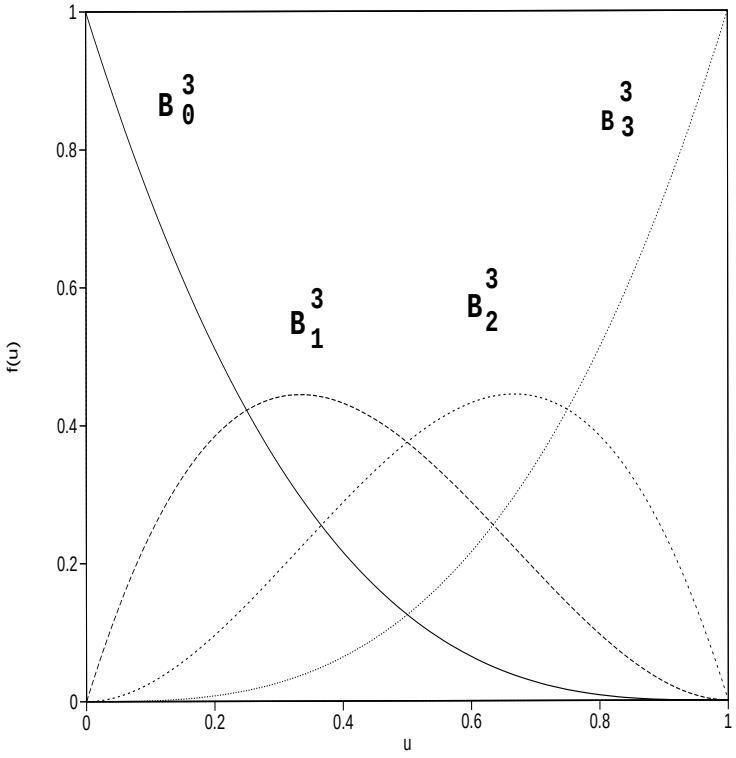


Abbildung 4.6: Die Bernsteinpolynome vom Grad 3 bilden eine Basis des $\mathbb{P}^3$.
Die Darstellung eines Polynoms in Bernsteinbasis heißt Bézier-Darstellung .

ten:

$$\begin{aligned}
\sum_{i=0}^{n} {}_s^t B_i^n(u) &= 1 \text{ für } u \in [s,t] && \text{Partition der 1}\\
{}_s^t B_i^n(u) &\geq 0 \text{ für } u \in [s,t] && \text{Positivität}\\
{}_s^t B_i^n(u) &= \tfrac{u-s}{t-s}\,{}_s^t B_{i-1}^{n-1}(u) + \tfrac{t-u}{t-s}\,{}_s^t B_i^{n-1}(u) && \text{Rekursion}\\
{}_s^t B_i^n(u) &= {}_s^t B_{n-i}^n(t-(u-s)) && \text{Symmetrie}
\end{aligned}$$

Häufig wird als Parameterintervall das Intervall $[0,1]$ gewählt. In diesem Fall schreibt man statt ${}_0^1 B_i^n$ einfach B_i^n.
Die Bernsteinpolynome vom Grad 3 über dem Intervall $[0,1]$ sind in Bild 4.6 dargestellt.

4.3.2 Bézier-Kurven

Die Darstellung von q mit

$$q(u) = \sum_{i=0}^{n} {}_s^t B_i^n(u) \cdot b_i, \quad b_i \in \mathbb{R}^3, \tag{4.20}$$

heißt *Bézier-Darstellung* im $\mathbb{R}^3$. Die Punkte $b_i,\ i = 0,\cdots,n,$ heißen Bézier-Punkte und definieren das Bézier-Polygon (auch Kontrollpolygon genannt).
Da

$$\sum_{i=0}^{n} {}_s^t B_i^n(u) = 1, \ \ u \in [s,t]$$

gilt, sind die Bézier-Kurven invariant unter affinen Transformationen.
Ist

$$q(u) = \sum_{i=0}^{n} {}_s^t B_i^n(u) \cdot b_i, \quad b_i \in \mathbb{R}^3,$$

eine Bézier-Kurve, so hängt die Gestalt der Kurve mit dem Kontrollpolygon wie folgt zusammen:

1. Für $u \in [s,t]$ liegt die Kurve $q(u)$ innerhalb der konvexen Hülle des Kontrollpolygons.

2. Es gilt
$$q(s) = b_0, \quad q(t) = b_n,$$
d.h. die Kurve verläuft durch Anfangs- und Endpunkt des Polygons.

3. Für die Ableitungen in den Anfangs- und Endpunkten gilt
$$q'(s) = \frac{n}{t-s}(b_1 - b_0), \quad q'(t) = \frac{n}{t-s}(b_n - b_{n-1}),$$
d.h. die Kurve besitzt als Tangenten die Anfangs- und Endseiten des Polygons.

4. Für die i–te Ableitung von q in den Randwerten $u = s$ bzw. $u = t$ gilt:

$$q^{(i)}(s) = \frac{1}{(t-s)^i} \frac{n!}{(n-i)!} \cdot \sum_{j=0}^{i} \begin{pmatrix} i \\ j \end{pmatrix} (-1)^{i-j} \cdot b_j$$

bzw.

$$q^{(i)}(t) = \frac{1}{(t-s)^i} \frac{n!}{(n-i)!} \cdot \sum_{j=0}^{i} \begin{pmatrix} i \\ j \end{pmatrix} (-1)^{j} \cdot b_{n-j},$$

d.h. die i-te Ableitung von q im Randwert $u = s$ hängt nur von dem Randpunkt selbst und seinen i unmittelbar benachbarten Bézier-Punkten ab. Analoges gilt im Randwert $u = t$.

Der Zusammenhang zwischen einer Bézier-Kurve und ihrem Kontrollpolygon ist in Bild 4.7 dargestellt.

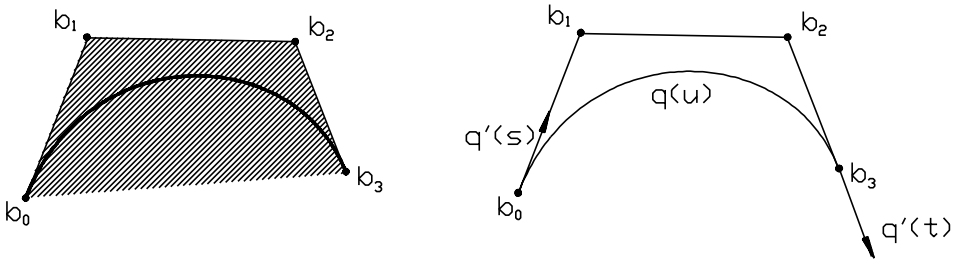


Abbildung 4.7: Die Bézier-Kurve verläuft durch Anfangs- und Endpunkt des Kontrollpolygons. In diesen Punkten ist die Bézier-Kurve tangential zum Kontrollpolygon. Alle Kurvenpunkte einer Bézier-Kurve liegen in der konvexen Hülle der Kontrollpunkte. Diese Eigenschaft heißt konvexe Hülleneigenschaft .

4.3.2.1 Der Algorithmus von de Casteljau

Der wichtigste Algorithmus zur Manipulation und Berechnung von Bézier-Kurven ist der Algorithmus von de Casteljau. Neben der stabilen Berechnung der Funktionswerte einer Bézier-Kurve (durch fortgesetzte Bildung von Konvexkombinationen) erlaubt er darüber hinaus die Bestimmung von Ableitungen und die Unterteilung der Kurve in mehrere Segmente. Die bei fortgesetzter Unterteilung entstehenden neuen Bézier-Polygone konvergieren gegen die Bézier-Kurve und können zur graphischen Darstellung der Kurve verwendet werden.

Ist

$$q(u) = \sum_{i=0}^{n} {}_s^t B_i^n(u) \cdot b_i, \qquad b_i \in \mathbb{R}^3,$$

eine Bézier-Kurve, so betrachten wir das durch die Rekursion

$$\begin{aligned} b_i^0(u) &= b_i \\ b_i^r(u) &= \frac{(u-s)}{(t-s)} \cdot b_{i+1}^{r-1}(u) + (1 - \frac{(u-s)}{(t-s)}) \cdot b_i^{r-1}(u) \end{aligned}$$

in Bild 4.8 definierte Schema. Mit Hilfe des de Casteljau-Algorithmus läßt sich der Funktionswert $q(u)$ an der Stelle u berechnen. Es gilt

$$q(u) = b_0^n(u).$$

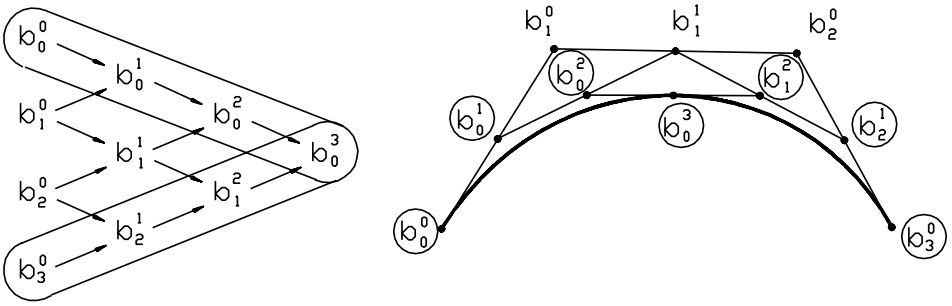


Abbildung 4.8: Schematische Darstellung des de Casteljau-Algorithmus im kubischen Fall
Der obere und untere Rand des Schemas enthält die Bézier-Punkte der Teilsegmente. Der Graph zeigt den Fall $u = \frac{s+t}{2}$.

Wenn Sie die PDF-Version des Kurses bearbeiten, können Sie jetzt das Interaktionselement 'bezier' anwählen.

4.3.2.2 Ableitung von Bézier-Kurven

Die i-te Ableitung $q^{(i)}(u)$ ergibt sich aus dem de Casteljau-Schema in der $(n-i)$-ten Stufe zu

$$q^{(i)}(u) = \frac{1}{(t-s)^i} \frac{n!}{(n-i)!} \sum_{k=0}^{i} \binom{i}{k} (-1)^{i-k} \cdot b_k^{n-i}(u). \tag{4.21}$$

Für die erste Ableitung gilt daher

$$q'(u) = \frac{n}{(t-s)} (b_1^{n-1}(u) - b_0^{n-1}(u)). \tag{4.22}$$

Im Fall einer kubischen Bézier-Kurve wird die erste Ableitung durch die Punkte b_1^2 und b_0^2 des de Casteljau-Schemas bestimmt. Insbesondere stimmt die Gerade durch diese beiden Punkte mit der Tangente an die Kurve im Punkt $q(u)$ überein (vgl. Bild 4.8).

4.3.2.3 Unterteilung von Bézier-Kurven

Mit Hilfe des de Casteljau-Algorithmus läßt sich eine Bézier-Kurve in zwei Teilsegmente zerlegen, die wieder in Bézier-Darstellung gegeben sind. Die neuen Bézier-Punkte der Teilsegmente ergeben sich aus dem oberen und unteren Rand

des Schemas (vgl. Bild 4.8).
Für $s \leq r \leq t$ gilt also die Beziehung

$$q(u) = \begin{cases} \sum_{i=0}^{n} {}_{s}^{r}B_i^n(u) \cdot b_0^i(r) & \text{für } s \leq u \leq r \\ \\ \sum_{i=0}^{n} {}_{r}^{t}B_i^n(u) \cdot b_i^{n-i}(r) & \text{für } r \leq u \leq t \end{cases} . \tag{4.23}$$

Hierbei bezeichnen ${}_{s}^{r}B_i^n(u)$ bzw. ${}_{r}^{t}B_i^n(u)$ die Bernstein-Polynome bezüglich der Teilintervalle $[s,r]$ bzw. $[r,t]$.

Bei fortgesetzter Unterteilung konvergiert die Folge der verfeinerten Kontrollpolygone gleichmäßig gegen die Kurve [LF80]. Bei fortgesetzter Halbierung des Parameterintervalls ist diese Konvergenz exponentiell und liefert ein praktisches Verfahren zur Darstellung einer Bézier-Kurve durch eine stückweise lineare Approximation.

Die Bézier-Kurven im $\mathbb{R}^3$ haben die Eigenschaft, daß eine beliebige Ebene im $\mathbb{R}^3$ die Kurve nicht öfter als das Kontroll-Polygon schneidet (Variation Diminishing Property).

4.3.2.4 Graderhöhung bei Bézier-Kurven

Werden zur genaueren Approximation einer geometrischen Figur mittels einer

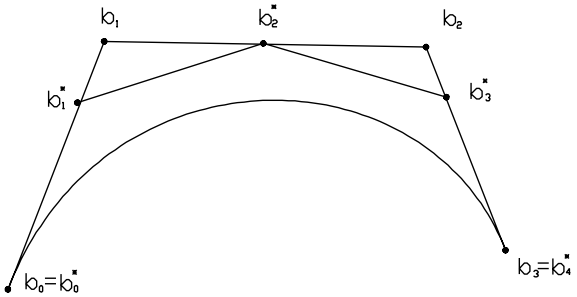


Abbildung 4.9: Graderhöhung einer kubischen Bézier-Kurve
Die neuen Bézier-Punkte liegen auf den Seiten des Kontrollpolygons.

Bézier-Kurve mehr Freiheitsgrade benötigt als vorhanden sind, so kann der Grad der Kurve erhöht werden. Bei Graderhöhung um den Grad eins wird dadurch ein weiterer Kontrollpunkt in das Kontrollpolygon eingefügt, ohne daß sich die geometrische Form der Kurve ändert.
Ist eine Bézier-Kurve q vom Grad n mit den Bézier-Punkten $b_0, \cdots, b_n$ gegeben, so ist q als Bézier-Kurve vom Grad $n+1$ darstellbar mit den Bézier-Punkten

$$b_0^* = b_0, \tag{4.24}$$

$$b_j^* = \frac{j}{n+1} \cdot b_{j-1} + (1 - \frac{j}{n+1}) \cdot b_j, \quad j = 1, \cdots, n, \tag{4.25}$$

$$b_{n+1}^* = b_n. \tag{4.26}$$

Die Graderhöhung einer Bézier-Kurve ist in Bild 4.9 dargestellt.

Wegen ihrer geometrischen Bedeutung wird die Graderhöhung auch als “Corner Cutting ” bezeichnet.

4.3.2.5 Vor- und Nachteile von Bézier-Kurven

Zusammenfassend besitzen die Bézier-Kurven folgende Vor- und Nachteile:

+ Das Bézier-Polygon vermittelt eine schnelle Übersicht über den möglichen Kurvenverlauf.
+ Änderungen der Kontrollpunkte erlauben kontrollierte Änderungen des Kurvenverlaufs.
- Der Grad der Kurve ist an die Anzahl der Ecken des Bézier–Polygons gekoppelt, was zu hohen Polynomgraden führt.
- Die Änderung eines Bézier-Punktes wirkt sich auf die gesamte Kurve aus und damit auch auf Bereiche, die eventuell nicht mehr geändert werden sollen.

4.4 Splines und ihre Stetigkeiten

Bei der Verwendung von Polynomen zur Darstellung von Kurven hängt der Polynomgrad direkt mit der Anzahl der Kontrollpunkte zusammen. Einen Ausweg aus dieser Situation bieten Splines.

Ein *Spline* ist eine stetige Abbildung q von einer Menge von Intervallen in den $\mathbb{R}^3$. Die Intervalle $[t_i, t_{i+1}],\ i = 0, \cdots, n-1$, werden durch einen *Stützstellenvektor*

$$T = (t_0, t_1 \cdots, t_n) \quad \text{mit} \quad t_0 \leq t_1 \leq \cdots \leq t_n$$

definiert. Jedes Intervall $[t_i, t_{i+1}]$ wird dabei auf ein Polynomsegment, das *Spline–Segment*, abgebildet. An den Stützstellen $t_i,\ i = 0, \cdots, n$, stoßen die Spline-Segmente nach Definition zusammen. Dort müssen die Splinesegmente aber, z.B. aufgrund unterschiedlicher Parametrisierungen, nicht unbedingt dieselben Tangentenvektoren besitzen. Sie können sich sowohl in Betrag als auch Richtung unterscheiden. Ist nur die Richtung der Tangentenvektoren gleich, ihre Länge jedoch verschieden, so besitzen die Splinesegmente an der Stützstelle dieselbe Tangente, allerdings ist die Kurve dort nicht differenzierbar. Analoges gilt für höhere Ableitungen. Daher unterscheidet man die Begriffe der parametrischen und der geometrischen Stetigkeit:

4.4.1 Parametrische und geometrische Stetigkeit

1. *Parametrisch stetiger Anschluß (C^n–stetiger Übergang)*:
   Seien $q_1 : [a_1, b_1] \to \mathbb{R}^3$, $q_2 : [a_2, b_2] \to \mathbb{R}^3$ zwei n-mal stetig differenzierbare

reguläre Kurven. q_1 und q_2 schließen an der Stelle b_1, a_2 C^n–stetig genau dann aneinander, wenn

$$q_1^{(k)}(b_1) = q_2^{(k)}(a_2) \quad \text{für } alle\ k = 0, \cdots, n. \tag{4.27}$$

2. *Geometrisch stetiger Anschluß (G^n–stetiger Übergang, G^n–Stetigkeit)*: Seien $q_1 : [a_1, b_1] \to \mathbb{R}^3$, $q_2 : [a_2, b_2] \to \mathbb{R}^3$ zwei n-mal stetig differenzierbare reguläre Kurven. q_1 und q_2 schließen an der Stelle b_1, a_2 G^n–stetig aneinander, falls es eine zu q_1 äquivalente Kurve $r_1 : [a_0, b_0] \to \mathbb{R}^3$ gibt, so daß r_1 und q_2 an der Stelle b_0, a_2 C^n–stetig aneinanderschließen.

Die G^n–Stetigkeit ist unabhängig von der Parametrisierung. Ferner sieht man aus dieser Definition, daß für reguläre Kurven aus C^n–Stetigkeit stets G^n–Stetigkeit folgt. Für nicht reguläre Kurven braucht dies nicht der Fall zu sein. Betrachtet man die durch $q(u) = (u^3, u^2)$, $u \in [-4, 4]$, definierte Kurve aus Bild 4.1 als zwei im Punkt 0 aneinanderschließende Kurven, so sind diese dort C^n–stetig, aber nicht G^n–stetig.

Geometrisch können wir die G^n-Stetigkeit nun so deuten:
G^0-Stetigkeit entspricht der C^0-Stetigkeit.
G^1-Stetigkeit am Übergang von q_1 und q_2 ist äquivalent dazu, daß der Übergang stetig ist und beide Tangentenvektoren am Übergang gleiche Richtung besitzen (vgl. Bild 4.10).

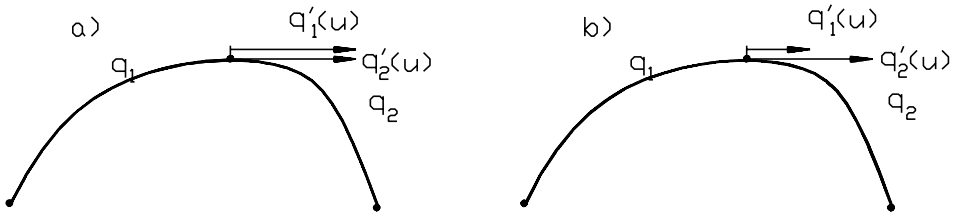


Abbildung 4.10: a) Bei einem C^1-stetigen Übergang stimmt sowohl die Richtung als auch die Länge der Tangentenvektoren von zwei Splinesegmenten überein.
b) Bei geometrischer Stetigkeit genügt das Übereinstimmen der Tangentenrichtungen.

Die G^2–Stetigkeit ist äquivalent dazu, daß q_1 und q_2 stetig sind, die Tangenten am Übergang dieselbe Richtung besitzen und ihre Krümmung dort übereinstimmt.

4.5 Bézier-Splines

Definiert man die Segmente eines Splines durch Bézier-Polynome, so spricht man von *Bézier-Splines*. Dabei können auch verschiedene Stetigkeitsanforderungen an die Knotenpunkte gestellt werden, die wir im folgenden diskutieren wollen (vgl. Bild 4.11).

Ist $q : \mathbb{R} \to \mathbb{R}^3$ ein Spline vom Grad n über $T = (t_0, \cdots, t_l)$ und

$$q_i(u) := \sum_{j=0}^{n} {}^{t_{i+1}}_{t_i}B_j^n(u) \cdot b_{i,j},\ u \in [t_i, t_{i+1}],$$

die Bézier-Darstellung des i–ten Splinesegments mit $\Delta_i := t_{i+1} - t_i$, dann gilt (ohne Beweis):

$$q \in C^0 \quad \Leftrightarrow \quad b_{i,n} = b_{i+1,0} \quad i = 0, \cdots, l-1 \tag{4.28}$$

$$\begin{aligned} q \in C^1 \quad \Leftrightarrow \quad & q \in C^0 \quad \text{und} \\ & b_{i,n} \cdot (\Delta_{i+1} + \Delta_i) = \Delta_i \cdot b_{i+1,1} + \Delta_{i+1} \cdot b_{i,n-1}, \\ & i = 0, \cdots, l-1 \end{aligned} \tag{4.29}$$

$$\begin{aligned} q \in C^2 \quad \Leftrightarrow \quad & q \in C^1 \quad \text{und} \\ & b_{i,n-1} + \frac{\Delta_{i+1}}{\Delta_i} \cdot (b_{i,n-1} - b_{i,n-2}) = b_{i+1,1} + \frac{\Delta_i}{\Delta_{i+1}}(b_{i+1,1} - b_{i+1,2}), \\ & i = 0, \cdots, l-1. \end{aligned} \tag{4.30}$$

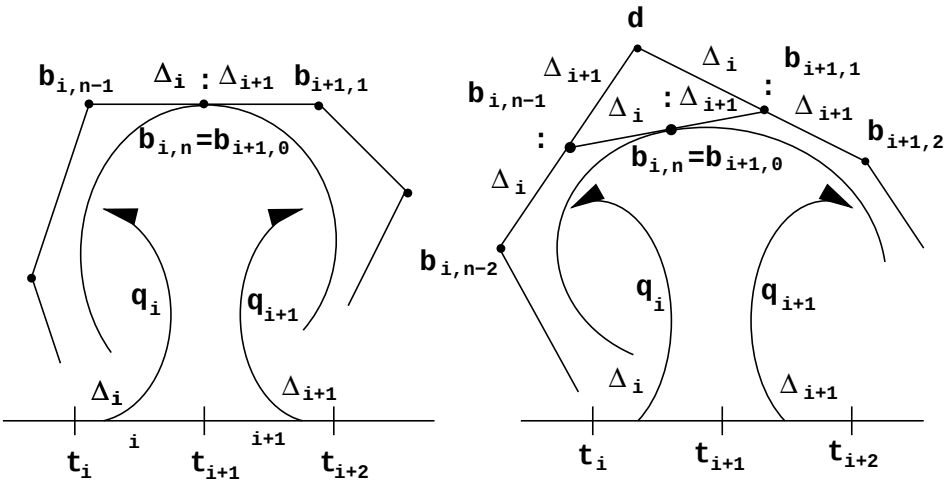


Abbildung 4.11: Bedingungen für C^1- und C^2-stetige Anschlüsse zwischen Bézier- Segmenten

4.6 B-Splines

Definiert man die Segmente eines Splines durch Verallgemeinerungen der Bernstein-Polynome, so spricht man von B-Splines bzw. B-Spline-Kurven. Ähnlich wie im Bézier-Fall haben die Koeffizienten einer B-Spline-Darstellung eine geometrische Bedeutung. Dadurch wird es möglich, diese Koeffizienten wieder als Kontrollpunkte für interaktiven Kurven- und Flächenentwurf zu verwenden. Ein weiterer Vorteil von B-Splines ist ihre Lokalität: Änderungen eines Kontrollpunktes wirken nicht mehr auf die gesamte Kurve, sondern nur noch lokal in der Nähe des geänderten Kontrollpolygons.

Anfangs wurden B-Splines vor allem im Kontext der Approximationstheorie untersucht [dB72], [dB78], [Sch81].

Gordon und Riesenfeld [Rie73], [GR74] führten B-Splines anschließend in den Bereich des Computer Aided Geometric Design ein. Seit dieser Zeit wurden - etwa durch die Algorithmen zum Einfügen neuer Knoten bei B-Spline-Kurven [Boe80], [CLR80] - erhebliche Fortschritte und zahlreiche Verallgemeinerungen erreicht. Eine umfassende Darstellung zur Theorie der B-Splines findet man z.B. in den ersten Kapiteln von [Sei89] oder in [HL89]. Dort findet man auch die Beweise zu den im folgenden beschriebenen Eigenschaften.

4.6.1 Die B-Spline-Basisfunktionen

Definition und elementare Eigenschaften

Gegeben seien $n \leq m \in \mathbb{N}$ sowie eine schwach monotone Folge

$$T = (t_0 = ... = t_n, t_{n+1}, ..., t_m, t_{m+1} = ... = t_{m+n+1})$$

von *Knoten* mit $t_i < t_{i+n+1}, 0 \leq i \leq m$.

Die normalisierten *B-Splines* N_i^n vom Grad n über T sind rekursiv definiert durch

$$N_i^0(u) := \begin{cases} 1 & \text{falls } t_i \leq u < t_{i+1} \\ 0 & \text{sonst} \end{cases} \tag{4.31}$$

und

$$N_i^r(u) \quad := \quad \frac{u - t_i}{t_{i+r} - t_i} \cdot N_i^{r-1}(u) \; + \frac{t_{i+1+r} - u}{t_{i+1+r} - t_{i+1}} \cdot N_{i+1}^{r-1}(u) \tag{4.32}$$

für $1 \leq r \leq n$.
Für $t_{i+r} = t_i$ bzw. $t_{i+r+1} = t_{i+1}$ ist in Gleichung 4.32 der entsprechende Summand gleich 0 zu setzen. Das folgt aus der Tatsache, daß dann $N_i^{r-1}(u) = 0$ bzw. $N_{i+1}^{r-1}(u) = 0$ verschwindet.

Da der Abstand $t_i - t_{i-1}, i = 0, \ldots, m+n+1$, aufeinanderfolgender Knoten nicht konstant ist, werden die normalisierten B-Splines als nicht uniforme normalisierte B-Splines bezeichnet.

Die normalisierten B-Splines $N_i^n(u)$ besitzen folgende Eigenschaften:

1. $N_i^n(u)$ besteht stückweise aus Polynomen vom Grad n über T.
2. Die Funktionen N_i^n besitzen einen lokalen Träger.
   In Formeln heißt dies
   $$N_i^n(u) = 0 \quad \text{für} \quad u \notin [t_i, t_{i+n+1}].$$
3. Es gilt $N_i^n(u) \geq 0$ für alle $u \in [t_0, t_{m+n+1}]$.
4. Wie die Bernsteinpolynome summieren sich die B-Splines für jeden Parameterwert $u \in [t_0, t_{m+n+1}]$ auf zu Eins:
   $$\sum_i N_i^n(u) = 1.$$

5. Ist t_j, $j \in \{0,\cdots,m+n+1\}$, ein einfacher Knoten, d.h. $t_{j-1} \neq t_j \neq t_{j+1}$, so ist $N_i^n(t_j)$ mindestens C^{n-1} stetig.

Normalisierte B-Splines sind in Bild 4.12 dargestellt.

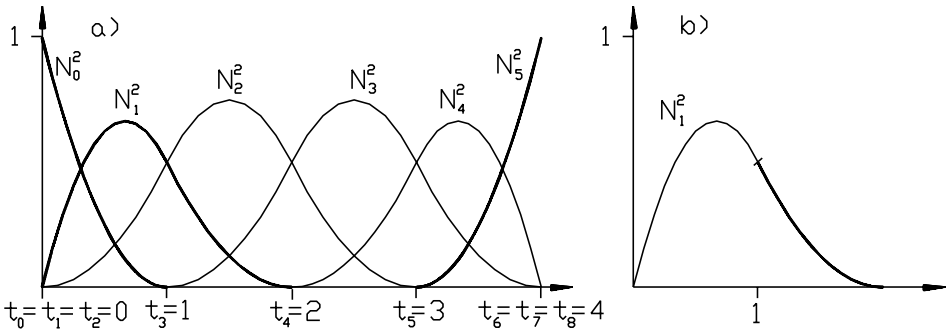


Abbildung 4.12: Normalisierte B-Splines
In a) sind die B-Splines N_j^2, $j = 0,\cdots,5$ über dem Knotenvektor $T = (0,0,0,1,2,3,4,4,4)$ dargestellt.
In b) ist der Übergang zwischen $N_1^2|_{[0,1)}$ und $N_1^2|_{[1,2)}$ dargestellt. Da $t_3 = 1$ ein einfacher Knoten ist, ist dieser Übergang C^1-stetig.

Alle aufgeführten Eigenschaften ergeben sich mittels Induktion unmittelbar aus der Definition.
Bezüglich der Stetigkeit an den Übergängen gilt folgendes:
Bei einem Mehrfachknoten $s = t_{j+1} = ... = t_{j+\mu}$ der Multiplizität μ sind die normalisierten B-Splines N_i^n vom Grad n mindestens $C^{n-\mu}$-stetig. Diese Eigenschaft ist in Bild 4.12 b) für $n = 2$ und $\mu = 1$ dargestellt.

4.6.2 B-Spline-Kurven

Gegeben seien $n \leq m \in \mathbb{N}$ sowie eine schwach monotone Folge

$$T = (t_0 = ... = t_n, t_{n+1}, ..., t_m, t_{m+1} = ... = t_{m+n+1})$$

von Knoten mit $t_i < t_{i+n+1}$ und Punkte $d_0, ..., d_m \in \mathbb{R}^3$.
Dann heißt eine Kurve

$$q(u) = \sum_{i=0}^{m} N_i^n(u) \cdot d_i, \qquad d_i \in \mathbb{R}^3, \tag{4.33}$$

eine *B-Spline-Kurve* (oder einfach ein *B-Spline*) vom Grade n über T.
Der Grad kann dabei vom Benutzer gewählt werden. Häufig beschränkt man sich jedoch auf kubische B-Splines.
Die Punkte $d_0, ..., d_m$ heißen *Kontroll-* oder *de Boor-Punkte* von q. Sie bilden das *Kontroll-* oder *de Boor-Polygon.*

4.6.2.1 Lokale Wirkung der de Boor-Punkte

Für die B-Splines gilt $N_i^n(u) = 0$, falls $u \notin [t_i, t_{i+n+1}]$. Daher beeinflußt der i-te de Boor-Punkt d_i die Kurve nur über dem Parameterbereich $[t_i, t_{i+n+1}]$. Die Form der Kurve wird über dem Intervall $[t_i, t_{i+n+1}]$ nur von den de Boor-Punkten $d_{i-n}, \cdots, d_{i+n}$ beeinflußt. Man kann die N_i^n daher als Schalter betrachten: Beim Segmentübergang schalten sie einen de Boor-Punkt ab und einen neuen ein (vgl. Bild 4.13).

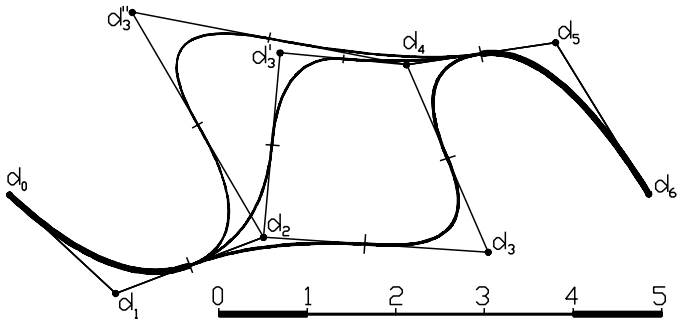


Abbildung 4.13: Lokale Wirkung der Kontrollpunkte:
$n = 2$, $T = (0,0,0,1,2,3,4,5,5,5)$
Durch Verschieben von d_3 nach d_3' werden lediglich die Kurvensegmente mit Parameterwerten $1 = t_3 \leq t < t_6 = 4$ beeinflußt.

4.6.2.2 Der Algorithmus von de Boor

Die Verallgemeinerung des de Casteljau-Algorithmus von Bézier-Kurven auf B-Splines ist der *de Boor-Algorithmus.*

Gegeben sei ein B-Spline

$$q(u) = \sum_{i=0}^{m} N_i^n(u) \cdot d_i$$

vom Grad n über T.
Für $t_l \leq u < t_{l+1}$ betrachten wir das durch die Rekursion

$$d_i^0(u) := d_i, \qquad i = l-n, ..., l, \tag{4.34}$$

und

$$\begin{aligned} d_i^r(u) \quad &:= \quad (1 - \frac{u - t_{i+r}}{t_{i+n+1} - t_{i+r}}) \cdot d_i^{r-1}(u) \\ &+ \quad \frac{u - t_{i+r}}{t_{i+n+1} - t_{i+r}} \cdot d_{i+1}^{r-1}(u), \; i = l-n, ..., l-r, \end{aligned} \tag{4.35}$$

für $0 \leq r \leq n$ definierte Dreiecksschema (vgl. Bild 4.14).

Dann gilt $q(u) = d_{l-n}^n(u)$.

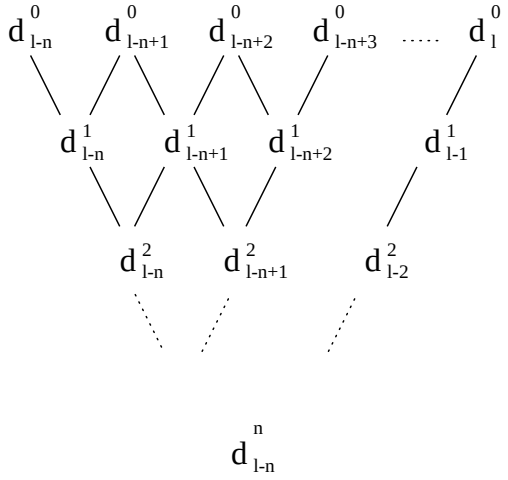


Abbildung 4.14: Der de Boor-Algorithmus

Im Spezialfall $T = (\underbrace{s,...,s}_{n+1},\underbrace{t,...,t}_{n+1})$ geht der de Boor-Algorithmus in den Algorithmus von de Casteljau über. Insbesondere stimmt dann die B-Spline-Darstellung einer B-Spline-Kurve q über T mit der Bézier-Darstellung von q über $[s,t]$ überein. Eine geometrische Deutung des de Boor-Algorithmus ist in Bild 4.15 dargestellt.

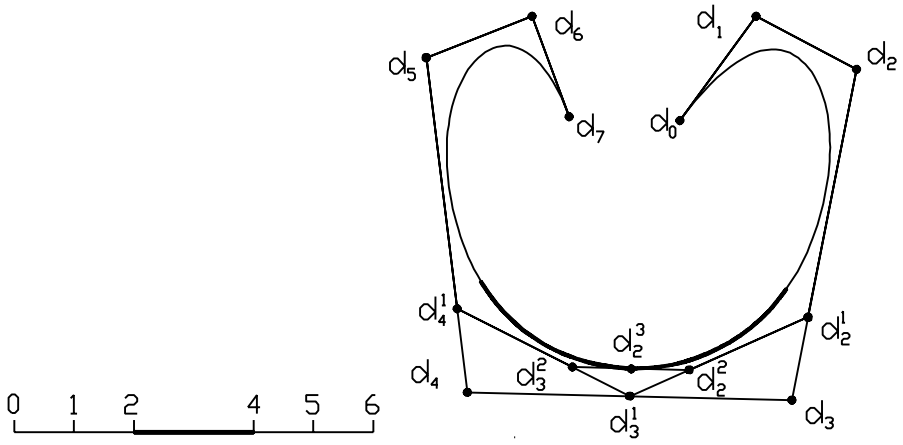


Abbildung 4.15: Geometrische Deutung des de Boor-Algorithmus
Dargestellt ist ein kubischer B-Spline über dem Knotenvektor
$T = (0,0,0,0,1,2,4,5,6,6,6,6)$
und die Auswertung für den Parameterwert $u = 3$.
Man beachte die Ähnlichkeit des Algorithmus von de Boor mit dem Algorithmus von de Casteljau. Der erste Schritt im de Boor-Algorithmus entspricht dem Einfügen eines neuen Knotens an der Stelle $u = 3$.

4.6.2.3 Der Zusammenhang zwischen Kontrollpolygon und B-Spline

Ist

$$q(u) = \sum_{i=0}^{m} N_i^n(u) \cdot d_i$$

eine B-Spline-Kurve vom Grad n über T in $\mathbb{R}^3$, dann hängt die Gestalt des Splines q mit dem Kontrollpolygon wie folgt zusammen:

1. Für $t_l \leq u \leq t_{l+1}$ liegt $q(u)$ in der konvexen Hülle der $n+1$ Kontrollpunkte $d_{l-n}, ..., d_l$.

2. Fallen n Kontrollpunkte $d_{l-n+1} = ... = d_l =: d$ zusammen, so gilt

   $$q(t_{l+1}) = d,$$

   d.h. die Kurve verläuft durch diesen n-fachen Kontrollpunkt.

3. Liegen $n+1$ Kontrollpunkte $d_{l-n}, ..., d_l$ auf einer Geraden L, so gilt

   $$q(u) \in L \quad \text{für} \quad t_l \leq u \leq t_{l+1},$$

   d.h. die Kurve hat mit der Geraden L ein Stück gemeinsam.

4. Fallen n Knoten

   $$t_{l+1} = ...t_{l+n} =: t$$

   zusammen, so ist

   $$q(t) = d_l$$

   ein Kontrollpunkt, und die Kurve ist dort tangential an das Kontrollpolygon. Insbesondere verläuft die Kurve bei $(n+1)$-fachen Anfangs- und Endknoten durch die Endpunkte des Polygons und ist dort tangential an das Kontrollpolygon.

Die hier aufgelisteten Eigenschaften sind in den Bildern 4.16-4.19 dargestellt.

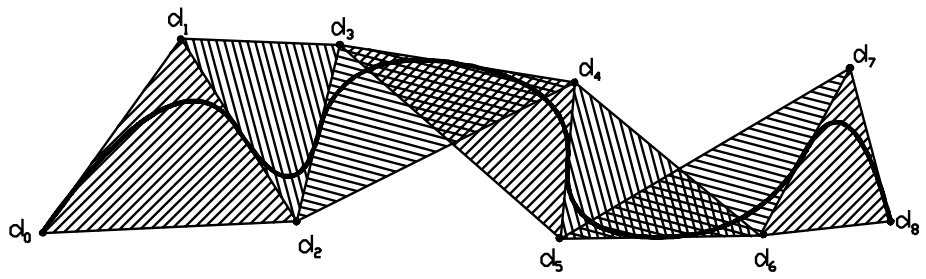


Abbildung 4.16: Die konvexe Hülleneigenschaft
Dargestellt wird die konvexe Hülle der Kurvensegmente für $n = 2$.

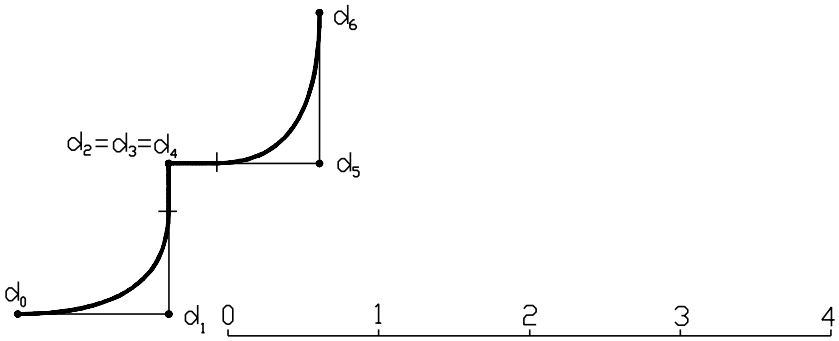


Abbildung 4.17: Die B-Splinekurve interpoliert einen n-fachen Kontrollpunkt.
$n = 3$, $T = (0,0,0,0,1,2,3,4,4,4,4)$.

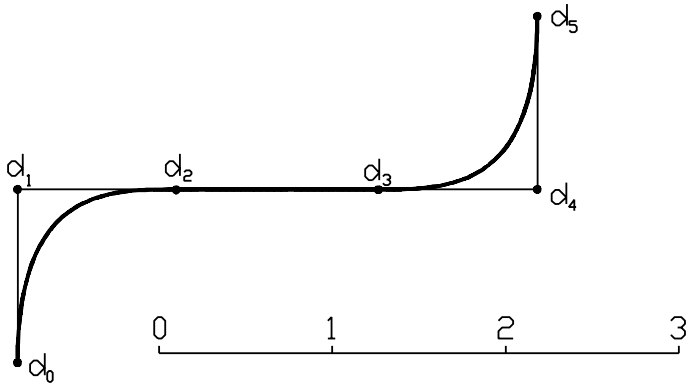


Abbildung 4.18: B-Splinekurve mit $n = 3,\ T = (0,0,0,0,1,2,3,3,3,3)$ mit d_1, d_2, d_3, d_4 kollinear
Die Kurve hat ein Geradenstück mit dem Polygon gemeinsam.

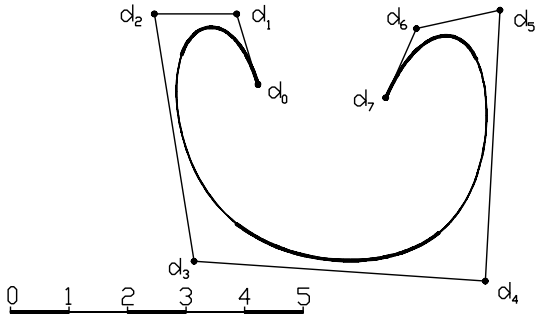


Abbildung 4.19: B-Splinekurve mit $n = 3,\ T = (0,0,0,0,1,2,3,4,5,5,5,5)$
Die einzelnen Polynomsegmente sind durch verschiedene Linientypen dargestellt.

4.6.2.4 Einfügen von Knoten

Von großer praktischer Bedeutung für das interaktive Arbeiten mit B-Spline-Kurven und -Flächen sind Algorithmen zum Einfügen neuer Knoten in einen gegebenen Knotenvektor [Boe80], [CLR80].

Einmal kann mit der Anzahl der Knoten auch die Anzahl der Kontrollpunkte erhöht werden. Der B-Spline läßt sich flexibler gestalten. Außerdem konvergieren die verfeinerten Kontrollnetze bei wiederholter Verfeinerung des Knotenvektors gleichmäßig gegen die Kurve und erlauben damit die rasche graphische Darstellung von B-Splines mittels einer stückweise linearen Approximation. Schließlich gestattet das Einfügen von Knoten bis zur Multiplizität n die Konvertierung der B-Spline-Darstellung in die Darstellung als stückweise Bézier-Kurve. Da die Verwendung stückweiser Bézier-Kurven vor allem in der Automobilindustrie stark verbreitet ist, ist auch diese Anwendung von erheblicher praktischer Bedeutung.

Gegeben sei eine B-Spline-Kurve

$$q(u) = \sum_{i=0}^{m} N_i^n(u) \cdot d_i$$

vom Grad n über dem Knotenvektor $T = (t_0, ..., t_{m+n+1})$. Nach Einfügen eines zusätzlichen Knotens t, etwa

$$t_l \leq t < t_{l+1}, \tag{4.36}$$

besitzt q eine Darstellung

$$q(u) = \sum_{i=0}^{m+1} N_i^n(u) \cdot d_i^* \tag{4.37}$$

als B-Spline vom Grad n über dem verfeinerten Knotenvektor

$$T^* = (t_0, ..., t_l, t, t_{l+1}, ..., t_{n+m+1}). \tag{4.38}$$

Die neuen Kontrollpunkte d_i^* werden wie folgt berechnet:

$$d_i^* = (1 - a_i) \cdot d_{i-1} + a_i \cdot d_i \tag{4.39}$$

mit

$$a_i = \begin{cases} 1 & \text{für } i \leq l - n \\ \frac{t - t_i}{t_{i+n} - t_i} & \text{für } l - n + 1 \leq i \leq l. \\ 0 & \text{für } l + 1 \leq i \end{cases} \tag{4.40}$$

Ein Vergleich dieses *Algorithmus von Boehm* mit dem de Boor-Algorithmus zeigt

$$d_i^* = d_{i-1}^1(t),$$

d.h. das Einfügen eines Knotens t bis zur Multiplizität n liefert den de Boor-Algorithmus (vgl. Bild 4.15).

Ausgabe der B-Splines

Bei fortgesetzter Unterteilung konvergiert die Folge der verfeinerten Kontrollpolygone gleichmäßig gegen die Kurve [LF80], [Pra84]. Hieraus ergibt sich ein praktisches Verfahren zur Darstellung einer B-Spline-Kurve durch eine stückweise lineare Approximation, ähnlich wie bei Bézier-Kurven.

Wenn Sie die PDF-Version des Kurses bearbeiten, können Sie jetzt das Interaktionselement 'bspline' anwählen.

Wenn Sie die PDF-Version des Kurses bearbeiten, können Sie jetzt das Interaktionselement 'curves' anwählen.

4.7 Parametrisierte Flächen

Wie in Abschnitt 4.1 'Parameterdarstellungen von Kurven' über parametrisierte Kurven rufen wir zunächst die wichtigsten mathematischen Begriffe wie parametrisierte Fläche, Tangentialebene und Normalenvektor ins Gedächtnis. Danach werden unterschiedliche Flächendarstellungen diskutiert.

4.7.1 Mathematische Grundlagen

4.7.1.1 Definition einer parametrisierten Flächen

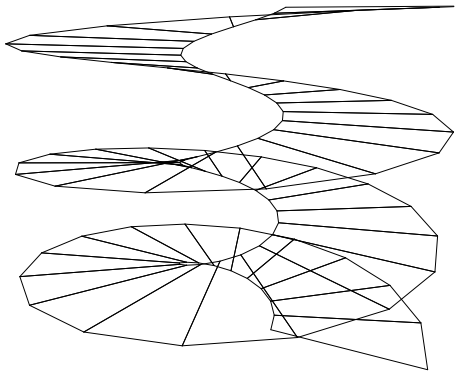


Abbildung 4.20: Wendelfläche

Sei $\emptyset \neq D \subset \mathbb{R}^2$ gegeben. Eine Abbildung $q : D \to \mathbb{R}^3$ heißt *parametrisierte Fläche*.
Oft verwenden wir $D = [0,1] \times [0,1]$ oder $D = \Delta(p_1, p_2, p_3)$ mit $p_1 = (0,0)$, $p_2 = (1,0)$ und $p_3 = (0,1)$.

Wenn Sie die PDF-Version des Kurses bearbeiten, können Sie jetzt das Interaktionselement 'parampatch' anwählen.

Eine Fläche heißt *n-mal stetig differenzierbar,* wenn die Abbildung q mindestens n-mal stetig differenzierbar ist, d.h. wenn q n-mal stetige partielle Ableitungen besitzt.

Anschaulich wird durch die Differenzierbarkeit der Fläche ihre Glattheit ausgedrückt.
Eine Fläche heißt *regulär,* wenn q einmal stetig differenzierbar ist und

$$q_u(u,v) := \frac{\partial q(u,v)}{\partial u}, \; q_v(u,v) := \frac{\partial q(u,v)}{\partial v} \text{ für alle } (u,v) \in D$$

linear unabhängig sind.

Die Vektoren $q_u(u,v)$ und $q_v(u,v)$ heißen *u-Tangente* bzw. *v-Tangente* von der Stelle (u,v).

Ist $q : D \to \mathbb{R}^3$ eine reguläre parametrisierte Fläche, so heißt die von den Vektoren $q_u(u_0,v_0)$ und $q_v(u_0,v_0)$ aufgespannte Ebene *Tangentialebene* $T_{q(u_0,v_0)}$ im Flächenpunkt $q(u_0,v_0)$.

Der Vektor

$$n(u,v) := \frac{q_u(u,v) \times q_v(u,v)}{\|q_u(u,v) \times q_v(u,v)\|}$$

heißt *Normalenvektor* im Punkt $q(u,v)$. Er steht senkrecht auf der Tangentialebene und ist unabhängig von der speziellen Wahl der Parametrisierung.

Ein Beispiel einer regulären parametrisierten Fläche ist die Wendelfläche

$$q : \mathbb{R}^2 \to \mathbb{R}^3, q(u,v) = (v\cos u, v\sin u, cu) \text{ mit } c \in \mathbb{R}.$$

Sie ist über dem Definitionsbereich $[0,6\pi] \times [0.25,1]$ in Abb. 4.20 dargestellt.

4.7.2 Tensorprodukt-Flächen

Sind $\{F_i^m,\ i = 0,\cdots,m\}$ und $\{G_j^n,\ j = 0,\cdots,n\}$ Basen von zwei Funktionenräumen R_1, R_2 mit reellwertigen univariaten Funktionen (Funktionen in einer Variablen) über den Intervallen $[a,b]$ bzw. $[c,d]$, so bilden die bivariaten reellwertigen Funktionen (Funktionen in zwei Variablen)

$$F_i^m G_j^n(u,v) = F_i^m(u) \cdot G_j^n(v), \quad i = 0,\cdots,m,\ j = 0,\cdots,n, \quad (u,v) \in [a,b] \times [c,d] \tag{4.41}$$

eine Basis des *Tensorproduktraumes* $R_1 \otimes R_2$. Dieser Raum hat die Dimension $(m+1)\cdot(n+1)$.

Als Beispiel betrachten wir die beiden Polynomräume $\mathbb{P}^m, \mathbb{P}^n$ über dem Intervall $[0,1]$ und als Basis dieser Räume die Bernsteinpolynome $B_i^m,\ i = 0,\cdots,m, B_j^n,\ j = 0,\cdots,n$.
Dann bilden die bivariaten Polynome

$$B_i^m B_j^n(u,v) = B_i^m(u) \cdot B_j^n(v), \quad i = 0,\cdots,m,\ j = 0,\cdots,n, \quad (u,v) \in [0,1] \times [0,1] \tag{4.42}$$

eine Basis des Tensorproduktraumes $\mathbb{P}^m \otimes \mathbb{P}^n$.

Ist eine Basis $F_i^m G_j^n,\ i = 0,\cdots,m,\ j = 0,\cdots,n$, eines Tensorproduktraumes $R_1 \otimes R_2$ von Funktionen über $[a,b] \times [c,d] \longmapsto \mathbb{R}$ und Koeffizienten $c_{ij} \in \mathbb{R}^3,\ i = 0,\cdots,m,\ j = 0,\cdots,n$, gegeben, so heißt die bezüglich der Tensorprodukt-Basis

dargestellte Funktion

$$q(u,v) = \sum_{i=0}^{m}\sum_{j=0}^{n} c_{ij}F_i^m G_j^n(u,v) = \sum_{i=0}^{m}\sum_{j=0}^{n} c_{ij}F_i^m(u)\cdot G_j^n(v), \qquad (u,v)\in[a,b]\times[c,d] \tag{4.43}$$

Tensorprodukt-Fläche.

In Matrizenschreibweise ergibt sich

$$q(u,v) = (F_0^m(u)\cdots F_m^m(u)) \begin{pmatrix} c_{00} & \cdots & c_{0n} \\ \cdot & & \cdot \\ \cdot & & \cdot \\ \cdot & & \cdot \\ c_{m0} & \cdots & c_{mn} \end{pmatrix} \begin{pmatrix} G_0^n(v) \\ \cdot \\ \cdot \\ \cdot \\ G_n^n(v) \end{pmatrix}. \tag{4.44}$$

Die Tensorprodukt-Flächen lassen sich leicht über Kurvendarstellungen gewinnen. Ist eine Polynom-Kurve oder ein Spline q im $\mathbb{R}^3$ durch

$$q(u) = \sum_{i=0}^{m} a_i F_i^m(u),\ \ u\in[a,b],\ a_i\in\mathbb{R}^3,$$

mit den F_i, $i=0,\cdots,m$, als Basisfunktionen gegeben, so kann man sich das Entstehen einer Fläche aus dieser Kurve folgendermaßen vorstellen: Die Kurve q wird durch den Raum bewegt, wobei auch Änderungen des Kurvenverlaufs von q zugelassen sind (vgl. Bild 4.21).

Die Änderung der Kurve beschreiben wir durch die Veränderung der Koeffizienten $a_i = a_i(v)$ und stellen diese bezüglich einer Basis $G_j^n,\ j=0,\cdots,n$ dar:

$$a_i(v) = \sum_{j=0}^{n} c_{ij}G_j^n(v),\ v\in[c,d].$$

Diese sich entlang einer Kurve verändernden Koeffizienten werden nun in obige Formel eingesetzt. Die dabei entstehende Fläche

$$q(u,v) = \sum_{i=0}^{m}\sum_{j=0}^{n} \underbrace{c_{ij}G_j^n(v)}_{a_i(v)} F_i^m(u),\ \ a\le u\le b,\ c\le v\le d, \tag{4.45}$$

ist eine Tensorprodukt-Fläche in obigem Sinne.

Als Beispiel seien vier beliebige, aber verschiedene Punkte $P_{00}, P_{01}, P_{10}, P_{11}$ im $\mathbb{R}^3$ gegeben, die nicht notwendig in einer Ebene liegen (vgl. Bild 4.21), z.B.

$$P_{00} = \begin{pmatrix} 0\\0\\0 \end{pmatrix},\quad P_{01} = \begin{pmatrix} 1\\0\\0 \end{pmatrix},\quad P_{10} = \begin{pmatrix} 0\\1\\0 \end{pmatrix},\quad P_{11} = \begin{pmatrix} 1\\1\\1 \end{pmatrix}.$$

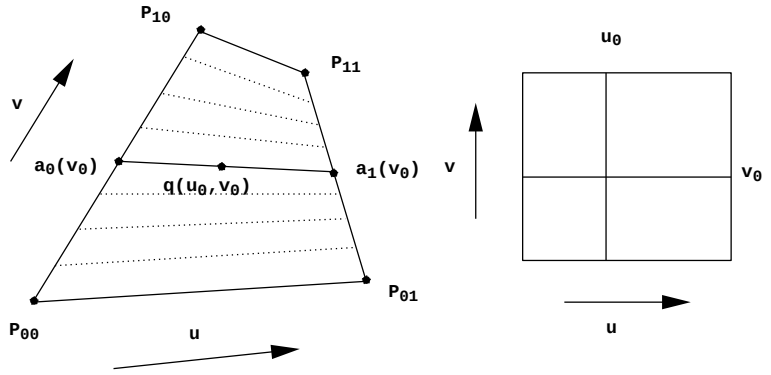


Abbildung 4.21: Konstruktion einer einfachen Tensorprodukt-Fläche

Zunächst interpolieren wir zwischen den Punkten P_{00}, P_{10} und den Punkten P_{01}, P_{11} durch zwei über dem Intervall $[0,1]$ parametrisierte, im allgemeinen windschiefe Geraden

$$a_0(v) = (1-v)P_{00} + vP_{10} \quad \text{und} \quad a_1(v) = (1-v)P_{01} + vP_{11}.$$

Zu jedem Parameterwert v_0 gehören genau zwei Punkte $a_0(v_0), a_1(v_0)$ auf je einer der Geraden a_0 und a_1. Zwischen diesen beiden Punkten interpolieren wir wieder durch eine über dem Intervall $[0,1]$ definierte Gerade und erhalten die folgende Tensorproduktfläche:

$$\begin{aligned}
q(u,v) &= (1-u)a_0(v) + ua_1(v) \\
&= (1-u)(1-v)P_{00} + (1-u)vP_{10} + u(1-v)P_{01} + uvP_{11} \\
&= \begin{pmatrix} 1-u & u \end{pmatrix} \begin{pmatrix} \begin{pmatrix} P_{00_1} \\ P_{00_2} \\ P_{00_3} \end{pmatrix} & \begin{pmatrix} P_{10_1} \\ P_{10_2} \\ P_{10_3} \end{pmatrix} \\ \begin{pmatrix} P_{01_1} \\ P_{01_2} \\ P_{01_3} \end{pmatrix} & \begin{pmatrix} P_{11_1} \\ P_{11_2} \\ P_{11_3} \end{pmatrix} \end{pmatrix} \begin{pmatrix} 1-v \\ v \end{pmatrix} \\
&= \begin{pmatrix} 1-u & u \end{pmatrix} \begin{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} & \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \\ \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} & \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \end{pmatrix} \begin{pmatrix} 1-v \\ v \end{pmatrix} \\
&= \begin{pmatrix} u & v & uv \end{pmatrix}.
\end{aligned}$$

Die so erhaltene Fläche $q(u,v) = (u,v,uv)$ ist ein hyperbolisches Paraboloid.

Wenn Sie die PDF-Version des Kurses bearbeiten, können Sie jetzt das Interaktionselement 'tensorproduct' anwählen.

Tensorprodukt-Flächen können über verschiedenen Basis-Funktionen erzeugt werden. Je nach Wahl der Basis können die Koeffizienten wie im Fall von Kurven geometrisch interpretiert werden.

Sind die Basisfunktionen F_i^m, G_j^n, $i = 0, \cdots, m$, $j = 0, \cdots, n$, Monome, so lautet die Tensorprodukt-Flächendarstellung

$$q(u,v) = \sum_{i=0}^{m} \sum_{j=0}^{n} c_{ij} u^i v^j, \quad a \le u \le b, \; c \le v \le d. \tag{4.46}$$

4.7.3 Spline-Flächen-Interpolation

Für die Spline-Flächen-Interpolation verwenden wir dieselben Methoden wie für die Interpolation von Kurven. Die so erzeugten Flächen haben die gleichen Vor- und Nachteile wie die Kurven und können daher aus den dort aufgelisteten Eigenschaften abgeleitet werden.

Die im weiteren interessierende Interpolationsaufgabe läßt sich folgendermaßen formulieren:
Gesucht ist eine Funktion q, so daß folgende Bedingungen erfüllt sind:

$$P_{ik} = q(u_i, v_k), \quad P_{ik} \in \mathbb{R}^3, \; u_i, v_k \in \mathbb{R}, \quad i = 0, \cdots, n, \quad k = 0, \cdots, m, \tag{4.47}$$

wobei die (u_i, v_k) die Parameterwerte der zu interpolierenden Punkte P_{ik} sind.

4.7.3.1 Monom- und Lagrange-Interpolation

Ist eine Tensorproduktfläche wie in Formel 4.43 durch

$$q(u,v) = \sum_{i=0}^{m} \underbrace{\sum_{j=0}^{n} c_{ij} G_j^n(v)}_{a_i(v)} F_i^m(u), \quad a \le u \le b, \; c \le v \le d, \tag{4.48}$$

und eine $(m+1) \times (n+1)$ große Matrix von Stützpunkten $P_{kl} \in \mathbb{R}^3$ und Stützstellen (u_k, v_l), $k = 0, \cdots, m$, $l = 0, \cdots, n$, gegeben und soll die Tensorproduktfläche diese Datenpunkte interpolieren, so muß

$$q(u_k, v_l) = \sum_{i=0}^{m} \sum_{j=0}^{n} c_{ij} F_i^m(u_k) G_j^n(v_l) = P_{kl}, \quad k = 0, \cdots, m, \; l = 0, \cdots, n, \tag{4.49}$$

für alle $(m+1) \times (n+1)$ Parameterpaare (u_l, v_k), $k = 0, \cdots, m$, $l = 0, \cdots, n$, und Datenpunkte P_{kl} erfüllt sein.

In Matrizenschreibweise lauten diese Gleichungen

$$q(u_k, v_l) = (F_0^m(u_k) \cdots F_m^m(u_k)) \begin{pmatrix} c_{00} & \cdots & c_{0n} \\ \cdot & & \cdot \\ \cdot & & \cdot \\ \cdot & & \cdot \\ c_{m0} & \cdots & cmn \end{pmatrix} \begin{pmatrix} G_0^n(v_l) \\ \cdot \\ \cdot \\ \cdot \\ G_n^n(v_l) \end{pmatrix} = P_{kl},$$
$$k = 0, \cdots, m,\ l = 0, \cdots, n.$$

Wir erhalten $(m+1) \cdot (n+1)$ Gleichungen, die wir zusammen in Matrizenform schreiben können:

$$\mathbf{P} = F\mathbf{C}G, \tag{4.50}$$

wobei

$$\mathbf{P} = \begin{bmatrix} P_{00} & \cdots & P_{0n} \\ \vdots & & \vdots \\ P_{m0} & \cdots & P_{mn} \end{bmatrix}, \tag{4.51}$$

$$F = \begin{bmatrix} F_0^m(u_0) & \cdots & F_m^m(u_0) \\ \vdots & & \vdots \\ F_0^m(u_m) & \cdots & F_m^m(u_m) \end{bmatrix}, \tag{4.52}$$

$$\mathbf{C} = \begin{bmatrix} c_{00} & \cdots & c_{0n} \\ \vdots & & \vdots \\ c_{m0} & \cdots & c_{mn} \end{bmatrix}, \tag{4.53}$$

$$G = \begin{bmatrix} G_0^n(v_0) & \cdots & G_0^n(v_n) \\ \vdots & & \vdots \\ G_n^n(v_0) & \cdots & G_n^n(v_n) \end{bmatrix}. \tag{4.54}$$

Sind die Parameterwerte $u_0, \cdots, u_m$ und $v_0, \cdots, v_n$ paarweise verschieden und verwendet man die Monom-Basis, so sind die Matrizen F und G Vandermond'sche Matrizen und daher invertierbar. Die Koeffizientenmatrix $\mathbf{C}$ ist dann gegeben durch

$$\mathbf{C} = F^{-1}\mathbf{P}G^{-1}. \tag{4.55}$$

Wird die Lagrange-Basis verwendet, d.h. die Funktionen F_i^m, $i = 0, \cdots, m$, G_j^n, $j = 0, \cdots, n$, sind die Lagrange'schen Basisfunktionen zu den Parameterwerten $u_0, \cdots, u_m$ bzw. $v_0, \cdots, v_n$, so gilt

$$F_i^m(u_k) = \delta_{ik} = \begin{cases} 1 & \text{für } i = k \\ 0 & \text{sonst} \end{cases}$$

und

$$G_j^n(v_l) = \delta_{jl} = \begin{cases} 1 & \text{für } j = l \\ 0 & \text{sonst} \end{cases}.$$

Daher sind F und G in diesem Fall Einheitsmatrizen und es gilt $\mathbf{C} = \mathbf{P}$. Wie im Fall von Kurven sind die Koeffizienten direkt durch die zu interpolierenden Punkte gegeben, d.h.

$$q(u,v) = \sum_{i=0}^{m} \sum_{j=0}^{n} P_{ij} L_i^m(u) L_j^n(v).$$

4.7.3.2 Bikubische Hermite-Interpolation

Ein bikubisches Hermite-Pflaster ist gegeben durch

$$q(u,v) = \sum_{i=0}^{3} \sum_{j=0}^{3} H_i^3(u) h_{ij} H_j^3(v),\ 0 \le u,v \le 1, \tag{4.56}$$

wobei die $H_i^3,\ i = 0, \cdots, 3$ die Hermite-Basisfunktionen sind. Verwendet man die Interpolationseigenschaften der Hermite-Basisfunktionen aus den Formeln 4.15 und bildet die einfachen und gemischten Ableitungen in den Eckpunkten, so folgt

$$[h_{ij}] = \begin{bmatrix} q(0,0) & q_v(0,0) & q_v(0,1) & q(0,1) \\ q_u(0,0) & q_{uv}(0,0) & q_{uv}(0,1) & q_u(0,1) \\ q_u(1,0) & q_{uv}(1,0) & q_{uv}(1,1) & q_u(1,1) \\ q(1,0) & q_v(1,0) & q_v(1,1) & q(1,1) \end{bmatrix}. \tag{4.57}$$

Dabei sind $q_u = \frac{\partial q}{\partial u}$, $q_v = \frac{\partial q}{\partial v}$ und $q_{uv} = \frac{\partial^2 q}{\partial u \partial v}$.
Die gemischten Ableitungen q_{uv} an den Ecken der Pflaster heißen *Twistvektoren*.
Ein Beispiel ist in Bild 4.22 dargestellt.

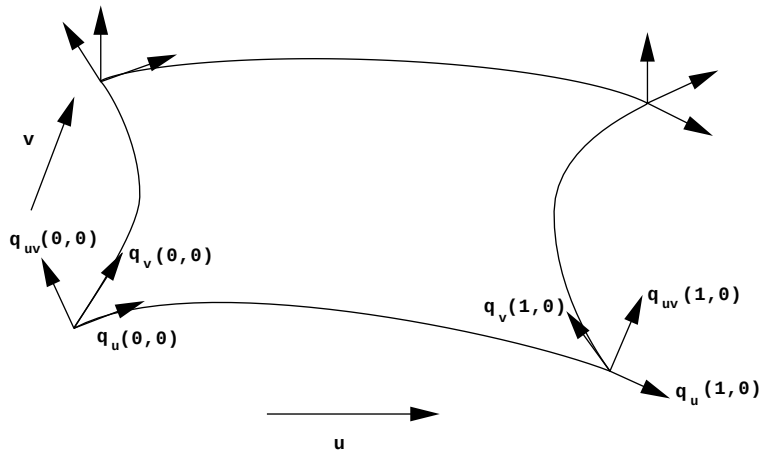


Abbildung 4.22: Bestimmung eines Pflasters durch Punkte und Tangentenvektoren bei der Hermite-Interpolation

4.7.4 Spline-Flächen-Approximation

4.7.4.1 Bézier-Spline-Flächen

Verwendet man die Bernstein-Bézier-Basis zur Darstellung von Pflastern, so haben die Koeffizienten geometrische Bedeutung.
Sind $o < r$ und $s < t \in \mathbb{R}$, so definieren wir ein Tensorprodukt-Bézier-Pflaster vom Grad (m,n) durch

$$q(u,v) = \sum_{i=0}^{m} \sum_{j=0}^{n} b_{ij}\, {}_o^r B_i^m(u)\, {}_s^t B_j^n(v), \quad b_{ij} \in \mathbb{R}^3, \quad i = 0 \cdots m, \; j = 0 \cdots n. \tag{4.58}$$

Dabei ist $u \in [o,r]$ und $v \in [s,t]$.
Die Koeffizienten b_{ij} heißen *Bézier-Punkte* und bilden das *Bézier-Netz.*
Ein Beispiel ist in Bild 4.23 dargestellt.

Die Parameterlinien mit konstantem v liefern Bézier-Kurven

$$q^v : [o,r] \to \mathbb{R}^3, \; u \longmapsto \sum_{i=0}^{m} {}_o^r B_i^m(u) \cdot b_i(v) \tag{4.59}$$

vom Grad m mit den Bézier-Punkten

$$b_i(v) = \sum_{j=0}^{n} {}_s^t B_j^n(v) \cdot b_{ij}. \tag{4.60}$$

Analoges gilt für konstantes u.

Ein wie in Gleichung 4.58 definiertes Bézier-Pflaster besitzt folgende Eigenschaften:

1. Wegen
$$\sum_{i=0}^{m} \sum_{j=0}^{n} {}_o^r B_i^m(u)\, {}_s^t B_j^n(v) = \sum_{i=0}^{m} {}_o^r B_i^m(u) \sum_{j=0}^{n} {}_s^t B_j^n(v) = 1$$
und
$${}_o^r B_i^m(u) \cdot {}_s^t B_j^n(v) \geq 0, \; (u,v) \in [o,r] \times [s,t],$$
liegt für $(u,v) \in [o,r] \times [s,t]$ die Fläche $q(u,v)$ in der konvexen Hülle des Bézier-Netzes.

2. Die Bézier-Punkte der Randkurven sind die Randpunkte des Bézier-Netzes. In Formeln gilt
$$q(u,s) = \sum_{i=0}^{m} {}_o^r B_i^m(u) \cdot b_{i0}, \tag{4.61}$$
$$q(u,t) = \sum_{i=0}^{m} {}_o^r B_i^m(u) \cdot b_{in}, \tag{4.62}$$
$$q(o,v) = \sum_{j=0}^{n} {}_s^t B_j^n(v) \cdot b_{0j}, \tag{4.63}$$
$$q(r,v) = \sum_{j=0}^{n} {}_s^t B_j^n(v) \cdot b_{mj}. \tag{4.64}$$

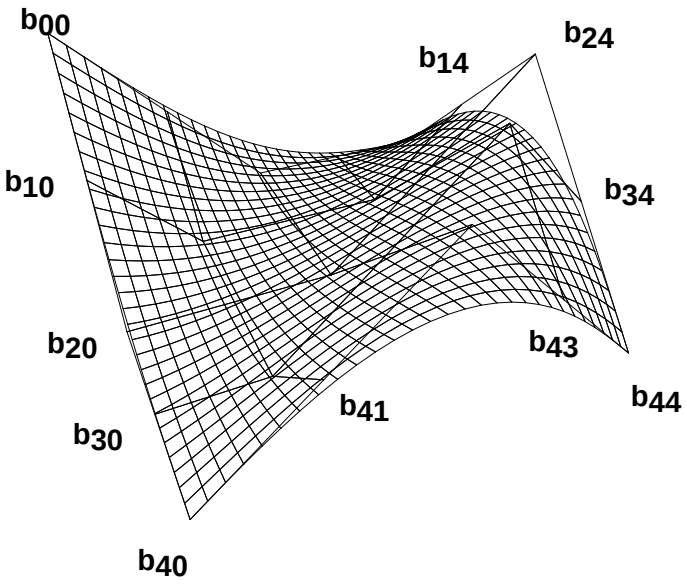


Abbildung 4.23: Bézier-Pflaster mit Bézier-Punkten und Parameterlinien für $m = n = 5$

3. Das Pflaster verläuft durch die Eckpunkte, d.h.
   $q(o,s) = b_{00}$, $q(o,t) = b_{0n}$, $q(r,s) = b_{m0}$, $q(r,t) = b_{mn}$.

4. Die Ableitungen in den Randpunkten lassen sich wie bei den Kurven aus den Bézier-Punkten berechnen. Es gilt:

$$\frac{\partial}{\partial u} q(u,v)|_{u=o} = \frac{m}{r-o} \sum_{j=0}^{n} {}_s^t B_j^n(v) \cdot (b_{1j} - b_{0j}). \tag{4.65}$$

   Die gemischte Ableitung, d.h. der Twistvektor T_{os}, z.B. im Eckpunkt mit den Parameterwerten o,s, berechnet sich aus

$$T_{os} = \frac{\partial^2}{\partial u \partial v} q(u,v)|_{u=o,v=s} = \frac{mn}{(r-o)(t-s)} (b_{00} - b_{01} + b_{11} - b_{10}). \tag{4.66}$$

   Um den Twist geometrisch zu interpretieren, formen wir die obige Gleichung um:

$$T_{os} = \frac{mn}{(r-o)(t-s)} (b_{11} - \underbrace{(b_{00} + (b_{10} - b_{00}) + (b_{01} - b_{00}))}_{P_{11}} \tag{4.67}$$

   Der Punkt P_{11} ist der vierte Punkt des von b_{00}, b_{01}, b_{10} definierten Parallelogramms. Der Twistvektor T_{os} ist proportional zur Differenz zwischen den Punkten b_{11} und P_{11} (vgl. Bild 4.24).

Der Twistvektor in diesem Eckpunkt kann auch als Änderung der u-Tangente in Richtung v interpretiert werden und umgekehrt als Änderung der v-Tangente in Richtung u.

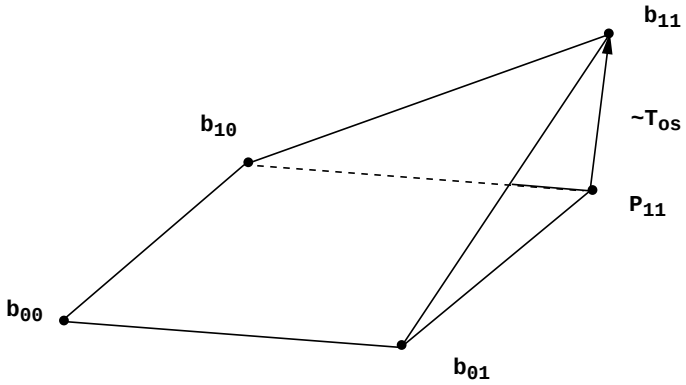


Abbildung 4.24: Twist: Die Koeffizienten des Twistvektors sind proportional zu der Abweichung des Punktes b_{11} von dem vierten Punkt P_{11} des durch b_{00}, b_{01}, b_{10} definierten Parallelogramms.

Algorithmen zur Manipulation und Auswertung von Bézier-Kurven übertragen sich direkt auf die Tensorprodukt-Bézier-Pflaster.

Wenn Sie die PDF-Version des Kurses bearbeiten, können Sie jetzt das Interaktionselement 'bezpatch' anwählen.

4.7.4.2 Zusammengesetzte Bézier-Pflaster

Um zwei Pflaster

$$q_1(u,v) = \sum_{i=0}^{m}\sum_{j=0}^{n} b_{ij}{}_{o_1}^{r_1} B_i^m(u)_{s_1}^{t_1} B_j^n(v),$$
$$i = 0,\cdots m,\ j = 0,\cdots n,\ (u,v) \in [o_1,r_1]\times[s_1,t_1],$$

und

$$q_2(u,v) = \sum_{i=0}^{m}\sum_{j=0}^{n} c_{ij}{}_{o_2}^{r_2} B_i^m(u)_{s_2}^{t_2} B_j^n(v),$$
$$i = 0,\cdots m,\ j = 0,\cdots n,\ (u,v) \in [o_2,r_2]\times[s_2,t_2],$$

entlang der Randkurve in v-Richtung aneinanderschließen zu können, müssen erstens die beiden Pflaster dieselbe Randkurve in v-Richtung besitzen, d.h. für die Bézier-Punkte gilt

$$b_{mj} = c_{0j},\ j = 0,\cdots,n,$$

und zweitens müssen entlang der Randkurve, d.h. zu jedem festen v, die Ableitungen in u-Richtung übereinstimmen. Diese stimmen genau dann überein, wenn gilt

$$\frac{1}{(o_1-r_1)}(b_{mj}-b_{m-1,j}) = \frac{1}{(o_2-r_2)}(c_{1j}-c_{0j}),\quad j = 0,\cdots,n. \tag{4.68}$$

Dies bedeutet, daß die Polygonsegmente kollinear sein und zudem ein Längenverhältnis $(o_1-r_1):(o_2-r_2)$ besitzen müssen (Bild 4.25).

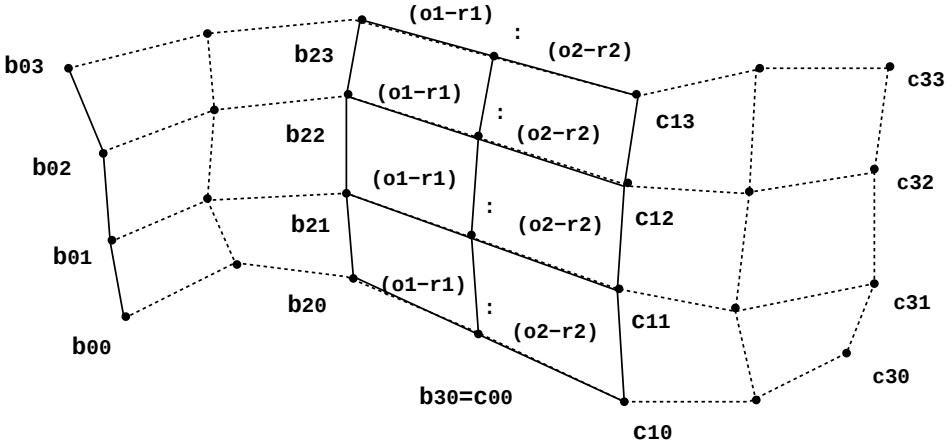


Abbildung 4.25: Kantenfreies (C^1)-stetiges Aneinanderfügen zweier kubischer Bézier-Flächen
Die Kontrollpunkte $b_{3-1,j}, b_{3,j} = c_{0,j}$ und $c_{1,j},\ j = 0,\cdots,3$ müssen auf einer Geraden liegen, und die in der Abbildung angegebenen Verhältnisse müssen gelten.

Fordert man beim Aneinanderfügen nur paarweise kollineare Tangentenvektoren – also gleiche Tangenten –, was der G^1-Stetigkeit entlang der gesamten Nahtlinie entspricht, so müssen die Polygonsegmente zwar kollinear sein, aber in keinem bestimmten Längenverhältnis stehen.

4.7.4.3 B-Spline-Flächen

Seien $m < o$ und $n < p$ sowie

$$S = (s_0 = \cdots = s_m, \cdots, s_{o+1} = \cdots = s_{m+o+1})$$

und

$$T = (t_0 = \cdots = t_n, \cdots, t_{p+1} = \cdots = t_{p+n+1})$$

schwach monotone Folgen von Knoten und seien

$$N_i^m(u),\ i = 0,\cdots,o, \quad \text{bzw.} \quad N_j^n(v),\ j = 0,\cdots,p,$$

B-Splines über S und T. Dann werden analog zu den Bézier-Tensorprodukt-Flächen die Tensorprodukt- B-Spline-Flächen vom Grad (m,n) definiert durch

$$q : [s_0, s_{m+o+1}] \times [t_0, t_{n+p+1}] \to \mathbb{R}^3 :$$

$$q(u,v) \longmapsto \sum_{i=0}^{o} \sum_{j=0}^{p} d_{ij} N_i^m(u) N_j^n(v),\ d_{ij} \in \mathbb{R}^3. \tag{4.69}$$

Die Punkte d_{ij} heißen *de Boor-Punkte* und bilden das *de Boor-Netz*.

Ein Beispiel einer B-Spline-Fläche zeigt Bild 4.26.
Es gelten alle Eigenschaften analog zu den B-Spline-Kurven, insbesondere die konvexe Hülleneigenschaft. Wichtig ist z.B. auch die lokale Wirkung der de Boor-Punkte. So wird z.B. das Flächensegment $u \in [s_i, s_{i+1}]$, $v \in [t_j, t_{j+1}]$ nur von den

Punkten $d_{(i-m)(j-n)}, \cdots, d_{ij}$ beeinflußt, oder umgekehrt hat der de Boor-Punkt d_{ij} nur Einfluß auf die Fläche über dem Parameterbereich $u \in [s_i, s_{i+m+1}]$, $v \in [t_j, t_{j+n+1}]$.
Zur Berechnung von Flächenpunkten $q(u_0, v_0)$ kann wieder der de Boor-Algorithmus eingesetzt werden. Dabei wird der Algorithmus zunächst für $u = u_0$ angewandt, d.h. es werden die de Boor-Punkte

$$d_j(u_0) = \sum_{i=0}^{o} d_{ij} N_i^m(u_0),\ j = 0, \cdots, p,$$

berechnet. Danach wird der de Boor-Algorithmus noch einmal mit $v = v_0$ und den Punkten $d_j(u_0)$ durchlaufen:

$$q(u_0, v_0) = \sum_{j=0}^{p} d_j(u_0) N_j^n(v_0).$$

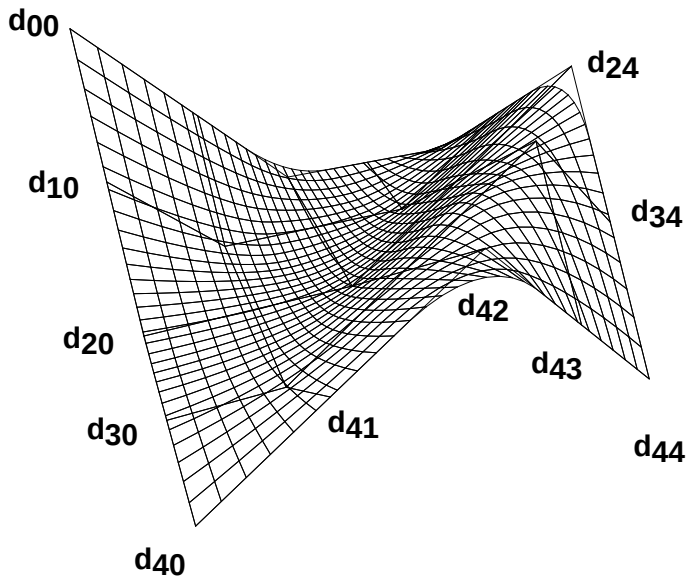


Abbildung 4.26: Eine kubische B-Spline-Tensorprodukt-Fläche und ihr Kontrollnetz mit (5×5) Kontrollpunkten

Wenn Sie die PDF-Version des Kurses bearbeiten, können Sie jetzt das Interaktionselement 'bsplinepatch' anwählen.

4.8 Bézier-Flächen über Dreiecken

Erstmals im rechnerunterstützten Flächenentwurf wurde dieser Flächentyp gegen Ende der 50er Jahre von de Casteljau bei Citroën verwendet [dC63], erlangte jedoch aufgrund der Tatsache, daß die Arbeiten de Casteljau's nicht publiziert wurden, nur geringe Verbreitung in anderen Systemen. Diese Situation änderte sich vor allem mit den grundlegenden Arbeiten von Farin [Far79], [Far80]. Seither haben Bézier-Polynome über Dreiecken innerhalb des Computer Aided Geometric

Design wichtige Bedeutung erlangt [Bar85], [Far86], [dB87]. Eine Zusammenstellung findet man in [Slu89]. Einen einfachen und eleganten Zugang mittels Polarformen findet man in [Sei89].

4.8.1 Verallgemeinerte Bernstein-Polynome

Sind drei affin-unabhängige Punkte $R,S,T \in \mathbb{R}^2$ gegeben und bezeichnen $\rho(U)$, $\sigma(U)$, $\tau(U) \in \mathbb{R}$ die baryzentrischen Koordinaten eines Punktes $U \in \mathbb{R}^2$, d.h.

$$U = \rho(U)\cdot R + \sigma(U)\cdot S + \tau(U)\cdot T \in \mathbb{R}^2, \qquad (4.70)$$
$$\rho(U) + \sigma(U) + \tau(U) = 1,$$

(vgl. Kapitel 3, Abschnitt 3.1.2.1 'Baryzentrische Koordinaten'), dann heißen die Polynome

$${}_{RST}B^n_{i,j,k}(U) := \begin{pmatrix} n \\ i,j,k \end{pmatrix} \rho(U)^i \sigma(U)^j \tau(U)^k, \quad i+j+k=n, \qquad (4.71)$$

Bernstein-Polynome bezüglich des Referenzdreiecks $\triangle(R,S,T)$.

Hierbei bezeichnet $\begin{pmatrix} n \\ i,j,k \end{pmatrix}$ den Koeffizienten

$$\begin{pmatrix} n \\ i,j,k \end{pmatrix} := \frac{n!}{i!\,j!k!}. \qquad (4.72)$$

An den Dreiecksrändern, d.h. für $\rho(U)=0$, $\sigma(U)=0$ oder $\tau(U)=0$ entarten die verallgemeinerten Bernsteinpolynome zu den gewöhnlichen Bernsteinpolynomen. Es gilt z.B. auf der Randkurve zwischen den Punkten R,S des Dreiecks $\Delta(R,S,T)$

$${}_{RST}B^n_{ij0}(\rho,\sigma,0) = B^n_i(\rho) = B^n_j(\sigma).$$

4.8.1.1 Basiseigenschaft der Bernstein-Polynome

Wie die Bernstein-Polynome eine Basis für den Raum der reellwertigen univariaten Polynome (Polynome in einer Variablen) über einem Intervall bilden, so bilden die bivariaten Bernstein-Polynome (Polynome in zwei Variablen)

$${}_{RST}B^n_{i,j,k},\ i+j+k=n,$$

eine Basis für den Raum aller reellwertigen bivariaten Polynome vom Grad n.

4.8.2 Dreiecksflächen in Bernsteinbasis

Gegeben sei ein Referenzdreieck $\triangle(R,S,T)$ und ein bivariates Polynom
$q : \mathbb{R}^2 \to \mathbb{R}^3$.
Die Darstellung

$$q(U) = \sum_{i+j+k=n} {}_{RST}B^n_{i,j,k}(U)\cdot b_{i,j,k}, \qquad b_{i,j,k} \in \mathbb{R}^3, \qquad (4.73)$$

von q in der Bernstein-Basis bezüglich $\triangle(R,S,T)$ mit Koeffizienten $b_{i,j,k} \in \mathbb{R}^3$ heißt *Bézier-Darstellung* von q bezüglich $\triangle(R,S,T)$.
Die Punkte $b_{i,j,k}$ heißen *Kontroll-* oder *Bézier-Punkte* von q. Sie bilden das *Kontroll-* oder *Bézier-Netz* (vgl. Bild 4.27).

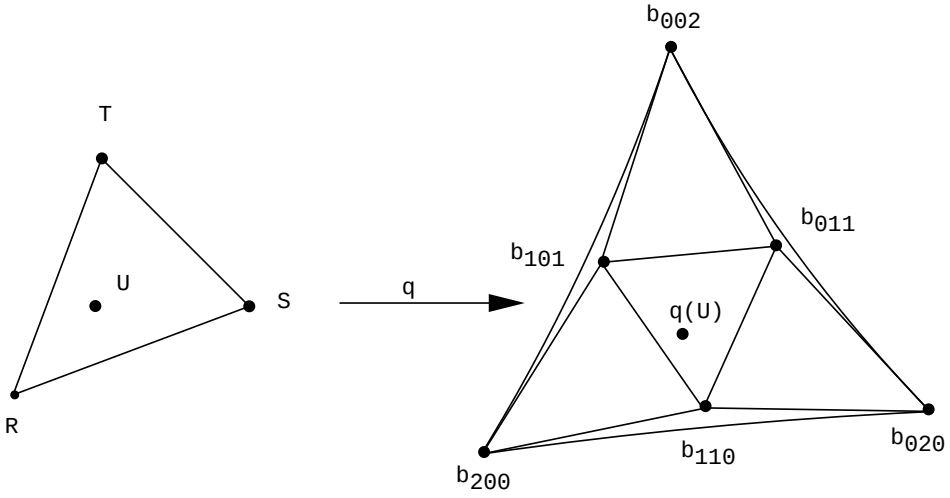


Abbildung 4.27: Referenzdreieck und Dreiecks-Bézier-Fläche vom Grad $n = 2$ mit Bézier-Netz

Analog zum Kurvenfall können mit Hilfe des entsprechenden Algorithmus von Casteljau Funktionswerte von $q(U)$ sowie Ableitungen und Normalenvektoren bestimmt werden. Auch das Unterteilen, das Aneinanderfügen und die Graderhöhung von Bézier-Dreiecksflächen erfolgt mit entsprechenden, in der Notation erheblich komplexeren Algorithmen. Wegen Einzelheiten sei auf [HL89] verwiesen.

4.8.3 Eigenschaften von Bézier-Dreiecksflächen

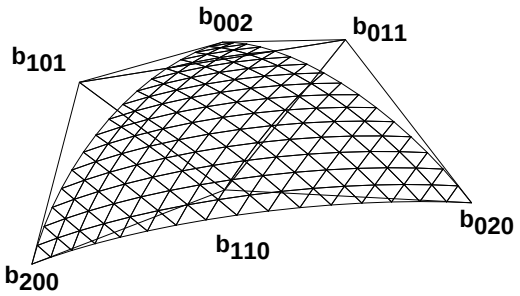


Abbildung 4.28: Eine quadratische Bézier-Dreiecksfläche

Es sei

$$q(U) = \sum_{i+j+k=n} {}_{RST}B^n_{i,j,k}(U) \cdot b_{i,j,k}, \quad b_{i,j,k} \in \mathbb{R}^d,$$

eine quadratische Bézier-Dreiecksfläche bezüglich des Referenzdreiecks $\triangle(R,S,T)$ (vgl. Bild 4.28). Dann hängt die Gestalt der Fläche mit dem Kontrollnetz wie folgt zusammen:

1. Für $U \in \triangle(R,S,T)$ liegt $q(U)$ innerhalb der abgeschlossenen konvexen Hülle des Kontrollpolygons.

2. $q(R) = b_{n,0,0}$, $q(S) = b_{0,n,0}$ und $q(T) = b_{0,0,n}$, d.h. die Fläche verläuft durch die drei Eckpunkte des Bézier-Polygons.

3. Die Fläche ist in den Eckpunkten $q(R)$, $q(S)$ und $q(T)$ tangential an das Kontrollnetz.

4. Die Einschränkung von q auf die Gerade (S,T) ist eine Bézier-Kurve mit den Bézier-Punkten $b_{0,n-k,k}$, d.h. für $U \in (S,T)$ gilt

$$q(U) = \sum_{k=0}^{n} {}^{T}_{S}B^n_k(U) \cdot b_{0,n-k,k}.$$

   Analoges gilt für die Geraden (R,S) und (T,R).

5. Der Zusammenhang zwischen Kurve und Kontrollpolygon ist affin invariant, d.h.

$$\begin{aligned} \Phi(q(U)) &= \Phi\left(\sum_{i+j+k=n} {}_{RST}B^n_{i,j,k}(U) \cdot b_{i,j,k}\right) \\ &= \sum_{i+j+k=n} {}_{RST}B^n_{i,j,k}(U) \cdot \Phi(b_{i,j,k}). \end{aligned}$$

4.9 Coons-Pflaster und Gordon-Flächen - Interpolation von Kurven

Im Gegensatz zu den bisherigen Flächenkonstruktionen wird im folgenden die Konstruktion von Flächen mittels Interpolation von Kurven beschrieben. Die einfachste Form führt auf das Coons-Pflaster. Eine Verallgemeinerung davon stellt die Gordon-Fläche dar. Da bei dieser Art von Interpolation nicht nur endlich viele Punkte interpoliert werden, sondern ganze Kurven, spricht man auch von *transfiniter* Interpolation. Motiviert wird diese Art der Interpolation durch verschiedene Anwendungen. Um z.B. ein 3D-Objekt aus Kunststoff in eine CAD-Datenbank zu übernehmen, wird ein Sensor über vordefinierte Linien des Modells geführt. Entlang dieser Linien werden Punkte gemessen und abgespeichert. Danach werden Kurvenpunkte interpoliert. Aus dem entstehenden Netz von Kurven kann dann die Oberfläche rekonstruiert werden. Dazu werden Coons - und Gordonflächen verwendet.

4.9.1 Coons-Pflaster

Um ein viereckiges Pflaster

$$q(u,v),\ (u,v) \in [0,1] \times [0,1] \subset \mathbb{R}^2,$$

zu konstruieren, dessen vier parametrisierte Randkurven

$$q(u,0), \quad q(u,1),\ u \in [0,1], \quad q(0,v), \quad q(1,v),\ v \in [0,1],$$

gegeben sind, gehen wir schrittweise vor.
Zuerst konstruieren wir zu je zwei sich gegenüberliegenden Randkurven jeweils eine Fläche, die diese Kurven als Randkurven enthält, und kombinieren diese beiden entstehenden Flächen in einem zweiten Schritt geeignet (vgl. Bild 4.29).

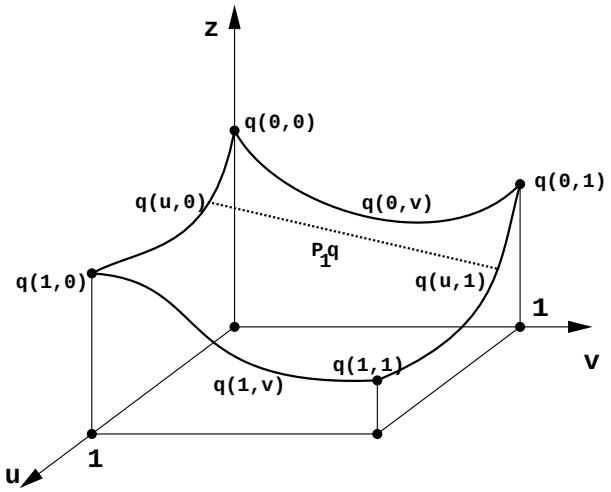


Abbildung 4.29: Konstruktion der Bindefunktionen eines Coons-Pflaster

Das einfachste Verfahren hierzu ist lineare Interpolation.
Wir definieren

$$Q_1(u,v) = (1-v)q(u,0) + vq(u,1), \quad (u,v) \in [0,1] \times [0,1], \tag{4.74}$$
$$Q_2(u,v) = (1-u)q(0,v) + uq(1,v), \quad (u,v) \in [0,1] \times [0,1]. \tag{4.75}$$

Die Fläche Q_1 interpoliert die Randkurven $q(u,0)$ und $q(u,1),\ u \in [0,1]$,
die Fläche Q_2 interpoliert die Randkurven $q(0,v)$ und $q(1,v),\ v \in [0,1]$.

Die Funktionen $u \mapsto (1-u)$ und $u \mapsto u$ bzw. $v \mapsto (1-v)$ und $v \mapsto v$ werden als *Bindefunktionen (blending functions)* bezeichnet.
Das hier beschriebene Verfahren zur Interpolation von zwei sich gegenüberliegenden Kurven durch eine Fläche heißt *lofting*.

Um diese Flächen geeignet zu kombinieren, addieren wir sie zunächst. Da die Summe der beiden Flächen die gegebenen Randkurven nicht mehr interpoliert, müssen wir eine Korrektur vornehmen (vgl. Bild 4.30).

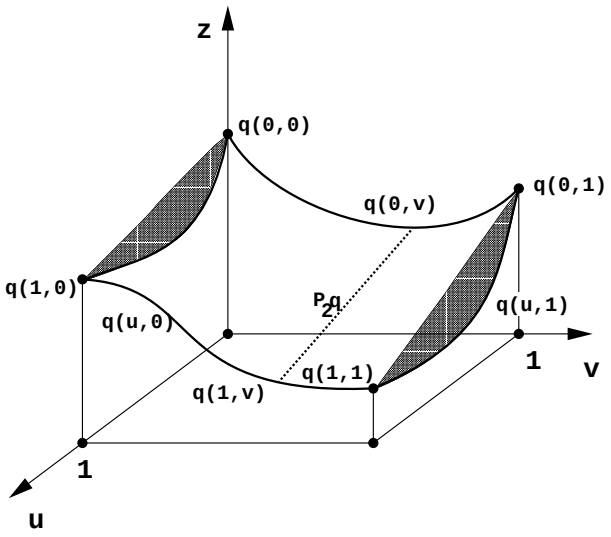


Abbildung 4.30: Konstruktion des Korrekturglieds eines Coons-Pflasters

Wir setzen

$$Q(u,v) \quad = \quad Q_1(u,v) + Q_2(u,v) - ((1-u),u) \begin{pmatrix} q(0,0) & q(0,1) \\ q(1,0) & q(1,1) \end{pmatrix} \begin{pmatrix} 1-v \\ v \end{pmatrix} \tag{4.76}$$

Daß diese Fläche die vorgebenen Randkurven interpoliert, läßt sich einfach nachrechnen, z.B. für die Randkurve $q(u,0),\ u \in [0,1]$:

$$\begin{aligned} Q(u,0) \quad &= \quad Q_1(u,0) + Q_2(u,0) \\ &\qquad -((1-u),u) \begin{pmatrix} q(0,0) & q(0,1) \\ q(1,0) & q(1,1) \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} \\ &= \quad Q_1(u,0) + (1-u)q(0,0) + uq(1,0) \\ &\qquad -((1-u),u) \begin{pmatrix} q(0,0) & q(0,1) \\ q(1,0) & q(1,1) \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} \\ &= \quad Q_1(u,0) = q(u,0). \end{aligned}$$

Das so definierte, alle vier Randkurven interpolierende Pflaster $Q(u,v)$ heißt *Coons-Pflaster*.
Das Korrekturglied

$$((1-u),u) \begin{pmatrix} q(0,0) & q(0,1) \\ q(1,0) & q(1,1) \end{pmatrix} \begin{pmatrix} 1-v \\ v \end{pmatrix}$$

ist eine Tensorprodukt-Fläche.

Wie man leicht an den Formeln 4.74, 4.75, 4.76 sieht, lassen sich anstelle linearer Bindefunktionen auch andere Paare von Bindefunktionen $f_1(u), f_2(u)$ und

$g_1(v), g_2(v)$ verwenden, z.B. kubische Hermite-Polynome. Diese müssen am Rand ihres Definitionsintervalls $[0,1]$ folgende Bedingungen erfüllen:

$$\begin{array}{rclcrcl} f_1(0) & = & 1 & \text{und} & f_1(1) & = & 0 \\ f_2(0) & = & 0 & \text{und} & f_2(1) & = & 1 \\ g_1(0) & = & 1 & \text{und} & g_1(1) & = & 0 \\ g_2(0) & = & 0 & \text{und} & g_2(1) & = & 1. \end{array} \tag{4.77}$$

Man erhält dann ein *verallgemeinertes Coons-Pflaster*.
Bezeichnet man die Matrix der Eckpunkte

$$\begin{pmatrix} q(0,0) & q(0,1) \\ q(1,0) & q(1,1) \end{pmatrix}$$

mit A und wählt sonst dieselben Bezeichnungen wie oben, so läßt sich das verallgemeinerte Coons-Pflaster Q schreiben als

$$Q = Q_1 + Q_2 - (f_1, f_2) A \begin{pmatrix} g_1 \\ g_2 \end{pmatrix}. \tag{4.78}$$

Durch die Wahl der Funktionen f_1, f_2, g_1, g_2 ist die Interpolationsfläche festgelegt. Sie besitzt keine weiteren freien Parameter. Das ist ein gewisser Nachteil der Coons-Darstellung.

Wenn Sie die PDF-Version des Kurses bearbeiten, können Sie jetzt das Interaktionselement 'coons' anwählen.

4.9.2 Gordon-Pflaster

Die Ausdehnung der Konstruktion von Coons auf Systeme von Flächen führt auf die sogenannten *Gordon-Flächen* [Gor69], [Gor71]:
Ist ein Netz von parametrisierten Kurven

$$q(u_i, v),\ i = 0, \cdots, m \quad \text{und} \quad q(u, v_j),\ j = 0, \cdots, n$$

gegeben (vgl. Bild 4.31), so interpoliert eine Gordon-Fläche dieses Netz mit Hilfe eines Systems von Bindefunktionen

$$g_i(u),\ i = 0, \cdots, m \quad \text{und} \quad h_j(v),\ j = 0, \cdots, n.$$

Die Konstruktion von Gordon-Flächen verläuft analog zur Konstruktion von Coons-Pflastern. Zunächst konstruiert man zwei Lofting-Flächen wie in 4.74 und 4.75. Da bei Gordon-Flächen in u und v-Richtung mehr als zwei Werte interpoliert werden müssen, verwendet man zur Interpolation Polynome entsprechenden Grades. Man kann die Bindefunktionen z.B. mit Hilfe der Lagrange-Interpolation (vgl. Kapitel 4.2.3 'Interpolation mit Lagrange-Polynomen') wie folgt ansetzen:

$$g_1(u, v) = \sum_{i=0}^{m} q(u_i, v) L_i^m(u), \tag{4.79}$$

$$g_2(u, v) = \sum_{j=0}^{n} q(u, v_j) L_j^n(v). \tag{4.80}$$

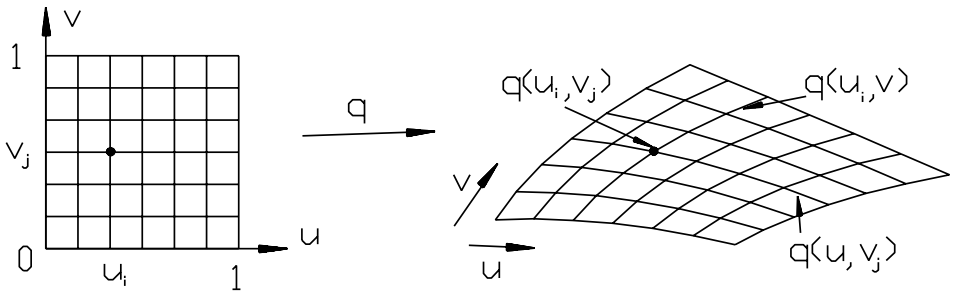


Abbildung 4.31: Beispiel eines Netzes von Kurven
Mit Hilfe einer Gordon-Fläche können diese Kurven interpoliert werden.

Das Korrekturglied ergibt sich wie bei den Coons-Pflastern als Tensorprodukt:

$$g_{12}(u,v) = \sum_{i=0}^{m} \sum_{j=0}^{n} q(u_i, v_j) L_i^m(u) L_j^n(v). \tag{4.81}$$

Mit diesen Flächen kann man die Gordon-Fläche wie folgt schreiben:

$$R(u,v) = g_1(u,v) + g_2(u,v) - g_{12}(u,v). \tag{4.82}$$

Da dieselbe Konstruktionsweise wie für Coons-Pflaster verwendet wurde, ist jedes Pflasterstück

$$R(u,v) \quad \text{mit} \quad (u,v) \in [u_i, u_{i+1}] \times [v_j, v_{j+1}],\ i = 0, \cdots, m,\ j = 0, \cdots, n,$$

selbst ein Coons-Pflaster.

Anstatt die Bindefunktionen durch Polynom-Interpolation zu berechnen, könnte man auch andere Interpolanten wählen, z.B. Spline-Interpolanten. Man spricht dann von *spline-blended* Gordon-Flächen. Gordon-Flächen lassen sich auch auf dreieckige Parametergebiete ausdehnen, worauf wir hier aber nicht eingehen. Nähere Informationen findet man in dem Buch von Hoschek und Lasser [HL89].

4.10 Übungsaufgaben

Aufgabe 1:

Gegeben sei das reelle Polynom

$$p(u) = 7u^3 + 6u^2 - 3u + 5.$$

a) Schreiben Sie den Graphen G(u) = $(u, p(u))$ über dem Intervall [0,1] als kubische Bézier-Kurve. Skizzieren Sie das Kontrollpolygon des Graphen.

b) Zerlegen Sie den Graphen G(u) = $(u, p(u))$ an der Stelle $u = \frac{1}{2}$ in zwei Teilsegmente und geben Sie die Bézierpunkte der beiden Teilsegmente an. Zeichnen Sie eine Skizze und tragen Sie die Bézierpunkte der Teilsegmente ein. (Hinweis: Verwenden Sie den Algorithmus von de Casteljau.)

c) Schreiben Sie den Graphen G(u) = $(u, p(u))$ als Bézierkurve vom Grad 4 über dem Intervall [0, 1]. Geben Sie die dazugehörigen Bézierpunkte an und machen Sie eine Skizze.

Aufgabe 2:

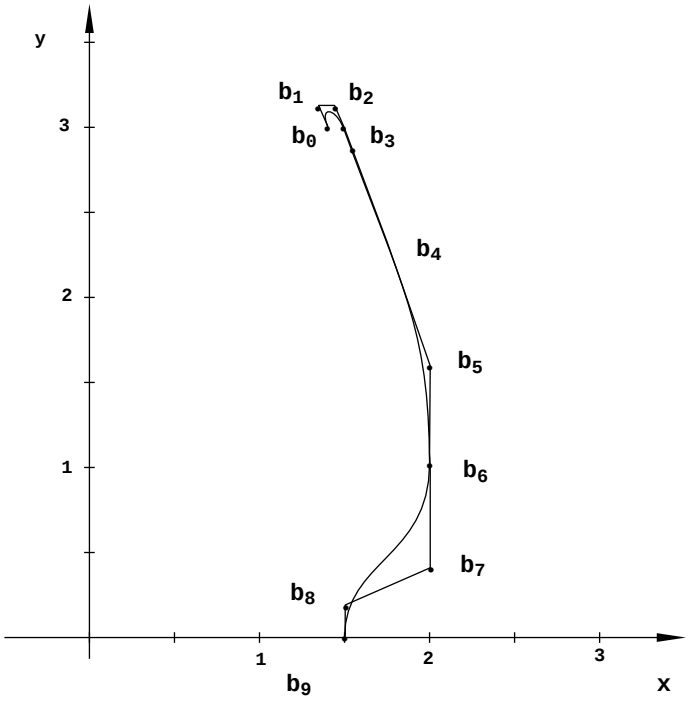


Abbildung 4.32: Ebene Bézier-Spline-Kurve vom Grade drei mit drei Segmenten

Gegeben ist eine ebene Bézier-Spline-Kurve vom Grad 3 mit drei Segmenten je-

weils über dem Intervall $[0,1]$ und den Kontrollpunkten

$$\begin{aligned} b_0 &= (1.4,3) \\ b_1 &= (1.3375,3.1750) \\ b_2 &= (1.4375,3.1750) \\ b_3 &= (1.59375,2.7375) \\ b_4 &= (1.75,2.3) \\ b_5 &= (2.0,1.6) \\ b_6 &= (2.0,1.0) \\ b_7 &= (2.0,0.4) \\ b_8 &= (1.5,0.2) \\ b_9 &= (1.5,0.0) \end{aligned}$$

(vgl. Abb. 4.32.)

a) Geben Sie die drei Bézier-Kurven explizit an und zeigen Sie, daß die Spline-Kurve an den Übergängen C^1–stetig ist. Ist die Spline-Kurve C^2–stetig?

b) Schreiben Sie die Bézier–Spline-Kurve als kubische B–Spline-Kurve mit denselben Kontrollpunkten. Hinweis: Wählen Sie einen geeigneten Knotenvektor.

c) Da die Splinekurve an den Übergängen C^1– stetig ist (vgl. a)), kann die Spline-Kurve als kubische B–Spline-Kurve mit nur 8 Kontrollpunkten geschrieben werden. Bestimmen Sie einen geeigneten Knotenvektor und geben Sie die 8 Kontrollpunkte der B–Spline-Kurve an.
Hinweis: Knoten einfügen.

d) Welche der 8 Kontrollpunkte aus c) beeinflussen das erste, zweite und dritte Kurvensegment?

Aufgabe 3:

Gegeben sei ein Tensorprodukt-Bézier–Pflaster

$$q(u,v) = \sum_{i=0}^{m} \sum_{j=0}^{n} B_i^m(u) B_j^n(v) \cdot b_{ij}$$

über [0, 1]×[0, 1] mit n ≥ m. Betrachten Sie folgendes Rekursionsschema (Algorithmus von de Casteljau für Tensorprodukt-Flächen):

$$b_{ij}^{00} := b_{ij} \quad i = 0,\cdots,m,\ j = 0,\cdots,n,$$

$$b_{ij}^{r_1+1,r_2} := (1-u)b_{ij}^{r_1 r_2} + u b_{i+1,j}^{r_1 r_2}, \quad i = 0,\cdots,m-r_1-1,\ j = 0,\cdots,n-r_2,$$

$$b_{ij}^{r_1,r_2+1} := (1-v)b_{ij}^{r_1 r_2} + v b_{i,j+1}^{r_1 r_2}, \quad i = 0,\cdots,m-r_1,\ j = 0,\cdots,n-r_2-1,$$

mit $r_1 = 1,\cdots,m,\ r_2 = 1,\cdots n.$

a) Zeigen Sie, daß Rekursionsschritte in $u-$ und $v-$Richtung vertauschbar sind, d.h. daß b_{00}^{mn} nicht von der Reihenfolge der Rekursionschritte in $u-$ und $v-$Richtung abhängt.

b) Veranschaulichen Sie graphisch jeweils einen Rekursionsschritt in $u-$ und $v-$Richtung anhand eines einfachen Kontrollpolygons für den Fall $n = m = 2$. Wählen Sie $u = v = \frac{1}{2}$.

c) Zeigen Sie, daß $q(u,v) = b_{00}^{mn}$ gilt, d.h. mittels obiger Rekursion kann der Funktionswert $q(u,v)$ an der Stelle (u,v) berechnet werden.
Hinweis:
Führen Sie zunächst für $i = 0, \cdots, m$ jeweils n Rekursionsschritte in $v-$Richtung durch und stellen Sie das Ergebnis als Bézier-Kurve in v dar (Algorithmus von de Casteljau im Kurvenfall, siehe Abschnitt 4.3.2 'Der Algorithmus von de Casteljau').

d) Nach a) sind Rekursionschritte in $u-$ und $v-$Richtung vertauschbar. Die Punkte $b_{ij}^{r_1 r_2}$ lassen sich für festes r_1 und r_2 in einem rechteckigen Schema anordnen, z.B.

$$\begin{array}{lll} b_{00} & b_{01} & b_{02} \\ b_{10} & b_{11} & b_{12} \\ b_{20} & b_{21} & b_{22} \end{array} .$$

Tragen Sie für $n = m = 2$ die Punkte

$$b_{ij}^{r_1 r_2},\ 0 \le i \le m - r_1,\ 0 \le j \le n - r_2,\ 0 \le r_1 \le m,\ 0 \le r_2 \le n,$$

der obigen Rekursion in solche Schemata ein und ordnen Sie diese auf solche Weise an, wie sie mittels der Rekursion aus ihren Vorgängern berechnet werden.

e) Vergleichen Sie im Fall $m \le n$ den Aufwand zur Berechnung eines Funktionswertes $q(u,v)$ für folgende beiden Fälle:

I. Es werden für $i = 0, \cdots, m$ zunächst n Rekursionsschritte in $v-$Richtung und danach m Rekursionsschritte in $u-$Richtung durchgeführt.

II. Es werden für $j = 0, \cdots, n$ zunächst m Rekursionsschritte in $u-$Richtung und danach n Rekursionsschritte in $v-$Richtung durchgeführt.

f) Im Fall $n = m$ können Rekursionsschritte in $u-$Richtung und in $v-$Richtung abwechselnd ausgeführt werden. Wieviele Affinkombinationen sind in diesem Fall zur Berechnung von b_{00}^{mn} notwendig? Vergleichen Sie den Rechenaufwand in diesem Fall mit dem Rechenaufwand des Verfahrens aus Teilaufgabe e).

Aufgabe 4:

a) *Coonspflaster*
Gegeben seien die folgenden parametrisierten Randkurven mit $u \in [0,1]$ und $v \in [0,1]$:

$$\begin{aligned} q(u,0) &= u^2+1, \\ q(u,1) &= u^2-2u+2, \\ q(1,v) &= v^2-2v+2, \\ q(0,v) &= v^2+1. \end{aligned}$$

Berechnen Sie das zugehörige Coonspflaster.

b) *Verallgemeinerte Coonspflaster*
Bei der Berechnung von Coonspflastern lassen sich auch nichtlineare Bindefunktionen f_1, f_2, g_1, g_2 verwenden, z.B. kubische Hermite-Polynome.
Dabei müssen sie am Rand ihres Definitionsbereichs $[0,1]$ die folgenden Bedingungen erfüllen:

$$\begin{aligned} f_1(0) &= 1, & f_1(1) &= 0, \\ f_2(0) &= 0, & f_2(1) &= 1, \\ g_1(0) &= 1, & g_1(1) &= 0, \\ g_2(0) &= 0, & g_2(1) &= 1. \end{aligned}$$

1.) Wählen Sie kubische Hermite-Polynome für f_1, f_2 und g_1, g_2 aus, die diese Bedingungen erfüllen.

2.) Berechnen Sie mit diesen Hermite-Polynomen als Bindefunktionen das verallgemeinerte Coonspflaster zu den Randkurven

$$\begin{aligned} q(u,0) &= u^3, \\ q(u,1) &= u^3+1, \\ q(0,v) &= v^3, \\ q(1,v) &= v^3+1. \end{aligned}$$

4.11 Lösungen zu den Übungsaufgaben

Lösung 1:

a) Die Kontrollpunkte b_i des Kontrollpolygons haben die Gestalt

$$\begin{pmatrix} 0 \\ b_{02} \end{pmatrix} \quad \begin{pmatrix} \frac{1}{3} \\ b_{12} \end{pmatrix} \quad \begin{pmatrix} \frac{2}{3} \\ b_{22} \end{pmatrix} \quad \begin{pmatrix} 1 \\ b_{32} \end{pmatrix},$$

wenn wir das Intervall $[0,1]$ zugrundelegen.
Die Bernsteinpolynome vom Grade 3 sind über dem Intervall $[0,1]$ gegeben durch:

$$\begin{aligned} B_0^3(u) &= (1-u)^3 &&= 1-3u+3u^2-u^3 \\ B_1^3(u) &= 3u(1-u)^2 &&= 3u-6u^2+3u^3 \\ B_2^3(u) &= 3u^2(1-u) &&= 3u^2-3u^3 \\ B_3^3(u) &= u^3 \end{aligned}$$

Daher lassen sich die b_{i2} aus der folgenden Gleichung berechnen:

$$\begin{aligned} & b_{02}B_0^3(u)+b_{12}B_1^3(u)+b_{22}B_2^3(u)+b_{32}B_3^3(u) \\ = \; & b_{02}(1-3u+3u^2-u^3)+b_{12}(3u-6u^2+3u^3)+b_{22}(3u^2-3u^3)+b_{32}u^3 \\ = \; & 7u^3+6u^2-3u+5 \end{aligned}$$

Koeffizientenvergleich liefert die Gleichungen

$$\begin{aligned} b_{02} &= 5 \\ -3b_{02}+3b_{12} &= -3 \\ 3b_{02}-6b_{12}+3b_{22} &= 6 \\ -b_{02}+3b_{12}-3b_{22}+b_{32} &= 7 \end{aligned}$$

und damit

$$\begin{aligned} b_{02} &= 5 \\ b_{12} &= 4 \\ b_{22} &= 5 \\ b_{32} &= 15 \end{aligned}$$

Für den Graphen ergeben sich somit die Bézier-Punkte

$$\begin{pmatrix} 0 \\ 5 \end{pmatrix} \quad \begin{pmatrix} \frac{1}{3} \\ 4 \end{pmatrix} \quad \begin{pmatrix} \frac{2}{3} \\ 5 \end{pmatrix} \quad \begin{pmatrix} 1 \\ 15 \end{pmatrix},$$

(vgl. Abb. 4.33 a)).

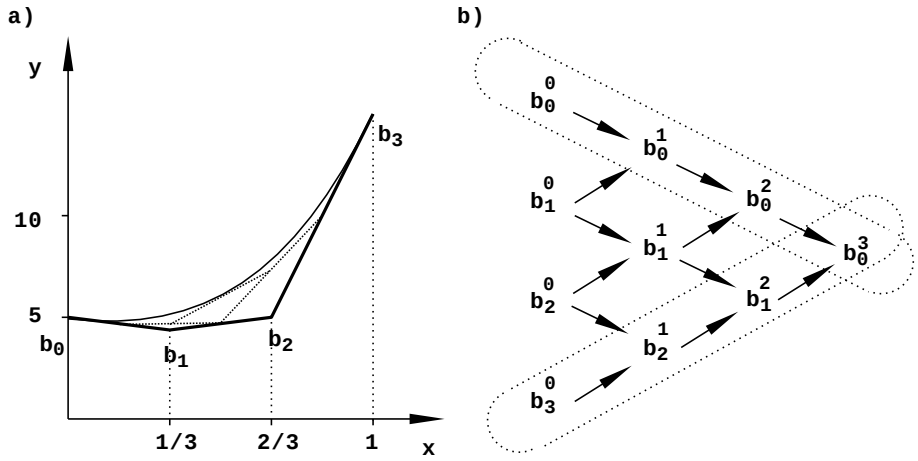


Abbildung 4.33: a) Bézierpunkte, b) de Casteljau-Schema

b) Aus dem de Casteljau-Schema für $u = 1/2$ lassen sich die Bézierpunkte der Teilsegmente p_1, p_2 ablesen (vgl. Abb. 4.33 b)). Man erhält

$$\begin{aligned} p_1(u) &= \sum_{i=0}^{3} b_0^i B_i^3(u), \\ p_2(u) &= \sum_{i=0}^{3} b_i^{3-i} B_i^3(u), \end{aligned}$$

mit

$$\begin{aligned} b_0^0 &= (0,5) \\ b_0^1 &= (1/6, 9/2) \\ b_0^2 &= (1/3, 9/2) \\ b_0^3 &= (1/2, 47/8) \\ b_0^3 &= (1/2, 47/8) \\ b_1^2 &= (2/3, 29/4) \\ b_2^1 &= (5/6, 10) \\ b_3^0 &= (1, 15) \end{aligned}$$

(vgl. Abb. 4.34 a)).

c) Um den Graph als Bézier-Kurve vom Grad 4 zu schreiben, führen wir eine Graderhöhung durch (vgl. Formeln 4.24 - 4.26 und Abb. 4.34 b)):

$$\begin{aligned} b_0^* &= b_0 = (0,5) \\ b_1^* &= 1/4 \cdot b_0 + 3/4 \cdot b_1 = (1/4, 17/4) \\ b_2^* &= 1/2 \cdot b_1 + 1/2 \cdot b_2 = (1/2, 9/2) \\ b_3^* &= 3/4 \cdot b_2 + 1/4 \cdot b_3 = (3/4, 30/4) \\ b_4^* &= b_3 = (1, 15). \end{aligned}$$

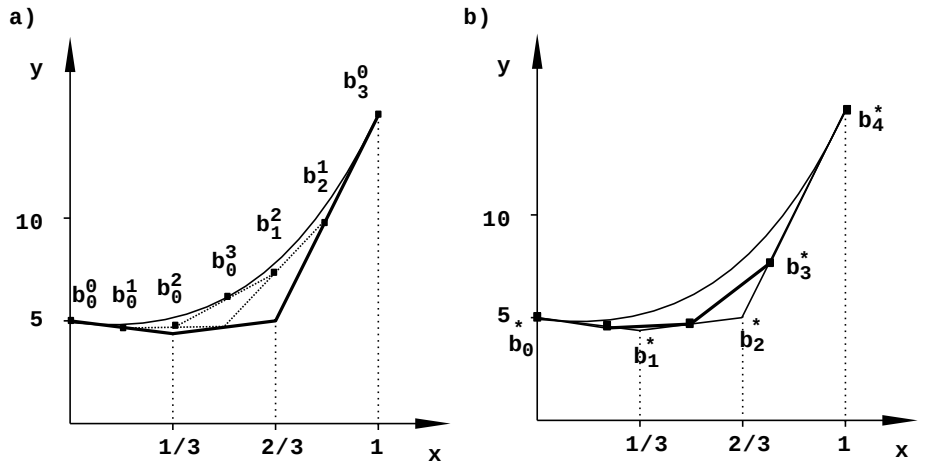


Abbildung 4.34: a) Bézierpunkte der Teilsegmente, b) Graderhöhung

Lösung 2:

a) Die drei Bezierkurven sind gegeben durch

$$\begin{aligned} q_1(u) &= b_0B_0^3(u) + b_1B_1^3(u) + b_2B_2^3(u) + b_3B_3^3(u), \\ q_2(u) &= b_3B_0^3(u) + b_4B_1^3(u) + b_5B_2^3(u) + b_6B_3^3(u), \\ q_3(u) &= b_6B_0^3(u) + b_7B_1^3(u) + b_8B_2^3(u) + b_9B_3^3(u). \end{aligned}$$

Für C^1–stetige Übergänge muß gelten (vgl. Formel 4.27):

$$q_1(1) = q_2(0) \text{ und } q_1'(1) = q_2'(0)$$

sowie

$$q_2(1) = q_3(0) \text{ und } q_2'(1) = q_3'(0).$$

Die erste Bedingung ist in beiden Fällen offensichtlich erfüllt.
Die zweite Bedingung wird jeweils nachgerechnet:

$$\begin{aligned} q_1'(1) &= 3(b_3 - b_2) = 3 \cdot (0.15625, -0.4375), \\ q_2'(0) &= 3(b_4 - b_3) = 3 \cdot (0.15625, -0.4375) \end{aligned}$$

und

$$\begin{aligned} q_2'(1) &= 3(b_6 - b_5) = 3 \cdot (0, -0.6), \\ q_3'(0) &= 3(b_7 - b_6) = 3 \cdot (0, -0.6). \end{aligned}$$

Also ist die Spline-Kurve an den Übergängen C^1-stetig.

Für die C^2– Stetigkeit der Spline-Kurve müßte gelten:

$$b_2 + (b_2 - b_1) = b_4 + (b_4 - b_5),$$

und

$$b_5 + (b_5 - b_4) = b_7 + (b_7 - b_8).$$

Einsetzen und Nachrechnen zeigt, daß diese Bedingungen nicht erfüllt sind, d.h. die Splinekurve ist nicht überall C^2– stetig.

b) Werden alle inneren Knoten mit Vielfachheit 3 gezählt, so ist die B–Spline-Darstellung einer Spline-Kurve und die Darstellung der Spline-Kurve als stückweise Bezier-Kurve identisch. Wir wählen daher den Knotenvektor

$$T = (0,0,0,0,1,1,1,2,2,2,3,3,3,3).$$

Die Spline-Kurve hat über diesem Knotenvektor die Darstellung

$$q(u) = \sum_{i=0}^{9} b_i N_i^3(u).$$

c) Da die Spline-Kurve an den Segmentübergängen stetig ist, genügt an den Übergangsstellen eine doppelte Vielfachheit des Knotens. Man kann daher den Knotenvektor

$$T = (0,0,0,0,1,1,2,2,3,3,3,3)$$

wählen. Die Spline-Kurve hat bezüglich dieses Knotenvektors die Darstellung

$$q_(u) = \sum_{i=0}^{7} d_i N_i^3(u).$$

Werden in diesen Knotenvektor die beiden Knoten 1 und 2 geeignet eingefügt, so erhält man den Knotenvektor

$$T = (0,0,0,0,1,1,1,2,2,2,3,3,3,3)$$

aus Aufgabenteil b) und die Darstellung

$$q(u) = \sum_{i=0}^{9} b_i N_i^3(u)$$

und daraus Bedingungen für die Kontrollpunkte d_i. Einfügen des Knotens $t_5 \le t = 1 \le t_6$ gemäß den Formeln 4.39, 4.40

$$d_i^* = (1-a_i)\cdot d_{i-1} + a_i \cdot d_i$$

mit

$$a_i = \begin{cases} 1 & \text{für } i \le 5-3 \\ \dfrac{t-t_i}{t_{i+n}-t_i} & \text{für } 5-3+1 \le i \le 5 \\ 0 & \text{für } 5+1 \le i \end{cases}$$

liefert:

$$\begin{array}{lllll} d_0^* &=& d_0, & d_1^* &= d_1,\ d_2^* = d_2 \\ d_3^* &=& \dfrac{d_2+d_3}{2}, & d_4^* &= d_3,\ d_5^* = d_4 \\ d_6^* &=& d_5, & d_7^* &= d_6,\ d_8^* = d_7 \end{array}$$

mit dem Knotenvektor

$$T = (0,0,0,0,1,1,1,2,2,3,3,3,3).$$

Einfügen des Knotens $t_8 \leq t = 2 \leq t_9$ in diesen Knotenvektor gemäß den Formeln 4.39, 4.40

$$d_i^{**} = b_i = (1 - a_i) \cdot d_{i-1}^{*} + a_i \cdot d_i^{*}$$

mit

$$a_i = \begin{cases} 1 & \text{für } i \leq 8-3 \\ \dfrac{t - t_i}{t_{i+n} - t_i} & \text{für } 8-3+1 \leq i \leq 8 \\ 0 & \text{für } 8+1 \leq i \end{cases}$$

liefert:

$$b_0 = d_0^{*} = d_0 \qquad b_1 = d_1^{*} = d_1, \qquad b_2 = d_2^{*} = d_2$$

$$b_3 = d_3^{*} = \underbrace{\frac{d_2 + d_3}{2}}_{C^1\text{--Bed.}}, \qquad b_4 = d_4^{*} = d_3, \qquad b_5 = d_5^{*} = d_4$$

$$b_6 = d_6^{*} = \underbrace{\frac{d_5^{*} + d_6^{*}}{2}}_{C^1\text{--Bed.}} = \frac{d_4 + d_5}{2}, \qquad b_7 = d_6^{*} = d_5$$

$$b_8 = d_7^{*} = d_6, \qquad b_9 = d_8^{*} = d_7.$$

Daraus erhalten wir

$d_0 = b_0$, $d_1 = b_1$, $d_2 = b_2$, $d_3 = b_4$, $d_4 = b_5$, $d_5 = b_7$, $d_6 = b_8$ und $d_7 = b_9$.

d) Das erste Kurvensegment wird von den Kontrollpunkten d_0, d_1, d_2, d_3, das zweite von den Kontrollpunkten d_3, d_4, d_5, d_6 und das dritte von den Kontrollpunkten d_4, d_5, d_6, d_7 beeinflußt.

Lösung 3:

a) Für die Rekursionsschritte in $u-$ bzw. $v-$Richtung gilt:

$$b_{ij}^{r_1+1,r_2} = (1-u)b_{ij}^{r_1r_2} + ub_{i+1,j}^{r_1r_2}$$

$$b_{ij}^{r_1,r_2+1} = (1-v)b_{ij}^{r_1r_2} + vb_{i,j+1}^{r_1r_2}$$

Daraus erhalten wir

$$\begin{aligned} b_{ij}^{r_1+1,r_2+1} &= (1-u)b_{ij}^{r_1,r_2+1} + ub_{i+1,j}^{r_1,r_2+1} \\ &= (1-u)((1-v)b_{ij}^{r_1r_2} + vb_{i,j+1}^{r_1r_2}) + u((1-v)b_{i+1,j}^{r_1r_2} + vb_{i+1,j+1}^{r_1r_2}) \\ &= (1-v)((1-u)b_{ij}^{r_1r_2} + ub_{i+1,j}^{r_1r_2}) + v((1-u)b_{i,j+1}^{r_1r_2} + ub_{i+1,j+1}^{r_1r_2}) \\ &= (1-v)b_{ij}^{r_1+1,r_2} + vb_{i,j+1}^{r_1+1,r_2}, \end{aligned}$$

d.h. die Rekursionsschritte in $u-$ und $v-$Richtung sind vertauschbar.

b) Die graphische Veranschaulichung ist in Bild 4.35 dargestellt.

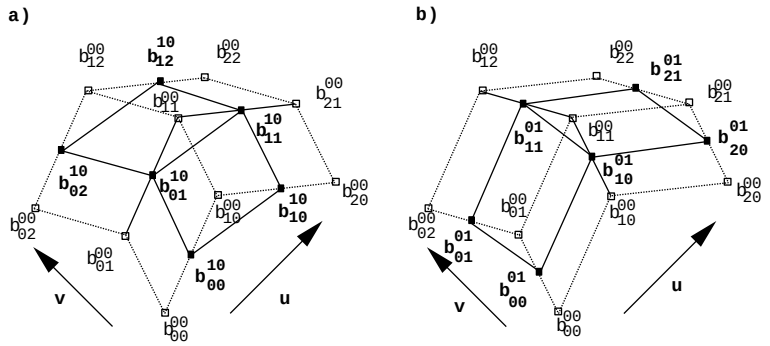


Abbildung 4.35: In a) ist ein Rekursionsschritt in $u-$Richtung und in b) ein Rekursionsschritt in $v-$ Richtung dargestellt.

c) Da nach a) Rekursionsschritte in $u-$ und $v-$Richtung vertauscht werden können, kann man zunächst für $i = 0, \cdots, m$ jeweils n Rekursionsschritte in $v-$Richtung durchführen. So erhält man für $i = 0, \cdots, m$ gemäß dem Algorithmus von de Casteljau für Polynome in einer Variablen

$$b_{i0}^{0n} = \sum_{j=0}^{n} b_{ij} B_j^n(v).$$

Danach werden m Rekursionsschritte in $u-$Richtung durchgeführt und man erhält wieder gemäß dem Algorithmus von de Casteljau für Polynome in einer Variablen

$$b_{00}^{mn} = \sum_{i=0}^{m} b_{i0}^{0n} B_i^m(u) = \sum_{i=0}^{m}\sum_{j=0}^{n} b_{ij} B_j^n(v) B_i^m(u) = q(u,v).$$

d) Das Ergebnis ist in Bild 4.36 dargestellt.

e) I. Es müssen $(m+1)$ mal jeweils $\frac{n\cdot(n+1)}{2}$ Affinkombinationen zur Berechnung von b_{i0}^{0n} durchgeführt werden.

Danach sind noch $\frac{m\cdot(m+1)}{2}$ Affinkombinationen zur Berechnung von b_{00}^{mn} erforderlich. Insgesamt sind daher

$$A = (m+1)\cdot\frac{m+n(n+1)}{2}$$

Affinkombinationen auszuführen.

II. Analog erhält man, daß in diesem Fall

$$B = (n+1)\cdot\frac{n+m(m+1)}{2}$$

Affinkombinationen auszuführen sind.

Daraus erhalten wir

$$A - B = \frac{nm(n-m)}{2},$$

d.h. für $n > m$ ist Verfahren II. günstiger.

f) Werden Rekursionsschritte in $u-$ und $v-$Richtung abwechselnd ausgeführt, so werden in je zwei Schritten aus $i*i$ Kontrollpunkten $(i-1)*(i-1)$ Kontrollpunkte berechnet. Es sind dazu $n = m$ Doppelschritte (ein Rekursionsschritt in $u-$Richtung und ein Rekursionsschritt in $v-$Richtung) notwendig. In jedem Doppelschritt müssen

$$i*(i-1)+(i-1)*(i-1) = (i-1)*(2i-1),$$

und daher insgesamt

$$C = \sum_{i=2}^{n+1}(i-1)(2i-1) = \frac{5}{6}n+\frac{3}{2}n^2+\frac{2}{3}n^3$$

Affinkombinationen berechnet werden.
Für $n = m$ ergibt sich in e) ein Aufwand von

$$A = (n+1)\cdot\frac{n+n(n+1)}{2} = \frac{n\cdot(n+1)\cdot(n+2)}{2}$$

Affinkombinationen und daraus

$$C - A = \frac{1}{6}(n^3-n),$$

d.h. die Methode aus Aufgabe e) ist für $n > 1$ günstiger.
Vergleichen Sie hierzu auch das Schema aus Aufgabenteil d). Dort ergibt sich

$$C = 6+4+2+1 = 13, \quad A = 6+3+2+1 = 12 \quad \text{und} \quad C-A = 1.$$

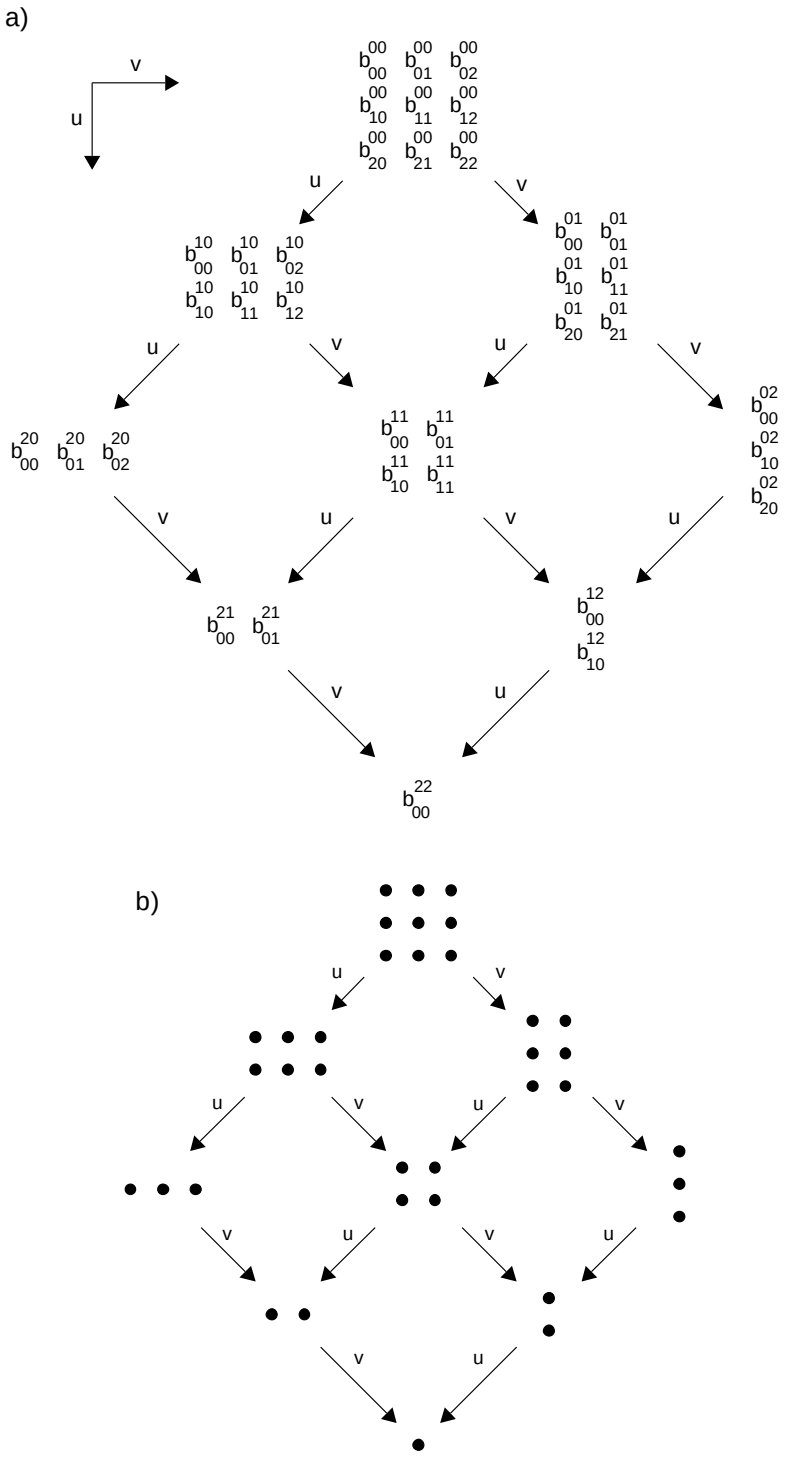


Abbildung 4.36: a) zeigt die Zwischenergebnisse der einzelnen Rekursionsschritte für $n = m = 2$ mit Kontrollpunkten. Eine komprimiertere Darstellung der Rekursion gibt b). Die einzelnen Quadrate repräsentieren die Kontrollpunkte $b_{ij}^{r_1 r_2}$ der Rekursion.

Lösung 4:

a) Das Coonspflaster Q hat die folgende Gestalt:

$$Q(u,v) = Q_1(u,v) + Q_2(u,v) - (1-u,u) \begin{pmatrix} q(0.0) & q(0,1) \\ q(1,0) & q(1,1) \end{pmatrix} \begin{pmatrix} 1-v \\ v \end{pmatrix}$$

mit

$$\begin{aligned} Q_1(u,v) &= (1-v)q(u,0) + vq(u,1) \\ &= (1-v)(u^2+1) + v(u^2-2u+2) \\ &= u^2 - 2uv + v + 1 \end{aligned}$$

und

$$\begin{aligned} Q_2(u,v) &= (1-u)q(0,v) + uq(1,v) \\ &= (1-u)(v^2+1) + u(v^2-2v+2) \\ &= v^2 - 2uv + u + 1 \end{aligned}$$

sowie

$$\begin{pmatrix} q(0,0) & q(0,1) \\ q(1,0) & q(1,1) \end{pmatrix} = \begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix}.$$

Wir erhalten also:

$$\begin{aligned} Q(u,v) &= (u^2 - 2uv + v + 1) + (v^2 - 2uv + u + 1) - (1-u,u) \begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix} \begin{pmatrix} 1-v \\ v \end{pmatrix} \\ &= u^2 + v^2 - 2uv + 1. \end{aligned}$$

b) 1.) Wir wählen:

$$\begin{aligned} f_1(u) &:= H_0^3(u) = (1-u)^2(1+2u) = 2u^3 - 3u^2 + 1 \\ f_2(u) &:= H_3^3(u) = (3-2u)u^2 = -2u^3 + 3u^2 \\ g_1(v) &:= H_0^3(v) = (1-v)^2(1+2v) = 2v^3 - 3v^2 + 1 \\ g_2(v) &:= H_3^3(v) = (3-2v)v^2 = -2v^3 + 3v^2 \end{aligned}$$

(Man vergleiche hierzu Formel 4.15 im Abschnitt 4.2.4 'Interpolation mit kubischen Polynomen'.)

2.) Das verallgemeinerte Coonspflaster Q hat die folgende Gestalt:

$$Q(u,v) = Q_1(u,v) + Q_2(u,v) - (f_1(u), f_2(u)) \begin{pmatrix} q(0,0) & q(0,1) \\ q(1,0) & q(1,1) \end{pmatrix} \begin{pmatrix} g_1(v) \\ g_2(v) \end{pmatrix}$$

mit

$$\begin{aligned} Q_1(u,v) &= g_1(v)q(u,0) + g_2(v)q(u,1) \\ &= (2v^3 - 3v^2 + 1)u^3 + (-2v^3 + 3v^2)(u^3 + 1) \\ &= u^3 - 2v^3 + 3v^2 \end{aligned}$$

und

$$\begin{aligned} Q_2(u,v) &= f_1(u)q(0,v) + f_2(u)q(1,v) \\ &= (2u^3 - 3u^2 + 1)v^3 + (-2u^3 + 3u^2)(v^3 + 1) \\ &= v^3 - 2u^3 + 3u^2 \end{aligned}$$

sowie

$$\begin{pmatrix} q(0,0) & q(0,1) \\ q(1,0) & q(1,1) \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 2 \end{pmatrix}.$$

Wir erhalten also:

$$\begin{aligned} Q(u,v) &= (u^3 - 2v^3 + 3v^2) + (v^3 - 2u^3 + 3u^2) \\ &\quad -(2u^3 - 3u^2 + 1, -2u^3 + 3u^2) \begin{pmatrix} 0 & 1 \\ 1 & 2 \end{pmatrix} \begin{pmatrix} 2v^3 - 3v^2 + 1 \\ -2v^3 + 3v^2 \end{pmatrix} \\ &= u^3 + v^3. \end{aligned}$$

Literaturverzeichnis

[Bar85] R. E. Barnhill, *Surfaces in computer aided geometric design: A survey with new results*, Computer Aided Design, 1985, 1–17.

[Boe80] W. Boehm, *Inserting new knots into a B-spline curve*, Computer Aided Design, Vol. 12, 1980, 50–60.

[CLR80] E. Cohen, T. Lyche , R. F. Riesenfeld, *Discrete B-splines and subdivision techniques in computer aided geometric design and computer graphics*, Computer Graphics and Image Processing, Vol. 14, 1980, 87–111.

[dB72] C. de Boor, *On calculating with B-splines*, J. Approx. Theory, Vol. 6, 1972, 50–62.

[dB78] C. de Boor, *A Practical Guide to Splines*, Springer, 1978.

[dB87] C. de Boor, *B-Form Basics*, *Geometric Modeling, Algorithms and New Trends*, SIAM, 1987, 131–148.

[dC63] P. de Casteljau, *Courbes et surfaces a poles*, Tech. report, A. Citroen, Paris, 1963.

[Far79] G. E. Farin, *Subsplines über Dreiecken*, Ph.D. thesis, Braunschweig, F.R.G., 1979.

[Far80] G. E. Farin, *Bézier polynomials over triangles and the construction of piecewise C^r-polynomials*, Tech. Report TR/91, Dept. Mathematics, Brunel Univ., Uxbridge, Middlessex, 1980.

[Far86] G. E. Farin, *Triangular Bernstein-Bézier patches*, Computer Aided Geometric Design, Vol. 3, 1986, 83–127.

[Gor69] W. J. Gordon, *Distibutive lattices and the approximation of multivariate functions*, Approximation with Special Emphasis on Spline Functions (I. J. Schoenberg, Ed.), Academic Press, 1969, 223–277.

[Gor71] W. J. Gordon, *Blending function methods of bivariate and multivariate interpolation and approximation*, SIAM J. Numer. Anal., Vol. 8, 1971, 158–177.

[GR74] W. J. Gordon, R. F. Riesenfeld, *B-spline curves and surfaces*, Computer Aided Geometric Design (R. E. Barnhill, R. F. Riesenfeld, Eds.), Academic Press, 1974.

[HL89] Hoscheck, Lasser, *Grundlagen der geometrischen Datenverarbeitung*, Teubner, Stuttgart, 1989.

[LF80] J. M. Lane, R. F.Riesenfeld, *A theoretical development for the computer generation of piecewise polynomial surfaces*, IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. 2, 1980, 35–46.

[Pra84] H. Prautzsch, *Unterteilungsalgorithmen für multivariate Splines*, Ph.D. thesis, Braunschweig, 1984.

[Rie73] R. F. Riesenfeld, *Applications of B-Spline Approximation to Geometric Problems of Computer Aided Design*, Ph.D. thesis, Syracuse University, 1973.

[Sch81] L. L. Schumaker, *Spline Functions: Basic Theory*, Wiley & Sons, New York, 1981.

[Sei89] H.-P. Seidel, *Polynome, Splines und symmetrische rekursive Algorithmen im Computer Aided Geometric Design*, Habilitationsschrift, Tübingen, 1989.

[Slu89] P. B. Slusallek, *Flächenmodellierung mit Splines über Dreiecken*, Master's thesis, Tübingen, 1989.

Index

Glossar

parametrisierte Kurve (4.1.1.1, S.169)

Eine Abbildung $q : \mathbb{R} \supset I = [a,b] \longrightarrow \mathbb{R}^3$ mit $a < b$ heißt *parametrisierte Kurve*.

n–mal stetig differenzierbare Kurve (4.1.1.1, S.169)

Eine Kurve heißt *n–mal stetig differenzierbar*, wenn die zugehörige Abbildung q mindestens n-mal stetig differenzierbar (kurz: $C^n - stetig$) ist.

reguläre Kurve (4.1.1.1, S.169)

Eine Kurve heißt *regulär*, wenn die zugehörige Abbildung q einmal stetig differenzierbar ist und $q'(u) \neq 0$ für alle $u \in [a,b]$ gilt.
Der *Gradient* $q'(u)$ von q an der Stelle u ist ein Vektor im $\mathbb{R}^3$, der die Tangentenrichtung der Kurve an der Stelle u angibt.

Parametertransformation (4.1.1.1, S.170)

Zwei reguläre Kurven $q_1 : [a,b] \to \mathbb{R}^3$, $q_2 : [c,d] \to \mathbb{R}^3$ heißen *äquivalent*, wenn es eine bijektive differenzierbare Abbildung $\varphi : [a,b] \to [c,d]$ mit $\varphi'(u) > 0$ gibt, so daß $q_1 = q_2 \circ \varphi$ gilt.
Wir sagen in diesem Fall auch, daß q_2 durch φ *reparametrisiert* wurde und nennen φ einen richtungserhaltenden Parameterwechsel oder eine *Reparametrisierung* von q_1.

Bogenlänge (4.1.1.1, S.171)

Die *Bogenlänge* $s(u)$ einer parametrisierten regulären Kurve $q : [a,b] \to \mathbb{R}^3$ läßt sich gemäß folgender Formel berechnen:

$$s : [a,b] \to [0, s(b)], \quad u \mapsto s(u) := \int_a^u \|q'(t)\| dt.$$

Dabei bezeichnet $\|.\|$ die euklidische Norm im $\mathbb{R}^3$.

nach der Bogenlänge parametrisiert (4.1.1.1, S.171)

Entspricht für $u \in [a,b]$ der Wert der Bogenlänge $s(u)$ der zugrundeliegenden Intervallänge $u - a$, so nennt man q *nach der Bogenlänge parametrisiert.*
Das ist äquivalent zu der Bedingung $\|q'(u)\| = 1, \quad u \in [a,b]$.
Im allgemeinen läßt sich jede reguläre Kurve nach der Bogenlänge parametrisieren.

Funktionenraum (4.1.1.2, S.171)

$C[a,b]$ sei die Menge aller stetigen, reellen Funktionen auf dem Intervall $[a,b] \subset \mathbb{R}$. Definiert man für Elemente $f, g \in C[a,b]$ eine Addition und Skalarmultiplikation durch

$$(\alpha f + \beta g)(t) = \alpha f(t) + \beta g(t),$$

mit $\alpha,\ \beta \in \mathbb{R}$, so ist $C[a,b]$ ein reeller Vektorraum.
In diesem Vektorraum heißen n Funktionen $f_1, \cdots, f_n \in C[a,b]$ *linear unabhängig*, falls aus $\sum c_i f_i = 0$ auf $[a,b]$ immer $c_1 = c_2 = \cdots = c_n = 0$ folgt.
Dabei ist $f = 0$ auf $[a,b]$, falls $f(t) = 0$ für alle $t \in [a,b]$ gilt.

Polynomraum (4.1.1.3, S.172)

Die Menge aller Polynome $\mathbb{P}^n[a,b]$ vom Grad n bildet einen Unterraum von $C[a,b]$ der Dimension $n+1$. Die Monome $1,t,t^2\cdots t^n$ bilden eine Basis, die sogenannte *Taylorbasis (Monombasis)* des Polynomraums. In dieser Basis ist ein reelles Polynom p durch

$$p(t) = c_n t^n + c_{n-1}t^{n-1} + \cdots + c_1 t + c_0$$

mit Taylorkoeffizienten $c_n,\cdots,c_0 \in \mathbb{R}$ gegeben.
Wählt man statt reeller Koeffizienten c_i Vektoren aus dem $\mathbb{R}^3$, so erhält man Polynomkurven im $\mathbb{R}^3$.

Nachteil dieser Darstellung für den interaktiven Entwurf:
Die Taylorkoeffizienten entziehen sich einer unmittelbaren geometrischen Deutung.

Interpolationsaufgabe (4.2.1, S.172)

Gesucht ist eine Funktion q, die folgende Bedingungen erfüllt:

$$P_i = q(u_i),\ u_i \in \mathbb{R},\ P_i \in \mathbb{R}^3,\ i = 0,\cdots,n,$$

wobei die u_i die Parameterwerte der zugehörigen Punkte P_i darstellen.
Die Punkte P_i heißen *Stützpunkte,* die u_i *Stützstellen* oder Parameterwerte, der Vektor $(u_0,\cdots,u_n)$ heißt *Stützstellenvektor.*
Die Paare (u_i,P_i) legen die Knoten des Interpolationspolynoms fest.

Invarianz bei affinen Abbildungen (4.2.1, S.173)

Gilt für die Basispolynome $f_0,\cdots,f_n$ einer Polynombasis von $\mathbb{P}^n[a,b]$, $a<b, a,b \in \mathbb{R}$, und für alle $u \in [a,b]$ die Bedingung

$$f(u) = \sum_{i=0}^{n} f_i(u) = 1,$$

so gilt für eine affine Transformation Φ und Stützpunkte $P_i,\ i = 1,\cdots,n$:

$$\Phi\left(\sum_{i=0}^{n} P_i f_i(u)\right) = \sum_{i=0}^{n} \Phi(P_i) f_i(u).$$

Anschaulich gesprochen erhält man dieselbe Interpolationskurve, unabhängig davon, ob man zuerst die Stützpunkte affin verschiebt und dann die Kurve berechnet, oder ob man zuerst die Kurve berechnet und dann die einzelnen Kurvenpunkte affin transformiert.

Interpolation mit Monomen (4.2.2, S.173)

Sind $n+1$ Knoten (u_i,P_i) mit paarweise verschiedenen Stützstellen u_i, $i = 0,\cdots,n$, gegeben, so suchen wir ein Polynom

$$q(u) = \sum_{j=0}^{n} c_j u^j, \quad c_j \in \mathbb{R}^3,$$

mit

$$P_i = q(u_i) = \sum_{j=0}^{n} c_j (u_i)^j, \quad i = 0,\cdots,n.$$

Dazu sind die Koeffizienten c_i aus folgendem Gleichungssystem zu bestimmen:

$$\begin{pmatrix} 1 & u_0 & \cdots & u_0^n \\ \cdot & & & \cdot \\ \cdot & & & \cdot \\ \cdot & & & \cdot \\ 1 & u_n & \cdots & u_n^n \end{pmatrix} \begin{pmatrix} c_0 \\ \cdot \\ \cdot \\ \cdot \\ c_n \end{pmatrix} = \begin{pmatrix} P_0 \\ \cdot \\ \cdot \\ \cdot \\ P_n \end{pmatrix}.$$

Da die Stützstellen als paarweise verschieden vorausgesetzt sind, ist die zu dem obigen Gleichungssystem gehörende Matrix (*Vandermondsche Matrix*) invertierbar und die Koeffizienten c_i lassen sich eindeutig bestimmen.

Nachteile dieser Interpolationsmethode:

1. Die Lösung des obigen Gleichungssystems kann für großes n zu numerischen Problemen führen.
2. Wird nur ein Knoten geändert, so muß im allgemeinen das ganze System neu gelöst werden.
3. Kurven in Taylorbasis sind nicht affin invariant.
4. Durch Monominterpolation entstehende Kurvensegmente können in der Regel nur mit C^0-Stetigkeit aneinandergefügt werden.
5. Die Koeffizienten c_j sind nicht unmittelbar geometrisch interpretierbar.

Interpolation mit Lagrange-Polynomen (4.2.3, S.174)

Bei dieser Interpolationsform wird anstelle der Taylorbasis die sogenannte *Lagrangebasis* verwendet.
Für $n+1$ Knoten $(u_i, P_i), \quad i = 0, \cdots, n,$ mit $u_0 < u_1 < \cdots < u_n$ definiert man die Basisfunktionen durch

$$L_i^n(u) = \frac{(u-u_0)(u-u_1)\cdots(u-u_{i-1})(u-u_{i+1})\cdots(u-u_n)}{(u_i-u_0)(u_i-u_1)\cdots(u_i-u_{i-1})(u_i-u_{i+1})\cdots(u_i-u_n)}.$$

Das Interpolationspolynom hat in dieser Basis die Form

$$q(u) = \sum_{i=0}^{n} P_i L_i^n(u) = \sum_{i=0}^{n} P_i \prod_{\substack{j=0 \\ j\neq i}}^{n} \frac{(u-u_j)}{(u_i-u_j)}$$

Für die Basispolynome L_i^n gilt:

$$L_i^n(u_k) = \delta_{ik} = \left\{ \begin{array}{ll} 1 & \text{für} \quad i = k \\ 0 & \text{für} \quad i \neq k \end{array} \right\},$$

wobei δ_{ik} das Kronecker-Symbol ist,
und

$$\sum_{i=0}^{n} L_i^n(u) = 1 \quad u \in [0,1],$$

woraus die affine Invarianz der Lagrangeschen Interpolationskurve folgt.

Vorteil der Lagrangeinterpolation:

Es müssen keine Polynomkoeffizienten berechnet werden, da diese mit den

zu interpolierenden Punkten übereinstimmen.

Nachteile:

1. Der Grad der Interpolationsfunktion hängt von der Anzahl der Bestimmungsstücke ab. Entsprechendes gilt für den Rechenaufwand zur numerischen Auswertung.
2. Lokale Änderungen sind nicht möglich.
3. Bei höheren Polynomgraden zeigt die Lagrange-Interpolation eine unerwünschte Welligkeit.
4. Wie bei den Monomen können Kurvensegmente nur mit C^0-Stetigkeit aneinandergefügt werden.

Interpolation mit kubischen Hermite-Polynomen (4.2.4, S.175)

Basisfunktionen sind die *kubische Hermite-Polynome* $H_i^3,\ i = 0,\cdots,3$, auf dem Intervall $[0,1]$:

$$\begin{aligned} H_0^3(u) &= (1-u)^2(1+2u) \\ H_1^3(u) &= u(1-u)^2 \\ H_2^3(u) &= -u^2(1-u) \\ H_3^3(u) &= (3-2u)u^2. \end{aligned}$$

In der *Hermitebasis* gelten für die Koeffizienten $c_0,\cdots,c_3$ eines Polynoms dritten Grades

$$q(u) = c_0H_0^3(u) + c_1H_1^3(u) + c_2H_2^3(u) + c_3H_3^3(u)$$

$$c_0 = q(0), \quad c_1 = q'(0), \quad c_2 = q'(1), \quad c_3 = q(1).$$

Daher können die Koeffizienten geometrisch gedeutet werden:
Sind zwei Punkte P_0, P_1 und zwei Tangentenvektoren m_0 und m_1 der Kurve in diesen Punkten zu interpolieren, so löst das Polynom

$$q(u) = P_0H_0^3(u) + m_0H_1^3(u) + m_1H_2^3(u) + P_1H_3^3(u)$$

die Interpolationsaufgabe.

Bernsteinpolynom (4.3.1, S.177)

Die *Bernsteinpolynome* über dem Intervall $[s,t]$ lassen sich aus den binomischen Formeln

$$(t-s)^n = ((t-u)+(u-s))^n = \sum_{i=0}^{n} \binom{n}{i} (u-s)^i(t-u)^{n-i}$$

entwickeln. Die durch $(t-s)^n$ dividierten Summanden

$${}_s^tB_i^n(u) = \frac{1}{(t-s)^n} \binom{n}{i} (u-s)^i(t-u)^{n-i}, \quad i = 0,1,\cdots,n$$

sind Polynome vom Grad n und heißen *Bernsteinpolynome* vom Grad n.
Sie bilden eine Basis des Polynomraumes $\mathbb{P}^n$ und besitzen folgende Eigenschaften:

$$\begin{aligned}
\textstyle\sum_{i=0}^{n} {}_s^t B_i^n(u) &= 1 \text{ für } u \in [s,t] && \text{Partition der 1}\\
{}_s^t B_i^n(u) &\geq 0 \text{ für } u \in [s,t] && \text{Positivität}\\
{}_s^t B_i^n(u) &= \tfrac{u-s}{t-s}\,{}_s^t B_{i-1}^{n-1}(u) + \tfrac{t-u}{t-s}\,{}_s^t B_i^{n-1}(u) && \text{Rekursion}\\
{}_s^t B_i^n(u) &= {}_s^t B_{n-i}^n(t-(u-s)) && \text{Symmetrie.}
\end{aligned}$$

Häufig wird als Parameterintervall das Intervall $[0,1]$ gewählt. In diesem Fall schreibt man statt ${}_0^1 B_i^n$ einfach B_i^n.

Bézier-Kurve (4.3.2, S.178)
Die Darstellung von q mit

$$q(u) = \sum_{i=0}^{n} {}_s^t B_i^n(u) \cdot b_i, \quad b_i \in \mathbb{R}^3,$$

heißt *Bézier-Darstellung* im $\mathbb{R}^3$.
Die Punkte b_i, $i = 0, \cdots, n$, heißen *Bézier-Punkte* und definieren das *Bézier-Polygon.*
Da

$$\sum_{i=0}^{n} {}_s^t B_i^n(u) = 1, \quad u \in [s,t],$$

gilt, sind die Bézier-Kurven invariant unter affinen Transformationen.

Zusammenhang zwischen Kurve und Bézier-Polygon:

1. Für $u \in [s,t]$ liegt die Kurve $q(u)$ innerhalb der konvexen Hülle des Bézier-Polygons.
2. Es gilt $q(s) = b_0$, $q(t) = b_n$, d.h. die Kurve verläuft durch Anfangs- und Endpunkt des Polygons.
3. Für die Ableitungen in den Anfangs- und Endpunkten gilt

$$q'(s) = \frac{n}{t-s}(b_1 - b_0),$$

$$q'(t) = \frac{n}{t-s}(b_n - b_{n-1}),$$

   d.h. die Kurve besitzt als Tangenten die Anfangs- und Endseiten des Polygons.
4. Für die i–te Ableitung von q in den Randwerten $u = s$ bzw. $u = t$ gilt:

$$q^{(i)}(s) = \frac{1}{(t-s)^i}\frac{n!}{(n-i)!} \cdot \sum_{j=0}^{i} \binom{i}{j} (-1)^{i-j} \cdot b_j$$

   bzw.

$$q^{(i)}(t) = \frac{1}{(t-s)^i}\frac{n!}{(n-i)!} \cdot \sum_{j=0}^{i} \binom{i}{j} (-1)^{j} \cdot b_{n-j},$$

d.h. die i-te Ableitung von q im Randwert $u = s$ hängt nur von dem Randpunkt selbst und seinen i unmittelbar benachbarten Bézier-Punkten ab. Analoges gilt im Randwert $u = t$.

Vor- und Nachteile:

+ Das Bézier-Polygon vermittelt eine schnelle Übersicht über den möglichen Kurvenverlauf.
+ Änderungen der Kontrollpunkte erlauben kontrollierte Änderungen des Kurvenverlaufs.
- Der Grad der Kurve ist an die Anzahl der Ecken des Bézier–Polygons gekoppelt, was zu hohen Polynomgraden führt.
- Die Änderung eines Bézier-Punktes wirkt sich auf die gesamte Kurve aus und damit auch auf Bereiche, die eventuell nicht mehr geändert werden sollen.

Algorithmus von de Casteljau (4.3.2.1, S.179)

Neben der stabilen *Berechnung der Funktionswerte* einer Bézier-Kurve (mittels fortgesetzter Bildung von Konvexkombinationen) erlaubt dieser Algorithmus darüber hinaus die *Bestimmung von Ableitungen* und die *Unterteilung der Kurve* in mehrere Segmente. Die bei fortgesetzter Unterteilung entstehenden neuen Bézier-Polygone konvergieren gegen die Bézier-Kurve und können zur graphischen Darstellung der Kurve verwendet werden.
Berechnung des Funktionswertes $q(u)$ an der Stelle u:
Rekursion:

$$\begin{aligned} b_i^0(u) &= b_i \\ b_i^r(u) &= \frac{(u-s)}{(t-s)} \cdot b_{i+1}^{r-1}(u) + (1 - \frac{(u-s)}{(t-s)}) \cdot b_i^{r-1}(u) \end{aligned}$$

Ableitung von Bézier-Kurven:
Die i-te Ableitung $q^{(i)}(u)$ ergibt sich aus dem de Casteljau-Schema in der $(n-i)$-ten Stufe zu

$$q^{(i)}(u) = \frac{1}{(t-s)^i} \frac{n!}{(n-i)!} \sum_{k=0}^{i} \begin{pmatrix} i \\ k \end{pmatrix} (-1)^{i-k} \cdot b_k^{n-i}(u).$$

Für die *erste Ableitung* gilt daher

$$q'(u) = \frac{n}{(t-s)} (b_1^{n-1}(u) - b_0^{n-1}(u)).$$

Unterteilung einer Bézier-Kurve in zwei Teilsegmente:
Für $s \leq q \leq t$ gilt die Beziehung

$$q(u) = \begin{cases} \sum_{i=0}^{n} {}_s^q B_i^n(u) \cdot b_0^i(q), & s \leq u \leq q \\ \\ \sum_{i=0}^{n} {}_q^t B_i^n(u) \cdot b_i^{n-i}(q), & q \leq u \leq t \end{cases}.$$

Hierbei bezeichnen ${}_s^q B_i^n(u)$ bzw. ${}_q^t B_i^n(u)$ die Bernstein-Polynome bezüglich der Teilintervalle $[s,q]$ bzw. $[q,t]$.

Darstellung einer Bézier-Kurve:
Bei fortgesetzter Unterteilung konvergiert die Folge der verfeinerten Kontrollpolygone gleichmäßig gegen die Kurve. Bei fortgesetzter Halbierung des Parameterintervalls ist diese Konvergenz exponentiell und liefert ein praktisches Verfahren zur Darstellung einer Bézier-Kurve durch eine *stückweise lineare Approximation.*

Variation Diminishing Property (4.3.2.3, S.181)
Die Bézier-Kurven im $\mathbb{R}^3$ haben die Eigenschaft, daß eine beliebige Ebene im $\mathbb{R}^3$ die Kurve nicht öfter als das Kontrollpolygon schneidet (*Variation Diminishing Property*).

Graderhöhung bei Bézier-Kurven: Corner Cutting (4.3.2.4, S.181)
Werden zur genaueren Approximation einer geometrischen Figur mittels einer Bézier-Kurve mehr Freiheitsgrade benötigt als vorhanden sind, so kann der Grad der Kurve erhöht werden. Bei Graderhöhung um den Grad eins wird dadurch ein weiterer Kontrollpunkt in das Kontrollpolygon eingefügt, ohne daß sich die geometrische Form der Kurve ändert.
Ist eine Bézier-Kurve q vom Grad n mit den Bézier-Punkten $b_0, \cdots, b_n$ gegeben, so ist q als Bézier-Kurve vom Grad $n+1$ darstellbar mit den Bézier-Punkten

$$\begin{aligned} b_0^* &= b_0, \\ b_j^* &= \frac{j}{n+1} \cdot b_{j-1} + (1 - \frac{j}{n+1}) \cdot b_j, \quad j = 1, \cdots, n, \\ b_{n+1}^* &= b_n. \end{aligned}$$

Wegen ihrer geometrischen Bedeutung wird die Graderhöhung auch als “Corner Cutting” bezeichnet.

Spline (4.4, S.182)
Ein *Spline* ist eine stetige Abbildung q von einer Menge von Intervallen in den $\mathbb{R}^3$. Die Intervalle $[t_i, t_{i+1}],\ i = 0, \cdots, n-1$, werden durch einen *Stützstellenvektor* $T = (t_0, t_1 \cdots, t_n)$ mit $t_0 \leq t_1 \leq \cdots \leq t_n$ definiert. Jedes Intervall $[t_i, t_{i+1}]$ wird dabei auf ein Polynomsegment, das *Spline–Segment*, abgebildet. An den Stützstellen $t_i,\ i = 0, \cdots, n$, stoßen die Spline-Segmente nach Definition zusammen. Dort müssen die Splinesegmente aber nicht unbedingt dieselben Tangentenvektoren besitzen. Sie können sich sowohl in Betrag als auch Richtung unterscheiden.

parametrische Stetigkeit (4.4.1, S.182)
Parametrisch stetiger Anschluß (C^n–stetiger Übergang):
Seien $q_1 : [a_1, b_1] \to \mathbb{R}^3$, $q_2 : [a_2, b_2] \to \mathbb{R}^3$ zwei n-mal stetig differenzierbare reguläre Kurven. q_1 und q_2 schließen an der Stelle b_1, a_2 C^n–stetig genau dann aneinander, wenn

$$q_1^{(k)}(b_1) = q_2^{(k)}(a_2) \quad \text{für alle } k = 0, \cdots, n.$$

geometrische Stetigkeit (4.4.1, S.182)
Seien $q_1 : [a_1, b_1] \to \mathbb{R}^3$, $q_2 : [a_2, b_2] \to \mathbb{R}^3$ zwei n-mal stetig differenzierbare reguläre Kurven. q_1 und q_2 schließen an der Stelle b_1, a_2 G^n–stetig

aneinander, falls es eine zu q_1 äquivalente Kurve $r_1 : [a_0, b_0] \to \mathbb{R}^3$ gibt, so daß r_1 und q_2 an der Stelle b_0, a_2 C^n–stetig aneinanderschließen.

Die G^n–Stetigkeit ist unabhängig von der Parametrisierung. Ferner sieht man aus dieser Definition, daß für reguläre Kurven aus C^n–Stetigkeit stets G^n–Stetigkeit folgt. Für nicht reguläre Kurven braucht dies nicht der Fall zu sein.

Bézier-Spline (4.5, S.183)

Definiert man die Segmente eines Splines durch Bézier-Polynome, so spricht man von *Bézier-Splines.* Dabei können verschiedene Stetigkeitsanforderungen an die Knotenpunkte gestellt werden.

B-Spline-Basisfunktionen (4.6.1, S.185)

Gegeben seien $n \leq m \in \mathbb{N}$ sowie eine schwach monotone Folge

$$T = (t_0 = ... = t_n, t_{n+1}, ..., t_m, t_{m+1} = ... = t_{m+n+1})$$

von *Knoten* mit $t_i < t_{i+n+1}, 0 \leq i \leq m$.
Die normalisierten *B-Splines* N_i^n vom Grad n über T sind rekursiv definiert durch

$$N_i^0(u) := \begin{cases} 1, & \text{falls } t_i \leq u < t_{i+1} \\ 0, & \text{sonst} \end{cases}$$

und

$$N_i^r(u) \quad := \quad \frac{u - t_i}{t_{i+r} - t_i} \cdot N_i^{r-1}(u) \; + \frac{t_{i+1+r} - u}{t_{i+1+r} - t_{i+1}} \cdot N_{i+1}^{r-1}(u)$$

für $1 \leq r \leq n$.
Für $t_{i+r} = t_i$ bzw. $t_{i+r+1} = t_{i+1}$ ist in obiger Gleichung der entsprechende Summand gleich 0 zu setzen.

Eigenschaften der normalisierten B-Splines $N_i^n(u)$:

1. $N_i^n(u)$ besteht stückweise aus Polynomen vom Grad n über T.
2. Die Funktionen N_i^n besitzen einen lokalen Träger, d.h. $N_i^n(u) = 0$ für $u \notin [t_i, t_{i+n+1}]$.
3. Es gilt $N_i^n(u) \geq 0$ für alle $u \in [t_0, t_{m+n+1}]$.
4. Für $u \in [t_0, t_{m+n+1}]$ gilt: $\sum_i N_i^n(u) = 1$.
5. Ist t_j, $j \in \{0, \cdots, m+n+1\}$ ein einfacher Knoten, d.h. $t_{j-1} \neq t_j \neq t_{j+1}$, so ist $N_i^n(t_j)$ mindestens $C^{n-1}-$stetig.

Bei einem Mehrfachknoten $s = t_{j+1} = ... = t_{j+\mu}$ der Multiplizität μ sind die normalisierten B-Splines N_i^n vom Grad n mindestens $C^{n-\mu}$-stetig.

B-Spline-Kurve (4.6.2, S.186)

Gegeben seien $n \leq m \in \mathbb{N}$ sowie eine schwach monotone Folge

$$T = (t_0 = ... = t_n, t_{n+1}, ..., t_m, t_{m+1} = ... = t_{m+n+1})$$

von Knoten mit $t_i < t_{i+n+1}$ und Punkte $d_0, ..., d_m \in \mathbb{R}^3$.
Dann heißt eine Kurve

$$q(u) = \sum_{i=0}^{m} N_i^n(u) \cdot d_i, \qquad d_i \in \mathbb{R}^3,$$

eine *B-Spline-Kurve* (oder einfach ein *B-Spline*) vom Grade n über T.
Der Grad kann dabei vom Benutzer gewählt werden. Häufig beschränkt man sich jedoch auf kubische B-Splines.
Die Punkte $d_0, ..., d_m$ heißen *Kontroll-* oder *de Boor-Punkte* von q. Sie bilden das *Kontroll-* oder *de Boor-Polygon*.

Lokale Wirkung der de Boor-Punkte (4.6.2.1, S.187)
Für die B-Splines gilt $N_i^n(u) = 0$, falls $u \notin [t_i, t_{i+n+1}]$. Daher beeinflußt der i-te de Boor-Punkt d_i die Kurve nur über dem Parameterbereich $[t_i, t_{i+n+1}]$.
Die Form der Kurve wird über dem Intervall $[t_i, t_{i+n+1}]$ nur von den de Boor-Punkten $d_{i-n}, \cdots, d_{i+n}$ beeinflußt.

Algorithmus von de Boor (4.6.2.2, S.187)
Die Verallgemeinerung des de Casteljau-Algorithmus von Bézier-Kurven auf B-Splines ist der *de Boor-Algorithmus.*
Gegeben sei ein B-Spline

$$q(u) = \sum_{i=0}^{m} N_i^n(u) \cdot d_i$$

vom Grad n über T.
Für $t_l \le u < t_{l+1}$ betrachten wir das durch die Rekursion

$$d_i^0(u) := d_i, \qquad i = l-n, ..., l,$$

und

$$\begin{aligned} d_i^r(u) \quad &:= \quad (1 - \frac{u - t_{i+r}}{t_{i+n+1} - t_{i+r}}) \cdot d_i^{r-1}(u) \\ &+ \quad \frac{u - t_{i+r}}{t_{i+n+1} - t_{i+r}} \cdot d_{i+1}^{r-1}(u), \; i = l-n, ..., l-r, \end{aligned}$$

für $0 \le r \le n$ definierte Dreiecksschema.
Dann gilt $q(u) = d_{l-n}^n(u)$.

Im Spezialfall $T = (\underbrace{s, ..., s}_{n+1}, \underbrace{t, ..., t}_{n+1})$ geht der de Boor-Algorithmus in den Algorithmus von de Casteljau über.

Zusammenhang zwischen Kontrollpolygon und B-Spline (4.6.2.3, S.189)
Ist

$$q(u) = \sum_{i=0}^{m} N_i^n(u) \cdot d_i$$

eine B-Spline-Kurve vom Grad n über T in $\mathbb{R}^3$, dann hängt die Gestalt des Splines q mit dem Kontrollpolygon wie folgt zusammen:

1. Für $t_l \le u \le t_{l+1}$ liegt $q(u)$ in der konvexen Hülle der $n+1$ Kontrollpunkte $d_{l-n}, ..., d_l$.

2. Fallen n Kontrollpunkte $d_{l-n+1} = ... = d_l =: d$ zusammen, so gilt $q(t_{l+1}) = d$, d.h. die Kurve verläuft durch diesen n-fachen Kontrollpunkt.
3. Liegen $n+1$ Kontrollpunkte $d_{l-n}, ..., d_l$ auf einer Geraden L, so gilt $q(u) \in L$ für $t_l \leq u \leq t_{l+1}$, d.h. die Kurve hat mit der Geraden L ein Stück gemeinsam.
4. Fallen n Knoten $t_{l+1} = ...t_{l+n} =: t$ zusammen, so ist $q(t) = d_l$ ein Kontrollpunkt, und die Kurve ist dort tangential an das Kontrollpolygon. Insbesondere verläuft die Kurve bei $(n+1)$-fachen Anfangs- und Endknoten durch die Endpunkte des Polygons und ist dort tangential an das Kontrollpolygon.

Knoteneinfügen in B-Splines (4.6.2.4, S.191)

Algorithmus von Boehm:
Gegeben sei eine B-Spline-Kurve $q(u) = \sum_{i=0}^{m} N_i^n(u) \cdot d_i$ vom Grad n über dem Knotenvektor $T = (t_0, ..., t_{m+n+1})$. Nach Einfügen eines zusätzlichen Knotens t, etwa

$$t_l \leq t < t_{l+1},$$

besitzt q eine Darstellung

$$q(u) = \sum_{i=0}^{m+1} N_i^n(u) \cdot d_i^*$$

als B-Spline vom Grad n über dem verfeinerten Knotenvektor

$$T^* = (t_0, ..., t_l, t, t_{l+1}, ..., t_{n+m+1}).$$

Die neuen Kontrollpunkte d_i^* werden wie folgt berechnet:

$$d_i^* = (1 - a_i) \cdot d_{i-1} + a_i \cdot d_i$$

mit

$$a_i = \begin{cases} 1 & \text{für } i \leq l-n \\ \frac{t-t_i}{t_{i+n}-t_i} & \text{für } l-n+1 \leq i \leq l. \\ 0 & \text{für } l+1 \leq i \end{cases}$$

Parametrisierte Flächen (4.7.1.1, S.193)

Sei $\emptyset \neq D \subset \mathbb{R}^2$ gegeben. Eine Abbildung $q : D \to \mathbb{R}^3$ heißt *parametrisierte Fläche*.

stetig differenzierbare Fläche (4.7.1.1, S.193)

Eine Fläche heißt *n-mal stetig differenzierbar*, wenn die Abbildung q mindestens n-mal stetig differenzierbar ist, d.h. wenn q n-mal stetige partielle Ableitungen besitzt.

reguläre Fläche (4.7.1.1, S.193)

Eine Fläche heißt *regulär*, wenn q einmal stetig differenzierbar ist und

$$q_u(u,v) := \frac{\partial q(u,v)}{\partial u}, q_v(u,v) := \frac{\partial q(u,v)}{\partial v} \text{ für alle } (u,v) \in D$$

linear unabhängig sind.
Die Vektoren $q_u(u,v)$ und $q_v(u,v)$ heißen *u-Tangente* bzw. *v-Tangente* von der Stelle (u,v).

Tangentialebene (4.7.1.1, S.194)

Ist $q : D \to \mathbb{R}^3$ eine reguläre parametrisierte Fläche, so heißt die von den Vektoren $q_u(u_0,v_0)$ und $q_v(u_0,v_0)$ aufgespannte Ebene *Tangentialebene* $T_{q(u_0,v_0)}$ im Flächenpunkt $q(u_0,v_0)$.

Normalenvektor einer Fläche (4.7.1.1, S.194)

Der Vektor

$$n(u,v) := \frac{q_u(u,v) \times q_v(u,v)}{\|q_u(u,v) \times q_v(u,v)\|}$$

heißt *Normalenvektor* im Punkt $q(u,v)$.
Er steht senkrecht auf der Tangentialebene und ist unabhängig von der speziellen Wahl der Parametrisierung.

Wendelfläche (4.7.1.1, S.194)

$$q : \mathbb{R}^2 \to \mathbb{R}^3, q(u,v) = (v\cos u, v\sin u, cu) \text{ mit } c \in \mathbb{R}$$

Tensorproduktraum (4.7.2, S.194)

Sind $\{F_i^m,\ i = 0,\cdots,m\}$ und $\{G_j^n,\ j = 0,\cdots,n\}$ Basen von zwei Funktionenräumen R_1, R_2 mit reellwertigen univariaten Funktionen (Funktionen in einer Variable) über den Intervallen $[a,b]$ bzw. $[c,d]$, so bilden die bivariaten reellwertigen Funktionen (Funktionen in zwei Variablen)

$$F_i^m G_j^n(u,v) = F_i^m(u) \cdot G_j^n(v),$$

$i = 0,\cdots,m,\ j = 0,\cdots,n,\ (u,v) \in [a,b] \times [c,d]$, eine *Basis des Tensorproduktraumes* $R_1 \otimes R_2$.
Dieser Raum hat die Dimension $(m+1)\cdot(n+1)$.

Tensorprodukt-Fläche (4.7.2, S.195)

Ist eine Basis $F_i^m G_j^n,\ i = 0,\cdots,m,\ j = 0,\cdots,n$ eines Tensorproduktraumes $R_1 \otimes R_2$ von Funktionen über $[a,b] \times [c,d] \longmapsto \mathbb{R}$ und Koeffizienten $c_{ij} \in \mathbb{R}^3,\ i = 0,\cdots,m,\ j = 0,\cdots,n$, gegeben, so heißt die bezüglich der Tensorprodukt-Basis dargestellte Funktion

$$q(u,v) = \sum_{i=0}^{m}\sum_{j=0}^{n} c_{ij} F_i^m G_j^n(u,v) = \sum_{i=0}^{m}\sum_{j=0}^{n} c_{ij} F_i^m(u) \cdot G_j^n(v),$$

$(u,v) \in [a,b] \times [c,d]$, *Tensorprodukt-Fläche*.
In Matrizenschreibweise ergibt sich

$$q(u,v) = (F_0^m(u) \cdots F_m^m(u)) \begin{pmatrix} c_{00} & \cdots & c_{0n} \\ \cdot & & \cdot \\ \cdot & & \cdot \\ \cdot & & \cdot \\ c_{m0} & \cdots & c_{mn} \end{pmatrix} \begin{pmatrix} G_0^n(v) \\ \cdot \\ \cdot \\ \cdot \\ G_n^n(v) \end{pmatrix}.$$

Spline-Flächen-Interpolation: Interpolationsaufgabe (4.7.3, S.197)

Gesucht ist eine Funktion q, so daß folgende Bedingungen erfüllt sind:

$$P_{ik} = q(u_i, v_k), \quad P_{ik} \in \mathbb{R}^3,\ u_i, v_k \in \mathbb{R}, \quad i = 0,\cdots,n, \quad k = 0,\cdots,m,$$

wobei die (u_i, v_k) die Parameterwerte der zu interpolierenden Punkte P_{ik} sind.

Spline-Flächen-Interpolation: Monom- und Lagrange (4.7.3.1, S.198)

Monom-Interpolation:
Ist eine Tensorproduktfläche durch

$$q(u,v) = \sum_{i=0}^{m} \underbrace{\sum_{j=0}^{n} c_{ij} G_j^n(v)}_{a_i(v)} F_i^m(u), \quad a \le u \le b,\ c \le v \le d,$$

und eine $(m+1) \times (n+1)$ große Matrix von Stützpunkten $P_{kl} \in \mathbb{R}^3$ und Stützstellen $(u_k, v_l),\ k = 0, \cdots, m,\ l = 0, \cdots, n$, gegeben und soll die Tensorproduktfläche diese Datenpunkte interpolieren, so muß

$$q(u_k, v_l) = \sum_{i=0}^{m} \sum_{j=0}^{n} c_{ij} F_i^m(u_k) G_j^n(v_l) = P_{kl}, \quad k = 0, \cdots, m,\ l = 0, \cdots, n,$$

für alle $(m+1) \times (n+1)$ Parameterpaare $(u_l, v_k),\ k = 0, \cdots, m,\ l = 0, \cdots, n$, und Datenpunkte P_{kl} erfüllt sein.
In Matrizenschreibweise lauten diese Gleichungen

$$q(u_k, v_l) = (F_0^m(u_k) \cdots F_m^m(u_k)) \begin{pmatrix} c_{00} & \cdots & c_{0n} \\ \cdot & & \cdot \\ \cdot & & \cdot \\ \cdot & & \cdot \\ c_{m0} & \cdots & cmn \end{pmatrix} \begin{pmatrix} G_0^n(v_l) \\ \cdot \\ \cdot \\ \cdot \\ G_n^n(v_l) \end{pmatrix} = P_{kl},$$

$$k = 0, \cdots, m,\ l = 0, \cdots, n.$$

Wir erhalten $(m+1) \cdot (n+1)$ Gleichungen, die wir zusammen in Matrizenform schreiben können:

$$\mathbf{P} = F\mathbf{C}G$$

mit

$$\mathbf{P} = \begin{bmatrix} P_{00} & \cdots & P_{0n} \\ \vdots & & \vdots \\ P_{m0} & \cdots & P_{mn} \end{bmatrix},$$

$$F = \begin{bmatrix} F_0^m(u_0) & \cdots & F_m^m(u_0) \\ \vdots & & \vdots \\ F_0^m(u_m) & \cdots & F_m^m(u_m) \end{bmatrix},$$

$$\mathbf{C} = \begin{bmatrix} c_{00} & \cdots & c_{0n} \\ \vdots & & \vdots \\ c_{m0} & \cdots & c_{mn} \end{bmatrix},$$

$$G = \begin{bmatrix} G_0^n(v_0) & \cdots & G_0^n(v_n) \\ \vdots & & \vdots \\ G_n^n(v_0) & \cdots & G_n^n(v_n) \end{bmatrix}.$$

Sind die Parameterwerte $u_0, \cdots, u_m$ und $v_0, \cdots, v_n$ paarweise verschieden und verwendet man die Monom-Basis, so sind die Matrizen F und G Vandermond'sche Matrizen und daher invertierbar. Die Koeffizientenmatrix $\mathbf{C}$ ist dann gegeben durch

$$\mathbf{C} = F^{-1}\mathbf{P}G^{-1}.$$

Lagrange-Interpolation:
Wird die Lagrange-Basis verwendet, d.h. die Funktionen F_i^m, $i = 0, \cdots, m$, G_j^n, $j = 0, \cdots, n$, sind die Lagrange'schen Basisfunktion zu den Parameterwerten $u_0, \cdots, u_m$ bzw. $v_0, \cdots, v_n$, so gilt

$$F_i^m(u_k) = \delta_{ik} = \begin{cases} 1 & \text{für } i = k \\ 0 & \text{sonst} \end{cases}$$

und

$$G_j^n(v_l) = \delta_{jl} = \begin{cases} 1 & \text{für } j = l \\ 0 & \text{sonst} \end{cases}.$$

Daher sind F und G in diesem Fall Einheitsmatrizen und es gilt $\mathbf{C} = \mathbf{P}$. Wie im Fall von Kurven sind die Koeffizienten direkt durch die zu interpolierenden Punkte gegeben, d.h.

$$q(u,v) = \sum_{i=0}^{m}\sum_{j=0}^{n} P_{ij} L_i^m(u) L_j^n(v).$$

Spline-Flächen-Interpolation: bikubische Hermite-Int. (4.7.3.2, S.199)

Ein bikubisches Hermite-Pflaster ist gegeben durch

$$q(u,v) = \sum_{i=0}^{3}\sum_{j=0}^{3} H_i^3(u) h_{ij} H_j^3(v),\ 0 \le u,v \le 1,$$

wobei die H_i^3, $i = 0, \cdots, 3$ die Hermite-Basisfunktionen sind.
Dabei sind die h_{ij} gegeben durch

$$[h_{ij}] = \begin{bmatrix} q(0,0) & q_v(0,0) & q_v(0,1) & q(0,1) \\ q_u(0,0) & q_{uv}(0,0) & q_{uv}(0,1) & q_u(0,1) \\ q_u(1,0) & q_{uv}(1,0) & q_{uv}(1,1) & q_u(1,1) \\ q(1,0) & q_v(1,0) & q_v(1,1) & q(1,1) \end{bmatrix}.$$

Dabei sind

$$q_u = \frac{\partial q}{\partial u}, q_v = \frac{\partial q}{\partial v} \quad \text{und} \quad q_{uv} = \frac{\partial^2 q}{\partial u \partial v}.$$

Die gemischten Ableitungen q_{uv} an den Ecken der Pflaster heißen *Twistvektoren*.

Bézier-Spline-Flächen (4.7.4.1, S.200)

Sind $o < r$ und $s < t \in \mathbb{R}$, so definieren wir ein Tensorprodukt-Bézier-Pflaster vom Grad (m,n) durch

$$q(u,v) = \sum_{i=0}^{m}\sum_{j=0}^{n} b_{ij}\,{}_o^r B_i^m(u)\,{}_s^t B_j^n(v),\ \ b_{ij} \in \mathbb{R}^3,\ \ i = 0 \cdots m,\ \ j = 0 \cdots n.$$

Dabei ist $u \in [o,r]$ und $v \in [s,t]$.
Die Koeffizienten b_{ij} heißen *Bézier-Punkte* und bilden das *Bézier-Netz*.

Eigenschaften von Bézier-Pflastern:

1. Die Fläche $q(u,v)$ liegt in der konvexen Hülle des Bézier-Netzes.
2. Die Bézier-Punkte der Randkurven sind die Randpunkte des Bézier-Netzes:

$$\begin{aligned} q(u,s) &= \sum_{i=0}^{m} {}_{o}^{r}B_i^m(u) \cdot b_{i0}, \\ q(u,t) &= \sum_{i=0}^{m} {}_{o}^{r}B_i^m(u) \cdot b_{in}, \\ q(o,v) &= \sum_{j=0}^{n} {}_{s}^{t}B_j^n(v) \cdot b_{0j}, \\ q(r,v) &= \sum_{j=0}^{n} {}_{s}^{t}B_j^n(v) \cdot b_{mj}. \end{aligned}$$

3. Das Pflaster verläuft durch die Eckpunkte, d.h.

$$q(o,s) = b_{00},\ q(o,t) = b_{0n},\ q(r,s) = b_{m0},\ q(r,t) = b_{mn}.$$

4. Die Ableitungen in den Randpunkten lassen sich aus den Bézier-Punkten berechnen. Es gilt:

$$\frac{\partial}{\partial u} q(u,v)|_{u=o} = \frac{m}{r-o} \sum_{j=0}^{n} {}_{s}^{t}B_j^n(v) \cdot (b_{1j} - b_{0j}).$$

5. Der Twistvektor T_{os} im Eckpunkt mit den Parameterwerten o,s, berechnet sich aus

$$T_{os} = \frac{\partial^2}{\partial u \partial v} q(u,v)|_{u=o,\,v=s} = \frac{mn}{(r-o)(t-s)}(b_{00} - b_{01} + b_{11} - b_{10}).$$

Algorithmen zur Manipulation und Auswertung von Bézier-Kurven übertragen sich direkt auf die Tensorprodukt-Bézier-Pflaster.

zusammengesetzte Bézier-Pflaster (4.7.4.2, S.202)
Um zwei Pflaster

$$q_1(u,v) = \sum_{i=0}^{m}\sum_{j=0}^{n} b_{ij}\, {}_{o_1}^{r_1}B_i^m(u)\, {}_{s_1}^{t_1}B_j^n(v),$$

$$i = 0,\cdots m,\ j = 0,\cdots n,\ (u,v) \in [o_1,r_1] \times [s_1,t_1],$$

und

$$q_2(u,v) = \sum_{i=0}^{m}\sum_{j=0}^{n} c_{ij}\, {}_{o_2}^{r_2}B_i^m(u)\, {}_{s_2}^{t_2}B_j^n(v),$$

$$i = 0,\cdots m,\ j = 0,\cdots n,\ (u,v) \in [o_2,r_2] \times [s_2,t_2],$$

entlang der Randkurve in v-Richtung aneinanderschließen zu können, müssen erstens die beiden Pflaster dieselbe Randkurve in v-Richtung besitzen, d.h. für die Bézier-Punkte gilt

$$b_{mj} = c_{0j},\ j = 0,\cdots,n,$$

und zweitens müssen entlang der Randkurve, d.h. zu jedem festen v, die Ableitungen in u-Richtung übereinstimmen. Diese stimmen genau dann überein, wenn gilt

$$\frac{1}{(o_1 - r_1)}(b_{mj} - b_{m-1,j}) = \frac{1}{(o_2 - r_2)}(c_{1j} - c_{0j}),\quad j = 0,\cdots,n.$$

Dies bedeutet, daß die Polygonsegmente kollinear sein und zudem ein Längenverhältnis $(o_1 - r_1) : (o_2 - r_2)$ besitzen müssen.

B-Spline-Fläche (4.7.4.3, S.203)

Seien $m < o$ und $n < p$ sowie

$$S = (s_0 = \cdots = s_m, \cdots, s_{o+1} = \cdots = s_{m+o+1})$$

und

$$T = (t_0 = \cdots = t_n, \cdots, t_{p+1} = \cdots = t_{p+n+1})$$

schwach monotone Folgen von Knoten und seien

$$N_i^m(u),\ i = 0,\cdots,o, \quad \text{bzw.} \quad N_j^n(v),\ j = 0,\cdots,p,$$

B-Splines über S und T.
Dann werden analog zu den Bézier-Tensorprodukt-Flächen die Tensorprodukt B-Spline-Flächen vom Grad (m,n) definiert durch

$$q : [s_0, s_{m+o+1}] \times [t_0, t_{n+p+1}] \to \mathbb{R}^3$$

mit

$$q(u,v) \longmapsto \sum_{i=0}^{o}\sum_{j=0}^{p} d_{ij} N_i^m(u) N_j^n(v),\ d_{ij} \in \mathbb{R}^3.$$

Die Punkte d_{ij} heißen *de Boor-Punkte* und bilden das *de Boor-Netz*.

Algorithmus von de Boor für B-Spline-Flächen (4.7.4.3, S.204)

Zur Berechnung von Flächenpunkten $q(u_0, v_0)$ kann wieder der de Boor-Algorithmus eingesetzt werden. Dabei wird der Algorithmus zunächst für $u = u_0$ angewandt, d.h. es werden die de Boor-Punkte

$$d_j(u_0) = \sum_{i=0}^{o} d_{ij} N_i^m(u_0),\ j = 0,\cdots,p,$$

berechnet. Danach wird der de Boor-Algorithmus noch einmal mit $v = v_0$ und den Punkten $d_j(u_0)$ durchlaufen:

$$q(u_0, v_0) = \sum_{j=0}^{p} d_j(u_0) N_j^n(v_0).$$

Bernstein-Polynom bezüglich eines Referenzdreiecks (4.8.1, S.205)

Sind drei affin-unabhängige Punkte $R,S,T \in \mathbb{R}^2$ gegeben und bezeichnen $\rho(U)$, $\sigma(U)$, $\tau(U) \in \mathbb{R}$ die baryzentrischen Koordinaten eines Punktes $U \in \mathbb{R}^2$, d.h.

$$U = \rho(U)\cdot R + \sigma(U)\cdot S + \tau(U)\cdot T \in \mathbb{R}^2, \quad \rho(U)+\sigma(U)+\tau(U) = 1,$$

dann heißen die Polynome

$${}_{RST}B^n_{i,j,k}(U) := \begin{pmatrix} n \\ i,j,k \end{pmatrix} \rho(U)^i \sigma(U)^j \tau(U)^k, \quad i+j+k = n,$$

Bernstein-Polynome bezüglich des Referenzdreiecks $\triangle(R,S,T)$.
Dabei ist

$$\begin{pmatrix} n \\ i,j,k \end{pmatrix} := \frac{n!}{i!\,j!\,k!}.$$

An den Dreiecksrändern, d.h. für $\rho(U) = 0$, $\sigma(U) = 0$ oder $\tau(U) = 0$ entarten die verallgemeinerten Bernsteinpolynome zu den gewöhnlichen Bernsteinpolynomen.

Basiseigenschaft der Bernstein-Polynome (4.8.1.1, S.205)

Wie die Bernstein-Polynome eine Basis für den Raum der reellwertigen univariaten Polynome (Polynome in einer Variablen) über einem Intervall bilden, so bilden die bivariaten Bernstein-Polynome (Polynome in zwei Variablen)

$${}_{RST}B^n_{i,j,k},\ i+j+k = n,$$

eine Basis für den Raum aller reellwertigen bivariaten Polynome vom Grad n.

Dreiecksflächen in Bernsteinbasis (4.8.2, S.205)

Gegeben sei ein Referenzdreieck $\triangle(R,S,T)$ und ein bivariates Polynom $q : \mathbb{R}^2 \to \mathbb{R}^3$. Die Darstellung

$$q(U) = \sum_{i+j+k=n} {}_{RST}B^n_{i,j,k}(U)\cdot b_{i,j,k}, \qquad b_{i,j,k} \in \mathbb{R}^3,$$

von q in der Bernstein-Basis bezüglich $\triangle(R,S,T)$ mit Koeffizienten $b_{i,j,k} \in \mathbb{R}^3$ heißt *Bézier-Darstellung* von q bezüglich $\triangle(R,S,T)$.
Die Punkte $b_{i,j,k}$ heißen *Kontroll-* oder *Bézier-Punkte* von q. Sie bilden das *Kontroll-* oder *Bézier-Netz*.

Eigenschaften von Bézier-Dreiecksflächen:

1. Für $U \in \triangle(R,S,T)$ liegt $q(U)$ innerhalb der abgeschlossenen konvexen Hülle des Kontrollpolygons.
2. $q(R) = b_{n,0,0}$, $q(S) = b_{0,n,0}$ und $q(T) = b_{0,0,n}$, d.h. die Fläche verläuft durch die drei Eckpunkte des Bézier-Polygons.
3. Die Fläche ist in den Eckpunkten $q(R)$, $q(S)$ und $q(T)$ tangential an das Kontrollnetz.

4. Die Einschränkung von q auf die Gerade (S,T) ist eine Bézier-Kurve mit den Bézier-Punkten $b_{0,n-k,k}$, d.h. für $U \in (S,T)$ gilt

$$q(U) = \sum_{k=0}^{n} {}^{T}_{S}B^n_k(U) \cdot b_{0,n-k,k}.$$

   Analoges gilt für die Geraden (R,S) und (T,R).

5. Die Kurve ist affin invariant, d.h.

$$\Phi\left(\sum_{i+j+k=n} {}_{RST}B^n_{i,j,k}(U) \cdot b_{i,j,k}\right) = \sum_{i+j+k=n} {}_{RST}B^n_{i,j,k}(U) \cdot \Phi(b_{i,j,k}).$$

Flächenkonstruktion mittels Kurven (4.9, S.207)

Konstruktion von Flächen mittels Interpolation von Kurven:
a) Coons-Pflaster
b) Gordon-Fläche

Coons-Pflaster (4.9.1, S.208)

Konstruktion eines viereckigen Pflaster

$$q(u,v),\ (u,v) \in [0,1] \times [0,1] \subset \mathbb{R}^2,$$

dessen vier parametrisierte Randkurven

$$q(u,0), q(u,1),\ u \in [0,1], \quad q(0,v), q(1,v),\ v \in [0,1]$$

gegeben sind:

Coons-Pflaster:

$$Q(u,v) = Q_1(u,v) + Q_2(u,v) - ((1-u),u) \begin{pmatrix} q(0,0) & q(0,1) \\ q(1,0) & q(1,1) \end{pmatrix} \begin{pmatrix} 1-v \\ v \end{pmatrix}$$

mit

$$\begin{aligned} Q_1(u,v) &= (1-v)q(u,0) + vq(u,1), \quad (u,v) \in [0,1] \times [0,1], \\ Q_2(u,v) &= (1-u)q(0,v) + uq(1,v), \quad (u,v) \in [0,1] \times [0,1]. \end{aligned}$$

.

Verallgemeinertes Coons-Pflaster:

Es lassen sich anstelle linearer Bindefunktionen auch andere Paare von Bindefunktionen $f_1(u), f_2(u)$ und $g_1(v), g_2(v)$ verwenden, z.B. kubische Hermite-Polynome. Diese müssen am Rand ihres Definitionsintervalls $[0,1]$ folgende Bedingungen erfüllen:

$$\begin{array}{lllllll} f_1(0) & = & 1 & \text{und} & f_1(1) & = & 0 \\ f_2(0) & = & 0 & \text{und} & f_2(1) & = & 1 \\ g_1(0) & = & 1 & \text{und} & g_1(1) & = & 0 \\ g_2(0) & = & 0 & \text{und} & g_1(1) & = & 1. \end{array}$$

Bezeichnet man die Matrix der Eckpunkte $\begin{pmatrix} q(0,0) & q(0,1) \\ q(1,0) & q(1,1) \end{pmatrix}$ mit A und wählt sonst dieselben Bezeichnungen wie oben, so läßt sich das verallgemeinerte Coons-Pflaster Q schreiben als

$$Q = Q_1 + Q_2 - (f_1, f_2) A \begin{pmatrix} g_1 \\ g_2 \end{pmatrix}.$$

Durch die Wahl der Funktionen f_1, f_2, g_1, g_2 ist die Interpolationsfläche festgelegt.

Gordon-Pflaster (4.9.2, S.210)

Ausdehnung der Konstruktion von Coons auf Systeme von Flächen:
Ist ein Netz von parametrisierten Kurven

$$q(u_i, v),\ i = 0, \cdots, m, \quad \text{und} \quad q(u, v_j),\ j = 0, \cdots, n,$$

gegeben, so interpoliert eine Gordon-Fläche R dieses Netz mit Hilfe eines Systems von Bindefunktionen

$$g_i(u),\ i = 0, \cdots, m \quad \text{und} \quad h_j(v),\ j = 0, \cdots, n:$$

$$R(u,v) = g_1(u,v) + g_2(u,v) - g_{12}(u,v)$$

mit

$$\begin{aligned} g_1(u,v) &= \sum_{i=0}^{m} q(u_i, v) L_i^m(u), \\ g_2(u,v) &= \sum_{j=0}^{n} q(u, v_j) L_j^n(v). \end{aligned}$$

und

$$g_{12}(u,v) = \sum_{i=0}^{m} \sum_{j=0}^{n} q(u_i, v_j) L_i^m(u) L_j^n(v).$$

.

Lehrziele

Visibilität sowie Beleuchtungsmodelle und Algorithmen gehören thematisch zusammen und liefern die Werkzeuge zur Berechnung photorealistischer Bilder. Während Beleuchtungsmodelle in Kurseinheit 7 ’Einführung in Körpermodelle und Beleuchtungsrechnung’ vorgestellt werden, beschäftigt sich diese Kurseinheit mit dem Thema Visibilität.

Nach dem Studium dieser Lehreinheit sollten Sie über folgendes Wissen verfügen:

- Clipping-Algorithmen und ihre wesentlichen Merkmale;
  - Was geschieht beim Clipping?
  - Wie arbeitet der Cohen-Sutherland-Algorithmus?
  - Welche Probleme treten beim Clipping von Polygonen auf?
  - Wie funktioniert das 3D-Clipping?
  - Was macht der w-Clip?
- Unterschiedliche Lösungsstrategien für das Visibilitätsproblem und ihre Bewertung hinsichtlich erreichbarer Bildqualität;
  - Welche Grundfunktionen werden von den Visibilitätsverfahren angewendet?
  - Wie funktionieren punktorientierte Visibilitätsverfahren?
    - Wie funktioniert der z-Buffer-Algorithmus?
    - Wie können die Fehler des z-Buffers durch zusätzlichen Aufwand beseitigt werden?
    - Was sind die wesentlichen Konzepte des Überlagerungsverfahrens?
  - Wie arbeiten linienorientierte Visibilitätsverfahren?
    - Wie funktioniert der Test von Rasterzeilen?
    - Was macht der Algorithmus von Watkins?
    - Wie arbeiten Verfahren, die Objekträndern und Konturen ermitteln?
  - Was machen regionenorientierte Visibilitätsverfahren?
    - Wie testet man Rasterflächen?
    - Wie arbeitet der Warnock-Algorithmus?
    - Wie testet man Objektflächen?
    - Wie arbeitet der Weiler-Artherton-Algorithmus?

Kapitel 5

Visibilität

Auf dem Weg von der Definition im Anwendungsprogramm bis zur Darstellung auf dem Bildschirm durchläuft ein graphisches Objekt mehrere Stufen in der Bildgenerierungspipeline (vgl. Bild 2.32), in denen über seine Visibilität (Sichtbarkeit) entschieden wird. Die erste Entscheidung fällt bei der Definition des interessierenden Ausschnitts aus der gesamten Bildinformation. An einem Fenster (window) wird die außerhalb liegende Information abgeschnitten („geclippt"). Eine ähnliche Entscheidung fällt bei der Auswahl der im Bildschirmfenster (viewport) darzustellenden Informationen. In der sogenannten Window-Viewport-Transformation werden diese Abbildungsvorgänge zusammengefaßt (siehe Abschnitt 5.2.3 'Viewing-Pipeline (Ausgabe-Darstellungsreihe')). Die hierzu notwendigen Algorithmen werden im Abschnitt 5.1 'Ausschnittsbildung' behandelt.

In der nächsten Stufe wird überprüft, ob das Objekt vom Blickpunkt des Betrachters aus sichtbar ist. Meistens beschränkt man sich in diesem Stadium darauf, die Rückseiten der Objekte zu beseitigen. Dadurch wird der zu behandelnde Datensatz bereits um 50 % reduziert. Im Abschnitt 5.3 'Visibilitätsverfahren' werden aus der Vielzahl der bekannten Algorithmen zur Bestimmung und Beseitigung verdeckter Objekte die für die Praxis wichtigsten vorgestellt.

Hat ein Objekt den Sichtbarkeitstest überstanden, wird über sein Erscheinen auf dem Bildschirm noch durch die Beleuchtungsberechnung entschieden. Nur dann, wenn dieser Bearbeitungsschritt dem Objekt eine sichtbare Farbe zuordnet, wird es in der Szene auch sichtbar sein. Die hierfür notwendigen Verfahren werden im Kurs Graphische Datenverarbeitung II behandelt, wobei zur Ermittlung von Schatten die hier vorgestellten Visibilitätsverfahren mit dem Blickpunkt an der Stelle der Lichtquelle eingesetzt werden können.

5.1 Ausschnittsbildung (Windowing und Clipping)

Begrenzung der auszugebenden Bildinformation

Bei der Ausgabe von Bildern tritt oft das Problem auf, aus der gesamten vorhandenen Bildinformation nur einen Ausschnitt darzustellen. Zu diesem Zweck

wird ein üblicherweise rechteckiges Fenster (Window) definiert, dessen Ränder den interessierenden Bildausschnitt begrenzen. Um ein fehlerfreies Bild zu erhalten, muß die außerhalb des Fensters liegende Bildinformation vor der Bildausgabe abgeschnitten werden (Clipping = Klippen). Wird dies unterlassen, können unerwünschte Effekte auftreten:

5.1.1 Wraparound

Bildfehler

Wird das Fenster (Window) auf die gesamte zur Verfügung stehende Fläche des Ausgabegeräts abgebildet, so erzeugen Bildelemente außerhalb des Fensters möglicherweise einen Überlauf der Koordinatenadressierung des Geräts. Dies führt in der Regel zu Anomalien des Bildes, wie z.B. zum sogenannten *Wraparound*. Die auftretenden Fehler hängen dabei stark vom Prinzip der Berechnung der einzelnen Bildpunkte (z.B. im Vektorgenerator) im Gerät ab. Bild 5.1 zeigt mögliche Wraparound–Fehler.

Sehr oft will man den Bildschirm in verschiedene Darstellungsflächen (Viewport) aufteilen. Ohne Clipping würden die Inhalte der einzelnen Viewports sich gegenseitig beeinflussen, also falsche Bilder erzeugen etc. Deshalb ist in den meisten Anwendungen Clipping unumgänglich.

In der Regel wird das Fenster in Weltkoordinaten spezifiziert und mit “Window” bezeichnet. Entsprechend wird das Ausgabefenster in Bildschirmkoordinaten angegeben und “Viewport” genannt. Die Transformation zwischen beiden heißt Window–Viewport–Transformation.

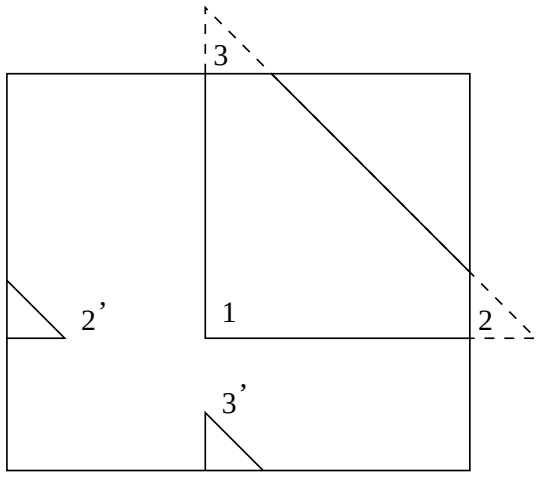


Abbildung 5.1: Wraparound–Fehler

5.1.2 Clipping

Der Vorgang des Clipping teilt die vorhandene Bildinformation in für den Benutzer sichtbare und unsichtbare Bereiche ein. Im zweidimensionalen Fall geschieht

dies durch Vergleich der Bildpunktkoordinaten (X,Y) mit den Fensterkoordinaten $(X_{max}, X_{min}, Y_{max}, Y_{min})$:

$$X_{min} \leq X \leq X_{max}, \quad Y_{min} \leq Y \leq Y_{max} \tag{5.1}$$

Bildpunkte, deren Koordinaten X,Y diese Ungleichungen nicht erfüllen, liegen außerhalb des Fensters und sind deshalb unsichtbar.

Das Clipping von Bildern nach dieser einfachen Vorschrift würde eine Punktzerlegung erfordern, was aus zeitlichen Gründen (Bildregenerierung) nicht in Frage kommt. Stattdessen müssen Clipping–Vorschriften für größere Bildteile — an den Ausgabeelementen (–Primitiva) des Systems orientiert — gefunden werden.

5.1.3 Clipping von Vektoren (Geraden)

Bild 5.2 zeigt unterschiedliche Lagen von Vektoren bezüglich des Fensters. Man erkennt, daß bei Unterteilung einer Geraden in sichtbare und unsichtbare Teile an einem rechteckigen (allgemein konvexen) Fenster nur **ein** sichtbarer Teil entstehen kann.

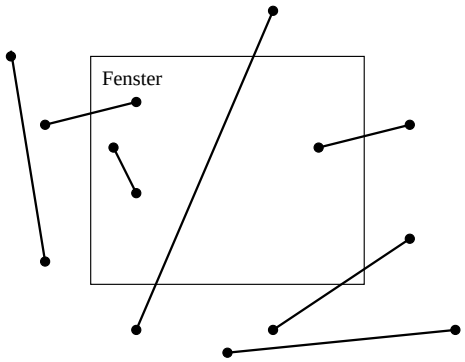


Abbildung 5.2: Vektor–Clipping am Fenster

Vektoren, deren beide Endpunkte oberhalb, unterhalb, rechts oder links des Fensters liegen, sind völlig unsichtbar. Können diese Vektoren einfach aussortiert werden, so ist eine erhebliche Beschleunigung des Clipping zu erwarten. Der Cohen–Sutherland–Algorithmus [SS68] nutzt diese Eigenschaft:

5.1.4 Cohen–Sutherland–Clipping

In der ersten Stufe dieses Algorithmus wird jedem Vektorendpunkt, entsprechend seiner Lage in einer der 9 Regionen, die durch die Fensterbegrenzungen gebildet werden (Bild 5.3), ein 4-bit–Code zugeordnet.

Dabei bedeuten (von rechts nach links):

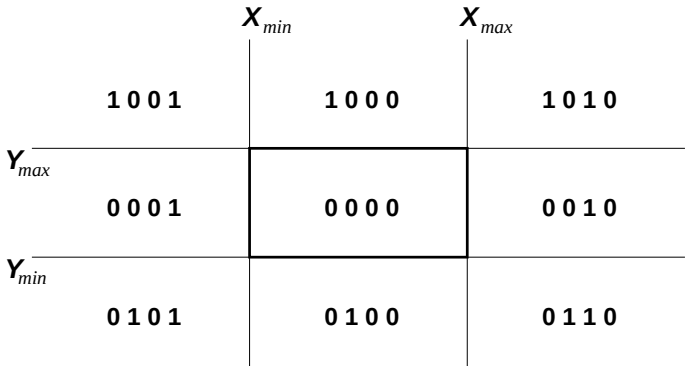


Abbildung 5.3: Codes für Fenster und umgebende Regionen

$$\begin{array}{lcllcl} X < X_{min} & : & \text{Bit } 1 = 1 & \text{für} & X - X_{min} & < & 0 \\ X > X_{max} & : & \text{Bit } 2 = 1 & \text{für} & X_{max} - X & < & 0 \\ Y < Y_{min} & : & \text{Bit } 3 = 1 & \text{für} & Y - Y_{min} & < & 0 \\ Y > Y_{max} & : & \text{Bit } 4 = 1 & \text{für} & Y_{max} - Y & < & 0 \end{array} \tag{5.2}$$

Ein Vektor liegt völlig

- innerhalb des Fensters, wenn der Code für beide Endpunkte Null ist.
- außerhalb des Fensters, wenn der Durchschnitt (logisches UND) der Codes beider Endpunkte verschieden von Null ist.

Wegen dieser Eigenschaft nennt man diesen 4–bit–Code auch „Outcode".

Ist beides nicht der Fall, so wird in der zweiten Stufe des Algorithmus der Schnittpunkt des Vektors mit einer geeigneten Fensterbegrenzung berechnet. Jede Schnittpunktberechnung zerlegt den Vektor in zwei Teile, die wieder nach Stufe 1 behandelt werden. Dabei kann jeweils ein außerhalb des Fensters liegender Teil beseitigt werden. Wird der verbleibende Teil weder als völlig innerhalb noch als völlig außerhalb des Fensters erkannt, so wird Stufe 2 mit einer anderen Fensterbegrenzung durchgeführt.

Die tatsächlich erforderlichen Schnittpunktberechnungen ergeben sich durch Vergleich der Outcodes der Endpunkte. Bei ungleichem Wert in einer Bitstelle wird mit der entsprechenden Fensterbegrenzung geschnitten. Mit Bild 5.4 wird diese Vorgehensweise verdeutlicht.

Vektor *AD*: Die Codes der Endpunktkoordinaten lassen keine Entscheidung zu. Die Schnittpunktberechnung mit der linken Fenstergrenze liefert Punkt *C*. Teil *AC* liegt links vom Fenster und wird deshalb eliminiert. Es verbleibt Vektor *CD*, dessen Endpunkte oberhalb des Fensters liegen und der deshalb ebenfalls als unsichtbar eliminiert werden kann.

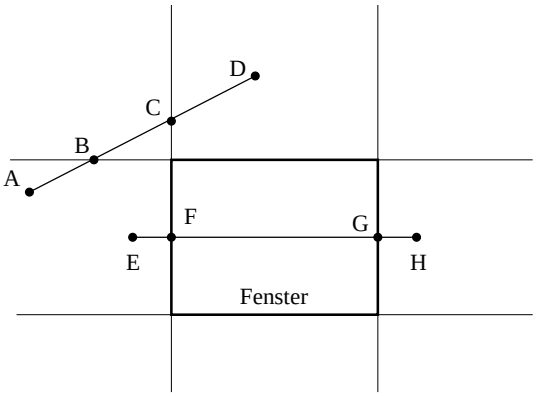


Abbildung 5.4: Zwei Beispiele, die mit Schnittpunktberechnung entschieden werden müssen

Vektor EH: Hier liefert die erste Schnittpunktberechnung den Punkt F. Teil EF wird eliminiert. Der Endpunkttest für FH liefert keine Entscheidung. Die zweite Schnittpunktberechnung liefert den Punkt G. GH wird eliminiert. FG wird als sichtbar erkannt.

Zur Schnittpunktberechnung verwenden Cohen–Sutherland die Geradengleichung in der Form $(P_1(X_1,Y_1),P_2(X_2,Y_2))$:

$$\frac{Y-Y_1}{X-X_1} = \frac{Y_2-Y_1}{X_2-X_1} = m, \tag{5.3}$$

wobei der Fall $X_2 = X_1$ (vertikale Gerade) bzw. $Y_2 = Y_1$ (horizontale Gerade) besonders behandelt wird.

Die Schnittpunktskoordinaten X_{smin}, X_{smax} bzw. Y_{smin}, Y_{smax} ergeben sich dann für die Y–Begrenzungen als

$$X_{smin} = X_1 + (Y_{min} - Y_1)\cdot\frac{1}{m}, \quad X_{smax} = X_1 + (Y_{max} - Y_1)\cdot\frac{1}{m} \tag{5.4}$$

bzw. für die X-Begrenzungen

$$Y_{smin} = Y_1 + (X_{min} - X_1)\cdot m, \quad Y_{smax} = Y_1 + (X_{max} - X_1)\cdot m. \tag{5.5}$$

Die Ergebnisse aus den Berechnungen der Outcodes (vgl. Gleichung 5.2) können verwendet werden. In [YDL84] und [CB78] wird eine Parameterdarstellung der Geraden gewählt:

$$P = P(\lambda) = P_1 + \lambda(P_2 - P_1). \tag{5.6}$$

Der gesuchte Schnittpunkt P(λ_s) muß die Gleichung der entsprechenden Fensterbegrenzung erfüllen mit $0 < \lambda_s < 1$. Auch in diesem Fall lassen sich Ergebnisse der Outcode-Berechnung für die Schnittpunktberechnung verwenden (vgl. Übungsaufgabe 1).

Polygone sind besonders als Begrenzungen von Flächen bei der Rastergraphik wichtig. Ein Clipping–Algorithmus muß diesem Flächencharakter entsprechen

und als Ergebnis des Clippingvorgangs wieder geschlossene Polygone liefern. Dies ist nur durch richtige Einbeziehung von Teilen der Fensterbegrenzung in das „geclippte" Polygon möglich.

Prinzipiell kann diese Aufgabe mit dem oben beschriebenen Vektor–Clipping gelöst werden. Das Polygon wird als Menge von n Einzelvektoren aufgefaßt und jeder Vektor für sich nacheinander an den Fenstergrenzen „geclippt" (vgl. Bild 5.5).

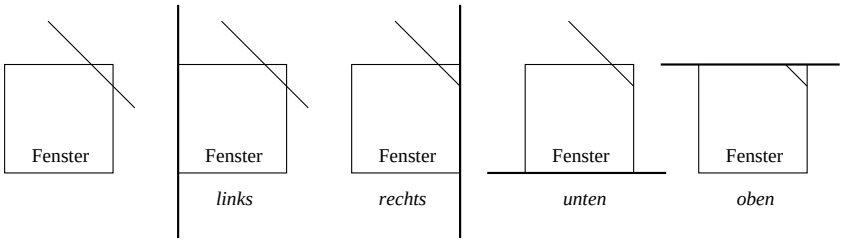


Abbildung 5.5: Polygon–Clipping durch n–faches Vektor–Clipping

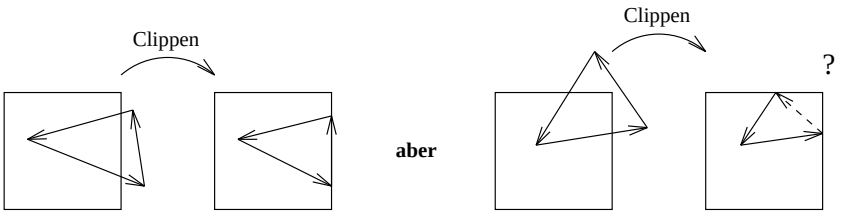


Abbildung 5.6: Einfügen von Fenstergrenzen beim Polygon–Clipping

Verläßt ein Vektor des Polygons das Fenster, so wird der Schnittpunkt mit dem Wiedereintrittspunkt verbunden, also ein neuer Vektor dem Polygon hinzugefügt (vgl. Bild 5.6).

Probleme entstehen, wenn das zu „clippende" Polygon Ecken des Fensters umschließt. Verschiedene Methoden wurden vorgeschlagen, um diese Sonderfälle behandeln zu können. Die einfachste Methode ist die Umkehrung der Schleifenschachtelung im Algorithmus: Das gesamte Polygon (alle Vektoren) wird zunächst an einer Fenstergrenze geclippt, anschließend wird an der nächsten Fenstergrenze geclippt etc. Dies ist die wesentliche Idee des Sutherland–Hodgman–Algorithmus [SH74] (vgl. Bild 5.7).

Die Ergebnisse jedes Zwischenschritts müssen gespeichert werden. Dies kann durch eine rekursive Definition des Algorithmus vermieden werden [SH74]. Eine Hardware–Realisierung besteht dann aus einer Pipeline von vier identischen Clipper–Stufen ohne Zwischenspeicherung [Cla80].

Das Clipping von konkaven Polygonen kann mehrere Einzelpolygone erzeugen, die beim beschriebenen Algorithmus durch Teile der Fenstergrenzen, also entartete Flächen, verbunden sind (vgl. Bild 5.8).

Diese, für manche Anwendungen unerwünschten Verbindungen, können durch

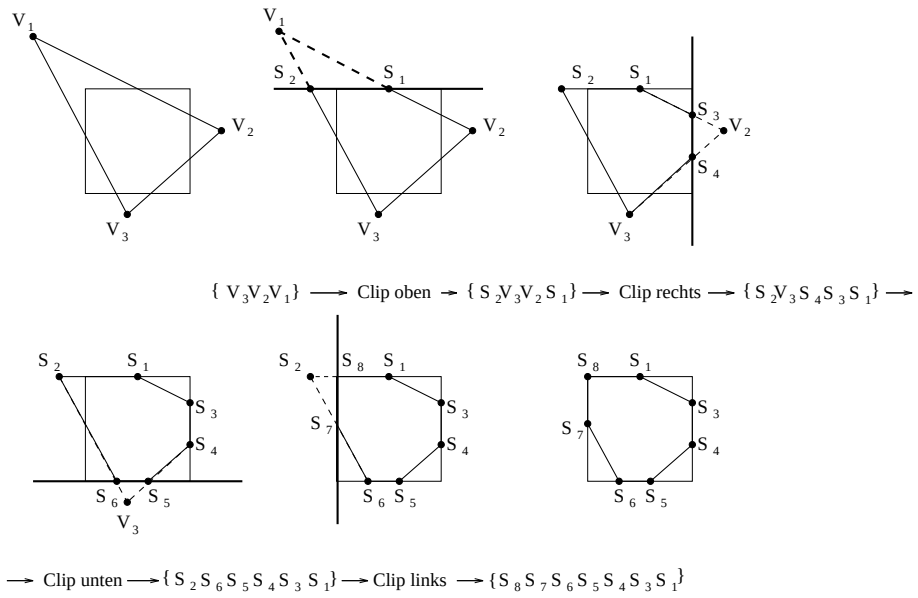


Abbildung 5.7: Prinzip des Sutherland–Hodgman–Clipping

zusätzliche Rechenschritte beseitigt werden. Oder man verwendet den sehr allgemeinen Weiler–Atherton–Algorithmus [WA77], der ein konkaves Polygon gegen ein beliebiges konkaves Fenster „clippt” und diese unerwünschten Verbindungen nicht erzeugt.

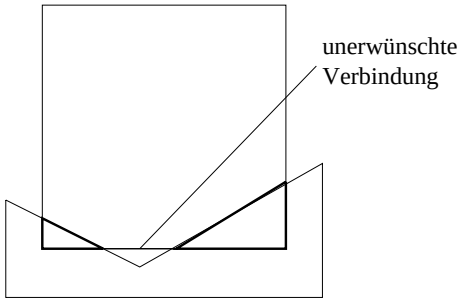


Abbildung 5.8: Clipping konkaver Polygone

Wenn Sie die PDF-Version des Kurses bearbeiten, können Sie jetzt das Interaktionselement ’clipping’ anwählen.

5.2 3D–Clipping

Im räumlichen Fall wird der interessierende Teil der Bildinformation durch ein sogenanntes Sichtvolumen (view volume) begrenzt. Im Fall der Parallelprojektion ist das Sichtvolumen ein in Projektionsrichtung unendlich ausgedehnter Qua-

der, im Fall der perspektivischen Projektion eine Pyramide. Beide Volumina werden aus praktischen Gesichtspunkten durch eine vordere und hintere Ebene begrenzt, vgl. Bilder 5.15 und 5.16 im Abschnitt 5.2.3 'Viewing Pipeline (Ausgabe-Darstellunsgreihe)'. Die oben beschriebenen Verfahren für den ebenen Fall können sinngemäß auf 3D erweitert werden. Die Abfragen der Clipping–Algorithmen werden besonders einfach und müssen nur einmal implementiert werden, wenn an normierten Sichtvolumina "geclippt" wird, wie z.B. bei Parallelprojektion an den sechs Ebenen

$$X = 0\,, \quad X = 1\,, \quad Y = 0\,, \quad Y = 1\,, \quad Z = 0\,, \quad Z = 1$$

und bei perspektivischer Projektion an den sechs Ebenen (vgl. Bild 5.9) des (rechtwinkligen) Pyramidenstumpfs

$$X = Z\,, \quad X = -Z\,, \quad Y = Z\,, \quad Y = -Z\,, \quad Z = Z_{min}\,, \quad Z = Z_{max}.$$

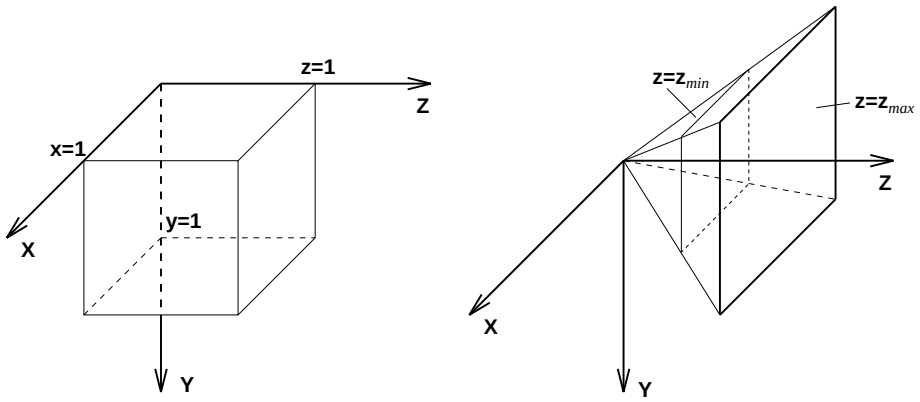


Abbildung 5.9: Beispiele für normierte Sichtvolumen: a) für Parallelprojektion, b) für Perspektive

Die vom Benutzer tatsächlich benötigten Sichtvolumen werden vor dem Clipping durch affine Koordinatentransformationen, die sich in einer 4×4–Matrix zusammenfassen lassen (vgl. Abschnitt 5.2.3 'Viewing Pipeline (Ausgabe-Darstellungsreihe)'), in die normierten Sichtvolumen überführt. Ist das Sichtvolumen ein Einheitswürfel, so ist ein auf 3D erweiterter Cohen–Sutherland–Clipping–Algorithmus unmittelbar zum Clipping dieser Szene geeignet. Ist das Sichtvolumen eine Pyramide, so kann das parametrische Linien-Clipping auf 3D erweitert werden.

Dieses Clipping hat folgende Nachteile:

1) Für parallele und perspektivische Projektionen wird an unterschiedlichen Sichtvolumen geclippt. Man benötigt also verschiedene Algorithmen.

2) Da das Clipping in affinen Koordinaten durchgeführt wird, dürfen bis zum Clipping nur affine Transformationen verwendet werden. Die Perspektive muß deshalb in einem separaten Schritt danach berechnet werden. Das Ziel ist jedoch eine Darstellung, die auch die Perspektive einschließt (siehe Abschnitt 5.2.3 'Viewing Pipeline (Ausgabe-Darstellungsreihe)').

5.2.1 Clipping in homogenen Koordinaten

In Abschnitt 3.3 'Affine und projektive Abbildungen in Matrizenschreibweise' über affine und projektive Abbildungen haben wir gezeigt, wie alle notwendigen geometrischen Transformationen nach einem einheitlichen, zweistufigen Schema in homogenen Koordinaten berechnet werden können. Um für parallele und perspektivische Projektionen gleiche Sichtvolumen zu haben, an denen geclippt wird, bietet sich an, *nach* der Zentralprojektion an einem Einheitswürfel zu clippen. Dabei treten jedoch Probleme auf, die am Beispiel einer perspektivischen Projektion erläutert werden.

Da es bei der perspektivischen Transformation endliche Punkte gibt, die auf unendlich ferne Punkte abgebildet werden und umgekehrt, kann ein Wraparound auftreten. Befindet sich der Augpunkt (Projektionszentrum) bei der perspektivischen Projektion innerhalb der Szene, wie es z.B. beim Durchwandern eines Zimmers der Fall ist, so werden Liniensegmente, deren Anfangs- und Endpunkt auf unterschiedlichen Seiten der Sichtebene liegen, auf zwei Teilsegmente abgebildet. Diese Situation ist in Bild 5.10 dargestellt.

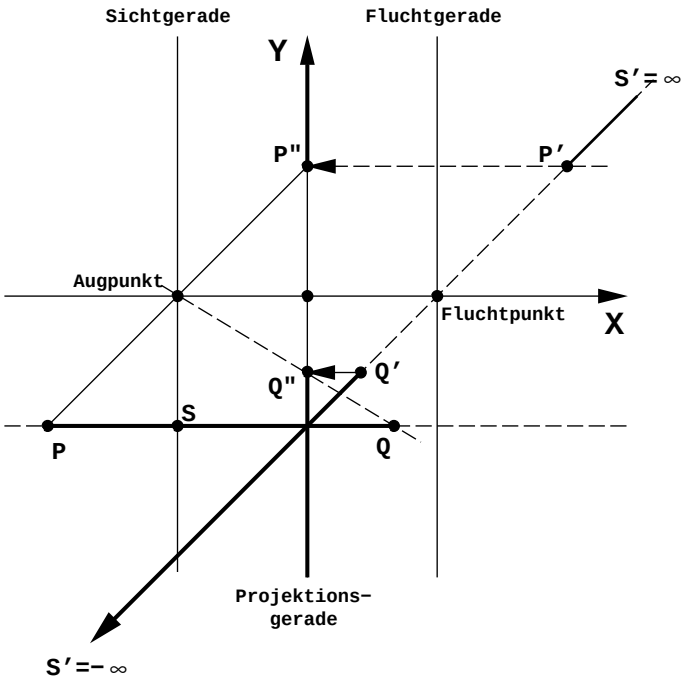


Abbildung 5.10: Perspektivische Transformation von $\overline{PQ}$ auf $\overline{Q'S'}$ und $\overline{S'P'}$
Der hinter dem Auge liegende Teil $\overline{SP}$ erscheint in $\overline{S'P'}$ vor dem Auge (Wraparound). Nach der Parallelprojektion auf die Projektionsgerade $Z = 0$ entstehen zwei Segmente $\overline{Q''Y(-\infty)}$ und $\overline{P''Y(+\infty)}$.

Die Ursache dafür, daß das Liniensegment $\overline{PQ}$ in Bild 5.10 nach der perspektivischen Transformation in zwei Segmente zerfällt, liegt darin begründet, daß es die Sichtgerade, die bei der perspektivischen Transformation auf die unendlich ferne Gerade abgebildet wird (vgl. Abschnitt 3.3.5 'Ebene geometrische Projektionen'), in S schneidet. Dieser Schnittpunkt S wird auf einen unendlich fernen Punkt transformiert. Daher enthält auch das Liniensegment $P'Q'$ einen unendlich fernen

Punkt S'. In homogener Darstellung besitzt dieser Punkt die w-Koordinate Null. Stehen nach der Zentralprojektion (Division durch w) nur noch die affinen Koordinaten der Punkte P'' und Q'' zur Verfügung, so ist es schwierig zu entscheiden, ob das innere Segment ($\overline{P''Q''}$) oder die beiden äußeren Teilsegmente das projizierte Geradensegment beschreiben. Um diese Mehrdeutigkeit zu vermeiden, muß vor der Zentralprojektion auf die Ebene $w = 1$ in homogenen Koordinaten geclippt werden. Dem dreidimensionalen Sichtvolumen entspricht ein vierdimensionales Gebilde, an dem geclippt werden muß. Der w–Clip [Her92] bietet eine elegante Möglichkeit, das Clipping an diesem vierdimensionalen Gebilde zu vermeiden.

5.2.2 Der w – Clip

Beim sogenannten w–Clip [Her92] wird das Wraparound–Problem in euklidischen 4D–Koordinaten beschrieben und gelöst. Wir betrachten diese Methode zunächst im 2D-Fall:

Die zweidimensionale, perspektivische Transformation T_p wird als lineare Transformation im $\mathbb{R}^3$ aufgefaßt. Affine Liniensegmente in der Hyperebene $\Psi : w = 1$ werden auf Liniensegmente in der Hyperebene $\Psi' = T_p \cdot \Psi$ abgebildet. Diese Situation zeigt Bild 5.11 in der xw–Ebene.

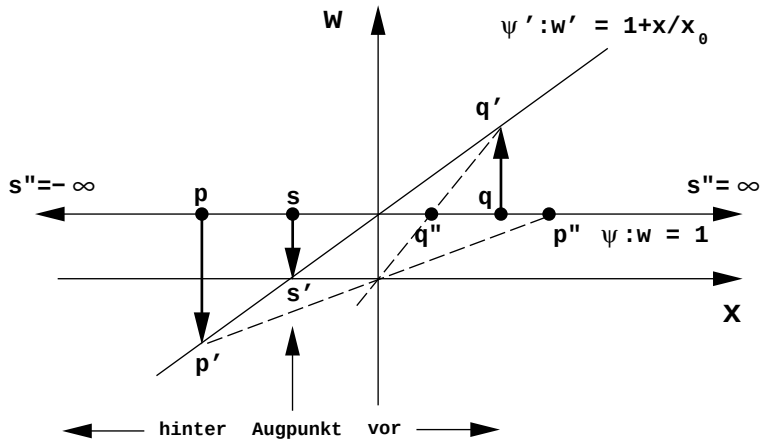


Abbildung 5.11: Bei der perspektivischen Transformation wird die Hyperebene $\Psi : w = 1$ auf die Hyperebene Ψ' abgebildet (durch Scherung, vgl. Gleichung 3.54). Enthält das Liniensegment pq den Punkt s, so enthält $p'q'$ den Punkt s' in der Hyperebene $w = 0$. Die entsprechenden inhomogenen Darstellungen von p'' und q'' ergeben sich durch Zentralprojektion auf die Ebene $w = 1$. Das Segment $p'q'$ wird durch die Zentralprojektion auf die Segmente $s''q''$ und $p''s''$ abgebildet.

Wenn Sie die PDF-Version des Kurses bearbeiten, können Sie jetzt das Interaktionselement 'wclip' anwählen.

Schneidet das transformierte affine Liniensegment die Hyperebene $w = 0$, so zerfällt es bei der Zentralprojektion auf die affine Hyperebene $w = 1$ in zwei Teilsegmente. Beim w–Clip wird daher zunächst ein Clipping an den Ebenen $w = \varepsilon$

und $w = -\varepsilon$ ($\varepsilon > 0$) durchgeführt; damit wird der kritische Bereich nahe $w = 0$ beseitigt (vgl. Abb. 5.12). Dieselben Überlegungen gelten auch für den 3D–Fall.

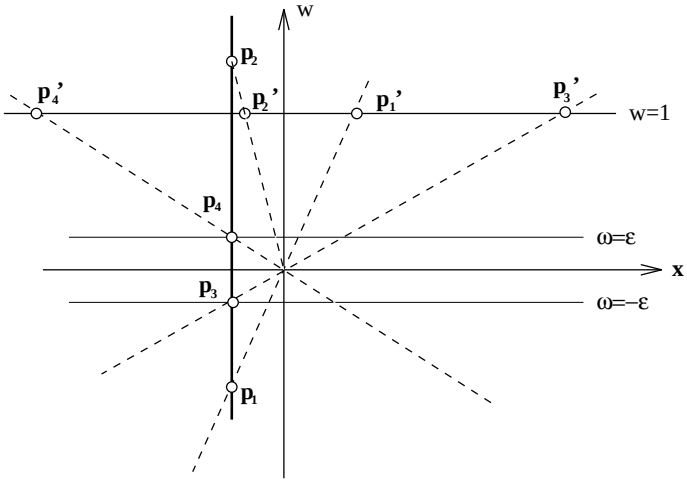


Abbildung 5.12: Das transformierte Liniensegment $\overline{p_1\,p_2}$ wird an den beiden affinen Ebenen $w = \varepsilon$ und $w = -\varepsilon$ geclippt. Man erhält zwei Liniensegmente $\overline{p_1\,p_3}$ bzw. $\overline{p_2\,p_4}$. Bei der Zentralprojektion auf die Ebene $w = 1$ (Übergang auf inhomogene Koordinaten) werden diese Segmente auf $\overline{p'_1\,p'_3}$ bzw. $\overline{p'_2\,p'_4}$ abgebildet.

Jedes der beiden entstehenden Geradensegmente liegt nun entweder ganz in $w > 0$ oder $w < 0$. Nach dem Clipping an den beiden Ebenen $w = \varepsilon$ und $w = -\varepsilon$ wird die Zentralprojektion auf die Ebene $w = 1$ durchgeführt. Das affine Sichtvolumen in Form eines Pyramidenstumpfes wird bei der perspektivischen Transformation auf einen Quader abgebildet, (vgl. Kapitel 3, Abschnitt 3.3.5.5 'Perspektivische Transformation'). Daher kann auf diese Weise auch im Fall der perspektivischen Projektion am Einheitswürfel geclippt werden.

Die Lösung des Wraparound durch den *w*–Clip bedeutet einen zusätzlichen Clippingvorgang. In [AEW90] wurde jedoch gezeigt, wie durch geeignete Faktorisierung der Viewing-Transformation (vgl. Abschnitt 5.2.3 'Viewing Pipeline (Ausgabe-Darstellungsreihe)') der *w*–Clip implizit durch das Clipping an der vorderen und hinteren Begrenzungsfläche des Sichtvolumens erledigt wird.

Das Clipping in homogenen Koordinaten ist also ein sehr allgemein einsetzbares Verfahren, das auch ungewöhnliche (nicht-physikalische), aber in Graphiksystemen spezifizierbare, Situationen, wie gleichzeitiges Sehen nach vorn und hinten oder Objekte im Unendlichen, korrekt bearbeitet. Es sei darauf hingewiesen, daß auch Clipping möglich ist, ohne die homogenen Koordinaten affin zu interpretieren [Kra89].

5.2.3 Viewing Pipeline (Ausgabe-Darstellungsreihe)

Im Bild 5.13 ist der Ablauf einer Viewing Pipeline dargestellt. Die jeweilige Viewing-Pipeline ist abhängig vom verwendeten Graphiksystem, vgl. Kapitel 6 'Graphische Programmierung'.

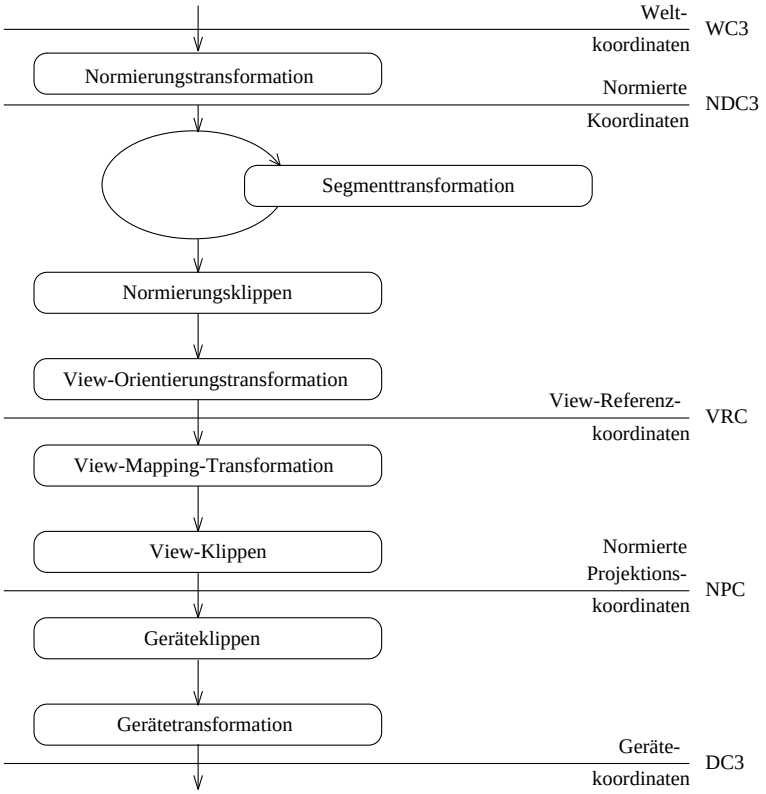


Abbildung 5.13: Beispiel einer Viewing Pipeline

2D- und 3D-Darstellungselemente durchlaufen in diesem Beispiel die gleiche Viewing Pipeline. Die Koordinaten der 2D-Darstellungselemente werden durch Hinzufügen des Wertes $z = 1$ zu 3D-Koordinaten ergänzt. Die 2D-Ausgabe liegt also in der Ebene $z = 1$.

Alle Koordinatensysteme sind Rechtssysteme.

Die Definition der Primitiva erfolgt in 3D-Weltkoordinaten (WC3). Mehrere solcher Weltkoordinatensysteme können als 3D-Fenster (Quader) bestehen. Die Abbildung erfolgt in das normierte Gerätekoordinatensystem (NDC3, normalized device coordinates).

Zur Festlegung von Ansichten (Viewing) gibt es zwei neue Koordinatensysteme, die View-Referenzkoordinaten (VRC) und die normierten Projektionskoordinaten (NPC). Sie dienen besonders zur Einführung eines View-Klippens und der Visibilitätsberechnung. Das Ergebnis der Viewing-Transformation ist in NPC gegeben.

Die abschließende Gerätetransformation (workstation transformation) bildet die NPC in 3D-Gerätekoordinaten (DC3, device coordinates 3) ab.

Die Normierungstransformation wird durch ein 3D-Fenster (window) und ein 3D-Darstellungsfeld (viewport) spezifiziert. Beide Funktionen haben als Parameter u.a. 6 Real-Werte, die die Grenzwerte (X_{min}, X_{max}; Y_{min}, Y_{max}; Z_{min}, Z_{max}) von

achsenparallelen Quadern festlegen.

Wenn Sie die PDF-Version des Kurses bearbeiten, können Sie jetzt das Interaktionselement 'camtrans' anwählen.

Viewing-Transformation

Die *Viewing-Transformation* (Viewing) dient dazu, Ansichten von dreidimensionalen Objekten an einem Arbeitsplatz (workstation) zu definieren. Sie stellt eine Abbildung von normierten Gerätekoordinaten (NDC3) in normierte Projektionskoordinaten (NPC) dar. Sie liefert ebenso die Möglichkeit, eine Projektion der 3D-Objekte auf ein Ausgabegerät durchzuführen. Jedoch wird im Ergebnis die z-Koordinate nicht entfernt. Hierdurch ist es möglich, eine Visibilitätsberechnung anzuschließen (vgl. Abschnitt 5.3 'Visibilitätsverfahren (Hidden-line-Hidden-surface-Algorithmen)').

Das Viewing wird durch zwei Teiltransformationen über je eine 4×4-Matrix definiert: die *Orientierungsmatrix* und die *View-Mapping-Matrix* . Damit werden drei Stufen des Viewings festgelegt (vgl. Bild 5.13): die View-Orientierungstransformation, die View-Mapping-Transformation und das View-Klippen.

Die Orientierungsmatrix definiert als Ergebnis der Transformation ein zusätzliches Koordinatensystem, das VRC (View Reference Coordinate System), in dem die Koordinatenachsen mit U, V, N bezeichnet werden. Dieses Zwischensystem wird dazu benutzt, die Parameter der Projektion zu spezifizieren und in der View-Mapping-Matrix festzulegen. Das Ergebnis dieser Projektionstransformation sind Koordinaten in NPC. Das VRC-System ist wie folgt definiert (vgl. Bild 5.14): Definition des VRC-Systems

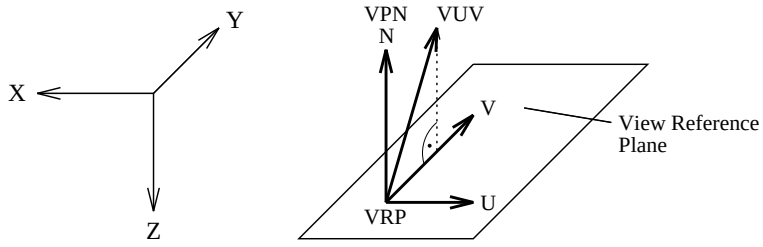


Abbildung 5.14: Definition des VRC-Systems (oft auch UVN-System genannt)

Der View-Referenz-Punkt (VRP) ist der Ursprung des VRC-Systems. Er sollte normalerweise so gewählt werden, daß er in der Nähe der darzustellenden Objekte liegt, wenn eine einfache interaktive Kontrolle der Ansichten gewünscht ist.

Der Normalenvektor der View-Ebene (VPN) geht vom VRP aus und legt die N-Achse des VRC-Systems fest. Diese Achse entspricht der Blickrichtung. Der Vektor bestimmt eine unendliche Menge paralleler Ebenen, von denen eine die View-Ebene ist. Die Ebene, in der der VRP liegt, wird als View-Referenzebene (view reference plane) bezeichnet.

Der VUV erlaubt es, eine Obenrichtung anzugeben, ohne einen rechten Winkel

mit der Blickrichtung einhalten zu müssen. Seine Projektion parallel zu VPN auf die View-Referenzebene legt die V-Achse des VRC-Systems fest.

Die U-Achse wird so gebildet, daß UVN ein Rechtssystem bilden (vgl. Bild 5.14).

Ausgehend von dieser Definition des VRC-Systems können folgende weitere Definitionen getroffen werden:
Das View-Fenster ist ein rechteckiger Bereich in der View-Ebene, der parallel zu den U-, V-Achsen liegt. Die Größe des View-Fensters wird durch die Angaben U_{min}, U_{max}, V_{min} und V_{max} in VRC-Einheiten bestimmt.
Legt man nun durch die Eckpunkte des View-Fensters Geraden, so ergibt sich ein Viewing-Bereich. Diese Geraden verlaufen bei der Parallelprojektion parallel zur Blickrichtung (vgl. Bild 5.15), bei der perspektivischen Projektion durch den Projektions-Referenzpunkt (PRP) (vgl. Bild 5.16).

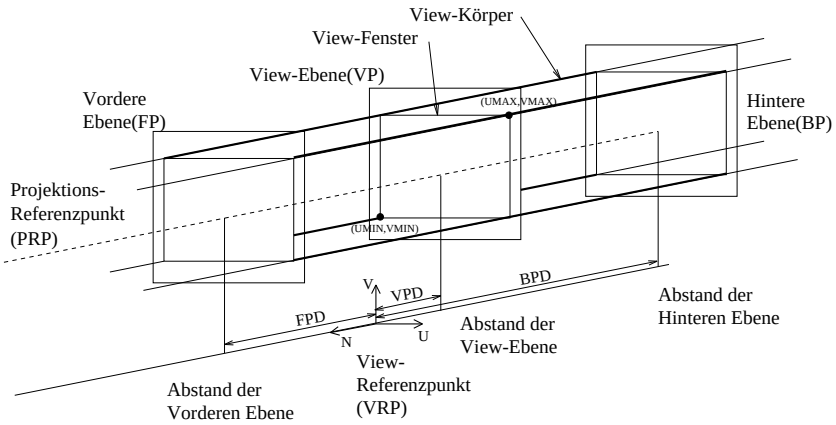


Abbildung 5.15: Viewing-Modell für Parallelprojektion

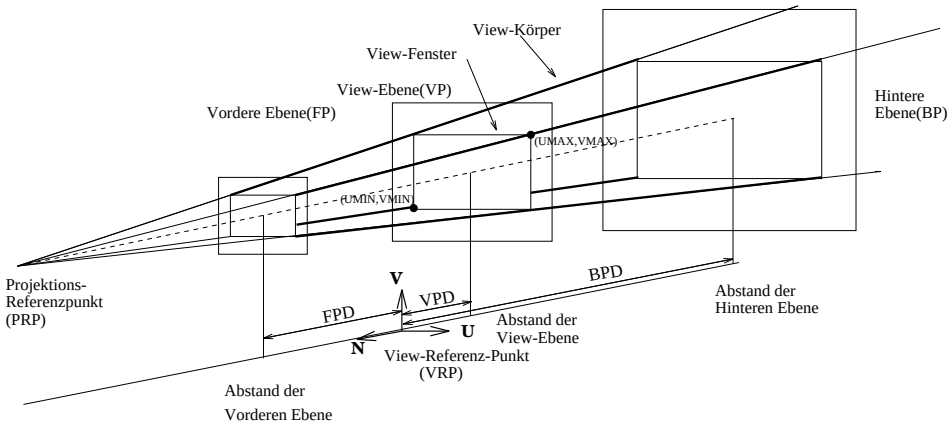


Abbildung 5.16: Viewing-Modell für perspektivische Projektion

Durch die Spezifikation von zwei weiteren Werten wird aus diesem (unendlich ausgedehnten) Viewing-Bereich ein View-Klippkörper (Sichtvolumen = view vol-

ume): der Abstand der vorderen und hinteren Ebene (front/back plane distance, FPD/BPD) vom VRP definiert zwei Ebenen, die parallel zur View-Ebene (und damit auch zur U-V-Ebene) sind. Damit kann die Region im NPC-System festgelegt werden, in der die sichtbaren Daten ausgegeben werden.

Neben den Transformationen wird auch der View-Klippkörper festgelegt (vgl. Bild 5.15).

Berechnung der Orientierungsmatrix

Die Orientierungsmatrix läßt sich folgendermaßen berechnen, wenn VRP, VUV und VPN in WC3 oder NDC3 gegeben sind und VRC ein Rechtssystem sein soll:

$$V = \frac{VPN \times VUV}{|VPN \times VUV|} \quad \text{und} \quad U = \frac{V \times VPN}{|V \times VPN|}.$$

Die Transformation von Weltkoordinaten in das VRC wird dann durch

1) Translation in den Ursprung
2) Rotation, so daß U auf $(1,0,0)$, V auf $(0,1,0)$ und N auf $(0,0,1)$ abgebildet wird

(vgl. Übungsaufgabe 4) ausgeführt.

Einbettung in eine 4×4-Matrix (R') und Multiplikation mit T ergibt die gesuchte Orientierungsmatrix

$$M_{vo} = T \cdot R'.$$

Bei der anschließend durchzuführenden Projektion unterscheidet man zwischen den Projektionsarten "Parallel" (vgl. Bild 5.15) und "Perspektive" (vgl. Bild 5.16). Im folgenden wird die Perspektive behandelt.

Innerhalb des VRC-Systems wird die Projektion definiert. Dazu dient der Parameter *Projektionsreferenzpunkt* (projection reference point) PRP, die Angabe eines *View-Körpers*, der über ein *View-Fenster* (view window) innerhalb der View-Ebene sowie eine hintere und vordere Ebene (back plane, front plane) bestimmt ist, und ein *Projektionsdarstellungsfeld* (projection viewport).

Der Blickpunkt eines imaginären Beobachters liegt im PRP. Die Orientierung der View-Ebene wird, wie oben beschrieben, durch die View-Ebenen-Normale VPN bestimmt. Der Abstand der Ebene vom VRP wird durch den View-Ebenen-Abstand (view plane distance) VPD festgelegt. Die View-Ebene ist immer parallel zur U-V-Ebene (vgl. Bilder 5.15, 5.16).

Geklippt wird am View-Körper. Dieser Körper ist bei perspektivischer Projektion ein Pyramidenstumpf, der durch die Angaben Projektionsreferenzpunkt, View-Fenster, hintere und vordere Ebene festgelegt ist (vgl. Bild 5.16).

Da die Gerade vom PRP zum Zentrum des View-Fensters im allgemeinen nicht parallel zu VPN ist, entsteht eine schiefe Projektion (vgl. Bild 5.17), die durch eine Scherung in eine senkrechte Projektion überführt wird. Dies ist notwendig, weil u.a. die Clipping-Algorithmen mit normierten Sichtvolumina arbeiten.

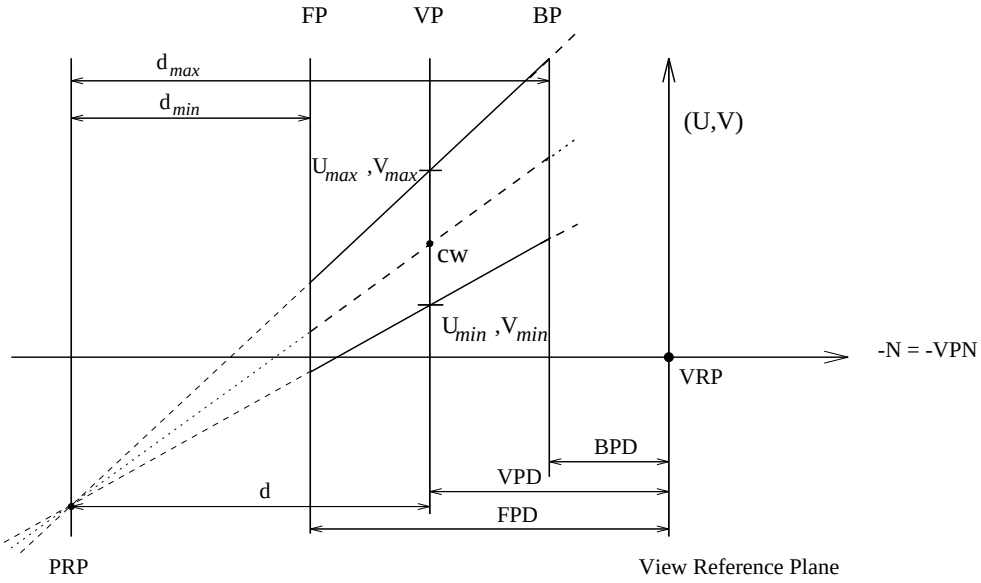


Abbildung 5.17: Die Gerade von PRP zum Zentrum des View-Fensters (CW) definiert die Blickrichtung. Zur Verbesserung der Übersicht wurde der View-Körper zwischen PRP und VRP gelegt.

Berechnung der View-Mapping-Matrix

Die View-Mapping-Matrix wird nun folgendermaßen berechnet, wenn PRP, BPD, VPD und FPD in VRC-Einheiten (U,V,N) gegeben sind, das normierte Sichtvolumen $[0,1]\times[0,1]\times[0,1]$ (in NPC) ist und $d = |VPD - PRP_z|$ gilt (vgl. Bild 5.17). (O.B.d.A. nehmen wir für unsere folgende Betrachtung der Perspektive an, daß $|VPD - BPD| = |FPD - VPD|$.)

Die einzelnen Elementartransformationen sind:

1) Translation von PRP in den VRP (Ursprung des VRC-Koordinatensystems) unter Mitnahme des View-Körpers (Sichtvolumens).

2) Scherung, so daß die Sichtgerade (bestimmt durch PRP und das Zentrum des View-Fensters CW) auf die VPN-Achse abgebildet wird.

3) Skalierung, so daß die Eckpunkte des View-Fensters $U_{min/max}, V_{min/max}$ wegen des Öffnungswinkels von 90° der normierten Sichtpyramide auf $-|d|$, $|d|$ abgebildet werden.

4) Transformation, so daß die View-Plane mit der $U,V,0$-Ebene übereinstimmt.

5) Perspektivische Transformation entlang der $(-N)$-Achse.

6) Skalierung und Translation, so daß das Sichtvolumen mit dem Einheitswürfel übereinstimmt.

Durch diese Transformationen (vgl. auch Abschnitt 3.3 'Affine und perspektivische Abbildungen in Matrizenschreibweise') wird das Sichtvolumen aus Bild 5.16 (View-Körper) auf den Einheitswürfel in Bild 5.18 abgebildet, und die hintere linke untere Ecke des Sichtvolumens liegt im Ursprung.

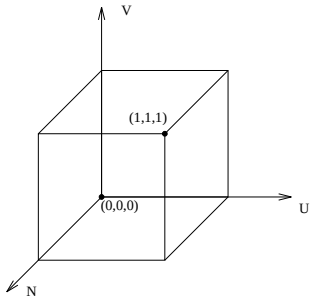


Abbildung 5.18: Sichtvolumen nach Transformationen

Die View-Mapping-Matrix transformiert dabei die VRC-Koordinaten des normierten Projektions-Koordinatensystems in die normierten Projektionskoordinaten NPC. Liefern die Viewing- und Mapping-Matrizen die gewünschten Ansichten, so wird durch Zusammenfassen in einer Matrix direkt von NDC3 in NPC transformiert.

5.3 Visibilitätsverfahren (Hidden–line– Hidden–surface–Algorithmen)

Das Visibilitätsproblem ist so alt wie die GDV. Eine möglichst photorealistische Bildqualität setzt die korrekte Ermittlung der sichtbaren und die Beseitigung der unsichtbaren Bildteile voraus. Der Vollständigkeit halber sei jedoch darauf hingewiesen, daß es häufig genügt, einen möglichst guten dreidimensionalen Eindruck eines Objekts ohne Beseitigung der verdeckten Bildteile zu gewinnen. Hierfür existieren verschiedene Sehhilfen, die in der Literatur unter dem Begriff „depth cueing" zusammengefaßt werden. Dazu gehört der Einsatz der Perspektive, die tiefenabhängige Abschwächung der Helligkeit und der Strichstärke. Je nach Anwendung können diese Sehhilfen kombiniert werden.

Durch die großen Fortschritte auf dem Gebiet der Mikroelektronik und wegen seiner leichten Realisierbarkeit in Hardware hat der einfachste aller Visibilitätsalgorithmen, der sogenannte *z*-Buffer, heute die größte Bedeutung erlangt.

Seine Erweiterung zur korrekten Behandlung des Aliasingproblems und der Transparenz wird deshalb im Detail dargestellt.

Im folgenden werden darüber hinaus aus der Vielzahl bekannter anderer Verfahren einige typische ausgewählt und exemplarisch vorgestellt. Dabei wird nicht die

oft vorgenommene Einteilung in Objekt– und Bildraumverfahren benutzt, weil sie keine prinzipiellen algorithmischen Merkmale, sondern nur die Rechengenauigkeit unterscheidet: *Objektraumverfahren* arbeiten mit Maschinengenauigkeit, während *Bildraumverfahren* sich auf die Darstellungsgenauigkeit des Ausgabegeräts beschänken. (Daraus ergibt sich natürlich Geräteunabhängigkeit der Objektraumverfahren bzw. Geräteabhängigkeit der Bildraumverfahren.) Vielmehr wird hier nach den im Algorithmus auf Verdeckung getesteten Objekten, d.h. Punkt, Linie, Fläche klassifiziert, und es werden für jede Klasse zwei Beispiele angegeben. Eine weitere Unterscheidung könnte dann mit den verdeckenden Objekten eingeführt werden, wobei die Dimension der verdeckten Objekte immer kleiner gleich der Dimension der verdeckenden Objekte sein muß (z.B. Linien verdecken Linien und Punkte, Flächen verdecken Flächen, Linien und Punkte).

Visibilitätsverfahren wenden die in den folgenden Unterkapiteln beschriebenen Grundfunktionen an.

5.3.1 Perspektivische Transformation

5.3.1.1 Vorverarbeitung für alle Verfahren

Verdeckung bzw. Unsichtbarkeit entsteht immer dann, wenn mehrere Objekte bei der Abbildung von 3D nach 2D die gleichen Bildschirmkoordinaten aufweisen (*Projektionsäquivalenz*), also vom gleichen Sehstrahl getroffen werden (vgl. Bild 5.19).

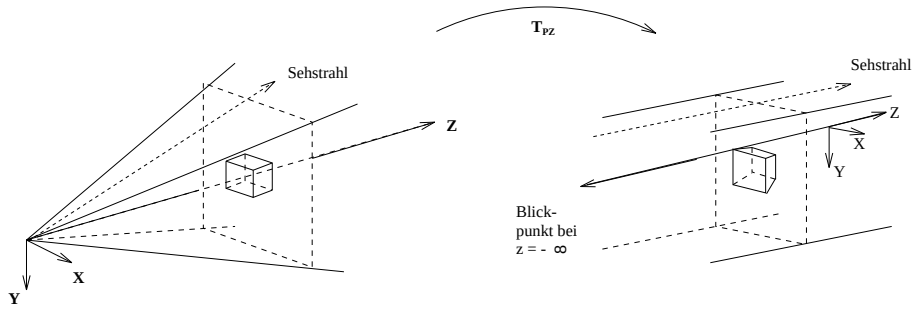


Abbildung 5.19: Perspektivische Transformation

Sichtbar ist jeweils der dem Auge am nächsten liegende Punkt. Ist dieses Objekt durchsichtig (transparent), wird der dahinterliegende Punkt auch sichtbar, usw. Um die Feststellung der Projektionsäquivalenz wesentlich zu vereinfachen, wird die zu untersuchende Szene zuvor perspektivisch transformiert. Dabei wird der in Abschnitt 3.3.5 ’Ebene geometrische Projektionen’ und Bild 3.26 gezeigte Sachverhalt ausgenutzt, daß die perspektivische *Transformation* mit anschließender Parallelprojektion äquivalent der perspektivischen *Projektion* ist, d.h. Punkte mit gleichen x, y–Koordinaten liegen auf demselben Sehstrahl. Da die perspektivische Transformation die Tiefenrelation nicht verändert, reduziert sich die Bestimmung des sichtbaren Punktes auf einen Tiefenvergleich, d.h. bei der Projektion

auf die x,y–Ebene müssen dann nur die z–Werte ermittelt und verglichen werden. Bild 5.19 verdeutlicht diese Vorgehensweise. Im folgenden wird angenommen, daß die Blickrichtung mit der positiven z-Achse übereinstimmt. Der Blickpunkt liegt dann bei $z = -\infty$ (Parallelprojektion).

5.3.2 Beseitigung der Rückseiten (Back face culling)

Rückseiten von undurchsichtigen Körpern sind definitionsgemäß unsichtbar. Da sie etwa die Hälfte der in der Szene insgesamt vorkommenden Flächen ausmachen können, werden sie vor Beginn des eigentlichen Visibilitätsalgorithmus festgestellt und von der weiteren Betrachtung ausgeschlossen. Rückseiten sind dadurch

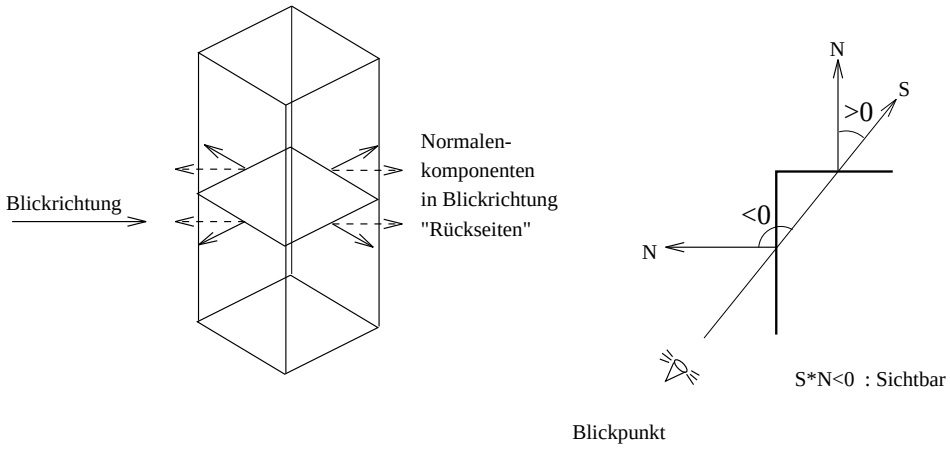


Abbildung 5.20: Ermittlung der Rückseiten

gekennzeichnet, daß ihr Normalenvektor eine positive z–Komponente aufweist. Besteht die Szene nur aus einem konvexen Körper, dann ist mit der Bestimmung der Rückseiten das Visibilitätsproblem bereits gelöst. Bei einem konvexen Polyeder ist das besonders einfach (vgl. Bild 5.20).

Wurde die perspektivische Transformation nicht durchgeführt, so kann aus dem Vorzeichen des Skalarprodukts <Sehstrahl, Normale> die Rückseite ermittelt werden (vgl. Bild 5.20).

5.3.3 Punktklassifizierung

Mit diesem Test wird in der x,y–Ebene untersucht, ob ein Punkt innerhalb oder außerhalb eines geschlossenen Polygons liegt. Vom Testpunkt ausgehend wird die Anzahl der Schnittpunkte eines Strahls (zweckmäßigerweise parallel zur x– oder y–Achse) mit dem Polygon ermittelt. Ist die Anzahl ungerade, liegt der Testpunkt innerhalb, sonst außerhalb des Polygons (vgl. Bild 5.21).

Häufig liegt eine Szene trianguliert vor. Der Test kann dann sehr einfach mit den

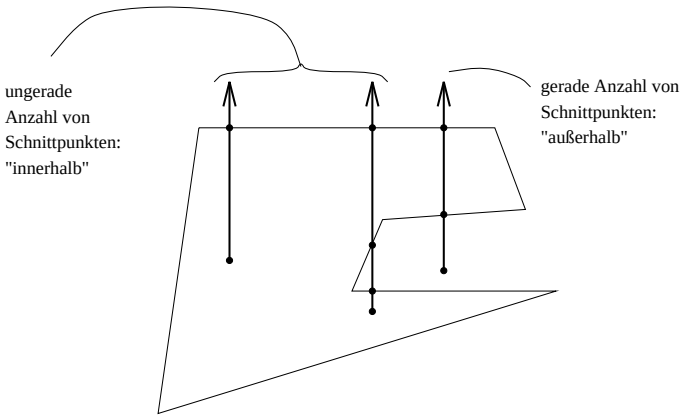


Abbildung 5.21: Innere und äußere Punkte

in Abschnitt 3.1.2 'Affine Unterräume' eingeführten baryzentrischen Koordinaten durchgeführt werden (vgl. Übungsaufgabe 2).

5.3.4 Min/max–Test

Verdeckung zwischen Flächen, z.B. Polygonen, kann nur dann auftreten, wenn ihre Projektionen sich in der x, y–Ebene überlappen. Mit dem Min/max–Test kann man durch einfachen Vergleich feststellen, ob sich zwei Flächen nicht überlappen und auf diese Weise kompliziertere Tests vermeiden. Ist z.B. der kleinste x–Wert eines Polygons größer als der größte x–Wert eines anderen, so kann keine Überlappung auftreten. Dasselbe Argument gilt für die y–Koordinate. Fällt der Min/max–Test negativ aus, muß trotzdem nicht notwendigerweise eine Überlappung vorhanden sein (vgl. Bild 5.22). Dies muß durch Berechnung der Schnittpunkte zwischen den einzelnen Polygonkanten festgestellt werden.

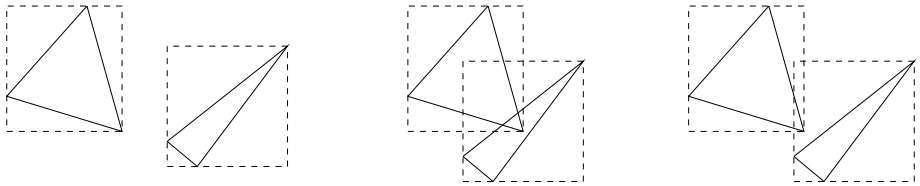


Abbildung 5.22: Min/max–Test

5.3.5 Punktorientierte Visibilitätsverfahren

Bei diesen Algorithmen werden einzelne Punkte auf Verdeckung bzw. Sichtbarkeit untersucht. Sie stellen einerseits die einfachste Art der Visibilitätsrechnung dar, benötigen andererseits aber die größte Anzahl von Tests. Punkttests liefern die Sichtbarkeit explizit, während Linien– und Flächentests die Sichtbarkeit nur

implizit aus wenigen Punkttests und unter Ausnutzung von Nachbarschaftsbeziehungen (Kohärenz) auf den darzustellenden Objekten gewinnen.

Punktorientierte Verfahren werden zweckmäßigerweise für die Bildausgabe auf Rastergeräten eingesetzt, weil dann Testelement und Ausgabeelement übereinstimmen. Der bekannteste Vertreter dieser Klasse ist der sogenannte *z*–Buffer–Algorithmus.

5.3.5.1 z–Buffer–Algorithmus (Tiefenspeicher)

Der Tiefenspeicher wurde bereits 1974 [Str74] vorgeschlagen, aber aus Kostengründen damals nicht realisiert. Das Prinzip dieses Algorithmus ist denkbar einfach: Neben dem Grau(Farb-)wert wird für jeden Bildpunkt auch ein *z*–Wert gespeichert. Der Bildspeicher wird mit der Hintergrundfarbe und der *z*–Speicher dementsprechend mit dem maximal darstellbaren *z*–Wert initialisiert. Nun werden alle Objekte der Szene nacheinander mit der Bildauflösung abgetastet (gerastert oder scankonvertiert). Eine besondere Reihenfolge ist nicht erforderlich. Für jeden Punkt (x,y) des Bildschirmrasters und für jedes Objekt durchläuft der Algorithmus folgende Schritte:

1. Berechne $z(x,y)$.
2. Ist $z(x,y)$ kleiner als der unter (x,y) bereits gespeicherte Wert, dann schreibe $z(x,y)$ in den *z*–Speicher und den zugehörigen Grau(Farb-)wert an der Stelle (x,y) in den Bildspeicher.

Nachdem alle Objekte auf diese Weise bearbeitet wurden, ist das Visibilitätsproblem gelöst, und im Bildspeicher steht das gewünschte Bild.

Der Vorteil des *z*–Buffers, Bildpunkte in beliebiger Reihenfolge erzeugen zu können, begründet auch seinen größten Nachteil: exakte Glättung (Filterung, siehe Graphische Datenverarbeitung II, Kapitel 5, 'Grundlagen der Bildverarbeitung', Antialiasing, vgl. Abschnitt 2.5.4 'Glätten von Polygonkanten') und Transparenz sind prinzipiell nicht realisierbar, weil nur ein Objekt pro Bildpunkt gespeichert wird. Durch Erhöhung der Speicherauflösung (Supersampling) gegenüber der Bildschirmauflösung lassen sich die Abtastfehler durch Nachfilterung (Mittelung) stark reduzieren, jedoch nicht völlig beseitigen.

Dies erhöht allerdings den ohnehin großen Speicheraufwand. Prinzipiell kann der Aufwand für den Tiefenspeicher dadurch reduziert werden, daß das Bild in Teilbilder zerlegt und ein entsprechend kleinerer Tiefenspeicher nacheinander für jedes Teilbild genutzt wird.

Ein wichtiger Sonderfall entsteht, wenn der *z*–Buffer nur noch eine Bildzeile umfaßt. Dies führt zu den rasterlinienorientierten Verfahren.

Ein weiteres Problem entsteht durch die beschränkte Genauigkeit des *z*–Buffers. Insbesondere bei in der Tiefe weit ausgedehnten Szenen mit starker Perspektive

passiert es häufig, daß getrennte Objekte denselben *z*–Wert erhalten. Eine Verkleinerung, Änderung der Perspektive oder andere Plazierung des Sichtvolumens kann hier in beschränktem Umfang helfen.

Die Vorteile des *z*–Buffers überwiegen jedoch die Nachteile:

- Jede beliebige Szene kann, unabhängig von der Komplexität der Einzelobjekte, behandelt werden.
- In einer fertigen Szene können nachträglich Objekte eingefügt werden, sofern deren *z*–Werte zur Verfügung stehen oder berechnet werden können. Dies ist besonders für die Kombination mit Kameraaufnahmen interessant.
- Spezielle Objekte, wie z.B. ein 3D–Cursor, können in der Szene mit korrekter Verdeckung dargestellt werden.
- Der *z*–Buffer kann für spezielle Aufgaben in aufwendigeren Verfahren eingesetzt werden. Beispielsweise kann die erste Stufe eines Raytracers oder die Auswertung des Hemicubes beim Radiosity–Verfahren beschleunigt werden (siehe Kapitel 7 ’Einführung in Körpermodelle und Beleuchtungsrechnung’ oder Graphische Datenverarbeitung II, Kapitel 3 ’Globale Beleuchtungsmodelle’).

Wenn Sie die PDF-Version des Kurses bearbeiten, können Sie jetzt das Interaktionselement ’zbuffer’ anwählen.

Darüber hinaus können die Fehler des *z*–Buffers durch zusätzlichen Aufwand leicht beseitigt werden. Dies soll im folgenden etwas genauer besprochen werden.

5.3.5.2 Der exakte A–Buffer

In Abschnitt 2.5.4 wurde bereits gezeigt, daß das Aliasing durch Berücksichtigen des von einem Objekt überdeckten Anteils der Pixelgesamtfläche beseitigt werden kann. Wird die Fläche des Pixels mit A und die durch das Objekt überdeckte Fläche mit A_i bezeichnet, so berechnet sich der korrekte Grau– oder Farbwert (vgl. Graphische Datenverarbeitung II) des Pixels zu

$$\begin{aligned} I &= \tfrac{1}{A}\left[I_i \cdot A_i + I_0(\underbrace{A - A_i}_{A_0})\right] \\ &= I_i \cdot \alpha_i + I_0 \cdot \alpha_0, \quad \alpha_0 + \alpha_i = 1, \end{aligned} \tag{5.7}$$

wenn durch I_0, A_0 der Einfluß des Hintergrundes erfaßt wird. Diese Vorgehensweise wird jetzt auf *n* Objekte, die das Pixel teilweise überdecken, auf sogenannte Fragmente, erweitert:

$$I = \sum_{i=0}^{n} I_i \cdot \alpha_i \quad \text{mit} \quad \sum_{i=0}^{n} \alpha_i = 1. \tag{5.8}$$

Wird diese Gleichung sequentiell ausgewertet, entsteht das sogenannte α–Blending [PD84]. Dabei ist zu beachten, daß die A_i sichtbare Flächen sind. (Gleichung 5.8 beschreibt die Funktion eines Boxfilters (vgl. Graphische Datenverarbeitung II, Kapitel 5, Abschnitt 5.3.6 'Abtasttheorem und Bildfilterung'). Die nach Abschnitt 2.5.4 'Glätten von Polygonkanten' gewonnenen Subpixelmasken für einzelne Fragmente müssen auf gegenseitige Verdeckung überprüft werden. Das Visibilitätsproblem wird also auf Subpixelebene gelöst.

Eine Approximation dieses exakten Verfahrens ist der A–Buffer. Fragmente werden dort für jedes Pixel in einer nach Z sortierten Liste (front-to-back) gesammelt. Für jedes Fragment werden Z_{min}, Z_{max} sowie eine Subpixelmaske gespeichert, die die vom Fragment überdeckten Subpixel angibt (vgl. Abschnitt 2.5.4 'Glätten von Polygonkanten'). Bei der Abarbeitung der Fragmente wird durch die Sortierung ein dem z–Buffer ähnliches Verhalten auf Subpixelebene realisiert. Da jedoch nicht für jedes Subpixel der Z–Wert, sondern nur Z_{min} und Z_{max} für das ganze Fragment (alles Subpixel der Subpixelmaske) zur Verfügung stehen, können Fragmente, deren Z–Bereiche sich überlappen, nur näherungsweise behandelt werden. Dabei wird grundsätzlich angenommen, daß sich solche Objekte, wie in Bild 5.23 gezeigt, durchdringen.

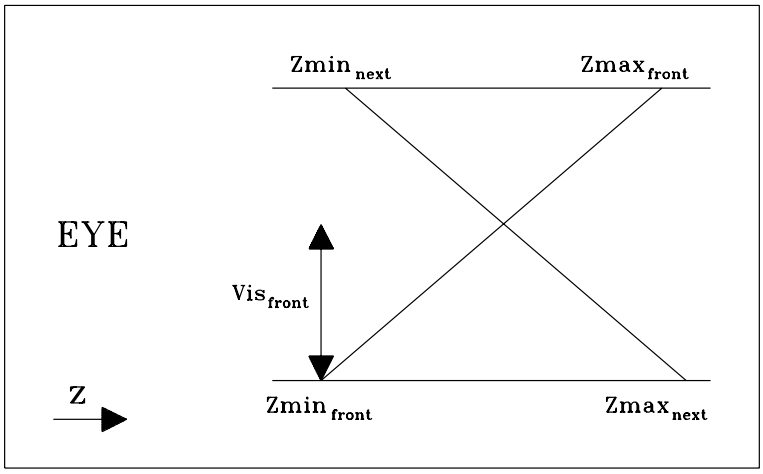


Abbildung 5.23: Sichtbares Fragment nach [Car89]

Die sichtbare Fläche Vis_{front} des vorderen Fragments wird berechnet nach [Car89]

$$Vis_{front} = \frac{Z_{max_{next}} - Z_{min_{front}}}{(Z_{max} - Z_{min})_{front} + (Z_{max} - Z_{min})_{next}}. \tag{5.9}$$

Die Annahmen für diese Methode werden selten erfüllt. Ein Beispiel für die dadurch entstehenden Fehler zeigt die Situation in Bild 5.24, die zur gleichen Ausgabe wie in Bild 5.23 führt, obwohl das vordere Fragment („front") das ganze Pixel überdeckt.

Diese A–Buffer–Lösung ist also nicht besonders wirkungsvoll.

Beim z–Buffer steht sogar nur der Z–Wert im Pixelmittelpunkt zur Verfügung.

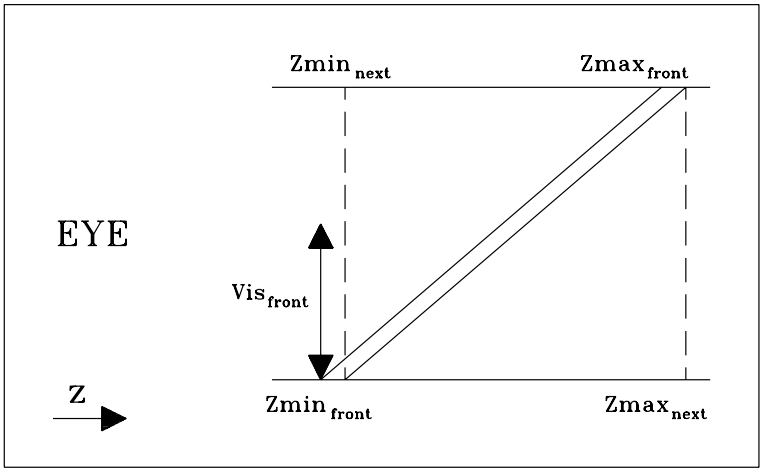


Abbildung 5.24: Front–Fragment überdeckt ganzes Pixel

Die dabei entstehenden Fehler illustrieren die Bilder 5.25 bis 5.27. Dabei wird angenommen, daß Subpixelmasken wie beim A–Buffer vorhanden sind. Die gestrichelten Fragmente werden nicht berücksichtigt, obwohl sie sichtbar sind.

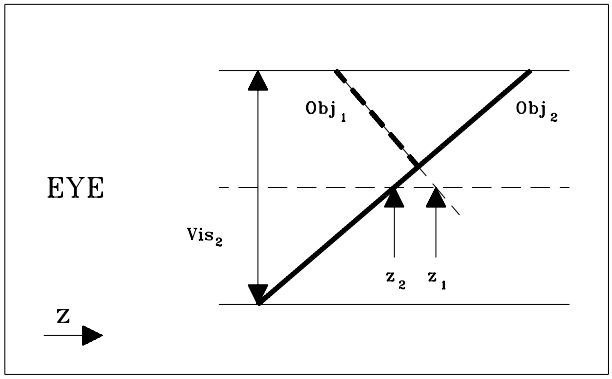


Abbildung 5.25: Objekt 1 verschwindet, Objekt 2 definiert das Pixel

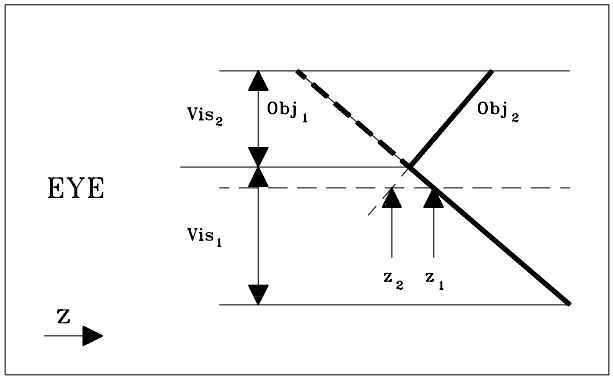


Abbildung 5.26: Objekt 2 definiert fast die Hälfte des Pixels

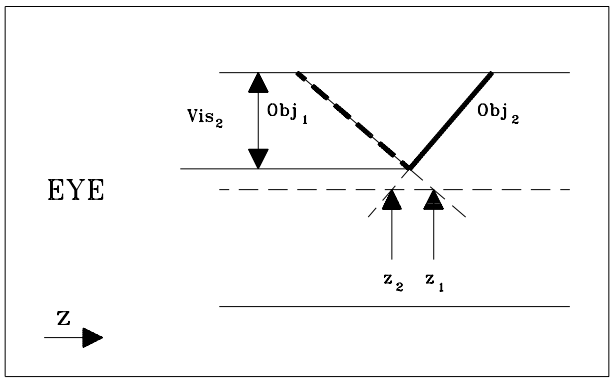


Abbildung 5.27: Objekt 1 verschwindet, Objekt 2 definiert das Pixel wie in Bild 5.20

Diese Probleme können mit einer Prioritätsmaske P im sogenannten EXACT–Algorithmus gelöst werden (vgl. Bild 5.28).

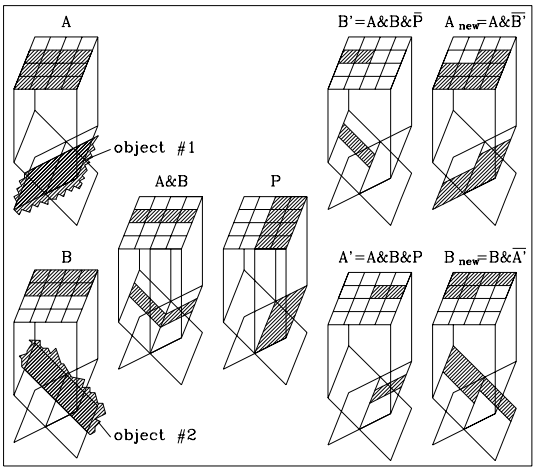


Abbildung 5.28: Wirkung der Prioritätsmaske P auf die Subpixelmaske A und B

P gibt an, in welchem Gebiet des Pixels das Objekt 1 vor Objekt 2 liegt. Das dem Subpixel entsprechende Bit ist dann 1 gesetzt. Aus den ursprünglichen Subpixelmasken A und B der Objekte 1 und 2 können nun durch logische Verknüpfungen mit P die sichtbaren Masken A_{new} und B_{new} gewonnen werden:

$$\begin{aligned} A_{new} &= A\&\overline{A\&B\&\overline{P}}, \\ B_{new} &= B\&\overline{A\&B\&P}. \end{aligned} \tag{5.10}$$

Die Berechnung der Prioritätsmaske beruht auf der Annahme, daß innerhalb des Pixels die Fragmente in guter Näherung durch Ebenen approximiert werden können, deren Steigungen in x– und y–Richtung bekannt und durch

$$dz_x = \frac{\partial z(x,y)}{\partial x} \quad \text{bzw.} \quad dz_y = \frac{\partial z(x,y)}{\partial y} \tag{5.11}$$

definiert sind.

Die Z–Werte $z(0,0)$ im Pixelmittelpunkt sind ebenfalls bekannt (vgl. Bild 5.29).

$$\begin{aligned} z(0,0) &= z_1(0,0) - z_2(0,0) \\ dz_x &= dz_{1,x} - dz_{2,x} \\ dz_y &= dz_{1,y} - dz_{2,y} \end{aligned} \tag{5.12}$$

Die Gleichungen (5.12) beschreiben dann eine Ebene (mit dem Koordinatenursprung im Pixelmittelpunkt)

$$z(x,y) = z(0,0) + x \cdot dz_x + y \cdot dz_y \tag{5.13}$$

mit der Eigenschaft

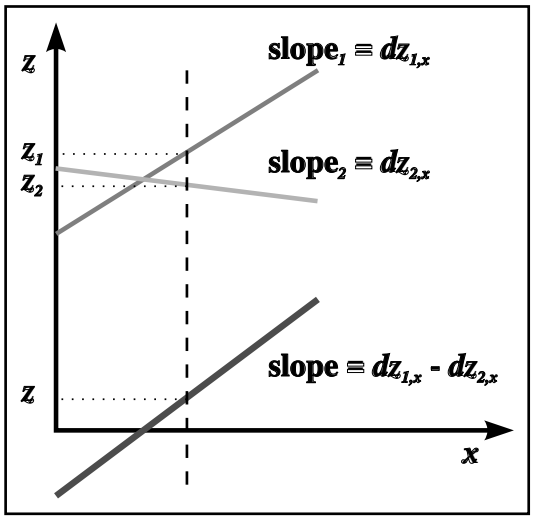


Abbildung 5.29: Berechnung der P–Maske

$z(x,y) > 0$: Objekt 1 vor Objekt 2,

$z(x,y) = 0$: Objekt 1 gleich Objekt 2 (Schnittkante),

$z(x,y) < 0$: Objekt 2 vor Objekt 1.

$z(x,y) = 0$ entspricht der in Abschnitt 2.5.4 'Glätten von Polygonkanten' beschriebenen Kantenfunktion $E(x,y)$. Nach der Normierung mit $|dz_x| + |dz_y|$ kann die P–Maske aus der Tabelle für die Subpixelmasken entnommen werden. Sind in der z–sortierten Liste mehr als zwei Fragmente für ein Pixel abgespeichert, müssen unter Umständen mehrere Prioritätsmasken nacheinander berechnet werden. Allerdings wird man versuchen, dies auf die absolut notwendigen Fälle zu beschränken, die gekennzeichnet sind durch

$$A \& B \neq 0 \tag{5.14}$$

$$|z_1(0,0) - z_2(0,0)| < (|dz_{1,x} - dz_{2,x}| + |dz_{1,y} - dz_{2,y}|)/2. \tag{5.15}$$

Mit der ersten Bedingung wird überprüft, ob die Subpixelmasken sich überhaupt überlappen, mit der zweiten, ob ein Schnitt innerhalb des Pixels stattfindet. Der zweite Test ist nur erforderlich, wenn der erste eine Überlappung anzeigt. Auf die Situation in Bild 5.24 angewandt, wäre P vollständig mit 1 oder 0 besetzt. Ein Beispiel für die Leistungsfähigkeit des vorgestellten EXACT–Algorithmus zeigt Bild 5.30.

Transparente Fragmente werden im EXACT–Algorithmus in einen verdeckenden und in einen verdeckten Teil zerlegt. Der Grau– oder Farbwert des verdeckenden Teils I_{front} wird anschließend mit dem entsprechenden Wert des von ihm verdeckten Fragments I_{next} unter Berücksichtigung des Transparenzfaktors t, $0 \leq t \leq 1$, (vgl. Graphische Datenverarbeitung II, Kapitel 7 'Texturen', Abschnitt 7.1.4 'Beeinflussung der Beleuchtungsrechnung') berechnet:

$$I_{ges} = (1-t) \cdot I_{front} + t \cdot I_{next}. \tag{5.16}$$

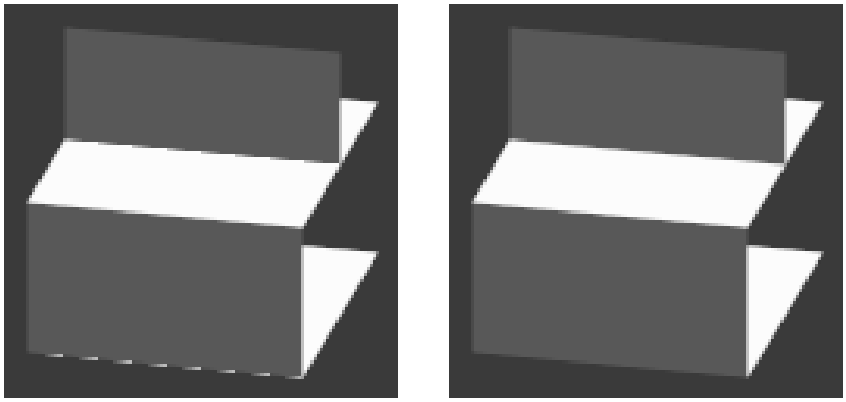


Abbildung 5.30: Links: Standard *z*–Buffer, rechts: EXACT–Algorithmus

Der EXACT–Algorithmus eignet sich für Hardware– und Softwareimplementierungen. Seine besondere Bedeutung liegt jedoch darin, daß er die Basis für eine neue Architektur von Hochleistungsgraphiksystemen bildet, deren Leistungsfähigkeit über einen weiten Bereich linear mit der Anzahl eingesetzter Prozessoren steigt [Sch91].

Im folgenden wird ein weiterer punktorientierter Algorithmus beschrieben, der nicht am Punktraster des Ausgabegeräts, sondern an Punkten der Objektdefinition orientiert ist.

5.3.5.3 Überlagerungsverfahren

Dieses historische Verfahren [Enc75] aus der Zeit der Vektorgraphik wird beschrieben, weil es beispielhaft zwei wichtige Konzepte zur Beschleunigung von Visualisierungsverfahren nutzt:

- Zum einen werden durch Feststellung des räumlichen Zusammenhangs Nachbarschaftsbeziehungen (Kohärenz) zur Reduzierung der Anzahl notwendiger Tests ausgenutzt,
- zum anderen wird durch sukzessive Unterteilung die Komplexität des Visibilitätsproblems soweit reduziert, daß es leicht entschieden werden kann.

Das Ziel des Verfahrens ist, Primitiva höherer Ordnung darstellen zu können, speziell Parameterflächen $Q(u,v)$ (vgl. Abschnitt 4.7 'Parametrisierte Flächen'). Diese Flächen werden hier durch *u*-*v*-Parameterlinien $Q(u_i,v)$ bzw. $Q(u,v_j)$ dargestellt.
Die Sichtbarkeit wird zunächst nur für die Schnittpunkte $Q(u_i,v_j)$ dieser Parameterlinien bestimmt.
Teilflächen zwischen benachbarten Punkten $(Q_{i,j},Q_{i+1,j},Q_{i,j+1},Q_{i+1,j+1})$ werden durch Dreiecke approximiert (vgl. Bild 5.32). Im weiteren wird die Verdeckung mit Hilfe dieser Approximation bestimmt. Zur Vermeidung unnötiger Tests wird der Projektion der Szene ein quadratisches Raster überlagert (vgl. Bild 5.31). Die

Projektionen der einzelnen Teilflächen werden den Quadraten dieses überlagerten Rasters zugeordnet (z.B. durch Min/max–Tests).

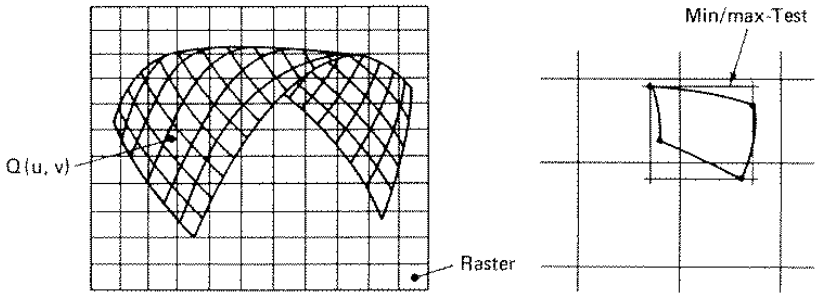


Abbildung 5.31: Das Überlagerungsverfahren

Zum Test eines Punktes Q_{i_0,j_0} wird zunächst festgestellt, in welchem Rasterquadrat seine Projektion liegt. Anschließend werden nur die Teilflächen zur Verdeckungsrechnung herangezogen, deren Projektionen dieses Rasterquadrat überlappen. Im einzelnen durchläuft der Algorithmus folgende Schritte:

1. Projektion der Parameterflächen in die Bildebene.
2. Bestimmung der minimalen und maximalen x– und y–Werte der Projektionen der darzustellenden Teilflächen.
3. Ein $n \times n$ Raster wird über die projizierte Fläche gelegt, so daß eine möglichst „vernünftige” Unterteilung erfolgt.
4. Jedem Rasterquadrat werden alle projizierten Teilflächen zugeordnet, deren Min/max–Test–Rechteck das Rasterquadrat schneidet.
5. Die Visibilität aller $Q_{i,j}$ wird bestimmt, indem nur diejenigen Teilflächen berücksichtigt werden, die dem Rasterquadrat zugeordnet sind. Alle Rasterquadrate, denen mehr als eine Teilfläche zugeordnet ist, werden folgendermaßen bearbeitet:

   (I.) Ist die Anzahl der Teilflächen kleiner oder gleich einer vorgegebenen Schranke?

   (II.) Sind entweder zuviele Teilflächen im Rasterquadrat, oder liegen die zu Q_{i_0,j_0} benachbarten Punkte in anderen Rasterquadraten, so wird es in 4 gleiche Teile zerlegt, die Zugehörigkeit der Teilflächen zu den neuen Rasterquadraten wird gekennzeichnet und bei Schritt (I) fortgefahren, sonst bei (III).

   (III.) Die Teilflächen werden durch Dreiecke approximiert (vgl. Bild 5.32). Da Teilflächen im allgemeinen gekrümmt sind, werden zwei Triangulierungen betrachtet.

   (IV.) Es wird festgestellt, in welchen beiden Dreiecken (in 2D) der Testpunkt liegt (vgl. Bild 5.32).

(V.) Aus den Ebenengleichungen der Dreiecke werden für die *xy*–Koordinaten des Testpunkts die *z*–Werte der beiden Dreiecke ermittelt.

(VI.) Vergleich dieser Werte mit dem *z*–Wert des Testpunkts. Liegt mindestens eine Dreiecksfläche vor dem Testpunkt, so wird er für unsichtbar erklärt.

6. Nachdem die Visibilität aller Flächenpunkte $Q_{i,j}$ festgestellt wurde, können die Flächen durch Verbinden der sichtbaren Punkte ausgegeben werden. Ist die Approximation zu grob, lassen sich auch Punkte auf den Verbindungslinien zwischen den $Q_{i,j}$ auf Visibilität testen.

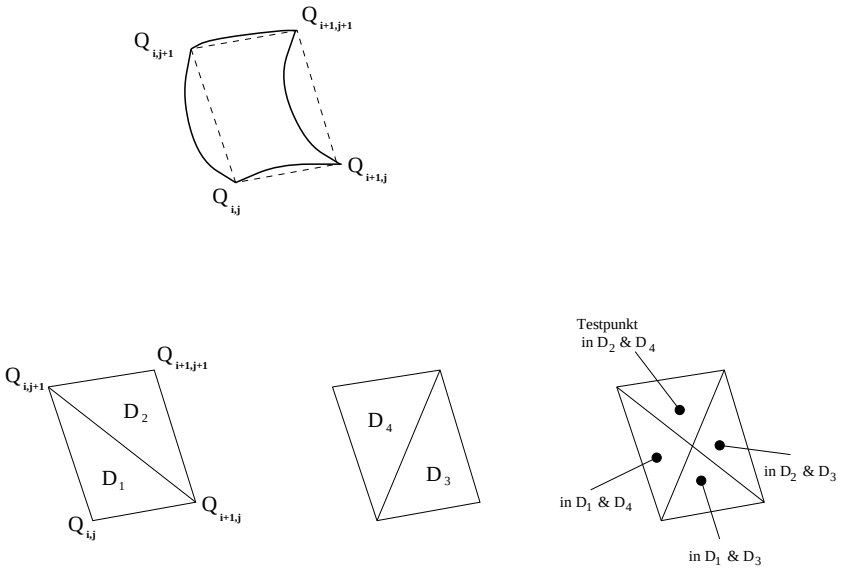


Abbildung 5.32: Dreiecksapproximation der Pflaster

5.3.6 Linienorientierte Visibilitätsverfahren

Wie der Name schon andeutet, werden bei diesen Algorithmen Linien auf Sichtbarkeit untersucht. Diese Vorgehensweise ist naheliegend, lehrt doch die Anschauung, daß sich die Sichtbarkeit nur an Körperkanten und Konturen ändern kann. Sind die Punkte mit Änderung der Sichtbarkeit feststellbar, so kann unmittelbar auf die Sichtbarkeit ganzer Linienteile geschlossen werden. Die Nachbarschaftsbeziehung auf der Linie wird also zur Einsparung von Einzelpunkttests genutzt. Diese Überlegungen führen zum Test objektorientierter Linien.

Im Gegensatz dazu steht der Test von Rasterzeilen. Diese Alternative ergab sich bereits bei der Behandlung punktorientierter Verfahren als Sonderfall des *z*–Buffers. Sie diente zunächst nicht der algorithmischen Verbesserung des Visibilitätsverfahrens, sondern der Speicherökonomie der Implementierung. Die geringen Änderungen des Bildinhalts aufeinanderfolgender Rasterzeilen weisen aber darauf hin, daß auch in diesem Fall Nachbarschaftsbeziehungen ausgenutzt werden

können. Wegen der großen praktischen Bedeutung für die Rastergraphik soll dieser Algorithmustyp zuerst besprochen werden.

5.3.6.1 Test von Rasterzeilen

Bei dieser Klasse von Algorithmen werden Segmente von Rasterzeilen auf Verdeckung getestet. Als Ergebnis steht die sichtbare Information auf einer Zeile zur Verfügung (vgl. Bild 5.33 und [GMSG82]).

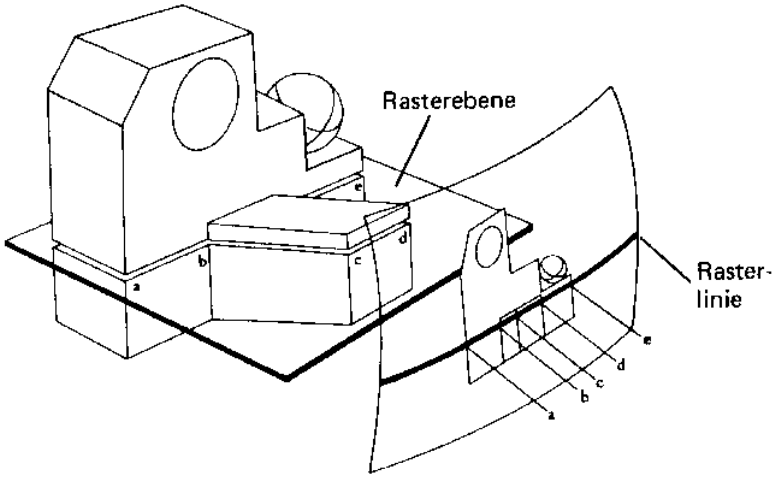


Abbildung 5.33: Prinzip der Rasterlinienalgorithmen

Das Gesamtbild ergibt sich als Summe der Zeilen und wird in Anlehnung an den Bildausgabevorgang beim Rastersichtgerät von oben nach unten erzeugt. Zeilenrasterung (Scankonvertierung) der Szene und Visibilitätsberechnung sind eng miteinander verbunden (sie benutzen dieselbe Kohärenz) und bieten so eine günstige Voraussetzung zur schnellen hardwareunterstützten Ausführung. Der erste Algorithmus zur Echtzeiterzeugung von Rasterbildern stammt deshalb auch aus dieser Klasse und soll im folgenden exemplarisch behandelt werden.

5.3.6.2 Watkins' Algorithmus

Der Watkins' Algorithmus [Wat70] setzt die Approximation der Szene durch ebene Polygone voraus, was z.B. durch eine Dreieckszerlegung erreicht werden kann. Rasterzeilen werden hier also auf Verdeckung durch geschlossene Polygone getestet. Zwei Arten von Nachbarschaftsbeziehungen (Kohärenz) können dadurch einfach zur Verringerung der Anzahl notwendiger Tests ausgenutzt werden:

- Interpolation in x–Richtung innerhalb eines Polygons (*Spaltenkohärenz*);
- Interpolation in y–Richtung (*Kantenkohärenz*) .

Um unnötige Tests mit Polygonen zu vermeiden, die zur betrachteten Rasterzeile gar keinen Beitrag liefern können, werden vor dem eigentlichen Algorithmus alle Polygonkanten nach dem Index der Rasterzeile, auf der der obere Kantenpunkt liegt, sortiert (vgl. Min/max-Test), also von oben nach unten (Y–Sortierung) und von links nach rechts (X–Sortierung) (vgl. Bild 5.34). Das Ergebnis wird in der Kantenliste KL gespeichert. Jede Kante taucht nur einmal darin auf.

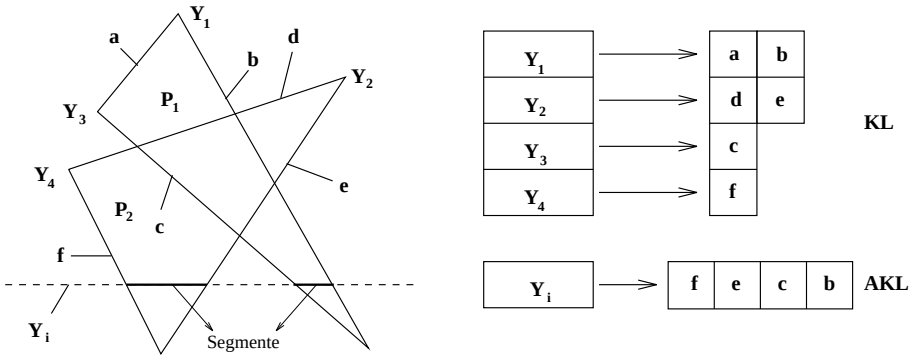


Abbildung 5.34: Kantenliste *KL* und aktive Kantenliste *AKL*

Die so geordnete Kantenliste *KL* enthält mindestens folgende Information über jede Kante:

- X–Koordinate des oberen Endes,
- Y–Koordinate des unteren Endes (alternativ ΔY = Anzahl der geschnittenen Zeilen),
- $\frac{\Delta X}{\Delta Y}$, um inkrementell den Schnittpunkt mit der nächsten Zeile zu erhalten

  (vgl. Abschnitt 2.5.3 ’Rasterung von Polygonen’),
- den Polygonnamen.

Der Polygonname dient als Zeiger in eine Polygonliste, in der Informationen zur Berechnung der Z–Werte, der Grau- oder Farbwerte und eventuell der Normalen gespeichert sind. Zum Beispiel können der Z–Wert und die Farbwerte an der obersten Ecke sowie deren Änderungen in X- und Y-Richtung gespeichert sein. Die Berechnung dieser Größen als lineare Funktion von X,Y kann inkrementell erfolgen (vgl. Kantenfunktion in Abschnitt 2.5.4 ’Glätten von Polygonkanten’).

Aus der Kantenliste wird beim zeilenweisen Bearbeiten des Bildes eine Liste *AKL* der aktiven Kanten und damit auch Segmente aufgebaut (Segment = Schnitt des projizierten Polygons mit der Rasterzeile) (vgl. Bild 5.34). Sie wird von Zeile zu Zeile auf den aktuellen Stand gebracht, das heißt:

- Löschen aller Kanten, die in der nächsten Rasterzeile nicht mehr auftreten;
- Weiterführen aller auch in der nächsten Rasterzeile aktiven Kanten;
- Einfügen (X–sortiert) aller neuen Kanten.

Nach diesen vorbereitenden Maßnahmen, die zur Zeilenrasterung gehören, betrachten wir das eigentliche Visibilitätsverfahren. Es besteht aus zwei Schritten:

1. Bestimmung von Segmenten ohne Visibilitätsänderung (*sample spans*) auf der betrachteten Rasterzeile;
2. Bestimmung der Visibilität.

Die *sample spans* werden aus den Segmenten der einzelnen Polygone, die die *AKL* bereithält, gewonnen. Weisen die Polygonsegmente in X–Richtung keine Überlappung auf, kann auch keine Visibilitätsänderung auftreten. Die Segmente sind dann gleichzeitig sichtbare sample spans (vgl. Bild 5.35).

Sind Überlappungen vorhanden, muß wegen möglicher Polygondurchdringungen die Bestimmung der sample spans in der XZ–Ebene vorgenommen werden. Ein sample span liegt vor, wenn in einem Bereich der Rasterzeile

– die Anzahl der Segmente konstant bleibt und
– diese Segmente sich in der XZ–Ebene nicht schneiden.

Die Visibilität innerhalb eines sample spans ist durch einen Z-Werte–Vergleich an beliebiger Stelle, zweckmäßigerweise aber beim kleinsten X–Wert, festzustellen.

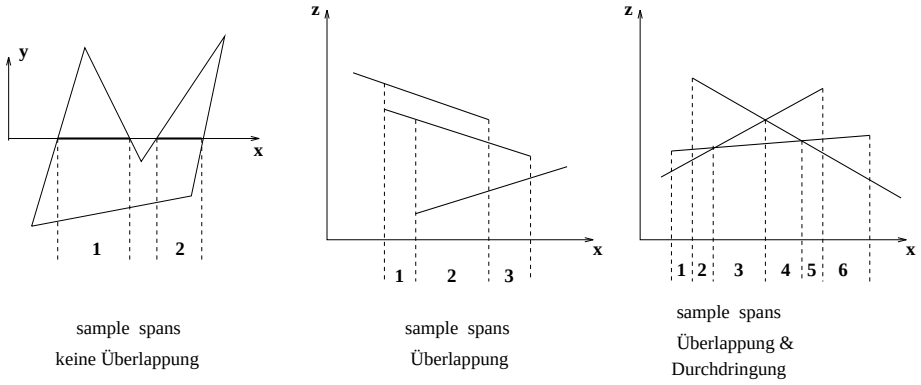


Abbildung 5.35: Bestimmung der sample spans in der XZ–Ebene

Beim Übergang zur nächsten Rasterlinie kann sich die Sichtbarkeit auf den sample spans nur ändern, sofern

– Polygone hinzukommen oder wegfallen
– sich zwei Polygonkanten zwischen zwei Rasterzeilen schneiden, die X–Sortierung in der *AKL* sich also ändert.

Watkins' Algorithmus eignet sich gut zur Grau(Farb-)wertdarstellung. Sind die sample spans ermittelt, können die einzelnen Bildpunkte nach vorgegebenen Schattierregeln berechnet werden. Aus diesem Grund wird zu den Kantenlisten auch

eine Polygonliste geführt, die die entsprechende Farbinformation bereithält. Zusätzlich können z.B. auch die Koeffizienten der Ebenengleichungen gespeichert werden, was ein schnelles Berechnen von Z–Werten, Normalen usw. gestattet. Zu Watkins' Algorithmus gibt es viele Varianten. Eine gute Übersicht historischer Verfahren ist in [SH74], [NS79] und [Rog85] zu finden.

Wenn Sie die PDF-Version des Kurses bearbeiten, können Sie jetzt das Interaktionselement 'watkins' anwählen.

5.3.6.3 Test von objektorientierten Linien

Diese Klasse von Algorithmen verfolgt den zu Beginn dieses Abschnitts skizzierten natürlichen Weg und ermittelt die Objektränder und die Konturen. Die bekanntesten Verfahren stammen von Appel [App67], Galimberti–Montanari [GM69] und Loutrel [Lou70]. Da im Objektraum gearbeitet wird, ist das Ergebnis geräteunabhängig.

Diesem Verfahren liegt folgende Idee zugrunde: In einem ersten Schritt werden durch Beseitigung der Rückseiten die Objektränder und die Konturen der Szene ermittelt. Bei Objekten mit ebener Polygonbegrenzung sind das die Kanten, die ein sichtbares und ein unsichtbares Polygon trennen. Das heißt, die Änderung der Sichtbarkeit findet nur an diesen Kanten statt. Deshalb können sich Tests auf Punkte dieser Kanten beschränken. Verfolgt man die Kanten einer potentiell sichtbaren Fläche, so muß man nach einem Vorschlag von Appel [App67] bei jedem Schnitt mit einer Kontur nur die "quantitative Unsichtbarkeit " QI ermitteln. QI gibt an, wieviele Polygone die Testkante verdecken. Sichtbarkeit ist dann für QI=0 gegeben (vgl. Bild 5.36). Da bei diesem Verfahren alle Kanten gegen alle Flächen getestet werden, hängt das Zeitverhalten quadratisch von der Zahl der Objekte ab.

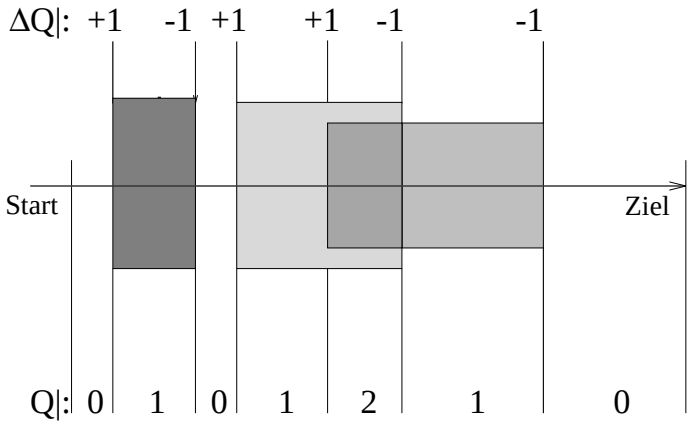


Abbildung 5.36: Quantitative Unsichtbarkeit entlang einer potentiell sichtbaren Kante

Deshalb wird in [Hor84] ein regionenorientiertes Verfahren entwickelt, das durch stärkere Ausnutzung von Nachbarschaftsbeziehungen zu einer Reduktion der Anzahl notwendiger Tests gelangt und ein lineares Verhalten zeigt. Dabei wird nicht

nur festgestellt, von wievielen Objekten ein Punkt verdeckt wird, sondern auch, wieviele er selbst verdeckt.

5.3.6.4 Regionenorientiertes Visibilitätsverfahren

In Anlehnung an Appel [App67] wird beim regionenorientierten Visibilitätsverfahren [Hor84] jedem Punkt P der Szene ein sogenannter quantitativer Visibilitätsstatus

$$qvs = (qc, qh)$$

zugeordnet. Dabei ist

qc = Anzahl der von P verdeckten (covered) Punkte,

qh = Anzahl der P verdeckenden (hiding) Punkte.

Beseitigung der Rückseiten

Dieses Verfahren arbeitet nach folgenden Grundprinzipien: In einem ersten Schritt werden die potentiell sichtbaren Gebiete, hier Regionen genannt, ermittelt, d.h. die Rückseiten werden entfernt. Dann wird die Visibilität (qvs) der Regionenränder durch Test der Ränder gegeneinander festgestellt. Damit ist qvs für alle Randpunkte bestimmt. Daran schließt sich die Projektion der sichtbaren verdeckenden Randsegmente auf die verdeckten Regionen an. Jetzt sind die sichtbaren Gebiete bekannt und können ausgegeben werden.

Gegenüber den konturorientierten weist das regionenorientierte Verfahren einige prinzipielle Vorteile auf. Auf das Testen aller Objekte gegeneinander kann vollständig verzichtet werden. Die Visibilität der Randpunkte wird mit minimalem Aufwand bestimmt, da nur an Stellen einer qvs–Änderung gerechnet wird. Ebenso werden bei der Projektion nur die Flächen betrachtet, an deren Rand sich die Visibilität von sichtbar nach unsichtbar ändert. Am Beispiel zweier polygonal approximierter Tori soll die Wirkungsweise des Algorithmus schrittweise verdeutlicht werden.

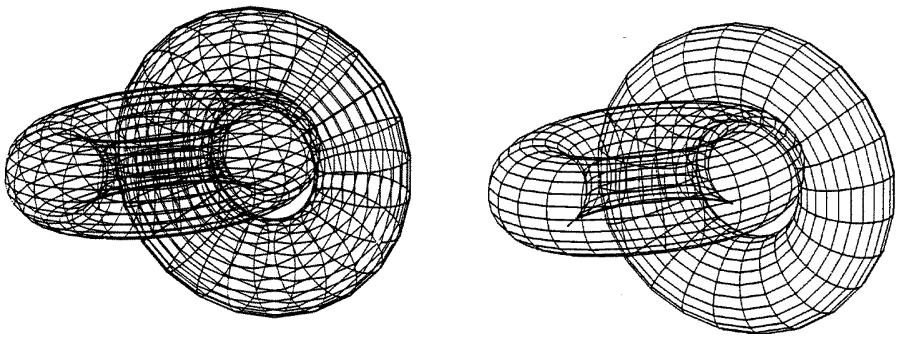


Abbildung 5.37: 2 Tori, durch Polygone approximiert, und Regionen (Rückseiten beseitigt)

Wenn Sie die PDF-Version des Kurses bearbeiten, können Sie jetzt das Interaktionselement 'region' anwählen.

5.3.6.5 Bestimmung der Regionen

Dieser erste Schritt entspricht der in 5.3.2 'Beseitigung der Rückseiten (Back Face Culling)' beschriebenen Beseitigung der Rückseiten (Culling). Dementsprechend müssen die Kanten der Polygonapproximation nach Konturen durchsucht werden. Ist eine Konturkante gefunden worden, so werden alle weiteren Kanten desselben Regionenrandes iterativ unter Nutzung der Nachbarschaftsinformation bestimmt. Dieser Schritt wird nacheinander für alle Ränder und Regionen durchgeführt (vgl. Bild 5.37). Im Gegensatz zu konturorientierten Verfahren, die jeweils isolierte Konturkanten betrachten, wird hier der Visibilitätsstatus entlang der Ränder und innerhalb der Regionen fortgeschaltet. (Natürlich muß hierzu eine entsprechende Objektdatenstruktur aufgebaut werden.)

5.3.6.6 Visibilitätstest der Ränder

Nach der Bestimmung der Regionen müssen alle Ränder gegeneinander getestet werden (vgl. Bild 5.38), um die sichtbaren Segmente der Ränder ($qh = 0$) zu erhalten.

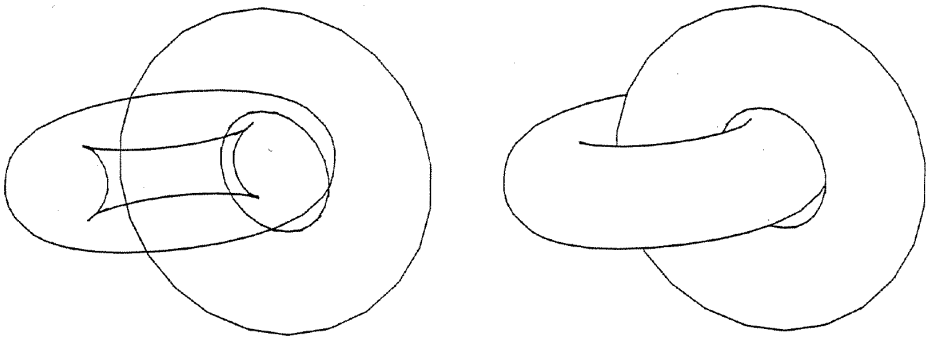


Abbildung 5.38: Ränder der Regionen und sichtbare Ränder

5.3.6.7 Projektion der sichtbaren, verdeckenden Randsegmente

Für diese Segmente gilt $qc > 0$ und $qh = 0$. In jedem Startpunkt eines solchen Segments gibt es einen verdeckten Punkt eines anderen Segments mit $qh = 1$. Von diesem Punkt ausgehend kann nun das verdeckende Randsegment auf die verdeckte Region projiziert werden (vgl. Bild 5.39).

Das vorgestellte Verfahren ist nicht auf polygonal begrenzte Objekte beschränkt. Vielmehr kann es vorteilhaft zur analytischen Visibilitätsberechnung bei Tensorprodukt-Flächen $Q(u,v)$ (siehe Abschnitt 4.7.2 'Tensorproduktflächen' und Abschnitt 4.9 'Coons-Pflaster und Gordon-Flächen – Interpolation von Kurven') eingesetzt werden. Der algorithmische Vorteil, das Innere der sichtbaren Gebiete

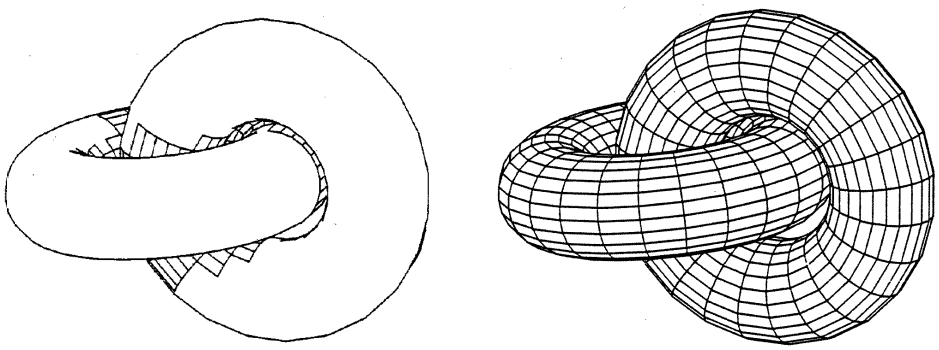


Abbildung 5.39: Projektion der verdeckenden, sichtbaren Randsegmente und Darstellung der sichtbaren Gebiete

ohne Visibilitätsrechnung ausfüllen zu können, kommt bei höheren Funktionen besonders zum Tragen. Darüber hinaus ist die analytische Lösung geräteunabhängig und invariant gegenüber affinen Bildtransformationen [RH84], [HLRS85] und [EC90].

Die Begriffe *Region* und *Gebiet* werden in der Parameterebene besonders deutlich, weshalb die einzelnen Schritte des Algorithmus anhand von Bildbeispielen nochmals verdeutlicht werden. Bild 5.40 zeigt die Bestimmung der Regionen.

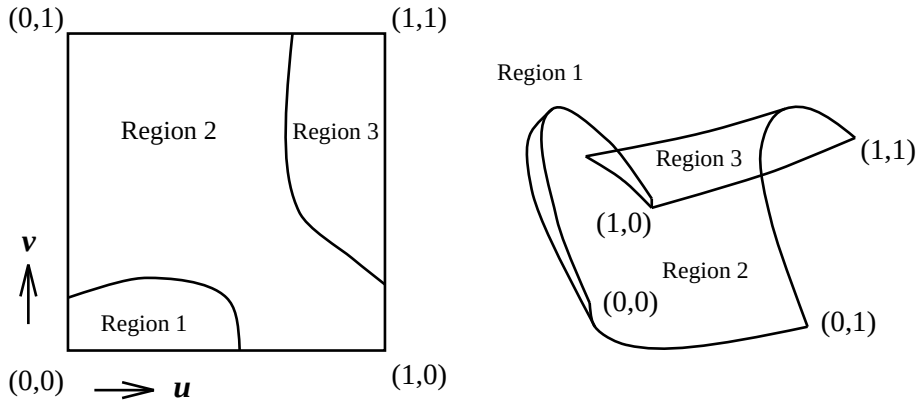


Abbildung 5.40: Die Regionen einer Fläche in *uv*– und *XY*–Ebene

Im nächsten Schritt müssen die sichtbaren Segmente der Regionenränder bestimmt werden. Dazu wird jedem Punkt der Ränder ein *qvs*–Status zugeordnet.

Beim Durchlaufen der Ränder können folgende 4 Statusänderungen auftreten:

ic	(inkrementiere *qc*)	*qc*	=	$qc+1$
dc	(dekrementiere *qc*)	*qc*	=	$qc-1$
ih	(inkrementiere *qh*)	*qh*	=	$qh+1$
dh	(dekrementiere *qh*)	*qh*	=	$qh-1$

Bild 5.41 zeigt diese Zuordung in der *uv*– und *XY*–Ebene.

Die Werte der *qvs* erhält man aus der Überlegung, daß auf einem geschlosse-

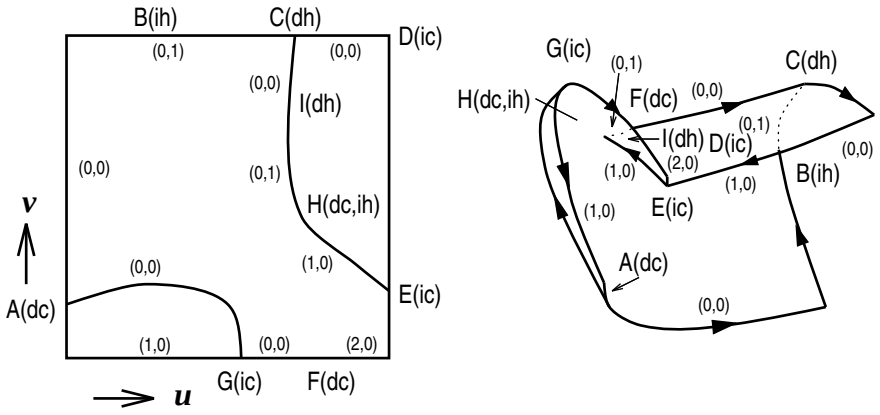


Abbildung 5.41: Zuordnung des *qvs*

nen Regionenrand ein sichtbares Segment mit $qvs = (0,0)$ vorhanden sein muß. Hieraus lassen sich die absoluten *qvs* aller anderen Segmente bestimmen. Bild 5.42 zeigt die sichtbaren Regionenränder.

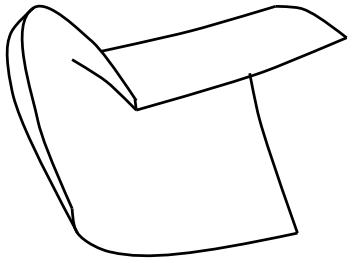


Abbildung 5.42: Sichtbare Segmente der Regionenränder

Die sichtbaren Gebiete erhält man schließlich durch Projektion der sichtbaren, verdeckenden Segmente ($qc > 0$, $qh = 0$) auf die Fläche. Ein Auffüllen der sichtbaren Gebiete durch Parameterlinien ist nun problemlos möglich (vgl. Bild 5.43).

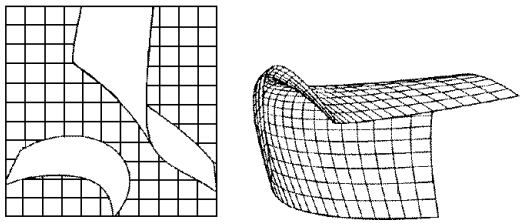


Abbildung 5.43: Sichtbare Gebiete in *uv*– und *XY*–Ebene

5.3.7 Flächenorientierte Visibilitätsverfahren

Bei diesen Algorithmen werden Flächen auf Verdeckung getestet. Grundsätzlich ist zwischen Rasterflächen (RF = Ausschnitt des Bildschirms) und Objektflächen (OF) zu unterscheiden.

5.3.7.1 Test von Rasterflächen

Das bekannteste Verfahren dieses Typs ist der *Warnock-Algorithmus* [War69], bei dem eine RF auf Verdeckung durch OF getestet wird und das „divide et impera"-Prinzip zur Vereinfachung des Visibilitätsproblems benutzt wird. Die Visibilität für eine RF kann unmittelbar bestimmt werden, wenn eine der beiden Bedingungen erfüllt ist:

1. Die dem Betrachter am nächsten liegende OF überdeckt die gesamte RF.
2. Es existiert keine oder nur eine OF innerhalb der RF.

Ist keine der beiden Bedingungen erfüllt, so wird die Rasterfläche unterteilt und beide Bedingungen werden erneut überprüft. Entartet die Rasterfläche zu einer Zeile, so wird eindimensional weiter unterteilt. Spätestens bei der Reduktion der Rasterfläche auf einen Rasterpunkt wird dann die erste Bedingung als erfüllt angesehen, und die Sichtbarkeit kann durch Tiefentest (z-Buffer) entschieden werden. Zur Vermeidung von Aliasing könnte hier auch nach dem EXACT–Algorithmus weiter verfahren werden. In der Originalarbeit [War69] wird die Rasterfläche bei jedem Unterteilungsschritt in vier gleiche Teile zerlegt, was eine gewisse Ineffizienz zur Folge hat. Bis auf den Tiefentest ist dieses Verfahren ein Clipping–Algorithmus. Ein Clipping an den gegenseitigen Flächengrenzen anstelle der regelmäßigen Unterteilung bietet sich an. Im Weiler–Atherton–Algorithmus [WA77] wird dies (im Objektraum) realisiert.

5.3.7.2 Test von Objektflächen

Der Weiler–Atherton–Algorithmus erlaubt das Clipping beliebiger Polygone an beliebigen Fensterpolygonen.

Der Algorithmus verfolgt dabei die Kanten des Testpolygons in einer festen Richtung (z.B. Uhrzeigersinn), bis ein Schnitt mit dem Fensterpolygon festgestellt wird (vgl. Bild 5.44). Tritt die Testpolygonkante an dieser Stelle in das Fenster ein, so verfolgt der Algorithmus weiterhin die Testpolygonkante. Verläßt die Polygonkante das Fenster, so wird die rechts liegende Fensterkante weiterverfolgt. Das Ergebnis des Schnittpunkts wird abgespeichert und sichergestellt, damit kein Pfad zweimal durchlaufen wird.

Nach dem Clipping kann die Sichtbarkeit des Fensterpolygons sofort, ohne weitere Unterteilung, entschieden werden. Anschaulich gesprochen werden die sicht-

baren Gebiete wie mit Plätzchenformen ausgestanzt. Gegenüber dem Warnock–Algorithmus sind erheblich weniger Unterteilungen vorzunehmen, allerdings sind sie mit komplexeren Operationen verbunden.

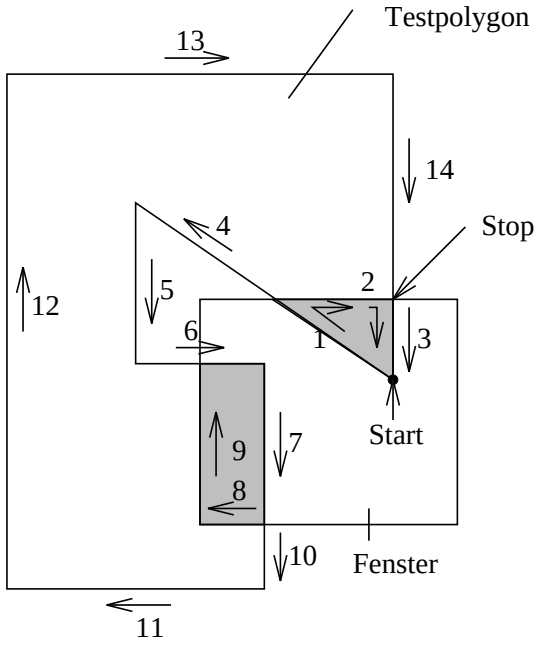


Abbildung 5.44: Weiler–Atherton–Clipping

5.3.8 Bewertung von Visibilitätsverfahren

Bei der Beschreibung von Visibilitätsverfahren wurde in jeder Klasse zwischen analytischen (objektorientierten) und Rasteralgorithmen unterschieden. Intuitiv wurde klar, daß die analytischen Algorithmen mit wenigen komplizierten Tests auskommen, die Rasteralgorithmen dagegen viele einfache Tests durchführen. Eine sehr grobe Abschätzung ergibt für erstere einen Aufwand proportional P^2, für letztere proportional PN. Dabei ist P die Anzahl der zu testenden Objekte und N die Anzahl der zu berechnenden Bildpunkte. Viele Algorithmen wurden entwickelt, um unterhalb dieser Schranken zu bleiben. Aus prinzipiellen Gründen sind die algorithmisch minimalen Verfahren in der Klasse der analytischen Verfahren zu finden [Hor84]. Diese algorithmische Minimalität sagt aber nur bedingt etwas über die Nützlichkeit eines Verfahrens aus, z.B. sind alle Echtzeitverfahren Rasteralgorithmen.

Soll die Ausgabe als Strichzeichnung erfolgen (z.B. zur präzisen, maßstäblichen Illustration von Büchern), sollte ein analytischer Algorithmus gewählt werden. Umgekehrt ist ein Rasteralgorithmus für nur flächenhafte Ausgabe sehr gut geeignet.

Die Anwendung bestimmt letztlich den zu verwendenden Algorithmus. Dabei

sind hohe Interaktionsrate einerseits und exakte Darstellung andererseits die sich widersprechenden Anforderungen. Der Anwender muß hier Prioritäten setzen.

Durch weitere Verbesserungen der Hardware, insbesondere der Speicherbausteine für die direkte Realisierung von Tiefenvergleichen auf dem Chip, wird der *z*–Buffer das dominierende Verfahren sein. Wie mit dem EXACT–Algorithmus gezeigt wurde, kann der verbesserte *z*–Buffer alle Qualitätsansprüche erfüllen. Interessant ist in diesem Zusammenhang auch, daß die Kosten des Verfahrens im wesentlichen von der Zahl der Bildpunkte des Ausgabegeräts und nicht von der Komplexität der Szene abhängen. Dies liegt daran, daß mit zunehmender Szenenkomplexität (größere Anzahl von Dreiecken) die Zahl der überdeckten Pixel pro Dreieck geringer wird und sich über die gesamte Bildschirmfläche gleichmäßig verteilt. Bei relativ großen Kosten für einfache Szenen wird das Preis-Leistungs-Verhältnis des *z*-Buffers mit steigender Szenenkomplexität immer besser.

5.4 Übungsaufgaben

Aufgabe 1:

Beim parametrischen Linien-Clipping wird die mit dem Fenster zu schneidende Strecke $\overline{P_1 P_2}$ durch Gleichung (5.6)

$$P = P(\lambda) = P_1 + \lambda(P_2 - P_1)$$

beschrieben.

a) Geben Sie die Parameterwerte $\lambda_i,\ 1 \leq i \leq 4,$ für die vier möglichen Schnitte der Geraden durch P_1, P_2 mit den Fenstergrenzen $X_{min(max)}$, $Y_{min(max)}$ an.

b) Der sichtbare Teil der Strecke $\overline{P_1 P_2}$ werde durch die Parameterwerte λ_{S1} und λ_{S2}, $\lambda_{S1} < \lambda_{S2}$, bestimmt. Geben Sie Gleichungen zur Berechnung von λ_{S1} und λ_{S2} an. Welche Bedingungen entsprechen dem Outcode des Cohen-Sutherland-Algorithmus?

c) Geben Sie den vollständigen Clipping-Algorithmus in Pascal-, C- oder Pseudocode an.

Aufgabe 2:

a) In Abschnitt 5.3.3 'Punktklassifizierung' wurde die Punktklassifizierung bezüglich eines beliebigen Polygons in der XY-Ebene besprochen. Nehmen Sie ein geschlossenes Polygon mit n Ecken P_i an und führen Sie den Punkttest mit einem Strahl in Richtung der positiven X-Achse durch.

b) Testen Sie, ob ein Punkt innerhalb eines Dreiecks liegt. Verwenden Sie baryzentrische Koordinaten.

Aufgabe 3:

Für eine aus zwei undurchsichtigen Flächen bestehende Szene soll unter Anwendung des Warnock-Unterteilungsverfahrens die Visibilität bestimmt werden. Die Projektionsebene sei die XY-Ebene $(Z = 0)$. Das Koordinatensystem ist linkshändig, und der Blickpunkt liegt im Unendlichen (bei $Z = -\infty$).

a) Beschreiben Sie (evtl. mit Skizzen), welche Situationen allgemein bei Anwendung des Warnock-Unterteilungsverfahrens nach jedem Unterteilungsschritt auftreten können und geben Sie an, wie Sie unter Berücksichtigung der in Abschnitt 5.3.7 'Flächenorientierte Visibilitätsverfahren' angegebenen Kriterien in jeder dieser Situationen vorgehen.

b) Gegeben seien nun folgende Objektflächen (OF):

– ein durch

$$A = (A1(18.5, 12.5, 0); \quad A2(22.5, 16.5, 0); \quad A3(14.5, 22.5, 14))$$

bestimmtes rotes Dreieck, und

– ein durch

$$\begin{aligned} B = (&B1(12.5, 14.5, 6); \qquad B2(20.5, 14.5, 6); \\ &B3(20.5, 20.5, 6); \qquad B4(12.5, 20.5, 6)) \end{aligned}$$

definiertes blaues Rechteck.

Die Pixelmittelpunkte haben ganzzahlige Koordinaten. Die Hintergrundfarbe sei weiß.

1) Führen Sie für diese Szene das regelmäßige, quadratische Warnock-Unterteilungsverfahren graphisch anhand von Skizzen durch.
Die Rasterfläche, von der auszugehen ist, sei durch

$$\begin{aligned} C = \quad &(C1(9.5, 9.5); \qquad C2(25.5, 9.5); \\ &C3(25.5, 25.5); \qquad C4(9.5, 25.5)) \end{aligned}$$

definiert, und ein Pixel habe die Kantenlänge Eins, d.h. die Ausgangsrasterfläche umfaßt 16×16 Pixel.

Zur Entscheidung, ob eine Objektfläche, die die Rasterfläche ganz bedeckt, dem Betrachter am nächsten ist, werden die Z-Koordinaten aller Objektflächen an den vier Eckpunkten der betrachteten Rasterfläche herangezogen: Nur wenn die OF an allen vier Punkten den kleinsten Z-Wert aufweist, verdeckt sie sicher alle anderen OF.

In Rasterflächen, in denen die Visibilität entschieden ist, soll die sichtbare OF eingezeichnet werden: Wenn der Pixelmittelpunkt auf der OF oder auf ihren Rand liegt, wird das Pixel eingefärbt.

2) Beendet man das Warnock-Unterteilungsverfahren, sobald die unterteilten Flächen Pixelgröße erreicht haben, so kann, sofern dann immer noch mehrere potentiell sichtbare Fragmente vorhanden sind, noch keine endgültige Farbe angegeben werden. Diese Pixel müßten eigentlich weiter unterteilt werden.

Wenden Sie auf diese „übriggebliebenen" Pixel, wie in Abschnitt 5.3.5 'Punktorientierte Visibilitätsverfahren' angegeben, den z-Buffer-Algorithmus an, indem Sie die Visibilität am Pixelmittelpunkt bestimmen. Zeichnen Sie das Ergebnis. (Zeichnen Sie Objekt A, wenn A und B in einem Punkt den gleichen Z-Wert haben.)

3) Bestimmen Sie das exakte Ergebnis durch analytische Berechnung.

Aufgabe 4

Berechnen Sie die im Abschnitt 5.2.3 ’Viewing Pipeline (Darstellungsreihe)’ vorgestellte Orientierungsmatrix.

Aufgabe 5:

Skizzieren Sie die in Abschnitt 5.2.3 ’Viewing Pipeline (Darstellungsreihe)’ angegebenen 6 Elementartransformationen zur Berechnung der View-Mapping-Matrix (Seite 266 ff.). Orientieren Sie sich dabei an den Bildern 5.14–5.17.

5.5 Lösungen zu den Übungsaufgaben

Lösung 1:

a) Nach (5.6) gilt:

$$\begin{aligned} P(\lambda) &= P_1 + \lambda \cdot (P_2 - P_1) \\ &= P_1 + \lambda \cdot \Delta P, \end{aligned}$$

mit

$$P(0) = P_1,\ P(1) = P_2,\ \Delta P = [\Delta X, \Delta Y].$$

Das Innere des Fensters (der sichtbare Teil des Objekts) ist gegeben durch Gleichung (5.1)

$$X_{min} \le X \le X_{max}, \quad Y_{min} \le Y \le Y_{max}.$$

Setzt man (5.6) in (5.1) ein, so erhält man für die vier Begrenzungen des Fensters:

$$\begin{array}{lllllll} - & \Delta X \cdot \lambda & \le & X_1 - X_{min}, & \Delta X \cdot \lambda & \le & X_{max} - X_1 \\ \text{und} & & & & & & \\ - & \Delta Y \cdot \lambda & \le & Y_1 - Y_{min}, & \Delta Y \cdot \lambda & \le & Y_{max} - Y_1. \end{array} \tag{5.17}$$

Diese Ungleichungen lassen sich kompakter schreiben als

$$p_i \cdot \lambda_i \le q_i, \quad i = 1, \ldots, 4,$$

mit der Zuordnung

$$i = 1 : \text{links}, \quad i = 2 : \text{rechts}, \quad i = 3 : \text{unten}, \quad i = 4 : \text{oben},$$

d.h.

$$\begin{array}{llllll} p_1 &= -\Delta X, & q_1 &= X_1 - X_{min}, \\ p_2 &= \Delta X, & q_2 &= X_{max} - X_1, \\ p_3 &= -\Delta Y, & q_3 &= Y_1 - Y_{min}, \\ p_4 &= \Delta Y, & q_4 &= Y_{max} - Y_1. \end{array}$$

Die vier gesuchten Parameterwerte ergeben sich dann aus

$$\lambda_i = \frac{q_i}{p_i}. \tag{5.18}$$

b) Die beiden Parameterwerte, die den sichtbaren Teil von $\overline{P_1 P_2}$ begrenzen, sind aus Gleichung (5.18) mit der zusätzlichen Bedingung $0 \le \lambda \le 1$ zu gewinnen. Aus dem folgenden Bild erkennt man, daß für den Fall, daß $\overline{P_1 P_2}$ nicht völlig außerhalb des Fensters liegt, gilt

$$\begin{aligned} \lambda_{S1} &= \max(\{\tfrac{q_i}{p_i} \mid p_i < 0, 1 \le i \le 4\} \cup \{0\}), \\ \lambda_{S2} &= \min(\{\tfrac{q_i}{p_i} \mid p_i > 0, 1 \le i \le 4\} \cup \{1\}). \end{aligned} \tag{5.19}$$

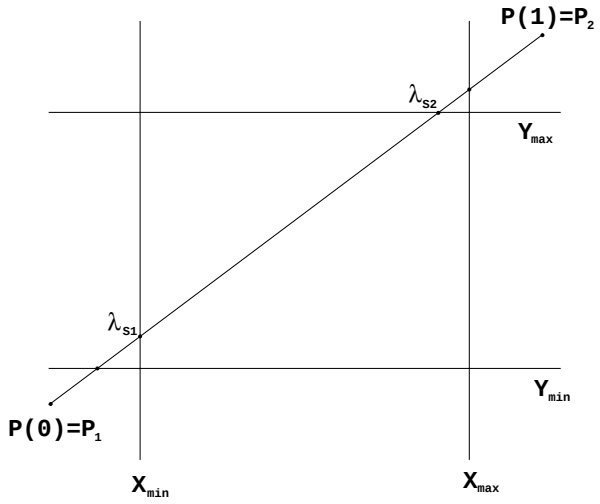


Abbildung 5.45: $\lambda_{S1}, \lambda_{S2}$ für den Fall, daß $\overline{P_1P_2}$ nicht völlig außerhalb des Fensters liegt

$\overline{P_1P_2}$ liegt völlig außerhalb des Fensters, wenn mindestens eine der fünf Bedingungen erfüllt ist:

$$\lambda_{S1} > \lambda_{S2} \text{ oder } p_i = 0 \ \& \ q_i < 0,\ 1 \leq i \leq 4. \tag{5.20}$$

$\overline{P_1P_2}$ liegt auf dem Rand oder im Inneren des Fensters, wenn $\lambda_{S1} = 0$ und $\lambda_{S2} = 1$ sind.

c) Der Algorithmus zum 2D-Clipping kann dann folgendermaßen geschrieben werden, wenn (X_1, Y_1), (X_2, Y_2) die Endpunkte der Strecke sind und das Fenster durch $X_{min} \leq X \leq X_{max}$ und $Y_{min} \leq Y \leq Y_{max}$ definiert ist.

```
procedure clip2d (var X1,Y1,X2,Y2 : real, accept : Boolean);
var λS1, λS2, deltax, deltay : real;

function clipt ((* value *) p,q : real; var λS1, λS2 : real) : Boolean;
var r : real;
begin (* clipt *)
    accept:=true;
    if p < 0 then
    begin
        r:=q/p;
        if r > λS2 then accept:=false (* außerhalb *)
        else if r > λS1 then λS1:=r
    end
    else
    if p > 0 then
    begin
```

```
        r:=q/p;
        if r < λS1 then accept:=false (* außerhalb *)
        else if r < λS2 then λS2:=r
    end

    else (* p = 0 *)
    if q < 0 then accept:=false; (* außerhalb *)
    clipt:=accept;

end (* clipt *);

begin (* clip2d *)

    λS1:=0; λS2:=1;
    deltax:=X2 – X1;
    accept:=false;
    if clipt (– deltax, X1 – Xmin, λS1, λS2) then
    if clipt (deltax, Xmax – X1, λS1, λS2) then
    begin
    deltay:=Y2 – Y1;
    if clipt (– deltay, Y1 – Ymin, λS1, λS2) then
    if clipt (deltay, Ymax – Y1, λS1, λS2) then
    begin
        accept:=true;
        if λS2 < 1 then
        begin
            X2:=X1 + λS2 * deltax;
            Y2:=Y1 + λS2 * deltay;
        end;
        if λS1 > 0 then
        begin
            X1:=X1 + λS1 * deltax;
            Y1:=Y1 + λS1 * deltay;
        end
    end
    end

end (* clip2d *);
```

Lösung 2:

a) Seien der Testpunkt $T(X_t, Y_t)$ und die Polygonecken $P_i(X_i, Y_i)$ gegeben. Zunächst wird der Ursprung des Koordinatensystems in den Testpunkt gelegt:

$$T' = (0,0) \quad \text{und} \quad P_i' = P_i - T.$$

Dann wird gezählt, wieviele Polygonkanten $\overline{P_i P_{i+1}}$ die positive X'-Achse schneiden. Ist die Anzahl ungerade, befindet sich der Testpunkt im Polygon. Die zu zählenden Kanten sind charakterisiert durch:

1. $\text{sign } Y_i' \neq \text{sign } Y_{i+1}' \;\&\; X_i' > 0 \;\&\; X_{i+1}' > 0$

2.

$$\begin{array}{rcl} \text{sign } Y_i' & \neq & \text{sign } Y_{i+1}' \\ & \& & \\ X_i' & > & 0 \\ & \& & \\ X_{i+1}' < 0 & \cup & X_{i+1}' > 0 \\ & \& & \\ X_i' & < & 0 \\ & \& & \end{array}$$

$$\text{Schnittpunkt} \quad \text{mit} \quad Y' = 0 \quad \text{liegt} \quad \text{in} \quad X' > 0.$$

Die Fälle, bei denen Ecken auf dem Teststrahl liegen, müssen gesondert behandelt werden, was kompliziert ist. Deshalb empfehlen sich vorher Tests entlang der anderen drei Halbachsen.

b) Die baryzentrischen Koordinaten (α, β) eines Punktes T bezüglich des Dreiecks $P_1P_2P_3$ sind definiert durch

$$T = \alpha P_2 + \beta P_3 + (1 - \alpha - \beta) P_1$$

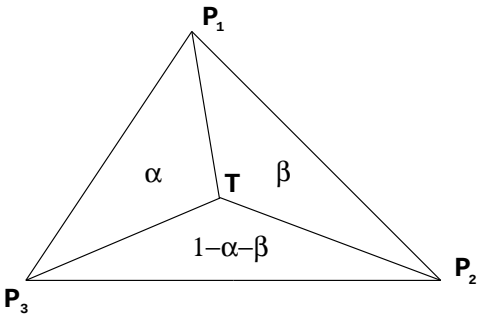


Abbildung 5.46: Baryzentrische Koordinaten eines Punktes T, der im Dreieck $P_1P_2P_3$ liegt

mit

$$\alpha = \frac{\Delta(P_1 T P_3)}{\Delta(P_1 P_2 P_3)} \text{ und } \beta = \frac{\Delta(P_1 P_2 T)}{\Delta(P_1 P_2 P_3)}.$$

T liegt in $P_1P_2P_3$ (einschließlich des Randes), wenn $0 \leq \alpha \leq 1$ und $0 \leq \beta \leq 1$ gilt.

Lösung 3:

a) Zu Beginn und nach jedem Unterteilungsschritt werden die Rasterflächen (RF) auf das Vorliegen der folgenden Bedingungen überprüft:

1. Die dem Betrachter am nächsten liegende OF überdeckt die gesamte RF.
2. Es existiert keine oder nur eine OF innerhalb der RF.

Es wird dann wie folgt vorgegangen:

1. Die gesamte RF erhält die Farbe der am nächsten liegenden, alles andere verdeckenden OF.
2. Existiert eine OF innerhalb der RF, so wird diese gerastert und dargestellt.

Ist keine der beiden Bedingungen erfüllt, so wird die Rasterfläche unterteilt, und die vier Teilflächen werden gesondert auf das Vorliegen der oben genannten Bedingungen überprüft.

b) Das folgende Bild 5.47 zeigt die beiden Objektflächen *A* und *B*.

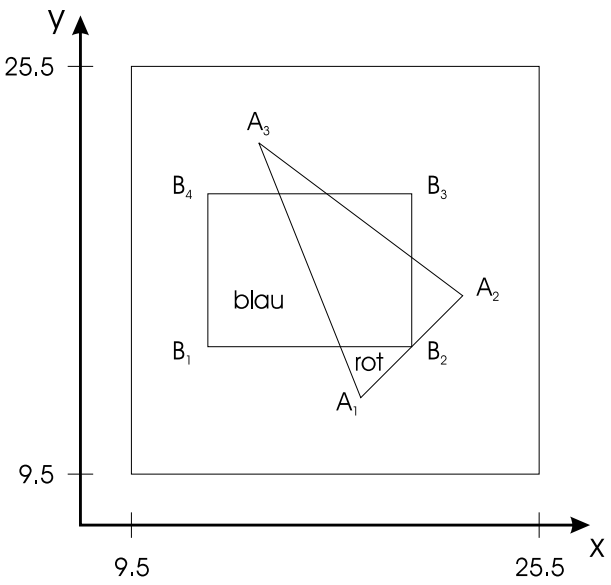


Abbildung 5.47: Objektflächen A und B

Werden Rechteck und Dreieck einzeln gerastert, erhält man die folgende Darstellung (die durchgezogenen Linien begrenzen die exakten analytischen Flächen):

b1) Der Warnock-Unterteilungsalgorithmus und das anschließende Färben der zu den sichtbaren Flächen gehörenden Pixel liefert das in Bild 5.50 dargestellte Ergebnis. Die Zahlen in der Abbildung bezeichnen die beiden in a) besprochenen Fälle; die mit einem Punkt bezeichneten Pixel enthalten noch mehrere potentiell sichtbare Flächen (keine

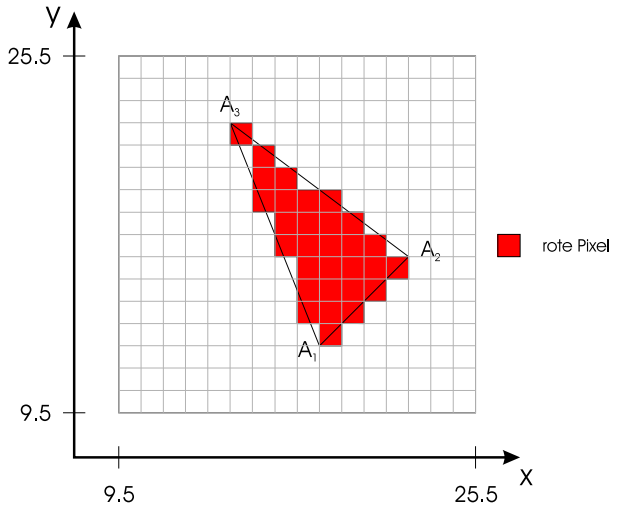


Abbildung 5.48: Objektfläche A mit roten Pixeln

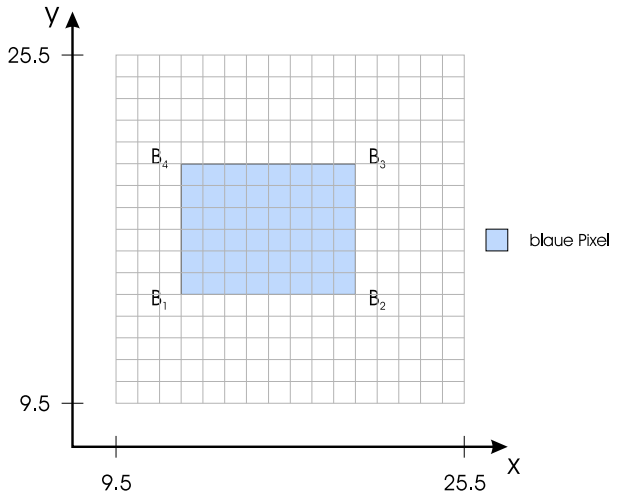


Abbildung 5.49: Objektfläche B mit blauen Pixeln

der beiden Bedingungen ist erfüllt). Sie müßten eigentlich weiter unterteilt werden.

b2) Die Anwendung des z-Buffers für alle in b1) mit einem Punkt bezeichneten Pixel ergibt das in 5.51 dargestellte Bild.

b3) Unter Berücksichtigung der Gleichung für eine Ebene

$$Z = aX + bY + c$$

ergibt sich für die das Dreieck enthaltende Ebene die Gleichung

$$Z = -X + Y - 6$$

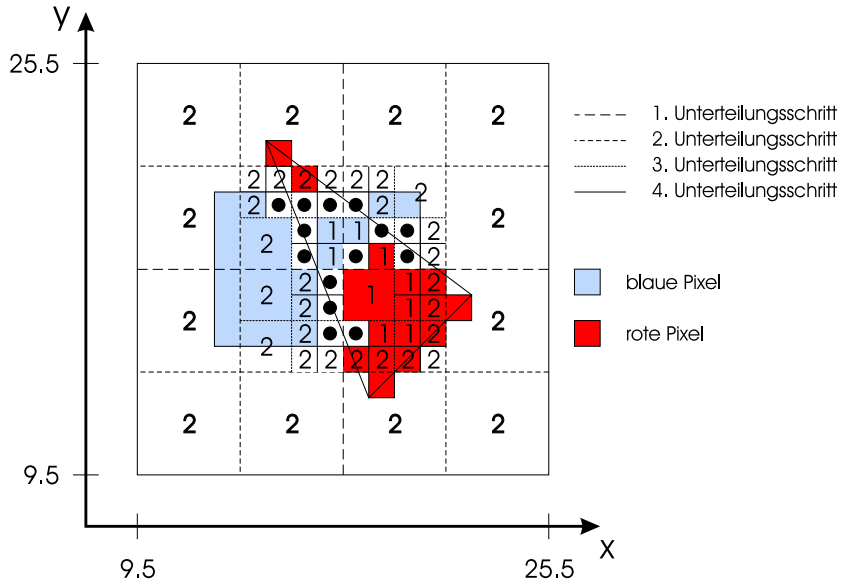


Abbildung 5.50: Ergebnis des Warnock-Unterteilungsalgorithmus und des anschließenden Färbens der zu den sichtbaren Flächen gehörenden Pixel

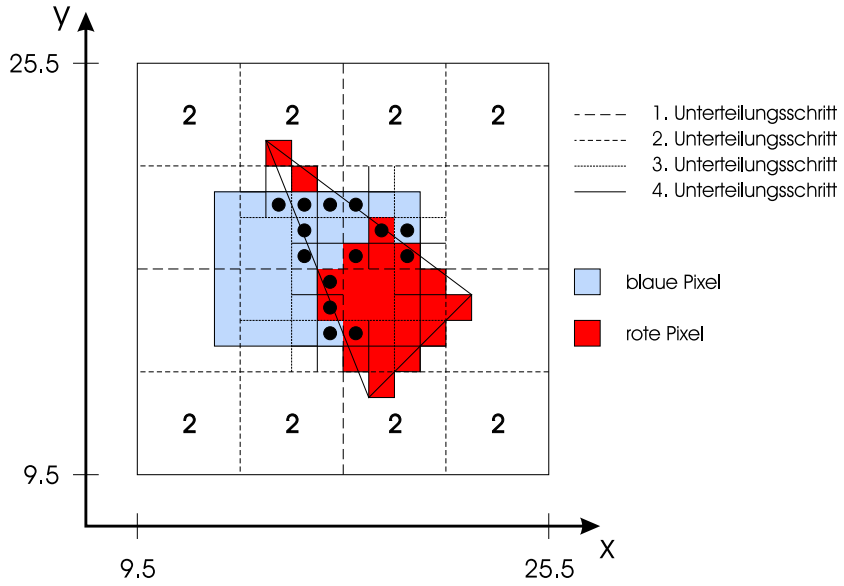


Abbildung 5.51: Anwendung des z-Buffers auf die in Bild 5.50 mit einem Punkt bezeichneten Pixel

und für die das Rechteck enthaltende Ebene die Gleichung

$$Z = 6.$$

Durch Gleichsetzen erhält man die Schnittgerade

$$Y = X, \quad Z = 6,$$

die den sichtbaren unteren Teil des Dreiecks vom verdeckten trennt. Das Bild 5.52 zeigt die sichtbaren Flächen nach der Projektion auf die X-/Y-Ebene.

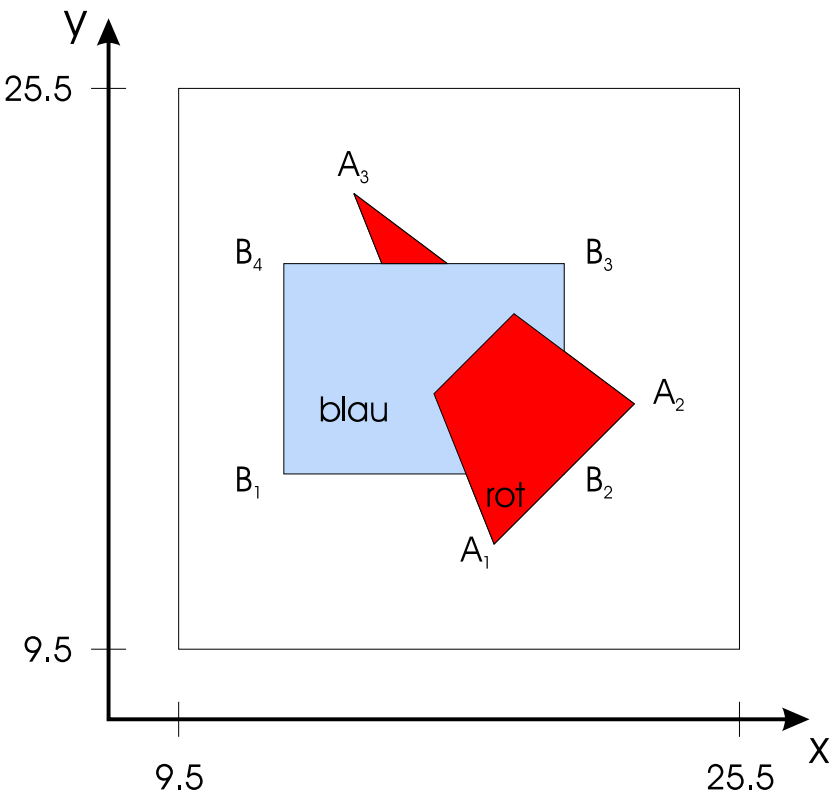


Abbildung 5.52: Sichtbare Flächen nach der Projektion auf die X-/Y-Ebene

Lösung 4:

Die Orientierungsmatrix läßt sich folgendermaßen berechnen (vgl. Abschnitt 3.3.5 'Ebene geometrische Projektionen'), wenn VRP, VUV und VPN in WC3 oder NDC3 gegeben sind und VRC ein Rechtssystem sein soll:

$$V = \frac{VPN \times VUV}{|VPN \times VUV|} \quad \text{und} \quad U = \frac{V \times VPN}{|V \times VPN|}.$$

Die Transformation von Weltkoordinaten in das VRC wird dann in zwei Schritten ausgeführt:

1) Translation in den Ursprung

$$T = T(-VRP) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -VRP_x & -VRP_y & -VRP_z & 1 \end{bmatrix}.$$

2) Rotation, so daß U auf $(1,0,0)$, V auf $(0,1,0)$ und N auf $(0,0,1)$ abgebildet wird, d.h.:

$$\begin{aligned} U \cdot R &= [1,0,0] \\ V \cdot R &= [0,1,0] \\ N \cdot R &= [0,0,1] \end{aligned}$$

bzw. (unter Berücksichtigung der Orthogonalität von U,V,N)

$$R = \begin{bmatrix} U_x & U_y & U_z \\ V_x & V_y & V_z \\ N_x & N_y & N_z \end{bmatrix}^{-1} = \begin{bmatrix} U_x & V_x & N_x \\ U_y & V_y & N_y \\ U_z & V_z & N_z \end{bmatrix}$$

Einbettung in eine 4×4-Matrix (R')

$$R' = \begin{bmatrix} U_x & V_x & N_x & 0 \\ U_y & V_y & N_y & 0 \\ U_z & V_z & N_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

und Multiplikation mit T ergibt die gesuchte Orientierungsmatrix

$$M_{vo} = T \cdot R'.$$

Lösung 5:

Die Skizze der Ausgangssituation ist Bild 5.53:

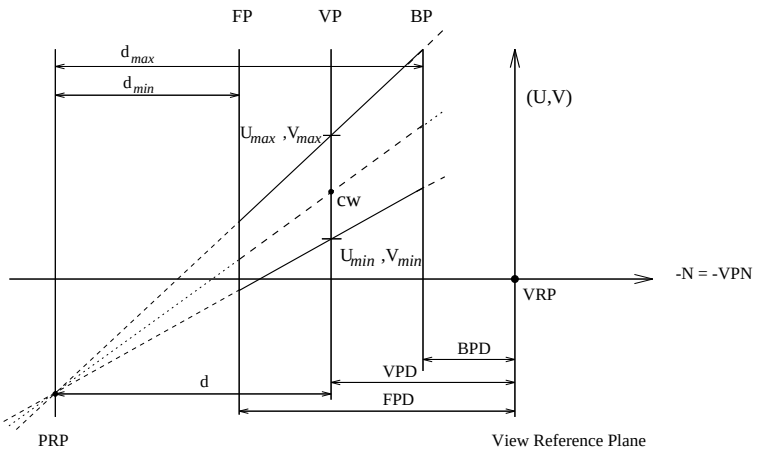


Abbildung 5.53: Ausgangssituation

Mit d bezeichnen wir den Abstand des PRP zur VP mit

$$\begin{aligned} d &= |VPD - PRP_z| \\ d_{max} &= |BPD - PRP_z| \\ d_{min} &= |FPD - PRP_z|. \end{aligned}$$

1.) Translation von PRP in den VRP (Ursprung des VRC-Koordinatensystems):

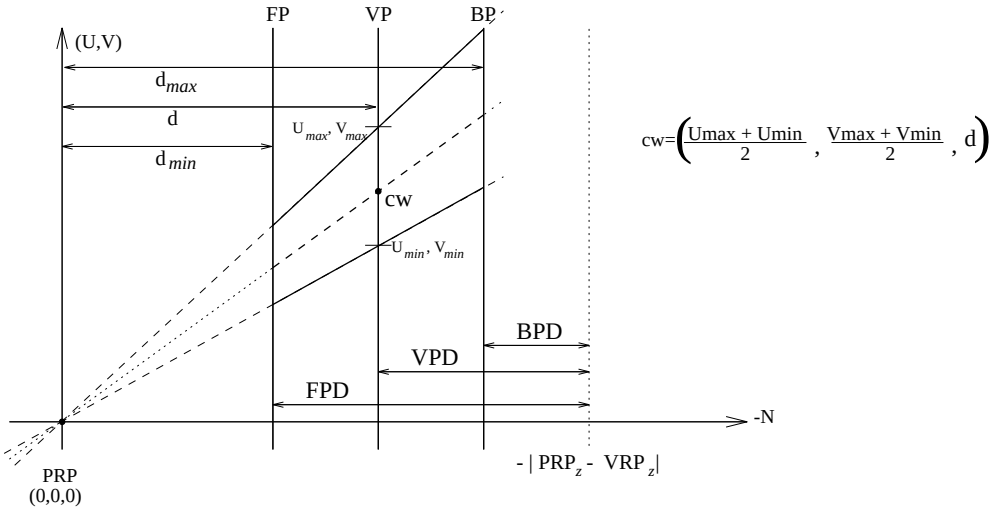


Abbildung 5.54: Translation

2.) Scherung, so daß die Sichtgerade (bestimmt durch PRP und CW) auf die VPN-Achse abgebildet wird:

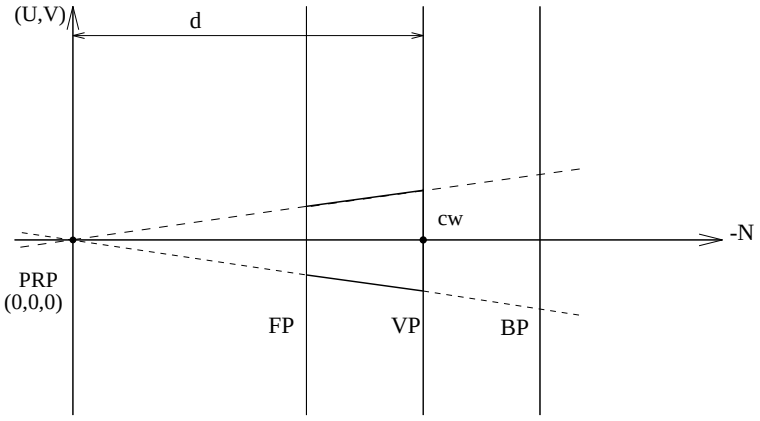


Abbildung 5.55: Scherung

3.) Skalierung, so daß die Eckpunkte des View-Fensters $U_{min/max}, V_{min/max}$ wegen des Öffnungswinkels von 90° der normierten Sichtpyramide auf $-|d|, |d|$ abgebildet werden:

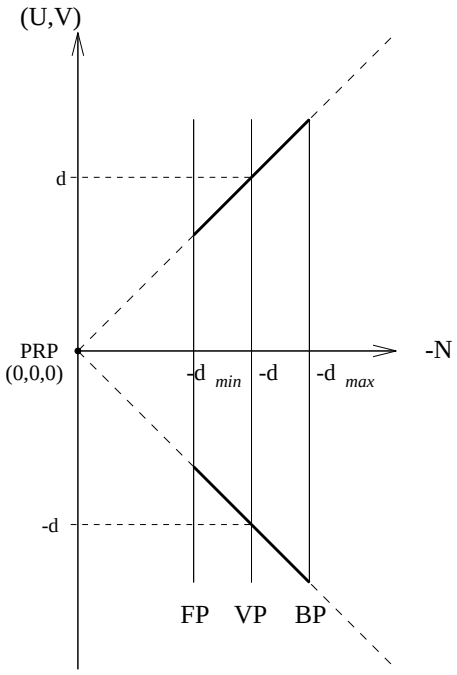


Abbildung 5.56: Skalierung

4.) Transformation, so daß die VP mit der $U,V,0$-Ebene übereinstimmt:

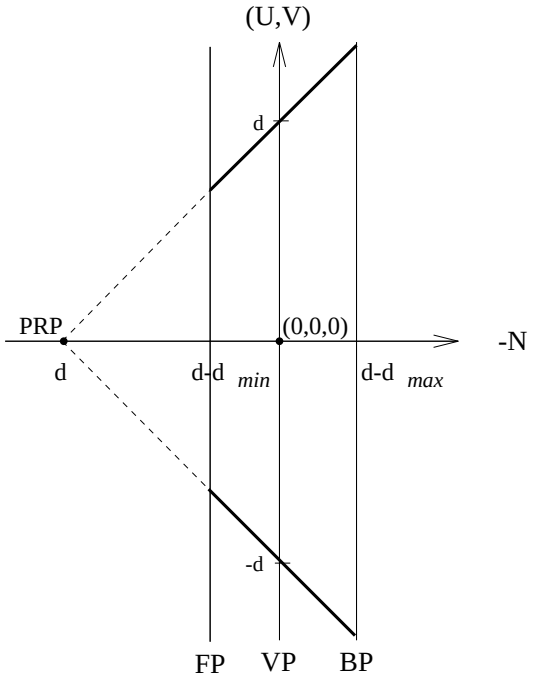


Abbildung 5.57: Transformation

5.) Perspektivische Transformation entlang der $(-N)$-Achse:

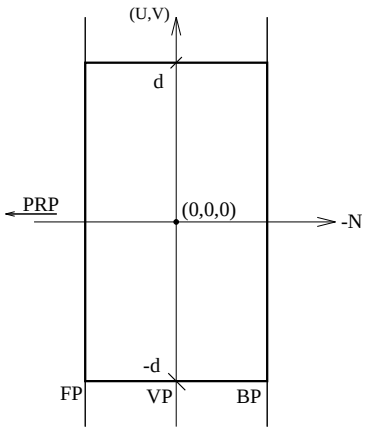


Abbildung 5.58: perspektivische Transformation

Die Perspektive ist durch den Abstand d des PRP zur Bildebene VP festgelegt.
Bei der Perspektive werden die Ebenen

$$FP = d - d_{min} \qquad \text{bzw.} \qquad BP = d - d_{max}.$$

auf die Ebenen

$$FP = \frac{d(d-d_{min})}{d_{min}} \qquad \text{bzw.} \qquad BP = \frac{d(d-d_{max})}{d_{max}}$$

abgebildet, vorausgesetzt, das Sichtvolumen enthält nicht die Ebene $N = d$. Sonst tritt das wrap-around-Problem (vgl. Kap. 5) auf.

6.) Skalierung und Translation, so daß das Sichtvolumen mit dem Einheitswürfel übereinstimmt:
Nach der perspektivischen Transformation besteht das Sichtvolumen aus einem Quader, der nun so skaliert wird, daß ein Würfel mit Kantenlänge 1 entsteht. Dieser wird anschließend so transliert, daß er mit dem Einheitswürfel übereinstimmt.

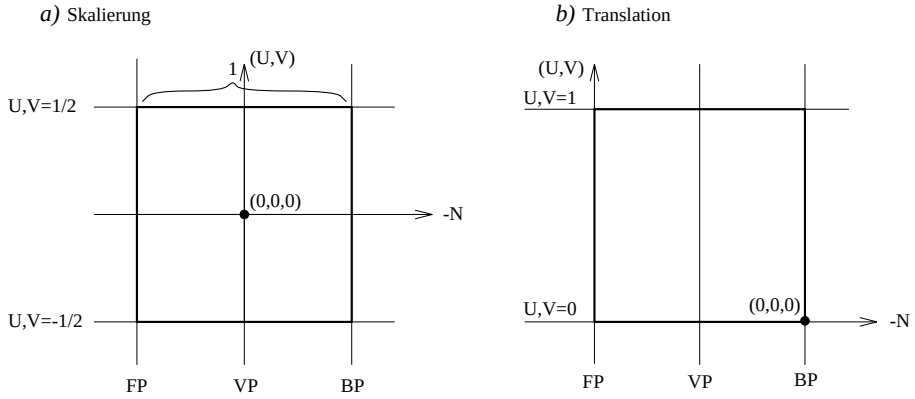


Abbildung 5.59: abschließende Skalierung und Translation

Literaturverzeichnis

[AEW90] Salim S. Abbi-Ezzi, Michael J. Wozny, *Factoring a Homogeneous Transformation for a more Efficient Graphics Pipeline*, Eurographics '90 (C. E. Vandoni, D. A. Duce, Eds.), North-Holland, September 1990, 159–175.

[App67] Arthur Appel, *The Notion of Quantitative Invisibility and the Machine Rendering of Solids*, Proc. ACM Natl. Mtg., 1967, 387.

[Car89] L. Carpenter, *The a–buffer, an antialiasing hidden surface method*, Computer Graphics, Vol. 18, 1989, Nr. 3, 103–108.

[CB78] M. Cyrus, J. Beck, *Generalized Two- and Three-Dimensional Clipping*, Computers & Graphics, Vol. 3, 1978, 23–28.

[Cla80] J. Clark, *A VLSI Geometry Processor for Graphics*, Computer IEEE, Vol. 13, 1980, 59–62, 64, 66–68.

[EC90] Gershon Elber, Elaine Cohen, *Hidden Curve Removal for Free Form Surfaces*, Computer Graphics (SIGGRAPH '90 Proceedings) (Forest Baskett, Ed.), Vol. 24, August 1990, 95–104.

[Enc75] J. L. Encarnaçao, *Computer-Graphics: Programmierung und Anwendung von graphischen Systemen*, R. Oldenburg Verlag, München, West Germany, 1975.

[GM69] R. Galimberti, U. Montanari, *An algorithm for hidden-line elimination*, Communications of the ACM, Vol. 12, 1969, 206.

[GMSG82] D. Greenberg, A. Marcus, A. H. Schmidt , V. Gort, *The Computer Image: Applications of Computer Graphics*, Addison-Wesley, 1982.

[Her92] I. Herman, *The Use of Projective Geometry in Computer Graphics*, Lecture Notes in Computer Science, Springer, 1992.

[HLRS85] C. Hornung, W. Lellek, P. Rehwald , W. Strasser, *An area-oriented analytical visibility method for displaying parametrically defined tensor-product surfaces*, Computer Aided Geometric Design, Vol. 2, 1985, Nr. 1-3, 197–205.

[Hor84] C. Hornung, *A Method for Solving the Visibility Problem*, IEEE Comput. Graphics and Appl. (USA), Vol. 4, 1984, 26–33.

[Kra89] G. Krammer, *Notes on the Mathematics of the PHIGS Viewing Pipeline*, Computer Graphics Forum, Vol. 8, 1989, Nr. 3, 219–226.

[Lou70] P. P. Loutrel, *A solution to the hidden-line problem for computer drawn polyhedra*, IEEE Trans. Comput., Vol. C-19, 1970, 205–213.

[NS79] W. M. Newman, R. F. Sproull, *Principles of Interactive Computer Graphics*, McGraw-Hill, New York, NY, 1979.

[PD84] Thomas Porter, Tom Duff, *Compositing digital images*, Computer Graphics (SIGGRAPH '84 Proceedings) (Hank Christiansen, Ed.), Vol. 18, July 1984, 253–259.

[RH84] P. Rehwald, C. Hornung, *An Analytical Visibility Method for Displaying Parametrically Defined Surfaces*, Eurographics '84 (K. Bo, H. A. Tucker, Eds.), North-Holland, 1984, 137–149.

[Rog85] D. F. Rogers, *Procedural Elements for Computer Graphics*, McGraw-Hill, 1985.

[Sch91] A. Schilling, *A new simple and efficient antialiasing with Subpixel masks*, Computer Graphics, Vol. 25, 1991, Nr. 4, 133–141.

[SH74] I. E. Sutherland, G. W. Hodgman, *Reentrant polygon clipping*, Communications of the ACM, Vol. 17, 1974, 32–42.

[SS68] Robert F. Sproull, Ivan E. Sutherland, *A Clipping Divider*, FJCC, 1968.

[Str74] Wolfgang Straßer, *Schnelle Kurven– und Flächendarstellung auf graphischen Sichtgeräten*, Ph.D. thesis, TU Berlin, 1974.

[WA77] K. Weiler, K. Atherton, *Hidden surface removal using polygon area sorting*, Computer Graphics (SIGGRAPH '77 Proceedings), Vol. 11, 1977, Nr. 2, 214–222.

[War69] J. Warnock, *A Hidden-Surface Algorithm for Computer Generated Half-Tone Pictures*, Tech. Report TR 4–15, NTIS AD-733 671, University of Utah, Computer Science Department, 1969.

[Wat70] G. S. Watkins, *A real-time visible surface algorithm*, Tech. Report UTEC-CS-70-101, Dept. Comput. Sci., Univ. Utah, Salt Lake City, UT, 1970.

[YDL84] B. A. Barsky Y.-D. Liang, *A New Concept and Method for Line Clipping*, ACM Trans. Graphics (USA), Vol. 3, 1984, 1–22.

Index

Glossar

Clipping (5.1, S.252)
Bei der Ausgabe von Bildern tritt oft das Problem auf, aus der gesamten vorhandenen Bildinformation nur einen Ausschnitt darzustellen. Um ein fehlerfreies Bild zu erhalten, muß die außerhalb des Fensters liegende Bildinformation vor der Bildausgabe abgeschnitten werden (clipping = Klippen). Bildelemente außerhalb des Fensters können zum Überlauf der Koordinatenadressierung des Gerätes führen (Wraparound).

- Der Cohen-Sutherland-Algorithmus (**Cohen-Sutherland-Clipping**) beschreibt das Clippen von Vektoren an rechteckigen Fenstern.
- Der **Weiler-Atherton-Algorithmus** beschreibt das Clippen von Polygonen an beliebigen Fenster-Polygonen.
- Der **w-Clip** löst das dreidimensionale Clippen in homogenen Koordinaten und löst elegant das Wraparound-Problem.

Cohen-Sutherland-Algorithmus (5.1.3, S.253)
Der Cohen-Sutherland-Algorithmus erlaubt das **Clippen** von Vektoren an rechteckigen Fenstern. Im zweidimensionalen Fall wird jedem Vektorendpunkt ein Outcode zugeordnet, mit Hilfe dessen einfach bestimmt werden kann, ob ein Vektor außerhalb oder innerhalb des Fensters liegt oder die Fensterbegrenzung schneidet. Im letzteren Fall müssen Schnittpunktberechnungen mit den Fenstergrenzen durchgeführt werden.

Weiler-Atherton-Algorithmus (5.1.4, S.257)
Der Weiler-Atherton-Algorithmus erlaubt das **Clippen** beliebiger Polygone an beliebigen Fensterpolygonen, indem er den zu clippenden Polygonzug sy stematisch verfolgt und die sichtbaren Gebiete des zu clippenden Polygons 'ausstanzt'.

w-Clipping (5.2.2, S.260)
Bei der perspektivischen Transformation gibt es endliche Punkte, die auf unendliche Punkte abgebildet werden, und unendliche Punkte, die auf endliche Punkte abgebildet werden. Somit können Liniensegmente auf unterschiedliche Teilsegmente abgebildet werden. Beim w-Clipping wird das Wraparound-Problem in euklidischen 4D-Koordinaten beschrieben und gelöst.

Visibilitätsverfahren (5.3, S.267)
Eine möglichst photorealistische Bilddarstellung setzt die korrekte Ermittlung sichtbarer und die Beseitigung unsichtbarer Bildteile voraus. Hierzu dienen Visibilitätsverfahren (Hidden-line-, Hidden-surface-Algorithmen).
Dem Einsatz des eigentlichen Visibiltätsverfahrens gehen einfache Verarbeitungsschritte voraus:

- Die perspektivische Transformation aller darzustellenden Objekte reduziert die Bestimmung sich gegenseitig verdeckender Punkte auf einen Vergleich ihrer z-Koordinaten.
- Die Rückseiten von Körpern werden, falls keine perspektivische Transformation durchgeführt wurde, anhand ihrer Oberflächen-Normalenvektoren eliminiert.

- Verfahren zur Punkteklassifizierung testen, ob ein Punkt innerhalb oder außerhalb eines geschlossenen Polygons liegt.
- Min/max-Tests testen, ob sich Polygone mit Sicherheit nicht verdecken.

Je nachdem, ob Punkte, Linien oder Flächen auf Verdeckung getestet werden, unterscheidet man

- punktorientierte Visibilitätsverfahren:
  z-Buffer-Algorithmus (Tiefenspeicher) und Varianten
- linienorientierte Visibilitätsverfahren:
  Watkins-Algorithmus
  regionenorientierter Algorithmus
- flächenorientierte Visibilitätsverfahren:
  Warnock-Algorithmus

z-Buffer-Algorithmus (5.3.5.1, S.271)

Der z-Buffer-Algorithmus ist ein punktorientiertes Visibilitätsverfahren.
Beim z-Buffer-Algorithmus werden alle Objekte der Szene nacheinander mit der Bildauflösung abgetastet. Alle Punkte des z-Speichers werden mit dem Wert 'unendlich', alle Punkte des Bildspeichers mit der Hintergrundfarbe vorbelegt; besitzt ein Objektpunkt einen z-Wert kleiner als den im z-Speicher aktuell eingetragenen, so wird dieser in den z-Speicher und seine Farbe in den Bildspeicher eingetragen.
Bei diesem Verfahren entsteht durch die Rasterung Aliasing. Das *A-Buffer-Verfahren* und das *EXACT-Verfahren* bestimmen genauer, welche Anteile eines Pixels von einzelnen Objekten überdeckt werden, und bestimmen aus diesen Anteilen den Farbwert des Bildspeichers wesentlich genauer.

Watkins-Algorithmus (5.3.6.2, S.281)

Der Watkins-Algorithmus ist ein linienorientiertes Visibilitätsverfahren.
Beim Watkins-Verfahren werden ebene Polygone auf gegenseitige Verdekkung getestet. Bestimmt wird die sichtbare Information auf einer Rasterzeile des Bildschirms. Um unnötige Schnittpunktberechnungen zu vermeiden, werden alle Polygonkanten entsprechend der y-Werte ihrer Endpunkte vorsortiert.

Regionenorientierter Visibilitäts-Algorithmus (5.3.6.4, S.285)

Regionenorientierte Visibilitätsverfahren zählen zu den linienorientierten Visibilitätsverfahren. In einem ersten Schritt werden die Ränder potentiell sichtbarer Gebiete (Regionen) ermittelt. Anschließend wird die Tatsache ausgenutzt, daß sich die Visibilität eines Objekts nur auf Regionenrändern ändern kann, es also ausreicht, die Visibilitätsverhältnisse auf den Rändern zu verfolgen.

Warnock-Algorithmus (5.3.7.1, S.289)

Der Warnock-Algorithmus ist ein flächenorientiertes Visibilitätsverfahren, bei dem geprüft wird, ob eine Objektfläche

- außerhalb einer Rasterfläche liegt,
- als einziges Objekt über der Rasterfläche liegt,

- der Rasterfläche am nächsten liegt und diese vollständig überdeckt.

Ist eine der Bedingungen erfüllt, liegt die Sichtbarkeit des Objekts fest, andernfalls wird die Rasterfläche weiter unterteilt.

Lehrziele

Nach dem Durchlesen sollten Sie

- wissen, warum und wo Software-Bibliotheken eingesetzt werden,
- die wesentlichen Eigenschaften der Software-Bibliothek Java3D kennen,
- die wesentlichen Eigenschaften der Schnittstellen-Spezifikation OpenGL kennen,
- den Zusammenhang von Java3D und OpenGL und deren prinzipielle Unterschiede kennen,
- wissen, was ein Szenegraph ist und wie ein Szenegraph in Java3D aufgebaut wird,
- die Struktur einfacher Java3D-Beispielprogramme beschreiben können,
- zu einem einfachen gegebenen Java3D-Beispielprogramm den zugehörigen Szenegraph darstellen und das Ergebnis des Beispielprogramms erkennen können,
- die grafischen Standard-Geometrien von Java3D und die Primitive von OpenGL kennen,
- die Prinzipien bei der Implementierung von Animationen mit Java3D kennen,
- die Rendering-Pipeline von OpenGL erläutern können,
- einen Überblick über die historische Entwicklung von Grafiksystemen gewonnen haben.

Kapitel 6

Grafik-Bibliotheken

In den vorigen Kapiteln dieses Kurses konnten Sie erfahren, wie dreidimensionale virtuelle Szenen modelliert werden und wie aus dem entstandenen Modell ein gerastertes zweidimensionales Bild generiert wird. In diesem Kapitel werden kommerzielle Software-Bibliotheken vorgestellt, in deren Routinen die wichtigsten Algorithmen zur Modellierung und Bildgenerierung implementiert sind.

6.1 Zweck und Anwendung von Grafik-Bibliotheken

Software-Bibliotheken bestehen aus einer Menge von Prozeduren bzw. Klassen und Methoden (im Folgenden Routinen genannt). Software-Bibliotheken sind in einem Programm aufrufbar bzw. deklarierbar. Sie dienen der Entlastung der Software-Entwickler, unter anderem weil sich die Entwickler nicht mit der Implementierung der Routinen auseinandersetzen müssen. Für ihre Arbeit ist die genaue Kenntnis der *Schnittstellen* sowie der Zusammenarbeit der Routinen in der Regel ausreichend. Die Schnittstellenbeschreibung einer Routine enthält die folgenden Angaben:

- Name
- Parameter
- Rückgabewert
- Wirkung/Funktion
- mögliche Fehlerbedingungen und die Reaktion der Routine auf diese Fehler

Die Schnittstelle einer Software-Bibliothek wird auch *API* (*application programming interface*) genannt. Wegen der großen Bedeutung der Schnittstelle wird in der Regel auch die gesamte Software-Bibliothek mit API bezeichnet.

Grafik-Bibliotheken sind Software-Bibliotheken, mit deren Routinen man komfortabel grafische Anwendungen erstellen kann. So können aus Anwendersicht

schon mit wenigen Zeilen Quellcode einfache dreidimensionale Anwendungen programmiert werden. In der Regel ermöglichen Grafik-Bibliotheken vor allem

1. die *Modellierung* von dreidimensionalen Objekten und Szenen aus Objekten und Lichtquellen,
2. die dreidimensionale *Darstellung* von Szenen aus Objekten und Lichtquellen auf einem Ausgabemedium (Bildgenerierung genannt) und
3. die *Interaktion* wie beispielsweise das Drehen von Objekten durch die Bewegung der Maus.

Grafik-Bibliotheken entlasten Software-Entwickler insbesondere aufgrund ihrer folgenden drei Eigenschaften:

- *Verbergen komplexer mathematischer Zusammenhänge*
  In den vorigen Kapiteln haben wir uns mit Hilfe der linearen Algebra der Thematik der Computer-Grafik genähert. Dabei wurden Algorithmen vorgestellt, mit denen aus modellierten Szenen Bilder generiert werden. Beispielsweise wurde ein Algorithmus für das Clipping und der Z-Buffer-Algorithmus erläutert. Grafik-Bibliotheken kapseln (verbergen) einen Großteil der Algorithmen in den Routinen, der Entwickler muss zum Benutzen der Routinen nur Kenntnisse über die Schnittstelle haben. Insbesondere wird die komplette Bildgenerierung (Rendering-Pipeline) inklusive Projektion, Clipping und Z-Buffering automatisch ausgeführt.
- *Plattformunabhängigkeit der Anwendungssoftware*
  Um plattformunabhängig zu sein, muß eine Anwendungssoftware zwei Bedingungen erfüllen: Sie greift nicht auf Betriebssystem spezifische Routinen wie beispielsweise die Routinen des Fenstermanagements zu, sondern statt dessen auf die Routinen von Software-Bibliotheken; äquivalente Software-Bibliotheken stehen auf verschiedenen Betriebssystemen bzw. Plattformen wie Windows, Mac, Linux und Unix zur Verfügung. Einmal geschrieben, kann die mit Hilfe von Software-Bibliotheken erstellte Software auf verschiedenen Rechnern genutzt werden. Die Bibliotheken sind plattformabhängig, die mit ihnen implementierten Programme jedoch nicht. Die Plattformunabhängigkeit wird auch als Portabilität bezeichnet.
- *Resourcen optimierte Nutzung der Grafik-Hardware*
  Anwendungssoftware greift i.d.R. über das Betriebssystem auf die Rechner-Hardware zu, beispielsweise bei Dateizugriffen. Greift die Anwendungssoftware bei Grafikoperationen nicht direkt auf Routinen des Betriebssystems zu, sondern auf Routinen einer Grafik-Bibliothek, ermöglichen diese Routinen in der Regel die Resourcen-optimierte Nutzung der Grafik-Hardware wie Grafikkarten. Grafik-Bibliotheken greifen wie auch Betriebssysteme auf die Grafik-Hardware mit Hilfe der zur Hardware zugehörigen Treiber zu.

Somit erhöhen Grafik-Bibliotheken die Wirtschaftlichkeit der Grafik- und Software-Entwicklung. Die Entwickler müssen nur noch die darzustellende Szene modellieren, die aus virtuellen Objekten und Lichtquellen besteht. Bei komplexen

Szenen wird zur Modellierung spezielle Software eingesetzt. Die Generierung eines gerasterten Bildes aus dem Modell übernehmen die Routinen der Grafik-Bibliotheken. Mit Hilfe des Betriebssystems, seines Fenstermanagement-Systems und der geeigneten Treiber wird das gerasterte Bild schließlich auf dem Bildschirm oder einem anderen Ausgabegerät dargestellt.

Die meisten Anwendungen, die dreidimensionale Szenen darstellen, nutzen Grafik-Bibliotheken. Dies gilt beispielsweise für CAD-Software für Ingenieure und Architekten, für visualisierte physikalische oder technische Simulationen, für Flugsimulatoren, Computer simulierte Auto-Crashtests sowie für Computer-Spiele.

6.2 Java3D

6.2.1 Was ist Java3D?

Java3D (auch mit Java 3D bezeichnet) besteht aus einer Hierarchie von Java-Klassen. Durch die Benutzung geeigneter Klassen können anspruchsvolle dreidimensionale Szenen modelliert werden. Eine derart programmierte Anwendung stellt nach ihrem Start die Szene in einem Fenster dar.

Java3D ist in Zusammenarbeit zwischen Sun, Silicon Graphics (SGI), Intel und Apple entstanden. Alle vier Unternehmen entwickelten vor einiger Zeit eigene Grafik-Bibliotheken. Diese Bibliotheken waren *plattformabhängig*, da sie bestimmte Hardware voraussetzten. Die einzelnen Unternehmen überlegten sich, ob es besser sei, die Erstellung jeweils einer Java-Version ihrer Grafik-Bibliothek zu diskutieren. Schließlich entschieden sie sich, eine einzige gemeinsame und kompatible Grafik-Bibliothek in Java aufzubauen. Es entstand das plattformunabhängige Java3D. Die ersten konkreten Schritte erfolgten 1997. Federführend bei der Entwicklung von Java3D ist das Unternehmen Sun, das auch die Programmiersprache Java ins Leben gerufen hat. Sun stellte es Personen oder anderen Unternehmen frei, selbst Java3D-Bibliotheken zu entwickeln, die jedoch der Schnittstellen-Spezifikation von Suns Java3D entsprechen müssen.

Java3D hat sich bisher noch nicht etablieren können. Gründe hierfür sind die Beschränkung auf die Programmiersprache Java sowie eine im Vergleich zu anderen Grafik-Bibliotheken geringere Darstellungsgeschwindigkeit.

Java3D ist nicht im *Java 2 JDK* (**J**ava **D**evelopment **K**it) enthalten, sondern eine Standard-Erweiterung (standard extension) vom Java 2 JDK: Die Klassen von Java3D erweitern die Klassenhierarchie von *Java 2*. Vor der Installation von Java3D ist zunächst Java 2 ab Version 1.2 zu installieren. Für Java3D muss neben Java auch OpenGL installiert sein. Ab Windows 95 Service Release 2 und ab Windows NT 4.0 (Service Pack 3) ist OpenGL vorinstalliert. Weitere Informationen zur Installation von Java3D und OpenGL unter Windows und unter Linux sind im Abschnitt 6.6 ’Installation von Java3D’ zu finden.

Im Folgenden wird Java3D in der Version 1.2 vom Mai 2000 beschrieben.

6.2.2 Wieso Java3D?

Für die Wahl von Java3D als Beispiel-Bibliothek für den vorliegenden Kurs sprechen zwei Gründe: Die Schnittstelle von Java3D ist in der benutzerfreundlichen Programmiersprache Java implementiert. Außerdem können mit Java3D komfortabler Grafik-Anwendungen erstellt werden als mit den meisten anderen kostenlos zu benutzenden Grafik-Bibliotheken. Ziel des Unterkapitels zu Java3D ist, einen Eindruck von der **Implementierung** einer Grafik-Anwendung zu geben.

Neben der Implementierung stehen die Begriffe „Szenegraph” und „Geometri-

en" im Mittelpunkt des Unterkapitels zu Java3D: Mit Java3D wird eine Szene im Rechner modelliert. Diese Szene besteht aus virtuellen „dreidimensionalen" Objekten (*Geometrien*) wie Linien, Polygonen und Kugeln. Alle die Szene definierenden Daten inklusive der Daten zu den Geometrien sind Teil einer baumartigen Struktur, des sogenannten *Szenegraphs*.

Weitere Themen sind die geometrische Transformation von Geometrien sowie die einfache Art der Interaktion mit einer Szene per Maus.

6.2.3 Inhaltliche Voraussetzungen

Dieses Kapitel über Java3D setzt grundlegende Kenntnisse in Java voraus. Zudem werden Algorithmen zur Bildgenerierung und weitere Inhalte aus vorigen Kapiteln dieses Kurses angewendet.

Als kostenlose Literatur empfehlen wir das „Java 3D Tutorial", auch „Collateral" genannt [Bou99]. Wie Sie die Dateien im PDF-Format erhalten, erfahren Sie im Abschnitt 6.8 'Literaturverzeichnis'. Unter anderem enthält das Tutorial ein empfehlenswertes Glossar.

6.2.4 Java3D und OpenGL

6.2.4.1 Grafik-Bibliotheken im Schichtenmodell

Die meisten Grafik-Bibliotheken enthalten keine ausreichenden Routinen für das Handhaben von Fenstern und Eingabegeräten wie Tastatur und Maus. Diese Routinen werden im Folgenden *Windowhandling-Routinen* genannt. Benutzt eine 3D-Anwendung eine solche Grafik-Bibliothek, muss sie zusätzlich zur Grafik-Bibliothek auf ein sogenanntes Fenstersystem aufsetzen, das die Windowhandling-Routinen zur Verfügung stellt.

Die Schnittstelle einer Grafik-Bibliothek zur Erstellung von dreidimensionalen Anwendungen oder aber die Grafik-Bibliothek selbst wird als *3D-API* bezeichnet.

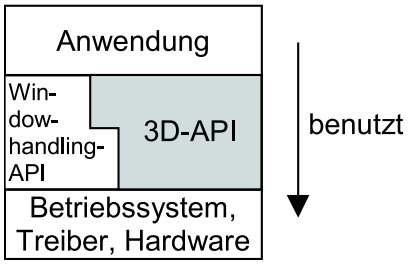


Abbildung 6.1: Schichtenmodell beim Einsatz der meisten Grafik-Bibliotheken (3D-API)

Die verschiedenen Fenstersysteme werden wie Grafik-Bibliotheken durch ihre Schnittstelle (Windowhandling-API) beschrieben. Bei Windows-Systemen könn-

te die Windowhandling-API beispielsweise aus den beiden nicht weiter beschriebenen APIs *Win API* und *WGL* bestehen. Bei Unix/Linux-Systemen könnte als Windowhandling-API *X Window (Xlib)* zusammen mit der API *GLX* eingesetzt werden, eventuell zusätzlich *KDE* oder *Motiv*. Der genauere Zusammenhang zwischen Fenstersystem bzw. Windowhandling-API und der Grafik-Bibliothek wird im Abschnitt 6.3.4.9 'Zusätzliche Bibliotheken' erläutert.

6.2.4.2 Die Grafik-Bibliothek Java3D im Schichtenmodell

Um Grafik-Bibliotheken nicht mit erheblichem Aufwand selbst entwickeln zu müssen, werden kommerzielle und nichtkommerzielle Grafik-Bibliotheken eingesetzt. Exemplarisch wird in diesem Kapitel die Grafik-Bibliothek „Java3D" vorgestellt. Java3D ist auch kommerziell ohne kostenpflichtige Lizensierung einsetzbar.

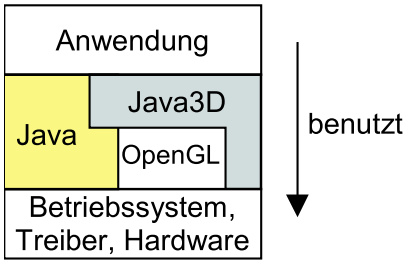


Abbildung 6.2: Schichtenmodell beim Einsatz der Grafik-Bibliotheken Java3D in Kombination mit OpenGL

Wie die Abbildung zeigt, wird die Windowhandling-API durch Java ersetzt. Anstatt der 3D-API wird eine Kombination von Java3D und OpenGL eingesetzt.

Java3D setzt auf der Grafik-Bibliothek *OpenGL* auf. OpenGL ist eine *Low-Level-Bibliothek*, die grundlegende Funktionalitäten bietet. Diese Funktionalitäten bestehen unter anderem aus dem Modellieren und Darstellen von Linien, gefüllten und nicht gefüllten Polygonen und Drahtgitternetzen. Um mit Hilfe von OpenGL komfortabel Grafik-Anwendungen zu implementieren, werden weitere Bibliotheken verwendet, die auf OpenGL aufsetzen. Eine solche *High-Level-Bibliothek* ist Java3D.

Das Schichten-Modell offenbart einen Vorteil von Java3D: Der Anwendungsprogrammierer muss sich ausschließlich mit den gut dokumentierten Schnittstellen von Java und von Java3D auseinandersetzen.

6.2.5 Implementierung (Programmierung und Syntax), Teil 1

6.2.5.1 Allgemeiner Ablauf bei der Implementierung

Stark vereinfacht gibt es zwei Vorgehensweisen, um mit Hilfe von Java3D (oder OpenGL) eine Anwendung zu implementieren:

Variante A:

1. Schritt: Es wird eine dreidimensionale Szene mit Hilfe geeigneter Aufrufe von Routinen der Grafik-Bibliothek programmiert. Dabei wird festgelegt, welche Objekte mit welchen Eigenschaften wo in der Szene positioniert werden. Außerdem wird die Beleuchtung sowie der Augpunkt und die Blickrichtung des Betrachters definiert. Eventuell kommen weitere Angaben hinzu, die Effekte wie Animationen und Interaktionen definieren.

2. Schritt: Das Programm wird compiliert.

3. Schritt: Das Programm wird ausgeführt. Die programmierte Szene wird (in der Regel auf einem Bildschirm) dargestellt.

Variante B (bei OpenGL wird dazu eine Zusatz-Bibliothek benötigt):

1. Schritt: Es wird mit Hilfe geeigneter Aufrufe von Routinen der Grafik-Bibliothek ein Programm erstellt, das eine bestimmte Datei lädt. Die Datei enthält Daten, die eine dreidimensionale Szene definieren, siehe den 1. Schritt bei Variante A.

2. Schritt: Das Programm wird compiliert.

3. Schritt: Das Programm wird ausgeführt. Die durch die Datei definierte Szene wird (in der Regel auf einem Bildschirm) dargestellt.

Bei dem folgenden Beispiel wird die Variante A beschrieben. Bei allen weiteren Java3D-Beispielen konzentrieren wir uns ausschließlich auf den 1. Schritt der Variante A. Die Variante B kommt in der Praxis häufig zum Einsatz, insbesondere bei komplexen Szenen. Aus Platzgründen wird jedoch in diesem Kurs darauf verzichtet, auf den Umgang mit Dateien und Dateiformaten einzugehen.

6.2.5.2 Beispiel Nr. 1: „Wuerfel1”, Teil 1

Zunächst wird das vermutlich einfachste Beispielprogramm vorgestellt, das mit Hilfe einiger Routinen von Java3D eine Szene darstellt. Das Programm öffnet ein Fenster (eine Instanz der Klasse `java.awt.Frame`) auf dem Bildschirm. In dem Fenster ist ein farbiger Würfel zu sehen.

Auf der CD-Rom zum Kurs sind die Quellcode-Dateien aller Beispiele dieses Kapitels zu finden. Einige der Beispiele zu Java3D sind dem Dokument „The Java3D Tutorial” der Firma Sun entnommen [Bou99]. In dem Tutorial von Sun werden die Beispiele noch ausführlicher erläutert.

Ausführen von Wuerfel1
Nachdem Java 2 und Java3D installiert sind, kann das Beispielprogramm ausgeführt werden. Wie das Beispielprogramm auf einem Windows 95/98- oder Windows Millenium-PC ausgeführt wird, ist im Folgenden beschrieben; bei anderen Betriebssystemen ist analog zu verfahren.

1. Kopieren Sie das Verzeichnis `k6` der CD auf ihre Festplatte.

2. Öffnen Sie eine DOS-Box
   (Start|Programme|MSDOS-Eingabeaufforderung).

3. Gehen Sie in der DOS-Box zum Verzeichnis der Festplatte mit den Beispielen:

   ```
   ... \k6\java3d\examples.
   ```

4. Geben Sie ein:

   ```
   javac Wuerfel1.java
   ```

   Falls es beim Compilieren Probleme gibt, kann dieser Schritt auch übergangen werden, da das Wuerfel1-Beispiel schon in einer compilierten Version (`Wuerfel1.class`) im gleichen Verzeichnis wie `Wuerfel1.java` liegt. Dieses class-File ist auf allen Plattformen mit installiertem Java 2, Java3D und OpenGL ausführbar.

5. Nach dem Compilieren ist die so erzeugte Binärcode-Datei auszuführen. Geben Sie dazu ein:

   ```
   java Wuerfel1
   ```

6. Nun öffnet sich ein Fenster mit schwarzem Inhalt. Nach kurzer Zeit wird ein rotes Quadrat dargestellt, siehe Abb. 6.3. Das Quadrat stellt die Seitenfläche eines Würfels dar. In Abbildung 6.13 ist der Würfel von der Seite zu sehen.

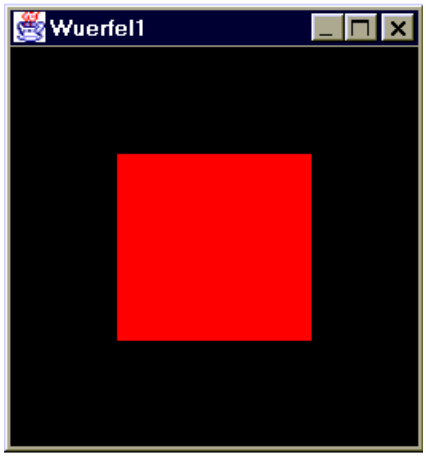


Abbildung 6.3: Beispiel Nr. 1: `Wuerfel1.java`
Zu sehen ist eine Seitenfläche des Würfels.

Quellcode von Wuerfel1
Im Quellcode des Beispiels sind die einzelnen Zeilen nummeriert. Bei Zeilen mit import-Anweisungen ist an die Zeilennummer ein „i” angehängt.

Es folgt der Quellcode von `Wuerfel1.java`.

```
01i.  import java.applet.Applet;
02i.  import java.awt.BorderLayout;
```

```
03i.  import java.awt.Frame;
04i.
05i.  import javax.media.j3d.BranchGroup;
06i.  import javax.media.j3d.Canvas3D;
07i.
08i.  import com.sun.j3d.utils.universe.SimpleUniverse;
09i.  import com.sun.j3d.utils.geometry.ColorCube;
10i.  import com.sun.j3d.utils.applet.MainFrame;

01.   public class Wuerfel1 extends Applet {
02.
03.     public Wuerfel1() {
04.       setLayout(new BorderLayout());
05.        Canvas3D canvas3D = new Canvas3D(
                SimpleUniverse.getPreferredConfiguration());
06.        add("Center", canvas3D);
07.
08.       SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
09.       simpleU.getViewingPlatform().setNominalViewingTransform();
10.
11.       BranchGroup scene = createSceneGraph();
12.
13.       scene.compile();
14.       simpleU.addBranchGraph(scene);
15.     }
16.
17.     public BranchGroup createSceneGraph() {
18.       BranchGroup rootBg = new BranchGroup();
19.        rootBg.addChild(new ColorCube(0.4));
20.       return rootBg;
21.     }
22.
23.     public static void main(String[] args) {
24.       Frame frame = new MainFrame(new Wuerfel1(), 256, 256);
25.     }
26.   }
```

Allgemeiner Aufbau der Beispielprogramme

Die Klasse `Wuerfel1` besteht wie alle Beispiele dieses Kapitels aus drei Methoden:

- Aus dem Konstruktor (bei diesem Beispiel `Wuerfel1`),
- der Methode `createSceneGraph` und
- der Methode `main`, die der Startpunkt des Programms ist.

Beim Ausführen des Programms wird automatisch die Methode `main` aufgerufen, die den Konstruktor aufruft, welcher wiederum `createSceneGraph` aufruft.

Anschaulich ausgedrückt, wird im Konstruktor der virtuelle Raum definiert, in dem sich die virtuellen Objekte der Szene befinden. Außerdem werden der Augpunkt des Betrachters und seine Blickrichtung festgelegt. Da in diesem Kapitel stets der gleiche „Standard-Raum" und default-Werte für den Augpunkt und die Blickrichtung gewählt werden, unterscheiden sich die Konstruktoren der einzelnen Beispiele nur durch ihren Namen. Auch die Methode `main` bleibt nahezu unverändert.

In `createSceneGraph` fügt der Entwickler den Quellcode ein, der die Szene definiert. Dabei werden alle Geometrien mit ihren Eigenschaften sowie eventuelle Lichtquellen modelliert.

Betrachtung einzelner Code-Passagen
Bevor wir uns genauer mit dem gesamten Beispielprogramm auseinandersetzen, befassen wir uns zunächst mit den fett gedruckten Stellen:

Zeile 01: Alle dargestellten Beispiele beginnen mit der Definition einer neuen Klasse (hier `Wuerfel1`), die von der Klasse `Applet` abgeleitet ist. Zwar könnten alle Beispiele anstatt als Applet auch als Application implementiert werden, jedoch sind mit Applets wesentlich einfacher Programme zu erstellen, die ein Fenster (Frame) öffnen und darin Menüs oder Ausgaben darstellen.
Übrigens haben die von uns in dieser Kurseinheit verwendeten englischen Begriffe wie „Application" in ihrem hier verwendeten Kontext eine spezielle Bedeutung, die durch das Übersetzen in einen deutschen Begriff nicht exakt zu erfassen ist.

Zeilen 05 und 06: In Zeile 05 wird die Instanz `canvas3D` von `Canvas3D` erzeugt. `Canvas3D` ist von der Klasse `Canvas` aus dem Paket `java.awt` (Abstract Windowing Toolkit) abgeleitet. Die Instanz `canvas3D` hat die Funktion eines Containers, der dreidimensionale GUI-Komponenten aufnimmt. GUI bedeutet graphische Benutzerschnittstelle (graphic user interface). Beispiele für zweidimensionale GUI-Komponenten sind grafische Benutzeroberflächenelemente wie Buttons oder Menüs. Neben solchen zweidimensionalen GUI-Komponenten können Instanzen der Klasse `Canvas3D` 3D-bezogene visuelle Objekte aufnehmen wie Dreiecke im Dreidimensionalen, Kugeln oder ein- und mehrfarbige Würfel.

In dem Beispiel „Wuerfel1" wird die dreidimensionale Szene, in der sich der Würfel befindet, in `canvas3D` aufgenommen. Zeile 06 bewirkt das Einfügen von `canvas3D` in das Fenster des Applets.

Beachten Sie bitte, dass in Abschnitt 6.2.12 'Darstellung der Klassenhierarchien' zwei Klassenhierarchien abgebildet sind, die sämtliche im vorliegenden Kapitel benutzten Klassen enthalten.

Zeile 19: Für den Würfel wird eine Instanz der Klasse `ColorCube` (Farbwürfel) aus dem Paket `com.sun.Java3D.utils.geometry` erzeugt (Zeile 19) und im virtuellen Raum (`canvas3D`) positioniert. Wie das genau funktioniert, ist Thema des Abschnitts 6.2.8.2 'Beispiel Nr.5: „Wuerfel 1", Teil 2'.

Zeilen 23 bis 25: Wie beschrieben, ist die Klasse `Wuerfel1` abgeleitet von `Applet`. Dank der neuen Klasse `MainFrame` (siehe Zeile 23) kann `Wuerfel1` je-

doch als Application ausgeführt werden. Die Klasse `MainFrame` stammt aus dem Package `com.sun.Java3D.utils.applet`. Sie ist abgeleitet von der Klasse `java.awt.Frame`. So erzeugt der Konstruktor `MainFrame()` ein Fenster (Frame), in dem `Wuerfel1` dargestellt wird.

Dieses unnötig kompliziert erscheinende Konzept - Applet und Application zugleich - hat zwei Vorteile: Einerseits ist ein Applet einfach zu programmieren, andererseits ist eine Application ohne Internet-Browser oder Applet-Viewer ausführbar.

Genau genommen ist `Wuerfel1` vor allem eine Application mit der Besonderheit, dass sie von `Applet` abgeleitet ist. Das wird deutlich, wenn man sich die implementierten Methoden ansieht: Keine der Methoden der `Applet`-Klasse (z.B. `init()`, `start()`, `paint()`) wird aufgerufen oder überschrieben, jedoch wird die für eine Application typische Methode `main()` implementiert.

6.2.6 Eigenschaften von Java3D

Eine Auflistung der *grafischen* Eigenschaften von Java3D ist im Abschnitt 6.4 'Vergleich: Java3D und OpenGL' zu finden.

6.2.6.1 Benutzungsfreundlichkeit

Ein Vorteil von Java3D ist die Benutzungsfreundlichkeit:

1. Ähnlich wie bei Java werden dem Programmierer einige lästige und fehlerträchtige Aufgaben abgenommen, wie die Freigabe von reserviertem Speicher.

2. Bei der Erstellung und Darstellung von Geometrien kann mit sogenannten High-Level-Konstrukten (z.B. Objekte wie Quader, Kegel, Zylinder oder Kugeln) gearbeitet werden, anstatt ausschließlich mit Punkten, Linien, Polygonen und Drahtgitternetzen.

3. Bei der Programmierung mit Java3D werden die Objekte einer Szene hierarchisch und objektorientiert strukturiert. Die Daten der modellierten visuellen Objekte und Szenen können in Dateien gespeichert und aus Dateien geladen werden. Zudem benötigt man selbst zur Erstellung und Darstellung komplexer Geometrien in der Regel außer OpenGL keine weiteren Grafik-Bibliotheken. OpenGL ist Bestandteil einer immer größeren Anzahl von Betriebssystemen.

6.2.6.2 Plattformunabhängigkeit der mit Java3D erstellten Programme

Wie bereits im Abschnitt 6.2.1 'Was ist Java3D?' erwähnt, ist Java3D plattformunabhängig: Java3D ist seit 1999 für die Betriebssysteme Windows95/98, Win-

dows ME/NT/2000 und für Solaris (von Sun) verfügbar. MacOS, Linux, JavaOS und andere Betriebssysteme folgten. Wie bei Java ist die Java3D-Umgebung (z.B. das SDK) plattformabhängig, nicht jedoch ein mit Java3D erstelltes Programm (Application). Das Unternehmen Sun drückt die Eigenschaft der Plattformunabhängigkeit mit der Aussage „write once, run anywhere" aus.

Java3D stützt sich als High-Level-Grafik-Bibliothek teilweise auf plattformabhängige Low-Level-Grafik-Bibliotheken. Beispielsweise ruft unter Windows ein Teil der Java3D-Methoden passende Methoden von OpenGL auf. Eine Java3D-Version für Direkt3D, eine Grafik-Bibliothek von Microsoft für Windows-Betriebssysteme, ist in Arbeit.

Das Aufsetzen auf Low-Level-Bibliotheken hat zwei Gründe: Erstens müssen die Low-Level-Routinen für die einzelnen Plattformen (Betriebssysteme und Hardware-Ausstattungen) nicht noch einmal entwickelt werden, zweitens sind die Routinen der Low-Level-Grafik-Bibliotheken in C oder Assembler geschrieben und auf eine bestimmte Hardware abgestimmt. Das erhöht die Geschwindigkeit bei der Darstellung der Szene. So profitiert Java3D durch das Aufsetzen auf Low-Level-Bibliotheken beispielsweise von 3D-Beschleuniger-Karten. Ebenfalls aus Geschwindigkeitsgründen sind einige Methoden von Java3D nicht in Java, sondern in C geschrieben.

Java3D ist für den Einsatz auf nahezu allen Arten von Rechnern gedacht, vom einfachen PC bis hin zum spezialisierten 3D- Image-Generator für mehrere Millionen Euro.

Da Java3D auf die Bibliothek OpenGL oder Direct3D (auf einem Mac OpenGL oder QuickDraw3D) aufsetzt, laufen Java3D-Programme (Applications) auf allen Plattformen, auf denen eine solche Bibliothek und ein Java-Interpreter verfügbar ist.

6.2.7 Der Szenegraph von Java3D (Definition einer Szene)

6.2.7.1 Virtuelles Universum

Eine darzustellende Szene besteht in der Regel aus Geometrien wie beispielsweise unserem „Würfel" aus dem vorigen Beispiel. Die Szene existiert in Java3D in einem virtuellen Raum. Dieser Raum wird das *virtuelle Universum* genannt. Es gibt in einer Java3D-Application oder in einem Java3D-Applet stets mindestens ein virtuelles Universum. Applications oder Applets mit mehreren virtuellen Universen sind selten.

6.2.7.2 Beispiel Nr. 2: Haus, Teil 1

Angenommen, die darzustellende Szene besteht aus einem dreidimensionalen Haus (siehe Abbildung 6.4). Das Haus besteht aus einem Dach und vier Wänden. Das

Dach besteht aus zwei Hälften. Die vordere Wand besitzt eine Tür. Die beiden Dachhälften und je zwei gegenüberliegende Wände haben eine identische Form. Das Fundament des Hauses, die Fenster und die Farben bleiben unbeachtet.

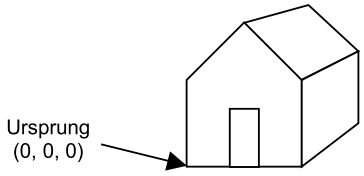


Abbildung 6.4: Beispiel Nr. 2 „Haus": Die Geometrie eines einfachen Hauses

Das 3D-Objekt „Haus" besteht also aus mehreren virtuellen geometrischen Objekten, bei Java3D Geometrien genannt. Die einzelnen Geometrien stehen bezüglich ihrer Form und ihrer Position zueinander in Beziehung. Solche Beziehungen sind typisch für nahezu alle dreidimensionalen Szenen. Dies gilt auch für unser Würfelbeispiel. Die dortigen Beziehungen betreffen die Lage der sechs Seitenflächen.

Die Beziehungen der Geometrien des Hauses lassen sich in Form einer Baumstruktur darstellen:

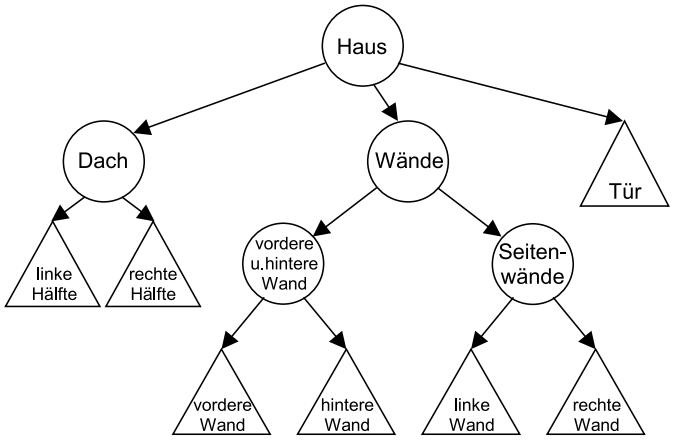


Abbildung 6.5: Beispiel Nr. 2 „Haus": Die Hierarchie der Geometrien eines einfachen Hauses

Die Blätter (leafes) des Baumes sind als Dreiecke dargestellt, die inneren Knoten (nodes) als Kreise. Statt von einem Baum wird auch von einem *gerichteten Graphen ohne Zyklen* gesprochen. Auch in Java3D werden aus einfachen Geometrien wie Vier- oder Fünfecken komplexere Geometrien wie ein Haus zusammengesetzt. Diese Graphen-Struktur wird bei Java3D Szenegraph genannt.

6.2.7.3 Beispiel Nr. 3: Zwei Zylinder (ein einfacher Szenegraph)

In Java3D werden die Geometrien einer Szene stets mit Hilfe eines Szenegraphen (scene graph) strukturiert. Abb. 6.6 zeigt einen einfachen Szenegraphen. Der Knoten mit dem Kreis-Symbol und der Beschriftung „BG" ist vergleichbar mit der Wurzel des vorigen Beispiels „Haus".

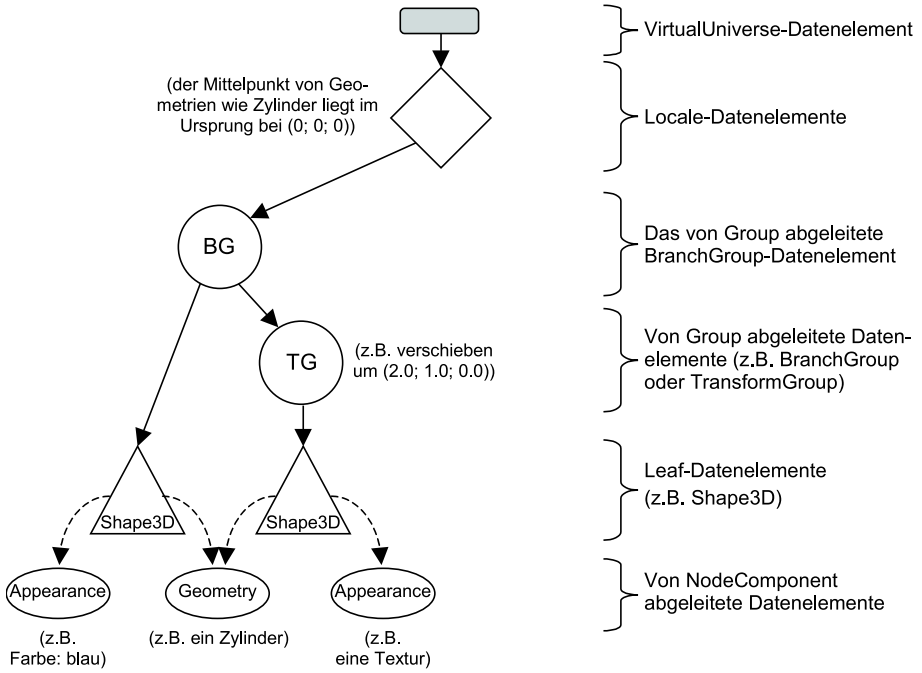


Abbildung 6.6: Beispiel Nr. 3 „Zwei Zylinder”: Ein Szenegraph, mit dem eine Szene, bestehend aus zwei Zylindern, definiert wird

Szenegraphen lassen sich als baumartige Struktur darstellen, wenn man von den gestrichelten Pfeilen auf die ellipsenförmigen Elemente unten in der Abbildung 6.6 absieht. Die Knoten der Struktur (die Wurzel und die anderen Knoten, hier Datenelement genannt) lassen sich in sechs Gruppen unterteilen. In der Abbildung liegen die Gruppen von Elementen auf jeweils einem horizontalen Bereich, der in der Abbildung rechts mit einer geschweiften Klammer gekennzeichnet ist. Bis zum kommenden Abschnitt sind jedoch die beiden obersten Bereiche „VirtualUniverse-Datenelement” und „Locale-Datenelement” nicht von Interesse.

Der Szenegraph aus der Abbildung beschreibt eine Szene mit zwei Zylindern. Der Szenegraph könnte so aussehen, wenn beispielsweise der eine Zylinder blau wäre, sein Mittelpunkt im Ursprung läge (Standard-Einstellung) und der andere Zylinder eine Textur trüge und um den Vektor (2, 1, 0) versetzt zum ersten Zylinder positioniert wäre. In den kommenden Abschnitten werden alle Elemente des in der Abbildung 6.6 dargestellten Szenegraphen ausführlich erläutert.

Den Zylindern sind die beiden mit „Shape3D” bezeichneten Dreiecke unten im Szenegraphen zugeordnet. Die Eigenschaften werden durch die drei Ellipsen darunter definiert. Die Form (Zylinder) und Größe (Durchmesser und Höhe) ist in der mittleren Ellipse festgelegt und für beide Zylinder identisch.

Wie der obige Szenegraph zeigt, sind in einem Szenegraphen Informationen wie der Typ und die Anzahl der in der Szene vorhandenen Geometrien (die Form der Objekte), ihre Position, Größe und Orientierung, ihre Farbe und weitere Oberflächeneigenschaften wie Texturen abgelegt. Zudem enthält ein Szenegraph auch alle Informationen zu den eventuellen Lichtquellen in der Szene und Informationen über den Sound in Form von Audio-Objekten. Hinzu kommt außerdem die

Position des Betrachters im virtuellen Raum und seine Blickrichtung. Ein Szenegraph enthält somit alle zur Bildgenerierung verwendeten Daten bis auf die Größe des Ausgabebereichs.

Ein Szenegraph besteht wie jeder Graph aus *Knoten* und *Kanten* und enthält in der Regel zusätzlich *Referenzen*.

6.2.7.4 Knoten

Die *Knoten* sind Datenelemente, implementiert durch Instanzen von Java3D-Klassen. In der Abbildung 6.6 werden ganz rechts die Knoten in sechs Gruppen eingeteilt. Fasst man die Gruppen *BranchGroup* und *Group* zu einer Gruppe zusammen, erhält man die fünf Arten von Knoten:

- *VirtualUniverse:*
  Die Wurzel des Szenegraphen ist bei Java3D keine Geometrie, sondern stets das virtuelle Universum, siehe 6.2.7.1 'Virtuelles Universum'.
- *Locale:*
  Das Locale-Datenelement ist ein Punkt im virtuellen Universum. Dieser Punkt hat die Aufgabe einer Landmarke: Die Position der Geometrien ist relativ zum Locale-Punkt. Der Locale-Punkt liegt für diese Geometrien also im Ursprung. Locale ist für alle Geometrien der Ursprung, die ein Element des Teilbaums sind, dessen Wurzel das Locale-Datenelement ist. Der Default-Wert des Punktes, bezogen auf das virtuelle Universum, ist (0.0; 0.0; 0.0). Wir werden ausschließlich Szenegraphen mit genau einem Locale-Datenelement behandeln.
- *Group* inklusive *BranchGroup* und *TransformGroup:*
  Die Hauptaufgabe eines Group-Datenelements ist das Gruppieren von anderen Datenelementen, die Söhne des Group-Datenelements sind. Die Group-Datenelemente halten demnach den Szenegraphen zusammen. Außerdem lohnt sich ein vorausschauendes Gruppieren auch aus Effizienzgründen, da teilweise Gruppen von Datenelementen aus Geschwindigkeitsgründen separat compiliert werden können. Falls später andere Teile des Szenegraphen zur Laufzeit verändert werden, ist ein erneutes Compilieren der Elemente der Gruppe nicht nötig.

In einer Anwendung, die auf Java3D aufsetzt, wird der Szenegraph objektorientiert implementiert. Dabei wird für jedes Datenelement (das heißt für jeden Knoten des Szenegraphen) eine Instanz (Objekt) einer bestimmten Klasse erzeugt. So wird das BranchGroup-Datenelement durch eine Instanz der Klasse `BranchGroup` repräsentiert. Anschließend werden die Kanten des Szenegraphen durch den Aufruf geeigneter Methoden erzeugt. Häufig beginnen diese Methoden mit „`add`", zum Beispiel die Methode `addChild`.

Kommen wir nochmals zum Group-Datenelement. Group ist eine abstrakte Klasse, weshalb nur davon abgeleitete Klassen benutzt werden können. In diesem Dokument verwenden wir ausschließlich die beiden abgeleiteten Klassen `Branch-`

`Group` und `TransformGroup`, weil der Einsatz der restlichen von `Group` abgeleiteten Klassen nur in speziellen Fällen sinnvoll ist. BranchGroup-Datenelemente dienen ausschließlich der Gruppierung und stellen eine Verzweigung (branch) dar. TransformGroup-Datenelemente dienen erstens der Gruppierung, zweitens werden die geometrischen Daten der gruppierten Datenelemente transformiert, indem eine vorher definierte und der TransformGroup zugeordnete Matrix mit ihnen multipliziert wird. Die obige Abbildung 6.6 enthält ein BranchGroup- und ein TransformGroup-Datenelement.

Leaf:
Ein Leaf-Datenelement steht für eine Geometrie. Es definiert (mit Hilfe des Datenelements NodeComponent) Aussehen (appearance), Geräusche (sound), Verhalten (behavior) und weitere Eigenschaften des visuellen Objekts.

Leaf ist eine abstrakte Klasse, weshalb nur davon abgeleitete Klassen benutzt werden können. In dem vorliegenden Dokument spielen insbesondere die beiden abgeleiteten Klassen `Shape3D` und `Behavior` eine Rolle. Mit `Shape3D` wird die Form der Geometrie festgelegt, z.B. Würfel, Zylinder oder Kugel. Mit `Behavior` werden Animationen und Interaktionen realisiert.

`Group` und `Leaf` sind die beiden Subklassen der Klasse `Node` (Knoten), siehe den folgenden Ausschnitt aus der Klassenhierarchie.

NodeComponent:
Diese Datenelemente stehen in enger Beziehung mit den Leaf-Datenelementen, in der Regel mit Shape3D-Objekten. Mit der NodeComponent-Klasse werden geometrische Eigenschaften, Aussehen, Textur und Material-Eigenschaften definiert. Eigentlich sind NodeComponent-Datenelemente nicht Teil des Szenegraphen, sondern die Shape3D-Datenelemente besitzen nur eine *Referenz* darauf. Da NodeComponent-Datenelemente jedoch einen entscheidenden Einfluss auf die Szene haben, sind sie ebenfalls Bestandteil der Abbildungen, die einen Szenegraphen darstellen.

Außerdem gibt es weitere Knoten-Arten, die hier jedoch nicht behandelt werden.

Die Elemente eines Szenegraphen bezeichnen wir als „Datenelemente”, um den mehrdeutigen Begriff „Objekt” zu vermeiden. Wird eine Szene programmiert, wird für jeden Knoten des Szenegraphen sowie für jedes NodeComponent-Element eine Instanz der betreffenden Klasse definiert. Beispielsweise wird für ein BranchGroup-Datenelement eine Instanz der Klasse `BranchGroup` definiert. Kanten und Referenzen werden durch jeweils einen Methodenaufruf implementiert. Bei Kanten ist das häufig die Methode `addChild`. Die bisher genannten Knoten bzw. deren zugeordnete Klassen sind in der Klassenhierarchie des Packages `java.media.j3d` enthalten. Weitere Klassen kann man durch zusätzliche Packages hinzufügen. Es folgt eine Ausschnitt aus der Klassenhierarchie:

```
Object
    VirtualUniverse
    Locale
    SceneGraphObject
```

```
Node
    Group
    Leaf
        Shape3D
NodeComponent
    Geometry
    Appearance
    Textur
    Material
```

Ein Ausschnitt der Klassenhierarchie von `java.media.j3d`, die alle in diesem Kapitel verwendeten Klassen enthält, ist in Abschnitt 6.2.12.1 'Klassenhierarchie von `javax.media.j3d`' zu finden. Eine vollständige Darstellung der Klassenhierarchie von `java.media.j3d` ist auf der CD-ROM zum Kurs in der Datei j3d `k6/java3d/klassenhierarchie.html` enthalten, siehe dazu Kapitel 6.7 'Die CD-ROM zum Kurs'.

6.2.7.5 Kanten

Jede *Kante* eines Szenegraphen steht für die hierarchische Beziehung zwischen zwei Datenelementen. Die Kanten sind vergleichbar mit den Kanten aus dem obigen Beispiel mit dem Haus, siehe Abb. 6.5. Die Kanten des Szenegraphen sind gerichtet und führen von einem Start- zu einem Ziel-Datenelement. Sie verbinden ausschließlich folgende Datenelemente miteinander, siehe dazu Abbildung 6.7:

- Vom VirtualUniverse können Kanten zu beliebig vielen Locales führen (genau eine Kante pro Ziel-Datenelement, hier Locale-Datenelemente).
- Von einem Local können Kanten zu beliebig vielen BranchGroups führen (genau eine Kante pro Ziel).
- Von einer BranchGroup oder TransformGroup können Kanten zu beliebig vielen anderen BranchGroups und/oder anderen TransformGroups und/oder Leafs führen (genau eine Kante pro Ziel).

Zudem ist eine weitere Regel zu beachten, damit aus dem Szenegraph ein Bild generiert werden kann: Zu jedem Datenelement führt genau eine Kante, nur zum VirtualUniverse führt keine Kante. Zyklen und nicht im Szenegraphen enthaltene Datenelemente sind demnach ausgeschlossen. Es handelt sich beim Szenegraphen um einen gerichteten azyklischen Graphen (directed acyclic graph, DAG); zusammen mit den obigen Einschränkungen ergibt sich eine Baumstruktur.

Kanten werden mit Methoden in den Szenegraphen eingefügt, die mit „add" beginnen. In dem Beispiel Nr. 1 (Wuerfel1), siehe Abb. 6.12, wird unter anderem eine Kante von einer BranchGroup zu einem Leaf-Datenelement erzeugt:

```
28.         BranchGroup rootBg = new BranchGroup();
29.         rootBg.addChild(new ColorCube(0.4));
```

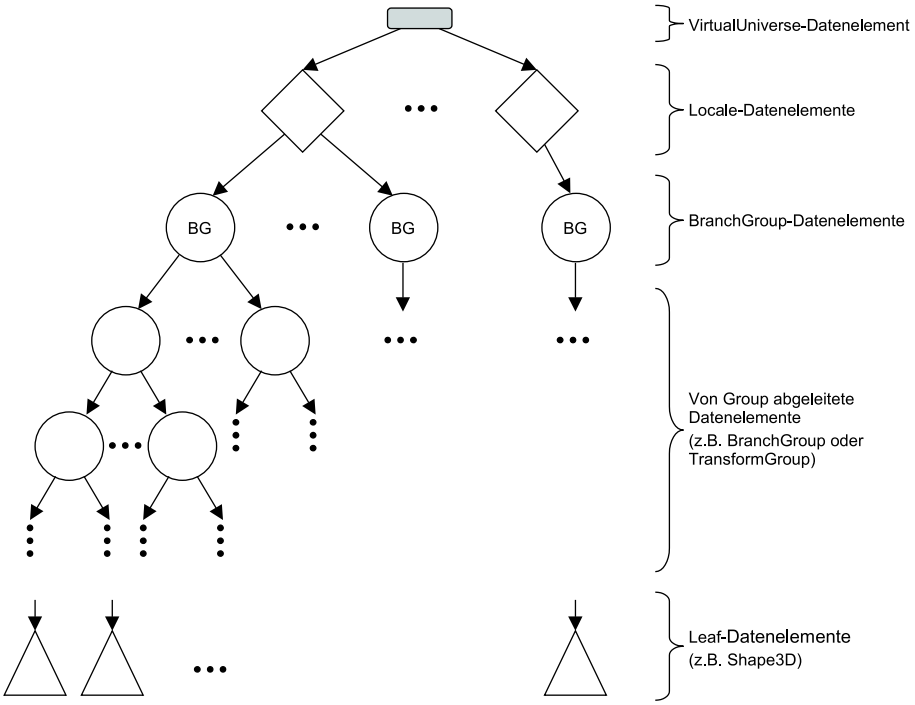


Abbildung 6.7: Ein allgemeiner Szenegraph ohne Referenzen

Das Leaf ist in diesem Fall eine Instanz von `ColorCube`. Die Klasse `ColorCube` ist eine Subklasse von `Shape3D`.

6.2.7.6 Referenzen

Bei der Erklärung des NodeComponent-Knotens wurde der Begriff „Referenz" eingeführt. Eine Referenz ist keine Kante und wird als gestrichelter Pfeil dargestellt. Eine Referenz kann nur von einem Leaf- zu einem NodeComponent-Datenelement führen. Von jedem Leaf können Kanten zu beliebig vielen NodeComponents führen (genau eine pro Ziel). Zu einem NodeComponent führt *mindestens* eine Referenz.

6.2.7.7 Traversierung des Graphen

Java3D generiert aus einer Szene das entsprechende gerasterte Bild, indem der Szenegraph von der Wurzel (VirtualUniverse-Datenelement) aus durchlaufen (traversiert) wird. Dabei wird für jedes Leaf-Datenelement der Pfad von der Wurzel zum Leaf-Datenelement ermittelt und werden die auf dem Pfad liegenden Datenelemente miteinander verknüpft.

Für jedes Leaf-Datenelement werden folgende fünf Schritte ausgeführt:

1. Es wird der Pfad von der Wurzel zum Leaf-Datenelement ermittelt. Von der

Wurzel zu jedem Leaf-Datenelement existiert genau ein Pfad. Ein Leaf ist damit eindeutig durch den zu ihm führenden Pfad definiert.

2. Die Datenelemente zwischen der Wurzel und den Leaf-Datenelementen sind in der Regel BranchGroup- und TransformGroup-Datenelemente. Die BranchGroup-Datenelemente in einem Pfad spielen keine Rolle und werden deshalb ignoriert. Die TransformGroup-Datenelemente in einem Pfad beeinflussen beispielsweise die Position und Größe der Geometrie des Leaf-Datenelements. Sie stehen für geometrische Transformationen wie Translation und Skalierung.

   Geometrische Transformationen werden in ihrer Reihenfolge im Pfad miteinander verknüpft. Jede Transformation wird Java3D-intern durch eine 4×4-Matrix repräsentiert. Die Verknüpfung der Transformationen geschieht durch Matrizenmultiplikation. Das Ergebnis ist wiederum eine Transformation bzw. Matrix.

   Bei der Verknüpfung der Transformationen ist die Reihenfolge zu beachten; Transformationen werden entsprechend ihrer Nähe zum Leaf-Datenelement ausgeführt: Liegt z.B. direkt über dem Leaf-Datenelement ein TransformGroup-Datenelement, das eine Rotation um die x-Achse um 10 Grad repräsentiert und liegt darüber ein weiteres TransformGroup-Datenelement, das eine Rotation um die y-Achse um 90 Grad repräsentiert, so wird die Geometrie des Leaf-Datenelements zunächst 10 Grad um die x-Achse und anschließend 90 Grad um die y-Achse gedreht. Die umgekehrte Reihenfolge hätte ein anderes Ergebnis.

3. Von den im Szenegraph enthaltenen Elementen werden nur die Geometrien dargestellt, also die Blätter (Leaf-Datenelemente). Diese Leaf-Datenelemente enthalten jedoch in der Regel keine für die Darstellung relevanten Informationen, der ColorCube ist da eine Ausnahme. Die Form wird hingegen von einer oder mehreren Referenzen festgelegt, beispielsweise, ob es sich um ein Dreieck oder eine Kugel handelt. Zunächst werden Körper mit Volumen wie Würfel, Zylinder, Kegel oder Kugeln in Flächen umgewandelt. Anschließend werden die nicht ebenen Flächen trianguliert.

4. Ab jetzt wird die Form einer Geometrie bzw. eines Leaf-Datenelements durch die Position seiner Eckpunkte oder die Position der Eckpunkte seiner Dreiecke definiert. Daraufhin werden die Eckpunkte bzw. die Vektoren zu diesen Eckpunkten mit der Matrix aus Schritt 2 multipliziert. So erhält man für jeden Eckpunkt den transformierten Eckpunkt.

   Hierbei ist eines zu beachten: Jede Transformation bzw. Matrix steht für eine affine Abbildung. Bei der Anwendung von affinen Abbildungen bleibt die „grundsätzliche" Form einer Geometrie erhalten, aus einem Dreieck kann kein Viereck oder eine Kugel werden.

5. Jede Geometrie kann neben den Referenzen für die Form auch eine Referenz für weitere Eigenschaften wie die Oberflächeneigenschaften (Texturen, Oberflächeneigenschaften für die Lichtreflexion und ähnliches) besitzen. Im letzten Schritt werden die durch die Referenzen festgelegten Eigenschaften auf die durch die transformierten Eckpunkte definierten Geometrien angewendet.

6.2.7.8 Inhalts- und Sichtgraphen

Bislang haben wir uns bezüglich des Szenegraphen ausschließlich mit der Frage beschäftigt, wie visuelle Objekte (Geometrien) mit ihren Eigenschaften in ein virtuelles Universum eingefügt werden. Es ging also um den Inhalt einer Szene.

Die andere Frage ist: „Von woher schauen wir wohin in die Szene?" Hierbei geht es also um die Position des Auges (Augpunkt), mit dem wir innerhalb der Szene die visuellen Objekte betrachten, und um die Richtung, in die wir blicken. Im Folgenden wird dies als „Sichtweise" oder kurz „Sicht" (view) bezeichnet. Diese Informationen und weitere technische Details sind ebenfalls Teil des Szenegraphen. Allerdings wurden sie bisher der besseren Übersicht wegen in den Abbildungen weggelassen.

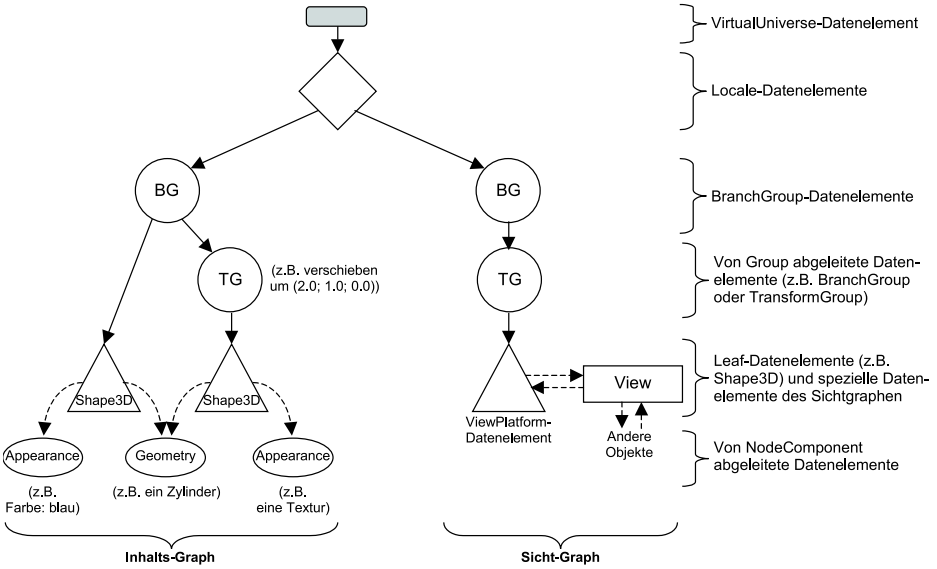


Abbildung 6.8: Beispiel Nr. 3 „Zwei Zylinder"
Im Gegensatz zur Abb. 6.6 wird der Szenegraph diesmal vollständig dargestellt. Er besteht nicht nur aus dem Inhaltsgraph, sondern aus Inhalts- und Sichtgraph.

Um die Teile des Szenegraphen eindeutig auseinanderhalten zu können, die einerseits den Inhalt und die andererseits die Sicht beschreiben, werden drei Begriffe eingeführt:

Wie oben bereits dargestellt, ist das VirtualUniverse die Wurzel des gesamten Szenegraphen. Im Unterschied dazu ist ein BranchGroup-Datenelement die Wurzel eines sogenannten **Verzweigungsgraphen** (branch graph). Es werden zwei Arten von Verzweigungsgraphen unterschieden:

Ein **Inhaltsgraph** (content branch graph) beschreibt ausschließlich den Inhalt der Szene.

Der **Sichtgraph** (view branch graph) beschreibt ausschließlich die Sichtweise.

In der Abbildung 6.8 wird erneut der Szenegraph aus Abb. 6.6 dargestellt, jedoch diesmal vollständig inklusive Sichtgraph.

Ein Szenegraph enthält genau einen Sichtgraphen und in der Regel mindestens einen Inhaltsgraphen.

In den Datenelementen des Sichtgraphen sind die Parameter bezüglich der Sichtweise spezifiziert. Zwei dieser Parameter sind die Position des Augpunktes und die Blickrichtung des virtuellen Betrachters in der Szene. Wir beschäftigen uns jedoch nicht weiter mit den Elementen des Sichtgraphen, denn statt des Sichtgraphen verwendet man bei einfachen Beispielprogrammen die im folgenden Absatz behandelte Klasse `SimpleUniverse`. Sie ist einfach zu handhaben, dafür jedoch unflexibel. Wer auf `SimpleUniverse` verzichten will, kann in der Spezifikation von Java3D ([SUN00], vgl. Abschnitt 6.8 'Literaturverzeichnis') Näheres zum Sichtgraphen erfahren.

6.2.7.9 SimpleUniverse

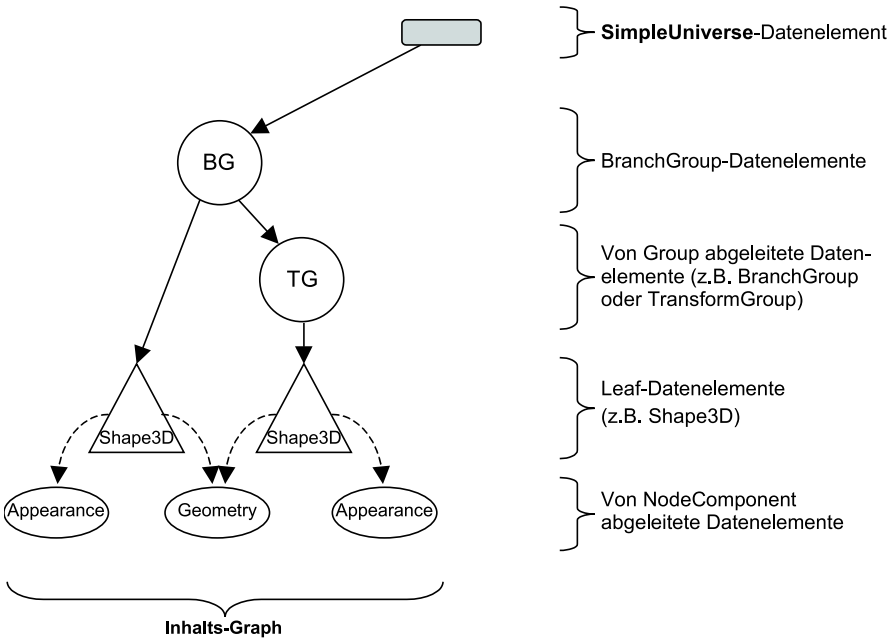


Abbildung 6.9: Beispiel Nr. 3 „Zwei Zylinder"
Mit SimpleUniverse wird der Szenegraph wesentlich übersichtlicher.

Die Klasse SimpleUniverse kapselt

- das VirtualUniverse-Datenelement,
- die Locale-Datenelemente und
- den Sichtgraphen.

Die Abbildung 6.9 macht die Vereinfachung deutlich.

6.2.7.10 Beispiel Nr. 4: Haus, Teil 2

In der Abbildung 6.10 wird der Szenegraph für das Beispiel „Haus” aus Abschnitt 6.2.7.2 ’Beispiel Nr. 2: Haus, Teil 1’ dargestellt. Dabei wurde auf die Darstellung der Referenzen und NodeComponent-Datenelemente verzichtet. Es wird angenommen, dass alle Shape3D-Datenelemente (hier sind es Rechtecke und Polygone) anfangs in der x-y-Ebene liegen und deren unterster linker Punkt im Ursprung liegt. Das Haus aus Abbildung 6.4 ist in der folgenden Abbildung nochmals dargestellt. Übrigens könnte die gleiche Szene auch mit völlig anders strukturierten Szenegraphen beschrieben werden.

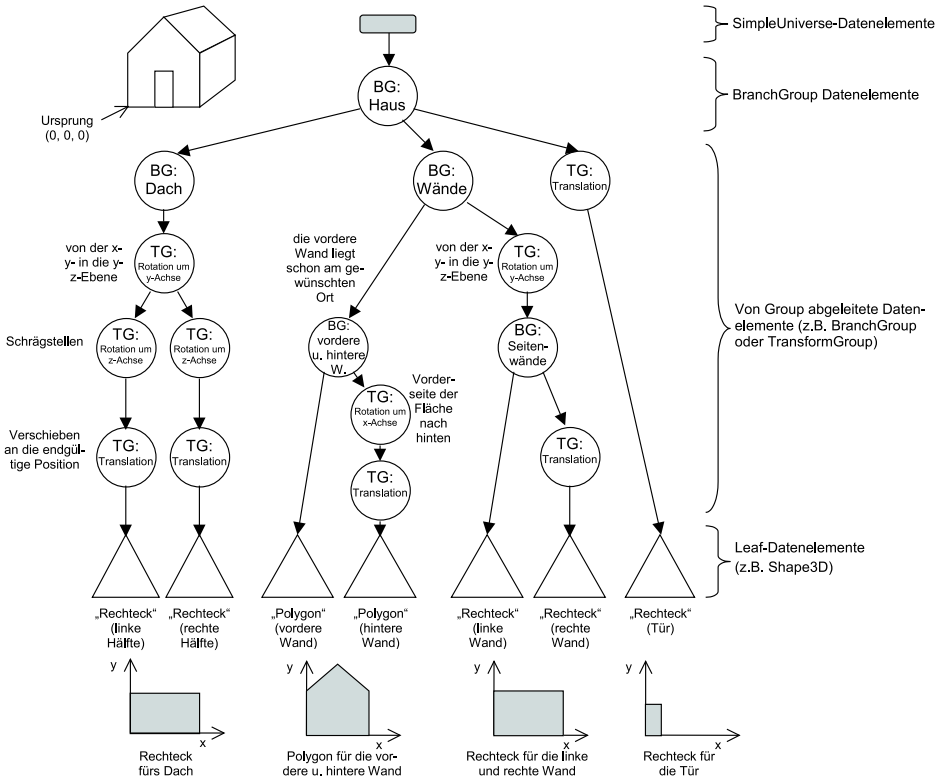


Abbildung 6.10: Beispiel Nr. 4 „Haus”
Auch einfache Szenen können zu komplexen Szenegraphen führen.

6.2.8 Implementierung (Programmierung und Syntax), Teil 2

In diesem Abschnitt befassen wir uns mit dem Implementieren von Programmen, die Java3D benutzen. Dabei wird in jedem Programm ein Szenegraph aufgebaut.

6.2.8.1 Kochrezept zum Erstellen von Java3D-Programmen

Das Kochrezept zum Erstellen von Programmen, die mit Hilfe von Java3D und insbesondere mit SimpleUniverse eine Szene darstellen, ist recht simpel:

1. Erzeugen einer Instanz der Klasse `Canvas3D`;

2. Erzeugen einer Instanz der Klasse `SimpleUniverse`. Dabei ist ein „Verweis" von der `SimpleUniverse`- zur `Canvas3D`-Instanz zu erstellen;

   a) Eventuell: Anpassen der `SimpleUniverse`-Instanz;

3. Aufbauen eines Inhaltsgraphen;

4. Compilieren des Inhaltsgraphen;

5. Einfügen des Inhaltsgraphen in das Locale-Datenelement der `SimpleUniverse`-Instanz.

Dieses Kochrezept wird im folgenden Abschnitt an Hand des Beispiels „Wuerfel1" erläutert.

6.2.8.2 Beispiel Nr. 5: „Wuerfel1", Teil 2

Es folgt der Quellcode von `Wuerfel1.java`. In Abschnitt 6.2.5.2 'Beispiel Nr.1: „Wuerfel 1", Teil 1' wurde der Quellcode bereits aufgelistet. Alle fünf Schritte aus dem vorigen Abschnitt werden im Konstruktor der Klasse `Wuerfel1` implementiert.

Die Zeilen 01i bis 09i enthalten die import-Anweisungen. Da diese Zeilen schon Teil des Quellcodes von „Wuerfel1" sind, werden sie im folgenden Quellcode nicht erneut abgedruckt.

```
01i. bis 09i.: ...

01.  public class Wuerfel1 extends Applet {
02.
03.    public Wuerfel1() {
04.      setLayout(new BorderLayout());
```

Schritt 1:

```
05.      Canvas3D canvas3D = new Canvas3D(
                    SimpleUniverse.getPreferredConfiguration());
06.      add("Center", canvas3D);
07.
```

Schritt 2:

```
08.      SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
```

Schritt 2 a):

```
09.      simpleU.getViewingPlatform().setNominalViewingTransform();
10.
```

Schritt 3, siehe auch unten die Methode createSceneGraph()*:*

```
11.      BranchGroup scene = createSceneGraph();
12.
```

Schritt 4:

```
13.         scene.compile();
```

Schritt 5:

```
14.         simpleU.addBranchGraph(scene);
15.      }

16.
17.      public createSceneGraph() {
18.         BranchGroup rootBg = new BranchGroup();
19.         rootBg.addChild(new ColorCube(0.4));
20.         return rootBg;
21.      }
22.
23.      public static void main(String[] args) {
24.         Frame frame = new MainFrame(new Wuerfel1(), 256, 256);
25.      }
26.   }
```

Die fünf Schritte definieren eine Szene mit einem Würfel. Dabei wird der Würfel im virtuellen Raum (`SimpleUniverse`) positioniert und in einem Fenster (`Canvas3D`) dargestellt.

Schritt 1 (Erzeugen einer Instanz der Klasse `Canvas3D`) :
Dieser Schritt wurde schon in Abschnitt 6.2.5.2 'Beispiel Nr. 1: „Wuerfel 1", Teil 1' behandelt.

Schritt 2 (Erzeugen einer Instanz der Klasse `SimpleUniverse`) :
Wie im Kochrezept beschrieben, wird ein „Verweis" (keine Referenz nach der bisherigen Terminologie) von der `SimpleUniverse`- zur `Canvas3D`-Instanz `canvas3D` erstellt. Damit wird festgelegt, dass die Ausgabe des generierten Bildes über das Objekt `canvas3D` erfolgt.

Schritt 2 a) (Anpassen der `SimpleUniverse`-Instanz) :
Dieser Schritt enthält mit dem Aufruf der Methode `getViewingPlatform` den einzigen Zugriff auf ein Datenelement aus dem Sichtgraphen. Dieses Datenelement ist eine Instanz der Klasse `ViewingPlatform`. Mit der parameterlosen Methode `setNominalViewingTransform` wird der Augpunkt des Betrachters auf

$$(0, 0, \sqrt{2}+1)$$

festgelegt, die Sichtrichtung ist die entgegengesetzte Richtung der positiven z-Achse. Der Augpunkt ist so gewählt, dass ein 2 Meter breites und hohes Quadrat

in der x-y-Ebene mit dem Mittelpunkt im Ursprung bei einem quadratischen Ausgabebereich den gesamten Bereich ausfüllt. Die dritte Koordinate des Augpunktes ergibt sich aus

$$\frac{1}{tan(\pi/8)} = \sqrt{2}+1 = 2,41...$$

Die folgende Abbildung 6.11 zeigt die Projektion eines Würfels auf die Bildebene und die Position des Augpunktes relativ zum Ursprung. Der Augpunkt befindet sich vom Ursprung aus hinter der Bildebene bei

$$(0, 0, \sqrt{2}+1).$$

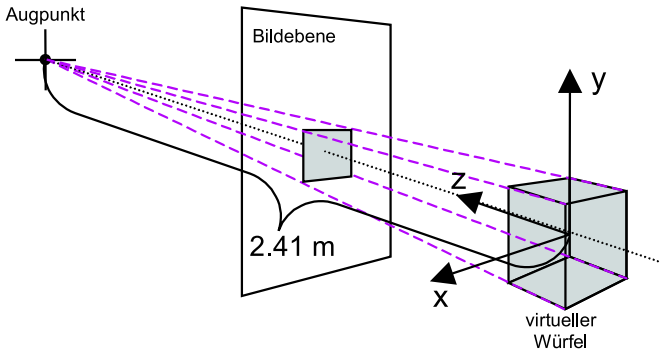


Abbildung 6.11: Beispiel Nr. 5 `Wuerfel1.java`
Die virtuellen Objekte werden auf die Bildebene (viewing plate) projeziert.

Schritt 3 (Aufbauen eines Inhaltsgraphen):
Die Instanz `scene` von `BranchGroup` dient als Wurzel für den Inhaltsbaum bzw. -graphen. Es entsteht im Schritt 3 und 5 der Szenegraph aus Abbildung 6.12.

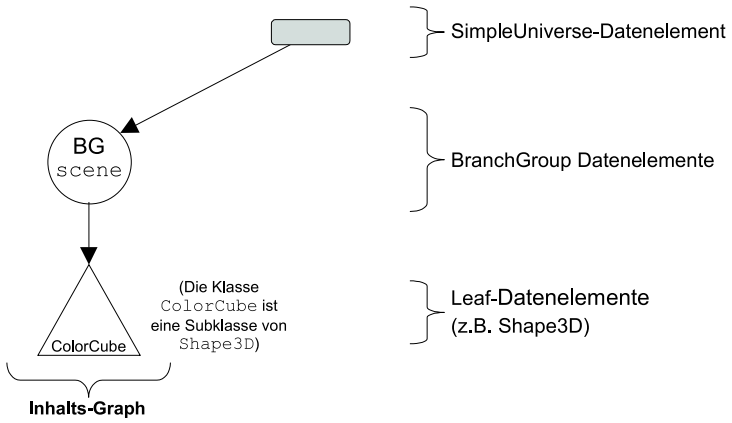


Abbildung 6.12: Beispiel Nr. 5 `Wuerfel1.java`
Der einfachste Szenegraph, mit dem eine Geometrie dargestellt werden kann

Ein BranchGroup-Datenelement kann nicht mehr verändert werden, sobald es an ein SimpleUniverse-Datenelement angehängt oder in einen bestehenden und mit dem SimpleUniverse-Datenelement verbundenen Inhaltsgraph eingefügt wird. Um die Eigenschaften eines BranchGroup-Datenelements trotzdem im Nachhinein

verändern zu können, muss vor dem Einfügen ein „Befähigungs”-Attribut (capability) des BranchGroup-Datenelements bzw. -Objekts entsprechend gesetzt werden. Im Abschnitt 6.2.11 ’Interaktion’ wird genauer darauf eingegangen.

Schritt 4 (Compilieren des Inhaltsgraphen):
Der Baum, dessen Wurzel `scene` ist, wird compiliert. Das teilweise oder vollständige Compilieren des Szenegraphen ist zwar nicht zwingend notwendig, da der gesamte Szenegraph beim Compilieren des Java-Programms ebenfalls compiliert wird. Das vorzeitige Compilieren dient jedoch dazu, Java3D das Optimieren der compilierten Strukturen zu ermöglichen.

Schritt 5:
Letztendlich wird der Inhaltsgraph an das SimpleUniverse-Datenelement angehängt.

Nach dem Abspeichern, Compilieren und Ausführen des Beispielprogramms `Wuerfel1` erhält man die in Abb. 6.3 dargestellte Ausgabe.

6.2.9 Transformation von Geometrien

6.2.9.1 Beispiel Nr. 6: „Wuerfel2” (Würfel in Schrägansicht)

Das Beispiel „Wuerfel2” zeigt den Würfel aus Beispiel „Wuerfel1” nach einer Drehung (rotation) um π/14 (1/28 Drehung) um die x-Achse und π/8 um die y-Achse.

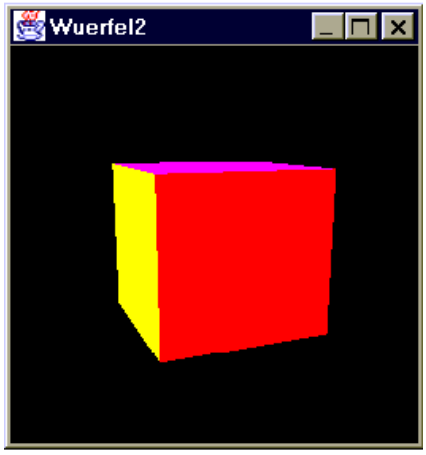


Abbildung 6.13: Beispiel Nr. 6 `Wuerfel2.java`
Im Unterschied zu „Wuerfel1” ist dieser Würfel vor der Darstellung um je einen konstanten Winkel um die x- und y- Achse gedreht worden.

Es folgt der Quellcode von `Wuerfel2.java`. Die anschließend behandelten Zeilen sind fett gedruckt.

```
01i.  import java.applet.Applet;
02i.  import java.awt.BorderLayout;
```

```
03i.  import java.awt.Frame;
04i.
05i.  import javax.media.j3d.Canvas3D;
06i.  import javax.media.j3d.BranchGroup;
07i.  import javax.media.j3d.Transform3D;
08i.  import javax.media.j3d.TransformGroup;
09i.
10i.  import com.sun.j3d.utils.universe.SimpleUniverse;
11i.  import com.sun.j3d.utils.geometry.ColorCube;
12i.  import com.sun.j3d.utils.applet.MainFrame;

01.   public class Wuerfel2 extends Applet {
02.
03.     public Wuerfel2() {
04.       setLayout(new BorderLayout());
05.       Canvas3D canvas3D = new Canvas3D(
                SimpleUniverse.getPreferredConfiguration());
06.       add("Center", canvas3D);
07.
08.       SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
09.       simpleU.getViewingPlatform().setNominalViewingTransform();
10.
11.       BranchGroup scene = createSceneGraph();
12.
13.       scene.compile();
14.       simpleU.addBranchGraph(scene);
15.     }
16.
17.     public BranchGroup createSceneGraph() {
18.       BranchGroup rootBg = new BranchGroup();
19.
20.        Transform3D rotate1 = new Transform3D();
21.        Transform3D rotate2 = new Transform3D();
22.
23.        rotate1.rotX(Math.PI/14.0d);
24.        rotate2.rotY(Math.PI/8.0d);
25.        rotate1.mul(rotate2);
26.
27.        TransformGroup rotateTg = new TransformGroup(rotate1);
28.
29.        rootBg.addChild(rotateTg);
30.        rotateTg.addChild(new ColorCube(0.4));
31.       return rootBg;
32.     }
33.
34.     public static void main(String[] args) {
35.       Frame frame = new MainFrame(new Wuerfel2(), 256, 256);
36.     }
37.   }
```

Zeile 07i und 08i: Mit den beiden Klassen `Transform3D` und `TransformGroup` wird die Drehung des `ColorCube` realisiert. `Transform3D` und `TransformGroup` werden in den folgenden Absätzen erläutert.

Zeile 20 bis 25: Das Beispiel `Wuerfel2` stellt einen um zwei Achsen gedrehten Würfel dar. Ein virtuelles Objekt wird gedreht, indem (wie bei jeder Transformation) die Eckpunkte des Objekts (in homogenen Koordinaten) mit einer geeigneten Matrix multipliziert werden. Die Transformationsmatrix A für die Drehung um zwei Achsen wird definiert, indem zunächst eine Transformationsmatrix A_1 für die Drehung um die erste Achse und anschließend eine Transformationsmatrix A_2 für die Drehung um die zweite Achse definiert werden. Werden die beiden Matrizen A_1 und A_2 in der richtigen Reihenfolge multipliziert, ergibt sich die gesuchte Matrix A. Laut Kurseinheit 3 'Affine und perspektivische Abbildungen', Abschnitt 3.3 'Affine und projektive Abbildungen in Matrizenschreibweise', ergibt sich folgende Reihenfolge:

$$A = A_1 \cdot A_2 \, .$$

Aus dem Eckpunkt p wird wie folgt der transformierte Eckpunkt p'' ermittelt:

$$p'' = p \cdot A_1 \cdot A_2 \, .$$

Stellvertretend für eine Matrix steht in Java3D die Klasse `Transform3D`. Sie wird nicht nur zur Repräsentation einer 4×4-Matrix genutzt, sondern bietet für die vier Transformationsarten Translation, Rotation, Skalierung und Scherung entsprechende Methoden an.

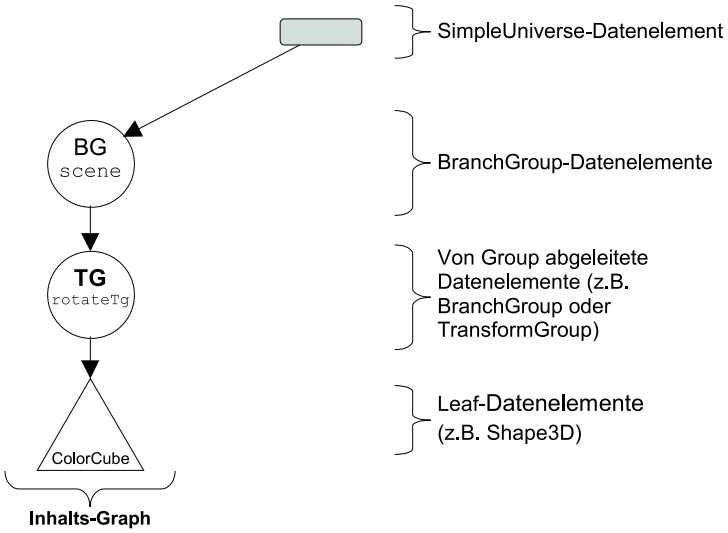


Abbildung 6.14: Szenegraph zum Beispiel `Wuerfel2.java`
Im Gegensatz zum vorigen Beispiel „Wuerfel1" soll bei „Wuerfel2" der ColorCube in einer gedrehten Lage dargestellt werden. Dazu wird der ColorCube nicht direkt an ein BranchGroup-Datenelement angehängt, sondern dazwischen wird ein TransformGroup-Datenelement eingefügt. Mit dem TransformGroup-Datenelement wird die Rotation des ColorCube realisiert.

In den Zeilen 20 und 21 werden zwei `Transform3D`-Instanzen (`rotate1` und `rotate2`) deklariert, die für die Matrizen A_1 und A_2 stehen.
In den Zeilen 23 und 24 werden die beiden Instanzen initialisiert. Der Würfel soll

einmalig um einen bestimmten Winkel um eine Achse rotiert werden.
In Zeile 23 wird mit der Methode `rotX` die Achse festgelegt und mit dem Parameter der Methode der Winkel. Ein Winkel von 2π entspricht einer ganzen Drehung, $1\pi/14$ etwa 25,7 Grad und $1\pi/8$ 45 Grad.
Nachdem in Zeile 24 auch die Drehung um die y-Achse definiert ist, werden in Zeile 25 beide Matrizen mit der Methode `mul` multipliziert. Anstatt eine neue `Transform3D`-Instanz für das Ergebnis zu erzeugen, wird das Ergebnis in `rotate1` abgelegt. Die Zeile 24 lässt sich mit Pseudocode wie folgt darstellen:

```
rotate1 := rotate1 * rotate2.
```

Zeile 27 bis 30: Nachdem die Transformationsmatrix ermittelt wurde, muss sie auf die Eckpunkte des virtuellen Würfels angewendet werden. Dazu wird eine `ColorCube`-Instanz definiert, die im Szenegraphen nicht wie bei dem Beispiel Wuerfel1 direkt an einen BranchGroup-Knoten angehängt wird, sondern an einen TransformGroup-Knoten. Diesen hängt man wiederum an den BranchGroup-Knoten, siehe Abbildung 6.14.

Mit einer `TransformGroup`-Instanz läßt sich eine Transformationsmatrix auf alle Söhne des `TransformGroup`-Instanz anwenden. Dazu wird in Zeile 27 eine `TransformGroup`-Instanz erzeugt und mit der Transformationsmatrix `rotate1` initialisiert.

6.2.10 Standard-Geometrien (GeometryArray und Geometry)

6.2.10.1 Beispiel Nr. 7: „Dreieck1" (Coloriertes Dreieck)

Bislang haben wir uns mit den Beispielprogrammen Wuerfel1 und Wuerfel2 beschäftigt. Beide stellen einen Würfel mit sechs verschiedenfarbigen Seitenflächen dar. Dazu haben wir die Klasse `ColorCube` benutzt. Instanzen dieser Klasse sind einfach zu handhaben, da die Geometrie wie auch die Farben der einzelnen Seitenflächen schon festgelegt sind. `ColorCube` eignet sich deshalb für schnelle Anfangserfolge beim Programmieren mit Java3D.

In der Praxis wird man Szenen eher mit anderen Geometrien wie Linien, Dreiecken, anderen Polygonen, Drahtgitternetzen sowie mit einfarbigen Würfeln, Kegeln, Zylindern und Kugeln modellieren.

Als Beispiel wird in dem folgenden Programm „Dreieck1" ein Dreieck definiert. Das Dreieck liegt in der x-y-Ebene. Den Ecken des Dreiecks wird jeweils eine Farbe zugewiesen. Die Farbwerte der anderen Pixel des Dreiecks werden automatisch durch Interpolation der Farben an den Eckpunkten ermittelt. Das Ergebnis sehen Sie in der Abbildung 6.15.

Der Szenegraph von „Dreieck1" erinnert an den von „Wuerfel1", siehe Abbildung 6.12 und die folgende Abbildung 6.16. Allerdings wird aus dem Leaf-Datenelement `ColorCube` das flexibel einsetzbare `Shape3D`. Mit Hilfe von `Shape3D` lassen

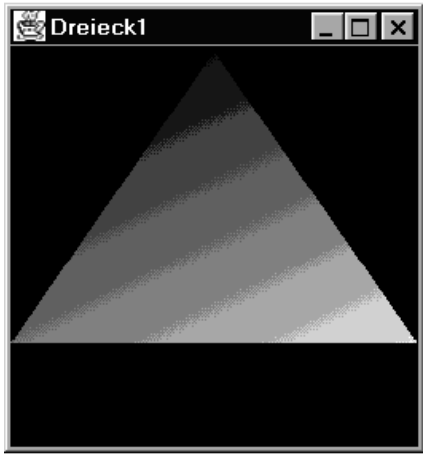


Abbildung 6.15: Beispiel Nr. 7, `Dreieck1.java`
Die Ecken des Dreiecks haben die Farben rot (linke Ecke), grün (rechts) und blau (oben).

sich alle im vorigen Absatz aufgelisteten Geometrien definieren. Zusätzlich zum `Shape3D`-Objekt muss eine Referenz vom `Shape3D`-Objekt zu einem `Geometry`-Objekt definiert werden. Mit dem `Geometry`-Objekt werden die Form und bestimmte Eigenschaften einer Geometrie definiert. In unserem Beispiel werden mit einem `Geometry`-Objekt die Art „Dreieck" sowie die Koordinaten und Farben der drei Eckpunkte festgelegt. Bei der Bildgenerierung wird das Dreieck automatisch coloriert.

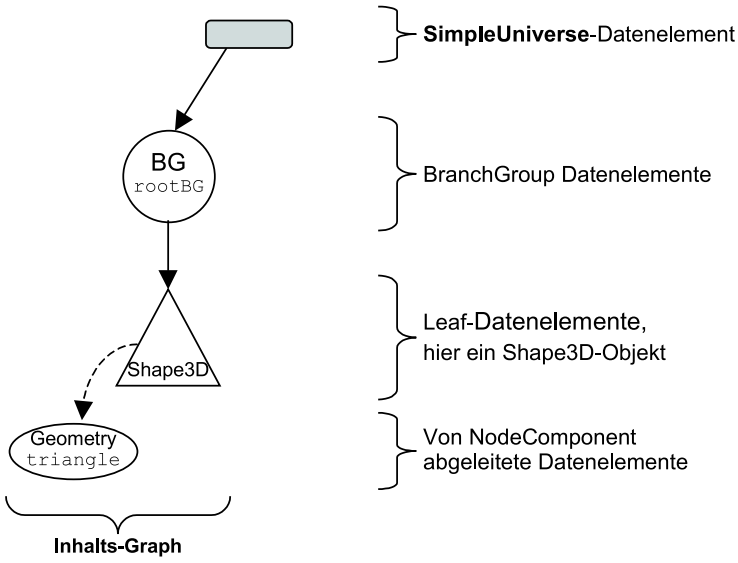


Abbildung 6.16: Szenegraph vom Beispiel `Dreieck1.java`
Nicht das Shape3D-, sondern das Geometry-Datenelement legt fest, dass ein Dreieck dargestellt wird und welche Position und Farbe die drei Eckpunkte haben sollen.

Es folgt der Quellcode von `Dreieck1.java`.

```
01i.  import java.applet.Applet;
02i.  import java.awt.BorderLayout;
03i.  import java.awt.Frame;
```

```
04i.
05i.  import javax.media.j3d.Canvas3D;
06i.  import javax.media.j3d.BranchGroup;
07i.  import javax.media.j3d.TriangleArray;
08i.  import javax.media.j3d.Shape3D;
09i.  import javax.vecmath.Point3f;
10i.  import javax.vecmath.Color3f;
11i.
12i.  import com.sun.j3d.utils.universe.SimpleUniverse;
13i.  import com.sun.j3d.utils.applet.MainFrame;

01.   public class Dreieck1 extends Applet {
02.
03.     public Dreieck1() {
04.       setLayout(new BorderLayout());
05.       Canvas3D canvas3D = new Canvas3D(
                SimpleUniverse.getPreferredConfiguration());

06.       add("Center", canvas3D);
07.
08.       SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
09.       simpleU.getViewingPlatform().setNominalViewingTransform();
10.
11.       BranchGroup scene = createSceneGraph();
12.
13.       scene.compile();
14.       simpleU.addBranchGraph(scene);
15.     }
16.
17.     public BranchGroup createSceneGraph() {
18.       BranchGroup rootBg = new BranchGroup();
19.
20.        TriangleArray triangle
21.           = new TriangleArray(3, TriangleArray.COORDINATES |
22.                                  TriangleArray.COLOR_3);
23.
24.        triangle.setCoordinate(0,new Point3f(-1.0f,-0.5f, 0.0f));
25.        triangle.setCoordinate(1,new Point3f( 1.0f,-0.5f, 0.0f));
26.        triangle.setCoordinate(2,new Point3f( 0.0f, 1.0f, 0.0f));
27.
28.        triangle.setColor(0,new Color3f( 1.0f, 0.0f, 0.0f ));//rot
29.        triangle.setColor(1,new Color3f( 0.0f, 1.0f, 0.0f ));//gruen
30.        triangle.setColor(2,new Color3f( 0.0f, 0.0f, 1.0f ));//blau
31.
32.        rootBg.addChild(new Shape3D(triangle));
33.       return rootBg;
34.     }
35.
36.     public static void main(String[] args) {
37.       Frame frame = new MainFrame(new Dreieck1(), 256, 256);
```

```
38.     }
39.   }
```

In der Methode `createSceneGraph` wird in drei Schritten das Dreieck mit Hilfe eines `Shape3D`-Objekts definiert und in den Szenegraphen eingefügt. Das `Shape3D`-Objekt steht für das virtuelle Objekt (Geometrie). Die Form einer Geometrie wird nicht durch das `Shape3D`-Objekt festgelegt, sondern durch eine Referenz auf eine Instanz einer von `Geometry` abgeleiteten Klasse.

Schritt 1: Zeile 20 bis 22:

In den Zeilen wird die Instanz `triangle` der von `Geometry` abgeleiteten Klasse `TriangleArray` definiert.

Aber zunächst ein paar Sätze zur allgemeinen Definition von Geometrien. Im Bereich der 3D-Grafik werden Geometrien vom Rechner in der Regel als eine *Menge von Eckpunkten* (vertices) mit deren Eigenschaften und mit deren Beziehungen untereinander gespeichert und verarbeitet. Deshalb müssen beim Beispiel „Dreieck1" nur die drei Eckpunkte und deren Farben definiert werden, Angaben zu den Kanten und zur Fläche des Dreiecks sind nicht notwendig. Beispielsweise ergibt sich aus der Reihenfolge der Eckpunkte, welche der beiden Seiten des Dreiecks die Vorderseite ist.

Als „*Eckpunkt-Datum*" bezeichnen wir im Folgenden einen Datentyp, der folgende Daten enthält:

- Die Position eines Punktes,
- optional: die Farbe des Punktes,
- optional: Normalenvektor an diesem Punkt und
- optional: die Texturkoordinaten (siehe Kurs 1693 'GDV II', KE 7 'Texturen', Abschnitt 7.1.2 'Texturabbildungen', dort werden die Texturkoordinaten (u, v)-Koordinaten genannt).

In Java3D gibt es keine Klasse, mit der man ein Eckpunkt-Datum implementiert, da dieses stets Teil einer *Liste von Eckpunkt-Daten* ist. Zur Definition solcher Listen steht die abstrakte Klasse `GeometryArray` zur Verfügung. Subklassen von `GeometryArray` sind `PointArray`, `LineArray`, `TriangleArray` und `QuadArray`. Diese Klassen kann man sich als Arrays vorstellen, bei denen jedes Feld ein Eckpunkt-Datum enthält.
Ein `PointArray` steht nur für eine Folge von Eckpunkten, die bis auf die Reihenfolge keine weitere Beziehung zueinander haben.
`LineArray` kann zur Definition von Linien eingesetzt werden: Je zwei aufeinanderfolgende Array-Einträge ergeben eine Linie; die Anzahl der Array-Einträge muss demnach gerade sein.
Bei `TriangleArray` ergeben je drei aufeinanderfolgende Array-Einträge ein Dreieck; die Anzahl der Array-Einträge muss demnach durch drei teilbar sein.

Mit `QuadArray` werden konvexe Polygone mit vier Eckpunkten dargestellt; die Anzahl der Array-Einträge muss durch vier teilbar sein.

Es folgt ein Auszug aus der Klassenhierarchie:

```
Object
     SceneGraphObject
          NodeComponent
               Geometry
                    GeometryArray
                         PointArray
                         LineArray
                         TriangleArray
                         QuadArray
                         ...
```

In Zeile 20 bis 22 wird eine Instanz (`triangle`) von `TriangleArray` deklariert und initialisiert. Der gewählte Konstruktor ist von `Geometry` geerbt, mit ihm werden zwei Parameter vom Typ `int` übergeben:

1. Die Gesamtanzahl der Eckpunkt-Daten, mit der man anschließend das Array füllen wird;
2. das Eckpunkte-Format (vertex format), mit dem festgelegt wird,
   - ob später die Koordinaten der Eckpunkte definiert werden,
   - ob später die Normalenvektoren an den Eckpunkten definiert werden und
   - ob später die Farben der Eckpunkte definiert werden,
   - ob später Texturkoordinaten definiert werden.

Für jede dieser Optionen existiert als Teil der Klasse `TriangleArray` eine Konstante. Will der Entwickler später beispielsweise die Position der Eckpunkte definieren, steht hierfür die Konstante `COORDINATES` zur Verfügung. Da man stets die Koordinaten angeben wird, wird die Konstante immer als Parameter der Funktion `TriangleArray` übergeben.

```
20.    TriangleArray triangle
21.      = new TriangleArray(3, TriangleArray.COORDINATES |
22.                          TriangleArray.COLOR_3);
```

Will der Entwickler später (allen Eckpunkten jeweils) eine Farbe zuordnen, übergibt er eine der beiden Konstanten `COLOR_3` und `COLOR_4`. Der Entwickler wählt `COLOR_4` statt `COLOR_3`, falls er zusätzlich zu den drei Farbwerten einen Wert für die Transparenz definieren will. Werden mehrere Konstanten übergeben, werden sie mit OR verknüpft. Die Konstanten sind unterschiedliche Integer-Werte, bei denen jeweils nur ein Bit gesetzt ist.

In Zeile 21 und 22 werden mit COORDINATES und COLOR_3 genau diese beiden Bits gesetzt, die anderen sind defaultmäßig nicht gesetzt.

Schritt 2: Zeile 24 bis 30:

Zu jedem der drei Eckpunkte werden die Koordinaten und Farben definiert. Der erste Parameter von `setCoordinate` und `setColor` legt die Position des Eckpunktes im `TriangleArray` fest. Die Positionen der Eckpunkte im `TriangleArray` (Array-Feld 0 bis 2) ergeben die Reihenfolge der Eckpunkte.

Schritt 3: Zeile 32:

Es wird ein neues Shape3D-Objekt erzeugt und in den Szenegraphen eingefügt, indem es an das BranchGraph-Datenelement `rootBg` angehängt wird:

```
32.         rootBg.addChild(new Shape3D(triangle));
```

Es folgt eine Überprüfung der Datentypen. Der gewählte Konstruktor von `Shape3D` erwartet einen Parameter des Typs `Geometry`. Wie bereits angesprochen, ist `triangle` eine Instanz einer Subklasse von `Geometry`. Die Methode `addChild` in Zeile 30 benötigt einen Parameter des Typs `Node`. Eine Subklasse von `Node` ist `Leaf` und eine Subklasse von `Leaf` ist `Shape3D`.

6.2.10.2 Alle Standard-Geometrien ohne Volumen im Überblick

In diesem Abschnitt werden diejenigen Standard-Geometrien behandelt, die ausschließlich aus Punkten, Linien und ebenen Flächen bestehen. Diese volumenlosen Geometrien sind in der folgenden Abbildung 6.17 zu sehen.

Es folgt die Klassenhierarchie der volumenlosen Standard-Geometrien. Alle Klassen sind Teil des Packages `javax.media.j3d`.

```
Object
     SceneGraphObject
          NodeComponent
               Geometry
                    GeometryArray
                         PointArray
                         LineArray
                         TriangleArray
                         QuadArray
                         GeometryStripArray
                              LineStripArray
                              TriangleStripArray
                              TriangleFanArray
```

Alle diese Standard-Geometrien werden analog zum Beispiel „Dreieck1" definiert. Beim Definieren von Standard-Geometrien werden unter anderem die Eckpunkte der Geometrie aufgelistet. Wie die folgende Abbildung zeigt, ist dabei die Reihenfolge zu beachten, in der die Eckpunkte aufgelistet werden. Beispielsweise werden Geometrien der Klasse `QuadArray` (Vierecke) nur dann dargestellt, wenn

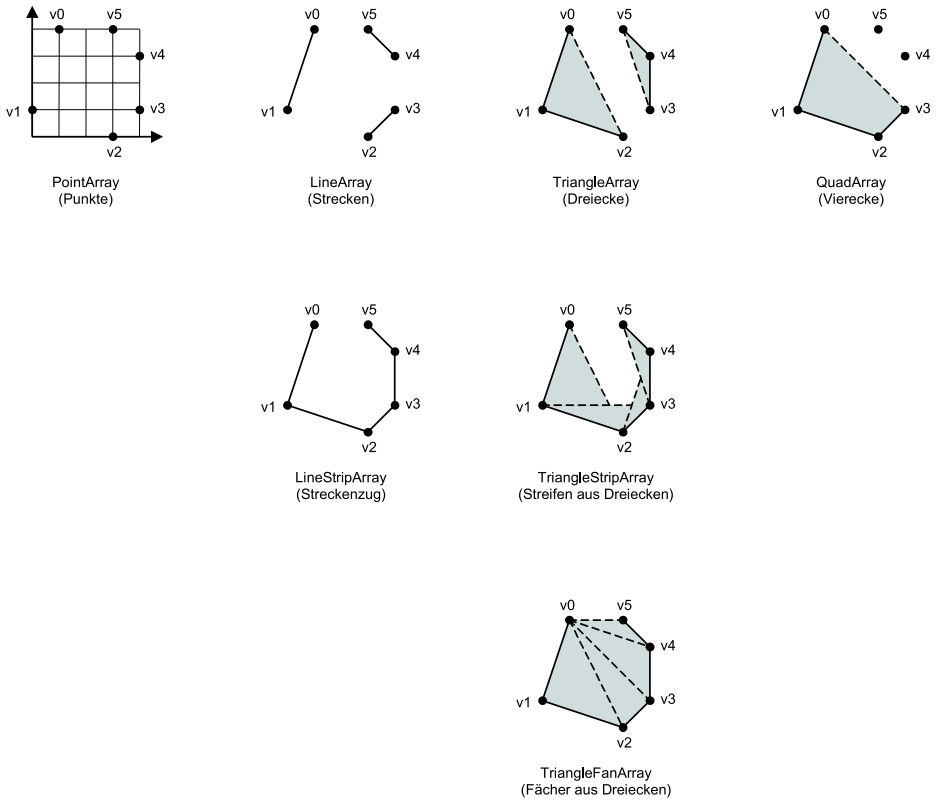


Abbildung 6.17: Diese sieben volumenlosen Standard-Geometrien stehen bei Java3D zur Verfügung. Die gestrichelt dargestellten Kanten werden automatisch hinzugefügt. Zur Orientierung werden bei allen Geometrien die Position der Eckpunkte durch schwarze kleine Kreise markiert.

die Eckpunkte jedes Vierecks bei der Definition in einer Ebene liegen und das Viereck konvex ist. Anderenfalls ist das Ergebnis undefiniert.

Neben Punkten, Linien, Dreiecken und Vierecken stehen bei Java3D weitere drei Standard-Geometrieen zur Verfügung:

- Linienzüge (Instanzen der Klasse `LineStripArray`),
- Streifen aus Dreiecken (stripes, Klasse `TriangleStripArray`),
- Fächer aus Dreiecken (fans, Klasse `TriangleFanArray`).

Die Abbildung 6.17 zeigt alle geometrischen Objekte im Überblick. Bei *Linienzügen* werden jeweils zwei Eckpunkte, die in der Reihenfolge aufeinanderfolgen, verbunden. Diese Verbindung wird durch eine Kante dargestellt.

Bei *Streifen aus Dreiecken* wird im Vergleich zu Linienzügen zusätzlich der dritte Eckpunkt automatisch mit dem zwei Eckpunkte davor liegenden Eckpunkt verbunden, also mit dem ersten. Der erste, zweite und dritte Eckpunkt bilden ein Dreieck. Anschließend wird der vierte Eckpunkt mit dem zweiten verbunden. Der zweite, dritte und vierte Eckpunkt bilden ein zweites Dreieck. Analog wird mit allen weiteren Eckpunkten verfahren.

Bei *Fächern aus Dreiecken* wird im Vergleich zu Linienzügen zusätzlich der dritte Eckpunkt automatisch mit dem ersten Eckpunkt verbunden. Wie auch beim *Streifen aus Dreiecken* bilden der erste, zweite und dritte Eckpunkt ein Dreieck. Anschließend wird der vierte Eckpunkt ebenfalls mit dem ersten Eckpunkt verbunden. Der erste, dritte und vierte Eckpunkt bilden ein zweites Dreieck. Analog werden alle weiteren Eckpunkte mit dem ersten Eckpunkt verbunden.

Demnach bestehen *Streifen aus Dreiecken* und *Fächer aus Dreiecken* aus einer Folge von aneinanderliegenden Dreiecken. Bei Streifen und Fächern beranden bestimmte Kanten jeweils zwei Dreiecke. In bestimmten Fällen werden Eckpunkte jedoch im Gegensatz zu den vorigen beiden Absätzen nicht verbunden und demzufolge die durch diese Verbindungen berandeten Dreiecke nicht dargestellt, siehe hierzu die Abbildung 6.18, TriangleFanArray. Die Abbildung zeigt das Ergebnis, wenn die Reihenfolge der Eckpunkte verändert wird.

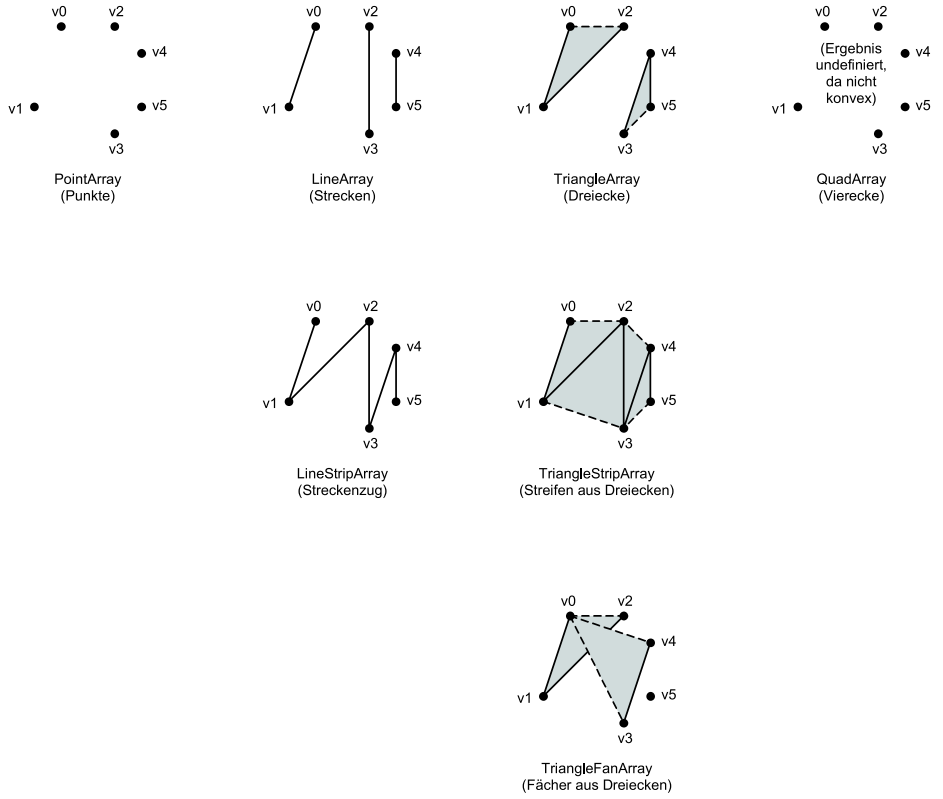


Abbildung 6.18: Im Vergleich zur Abbildung 6.17 entstehen mit den gleichen Eckpunkten, aber anderer Reihenfolge der Eckpunkte, andere geometrische Objekte.

6.2.10.3 Alle Standard-Geometrien mit Volumen im Überblick

Die Abbildung 6.19 stellt die vier Standard-Geometrien dar, die für Körper stehen, die ein Volumen besitzen: Würfel (Klasse Box), Zylinder (Cylinder), Kegel (Cone) und Kugel (Sphere). Im Folgenden werden diese Standard-Geometrien als Grundkörper bezeichnet.

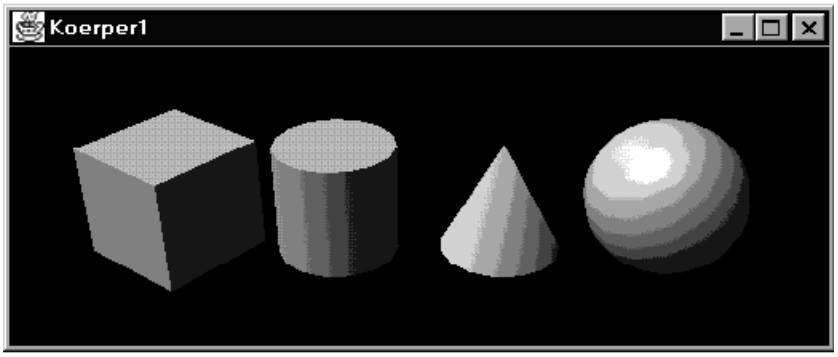


Abbildung 6.19: Beispiel Nr. 8, `Koerper1.java`
Darstellung der vier Grundkörper von Java3D

Die Klassenhierarchie der vier Grundkörper zeigt, dass sie keine Subklassen von `Leaf` oder `Shape3D` sind:

```
java.lang.Object
   javax.media.j3d.SceneGraphObject
      javax.media.j3d.Node
         javax.media.j3d.Group
            com.sun.j3d.utils.geometry.Primitive
               com.sun.j3d.utils.geometry.Box
               com.sun.j3d.utils.geometry.Cone
               com.sun.j3d.utils.geometry.Cylinder
               com.sun.j3d.utils.geometry.Sphere
```

Kern von Java3D ist das Package `javax.media.j3d`. Allerdings erhält man mit Java3D bei der Installation auch einige Packages, die mit `com.sun.j3d` beginnen. Deshalb kann auch das Package `com.sun.j3d.utils.geometry` als Teil von Java3D bezeichnet werden.

6.2.10.4 Beispiel Nr. 8: „Koerper1" (Standard-Geometrien mit Volumen)

Aus dem Quellcode des Beispiels „Koerper1" werden im Folgenden nur die Zeilen der Methode `createSceneGraph` abgedruckt, die sich auf den ersten Körper (Box) beziehen. Stellen des Quellcodes, die sich auf die Licht- und Schattierungs-Effekte des Beispiels beziehen, sind ebenfalls nicht angegeben.

Es folgt ein Ausschnitt aus dem Quellcode von `Koerper1.java`:

```
01.  public BranchGroup () {
02.    BranchGroup rootBg = new BranchGroup();
03.
04.    Transform3D translate = new Transform3D();
05.    translate.set( new Vector3f( -0.6f, 0.0f, 0.0f ) );
06.    TransformGroup translate1Tg = new TransformGroup(translate);
07.
08.    Transform3D rotateX = new Transform3D();
09.    Transform3D rotateY = new Transform3D();
```

```
10.    rotateX.rotX(Math.PI/6.0d);
11.    rotateY.rotY(Math.PI/3.5d);
12.    rotateX.mul(rotateY);
13.    TransformGroup rotate1Tg = new TransformGroup(rotateX);
14.
15.    Box box = new Box(0.15f,0.15f,0.15f,...); x-,y-,z-Richtung
16.
17.    rootBg.addChild(translate1Tg);
18.    translate1Tg.addChild(rotate1Tg);
19.    rotate1Tg.addChild(box);
20.
21.    return rootBg;
22.   }
```

In Zeile 15 wird eine Instanz der Klasse Box erzeugt und dabei initialisiert. Nach dem Initialisieren liegt der Mittelpunkt des Körpers im Ursprung. Um den Körper wie in der Abbildung 6.19 anzuordnen, werden auf den Körper zwei Transformationen angewendet: In Zeile 04 bis 06 wird der Würfel auf der x-Achse verschoben, in Zeile 08 bis 13 wird der Würfel rotiert. Der Szenegraph ab dem obersten BranchGroup-Datenelement wird in den Zeilen 17 bis 19 aufgebaut.

Im folgenden Szenegraphen wird deutlich, in welcher Reihenfolge in einem Pfad des Szenegraphen die zwei Transformationen pro Körper ausgeführt werden, nämlich von unten nach oben.

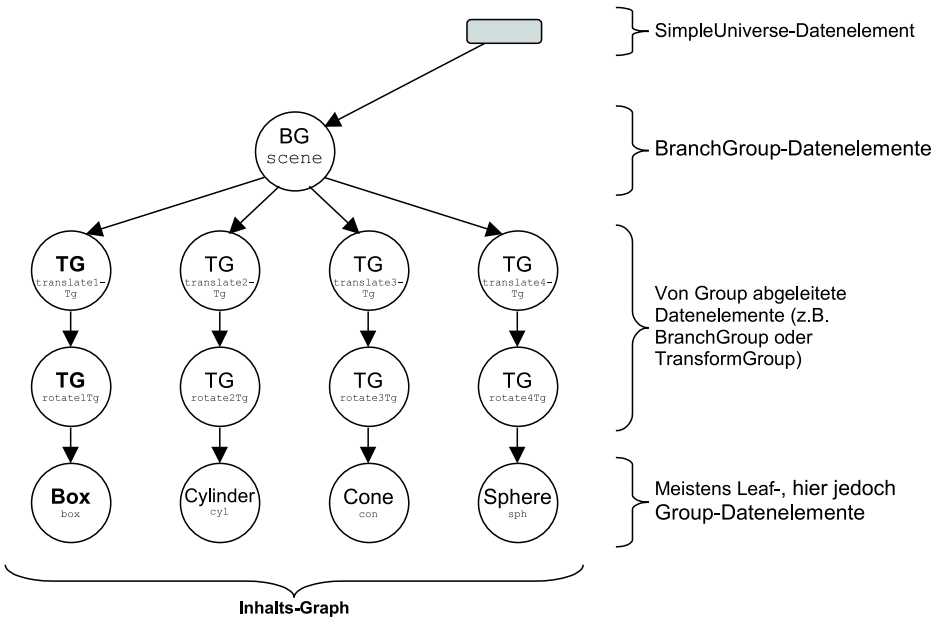


Abbildung 6.20: Szenegraph zum Beispiel `Koerper1.java` (ohne das Datenelement für die vorhandene Lichtquelle)

Die das Erscheinungsbild der Standard-Geometrien definierenden Werte wie Breite, Höhe, Tiefe und Radius können frei gewählt werden. Eine Kreisscheibe erhält man beispielsweise durch einen Zylinder mit der Höhe null. Durch die Anwendung von TransformGroup-Objekten können die Standard-Geometrien beliebig routiert und skaliert werden. So erhält man z.B. aus einer Kugel ein Ellipsoid.

Komplexere graphische Objekte (ohne oder mit Volumen) als die vorgestellten

müssen mit nicht in Java3D enthaltenen Klassen speziell definiert oder aus den gegebenen Geometrien zusammengesetzt werden.

6.2.11 Interaktion

6.2.11.1 Animation und Interaktion

In den bisherigen Beispielen wurden nur Programme behandelt, die eine Szene wie ein Standbild darstellen. Thema dieses Abschnitts sind Szenen, die oder deren Darstellung sich zur Laufzeit ändert. Ändern können sich virtuelle Geometrien und Lichtquellen, ihre Eigenschaften sowie der Augpunkt und die Blickrichtung des virtuellen Betrachters. Bei solchen Veränderungen wird zwischen Animationen und Interaktionen unterschieden:

- *Animationen* reagieren auf den Fortgang der Zeit. Sie laufen nach einem fest definierten und zur Laufzeit nicht veränderbaren Muster ab. Beispielsweise sind sich verändernde Trickszenen in Kinofilmen Animationen.

- *Interaktionen* reagieren auf Eingabe-Informationen von Benutzern oder von anderen Rechnern. Beispielsweise sind 3D-Spiele Interaktionen.

Animationen und Interaktionen sind komplexe Bereiche der 3D-Grafik. Deshalb werden wir uns nur mit einer einfachen Form der Interaktion beschäftigen, die sich mit Java3D in wenigen Zeilen programmieren lässt. Als Beispiel wählen wir einen Würfel (Instanz der Klasse `ColorCube`), der durch Bewegung der Maus um seinen Mittelpunkt gedreht werden kann. Für die Implementierung einer solchen Interaktion spielen die Begriffe *Behavior*, *Capabilities* und *Bound* eine Rolle. Diese Begriffe sind Thema der folgenden Abschnitte.

6.2.11.2 Behavior (Verhalten)

Bei Java3D werden Animationen und Interaktionen mit Hilfe der abstrakten Klasse `Behavior` realisiert. Mit Subklassen dieser Klasse kann der Entwickler festlegen, dass folgende Änderungen am Szenegraphen zur Laufzeit vorgenommen werden:

- Das Hinzufügen, Entfernen oder Austauschen von Datenelementen,

- Veränderungen von TransformGroup-Datenelemente, wodurch visuelle Objekte ihre Lage und Größe verändern,

- Veränderungen von Geometry- und Appearance-Referenzen, wodurch visuelle Objekte ihre Form und Oberflächeneigenschaften wie Farbe und Texturen verändern.

Es stehen einige Subklassen von `Behavior` zur Verfügung, weitere können kreiert werden. Ein Beispiel für eine solche Subklasse ist `MouseRotate`. Wird ein visuelles Objekt einer Szene mit einer Instanz von `MouseRotate` „verbunden", kann das visuelle Objekt interaktiv mit der Maus um den Mittelpunkt gedreht werden.

Jedes visuelle Objekt einer Szene kann mit Instanzen bestimmter Subklassen von `Behavior` verbunden werden. Die Erzeugung einer „Verbindung" verläuft in drei Schritten. Im ersten Schritt definiert der Entwickler eine solche Instanz und ein Leaf-Datenelement, z.B. eine mit `ColorCube` initialisierte `Shape3D`-Instanz. Im zweiten Schritt fügt der Entwickler beide Elemente in den Szenegraphen ein. Betrachten wir dazu die Klassenhierarchie von `Behavior`:

```
java.lang.Object
   javax.media.j3d.SceneGraphObject
      javax.media.j3d.Node
         javax.media.j3d.Leaf
            javax.media.j3d.Behavior
```

Subklassen von `Behavior` sind demnach, wie z.B. die Klasse `Shape3D`, indirekte Subklassen von `Leaf`. Sie sind deshalb im Szenegraph ganz unten angeordnet und an ein `BranchGroup`- oder `TransformGroup`-Datenelement angehängt.

Nachdem das `Behavior`- und das `Shape3D`-Datenelement Teil des Szenegraphen ist, erzeugt der Entwickler im dritten Schritt von der `Behavior`-Instanz eine Referenz zur `Shape3D`-Instanz oder zu einem darüberliegenden `TransformGroup`- oder `BranchGroup`-Datenelement.

Bemerkung: Um Quellcode zu sparen, entspricht bei der Implementierung die Reihenfolge und Zahl der Schritte beim Erstellen einer Verbindung nicht dem oben Dargestellten. Das Erstellen einer Verbindung wird weiter unten im Rahmen eines Kochrezeptes genauer betrachtet.

Der in der Abbildung 6.21 dargestellte Szenegraph steht für eine Szene mit einem farbigen Würfel (`ColorCube`), der mit Hilfe von `MouseRotate` interaktiv um den Mittelpunkt gedreht werden kann. Die Referenz von `MouseRotate` zu TG hat folgenden Effekt: Je nach Bewegung der Maus wird zu der `TransformGroup` TG automatisch eine entsprechende Rotation hinzugefügt und die Szene erneut dargestellt. Bei ausreichender Darstellungsgeschwindigkeit scheint der Würfel sich fließend zu drehen.

6.2.11.3 Capabilities (Fähigkeiten)

Wie im letzten Abschnitt dargestellt, werden Animationen und Interaktionen mit Hilfe der Klasse `Behavior` realisiert. So können Veränderungen von TransformGroup-Datenelementen und von Geometry- und Appearance-Referenzen implementiert werden. Sind jedoch diese Datenelemente und Referenzen bereits in einen Szenegraphen (z.B. mit `addChild()`) eingefügt worden, und enthält der Szenegraph schon das `Locale`- oder `SimpleUniverse`-Datenelement, können sie

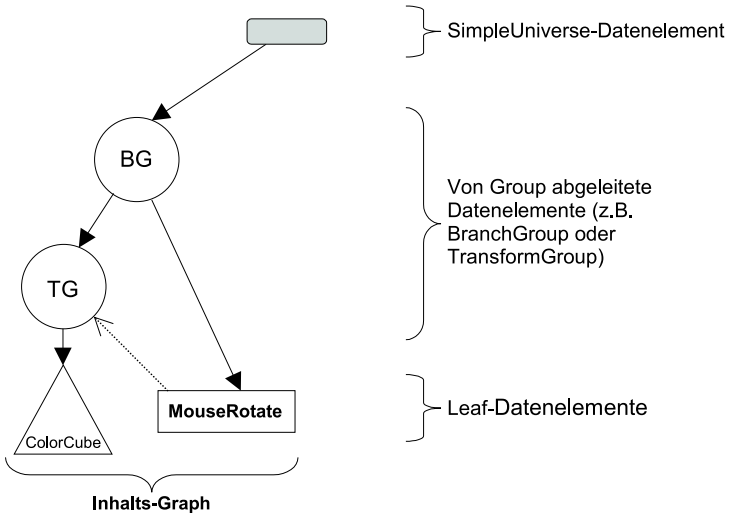


Abbildung 6.21: (Unvollständiger) Szenegraph des Beispiels `Wuerfel3.java`
Die Verbindung vom MouseRotate- zum TG-Datenelement (gestrichelter Pfeil) wird im Szenegraph in der Regel nicht dargestellt.

ab dem Compilieren der Datei nicht mehr verändert werden. Auch zur Laufzeit sind dann keine Änderungen mehr möglich. Wird beim Ausführen des Programms ein Befehl aufgerufen, der beispielsweise einer Instanz von `TransformGroup` eine Rotation hinzufügt, nachdem die Instanz bereits in den Szenegraph eingefügt worden ist, der ein `Locale`- oder `SimpleUniverse`-Datenelement enthält, kommt es zu einem Laufzeitfehler. Java3D übersetzt beim Compilieren den Szenegraphen in eine sogenannte interne Repräsentation (internal representation). Diese Repräsentation ist eine Art optimierte Version des Szenegraphen. Eine Art der Optimierung ist die Zusammenführung von `TransformGroup`-Datenelementen. Enthält ein Pfad im Szenegraphen beispielsweise mehrere `TransformGroup`-Datenelementen hintereinander, können die entsprechenden Transformations-Matrizen schon beim Compilieren miteinander multipliziert werden. Mit der entstehenden Matrix wird ein neues `TransformGroup`-Datenelement definiert, das die alten ersetzt.

Da durch die Optimierung ein möglichst großer Teil der zur Darstellung durchzuführenden Berechnungen bereits beim Compilieren ausgeführt wird, können die Elemente des Szenegraphen nicht im nachhinein verändert werden. Elemente wie `TransformGroup`-Datenelemente stellen für die Berechnungen einerseits die Eingabewerte der Berechnungen bereit und andererseits definieren sie die Rechenanweisungen, z.B.: „Rotiere um Achse z um 45 Grad”. Weder die Eingabewerte noch die Rechenanweisungen können nach der Berechnung verändert werden.

Mit einer Vorkehrung lassen sich Szenen zur Laufzeit jedoch verändern. Dazu muss explizit angegeben werden, inwiefern welche Datenelemente bzw. Referenzen zur Laufzeit veränderbar sind. Die zur Laufzeit veränderbaren Eigenschaften bzw. Parameter werden Capabilities (Fähigkeiten) genannt. Jeder dieser Capabilities ist ein Flag (Bit) zugeordnet. Nur Capabilities, deren Flag gesetzt ist, können zur Laufzeit verändert werden. Die Flags können mit der Methode `setCapability` der Klasse `SceneGraphObject` gesetzt werden.

Zur Erinnerung:
Von `SceneGraphObject` sind die Klassen `Node` und `NodeComponent` abgeleitet und damit auch `BranchGroup`, `TransformGroup`, `Leaf`, `Shape3D`, `Appearance` und `Geometry`. Da somit alle in diesem Kapitel beschriebenen Szenegraph-Elemente von `SceneGraphObject` die Methode `setCapability` erben, lassen sich mit Hilfe bestimmter Flags Eigenschaften aller Elemente zur Laufzeit verändern.

Die Syntax von `setCapability` lautet:

```
void setCapability(int bit)
```

Der Parameter `bit` gibt die Nummer des zu setzenden Flags an. Welche Flags gesetzt werden können, ist aus der Liste der Capabilities der entsprechenden Klasse zu entnehmen, siehe die Dokumentation von Java3D. Jeder Capability entspricht ein bestimmter `int`-Wert.

Beispiel: Um Geometrien zur Laufzeit drehen zu können, müssen sie im Szenegraph an ein `TransformGroup`-Objekt angehängt werden. Die `TransformGroup`-Klasse enthält neben den geerbten nur die beiden Capabilities:

```
TransformGroup.ALLOW_TRANSFORM_READ
TransformGroup.ALLOW_TRANSFORM_WRITE
```

Diese beiden `int`-Konstanten stehen für zwei Capabilities, mit denen diejenigen Daten gelesen bzw. verändert werden können, welche die Transformierung der Geometrien definieren. Für das Drehen von Geometrien zur Laufzeit muss der Entwickler beide Capabilities setzen.

6.2.11.4 Bound (Begrenzung)

Bei der Darstellung von Animationen und Interaktionen ändert sich zur Laufzeit die Szene. Die Darstellung solcher Szenen ist häufig sehr rechenintensiv, da bei Animationen und Interaktionen die Optimierung beim Compilieren und Darstellen nur im geringeren Maße möglich ist. Der Aufwand bei der Darstellung wird zusätzlich erhöht, wenn mehrere Geometrien unterschiedlich animiert werden bzw. der Entwickler mit verschiedenen Geometrien interagiert. Beispielsweise wird bei einer Fußball-Simulation jede Spielfigur einzeln animiert.

Um den Rechenaufwand zu minimieren, definiert der Entwickler für jede Animation bzw. Interaktion einen räumlichen Bereich. Die Animation bzw. Interaktion wirkt sich ausschließlich auf visuelle Objekte (Geometrien und Lichtquellen) aus, die zumindest teilweise in diesem Bereich enthalten sind. Dieser Bereich wird in Java3D *scheduling region* genannt, die Begrenzung dieses Bereiches *scheduling bound*. Jeder Animation und jeder Interaktion muss man genau eine scheduling bound zuweisen. Dabei kann man mehreren Animationen und Interaktionen die gleiche scheduling bound zuordnen. Wird einer Animation oder Interaktion keine scheduling bound zugeordnet, kann weder die Animation noch die Interaktion

ausgeführt werden - die entsprechenden Geometrien bleiben in ihrem Anfangszustand.

Eine scheduling region hat in der Regel die Form einer Kugel (bounding sphere) oder eines Würfels (bounding box). In diesem Kapitel wird ausschließlich die kugelförmige Begrenzung verwendet. Die scheduling region bzw. bound wird mit der folgenden Methode gesetzt:

```
void setSchedulingBounds( Bounds region )
```

Diese Methode stammt von der abstrakten Klasse `Behavior`, die oben bereits vorgestellt wurde.

Die Klasse `BoundingSphere` ist eine Subklasse von `Bounds`. Im Folgenden geben wir die Syntax zweier Konstruktoren von `BoundingSphere` an:

```
BoundingSphere()
```

und

```
BoundingSphere( Point3d center, double radius )
```

Der erste Konstruktor definiert einen kugelförmigen Bereich mit dem Mittelpunkt (0, 0, 0) und dem Radius 1. Den Aufruf des zweiten Konstruktors zeigt das folgende Beispiel. In dem Beispiel wird eine bounding sphere um den Ursprung mit dem Radius 100 definiert und gesetzt.

```
...
import javax.media.j3d.BoundingSphere;
import javax.vecmath.Point3d;
...    BoundingSphere bs =
   new BoundingSphere(new Point3d(0.0,0.0,0.0), 100.0;
... .setSchedulingBounds( bs )
```

Der Vollständigkeit halber folgt ein Auszug aus der Klassenhierarchie:

```
java.lang.Object
   javax.media.j3d.Bounds
      javax.media.j3d.BoundingBox
      javax.media.j3d.BoundingSphere
```

6.2.11.5 Kochrezept für die Benutzung der Subklassen von MausBehavior

Interaktive Programme nutzen von der Klasse `Behavior` abgeleitete Klassen. Für Interaktionen, bei der das Eingabegerät die Maus ist, wird folgende von `Behavior` abgeleitete Klasse eingesetzt:

```
com.sun.j3d.utils.behaviors.mouse.MouseBehavior
```

Diese abstrakte Klasse besitzt wiederum drei Subklassen. Eine dieser Subklassen ist `MouseRotate`, die bereits oben eingeführt wurde. `MouseRotate` ermöglicht folgenden Effekt: Bewegt man auf dem Fenster der Java3D-Anwendung mit gedrückter linker Maustaste die Maus, werden die mit dem `MouseRotate`-Objekt „verbundenen" visuellen Objekte um den Mittelpunkt rotiert. Das weiter unten vorgestellte Beispiel „Wuerfel3" implementiert eine solche Interaktion.

Da `MouseRotate` von `Behavior` abgeleitet ist, kann für ein `MouseRotate`-Objekt eine scheduling bound definiert werden. Es folgt ein Ausschnitt aus der Klassenhierarchie:

```
java.lang.Object
      javax.media.j3d.SceneGraphObject
         javax.media.j3d.Node
            javax.media.j3d.Leaf
               javax.media.j3d.Behavior
                  com.sun.j3d.utils.behaviors.mouse.MouseBehavior
                     com.sun.j3d.utils.behaviors.mouse.MouseRotate
```

Korrekt ausgedrückt verbindet der Entwickler keine visuellen Objekte, sondern ausschließlich `TransformGroup`-Objekte mit dem `MouseRotate`-Objekt. Die Interaktion wird auf alle visuellen Objekte angewendet, die an einem solchen `TransformGroup`-Objekt (mit `addChild`) angehängt werden und die zumindest teilweise im Inneren der scheduling bound liegen. Die folgende Abbildung des Szenegraphen des Beispiels „Wuerfel3" soll diese Verbindung verdeutlichen.

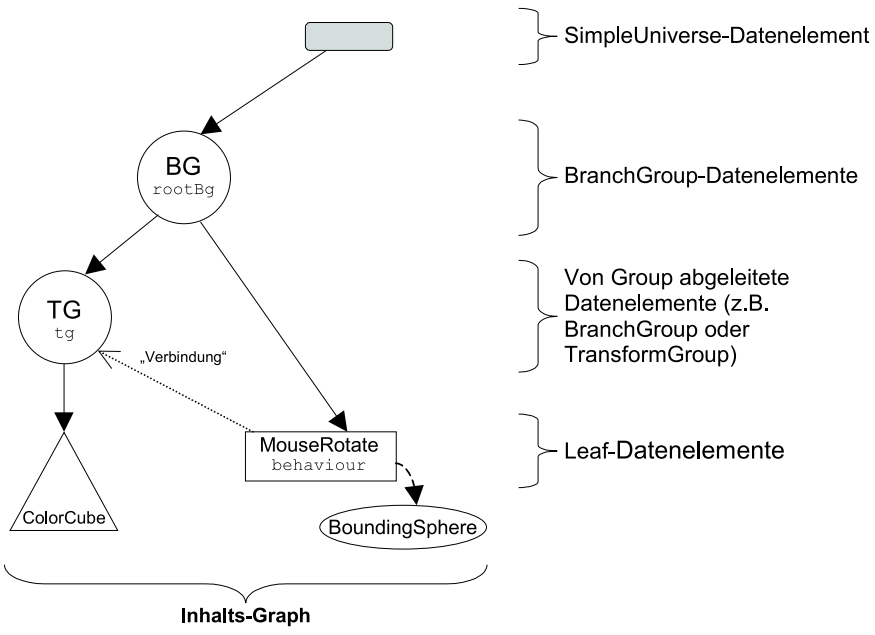


Abbildung 6.22: (Vollständiger) Szenegraph des Beispiels `Wuerfel3.java`

Einer der Konstruktoren von MouseRotate hat die folgende Syntax:

```
MouseRotate( TransformGroup transformGroup )
```

Das anschließende Kochrezept stellt die Schritte bei der Benutzung der Subklassen von `MouseBehavior` vor. Die mit der Maus drehbaren visuellen Objekte werden an ein TransformGroup-Datenelement angehängt. Zur Laufzeit soll die TransformGroup verändert werden können. Der Entwickler hat bereits die Instanz der Klasse `TransformGroup` definiert, die nötigen Capabilities gesetzt und das `TransformGroup`-Objekt bzw. -Datenelement in den Szenegraphen eingefügt.

1. Setzen der Fähigkeit (**capability**) des `TransformGroup`-Objekts;
2. Definieren eines `MouseBehavior`-Objekts (z.B. **MouseRotate**);
3. Erstellen einer **Verbindung** vom `MouseBehavior`- zum `TransformGroup`-Objekt;
4. Definieren einer räumlichen Begrenzung (**bound**) für das `MouseBehavior`-Objekt;
5. Einfügen des `MouseBehavior`-Objekts in den Szenegraphen.

Das Kochrezept wird im folgenden Beispiel angewendet.

6.2.11.6 Beispiel Nr. 9: „Wuerfel3" (interaktives Drehen mit der Maus)

Im folgenden Beispiel kommen die zuvor im Abschnitt 6.2.11 'Interaktion' vorgestellten Konzepte und Klassen zur Anwendung. Dazu wird das Programm „Wuerfel1" entsprechend erweitert. Bei „Wuerfel3" ist es möglich, mit der Maus den Würfel (`ColorCube`) beliebig um den Mittelpunkt zu drehen. Der Szenegraph zu diesem Beispiel findet sich im vorigen Abschnitt.

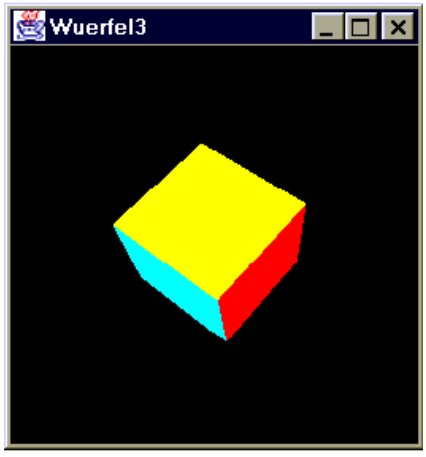


Abbildung 6.23: Beispiel Nr. 9, `Wuerfel3.java`
Nach einer bestimmten Bewegung mit der Maus wird die obere Seite des Würfels sichtbar.

Es folgt der Quellcode von `Wuerfel3.java`.

```
01i.  import java.applet.Applet;
02i.  import java.awt.BorderLayout;
03i.  import java.awt.Frame;
04i.
05i.  import javax.media.j3d.Canvas3D;
06i.  import javax.media.j3d.BranchGroup;
07i.  import javax.media.j3d.TransformGroup;
08i.  import javax.media.j3d.BoundingSphere;
09i.
10i.  import com.sun.j3d.utils.universe.SimpleUniverse;
11i.  import com.sun.j3d.utils.behaviors.mouse.MouseRotate;
12i.  import com.sun.j3d.utils.geometry.ColorCube;
13i.  import com.sun.j3d.utils.applet.MainFrame;

01.   public class Wuerfel3 extends Applet {
02.
03.      public Wuerfel3() {
04.         setLayout(new BorderLayout());
05.         Canvas3D canvas3D = new Canvas3D(
                  SimpleUniverse.getPreferredConfiguration());
06.         add("Center", canvas3D);
07.
08.         SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
09.         simpleU.getViewingPlatform().setNominalViewingTransform();
10.
11.         BranchGroup scene = createSceneGraph();
12.
13.         scene.compile();
14.         simpleU.addBranchGraph(scene);
15.      }
16.
17.      public BranchGroup createSceneGraph() {
18.         BranchGroup rootBg = new BranchGroup();
19.
20.         TransformGroup tg = new TransformGroup();
21.
```

1. Setzen der Fähigkeit (capability) des `TransformGroup`-Objekts:

```
22.         tg.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
23.         tg.setCapability(TransformGroup.ALLOW_TRANSFORM_READ);
24.         rootBg.addChild(tg);
25.
```

2. Definieren eines `MouseRotate`-Objekts:

und

3. Erstellen einer Verbindung vom `MouseBehavior`- zum `TransformGroup`-Objekt:

```
26.         MouseRotate behavior = new MouseRotate(tg);
27.
```

4. Definieren eine räumlichen Begrenzung (bound) für das `MouseBehavior`-Objekt:

```
28.         behavior.setSchedulingBounds(new BoundingSphere());
```

5. Einfügen des `MouseBehavior`-Objekts in den Szenegraphen:

```
29.         rootBg.addChild(behavior); //Oder: tg.addChild(behavior);
30.
31.         tg.addChild(new ColorCube(0.3));
32.
33.         return rootBg;
34.       }
35.
36.       public static void main(String[] args) {
37.         Frame frame = new MainFrame(new Wuerfel3(), 256, 256);
38.       }
39.     }
```

6.2.12 Darstellung der Klassenhierarchien

6.2.12.1 Klassenhierarchie von `javax.media.j3d`

Es folgt in Abbildung 6.24 eine graphische Darstellung eines Auszugs aus der Klassenhierarchie des Packages `javax.media.j3d`. Es werden genau diejenigen Klassen angegeben, die in dem Kapitel zu Java3D angesprochen wurden.

6.2.12.2 Weitere Klassenhierarchien

Die in Abbildung 6.25 abgebildete Klassenhierarchie enthält weitere Klassen, die wir im Abschnitt 6.2 'Java3D' kennengelernt haben.

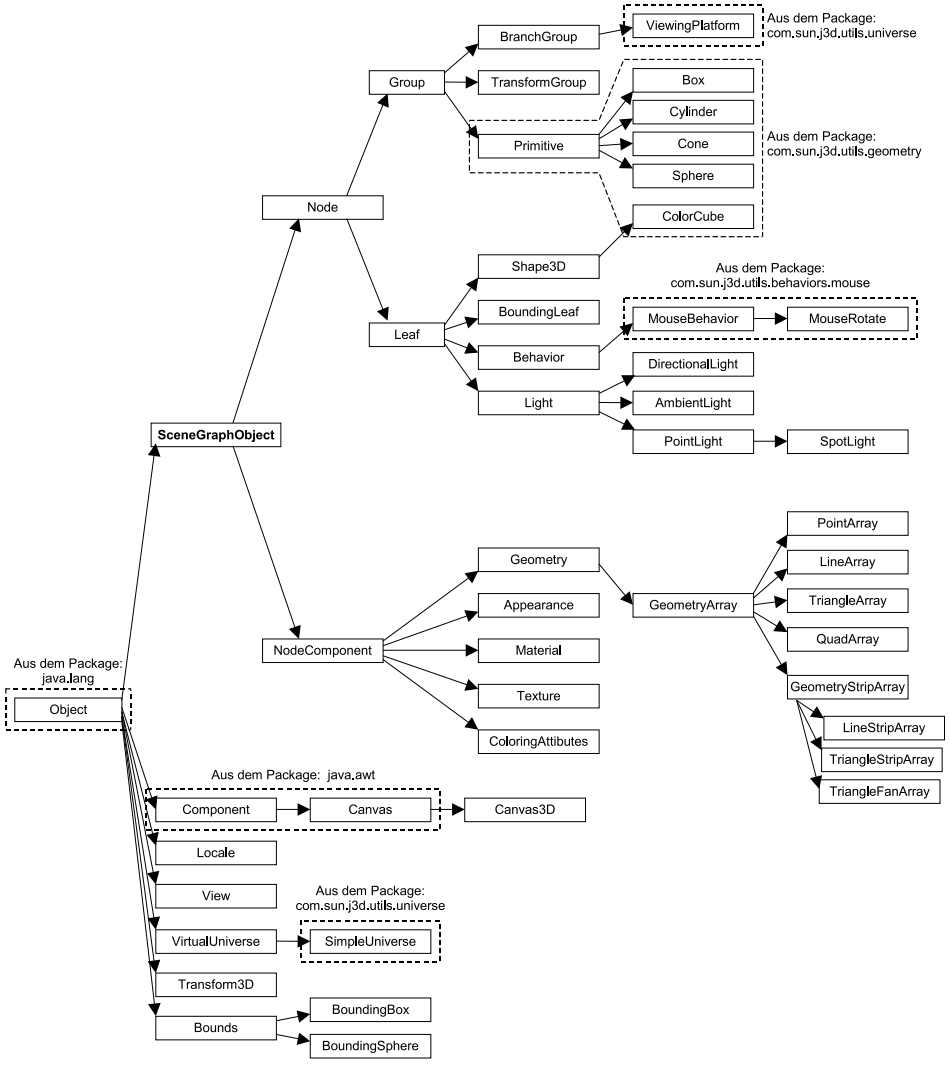


Abbildung 6.24: Ausschnitt aus der Klassenhierarchie des `javax.media.j3d`-Packages

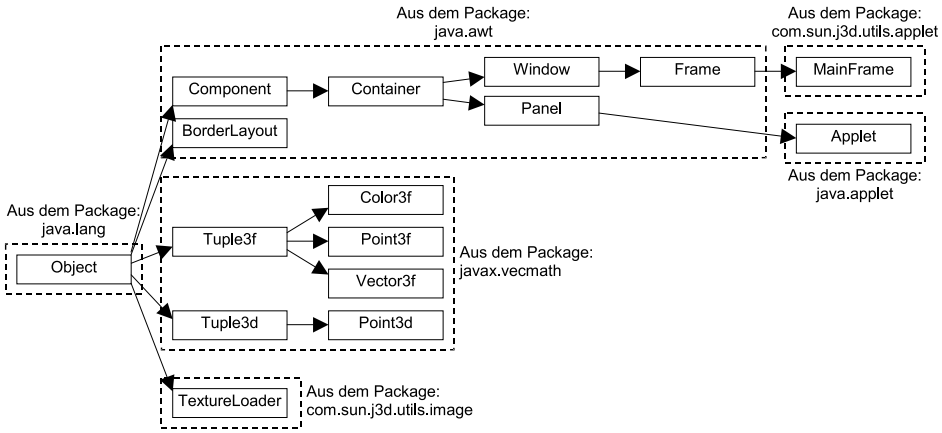


Abbildung 6.25: Eine weitere Klassenhierarchie

6.2.13 Standard-Importliste

In der Import-Liste am Anfang eines Programms können die einzelnen Klassen entweder mit ihrem vollständigen oder mit einem verkürzten allgemeinen Namen importiert werden, z.B.:

```
import javax.media.j3d.BranchGroup;
import javax.media.j3d.TransformGroup;
```

oder

```
import javax.media.j3d.*;
```

In den Beispielprogrammen wurde in der Import-Liste stets der vollständige Name angegeben, damit Sie die Klassen (`BranchGroup`) sofort ihrem Package (`javax.media. j3d`) zuordnen können. Will man jedoch selbst Programme mit Java3D erstellen, ist es einfacher und kürzer, nur die verkürzten allgemeinen Namen zu benutzen. Wir empfehlen insbesondere für einfache Beispielprogramme folgende Import-Liste, die wir Standard-Importliste nennen:

```
01i.  import java.applet.Applet;
02i.  import java.awt.BorderLayout;
03i.  import java.awt.Frame;
04i.
05i.  import javax.media.j3d.*;
06i.  import javax.vecmath.*;
07i.
08i.  import com.sun.j3d.utils.universe.SimpleUniverse;
09i.  import com.sun.j3d.utils.behaviors.mouse.*;
10i.  import com.sun.j3d.utils.geometry.*;
11i.  import com.sun.j3d.utils.applet.MainFrame;
```

6.2.14 Weiterführende Beispiele

Die folgenden drei Beispiele sollen einen Eindruck davon vermitteln, wie in Java3D Texturen und einfache Beleuchtungseffekte aussehen. Auf die Implementierung der Beispielprogramme wird hier nicht näher eingegangen.

6.2.14.1 Beispiel Nr. 10: „Koerper2” (Oberflächeneigenschaften, ambiente Lichtquelle)

Die Abbildung 6.26 zeigt eine Anordnung von jeweils drei Würfeln, Zylindern, Kegeln und Kugeln. Diese vier Grundkörper in Java3D (Box, Cylinder, Cone, Sphere) hat der Entwickler jeweils durch Transformationen verschoben, so dass ihre Mittelpunkte nicht im Ursprung, sondern in der x-y-Ebene liegen. Die Körper sind danach wie die Werte in einer 4×3-Matrix angeordnet. Anschließend wurden alle 12 virtuellen Körper gemeinsam etwas um die x- und y-Achse gedreht.

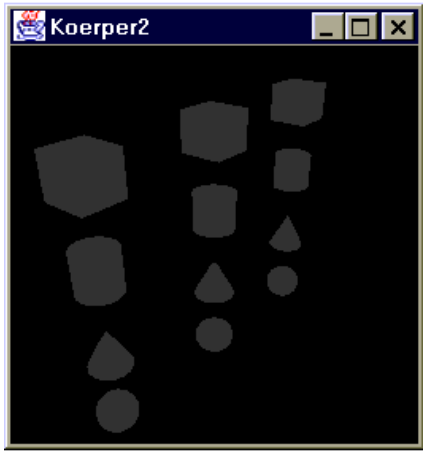


Abbildung 6.26: Beispiel `Koerper2.java`
Oberflächeneigenschaften und ambiente Lichtquelle: Die Körper sind kaum zu erkennen.

Die geometrischen Transformationen sind bei dem Beispiel `Koerper3` jedoch nicht so sehr von Interesse wie die Lichteffekte. Dazu werden die Oberflächeneigenschaften, die die Lichtreflexion und die Emmission von Licht betreffen, mit einer Instanz der Klasse `Material` und mit einer Instanz der Klasse `Appearance` festgelegt. Mit diesen Instanzen werden nicht die geometrischen Eigenschaften der Körper festgelegt, sondern z.B. Farbeigenschaften und Texturen. Die `Material`-Instanz wurde mit dem Standard-Konstruktor der Klasse `Material` initialisiert.

Allein mit der Definition eines `Appearance`- und `Material`-Objekts bliebe die Szene jedoch schwarz. Denn mit den Default-Werten vom `Material` wird die Oberfläche der 12 Körper bei schwarzem Hintergrund ausschließlich dann sichtbar, wenn sie Licht reflektiert. Deshalb wurde der Szene eine Lichtquelle hinzugefügt, die ambientes Licht aussendet. Ambientes Licht simuliert Licht, das von anderen visuellen Objekten diffus reflektiert wird. In der Realität kann man beispielsweise den Bereich der Straße direkt unter einem Auto selbst dann noch erkennen, wenn die Sonne von oben scheint. Der gleiche Effekt ist auch bei der Unterseite von Tischen zu beobachten.

Da es in der Szene ausschließlich ambientes Licht gibt, haben alle Seiten der Körper die gleiche Farbe (dunkelgrau). Ein räumlicher Eindruck kann so kaum entstehen. Um das zu ändern, kommt im folgenden Beispiel zur ambienten Lichtquelle eine andere Lichtquelle hinzu.

6.2.14.2 Beispiel Nr. 10: „Koerper3” (Oberflächeneigenschaften, ambiente und parallele Lichtquelle)

In dem Beispiel existiert in der Szene neben der ambienten eine parallele Lichtquelle. Eine parallele Lichtquelle strahlt Licht nur in einer Richtung aus, die „Lichtstrahlen” sind demnach parallel. Das Licht kommt von links oben hinter dem Beobachter und strahlt nach rechts unten und in die Blickrichtung des Beob-

achters.

Java3D bietet neben ambientem und parallelem Licht auch noch von einem Punkt aus abgestrahltes Licht und Lichtquellen, die, ähnlich wie ein Spot, nur einen bestimmten Bereich der Szene beleuchten. Die letzten beiden Lichtarten führen allerdings zu einer erheblich größeren Komplexität der Berechnungen bei der Darstellung der Szene. Deshalb begnügt man sich in der Regel mit ambientem und parallelem Licht.

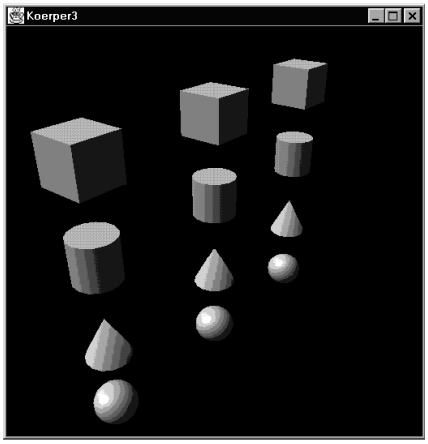


Abbildung 6.27: Beispiel Nr. 10, `Koerper3.java`
Oberflächeneigenschaften, ambiente und parallele Lichtquelle

Bei Java3D muss für jede Lichtquelle eine *scheduling bound* definiert werden: Die Lichtquelle beeinflusst nur das Aussehen von visuellen Objekten, die zumindest teilweise innerhalb der scheduling bound liegen.

6.2.14.3 Beispiel Nr. 11: „Koerper4" (Textur)

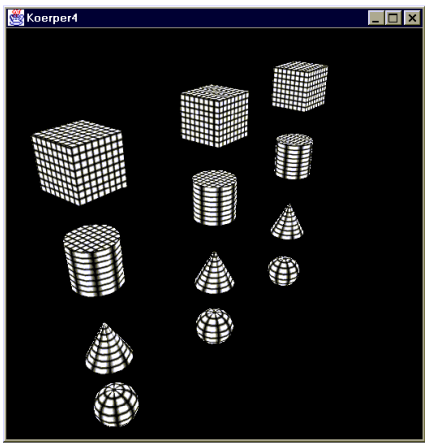


Abbildung 6.28: Beispiel Nr. 11, `Koerper4.java`
Mit Textur, ohne Oberflächeneigenschaften und Lichtquellen

Das Beispiel „Koerper4" baut auf dem Beispiel „Koerper1" auf. Jetzt werden die 12 Körper mit derselben Textur versehen, die quadratisch ist und ein schwarzes

Gitter auf weißem Hintergrund darstellt. Sie besteht ausschliesslich aus schwarzen und weißen Pixeln, die Grauwerte am Rand der einzelnen Gitterstäbe entstehen durch Interpolation. Die Interpolation ist durch die Transformation auf die virtuellen Körper bedingt. Auf der Oberseite des rechten Würfels und des rechten Zylinders sind Darstellungsfehler durch Aliasing zu erkennen.
Weitere Beispiele sind im Tutorial von Sun beschrieben ([Bou99].

6.3 OpenGL

6.3.1 Was ist OpenGL?

OpenGL steht für **Open G**rafics **L**ibrary. OpenGL ist eine Schnittstellen-Spezifikation von Routinen zur vielseitigen und schnellen 2D- und 3D-Visualisierung.

OpenGL entstammt der Softwareschmiede des renommierten Hauses Silicon Grafics bzw. *Silicon Grafics Interfaces*, kurz *SGI*. SGI stellt auf Grafik-Anwendungen spezialisierte Rechner her und ist für seine Highend-Grafik-Workstations bekannt.

SGI entwickelte in den 80er Jahren für die eigenen Rechner die 3D-Entwicklungsumgebung *IrisGL*. SGI trennte schließlich die Grafikbibliothek von IrisGL ab. Die Schnittstellen-Spezifikation zu dieser Bibliothek wurde 1992 als Grafik-Standard verabschiedet, die Schnittstelle erhielt den Namen OpenGL V 1.0.

Die Version 1.1 folgte im Jahre 1995, seit 1998 existiert die Version 1.2.

Anders als Java3D ist OpenGL keine Grafik-Bibliothek im Sinne von implementierter Software. OpenGL ist ein Standard, der die Schnittstelle bestimmter Grafik-Bibliotheken beschreibt. Alle diese Bibliotheken besitzen dieselbe Schnittstelle. OpenGL wird auch Schnittstellen-Spezifikation genannt. Inzwischen gibt es für viele Plattformen Implementierungen von Grafik-Bibliotheken, die der OpenGL-Schnittstellen-Spezifikation entsprechen. In der Praxis wird mit OpenGL jedoch nicht nur die Schnittstellen-Spezifikation bezeichnet, sondern ebenso die Implementierung der Spezifikation, das heißt die Grafik-Bibliothek bzw. die API.

OpenGL soll wie Java3D die Entwicklung von 3D-Grafik-Anwendungen erleichtern. Anders als bei der Grafik-API Direct3D (von Microsoft) und der Open Source-Grafik-API Glide (von 3dfx) liegt der Fokus von OpenGL nicht auf der 3D-Computerspiele-Entwicklung, sondern bei den professionellen Anwendungsbereichen von 3D-Computergrafik wie CAD und der Darstellung virtueller Welten.

Für die Pflege und Weiterentwicklung der OpenGL-Schnittstellen-Spezifikation ist ein Gremium aus Vertretern von Hard- und Softwarefirmen verantwortlich, das so genannte *Architecture Review Board* (*ARB*) .

Eine weitere Aufgabe des ARB ist die Vergabe von OpenGL-Lizenzen. Will ein Unternehmen eine OpenGL-Implementierung erstellen und für diese Software auch den Ausdruck „OpenGL” benutzen, muss es eine Lizenz von der ARB erwerben. Vor der Vergabe einer Lizenz überprüft die ARB, ob die Implementierung die Schnittstellen-Spezifikation umsetzt. Dafür muss die OpenGL-Implementierung eine Konformitätsprüfung (conformance test) durchlaufen. Jede OpenGL-Implementierung muss die komplette Schnittstelle implementieren, also sämtliche durch die Schnittstellen beschriebenen Routinen mit ihren definierten Eigenschaften. Dies stellt sicher, dass alle lizensierten OpenGL-Implementierungen vom Entwickler identisch zu handhaben sind. Wie effizient die Routinen im Einzelnen implementiert werden, ist dabei ohne Bedeutung. Nur die Implementierungen, die

den Test bestehen, dürfen laut ARB als OpenGL bezeichnet werden. Das ARB ist für die Durchführung und für die Richtlinien dieser Prüfung verantwortlich.

Die bekannteste lizensierte und freie (kostenlos zu benutzende) OpenGL-Implementierung ist *Mesa*. Mesa (oder „*the Mesa 3-D graphics library*") ist auf den meisten bedeutenden Unix/X Systemen sowie unter Linux compiliert und getestet worden. Zur Nutzung wird ein ANSI-C-Compiler benötigt. Auch für Windows-Systeme existiert eine OpenGL-Implementierung. In Windows 98/ME und Windows 2000 ist eine solche Implementierung vorinstalliert.

6.3.2 Wieso OpenGL?

Für die Entscheidung, OpenGL in diesem Kurs vorzustellen, sprechen drei Gründe: OpenGL wird von vielen Anwendungen genutzt, ist also ein Industriestandard. OpenGL eignet sich dazu, das **Zusammenwirken der Stationen der Bildgenerierung (Rendering-Pipeline)** zu erläutern. Der dritte Grund ist, dass Java3D auf OpenGL aufsetzt. Leider hat OpenGL zumindest einen Nachteil: Die Schnittstelle von OpenGL ist in C spezifiziert und deshalb nicht objektorientiert. Für das Verständnis dieses Kapitels über OpenGL reichen Kenntnisse in Java aus.

Für OpenGL wie auch für Java3D spricht außerdem der Vorteil, dass es Entwickler kostenlos zur Programmierung benutzen dürfen. Hinzu kommt die Tatsache, dass beide kontinuierlich gepflegt und weiterentwickelt werden.

Ein Schwerpunkt des vorigen Unterkapitels 6.2 'Java3D' war die **Implementierung** von Programmen, die auf Java3D aufsetzen. Im Gegensatz dazu widmen wir uns in dem vorliegenden Kapitel 6.3 über OpenGL einerseits den **Eigenschaften** von OpenGL und andererseits der **Rendering-Pipeline** von OpenGL.

6.3.3 Rendering-Pipeline, Teil 1

Die Hauptaufgabe von Java3D wie auch von OpenGL liegt in der Darstellung von dreidimensionalen Szenen. Diese Darstellung erfolgt in mehreren Schritten. Die Summe dieser Schritte ist die „Rendering-Pipeline". Die Rendering-Pipeline ist in der KE 1/2 (Einführung-Gerätetechnik), Abschnitt 2.4.3.1 'Grafiksystem mittlerer Leistungsfähigkeit' bereits erläutert worden. Deshalb werden im Folgenden die Schritte nur kurz beschrieben, ohne auf die hinter diesen Schritten steckenden Algorithmen einzugehen. Wir betrachten auch nicht den internen Aufbau von OpenGL-Implementierungen, da dieser je nach Implementierung variieren kann.

6.3.3.1 OpenGL als Black Box

Wie auch mit Java3D können mit OpenGL virtuelle Objekte und Szenen modelliert werden. Anschließend wird aus diesen dreidimensionalen Modellen ein zweidimensionales gerastertes Bild aus Pixeln erzeugt. Die Pixel schreibt OpenGL

schließlich in den Framebuffer (Bildspeicher).

Zunächst betrachten wir OpenGL als Black Box mit In- und Output. In der Black Box transformiert die Rendering-Pipeline von OpenGL die Input-Daten in Output-Daten, siehe die folgende Abb. 6.29.

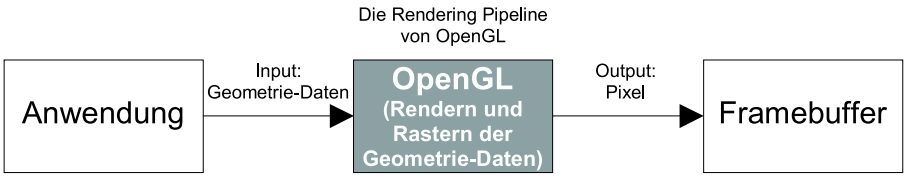


Abbildung 6.29: Die Rendering-Pipeline von OpenGL als Black Box

Input: Das Anwendungsprogramm ruft Routinen von OpenGL auf und übergibt dabei drei Arten von Input-Daten:

- *Geometriedaten*
  Sie bestehen vor allem aus den Daten der Eckpunkte von geometrischen Objekten. Zum Definieren solcher Eckpunkte ruft die Anwendung spezielle OpenGL-Funktionen auf. Solche Definitionen dürfen nur zwischen dem Aufruf der beiden Funktionen `glBegin` und `glEnd` stehen. Die Funktionen `glBegin` und `glEnd` legen fest, dass Eckpunkte, die zwischen den Funktionen definiert werden, zusammengehören. Mit einem der Funktion `glBegin` übergebenen Parameter wird festgelegt, welcher „Typ" eines geometrischen Objekts mit den Eckpunkten modelliert werden soll. So kann beispielsweise als Typ die Konstante `GL_TRIANGLES` für ein Dreieck (oder eine Folge von Dreiecken) angegeben werden. Auch der Typ des geometrischen Objekts ist Teil der Geometriedaten.

- *Bilddaten*
  Sie beschreiben mit Farbwerten gefüllte rechteckige Bereiche (Pixelmaps). Ein Farbwert ist entweder ein Farbtabellenindex oder ein Tripel aus Werten für die Farbkomponenten, z.B. bei der Echtfarbdarstellung (true color) ein Tripel aus je einem Wert für die rote, grüne und blaue Farbkomponente. Bilddaten werden in der Regel zur Definition von Texturen und Hintergrundbildern eingesetzt. Ein mit Schwarzweißwerten (Bits) statt Farbwerten gefüllter rechteckiger Bereich wird Bitmap genannt. Mit Bitmaps können Schwarzweißtexturen und binäre Masken definiert werden.

- *Zustandsdaten*
  Startet man eine Anwendung, die auf OpenGL aufsetzt, dann befindet sich die OpenGL-Implementierung in einem so genannten Zustand (state), der sich mit Ablauf der Anwendung in der Regel ändert. Der Zustand besteht aus rund 250 Variabeln. Mit geeigneten Routinen können diese Variablen verändert werden. Ihre Werte bezeichnen wir mit Zustandsdaten. Der Zustand ist Thema des Abschnitts 6.3.4.10 'Zustandsmaschine (State machine)'.

In der obigen Abbildung 6.29 werden als Input stellvertretend für alle Input-Daten nur die Geometriedaten genannt.

Output: Die von OpenGL gerenderten und gerasterten Daten ergeben eine Folge von Pixeln. Ein Pixel besteht entweder aus den Werten der Farbanteile (z.B. rot, grün und blau) oder aus einem Wert, der die Position der Farbe in der Farbtabelle angibt, dem so genannten Farbtabellen-Index. Die Farbanteile oder Farbtabellen-Indizes werden in den Framebuffer geschrieben. Der Framebuffer (Bildspeicher) ist ein Speicherbereich, der den Bildschirminhalt aufnimmt. Der Monitor stellt letztendlich diese Daten dar. Der Framebuffer ist als dreidimensionales Array zu verstehen. Bei einer Bildauflösung mit beispielsweise 1280 x 1024 Pixeln und bei durch die drei Farbkomponenten rot, grün und blau definierten Farben handelt es sich um ein Array mit 1280 x 1024 x 3 Zellen. Werden die Farben mit Farbtabellenindizes anstatt mit drei Farbkomponenten festgelegt, ergibt sich ein Array aus 1280 x 1024 x 1 Zellen.

6.3.3.2 Die Rendering-Pipeline als Blockdiagramm

Wir betrachten nun die einzelnen Schritte der Rendering-Pipeline (auch Bildgenerierungs-Pipeline genannt) von OpenGL. Dabei gliedern wir die Datenverarbeitung in der Pipeline in vier Schritte, siehe folgende Abbildung 6.30.

Die Rendering-Pipeline von OpenGL ist vergleichbar mit den ersten vier Funktionseinheiten (Host bis Bildspeicher) der in der KE 1/2 (Einführung-Gerätetechnik), Abschnitt 2.4.3.1 'Grafiksystem mittlerer Leistungsfähigkeit' in der dortigen Abbildung 2.33 bereits dargestellten Rendering-Pipeline.

1) Evaluatoren
Der Input - dies sind vor allem die Geometriedaten - definiert visuelle geometrische Objekte wie z.B. Linien oder Dreiecke. Mit OpenGL kann man jedoch auch durch polynomielle Funktionen definierte Kurven und gekrümmte Flächen modellieren, z.B. Bézier-Kurven und Bézier-Flächen, und mit der Zusatzbibliothek GLU auch „non-uniform rational B-Splines" (NURBS). Dabei gibt man unter anderem den Kontrollpolygonzug der Kurve bzw. das Kontrollnetz der Fläche an. Mit Hilfe spezieller Funktionen, so genannter Evaluatoren, werden aus dem Kontrollpolygonzug bzw. dem Kontrollnetz Linienzüge bzw. Gitternetze bestimmt, welche die Kurve bzw. Fläche approximieren. Die Linienzüge bzw. Gitternetze sind durch ihre Eckpunkte definiert. Zur Approximation werden die polynomielle Funktionen mit bestimmten Eingabewerten ausgewertet (evaluated).

Generell werden im Evaluator-Schritt komplexe geometrische Objekte in eine Menge von Eckpunkten (vertices) und Informationen über die Beziehungen der Eckpunkte untereinander überführt. Die Beziehungen legen unter anderem fest, welche Gruppen von Eckpunkten eine gemeinsame Linie, einen Linienzug, ein Polygon oder ein Gitternetz definieren.

2) Transformation, Beleuchtung, Clipping, Projektion
Die Eckpunkte werden transformiert, beispielsweise verschoben oder um eine Koordinatenachse rotiert. Anschließend werden die Farben der Eckpunkte in Abhängigkeit von den Lichtquellen in der Szene, den Zustandsvariablen (siehe unten, Abschnitt 6.3.4.10 'Zustandsmaschine (State machine)') sowie der Beleuchtung

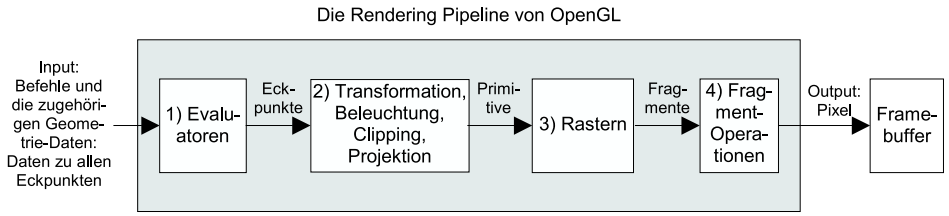


Abbildung 6.30: Die Rendering-Pipeline von OpenGL beschreibt die Reihenfolge der Schritte beim Rendern eines Bildes.

und den Eigenschaften (Materialeigenschaften und eventuell der Normalenvektor) des Punktes berechnet.

Daraufhin werden zwei Arten von Eckpunkten unterschieden: Zum einen die Eckpunkte, die Teil einer Linie, eines Linienzugs, eines Polygons oder Gitternetzes sind. Sie werden zusammen mit den jeweils zugehörigen anderen Eckpunkten der jeweiligen Geometrie (Linie, Linienzug, Polygon oder Gitternetz) behandelt; im weiteren Verlauf der Rendering-Pipeline wird jede dieser Geometrien als Ganzes behandelt, anstatt als eine Folge von Eckpunkten. Zum anderen sind die restlichen (isolierten) Eckpunkte zu betrachten, die nicht Teil einer Linie, eines Linienzugs, eines Polygons oder Gitternetzes sind. Aus ihnen werden Punkte. In den folgenden Schritten werden alle diese Geometrien (Punkte, Linien, Linienzüge und Polygone) als „geometrische Objekte" bezeichnet.

Es folgt das Clippen der geometrischen Objekte an den Grenzen des Ausgabebereichs. Das Clippen der Punkte wird auch point culling genannt. Es schließt sich die Perspektiv-Projektion an.

All diese rechenaufwändigen Phasen von der Transformation bis hin zur Projektion müssen nicht zwingend in dieser Reihenfolge durchlaufen werden. Das Ergebnis dieses Schrittes ist die auf die Bildebene (viewing plane) projizierte geclippte Szene. Im Detail werden zu jedem geclippten und projizierten geometrischen Objekt die zweidimensionalen Koordinaten der Eckpunkte mit ihrer Kolorierung ermittelt. Für spätere Operationen beispielsweise im Zusammenhang mit dem z-Buffer-Algorithmus wird zusätzlich der Tiefenwert berechnet.

3) Rastern
Beim Rastern werden aus den Eckpunkten der geometrischen Objekte die einzelnen Pixel berechnet. Jedes Pixel besteht aus je einem Wert für die x- und y-Position, an der sie später im Framebuffer abgelegt werden (Framebuffer-Adresse) und dem Farbwert (drei Farbkomponenten oder Farbtabellenwert). Zusätzlich werden für jedes Pixel der Transparenzwert (fürs α-Blending siehe Abschnitt 5.3.5.2 'Der exakte A–Buffer'), der Tiefenwert (für den z-Buffer) und die Texturkoordinate ermittelt. Zusammen werden all diese Werte von der x-Position bis hin zur Texturkoordinate eines Pixels als Fragment bezeichnet.

4) Fragment-Operationen
Auf jedes Fragment werden so genannte Fragment-Operationen angewendet. Fragment-Operationen sind die Berechnung der Textur, des Nebels, Antialiasing-Operationen, die Berechnung der Transparenz (α-Blending sowie weitere Blendig-Ver-

fahren), der z-Buffer-Test, die Maskierung und andere auf die Fragmente angewandte logische Operationen. Für den z-Buffer-Test greift OpenGL auf den z-Buffer zu. Der z-Buffer wurde vor dem Rendern der Szene initialisiert.

Beim Anwenden der Fragment-Operationen auf die Fragmente können sich die Farbwerte ändern. Wenn die Anwendung der Fragment-Operationen ergibt, dass das Fragment bzw. das Pixel nicht sichtbar ist, wird es nicht weitergegeben; andernfalls wird das Pixel an den Framebuffer übergeben.

Die abschließenden Aktionen sind unabhängig von OpenGL. Um den Inhalt des Framebuffers auszugeben oder weiterzuverarbeiten, liest das Fenstersystem des Betriebssystems den Framebuffer aus.

Zu beachten ist, dass weder die Schritte innerhalb der Rendering-Pipeline noch die Reihenfolge der Schritte in der OpenGL-Spezifikation festgelegt sind. Daher sind die oben beschriebenen Schritte und ihre Reihenfolge nur ein Beispiel, wie eine OpenGL-Implementierung die Rendering-Pipeline realisieren kann.

In den Abschnitten 6.3.4.12 ’Darstellungslisten (Display-Lists)’ und 6.3.5 ’Rendering Pipeline, Teil 2’ widmen wir uns ausführlicher der Rendering-Pipeline.

6.3.4 Eigenschaften

Jede der folgenden Eigenschaften wird weiter unten erläutert.

Allgemeine Eigenschaften:

1. OpenGL ist ein offener Standard und zugleich ein Industriestandard.
2. OpenGL ist zuverlässig.

Hardware-nahe Eigenschaften:

3. OpenGL ist plattformunabhängig.
4. OpenGL hat eine (Grafik-) Hardware-nahe Ausrichtung und ist deshalb effizient.
5. OpenGL hat eine Client/Server-Architektur.
6. OpenGL verarbeitet die Geometrie- und Bilddaten separat voneinander.

Software-nahe Eigenschaften:

7. OpenGL ist prozedural organisiert.
8. OpenGL ist nur eine Low-level-Bibliothek und stellt zur Modellierung als grafische Objekte nur so genannte Primitive zur Verfügung.

9. OpenGL wird durch zusätzliche Bibliotheken (z.B. GLU und GLUT) um elementare Fähigkeiten erweitert.
10. OpenGL wird als „state machine” bezeichnet.
11. OpenGL-Programme laufen im „Immediate Mode”.
12. OpenGL sieht zur Strukturierung des Codes so genannte Darstellungslisten vor.

6.3.4.1 Offener Standard, Industriestandard, OpenGL-Treiber

Wie bereits erläutert wurde, kontrolliert ein unabhängiges Gremium namens Architecture Review Board (ARB) den mit OpenGL definierten Schnittstellen- bzw. Grafikstandard. Demnach handelt es sich bei OpenGL um einen **offenen Standard**. Die Mitglieder der ARB Anfang des Jahres 2000 sind Compaq/Digital Equipment, Evans & Sutherland, Hewlett-Packard, IBM, Intel, Intergraph, Microsoft und SGI.

In bestimmten professionellen Bereichen wie dem computer aided design (CAD) ist OpenGL die am häufigsten benutzte Grafik-Schnittstelle. Wegen des kommerziellen Erfolgs spricht man bei OpenGL auch von einem **Industriestandard**.

Früher war OpenGL nur im Highend-Bereich ein Begriff. Inzwischen werden auch Grafikkarten der Mittelklasse in der Regel mit Treibern verkauft, die die Installation von OpenGL voraussetzen. Solche **OpenGL-Treiber** nutzen bei der Darstellung von dreidimensionalen Szenen die jeweiligen Eigenschaften der installierten Grafik-Hardware. Dabei werden möglichst viele Aufgaben im Rahmen des Renderns an die optimierte Grafik-Hardware delegiert. Solche typischen Aufgaben sind die Verwaltung des z-Buffers, bestimmte Bereiche der Berücksichtigung von Texturen bis hin zur gesamten Geometrie-Berechnung (Transformationen und Clipping) und der Schattierung.

6.3.4.2 Zuverlässigkeit

Die Spezifikation der OpenGL-Schnittstelle wird nur in größeren zeitlichen Abständen geändert. Von 1992 bis 2000 gab es gerade einmal drei Versionen. Diese vorsichtige Strategie hat zwei Vorteile: Anwendungs-Entwickler müssen sich nur in größeren Zeitabständen mit den Folgen von Veränderung der Spezifikation befassen. Weil Änderungen in der Regel erst nach einem längeren Diskussionsprozess vorgenommen werden, sind die Entscheidungen entsprechend ausgereift. Zudem haben OpenGL-Implementierungen den Ruf, auch komplexe Szenen zuverlässig darzustellen.

Noch eine weitere Eigenschaft macht die Zuverlässigkeit der OpenGL-Spezifikation aus: Die Versionen der OpenGL-Spezifikation sind abwärts-kompatibel. Selbst Anwendungen, welche auf älteren OpenGL-Spezifikationen aufbauen, bleiben somit weiterhin benutzbar.

6.3.4.3 Plattformunabhängigkeit

OpenGL ist für die Plattformen Windows 95/98/ME/NT/2000, Unix (Solaris (von Sun), AIX, HP/UX, IRIX (von SGI)), Linux, MacOS, BeOS, OS/2 und AmigaOS verfügbar.

OpenGL ist eine offene, systemübergreifende Software-Schnittstelle zur Grafikhardware. Falls eine Anwendung für mehrere Plattformen implementiert wurde, unterscheiden sich die Aufrufe der OpenGL-Routinen im Quelltext kaum. Lediglich bestimmte Quelltext-Passagen sind system- und programmiersprachenspezifisch:

1. Quelltext-Passagen, die das Fenstersystem betreffen:
   Das Fenstersystem handhabt die Elemente der grafischen Benutzeroberfläche wie Fenster, Menüs und Buttons und verwaltet alle Betriebssystem nahen Einstellungen, die für OpenGL relevant sind. Die Gesamtheit dieser Einstellungen wird Rendering-Kontext genannt.
2. Quelltext-Passagen, in denen plattformabhängige Eigenschaften des benutzten Compilers genutzt werden.

Trotz der beiden Ausnahmen werden Anwendungen, die eine OpenGL-Implementierung einbinden, als plattformunabhängig bezeichnet.

6.3.4.4 Hardware nahe Ausrichtung, Extensions

Wie bereits oben erläutert, wird die OpenGL-Spezifikation nur in größeren zeitlichen Abständen geändert. Um trotz der eher seltenen Versionssprünge die Soft- und Hardware-Entwickler nicht bei Neuentwicklungen einzuschränken, können einer OpenGL-Implementierung mit so genannten Erweiterungen (Extensions) neue Eigenschaften hinzugefügt werden.

Das ARB hat eine Schnittstellen-Spezifikation für Extensions definiert. Eine Extension besteht aus einer oder mehreren Funktionen, die nicht Teil der OpenGL-Spezifikation sind.

Auf Grund der plattformunabhängigen Architektur kann OpenGL selbst nicht mit den Elementen des Betriebssystems, die spezielle Systemressourcen wie Hardware zur Ein- und Ausgabe von Daten oder Grafikkarten verwalten, kommunizieren. Der Zweck von Extensions ist, die Kommunikation von OpenGL mit den Elementen des Betriebssystems zu ermöglichen. Da jedoch nahezu jedes Programm Tastatur- oder Mauseingaben benötigt und seine Ausgaben in einem gerasterten Bereich darstellen muss, enthält jedes OpenGL-System entsprechende Erweiterungen, die diese Kommunikation ermöglichen.

Extensions haben zwei Vorteile:

- Da Extensions kein Bestandteil der OpenGL-Spezifikation sind, bleibt OpenGL selbst plattformunabhängig.

- Spezielle, eventuell brandneue Hardware, kann eingebunden und alle ihre speziellen Eigenschaften genutzt werden. Durch diese Hardware-nahe Ausrichtung kann eine effektive Umsetzung der OpenGL-Funktionsaufrufe erreicht werden.

Die OpenGL-Implementierung ist generell plattformabhängig. Sie kann spezielle Grafik-Hardware unterstützen und durch Extensions auf bestimmte Hardware- oder Software-Eigenschaften zugeschnitten werden. Sollen bestimmte beschleunigende Eigenschaften der Hardware genutzt werden, müssen für diese Hardware die passenden OpenGL-Treiber installiert sein.

6.3.4.5 Client/Server-Architektur

OpenGL nutzt das Modell einer Client/Server-Architektur: 3D-Arbeitsplätze (die Clients) können beispielsweise bestimmte Aufgaben an einen speziell ausgerüsteten Server weiterleiten. Der Server führt anschließend die notwendigen Berechnungen aus und liefert die Ergebnisse an die Clients zurück. So kann die rechenintensive Arbeit auf ein zentrales System in Form eines Highend-Servers oder einer so genannten Render-Farm aus mehreren Servern ausgelagert werden. Der Client-Rechner dient nur der Darstellung und verarbeitet die Benutzereingaben, die dann an den Server weitergeleitet werden.

Der Vorteil einer solchen Client/Server-Architektur ist offensichtlich: Es muss nicht jeder 3D-Arbeitsplatz über die notwendige Rechenleistung verfügen, um anspruchsvolle 3D-Grafiken ausreichend schnell darstellen zu können.

6.3.4.6 Separates Verarbeiten von Geometrie- und Bilddaten

In der Abbildung 6.30 ist die Rendering-Pipeline in einer stark vereinfachten Form dargestellt. In der Realität zerlegen die OpenGL-Routinen im Rahmen der Rendering-Pipeline die Daten zunächst in Geometrie- und Bilddaten: Beide Arten von Daten können parallel jeweils von entsprechender Hardware verarbeitet werden. Der genauere Ablauf der separaten Verarbeitung von Geometrie- und Bilddaten wird im Abschnitt 6.3.5 ’Rendering-Pipeline, Teil 2’ beschrieben, siehe auch [c’t, Heft 20, S.244ff].

6.3.4.7 Prozedurale Organisation

Die OpenGL-Schnittstelle ist prozedural organisiert. Die Syntax der Schnittstellen entspricht derjenigen der Programmiersprache C. Für OpenGL wurde demnach kein objektorientierter Ansatz gewählt. Das liegt zum einen an der recht langen Entwicklungsgeschichte (früher war der objektorientierte Ansatz noch nicht so bedeutend) und zum anderen an der großen Gewichtung der Grafikleistung, compilierter C-Code ist schneller als z.B. compilierter objektorientierter C++-Code.

In der Praxis verwenden inzwischen die meisten Entwickler von Grafik-Anwendungen objektorientierte Programmiersprachen, meistens C++. Bei einer in C++ implementierten Anwendung, die auf OpenGL aufsetzt, folgen die Aufrufe der OpenGL-Routinen jedoch weiterhin der Syntax von C.

6.3.4.8 Low-level-Bibliothek, Primitive, Funktionen

OpenGL ist eine Low-level-Bibliothek. Als solche stellt OpenGL zur Modellierung ausschließlich einfache geometrische Objekte zur Verfügung: Punkte, Linien, Linienzüge und Polygone, sowie Streifen und Fächer. Die Abbildung 6.31 zeigt alle geometrischen Objekte im Überblick.

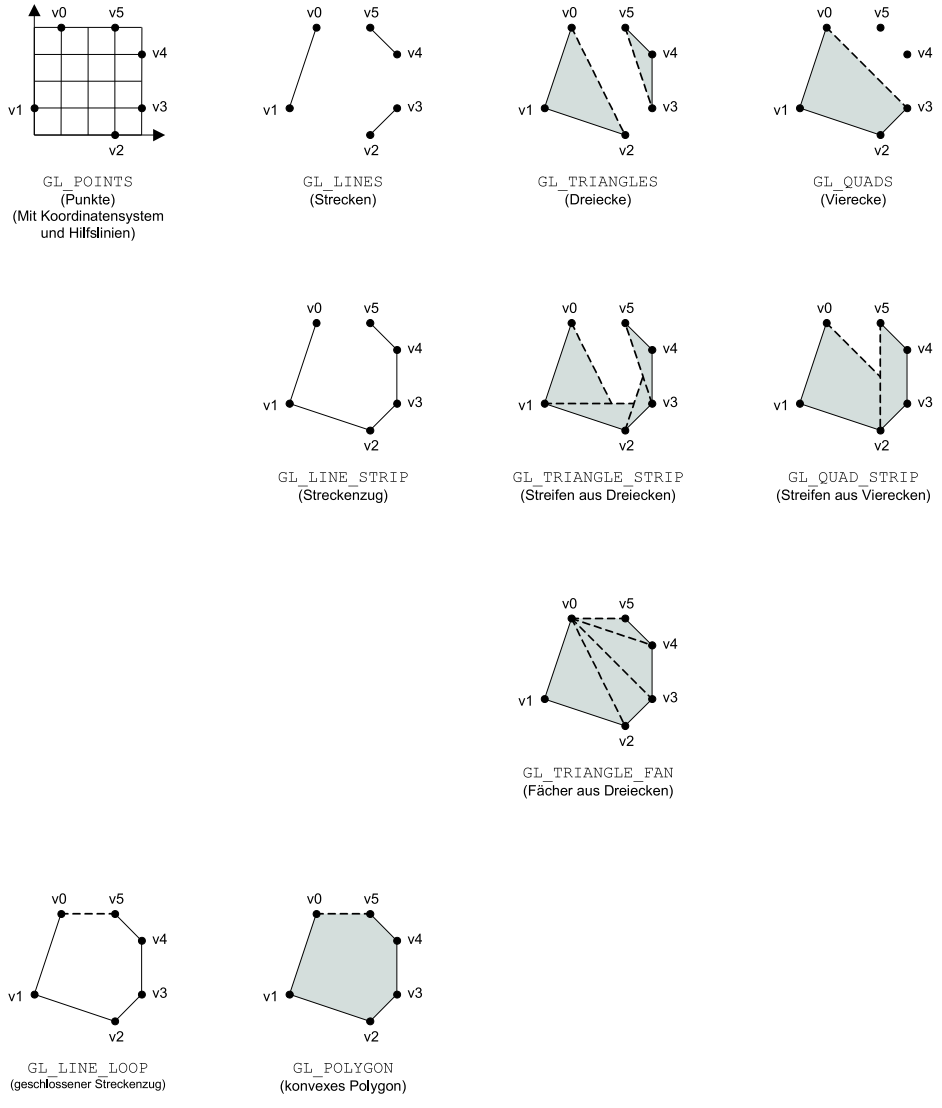


Abbildung 6.31: Ohne Berücksichtigung von Bit- und Pixelmaps stehen bei OpenGL ausschließlich diese 10 Primitive zur Verfügung. Die gestrichelt dargestellten Kanten werden automatisch hinzugefügt.

Die geometrischen Objekte Punkte, Linien, Linienzüge, Polygone, Streifen, Fächer, Bitmaps und Pixelmaps werden als grafische Primitive oder *Primitive* be-

zeichnet, bei Java3D heißen sie Geometrien.

Damit eine durch ein Viereck oder Polygon berandete Fläche von der Rendering-Pipeline von OpenGL verarbeitet werden kann, müssen die Eckpunkte der Fläche nicht genau in einer Ebene liegen, da sonst wegen numerischer Ungenauigkeiten bei Transformationen und beim gesamten Render-Prozess ein großer Teil solcher Flächen nicht darstellbar wäre. Bei zu großen Abweichungen kann es jedoch zu Darstellungsfehlern kommen.

Als Primitive werden auch die Grafik-Objekte aus zusätzlichen Bibliotheken (siehe Abschnitt 6.3.4.9 'Zusätzliche Bibliotheken') bezeichnet. Falls man nicht auf solche Bibliotheken zurückgreift, müssen komplexe grafische Objekte wie ein Würfel oder eine Kugel bei OpenGL aus den gegebenen Primitiven zusammengesetzt werden.

Wenden wir uns nun der Implementierung zu. Will der Entwickler in einer Anwendung ein Primitiv definieren, beispielsweise ein bestimmtes Dreieck, beginnt die Definition immer mit dem Aufruf von `glBegin(Mode)` und endet mit `glEnd()`. Das Argument `Mode` teilt dem System mit, welche Art von Objekt es durch die Aufrufe der Kernfunktionen zusammensetzen soll. Bei einem Dreieck wäre für `Mode` die Konstante `GL_TRIANGLES` anzugeben. Zwischen den beiden Zeilen würden die drei Eckpunkte des Dreiecks mit jeweils einem Aufruf der Prozedur `glVertex` definiert („Vertex" von „Eckpunkt").

Bei OpenGL wird generell statt „Prozedur" der Ausdruck „Funktion" verwendet. Von jeder OpenGL-Basisfunktion wie `glVertex` existieren im Allgemeinen mehrere Typen mit jeweils unterschiedlichen Parametern. Beispielsweise kann ein Eckpunkt mit `glVertex` definiert werden, indem zwei oder drei Integer-Werte oder zwei oder drei dezimale Werte als Parameter übergeben werden. Bei zwei Integer-Werten wäre ein Eckpunkt im Zweidimensionalen mit zwei ganzzahligen Koordinatenwerten definiert worden. Zur Unterscheidung der Funktionen mit gleichem Funktionsnamen, aber unterschiedlichen Parametern, hängt der Entwickler an die Funktionsnamen entsprechende Suffixe an. Der Suffix besteht meistens aus einer Ziffer und einem Buchstaben. Die Ziffer gibt die Anzahl der Parameter an. Der Buchstabe bestimmt den Typ der Parameter, beispielsweise `b` für GLbyte (8 Bit Integer mit Vorzeichen) oder `i` für GLint (32 Bit Integer mit Vorzeichen). Zum Beispiel erwartet `glVertex3i()` drei Werte vom Typ GLint. Die `glVertex`-Funktionen werden in der Spezifikation von OpenGL allgemein mit „`glVertex*()`" benannt. Das Sternchen steht dabei für die unterschiedlichen Suffixe. Analoges gilt auch für die übrigen Funktionen.

Die folgende Definition eines Primitivs erzeugt ein rotes Dreieck:

```
glBegin(GL_TRIANGLES);
   glColor(1.0, 0.0, 0.0);
   glVertex3i(0,0,0);
   glVertex3i(2,0,0);
   glVertex3i(1,2,0);
glEnd();
```

Die nächste Primitiven-Konstruktion erzeugt jede der 10 Primitiven der Abbildung 6.31 im Zweidimensionalen, wobei für `Mode` die betreffende Konstante anzugeben ist:

```
glBegin(Mode);
   glColor(1.0, 0.0, 0.0);
   glVertex2i(1,4);
   glVertex2i(0,1);
   glVertex2i(3,0);
   glVertex2i(4,1);
   glVertex2i(4,3);
   glVertex2i(3,4);
glEnd();
```

In der Regel wird eine Kante nur dann dargestellt, wenn die zwei Eckpunkte der Kante nacheinander aufgerufen werden. Jedoch werden bestimmte Kanten automatisch dargestellt; diese Kanten wurden in Abbildung 6.31 gestrichelt gezeichnet.

Beim Definieren von Primitiven durch ihre Eckpunkte ist wie bei Java3D die Reihenfolge zu beachten, in der die Eckpunkte aufgelistet werden. Wie bei Java3D lassen sich Primitive auch aus anderen Primitiven zusammensetzen. Was bei OpenGL die Definition eines Primitivs ist, entspricht bei Java3D der Definition der entsprechenden Knoten, Kanten und Referenzen und dem anschließenden Einfügen dieser Elemente in den Szenegraphen.

Mit `glBegin`, `glEnd`, `glVertex*` und `glColor` haben wir schon einige Basisfunktionen kennengelernt. OpenGL besteht aus einer Schnittstellen-Spezifikation von über 120 (Version 1.0), 150 (V 1.1) oder 200 (V 1.2) Basisfunktionen.

6.3.4.9 Zusätzliche Bibliotheken

OpenGL hat als Low-level-Bibliothek einen Vorteil, aber auch einen Nachteil: Einerseits sind die Routinen wegen ihrer fenstersystem-unabhängigen Definitionen und dem Prinzip „Konzentration aufs Wesentliche” allgemein verwendbar und effizient. Andererseits sind nicht triviale Anwendungen mit OpenGL allein kaum zu realisieren, da OpenGL umständlich zu handhaben ist:

- OpenGL bietet nur wenige grafische Primitive (Punkte, Linien, Polygone, Streifen, Fächer, Bit- und Pixelmaps). Beim Modellieren müssen diese erst mühsam zu grafischen Objekten zusammengesetzt werden. Zum Beispiel sind Körper mit Volumen wie Zylinder oder Quader aus einzelnen Flächen zusammenzusetzen.
- OpenGL bietet keine Mechanismen oder Werkzeuge zur hierarchischen Modellierung komplexer grafischer Objekte. Im Gegensatz dazu wird bei Java3D das hierarchische Modellieren durch den Aufbau eines Szenegraphen realisiert.

- OpenGL bietet keine Mechanismen oder Werkzeuge zum Einlesen und Ausgeben grafischer Daten aus einer bzw. in eine Datei.
- OpenGL bietet nur detaillierte und damit umständliche Möglichkeiten zur Texturierung von Geometrien.

OpenGL bietet also umfassende Möglichkeiten um festzulegen, wie aus der Definition komplexer grafischer Objekte Bilder generiert werden sollen; aber mit OpenGL lassen sich komplexe grafische Objekte nur umständlich beschreiben bzw. modellieren. OpenGL stellt demnach nur eine „Basisbibliothek" dar, die in der Regel ohne Zusatz-Bibliotheken nicht effektiv genutzt werden kann. Die bekanntesten auf OpenGL aufbauenden Bibliotheken sind GLU, GLUT und der Open Inventor.

GLU
GLU ist die Abkürzung von Open**GL U**tility Library.

GLU stellt mehrere Dutzend zusätzlicher Funktionen zur Verfügung; es folgt eine Auswahl:

- Funktionen zur Modellierung von **vordefinierten Primitiven** wie Zylindern, Kreisscheiben oder Kugeln;
- Funktionen zur Erstellung von **NURBS** (Non-Uniform Rational B-Spline) zur komfortablen und flexiblen Modellierung von Kurven und gekrümmten Flächen;
- Funktionen zum einfacheren Arbeiten mit **Texturen**.

Jede Funktion von GLU beginnt mit ş`glu`ş. Bei den gängigen Implementierungen von OpenGL wird eine Implementierung von GLU mitgeliefert und mit OpenGL installiert: Für den Entwickler entsteht der Eindruck, die GLU-Funktionen seien Teil von OpenGL. Dann lassen sich die OpenGL-Funktionen in zwei Kategorien einteilen:

- Die Kernfunktionen, deren Name mit einem vorangestellten ş`gl`ş beginnen, und
- die Hilfsfunktionen, deren Name mit ş`glu`ş beginnt.

Hilfsfunktionen setzen intern auf den Kernfunktionen auf. Hilfsfunktionen kommen in nahezu jedem nicht trivialen 3D-Programm vor, das auf OpenGL aufsetzt.

GLUT
GLUT ist die Abkürzung von Open**GL U**tility Library **T**oolkit, ausgesprochen wie der englische Begriff gluttony (Völlerei, Gefräßigkeit). GLUT zeichnet sich durch folgende Eigenschaften aus:

- GLUT bietet **vordefinierte Primitive** wie Würfel, Kegel oder Kugeln.

- GLUT stellt eine **einheitliche Schnittstelle für Fenstersystem-Routinen** zur Verfügung:
  Auf Grund der Plattformunabhängigkeit verfügt OpenGL über keinerlei systemspezifische Routinen. Mechanismen zum Zugriff auf Maus, Tastatur oder andere Hardware für die Datenein- und -ausgabe sind nicht Bestandteil von OpenGL. Diese so genannten Fenstersystem-Routinen lassen sich mit zusätzlichen Bibliotheken zwar für einzelne Plattformen einbinden, sie sind jedoch für jede Plattform unterschiedlich zu handhaben. GLUT stellt diese Routinen über eine für alle diese Plattformen identische Schnittstelle zur Verfügung.
  Neben Funktionen für die I/O-Steuerung (Maus, Tastatur, Joystick) bietet GLUT eine einfache Verbindung von OpenGL mit Fenstern des Betriebssystems bzw. des Window Managers.

GLUT ist zwangsläufig plattformabhängig, jedoch nicht eine mit GLUT programmierte Anwendung. GLUT ist für Unix/Linux, Windows NT, Windows 95, für den Mac und OS/2 implementiert. GLUT ist frei verfügbar, jedoch keine Public Domain. Im Gegensatz zu GLU muss GLUT separat installiert werden.

Jede Funktion von GLUT beginnt mit ş`glut`ş.

GLUT ist leicht zu handhaben. Insbesondere zum Erlernen von OpenGL ist GLUT geeignet, ebenso wie für die Erstellung kleiner und mittelgroßer Programme, die OpenGL-Routinen aufrufen. Für die Erstellung von größeren und professionellen Anwendungen ist GLUT weniger zu empfehlen, da GLUT ausschließlich Routinen bereitstellt, die sich auf die Routinen der Fenster-Systeme der verschiedenen Plattformen abbilden lassen. Aufgrund dieser Verallgemeinerung können mit GLUT fenstersystem-spezifische Eigenschaften nicht genutzt werden. Für professionelle Anwendungen reicht die verringerte Funktionalität häufig nicht aus.

Open Inventor
Der Open Inventor („offener Entwickler”) besteht aus mehreren Komponenten. Eine der Komponenten ist eine Bibliothek, mit der OpenGL um weitere Funktionalität ergänzt wird, ähnlich wie bei GLU und GLUT. Eine weitere Komponente ist eine Spezifikation für ein Dateiformat zum Einlesen und Ausgeben von grafischen Daten.

Der Open Inventor ergänzt OpenGL durch die folgenden Eigenschaften:

- Neue **vordefinierte Primitive** wie Würfel, Zylinder, Kegel und Kugel.

- Visuelle Objekte können hierarchisch zu komplexeren Objekten und Szenen modelliert werden. Die erzeugte Struktur ist ein **Szenegraph**, der dem Szenegraphen von Java3D ähnelt.

- Eine der Komponenten vom Open Inventor ist das so genannte **Toolkit** (Werkzeugkasten). Das Toolkit ist eine Anwendung mit grafischer Benutzeroberfläche, mit der Geometrien betrachtet werden können und mit der ein Szenegraph zusammengeklickt werden kann, siehe Abbildung 6.32.

- Grafische Daten können einfach **eingelesen und ausgegeben** werden. Die Daten werden in der Regel in einer Datei abgelegt.

- Er erspart ähnlich wie GLUT die Auseinandersetzung mit dem Fenstersystem. Beispielsweise stellt der Open Inventor **Objekte zur Implementierung der Benutzerschnittstelle** zur Verfügung, z.B. Buttons und Menüs. Mit dem Open Inventor erstellte Anwendungen haben ein einheitliches und typisches „Look and Feel", sehen also ähnlich aus und folgen einer ähnlichen Benutzungslogik.

- Der Open Inventor besitzt eine Klassenbibliothek in C++, über die er individuell **erweiterbar** ist. Der Inventor bietet jedoch für jedes Objekt auch eine Schnittstelle in C.

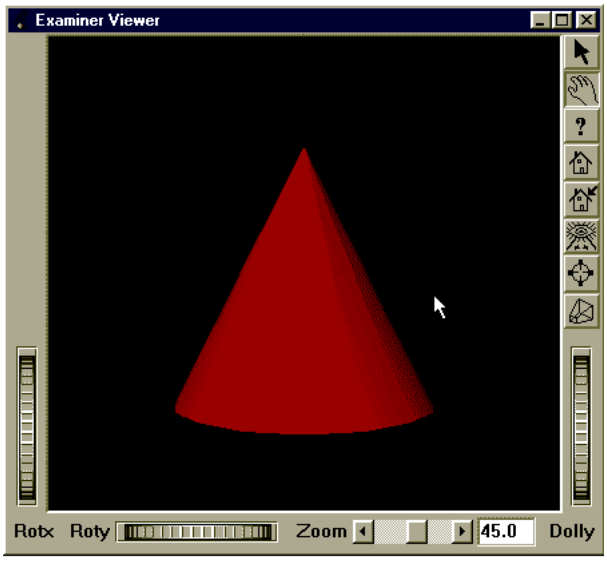


Abbildung 6.32: Die Benutzeroberfläche der Komponente des Open Inventor, mit der man die modellierte Szene betrachten kann: Die drei „Stellrädchen" (thumbwheels) im unteren Drittel sind typisch für mit dem Open Inventor programmierte Anwendungen.

6.3.4.10 Zustandsmaschine (State machine)

Geometrie- und Bilddaten (siehe Abschnitt 6.3.3.1 'OpenGL als Black Box') werden in der Rendering-Pipeline verarbeitet und schließlich in den Framebuffer geschrieben. Geometrie- und Bilddaten definieren, *welche* geometrischen Objekte dargestellt werden. Im Unterschied dazu definieren Zustandsdaten, *wie* Geometrie- und Bilddaten dargestellt werden. Der Zustand einer OpenGL-Implementierung besteht aus mehr als 250 Variablen (Schalter oder auch Attribute genannt). Jede Variable beschreibt dabei einen spezifischen Aspekt der Ausführung, z.B. die aktuelle Farbe, die geometrische Transformation oder die Textur, sowie den Augpunkt und die Blickrichtung des virtuellen Beobachters in der Szene. Wird eine der Variablen verändert, hat dies einen Einfluss auf die Darstellung der im Quellcode nach der Variablenänderung (bzw. auf die Darstellung aller zukünftig) definierten Geometrien.

In einem auf OpenGL aufsetzenden Programm wechseln sich zwei Arten von Befehlen ab: Die erste Art beschreibt die Art der Verarbeitung, also was mit den Geometrie- und Bilddaten zu tun ist. Die zweite Art definiert die Geometrie- und Bilddaten.
Es folgt ein Beispiel zu diesem Wechselspiel:
Ein Programm bestehe aus folgenden fünf Befehlen:

1. Stelle aktuelle Farbe auf rot.
2. Definiere eine Dreieck.
3. Stelle aktuelle Farbe auf blau.
4. Transformiere: Skaliere auf die Hälfte (die Transformationsmatrix wird verändert).
5. Definiere ein Polygon.

Die Zeilen 1, 3 und 4 geben an, was mit geometrischen Daten zu tun ist. Die anderen beiden Zeilen definieren die Geometriedaten. Alle Variablen behalten solange ihre Einstellung, bis sie durch explizites Aufrufen bestimmter hierfür vorgesehener Funktionen verändert werden oder bis das Programm terminiert. Beispielsweise wäre ohne den dritten Befehl auch das Polygon rot. Auch die Transformationsmatrix kann als Variable verstanden werden. Anders als bei Java3D gibt es nur eine Transformationsmatrix. Eine Änderung der Matrix hat Auswirkungen auf alle nach der Änderung (bis zur erneuten Änderung) definierten visuellen Objekte.

Wegen der zentralen Bedeutung des Zustands für die Rendering-Pipeline von OpenGL wird OpenGL auch als *Zustandsmaschine* („state machine”) oder Zustandsautomat bezeichnet. Statt „Zustand” werden in der Literatur auch die Begriffe „Status” und „Modus” verwendet.

Variablen vom Typ `boolean` lassen sich mit den Funktionen `glEnable()` und `glDisable()` „ein- und ausschalten”. Der Status dieser Variablen läßt sich mit `glIsEnable()` ermitteln.

Zusammengefasst ist das Ergebnis der Rendering-Pipeline (in Form von Pixeln) abhängig von

- den Geometrie- und Bilddaten,
- dem Augpunkt und der Blickrichtung des virtuellen Beobachters in der Szene,
- dem Zustand von OpenGL.

6.3.4.11 Immediate Mode

Bei OpenGL werden die Szenen im *Immediate Mode* („direkten Modus”) dargestellt: Jede Codezeile wird so bald wie möglich ausgeführt. Wenn beispielsweise

im Quellcode ein Primitiv definiert wird, können bei der Ausführung des compilierten Programms die ersten Befehle der Definition schon ausgeführt werden, selbst wenn das Ende der Definition noch nicht erreicht ist. Im Unterschied dazu wird bei Java3D erst der gesamte Szenegraph aufgebaut, bevor Java3D mit der Darstellung der Szene beginnt. Der Immediate Mode ist insbesondere für interaktive Anwendungen geeignet, die ohne spürbare Verzögerung die veränderte Szene darstellen sollen.

Der Immediate Mode hat jedoch auch Nachteile, insbesondere bietet er keine Möglichkeit, einem Primitiv einen Namen oder eine Identifikationsnummer (ID) zuzuordnen, um es auch nach der Definition erneut aufrufen zu können. Selbst wenn man die Definition des Primitivs in eine eigene Prozedur packt, wird bei jedem Aufruf der Prozedur das Primitiv neu erzeugt. Dieses Vorgehen ist ineffektiv, vor allem wenn die OpenGL-Anwendung auf einem Client ausgeführt wird und die Befehle zu einem rechenstarken Server weitergeleitet werden: Wenn identische Codesequenzen mehrmals ausgeführt werden sollen, müssen sie auch mehrmals übertragen werden. So muss z.B. bei jedem Aufruf einer bestimmten Textur (texture image) die Textur neu definiert und die zugehörige Bilddatei übertragen werden.

Zur Umgehung der Nachteile des Immediate Mode kann man so genannte „Darstellungslisten" einsetzen.

6.3.4.12 Darstellungslisten (display lists)

Eine Darstellungsliste (display list) ist die Zusammenfassung von Befehlen zu einer Einheit. Die Einheit erhält einen Namen, über den sie im Programm aufgerufen werden kann. Darstellungslisten sind bezüglich der Definition, des Aufrufs und deren Aufgabe vergleichbar mit Prozeduren.

Definition und Aufruf von Darstellungslisten
Bei der Definition von Primitiven definieren die Funktionen `glBegin(...)` und `glEnd()` den Anfang und das Ende der Definition. Bei der Definition einer Darstellungsliste treten die Funktionen `glNewList(...)` und `glEndList()` an deren Stelle. Dazwischen fügt der Entwickler wie bei der Definition von Primitiven entsprechende Befehle ein. Dabei können keine oder beliebig viele Primitive definiert, Zustandsvariablen gesetzt oder andere Befehle eingesetzt werden. Eine Darstellungsliste kann zur Laufzeit nicht verändert, sondern nur definiert, gelöscht oder durch erneute Definition überschrieben werden.
Die Syntax von `glNewList` lautet:

```
void glNewList( Glunit n, Glenum mode );
```

Der Parameter `n` ist ein Integer-Wert größer gleich 1, der die Darstellungsliste identifiziert, sozusagen der Name oder die ID der Darstellungsliste. Da Darstellungslisten mit Prozeduren vergleichbar sind, hat `n` die gleiche Aufgabe wie der Name einer Prozedur. Im folgenden Programmausschnitt ist in der ersten Zeile als

„Name der Darstellungsliste" eine Konstante mit dem Wert 1 definiert. Der Parameter `mode` gibt an, wie die Darstellungsliste compiliert wird, worauf wir jedoch nicht näher eingehen werden.

Die im Folgenden definierte Darstellungsliste erzeugt bei ihrem anschließenden Aufruf ein rotes Dreieck:

```
#define MyTriangle 1
glNewList( MyTriangle, GL_COMPILE );
   glColor(1.0, 0.0, 0.0);
   glVertex3i(0,0,0);
   glVertex3i(2,0,0);
   glVertex3i(1,2,0);
glEndList();
```

Die definierte Darstellungsliste wird wie folgt aufgerufen:

```
glCallList( MyTriangle );
```

In der Definition einer Darstellungsliste lassen sich andere Darstellungslisten zwar aufrufen, jedoch nicht definieren. Demnach muss nach dem Aufruf von `glNewList` vor dem nächsten `glNewList` ein `glEndList` stehen. Bei Primitiven ist hingegen das Verschachteln der Definitionen zulässig. Außerdem können in der Definition eines Primitivs Darstellungslisten aufgerufen und definiert werden und umgekehrt.

Ein Beispiel für die Programmierung ohne Darstellungslisten
Die meisten Objekte einer 3D-Szene ändern sich beim Ausführen eines Programms nicht. Soll z.B. ein Schachbrett samt Figuren aus der Sicht eines Spielers dreidimensional dargestellt werden, so wird jede der 32 Schachfiguren mit dem Aufruf passender OpenGL-Funktionen aufgebaut, z.B. durch Funktionen zur Definition von Polygonen, Streifen, Transformationen und Funktionen zur Änderungen der Zustandsvariablen, welche z.B. die Farbe festlegt.

In welchen Situationen muss die Geometrie einer Schachfigur und ihre weiteren Eigenschaften wie ihre Position und Farbe definiert werden?

- Immer wenn eine Schachfigur dargestellt wird, muss sie vollständig neu definiert werden, hierzu sind eventuell mehr als einhundert Zeilen Quellcode notwendig, welche die Geometrie festlegen. Obwohl z.B. die acht weißen Bauern identisch aussehen, müsste jeder weiße Bauer neu definiert werden. Und obwohl die schwarzen und weißen Bauern sich nur in der Farbe unterscheiden, müssten auch die schwarzen Bauern neu definiert werden.

- Bei jeder Änderung der Eigenschaften von visuellen Objekten der Szene müssen die Objekte neu definiert werden. Das ist beispielsweise der Fall, wenn sich die Position einer Schachfigur ändert.

Folge der Ineffizienz sind zum einen längere und damit schwerer zu handhabende Quellcodes. Dieses Problem lässt sich lösen, indem man für jeden Typ von Schachfigur (Bauer, Springer, ...) eine Prozedur implementiert. Vor jedem Aufruf der Prozedur übergibt man der Zustandsvariablen für die Transformation einen Wert, der die Position der Figur auf dem Schachbrett festlegt und weist eventuell der Zustandsvariablen für die Farbe den Wert für weiß oder schwarz zu.

Zum anderen hat die Ineffizienz negative Konsequenzen, wenn das Programm auf einem Client ausgeführt wird und ein Server die Rechenarbeit leistet. Hierbei werden die Aufrufe der OpenGL-Routinen zum Server übertragen. Anschließend generiert der Server aus der Szene das zweidimensionale Bild. Daraufhin wird das Ergebnis wieder zum Client gesendet. Bei jeder Definition einer Schachfigur werden die Aufrufe aller zu dieser Schachfigur gehörenden OpenGL-Routinen vom Client zum Server übertragen. Dies geschieht unabhängig davon, ob der Quellcode in einer Prozedur gekapselt ist oder nicht. Die Übertragung großer Datenmengen zwischen Client und Server kostet Zeit, insbesondere wenn Texturen definiert werden.

Anwendungsgebiete für Darstellungslisten
Falls eine Anwendung auf einem System aus Client und Server zum Einsatz kommen soll, sollten Darstellungslisten eingesetzt werden. Einerseits lässt sich der Quellcode mit Darstellungslisten genauso gliedern wie mit Prozeduren, andererseits lässt sich mit Darstellungslisten die mehrfache Defininition eines visuellen Objekts vermeiden: Der Entwickler definiert mit einer Darstellungsliste sozusagen eine Art statisches Primitiv. Eine Darstellungsliste kann zwar zur Laufzeit des Programms nicht mehr verändert werden, bei geschickter Modellierung lässt sich jedoch eine Darstellungsliste an vielen Stellen wiederverwenden.

Darstellungslisten sind sozusagen Inseln, auf denen die Regeln des Immediate Mode nicht gelten, wie ein Vergleich zwischen einem Primitiv und einer Darstellungsliste verdeutlicht: Primitive bestehen wie auch Darstellungslisten aus einer Auflistung von Befehlen. Bei der Ausführung eines Programms wird bei einem Primitiv zunächst der erste Befehl definiert und direkt anschließend ausgeführt; analog wird mit den restlichen Befehlen verfahren. Hingegen werden Darstellungslisten zunächst vollständig definiert. Erst durch Aufruf eines entsprechenden Befehls wird die Darstellungsliste ausgeführt.

Es sind drei Anwendungsgebiete für Darstellungslisten hervorzuheben ([Cla97], S. 56):

- Die hierarchische Modellierung,
- das effiziente Caching von Texturen auf dem Server,
- Die Ablage immer wieder benötigter Parameterkonstellationen.

Vorteile von Darstellungslisten
Der Einsatz von Darstellungslisten hat zwei Vorteile:

- Sparen von Entwicklungszeit durch das Wiederverwenden von Darstellungslisten.

- Eine häufig höhere Grafikleistung:
  Erstens erhält die Grafikhardware durch Darstellungslisten die Möglichkeit, den Inhalt der Darstellungslisten zu optimieren, was sich jedoch in der Regel nur bei mehrmaligem Gebrauch einer Darstellungsliste lohnt. Zweitens müssen in einem Netzwerk aus Clients und Server die Daten der Darstellungsliste nur einmal vom Client zum rechenstarken Server übertragen werden. Für jedes erneute Rendern der Elemente einer Darstellungsliste braucht nur der Name der Darstellungsliste übertragen zu werden. Drittens könnten Darstellungslisten zum schnelleren Zugriff in einem Geometrie-Cache gelagert werden. Und viertens können die Darstellungslisten auf mehrere interne Pipelines aufgeteilt werden, damit die Daten parallel verarbeitet werden können.

Speichern des Zustands
Darstellungslisten speichern nicht automatisch den Zustand ab. Das gilt sowohl für den beim Definieren als auch beim Ausführen gültigen Zustand. Die Ausführung einer Darstellungsliste wird ausschließlich durch den bei der Ausführung aktiven Zustand beeinflusst. Sollen Werte bestimmter Variablen des Zustandes oder der gesamte Zustand an eine Darstellungsliste gekoppelt werden, müssen sie bzw. er explizit in der Darstellungsliste gesetzt werden.

Die Unabhängigkeit vom Zustand birgt einen Vor- und einen Nachteil: Zum einen besteht eine Darstellungsliste aus weniger Daten, da sie nicht den Zustand abspeichert, so kann sie schneller übertragen werden. Zum anderen ist es für den Entwickler schwierig, Vorkehrungen für die korrekte parallele Bearbeitung mehrerer Darstellungslisten zu treffen, falls die Darstellungslisten z.B. auf einem Mehrprozessor-System parallel abgearbeitet werden sollen.

Wegen dieses Problems sollte man nicht immer Darstellungslisten anstatt Primitive einsetzen, auch wenn sich das visuelle Objekt nicht verändert. In der Regel sind in Darstellungslisten nur Gruppen von Befehlen zum Setzen von Zustandsvariablen oder Befehle zum Definieren einfacher Geometrien enthalten.

Behandlung von Darstellungslisten in der Rendering-Pipeline
Im Abschnitt 6.3.3 ’Rendering-Pipeline, Teil 1’ wurde bereits die Rendering-Pipeline dargestellt. Abbildung 6.33 zeigt, welche Rolle Darstellungslisten im Rahmen der Rendering-Pipeline spielen. Werden Geometrie- oder Bilddaten als Darstellungsliste definiert, wird die Darstellungsliste am Anfang der Rendering-Pipeline zwischengespeichert. Falls die Darstellungsliste an Hand ihres Namens bzw. ihrer Nummer mit `glCallList` aufgerufen wird, werden die Daten der Darstellungsliste aus diesem Zwischenspeicher gelesen.

6.3.4.13 Lizensierung

OpenGL muss nur von Unternehmen kostenpflichtig lizensiert werden, die ausführbare Dateien einer OpenGL-Implementierung erstellen oder verkaufen. Ausführbare Dateien werden beim Compilieren einer OpenGL-Implementierung erstellt, wenn diese verändert oder neu erstellt wird oder wenn sie auf einer an-

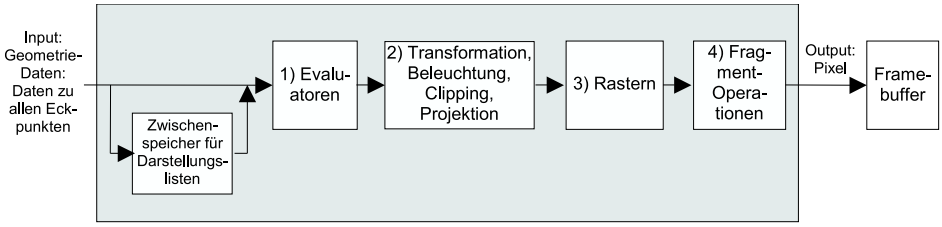


Abbildung 6.33: Die Rendering-Pipeline von OpenGL
In einem Schritt vor den Evaluatoren werden Darstellungslisten zwischengespeichert oder aus dem Zwischenspeicher gelesen.

deren Plattform compiliert wird. Demnach braucht man OpenGL nicht zu lizenzieren, wenn man Anwendungen implementiert, die auf OpenGL aufsetzen. Dazu müssen nur die ausführbaren Dateien der OpenGL-Implementierung eingebunden werden.

Die Lizenzierung von OpenGL erfolgt bei der ARB. Universitäten können eine Muster-Implementierung von OpenGL lizenzieren, von der ihnen auch der Quellcode zur Verfügung gestellt wird.

6.3.5 Rendering-Pipeline, Teil 2 (Separates Verarbeiten von Geometrie- und Bilddaten)

Wie im Abschnitt 6.3.4.6 'Separates Verarbeiten von Geometrie- und Bilddaten' bereits angesprochen, zerlegt eine OpenGL-Implementierung innerhalb der Rendering-Pipeline die Daten zunächst in Geometrie- und Bilddaten. Die Geometrie- und Bilddaten werden in jeweils separaten Pipelines verarbeitet und anschließend wieder zusammengeführt.

Der Vorteil von zwei separaten Pipelines liegt in der möglichen Steigerung der Effizienz, da so Geometrie- und Bilddaten im Prinzip parallel von jeweils spezialisierter Hardware verarbeitet werden können. Rechner mit Grafik-Hardware mit mehreren parallelen Pipelines sind inzwischen sogar im PC-Bereich zu finden. Eine Effizienzsteigerung ist auch möglich, indem die Daten parallel von mehreren Prozessoren verarbeitet werden.

6.3.5.1 Verarbeitung der Bilddaten

Zu Beginn der Verarbeitung der Bilddaten können Daten (wie auch bei der Verarbeitung der Geometriedaten) mit Hilfe der Darstellungslisten zwischengespeichert und später wieder verwendet werden, siehe Abbildung 6.34. Anschließend werden auf die Bilddaten so genannte Pixel-Operationen angewendet. Eine der Pixel-Operationen ist das Normieren auf ein einheitliches Farbformat, denn die Farbinformation der Pixel am Input kann in unterschiedlichen Formaten vorliegen.

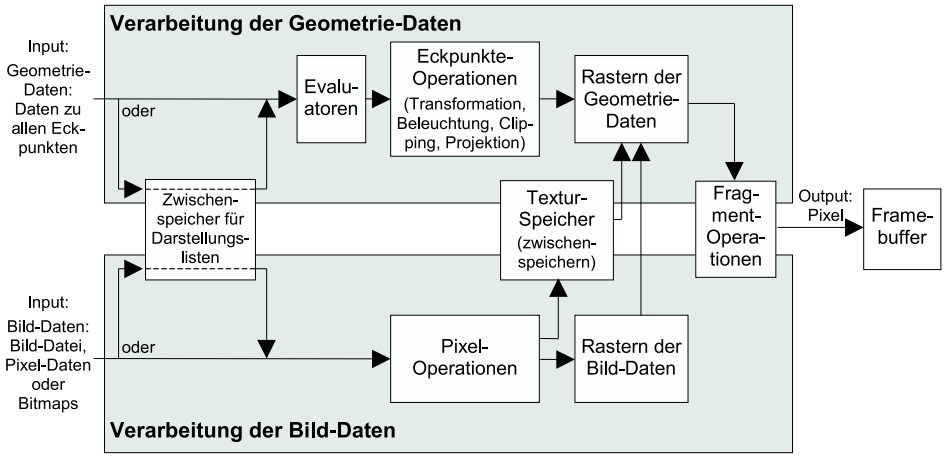


Abbildung 6.34: Die Rendering-Pipeline von OpenGL
Es wird zwischen der Verarbeitung von Geometrie- und Bilddaten unterschieden.

Anschließend werden die Bilddaten mit ihren transformierten Eckpunkten entweder gerastert oder sie werden im Textur-Speicher zur späteren Verwendung zwischengespeichert.

Bei der Verarbeitung der Bilddaten entfallen die beiden rechenintensiven Phasen der Transformation und Projektion: Bilddaten, die als Textur verwendet werden sollen, werden durch geeignete Funktionsaufrufe einem geometrischen Objekt zugewiesen. Dazu wird jedem zweidimensionalen Eckpunkt der Textur ein dreidimensionaler Eckpunkt des geometrischen Objekts zugeordnet. Bei der Verarbeitung der Geometriedaten werden die Eckpunkte des geometrischen Objekts transformiert und projiziert. Dadurch werden die Koordinaten der Eckpunkte der geometrischen Objekte auf der Bildebene ermittelt. Die Eckpunkte der Textur sind den entsprechenden Eckpunkten von geometrischen Objekten und somit den Eckpunkten auf der Bildebene zugeordnet. Deshalb brauchen bei der Verarbeitung der Bilddaten die Eckpunkte der Textur nicht nochmals transformiert und projiziert zu werden.

6.3.5.2 Verarbeitung der Geometriedaten

Die Verarbeitung der Geometriedaten benötigt viel Rechenzeit, da in diesen Bereich die rechenintensiven Eckpunkte-Operationen wie Clipping und Projektion fallen. Die Rastereinheit bereitet anschließend die Daten für den Framebuffer auf, indem sie für jedes Bildschirmpixel eine Farbe berechnet, wobei Textur-Daten aus dem Textur-Speicher oder der Rastereinheit für Bilddaten einbezogen werden können. Nach den schon im Abschnitt 6.3.3.2 ’Die Rendering-Pipeline als Blockdiagramm’ beschriebenen Fragment-Operationen erfolgt im letzten Schritt die Übertragung in den Framebuffer.

6.4 Vergleich: Java3D und OpenGL

6.4.1 Tabellarischer Vergleich der Leistungsmerkmale

Es folgt ein tabellarischer Vergleich der wichtigsten Leistungsmerkmale von Java3D und OpenGL.

Leistungsmerkmal	**Java3D**	**OpenGL**
Transformationen (verschieben, rotieren, skalieren)	X	X
Einfache geometrische Objekte (Punkte, Linien, Flächen)	X	X
Komplexere geometrische Objekte (Sphäre, Quader, ...)	X 1)	
Bézier–Kurven und –Flächen, Nurbs und Quadrigen		X
Szenegraph oder ähnliches	X	
Darstellungslisten (display lists)		X
Materialeigenschaften von Geometrien, inklusive Farben	X	X
Ambientes und paralleles Licht, punktförmige Lichtquelle und Spotlicht (mehrere Lichtquellen sind möglich)	X	X
Beeinflussbare Schattierungseffekte (Shading)	X	X
Nebel	X	X
Transparenz (α-Blending)	X	X
Texturen	X	X
2D-Text (rechteckiges Polygon, auf das eine Textur aufgebracht wird, die den Text darstellt)	X	
3D-Text (dreidimensionale geometrische Objekte, die in die Tiefe extrudierte Buchstaben ergeben)	X	
Hintergrundgestaltung (Background)	X	
Level of Detail (vom Betrachter weit entfernte Geometrien werden durch aus der Ferne ähnlich aussehende, aber weniger detaillierte Geometrien ersetzt)	X	
Antialiasing	X	X
Animationen und Interaktionen (Veränderung bestimmter Eigenschaften von visuellen Objekten inklusive der Größe und Lage) (Interpolatoren und Zeitgeber)	X	
Picking (Auswahl eines visuellen Objektes durch einen Mausklick auf das Objekt)	X	
Morphing (Veränderung der Form von visuellen Objekten; durch Interpolation wird aus wenigen ähnlichen visuellen Objekten eine Bildfolge, die den fließenden Übergang dieser Objekte ineinander zeigt)	X	
Pixelweise Beeinflussung des Framebuffers	X	X
Abspeichern und Auslesen von Szenen und Geometriedaten in/aus Dateien (Loaders, dabei sind mehr als ein Dutzend Dateiformate wählbar)	X	
Tonausgabe (Sound)	X	
Schnittstelle zu anderen Applikationen	X	X

1) Mit Hilfe der Klasse `com.sun.j3d.utils.geometry`

6.4.2 Prinzipielle Vorteile

Java3D und OpenGL haben völlig andere Zielsetzungen: Java3D setzt auf Benutzerfreundlichkeit und Plattformunabhängigkeit und OpenGL auf Flexibilität und Effizienz. Außerdem ist OpenGL ein Schnittstellen-Standard ohne Implementierung des Standards, Java3D ist ein Schnittstellen-Standard inklusive implementierter Bibliothek.

Vorteile von Java3D:

- Bei Java3D werden weder weitere Bibliotheken (außer OpenGL) benötigt noch Extensions. Java3D ist deshalb vollständig und zudem völlig plattformunabhängig.

  Insbesondere ist durch das in Java integrierte Paket Swing die Erstellung von anspruchsvollen Benutzeroberflächen möglich, die erstens plattformunabhängig zu implementieren sind und zweitens auf den verschiedenen Plattformen annähernd gleich aussehen. Bei OpenGL muss man hierfür z.B. auf GLUT zurückgreifen, das nur eine eingeschränkte Funktionalität bietet, oder es müssen je nach Betriebssystem bzw. Fenstersystem spezielle Zusätze eingebunden werden (z.B. GLX oder WGL), die unterschiedlich zu integrieren sind.

  Java3D gibt einen eigenen Szenegraphen vor. Außerdem können bei Java3D Geometrie-Modelle abgespeichert werden. OpenGL bietet diese Funktionalitäten nicht an. Um auf solche Funktionalitäten nicht verzichten zu müssen, greift der Entwickler in der Regel auf zusätzliche Software wie den Open Inventor zurück.

- Im Vergleich zu C oder C++ bei OpenGL ist die Programmiersprache Java benutzerfreundlicher und hilft dem Entwickler dabei, typische Programmierfehler zu vermeiden, man denke etwa an das Fehlen von Pointern, die Garbage Collection, die strikte Typisierung, das Fehlen von globalen Variablen und Konstanten, die kompromisslose Objektorientierung, genau eine Klassenhierarchie mit der Klasse `Object` als Wurzel u.a.

Vorteile von OpenGL:

- Im Gegensatz zu Java3D ist OpenGL ein De-facto-Standard, da er sich auf dem Markt durchgesetzt hat und von vielen Anwendungsprogrammen genutzt wird.

- OpenGL bietet nur die Grundfunktionalität, was den darauf aufbauenden Zusatzbibliotheken und Anwendungen viel Raum für Optimierungen läßt. Außerdem führt die Hardwarenähe zur höheren Darstellungsgeschwindigkeit.

- Wie der Name andeutet, ist OpenGL ein gewissermaßen offener Standard, da der Standard vollständig veröffentlicht ist und der Kreis der Unternehmen in der ARB prinzipiell nicht beschränkt ist. Die Protokolle der Sitzungen der ARB und damit auch die Hintergründe von Entscheidungen sind öffentlich zugänglich. Gegen eine Lizenzgebühr und unter Auflagen wird Universitäten der Sourcecode einer Muster-Implementierung des OpenGL-Standards zur Verfügung gestellt. Im Unterschied zu OpenGL ist bei Java3D nur die Schnittstelle veröffentlicht.

6.5 Ältere Grafik-Standards

Dank der kontinuierlichen Weiterentwicklung von OpenGL sind Grafik-Bibliotheken mit OpenGL-Schnittstelle inzwischen die meistverwendeten Grafik-Bibliotheken. Dabei wurden andere Schnittstellen-Spezifikationen wie GKS und PHIGS verdrängt. Trotzdem lohnt sich ein kurzer Rückblick auf die 3D-Grafik-Schnittstellen von 1980 bis heute.

Interessant sind aus historischen Gründen vor allem die Schnittstellen GKS bzw. GKS-3D sowie PHIGS inklusive PHIGS+ . Im Unterschied zu Java3D und OpenGL sind alle im Folgenden vorgestellten Schnittstellen in ISO-Normen festgeschrieben.

Abbildung 6.35: Die Entwicklung der Grafik-Bibliotheken bzw. -Schnittstellen

6.5.1 GKS

GKS steht für **G**rafisches **K**ern**s**ystem. Im Gegensatz zu Java3D und OpenGL wurde die Schnittstelle der Grafik-Bibliothek GKS 1982 in der GKS-ISO- und GKS-DIN-Norm festgelegt [ISO82],[DIN82]. Die wichtigsten Ziele der drei Grafik-Bibliotheken bzw. Schnittstellen sind jedoch identisch:

- Die Bibliotheken führen zur effizienteren Software-Erstellung.
- Die Konzentration auf die mehr oder weniger wesentlichen Eigenschaften und Fähigkeiten soll eine vielseitige Einsetzbarkeit eröffnen.
- Eine plattformunabhängige Schnittstelle soll zur weitgehenden Portabilität führen.

GKS war einer der ersten Standards im Grafik-Bereich. Mit seiner Terminologie hat GKS zur Vereinheitlichung der Bezeichnungen im Grafik-Bereich beigetragen.

GKS ähnelt OpenGL mehr als Java3D. Eine Implementierung des GKS-Standards ist wie auch jede OpenGL-Implementierung eine Zustandsmaschine. GKS stellt etwa 180 Funktionen zur Verfügung. Anders als bei OpenGL muss allerdings nicht jede Realisierung des GKS-Standards alle Funktionen implementieren.

Die grafischen Objekte (Geometrien) werden bei GKS Darstellungselemente genannt. Dazu gehören Polygonzüge (in GKS „Linienzüge” genannt), Punkte („Polymarken”), Text und gefüllte Polygone („Füllgebiete”).

Die Eigenschaften der Darstellungselemente wie Form und Farbe werden durch Darstellungsattribute festgelegt. Anders als OpenGL und Java3D berücksichtigt GKS zusätzlich eine spezielle Anforderung: Es kann vorkommen, dass auf unterschiedlichen Ausgabemedien die Darstellungselemente unterschiedlich dargestellt werden sollen. Beispielsweise sollen die Linien auf dem Bildschirm doppelt so dick dargestellt werden wie auf einem Plotter. Auch können die Darstellungsattribute abhängig vom Ausgabegerät gesetzt werden. Vor der Ausgabe ermittelt das Programm abhängig vom Ausgabegerät, auf welche Art die Ausgabe erfolgen soll. OpenGL und Java3D sehen solche Unterscheidungen nicht vor und überlassen die Darstellung der Ergebnisse dem Betriebssystem bzw. den Geräten und ihren Treibern.

Im Unterschied zu OpenGL und Java3D gibt es in GKS sechs Gruppen (in GKS „Klassen” genannt) logischer Eingabegeräte, die eine Abstraktion realer Eingabegeräte darstellen. Beispielsweise liefert die Klasse „Lokalisierer” die Koordinaten eines Punktes, und die Klasse „Text” liefert eine eingegebene Zeichenfolge. Der Vorteil liegt in der einfacheren Implementierung von interaktiven Anwendungen.

OpenGL geht bei der Abstraktion realer Eingabegeräte noch einen Schritt weiter, denn OpenGL definiert keine Routinen zur Abfrage der Daten von Eingabegeräten. Java bzw. Java3D kennt ausschließlich Eingabegeräte, die wie eine Maus funktionieren, sowie die Eingabemöglichkeit von Text, was für die allermeisten

Anwendungen ausreichend ist. Für weitere Anwendungen können durch die Objektorientiertheit und damit Erweiterbarkeit von Java und Java3D komfortabel weitere Eingabemöglichkeiten implementiert werden.

6.5.2 GKS-3D

Die Schnittstelle der Grafik-Bibliothek GKS-3D wurde in der GKS-3D-ISO-Norm festgelegt [ISO]. GKS-3D enthält die gesamte GKS-Schnittstelle ohne Änderung und zusätzliche Funktionen zur 3D-Eingabe und -Verarbeitung. Prinzipielle Unterschiede zu GKS gibt es vor allem bei der Rendering-Pipeline (bei GKS-3D „Viewing Pipeline" genannt, vgl. Kurseinheit 5 'Visibilität', Abschnitt 5.2.3 'Viewing Pipeline (Ausgabe-Darstellungsreihe)'), die um die dritte Dimension erweitert wurde. Die Pipeline von GKS-3D verarbeitet zweidimensionale Darstellungselemente mit dem gleichen Ergebnis wie GKS.

6.5.3 PHIGS

PHIGS steht für das **P**rogrammers's **H**ierarchical **I**nteractive **G**rafics **S**ystem. Die Schnittstelle der Grafik-Bibliothek PHIGS wurde 1989 in der PHIGS-ISO-Norm festgelegt [ISO89]. Neben der Darstellung von grafischen Objekten unterstützt es auch die Gruppierung und Hierarchisierung von Objekten. Diese Strukturierung ist eine wichtige Voraussetzung für das Modellieren von komplexen Szenen.

Die elementaren Objekte sind nicht wie bei GKS die Darstellungselemente, sondern die so genannten Strukturelemente. Strukturelemente enthalten alle Informationen zur Definition von Darstellungselementen. Ein Strukturelement kann z.B. aus den beiden Daten „Liniendicke: 4" und „Darstellungselement: Linienzug" bestehen.

Es ergeben sich Ähnlichkeiten zum Szenegraphen von Java3D: Die Strukturelemente können hierarchisch angeordnet werden, indem Einträge einer Struktur auf andere Strukturen verweisen. Durch die Verweise entsteht ein gerichteter Graph. Ein Strukturelement kann auch eine Transformation oder andere Operationen festlegen. Darstellungselemente existieren nur innerhalb von Strukturen. Für die Darstellung der Szene wird der gerichtete Graph traversiert. Erst bei der Traversierung werden die Attribute der Darstellungselemente durch die Strukturelemente auf dem jeweiligen Pfad ermittelt.

6.5.4 PHIGS+

PHIGS+ bzw. PHIGS PLUS bedeutet „PHIGS plus Beleuchtung und Flächen". Die Schnittstelle der Grafik-Bibliothek PHIGS+ wurde 1991 in der PHIGS PLUS-ISO-Norm festgelegt [ISO91]. PHIGS+ ist eine Erweiterung von PHIGS um

- Darstellungselemente auf höherem Level (z.B. ein Menge von Polygonen,

Gitternetze aus Dreiecken (Triangle Strip), Gitternetze aus Vierecken (Quadrilateral Mesh), NURBS).
Den Eckpunkten können bestimmte Informationen wie Farbe oder Normalenvektor zugeordnet werden.

- Lichtquellen, Schattierung, Rückseitenbearbeitung (z.B. Backface-Culling).
- Direkte Farbwahl:
  Bei PHIGS konnten Farben nur indirekt, d.h. über die Indizes einer Farbtabelle spezifiziert werden.

6.5.5 Vergleich von ISO-Standards und Industriestandards

Schnittstellen-Standards lassen sich in ISO-Standards und Industriestandards unterteilen. Die Grafik-Schnittstellen-Standards GKS, GKS-3D, PHIGS und PHIGS+ wurden von bestimmten Gremien definiert und als ISO-Standards eingetragen. Im Unterschied dazu sind OpenGL, Java3D, Direkt3D und die meisten anderen Schnittstellen-Standards Industriestandards.

Ein Nachteil von ISO-Standards ist der, dass es entweder keine Organisation und kein Unternehmen gibt, die bzw. das für die Pflege und Weiterentwicklung des Standards verantwortlich ist. Oder die verantwortliche Organisation oder Firma ist finanziell oder in anderer Weise nicht abhängig von dem Erfolg des Standards. Zudem werden ISO-Normen nicht geschaffen, um regelmäßig überarbeitet zu werden. So wurden bei GKS und PHIGS neue Funktionalitäten hinzugefügt, indem man neue Standards (GKS-3D und PHIGS+) definierte, anstatt die alten Standards entsprechend anzupassen.

Bei Industriestandards wird für das Hinzufügen neuer Funktionalität stattdessen der Standard um die neue Funktionalität erweitert und die Versionsnummer erhöht oder wie bei OpenGL zusätzliche Bibliotheken wie GLU, GLUT oder Open Inventor kreiert. Außerdem sorgt bei Industriestandards schon allein der Wettbewerbsdruck auf den für die Standards verantwortlichen Unternehmen für eine hohe Bereitschaft, innovative Neuerungen im Bereich der Algorithmen, der Hardware oder der Anwendungen zu berücksichtigen.

Wegen der flexiblen Erweiterbarkeit und stetigen Weiterentwicklung haben bei Grafik-Bibliotheken Industriestandards die ISO-Standards zum größten Teil abgelöst.

6.6 Installation von Java3D

In diesem Abschnitt wird die Installation der Java3D 1.2 API auf einem Rechner mit dem Betriebssystem Windows 95/98 beschrieben. Die Installationsanleitung lässt sich jedoch auch auf andere Betriebssysteme übertragen. Da Java3D die Installation einer Implementierung von OpenGL sowie die Programmiersprache Java voraussetzt, erfolgt die Installation in drei Schritten. Im erste Schritt wird OpenGL installiert, im zweiten folgt die Installation vom Java 2 SDK, und im dritten Schritt befassen wir uns mit der Installation von Java3D.

6.6.1 Schritt 1: Installation von OpenGL

Für Java3D wird eine Implementierung von OpenGL in der Version 1.1 (oder höher) benötigt. In Windows 95 (ab OEM Service Release (OSR) 2), in Windows NT 4.0 (ab Service Pack 3) und in Windows 98 / 2000 / ME ist OpenGL 1.1 schon enthalten. In diesen Fällen gehen Sie bitte zu Schritt 2.

Andernfalls muss für Java3D auch OpenGL installiert werden. Dazu kann entweder OpenGL durch das Installieren von Treibern zu bestimmten Grafik- und Grafik-Beschleuniger-Karten automatisch mitinstalliert werden bzw. worden sein. Oder man erhält die nötigen Dateien einer OpenGL-Implementierung aus dem Internet, beispielsweise über die Seite:

```
http://www.opengl.org/
```

Wählen Sie dort den Eintrag „Downloads”.
Folgen Sie den Anweisungen.
Downloaden Sie schließlich die ausführbare Datei (`opengl95.exe` oder ähnlich) und speichern Sie sie in ein Verzeichnis ab, z.B. in

```
C:\Programme\OpenGL
```

Führen Sie die Datei durch einen Doppelklick aus. Die Datei wird daraufhin automatisch entpackt. Eine der so entstandenen Dateien ist die Readme-Datei.

Lesen Sie die Readme-Datei.
Kopieren Sie alle entpackten Dateien (ohne die Readme-Datei und die gerade angeklickte Datei `opengl95.exe`) in das Verzeichnis

```
c:\windows\system
```

bzw. in das in der Readme-Datei beschriebene Verzeichnis.

6.6.2 Schritt 2: Installation vom Java 2 SDK

Vor der Installation von Java 3D muss das Java 2 SDK (software development kit, Entwicklungsumgebung) ab Version 1.2.2 installiert worden sein. Diese Software wird auch als JDK (Java development kit) 1.2.2 bezeichnet. Um zu erfahren, ob bereits das Java 2 SDK ab Version 1.2.2 installiert worden ist, öffnen Sie bitte ein DOS-Fenster (Start | Programme | MS-DOS Eingabeaufforderung). Geben Sie ein:

```
javac -version
```

Vergessen Sie nicht das „c" in `javac`. Nach <Enter> sollte eine Ausgabe erscheinen, die wie die folgende beginnt:

```
javac version "1.2.2"
```

Falls die ausgegebene Versionsnummer mindestens 1.2.2 ist, gehen Sie bitte zu Schritt 3. Anderenfalls oder wenn noch kein Java SDK installiert ist, muss eine Java SDK-Version ab Version 1.2.2 installiert werden. Das Java 2 SDK ist kostenlos bei der Firma JavaSoft zu beziehen:

```
http://java.sun.com/j2se/
```

oder

```
http://java.sun.com/j2se/1.3/download-windows.html
```

Wählen Sie die neueste Version des Java 2 SDK. Im Folgenden gehen wir von der Version 1.3.0 aus. Das SDK wird in diesem Fall in das Verzeichnis

```
C:\jdk1.3
```

installiert. Die herunterzuladene Datei des Java 2 SDK Version 1.3.0 (Anfang 2001)

```
j2sdk-1_3_0_02-win.exe
```

ist etwa 31 Megabyte groß. Auf einer der Seiten, von der Sie das Java SDK herunterladen können, ist eine detaillierte Anleitung zur Installation des SDK zu finden. Folgen Sie den Anweisungen der Installationsanleitung und setzen Sie, wie in der Anleitung beschrieben, in der Datei `autoexec.bat` die Variablen `PATH` und eventuell `CLASSPATH`. In der Zeile, in der die `PATH`-Variable gesetzt wird, muss der Pfad zum SDK bzw. JDK vor dem Pfad zum Verzeichnis `Windows` und - falls der Pfad angegeben ist - vor dem Pfad zum Verzeichnis `Windows\System` stehen.

Es folgt ein Beispiel für das Setzen der Variable `PATH`. Die Variable `CLASSPATH` wird in dem Beispiel nicht gesetzt, da dies ab der Version 1.2 nicht mehr nötig ist.

```
SET PATH=.;C:;C:\jdk1.3\bin;C:\windows;C:\windows\command
```

Die Installation des Java SDK ist somit abgeschlossen. Nach der Installation sollte das Java SDK z.B. in das Verzeichnis

```
C:\jdk1.3
```

installiert worden sein. Fahren Sie sicherheitshalber den Rechner herunter und anschließend wieder hoch. Das soll sicherstellen, dass die Änderungen in der Datei `autoexec.bat` für alle anschließend geöffneten DOS-Fenster gültig werden.

Als Nächstes wird die Installation mit drei Tests überprüft.

Test 1: Öffnen Sie wie in Schritt 2 ein DOS-Fenster (Start | Programme | MS-DOS Eingabeaufforderung). Geben Sie ein:

```
javac -version
```

Nach <Enter> sollte eine Ausgabe erscheinen, die wie die folgende beginnt:

```
javac version "1.3.0"
```

Test 2: Geben Sie daraufhin ein:

```
appletviewer
```

Es sollte keine Fehlermeldung wie „Befehl oder Dateiname nicht gefunden" erscheinen, sondern Angaben zur Benutzung des Befehls. Ansonsten überprüfen Sie die `PATH`-Variable und starten Sie gegebenenfalls den Rechner neu.

Test 3: Erzeugen Sie das Verzeichnis (bzw. den Ordner)

```
C:\javatest.
```

Kopieren Sie anschließend von der CD-ROM zum Kurs die Datei `hello.java` in dieses Verzeichnis. Die Datei enthält das folgende kurze Beispielprogramm:

```
import java.applet.Applet;
import java.awt.Graphics;

public class hello extends Applet{
   public void paint(Graphics g){
      g.drawString("Hello World!", 20, 25);
   }
}
```

Beachten Sie bitte: Der Dateiname (ohne ".java") muss mit dem Namen der Klasse (vierte Zeile im Beispiel-Programm, nach `class`) übereinstimmen.

Gehen Sie im geöffneten DOS-Fenster in das Verzeichnis

```
C:\javatest
```

und geben Sie ein:

```
javac hello.java
```

Falls keine Fehlermeldung erscheint, führen Sie das Programm aus. Dazu müssen Sie die Datei `hello.html` von der CD-ROM zum Kurs in das Verzeichnis

```
C:\javatest
```

kopieren. Die Datei enthält nur drei Zeilen:

```
<HTML>
<applet code="hello.class" width=300 height=50></applet>
</HTML>
```

Geben Sie im DOS-Fenster ein:

```
appletviewer hello.html
```

Es sollte sich ein Fenster wie in der Abbildung 6.36 öffnen.

Abbildung 6.36: Applet-Viewer: `Hello.class`

Sie können nun die Datei

```
j2sdk-1_3_0_02-win.exe
```

wieder löschen.

Wir empfehlen, sich zusätzlich die umfangreiche Dokumentation (23 Megabyte) zum SDK herunterzuladen und zu installieren, wenn Sie sich näher mit Java auseinandersetzen wollen.

6.6.3 Schritt 3: Installation von Java 3D

1. Gehen Sie im Internet auf die Seite

   ```
   http://java.sun.com/products/java-media/3D/download.html
   ```

   Downloaden Sie die aktuellste Java 3D-Version. In der vorliegenden Installationsanleitung gehen wir von der Version 1.2 aus. Wählen Sie beim Downloaden "Java 3D Windows Runtime and Examples for the JDK". Beim Herunterladen erhalten Sie schließlich die ausführbare Datei

   ```
   java3d1_2-win-opengl_sdk.exe
   ```

   wobei

   ```
   ,,1_2''
   ```

   für die Version 1.2 steht. Die Datei ist etwa 4 Megabyte groß.

2. Machen Sie im Dateimanager bzw. Explorer einen Doppelklick auf diese Datei. Java3D wird nun installiert. Wenn Sie dabei nach dem Verzeichnis gefragt werden, in das Java3D installiert werden soll, geben Sie den Pfad zum Verzeichnis an, in dem das Java 2 SDK installiert ist, z.B.

   ```
   C:\jdk1.3
   ```

3. Sie können nun die Datei

   ```
   java3d1_2-win-opengl_sdk.exe
   ```

   wieder löschen.

4. Nun können Sie ebenfalls von der Seite

   ```
   .../3D/download.html
   ```

   die Dokumentation zu Java3D downloaden.
   Lesen Sie dazu die README-Datei zu Java3D.

6.6.4 Testen der Installation

Die Installation von OpenGL, Java und Java 3D ist somit abgeschlossen. Bei der Installation von Java3D erhält man automatisch unter anderem das Zusatz-Package

```
com.sun.j3d.utils.geometry
```

Das Package enthält insbesondere die Klassen `ColorCube`, `Box`, `Cylinder`, `Cone` und `Sphere`, mit denen einfache virtuelle geometrische Objekte in eine virtuelle Szene eingefügt werden können.

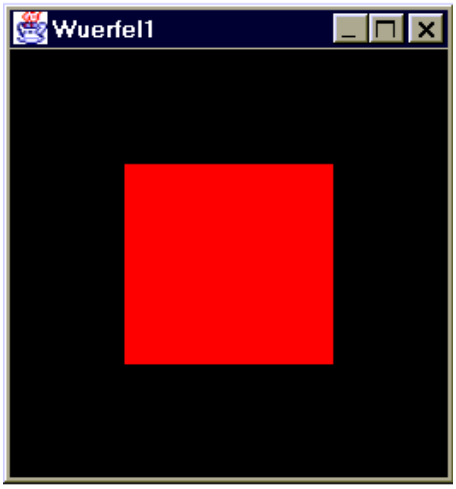


Abbildung 6.37: Screenshot des Programms `Wuerfel1.java`

Zuletzt testen wir die Installation. Dazu compilieren wir das einfache Beispiel-Programm `Wuerfel1.java`. Das Programm ist das Beispiel Nr. 1 aus dem Abschnitt 6.2.5.2 'Beispiel Nr.1:"Wuerfel1", Teil1'. Dort ist der Quellcode dieses Beispiels abgedruckt. Die Datei `Wuerfel1.java` und die zugehörige ausführbare Datei `Wuerfel1.class` sind auf der CD-ROM zum Kurs zu finden.

Gehen Sie im DOS-Fenster in das Verzeichnis

```
C:\javatest
```

(falls nicht schon geschehen).
Geben Sie ein:

```
javac Wuerfel1.java
```

und daraufhin

```
java Wuerfel1
```

Es öffnet sich ein Fenster, in dem ein rotes Quadrat auf schwarzem Grund zu sehen ist, siehe Abbildung 6.37.
Falls das Fenster zu sehen ist, ist die Installation erfolgreich abgeschlossen worden.

6.7 Die CD-ROM zum Kurs

Alle Beispiele sind auf der zum Kurs erscheinenden CD im Verzeichnis

```
\k6\java3d\examples
```

zu finden.

Die Dateien auf dieser CD sind auch zu beziehen unter:

```
http://www.informatik.fernuni-hagen.de/import/pi2/OnlineAngebote.html
```

Eventuell sind die dort zu erhaltenden Dateien aktueller als die auf der CD.

Einige der Beispiele sind dem Dokument „The Java 3D Tutorial” der Firma Sun entnommen ([Bou99]).

6.8 Bemerkungen zum Literaturverzeichnis

Zu Java3D wie auch zu OpenGL existieren mehrere Fachbücher und zahlreiche Artikel und Beispielsammlungen im Web.

6.8.1 Literatur zu Java3D

Bücher

Zu Java3D sind Ende 2000 folgende Bücher verfügbar:

- Java 3D Programming [Sel00]
- Ready-to-Run Java 3D [BP99]
- The Java 3D API Specification [SRD97]

Die Java3D (API) Spezifikation ist auch im Web erhältlich, siehe unten.

Im Web

Für die Literatursuche über Java3D im Web lohnt sich generell der Besuch der Seite

```
http://java.sun.com/products/java-media/3D/index.html
```

Zu Java3D stellt Sun ein ausführliches Tutorial zur Verfügung [Bou99]. Ende 2000 lag das Tutorial in der Version v1.5 vor. Inzwischen ist das Tutorial mit einem Umfang von etwa 300 Seiten eher ein Fachbuch als eine Einleitung für den schnellen Einstieg. Das Kapitel 6.2 über Java3D der vorliegenden Kurseinheit basiert vor allem auf dem Tutorial, insbesondere auf den Kapiteln 0, 1, 2, 4 und 6.

Um komplexere Beispielprogramme zu erstellen als diejenigen, die in der vorliegenden Kurseinheit oder im Tutorial beschrieben werden, ist eine intensive Auseinandersetzung mit der „Java3D (API) Spezifikation" unabdingbar. Sie ist im Web [SUN00] oder als Buch [SRD97] erhältlich.

Im Web sind außerdem eine Reihe von Beispielsammlungen und von Artikeln zu einzelnen Aspekten von Java3D zu finden. Falls man im Internet mit Hilfe von Suchmaschinen weiteres über Java3D erfahren will, geben wir einen Tip: Wenn man nach Seiten sucht, welche die Begriffe „java3d" oder „java 3D" oder „opengl" (ohne Berücksichtigung der Groß- und Kleinschreibung) enthalten, erhält man bis zu mehrere hunderttausend Treffer. Deshalb lohnt es sich, nur nach Seiten zu suchen, die auch zusätzlich einen der Begriffe „scene graph" oder „szene gra*" oder „szene-gra*" oder „szenegra*" enthalten. Das Sternchen in „gra*"

soll die unterschiedliche Schreibweise von „Graf” bzw. „Graph” berücksichtigen. Wesentlich mehr Seiten bekommt man, wenn man statt dessen „szene” oder „scene” angibt.

6.8.2 Literatur zu OpenGL

Wer Informationen zu OpenGL sucht, sollte zunächst auf der Seite

```
http://www. opengl.org/
```

suchen. Dort ist auch eine kurze Beschreibung zu allen im Folgenden angegebenen Büchern (bis auf das letzte) zu finden.

Bücher

Ein Ausschnitt aus den etwa ein Dutzend Büchern zu OpenGL:

- The OpenGL Programming Guide (The Official Guide to Learning OpenGL Version 1.2) [WND+99]
- OpenGL SuperBible [RSWS99]
- OpenGL Reference Manual [ARB99]
- Computer Graphics Using OpenGL [Hil00]
- Programmieren mit OpenGL [Cla97]

Das Kapitel 6.3 über OpenGL der vorliegenden Kurseinheit basiert vor allem auf dem ersten und letzten Buch. Das erste Buch ist in einer älteren Auflage auch in einer Online-Version verfügbar, siehe unten.

Im Web

Zu OpenGL findet man zahlreiche Beispiele unterschiedlichster Komplexität im Web. Leider gibt es zu OpenGL Ende 2000 kein Dokument, das von den Seiten von SGI oder der ARB kostenlos herunterzuladen und mit dem Tutorial zu Java3D vergleichbar ist.

Auf der Seite

```
http://www.opengl.org/
```

sind Links zu mehreren hundert Artikeln, die vorwiegend einzelne Aspekte von OpenGL thematisieren. Wer das erste Programm mit OpenGL erstellen will, dem sei auch [Wil95] und [Kun94] zu empfehlen.

Als Nachschlagewerke zum Programmieren sind zu empfehlen:

- The OpenGL Graphics System: A Specification (Version 1.2.1) [SA99]
- The OpenGL Programming Guide (The Official Guide to Learning OpenGL, Version 1.1) [SGI97]

6.9 Übungsaufgaben

Aufgabe 1:

Bei einigen Grafik-Bibliotheken werden virtuelle Szenen modelliert, indem man die Elemente der Szenen (virtuelle Objekte und Lichtquellen) unabhängig voneinander definiert. Bei Java3D stehen die Elemente durch die Anordnung im Szenegraphen in Beziehung zueinander.

Geben Sie Vor- und Nachteile des Modellierens mit Hilfe einer Struktur wie des Szenegraphens an, und zwar jeweils für

a) Entwickler, die ihre 3D-Anwendungen mit Hilfe von Grafik-Bibliotheken erstellen, und

b) Benutzer von 3D-Anwendungen.

Aufgabe 2:

a) In der folgenden Abbildung 6.38 ist ein geometrisches Objekt zu sehen. Bei dem Objekt handelt es sich um ein ebenes Achteck in der x-y-Ebene. Das Achteck approximiert eine Kreisscheibe.

   Erstellen Sie ein Programm mit dem Namen `Achteck1.java`, mit dem das geometrische Objekt erzeugt werden kann. Definieren Sie dazu das Achteck mit der Klasse `TriangleArray`. Der Abstand (”Radius”) der äußeren Eckpunkte zum Ursprung soll 0.5 sein. Definieren Sie das Achteck mit einer einzigen Instanz von `TriangleArray` und verzichten Sie auf Transformationen.

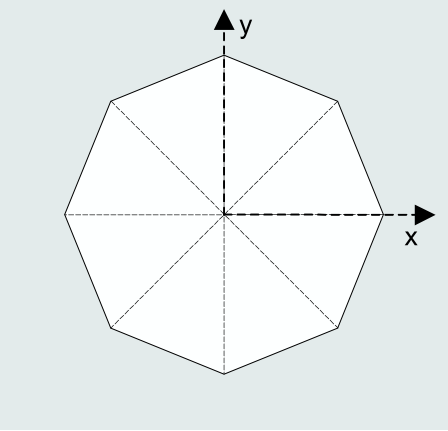


Abbildung 6.38: Schematische Darstellung des mit der Klasse `TriangleArray` erstellten Achtecks

Beim Ausführen des Programms erhalten Sie das in Abbildung 6.39 dargestellte Fenster:

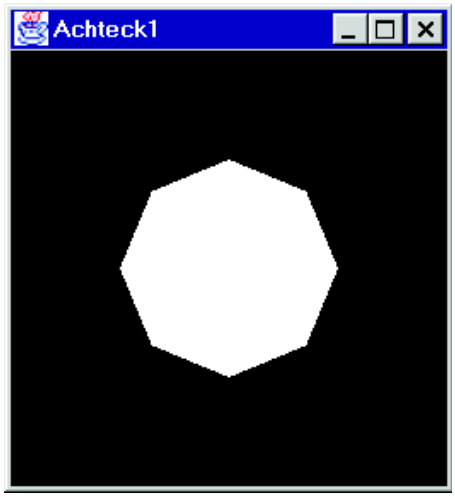


Abbildung 6.39: Screenshot der Ausgabe des Programms `Achteck1.java`

Wir empfehlen, den Konstruktor der Klasse `TriangleArray` mit

```
TriangleArray triangles = new
    TriangleArray(N,TriangleArray.COORDINATES);
```

aufzurufen anstatt mit

```
TriangleArray triangles = new
    TriangleArray(N,TriangleArray.COORDINATES
        | TriangleArray.COLOR_3);
```

Denn dann ist die Default-Farbe weiß statt schwarz, und Sie können sich die explizite Festlegung der Farbe jedes Eckpunktes (mit der Methode `triangles.setColor(...)`)
ersparen.

b) Geben Sie eine Methode `createSceneGraph()` an, mit der das geometrische Objekt aus Teil a) erzeugt werden kann, jedoch diesmal mit der Klasse `TriangleFanArray`. In der Abbildung 6.40 sind die 10 Eckpunkte des Fächers aus Dreiecken beispielhaft nummeriert.

Sie können den Konstruktor von `TriangleFanArray` wie folgt aufrufen:

```
int N = 10:     // Anzahl Eckpunkte
int stripVertexCounts[] = {N};  // siehe Erklaerung unten
TriangleFanArray tfa = new TriangleFanArray(N,
                                TriangleFanArray.COORDINATES,
                                   stripVertexCounts);
```

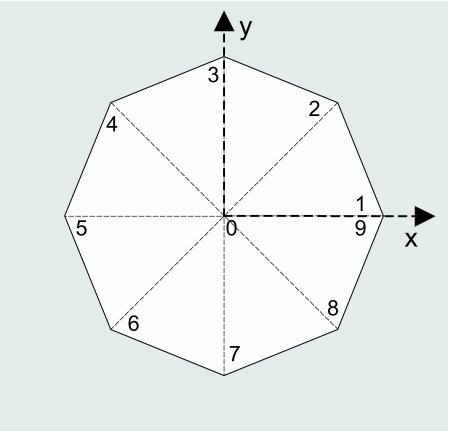


Abbildung 6.40: Schematische Darstellung des mit der Klasse `TriangleFanArray` erstellten Achtecks

Erklärung zu `stripVertexCounts`:
Mit einer einzigen Instanz von `TriangleFanArray` können mehrere geometrische Objekte (Fächer von Dreiecken) definiert werden. Die Variable `stripVertexCounts` gibt an, mit wie vielen Eckpunkten jeweils die einzelnen Objekte definiert werden.
Beispiel:
Wenn mit derselben Instanz von `TriangleFanArray` drei geometrische Objekte definiert werden sollen (das erste Objekt mit 10, das zweite mit 3 und das dritte mit 100 Eckpunkten), würde die zugehörige Zeile lauten:

```
int stripVertexCounts[] = {10, 3, 100};
```

Weitere Informationen über die Klasse `TriangleFanArray` entnehmen Sie bitte dem Tutorial der Forma SUN ([Bou99]) oder der ”Java3D API Specification” ([SRD97]).

Aufgabe 3:

a) Zeichnen Sie einen Szenegraphen, mit dem die folgende Szene dargestellt wird: Die Szene besteht aus fünf ebenen visuellen Objekten in der x-y-Ebene, siehe die Abbildung 6.41.

Bei den fünf Objekten handelt es sich um ein achsenparalleles Quadrat und vier Dreiecke. Die Dreiecke sind rechtwinklig und gleichschenklig. Die Seitenlänge des Quadrats ist 0,4. Die Katheten der Dreiecke sind ebenfalls achsenparallel und 0,4 lang. Der Mittelpunkt des Quadrates liegt im Ursprung bei (0;0;0). Alle fünf Objekte sind weiß.

Die Dreiecke sollen erzeugt werden, indem ein einziges Dreieck definiert und anschließend dreimal verschoben und/oder gedreht wird.

`TriangleArray` wird von der Klasse `NodeComponent` abgeleitet, siehe das Klassendiagramm in Abschnitt 6.2.12 ’Darstellung der Klassenhierarchie’.

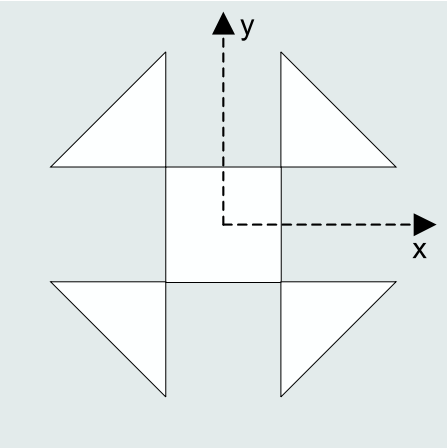


Abbildung 6.41: Schematische Darstellung der Szene aus einem Quadrat und vier Dreiecken

Da mehrere Referenzen auf das gleiche NodeComponent-Datenelement zeigen dürfen, reicht es aus, eine einzige `TriangleArray`-Instanz zu erzeugen, deren Form durch drei Eckpunkte festgelegt wird. Mit TransformGroup-Datenelementen wird das Dreieck verschoben und/oder gedreht.

Nummerieren Sie in Ihrer Zeichnung die TransformGroup-Datenelemente und beschreiben Sie zu jedem dieser Datenelemente die zugehörige Transformation.
Es sind verschiedene Lösungen möglich.

b) Schreiben Sie die zur Szene passende Methode `createSceneGraph()`.

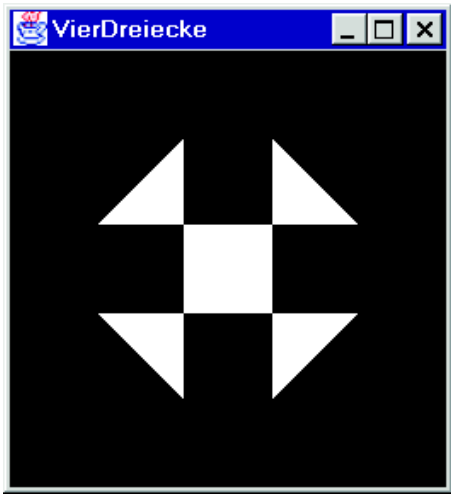


Abbildung 6.42: Screenshot der Ausgabe des Programms `VierDreiecke.java`

Der Konstruktor von `QuadArray` ist identisch zu handhaben wie der Konstruktor der Klasse `TriangleArray`, siehe das Beispiel Nr.7 aus dem Unterkapitel zu Java3D.

Wenn man das gesamte Beispiel-Programm `VierDreiecke.java` nennt, könnte das Ergebnis wie in Abbildung 6.42 aussehen.

Aufgabe 4:

a) OpenGL-Implementierungen zerlegen die Daten innerhalb der Rendering-Pipeline in Geometrie- und Bilddaten. Überlegen Sie, ob bei Java3D ähnlich vorgegangen wird. Es sind verschiedene Antworten möglich.

b) In der Abbildung 6.33 ist eine vereinfachte Darstellung der Rendering-Pipeline (inklusive der Handhabung von Darstellungslisten) zu sehen. Wie könnte diese Darstellung bei Java3D aussehen?

c) OpenGL kann Szenen in der Regel schneller darstellen als Java3D. Nennen Sie fünf Gründe für die höhere Effizienz.

Aufgabe 5:

a) Eine Darstellungsliste ist die Zusammenfassung von Befehlen zu einer Einheit. Mit welchen Elementen wird bei Java3D Ähnliches erreicht?

b) Nennen Sie alle im Unterkapitel 6.3 genannten Vor- und Nachteile von Darstellungslisten.

6.10 Lösungen zu den Übungsaufgaben

Lösung 1:

a) Vor- und Nachteile für Entwickler, die ihre 3D-Anwendungen mit Hilfe von Grafik-Bibliotheken erstellen:

Vorteile:

- Mehr Effizienz:
  Der Quellcode wird übersichtlicher, da Wiederholungen oft vermieden werden können. Daraus folgt ein geringerer Aufwand beim Modellieren und Ändern der Szene.
- Für Änderungen an der Szene muss häufig nur ein kleiner Teil des Codes verändert werden.
- Schnellere Bilddarstellung, falls bestimmte Teilbäume des Szenegraphen bei Veränderungen der Szene nicht neu compiliert werden müssen.
- Ein Szenegraph oder eine ähnliche Struktur setzt eine genaue Planung der Szene voraus und erzwingt somit ein strukturiertes Vorgehen. Das verringert die Fehlerquote und vermeidet umfassende Änderungen in einem fortgeschrittenen Stadium der Entwicklung von 3D-Anwendungen.

Nachteile:

- Eine eventuell langsamere Bilddarstellung:
  Eine komplexe verzeigerte Struktur wie ein umfangreicher Szenegraph muss erst von den Routinen der Grafik-Bibliothek in einzelne einfache Graphikbefehle überführt werden, da erst diese einfachen Grafikbefehle von der Hardware verarbeitet werden können.

b) Vor- und Nachteile für Benutzer von 3D-Anwendungen:

Vorteile:

- Die Anwendungen preiswerter und schneller ausgereift.

Nachteile:

- Eine eventuell langsamere Bilddarstellung:
  Im Gegensatz zur Gruppe der Entwickler ist dieser Nachteil für viele Anwender entscheidend, insbesondere, wenn sie mit interaktiven Anwendungen wie Computerspielen oder CAD-Software arbeiten.

Lösung 2:

a) Der Quellcode des Programms `Achteck1.java`:

```
import java.applet.Applet;
import java.awt.BorderLayout;
import java.awt.Frame;

import javax.media.j3d.Canvas3D;
import javax.media.j3d.BranchGroup;
import javax.media.j3d.Shape3D;

import javax.media.j3d.TriangleArray;

import javax.vecmath.Point3f;

import com.sun.j3d.utils.universe.SimpleUniverse;
import com.sun.j3d.utils.applet.MainFrame;


public class Achteck1 extends Applet {
  public Achteck1() {
    setLayout(new BorderLayout());
    Canvas3D canvas3D = new Canvas3D(
                  SimpleUniverse.getPreferredConfiguration());
    add(”Center”, canvas3D);

    SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
    simpleU.getViewingPlatform().setNominalViewingTransform();

    BranchGroup scene = createSceneGraph();


    scene.compile();
    simpleU.addBranchGraph(scene);
  }


  public BranchGroup createSceneGraph() {
    BranchGroup rootBg = new BranchGroup();
    TriangleArray triangles = new TriangleArray(24,
                                        TriangleArray.COORDINATES);

    double radius = 0.5f;
    double winkel = Math.PI/4.0d;

    for (int i=0; i < 8; i++){
      triangles.setCoordinate(i*3,
        new Point3f( 0.0f, 0.0f, 0.0f ));
      triangles.setCoordinate(i*3+1,
        new Point3f((float) (radius*Math.cos(i *winkel)),
                    (float) (radius*Math.sin(i *winkel)),0.0f ));
      triangles.setCoordinate(i*3+2,
        new Point3f((float) (radius*Math.cos((i+1)*winkel)),
                    (float) (radius*Math.sin((i+1)*winkel)),0.0f ));
    }

    rootBg.addChild(new Shape3D(triangles));
```

```
    return rootBg;
  }

  public static void main(String[] args) {
    Frame frame = new MainFrame(new Achteck1(), 256, 256);
  }
}
```

b) Quellcode der Methode `createSceneGraph()` des Programms `Achteck2.java`:

```
  public BranchGroup createSceneGraph() {
    BranchGroup rootBg = new BranchGroup();
    int N = 10; // Anzahl Eckpunkte
    int stripVertexCounts[] = {N};
    TriangleFanArray tfa = new TriangleFanArray(N,
                                    TriangleFanArray.COORDINATES,
                                    stripVertexCounts);
    double radius = 0.5f;
    double winkel = Math.PI/4.0d;
    tfa.setCoordinate( 0, new Point3f( 0.0f, 0.0f, 0.0f ));
    for (int i=1; i < N; i++){
            tfa.setCoordinate(i,
              new Point3f((float) (radius*Math.cos((i-1)*winkel)),
                          (float)(radius*Math.sin((i-1)*winkel)),0.0f ));
    }
    rootBg.addChild(new Shape3D(tfa));
    return rootBg;
  }
```

Übrigens ist in der Import-Liste am Anfang des Programms im Vergleich zum Programm aus Aufgabenteil a) die Zeile

```
import javax.media.j3d.TriangleArray;
```

auszutauschen gegen

```
import javax.media.j3d.Triangle<b>Fan</b>Array;
```

Lösung 3:

a) Der Szenegraph ist in Abbildung 6.43 dargestellt.
Wie in der Aufgabenstellung bereits erwähnt, sind verschiedene Lösungen möglich.

Die in der Abbildung 6.43 angegebenen Namen der Instanzen (z.B. `rotateTg1` für eine Instanz der Klasse `TransformGroup`) sind nicht notwendig, passen aber zum Muster-Quellcode der Methode `createSceneGraph()`, siehe die Lösung zum Aufgabenteil b).

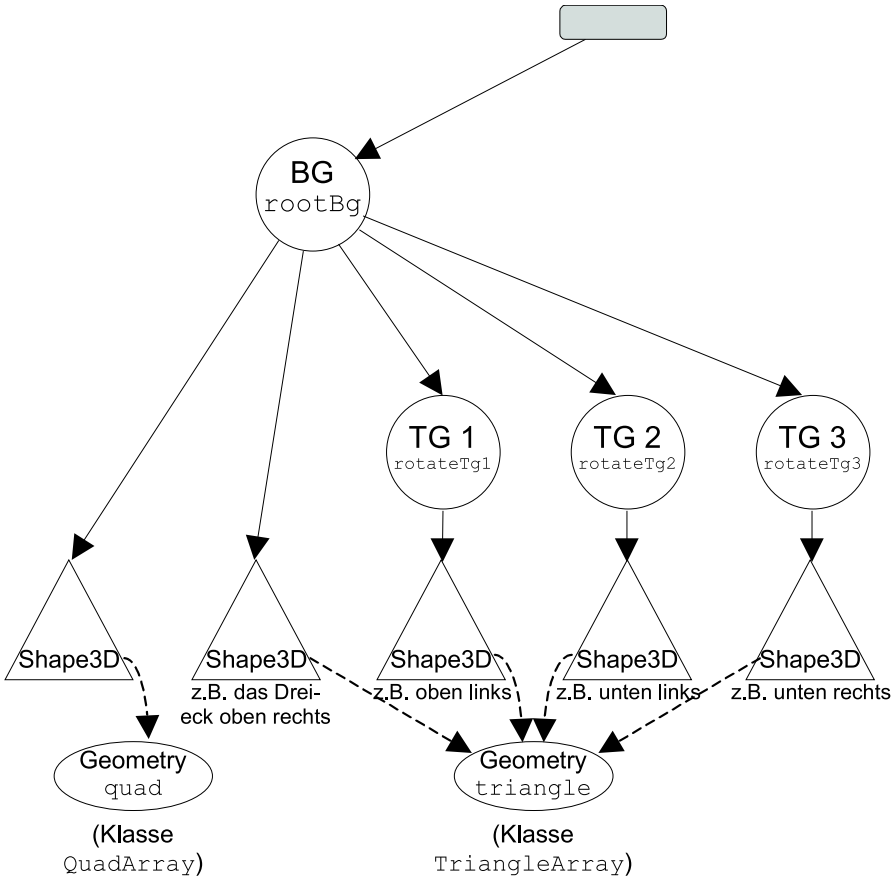


Abbildung 6.43: Ein Szenegraph zu der Szene aus einem Quadrat und vier Dreiecken

TG 1: Drehung um 90 Grad um die z-Achse in mathematisch positiver Richtung (also im Gegenuhrzeigersinn).

TG 2: Drehung um 180 Grad um die z-Achse in mathematisch positiver Richtung.

TG 3: Drehung um 270 Grad um die z-Achse in mathematisch positiver Richtung.

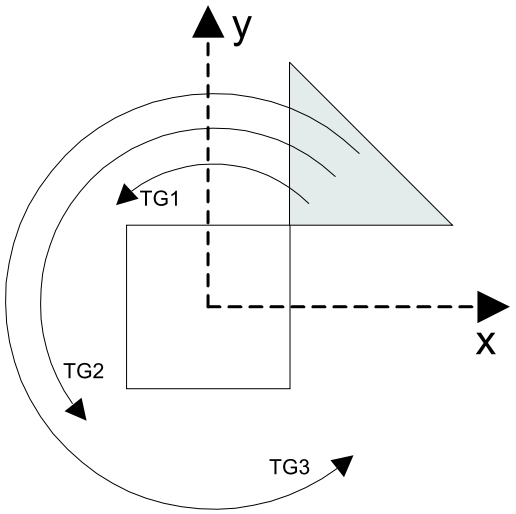


Abbildung 6.44: Die Szene besteht aus einem Quadrat und einem Dreieck. Beide Geometrien liegen in der xy-Ebene. Wir sehen in entgegengesetzter Richtung der z-Achse auf die Geometrien. Die z-Achse ist nicht eingezeichnet. Das Dreieck wird mit drei Transformationen um die z-Achse gedreht. Alle drei zugehörigen TransformGroup-Datenelemente sind Söhne desselben BranchGroup-Datenelements. Deshalb entstehen drei neue Dreiecke.

b) Es folgt nicht nur die Methode `createSceneGraph()`, sondern der gesamte Quellcode des Programms `VierDreiecke.java`:

```
import java.applet.Applet;
import java.awt.BorderLayout;
import java.awt.Frame;

import javax.media.j3d.Canvas3D;
import javax.media.j3d.BranchGroup;
import javax.media.j3d.Transform3D;
import javax.media.j3d.TransformGroup;
import javax.media.j3d.Shape3D;

import javax.media.j3d.TriangleArray;
import javax.media.j3d.QuadArray;

import javax.vecmath.Point3f;

import com.sun.j3d.utils.universe.SimpleUniverse;
import com.sun.j3d.utils.applet.MainFrame;

public class VierDreiecke extends Applet {
  public VierDreiecke() {
    setLayout(new BorderLayout());
    Canvas3D canvas3D = new Canvas3D(
                          SimpleUniverse.getPreferredConfiguration());
```

```
    add(”Center”, canvas3D);

    SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
    simpleU.getViewingPlatform().setNominalViewingTransform();

    BranchGroup scene = createSceneGraph();

    scene.compile();
    simpleU.addBranchGraph(scene);
}

public BranchGroup createSceneGraph() {
    BranchGroup rootBg = new BranchGroup();

    QuadArray quad = new QuadArray(4,
                                   QuadArray.COORDINATES);

    quad.setCoordinate(0, new Point3f( 0.2f, 0.2f, 0.0f ));
    quad.setCoordinate(1, new Point3f( -0.2f, 0.2f, 0.0f ));
    quad.setCoordinate(2, new Point3f( -0.2f, -0.2f, 0.0f ));
    quad.setCoordinate(3, new Point3f( 0.2f, -0.2f, 0.0f ));

    TriangleArray triangle = new TriangleArray(3,
                                   TriangleArray.COORDINATES);

    triangle.setCoordinate(0, new Point3f( 0.2f, 0.2f, 0.0f ));
    triangle.setCoordinate(1, new Point3f( 0.6f, 0.2f, 0.0f ));
    triangle.setCoordinate(2, new Point3f( 0.2f, 0.6f, 0.0f ));

    Transform3D rotate1 = new Transform3D();
    Transform3D rotate2 = new Transform3D();
    Transform3D rotate3 = new Transform3D();

    rotate1.rotZ(Math.PI*0.5d);
    rotate2.rotZ(Math.PI);
    rotate3.rotZ(Math.PI*1.5d);

    TransformGroup rotateTg1 = new TransformGroup(rotate1);
    TransformGroup rotateTg2 = new TransformGroup(rotate2);
    TransformGroup rotateTg3 = new TransformGroup(rotate3);

    rootBg.addChild(new Shape3D(quad));
    rootBg.addChild(new Shape3D(triangle));
    rootBg.addChild(rotateTg1);
    rootBg.addChild(rotateTg2);
    rootBg.addChild(rotateTg3);
```

```
        rotateTg1.addChild(new Shape3D(triangle));
        rotateTg2.addChild(new Shape3D(triangle));
        rotateTg3.addChild(new Shape3D(triangle));

        return rootBg;
    }

    public static void main(String[] args) {
        Frame frame = new MainFrame(new VierDreiecke(), 256, 256);
    }
}
```

Lösung 4:

a) Es ist nicht veröffentlicht, wie Java3D implementiert ist, also wie Java3D intern funktioniert. Deshalb können wir nur vermuten, wie bei Java3D intern vorgegangen wird. Bei Java3D steht weniger die Effizienz der Bilddarstellung im Vordergrund. Außerdem ist Java3D wie auch Java relativ plattformunabhängig. So ist die interne Zerlegung der Daten in Geometrie- und Bilddaten eher unwahrscheinlich, da sie ausschließlich bei bestimmter Hardware die Bilddarstellung beschleunigt.

b) Die vereinfachte Darstellung einer möglichen Rendering-Pipeline von Java3D finden Sie in Abbildung 6.45.

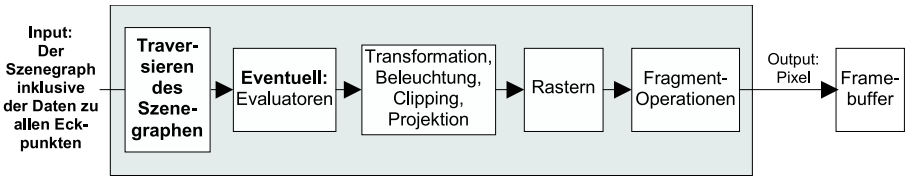


Abbildung 6.45: Die mögliche Rendering-Pipeline von Java3D

c) Gründe für die höhere Effizienz:

- Extensions:
  OpenGL-Treiber nutzen mit Hilfe der Extensions die speziellen Eigenschaften der Grafik-Hardware. Somit werden Aufgaben an optimierte/spezialisierte Grafik-Hardware delegiert.

- Client/Server-Architektur:
  Die Aufgaben können auf den Client und auf einen oder mehrere Server verteilt werden. Bei leistungsstarken Servern kann der Client von allen zeitkritischen und aufwändigen Aufgaben entlastet werden.

- Separates Verarbeiten von Geometrie- und Bilddaten:
  Die verschiedenen Arten von Daten können parallel verarbeitet werden.

- OpenGL-Implementierungen werden in C implementiert.

- Der Zustand wird durch Variablen definiert:
  Der direkte Zugriff auf Variablen ist in der Regel schneller als der Zugriff auf Variablen bzw. Felder von Objekten, denn der Compiler muss bei Objekten zunächst den Speicherort des Objektes (bzw. die Referenz auf das Objekt) ermitteln.

- Immediate Mode:
  Die Code-Zeilen werden so früh wie möglich ausgeführt.

- Darstellungslisten:
  Durch den Einsatz von Darstellungslisten kann die Menge der zum Server gesendeten Daten verringert werden.

Lösung 5:

a) Mit Group-Datenelementen (BranchGroup und TransformGroup)

b) Vorteile:

- Einsparung von Entwicklerzeit durch das Wiederverwenden von Darstellungslisten
- Eine häufig höhere Grafikperformance
  (Optimierung des Inhalts der Darstellungslisten, Verringerung des Datenaustausches zwischen Client und Server, Speichern von Darstellungslisten im Geometrie-Cache, Aufteilung auf mehrere interne Pipelines)

Nachteile:

- Es ist schwierig, Vorkehrungen für die korrekte parallele Bearbeitung mehrerer Darstellungslisten zu treffen.

Literaturverzeichnis

[ARB99] ARB, *OpenGL Reference Manual, dritte Auflage*, ISBN: 0-201-65765-1, Dezember 1999.

[Bou99] Dennis J. Bouvier, *Java 3D API Collateral (Tutorial)*, http://java.sun.com/products/java-media/3D/collateral/, 1999.

[BP99] Kirk Brown, Dan Petersen, *Ready-to-Run Java 3D*, ISBN: 0-471-31702-0, 1999.

[Cla97] Ute Claussen, *Programmieren mit OpenGL*, ISBN: 3-540-57977-X, 1997.

[DIN82] DIN–ISO 7942, *Information Processing – Graphical Kernel System (GKS), Functional Description*, Beuth Verlag GmbH, Berlin, 1982.

[Hil00] Francis S. Hill, *Computer Graphics Using OpenGL*, ISBN: 0-023-54856-8, Mai 2000.

[ISO] ISO 8805, *Graphical Kernel System for Three Dimensions (GKS–3D), Functional Description*.

[ISO82] ISO DIS 2382/13, *Data Processing Vocabulary — Computer Graphics*, 1982.

[ISO89] ISO 9592–(1-3), *Programmer's Hierarchical Interactive Graphics System (PHIGS), Part 1–3*, 1989.

[ISO91] ISO 9592–4, *Programmer's Hierarchical Interactive Graphics System (PHIGS), Part 4: Plus Lumière und Surfaces (PHIGS PLUS)*, 1991.

[Kun94] Michael Kunze, *3D-Grafikstandard OpenGL (Einführung in die 3D-Grafiksprache OpenGL)*, c't, Vol. 8, 1994, 198.

[RSWS99] Jr. Richard S. Wright, Michael Sweet, *OpenGL SuperBible, zweite Auflage*, ISBN: 1-571-69164-2, Dezember 1999.

[SA99] Mark Segal, Kurt Akeley, *The OpenGL Graphics System: A Specification (Version 1.2.1)*, http://www.sgi.com/software/opengl/manual.html, April 1999.

[Sel00] Daniel Selman, *Java 3D Programming*, ISBN: 1-884-77797-X, Juli 2000.

[SGI97] SGI, *The OpenGL Programming Guide (The Official Guide to Learning OpenGL, Version 1.1)*, http://heron.cc.ukans.edu/ebt-bin/nph-dweb/dynaweb/SGI_Developer/OpenGL_PG/, 1997.

[SRD97] Henry Sowizral, Kevin Rushforth , Michael Deering, *The Java 3D API Specification*, ISBN: 0-201-32576-4, 1997.

[SUN00] SUN, *The Java 3D API Specification (Version 1.2)*, http://java.sun.com/products/java-media/3D, April 2000.

[Wil95] Andreas Will, *OpenGL-Programmierung unter Windows 95 und NT*, c't, Vol. 11, 1995, 336ff.

[WND+99] Mason Woo, Jackie Neider, Tom Davis, Dave Shreiner , OpenGL Architectural Review Board, *The OpenGL Programming Guide (The Official Guide to Learning OpenGL Version 1.2), dritte Auflage*, ISBN 0-201-60458-2, August 1999.

Index

Glossar

Bild (image) (6, S.321)
Ein rechteckiges Feld von **Pixeln**. Es ist entweder im Arbeitsspeicher oder im **Bildspeicher** abgelegt. Der Begriff Bild wird im Zusammenhang mit der Bildgenerierung in OpenGL verwendet.

API (6.1, S.321)
Siehe **Application Programming Interface**.

Application Programming Interface (6.1, S.321)
Ausdruck für die Gesamtheit der Schnittstellen einer Reihe von Klassen oder Prozeduren einer Software-Bibliothek. Wird auch **API** genannt.

Projektion (6.1, S.322)
Die Überführung von dreidimensionalen virtuellen Objekten auf die zweidimensionale **Bildebene**. In der Regel werden zusätzlich die Tiefenwerte der Eckpunkte der Objekte gespeichert.

Modellierung (modelling) (6.1, S.323)
„Alle Aktivitäten zum Entwurf einer Szene. Dies umfasst die Festlegung der Geometrie, der Materialien, der Lichtverhältnisse und der Blickverhältnisse. Wird in eingeschränktem Sinn auch oft nur für die geometrische Modellierung verwendet." ([Cla97], S.345)

Anwendung (6.1, S.323)
Anwendungsprogramm, im Gegensatz zu beispielsweise Betriebssystemen und Bibliotheken.

Geometrie (Java3D) (6.2.2, S.326)
Bei Java3D ist Geometrie die Bezeichnung für ein **geometrisches Objekt**. Geometrien können aus den grundlegenden geometrischen Formen (**Standard-Geometrien** von Java3D) und weiteren Elementen aus Bibliotheken, die Java3D ergänzen, zusammengesetzt werden. Solche Elemente können beispielsweise Spiralen, Segmente aus Kreisen und Kugeln sowie komplexere Formen sein.
Im **Szenegraphen** repräsentieren Leaf-Datenelemente Geometrien.
In der Regel werden mit der Klasse **Shape3D** einfache Geometrien implementiert.

Szenegraph (scene graph) (Java3D) (6.2.2, S.326)
Die zentrale Datenstruktur in Java3D. Sie enthält alle Daten zur Beschreibung der Szene und die Daten, die die Sichtweise des virtuellen Beobachters in der Szene spezifizieren.
Der Szenegraph besteht aus genau einem **Sichtgraphen** und in der Regel

mindestens einem **Inhaltsgraphen**.
Der Inhaltsgraph besteht unter anderem aus **BranchGroup**-, **TransformGroup**- und **Leaf**-Datenelementen.
In der vorliegenden Kurseinheit enthält der Szenegraph alle Datenelemente, inklusive des **VirtualUniverse**-Datenelements. In der Literatur ist der Szenegraph jedoch manchmal anders definiert. Dann ist das VirtualUniverse-Datenelement nicht Teil des Szenegraphen, sondern der Vater aller Szenegraphen. In diesem Fall ist die Wurzel jedes Szenegraphen ein **Locale**-Datenelement. In einer Java3D-Anwendung lassen sich demzufolge mehrere Szenen definieren - ein Fall, der eher selten ist.
Bei OpenGL stellt der Open Inventor eine dem Szenegraphen von Java3D ähnliche Datenstruktur zur Verfügung.

Transformation (6.2.2, S.326)
Die mathematische Operation, die auf einen **Eckpunkt** oder eine Menge von Eckpunkten angewendet wird. Der Begriff Transformation steht allgemein für geometrische Transformationen von geometrischen Objekten. Übliche Transformationen sind die Translation, Skalierung, Scherung und die Rotation. Jede dieser Transformationen wird durch eine affinen Abbildung definiert. Die affine Abbildung wird als 4×4-Matrix dargestellt und in einem **TransformGroup**-Datenelement gespeichert. Die entsprechende Transformation wird auf alle geometrische Objekte angewendet, die im Teilgraphen liegen, dessen Wurzel das TransformGroup-Datenelement ist.
„Bei OpenGL der Übergang zwischen zwei Koordinatensystemen." ([Cla97], S. 348)

Bildgenerierung (6.2.3, S.326)
Allgemein: Die „Zusammenfassung einer Klasse von Darstellungsalgorithmen, die auf lokalen Beleuchtungsmodellen und interpolierenden Schattierungsalgorithmen beruhen." [Cla97], S. 346.
Bei Java3D: Der Vorgang beim Überführen des **Szenegraphen** in ein **gerastertes Bild**.

Fenstersystem (window system) (OpenGL) (6.2.4.1, S.326)
Darunter verstehen wir das OpenGL umgebende Betriebs- oder Fenstersystem. Es stellt u.a. **Framebuffer** und z-Buffer bereit, initialisiert und verwaltet den **Zustand** und sorgt für die Bearbeitung von Interaktionen.

Datenelement (Java3D) (6.2.7.3, S.335)
Ein **Szenegraph** besteht aus **Knoten** und **Kanten**. Die Knoten werden bei Java3D durch Instanzen bestimmter Klassen repräsentiert. Da es bei Java3D auch Node-Klassen gibt, verwenden wir den Begriff Datenelement anstelle von Knoten. Auch die Blätter des Szenegraphen bezeichnen wir als Datenelemente.

Leaf-Klasse; Leaf-Datenelement (Java3D), (6.2.7.2, S.334)
Leaf ist eine abstrakte Klasse. Leaf-Datenelemente stehen für die **visuellen**

Objekte und Audio-Objekte einer Szene. In dem vorliegenden Dokument spielen insbesondere die beiden von Leaf abgeleiteten Klassen **Shape3D** und **Behavior** eine Rolle.

Knoten und Kanten (6.2.7.3, S.336) Ein Szenegraph besteht wie jeder Graph aus Knoten und Kanten und enthält in der Regel zusätzlich **Referenzen**.
Die Knoten werden in Java3D als **Datenelemente** bezeichnet.

virtuelles Universum (Java3D) (6.2.7.4, S.336)
Der konzeptionelle Raum, in dem die **visuellen Objekte** existieren. In jeder **Anwendung**, die auf Java3D aufbaut, gibt es ein virtuelles Universum. Das virtuelle Universum wird durch ein **VirtualUniverse**-Datenelement bzw. eine VirtualUniverse-Klasse repräsentiert.

Group-Klasse (Java3D) (6.2.7.4, S.336)
Group ist eine abstrakte Klasse. Zwei Subklassen von Group sind **BranchGroup** und **TransformGroup**. BranchGroup- und TransformGroup-Datenelemente repräsentieren diese Klassen als Knoten im **Szenegraphen**. Die Hauptaufgabe eines Group-Datenelements ist das Gruppieren von anderen Datenelementen (z.B. **Leaf**- und weiteren BranchGroup- oder TransformGroup-Datenelementen), die Söhne des Group-Datenelements sind.
Die Group-Datenelemente halten den Szenegraphen zusammen.

VirtualUniverse-Klasse (Java3D) (6.2.7.4, S.336)
Ein zur VirtualUniverse-Klasse gehörendes Datenelement repräsentiert ein **virtuelles Universum**. Ein VirtualUniverse-Datenelement kann nur **Locale**-Datenelemente als Kinder haben. Somit ist in einer **Anwendung** die Wurzel aller **Szenegraphen** ein VirtualUniverse-Datenelement.

Locale-Klasse (Java3D) (6.2.7.4, S.336)
Ein zur Locale-Klasse gehörendes Datenelement ist ein Punkt im **virtuellen Universum**. Dieser Punkt hat die Aufgabe einer Landmarke: Die Position der **Geometrien** ist relativ zum Locale-Punkt. Der Locale-Punkt liegt für diese Geometrien also im Ursprung. Locale ist für alle Geometrien der Ursprung, die ein Element des Teilbaums sind, dessen Wurzel das Locale-Datenelement ist.

BranchGroup-Klasse (Java3D) (6.2.7.4, S.336)
Wie **TransformGroup** eine Subklasse von **Group**. BranchGroup-Datenelemente dienen ausschließlich der Gruppierung von weiteren **Datenelementen**. Jedes BranchGroup-Datenelement ist - wie jeder innere Knoten eines **Szenegraphen** - die Wurzel eines Teilgraphen des Szenegraphen. Alle Knoten bzw. Datenelemente in einem solchen Teilgraphen werden logisch gruppiert. Der Teilgraph wird **branch graph** genannt, branch von Verzweigung.
Für die Kinder eines BranchGroup-Datenelements gilt das gleiche wie für

TransformGroup-Datenelemente: Ihre Anzahl ist beliebig und alle Kinder sind Instanzen der Subklassen von **Group** oder/und der Klasse **Leaf**. Allerdings dürfen nur BranchGroup-Datenelemente an ein **Locale**-Datenelement angehängt werden.

TransformGroup-Klasse (Java3D) (6.2.7.4, S.336)

Wie **BranchGroup** eine Subklasse von **Group**. Wie BranchGroup-Datenelemente dienen TransformGroup-Datenelemente der Gruppierung weiterer **Datenelemente**. Jedes TransformGroup-Datenelement ist - wie jeder innere Knoten eines **Szenegraphen** - die Wurzel eines Teilgraphen des Szenegraphen. Alle Knoten bzw. Datenelemente in einem solchen Teilgraphen werden logisch gruppiert. Der Teilgraph wird **branch graph** genannt, branch von Verzweigung.
Zusätzlich werden die geometrischen Daten der gruppierten Datenelemente geometrisch transformiert, siehe **Transformation**.
Für die Kinder eines TransformGroup-Datenelements gilt das gleiche wie für BranchGroup-Datenelemente: Ihre Anzahl ist beliebig und alle Kinder sind Instanzen der Subklassen von Group oder/und der Klasse **Leaf**.

gerichteter azyklischer Graph (Java3D) (6.2.7.5, S.338)

Eine Datenstruktur aus Knoten und gerichteten Kanten, in der es keine Zyklen gibt. Ein Pfad, der den Kanten in der jeweiligen Richtung folgt, darf nicht mehrmals zum gleichen Knoten führen. Jeder **Szenegraph** ist (ohne Referenzen) ein gerichteter azyklischer Graph (directed acyclic graph).

Geometry-Klasse (Java3D) (6.2.7.4, S.338)

Eine abstrakte Klasse von Java3D, die die Superklasse von GeometryArray und damit der meisten **Standard-Geometrien** ist.

DAG (Java3D) (6.2.7.5, S.338)

directed acyclic graph, siehe **gerichteter azyklischer Graph**.

Referenz (6.2.7.6, S.339)

Ein Szenegraph besteht aus **Knoten** (**Datenelementen**) und **Kanten** und enthält in der Regel zusätzlich Referenzen. Eine Referenz ist keine Kante und wird als gestrichelter Pfeil dargestellt. Eine Referenz kann nur von einem Leaf- zu einem NodeComponent-Datenelement führen. Zu einem NodeComponent führt *mindestens* eine Referenz.

Pfad im Szenegraphen (Java3D) (6.2.7.7, S.339)

Pfad vom **Locale**-Datenelement oder von einem internen Knoten (ohne Blätter) im **Szenegraphen** zu einem Blatt des Szenegraphen.

Eckpunkt (vertex) (4, S.340)

Die Punkte, mit denen **geometrische Objekte** definiert werden. Beispielsweise der Anfangs- oder Endpunkt einer Strecke oder die Punkte in den

„Ecken” des Randes eines Polygons.

Inhalt (content) (Java3D), (6.2.7.8, S.341)
Die Gesamtheit der **visuellen Objekte** (inklusive der Eigenschaften der Objekte und der Lichtquellen) und Audio-Objekte eines **virtuellen Universums**.

branch graph (Java3D) (6.2.7.8, S.341)
Ein Teilbaum eines **Szenegraphen**, dessen Wurzel ein **BranchGroup**-Datenelement ist.

Inhaltsgraph (content branch graph) (Java3D) (6.2.7.8, S.341)
Der Teil des **Szenegraphen**, der die **visuellen Objekte** (inklusive der Eigenschaften der Objekte und der Lichtquellen) enthält. Die Wurzel eines Inhaltsgraphen ist ein **Locale**-Datenelement. Ein Szenegraph kann mehrere Inhaltsgraphen enthalten.

Sichtgraph (view branch graph) (Java3D) (6.2.7.8, S.341)
Der Teil des **Szenegraphen**, in dem die Parameter bezüglich der Sichtweise festgelegt sind. Zwei der Parameter sind der Augpunkt und die Blickrichtung des virtuellen Betrachters in der Szene.

Bildebene (image plate) (6.2.8.2, S.346)
Das imaginäre Rechteck im **virtuellen Universum**, auf das die Szene projiziert wird.

Pixel (von picture element) (6.2.10.1, S.350)
Ein einzelnes Bildelement des Rasterausgabegeräts (Displays). Ein Pixel besteht aus je einem x- und y-Wert und drei Werten für die Farbe (bei **Echtfarbdarstellung**) bzw. einen **Farbtabellenindex**-Wert. Jedes Pixel ist einer x- und y-Koordinate zugeordnet. In OpenGL und insbesondere Java3D arbeiten Anwendungsprogramme typischerweise nicht mit Pixeln, sondern mit dreidimensionalen geometrischen Elementen, die schließlich **gerastert** werden. Das Ergebnis sind **Fragmente**. Fragmente können von den Anwendungsprogrammen mit Hilfe von OpenGL oder Java3D manipuliert werden. Anschließend werden die Fragmente in Pixel überführt, die dem **Framebuffer** übergeben werden.

Shape3D-Klasse (Java3D) (6.2.10.1, S.351)
Mit Shape3D wird die Form einer **Geometrie** festgelegt, z.B. Würfel, Zylinder oder Kugel.

Standard-Geometrie (Java3D) (6.2.10.2, S.355)
Eine bestimmte Gruppe von **geometrischen Objekten** mit grundlegender

Form. In Java3D stehen nur 10 Standard-Geometrien zur Verfügung: Punkte, Strecken/Linien, Dreiecke, Vierecke, Streckenzüge/Linienzüge, Streifen aus Dreiecken und Fächer aus Dreiecken. Siehe auch **Geometrie**.

geometrisches Objekt (6.2.10.2, S.356)

(Virtuelles) **visuelles Objekt** mit geometrischer Form. Beispiele für geometrische Objekte sind Linien, Polygone und Kugeln. Geometrische Objekte sind neben dem Hintergrund und Effekten wie Nebel die einzigen sichtbaren Elemente der Szene.
Bei Java3D wird ein geometrisches Objekt mit **Geometrie** bezeichnet, bei bei OpenGL ist damit ein **Primitiv** oder eine Menge von Primitiven gemeint.

Streifen aus Dreiecken (6.2.10.2, S.356)

Bei Streifen aus Dreiecken wird im Vergleich zu Linienzügen zusätzlich der dritte Eckpunkt automatisch mit dem zwei Eckpunkte davor liegenden Eckpunkt verbunden, also mit dem ersten. Der erste, zweite und dritte Eckpunkt bilden ein Dreieck. Anschließend wird der vierte Eckpunkt mit dem zweiten verbunden. Der zweite, dritte und vierte Eckpunkt bilden ein zweites Dreieck. Analog wird mit allen weiteren Eckpunkten verfahren.

Fächer aus Dreiecken (6.2.10.2, S.357)

Bei Fächern aus Dreiecken wird im Vergleich zu Linienzügen zusätzlich der dritte Eckpunkt automatisch mit dem ersten Eckpunkt verbunden. Wie auch beim **Streifen aus Dreiecken** bilden der erste, zweite und dritte Eckpunkt ein Dreieck. Anschließend wird der vierte Eckpunkt ebenfalls mit dem ersten Eckpunkt verbunden. Der erste, dritte und vierte Eckpunkt bilden ein zweites Dreieck. Analog werden alle weiteren Eckpunkte mit dem ersten Eckpunkt verbunden.

Animation (Java3D) (6.2.11.1, S.360)

Bei Java3D die Veränderung des **virtuellen Universums** (inklusive der Szene), ohne dass Aktionen (bzw. Eingabedaten) des Benutzers einen Einfluss auf den Start und den Ablauf der Animation haben. In der Regel werden die Veränderungen durch den Ablauf bestimmter Zeitintervalle initiiert.
Es wird zwischen **Interaktionen** und **Animationen** unterschieden.
Im allgemeinen Sprachgebrauch ist eine Animation die „Erstellung von Bildserien, die bei hinreichend schneller Bildabfolge den Eindruck von kontinuierlicher Bewegung hervorrufen" ([Cla97], S. 337).

Interaktion (Java3D) (6.2.11.1, S.360)

Veränderung des **virtuellen Universums** (inklusive der **visuellen Objekte** der Szene) als Reaktion auf Eingabedaten des Benutzers oder in einem Kommando-Antwort-Wechselspiel. Es wird zwischen Interaktionen und **Animationen** unterschieden.

behavior (Java3D) (6.2.11.2, S.360)
Behavior (Verhalten) ist die Änderung einiger Eigenschaften von **visuellen Objekten** zur Laufzeit. Eigenschaften können beispielsweise die Position, Orientierung, Größe oder Farbe oder Kombinationen davon sein. Ein 'behavior' kann entweder eine **Animation** oder eine **Interaktion** sein.
Siehe auch **Behavior-Klasse**.

Behavior-Klasse (Java3D) (6.2.11.2, S.360)
Sie legt fest, wie sich bestimmte Parameter von **Datenelementen** im **Szenegraphen** zur Laufzeit ändern. Wird beispielsweise die durch ein **TransformGroup**-Datenelement definierte Matrix modifiziert, ändert sich die Position, Orientierung oder Größe. Behavior-Klassen werden bei der Erstellung von **Animationen** sowie von interaktiven Programmen (siehe **Interaktion)** eingesetzt.
Siehe auch **behavior**.

Capabilities (Java3D) (6.2.11.1, S.360)
Eine spezielle Art von Bit-Variablen. Bei Java3D wird der Zugriff auf die Parameter eines Datenelementes im **Szenegraphen** mit Capabilities kontrolliert. Beispielsweise kann der Wert eines Parameters eines **TransformGroup**-Datenelements nicht verändert werden, bis die zu diesem Parameter zugehörige Capability gesetzt wird.

visuelles Objekt (6.2.11.2, S.361)
Geometrische Objekte, je nach Situation sind zusätzlich auch virtuelle Lichtquellen, der Hintergrund oder andere Elemente gemeint. Die Gesamtheit der visuellen Objekte samt ihren Eigenschaften ergibt den **Inhalt** (content) eines **virtuellen Universums**. Der Ausdruck „visuelles Objekt" statt „Objekt" wird verwendet, um Verwechslungen mit Objekten im objektorientierten Sinn auszuschließen.

scheduling bound (Java3D) (6.2.11.4, S.363)
Wird eine 'Bound' (Grenze) im Zusammenhang mit **Interaktionen**, **Animationen** oder bestimmten anderen Effekten benutzt, spricht man von einer scheduling bound.

Bounds-Klasse; bounding volume (Java3D) (6.2.11.4, S.364)
Mit der abstrakten Klasse Bounds (Grenzen) definiert der Entwickler die Grenzen eines dreidimensionalen konvexen und abgeschlossenen Raums (bounding volume, Grenzbereich). Häufig ist der Raum kugel- oder quaderförmig. Dieser Raum ist mit einem **behavior**, mit Licht, Sound oder anderen Leistungsmerkmalen von Java3D verknüpft. Bounds können unter anderem mit einer **BoundingSphere** oder **BoundingBox** definiert werden.
Siehe auch **scheduling bound**.

BoundingSphere-Klasse (Java3D) (6.2.11.4, S.364)
Instanzen von BoundingSphere definieren einen kugelförmigen Grenzbe-

reich. Siehe auch **Bounds-Klasse**.

BoundingBox-Klasse (Java3D) (6.2.11.4, S.364)
Instanzen von BoundingBox definieren einen quaderförmigen Grenzbereich, dessen Kanten parallel zu den drei Achsen des Koordinatensystems sind. Siehe auch **Bounds-Klasse**.

Material-Klasse (Java3D) (6.2.14.1, S.371)
Damit werden bestimmte Eigenschaften von Oberflächen von **geometrischen Objekten** definiert. Eine dieser Eigenschaften ist die Art, wie ambientes, diffuses, punktförmiges und paralleles Licht reflektiert wird und ob die Oberfläche Licht emitiert (z.B. wie bei einer glühenden Herdplatte).

Framebuffer (6.3.3.1, S.377)
Siehe **Bildspeicher**.

Bildspeicher (Framebuffer/frame buffer) (6.3.3.1, S.377)
Ein „Speicherbereich, der die Farbinformation computergenerierter **Bilder** aufnimmt". Im Bildspeicher sind für jedes einzelne **Pixel** des Bildes die zugehörigen Farbinformationen abgelegt. Der Bildspeicher enthält zu jedem Pixel entweder die Farbinformation in **Echtfarbdarstellung** oder den **Farbtabellenindex**. Der Bildspeicher „existiert im Allgemeinen auch physikalisch im Ausgabegerät"([Cla97], S. 339).

Pixelmap (pixelmap) (OpenGL) (6.3.3.1, S.377)
Ein rechteckiges Feld von **Pixeln**. Eines der 12 **Primitiven** von OpenGL.

Farbtabellenindex (color index) (6.3.3.1, S.377)
Eine Farbe kann entweder in **Echtfarbdarstellung** oder durch einen Farbtabellenindex festgelegt werden. Ein Farbtabellenindex ist ein ganzzahliger Wert. Die durch den Farbtabellenindex beschriebenen Farben werden mit Hilfe einer **Farbtabelle** in die vom Ausgabegerät darstellbaren Farben umgesetzt.

Echtfarbdarstellung (true color) (6.3.3.1, S.377)
Eine Farbe kann entweder in Echtfarbdarstellung oder durch einen **Farbtabellenindex** festgelegt werden. Bei der Echtfarbdarstellung erfolgt „eine direkte Umsetzung von Farbkomponenten in darstellbare Werte des Ausgabegerätes. Die Berechnung der Farben erfolgt anhand ihrer Komponenten" ([Cla97], S. 341), z.B. je ein Wert für die Farbkomponenten rot, grün und blau.

Bitmap (bitmap) (OpenGL) (6.3.3.1, S.377)
Ein rechteckiges Feld von Bits, das als [visuelles] **Primitiv** oder zum Maskieren (siehe **Maske**) von Bildausschnitten des **Bildspeicher** dient.

Maske (mask) (OpenGL) (6.3.3.1, S.377)
„Zeichenmuster, das zur Auswahl oder zum Ausblenden von Teilen eines **Bildes** dient" ([Cla97], S. 344).
Häufig bestehen Masken aus rechteckigen **Bitmaps**. Eine solche Bitmap wird quasi über einen bestimmten Bereich des **gerasterten** Bildes gelegt. So können beispielsweise die Werte der Bitmap und die Farbwerte oder **Farbindizes** des Bildes mit Operationen verknüpft werden, dazu ein einfaches Beispiel: Alle Bildpunkte (**Fragmente** oder **Pixel)** mit einem Farbtabellenindex, bei dem das höchste Bit 0 ist, wird der Farbtabellenindex für schwarz zugewiesen.

Zustand (state) (OpenGL) (6.3.3.1, S.377)
„Der Zustand der OpenGL-„Maschine" bzw. der OpenGL-**Zustandsmaschine** steuert die Abarbeitung aller OpenGL-Befehle. Ein Teil des Zustands ist implementierungsabhängig und nicht durch die **Anwendung** beeinflussbar." ([Cla97], S. 348) Der Zustand von OpenGL besteht aus den Werten von mehr als 250 Variablen. Diese Zustandsvariablen werden auch Attribute genannt.

state (OpenGL) (6.3.3.1, S.377)
Siehe **Zustand**.

Farbtabelle (color look-up table) (6.3.3.1, S.378)
Tabelle, die die **Farbtabellenindizes** in darstellbare Farben des Ausgabegerätes umsetzt: Die Farbtabelle ordnet jedem Farbtabellenindex je einen Wert für jede Farbkomponente (z.B. rot. grün und blau) zu.

Evaluator (evaluator) (OpenGL) (6.3.3.2, S.378)
„In OpenGL werden mit Hilfe von Evaluatoren Koordinaten bzw. Parameter von Flächen auf der Basis von B*é*zierkurven oder -flächen berechnet" ([Cla97], S. 341).

Zustandsvariable (OpenGL) (6.3.3.2, S.378)
Siehe **Zustand**.

Rastern; Rasterung; Rasterisierung (6.3.3.2, S.379)
Die Überführung der (in der Regel auf die Bildebene projizierten) **virtuellen Objekte** in **Pixel** oder **Fragmente**. Fragmente sind die Vorstufe von Pixeln. In der Regel werden im z-Buffer die Tiefenwerte der Fragmente abgelegt. „In OpenGL wird die Rasterisierung auf alle grafischen **Primitive**, also auch auf **Bitmaps** und **Pixelmaps**, angewendet." ([Cla97], S. 345)

Fragment (fragment) (OpenGL) (6.3.3.2, S.379)
Ein Fragment ist die Vorstufe eines **Pixels** im Rahmen der **Rendering**-Pipeline. Ein Fragment besteht aus den Informationen des zukünfigen Pixels (x- und y-Wert und der Farbe in **Echtfarbdarstellung** bzw. dem **Farb-**

tabellenindex, Tiefenwert, Transparentwert und den Texturkoordinaten.)

rendern; Rendering (6.3.3.2, S.380)
Allgemein: Die „Zusammenfassung einer Klasse von Darstellungsalgorithmen, die auf lokalen Beleuchtungsmodellen und interpolierenden Schattierungsalgorithmen beruhen.” [Cla97], S. 346.
Bei Java3D: Der Vorgang beim Überführen des **Szenegraphen** in ein ge**rastertes Bild**.

Darstellungsliste (display list) (OpenGL) (6.3.3.2, S.380)
„Eine Zusammenfassung von grafischen Befehlen zu einer Einheit mit eigenem Namen. In OpenGL werden diese Listen auf dem Server abgelegt, so dass sie bei Aufruf im Allgemeinen effizienter zu bearbeiten sind und die Kommunikation zwischen Client und Server reduziert wird” ([Cla97], S. 340). Siehe auch **Client-Server-Architektur**.

Client-Server-Architektur (OpenGL) (6.3.4.5, S.383)
„Ein Modell der Betrachtung von Hard- und Softwarearchitekturen, in dem die eine Seite, der „Client”, Dienstleistungen nutzt, die die andere Seite, der „Server”, anbietet. Diese Form der Modellierung wird hauptsächlich im Kontext von Netzen und in der objektorientierten Programmierung angewendet.
OpenGL ist als ein „Server” konzipiert, der auf einem anderen Rechner ausgeführt werden kann als das ihn benutzende Programm.”([Cla97], S. 340)

Primitiv (OpenGL) (6.3.4.8, S.385)
Bei OpenGL ist Primitiv (das grafische/geometrische/visuelle Primitiv) die Bezeichnung für die einfachsten **geometrischen Objekte**. In OpenGL stehen 12 Primitive zur Verfügung: Punkte, Strecken, Streckenzüge, Dreiecke, Vierecke, Streifen aus Dreiecken, Streifen aus Vierecken, Fächer aus Dreiecken, geschlossene Streckenzüge, konvexe Polygone sowie **Bitmaps** und **Pixelmaps**.

Zustandsmaschine (state machine) (OpenGL) (6.3.4.10, S.390)
„Eine Modell der Informationsverarbeitung” [Cla97], S. 349. Eine Soft- oder/und Hardware, die intern eine oder mehrere Informationen speichert. Die Informationen werden als (interner) **Zustand** des Systems bezeichnet. Die Soft-/Hardware nimmt Eingaben entgegen, interpretiert diese auf Basis des internen Zustands, verändert eventuell abhängig von dem Ergebnis der Interpretation den eigenen Zustand und erzeugt eventuell Ausgaben. Setzt eine **Anwendung** auf OpenGL auf, kann die Gesamtheit aller OpenGL-Routinen und -Variablen inklusive des Zustandes als eine Zustandsmaschine betrachtet werden.

Status (OpenGL) (6.3.4.10, S.390)
Siehe **Zustand**.

Immediate Mode (6.3.4.11, S.390)

Bei OpenGL werden die Szenen im Immediate Mode („direkten Modus”) dargestellt: Jede Codezeile wird so bald wie möglich ausgeführt. Wenn beispielsweise im Quellcode ein Primitiv definiert wird, können bei der Ausführung des compilierten Programms die ersten Befehle der Definition schon ausgeführt werden, selbst wenn das Ende der Definition noch nicht erreicht ist. Der Immediate Mode ist insbesondere für interaktive Anwendungen geeignet, die ohne spürbare Verzögerung die veränderte Szene darstellen sollen. Der Immediate Mode bietet keine Möglichkeit, einem Primitiv einen Namen oder eine Identifikationsnummer (ID) zuzuordnen, um es auch nach der Definition erneut aufrufen zu können. Um dieses Problem zu umgehen, kann man so genannte **Darstellungslisten** einsetzen.

display list (OpenGL) (6.3.4.12, S.391)

Siehe **Darstellungsliste**.

Lehrziele

Nach dem Durcharbeiten des ersten Teils dieser Kurseinheit sollten Sie verstehen,

- welche prinzipiellen Unterschiede zwischen Kurven- und Flächenmodellierung einerseits und Körpermodellierung andererseits bestehen,
- was ein Repräsentationsschema ist und welche Bedeutung die Charakterisierungen *vollständig* und *eindeutig* haben,
- worin die Vor- und Nachteile der einzelnen Repräsentationen bestehen,
- welche Datenstrukturen in den verschiedenen Repräsentationen verwendet werden,
- worin die Schwierigkeiten beim Übergang von einer Repräsentation in die andere bestehen,
- warum das interaktive Entwerfen von komplexen Formen mit Körpermodellierern nur sehr eingeschränkt möglich ist.

Der zweite Teil dieser Kurseinheit vermittelt die physikalischen Grundlagen zur Berechnung realitätsnaher Bilder und gibt eine Einführung in die Farbtheorie. Zum Abschluß werden lokale (empirische) Beleuchtungsmodelle behandelt.

Nach dem Studium dieser Lehreinheit sollten Sie

- den Unterschied zwischen Radiometrie und Photometrie kennen und die wichtigsten strahlungsphysikalischen und photometrischen Größen zur Beschreibung und Berechnung der Lichtausbreitung in einer Szene angeben können,
- den Strahlungsaustausch zwischen Oberflächen beschreiben können,
- angeben können,
  - wie die Farbe eines Bildpunktes auf einem Monitor festgelegt wird,
  - wie die Farbe vom Auge wahrgenommen und bewertet wird,
  - welcher Zusammenhang zwischen Strahlungsspektrum und Farbe besteht,
- Systeme zur Farbdefinition, ihre Anwendbarkeit, die Unterschiede zwischen verschiedenen Systemen beschreiben können,
- lokale Beleuchtungsmodelle, Beleuchtungsalgorithmen und Beleuchtungsverfahren erklären können,
- insbesondere Gouraud- und Phong-Shading genau beschreiben können.

Nach den einfachen empirischen Beleuchtungsmodellen werden Sie in diesem dritten Teil der Kurseinheit mit zwei Verfahren vertraut gemacht, die möglichst genau die physikalischen Gesetzmäßigkeiten realisieren sollen, dem Raytracing-Verfahren und dem Radiosity-Verfahren. Allerdings sind auch hier zur Begrenzung des Rechenaufwandes vereinfachende Annahmen notwendig.
Diese Lehreinheit sollte Ihnen folgendes Wissen vermitteln:

- Was sind die wesentlichen Ideen beim Raytracing-Verfahren?
  Wie hoch ist der Aufwand des Verfahrens?
  Welche Möglichkeiten zur Beschleunigung des Raytracing-Verfahrens gibt es?
- Wie funktionieren die folgenden Algorithmen zur Reduzierung der Gesamtanzahl der Schnittpunkttests
  - Raumunterteilung mit regulären Gittern,
  - Raumunterteilung mit Octrees, insbesondere der SMART-Algorithmus,
  - Szeneunterteilung mit hierarchischen Bäumen,
  - Mailboxtechnik,
  - Schatten-Caches,
  - adaptive Rekursionstiefenkontrolle,
  - Pixel-Selected Raytracing?
- Was sind die wesentlichen Ideen beim Radiosity-Verfahren?
- Was versteht man unter einem Formfaktor?
  Welche Bedeutung hat er für das Radiosity-Verfahren?
- Wie lautet die Radiosity-Gleichung, und wie löst man sie im Fall einer 'Full-Matrix-Lösung'?
  Welche alternative Methode existiert hierzu, und was sind die Unterschiede zum ursprünglichen Verfahren?
- Wie kann man in Theorie und Praxis Formfaktoren berechnen, d.h. wie funktioniert
  - die numerische Integration,
  - Nusselts Analogon,
  - das Hemicube-Verfahren,
  - Formfaktorberechnung mit Raytracing?
- Wie hängt der Rechenaufwand beim Raytracing- und beim Radiosity-Verfahren von der Zahl der Objekte in der Szene und der Bildschirmauflösung ab?

Kapitel 7

Einführung in Körpermodelle und Beleuchtungsrechnung

Körper

In diesem Abschnitt stellen wir Ihnen einige der wesentlichen Konzepte der Festkörpermodellierung (Solid Modelling) vor. Der interessierte Leser sei auf den Kurs 1693 'Graphische Datenverarbeitung II', Kapitel 1 'Körper', hingewiesen.
Beim Solid Modelling geht man von realen Objekten und Körpern aus. Für jedes reale Objekt wird ein *Körpermodell* erzeugt, das geschlossene dreidimensionale Körper eindeutig und vollständig beschreibt. Dieses Modell beinhaltet Informationen über die Gestalt und die Geschlossenheit des dreidimensionalen Körpers sowie über seine geometrische Verbundenheit, d.h. die Anzahl seiner zusammenhängenden Komponenten. Darüber hinaus garantiert das Modell die Integrität und Konsistenz der repräsentierten Daten und bildet die Basis für alle nachfolgenden geometrischen und anderen Operationen der Anwendung.

7.1 Repräsentationsschemata

7.1.1 Modellbildung

In einem zweistufigen Prozeß bildet man zunächst das *mathematische Modell*, dann das *symbolische Modell*.
Das mathematische Modell ist eine Idealisierung des realen Objektes und beinhaltet in der Regel nur einen Teil der Eigenschaften, die das reale Objekt auszeichnen. Es wird in ein *symbolisches Modell* im sogenannten *Repräsentationsraum* überführt.
Der *Repräsentationsraum* ist abhängig vom *Modellierer* und umfaßt die Menge derjenigen Objekte, die mit dem Modellierer konstruiert werden können.

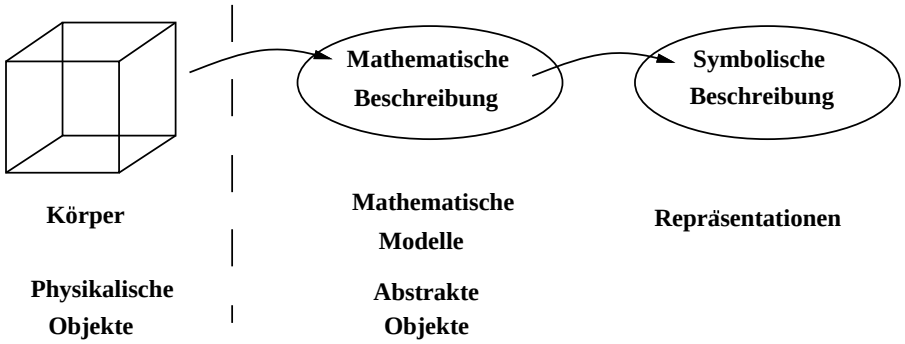


Abbildung 7.1: Die Modellbildung als zweistufiger Prozeß

7.1.2 Übersicht bekannter Repräsentationsschemata

Die bekannten Repräsentationsschemata für Körpermodelle lassen sich in drei Hauptgruppen einteilen:

- *Drahtmodelle:*
  Die Strukturelemente des Drahtmodells beschränken sich im allgemeinen auf die Konturelemente ‘gerade Kante’, ‘Kreisbogen’ oder auch ‘Spline’. Zwischen diesen Elementen bestehen innerhalb eines Drahtmodells keine Beziehungen; eine Zuordnung zu Flächen ist nicht definiert.
  Beurteilung:
  Die Drahtmodelle sind einfach und traditionell.
  Zur Repräsentation von Körpermodellen sind sie jedoch nicht geeignet, da die Darstellung als Drahtmodell unvollständig und mehrdeutig ist.

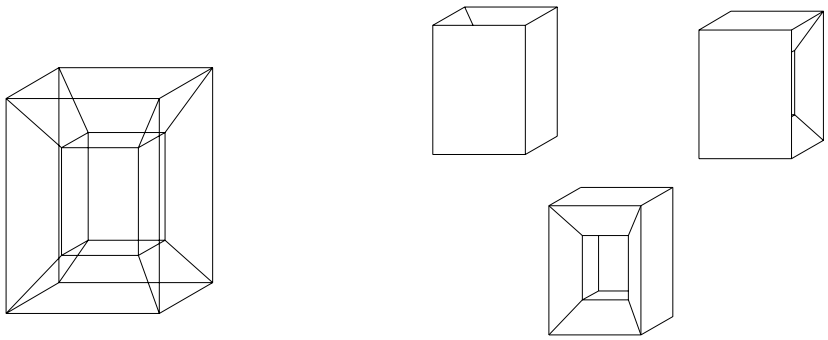


Abbildung 7.2: Mehrdeutigkeit bei Drahtmodellen

- *Flächenmodelle:*
  Der Hauptinformationsgehalt der Flächenmodellierer liegt in den einzelnen

Flächenbeschreibungen (vgl. Kurseinheit 4 'Kurven und Flächen'). Die Flächen stehen bei einem reinen Flächenmodellierer in keinem gegenseitigen Zusammenhang; insbesondere sind keine Nachbarschaftsbeziehungen zwischen ihnen abgespeichert.
Beurteilung:
Im Flächenmodell läßt sich nicht entscheiden, ob ein Punkt innerhalb oder außerhalb des dargestellten Objektes liegt.

- *Körpermodelle:*
  Hinter dem Begriff der Körpermodelle verbergen sich zahlreiche Repräsentationsschemata. Die wichtigsten sind:

  – Boundary Representation (BRep)
  – Constructive Solid Geometry (CSG)
  – Zellmodell
  – Hybridmodell (CSG-BRep)

  Beurteilung:
  Die Körpermodelle bilden eine vollständige Beschreibung eines dreidimensionalen Objektes.
  Durch die verwendeten Algorithmen zur Manipulation von Objekten kann deren Konsistenz gesichert werden.
  Sie können automatisch von Programmen interpretiert werden.

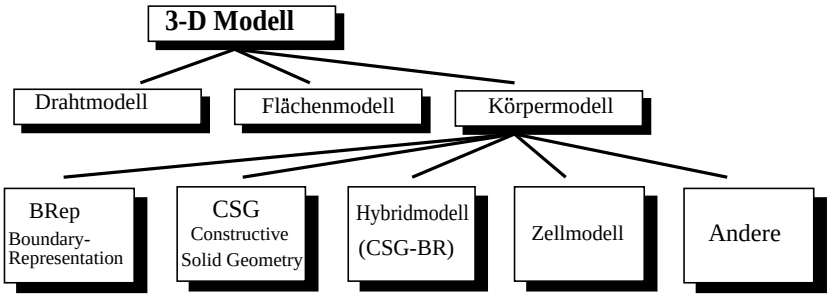


Abbildung 7.3: Die wichtigsten Repräsentationsschemata für 3D-Modelle

7.2 Allgemeine Erzeugungstechniken und Manipulationen von Körpermodellen

Unabhängig davon, welches spezielle Körpermodell betrachtet wird, müssen Konzepte zur Erzeugung, Manipulation und zur Speicherung von Objekten vorhanden sein. Wir betrachten zunächst die folgenden Grundkörperdefinitionen:

- *Grundkörper fester Form - Solid Shapes:*
  Jedes Primitivum wird aus einer vom System vorgegebenen Menge von einfachen geometrischen Körpern fester Form ausgewählt. Durch das Festlegen gewisser Dimensionsparameter wird der gewünschte Grundkörper dann eindeutig beschrieben.
  Für jedes Objekt wird ein Bezugspunkt festgelegt, der als Ursprung eines lokalen Koordinatensystems dient. Dieses lokale Koordinatensystem kann vom Anwender nicht verändert werden, da es a priori für jeden Grundkörper festgelegt ist.

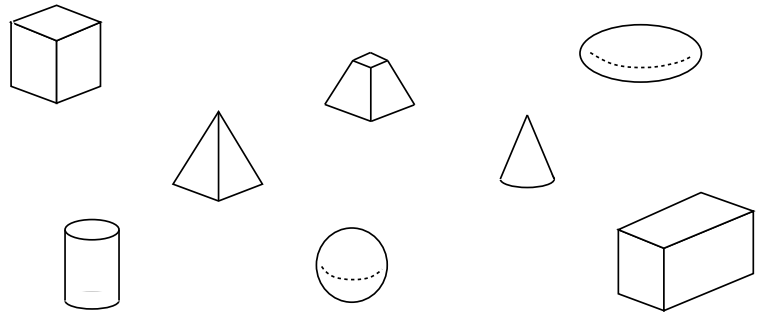


Abbildung 7.4: Grundkörper fester Form

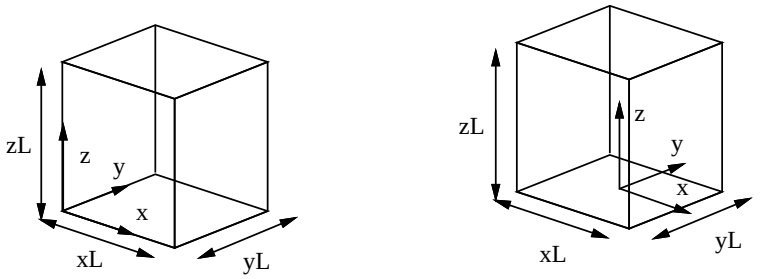


Abbildung 7.5: Grundkörper fester Form:
Dimensionsparameter und die Lage des Bezugspunktes bestimmen ein Primitivum eindeutig.

- *Sweeping ebener Konturzüge und ganzer Körper:*
  Beim *Sweeping* wird entweder eine Kontur oder ein ganzer Körper entlang einer Raumkurve bewegt. Die Kontur oder der Körper und die Raumkurve werden durch Parameter definiert.
  Man unterscheidet translatorisches Sweeping, rotatorisches Sweeping und das Sweeping vollständiger Körper.

- *Halbräume:*
  Ist eine reellwertige analytische Funktion $f : \mathbb{R}^3 \to \mathbb{R}$ gegeben, so bezeichnet man die Mengen

$$F_1 = \{x \in \mathbb{R}^3 : f(x) \geq 0\}$$

  und

$$F_2 = \{x \in \mathbb{R}^3 : f(x) \leq 0\}$$

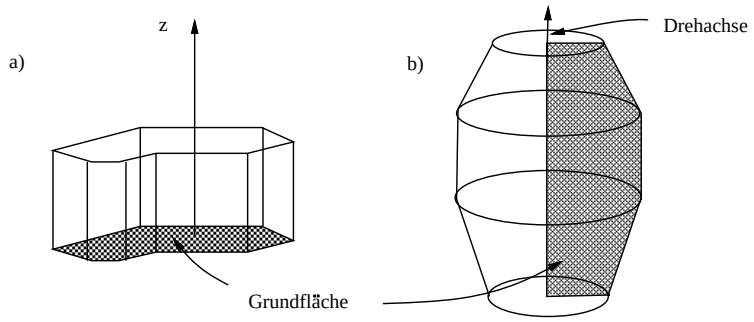


Abbildung 7.6: Sweeping eines ebenen Konturzuges: a) Translation, b) Rotation

als die durch f definierten *Halbräume*.
Die Menge $\{x : f(x) = 0\}$ trennt diese beiden Halbräume. Die Vereinigung der beiden Halbräume ergibt wieder den $\mathbb{R}^3$.
Die Einschränkung auf analytische Funktionen schließt unerwünschte Objekte aus.
Werden als Funktionen nur Polynome zugelassen, so nennt man die Halbräume *algebraisch.*

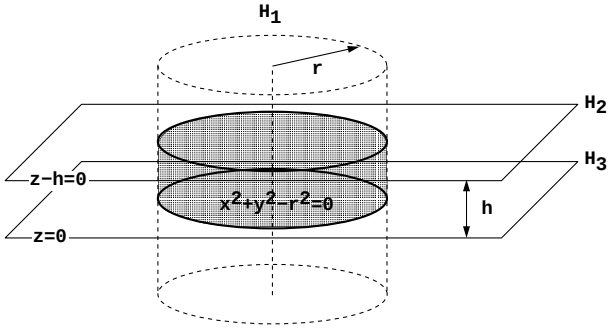


Abbildung 7.7: Konstruktion eines Zylinders aus Halbräumen

Im allgemeinen werden bei Körpermodellierern, ausgehend von einfachen Objekten (den Grundkörpern oder *Primitiva*), schrittweise kompliziertere Objekte durch Anwendung Boolescher Mengenoperationen konstruiert.
Gegeben: Objekte mit einem gemeinsamen Koordinatensystem.

Nach der Festlegung der Primitiva werden mit Hilfe von *regularisierten Boole'schen Mengenoperationen*, der *regularisierten Vereinigung* $\cup^*$, des *regularisierten Durchschnittes* $\cap^*$ und der *regularisierten Differenz* $-^*$, neue Körper erzeugt. Sie unterscheiden sich von den bekannten Boole'schen Mengenoperationen dadurch, daß das Ergebnis regularisiert wird. Beim Regularisieren werden die entstandenen 0D-, 1D- und 2D-Anteile eliminiert.

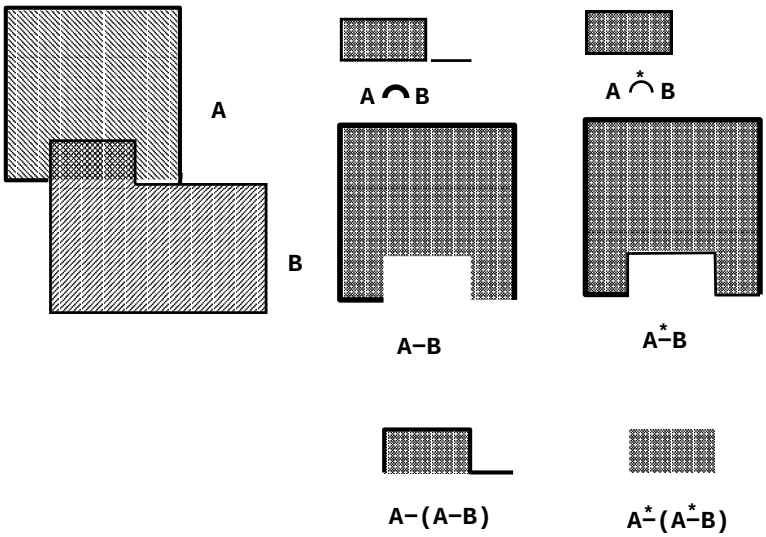


Abbildung 7.8: Eine regularisierte Mengenoperation im $\mathbb{R}^2$
Die entstehenden 1D-Anteile werden entfernt.

7.3 Randrepräsentationen (Boundary Representation)

Ein Körpermodell kann eindeutig durch seine Oberfläche und eine zugehörige topologische Orientierung beschrieben werden. Wegen der topologischen Orientierung ist für jeden Punkt der Oberfläche eindeutig festgelegt, auf welcher Seite das Innere des Objektes liegt.
Boundary Representations (BReps) benutzen diese Tatsache und beschreiben 3D-Objekte durch ihre Oberfläche.

Dabei können einfache Körper wie Würfel, Quader, Kugeln durch ihre Knoten, Kanten und Facetten beschrieben werden, wobei Facetten Quadrate, Rechtecke, Dreiecke, Oberflächen von Halbkugeln etc. sind (vgl. Abb. 7.9, 7.10):

v : Anzahl der Knoten (*vertices*)

e : Anzahl der Kanten (*edges*)

f : Anzahl der Facetten (*faces*)

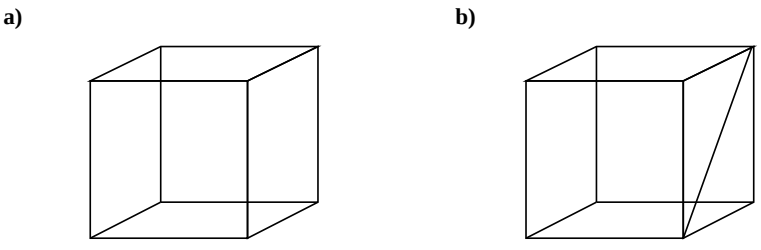


Abbildung 7.9: a) Würfel mit 6 Facetten, 8 Knoten und 12 Kanten
b) Wird eine zusätzliche Kante eingefügt, so besitzt der Würfel 7 Facetten, 8 Knoten und 13 Kanten.

Jeder Facette kann man dadurch eine Orientierung zuschreiben, daß auf ihrer Berandung ein Umlaufsinn definiert wird. Ein derartiger Umlaufsinn auf der Beran-

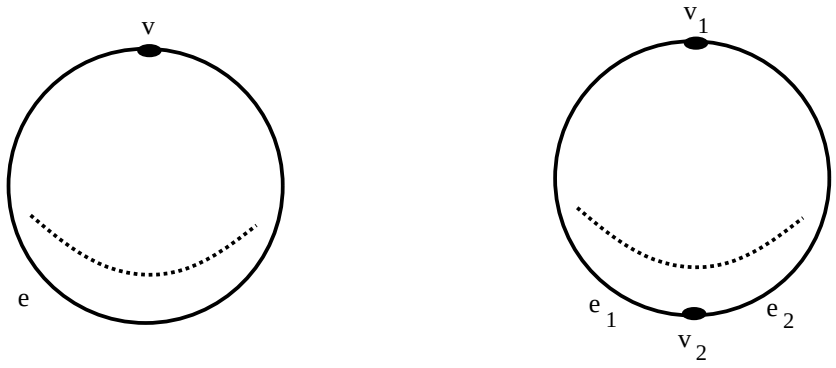


Abbildung 7.10: Knoten und Kanten auf der Kugeloberfläche:
Links ein Knoten, eine Kante und zwei Facetten: Die beiden Facetten sind die Oberflächen der beiden durch die Kante *e* getrennten Halbkugeln.
Rechts zwei Knoten, zwei Kanten und dieselben beiden Facetten.
Im Falle der Kugel bestehen die Kanten nicht aus Geraden, sondern aus Kreissegmenten.

dung wird durch einen Pfeil auf einer ihrer Kanten gekennzeichnet. Diesen Pfeil übertragen wir von einer Kante auf die ihr in Pfeilrichtung benachbarte Kante, indem wir auf dieser Kante einen Pfeil anbringen, der von dem gemeinsamen Knoten beider Kanten zum nächsten Knoten weist. Auf diese Weise versorgen wir, ausgehend von einem Pfeil auf einer Kante der Facette, sämtliche Kanten dieser Facette mit einem Pfeil. Man erkennt leicht, daß man nach einem einmaligen Umlauf der Facette auf der Ausgangskante einen Pfeil erhält, der dieselbe Richtung wie der Ausgangspfeil hat. Folglich läßt jede Facette genau zwei Orientierungen zu, die wir als zueinander entgegengesetzt bezeichnen.

Für Boundary-Modelle können wir die folgenden Datenstrukturen verwenden:

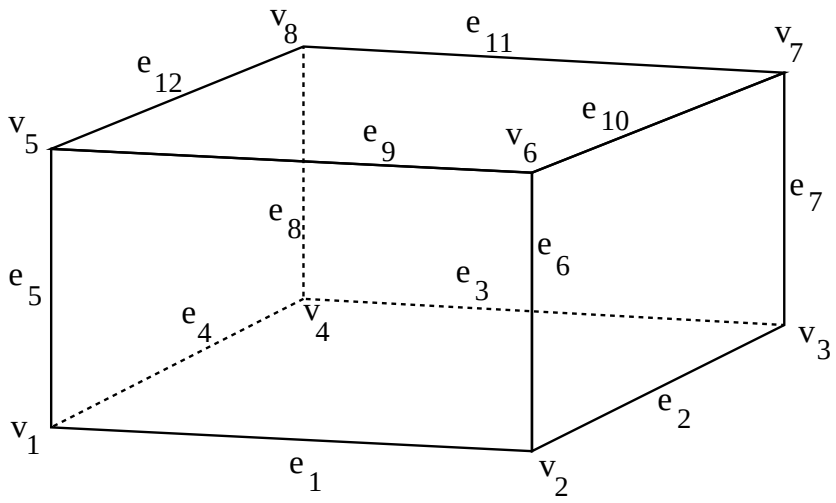


Abbildung 7.11: Quader mit gekennzeichneten Knoten, Kanten und Flächen

- *Polygonorientierte Datenstrukturen:*
  Der Körper besteht aus einer Sammlung von Facetten, welche in einer Ta-

belle mit Bezeichnern für jede Facette zusammengefaßt werden. Die Facetten werden als Polygone dargestellt, wobei jedes Polygon aus einer Folge von Koordinatentripeln besteht, siehe Tabelle 7.1.

Facette	Koordinaten			
f_1	(x_1,y_1,z_1)	(x_2,y_2,z_2)	(x_3,y_3,z_3)	(x_4,y_4,z_4)
f_2	(x_6,y_6,z_6)	(x_2,y_2,z_2)	(x_1,y_1,z_1)	(x_5,y_5,z_5)
f_3	(x_7,y_7,z_7)	(x_3,y_3,z_3)	(x_2,y_2,z_2)	(x_6,y_6,z_6)
f_4	(x_8,y_8,z_8)	(x_4,y_4,z_4)	(x_3,y_3,z_3)	(x_7,y_7,z_7)
f_5	(x_5,y_5,z_5)	(x_1,y_1,z_1)	(x_4,y_4,z_4)	(x_8,y_8,z_8)
f_6	(x_6,y_6,z_6)	(x_5,y_5,z_5)	(x_8,y_8,z_8)	(x_7,y_7,z_7)

Tabelle 7.1: Polygonorientierte Datenstruktur eines Quaders (vgl. Bild 7.11)

Wenn Sie die PDF-Version des Kurses bearbeiten, können Sie jetzt das Interaktionselement 'modeler_face' anwählen.

- *Knotenorientierte Datenstrukturen:*
  Knoten werden unabhängig gespeichert und dann den einzelnen Facetten zugeordnet. Tabelle 7.2 zeigt eine Darstellung des Quaders in einer knotenorientierten Datenstruktur.

Knoten	Koordinaten			Facette	Knoten			
v_1	x_1	y_1	z_1	f_1	v_1	v_2	v_3	v_4
v_2	x_2	y_2	z_2	f_2	v_6	v_2	v_1	v_5
v_3	x_3	y_3	z_3	f_3	v_7	v_3	v_2	v_6
v_4	x_4	y_4	z_4	f_4	v_8	v_4	v_3	v_7
v_5	x_5	y_5	z_5	f_5	v_5	v_1	v_4	v_8
v_6	x_6	y_6	z_6	f_6	v_6	v_5	v_8	v_7
v_7	x_7	y_7	z_7					
v_8	x_8	y_8	z_8					

Tabelle 7.2: Knotenorientierte Datenstruktur eines Quaders (vgl. Bild 7.11)

Wenn Sie die PDF-Version des Kurses bearbeiten, können Sie jetzt das Interaktionselement 'modeler' anwählen.

- *Kantenorientierte Datenstrukturen:*
  Ein kantenorientiertes Boundary-Modell stellt eine Facette als Zyklus von Kanten dar. Die Knoten einer Facette sind implizit durch die Kanten dargestellt, siehe Tabelle 7.3

Bestimmte Berechnungen sind bei einfachen Darstellungen teuer: So muß man die Kantenliste aller Facetten durchsuchen, um festzustellen, welche

Kante	Knoten		Knoten	Koordinaten			Facetten	Kanten			
e_1	v_1	v_2	v_1	x_1	y_1	z_1	f_1	e_1	e_2	e_3	e_4
e_2	v_2	v_3	v_2	x_2	y_2	z_2	f_2	e_9	e_6	e_1	e_5
e_3	v_3	v_4	v_3	x_3	y_3	z_3	f_3	e_{10}	e_7	e_2	e_6
e_4	v_4	v_1	v_4	x_4	y_4	z_4	f_4	e_{11}	e_8	e_3	e_7
e_5	v_1	v_5	v_5	x_5	y_5	z_5	f_5	e_{12}	e_5	e_4	e_8
e_6	v_2	v_6	v_6	x_6	y_6	z_6	f_6	e_{12}	e_{11}	e_{10}	e_9
e_7	v_3	v_7	v_7	x_7	y_7	z_7					
e_8	v_4	v_8	v_8	x_8	y_8	z_8					
e_9	v_5	v_6									
e_{10}	v_6	v_7									
e_{11}	v_7	v_8									
e_{12}	v_8	v_5									

Tabelle 7.3: Kantenorientierte Datenstruktur eines Quaders (vgl. Bild 7.11)

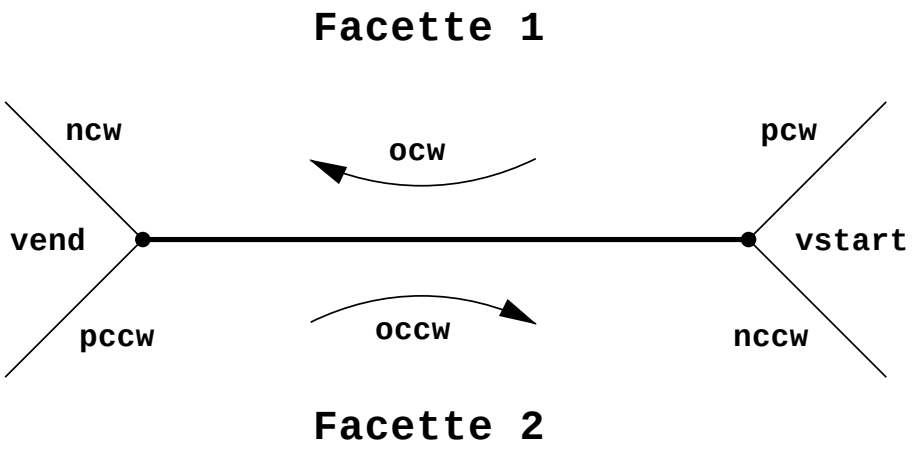


Abbildung 7.12: Die Winged-Edge-Datenstruktur:
Benachbarte Knoten, Kanten und Flächen werden im Uhrzeigersinn (cw) und im Gegenuhrzeigersinn (ccw) gespeichert. Eine Kante wird im Uhrzeigersinn vom Startknoten (vstart) zum Endknoten (vend) durchlaufen.
o=orientation, n=next, p=previous

Facetten eine gemeinsame Kante haben. Deshalb wurden komplexere Randrepräsentationen erfunden, um solche Berechnungskosten zu senken.
Das wichtigste Beispiel einer kantenorientierten Datenstruktur ist die *Winged-Edge-Datenstruktur*, [Sam90], siehe Tabelle 7.4.
Zusätzlich zu den in einer einfachen kantenorientierten Datenstruktur gespeicherten Beziehungen werden noch Nachbarschaftsbeziehungen zwischen den einzelnen Kanten explizit gespeichert, vgl. Abbildung 7.12.

Wenn Sie die PDF-Version des Kurses bearbeiten, können Sie jetzt das Interaktionselement 'modeler_edge' anwählen.

Wenn Sie die PDF-Version des Kurses bearbeiten, können Sie jetzt das Interaktionselement 'modeler_winged_edge' anwählen.

Kante	Knoten, Nachbarkanten				Knoten	Koordinaten			Facetten	Kanten	
e_1	v_1	v_2	e_2	e_5	v_1	x_1	y_1	z_1	f_1	e_1	$+$
e_2	v_2	v_3	e_7	e_1	v_2	x_2	y_2	z_2	f_2	e_5	$+$
e_3	v_3	v_4	e_4	e_7	v_3	x_3	y_3	z_3	f_3	e_2	$+$
e_4	v_4	v_1	e_1	e_8	v_4	x_4	y_4	z_4	f_4	e_8	$-$
e_5	v_1	v_5	e_9	e_4	v_5	x_5	y_5	z_5	f_5	e_8	$+$
e_6	v_2	v_6	e_{10}	e_2	v_6	x_6	y_6	z_6	f_6	e_{12}	$+$
e_7	v_3	v_7	e_{11}	e_2	v_7	x_7	y_7	z_7			
e_8	v_4	v_8	e_{12}	e_3	v_8	x_8	y_8	z_8			
e_9	v_5	v_6	e_{10}	e_5							
e_{10}	v_6	v_7	e_{11}	e_6							
e_{11}	v_7	v_8	e_{12}	e_{10}							
e_{12}	v_8	v_5	e_9	e_8							

Tabelle 7.4: (Vereinfachte) Winged-Edge-Datenstruktur eines Quaders (vgl. Bild 7.11) Dabei gibt das Orientierungsbit + oder - an, wie die Kanten der Facette durchlaufen werden: + entspricht dem Gegenuhrzeigersinn, - dem Uhrzeigersinn.

Eine gute Übersicht über weitere Datenstrukturen, deren Speicherplatzbedarf und deren Abfragekomplexität findet man in [NB94].

7.4 CSG

Bei der **C**onstructive **S**olid **G**eometry (CSG) werden Körper durch Bäume Boole'scher Operatoren und Primitiva, sogenannte CSG-Bäume, beschrieben. Die einzelnen Primitiva können dabei durch ein Boundary-Modell oder durch ein Halbraummodell definiert sein.

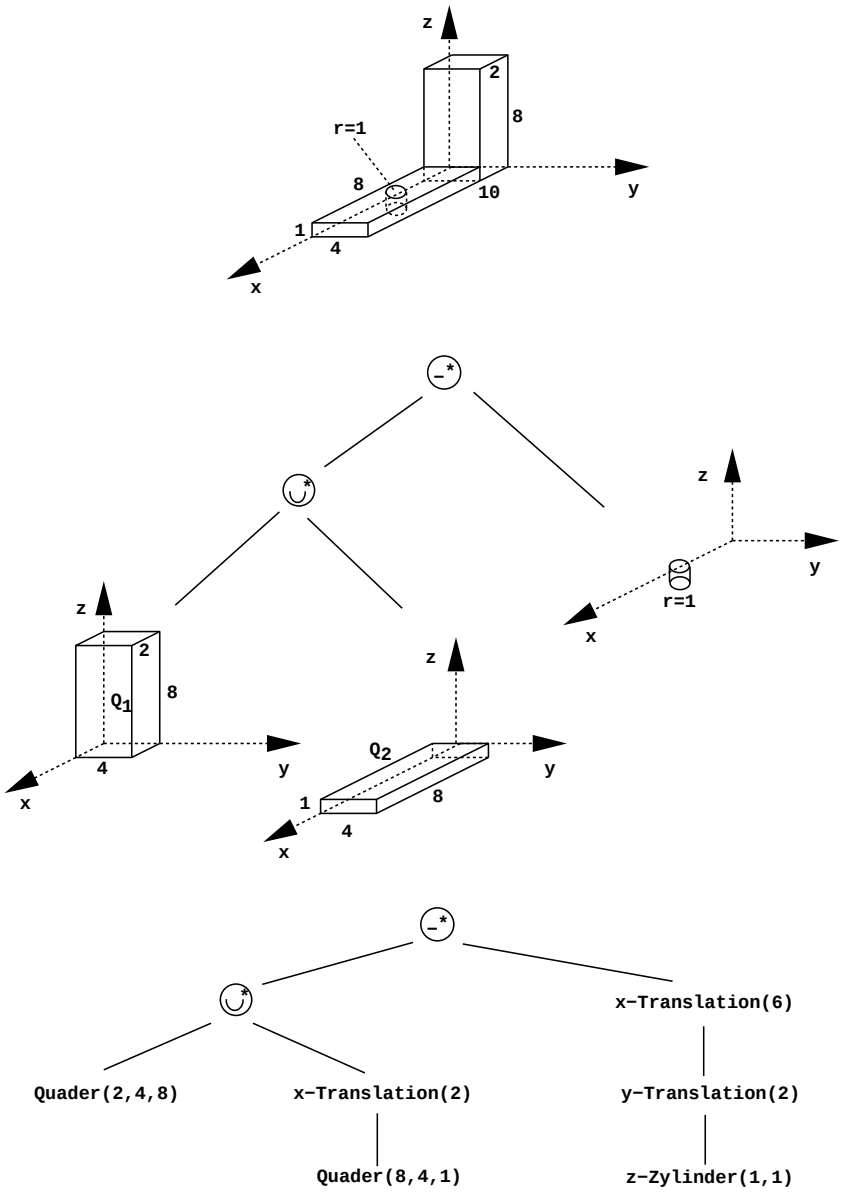


Abbildung 7.13: Darstellung eines Objekts als CSG-Baum

Die Darstellung von CSG-Objekten als Bäume ist eng mit der Konstruktion der Objekte verknüpft.
Durch einen CSG-Baum wird, ausgehend von vorhandenen Primitiva, die Vorgehensweise bei der Konstruktion eines Objektes beschrieben.

Die Knoten des Baumes repräsentieren regularisierte Boole'sche Mengenoperationen bzw. Transformationen im Raum, welche die Lage eines Primitivums im dreidimensionalen Raum definieren. Die Blätter des Baumes verweisen auf Primitiva. Die Konstruktionsmethode eines Objektes in CSG ist nicht eindeutig.

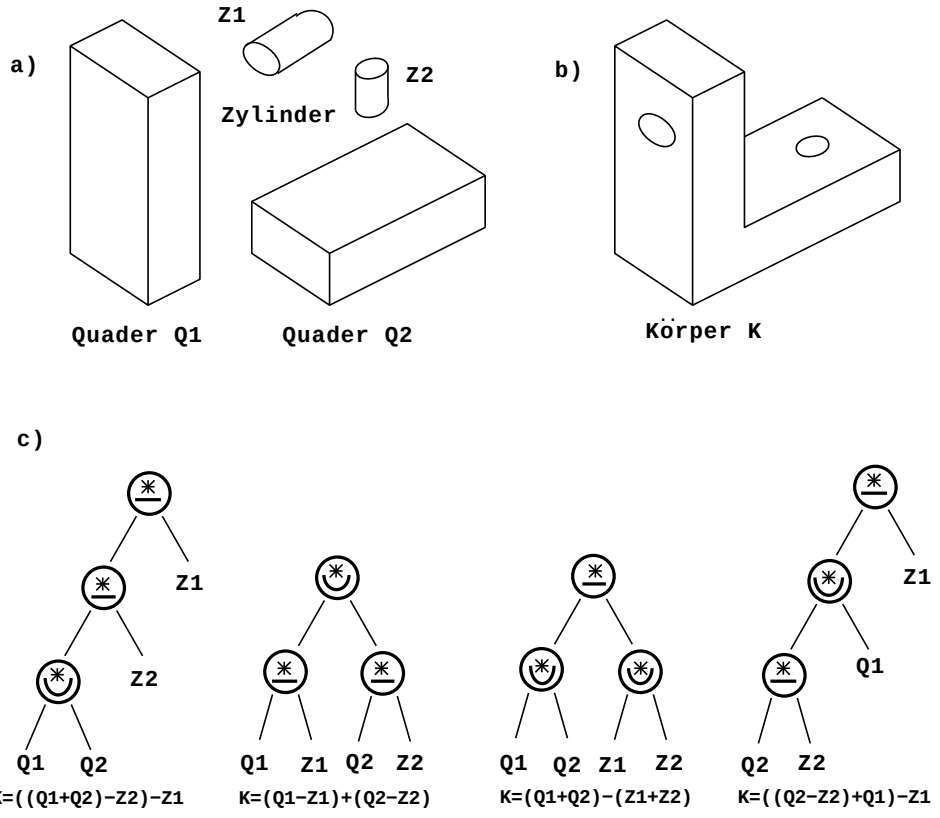


Abbildung 7.14: Darstellung eines Objektes durch vier verschiedene CSG-Bäume. Die Boole'schen Ausdrücke sind äquivalent.

Wenn Sie die PDF-Version des Kurses bearbeiten, können Sie jetzt das Interaktionselement 'csg' anwählen.

7.5 Zellmodelle

In Zellmodellen wird ein Körper in eine Menge sich nicht überlappender, benachbarter Zellen zerlegt. Diese können verschiedene Formen, Größen, Lagen und Orientierungen besitzen.
Man unterscheidet die folgenden Modelle:

- **Zellzerlegungen**
- **Räumliche Aufzählungsmethoden**
- **adaptive Darstellungen (Quadtrees, Octrees, Binary Space Partitioning Trees)**

Eine **Zellzerlegung** basiert auf einer bestimmten Anzahl von Zelltypen und einem einfachen Operator zum Zusammensetzen dieser Zellen. Die einzelnen Zellen können beliebige Objekte sein, die keine Löcher besitzen. Die Oberflächen der Zellen können dabei auch gekrümmte Flächen sein.
Ein Körper wird aus einer Menge halbdisjunkter Zellen zusammengesetzt, d.h. die verschiedenen Zellen haben keine inneren Punkte gemeinsam.
Nachteile dieser Darstellung:

- Die Darstellung selbst muß nicht eindeutig sein.
- Die Überprüfung einer Darstellung auf ihre Richtigkeit ist aufwendig, da jedes Paar von Zellen einer Zerlegung auf mögliche nicht erlaubte Überschneidungen hin überprüft werden muß.

Anwendungsgebiet: finite Elementmethoden

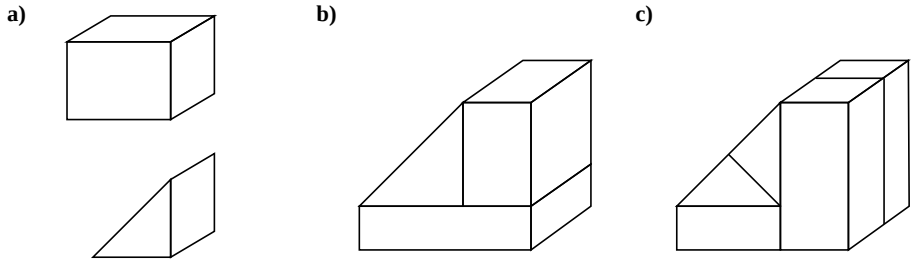


Abbildung 7.15: Die Darstellung eines Objektes bei einer Zellzerlegung ist nicht eindeutig. Um dasselbe Objekt zu beschreiben, können mehrere Zellformen verwendet werden.

Bei **räumlichen Aufzählungsmethoden** wird der $\mathbb{R}^3$ in ein festes Raster von kleinen Würfeln eingeteilt. Diese Elementarwürfel bezeichnet man als Voxel.
Wegen der Regelmäßigkeit des Rasters genügt es, für jedes Voxel die Koordinaten eines Knotens zu speichern.
Um einen Körper in einem solchen Raster darzustellen, werden alle Voxel des Rasters, die ganz oder teilweise durch den Körper ausgefüllt werden, gekennzeichnet. Je kleiner die Voxel gewählt sind, desto genauer kann ein Körper dadurch approximiert werden.

Vorteile:

- Körper können eindeutig dargestellt werden.
- Algorithmen zum Generieren von neuen Körpern aus schon bestehenden sind relativ einfach.

Nachteile:

- Großer Speicherbedarf
- Es können nur diejenigen Objekte exakt beschrieben werden, deren Oberflächen parallel zu den Oberflächen der Voxel verlaufen.

Bei einem **Quadtree** wird die Ebene in Quadranten eingeteilt. Eine gegebene Flä-

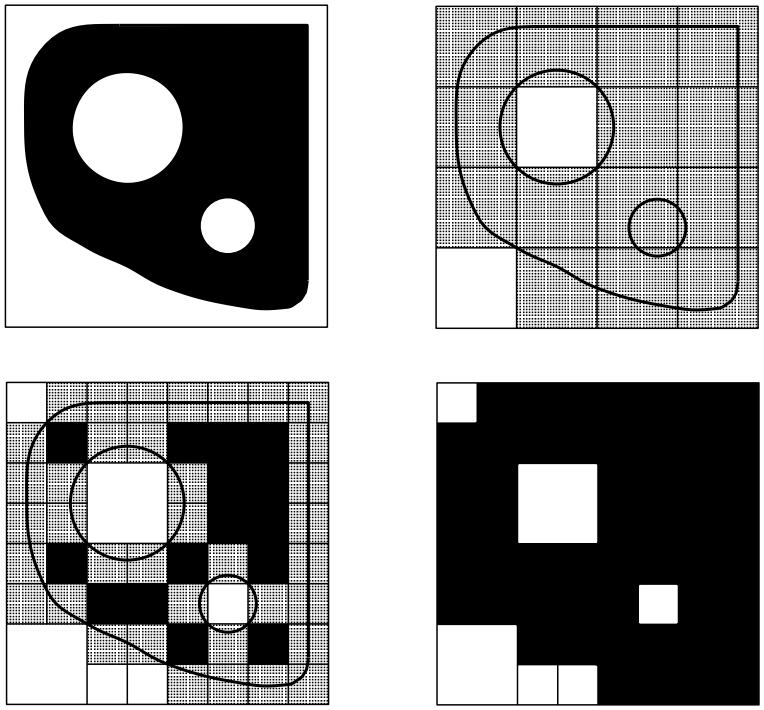
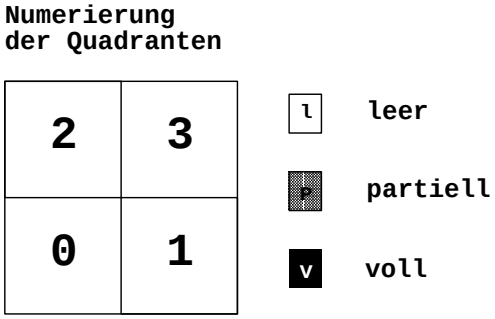


Abbildung 7.16: Quadtree-Darstellung einer einfachen Fläche. Weitere Unterteilung erhöht die Genauigkeit der Darstellung. In der tiefsten Rekursionsstufe werden alle partiell überdeckten Quadranten schwarz eingefärbt.

che überdeckt jeden dieser Quadranten ganz, teilweise oder überhaupt nicht. Mittels einer Klassifikation werden die Quadranten je nach Grad ihrer Überdeckung mit schwarz, grau oder weiß gekennzeichnet. Ein teilweise überdeckter Quadrant wird in vier Unterquadranten aufgeteilt, die wieder je nach Grad ihrer Überdeckung gekennzeichnet werden. Diese Operation wird rekursiv solange fortgesetzt, bis alle Quadranten entweder mit schwarz oder weiß gekennzeichnet sind oder eine vorgegebene Rekursionstiefe erreicht wird. Im dreidimensionalen Fall werden diese Operationen analog für acht Oktanden durchgeführt.
Die sukzessive Unterteilung kann durch einen Baum mit teilweise überdeckten inneren Knoten und vollen oder leeren Quadranten als Blätter dargestellt werden.

Um den Speicherbedarf zu verringern, werden für Octrees und Quadtrees auch lineare Datenstrukturen eingesetzt.
In der linearen Quadtree- und Octree-Darstellung besteht der lineare Baum somit aus einer sortierten Liste von Pfadadressen zu allen schwarzen Knoten, d.h. zu allen vollständig überdeckten Oktanden.



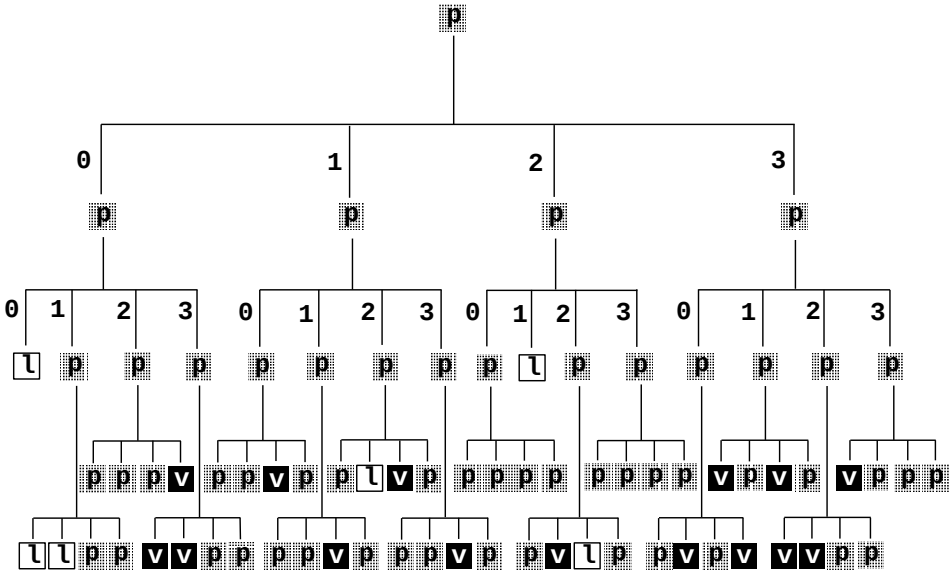


Abbildung 7.17: Quadtree-Datenstruktur der Fläche in Bild 1.34: Die schwarzen Knoten und Blätter repräsentieren vollständig überdeckte, die grauen teilweise überdeckte und die weißen leere Quadranten.

Wenn Sie die PDF-Version des Kurses bearbeiten, können Sie jetzt das Interaktionselement 'octree' anwählen.

Binary Space Partitioning Trees (BSP Trees)

Als Alternative zu einer Octree-Darstellung ist auch eine binäre Raumaufteilung möglich. Dabei werden die Baum-Knoten nicht in acht Oktanden, sondern in zwei Hälften aufgeteilt. Diese Aufteilungen können z.B. sukzessive in x-, y- und z-Richtung durchgeführt werden. Im allgemeinen Fall entstehen die Teilbäume durch eine Trennung mit Hilfe einer Ebene mit beliebiger Richtung, Orientierung und Lage. Zu jedem Knoten gehören dabei die Gleichung einer Fläche und zwei Zeiger auf die beiden, durch die Ebene definierten Halbräume (H_1 und H_2). Durch

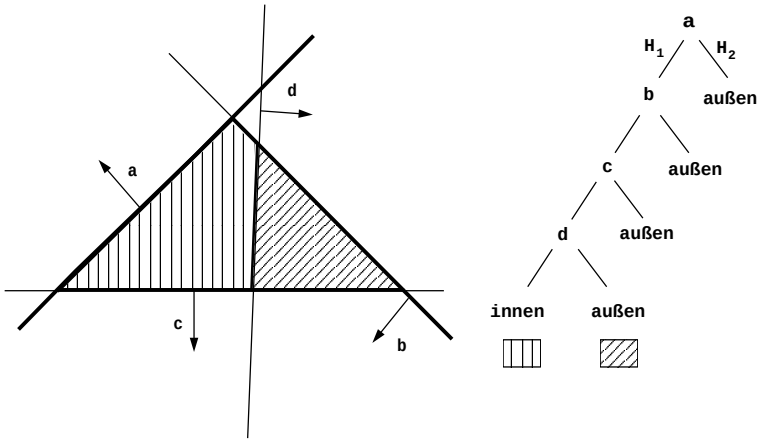


Abbildung 7.18: Konstruktion eines einfachen BSP-Baumes

die Wahl des Vorzeichens der Ebenengleichung wird ein Normalenvektor festgelegt, der in der Regel auf das Äußere des Objektes (H_2) zeigt.

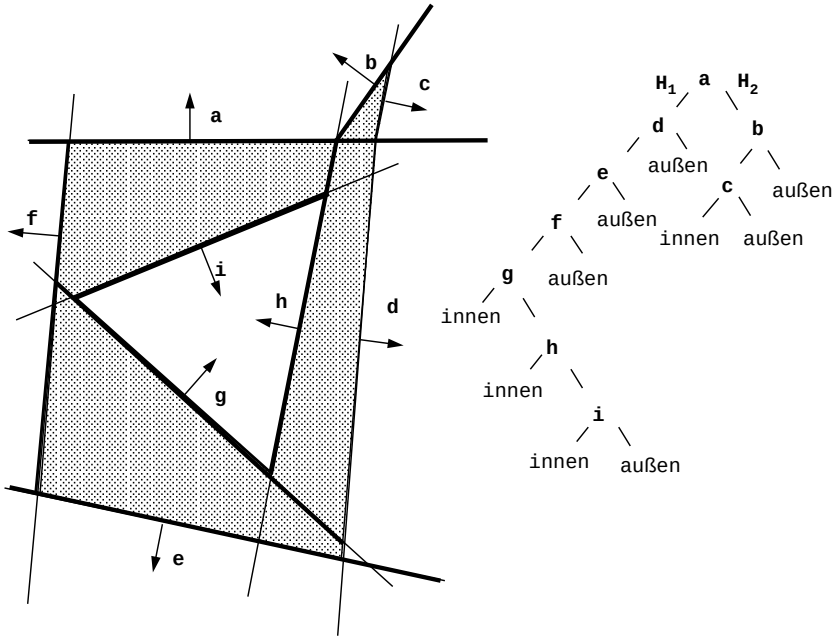


Abbildung 7.19: Polygon und dazugehörender BSP-Baum

Wenn Sie die PDF-Version des Kurses bearbeiten, können Sie jetzt das Interaktionselement 'bsptrees' anwählen.

7.6 Hybridmodelle

Keiner der drei wichtigsten Zugänge zu Solid Modelling (**Boundary-Darstellung, CSG-Darstellung** oder **Räumliche Zerlegungsmethoden**) ist für alle Anwendungen gleich gut geeignet. Daher werden bei der Implementierung oft mehrere Modelle gleichzeitig verwendet. Dabei muß auf die **Konsistenz** der Daten in den verschiedenen Repräsentationen geachtet werden. Um von einer Repräsentation in eine andere zu wechseln, sind entsprechende Konvertierungsalgorithmen notwendig.

Die CSG-Darstellung ist z.B. für den Anwender am besten als Repräsentationsart geeignet, da Modellierungsvorschriften mittels Boole'scher Operationen direkt angegeben werden können. Zur schnellen Visualisierung sind hingegen BReps besser geeignet, da das Neuauswerten der gesamten CSG-Bäume bei einer Neudarstellung nach affinen Transformationen entfällt. Ein typisches Beispiel eines Hybridmodellierers ist daher ein CSG-BRep-Modellierer. Die CSG-Datenstruktur wird dabei als Modelldatenstruktur verwendet.

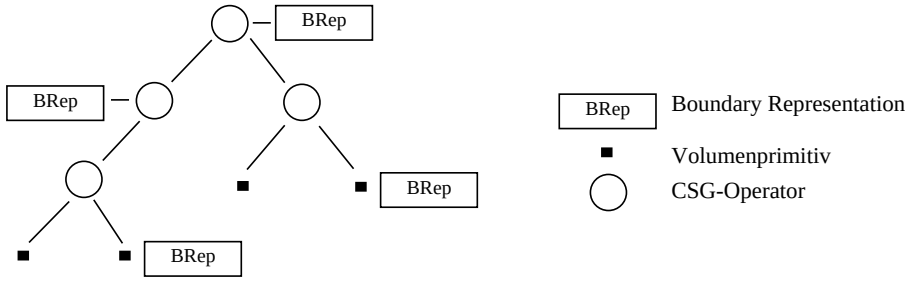


Abbildung 7.20: Datenstruktur eines Hybrid-Modellieres mit CSG- und Boundary-Repräsentation

Lokale Beleuchtungsmodelle

7.7 Lokale Beleuchtungsmodelle: Einleitung

Für die Erzeugung möglichst wirklichkeitsnaher Bilder muß neben einer genauen geometrischen Beschreibung und einem Visibilitätsverfahren auch eine Vorschrift zur Berechnung der Farb- und Grauwerte der einzelnen Bildpunkte – ein sogenanntes Beleuchtungsmodell – vorhanden sein. In einem solchen Modell werden die Einflüsse der Lichtquellen (Lage, Größe, Stärke, spektrale Zusammensetzung) sowie der Oberflächenbeschaffenheit (Geometrie, Reflexionseigenschaften) auf die Farbe eines Bildpunktes erfaßt. Das Modell muß also den Vorgang der Lichtausbreitung in einer definierten Umgebung beschreiben und sich auch (mit vernünftigem Aufwand) berechnen lassen. Die Grundlage zur Beschreibung dieser Zusammenhänge sind die Gesetze der Physik.

7.8 Physikalische Grundlagen

Die benötigten physikalischen Grundlagen müssen Auskunft darüber geben, wie Licht und die vom Menschen empfundenen Farbeindrücke zusammenhängen. Dazu gehören mathematische Modelle für die Lichtausbreitung in unterschiedlichen Medien und Maße zur quantitativen Beurteilung der interessierenden Größen. Licht ist der für den Menschen sichtbare Ausschnitt des elektromagnetischen Spektrums von Violett (360 nm = Nanometer) bis Rot (830 nm).

Einige Eigenschaften elektromagnetischer Strahlung werden zur Zeit von den Beleuchtungsmodellen der GDV noch nicht erfaßt. Da jedoch mit schnell wachsender Rechen- und Speicherkapazität zunehmend physikalisch korrektere Modelle entwickelt werden, sollen diese Eigenschaften hier trotzdem erwähnt werden.

Breitet sich Licht einer Frequenz, z.B. eines Lasers, so aus, daß sich die Phasenlage der transversalen Welle nicht ändert, so spricht man von kohärentem Licht. Eine Voraussetzung für Kohärenz ist, daß nur Licht einer Wellenlänge ausgestrahlt wird (monochromatisches Licht). Übliche Lichtquellen zur Raumbeleuchtung sind nicht monochromatisch und strahlen deshalb kein kohärentes Licht aus. Eine weitere Eigenschaft, die sich aus der Wellennatur des Lichts ergibt, ist die Polarisation. Licht ist polarisiert, falls es in einer Schwingungsebene schwingt. Diese Eigenschaft, die für Reflexion und Brechung des Lichts an Materialflächen wichtig ist, wird in der GDV meistens nicht berücksichtigt.

Die Physik unterscheidet drei Gebiete der Optik: die Strahlenoptik (geometrische Optik), die Wellenoptik und die Quantenoptik. Letztere gewinnt durch die Anwendung der Transporttheorie [Kr"91] zur Visualisierung wissenschaftlicher Ergebnisse (VISC = visualization in scientific computing) immer mehr an Bedeutung. Die Wellenoptik erklärt Phänomene, die sich im mikroskopischen Bereich abspielen, d.h. die Abmessungen der betrachteten Objekte liegen in der Größen-

ordnung der Wellenlänge des Lichts. Hier werden Beugung, Interferenz und z.B. die Entstehung von Hologrammen erklärt [Ben82].

Die Strahlenoptik hat für die GDV zur Zeit die größte Bedeutung, da sich die meisten eingesetzten Beleuchtungsmodelle auf makroskopische Vorgänge bei der Lichtausbreitung beschränken. Von großer praktischer Bedeutung ist die *Superpositionseigenschaft* des Lichtes. Diese Eigenschaft erlaubt, daß die Einflüsse mehrerer Lichtquellen und auch kleiner Wellenlängebereiche einzeln berechnet und überlagert werden können. In der GDV sind das z.B. die drei Farbkomponenten **R, G, B**.

7.8.1 Strahlungsphysikalische (radiometrische) Grundgrößen

Dieses Kapitel folgt der Darstellung in [BS93], Kapitel 5.2. Hierbei wird nicht die mathematische Strenge, sondern die Anschaulichkeit betont. Der mit der differentiellen Schreibweise nicht vertraute Leser kann sich vorstellen, daß Symbole, die durch ein vorgestelltes d gekennzeichnet sind, differentiell kleine Elemente bezeichnen, entlang derer alle Größen als konstant angesehen werden können.

Die *Radiometrie* beschäftigt sich mit dem ganzen elektromagnetischen Spektrum. Alle strahlungsphysikalischen Größen bekommen zur Unterscheidung von den entsprechenden photometrischen Größen (nächster Abschnitt) den Index e (für energetisch).

Eine wichtige Größe in der Strahlungsphysik ist der Raumwinkel Ω. Der Raumwinkel Ω, unter dem eine Fläche A von einem Punkt O aus erscheint, ist anschaulich der Flächeninhalt der Fläche, die man erhält, wenn man die Fläche A durch Zentralprojektion mit Zentrum O auf die Oberfläche der Einheitskugel mit Mittelpunkt O projiziert. Um den Zusammenhang zwischen einer differentiellen Fläche dA und dem zugehörigen differentiellen Raumwinkel $d\Omega$ zu bestimmen, werden beide Flächenelemente in Kugelkoordinaten (R, ϑ, φ) dargestellt (vgl. Bild 7.21).

Durch geometrische Betrachtung erhält man:

$$d\Omega = \sin\vartheta \, d\vartheta d\varphi \tag{7.1}$$

und

$$dA = \frac{R^2}{\cos\alpha} \sin\vartheta \, d\vartheta d\varphi. \tag{7.2}$$

Dabei ist R der Abstand des Flächenelements dA vom Punkt O und α der Winkel zwischen der Flächennormalen von dA und dem Verbindungsvektor von O nach dA.

Aus (7.1) und (7.2) folgt dann:

$$d\Omega = \frac{\cos\alpha}{R^2} dA. \tag{7.3}$$

Den Raumwinkel einer endlichen Fläche erhält man durch Integration:

$$\Omega = \int_A \frac{\cos\alpha}{R^2} \, dA. \tag{7.4}$$

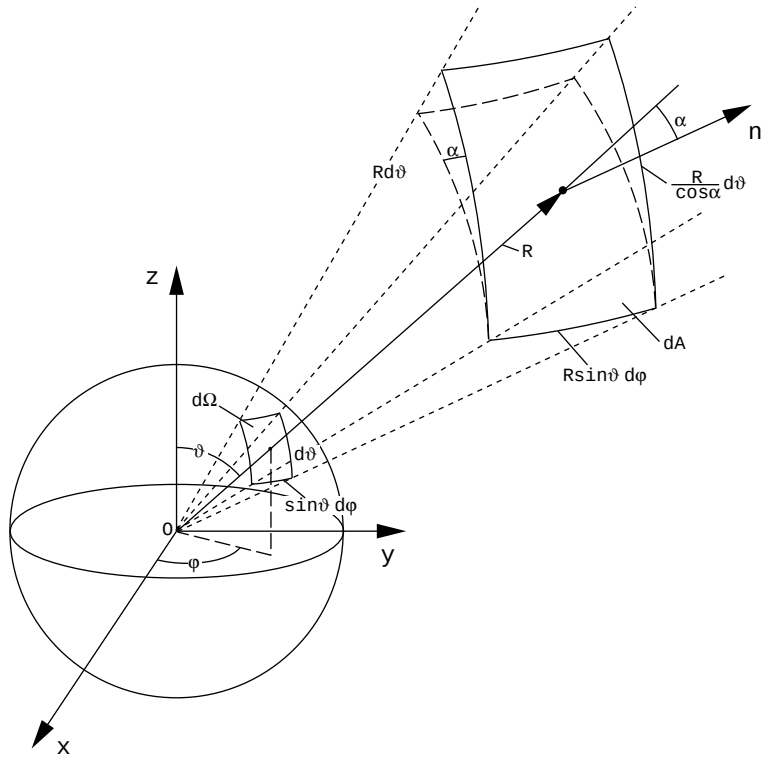


Abbildung 7.21: Zusammenhang zwischen Flächenelement und Raumwinkel

Aus der Gleichung 7.3 sieht man, daß der Raumwinkel dimensionslos ist. Um ihn dennoch zu kennzeichnen, bekommt er die Einheit *sr* (steradiant). Der volle Raumwinkel ist per Definition die Fläche einer Einheitskugel, er hat also die Größe $4\pi sr$.

Die wichtigste Größe in der Radiometrie ist die *Strahlungsenergie* (Radiant Energy) Q_e. Von ihr werden alle anderen Größen abgeleitet.

Die *Strahlungsleistung* (Radiant Flux) φ_e ist die zeitliche Ableitung der Strahlungsenergie. Sendet ein Körper im Zeitintervall dt die Strahlungsenergie dQ_e aus, so ist seine Strahlungsleistung durch

$$\varphi_e = \frac{dQ_e}{dt} \tag{7.5}$$

gegeben. Die Strahlungsleistung wird auch *Strahlungsfluß* genannt und hat die Einheit Watt. Um aufgenommene Strahlungsleistung von abgegebener unterscheiden zu können, bekommt φ_e noch einen Index 1, wenn es sich um eine Strahlungsquelle handelt und eine 2, falls es ein Strahlungsempfänger ist. (Auch die anderen Größen bekommen diesen Index, falls die Unterscheidung wichtig ist.)

Umgibt man eine *Punktlichtquelle* mit einer beliebigen geschlossenen Oberfläche, so darf die gesamte durch diese Oberfläche abgestrahlte Strahlungsleistung aus energetischen Gründen nicht von der Form und der Größe der Fläche abhängen.

Strahlt die Punktlichtquelle in alle Richtungen gleich ab, so muß die auf das differentielle Raumwinkelelement $d\Omega_1$ entfallende Strahlungsleistung $d\varphi_{e_1}$ diesem proportional sein. Es gilt also:

$$d\varphi_{e_1} = I_{e_1} d\Omega_1. \tag{7.6}$$

Die Proportionalitätsgröße I_e heißt *Strahlstärke* oder auch *Intensität* (Intensity). Die Einheit der Strahlstärke ist Wsr^{-1}. Da natürliche Lichtquellen nicht in alle Richtungen gleich stark abstrahlen, ist die Strahlstärke im allgemeinen eine Funktion der Abstrahlrichtung. Bei einem Abstand R zwischen Strahlungsquelle und einem differentiellen Flächenelement dA_2 ergibt sich mit Hilfe von Formel 7.3

$$d\varphi_{e_2} = I_e \frac{\cos\alpha_2}{R^2} dA_2 = E_e dA_2. \tag{7.7}$$

Die neu eingeführte Größe

$$E_e = I_e \frac{\cos\alpha_2}{R^2} \tag{7.8}$$

heißt *Bestrahlungsstärke* (Irradiance). Sie ist ein Maß für die pro Fläche auftreffende Strahlungsleistung und nimmt mit dem Quadrat der Entfernung von der Lichtquelle ab. Dieser Zusammenhang wird als *quadratisches Entfernungsgesetz* bezeichnet. Obwohl es in Strenge nur für punktförmige Strahler gilt, sind in der Praxis die Fehler auch für nichtpunktförmige Strahler klein, solange das Verhältnis zwischen der lateralen Abmessung des Strahlers und der Entfernung zwischen Strahler und Empfänger klein ist.

Die der Bestrahlungsstärke entsprechende Sendegröße heißt *spezifische Ausstrahlung* (Radiosity) M_e:

$$d\varphi_{e1} = M_e dA_1. \tag{7.9}$$

(Aus Symmetriegründen bezeichnen wir den aus der Definition des Raumwinkels in Bild 7.21 stammenden Winkel α im folgenden ebenfalls mit ϑ (vgl. Bild 7.22).)

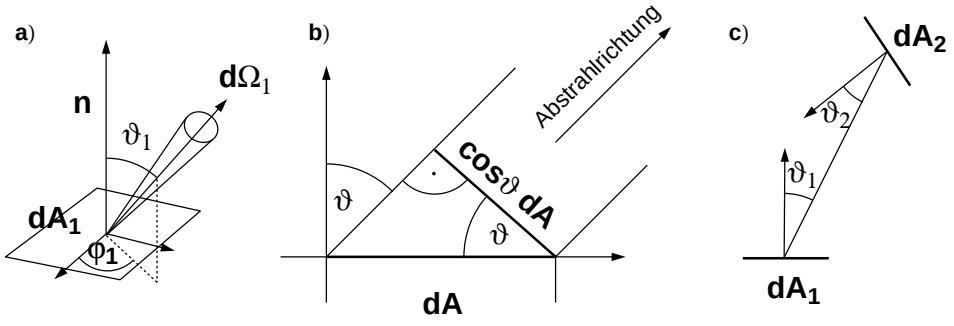


Abbildung 7.22: a) Die von einem Punkt auf dem differentiellen Flächenelement dA_1 in Richtung (ϑ_1, φ_1) in den differentiellen Raumwinkel $d\Omega_1$ abgestrahlte Strahlungsleistung
b) Das unter dem Winkel ϑ gesehene differentielle Flächenelement erscheint um den Faktor $\cos\vartheta$ verkleinert.
c) Strahlungsübertragung zwischen zwei differentiellen Flächenelementen dA_1 und dA_2

Die Einheit von Bestrahlungsstärke, wie auch von spezifischer Ausstrahlung, ist $W\,m^{-2}$. Bei flächenförmigen Lichtquellen ist die Strahlstärke nicht nur richtungsabhängig, sondern auch ortsabhängig. Die auf ein Flächenelement dA_1 bezogene

Strahlungsstärke in Richtung ϑ_1 (Bild 7.22) ist zum einen zur Fläche dieses Flächenelements und zum anderen zum Kosinus dieses Winkels proportional. Die Proportionalität zum Kosinus dieses Winkels rührt daher, daß die aus Richtung ϑ_1 gesehene Fläche dA_1 um den Faktor $\cos\vartheta_1$ verkürzt erscheint. Es gilt

$$dI_{e_1} = L_{e_1} \cos\vartheta_1 dA_1. \tag{7.10}$$

Die neu eingeführte Proportionalitätsgröße L_{e_1} heißt *Strahldichte* (Radiance). Sie hat die Einheit $W\,sr^{-1}\,m^{-2}$.

Multipliziert man diese Gleichung mit $d\Omega_1$, so erhält man mit Hilfe von Gleichung 7.6 daraus die differentielle Strahlungsleistung, die ein differentielles Flächenelement dA_1 in den differentiellen Raumwinkel $d\Omega_1$ abstrahlt:

$$d^2\varphi_{e_1} = L_{e_1} \cos\vartheta_1 dA_1 d\Omega_1. \tag{7.11}$$

Mit Hilfe von Formel 7.3 erhält man daraus das wichtige *Grundgesetz der Strahlungsübertragung*, das die differentielle Strahlungsleistung beschreibt, die ein differentielles Flächenelement dA_1 abstrahlt und die von einem differentiellen Flächenelement dA_2 im Abstand R von dA_1 aufgenommen wird:

$$d^2\varphi_e = L_e \frac{\cos\vartheta_1 \cos\vartheta_2}{R^2} dA_1 dA_2. \tag{7.12}$$

Wegen der Energieerhaltung ist ausgestrahlte und empfangene Leistung gleich. (Streng genommen gilt diese Beziehung nur im Vakuum, da keinerlei Dämpfung der Strahlung durch das zwischen dA_1 und dA_2 liegende Medium berücksichtigt wird.)

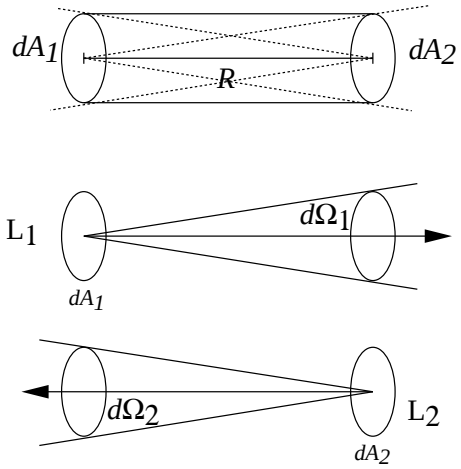


Abbildung 7.23: Zur Gleichheit von gesendeter und empfangener Leistung

Die Beziehung $d^2\varphi_{e_1} = d^2\varphi_{e_2}$ liefert $L_{e_1} = L_{e_2}$, d.h. die Strahldichte ändert sich auf dem Weg der Lichtausbreitung nicht. Bei Strahlverfolgungsverfahren (Raytracing) muß also diese Größe auf dem Weg des Lichts verfolgt werden. Allgemein wird die Leuchtdichte auf dem Bildschirm ausgegeben (vgl. Abschnitt 8.2.2 'Photometrische Grundgrößen').

Mit den bisher definierten strahlungsphysikalischen Größen lassen sich die Abhängigkeiten von der Geometrie vollständig charakterisieren. Noch nicht berücksichtigt wurde die Abhängigkeit von der Frequenz. Zu diesem Zweck wird jeder strahlungsphysikalischen Größe X_e eine *spektrale Dichte* $X_{e\lambda}$ zugeordnet:

$$X_{e\lambda} = \frac{\partial X_e}{\partial \lambda}. \tag{7.13}$$

Oft ist es zweckmäßig, mit einer *normierten spektralen Dichtefunktion*

$$\frac{X_{e\lambda}(\lambda)}{X_{e\lambda}(\lambda_0)} \tag{7.14}$$

zu arbeiten, wobei λ_0 eine zu wählende Bezugsfrequenz ist.

Ein Beispiel zur Berechnung einiger in diesem Abschnitt eingeführter Größen ist in Übungsaufgabe 1 gegeben.

7.8.2 Photometrische Grundgrößen

Im Gegensatz zur Radiometrie beschäftigt sich die Photometrie mit der Messung von elektromagnetischer Strahlung im sichtbaren Bereich von ca. 360 nm bis 830 nm. Der entscheidende Unterschied zur Radiometrie besteht darin, daß nicht nur physikalische Größen, sondern auch die physiologischen Eigenschaften des menschlichen Auges berücksichtigt werden. Jeder strahlungstechnischen Größe wird eine photometrische Größe zugeordnet, wobei zur Unterscheidung der Index v (für visuell) verwendet wird. Außerdem erhalten die photometrischen Größen neue Einheiten.

Der Zusammenhang zwischen radiometrischen und photometrischen Größen wird durch die *Hellempfindlichkeitsfunktion* $V(\lambda)$ hergestellt. Die Hellempfindlichkeitsfunktion gibt die relative Empfindlichkeit des Auges in Abhängigkeit von der betrachteten Wellenlänge an. Ist das Auge dunkeladaptiert (*skotopisches Sehen*), so sind nur die sogenannten Stäbchen des Auges für die Rezeption verantwortlich, während beim helladaptierten Auge (*photopisches Sehen*) nur die Farbzapfen beteiligt sind. Deshalb gibt es zwei voneinander abweichende Hellempfindlichkeitsfunktionen, V für photopisches und V' für skotopisches Sehen.

Durch Gewichtung der strahlungstechnischen Größen $X_{e\lambda}$ mit den Hellempfindlichkeitsfunktionen erhält man die entsprechenden photometrischen Größen $X_{v\lambda}$ und $X'_{v\lambda}$:

$$X_{v\lambda}(\lambda) = Km X_{e\lambda}(\lambda) V(\lambda) \tag{7.15}$$

und

$$X'_{v\lambda}(\lambda) = Km' X_{e\lambda}(\lambda) V'(\lambda). \tag{7.16}$$

Km bzw. Km' sind Proportionalitätsfaktoren.

Die *Lichtstärke* I_v ist das photometrische Gegenstück zur Strahlstärke I_e. Die Bedeutung der Lichtstärke für die Photometrie wird dadurch deutlich, daß sie eine

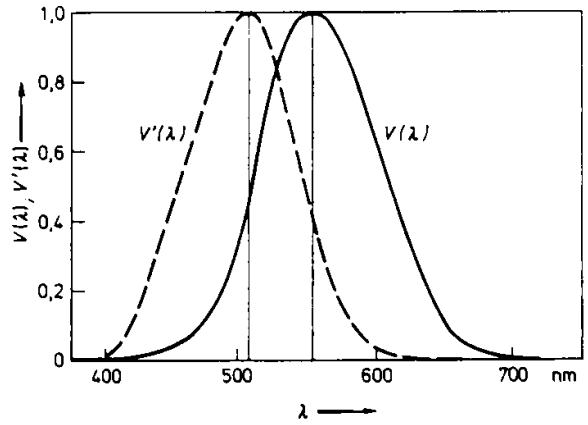


Abbildung 7.24: Hellempfindlichkeitskurven (V für helladaptiertes Sehen, V' für dunkeladaptiertes Sehen)

Radiometrie		
Größe	*Einheit*	*engl. Bezeichnung*
Strahlungsenergie Q_e	J	radiant energy
Strahlungsleistung φ_e	W	radiant power
Strahlstärke I_e	$W\,sr^{-1}$	radiant intensity
Spezifische Ausstrahlung M_e	$W\,m^{-2}$	radiosity
Bestrahlungsstärke E_e	$W\,m^{-2}$	radiance
Strahldichte L_e	$W\,sr^{-1}\,m^{-2}$	radiance

Photometrie		
Größe	*Einheit*	*engl. Bezeichnung*
Lichtmenge Q_v	Lumensek. $(lm\,s)$	quantity of light
Lichtstrom φ_v	Lumen (lm)	liminous flux
Lichtstärke I_v	Candela (cd)	liminous intensity
Spez. Lichtausstrahlung M_v	Phot $= lm\,m^{-2}$	luminosity
Beleuchtungsstärke E_v	Lux $(lx = lm\,m^{-2})$	illuminance
Leuchtdichte L_v	$cd\,m^{-2}$	luminance

Tabelle 7.5: Gegenüberstellung von radiometrischen und photometrischen Größen

eigene SI-Basiseinheit[1] bekommt, nämlich die Candela (cd). Die Definition der Candela ist:

Definition 2.2.1 (Candela): Eine Candela ist die Lichtstärke in einer bestimmten Richtung einer Strahlungsquelle, die monochromatische Strahlung der Frequenz $540\,THz$ (entspricht der Wellenlänge von ca. $555\,nm$) aussendet und deren Strahlstärke in dieser Richtung $\frac{1}{683} W\,sr^{-1}$ beträgt.

[1] SI = Système International (internationales Einheitensystem mit den Grundgrößen Länge, Zeit, Masse, Temperatur, elektrischer Strom, Substanzmenge und Lichtstärke).

Mit dieser Definition sind die Proportionalitätsfaktoren Km und Km' festgelegt. Für das helladaptierte Sehen gilt: $Km = 683\frac{cd}{W\,sr^{-1}}$ und für das dunkeladaptierte Sehen $Km' = 1725\frac{cd}{W\,sr^{-1}}$. Der höhere Wert für das dunkeladaptierte Sehen zeigt die wesentlich höhere Empfindlichkeit des Auges im dunkeladaptiven Zustand.

Das photometrische Gegenstück zur Bestrahlungsstärke ist die *Beleuchtungsstärke*. Die Einheit der Beleuchtungsstärke ist das *Lux*, abgekürzt *lx*. Das *Lux* ist in der Beleuchtungstechnik eine sehr gebräuchliche Einheit. Um eine Vorstellung von dieser Einheit zu bekommen, seien ein paar Beispiele genannt: Die Beleuchtungsstärke, die der Mond auf der Erde erzeugt, ist $0.2lx$, die Mittagssonne erzeugt bis zu $70000lx$. Zum Lesen braucht man wenigstens $100lx$. Für normales Arbeiten werden $500lx$, für Präzisionsarbeit sogar $1000lx$ benötigt.

Weitere photometrische Größen und ihre Maßeinheiten sind in Tabelle 7.5 aufgeführt.

7.8.3 Strahlungsaustausch zwischen Oberflächen

In diesem Abschnitt wird der Strahlungsaustausch zwischen Oberflächen beschrieben und dabei bereits auf verschiedene Beleuchtungsmodelle (vgl. Abschnitt 8.5 'Beleuchtungsmodelle') Bezug genommen, da der Strahlungsaustausch an einer Oberfläche deren Reflexionseigenschaften charakterisiert und dadurch auch messen läßt.

7.8.3.1 Emission

Die Emission wird mit der radiometrischen Größe Strahldichte bzw. ihrem photometrischen Gegenstück, der Leuchtdichte, beschrieben. Im allgemeinen Fall ist die Strahldichte L von der Emissionsrichtung und von der Wellenlänge λ abhängig. Die Richtungsabhängigkeit der Strahldichte L und ihrer spektralen Dichte L_λ wird mit einem Strich gekennzeichnet: $L'(\varphi,\vartheta)$ bzw. $L'_\lambda(\lambda,\varphi,\vartheta)$ (vgl. Bild 7.22 und 7.26). Die beiden Polarwinkel φ und ϑ legen die Abstrahlungsrichtung fest. Die gesamte Emission erhält man durch Integration über den sichtbaren Wellenlängenbereich

$$L'(\varphi,\vartheta) = \int_{\lambda=360nm}^{830nm} L'_\lambda(\lambda,\varphi,\vartheta)\,d\lambda. \tag{7.17}$$

Um die gesamte spektrale Strahlungsleistung pro Fläche, d.h. die spektrale spezifische Ausstrahlung zu bekommen, muß man über den Raumwinkel des ganzen Halbraums integrieren, der der Lichtquelle zugewandt ist :

$$\begin{aligned}\frac{d\varphi}{dA} &= \int_{Halbraum} L'(\varphi,\vartheta)\cos\vartheta\, d\Omega \\ &= \int_{\varphi=0}^{2\pi}\int_{\vartheta=0}^{\pi/2} L'(\varphi,\vartheta)\cos\vartheta\,\sin\vartheta\, d\vartheta\, d\varphi. \end{aligned}\tag{7.18}$$

Ist die Leuchtdichte richtungsunabhängig, kann man L vor das Integral ziehen. Die Integration ergibt

$$\frac{d\varphi}{dA} = L \cdot \pi. \tag{7.19}$$

Die Richtungsunabhängigkeit der Leuchtdichte ist eine wesentliche Voraussetzung für das Radiosity-Verfahren (vgl. Abschnitt 9.2 'Das Radiosity-Verfahren'). Oberflächen, die diese Eigenschaft aufweisen, heißen auch *Lambert'sche Strahler*. Da die Leuchtdichte unabhängig von der Betrachtungsrichtung ist, erscheint die Fläche aus jeder Richtung gleich hell. Löst man Gleichung 7.10 nach der Strahlstärke auf,

$$dI = L\, dA_1 \cos\vartheta_1, \tag{7.20}$$

so sieht man, daß die Strahlstärke eines Lambert'schen Strahlers proportional zum Cosinus des Winkels zwischen Flächennormaler und Ausstrahlungsrichtung ist (*Lambert'sches Cosinusgesetz* von 1760) (vgl. Bild 7.25).

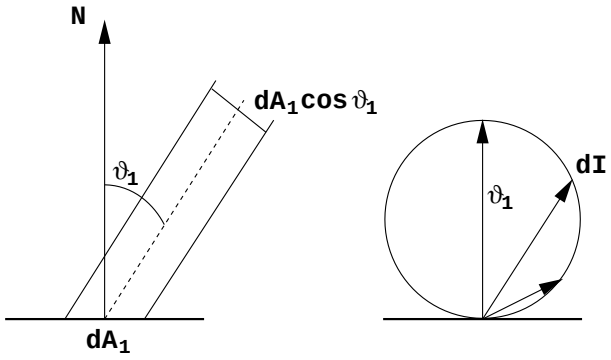


Abbildung 7.25: Lambert'sches Cosinusgesetz

7.8.3.2 Reflexion

Die Reflexion von Strahlung wird durch den *spektralen Reflexionsfaktor* ρ beschrieben, der das Verhältnis von reflektierter Strahldichte zur einfallenden Bestrahlungsstärke E angibt (vgl. Bild 7.26):

$$\rho_\lambda(\lambda, \varphi_r, \vartheta_r, \varphi_i, \vartheta_i) = \frac{L_{\lambda,r}(\lambda, \varphi_r, \vartheta_r)}{E_{\lambda,i}(\lambda, \varphi_i, \vartheta_i)} = \frac{L_{\lambda,r}(\lambda, \varphi_r, \vartheta_r)}{\int_{\Omega_i} L_{\lambda,i}(\lambda, \varphi_i, \vartheta_i) \cos\vartheta_i d\Omega_i} \tag{7.21}$$

Der Index i kennzeichnet die Größen der einfallenden, r die Größen der reflektierten Strahlung. In der englischen Literatur wird diese Größe „bidirectional reflection distribution function (BRDF)" genannt.

Wichtige Eigenschaften von ρ_λ (BRDF) für die Beleuchtungsmodelle der GDV sind:

1. Reziprozität: ρ_λ ändert sich nicht, wenn die Einfalls- und Ausfallsrichtungen vertauscht werden. Hierauf beruht u.a. das Raytracing.

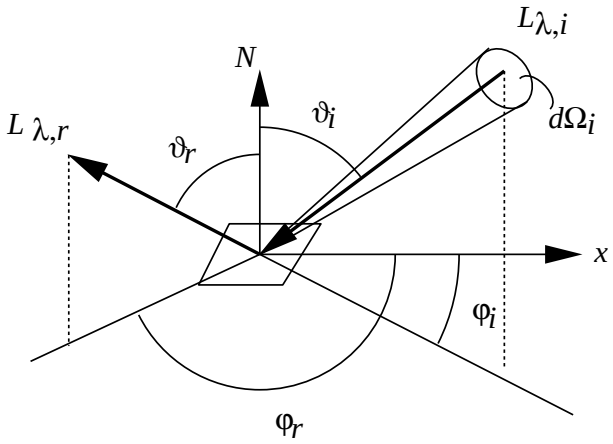


Abbildung 7.26: Zur Definition von ρ: Zusammenhang zwischen einfallendem und reflektiertem Licht

2. ρ_λ ist im allgemeinen anisotrop: Wird bei gleicher Einfalls- und Ausfallsrichtung die Fläche um die Normale verdreht, so ändert sich der Anteil des reflektierten Lichts. Typische Beispiele sind Stoffe oder Metalleffektlacke.

3. Trifft in einem Punkt Licht aus mehreren Richtungen ein, so beeinflussen sich die einzelnen Reflexionen nicht, sondern können linear überlagert werden, wie man aus (7.21) direkt ablesen kann:

$$L_r = \int_{\Omega_i} \rho\, L_i \cos\vartheta_i \, d\Omega_i. \tag{7.22}$$

4. Der Reflexionsfaktor ρ ist wegen der Energieerhaltung immer positiv. Da er auf den Raumwinkel bezogen ist und die Dimension (Raumwinkel^{-1}) besitzt, kann er im Extremfall den Wert ∞ annehmen. Deshalb wird in der GDV, besonders bei den empirischen Beleuchtungsmodellen (vgl. Abschnitt 8.5.2 'Lokale (empirische) Beleuchtungsmodelle'), stattdessen mit dem *Reflexionsgrad* r gearbeitet, der das Verhältnis von reflektierter zu einfallender Bestrahlungsstärke angibt und deshalb dimensionslos ist:

$$r_\lambda = \frac{E_{\lambda,r}}{E_{\lambda,i}}, \quad 0 \le r_\lambda \le 1. \tag{7.23}$$

   Den Zusammenhang zum Reflexionsfaktor ρ erhält man durch Einsetzen von $E_{\lambda,r}$ bzw. $E_{\lambda,i}$ in (7.23) unter Berücksichtigung von (7.21). Eine Herleitung ist in [CW93] zu finden.

Im allgemeinen werden drei Arten der Reflexion unterschieden (vgl. Bild 7.27) [Sil92].

7.8.3.3 Ideal diffuse Reflexion

Im einfachsten Fall ist die reflektierte Leuchtdichte unabhängig von der Abstrahlungsrichtung. Man nennt diesen Fall deshalb auch ideal diffuse oder *Lambert'sche*

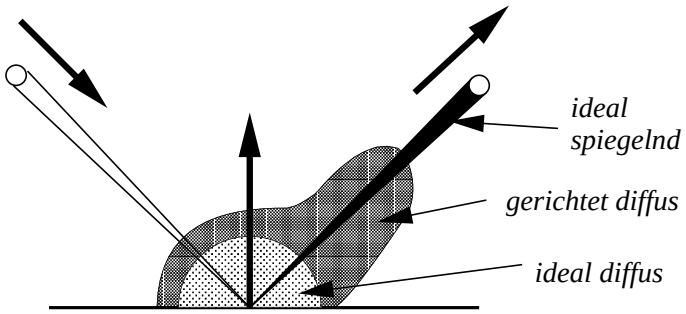


Abbildung 7.27: Drei Arten der Reflexion

Reflexion. Auf dieser Art der Reflexion wird im Radiosity-Verfahren (vgl. Abschnitt 9.2 'Das Radiosity-Verfahren') der Austauschmechanismus für Licht zwischen den Oberflächen der Objekte modelliert. Die reflektierte Leuchtdichte ist zwar unabhängig von den Winkeln (φ_r, ϑ_r), hängt aber von den Einstrahlungswinkeln (φ_i, ϑ_i) ab:

$$L_{\lambda,r} = \rho_\lambda E_{\lambda,i}(\lambda, \varphi_i, \vartheta_i). \tag{7.24}$$

Da die Leuchtdichte unabhängig von der Abstrahlrichtung ist, ergibt sich mit (7.19) für die spezifische Ausstrahlung (Radiosity)

$$M = \pi L_{\lambda,r} \tag{7.25}$$

und mit (7.24)

$$\rho_\lambda = \frac{M}{\pi E_\lambda} \left[\frac{1}{sr}\right]. \tag{7.26}$$

7.8.3.4 Ideal spiegelnde Reflexion

Die spiegelnde Reflexion wird durch das aus der Optik bekannte Reflexionsgesetz beschrieben (vgl. Bild 7.28):
Der einfallende und der reflektierte Strahl bilden mit der Normalen der reflektierenden Oberfläche gleiche Winkel. Einfallender Strahl, reflektierter Strahl und Flächennormale liegen in einer Ebene.
Dies ist die strahlenoptische Formulierung des Reflexionsgesetzes. Für elektromagnetische Wellen gilt das Reflexionsgesetz in gleicher Weise. Ist also die Richtung der einfallenden Strahlung (φ_i, ϑ_i), so wird nur in die Richtung

$$\vartheta_r = \vartheta_i \qquad \text{und} \qquad \varphi_r = \varphi_i + \pi$$

Strahlung reflektiert. Konventionelles Raytracing beruht auf diesem einfachen Reflexionsgesetz.

Der Vollständigkeit halber sei an dieser Stelle noch die ideale Brechung (Transmission) eingefügt (vgl. Bild 7.29).

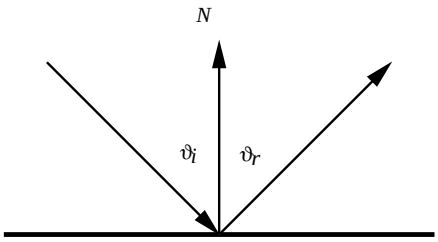


Abbildung 7.28: Geometrie des Reflexionsgesetzes

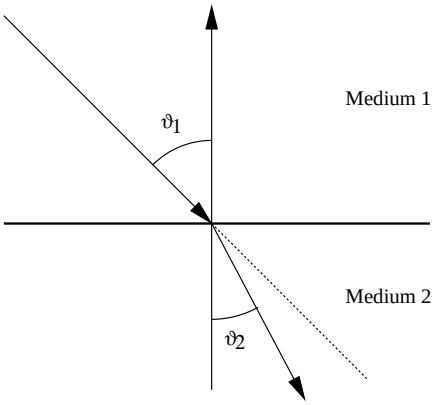


Abbildung 7.29: Geometrie des Brechungsgesetzes beim Übergang von einem optisch dünneren in ein optisch dichteres Medium

Das Brechungsgesetz (Snellius, 1620) besagt:
Einfallender Strahl, Normale und gebrochener Strahl liegen in einer Ebene. Der Sinus des Einfallwinkels steht zum Sinus des Brechungswinkels in einem konstanten Verhältnis, das nur von der Natur der beiden Medien abhängt.

In Formeln:

$$n_1 \sin\vartheta_1 = n_2 \sin\vartheta_2 \Leftrightarrow \frac{\sin\vartheta_1}{\sin\vartheta_2} = \frac{n_2}{n_1} = const. \tag{7.27}$$

n_1 bzw. n_2 sind dabei die *Brechzahlen* (Brechungsindizes) der Medien. Die Brechzahl ist definiert als das Verhältnis der Lichtgeschwindigkeit im Vakuum zur Lichtgeschwindigkeit im betreffenden Medium. Der Brechungsindex des Vakuums ist gleich 1.

Bei der Brechung eines Lichtstrahls ist noch zu beachten, daß beim Übergang von einem optisch dünneren in ein optisch dichteres Medium, also für $n_2 > n_1$, der Lichtstrahl stets zum Einfallslot hin gebrochen wird. Dies kann aus Gleichung 7.27 direkt hergeleitet werden. Trifft dagegen Licht von einem optisch dichteren in ein dünneres Medium ($n_2 < n_1$), so wird es vom Einfallslot weggebrochen. Es gibt dann einen Einfallswinkel ϑ_T, zu dem ein Brechungswinkel von 90° gehört (vgl. Bild 7.30). Das Brechungsgesetz liefert:

$$\sin\vartheta_T = \frac{n_2}{n_1}. \tag{7.28}$$

Wenn dieser Grenzwinkel ϑ_T überschritten wird, ist ein Übergang in das dün-

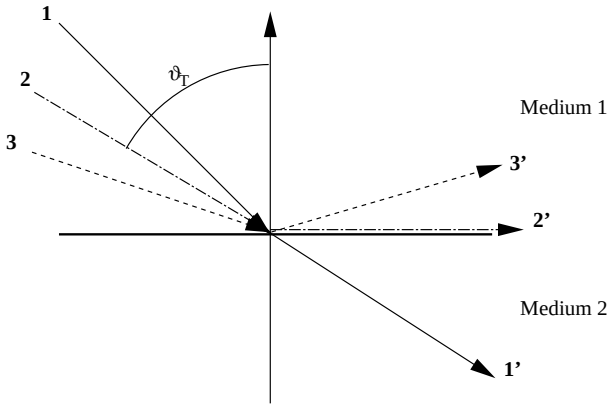


Abbildung 7.30: Totalreflexion
Strahl 2 fällt unter dem Grenzwinkel der Totalreflexion ein.

nere Medium nicht mehr möglich; vielmehr wird alles Licht an der Grenzfläche reflektiert (Totalreflexion).

7.8.3.5 Gerichtet diffuse Reflexion (spekulare Reflexion)

In der Natur trifft man die beiden idealen Reflexionsarten selten. Man muß also für alle Oberflächen die richtungsmäßige Verteilung von $\rho_\lambda(\lambda, \varphi_r, \vartheta_r, \varphi_i, \vartheta_i)$ bestimmen. Sehr häufig tritt der Fall auf, daß ρ ein deutliches Maximum in Richtung der spiegelnden Reflexion hat und kleiner wird, je weiter man sich von dieser Richtung entfernt. Man spricht hier auch von *spekularer Reflexion* (gerichtet diffus, glossy etc.). In der GDV ist es üblich, die spekulare Reflexion in einen richtungsunabhängigen, diffusen Anteil (Index d) und einen richtungsabhängigen Anteil (Index s) aufzuspalten.

Im empirischen Phong-Modell (vgl. Abschnitt 7.11.2 'Lokale (empirische) Beleuchtungsmodelle') wird der richtungsabhängige Anteil über den Reflexionsgrad (vgl. Gleichung 7.23) berechnet als

$$r_s = r_{s,0} \cos^m \gamma. \tag{7.29}$$

$r_{s,0}$ ist eine Konstante zwischen 0 und 1. γ ist der Winkel zwischen der Richtung des ideal reflektierten Strahls und der Beobachtungsrichtung. Der Exponent m gibt an, wie schnell das Reflexionsvermögen mit größer werdendem Winkel γ abfällt.

7.9 Farbmetrik

Aus Sicht der Beleuchtungstechnik wären nun die grundlegenden Größen und Gesetzmäßigkeiten eingeführt, um mit der Formulierung von Beleuchtungsmodellen und Algorithmen beginnen zu können. Das Ziel der GDV ist jedoch nicht nur, die Leuchtdichte an jeder Stelle der Szene zu bestimmen, sondern ein Raster-Bild zu

berechnen, das ein wirklichkeitsgetreues Abbild der Szene ist. Hall und Greenberg [Hal88] fordern sogar, daß das berechnete Bild unser visuelles Aufnahmesystem in den gleichen Zustand versetzt wie eine Betrachtung der realen Szene. Über die Analyse von Lichtverhältnissen in der Szene hinaus müssen also auch die Eigenschaften des Ausgabegerätes berücksichtigt werden.

Leider gibt es keine Ausgabegeräte, die direkt mit einer spektralen Leuchtdichteverteilung für jeden Bildpunkt ansprechbar sind. Üblicherweise wird ein Bildpunkt eines Ausgabegerätes durch seine **Farbe** charakterisiert. Farbe ist jedoch eine Sinnesempfindung, deren äußere Ursachen zwar in den Strahlungsspektren liegen, deren Eigenschaften und Gesetzmäßigkeiten aber nur durch Betrachtung der physiologischen Wahrnehmung des Lichts erschlossen werden können. Dies ist Aufgabe der Farbmetrik, der Lehre von den Maßbeziehungen der Farben untereinander [Sch20].

Um ein realitätsnahes Bild auf dem Bildschirm zu erzeugen, müssen drei Fragen geklärt werden:

1. Wie wird die Farbe eines Bildpunktes auf dem Monitor festgelegt?
2. Wie wird die Farbe vom Auge wahrgenommen und bewertet?
3. Welcher Zusammenhang besteht zwischen Strahlungsspektrum und Farbe?

In den folgenden drei Unterabschnitten soll auf diese Fragen eingegangen werden.

7.9.1 Grundlagen der additiven Farbmischung

Die Farbgebung auf einem Graphikmonitor geschieht auf der Grundlage der additiven Farbmischung. Jeder Bildpunkt des Monitors besteht eigentlich aus drei dicht beieinanderliegenden Bildpunkten. Aufgrund des beschränkten räumlichen Auflösungsvermögens des Auges werden die drei Farbkomponenten im Auge zu einem einheitlichen Farbreiz gemischt. Bei üblichen Monitoren setzt sich die Farbe aus einer roten, einer grünen und einer blauen Komponente zusammen (RGB-Monitor). Für jede Komponente können unterschiedliche Intensitätsstufen gewählt werden. Üblich sind heute 256 Intensitätsstufen, womit sich insgesamt 256^3, also ca. 16 Mio. unterschiedliche Farbkombinationen bilden lassen.

Es stellt sich sofort die Frage, ob drei Grundfarben ausreichen, um sämtliche Farbtöne darzustellen. Eine Antwort darauf gibt das *erste Graßmannsche Gesetz* [Gra53]:
Zwischen je vier Farben besteht immer eine eindeutige lineare Beziehung. Eine Farbe braucht zu ihrer Beschreibung drei voneinander unabhängige Bestimmungsstücke, d.h. die Farbe ist eine dreidimensionale Größe.

Farben können demnach als Vektoren eines dreidimensionalen Vektorraumes aufgefaßt werden. Die Vektoren dieses ‘Farbraums’ heißen *Farbvalenzen*. Die Länge eines Vektors ist ein Maß für die Leuchtdichte und heißt *Farbwert*, seine Richtung bestimmt die *Farbart*. Die Dreidimensionalität der Farbe liegt daran, daß das

menschliche Auge drei verschiedene Arten von Farbrezeptoren (Zapfen) hat. Wie in jedem dreidimensionalen Vektorraum benötigt man drei voneinander linear unabhängige Basisvektoren (*Primärvalenzen*), um den Raum aufzuspannen. In diesem Fall bedeutet linear unabhängig, daß eine Primärvalenz nicht durch Mischung der beiden anderen Primärvalenzen darstellbar ist. Mit drei Primärvalenzen $\mathcal{R}$, $\mathcal{G}$ und $\mathcal{B}$ läßt sich also für jede Farbvalenz $\mathcal{F}$ eine Farbgleichung aufstellen:

$$\mathcal{F} = r\mathcal{R} + g\mathcal{G} + b\mathcal{B}. \tag{7.30}$$

Die Werte für r, g und b können nur durch ein Mischexperiment gewonnen werden. Dabei werden die Farbwerte der Primärvalenzen solange verändert, bis der gleiche Farbton getroffen wird.

Wenn Sie die PDF-Version des Kurses bearbeiten, können Sie jetzt das Interaktionselement 'colormixing' anwählen.

In vielen Fällen wird man jedoch feststellen, daß eine unbekannte Farbvalenz $\mathcal{N}$ nicht durch Mischen erzeugt werden kann. Es gelingt indessen immer, Farbgleichheit herzustellen, wenn man der unbekannten Farbvalenz $\mathcal{N}$ Primärvalenzen hinzumischt, z.B.:

$$\mathcal{N} + r'\mathcal{R} = g'\mathcal{G} + b'\mathcal{B}\,,$$

d.h.:

$$\mathcal{N} = -r'\mathcal{R} + g'\mathcal{G} + b'\mathcal{B}.$$

Das Graßmannsche Gesetz wird also nicht verletzt, es müssen nur analog zum Vektorraum auch negative Farbwerte zugelassen werden. Man nennt diese Art von Farbmischung *äußere Farbmischung*; sind alle drei Farbwerte positiv, handelt es sich um eine *innere Mischung*.

Eine weitere Eigenschaft eines Vektorraums ist die *Linearität*.
So muß für

$$\mathcal{F} = \mathcal{U} + \mathcal{V} + \mathcal{W}$$

auch

$$k\mathcal{F} = k\mathcal{U} + k\mathcal{V} + k\mathcal{W}$$

gelten. Auch diese Eigenschaft wird durch das Experiment bestätigt.
Mit Farbvalenzen kann man rechnen wie mit Vektoren, insbesondere ist die Umrechnung der Darstellung bezüglich verschiedener Primärvalenztripel möglich.

Die Bedeutung der Farbvalenz für das Farbempfinden macht das *zweite Graßmannsche Gesetz* deutlich:

Zweites Grassmannsches Gesetz:
Gleich aussehende Farben ergeben mit einer dritten Farbe stets gleich aussehende Farbmischungen.

Das heißt, daß es bei der Beurteilung von Gleichheit zweier Farben nur auf die Farbvalenz ankommt. Die Zusammensetzung und insbesondere die Wahl der Primärvalenzen spielen keine Rolle.

Für die bildliche Darstellung einer Farbtafel ist das dreidimensionale Vektormodell der Farbvalenzen ungeeignet. Man benutzt daher häufig eine zweidimensionale Darstellung mit Hilfe von baryzentrischen Koordinaten (in der Literatur oft als Schwerpunktskonstruktion bezeichnet) (vgl. Abschnitt 3.1.2.1 'Baryzentrische Koordinaten').

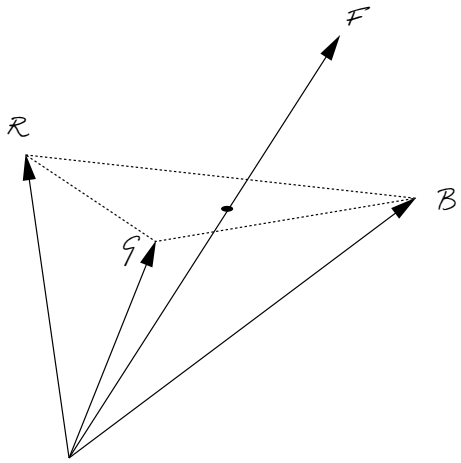


Abbildung 7.31: Schwerpunktskonstruktion der Farbvalenzen

Hierzu wird jede Farbvalenz auf die durch die Endpunkte der Primärvalenzen gegebene Ebene projiziert (vgl. Bild 7.31). Der *Farbort* auf der Ebene wird durch baryzentrische Koordinaten r_F, g_F und b_F bezüglich dem von den Primärvalenzen gebildeten Dreieck bestimmt.
Ist eine Farbvalenz F gegeben durch

$$\mathcal{F} = R_F\,\mathcal{R} + G_F\,\mathcal{G} + B_F\,\mathcal{B},$$

so sind die baryzentrischen Koordinaten

$$\begin{aligned} r_F &= \frac{R_F}{R_F + G_F + B_F} \\ g_F &= \frac{G_F}{R_F + G_F + B_F} \\ b_F &= \frac{B_F}{R_F + G_F + B_F}. \end{aligned} \tag{7.32}$$

Aus (7.32) entnimmt man, daß $r_F + g_F + b_F = 1$. Es genügen also zwei baryzentrische Koordinaten zur Festlegung des Farborts.

In der baryzentrischen Darstellung werden auch die Begriffe *innere* und *äußere Farbmischung* verständlich. Farben, die durch innere Farbmischung zustande kommen, befinden sich im Innern des Dreiecks, äußere Farbmischungen führen zu Farborten außerhalb des Dreiecks. Die übersichtlichere Darstellung wird allerdings damit erkauft, daß man keine Information mehr über den Farbwert hat.

7.9.2 Spektralwerte

In diesem Abschnitt soll geklärt werden, wie für eine bestimmte Verteilung (auch Spektralreiz genannt) die Farbgleichung bei vorgegebenen Primärvalenzen bestimmt werden kann. Oder anschaulich formuliert: Wie muß man seine Grundfarben mischen, um das Farbempfinden eines bestimmten Spektralreizes L_λ zu erreichen?

Dazu zerlegt man den sichtbaren Wellenlängenbereich in enge Spektralbänder $\lambda_{k,\Delta\lambda} \ldots \lambda_{l,\Delta\lambda}$ der Bandbreite $\Delta\lambda$ und betrachtet zunächst das Farbempfinden, das ein auf das Spektralband $\lambda_{i,\Delta\lambda}$ beschränkter 'monochromatischer' Spektralreiz erzeugt. $\Delta\lambda$ ist in der Größenordnung von 5 bis 10 nm. Man nimmt kein streng monochromatisches Licht, weil sonst die Lichtausbeute zu gering wäre und das Auge ohnehin keine höhere spektrale Auflösung hat. Für die zu diesem Farbreiz gehörende Farbvalenz $\mathbf{f}_{\lambda_i}$ kann man wiederum eine Farbgleichung aufstellen:

$$\mathbf{f}_{\lambda_i} = \mathbf{r}_{\lambda_i}\,\mathcal{R} + \mathbf{g}_{\lambda_i}\,\mathcal{G} + \mathbf{b}_{\lambda_i}\,\mathcal{B}. \tag{7.33}$$

Die Farbkoeffizienten $\mathbf{r}_{\lambda_i}, \mathbf{g}_{\lambda_i}$ und $\mathbf{b}_{\lambda_i}$ heißen *Spektralwerte* bzgl. der Primärvalenzen $\mathcal{R}$, $\mathcal{G}$, $\mathcal{B}$. Die Spektralwerte können nur mittels Mischexperimenten gewonnen werden. Führt man dieses Experiment für jedes Spektralband innerhalb des sichtbaren Spektralbereichs durch, erhält man die sogenannten *Spektralwertkurven* $\overline{r}$, $\overline{g}$ und $\overline{b}$ (vgl. Bild 7.32).

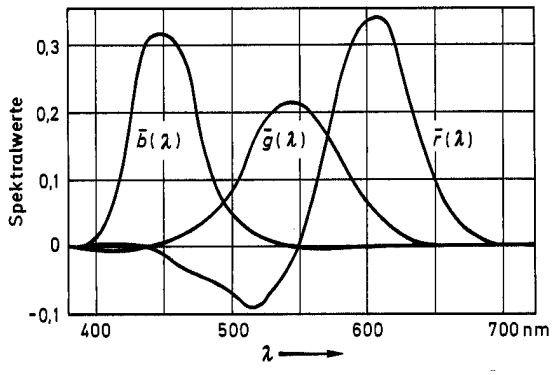


Abbildung 7.32: Spektralwertkurven bezogen auf die Primärvalenzen $\mathcal{R} = 700nm$, $\mathcal{G} = 546nm$ und $\mathcal{B} = 436nm$

Hat man die Spektralwertkurven für ein bestimmtes Primärvalenztripel bestimmt, so kann man daraus die Spektralwertkurven für jedes beliebige Primärvalenztripel berechnen. Mit Hilfe der Spektralwertkurven für ein gegebenes Primärvalenztripel kann man den Farbreiz, den ein ganzes Wellenlängenspektrum L_λ erzeugt, ermitteln. Man ersetzt den Spektralreiz L_λ durch eine Überlagerung 'monochromatischer' Spektralreize $L_{\lambda_i,\,\Delta\lambda}$:

$$L_\lambda = \sum_{i=k}^{l} L_{\lambda_i,\,\Delta\lambda}. \tag{7.34}$$

Die Koeffizienten der zugehörigen Farbvalenz

$$\mathcal{F}(L_\lambda) = R_{L_\lambda}\,\mathcal{R} + G_{L_\lambda}\,\mathcal{G} + B_{L_\lambda}\,\mathcal{B},$$

R_{L_λ}, G_{L_λ} und B_{L_λ}, erhält man durch Aufsummieren der mit den Leuchtdichten L_i der einzelnen Spektralwerte $L_{\lambda_i,\,\Delta\lambda}$ gewichteten Spektralwerte:
Mit

$$\begin{aligned} L_i &= \int_{\lambda_i-\frac{\Delta\lambda}{2}}^{\lambda_i+\frac{\Delta\lambda}{2}} L_{\lambda_i,\,\Delta\lambda}\,d\lambda \\ &\approx L_{\lambda_i,\,\Delta\lambda}\,\Delta\lambda \end{aligned}$$

erhält man

$$R_{L_\lambda} = \sum_{i=k}^{l} r_{\lambda_i} L_i\,, \quad G_{L_\lambda} = \sum_{i=k}^{l} g_{\lambda_i} L_i\,, \quad B_{L_\lambda} = \sum_{i=k}^{l} b_{\lambda_i} L_i\,. \tag{7.36}$$

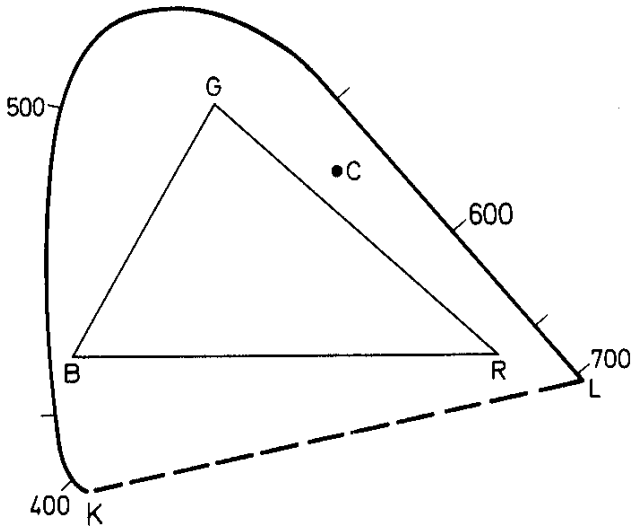


Abbildung 7.33: Spektralvalenzkurve in der Schwerpunktsdarstellung

An den Spektralwertkurven in Bild 7.32 fällt auf, daß auch negative Farbwerte vorkommen. Manche spektralen Farbvalenzen können also nur durch äußere Mischung erzeugt werden. Diese Tatsache wird noch deutlicher, wenn man sich die Farbtafel in der Schwerpunktsdarstellung betrachtet (vgl. Bild 7.33 und 7.34). Die spektralen Valenzen $\mathbf{f}(\lambda)$ bilden eine stetige konvexe Kurve. Die Kurve hat ein langwelliges und ein kurzwelliges Ende. Sie ist nicht geschlossen, weil das kurzwellige Ende im Violett-Bereich und das langwellige Ende im Rot-Bereich liegt, was natürlich unterschiedlichen Farbvalenzen entspricht. Alle real darstellbaren Farben liegen innerhalb der Kurve, weil jede reale Farbe einem Frequenzspektrum entspricht. Die einzelnen Frequenzen des Spektrums (und damit die spektralen Farbvalenzen) können aber nur einen positiven (oder keinen) Beitrag liefern.

Anders ausgedrückt: Sämtliche realen Farben sind innere Farbmischungen der spektralen Farbvalenzen. Das bedeutet aber auch, daß jedes reale Primärvalenztripel innerhalb der Spektralvalenzkurve liegt. Es gibt also immer Farbvalenzen, die nur durch äußere Farbmischung aus den Primärvalenzen erzeugt werden können. Das ist aber bei einem RGB-Monitor nicht möglich, da negative Farbwerte der Primärvalenzen nicht realisierbar sind. Selbst bei den über 16 Millionen Farben eines Hochleistungs-RGB-Monitors gibt es also Farbtöne, die dieser nicht darstellen kann. Wie groß das ‘Spektrum’ der darstellbaren Farben ist, hängt davon ab, wie groß die Fläche des Primärvalenzdreieckes innerhalb der Spektralvalenzkurve ist. Die Qualität eines Farbmonitors hängt also wesentlich von der Güte des Farbphosphors ab, mit dem die Primärvalenzen erzeugt werden. Um nicht für jedes Primärvalenztripel von neuem die Spektralwertkurven bestimmen zu müssen, wurde schon 1931 von der *Commision International de l’Éclairage* (CIE) ein Standardsystem von Primärvalenzen vorgeschlagen (vgl. Bild 7.34). Das besondere an diesen CIE-Primärvalenzen ist, daß sie außerhalb der Spektralvalenzkurve liegen. Sie sind also gar nicht darstellbar; man nennt solche Farbvalenzen deshalb *virtuell*.

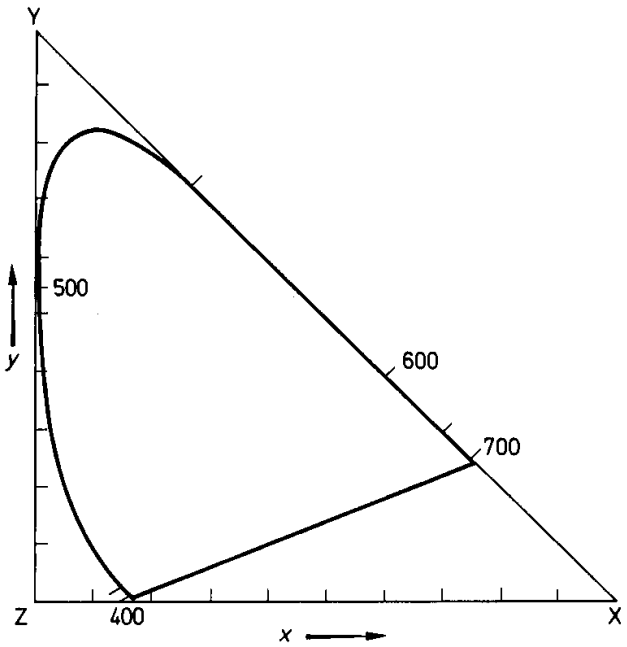


Abbildung 7.34: Spektralvalenzkurve im CIE-Farbsystem

Weil die Spektralvalenzkurve innerhalb des Dreiecks der CIE-Primärvalenzen $\mathcal{X}$, $\mathcal{Y}$ und $\mathcal{Z}$ liegt, sind die zugehörigen Spektralwertkurven an jeder Stelle positiv (vgl. Bild 7.35).

Diese Spektralwertkurven lassen sich, da es sich um virtuelle Farbvalenzen handelt, nicht direkt durch Mischexperimente gewinnen. Sie müssen aus den Spektralwertkurven eines realen Primärvalenztripels berechnet werden.

Obwohl virtuelle Farbvalenzen nicht darstellbar sind, läßt sich mit ihnen genau-

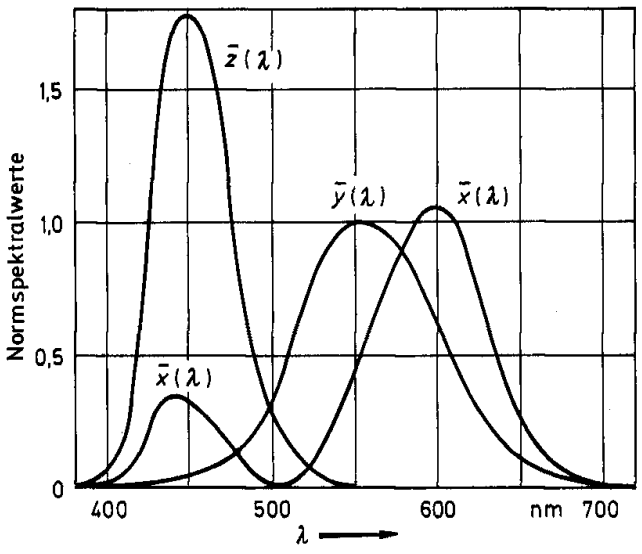


Abbildung 7.35: Spektralwertkurven im CIE-Farbsystem

so rechnen wie mit realen Farbvalenzen. Insbesondere ist die Umrechnung auf ein reales Primärvalenztripel jederzeit möglich. Sie entspricht, in Analogie zum Vektorraum, einer Basistransformation. Wir betrachten zur Erläuterung die Transformation von einem Primärvalenzsystem $S = R,G,B$ in ein System $S' = P,D,T$:

Zunächst drückt man die Primärvalenzen von S' in Koordinaten von S aus. Das heißt

$$\begin{aligned} P &= p_r \cdot R + p_g \cdot G + p_b \cdot B \\ D &= d_r \cdot R + d_g \cdot G + d_b \cdot B \\ T &= t_r \cdot R + t_g \cdot G + t_b \cdot B \end{aligned}$$

oder in Vektorschreibweise

$$P = \begin{pmatrix} p_r \\ p_g \\ p_b \end{pmatrix}_S, \; D = \begin{pmatrix} d_r \\ d_g \\ d_b \end{pmatrix}_S, \; T = \begin{pmatrix} t_r \\ t_g \\ t_b \end{pmatrix}_S.$$

Diese Vektoren bilden die Zeilen der Transformationsmatrix M

$$S' = MS = \begin{pmatrix} p_r \; p_g \; p_b \\ d_r \; d_g \; d_b \\ t_r \; t_g \; t_b \end{pmatrix} S.$$

Da die Primärvalenzen von $S' = P,D,T$ linear unabhängig sind, ist M invertierbar.

Um also die Spektralwertkurven für einen speziellen Farbmonitor zu bestimmen, reicht es aus, wenn der Hersteller angibt, wie sich die realen Primärvalenzen des Monitors mit den CIE-Valenzen ausdrücken lassen. Es müssen keine Mischexperimente durchgeführt werden. Eine ausführliche Darstellung mit Rechenbeispielen ist in [Jac92] zu finden.

Wenn Sie die PDF-Version des Kurses bearbeiten, können Sie jetzt das Interaktionselement 'orangevaluation' anwählen.

7.10 Farbmodelle

Durch die zunehmende Verbreitung der Farbrastertechnologie nutzen immer mehr Anwendungen die Möglichkeit der Farbdarstellung zur Gestaltung. So vielfältig die Nutzung der Farbe ist, so vielfältig sind auch die Probleme bei der Farbdefinition, -wiedergabe und -wahrnehmung. In diesem Abschnitt werden Systeme zur Farbdefinition beschrieben, auf ihre Anwendbarkeit untersucht und verglichen.

Wesentliche Einflüsse auf die optische Wirkung bei der Wiedergabe auf dem Bildschirm (Monitor) sind die Umgebung, wie z.B. Farbe des Hintergrundes, Helligkeit im Raum, Beleuchtungsquelle und daraus resultierende Reflexionen etc. [Hä81]. Zwei weitere wichtige Punkte sind die Wahl der Leuchtstoffe und die Einstellung des Monitors nach genormten Richtlinien [CCI82].

Die größte Zahl der Geräte wird entsprechend den Farbkathoden im Monitor über die drei Primärfarben Rot, Grün und Blau angesprochen. Die Mischfarben werden durch eine additive Überlagerung der Primärfarben dargestellt. Diese Definition der Farbe entspricht jedoch nicht dem Wahrnehmungsempfinden des Menschen, das sich eher an den Parametern Farbton, Helligkeit und Sättigung orientiert [JW63], [Wys60].

Im folgenden werden die unterschiedlichen Farbmodelle erklärt und klassifiziert. Sie lassen sich in technisch–physikalische und wahrnehmungsorientierte Farbsysteme oder –modelle unterteilen.

7.10.1 Technisch-physikalische Farbmodelle

Die technisch-physikalischen Modelle beschreiben eine Farbe als Mischung dreier Primärfarben. Die Unterschiede zwischen den einzelnen Modellen liegen in der Wahl der Primärfarben und der Art der Farbmischung.

7.10.1.1 Das RGB–Modell

Beim RGB–Modell werden die darstellbaren Farben als Punkte eines Einheitswürfels im Ursprung des kartesischen Koordinatensystems beschrieben (vgl. Bild

7.36). Auf den positiven Halbachsen liegen die Primärfarben Rot, Grün und Blau. Grauwerte, darstellbar durch gleichgroße Anteile von R, G und B, liegen auf der Hauptdiagonalen des Einheitswürfels mit Schwarz im Ursprung $[0,0,0]$ und Weiß im Punkt $[1,1,1]$. Eine Farbe wird dann durch Anteile von R, G und B beschrieben, die zu Schwarz addiert werden müssen. Farbrastersichtgeräte haben R, G, B–Leuchtstoffe, die über individuelle Kathoden angeregt werden, woraus sich die besondere Bedeutung des RGB-Modells ergibt: Alle anderen Farbbeschreibungen müssen vor der Farbausgabe in den äquivalenten Punkt des RGB–Würfels umgerechnet werden.

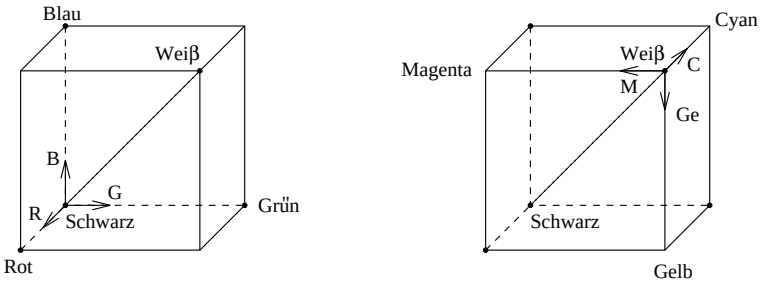


Abbildung 7.36: RGB–und Cyan, Magenta, Gelb–Einheitswürfel

Zur Bezeichnung von Farben für die Plotterausgabe ist ein zum RGB–Würfel komplementäres Modell zweckmäßig, bei dem Weiß im Koordinatenursprung liegt. Auf den Koordinatenachsen werden dann die Primärfarben Cyan, Magenta und Gelb abgetragen (vgl. Bild 7.36). Schwarz liegt bei $[1,1,1]$, weshalb Cyan, Magenta und Gelb im Gegensatz zu Rot, Grün und Blau als subtraktive Primärfarben bezeichnet werden. Die Umrechnung zwischen beiden Modellen ist einfach:
Von RGB nach CMGe:

$$[C,M,Ge] = [1,1,1] - [R,G,B], \tag{7.37}$$

und von CMGe nach RGB:

$$[R,G,B] = [1,1,1] - [C,M,Ge]. \tag{7.38}$$

7.10.1.2 Das YIQ–Modell

Farben können neben der Beschreibung durch drei Primärfarben auch durch Leuchtdichte (Luminanz, Helligkeit) und Farbart (Chrominanz) definiert werden. Diese Tatsache ist für das Fernsehen von größter Bedeutung, weil dadurch *Farb–* und *S/W–Fernsehen* kompatibel bleiben. Die Luminanz enthält dann die S/W–Information bzw. Helligkeit, während die *Chrominanz* die Farbinformation zur Kolorierung des S/W–Bildes darstellt. Ihre Kenngrößen sind der *Farbton* und die *Farbsättigung*. Der Farbton ist mit der dominierenden Wellenlänge des Lichtes gegeben, man spricht von Rot, Orange, Blau usw. Die Sättigung ist ein Maß für die spektrale Reinheit, d.h. wie kräftig die Farbe bzw. wieviel unbuntes Licht (weiß) beigemischt ist. Aus den RGB-Werten wird die Luminanz als bewertete Summe gebildet:

$$Y = 0,3R + 0,59G + 0,11B. \tag{7.39}$$

Die Bewertung der einzelnen Farbbereiche durch die verschieden großen Koeffizienten entspricht der unterschiedlichen Hellempfindung (vgl. Bild 7.37).

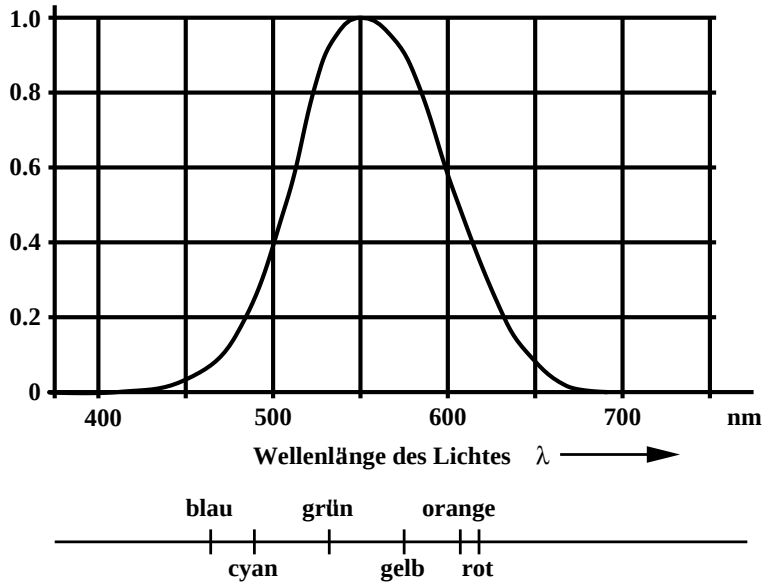


Abbildung 7.37: Hellempfindung bei energiegleicher Strahlung im Spektrum

Die Chrominanz wird mit den zwei Differenzen $R-Y$ und $B-Y$ angegeben. Zusammen mit der Luminanz, die ja bereits die Summeninformation von RGB enthält, ist dann die gewünschte Farbe vollständig beschrieben. Die Differenzen werden bewertet zu den Größen I und Q zusammengefaßt [The73]:

$$\begin{aligned} \mathrm{I} &= 0{,}74\ (\mathrm{R\text{-}Y}) - 0{,}27\ (\mathrm{B\text{-}Y}) \\ &= 0{,}6\mathrm{R} - 0{,}28\ \mathrm{G} - 0{,}32\ \mathrm{B} \\ \mathrm{Q} &= 0{,}41\ (\mathrm{B\text{-}Y}) + 0{,}48(\mathrm{R\text{-}Y}) \\ &= 0{,}21\ \mathrm{R} - 0{,}52\ \mathrm{G} + 0{,}31\ \mathrm{B}, \end{aligned} \tag{7.40}$$

bzw. in Matrixform:

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0{,}30 & 0{,}59 & 0{,}11 \\ 0{,}60 & -0{,}28 & -0{,}32 \\ 0{,}21 & -0{,}52 & 0{,}31 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}. \tag{7.41}$$

I und Q enthalten keine Helligkeitsinformation, was man sofort mit $R = G = B$ feststellt. Möchte man garantieren, daß bestimmte farbige Objekte auch im S/W–Bild zu unterscheiden sind, so muß man die RGB-Komponenten im Hinblick auf unterschiedliche Y–Werte wählen.

Das YIQ–Modell erlaubt, die unterschiedliche Auflösung des Gesichtssinns für Leuchtdichte und Farbart zur Datenreduktion bzw. Bandreduktion auszunutzen und wird im amerikanischen NTSC-System verwendet.

7.10.1.3 Das YUV–Modell

Die europäischen Systeme PAL (Deutschland) und SECAM (Frankreich) verwenden ebenfalls die Helligkeit Y und zwei Farbdifferenzen U und V. Die U- und V-Signale lassen sich durch eine einfache Rotation der Koordinaten im Farbraum in die I- und Q-Komponenten überführen [Pra78].

$$I = -U\sin(33^\circ) + V\cos(33^\circ) \tag{7.42}$$

$$Q = U\cos(33^\circ) + V\sin(33^\circ). \tag{7.43}$$

7.10.2 Wahrnehmungsorientierte Farbmodelle

Die technisch-physikalischen Farbmodelle sind an den Erfordernissen der Ausgabegeräte und Übertragungstechnik orientiert. Sie eignen sich wenig zur direkten Farbdefinition durch den Benutzer. Deshalb wurden Farbmodelle entwickelt, die mit Größen entsprechend der menschlichen Wahrnehmung arbeiten, nämlich Helligkeit, Farbton und Farbsättigung. Diese Größen müssen zur Ausgabe aber in ein technisch-physikalisches Modell transformiert werden.

7.10.2.1 Numerisch-symbolische Eingabe

Die numerisch-symbolische Eingabe wird in folgenden Systemen eingesetzt: HLS, HSV [FvDFH90], [Smi78], H'L'S', H'S'V' [Kau82], Ridgway, Ostwald, Munsell, DIN und Hesselgren [Wys60]. Die ersten vier erwähnten Systeme haben kontinuierliche, die anderen diskrete Wertebereiche für ihre Farbdimensionen.

Als ein Beispiel für die Systeme mit numerisch-symbolischer Eingabe sei hier das HLS-System (H=Hue (Farbton), L=Lightness (Helligkeit), S=Saturation (Sättigung)) näher betrachtet (vgl. Bild 7.38).

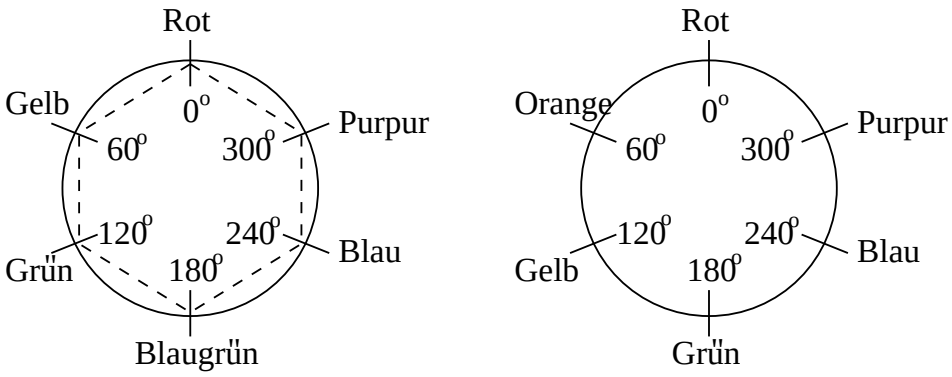


Abbildung 7.38: HLS- und H'L'S'–Farbkreise

Die Farbanordnung entspricht der senkrechten Projektion des RGB–Würfels von

Weiß nach Schwarz entlang der Hauptdiagonalen. Das entstehende regelmäßige Sechseck wird meist durch einen Kreis ersetzt, so daß der Farbton (H) als Winkel zwischen 0° und 360° anzugeben ist. Das H'L'S'-System entsteht durch Verschieben von Grün in Richtung Blau. Dadurch liegen Rot, Gelb und Blau gleich weit voneinander entfernt, was der Farbempfindung besser entspricht.

Die Helligkeit (L) wird als Wert zwischen 0 und 1 angegeben, wobei 0 Schwarz und 1 Weiß entspricht.

Die Sättigung (S) wird als Abstand einer Farbe vom Mittelpunkt des Farbkreises angegeben. Sie beträgt 0 für achromatische Farben und kann als höchsten Wert 1 für die gesättigten Farben auf dem Rand des Farbkreises annehmen. Bei Farben mit der Helligkeit $0,5$ ist die volle Sättigung 1 möglich. Mit zunehmender oder abnehmender Helligkeit nimmt die maximal mögliche Sättigung ab. Je nachdem, ob die Sättigung absolut oder relativ zur maximal bei einer bestimmten Helligkeit erreichbaren Sättigung angegeben wird, verwendet man deshalb das Doppelkegelmodell oder das Zylindermodell (vgl. Bild 7.39). Dabei wird auf der senkrech-

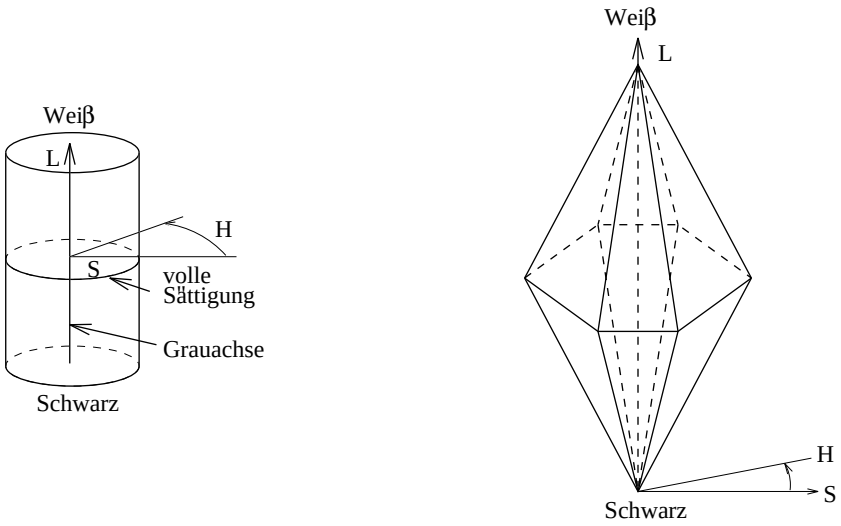


Abbildung 7.39: HLS-/H'L'S'–Farbkörper

ten Achse die Helligkeit abgetragen. Die absolute bzw. relative Sättigung ist durch den Abstand von der Achse gegeben.

In den Algorithmen zur Umrechnung zwischen HLS und RGB werden folgende Abkürzungen verwandt:

R Rotwert
G Grünwert
B Blauwert

min	Minimum	(R,G,B)
max	Maximum	(R,G,B)
H	hue	= Farbton
L	lightness	= Helligkeit
S	saturation	= Sättigung

7.10.2.2 RGB nach HLS-/H'L'S'

Für das HLS-/H'L'S'-System wird der Mittelwert aus *max* und *min* als Helligkeit gewählt. Die Sättigung wird als Funktion der Helligkeit errechnet. Ist sie kleiner oder gleich 0.5, so wird als Sättigung das Verhältnis von Abstand des Minimums von der Helligkeit zum Abstand des absoluten Minimums von der Helligkeit definiert:

$$S = \frac{L - min}{L - 0} = \frac{(max + min)/2 - min}{(max + min)/2} = \frac{max - min}{max + min} \tag{7.44}$$

Ist die Helligkeit größer als 0.5, wird die Sättigung als Verhältnis vom Abstand des Maximums von der Helligkeit zum Abstand des absoluten Maximums von der Helligkeit definiert:

$$S = \frac{max - L}{1 - L} = \frac{max - (max + min)/2}{1 - (max + min)/2} = \frac{max - min}{2 - max - min} \tag{7.45}$$

Die Berechnung des Farbtons erfolgt unter der Voraussetzung, daß $H = 0$ Rot entspricht und H für $S = 0$ undefiniert ist (vgl. Bild 7.38).

7.10.2.3 HLS nach RGB

Da sich Helligkeit und Sättigung allein aus *max* und *min* berechnen, lassen sich in umgekehrter Richtung *max* und *min* auch wieder aus Helligkeit und Sättigung berechnen.

Für $L \leq 0,5$ ergibt sich:

$$\begin{aligned} S &= \frac{max - min}{max + min} \\ &= \frac{2\,max - max - min}{max + min} \\ &= \frac{max - (max + min)/2}{(max + min)/2} \\ &= \frac{max - L}{L}. \end{aligned}$$

Daraus folgt

$$LS = max - L,$$

und somit ist

$$max = L(1+S).$$

Falls $L > 0,5$ ist, gilt

$$S = \frac{max - L}{1 - L}.$$

Daraus folgt

$$S(1-L) = max - L \quad \text{und} \quad max = L + S - LS.$$

Aus der Definition der Helligkeit erhält man

$$min = 2L - max.$$

Welche der drei Primärfarben *max* und welche *min* ist, wird durch H bestimmt (siehe Algorithmus).

Ist H nahe einer Primärfarbe, d.h. bei 0, 120 oder 240 Grad im HLS-System bzw. 0, 180 oder 240 Grad im H'L'S'-System, liegt der auf den Bereich von 0 bis 6 transformierte H-Wert nahe 0, 2, 4 oder 6. In solchen Fällen darf der Wert der zweitstärksten Farbe nur wenig mehr als das *min* betragen.
Bezeichnet H eine 1:1-Mischfarbe, d.h. liegt der transformierte H-Wert nahe 1, 3 oder 5, so muß die zweitstärkste Farbe das *max* erreichen. Das bedeutet, daß die Differenz *max* – *min* anteilig zum *min*-Wert addiert werden muß. Dieser Anteil ist je nach Sektor der gebrochenzahlige Anteil von H oder 1 minus dem gebrochenzahligen Anteil [FvDFH90], [Kau82].

7.11 Beleuchtungsmodelle

Beleuchtungsmodelle beschreiben die Zusammenhänge zwischen Raumgeometrie, Lichtquellen und Oberflächenbeschaffenheit zur Berechnung der Leuchtdichte in einem Punkt der darzustellenden Szene. Beleuchtungsalgorithmen berechnen aus der Leuchtdichte einiger ausgewählter Punkte die darzustellende Farbe aller Bildpunkte. Die Kombination von Beleuchtungsmodell und -algorithmus kennzeichnet spezielle Beleuchtungsverfahren.

7.11.1 Lichtquellen

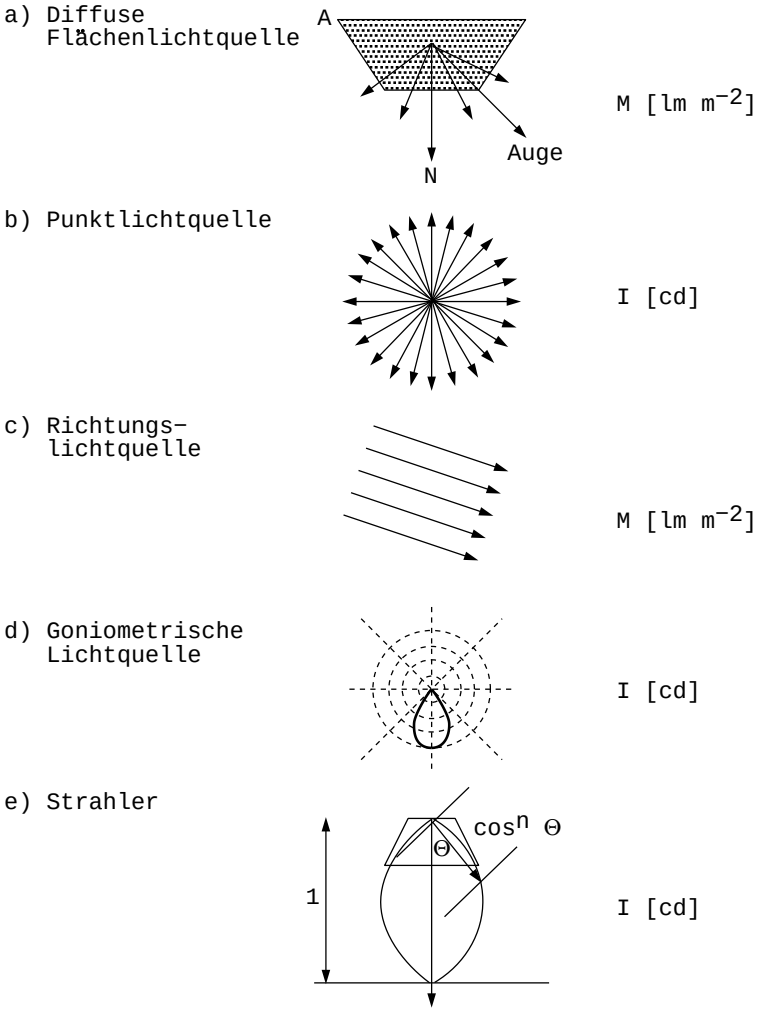


Abbildung 7.40: Zusammenstellung einiger einfacher Lichtquellen

Im folgenden werden zuerst die wichtigsten Lichtquellen vorgestellt, die Bild 7.40 in der Übersicht zeigt.
(Die Abhängigkeit von λ wird nur dort ausgeschrieben, wo es unbedingt zum Verständnis erforderlich ist. Ansonsten sind die Gleichungen immer als Vektorgleichungen für die drei Komponenten R,G,B zu verstehen.)

7.11.1.1 Umgebungslicht (ambientes Licht)

Diesen Begriff kennt die Physik nicht. Dennoch wird in allen Beleuchtungsverfahren eine ambiente Komponente berücksichtigt. Sie dient dazu, alle vom Modell nicht erfaßten, jedoch in Wirklichkeit vorhandenen, indirekten Beleuchtungen zu berücksichtigen. Damit werden auch nicht direkt beleuchtete Szenenteile sichtbar. Das ambiente Licht fällt auf allen Flächen der Szene mit gleicher Stärke ein und wird deshalb durch die Beleuchtungsstärke $E_a(\lambda)$ in Lux (lx) charakterisiert. Die ambiente Leuchtdichte (in der Literatur meist Intensität genannt) ergibt sich entsprechend (7.24) zu

$$L_{amb}(\lambda) = \rho_a(\lambda)\, E_a(\lambda), \tag{7.46}$$

wobei ρ_a der ambiente Reflexionsfaktor ist.
Mit dem dimensionslosen Reflexionsgrad $r_a(\lambda) = \rho_a(\lambda)\pi$ (vgl. Übungsaufgabe 2) erhält man unter der Annahme diffuser Reflexion

$$L_{amb}(\lambda) = r_a(\lambda)\frac{E_a(\lambda)}{\pi}, \;\; 0 \leq r_a \leq 1. \tag{7.47}$$

7.11.1.2 Diffuse Flächenlichtquelle

Diese Lichtquelle spielt für globale Beleuchtungsverfahren, bei denen alle beleuchteten und reflektierenden Flächen als Sender berücksichtigt werden, eine große Rolle (siehe Radiosity-Verfahren). Sie wird durch die Fläche A, die Normale N und die spezifische Lichtausstrahlung $M(\lambda)$ in Lumen/Quadratmeter (lm/m^2) angegeben. Da die Fläche definitionsgemäß diffus strahlt, ist ihre Leuchtdichte durch (7.19)

$$L = \frac{M}{\pi} \tag{7.48}$$

und die Lichtstärke durch (7.20) gegeben:

$$dI = \frac{M}{\pi} \cdot dA \cdot cos\vartheta, \tag{7.49}$$

wobei ϑ der Winkel zwischen N und der Richtung zum betrachteten Punkt ist. Ist die Ausdehnung der Fläche klein im Vergleich zum Abstand des Punktes von der Fläche, so ist

$$I = \frac{M}{\pi} \cdot A \cdot cos\vartheta, \tag{7.50}$$

wobei die Flächennormale im Mittelpunkt der Fläche angenommen wird.

7.11.1.3 Punktlichtquelle

Die Punktlichtquelle wird meist als in alle Richtungen gleichmäßig sendend (isotrop) angenommen. Neben der Lage im Raum wird sie durch die Lichtstärke $I(\lambda)$

in Candela gekennzeichnet. Im Abstand R erzielt diese Lichtquelle nach (7.7) eine Beleuchtungsstärke

$$E = \frac{I}{R^2} \cdot cos\vartheta,$$

wobei ϑ der Lichteinfallswinkel (zwischen Flächennormaler und Lichtausbreitungsrichtung) ist. Ist die Punktlichtquelle weit entfernt, so verhält sie sich wie eine Richtungslichtquelle.

7.11.1.4 Richtungslichtquelle

Die Richtungslichtquelle ist durch die Lichtausbreitungsrichtung $\vec{V}_L$ und die spezifische Lichtausstrahlung $M(\lambda)$ gegeben. Alle Lichtstrahlen treffen mit derselben Richtung $\vec{V}_L$ in der Szene auf, wodurch eine weit entfernte Lichtquelle, wie z.B. die Sonne, modelliert werden kann.

7.11.1.5 Goniometrische Lichtquelle

Bei dieser Lichtquelle wird die Ausbreitungscharakteristik in einem Diagramm beschrieben, das die Größe der Lichtstärke als Funktion des Winkels zur Hauptausbreitungsrichtung in einer Tabelle angibt. Zur Ermittlung der Lichtstärke in eine gewünschte Richtung muß unter Umständen zwischen Tabellenwerten interpoliert werden.

7.11.1.6 Strahler

Beim Strahler (engl. spot) wird die Lichtausbreitung der Quelle auf einen bestimmten Raumwinkel (Lichtkegel) beschränkt. Der Abfall der Lichtstärke vom größten Wert in Richtung der Symmetrieachse (vgl. Bild 7.40) zum Rand wird durch folgendes Gesetz bestimmt:

$$I = I_0 \cdot cos^n \vartheta. \tag{7.51}$$

Der Exponent n bestimmt dabei die Bündelung des Lichts.

7.11.2 Lokale (empirische) Beleuchtungsmodelle

Lokale Beleuchtungsmodelle beschreiben die in einem Punkt der Objektoberfläche wahrnehmbare Leuchtdichte. Dabei wird das Zusammenwirken von einfallendem Licht aus den Lichtquellen und dem Reflexionsverhalten der Objektflächen ausgewertet. Sekundäre Effekte, wie der Strahlungsaustausch benachbarter Flächen, werden nicht berücksichtigt.

Lokale Modelle wurden Anfang der siebziger Jahre entwickelt. Sie sollten mit minimalem Aufwand möglichst realitätsnahe Flächengraphik erzeugen. Sie basieren deshalb auch nicht auf strengen physikalischen Gesetzmäßigkeiten.

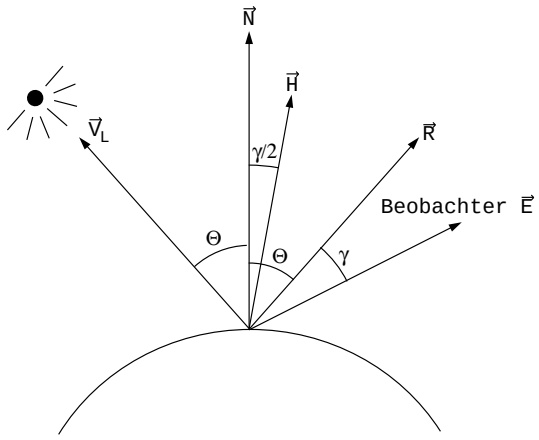


Abbildung 7.41: Geometrie der Beleuchtungsmodelle

Im Abschnitt 2.2.3 'Strahlungsaustausch zwischen Oberflächen' wurde gezeigt (vgl. Bild 7.27), daß neben dem Umgebungslicht noch drei Reflexionsarten zu berücksichtigen sind: Ideal diffus, ideal spiegelnd und gerichtet diffus. Im Falle durchsichtiger Objekte kommt unter Umständen noch ein transparenter Anteil hinzu.

Bild 7.41 zeigt die im folgenden verwendeten Vektoren zur Beschreibung der Beleuchtungsgeometrie in einem Punkt der Objektoberfläche. Alle Vektoren sind auf die Länge 1 normiert. (Dies muß auch nach geometrischen Transformationen gewährleistet sein.)

7.11.2.1 Ambientes Licht

(7.46) liefert mit (7.7) und (7.19):

$$L_{amb} = r_a \cdot \frac{E_a}{\pi} = r_a \cdot L. \tag{7.52}$$

(Alle Größen können eine Funktion der Wellenlänge sein!)

Dabei haben wir folgende Bezeichnungen benutzt:

L_{amb}	ambiente Leuchtdichte,
E_a	Beleuchtungsstärke auf Grund des Umgebungslichts,
r_a	ambienter Reflexionsgrad,
L	Leuchtdichte des einfallenden Lichts.

7.11.2.2 Ideal diffus reflektiertes Licht

Hier wird das Lambert'sche Cosinusgesetz (vgl. (7.20) und Bild 7.25) angewendet. Die Leuchtdichte hängt vom Einfallswinkel ϑ (vgl. Bild 7.41) ab, ist aber

unabhängig vom Betrachtungswinkel

$$\begin{aligned} L_{diff} &= \begin{cases} r_d \cdot L \cdot cos\vartheta, & |\vartheta| < 90° \\ 0 \text{ sonst} \end{cases} \\ &= \begin{cases} r_d \cdot L \cdot (\vec{N} \cdot \vec{V}_L), \text{ falls } (\vec{N} \cdot \vec{V}_L) > 0 \\ 0 \text{ sonst} \end{cases} \end{aligned} \tag{7.53}$$

Dabei haben wir folgende Bezeichnungen benutzt:

L_{diff}	diffus reflektierte Leuchtdichte,
r_d	diffuser Reflexionsgrad $0 \leq r_d \leq 1$,
L	Leuchtdichte des einfallenden Lichts,
ϑ	Winkel zwischen Flächennormale und Vektor zur Lichtquelle,
$\vec{N}$	Flächennormale (vgl. Bild 7.41),
$\vec{V}_L$	Vektor zur Lichtquelle (vgl. Bild 7.41).

7.11.2.3 Gerichtet diffus reflektiertes Licht (Phong Modell)

In (2.29) wurde bereits das auf Phong zurückgehende Modell dargestellt:

$$\begin{aligned} L_{spec} &= \begin{cases} r_s \cdot L \cdot \cos^m \gamma, \text{ falls } |\gamma| < 90° \\ 0 \text{ sonst} \end{cases} \\ &= \begin{cases} r_s \cdot L \cdot (\vec{R} \cdot \vec{E})^m, \text{ falls } (\vec{R} \cdot \vec{E}) > 0 \\ 0 \text{ sonst} \end{cases} \end{aligned} \tag{7.54}$$

oder zur Vermeidung der Berechnung von $\vec{R}$:

$$L'_{spec} = \begin{cases} r_s \cdot L \cdot (\vec{H} \cdot \vec{N})^m, \text{ falls } (\vec{H} \cdot \vec{N}) > 0 \\ 0 \text{ sonst} \end{cases} . \tag{7.55}$$

Dabei wird anstatt des Winkels γ der Winkel zwischen der Flächennormalen $\vec{N}$ und dem Halbvektor $\vec{H}$ zwischen Lichtquellenrichtung und Beobachtungsrichtung benutzt, der gleich $\frac{\gamma}{2}$ ist.

Es werden die folgenden Bezeichnungen benutzt:

$\vec{H}$	Einheitsvektor mit $\vec{H} = \frac{\vec{E}+\vec{V}_L}{\lvert\vec{E}+\vec{V}_L\rvert}$,
$\vec{E}$	Einheitsvektor in Richtung des Beobachters,
$\vec{R}$	Einheitsvektor in Reflexionsrichtung,
L	Leuchtdichte des einfallenden Lichts,
r_s	spiegelnder Reflexionsgrad, wellenlängenabhängig und $0 \leq r_s \leq 1$,
L_{spec}	spekulare Leuchtdichte,
m	gibt die Bündelung des reflektierten Lichts an.

Da der Winkel zwischen $\vec{H}$ und $\vec{N}$ $\gamma/2$ beträgt (vgl. Bild 7.41), liefern die beiden Formulierungen des Phong-Modells bei gleicher Wahl von m verschiedene

Ergebnisse L_{spec} bzw. L'_{spec}. Dies kann aber durch eine geeignete Wahl von m kompensiert werden.

Bild 7.42 zeigt die Zusammenfassung von ambientem und diffusem Anteil durch den (richtungsunabhängigen) Kreis und die Überlagerung des spekularen Anteils für verschiedene Werte von m.

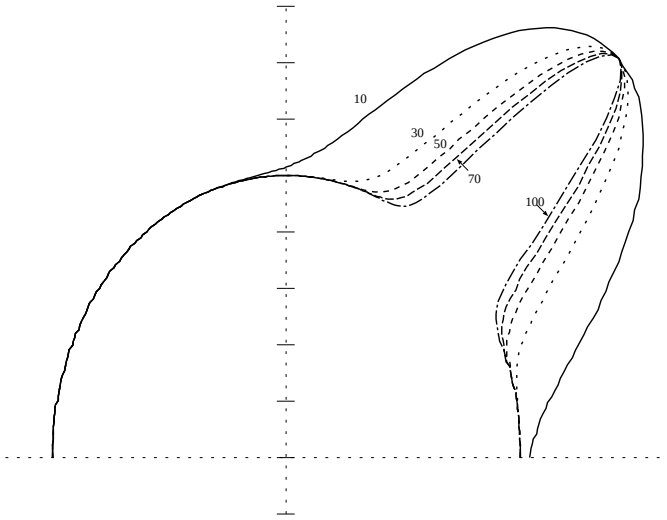


Abbildung 7.42: Überlagerung der einzelnen Komponenten des reflektierten Lichts für verschiedene Werte von m

Da r_s unabhängig von λ ist, hat L_{spec} die gleiche spektrale Zusammensetzung wie das einfallende Licht. Bei großem m wird dieser Anteil in Richtung R konzentriert und entspricht im Grenzfall der ideal spiegelnden Reflexion. Geometrische Abhängigkeiten, wie die Entfernung zwischen Lichtquelle und Objekt, werden nicht extra modelliert, sondern in die Leuchtdichte des einfallenden Lichts integriert. Nach dem Phong'schen Modell erhält man dann bei n Lichtquellen die Leuchtdichte in einem Punkt aus:

$$L_{Phong} = L_{amb} + \sum_{i=1}^{n} (L_{diff,i} + L_{spec,i}). \tag{7.56}$$

Eine Gleichung mit stärkerem physikalischen Bezug erhält man durch Verwendung des Reflexionsfaktors. Allerdings muß dann die einfallende Beleuchtungsstärke anstelle der Leuchtdichte berücksichtigt werden.

Wenn Sie die PDF-Version des Kurses bearbeiten, können Sie jetzt das Interaktionselement 'shadingmodel' anwählen.

7.11.3 Lokale Beleuchtungsalgorithmen

Im folgenden gehen wir davon aus, daß Objekte in triangulierter Form vorliegen. Als Beispiel dient uns der Utah teapot (vgl. Bild 7.43) [Bli87].

7.11.3.1 Konstante Beleuchtung

Der einfachste Algorithmus zur Beleuchtung polygonal begrenzter Körper belegt ein ganzes Polygon gleichmäßig mit einer Farbe, die aber von der Beleuchtung der Szene abhängt (flat shading). Das bedeutet, daß nur einmal pro Polygon eine Leuchtdichte zu berechnen ist und die Farbe als konstantes Attribut dieser Fläche mitgeführt werden kann.

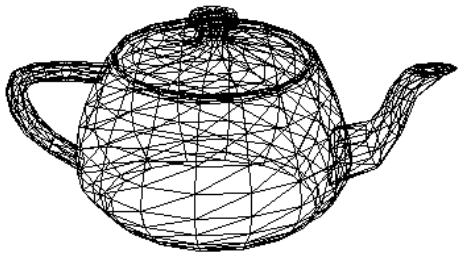


Abbildung 7.43: Trianguliertes Modell des teapots

Algorithmus 1

for jedes Polygon **do**

1. Berechne im Objektraum das Beleuchtungsmodell an einem Punkt des Polygons;
2. Projiziere das Polygon in die Bildebene;
3. Fülle das Polygon mit der in Schritt 1 berechneten Farbe.

endfor

Der Aufwand des Algorithmus hängt allerdings stark von der Stelle ab, an der er in die Bildgenerierungspipeline eingefügt wird. Kann er so integriert werden, daß die Berechnung des Beleuchtungsmodells nur für die sichtbaren Polygone durchgeführt wird und die Farbe anschließend als Attribut mitgeführt wird, so ist der Berechnungsaufwand linear abhängig von der Anzahl der sichtbaren Polygone.

Die resultierende Darstellung der Körper ist nicht realistisch (vgl. Bild 7.44). Obwohl ein Tiefeneindruck entsteht, sind die Polygone deutlich voneinander unterscheidbar. Neben diesem Nachteil bestehen Einschränkungen in der Auswahl des Beleuchtungsmodells. Durch die konstante Färbung eines Polygons ist die Verwendung von Modellen für die spiegelnde Reflexion nicht sinnvoll.

Entsprechen die Polygone jeweils nur wenigen Pixeln auf dem Bildschirm, so kann diese Art der Beleuchtungsberechnung dennoch durchaus sinnvoll sein, wie von Cook in [CCC87] gezeigt wurde. Die Polygone werden dann als Mikropolygone bezeichnet.

Abbildung 7.44: Teapot mit flat shading

7.11.3.2 Gouraud–Algorithmus

Der sogenannte Gouraud–Algorithmus wurde 1971 von Henri Gouraud in [Gou71] vorgestellt und basiert auf dem Scan–line–Algorithmus von Watkins (vgl. [Wat70]). Es handelt sich dabei um die Interpolation der an den Ecken eines Dreiecks berechneten Leuchtdichten. Diese Werte werden entlang der Kanten des Dreiecks und anschließend entlang einer Rasterzeile interpoliert, was sich in den Vorgang der Rasterung integrieren läßt. Dadurch entsteht ein inkrementelles Verfahren, das sehr effizient realisiert werden kann. Weil die Leuchtdichte innerhalb des Dreiecks aus den berechneten Leuchtdichten an den Eckpunkten ermittelt wird, ist die Berücksichtigung eines spekularen Anteils, der ja definitionsgemäß ein richtungsabhängiges Phänomen ist, nicht sinnvoll. Deshalb wird Gouraud-Shading meist mit diffuser Reflexion assoziiert.

Algorithmus 2

1. Berechne im Objektraum die Normalenvektoren in den Eckpunkten der Polygone;

for jedes Polygon **do**

2. Berechne im Objektraum das Beleuchtungsmodell an allen Eckpunkten des Polygons;

3. Projiziere das Polygon in die Bildebene;

   for alle vom Polygon überdeckten Scanlinien **do**

   4. Berechne die linear interpolierte Leuchtdichte an der linken und rechten Kante des Polygons;

      for jedes Polygonpixel der Scanlinie **do**

      5. Berechne die linear interpolierte Leuchtdichte des Pixels.

      endfor

   endfor

endfor

Bild 7.45 zeigt den teapot mit Gouraud-Shading. Der Algorithmus sei an einem Beispiel verdeutlicht.

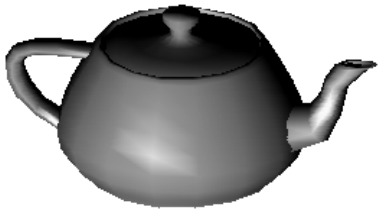


Abbildung 7.45: Teapot mit Gouraud-Shading

Gegeben sei ein Dreieck, das durch die Punkte P_1, P_2 sowie P_3 beschrieben ist. Die Normalenvektoren an den Eckpunkten können berechnet werden durch:

- Mittelung der Normalenvektoren der angrenzenden Flächen des Punktes für eine Beleuchtung mit “weichen” Übergängen (smooth shading);
- Mittelung eines Teils der Normalenvektoren der angrenzenden Flächen, wenn eine Kante erscheinen soll;
- Verwendung des (bekannten) Normalenvektors desjenigen Objektes, das durch die Polygone approximiert wird.

Man erkennt, daß für diesen Algorithmus bereits Datenstrukturen in einer bestimmten Weise aufgebaut werden müssen, um beispielsweise schnell auf die Normalenvektoren der angrenzenden Polygone zugreifen zu können.

Mit Hilfe des Beleuchtungsmodells werden die Leuchtdichten L_1, L_2 und L_3 berechnet. In Bild 7.46 wird dann das weitere Vorgehen verdeutlicht.

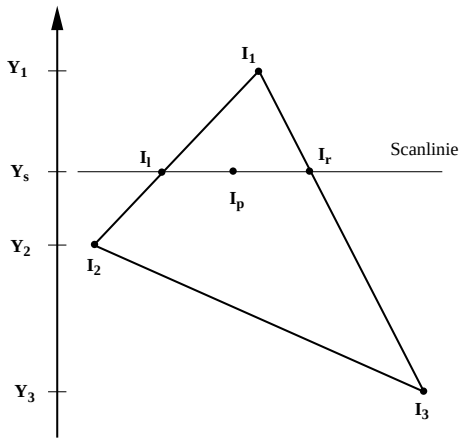


Abbildung 7.46: Interpolationsschema

Für einen beliebigen Punkt L_P innerhalb des betrachteten Dreiecks gelten die Interpolationsregeln

$$\begin{aligned} L_P &= (1-\alpha)\cdot L_l + \alpha \cdot L_r, \\ \alpha &= \frac{X_P - X_l}{X_r - X_l},\ 0 \le \alpha \le 1, \end{aligned}$$

wobei sich die Leuchtdichten L_l und L_r wiederum aus Interpolationen der Leuchtdichten an den Eckpunkten ergeben:

$$\begin{aligned} L_l &= (1-\beta)\cdot L_1 + \beta \cdot L_2 \\ \beta &= \frac{Y_1 - Y_s}{Y_1 - Y_2},\ 0 \le \beta \le 1 \\ L_r &= (1-\gamma)\cdot L_1 + \gamma \cdot L_3 \\ \gamma &= \frac{Y_1 - Y_s}{Y_1 - Y_3},\ 0 \le \gamma \le 1. \end{aligned}$$

Man erkennt, daß ein Berechnungsaufwand entsteht, der sich im günstigsten Fall additiv aus einem in der Anzahl der sichtbaren Polygone linearen und einem in der Anzahl der Pixel des Bildschirms linearen Aufwand zusammensetzt.

Die visuellen Ergebnisse sind befriedigend für diffuse Körper, wenn auch noch die sogenannten Machband–Effekte zu beobachten sind. Als Machband–Effekt werden helle oder dunkle Linien bezeichnet, die an Stellen auftreten, an denen sich die Leuchtdichten schnell ändern bzw. diskontinuierlich sind (bzw. L nicht stetig differenzierbar!).

7.11.3.3 Phong–Algorithmus

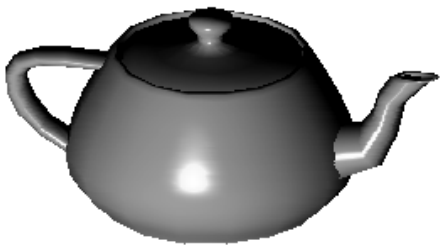


Abbildung 7.47: Phong-Shading mit m=10

Das Phong-Modell erfaßt auch spiegelnde Effekte. Dazu werden die Normalen benötigt. Deshalb werden nicht die Leuchtdichten, sondern die Normalenvektoren interpoliert. Diese Berechnung kann ebenfalls inkrementell sein und integriert in die Rasterung aufgenommen werden. Allerdings muß jeder Normalenvektor vor Auswertung des Beleuchtungsmodells normiert werden. Nach der Interpolation wird in jedem Punkt die Leuchtdichte berechnet.

Algorithmus 3

1. Berechne im Objektraum die Normalenvektoren in den Eckpunkten der Polygone;

for jedes Polygon **do**

2. Projiziere die Eckpunkte des Polygons in die Bildebene;

for alle vom Polygon überdeckten Scanlinien **do**

3. Berechne im Objektraum den linear interpolierten Polygonpunkt und den linear interpolierten Normalenvektor an der linken und rechten Kante des Polygons;

for jedes Polygonpixel der Scanlinie **do**

4. Berechne im Objektraum den linear interpolierten Polygonpunkt und den linear interpolierten Normalenvektor;
5. Normiere den Normalenvektor;
6. Berechne im Objektraum das Beleuchtungsmodell und setze das Pixel mit der berechneten Leuchtdichte.

endfor

endfor

endfor

Es entsteht im günstigsten Fall wiederum ein in der Anzahl der sichtbaren Polygone und in der Anzahl der Pixel sich additiv zusammensetzender linearer Aufwand. Allerdings sind die Konstanten hier wesentlich größer als bei dem Gouraud–Algorithmus, da ja sowohl die Farbe als auch vorher der (normierte) Normalenvektor für jeden Punkt berechnet werden muß. Die Resultate sind realistischer, die Mach–Band–Effekte sind reduziert. Diese Reduktion des Mach–Band–Effektes bedeutet aber nicht, daß der Phong–Algorithmus *in allen Fällen* bessere Ergebnisse liefert als der Gouraud–Algorithmus. Dies wurde von T. Duff demonstriert [Duf79]. Bild 7.47 zeigt den teapot mit Phong-Shading (m=10).

Phong-Shading liefert trotz des nichtphysikalischen Modells in vielen Fällen sehr realistisch wirkende Bilder. Allerdings ist der Aufwand im Vergleich zum Gouraud-Shading sehr groß. Insbesondere die Berechnung der normierten Normalenvektoren ist sehr aufwendig. Deshalb wurde versucht, algorithmische und Hardwarebeschleunigungen zu realisieren. In [PH90] wurde ein Verfahren vorgestellt, das spiegelnde Reflexion (“highlights”) dadurch erfaßt, daß Objektflächen in Bereichen starker Richtungsänderung feiner trianguliert werden. Auf diese Weise wird

das Phong-Shading auf Gouraud-Shading zurückgeführt. Hardware-Entwicklungen haben das Phong-Shading durch direkte Implementierung [Kni93] oder durch Anlegen von Reflektanztafeln [Jac92] echtzeitfähig gemacht.

Wenn Sie die PDF-Version des Kurses bearbeiten, können Sie jetzt das Interaktionselement 'shadingalgo' anwählen.

7.12 Strahlverfolgung (Raytracing)

Im Gegensatz zu den lokalen Beleuchtungsverfahren steht bei den globalen Verfahren nicht die möglichst einfache Realisierung, sondern die bestmögliche, realistische Wiedergabe einer Szene im Vordergrund. Man spricht auch von photorealistischen Darstellungen. Dieses Ziel kann natürlich nur erreicht werden, wenn man die physikalischen Vorgänge so exakt wie möglich modelliert.

In der GDV werden zwei unterschiedliche Verfahren eingesetzt: Raytracing und Radiosity. Im Rahmen dieses Einführungskurses können nur die grundlegenden Prinzipien beider Verfahren beschrieben werden. Viele die Verfahren beschleunigende Varianten werden z.B. im Kurs GDV II behandelt.

Wir folgen der geschichtlichen Entwicklung und betrachten zuerst das Raytracing.

Raytracing simuliert den Prozeß der Lichtausbreitung und arbeitet dabei nach den Gesetzen für ideale Spiegelung und Brechung. Daher ist Raytracing vor allem für Szenen mit hohem spiegelnden und transparenten Flächenanteil gut geeignet. Die Grundidee besteht darin, Lichtstrahlen auf ihrem Weg von der Quelle bis zum Auge zu verfolgen. Zur Vereinfachung werden beim konventionellen Raytracing nur ideal reflektierte und ideal gebrochene Strahlen weiterverfolgt. Da nur wenige Strahlen das Auge erreichen, kehrt man das Verfahren um (Reziprozität der Reflexion) und sendet durch jedes Pixel des Bildschirms einen vom Augpunkt ausgehenden Strahl in die Szene.

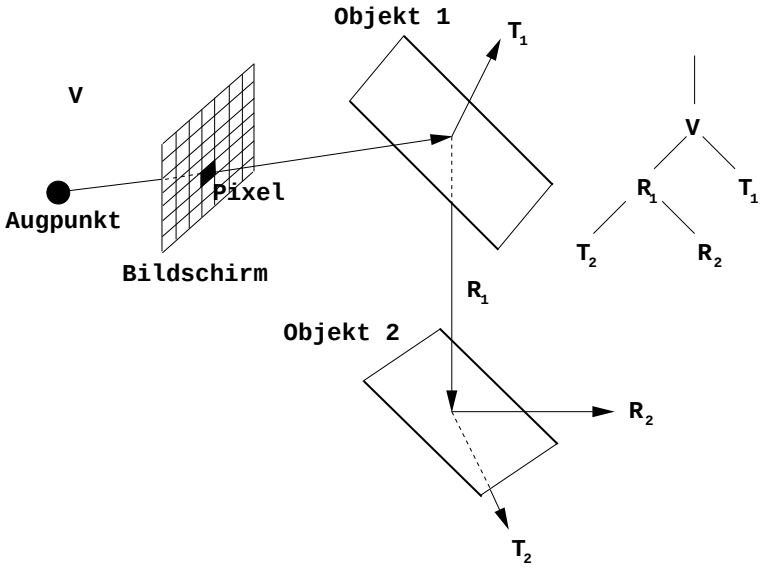


Abbildung 7.48: Geometrie der Strahlverfolgung und Strahlbaum

Trifft der Sehstrahl auf ein Objekt, so wird das lokale Beleuchtungsmodell berech-

net. Anschließend werden zwei neue Strahlen erzeugt, nämlich der reflektierte und der gebrochene (transmittierte) Sehstrahl. Der Leuchtdichtebeitrag dieser beiden Strahlen wird rekursiv berechnet.

Dieser Prozeß bricht ab, wenn eine Lichtquelle getroffen wird, die auf dem Strahl transportierte Energie zu gering wird oder wenn der Sehstrahl die Szene verläßt (vgl. Bild 7.48). Aus praktischen Erwägungen wird man eine Obergrenze für die Rekursionstiefe festlegen. Mit Hilfe dieses Algorithmus wird das Verdeckungsproblem implizit gelöst. Die Schattenberechnung wird durchgeführt, indem man von den Auftreffpunkten des Sehstrahls sogenannte Schattenstrahlen zu den Lichtquellen der Szene sendet. Nur wenn kein undurchsichtiges Objekt zwischen einer Lichtquelle und dem Auftreffpunkt liegt, trägt sie zur Beleuchtung bei.

Der Algorithmus läuft dann folgendermaßen ab:

```
für jedes Pixel des Bildschirms
{
    1. bestimme nächstliegenden Schnittpunkt des Seh-
           strahls mit einem Objekt der Szene
    2. berechne ideal reflektierten Lichtstrahl
→   3. berechne die Leuchtdichte aus dieser Richtung
    4. berechne ideal gebrochenen Lichtstrahl
→   5. berechne die Leuchtdichte aus dieser Richtung
    6. werte das Phongbeleuchtungsmodell im Schnitt-
          punkt aus und addiere die gewichteten Leucht-
          dichten der Sekundärstrahlen
}
```

Die mit → gekennzeichneten Befehle entsprechen dem Wiederaufruf des inneren Teils des Algorithmus – es handelt sich also um einen rekursiven Prozeß. Teilweise wird der Algorithmus auch als zweistufig beschrieben: In der ersten Phase wird der den Rekursionsstufen entsprechende Baum mit Hilfe der Schnittpunktberechnungen aufgebaut, und in einer zweiten Phase wird der Baum traversiert, wobei die Intensitäten berechnet werden. Als vernünftige maximale Baumtiefe gilt 5.

Wenn Sie die PDF-Version des Kurses bearbeiten, können Sie jetzt das Interaktionselement 'rayshow' anwählen.

Der Aufwand für die Berechnung des Schnittes eines Strahls mit einem Objekt kann sehr groß werden, je nachdem, welches Objekt man betrachtet. Whitted [Whi79] stellte nach Laufzeitmessungen fest, daß 75% der Zeit für die Schnittpunktberechnungen verwendet wurde und 12% für die Berechnung des Beleuchtungsmodells.

Die Anzahl der Schnittpunktberechnungen ist dabei proportional zum Produkt aus der Anzahl der Strahlen und der Anzahl der Objekte und wächst exponentiell mit der Anzahl der Rekursionsstufen.

Zum Beleuchtungsmodell nach Phong (vgl. Abschnitt 8.5.3 'Lokale Beleuchtungsalgorithmen'), das den lokalen Anteil modelliert, kommen noch der ideal spiegelnde und der transmittierte Anteil hinzu:

$$L_{ges} = L_{Phong} + r_r L_r + r_t L_t, \tag{7.57}$$

wobei

L_r *die Leuchtdichte auf dem reflektierten Strahl,*
L_t *die Leuchtdichte auf dem transmittierten Strahl,*
r_r *der Reflexionsgrad für die Idealreflexion,*
r_t *der Reflexionsgrad für die Idealtransmission ist.*

Dabei ist zu berücksichtigen, daß der reflektierte Strahl R die an der Tangentialebene des Objekts im Schnittpunkt reflektierte Blickrichtung ist.

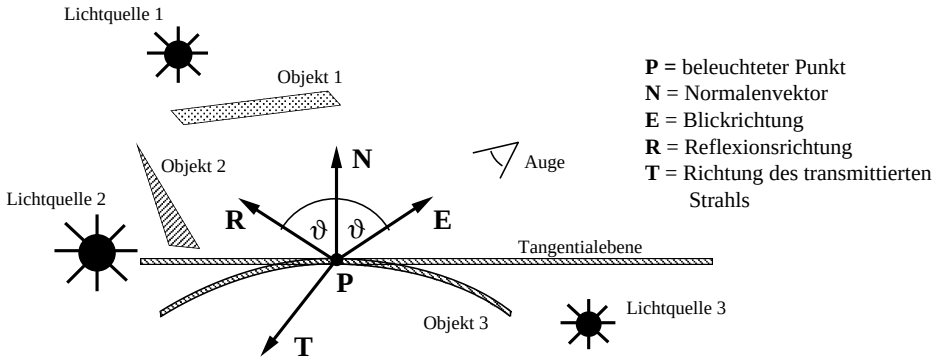


Abbildung 7.49: Beleuchtungsgeometrie des Raytracings

Beschleunigungstechniken
Es gibt zwei Möglichkeiten zur Beschleunigung des Raytracing-Verfahrens:

1. Verringerung der durchschnittlichen Kosten der Schnittpunktberechnung zwischen einem Strahl und einem Szenenobjekt, und

2. Verringerung der Gesamtanzahl der Strahl-Objekt-Schnittpunkttests.

Beide Möglichkeiten müssen in einer Implementierung genutzt werden. Die Minimierung der Kosten der Schnittpunktberechnung kann nur objektspezifisch geleistet werden und ist deshalb nicht szenenunabhängig.

Haines [Hai89] und Hanrahan [Han89] geben eine sehr gute Einführung in effiziente Schnittpunktalgorithmen für häufig vorkommende Objekte.

Beim konventionellen Raytracing wird jeder Primärstrahl sowie jeder reflektierte und gebrochene Strahl mit allen Objekten der Szene geschnitten, um den nächstliegenden Schnittpunkt zu finden, falls dieser existiert. Dasselbe gilt für Schattenstrahlen, die den Schnittpunkt des Strahls mit dem Objekt auf Sichtbarkeit für die

Lichtquellen überprüfen. Nur hier terminiert die Schnittpunktsuche beim Feststellen des ersten Schnittpunkts mit einem undurchsichtigen Objekt. Dieser Ansatz ist extrem rechenaufwendig und kann auf mehrere Arten beschleunigt werden (wegen Details sei auf den Kurs GDV II (K1693) verwiesen):

- Die Raumunterteilung ist eine Beschleunigungstechnik, die die Objektkohärenz ausnutzt. In einem Vorverarbeitungsschritt wird der dreidimensionale Objektraum in nichtüberlappende Unterräume unterteilt (sog. Voxel = Volumenelement). Bei der Strahlverfolgung bestimmt dann ein Traversierungsalgorithmus der Reihe nach die vom Strahl durchlaufenen Unterräume. Schnittpunktberechnungen müssen nur noch mit denjenigen Objekten durchgeführt werden, die zumindest teilweise in diesen Unterräumen liegen. Wurde ein Schnittpunkt gefunden und ist dieser Schnittpunkt der nächste im aktuellen Voxel, dann kann die Traversierung stoppen. Man unterscheidet
  - die Raumaufteilung mit regelmäßigen Gittern, bei der der Objektraum in endlich viele gleich große achsenparallele Voxel aufgeteilt wird,
  - die Raumunterteilung mit Octrees, bei der ein Voxel immer dann, wenn es zu komplex ist, d.h. zu viele Objektprimitive enthält, in acht gleichgroße Subvoxel unterteilt wird.

- Bei einer Szenenunterteilung mit hierarischen Bäumen werden Objekte einer Teilszene zu Boundary Volumes, d.h. Körpern, die diese Objekte vollständig umhüllen, zusammengefaßt, um anschließend mit anderen Boundary Volumes zu größeren Boundary Volumes zusammengefaßt zu werden. Der Schnitt eines Strahls mit einem Boundary Volume ist eine notwendige, aber keine hinreichende Bedingung dafür, daß die enthaltene Teilszene vom Strahl getroffen wird.

- Um das wiederholte Durchführen derselben Schnittpunktberechnungen zu vermeiden, wird jedem Objekt der Szene eine sogenannte *Mailbox* und jedem Strahl eine eindeutige Strahl-ID zugeordnet. Nach jedem Schnittpunkttest wird das Ergebnis der Berechnung zusammen mit der Strahl-ID in der Mailbox des Objektes gespeichert.

- Das Verfolgen von Schattenstrahlen ist einfacher als das Verfolgen von Sehstrahlen, da es nicht notwendig ist, den nächsten Schnittpunkt des Strahls mit den Primitiven der Szene zu finden. Das Wissen, daß mindestens ein Schnittpunkt des Strahls mit einem undurchsichtigen Objekt existiert, ist bereits ausreichend. Aus diesem Grund verwaltet jede Lichtquelle einen Schatten-Cache [HG86], [PJ91], der das Resultat der Strahlverfolgung des letzten Schattenstrahls speichert. War dieser Schattenstrahl auf seinem Weg von der Lichtquelle zum Objektschnittpunkt auf ein undurchsichtiges Objekt gestoßen, so wird ein Pointer auf dieses schattenwerfende Objekt im Schatten-Cache gespeichert. Ansonsten wird ein Null-Pointer gespeichert. Bevor der nächste Schattenstrahl verfolgt wird, führt man einen Schnittpunkttest mit dem durch den Schatten-Cache referenzierten Objekt durch. Nur wenn dieses Schnittestergebnis negativ ist, wird der Schattenstrahl explizit durch die Szene verfolgt.

- Eine ad-hoc-Technik zur Vermeidung unnötiger Strahlverfolgungen besteht darin, (reflektierte/ transmittierte/...) Sekundärstrahlen nicht weiter zu verfolgen, wenn ihr Beitrag zur Pixelfarbe zu klein wird [HG83].

- Selbst wenn die Szenengeometrie fest vorgegeben ist, sind bis zu einem zufriedenstellenden Ergebnis häufig viele Bildberechnungen notwendig. Der Grund hierfür ist, daß auch viele andere nichtgeometrische Parameter den visuellen Eindruck der Szene beeinflussen: Kameraparameter, Anzahl, Positionen und Intensitäten der Lichtquellen und zahlreiche Materialparameter für jedes Primitiv der Szene, die vom Benutzer häufig nach einer Berechnung noch korrigiert werden müssen.

  Eine schnelle Previewing-Technik zur Beschleunigung der Visualisierung besteht darin, homogene Regionen in der Bildebene zu detektieren und zu interpolieren. Die Technik basiert auf einer *Divide–and–Conquer*–Strategie und führt die folgenden Schritte aus:

  1. Unterteile die Bildebene in Regionen von je D$x \times$ Dy Pixeln.
  2. Bestimme die Intensitäten der vier Eckpixel durch Emittieren von Strahlen.
  3. Sind die Pixel direkt benachbart, so gib Pixelfarben aus und stoppe.
  4. Ist die Intensität zwischen den Eckpixeln kleiner als ein Schwellwert, so fülle die Region durch bilineare Interpolation der vier Eckintensitäten und stoppe.
  5. Ansonsten unterteile die Region in zwei oder vier Unterregionen, berechne alle noch nicht berechneten Eckintensitäten durch das Raytracing-Verfahren und wende die Schritte 3 bis 5 rekursiv an.

Wenn Sie die PDF-Version des Kurses bearbeiten, können Sie jetzt das Interaktionselement 'shadowcache' anwählen.

7.12.1 Bewertung

Raytracing ist insbesondere für Szenen mit hohem spiegelnden und transparenten Flächenanteil geeignet und liefert hier gute Ergebnisse. Im folgenden werden die Vor- und Nachteile das Raytracing noch einmal zusammengestellt.

+ Die Szenenbeschreibung kann beliebig komplexe Objekte enthalten. Die einzige Bedingung besteht darin, daß Objektnormale und Schnittpunkte mit Strahlen berechenbar sind. Es besteht keine Notwendigkeit, alle Objekte durch Polygone zu approximieren.

+ Die Berechnung verdeckter Flächen, Schatten, Reflexionen und Transparenzen sind ein inhärenter Teil des Raytracing-Algorithmus.

+ Explizite perspektivische Transformationen der Objekte und Clipping-Berechnungen sind nicht notwendig.

+ Objekte dürfen sich gegenseitig durchdringen. Schnitte zwischen Objekten brauchen nicht berechnet zu werden.

+ Das Beleuchtungsmodell muß nur in sichtbaren Objektpunkten berechnet werden.

– Die Abtastung der Szene mit einem Strahl pro Pixel erzeugt in der Regel Aliasing, das mit Supersampling oder stochastischem Abtasten gemildert [Coo86], [Pur86], [Gla95] werden kann.

– Schnittpunktsberechnungen werden in der Regel in Floating-Point-Arithmetik durchgeführt. Da unter Umständen viele Millionen Strahlen verfolgt werden müssen, ist der Rechenaufwand sehr groß.

– Schatten haben stets scharfe Grenzen. Weiche Halbschatten sind nur durch rechenaufwendigere, stochastische Raytracing-Verfahren darstellbar.

– Außerdem müssen Schatten bei jeder Änderung der Kameraparameter neu berechnet werden, obwohl diese nur von den Lichtquellen und den Objekten der Szene abhängig sind.

– Globales diffuses Licht wird beim konventionellen Raytracing-Verfahren nicht berücksichtigt.

7.13 Das Radiosity-Verfahren

7.13.1 Einführung

Das *Radiosity-Verfahren* [CW93] wurde von Goral, Greenberg et al. [GTGB84] im Jahre 1983 vorgestellt und beruht auf der Energieerhaltung. Zwischen dem durch die Lichtquellen zugeführten Lichtstrom und der Absorption durch die Oberflächen besteht ein Gleichgewicht, das zu einer zeitunabhängigen Beleuchtungsstärke in der Szene führt. Ebenso wie beim Strahlverfolgungsverfahren handelt es sich um ein *globales* Beleuchtungsverfahren, allerdings für ideal diffus reflektierende Oberflächen. Global heißt, daß nicht nur die Wechselwirkung der Oberflächen mit den Lichtquellen, sondern darüber hinaus die Wechselwirkung der Oberflächen untereinander berücksichtigt wird. Trotz der Einschränkung auf diffus reflektierende Oberflächen bietet sich für das Radiosity-Verfahren doch ein weites Anwendungsgebiet. So sind in unserer natürlichen Umgebung weitgehend diffus reflektierende Oberflächen zu sehen, wogegen spiegelnde Oberflächen eher die Ausnahme sind. Vor allem für die Lichtverteilung in Gebäuden ist das Radiosity-Verfahren wie kein anderes Beleuchtungsverfahren geeignet.

Ein großer Vorteil diffuser Reflexion ist, daß die Leuchtdichte einer Fläche unabhängig von der Beobachtungsrichtung ist. Wurde eine Szene mit dem Radiosity-Verfahren berechnet, so ist es möglich, die Szene aus allen Richtungen zu betrachten, ja sogar durch die Szene hindurchzugehen, ohne sie neu berechnen zu müssen. Durch diese Eigenschaft der Betrachtungsunabhängigkeit wird auch die hohe Rechenzeit für das Verfahren akzeptabel.

Das Radiosity-Verfahren wird als *physikalisches* Beleuchtungsverfahren bezeichnet, obwohl auch im Radiosity-Verfahren von der Fülle der physikalischen Effek-

te, die bei der Lichtausbreitung eine Rolle spielen, nur der Mechanismus der rein diffusen Reflexion berücksichtigt wird. Die rein diffuse Reflexion kann aber dafür mit gewünschter Genauigkeit nachgebildet werden. Berechnet wird im Radiosity-Verfahren die Beleuchtungsstärke (gemessen in Lux) einer jeden Fläche. Es wird also eine photometrische Größe berechnet, für die es eine genaue Meßvorschrift gibt. Die Ergebnisse des Radiosity-Verfahrens konnten deshalb auch im Experiment nachgeprüft und verifiziert werden [MRC+86].

Da die Aussendung des Lichts von der Richtung unabhängig ist, ist die entscheidende photometrische Größe die spezifische Lichtausstrahlung M_v. In der englischsprachigen Literatur wird diese Größe als *Radiosity* bezeichnet (daher der Name des Verfahrens) und oft im Widerspruch zur Tabelle 2.1 in Abschnitt 2.2.3 'Strahlungsaustausch zwischen Oberflächen' mit B [2] bezeichnet. Der Lichtstrom einer infinitesimalen Fläche dA_j setzt sich aus dem Eigenlichtstrom φ_{Ej} (E für Emission) und dem reflektierten Anteil des einfallenden Lichtstroms φ_{Ij} (I für Incident) zusammen:

$$\varphi_j = \varphi_{Ej} + \rho_j \varphi_{Ij}. \tag{7.58}$$

(ρ_j ist hier der Reflexionsgrad.)

Drückt man den Lichtstrom durch Radiosities aus, so erhält man:

$$B_{dA_j}\, dA_j = E_{dA_j}\, dA_j + \rho_j \varphi_{Ij}. \tag{7.59}$$

Der einfallende Lichtstrom φ_{Ij} resultiert wiederum aus der Lichtausstrahlung aller anderen Oberflächen der Szene:

$$\varphi_{Ij} = \int_{\left(\substack{\cup A_i \\ i \neq j}\right)} F_{dA_i\text{-}dA_j} B_{dA_i}\, dA_i. \tag{7.60}$$

Die Integration geht hier über alle Oberflächen der Szene, die Fläche dA_j natürlich ausgenommen, da es keine Bestrahlung einer infinitesimalen Fläche auf sich selbst gibt. Die Größe $F_{dA_i\text{-}dA_j}$ nennt man den *Formfaktor* zwischen dem Flächenelement dA_i und dem Flächenelement dA_j. Der Formfaktor ist eine dimensionslose Zahl zwischen 0 und 1. Er gibt an, wieviel Lichtstrom des Flächenelements dA_i auf dem Flächenelement dA_j ankommt, und wird folgendermaßen definiert:

$$F_{dA_i\text{-}dA_j} = \frac{d^2\varphi_{dA_i \to dA_j}}{d\varphi_{dA_i}}. \tag{7.61}$$

Er entspricht dem Verhältnis des Lichtstroms $d^2\varphi_{dA_i \to dA_j}$, der von dA_i abgestrahlt wird und auf dA_j ankommt, zum gesamten Lichtstrom $d\varphi_{dA_i}$ des Flächenelements dA_i.

Setzt man Gleichung 7.60 in Gleichung 7.59 ein und dividiert durch dA_j, so bekommt man eine Gleichung für die Radiosity von dA_j:

$$B_{dA_j} = E_{dA_j} + \rho_j \frac{1}{dA_j} \int_{\left(\substack{\cup A_i \\ i \neq j}\right)} F_{dA_i\text{-}dA_j} B_{dA_i}\, dA_i. \tag{7.62}$$

[2] In der Literatur werden häufig Begriffe der Radiometrie und der Photometrie gemischt verwendet. Wir folgen, mit Ausnahme des Begriffs Radiosity, konsequent der photometrischen Terminologie.

Diese Gleichung stellt eine Vereinfachung der ‘rendering equation’ von Kajiya [Kaj86] dar. Die Vereinfachung liegt darin, daß nur die rein diffusen Reflexionen berücksichtigt worden sind. Beim Radiosity-Verfahren kommt noch eine zweite Vereinfachung hinzu: Die Oberflächen der Objekte werden durch eine endliche Zahl kleiner, ebener Flächen, sogenannter Patches, beschrieben. Jedes Patch hat einen einheitlichen Reflexionsgrad und eine einheitliche Radiosity. Die Radiosity einer Fläche j wird dann beschrieben durch

$$B_j = E_j + \rho_j H \tag{7.63}$$

mit

$$H = \sum_{i=1}^{N} F_{ij} B_i \frac{A_i}{A_j} , \tag{7.64}$$

wobei N die Anzahl der Flächen in der Szene ist. Analog zur obigen Definition beschreibt der Formfaktor F_{ij} den Bruchteil des auf der Fläche j ankommenden Lichtstroms der Gesamtausstrahlung der Fläche i. In den Formfaktoren ist die gesamte geometrische Information der Szene enthalten.

Aus den beiden obigen Gleichungen erhält man durch Einsetzen die Grundgleichung des Radiosity-Verfahrens:

$$B_j = E_j + \rho_j \sum_{i=1}^{N} F_{ij} B_i \frac{A_i}{A_j}. \tag{7.65}$$

Aus der Integralgleichung 7.62 ist nun ein lineares Gleichungssystem geworden, das sehr viel einfacher zu lösen ist. Vor der Lösung des Gleichungssystems müssen die Formfaktoren bestimmt werden. Es wird sich zeigen, daß dies ungleich aufwendiger ist als die Lösung des Gleichungssystems.

Der zentrale Begriff im Radiosity-Verfahren ist der des *Formfaktors*. Formfaktoren sind jedoch keine Erfindung der GDV. Sie sind bereits aus der Wärmetechnik gut bekannt. Ausführlich besprochen werden sie in dem Buch *Thermal Radiation Heat Transfer* von Robert Siegel [Sie81][3]. Hier soll gezeigt werden, wie sie mit den in Abschnitt 8.2 ’Physikalische Grundlagen’ eingeführten Gesetzen berechnet werden können. Der Lichtstrom, der von dem Flächenelement dA_i ausgeht und auf der Fläche dA_j auftrifft, ist nach dem Grundgesetz der Strahlungsübertragung (2.12) (vgl. Bild 7.50):

$$d^2\varphi_{dA_i \to dA_j} = L_{dA_i} \frac{\cos\vartheta_j \cos\vartheta_i}{R^2} dA_j dA_i. \tag{7.66}$$

L_{dA_i} ist die Leuchtdichte des Flächenelements dA_i. Sie gibt an, wieviel Lichtstrom pro Fläche und Raumwinkel emittiert wird. Dabei wird vorausgesetzt, daß die beiden Flächenelemente nicht durch andere Flächen gegenseitig verdeckt werden. Der gesamte Lichtstrom einer diffus reflektierenden Fläche dA_i ist mit Gleichung 2.19 gegeben als

$$d\varphi_{Ai} = \pi L_{dA_i} dA_i. \tag{7.67}$$

[3] Statt Formfaktor (form factor) werden sie dort als ‘configuration factor’ bezeichnet.

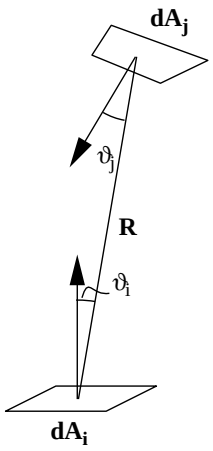


Abbildung 7.50: Geometrie der Formfaktoren

Einsetzen der Gleichungen 7.66 und 7.67 in Gleichung 7.61 liefert den Formfaktor $F_{dA_i\text{-}dA_j}$ zwischen den Flächenelementen dA_i und dA_j:

$$F_{dA_i\text{-}dA_j} = \frac{d^2\varphi_{dA_i \to dA_j}}{d\varphi_{dA_i}} = \frac{\cos\vartheta_j \cos\vartheta_i}{\pi R^2} dA_j. \tag{7.68}$$

Um auch die Verdeckung von Flächen mit zu beschreiben, wird für jedes differentielle Flächenpaar dA_i, dA_j eine Funktion H_{ij} eingeführt, die den Wert 1 hat, falls keine andere Fläche zwischen dA_i und dA_j liegt, andernfalls den Wert 0. Die allgemeine Formel lautet dann:

$$F_{dA_i\text{-}dA_j} = H_{ij} \frac{\cos\vartheta_j \cos\vartheta_i}{\pi R^2} dA_j. \tag{7.69}$$

Durch Integration über die Fläche A_j erhält man den Bruchteil des Lichtstroms der infinitesimalen Fläche dA_i, der auf der endlichen Fläche A_j landet:

$$F_{dA_i\text{-}A_j} = \int_{A_j} H_{ij} \frac{\cos\vartheta_j \cos\vartheta_i}{\pi R^2} dA_j. \tag{7.70}$$

Den Formfaktor zweier endlicher Flächen bekommt man schließlich noch durch Integration über die Fläche A_i und anschließende Mittelwertbildung (Division durch A_i):

$$F_{A_i\text{-}A_j} = F_{ij} = \frac{1}{A_i} \int_{A_i} \int_{A_j} H_{ij} \frac{\cos\vartheta_j \cos\vartheta_i}{\pi R^2} dA_j dA_i. \tag{7.71}$$

Die Formfaktoren F_{ij} hängen also nur von geometrischen Größen der Szene ab. Aus der Definition und den Bestimmungsgleichungen der Formfaktoren lassen sich eine Reihe von Eigenschaften ableiten:
Das Integral in Gleichung 7.71 ist invariant unter der Vertauschung von i und j. Dadurch ergibt sich folgende einfache Reziprozitätsbeziehung, von der noch häufiger Gebrauch gemacht werden wird:

$$A_i F_{ij} = A_j F_{ji}. \tag{7.72}$$

Aus Gründen der Energieerhaltung gilt für jede abstrahlende Fläche A_i:

$$\sum_{j=1}^{N} F_{ij} \leq 1. \tag{7.73}$$

Jede konvexe, im Grenzfall ebene, Fläche strahlt kein Licht auf sich selbst ab:

$$F_{ii} = 0. \tag{7.74}$$

Enthält die Szene nur planare Flächen, so sind also bei N Flächen nur $N(N-1)/2$ Formfaktoren zu berechnen. An der quadratischen Abhängigkeit der Formfaktorzahl von der Zahl der Flächen ändert das jedoch nichts und macht die Berechnung der Formfaktoren zum aufwendigsten Arbeitsschritt im Radiosity-Verfahren.

7.13.1.1 Lösung des Gleichungssystems

Hat man die Formfaktoren F_{ij} der Szene ermittelt, so kann man mit der Radiosity-Gleichung

$$B_j = E_j + \rho_j \sum_{i=1}^{N} F_{ij} B_i \frac{A_i}{A_j} \tag{7.75}$$

die Radiosity B_j einer jeden Fläche A_j berechnen. Durch Ausnutzen der Reziprozitätsbeziehung von Formfaktoren bekommt die Gleichung eine einfachere Gestalt:

$$B_j = E_j + \rho_j \sum_{i=1}^{N} F_{ji} B_i. \tag{7.76}$$

(Aus der Form der Gleichung wird der in der Literatur häufig gebrauchte Begriff "Sammeln" klar.) Die Gleichung wird noch übersichtlicher, wenn man in die Matrixschreibweise übergeht:

$$\vec{B} = \vec{E} + \rho \mathbf{F} \vec{B} \Rightarrow (\mathbf{1} - \rho \mathbf{F}) \vec{B} = \vec{E},$$

wobei nun $\vec{E}$ und $\vec{B}$ Vektoren der Länge N und $\mathbf{F}$ eine $N \times N$-Matrix ist. Auch ρ ist eine $N \times N$-Matrix, allerdings in Diagonalgestalt.
Ausgeschrieben lautet das Gleichungssystem:

$$\begin{pmatrix} 1 & -\rho_1 F_{12} & \cdots & -\rho_1 F_{1N} \\ -\rho_2 F_{21} & 1 & \cdots & -\rho_2 F_{2N} \\ \vdots & \vdots & & \vdots \\ -\rho_N F_{N1} & -\rho_N F_{N2} & \cdots & 1 \end{pmatrix} \begin{pmatrix} B_1 \\ B_2 \\ \vdots \\ B_N \end{pmatrix} = \begin{pmatrix} E_1 \\ E_2 \\ \vdots \\ E_N \end{pmatrix}, \tag{7.77}$$

wobei ausgenutzt wurde, daß $F_{ii} = 0$. Die Koeffizienten-Matrix $(\mathbf{1} - \rho \mathbf{F})$ ist regulär wegen der Diagonaldominanz der Matrix (siehe unten). Sind die Lichtquellenterme alle Null, so folgt damit sofort, daß die einzige Lösung des Gleichungssystems die triviale Lösung ist; alle Radiosities sind Null. Für die Lösung des Gleichungssystems kann man ein Standardlösungsverfahren, wie z.B. das Eliminationsverfahren von Gauss, anwenden. In der Praxis hat sich jedoch das iterative

Gauss-Seidel-Verfahren bewährt. Ein hinreichendes Kriterium für die Konvergenz des Gauss-Seidel-Verfahrens ist die *Diagonaldominanz* der Koeffizientenmatrix:

$$\sum_{j=1,\, j\neq i}^{N} \mid a_{ij} \mid < \mid a_{ii} \mid . \tag{7.78}$$

Die Summe der Elemente einer Zeile muß also kleiner sein als das Diagonalelement. Es muß also für jedes i gelten:

$$\sum_{i\neq j} \mid \rho_i F_{ij} \mid = \rho_i \sum_{i\neq j} F_{ij} < 1. \tag{7.79}$$

Da für geschlossene Szenen $\sum_{i=1}^{N} F_{ij} = 1$ ist, folgt damit, daß der Reflexionsgrad $\rho_i < 1$ sein muß. Die physikalische Forderung, daß ein Energieflußgleichgewicht nur erreicht wird, wenn Strahlungsleistung auch wieder abgeführt wird, drückt sich also hier in einem Konvergenzkriterium aus.

7.13.1.2 Farben im Radiosity-Verfahren

Bei den bisherigen Betrachtungen zum Radiosity-Verfahren wurde vernachlässigt, daß Größen wie das Emissionsvermögen E, der Reflexionsgrad ρ und somit natürlich auch die Radiosity B wellenlängenabhängig sind. Die Wellenlängenabhängigkeit ist dafür verantwortlich, daß Szenen farbig erscheinen. So beruht die Farbigkeit von nicht selbstleuchtenden Oberflächen darauf, daß ein Teil des einfallenden Lichts absorbiert wird. Der reflektierte Teil des Spektrums legt dann die Farbe der Oberfläche fest. Die Formfaktoren sind natürlich unabhängig von der Wellenlänge. Die exakte Radiosity-Gleichung lautet also:

$$B_j(\lambda) = E_j(\lambda) + \rho_j(\lambda) \sum_{i=1}^{N} F_{ji} B_i(\lambda) \qquad \text{für alle Wellenlängen } \lambda. \tag{7.80}$$

Da E und ρ stetige Funktionen der Wellenlänge λ sind, ist aufgrund der Regularität der Koeffizientenmatrix auch die Radiosity B stetig. Will man nun die genauen Beleuchtungsverhältnisse in der Szene bestimmen, müßte man für jede Wellenlänge das Gleichungssystem lösen. Da man weder unendlich viele Gleichungssysteme lösen kann, noch den Lichtstrom bei einer festen Wellenlänge genau bestimmen kann, ist es sinnvoll, die Radiosity-Gleichung für aneinandergrenzende Wellenlängenbänder $\Delta\lambda$ zu lösen. Da im Radiosity-Verfahren sämtliche Berechnungen im Objektraum stattfinden, sind die Ergebnisse vollkommen unabhängig von den Eigenschaften des Monitors, wie z.B. der Auflösung oder der Anzahl der darstellbaren Farben. Die Trennung von Berechnung und Ausgabe legt es eigentlich nahe, die Berechnung mit physikalisch relevanten Größen in der gewünschten Genauigkeit durchzuführen und anschließend die erhaltenen Radiosities $B_i(\lambda)$ mit den vorhandenen Farben des Ausgabegeräts möglichst gut zu approximieren, wie es in Abschnitt 8.3 'Farbmetrik' beschrieben wurde.

In der GDV wird aber für gewöhnlich die Radiosity-Gleichung nur für die Grundfarben R, G und B eines RGB-Monitors gelöst. Für jede Fläche werden also der Reflexionsgrad $\rho_{\mathcal{R}}$, $\rho_{\mathcal{G}}$ und $\rho_{\mathcal{B}}$ benötigt sowie $E_{\mathcal{R}}$, $E_{\mathcal{G}}$ und $E_{\mathcal{B}}$ für die Charakterisierung von Lichtquelleneigenschaften.

Nach der Berechnung der Radiosities $B_{\mathcal{R}}$, $B_{\mathcal{G}}$ und $B_{\mathcal{B}}$ stellt sich noch das Problem der Skalierung in den vom Ausgabegerät darstellbaren Intensitätsbereich, also z.B. in dem Bereich zwischen 0 und 1. Die Problematik liegt in der Unabhängigkeit des Radiosity-Verfahrens von der Position und der Blickrichtung des Beobachters. Ist in der Szene eine sehr helle Lichtquelle vorhanden, so führt die Skalierung, zumindest wenn sie linear ist, dazu, daß alle Flächen der Szene um einen entsprechenden Faktor verdunkelt werden, obwohl die helle Lichtquelle vom Beobachter aus möglicherweise gar nicht direkt gesehen werden kann. Die Folge ist, daß die ganze Szene ziemlich dunkel erscheint.

Der Übergang von den Radiosities auf die Farbvalenzen des Ausgabegerätes sollte also sinnvollerweise erst nach der Festlegung der Beobachtungsparameter vollzogen werden.

Wenn Sie die PDF-Version des Kurses bearbeiten, können Sie jetzt das Interaktionselement 'radio3d' anwählen.

7.13.1.3 Graphische Ausgabe der berechneten Szene

Das Ergebnis einer Radiosity-Rechnung liefert einen Radiosity-Wert für jedes Patch der Szene. Durch Skalierung kann dieser Wert in eine darstellbare Farbe umgerechnet werden.

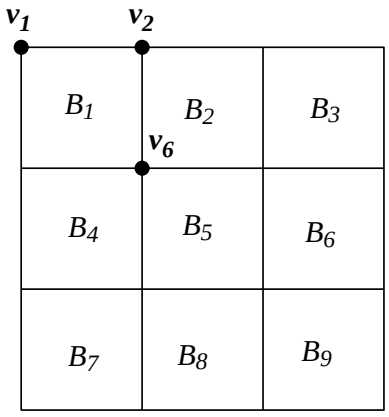


Abbildung 7.51: Radiosity-Werte an Eckpunkten

Für die graphische Ausgabe werden jedoch Radiosity-Werte an den Eckpunkten (Knoten) der Patches benötigt, damit durch eine Gouraud-Interpolation stetige Farbübergänge erzeugt werden können. Die Knoten-Radiosities können aus den Radiosities der Patches durch Mittelung berechnet werden (vgl. Bild 7.51). Die Radiosity am Punkt v_6 ist z.B:

$$B(v_6) = \frac{1}{4}(B_1 + B_2 + B_4 + B_5).$$

Bei Knoten an den Rändern muß man darauf achten, daß nur die angrenzenden Patches berücksichtigt werden:

$$B(v_1) = B_1$$

oder

$$B(v_2) = \frac{1}{2}(B_1 + B_2).$$

Die Farbwerte innerhalb der Patches können dann, wie bei der Gouraud-Interpolation üblich, durch bilineare Interpolation gewonnen werden.

7.13.2 Progressive Refinement

Eine alternative Methode, um die Radiosity-Gleichung zu lösen, wurde 1988 von Cohen et. al. [CCWG88] vorgeschlagen, nämlich das Progressive-Refinement-Verfahren.

Obwohl seit der Einführung des Radiosity-Verfahrens im Jahre 1984 sehr effiziente Verfahren zur Berechnung von Formfaktoren entwickelt wurden (siehe nächster Abschnitt), war für die praktische Anwendbarkeit des Radiosity-Verfahrens doch immer ein Hindernis, daß die Anzahl der Formfaktoren mit $O(N^2)$ wächst. Nicht nur der hohe Rechenaufwand, sondern auch der enorme Speicherbedarf für die Formfaktormatrix lieferten ausreichend Motivation für die Suche nach anderen Verfahren. So ist der Speicherbedarf für die Formfaktormatrix einer Szene mit 3000 Patches ca 17.5 Megabyte, selbst wenn man die Reziprozitätsbeziehung ausnutzt. Das Progressive Refinement ist, wie das Gauss-Seidel-Verfahren, ein iteratives Verfahren zur Lösung der Radiosity-Gleichung. Krämer [SK90] hat gezeigt, daß beide Iterationsverfahren gegen dieselbe Lösung konvergieren.

Das Progressive-Refinement-Verfahren beruht darauf, daß man die Ausbreitung des Lichts durch die Szene ausgehend von den primären Lichtquellen verfolgt. In der Radiosity-Gleichung 7.76 beschreibt der Term

$$\rho_j \sum_{i=1}^{N} F_{ji} B_i$$

den Teil der Radiosity B_j der Fläche A_j, der aus der Bestrahlung der Fläche A_j durch alle anderen Flächen resultiert. Dieser Vorgang wird deshalb auch als *Einsammeln* (Gathering) von Strahlung bezeichnet. Der Beitrag einer einzelnen Fläche A_i ist also:

$$B_{j \leftarrow i} = \rho_j F_{ji} B_i.$$

Die Symmetrie der Grundgleichung der Strahlungsübertragung zwischen Sende- und Empfängergrößen und die daraus folgende Eigenschaft der Formfaktoren $F_{ji} = \frac{A_i}{A_j} F_{ij}$ macht es möglich, den umgekehrten Prozeß, nämlich die *Verteilung* von Strahlung (Shooting) einer Fläche A_j an die anderen Flächen einfach durch

Vertauschen der Indizes zu beschreiben. Die Radiosity, die eine Fläche A_j an eine Fläche A_i weitergibt, ist gegeben als

$$B_{j\to i} = \rho_i B_j F_{ij} = \rho_i B_j F_{ji} \frac{A_j}{A_i}. \tag{7.83}$$

Zu Beginn des Progressive-Refinement-Verfahrens werden die Radiosities B_i mit den Lichtquellentermen E_i initialisiert. Um nicht die gleiche Radiosity mehrmals in der Szene zu verteilen, wird noch eine weitere Größe benötigt: die *unverteilte* Radiosity ΔB_i. Die ist am Anfang natürlich mit der Eigenausstrahlung E_i identisch. Vor dem ersten Iterationsschritt gilt also:

$$B_i := E_i \qquad \text{und} \qquad \Delta B_i := E_i \qquad \text{für } i = 1,\ldots,N. \tag{7.84}$$

Jeder Iterationsschritt besteht jetzt darin, die unverteilte Radiosity einer Fläche A_j der Szene auf alle anderen zu verteilen, wozu man lediglich einen Satz von Formfaktoren F_{ji} für $i = 1,\ldots,N$ benötigt. Die Radiosity der Flächen A_i wird dann entsprechend Gleichung 7.83 erhöht:

$$B_i := B_i + \rho_i \Delta B_j F_{ji} \frac{A_j}{A_i} \qquad \text{für } i = 1,\ldots,N. \tag{7.85}$$

Ebenso wird die unverteilte Radiosity erhöht:

$$\Delta B_i := \Delta B_i + \rho_i \Delta B_j F_{ji} \frac{A_j}{A_i} \qquad \text{für } i = 1,\ldots,N.$$

Anschließend wird die unverteilte Radiosity der Fläche A_j auf Null gesetzt:

$$\Delta B_j := 0.$$

Die schnelle Konvergenz des Verfahrens beruht darauf, daß man nicht beliebige Flächen auswählt, deren Radiosity dann verteilt wird, sondern man wählt für jeden Iterationsschritt die Fläche mit der jeweils größten unverteilten Strahlungsleistung aus. Am Anfang des Verfahrens wird folglich das Licht der stärksten Lichtquelle verteilt. Für die Konvergenz des Verfahrens ist wiederum entscheidend, daß der Reflexionsgrad kleiner als Eins ist. Die unverteilten Radiosities streben in diesem Fall gegen Null. Zudem gleichen sie sich immer mehr an. Als Abbruchkriterium kann man z.B. die Summe der unverteilten Radiosities nehmen, die einen bestimmten Wert unterschreiten soll. Das bedeutet, daß schon ein bestimmter Prozentsatz der von den Lichtquellen emittierten Leistung absorbiert wurde. Eine Alternative wäre, einen Schwellwert für die maximale unverteilte Radiosity festzulegen, der nicht unterschritten werden soll.

Der große Vorteil des Progressive-Refinement ist, daß für jeden Iterationsschritt nur die Formfaktoren F_{ij} eines einzelnen Patches i mit allen anderen Patchen $j = 1,\ldots,N$ berechnet werden müssen, d.h. zur Durchführung eines Iterationsschrittes genügt die Kenntnis einer Zeile der Formfaktormatrix. Da bereits nach wenigen Iterationsschritten sehr gute Ergebnisse erzielt werden, liegt der gesamte Rechenaufwand weit unterhalb der üblichen Vorgehensweise mit Bestimmung der Formfaktormatrix und Lösen des Gleichungssystems mit dem Gauss-Seidel-Verfahren. Außerdem wird jeweils nur der Speicherplatz für eine Zeile von Formfaktoren benötigt.

Nachteile des Progressive-Refinement-Verfahren gegenüber dem Full-Matrix-Verfahren sollen abschließend nicht verschwiegen werden. Während eine nachträgliche Änderung der Beleuchtungsparameter im Full-Matrix-Verfahren nur eine neuerliche Lösung des Gleichungssystems erfordert, muß das komplette Progressiv-Refinement-Verfahren mit Formfaktorberechnung wiederholt werden. Wird eine Fläche zum wiederholten Mal verteilende Fäche, so werden deren Formfaktoren jedesmal wieder neu berechnet.

Abbildung 7.52: Konvergenz des Progressive Refinement

Die Zeitersparnis beim Progressive-Refinement macht diesen Nachteil aber mehr als wett. Bild 7.52 zeigt die schnelle Konvergenz.
Auf Näherungsverfahren zur Berechnung von Formfaktoren kann im Rahmen dieser Einführung nicht eingegangen werden. Es wird auf den Kurs GDV II (K1693) verwiesen.

7.13.2.1 Nusselts Analogon

Schon 1928 stellte Wilhelm Nusselt [Nus28] ein Verfahren vor, mit dem sich das Integral in Gleichung 7.71 auf elegante Weise geometrisch lösen läßt. Zu diesem Zweck wird, wie in Skizze 7.53 zu sehen ist, eine Halbkugel mit Einheitsradius konstruiert, in deren Zentrum sich das Flächenelement dA_i befindet. Betrachtet man nochmals die Gleichung 7.70

$$F_{dA_i\text{-}A_j} = \frac{1}{\pi} \int_{A_j} \cos\vartheta_i \left(\frac{\cos\vartheta_j}{R^2} dA_j \right), \tag{7.88}$$

so erkennt man, daß der geklammerte Ausdruck der Definition des differentiellen Raumwinkels $d\Omega$ entspricht (s. Gleichung 7.3 im Abschnitt 8.2.1 'Strahlungsphy-

sikalische (radiometrische) Grundgrößen') :

$$d\Omega = \frac{\cos\vartheta_j \, dA_j}{R^2}. \tag{7.89}$$

Andererseits ist der Raumwinkel $d\Omega$ nach Definition genau die Projektion von dA_j auf die Einheitshalbkugel über dA_i:

$$dA_s = d\Omega. \tag{7.90}$$

Damit ergibt sich

$$F_{dA_i\text{-}A_j} = \frac{1}{\pi}\int_{A_j} \cos\vartheta_i \, d\Omega = \frac{1}{\pi}\int_{A_s} \cos\vartheta_i \, dA_s. \tag{7.91}$$

Der Integrand $\cos\vartheta_i \, dA_s = dA_p$ ist die Parallelprojektion von dA_s auf die Basis der Halbkugel. Die Integration über A_s ergibt also die Fläche der senkrechten Projektion A_p von A_s. Für den Formfaktor gilt somit:

$$F_{dA_i\text{-}A_j} = \frac{A_p}{\pi}. \tag{7.92}$$

Wenn man in Betracht zieht, daß π die Fläche des Einheitskreises ist, so ist der Formfaktor $F_{dA_i\text{-}A_j}$ der Anteil, der von der Parallelprojektion A_p in der Basis der Halbkugel eingenommen wird.

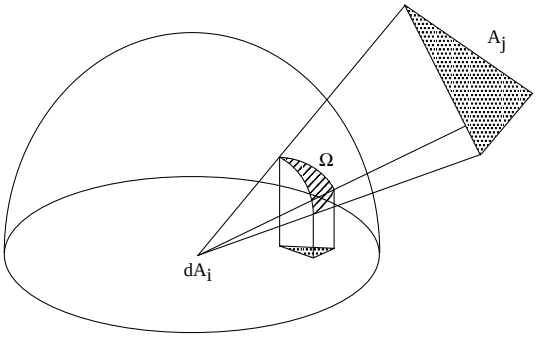


Abbildung 7.53: Halbkugel-Geometrie zur Berechnung von Formfaktoren

Für die Berechnung von Formfaktoren wird die Oberfläche der Halbkugel in kleine Segmente aufgeteilt. Für jedes dieser Segmente wird der Formfaktor zwischen dem Segment und der Fläche dA_i durch Parallelprojektion auf die Basis berechnet. Nachdem man diese sogenannten *Delta-Formfaktoren* berechnet hat, wird jede Fläche A_j der Szene auf die Halbkugel projiziert, wobei registriert wird, welche Segmente auf der Halbkugel von der Projektion der Fläche A_j bedeckt werden. Der gesamte Formfaktor der Fläche A_j ist dann einfach die Summe der Delta-Formfaktoren der von A_j bedeckten Segmente.

In der Praxis erwies sich das Verfahren als wenig brauchbar. Es gab Schwierigkeiten bei der Aufteilung der Halbkugel in möglichst gleich große Segmente und vor allem bei der Scan-Konvertierung der Szene auf die Halbkugel. Das Nusseltsche Analogon hat seine Bedeutung darin, daß es die Grundlage zu dem ersten brauchbaren Verfahren zur Berechnung von Formfaktoren ist, nämlich dem *Hemi-Cube-Verfahren.*

7.13.2.2 Das Hemi-Cube-Verfahren

Im Jahre 1985 wurde von Michael F. Cohen und Donald P. Greenberg [CG85] ein Verfahren zur Formfaktorberechnung vorgestellt, das effizient war und endlich auch das Problem mit der Verdeckung von Flächen löste, nämlich das Hemi-Cube-Verfahren. Erst mit diesem Verfahren wurde Radiosity auch für komplexe Szenen praktisch einsetzbar. Insbesondere mit dem Progressive-Refinement-Verfahren, das allerdings erst im Jahre 1987 entwickelt wurde, verhalf der Hemi-Cube zu Rechenzeiten, die nicht weit über denen lagen, die für Raytracing benötigt wurden.

Die Idee war, daß man alle Flächen der Szene auf eine das Flächenelement dA_i einhüllende Fläche projiziert. Diese Fläche soll wiederum in Segmente zerlegt werden und der Formfaktor $F_{dA_i\text{-}A_j}$ durch Aufsummieren der Delta-Formfaktoren gewonnen werden. Eine Halbkugel hat sich, wie erwähnt, als zu schwierig erwiesen. Cohen und Greenberg griffen nun auf Flächen zurück, für die in der GDV große Erfahrung vorhanden ist, nämlich rechteckige Rasterflächen. Aus der Halbkugel wurde also ein Halbwürfel, oder Hemi-Cube (vgl. Bild 7.54). Jede Seite wird mit einem rechteckigen Raster überzogen, den sogenannten Hemi-Cube-Pixeln. Die Projektion der Szene auf eine Fläche des Hemi-Cube stellt somit das gleiche Problem dar, wie die Darstellung einer dreidimensionalen Szene auf einem Raster-Display, weshalb auf die üblichen Scan-Konvertierungsverfahren zurückgegriffen werden kann. Beim Hemi-Cube muß auf fünf Rasterflächen projiziert werden: auf die obere Fläche und auf die vier Seitenflächen.

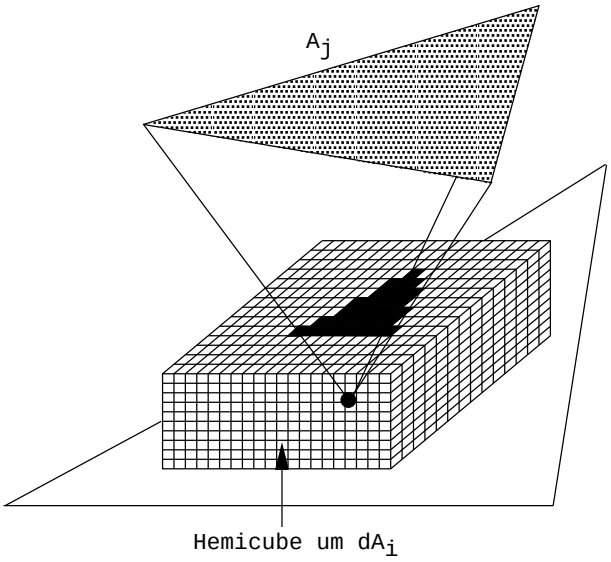


Abbildung 7.54: Der Hemi-Cube

Jedem Hemi-Cube-Pixel q wird ein Deltaformfaktor ΔF_q zugeordnet, der den Beitrag von q zum gesamten Formfaktor beschreibt. Den Formfaktor einer Fläche A_j erhält man, indem man die Deltaformfaktoren derjenigen Hemi-Cube-Pixel, die durch die Projektion von A_j bedeckt werden, aufsummiert. Je feiner die Unterteilung des Hemi-Cube ist, desto genauer wird letztlich der Formfaktor. Durch

Verwendung des **z-Buffer**-Algorithmus für die Scan-Konvertierung der Szene auf die Hemi-Cube-Flächen ist es nun möglich, auch die Verdeckung von Flächen zu berücksichtigen. Erstmals wird damit das Radiosity-Verfahren auch für komplexe Szenen anwendbar.

Analog zum Raster-Display wird durch das z-Buffer-Verfahren auf den Hemi-Cube-Flächen ein „Bild" der Szene aufgebaut, wobei den Hemi-Cube-Pixeln allerdings keine Farben, sondern die Indizes der projizierten Flächen zugeordnet sind.

Wird von diesem Bild ein Histogramm berechnet, also von jeder Farbe bzw. jedem Index die Häufigkeit berechnet, wobei die Hemi-Cube-Pixel entsprechend ihres Delta-Formfaktors gewichtet werden, so erhält man schließlich die Formfaktoren $F_{dA_i\text{-}A_j}$ für $j = 1,\ldots,N$. Mittels *eines* Hemi-Cubes werden also alle Formfaktoren eines Flächenelements dA_i zu allen anderen Flächen A_j bestimmt.

Beim ursprünglichen Radiosity-Verfahren, das auf der Lösung des vollständigen Gleichungssystems basiert, muß das Hemi-Cube-Verfahren für jede der N Flächen der Szene angewandt werden, um alle Formfaktoren zu bestimmen. Im Progressive-Refinement-Verfahren hingegen reicht ein einziger Hemi-Cube aus, um die Strahlung der hellsten Fläche an alle anderen Flächen zu verteilen. Die Verbindung des Hemi-Cube-Verfahrens mit dem Progressive-Refinement ermöglichte den Durchbruch des Radiosity-Verfahrens.

Wenn Sie die PDF-Version des Kurses bearbeiten, können Sie jetzt das Interaktionselement 'hemicube' anwählen.

7.13.2.3 Berechnung von Delta-Formfaktoren

Cohen und Greenberg gingen davon aus, daß die Hemi-Cube-Pixel so klein sind, daß man sie in guter Näherung als infinitesimale Flächenelemente auffassen kann. Der Delta-Formfaktor ΔF_q kann somit durch die Gleichung 7.68 berechnet werden. Die Rechnung wird für die Oberseite des Hemi-Cubes nachvollzogen, die sich im Abstand $h = 1$ von der Lichtquelle befinden soll. Die Rechnung für die Seitenflächen läuft völlig analog.

$$F_q = \frac{\cos\vartheta_j \cos\vartheta_i}{\pi R^2}\Delta A_p. \tag{7.93}$$

ΔA_p ist die Fläche des Hemi-Cube-Pixels. Da das Hemi-Cube-Pixel parallel zur Fläche dA_i ist, gilt $\cos\vartheta_j = \cos\vartheta_i$. Außerdem ist nach Pythagoras

$$R = \sqrt{x^2 + y^2 + 1}.$$

Damit läßt sich $\cos\vartheta_i$ wie folgt ausdrücken:

$$\cos\vartheta_i = \frac{1}{\sqrt{x^2 + y^2 + 1}}.$$

Für den Delta-Formfaktor ΔF_q ergibt sich somit folgende einfache Gleichung:

$$\Delta F_q = \frac{1}{\pi(x^2 + y^2 + 1)^2}\Delta A_p. \tag{7.96}$$

7.13.2.4 Genauigkeit des Hemi-Cube-Verfahrens

Es sind im wesentlichen drei Effekte, die zur ungenauen Berechnung von Formfaktoren mit dem Hemi-Cube-Verfahren führen. Zwei dieser Effekte hängen mit den Annahmen zusammen, die für Gleichung 7.71 gemacht wurden. Der dritte resultiert aus der endlichen Größe der Hemi-Cube-Pixel. Die Effekte sollen im einzelnen besprochen und Lösungsvorschläge gemacht werden.

1. *Fehler durch 'Fernnäherung':*

In jeder Szene gibt es Fälle, in denen die Annahme, daß die Entfernung zwischen zwei Flächen groß ist im Vergleich zu ihren Ausdehnungen, verletzt wird. Am augenscheinlichsten ist das bei aneinandergrenzenden, nichtparallelen Flächen der Fall (vgl. Bild 7.55).

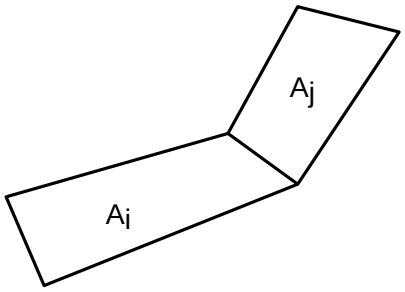


Abbildung 7.55: Fehler durch die Fernnäherung

Schon bei Betrachtung der Gleichung 7.70 wird deutlich, daß der resultierende Formfaktor zwischen A_i und A_j sehr von der Wahl des Bezugspunktes auf der Fläche A_i abhängt, da der Abstand R sogar quadratisch in die Formel eingeht. Wählte man wie üblich den Mittelpunkt von A_i als Bezugspunkt, so erhielte man einen wesentlich kleineren Formfaktor als bei einem Bezugspunkt in der Nähe der gemeinsamen Kante von A_i und A_j. Durch Unterteilung der Fläche A_i kann dieses Problem im Prinzip gelöst werden. Es werden damit zwar wieder Flächen geschaffen, die direkt an A_j angrenzen, deren wiederum fehlerbehaftete Formfaktoren fallen jedoch für den gesamten Formfaktor zwischen A_i und A_j kaum ins Gewicht. Durch entsprechend feine Unterteilung von A_i kann somit der Fehler unter jede gewünschte Grenze gebracht werden.

2. *Fehler durch teilweise Verdeckung:*

Ein zweiter Fehler tritt auf, wenn die Funktion H_{ij} in Gleichung 7.70 abhängig davon ist, wo man sich auf der Fläche A_i befindet. In Bild 7.56 sieht man, daß bei der gewählten Position des Hemi-Cubes auf A_i keinerlei Verdeckung der Fläche A_j durch A_k registriert wird, obwohl dies in der Tat eintritt. Die Folge ist, daß ein zu großer Formfaktor F_{ij} berechnet wird. Auch dieser Fehler läßt sich durch Unterteilen von A_i beliebig klein machen.

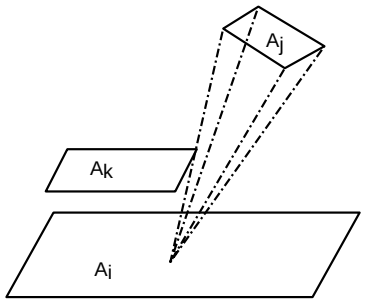


Abbildung 7.56: Teilweise Verdeckung der Fläche A_j durch A_k

3. *Fehler durch endliche Hemi-Cube-Auflösung:*

Mit der Diskretisierung der Hemi-Cube-Oberflächen in Hemi-Cube-Pixel treten Fehler durch Alias-Effekte zutage, ganz analog zur Darstellung von Bildern auf einem Raster-Display. So wird ein Hemi-Cube-Pixel, und damit der zugehörige Delta-Formfaktor, einer Fläche zugeordnet, obwohl sie das Hemi-Cube-Pixel nur teilweise bedeckt, wodurch auf der anderen Seite eine andere Fläche vollkommen leer ausgeht (vgl. Bild 7.57).

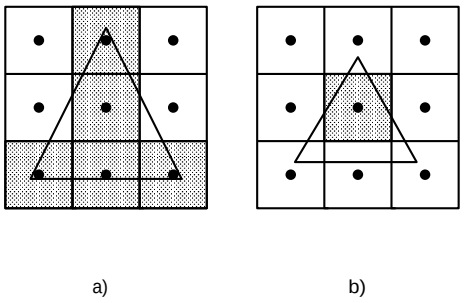


Abbildung 7.57: Alias-Effekte auf dem Hemi-Cube
Zu großer Formfaktor in a) und zu kleiner Formfaktor in b)

Um solche Alias-Fehler zu minimieren, kann auf bekannte Anti-Aliasing-Verfahren zurückgegriffen werden. So kann mit Hilfe einer Präfilterung der genaue Bedeckungsgrad eines Hemi-Cube-Pixels berechnet und der Delta-Formfaktor mit diesem Wert gewichtet werden. Damit werden die Alias-Effekte zwar weitgehend vermieden, jedoch mit sehr hohem Rechenaufwand. Gewöhnlich reicht es aber aus, die Auflösung des Hemi-Cubes zu vergrößern. Üblicherweise wird in der Literatur eine Auflösung von ca. 200 Hemi-Cube-Pixel angegeben.

7.13.3 Formfaktorberechnung mit Raytracing

Als letztes Verfahren zur Berechnung von Formfaktoren soll die Methode von Wallace et. al. [WEH89] vorgestellt werden. Auch dieses Verfahren berechnet Formfaktoren $F_{dA_i\text{-}A_j}$, also Formfaktoren von infinitesimalen Flächenelementen zu endlichen Flächen. Während beim Hemi-Cube-Verfahren die Lichtquelle die differentielle Fläche darstellt, über die der Hemi-Cube gelegt wird (nur so läßt sich

mit *einem* einzigen Hemi-Cube ein Iterationsschritt im Progressive Refinement vollziehen), ist es beim Verfahren von Wallace genau umgekehrt. Die Lichtquelle wird als endliche Fläche aufgefaßt und von ihr aus werden die Formfaktoren zu infinitesimalen Flächenelementen berechnet.

Der große Vorteil dieser Vorgehensweise ist, daß die Formfaktoren und damit die Radiosities genau an den gewünschten Stellen berechnet werden. Ist die Szene beispielsweise aus Polygonen aufgebaut, dann werden in jedem Schritt des Progressive Refinement die Formfaktoren vom hellsten Polygon zu allen Eckpunkten der Szene bestimmt. Dadurch werden von vornherein Fehler vermieden, die durch die Unterabtastung des Hemi-Cube zustande kommen, z.B. daß kleine Flächen bei der Verteilung von Radiosity unberücksichtigt bleiben.

Die eigentliche Formfaktorberechnung wird durch eine Kombination von Raytracing und analytischer Formfaktorberechnung bewerkstelligt. Das Raytracing dient dabei lediglich zur Verdeckungsrechnung. Da sich die Verdeckung innerhalb des Lichtquellenpolygons mit Fläche A_j ändern kann, wird eine variable Anzahl N von Referenzpunkten P_k ausgewählt, die gleichmäßig verteilt auf dem Lichtquellenpolygon liegen. Um jeden dieser Referenzpunkte werden gleich große Kreisscheiben mit Radius a_k und der Fläche $A_{j,k}$ gelegt, die das Lichtquellenpolygon approximieren sollen. Dabei soll gelten:

$$A_j = \sum_{k=1}^{N} A_{j,k}. \tag{7.97}$$

Bild 7.58 zeigt die Approximation eines Dreiecks durch neun Kreisscheiben.

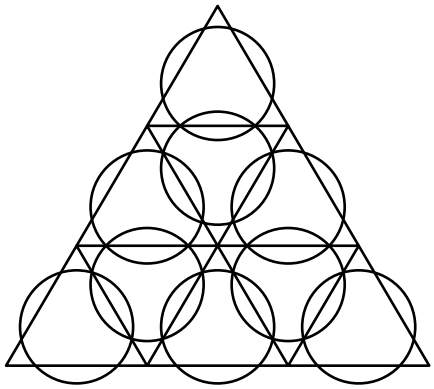


Abbildung 7.58: Approximation des Lichtquellenpolygons durch Kreisscheiben

Für die Berechnung des Formfaktors zwischen dA_i und einer Kreisscheibe $A_{j,k}$ gibt es eine analytische Formel, die am Anfang des Kapitels vorgestellt wurde:

$$F_{dA_i\text{-}A_{j,k}} = \frac{\cos\vartheta_{i,k}\cos\vartheta_{j,k}\,a_k^2}{r_k^2 + a_k^2}, \tag{7.98}$$

wobei a_k der Radius der Kreisscheibe $A_{j,k}$ und r_k der Abstand zwischen dA_i und P_k ist. Auch die Winkel $\vartheta_{i,k}$ und $\vartheta_{j,k}$ zwischen den Normalen und der Verbindungsstrecke sind für jeden Referenzpunkt verschieden und bekommen deshalb

den Index k.
Da die $A_{j,k}$ die emittierenden Flächen sind, wird jedoch der reziproke Formfaktor $F_{A_{j,k}\text{-}dA_i}$ benötigt. Mit Hilfe der Reziprozitätsbeziehung zwischen Formfaktoren und der Formel $A_{j,k} = \pi a_k^2$ erhält man

$$F_{A_{j,k}\text{-}dA_i} = \frac{\cos\vartheta_{i,k}\cos\vartheta_{j,k}}{\pi r_k^2 + A_{j,k}} dA_i. \tag{7.99}$$

Zur Verdeckungsrechnung wird nun von dA_i ein Strahl zu jedem Punkt P_k losgeschickt. Liegt ein Objekt zwischen dA_i und einem Referenzpunkt P_k, so trägt $A_{j,k}$ nichts zum Formfaktor $F_{A_j\text{-}dA_i}$ zwischen dA_i und A_j bei. Der gesamte Formfaktor kann damit wie folgt berechnet werden:

$$F_{A_j\text{-}dA_i} = \sum_{k=1}^{N} \delta_k F_{A_{j,k}\text{-}dA_i}, \tag{7.100}$$

wobei $\delta_k = 1$, wenn $A_{j,k}$ nicht verdeckt ist, sonst 0.

Mit Gleichung 7.99 ergibt sich schließlich:

$$F_{A_j\text{-}dA_i} = dA_i \frac{1}{N} \sum_{k=1}^{N} \delta_k \frac{\cos\vartheta_{i,k}\cos\vartheta_{j,k}}{\pi r_k^2 + A_{j,k}}. \tag{7.101}$$

An dieser Stelle geht ein, daß die Kreisflächen gleich groß sein müssen und deren Gesamtfläche der Fläche von A_j entspricht.

Für die Radiosity B_i, die das Flächenelement dA_i von der Fläche A_j erhält, ergibt sich aus den Gleichungen 7.101 und 7.83

$$B_i = \rho_i A_j \frac{1}{N} \sum_{k=1}^{N} \delta_k \frac{\cos\vartheta_{i,k}\cos\vartheta_{j,k}}{\pi r_k^2 + A_{j,k}} B_k. \tag{7.102}$$

Aufgrund der Reziprozitätsbeziehung hat sich dabei dA_i herausgekürzt.

7.13.4 Bewertung

Das Radiosity-Verfahren bietet als erstes Verfahren in der GDV die Möglichkeit, indirekte Beleuchtung und die Wirkung von flächenhaften diffusen Lichtquellen effizient zu berechnen. Insbesonders bei Innenräumen wird dadurch eine wesentlich realistischere Wirkung der Szene erzielt als mit anderen Verfahren. Die konsequente Verwendung physikalischer Größen macht das Radiosity-Verfahren zum wichtigsten Verfahren in der Beleuchtungsplanung von Gebäuden. Ein großer Vorteil ist die Unabhängigkeit des Verfahrens von den Beobachtungsparametern, die das Durchwandern einer berechneten Szene in Echtzeit möglich macht. Demgegenüber muß man auch die Nachteile des Verfahrens sehen. Es ist sowohl von der Rechenzeit als auch vom Speicherbedarf aufwendiger als das Raytracing. Insbesondere bei komplexen Szenen sind ausgeklügelte Datenstrukturen notwendig.

Auch die Beschränkung auf rein diffuse Lichtausbreitung ist ein gravierender Nachteil. Dieser kann jedoch aufgehoben werden, indem man das Radiosity-Verfahren mit Raytracing kombiniert. Diese Kombination der beiden sehr gegensätzlichen Verfahren, man nennt es oft auch Zwei-Schritt-Verfahren [WCG87], hat bisher die realistischsten Bilder erbracht.

7.14 Übungsaufgaben

Aufgabe 1:

Gegeben sei eine vereinfachte Winged-Edge-Datenstruktur eines Quaders:

Kante	vstart	vend	ncw	nccw
e_1	v_1	v_2	e_2	e_5
e_2	v_2	v_3	e_3	e_6
e_3	v_3	v_4	e_4	e_7
e_4	v_4	v_1	e_1	e_8
e_5	v_1	v_5	e_9	e_4
e_6	v_2	v_6	e_{10}	e_1
e_7	v_3	v_7	e_{11}	e_2
e_8	v_4	v_8	e_{12}	e_3
e_9	v_5	v_6	e_6	e_{12}
e_{10}	v_6	v_7	e_7	e_9
e_{11}	v_7	v_8	e_8	e_{10}
e_{12}	v_8	v_5	e_5	e_{11}

Knoten	Koordinaten
v_1	$x_1y_1z_1$
v_2	$x_2y_2z_2$
v_3	$x_3y_3z_3$
v_4	$x_4y_4z_4$
v_5	$x_5y_5z_5$
v_6	$x_6y_6z_6$
v_7	$x_7y_7z_7$
v_8	$x_8y_8z_8$

Facette	Startkante	Orientierungs-Bit (sign)
f_1	e_1	$+$
f_2	e_9	$+$
f_3	e_6	$+$
f_4	e_7	$+$
f_5	e_{12}	$+$
f_6	e_9	$-$

Dabei gibt das Orientierungs-Bit an, wie die Kanten innerhalb der Facetten orientiert sind. Das Innere der Facetten liegt rechts der orientierten Kanten.

a) Bestimmen Sie für alle Flächen die zugehörigen Knoten, und machen Sie eine Skizze des Quaders.

b) Beschreiben Sie einen Algorithmus, um eine vereinfachte Winged-Edge-Datenstruktur in die im Text (Abschnitt 7.3 ’Randrepräsentationen (Boundary Representation)’) beschriebene knotenorientierte Datenstruktur zu konvertieren.
Benutzen Sie dabei folgende Funktionen:

- vstart(Kante) — Startknoten der Kante
- vend(Kante) — Endknoten der Kante
- ncw(Kante) — nächste Kante clock-wise (cw)
- nccw(Kante) — nächste Kante counter-clock-wise (ccw)

Aufgabe 2:

Die Kantenlänge des zur Wurzel eines Quadtrees gehörenden Quadranten sei l. Seine linke untere Ecke besitze die Koordinaten $(0,0)$ (vgl. Bild 7.59 zur Numerierung).

Beschreiben Sie jeweils einen Algorithmus, um in dem Quadtree

a) dasjenige Blatt zu finden, das den Punkt p mit den Koordinaten (x,y), $0 \leq x,y \leq l$, enthält,

b) den (die) Nachbarn eines Knotens in N-Richtung zu finden (vgl. Abbildung 7.59).

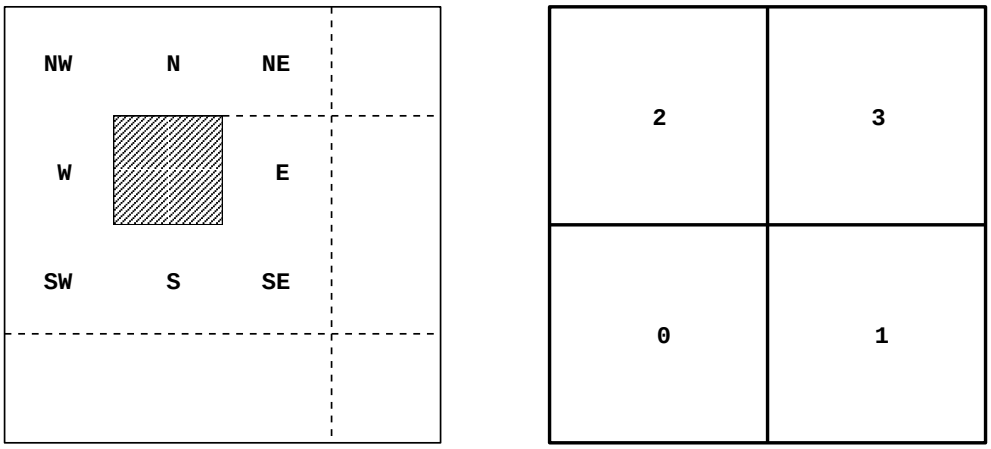


Abbildung 7.59: Numerierung des Quadranten

Um sich in dem Quadtree zu bewegen bzw. die Nummer der aktuellen Zelle abzufragen, sollten die Funktionen

- Vater() (gehe zu Vaterknoten)
- Kind(Nummer), Nummer = 0,1,2,3 (gehe zu Kindknoten)
- Nummer() (Nummer des aktuellen Knotens)
- Wurzel() (Knoten ist Wurzel)
- Blatt() (Knoten ist Blatt)

verwendet werden.

Aufgabe 3:

a) Geben Sie eine BSP-Darstellung des in Bild 7.60 dargestellten ebenen Modells an. Legen Sie die Normalenvektoren fest, und zeichnen Sie die Trenngeraden und Normalenvektoren.

b) Geben Sie den BSP-Baum für dasselbe Objekt an für den Fall, daß

   1. das Objekt um den Winkel α gegen den Uhrzeigersinn um den Ursprung rotiert wurde,
   2. das Objekt um den Vektor (t_x, t_y) transliert wurde.

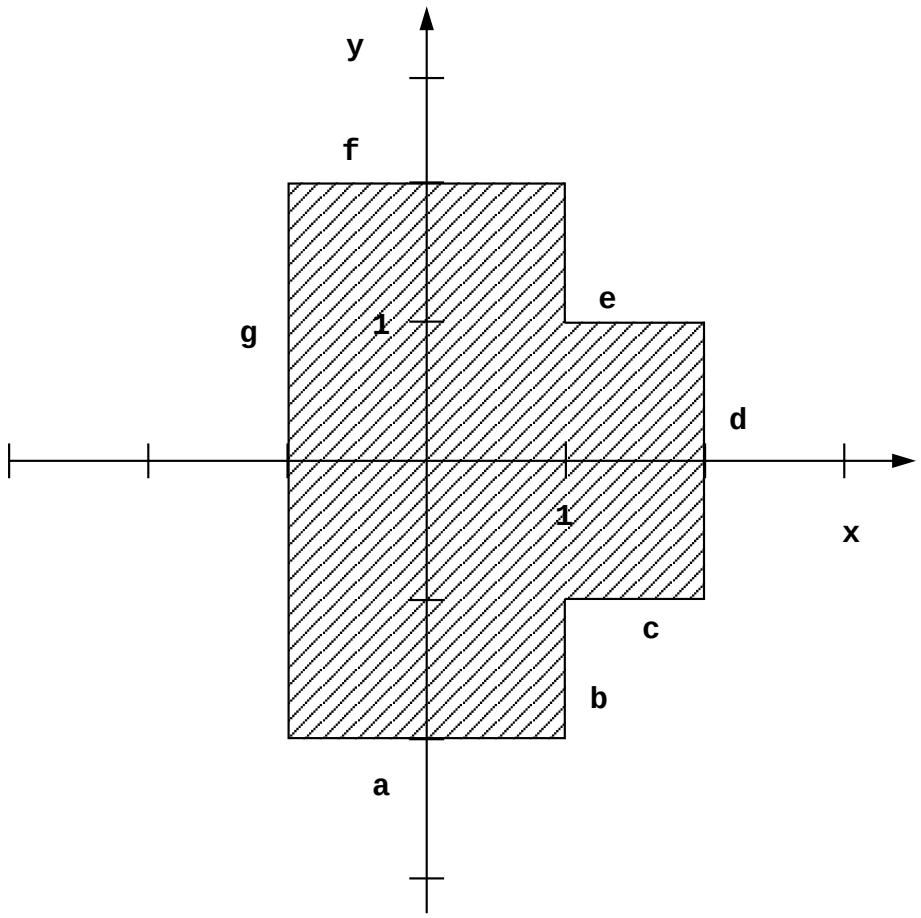


Abbildung 7.60: Ebenes Modell

Aufgabe 4:

a) Die Strahlungsleistung der Sonne beträgt $3,84 \cdot 10^{20}$ Megawatt, und die Entfernung Erde–Sonne beträgt $149,6 \cdot 10^{6}$ Kilometer. Berechnen Sie die Bestrahlungsstärke der Sonne auf die Erde, unter der Annahme, daß die Sonne senkrecht über der betrachteten Fläche steht (z.B. am Äquator, Frühlingsanfang 12.00 Uhr).

b) Berechnen Sie die Strahldichte der Sonne auf der Erde. Der Sonnenradius beträgt $6,965 \cdot 10^{8}$ Meter.

Aufgabe 5:

Historisch sind zuerst Visibilitätsverfahren und dann Beleuchtungsverfahren entwickelt worden. Geben Sie an, warum sich das linienorientierte Visibilitätsverfahren von Watkins gut mit dem Beleuchtungsalgorithmus von Gouraud kombinieren läßt (vgl. Kapitel 5 'Visibilität', 5.3.6.2 'Watkins' Algorithmus'). Begründen Sie Ihre Aussage durch Angabe der benötigten Operationen für die Beseitigung von Verdeckungen bzw. für die Leuchtdichteberechnung.

Aufgabe 6:

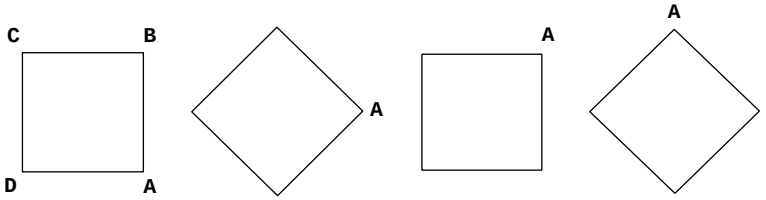


Abbildung 7.61: Verschiedene Lagen des Quadrates *ABCD*

In den Ecken eines Quadrates *ABCD* seien die Leuchtdichten wie folgt bestimmt worden:

$$L_A = L_0, \quad L_B = L_C = L_D = L_1, \quad \text{wobei} \quad 0 \leq L_0 < L_1 \leq 1.$$

Danach wird das Quadrat mittels Gouraud-Shading dargestellt.

a) Skizzieren Sie die Linien konstanter Leuchtdichte für die in Abbildung 7.61 angebenen Lagen des Quadrats (Abtastrichtung horizontal).

b) Angenommen $L_0 = 0$, $L_1 = 1$ und nur drei Intensitätsstufen ($1 =$ weiß, $\frac{1}{2} =$ grau und $0 =$ schwarz) sind möglich. Wie werden dann die vier oben skizzierten Quadrate dargestellt?

Aufgabe 7:

Zur Strahlerzeugung eines Raytracers sei ein einfaches Kameramodell durch folgende Parameter definiert:

E	Augpunkt (eye)
M	Blickbezugspunkt (view-reference-point)
U	Obenrichtung (View-Up-Vektor), muß nicht normalisiert sein
θ	horizontaler Öffnungswinkel des Sichtvolumens
φ	vertikaler Öffnungswinkel des Sichtvolumens
R_x	Anzahl Pixel in x-Richtung (horizontale Auflösung des Bildschirms)
R_y	Anzahl Pixel in y-Richtung (vertikale Auflösung des Bildschirms)

Berechnen Sie aus diesen Angaben den Strahl R, der vom Augpunkt **E** durch das Pixel (x,y) der virtuellen Bildebene geht.

Aufgabe 8:

a) Eine Kugel K sei durch ihren Mittelpunkt **c** und ihren Radius r gegeben. Gegeben sei außerdem ein Strahl D durch seinen Ursprung **o** und seine Einheitsrichtung **d**:

$$D: \qquad \mathbf{x} = \mathbf{o} + s\mathbf{d}, \qquad \mathbf{x}, \mathbf{o}, \mathbf{d} \in \mathbb{R}^3, \quad s \in \mathbb{R}, \quad ||\mathbf{d}|| = 1.$$

Berechnen Sie den vom Ursprung des Strahls aus gesehen ersten Schnittpunkt S zwischen D und K.

b) Berechnen Sie mit Hilfe der Kugelnormalen im Schnittpunkt S die Richtung des reflektierten Strahls.

Aufgabe 9:

a) Eine Ebene E sei durch ihren Einheitsnormalenvektor **n** und ihren Abstand a vom Koordinatenursprung (Hessesche Normalenform) definiert:

$$E: \qquad \mathbf{n}\mathbf{x} + a = 0, \qquad \mathbf{n}, \mathbf{x} \in \mathbb{R}^3, \quad a \in \mathbb{R}, \quad ||\mathbf{n}|| = 1.$$

Gegeben sei außerdem ein Strahl R durch seinen Ursprung **o** (Augpunkt) und seine Einheitsrichtung **d**:

$$R: \qquad \mathbf{x} = \mathbf{o} + s\mathbf{d}, \qquad \mathbf{x}, \mathbf{o}, \mathbf{d} \in \mathbb{R}^3, \quad s \in \mathbb{R}, \quad ||\mathbf{d}|| = 1.$$

Berechnen Sie den Schnittpunkt zwischen R und E.

b) Berechnen Sie, falls vorhanden, den Schnittpunkt eines Strahls $\mathbf{x} = \mathbf{o} + s\mathbf{d}$ mit einem ebenen Polygon P, das durch seine n Eckpunkte $\mathbf{p}_1, \ldots, \mathbf{p}_n \in \mathbb{R}^3$ definiert ist.

Hinweis: Transformieren Sie das Polygon so, daß der Strahl mit der positiven z-Achse übereinstimmt, und projizieren Sie anschließend das rotierte Polygon in die x/y-Ebene.

c) Berechnen Sie, falls vorhanden, den Schnittpunkt eines Strahls $\mathbf{x} = \mathbf{o} + s\mathbf{d}$ mit einem Dreieck D, das durch seine drei Eckpunkte $\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3 \in \mathbb{R}^3$ definiert ist (Hinweis: baryzentrische Koordinaten).

Aufgabe 10:

In dieser Übungsaufgabe sei der in die Szene gesandte Strahl definiert durch seinen Ausgangspunkt **o** und seine Richtung **d**. Die Bezeichnungen sind gleich gewählt wie in Aufgabe 8 und 9.

Die Szene selbst besteht aus einer Kugel mit Zentrum **c** und Radius r sowie einer Ebene E mit Normalenvektor **n** mit der Ebenengleichung $\mathbf{nx} + a = 0$.

Weiterhin sei eine Lichtquelle mit Position **l** vorhanden. Berechnen Sie die erste Hierarchieebene des durch den Strahl (**o**,**d**) aufgespannten Strahlbaumes, indem Sie den Schnitt des Strahls mit einem Objekt der Szene sowie den Reflektionsstrahl und die Schattenstrahlen bestimmen.

Beachten Sie auch, daß Schattenstrahlen von Lichtquellen ihrerseits durch Objekte geschnitten werden können. Die einzelnen Vektoren und Punkte seien wie folgt gegeben:

$$\mathbf{o} = (0, 0, 5)$$
$$\mathbf{d} = (1, 1, 0)$$
$$\mathbf{c} = (4, 4, 4)$$
$$r = 3$$
$$\mathbf{l} = (0, 0, 10)$$
$$\mathbf{n} = (0, 1, 0)$$
$$a = -6$$

7.15 Lösungen zu den Übungsaufgaben

Lösung 1:

a)

$$
\begin{aligned}
f_1 &: v_1\, v_2\, v_3\, v_4\\
f_2 &: v_5\, v_6\, v_2\, v_1\\
f_3 &: v_2\, v_6\, v_7\, v_3\\
f_4 &: v_3\, v_7\, v_8\, v_4\\
f_5 &: v_8\, v_5\, v_1\, v_4\\
f_6 &: v_6\, v_5\, v_8\, v_7
\end{aligned}
$$

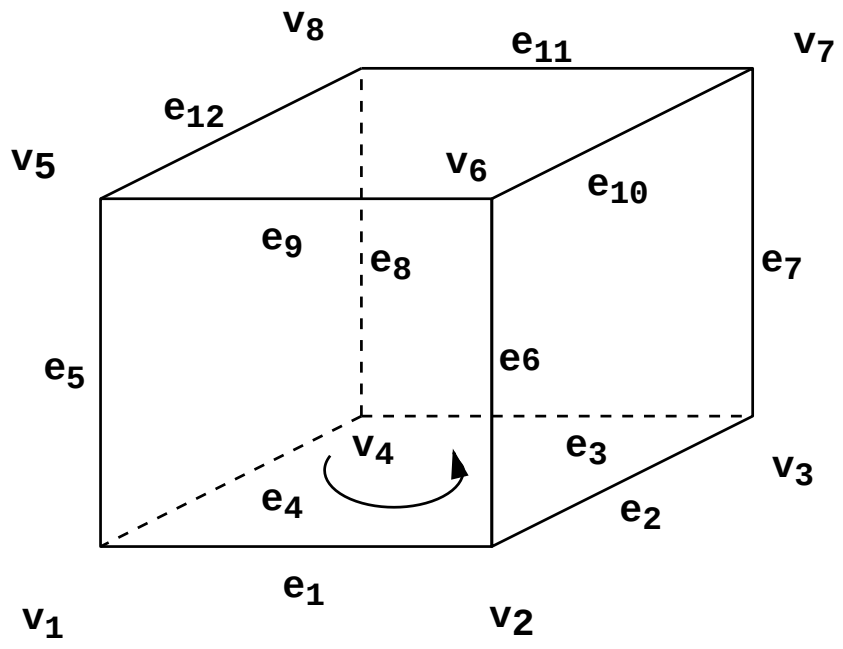


Abbildung 7.62: Skizze des Quaders

b)
```
Für alle Flächen in der Flächenliste
{
  Hilfs_sign = sign
  if (Hilfs_sign == +)
  {
    Addiere vstart(StartKante) zu Knoten(akt. Fläche)
    (d.h. vstart wird in die Knotenliste der aktuellen
    Fläche einsortiert.)
    Hilfs_v = vend(StartKante)
    Hilfs_Kante = ncw(StartKante)
  }
  else
  {
    Addiere vend(StartKante) zu Knoten(akt. Fläche)
    Hilfs_v = vstart(StartKante)
    Hilfs_Kante = nccw(StartKante)
```

```
  }
  Solange Hilfs_Kante ≠ StartKante
  {
    if ((Hilfs_sign == +) und (Hilfs_v ≠ vstart(Hilfs_Kante)))
        Hilfs_sign = -
    if ((Hilfs_sign == -) und (Hilfs_v ≠ vend(Hilfs_Kante)))
        Hilfs_sign = +
    if (Hilfs_sign == +)
    {
        Addiere vstart(Hilfs_Kante) zu Knoten(akt. Fläche)
        Hilfs_v = vend(Hilfs_Kante)
        Hilfs_Kante = ncw(Hilfs_Kante)
    }
    else
    {
        Addiere vend(Hilfs_Kante) zu Knoten(akt. Fläche)
        Hilfs_v = vstart(Hilfs_Kante)
        Hilfs_Kante = nccw(StartKante)
    }
  }
}
```

Lösung 2

a)
```
Tiefe = 1
Mittelpunktx = l/2
Mittelpunkty = l/2
Solange kein Blatt
{
  Tiefe = Tiefe + 1
  if (x ≤ Mittelpunktx)
  {
    Mittelpunktx = Mittelpunktx - 1/2^Tiefe · l
    if (y ≤ Mittelpunkty)
    {
      Kind(0)
      Mittelpunkty = Mittelpunkty - 1/2^Tiefe · l
    }
    else
    {
      Kind(2)
      Mittelpunkty = Mittelpunkty + 1/2^Tiefe · l
    }
  }
  else
  {
    Mittelpunktx = Mittelpunktx + 1/2^Tiefe · l
    if (y ≤ Mittelpunkty)
    {
      Kind(1)
      Mittelpunkty = Mittelpunkty - 1/2^Tiefe · l
    }
    else
    {
      Kind(3)
      Mittelpunkty = Mittelpunkty + 1/2^Tiefe · l
    }
  }
}
```

b) Um den Nachbarn in Richtung N zu finden, verwenden wir folgenden Algorithmus:
Knoten, deren Nummer im Vaterknoten 0 oder 1 ist, besitzen als nördliche Nachbarn die Geschwisterknoten mit Nummer 2 bzw. 3 im Vaterknoten.

Für Knoten, deren Nummer im Vaterknoten 2 oder 3 ist, gehen wir im Quadtree solange in Richtung der Wurzel, bis ein Vaterknoten Nummer 0 oder 1 besitzt. Dann gehen wir in Richtung des an der x-Achse gespiegelten Herkunftspfades ebenso oft nach unten, wie wir nach oben gegangen sind oder bis wir ein Blatt erreicht haben (vgl. Bild 7.63).

Falls im Herkunftspfad kein Knoten mit Nummer 0 oder 1 erreicht wird, gibt es keinen Nachbarn in N-Richtung. Dasselbe gilt für den Fall, daß beim Durchlaufen eines Baumes in umgekehrter Richtung ein Knoten zwar Kinder besitzt, aber nicht in der mit dem Herkunftspfad korrespondierenden Richtung.

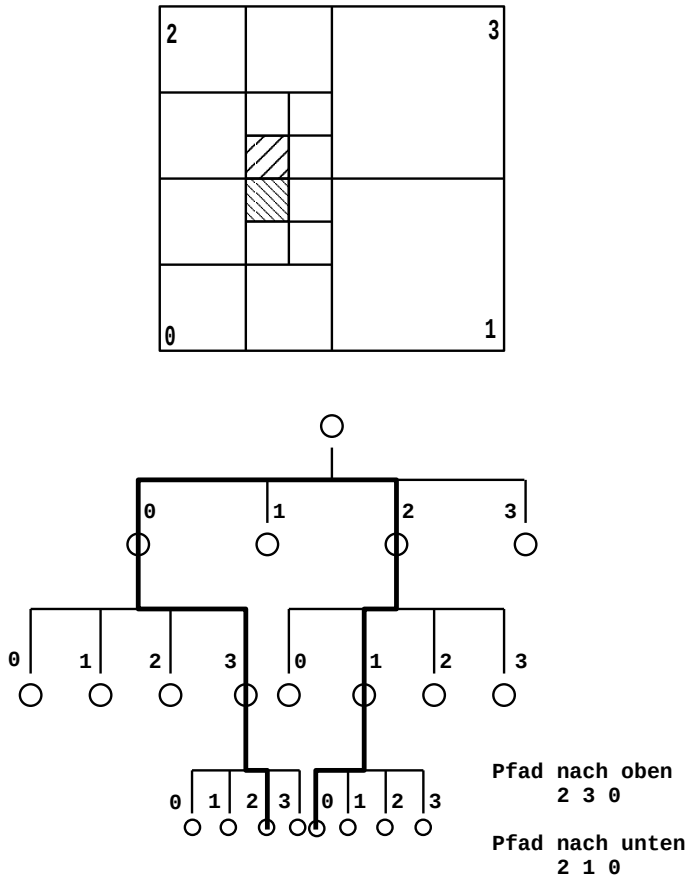


Abbildung 7.63: Graphische Veranschaulichung

```
if (Wurzel())        // kein Nachbar
  return
else
{
  Solange Pfad[Pfadlänge] ≠ 0 und Pfad[Pfadlänge] ≠ 1
  und nicht Wurzel()        // nach oben gehen
  {
    Pfadlänge = Pfadlänge + 1
    Pfad[Pfadlänge] = Nummer()
    Vater()
  }
  if (Wurzel() ≠ 0 und Pfad[Pfadlänge] ≠ 0
      und Pfad[Pfadlänge] ≠ 1)
    kein Nachbar
  else
  {
    Solange Pfadlänge ≠ 0 und nicht Blatt()
    {
      if Pfad[Pfadlänge] ==
      {
        0 : Kind(2)
           (falls nicht vorhanden: kein Nachbar)
        1 : Kind(3)
           (falls nicht vorhanden: kein Nachbar)
        2 : Kind(0)
           (falls nicht vorhanden: kein Nachbar)
```

```
            3 : Kind(1)
               (falls nicht vorhanden: kein Nachbar)
            Pfadlänge = Pfadlänge - 1
          }
        }
      }
    }
```

Lösung 3:

a) Für die Gleichungen der Geraden ergibt sich:

	Gleichung			Normale
a	$-y-2$	$=$	0	$(0,-1)$
b	$x-1$	$=$	0	$(1,0)$
c	$-y+1$	$=$	0	$(0,-1)$
d	$x-2$	$=$	0	$(1,0)$
e	$y-1$	$=$	0	$(0,1)$
f	$y-2$	$=$	0	$(0,1)$
g	$-x-1$	$=$	0	$(-1,0)$

Als BSP-Baum erhalten wir:

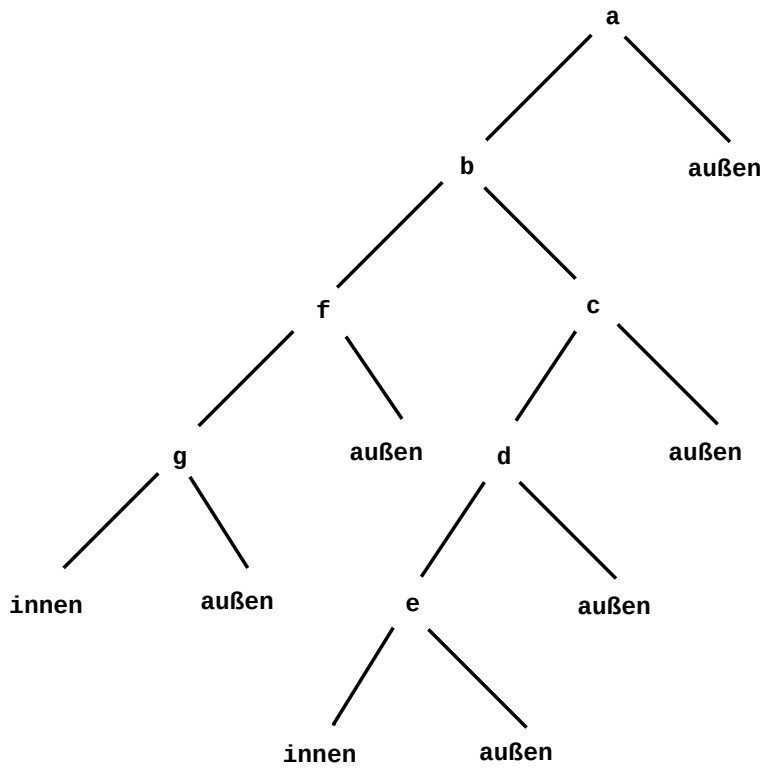


Abbildung 7.64: BSP-Baum

b) Eine Rotation um den Winkel α gegen den Uhrzeigersinn wird durch die

homogene Matrix

$$R(\alpha) = \begin{pmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

beschrieben. Da sich transponierte Normalen mit der inversen Matrix transformieren (vgl. Kurseinheit 3, Abschnitt 3.3.3 'Transformation von Normalen') und diese im Fall der Rotation mit der transponierten Matrix übereinstimmt, ergibt sich:

$$a' : (x,y,1) \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ -1 \\ -2 \end{pmatrix} = x\sin\alpha - y\cos\alpha - 2 = 0$$

$$b' : (x,y,1) \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ -1 \end{pmatrix} = x\cos\alpha + y\sin\alpha - 1 = 0$$

$$c' : (x,y,1) \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ -1 \\ 1 \end{pmatrix} = x\sin\alpha - y\cos\alpha + 1 = 0$$

$$d' : (x,y,1) \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ -2 \end{pmatrix} = x\cos\alpha + y\sin\alpha - 2 = 0$$

$$e' : (x,y,1) \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ -1 \end{pmatrix} = -x\sin\alpha + y\cos\alpha - 1 = 0$$

$$f' : (x,y,1) \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ -2 \end{pmatrix} = -x\sin\alpha + y\cos\alpha - 2 = 0$$

$$g' : (x,y,1) \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} -1 \\ 0 \\ -1 \end{pmatrix} = -x\cos\alpha + y\sin\alpha - 1 = 0$$

Eine Translation um den Vektor (t_x, t_y) wird durch die homogene Matrix

$$T = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{pmatrix}$$

beschrieben.
Die Inverse ist

$$T^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -t_x & -t_y & 1 \end{pmatrix}$$

Daraus ergibt sich

$$a' : (x,y,1) \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -t_x & -t_y & 1 \end{pmatrix} \begin{pmatrix} 0 \\ -1 \\ -2 \end{pmatrix} = -y + t_y - 2 = 0$$

$$b' : (x,y,1) \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -t_x & -t_y & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ -1 \end{pmatrix} = x - t_x - 1 = 0$$

$$c' : (x,y,1) \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -t_x & -t_y & 1 \end{pmatrix} \begin{pmatrix} 0 \\ -1 \\ 1 \end{pmatrix} = -y + t_y + 1 = 0$$

$$d' : (x,y,1) \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -t_x & -t_y & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ -2 \end{pmatrix} = x - t_x - 2 = 0$$

$$e' : (x,y,1) \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -t_x & -t_y & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ -1 \end{pmatrix} = y - t_y - 1 = 0$$

$$f' : (x,y,1) \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -t_x & -t_y & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ -2 \end{pmatrix} = y - t_y - 2 = 0$$

$$g' : (x,y,1) \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -t_x & -t_y & 1 \end{pmatrix} \begin{pmatrix} -1 \\ 0 \\ -1 \end{pmatrix} = -x + t_x - 1 = 0$$

Die Struktur des BSP-Baumes ändert sich durch die Transformationen nicht.

Lösung 4:

a) Die Sonne strahlt in jede Richtung gleich ab, d.h. die Intensität der Sonne ist unabhängig von der Abstrahlrichtung. Für die Intensität I_S der Sonne ergibt sich mit Gleichung (7.6)

$$\begin{aligned} I_S &= \frac{\varphi_S}{\text{Oberfläche der Einheitskugel}} \\ &= \frac{3{,}84 \cdot 10^{20} MW}{4\pi sr} \\ &= 305{,}577 \cdot 10^{20} \frac{KW}{sr} \end{aligned}$$

Bezeichnen wir mit R_{ES} den Abstand zwischen Erde und Sonne, so erhalten wir die Bestrahlungsstärke E_S der Sonne auf die Erde

$$\begin{aligned} E_S &= I_S \cdot (\text{ Raumwinkel, der 1 } m^2 \text{ auf der Erde entspricht}) \\ &= I_S \cdot \frac{1sr}{R_{ES}^2} \\ &= 305{,}577 \cdot 10^{20} \frac{\text{KW}}{sr} \cdot \frac{1}{(1496 \cdot 10^8)^2 m^2} sr \\ &= 1{,}365 \frac{\text{KW}}{m^2} \end{aligned}$$

(vgl. Gleichung 7.7).

b) Mit Gleichung 7.10 ergibt sich mit r_s = Sonnenradius:

$$\begin{aligned} I_s &= L_s \int_{Sonnenhalbkugel} \cos\vartheta_1 \, dA_1 \\ &= L_s \int_{\vartheta_1=0}^{\pi/2} \int_{\varphi=0}^{2\pi} \cos\vartheta_1 \, r_s^2 \, \sin\vartheta_1 \, d\vartheta_1 \, d\varphi \\ &= L_s \, \pi \, r_s^2 \end{aligned}$$

(Anschaulich:
Das Integral beschreibt die Projektion der Sonnenhalbkugel auf eine Ebene durch den Sonnenmittelpunkt.)

Damit ergibt sich L_s zu

$$L_s = \frac{I_s}{\pi \gamma_s^2} = 20{,}045 \frac{MW}{m^2 sr}$$

Lösung 5:

Das Visibilitätsverfahren von Watkins läßt sich gut mit dem Beleuchtungsalgorithmus von Gouraud kombinieren, da beide Algorithmen von den Informationen in den Polygonecken ausgehen und entlang der Polygonkanten und der Rasterlinien interpolieren.
Stehen die Steigungen in x- und y-Richtung für Farb(Grau)- und Koordinatenwerte zur Verfügung, so kann wegen Schrittweite 1 in beiden Richtungen inkrementell gearbeitet werden. Die für den Watkins-Algorithmus aufgebaute Datenstruktur mit Kantenliste und aktiver Kantenliste pro Rasterlinie kann leicht um die erforderliche Information für die Farbberechnung erweitert werden. Zum Beispiel wird zur x-Koordinate des oberen Endes und zur y-Koordinate des unteren Endes der Beleuchtungswert hinzugefügt. Weiterhin wird der Gradient des Farb(Grau)-Werts in x- bzw. y-Richtung gespeichert.

Vorteile der Kombination dieser beiden Verfahren sind:

- Inkrementelle Arbeitsweise. Eine Hardware-Realisierung des Watkins-Algorithmus ist leicht um identische Baugruppen für den Gouraud-Algorithmus erweiterbar und sequentiell nutzbar.
- Da zuerst die Verdeckungen beseitigt werden, wird nur für sichtbare Teile die Farbberechnung vorgenommen.

Lösung 6:

a) Die Quadrate mit den eingezeichneten Linien konstanter Leuchtdichte sind in Bild 7.65 dargestellt.

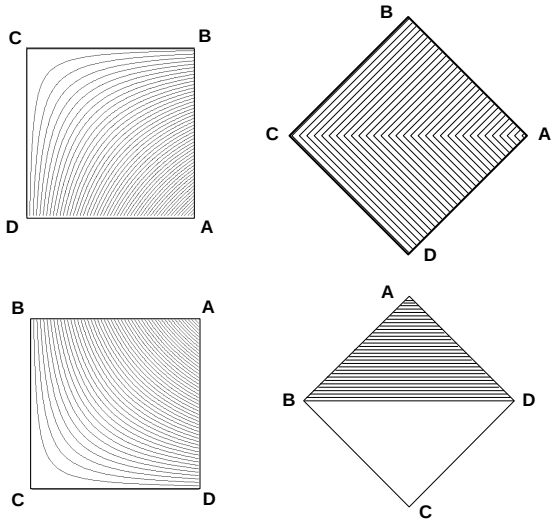


Abbildung 7.65: Linien konstanter Leuchtdichte

b) Die Quadrate mit den drei Leuchtdichten sind in Bild 7.66 dargestellt.

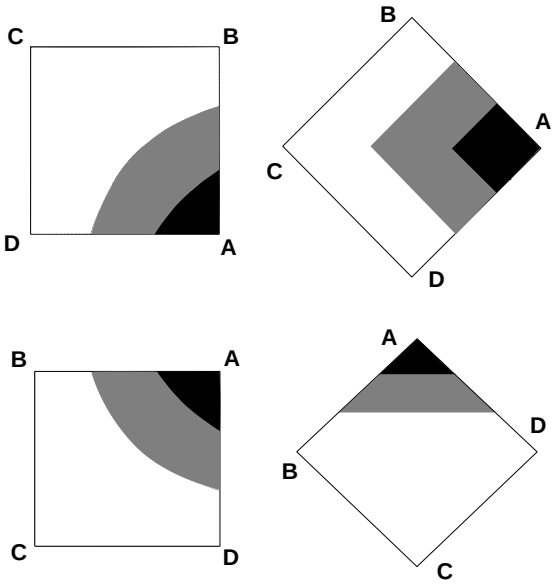


Abbildung 7.66: Darstellung mit drei Leuchtdichten

Lösung 7:

In Anlehnung an [Gla95] könnte die Lösung wie in Bild 7.67 aussehen.

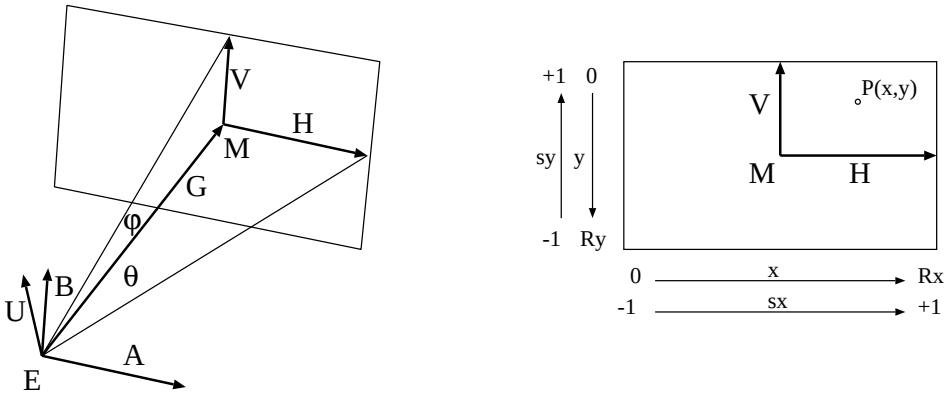


Abbildung 7.67: Raytracing: Berechnung des Strahls R

Für die interne Darstellung der Kamera benötigen wir die aufspannenden Vektoren **V** und **H**. Wie aus dem Bild leicht abzulesen ist, können diese wie folgt aus den gegebenen Kameraparametern berechnet werden:

$$\mathbf{G} = \mathbf{M} - \mathbf{E}, \quad \mathbf{A} = \frac{\mathbf{G} \times \mathbf{U}}{||\mathbf{G} \times \mathbf{U}||}, \quad \mathbf{B} = \frac{\mathbf{A} \times \mathbf{G}}{||\mathbf{A} \times \mathbf{G}||}$$

und damit

$$\mathbf{H} = \mathbf{A} \cdot ||\mathbf{G}|| \cdot \tan\theta, \quad \mathbf{V} = \mathbf{B} \cdot ||\mathbf{G}|| \cdot \tan\varphi.$$

Beachten Sie, daß die Vektoren **U**, **B**, **G** und **V** in einer Ebene liegen. Diese Ebene wird eindeutig durch die Kameraparameter **E**, **M** und **U** definiert.

Als nächstes wollen wir den Punkt $\mathbf{P}(x,y)$ in der Bildebene (**M**,**H**,**V**) berechnen, der durch seine Koordinaten (*x*,*y*) und die Kameraparameter eindeutig definiert ist. Wenn wir von der Annahme ausgehen, daß der Ursprung (0,0) in der linken oberen Bildschirmecke liegt, dann lautet die Lösung

$$\mathbf{P} = \mathbf{M} + s_x\mathbf{H} + s_y\mathbf{V} \qquad \text{mit} \qquad s_x = 2\frac{x}{R_x} - 1, \quad s_y = 1 - 2\frac{y}{R_y}\,.$$

Der Strahl verläuft somit vom Augpunkt **E** durch den Punkt $\mathbf{P}(x,y)$. Seine normalisierte Richtung ist also

$$\mathbf{D} = \frac{\mathbf{P} - \mathbf{E}}{||\mathbf{P} - \mathbf{E}||}.$$

Lösung 8:

a) Ein Punkt $\mathbf{x} = \mathbf{o} + s\mathbf{d}$ mit $s \geq 0$ ist genau dann Schnittpunkt des Strahls D mit der Kugel K, wenn die Gleichung

$$(\mathbf{o} + s\mathbf{d} - \mathbf{c})(\mathbf{o} + s\mathbf{d} - \mathbf{c}) = r^2$$

erfüllt ist. Durch Umformung erhält man

$$s^2(\mathbf{dd}) + 2s(\mathbf{o} - \mathbf{c})\mathbf{d} + (\mathbf{o} - \mathbf{c})(\mathbf{o} - \mathbf{c}) - r^2 = 0.$$

Wegen $\mathbf{dd} = 1$ ergibt sich

$$s_1 = -(\mathbf{o} - \mathbf{c})\mathbf{d} - \sqrt{Discr}$$

bzw.

$$s_2 = -(\mathbf{o} - \mathbf{c})\mathbf{d} + \sqrt{Discr}$$

mit

$$Discr := ((\mathbf{o} - \mathbf{c})\mathbf{d})^2 - (\mathbf{o} - \mathbf{c})(\mathbf{o} - \mathbf{c}) + r^2.$$

Ein Schnittpunkt des Strahls D mit der Kugel existiert also genau dann, wenn $Discr \geq 0$ und $s_2 \geq 0$ ist.
Der erste Schnittpunkt S ist dann gegeben durch

$$S := \left\{ \begin{array}{lll} \mathbf{o} + s_1\mathbf{d}, & falls & s_1 \geq 0. \\ \mathbf{o} + s_2\mathbf{d}, & falls & s_1 < 0. \end{array} \right\}$$

b) Wir zerlegen den Vektor $\mathbf{d}$ in zwei Komponenten $\mathbf{d} = \mathbf{d_p} + \mathbf{d_s}$, so daß $\mathbf{d_p}$ parallel und $\mathbf{d_s}$ senkrecht zur Normalen $\mathbf{n}$ ist (vgl. 7.68).

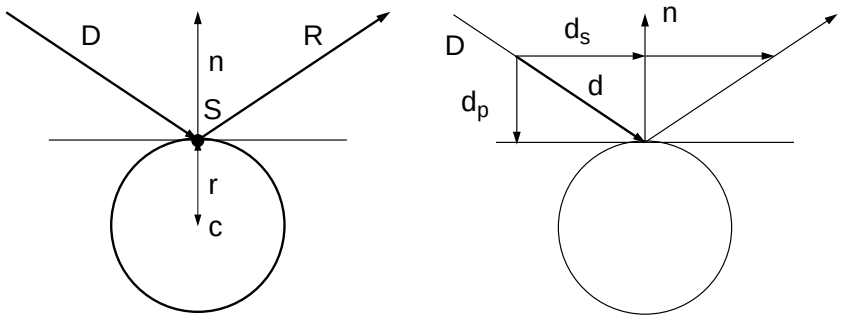


Abbildung 7.68: Berechnung der Richtung des reflektierten Strahls

Die Richtung des reflektierten Strahls ist dann gegeben durch

$$\mathbf{d}_{refl} = \mathbf{d} - \mathbf{2d_p}.$$

mit

$$\mathbf{d_p} = (\mathbf{dn})\mathbf{n}.$$

Lösung 9:

a) Ein Punkt $\mathbf{x}$ liegt in der Ebene E, falls $\mathbf{nx} + a = 0$.

Ein Punkt $\mathbf{x}$ liegt auf dem Strahl R, falls $\mathbf{x} = \mathbf{o} + s\mathbf{d}$.

Also liegt der Schnittpunkt $\mathbf{x}$ im Abstand s vom Ursprung $\mathbf{o}$ des Strahls, falls

$$\mathbf{n}(\mathbf{o} + s\mathbf{d}) + a = 0 \qquad \Longrightarrow \qquad s = -\frac{\mathbf{no} + a}{\mathbf{nd}}.$$

s ist nicht definiert, falls $\mathbf{nd} = 0$. Dies ist genau dann der Fall, wenn der Strahl R parallel zur Ebene E verläuft, d.h., ein Schnittpunkt $\mathbf{x}$ existiert nicht.

Beachten Sie, daß $s < 0$ werden kann, nämlich genau dann, wenn der Schnittpunkt in entgegengesetzter Strahlrichtung liegt. Beim Raytracing sind wir jedoch nur an Schnittpunkten in positiver Strahlrichtung ($s > 0$) interessiert.

b) a) Translation des Augpunktes o in den Ursprung:

$$\text{Translationsmatrix } T = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -T_x & -T_y & -T_z & 1 \end{pmatrix}.$$

b) Rotation der Strahlrichtung auf die z-Achse:

$$\text{Rotationsmatrix } R = \begin{pmatrix} e'_{xx} & e'_{xy} & e'_{xz} & 0 \\ e'_{yx} & e'_{yy} & e'_{yz} & 0 \\ e'_{zx} & e'_{zy} & e'_{zz} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

mit

$$\begin{aligned} e'_z &:= d \\ e'_x &:= \tfrac{e_y \times d}{\|e_y \times d\|},\ e_y \neq \alpha d \\ e'_y &:= d \times e'_x. \end{aligned}$$

Dabei bezeichnen e_x, e_y und e_z die Einheitsvektoren in x-, y- bzw. z-Richtung und e'_x, e'_y und e'_z die Bilder dieser Einheitsvektoren unter R^{-1}.

Falls $e_y = \alpha d$, so wählen Sie:

$$\begin{aligned} e'_x &:= e_x \\ e'_z &:= d \\ e'_y &:= d \times e_x = -e_z. \end{aligned}$$

c) Projektion in die x/y-Ebene:

$$\text{Projektionsmatrix } P = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

d) Mittels der Matrix $T \cdot R \cdot P$ werden die Eckpunkte des Polygons in die x/y-Ebene transformiert. Aufgrund der so gewählten Transformation wird der Schnittpunkt des Strahls mit der durch das Polygon definierten Ebene auf den Ursprung abgebildet.

e) Nun wird festgestellt, ob der Ursprung innerhalb des transformierten Polygons liegt. Dazu werden alle Polygonkanten auf Schnitt mit der positiven x-Achse getestet (vgl. GDV I, Abschnitt 5.3.3 'Punktklassifizierung'). Ist die Anzahl der Schnittpunkte gerade, so liegt der Schnittpunkt im Polygon, sonst außerhalb.

f) Liegt der Schnittpunkt innerhalb des Polygons, so wird der Schnittpunkt des Strahls mit der durch das Polygon definierten Ebene bestimmt:
Dazu wird die Hessesche Normalform der Ebene bestimmt und dann Aufgabenteil a) verwendet. Die Normale wird mit Hilfe von Newell bestimmt (vgl. GDV II, Abschnitt 1.3.9 'Algorithmen für Boundary-Modelle', Formel 1.7):

$$n = \frac{\sum_{i=1}^{n}(p_i - p_{i+1}) \times (p_i + p_{i+1})}{||\sum_{i=1}^{n}(p_i - p_{i+1}) \times (p_i + p_{i+1})||},$$

wobei $p_{n+1} = p_1$ ist.

Der Abstand a der Ebene zum Ursprung berechnet sich dann durch Einsetzen eines Punktes in die Ebenengleichung

$$\begin{array}{rrcl} & np_1 + a & = & 0 \\ \Leftrightarrow & a & = & -np_1. \end{array}$$

Daraus erhalten wir die Ebenengleichung: $np + a = 0$.

c) Die Lösungsidee lautet wie folgt: Man berechne zunächst den Schnittpunkt des Strahls mit der Dreiecksebene und prüfe dann, ob dieser Schnittpunkt innerhalb des Dreiecks liegt.

Diese Idee läßt sich wie folgt realisieren:

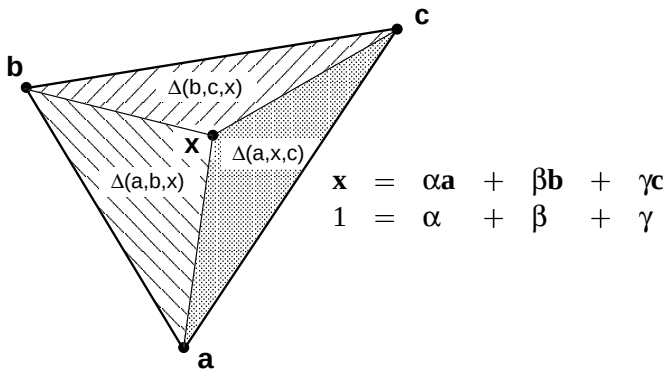
1.) Schnittpunkt des Strahls mit einer Dreiecksebene

Dieses Problem haben wir bereits in Teilaufgabe a) gelöst. Der Normalenvektor n und a werden, wie in Teil b) allgemein gezeigt, berechnet.

2.) Liegt der Schnittpunkt im Dreieck?

Wir wollen dieses Problem durch Ausnutzung des folgenden Hauptsatzes der affinen Geometrie lösen:

Eine Ebene E sei durch die drei nichtkollinearen Punkte $\mathbf{a}, \mathbf{b}, \mathbf{c} \in \mathbf{R}^3$ definiert. Dann existieren für jeden Punkt $\mathbf{x} \in E$ *baryzentrische Koordinaten* (α, β, γ), so daß gilt:



Liegt **x** innerhalb des Dreiecks (**a**, **b**, **c**), so gilt stets: $0 \leq \alpha \leq 1$, $0 \leq \beta \leq 1$ und $0 \leq \gamma \leq 1$.

Liegt **x** außerhalb des Dreiecks (**a**, **b**, **c**), so ist mindestens einer der baryzentrischen Koordinaten α, β, γ kleiner als Null.

Die konkrete Berechnung der baryzentrischen Koordinaten geschieht über Flächenverhältnisse:

$$\alpha = \frac{\Delta(\mathbf{b},\mathbf{c},\mathbf{x})}{\Delta(\mathbf{a},\mathbf{b},\mathbf{c})}, \qquad \beta = \frac{\Delta(\mathbf{a},\mathbf{x},\mathbf{c})}{\Delta(\mathbf{a},\mathbf{b},\mathbf{c})}, \qquad \gamma = \frac{\Delta(\mathbf{a},\mathbf{b},\mathbf{x})}{\Delta(\mathbf{a},\mathbf{b},\mathbf{c})} = 1 - \alpha - \beta.$$

Liegen drei Punkte **a**, **b**, **c** in der (*x*,*y*)-Ebene, so berechnet man $\Delta(\mathbf{a},\mathbf{b},\mathbf{c})$ durch

$$\Delta(\mathbf{a},\mathbf{b},\mathbf{c}) = (\mathbf{b}_x - \mathbf{a}_x) \cdot (\mathbf{c}_y - \mathbf{a}_y) - (\mathbf{b}_y - \mathbf{a}_y) \cdot (\mathbf{c}_x - \mathbf{a}_x).$$

Dies entspricht genau dem doppelten Flächeninhalt des Dreiecks (**a**,**b**,**c**).

Zur Lösung unseres Problems brauchen wir also nur noch die baryzentrischen Koordinaten des Schnittpunktes **x** bezüglich des Dreiecks $(\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3)$ berechnen und feststellen, ob $0 \leq \alpha \leq 1$ und $0 \leq \beta \leq 1$ gilt. Die Bedingung $0 \leq \gamma \leq 1$ gilt dann automatisch.

Um Rechenzeit zu sparen, reduzieren wir das 3D-Problem jedoch zunächst einmal auf ein 2D-Problem, indem wir das Dreieck auf eine der Hauptebenen (*xy*, *xz* oder *yz*) projizieren. Dies erfolgt durch Nullsetzen derjenigen *x*-, *y*- oder *z*-Komponenten in den Punkten $\mathbf{p}_1$, $\mathbf{p}_2$, $\mathbf{p}_3$ und **x**, bei denen der Normalenvektor der Dreiecksebene ein Maximum hat.

Lösung 10:

Zunächst wird der erste Schnittpunkt des Strahles

$$D: \qquad \mathbf{x} = \mathbf{o} + s\,\mathbf{d},\ x \in \mathbb{R}^3,\ s \in \mathbb{R} \tag{7.103}$$

mit der Kugel berechnet (vgl. Übungsaufgabe 3a)).
Für s ergeben sich die Werte $s_1 = 2$ und $s_2 = 6$.

Schneiden wir D mit der Ebene E (vgl. Übungsaufgabe 4a)), so erhalten wir $s = 6$.

Damit sind der Schnittpunkt mit der Ebene sowie der zweite Kugelschnittpunkt weiter vom Ausgangspunkt **o** als der erste Kugelschnittpunkt entfernt und daher unsichtbar.

Von den zwei Kugelschnittpunkten ist der in Strahlrichtung näher an **o** gelegene mit $s_1 = 2$ sichtbar.

Einsetzen von s_1 in (7.103) liefert den Schnittpunkt

$$\mathbf{ks} = (2, 2, 5).$$

Anschließend wird im Kugelschnittpunkt **ks** die Kugelnormale **kn** sowie der Reflektionsstrahl **r** wie in Übungsaufgabe 3b) bestimmt.

Wir erhalten als Ergebnis die Vektoren

$$\mathbf{kn} = (-0.66666,\ -0.66666,\ 0.33333)$$

und

$$\mathbf{r} = (-0.54995,\ -0.54995,\ 0.62853).$$

Als nächstes wird der Weg des reflektierten Strahles

$$R: \qquad \mathbf{x} = \mathbf{ks} + t\,\mathbf{r},\ x \in \mathbb{R}^3,\ t \in \mathbb{R}, \tag{7.104}$$

verfolgt.

In Reflektionsrichtung **r** ergibt sich ein Schnittpunkt des reflektierten Strahles mit der Ebene wie in Übungsaufgabe 4a) beschrieben. In (7.104) gilt für diesen Schnittpunkt $t = -7.2734$. Damit liegt der Schnittpunkt in negativer Strahlrichtung und der reflektierte Strahl schneidet die Ebene nicht.

Auch beim Test des reflektierten Strahles mit der Kugel zeigt sich, daß in Strahlrichtung kein Punkt der Kugel geschnitten wird. Für t ergeben sich die Werte $t_1 = 0$ und $t_2 = -5.6570$ ($t_1 = 0$ hat als Schnittpunkt den Strahlursprung *ks* von (7.104) zur Folge und wird deshalb als Schnittpunkt verworfen).

Als letztes muß noch geprüft werden, ob die Lichtquelle **l** von der Kugel oder der Ebene E verdeckt wird.

Ausgehend von der Lichtquelle **l** wird dazu der Schattenstrahl S zum Kugelschnittpunkt **ks** berechnet. Dieser bestimmt sich aus dem Differenzvektor **ss** von **ks** und **l** als

$$S: \qquad \mathbf{x} = \mathbf{l} + u\,\mathbf{ss},\ x \in \mathbb{R}^3,\ u \in \mathbb{R}, \tag{7.105}$$

mit

$$\mathbf{ss} = (2, 2, -5).$$

Im Fall der Ebene ergibt sich bei der Schnittberechnung in (7.105) der Wert von u als $u = 3$.

Beim Schnitt von S mit der Kugel erhalten wir die Werte $u_1 = 1.7879$ und $u_2 = 1$. Damit ist in dem zu u_2 gehörenden Schnittpunkt die Lichtquelle **l** sichtbar, da er in Strahlrichtung am nächsten an **l** liegt. Wird u_2 in (7.105) eingesetzt, so ergibt sich als Ergebnis **ks**. Damit ist die Lichtquelle **l** vom Kugelschnittpunkt **ks** aus zu sehen.

Literaturverzeichnis

[Ben82] S. A. Benton, *Survey of Holographic Stereograms*, Proceedings of SPIE, Vol. 8, 1982, 15–19.

[Bli87] Jim Blinn, *What, Teapots Again?*, IEEE CG&A, Vol. 9, 1987, 61–63.

[BS93] Bergmann, Schaefer, *Lehrbuch der Experimentalphysik, Bd. 3: Optik*, 9. Auflage, de Gruyter, 1993.

[CCC87] R. L. Cook, L. Carpenter , E. Catmull, *The Reyes Image Rendering Architecture*, ACM Computer Graphics, Vol. 21, 1987, Nr. 4, 95–102.

[CCI82] CCIR, *International Radio Consultative Committee, Recommendation 500-2, Kapitel Method for the Subjective Assessment of the Quality of Television Pictures*, 1982.

[CCWG88] M. F. Cohen, S. Chen, J. R. Wallace , D. P. Greenberg, *A Progressive Refinement Approach to Fast Radiosity Image Generation*, ACM Computer Graphics, Vol. 22, 1988, Nr. 4, 75–84.

[CG85] M. F. Cohen, D. P. Greenberg, *The Hemi-Cube: A Radiosity Solution for Complex Environments*, ACM Computer Graphics, Vol. 19, 1985, Nr. 3, 31–40.

[Coo86] R. L. Cook, *Stochastic sampling in computer graphics*, ACM Transactions on Graphics, Vol. 18, 1986, Nr. 3, 51–72.

[CW93] M. F. Cohen, J. R. Wallace, *Radiosity and Realistic Image Synthesis*, First Ed., Academic Press, 1993.

[Duf79] T. Duff, *Smoothly Shaded Renderings of Polyhedral Objects on Raster Displays*, ACM Computer Graphics, Vol. 13, 1979, Nr. 2, 270–275.

[FvDFH90] J. D. Foley, A. van Dam, S. T. Feiner , J. F. Hughes, *Computer Graphics - Principles and Practice*, Addison–Wesley, 1990.

[Gla95] A. S. Glassner, *Principles of Digital Image Synthesis*, Morgan Kaufmann, 1995.

[Gou71] H. Gouraud, *Continuous Shading of Curved Surfaces*, IEEE Transactions on Computers, Vol. C-20, 1971, Nr. 6, 623–628.

[Gra53] H. Graßmann, *Zur Theorie der Farbmischung*, Poggendorffs Annalen der Physik, Vol. 89, 1853.

[GTGB84] C. M. Goral, K. E. Torrance, D. P. Greenberg , B. Battaile, *Modeling the Interaction of Light between Diffuse Surfaces*, ACM Computer Graphics, Vol. 18, 1984, Nr. 3, 213–222.

[Hä81] M. Häusing, *Über die Farbkodierung von Informationen auf elektronischen Sichtgeräten*, Fortschritt der VDI-Zeitschriften, 1981, Nr. 10.

[Hai89] E. Haines, *Essential Ray Tracing Algorithms*, An Introduction to Ray Tracing (Andrew S. Glassner, Ed.), Academic Press, 1989, 33–77.

[Hal88] R. A. Hall, *Illumination and Color in Computer Generated Imagery*, Monographs in Visual Communications, Springer-Verlag, 1988.

[Han89] P. Hanrahan, *A Survey of Ray-Surface Intersection Algorithms*, An Introduction to Ray Tracing (Andrew S. Glassner, Ed.), Academic Press, 1989, 79–119.

[HG83] R. A. Hall, D. P. Greenberg, *A Testbed for Realistic Image Synthesis*, IEEE Computer Graphics & Applications, 1983, Nr. 11, 10–20.

[HG86] E. A. Haines, D. P. Greenberg, *The Light Buffer: A Shadow-Testing Accelerator*, IEEE Computer Graphics & Applications, 1986, Nr. 9, 6–16.

[Jac92] D. Jackél, *Grafik Computer Grundlagen, Architekturen und Konzepte computergrafischer Sichtsysteme*, First Ed., Springer Verlag, 1992.

[JW63] D. B. Judd, G. Wyszecki, *Color in Business, Science and Industry*, Wiley & Sons, 1963.

[Kaj86] J. T. Kajiya, *The Rendering Equation*, Computer Graphics, Vol. 20, 1986, Nr. 4, 143–150.

[Kau82] A. Kaufman, *High-Level Color Naming Standards and Conversion Algorithms*, Tech. report, Center of Computer Graphics, Ben Gurion University of the Negev Beer-Sheva, 1982.

[Kni93] G. Knittel, *PROVEN - Prompt Vector Normalizer*, Proceedings of the 6. IEEE ASIC Conference, Rochester, 1993, 112–115.

[Kr"91] W. Krüger, *Visualisierung 3-dimensionaler skalarer Datenfelder: Ein "physikalisches" Modell*, Informationstechnik, Vol. 33, 1991, Nr. 2, 72–76.

[MRC+86] G. W. Meyer, H. E. Rushmeier, M. F. Cohen, D. P. Greenberg , K. E. Torrance, *An Experimental Evaluation of Computer Graphics Imagery*, ACM Transactions on Graphics, Vol. 5, 1986, Nr. 1, 30–50.

[NB94] X. Ni, M. S. Bloor, *Performance Evaluation of Boundary Data Structures*, IEEE Computer Graphics and Applications, 1994, Nr. 11, 66–77.

[Nus28] W. Nusselt, *Graphische Bestimmung des Winkelverhältnisses bei der Wärmestrahlung*, VDI Z., 1928, Nr. 72, 673.

[PH90] J. Pöpsel, Ch. Hornung, *Highlight Shading: Lighting and Shading in a PHIGS+/PEX Environment*, Computers & Graphics, Vol. 14, 1990, Nr. 1.

[PJ91] A. Pearce, D. Jevans, *Exploiting Shadow Coherence in Ray Tracing*, Proceedings Graphics Interface '91, Nr. 6, 1991, 109–116.

[Pra78] W. K. Pratt, *Digital Image Processing*, John Wiley & Sons, 1978.

[Pur86] W. Purgathofer, *A Statistical Method for Adaptive Stochastic Sampling*, Eurographics '86 (A. A. G. Requicha, Ed.), Nr. 8, 1986, 145–152.

[Sam90] H. Samet, *Design and Analysis of Spatial Data Structures*, Addison Wesley, 1990.

[Sch20] E. Schrödinger, *Grundlinien einer Theorie der Farbmetrik im Tagessehen*, Annalen der Physik, Vol. 62, 1920.

[Sie81] R. Siegel, *Thermal Radiation Heat Transfer*, Second Ed., Hemisphere Publishing Corporation, 1981.

[Sil92] F. X. Sillion, *Extension of Radiosity Methods for Non-Diffuse Enviroments*, ACM SIGGRAPH Course Notes Global Illumination, 1992.

[SK90] P. Slusallek, M. Krämer, *Progressive Refinement Versus Full Matrix*, Tech. report, WSI/GRIS Universität Tübingen, Juni 1990.

[Smi78] A. R. Smith, *Color Gamut Transform Pairs*, Computer Graphics, Vol. 12, 1978, Nr. 3.

[The73] R. Theile, *Fernsehtechnik*, Vol. 1, Springer Verlag, 1973.

[Wat70] G. S. Watkins, *A real-time visible surface algorithm*, Tech. Report UTEC-CS-70-101, Dept. Comput. Sci., Univ. Utah, Salt Lake City, UT, 1970.

[WCG87] J. R. Wallace, M. F. Cohen , D. P. Greenberg, *A Two–Pass Solution to the Rendering Equation: a synthesis of ray tracing and radiosity method*, ACM Computer Graphics, Vol. 21, 1987, Nr. 4, 311–320.

[WEH89] J. R. Wallace, K. Elmquist , E. Haines, *A Ray Tracing Algorithm for Progressive Radiosity*, ACM Computer Graphics, Vol. 23, 1989, Nr. 3, 315–324.

[Whi79] T. Whitted, *An Improved Illumination Model for Shaded Display*, Special SIGGRAPH '79 Issue, Vol. 13, 1979, Nr. 8, 1–14.

[Wys60] G. Wyszecky, *Farbsysteme*, Musterschmidt, Göttingen, 1960.

Index

Glossar

Modellbildung beim Solid Modelling, (7.1.1, S.449)

In einem zweistufigen Prozeß bildet man zunächst das *mathematische Modell*, dann das *symbolische Modell*.
Das mathematische Modell ist eine Idealisierung des realen Objektes und beinhaltet in der Regel nur einen Teil der Eigenschaften, die das reale Objekt auszeichnen.
Es wird in ein *symbolisches Modell* im sogenannten *Repräsentationsraum* überführt.
Der *Repräsentationsraum* ist abhängig vom *Modellierer* und umfaßt die Menge derjenigen Objekte, die mit dem Modellierer konstruiert werden können.

Randrepräsentationen (Boundary Representation) (7.3, S.454)

Ein Körpermodell kann eindeutig durch seine Oberfläche und eine zugehörige topologische Orientierung beschrieben werden. Wegen der topologischen Orientierung ist für jeden Punkt der Oberfläche eindeutig festgelegt, auf welcher Seite das Innere des Objektes liegt.
Boundary Representations (BReps) benutzen diese Tatsache und beschreiben 3D-Objekte durch ihre Oberfläche.

Datenstrukturen für BReps (7.3, S.455)

Für Boundary-Modelle können wir die folgenden Datenstrukturen verwenden:

- *Polygonorientierte Datenstrukturen:*
  Der Körper besteht aus einer Sammlung von Facetten, welche in einer Tabelle mit Bezeichnern für jede Facette zusammengefaßt werden.
- *Knotenorientierte Datenstrukturen:*
  Knoten werden unabhängig gespeichert und dann den einzelnen Facetten zugeordnet.
- *Kantenorientierte Datenstrukturen:*
  Ein kantenorientiertes Boundary-Modell stellt eine Facette als Zyklus von Kanten dar. Die Knoten einer Facette sind implizit durch die Kanten dargestellt.
  Das wichtigste Beispiel einer kantenorientierten Datenstruktur ist die *Winged-Edge-Datenstruktur:*
  Zusätzlich zu den in einer einfachen kantenorientierten Datenstruktur gespeicherten Beziehungen werden noch Nachbarschaftsbeziehungen zwischen den einzelnen Kanten explizit gespeichert.

Hybridmodell (7.6, S.465)

Keiner der drei wichtigsten Zugänge zu Solid Modelling (**Boundary-Darstellung, CSG-Darstellung** oder **Räumliche Zerlegungsmethoden**) ist für alle Anwendungen gleich gut geeignet. Daher werden bei der Implementierung oft mehrere Modelle gleichzeitig verwendet. Dabei muß auf die **Konsistenz** der Daten in den verschiedenen Repräsentationen geachtet werden.

Um von einer Repräsentation in eine andere zu wechseln, sind entsprechende Konvertierungsalgorithmen notwendig.
Die CSG-Darstellung ist z.B. für den Anwender am besten als Repräsentationsart geeignet, da Modellierungsvorschriften mittels Boole'scher Operationen direkt angegeben werden können. Zur schnellen Visualisierung sind hingegen BReps besser geeignet, da das Neuauswerten der gesamten CSG-Bäume bei einer Neudarstellung nach affinen Transformationen entfällt. Ein typisches Beispiel eines Hybridmodellierers ist daher ein CSG-BRep-Modellierer. Die CSG-Datenstruktur wird dabei als Modelldatenstruktur verwendet.

Beleuchtungsmodell (7.7, S.467)

Ein *Beleuchtungsmodell* ist eine Vorschrift zur Berechnung der Farb- und Grauwerte der einzelnen Bildpunkte eines Bildes.
In einem solchen Modell werden die Einflüsse der Lichtquellen (Lage, Größe, Stärke, spektrale Zusammensetzung) sowie der Oberflächenbeschaffenheit (Geometrie, Reflexionseigenschaften) auf die Farbe eines Bildpunktes erfaßt.
Das Modell muß also den Vorgang der Lichtausbreitung in einer definierten Umgebung beschreiben und sich auch (mit vernünftigem Aufwand) berechnen lassen.

strahlungsphysikalische Grundgrößen (7.8.1, S.468)

Im Kurs werden die folgenden *strahlungsphysikalischen Grundgrößen* behandelt:

- **Raumwinkel**
- **Strahlungsenergie**
- **Strahlungsleistung**
- **Strahlstärke (Intensität)**
- **spezifische Ausstrahlung (Radiosity)**
- **Bestrahlungsstärke**
- **Strahldichte (Radiance)**

Radiometrie (7.8.1, S.468)

Die *Radiometrie* beschäftigt sich mit dem ganzen elektromagnetischen Spektrum. Alle strahlungsphysikalischen Größen bekommen zur Unterscheidung von den entsprechenden **photometrischen Größen** den Index e (für energetisch).

Raumwinkel (7.8.1, S.468)

Den *Raumwinkel* Ω, unter dem eine Fläche A von einem Punkt O aus erscheint, ist gegeben durch:

$$\Omega = \int_A \frac{\cos\alpha}{R^2}\, dA.$$

Er hat die Einheit *sr* (steradiant).
Der volle Raumwinkel ist per Definition die Fläche einer Einheitskugel, hat

also die Größe $4\pi sr$.

Strahlungsenergie (7.8.1, S.469)

Die wichtigste Größe in der Radiometrie ist die *Strahlungsenergie* (Radiant Energy) Q_e. Von ihr werden alle anderen Größen abgeleitet.

Strahlungsleistung (7.8.1, S.469)

Die *Strahlungsleistung* (Radiant Flux) φ_e ist die zeitliche Ableitung der Strahlungsenergie. Sendet ein Körper im Zeitintervall dt die Strahlungsenergie dQ_e aus, so ist seine Strahlungsleistung durch

$$\varphi_e = \frac{dQ_e}{dt}$$

gegeben. Die Strahlungsleistung wird auch *Strahlungsfluß* genannt und hat die Einheit Watt.
Um aufgenommene Strahlungsleistung von abgegebener unterscheiden zu können, bekommt φ_e noch einen Index 1, wenn es sich um eine Strahlungsquelle handelt und eine 2, falls es ein Strahlungsempfänger ist.

Strahlstärke (7.8.1, S.470)

Strahlt die Punktlichtquelle in alle Richtungen gleich ab, so muß die auf das differentielle Raumwinkelelement $d\Omega_1$ entfallende Strahlungsleistung $d\varphi_{e_1}$ diesem proportional sein. Es gilt also:

$$d\varphi_{e_1} = I_{e_1} d\Omega_1.$$

Die Proportionalitätsgröße I_e heißt *Strahlstärke* oder auch *Intensität* (Intensity). Die Einheit der Strahlstärke ist $W sr^{-1}$. Da natürliche Lichtquellen nicht in alle Richtungen gleich stark abstrahlen, ist die Strahlstärke im allgemeinen eine Funktion der Abstrahlrichtung.

Bestrahlungsstärke (7.8.1, S.470)

Die Größe

$$E_e = I_e \frac{\cos \alpha_2}{R^2}$$

heißt *Bestrahlungsstärke*. Sie ist ein Maß für die pro Fläche auftreffende Strahlungsleistung und nimmt (im Fall punktförmiger Strahler) mit dem Quadrat der Entfernung von der Lichtquelle ab. Dieser Zusammenhang wird als *quadratisches Entfernungsgesetz* bezeichnet.
Die Einheit der Bestrahlungsstärke ist $W m^{-2}$.

spezifische Ausstrahlung (Radiosity) (7.13.1, S.513)

Die der Bestrahlungsstärke entsprechende Sendegröße heißt *spezifische Ausstrahlung* (Radiosity) M_e:

$$d\varphi_{e1} = M_e dA_1.$$

Die Einheit der spezifischen Ausstrahlung ist $W m^{-2}$.

Strahldichte (Radiance) (7.8.1, S.471)

Bei flächenförmigen Lichtquellen ist die Strahlstärke nicht nur richtungs-, sondern auch ortsabhängig.
Die auf ein Flächenelement dA_1 bezogene Strahlungsstärke in Richtung ϑ_1 ist zum einen zur Fläche dieses Flächenelements und zum anderen zum Kosinus dieses Winkels proportional. Es gilt

$$dI_{e_1} = L_{e_1} \cos\vartheta_1 dA_1.$$

Die Proportionalitätsgröße L_{e_1} heißt *Strahldichte* (Radiance). Sie hat die Einheit $W\,sr^{-1}\,m^{-2}$.

Grundgesetz der Strahlungsübertragung (7.8.1, S.471)

Das *Grundgesetz der Strahlungsübertragung* beschreibt die differentielle Strahlungsleistung, die ein differentielles Flächenelement dA_1 abstrahlt und die von einem differentiellen Flächenelement dA_2 im Abstand R von dA_1 aufgenommen wird:

$$d^2\varphi_e = L_e \frac{\cos\vartheta_1 \cos\vartheta_2}{R^2} dA_1 dA_2.$$

spektrale Dichte (7.8.1, S.472)

Um die Abhängigkeit von der Frequenz zu erfassen, wird jeder strahlungsphysikalischen Größe X_e eine *spektrale Dichte* $X_{e\lambda}$ zugeordnet:

$$X_{e\lambda} = \frac{dX_e}{d\lambda}.$$

Oft ist es zweckmäßig, mit einer *normierten spektralen Dichtefunktion*

$$\frac{X_{e\lambda}(\lambda)}{X_{e\lambda}(\lambda_0)}$$

zu arbeiten, wobei λ_0 eine zu wählende Bezugsfrequenz ist.

Photometrie (7.8.2, S.472)

Im Gegensatz zur Radiometrie beschäftigt sich die Photometrie mit der Messung von elektromagnetischer Strahlung im sichtbaren Bereich von ca. 360 nm bis 830 nm. Der entscheidende Unterschied zur Radiometrie besteht darin, daß nicht nur physikalische Größen, sondern auch die physiologischen Eigenschaften des menschlichen Auges berücksichtigt werden.
Jeder strahlungstechnischen Größe wird eine photometrische Größe zugeordnet, wobei zur Unterscheidung der Index v (für visuell) verwendet wird. Außerdem erhalten die photometrischen Größen neue Einheiten.

Hellempfindlichkeitsfunktion (7.8.2, S.472)

Der Zusammenhang zwischen radiometrischen und photometrischen Größen wird durch die *Hellempfindlichkeitsfunktion* $V(\lambda)$ hergestellt. Die Hellempfindlichkeitsfunktion gibt die relative Empfindlichkeit des Auges in Abhängigkeit von der betrachteten Wellenlänge an. Ist das Auge dunkeladaptiert (*skotopisches Sehen*), so sind nur die sogenannten Stäbchen des Auges

für die Rezeption verantwortlich, während beim helladaptierten Auge (*photopisches Sehen*) nur die Farbzapfen beteiligt sind. Deshalb gibt es zwei voneinander abweichende Hellempfindlichkeitsfunktionen, V für photopisches und V' für skotopisches Sehen.

photometrische Grundgrößen (7.8.2, S.472)

Durch Gewichtung der strahlungstechnischen Größen $X_{e\lambda}$ mit den Hellempfindlichkeitsfunktionen erhält man die entsprechenden photometrischen Größen $X_{v\lambda}$ und $X'_{v\lambda}$:

$$X_{v\lambda}(\lambda) = Km X_{e\lambda}(\lambda) V(\lambda)$$

und

$$X'_{v\lambda}(\lambda) = Km' X_{e\lambda}(\lambda) V'(\lambda).$$

Km bzw. Km' sind Proportionalitätsfaktoren.

Im Kurs werden die folgenden photometrischen Grundgrößen behandelt:

- Lichtmenge
- Lichtstrom
- **Lichtstärke**
- Spezifische Lichtausstrahlung
- **Beleuchtungsstärke**
- **Leuchtdichte**

Lichtstärke (7.8.2, S.472)

Die *Lichtstärke* I_v ist das photometrische Gegenstück zur **Strahlstärke** I_e. Die Bedeutung der Lichtstärke für die Photometrie wird dadurch deutlich, daß sie eine eigene SI-Basiseinheit bekommt, nämlich die **Candela** (cd).

Candela (1, S.473)

Eine *Candela* ist die Lichtstärke in einer bestimmten Richtung einer Strahlungsquelle, die monochromatische Strahlung der Frequenz $540\,THz$ (entspricht der Wellenlänge von ca. $555\,nm$) aussendet und deren Strahlstärke in dieser Richtung $\frac{1}{683} W\,sr^{-1}$ beträgt.

Beleuchtungsstärke (1, S.474)

Das photometrische Gegenstück zur **Bestrahlungsstärke** ist die *Beleuchtungsstärke*. Die Einheit der Beleuchtungsstärke ist das *Lux*, abgekürzt *lx*.

Leuchtdichte (7.8.3.1, S.474)

Die *Leuchtdichte* ist das photometrische Gegenstück zur radiometrischen Größe **Strahldichte.**

Emission (7.8.3.1, S.474)

Die *Emission* wird mit der radiometrischen Größe **Strahldichte** bzw. ihrem photometrischen Gegenstück, der **Leuchtdichte**, beschrieben. Im allgemeinen Fall ist die Strahldichte L von der Emissionsrichtung und von der Wellenlänge λ abhängig. Die beiden Polarwinkel φ und ϑ legen die Abstrahlungsrichtung fest. Die gesamte Emission erhält man durch Integration über den sichtbaren Wellenlängenbereich

$$L'(\vartheta,\varphi) = \int_{\lambda=360nm}^{830nm} L'_\lambda(\lambda,\varphi,\vartheta)\,d\lambda.$$

Lambert'scher Strahler (7.8.3.1, S.475)

Lambert'sche Strahler sind Oberflächen, bei denen die **Leuchtdichte** unabhängig von der Betrachtungsrichtung ist. Die Fläche erscheint daher aus jeder Richtung gleich hell.

Die **Strahlstärke** eines Lambert'schen Strahlers ist proportional zum Cosinus des Winkels zwischen Flächennormaler und Ausstrahlungsrichtung (*Lambert'sches Cosinusgesetz*).

Reflexion (7.8.3.2, S.475)

Die *Reflexion* von Strahlung wird durch den *spektralen Reflexionsfaktor* ρ beschrieben, der das Verhältnis von reflektierter **Strahldichte** zur einfallenden **Bestrahlungsstärke** E angibt:

$$\rho_\lambda(\lambda,\varphi_r,\vartheta_r,\varphi_i,\vartheta_i) = \frac{L_{\lambda,r}(\lambda,\varphi_r,\vartheta_r)}{E_{\lambda,i}(\lambda,\varphi_i,\vartheta_i)} = \frac{L_{\lambda,r}(\lambda,\varphi_r,\vartheta_r)}{\int L_{\lambda,i}(\lambda,\varphi_i,\vartheta_i)\,cos\vartheta_i\,d\Omega_i}.$$

Der Index i kennzeichnet die Größen der einfallenden, r die Größen der reflektierten Strahlung.

Im allgemeinen werden drei Arten der Reflexion unterschieden:

- **ideal diffuse Reflexion**
- **ideal spiegelnde Reflexion**
- **gerichtet diffuse Reflexion** (spekulare Reflexion)

Reflexionsgrad (4, S.476)

Der *Reflexionsgrad* r gibt das Verhältnis von reflektierter zu einfallender **Bestrahlungsstärke** an und ist deshalb dimensionslos:

$$r_\lambda = \frac{E_{\lambda,r}}{E_{\lambda,i}},\quad 0 \le r_\lambda \le 1.$$

Er wird in der GDV, besonders bei den empirischen Beleuchtungsmodellen, statt des Reflexionsfaktors ρ verwendet (s. Stichwort **Reflexion**).

ideal diffuse Reflexion (7.8.3.3, S.476)

Im einfachsten Fall ist die reflektierte **Leuchtdichte** unabhängig von der Abstrahlungsrichtung. Man nennt diesen Fall deshalb auch *ideal diffuse*

oder *Lambert'sche Reflexion*. Auf dieser Art der Reflexion wird im **Radiosity-Verfahren** der Austauschmechanismus für Licht zwischen den Oberflächen der Objekte modelliert. Die reflektierte Leuchtdichte ist zwar unabhängig von den Winkeln (φ_r, ϑ_r), hängt aber von den Einstrahlungswinkeln (φ_i, ϑ_i) ab:

$$L_{\lambda,r} = \rho_\lambda E_{\lambda,i}(\lambda, \varphi_i, \vartheta_i).$$

Da die Leuchtdichte unabhängig von der Abstrahlrichtung ist, ergibt sich für die spezifische Ausstrahlung (Radiosity)

$$M = \pi L_{\lambda,r}$$

und

$$\rho_\lambda = \frac{M}{\pi E_\lambda} \left[\frac{1}{sr}\right].$$

ideal spiegelnde Reflexion (7.8.3.4, S.477)

Die spiegelnde Reflexion wird durch das aus der Optik bekannte Reflexionsgesetz beschrieben:
Der einfallende und der reflektierte Strahl bilden mit der Normalen der reflektierenden Oberfläche gleiche Winkel. Einfallender Strahl, reflektierter Strahl und Flächennormale liegen in einer Ebene.
Dies ist die strahlenoptische Formulierung des Reflexionsgesetzes. Für elektromagnetische Wellen gilt das Reflexionsgesetz in gleicher Weise.
Ist also die Richtung der einfallenden Strahlung (φ_i, ϑ_i), so wird nur in die Richtung

$$\vartheta_r = \vartheta_i \qquad \text{und} \qquad \varphi_r = \varphi_i + \pi$$

Strahlung reflektiert.
Konventionelles **Raytracing** beruht auf diesem einfachen Reflexionsgesetz.

Brechungsgesetz von Snellius (7.8.3.4, S.478)

Einfallender Strahl, Normale und gebrochener Strahl liegen in einer Ebene. Der Sinus des Einfallwinkels steht zum Sinus des Brechungswinkels in einem konstanten Verhältnis, das nur von der Natur der beiden Medien abhängt:

$$n_1 \sin\vartheta_1 = n_2 \sin\vartheta_2 \Leftrightarrow \frac{\sin\vartheta_1}{\sin\vartheta_2} = \frac{n_2}{n_1} = const.$$

n_1 bzw. n_2 sind dabei die **Brechzahlen** (Brechungsindizes) der Medien.

Brechzahl (7.8.3.4, S.478)

Die *Brechzahl* ist definiert als das Verhältnis der Lichtgeschwindigkeit im Vakuum zur Lichtgeschwindigkeit im betreffenden Medium. Der Brechungsindex des Vakuums ist gleich 1.

gerichtet diffuse Reflexion (spekulare Reflexion) (7.8.3.5, S.479)

Da man die beiden idealen Reflexionsarten in der Natur selten antrifft, muß man für alle Oberflächen die richtungsmäßige Verteilung der Größe $\rho_\lambda(\lambda, \varphi_r, \vartheta_r, \varphi_i, \vartheta_i)$ bestimmen. Sehr häufig tritt der Fall auf, daß ρ ein deutliches Maximum in Richtung der spiegelnden Reflexion hat und kleiner wird, je weiter man sich von dieser Richtung entfernt *(spekularer Reflexion)*.

In der GDV ist es üblich, die spekulare Reflexion in einen richtungsunabhängigen, diffusen Anteil (Index d) und einen richtungsabhängigen Anteil (Index s) aufzuspalten.

Im Kurs werden drei Modelle behandelt:

- **Phong-Modell**
- **Modell von Torrance und Sparrow**
- Modell von Cook und Torrance

Phong-Modell (7.8.3.5, S.479)

Im empirischen *Phong-Modell* wird der richtungsabhängige Anteil der spekularen Reflexion über den Reflexionsgrad berechnet als

$$r_s = r_{s,0} \cos^m \gamma.$$

$r_{s,0}$ ist eine Konstante zwischen 0 und 1. γ ist der Winkel zwischen der Richtung des ideal reflektierten Strahls und der Beobachtungsrichtung. Der Exponent m gibt an, wie schnell das Reflexionsvermögen mit größer werdendem Winkel γ abfällt (vgl. **gerichtet diffuse Reflexion (spekulare Reflexion)**.

Farbmetrik (7.9, S.479)

Farbmetrik ist die Lehre von den Maßbeziehungen der Farben untereinander. In der GDV wird sie aus folgendem Grund benötigt:

Die **Leuchtdichte** wird an jeder Stelle der Szene bestimmt und ein Rasterbild berechnet, das ein wirklichkeitsgetreues Abbild der Szene ist. Da es keine Ausgabegeräte gibt, die direkt mit einer spektralen Leuchtdichteverteilung für jeden Bildpunkt ansprechbar sind, wird üblicherweise ein Bildpunkt eines Ausgabegerätes durch seine Farbe charakterisiert.

Farbgebung (7.9.1, S.480)

Die *Farbgebung* auf einem Graphikmonitor geschieht auf der Grundlage der additiven Farbmischung. Jeder Bildpunkt des Monitors besteht aus drei dicht beieinanderliegenden Bildpunkten. Aufgrund des beschränkten räumlichen Auflösungsvermögens des Auges werden die drei Farbkomponenten im Auge zu einem einheitlichen Farbreiz gemischt.

Bei üblichen Monitoren setzt sich die Farbe aus einer roten, einer grünen und einer blauen Komponente zusammen (RGB-Monitor).

Für jede Komponente können unterschiedliche Intensitätsstufen gewählt werden. Üblich sind heute 256 Intensitätsstufen.

Es genügen drei Grundfarben, um sämtliche Farbtöne darzustellen (vgl. **Graßmannsches Gesetz**).

Darstellung von Farben (7.9.1, S.480)

Im Kurs werden zwei Möglichkeiten zur Farbdarstellung vorgestellt:

- Farben können als Vektoren eines dreidimensionalen Vektorraumes aufgefaßt werden. Die Vektoren dieses 'Farbraums' heißen *Farbvalenzen*. Die Länge eines Vektors ist ein Maß für die **Leuchtdichte** und heißt *Farbwert*, seine Richtung bestimmt die *Farbart*.
  Mit drei linear unabhängigen Basisvektoren *(Primärvalenzen)* $\mathcal{R}$, $\mathcal{G}$ und $\mathcal{B}$ läßt sich für jede Farbvalenz $\mathcal{F}$ eine Farbgleichung aufstellen:

  $$\mathcal{F} = r\mathcal{R} + g\mathcal{G} + b\mathcal{B}.$$

  Die Werte für r, g und b können nur durch ein Mischexperiment gewonnen werden.
- Für die bildliche Darstellung einer Farbtafel benutzt man häufig eine zweidimensionale Darstellung mit Hilfe von **baryzentrischen Koordinaten** (vgl. GDV I):
  Ist eine Farbvalenz F gegeben durch

  $$\mathcal{F} = R_F\,\mathcal{R} + G_F\,\mathcal{G} + B_F\,\mathcal{B},$$

  so sind die baryzentrischen Koordinaten

  $$\begin{aligned} r_F &= \frac{R_F}{R_F + G_F + B_F} \\ g_F &= \frac{G_F}{R_F + G_F + B_F} \\ b_F &= \frac{B_F}{R_F + G_F + B_F}. \end{aligned}$$

Graßmannsches Gesetz (7.9.1, S.480)

erstes Graßmannsches Gesetz:
Zwischen je vier Farben besteht immer eine eindeutige lineare Beziehung. Eine Farbe braucht zu ihrer Beschreibung drei voneinander unabhängige Bestimmungsstücke, d.h. die Farbe ist eine dreidimensionale Größe.
zweites Graßmannsches Gesetz:
Gleich aussehende Farben ergeben mit einer dritten Farbe stets gleich aussehende Farbmischungen.

Spektralwert (7.9.2, S.483)

Bestimmung der Farbgleichung bei vorgegebenen Primärvalenzen für einen beliebigen Spektralreiz:
Man zerlegt den sichtbaren Wellenlängenbereich in enge Spektralbänder $\lambda_{k,\Delta\lambda} \ldots \lambda_{l,\Delta\lambda}$ der Bandbreite $\Delta\lambda$ und betrachtet zunächst das Farbempfinden, das ein auf das Spektralband $\lambda_{i,\Delta\lambda}$ beschränkter 'monochromatischer' Spektralreiz erzeugt. Für die zu diesem Farbreiz gehörende Farbvalenz $\mathbf{f}_{\lambda_i}$ kann man eine Farbgleichung aufstellen:

$$\mathbf{f}_{\lambda_i} = \mathbf{r}_{\lambda_i}\,\mathcal{R} + \mathbf{g}_{\lambda_i}\,\mathcal{G} + \mathbf{b}_{\lambda_i}\,\mathcal{B}$$

Die Farbkoeffizienten $\mathbf{r}_{\lambda_i}, \mathbf{g}_{\lambda_i}$ und $\mathbf{b}_{\lambda_i}$ heißen *Spektralwerte* bzgl. der Primärvalenzen $\mathcal{R}$, $\mathcal{G}$, $\mathcal{B}$. Die Spektralwerte können nur mittels Mischexperimenten gewonnen werden. Führt man dieses Experiment für jedes Spektralband innerhalb des sichtbaren Spektralbereichs durch, erhält man die sogenannten *Spektralwertkurven* $\overline{r}$, $\overline{g}$ und $\overline{b}$. Hat man die Spektralwertkurven für ein Primärvalenztripel bestimmt, so kann man daraus die Spektralwertkurven für jedes beliebige Primärvalenztripel berechnen.
Mit Hilfe der Spektralwertkurven für ein gegebenes Primärvalenztripel kann man den Farbreiz, den ein ganzes Wellenlängenspektrum L_λ erzeugt, ermitteln.

CIE-Primärvalenzen (7.9.2, S.485)

Um nicht für jedes Primärvalenztripel von neuem die Spektralwertkurven (vgl. Stichwort **Spektralwert**) bestimmen zu müssen, wurde schon 1931 von der *Commision International de l'Éclairage* (CIE) ein Standardsystem von Primärvalenzen vorgeschlagen. Das besondere an diesen CIE-Primärvalenzen ist, daß sie außerhalb der Spektralvalenzkurve liegen. Sie sind also gar nicht darstellbar; man nennt solche Farbvalenzen deshalb *virtuell*.

Transformationen zwischen Primärvalenzsystemen (7.9.2, S.486)

Zunächst drückt man die Primärvalenzen von S' in Koordinaten von S aus:

$$\begin{aligned} P &= p_r \cdot R + p_g \cdot G + p_b \cdot B \\ D &= d_r \cdot R + d_g \cdot G + d_b \cdot B \\ T &= t_r \cdot R + t_g \cdot G + t_b \cdot B \end{aligned}$$

Aus den Koeffizienten erhält man die invertierbare Transformationsmatrix M:

$$S' = MS = \begin{pmatrix} p_r & p_g & p_b \\ d_r & d_g & d_b \\ t_r & t_g & t_b \end{pmatrix} S.$$

Farbmodell (7.10, S.487)

Die unterschiedlichen Farbmodelle lassen sich wie folgt unterteilen:

- *Technisch-physikalische Farbmodelle:*
  - **RGB-Modell**
  - **CMG-Modell**
  - **YIQ-Modell**
  - **YUV-Modell**

  Sie beschreiben eine Farbe als Mischung dreier Primärfarben. Die Unterschiede zwischen den einzelnen Modellen liegen in der Wahl der Primärfarben und der Art der Farbmischung.
  Diese Modelle sind an den Erfordernissen der Ausgabegeräte und Übertragungstechnik orientiert. Sie eignen sich wenig zur direkten Farbdefinition durch den Benutzer.

- *Wahrnehmungsorientierte Farbmodelle:*
  Sie arbeiten mit den Größen Helligkeit, Farbton und Farbsättigung.
  Je nach Art der Beschreibung dieser Größen lassen sich zwei Gruppen von wahrnehmungsorientierten Farbmodellen unterscheiden:

  a) Die Farbe wird in numerisch-symbolischer Form beschrieben:
     - **HLS-System**
     - **H'L'S'-System**
     - HSV-System, H'S'V'-System
     - Ridgway-, Ostwald-, Munsell-, DIN- und Hesselgren- Systeme

  b) Die Farbe wird umgangssprachlich beschrieben:
     - CNS = Color-Naming-System
     - ISCC-NBS-Lexikon
     - ACNS

  Diese Größen müssen zur Ausgabe in ein technisch-physikalisches Modell transformiert werden.
  Im Kurs behandelt wurden die Transformationsalgorithmen
  - RGB nach HLS-/H'L'S'
  - HLS /H'L'S' nach RGB.

RGB–Modell (7.10.1.1, S.487)

Beim *RGB–Modell* werden die darstellbaren Farben als Punkte eines Einheitswürfels im Ursprung des kartesischen Koordinatensystems beschrieben. Auf den positiven Halbachsen liegen die Primärfarben Rot, Grün und Blau. Grauwerte, darstellbar durch gleichgroße Anteile von R, G und B, liegen auf der Hauptdiagonalen des Einheitswürfels mit Schwarz im Ursprung $[0,0,0]$ und Weiß im Punkt $[1,1,1]$. Eine Farbe wird dann durch Anteile von R, G und B beschrieben, die zu Schwarz addiert werden müssen.
Farbrastersichtgeräte haben R, G, B–Leuchtstoffe, die über individuelle Kathoden angeregt werden, woraus sich die besondere Bedeutung des RGB-Modells ergibt: Alle anderen Farbbeschreibungen müssen vor der Farbausgabe in den äquivalenten Punkt des RGB–Würfels umgerechnet werden.

CMG–Modell (7.10.1.1, S.487)

Zur Bezeichnung von Farben für die Plotterausgabe ist ein zum RGB–Würfel komplementäres Modell zweckmäßig, bei dem Weiß im Koordinatenursprung liegt. Auf den Koordinatenachsen werden dann die Primärfarben Cyan, Magenta und Gelb abgetragen, Schwarz liegt bei $[1,1,1]$.
Die Umrechnung zwischen beiden Modellen ist einfach:
Von RGB nach CMGe:

$$[C,M,Ge] = [1,1,1] - [R,G,B],$$

und von CMGe nach RGB:

$$[R,G,B] = [1,1,1] - [C,M,Ge].$$

YIQ–Modell (7.10.1.2, S.488)

Farben können auch durch Leuchtdichte (Luminanz, Helligkeit) und Farbart (Chrominanz) definiert werden. Ihre Kenngrößen sind der *Farbton* und die *Farbsättigung*. Der Farbton ist mit der dominierenden Wellenlänge des Lichtes gegeben. Die Sättigung ist ein Maß für die spektrale Reinheit.
Aus den RGB-Werten wird die Luminanz als bewertete Summe gebildet:

$$Y = 0,3R + 0,59G + 0,11B.$$

Die Chrominanz wird mit den zwei Differenzen $R-Y$ und $B-Y$ angegeben. Die Differenzen werden bewertet zu den Größen I und Q zusammengefaßt:

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0,30 & 0,59 & 0,11 \\ 0,60 & -0,28 & -0,32 \\ 0,21 & -0,52 & 0,31 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}.$$

YUV–Modell (7.10.1.3, S.490)

Die europäischen Systeme PAL (Deutschland) und SECAM (Frankreich) verwenden wie das **YIQ-Modell** die Helligkeit Y und zwei Farbdifferenzen U und V. Die U- und V-Signale lassen sich durch eine einfache Rotation der Koordinaten im Farbraum in die I- und Q-Komponenten des YIQ-Modells überführen:

$$\begin{aligned} I &= -U\sin(33^\circ) + V\cos(33^\circ) \\ Q &= U\cos(33^\circ) + V\sin(33^\circ). \end{aligned}$$

HLS-System (7.10.2.1, S.490)

HLS-System = Hue (Farbton) - Lightness (Helligkeit) - Saturation (Sättigung) – System:
Die Farbanordnung entspricht der senkrechten Projektion des RGB–Würfels von Weiß nach Schwarz entlang der Hauptdiagonalen.
Das entstehende regelmäßige Sechseck wird meist durch einen Kreis ersetzt, so daß der *Farbton* (H) als Winkel zwischen 0° und 360° anzugeben ist.
Die *Helligkeit* (L) wird als Wert zwischen 0 und 1 angegeben, wobei 0 Schwarz und 1 Weiß entspricht.
Die *Sättigung* (S) wird als Abstand einer Farbe vom Mittelpunkt des Farbkreises angegeben. Sie beträgt 0 für achromatische Farben und kann als höchsten Wert 1 für die gesättigten Farben auf dem Rand des Farbkreises annehmen.

H'L'S'-System (7.10.2.1, S.491)

Das H'L'S'-System entsteht durch Verschieben von Grün in Richtung Blau.

Dadurch liegen Rot, Gelb und Blau gleich weit voneinander entfernt, was der Farbempfindung besser entspricht (vgl. Stichwort **HLS-System**).

Beleuchtungsverfahren (7.11, S.494)

Beleuchtungsmodelle beschreiben die Zusammenhänge zwischen Raumgeometrie, Lichtquellen und Oberflächenbeschaffenheit zur Berechnung der Leuchtdichte in einem Punkt der darzustellenden Szene.
Beleuchtungsalgorithmen berechnen aus der **Leuchtdichte** einiger ausgewählter Punkte die darzustellende Farbe aller Bildpunkte.
Die Kombination von Beleuchtungsmodell und -algorithmus kennzeichnet spezielle *Beleuchtungsverfahren*.

Lichtquelle (7.11.1, S.494)

Die wichtigsten Lichtquellen sind:

- **Umgebungslicht (ambientes Licht)**
- **diffuse Flächenlichtquelle**
- **Punktlichtquelle**
- **Richtungslichtquelle**
- **Goniometrische Lichtquelle**
- **Strahler**

Umgebungslicht (ambientes Licht) (7.11.1.1, S.495)

Eine ambiente Komponente dient in Beleuchtungsverfahren dazu, alle vom Modell nicht erfaßten, jedoch in Wirklichkeit vorhandenen, indirekten Beleuchtungen zu berücksichtigen. Damit werden auch nicht direkt beleuchtete Szenenteile sichtbar. Das ambiente Licht fällt auf allen Flächen der Szene mit gleicher Stärke ein und wird deshalb durch die Beleuchtungsstärke $E_a(\lambda)$ in Lux (lx) charakterisiert.
Die ambiente **Leuchtdichte** ergibt sich zu

$$L_{amb}(\lambda) = \rho_a(\lambda)\, E_a(\lambda),$$

wobei ρ_a der ambiente Reflexionsfaktor ist.

diffuse Flächenlichtquelle (7.11.1.2, S.495)

Diese Lichtquelle spielt für **globale Beleuchtungsverfahren**, bei denen alle beleuchteten und reflektierenden Flächen als Sender berücksichtigt werden, eine große Rolle (siehe **Radiosity-Verfahren**). Sie wird durch die Fläche A, die Normale N und die spezifische Lichtausstrahlung $M(\lambda)$ in Lumen/Quadratmeter (lm/m^2) angegeben. Da die Fläche definitionsgemäß diffus strahlt, ist ihre **Leuchtdichte** durch

$$L = \frac{M}{\pi}$$

und die **Lichtstärke** durch

$$dI = \frac{M}{\pi} \cdot dA \cdot cos\vartheta,$$

gegeben, wobei ϑ der Winkel zwischen N und der Richtung zum betrachteten Punkt ist.

Punktlichtquelle (7.11.1.3, S.495)

Die *Punktlichtquelle* wird meist als in alle Richtungen gleichmäßig sendend (isotrop) angenommen. Neben der Lage im Raum wird sie durch die Lichtstärke $I(\lambda)$ in Candela gekennzeichnet. Im Abstand R erzielt diese Lichtquelle eine **Beleuchtungsstärke**

$$E = \frac{I}{R^2} \cdot cos\vartheta,$$

wobei ϑ der Lichteinfallswinkel (zwischen Flächennormaler und Lichtausbreitungsrichtung) ist. Ist die Punktlichtquelle weit entfernt, so verhält sie sich wie eine Richtungslichtquelle.

Richtungslichtquelle (7.11.1.4, S.496)

Die *Richtungslichtquelle* ist durch die Lichtausbreitungsrichtung $\vec{V}_L$ und die spezifische Lichtausstrahlung $M(\lambda)$ gegeben. Alle Lichtstrahlen treffen mit derselben Richtung $\vec{V}_L$ in der Szene auf, wodurch eine weit entfernte Lichtquelle, wie z.B. die Sonne, modelliert werden kann.

goniometrische Lichtquelle (7.11.1.5, S.496)

Bei dieser Lichtquelle wird die Ausbreitungscharakteristik in einem Diagramm beschrieben, das die Größe der Lichtstärke als Funktion des Winkels zur Hauptausbreitungsrichtung in einer Tabelle angibt. Zur Ermittlung der Lichtstärke in eine gewünschte Richtung muß unter Umständen zwischen Tabellenwerten interpoliert werden.

Strahler (7.11.1.6, S.496)

Beim *Strahler* wird die Lichtausbreitung der Quelle auf einen bestimmten Raumwinkel (Lichtkegel) beschränkt. Der Abfall der Lichtstärke vom größten Wert in Richtung der Symmetrieachse zum Rand wird durch folgendes Gesetz bestimmt:

$$I = I_0 \cdot cos^n \vartheta.$$

Der Exponent n bestimmt dabei die Bündelung des Lichts.

lokales (empirisches) Beleuchtungsmodell (7.11.2, S.496)

Lokale Beleuchtungsmodelle beschreiben die in einem Punkt der Objektoberfläche wahrnehmbare **Leuchtdichte**. Dabei wird das Zusammenwirken von einfallendem Licht aus den Lichtquellen und dem Reflexionsverhalten der Objektflächen ausgewertet.
Sekundäre Effekte, wie der Strahlungsaustausch benachbarter Flächen, werden nicht berücksichtigt.
Lokale Beleuchtungsmodelle sollen mit minimalem Aufwand möglichst

realitätsnahe Flächengraphik erzeugen. Sie basieren deshalb auch nicht auf strengen physikalischen Gesetzmäßigkeiten.
Neben dem Umgebungslicht sind noch drei Reflexionsarten zu berücksichtigen: **ideal diffus, ideal spiegelnd und gerichtet diffus.**
Im Falle durchsichtiger Objekte kommt unter Umständen noch ein transparenter Anteil hinzu.

Bestimmung der Leuchtdichte für

- **ambientes Licht:**

$$L_{amb} = r_a \cdot \frac{E_a}{\pi} = r_a \cdot L.$$

- **ideal diffus reflektiertes Licht:**
  Die Leuchtdichte hängt vom Einfallswinkel ϑ ab, ist aber unabhängig vom Betrachtungswinkel:

$$\begin{aligned} L_{diff} &= \begin{cases} r_d \cdot L \cdot cos\vartheta, & |\vartheta| < 90° \\ 0 \text{ sonst} \end{cases} \\ &= \begin{cases} r_d \cdot L \cdot (\vec{N} \cdot \vec{V}_L), \text{ falls } (\vec{N} \cdot \vec{V}_L) > 0 \\ 0 \text{ sonst} \end{cases} \end{aligned}$$

- **Gerichtet diffus reflektiertes Licht (Phong-Modell):**
  Phong-Modell:

$$\begin{aligned} L_{spec} &= \begin{cases} r_s \cdot L \cdot \cos^m \gamma, \text{ falls } |\gamma| < 90° \\ 0 \text{ sonst} \end{cases} \\ &= \begin{cases} r_s \cdot L \cdot (\vec{R} \cdot \vec{E})^m, \text{ falls } (\vec{R} \cdot \vec{E}) > 0 \\ 0 \text{ sonst} \end{cases} \end{aligned}$$

  Phong-Modell mit Vermeidung der Berechnung von $\vec{R}$:

$$L'_{spec} = \begin{cases} r_s \cdot L \cdot (\vec{H} \cdot \vec{N})^m, \text{ falls } (\vec{H} \cdot \vec{N}) > 0 \\ 0 \text{ sonst} \end{cases} .$$

Beleuchtungsmodell nach Phong (7.11.2.3, S.499)
Nach dem Phong'schen Modell erhält man dann bei n Lichtquellen die Leuchtdichte in einem Punkt aus (vgl. **lokales (empirisches) Beleuchtungsmodell**):

$$L_{Phong} = L_{amb} + \sum_{i=1}^{n} (L_{diff,i} + L_{spec,i}).$$

lokale Beleuchtungsalgorithmen (7.11.3, S.499)
Im Kurs behandelt wurden die folgenden lokalen Beleuchtungsalgorithmen:

- **Konstante Beleuchtung**
- **Gouraud–Algorithmus**
- **Phong–Algorithmus**

Dabei wird davon ausgegangen, daß Objekte in triangulierter Form vorliegen.

konstante Beleuchtung (flat shading) (7.11.3.1, S.500)

Der einfachste Algorithmus zur Beleuchtung polygonal begrenzter Körper belegt ein ganzes Polygon gleichmäßig mit einer Farbe, die aber von der Beleuchtung der Szene abhängt *(flat shading)*. Das bedeutet, daß nur einmal pro Polygon eine Leuchtdichte zu berechnen ist und die Farbe als konstantes Attribut dieser Fläche mitgeführt werden kann.

Aufwand:

Er hängt stark von der Stelle ab, an der der Algorithmus in die Bildgenerierungspipeline eingefügt wird. Kann er so integriert werden, daß die Berechnung des Beleuchtungsmodells nur für die sichtbaren Polygone durchgeführt wird und die Farbe anschließend als Attribut mitgeführt wird, so ist der Berechnungsaufwand linear abhängig von der Anzahl der sichtbaren Polygone.

Bewertung:

Die resultierende Darstellung der Körper ist nicht realistisch. Obwohl ein Tiefeneindruck entsteht, sind die Polygone deutlich voneinander unterscheidbar.

Neben diesem Nachteil bestehen Einschränkungen in der Auswahl des Beleuchtungsmodells. Durch die konstante Färbung eines Polygons ist die Verwendung von Modellen für die spiegelnde Reflexion nicht sinnvoll.

Anwendungsgebiet:

Entsprechen die Polygone jeweils nur wenigen Pixeln auf dem Bildschirm (sog. Mikropolynome), so kann diese Art der Beleuchtungsberechnung durchaus sinnvoll sein.

Gouraud–Algorithmus (7.11.3.2, S.501)

Der *Gouraud–Algorithmus* basiert auf dem Scan–line–Algorithmus von Watkins. Es handelt sich dabei um die Interpolation der an den Ecken eines Dreiecks berechneten Leuchtdichten. Diese Werte werden entlang der Kanten des Dreiecks und anschließend entlang einer Rasterzeile interpoliert, was sich in den Vorgang der Rasterung integrieren läßt. Dadurch entsteht ein inkrementelles Verfahren, das sehr effizient realisiert werden kann.

Aufwand:

Der Berechnungsaufwand setzt sich im günstigsten Fall additiv aus einem in der Anzahl der sichtbaren Polygone linearen und einem in der Anzahl der Pixel des Bildschirms linearen Aufwand zusammen.

Bewertung:

Die visuellen Ergebnisse sind befriedigend für diffuse Körper, wenn auch noch die sogenannten **Machband–Effekte** zu beobachten sind.

Die Berücksichtigung eines spekularen Anteils ist nicht sinnvoll. Gouraud-Shading wird meist mit diffuser Reflexion assoziiert.

Machband–Effekt (7.11.3.2, S.503)

Als Machband–Effekt werden helle oder dunkle Linien bezeichnet, die an Stellen auftreten, an denen sich die Leuchtdichten schnell ändern bzw. dis-

kontinuierlich sind.

Phong–Algorithmus (7.11.3.3, S.503)

Das *Phong-Modell* erfaßt auch spiegelnde Effekte. Dazu werden die Normalen benötigt. Deshalb werden nicht die Leuchtdichten, sondern die Normalenvektoren interpoliert. Diese Berechnung kann ebenfalls inkrementell sein und integriert in die Rasterung aufgenommen werden. Allerdings muß jeder Normalenvektor vor Auswertung des Beleuchtungsmodells normiert werden. Nach der Interpolation wird in jedem Punkt die Leuchtdichte berechnet.

Aufwand:

Es entsteht im günstigsten Fall ein in der Anzahl der sichtbaren Polygone und in der Anzahl der Pixel sich additiv zusammensetzender linearer Aufwand. Die Konstanten sind hier wesentlich größer als bei dem Gouraud–Algorithmus.

Bewertung:

Die Resultate sind im allg. realistischer als bei **konstanter Beleuchtung** oder beim **Gouraud-Algorithmus**, die Mach–Band–Effekte sind reduziert.

globale Beleuchtungsverfahren (7.12, S.507)

Im Gegensatz zu den **lokalen Beleuchtungsverfahren** steht bei den globalen Verfahren nicht eine einfache Realisierung, sondern eine möglichst realistische Wiedergabe einer Szene im Vordergrund. Um eine solche photorealistische Darstellung zu erreichen, sollten die physikalischen Vorgänge so exakt wie möglich modelliert werden.

In der GDV werden zwei unterschiedliche Verfahren eingesetzt:

1. **Raytracing,**
2. **Radiosity.**

Raytracing (Strahlverfolgung) (7.12, S.507)

Raytracing simuliert den Prozeß der Lichtausbreitung und arbeitet dabei nach den Gesetzen für ideale Spiegelung und Brechung. Daher ist Raytracing vor allem für Szenen mit hohem spiegelnden und transparenten Flächenanteil gut geeignet.

Die Grundidee besteht darin, Lichtstrahlen auf ihrem Weg von der Quelle bis zum Auge zu verfolgen. Zur Vereinfachung werden beim konventionellen Raytracing nur ideal reflektierte und ideal gebrochene Strahlen weiterverfolgt. Da nur wenige Strahlen das Auge erreichen, kehrt man das Verfahren um (Reziprozität der Reflexion) und sendet durch jedes Pixel des Bildschirms einen vom Augpunkt ausgehenden Strahl in die Szene. Trifft der Sehstrahl auf ein Objekt, so wird das **lokale Beleuchtungsmodell** berechnet. Dann werden zwei neue Strahlen erzeugt, nämlich der reflektierte und der gebrochene (transmittierte) Sehstrahl. Der Leuchtdichtebeitrag dieser beiden Strahlen wird rekursiv berechnet.

Dieser Prozeß bricht ab, wenn eine Lichtquelle getroffen wird, die auf dem Strahl transportierte Energie zu gering wird oder wenn der Sehstrahl die Szene verläßt. Aus praktischen Erwägungen wird man eine Obergrenze für die Rekursionstiefe festlegen. Mit Hilfe dieses Algorithmus wird das

Verdeckungsproblem implizit gelöst. Die Schattenberechnung wird durchgeführt, indem man von den Auftreffpunkten des Sehstrahls sogenannte Schattenstrahlen zu den Lichtquellen der Szene sendet. Nur wenn kein undurchsichtiges Objekt zwischen einer Lichtquelle und dem Auftreffpunkt liegt, trägt sie zur Beleuchtung bei.
Zum **Beleuchtungsmodell nach Phong** (vgl. GDV II, Abschnitt 2.5.3 'Lokale Beleuchtungsalgorithmen'), das den lokalen Anteil modelliert, kommen der ideal spiegelnde und der transmittierte Anteil hinzu:

$$L_{ges} = L_{Phong} + r_r L_r + r_t L_t,$$

wobei

L_r	die Leuchtdichte auf dem reflektierten Strahl,
L_t	die Leuchtdichte auf dem transmittierten Strahl,
r_r	der Reflexionsgrad für die Idealreflexion,
r_t	der Reflexionsgrad für die Idealtransmission ist.

Beschleunigungstechniken beim Raytracing (7.12, S.509)

Es gibt die folgenden Möglichkeiten zur Beschleunigung des Raytracing-Verfahrens:

a) Verringerung der durchschnittlichen Kosten der Schnittpunktberechnung zwischen einem Strahl und einem Szeneobjekt

b) Verringerung der Gesamtanzahl der Strahl-Objekt-Schnittpunkttests:

- **Raumunterteilung mit regulären Gittern**
- **Raumunterteilung mit Octrees**
- **Szenenunterteilung mit hierarchischen Bäumen**
- **Mailbox-Technik**
- **Schatten-Caches**
- **adaptive Rekursionstiefenkontrolle**
- **Pixel-Selected Raytracing**

Raumunterteilung beim Raytracing (7.12, S.509)

In einem Vorverarbeitungsschritt wird der dreidimensionale Objektraum in nichtüberlappende Unterräume unterteilt (sog. Voxel = Volumenelement). Bei der Strahlverfolgung bestimmt dann ein Traversierungsalgorithmus der Reihe nach die vom Strahl durchlaufenen Unterräume. Schnittpunktberechnungen müssen dann nur noch mit denjenigen Objekten durchgeführt werden, die zumindest teilweise in diesen Unterräumen liegen. Wurde ein Schnittpunkt gefunden und ist dieser Schnittpunkt der nächste im aktuellen Voxel, dann kann die Traversierung stoppen.

Raumunterteilung mit regulären Gittern (Raytracing) (7.12, S.510)

Das regelmäßige 3D-Gitter teilt den Objektraum in endlich viele, gleich

große, achsenparallele Voxel. Es eignet sich sehr gut für kleine und mittelgroße Szenen mit nicht allzu ungleichmäßig verteilten Szenenprimitiven. Das 3D-Gitter ist leicht zu implementieren und kann mit sehr geringem Rechenaufwand von einem Strahl durchlaufen werden (etwa mit dem Strahltraversierungsalgorithmus von Amanatides und Woo) .

Schatten-Caches (Raytracing) (7.12, S.510)

Bei der Verfolgung von Schattenstrahlen beim Raytracing genügt es zu wissen, daß mindestens ein Schnittpunkt des Strahls mit einem undurchsichtigen Objekt existiert. Aus diesem Grund verwaltet jede Lichtquelle einen Schatten-Cache, der das Resultat der Strahlverfolgung des letzten Schattenstrahls speichert. War dieser Schattenstrahl auf seinem Weg von der Lichtquelle zum Objektschnittpunkt auf ein undurchsichtiges Objekt gestoßen, so wird ein Pointer auf dieses schattenwerfende Objekt im Schatten-Cache gespeichert. Ansonsten wird ein Null-Pointer gespeichert. Bevor der nächste Schattenstrahl verfolgt wird, führt man einen Schnittpunkttest mit dem durch den Schatten-Cache referenzierten Objekt durch. Nur wenn dieses Schnittestergebnis negativ ist, wird der Schattenstrahl explizit durch die Szene verfolgt.

adaptive Rekursionstiefenkontrolle (Raytracing) (7.12, S.511)

Eine ad-hoc-Technik zur Vermeidung unnötiger Strahlverfolgungen besteht darin, (reflektierte/ transmittierte/...) Sekundärstrahlen nicht weiter zu verfolgen, wenn ihr Beitrag zur Pixelfarbe zu klein wird.
Eine generelle Regel lautet, daß Strahlen weiter unten im Strahlbaum weniger zur Pixelfarbe beitragen. Bevor ein Sekundärstrahl durch die Szene verfolgt wird, wird sein maximaler Intensitätsbeitrag zur Pixelfarbe geschätzt. Liegt dieser Wert unterhalb eines Schwellwertes, so wird der Strahl nicht erzeugt.

Pixel-Selected-Raytracing (7.12, S.511)

Eine schnelle Previewing-Technik zur Beschleunigung der Visualisierung besteht darin, homogene Regionen in der Bildebene zu detektieren und zu interpolieren. Die Technik basiert auf einer *Divide-and-Conquer*-Strategie.

Formfaktor (2, S.513)

Die Formfaktoren treten in der GDV in der Radiosity-Gleichung des **Radiosity-Verfahrens** auf:
Die Größe $F_{dA_i\text{-}dA_j}$ nennt man den *Formfaktor* zwischen dem Flächenelement dA_i und dem Flächenelement dA_j. Der Formfaktor ist eine dimensionslose Zahl zwischen 0 und 1. Er gibt an, wieviel Lichtstrom des Flächenelements dA_i auf dem Flächenelement dA_j ankommt.
Der Formfaktor $F_{dA_i\text{-}dA_j}$ zwischen den Flächenelementen dA_i und dA_j ist gegeben durch

$$F_{dA_i\text{-}dA_j} = \frac{\cos\vartheta_j \cos\vartheta_i}{\pi R^2} dA_j.$$

Um auch die Verdeckung von Flächen mit zu beschreiben, wird für jedes differentielle Flächenpaar dA_i, dA_j eine Funktion H_{ij} eingeführt, die den

Wert 1 hat, falls keine andere Fläche zwischen dA_i und dA_j liegt, andernfalls den Wert 0. Die allgemeine Formel lautet dann

$$F_{dA_i\text{-}dA_j} = H_{ij}\,\frac{\cos\vartheta_j\,\cos\vartheta_i}{\pi R^2}\,dA_j.$$

Durch Integration über die Fläche A_j erhält man den Bruchteil des Lichtstroms der infinitesimalen Fläche dA_i, der auf der endlichen Fläche A_j landet:

$$F_{dA_i\text{-}A_j} = \int_{A_j} H_{ij}\,\frac{\cos\vartheta_j\,\cos\vartheta_i}{\pi R^2}\,dA_j.$$

Den Formfaktor zweier endlicher Flächen bekommt man durch Integration über die Fläche A_i und anschließende Mittelwertbildung (Division durch A_i):

$$F_{A_i\text{-}A_j} = F_{ij} = \frac{1}{A_i}\int_{A_i}\int_{A_j} H_{ij}\,\frac{\cos\vartheta_j\,\cos\vartheta_i}{\pi R^2}\,dA_j dA_i.$$

Eigenschaften der Formfaktoren:

a) Reziprozitätsbeziehung:

$$A_i F_{ij} = A_j F_{ji}.$$

b) Aus Gründen der Energieerhaltung gilt für jede abstrahlende Fläche A_i:

$$\sum_{j=1}^{N} F_{ij} \leq 1.$$

c) Jede konvexe, im Grenzfall ebene, Fläche strahlt kein Licht auf sich selbst ab:

$$F_{ii} = 0.$$

Enthält die Szene nur planare Flächen, so sind bei N Flächen nur $N(N-1)/2$ Formfaktoren zu berechnen.

Als Beispiele im Kurs berechnet werden:

a) Formfaktor zwischen differentieller Fläche und Kreisscheibe,

b) Formfaktor zwischen differentieller Fläche und Polygon.

Im Kurs behandelt werden die folgenden Verfahren zur Berechnung von Formfaktoren:

- **Nusselts Analogon** (Vorbereitung des Hemi-Cube-Verfahrens)
- **Hemi-Cube-Verfahren** (Berechnung von Delta-Formfaktoren)
- **Formfaktorberechnung mit Raytracing**

Radiosity-Verfahren (7.13, S.512)

Das Radiosity-Verfahren ist ein globales Beleuchtungsverfahren für ideal diffus reflektierende Oberflächen. Es werden nicht nur die Wechselwirkung der Oberflächen mit den Lichtquellen, sondern auch die Wechselwirkung der Oberflächen untereinander berücksichtigt.
Berechnet wird im Radiosity-Verfahren die Beleuchtungsstärke (gemessen in Lux) einer jeden Fläche.
Anwendungsgebiet:
Lichtverteilung in Gebäuden.
Vorteile:
Betrachtungsunabhängigkeit: Bei diffuser Reflexion ist die Leuchtdichte einer Fläche unabhängig von der Beobachtungsrichtung. Wurde eine Szene mit dem Radiosity-Verfahren berechnet, so ist es möglich, die Szene aus allen Richtungen zu betrachten, ja sogar durch die Szene hindurchzugehen, ohne sie neu berechnen zu müssen.
Vereinfachungen:
Beim Radiosity-Verfahren werden nur die rein diffusen Reflexionen berücksichtigt. Die Oberflächen der Objekte werden durch eine endliche Zahl kleiner, ebener Flächen, sogenannter Patches, beschrieben. Jedes Patch hat einen einheitlichen Reflexionsgrad und eine einheitliche Radiosity. Die Radiosity einer Fläche j wird dann beschrieben durch:

$$B_j = E_j + \rho_j H$$

mit

$$H = \sum_{i=1}^{N} F_{ij} B_i \frac{A_i}{A_j},$$

wobei N die Anzahl der Flächen in der Szene ist.
Der **Formfaktor** F_{ij} beschreibt den Bruchteil der auf Fläche j ankommenden Strahlungsleistung der Gesamtausstrahlung der Fläche i. In den Formfaktoren ist die gesamte geometrische Information der Szene enthalten.

Aus den beiden obigen Gleichungen erhält man durch Einsetzen die Grundgleichung des Radiosity-Verfahrens, ein lineares Gleichungssystem:

$$B_j = E_j + \rho_j \sum_{i=1}^{N} F_{ij} B_i \frac{A_i}{A_j}.$$

Vor der Lösung des Gleichungssystems müssen die Formfaktoren F_{ij} bestimmt werden, was ungleich aufwendiger ist als die Lösung des Gleichungssystems.

Verfahren zur Lösung des Gleichungssystems:

- Eliminationsverfahren von Gauß
- iteratives Gauß-Seidel-Verfahren
- iteratives **Progressive Refinement**-Verfahren

Farben im Radiosity-Verfahren:
Wenn man die Wellenlängenabhängigkeit der Größen in der Radiosity-Glei-

chung beachtet, erhält man:

$$B_j(\lambda) = E_j(\lambda) + \rho_j(\lambda) \sum_{i=1}^{N} F_{ji} B_i(\lambda) \qquad \text{für alle Wellenlängen } \lambda.$$

Es ist sinnvoll, die Radiosity-Gleichung für aneinandergrenzende Wellenlängenbänder $\Delta\lambda$ zu lösen. In der GDV wird aber für gewöhnlich die Radiosity-Gleichung nur für die Grundfarben R, G und B eines RGB-Monitors gelöst. Das Ergebnis einer Radiosity-Rechnung liefert einen Radiosity-Wert für jedes Patch der Szene. Durch Skalierung kann dieser Wert in eine darstellbare Farbe umgerechnet werden.

Progressive Refinement (Radiosity) (7.13.2, S.519)

Das Progressive-Refinement-Verfahren beruht darauf, daß man die Ausbreitung des Lichts durch die Szene ausgehend von den primären Lichtquellen verfolgt. Die Radiosity, die eine Fläche A_j an eine Fläche A_i weitergibt, ist dabei gegeben als

$$B_{j\to i} = \rho_i B_j F_{ij} = \rho_i B_j F_{ji} \frac{A_j}{A_i}.$$

Zu Beginn des Progressive-Refinement-Verfahrens werden die Radiosities B_i mit den Lichtquellentermen E_i initialisiert. Jeder Iterationsschritt besteht darin, die sogenannte unverteilte Radiosity einer Fläche A_j der Szene auf alle anderen zu verteilen.

Vorteile des Verfahrens:

- schnelle Konvergenz,
- zur Durchführung eines Iterationsschrittes genügt die Kenntnis einer Zeile der Formfaktormatrix,
- nach wenigen Iterationsschritten werden im allg. schon sehr gute Ergebnisse erzielt.

Nusselts Analogon (Radiosity) (7.13.2.1, S.521)

Wilhelm Nusselt konstruierte eine Halbkugel mit Einheitsradius, in deren Zentrum sich das Flächenelement dA_i befindet. Für die Berechnung von Formfaktoren wird die Oberfläche der Halbkugel in kleine Segmente aufgeteilt. Für jedes dieser Segmente wird der Formfaktor zwischen dem Segment und der Fläche dA_i durch Parallelprojektion auf die Basis berechnet. Nachdem man diese sogenannten *Delta-Formfaktoren* berechnet hat, wird jede Fläche A_j der Szene auf die Halbkugel projiziert, wobei registriert wird, welche Segmente auf der Halbkugel von der Projektion der Fläche A_j bedeckt werden. Der gesamte Formfaktor der Fläche A_j ist dann einfach die Summe der Delta-Formfaktoren der von A_j bedeckten Segmente.
Das Nusseltsche Analogon hat seine Bedeutung darin, daß es die Grundlage zu dem ersten brauchbaren Verfahren zur Berechnung von Formfaktoren ist, nämlich dem **Hemi-Cube-Verfahren.**

Hemi-Cube-Verfahren (Radiosity) (7.13.2.2, S.523)

Statt der Halbkugel bei **Nusselts Analogon** verwendet man bei diesem Verfahren Halbwürfel (Hemi-Cubes). Jede Seite des Halbwürfels wird mit einem rechteckigen Raster überzogen, den sogenannten Hemi-Cube-Pixeln. Die Projektion der Szene auf eine Fläche des Hemi-Cube stellt somit das gleiche Problem dar, wie die Darstellung einer dreidimensionalen Szene auf einem Raster-Display, weshalb auf die üblichen Scan-Konvertierungsverfahren zurückgegriffen werden kann. Beim Hemi-Cube muß auf fünf Rasterflächen projiziert werden: auf die obere Fläche und auf die vier Seitenflächen. Jedem Hemi-Cube-Pixel q wird ein Deltaformfaktor ΔF_q zugeordnet, der den Beitrag von q zum gesamten Formfaktor beschreibt. Den Formfaktor einer Fläche A_j erhält man, indem man die Deltaformfaktoren derjenigen Hemi-Cube-Pixel, die durch die Projektion von A_j bedeckt werden, aufsummiert.

Durch Verwendung des z-Buffer-Algorithmus für die Scan-Konvertierung der Szene auf die Hemi-Cube-Flächen ist es nun möglich, auch die Verdeckung von Flächen zu berücksichtigen.

Formfaktorberechnung mit Raytracing (7.13.3, S.526)

Während beim Hemi-Cube-Verfahren die Lichtquelle die differentielle Fläche darstellt, über die der Hemi-Cube gelegt wird, ist es beim *Verfahren von Wallace* genau umgekehrt: Die Lichtquelle wird als endliche Fläche aufgefaßt und von ihr aus werden die Formfaktoren zu infinitesimalen Flächenelementen berechnet.

Der große Vorteil dieser Vorgehensweise ist, daß die Formfaktoren und damit die Radiosities genau an den gewünschten Stellen berechnet werden.

Die eigentliche Formfaktorberechnung wird durch eine Kombination von **Raytracing** und analytischer Formfaktorberechnung bewerkstelligt. Das Raytracing dient dabei lediglich zur Verdeckungsrechnung.

Index

Glossar

adaptive Rekursionstiefenkontrolle (Raytracing) (7.12, S.511)

Eine ad-hoc-Technik zur Vermeidung unnötiger Strahlverfolgungen besteht darin, (reflektierte/ transmittierte/...) Sekundärstrahlen nicht weiter zu verfolgen, wenn ihr Beitrag zur Pixelfarbe zu klein wird.
Eine generelle Regel lautet, daß Strahlen weiter unten im Strahlbaum weniger zur Pixelfarbe beitragen. Bevor ein Sekundärstrahl durch die Szene verfolgt wird, wird sein maximaler Intensitätsbeitrag zur Pixelfarbe geschätzt. Liegt dieser Wert unterhalb eines Schwellwertes, so wird der Strahl nicht erzeugt.

affine Abbildung (3.1.3, S.105)

Eine Abbildung $\Phi\colon A_1 \to A_2$ zwischen zwei affinen Räumen A_1 und A_2 ist affin, wenn

$$\Phi\left(\sum_{i=0}^{n} \lambda_i \cdot p_i\right) = \sum_{i=0}^{n} \lambda_i \cdot \Phi(p_i)$$

für jede endliche Folge $\lambda_0, \cdots, \lambda_n \in \mathbb{R}$ mit $\sum_{i=0}^{n} \lambda_i = 1$ gilt.
Aus der Definition folgt, daß eine affine Abbildung eindeutig durch die Abbildung der affinen Basis festgelegt ist.

affiner Raum (3.1.1, S.101)

Eine Menge A^n heißt *n-dimensionaler affiner Raum*, falls ein n-dimensionaler reeller Vektorraum V^n existiert, so daß folgende drei Bedingungen erfüllt sind:

(i) Zu jedem geordneten Paar (p,q), $p,q \in A^n$, gehört ein Vektor $v \in V^n$. Man schreibt $v = (\vec{pq})$.

(ii) Zu jedem $p \in A^n$ und jedem $v \in V^n$ existiert ein eindeutig bestimmtes $q \in A^n$, so daß $v = (\vec{pq})$.

(iii) Ist $v = (\vec{pq})$ und $w = (\vec{qr})$, dann gilt $v + w = (\vec{pr})$.

Die Elemente des affinen Raumes A^n heißen *Punkte*;
V^n heißt der zu A^n *assoziierte Vektorraum*.
Die Elemente aus V^n heißen Vektoren. Aus (iii) folgt, daß $(\vec{pp}) = 0$ und $(\vec{pq}) = -(\vec{qp})$.

affiner Unterraum eines Vektorraumes (3.1.2, S.103)

Eine Teilmenge X eines Vektorraumes V heißt *affiner Unterraum* von V, falls es ein $v \in V$ und einen Untervektorraum $W \subset V$ gibt, so daß $X = v + W = \{u \in V |$ es gibt ein $w \in W$ mit $u = v + w\}$ gilt.

Beispiel:
Die eindimensionalen Unterräume sind Geraden.

Affinkombination (3.1.2, S.104)

Der Vektor v ist eine *Affinkombination* der Vektoren v_i, wenn

$$v = \sum_{i=0}^{n} \lambda_i v_i \quad \text{mit} \quad \sum_{i=0}^{n} \lambda_i = 1$$

gilt.

Algorithmus von de Boor für B-Spline-Flächen (4.7.4.3, S.204)

Zur Berechnung von Flächenpunkten $q(u_0, v_0)$ kann wieder der de Boor-Algorithmus eingesetzt werden. Dabei wird der Algorithmus zunächst für $u = u_0$ angewandt, d.h. es werden die de Boor-Punkte

$$d_j(u_0) = \sum_{i=0}^{o} d_{ij} N_i^m(u_0),\ j = 0, \cdots, p,$$

berechnet. Danach wird der de Boor-Algorithmus noch einmal mit $v = v_0$ und den Punkten $d_j(u_0)$ durchlaufen:

$$q(u_0, v_0) = \sum_{j=0}^{p} d_j(u_0) N_j^n(v_0).$$

Algorithmus von de Boor (4.6.2.2, S.187)

Die Verallgemeinerung des de Casteljau-Algorithmus von Bézier-Kurven auf B-Splines ist der *de Boor-Algorithmus.*
Gegeben sei ein B-Spline

$$q(u) = \sum_{i=0}^{m} N_i^n(u) \cdot d_i$$

vom Grad n über T.
Für $t_l \leq u < t_{l+1}$ betrachten wir das durch die Rekursion

$$d_i^0(u) := d_i, \qquad i = l-n, ..., l,$$

und

$$\begin{aligned} d_i^r(u) &:= (1 - \frac{u - t_{i+r}}{t_{i+n+1} - t_{i+r}}) \cdot d_i^{r-1}(u) \\ &+ \frac{u - t_{i+r}}{t_{i+n+1} - t_{i+r}} \cdot d_{i+1}^{r-1}(u),\ i = l-n, ..., l-r, \end{aligned}$$

für $0 \leq r \leq n$ definierte Dreiecksschema.
Dann gilt $q(u) = d_{l-n}^n(u)$.

Im Spezialfall $T = (\underbrace{s, ..., s}_{n+1}, \underbrace{t, ..., t}_{n+1})$ geht der de Boor-Algorithmus in den Algorithmus von de Casteljau über.

Algorithmus von de Casteljau (4.3.2.1, S.179)

Neben der stabilen *Berechnung der Funktionswerte* einer Bézier-Kurve (mittels fortgesetzter Bildung von Konvexkombinationen) erlaubt dieser Algorithmus darüber hinaus die *Bestimmung von Ableitungen* und die *Unterteilung der Kurve* in mehrere Segmente. Die bei fortgesetzter Unterteilung entstehenden neuen Bézier-Polygone konvergieren gegen die Bézier-Kurve und können zur graphischen Darstellung der Kurve verwendet werden.
Berechnung des Funktionswertes $q(u)$ an der Stelle u:
Rekursion:

$$\begin{aligned} b_i^0(u) &= b_i \\ b_i^r(u) &= \frac{(u-s)}{(t-s)} \cdot b_{i+1}^{r-1}(u) + (1 - \frac{(u-s)}{(t-s)}) \cdot b_i^{r-1}(u) \end{aligned}$$

Ableitung von Bézier-Kurven:
Die i-te Ableitung $q^{(i)}(u)$ ergibt sich aus dem de Casteljau-Schema in der $(n-i)$-ten Stufe zu

$$q^{(i)}(u) = \frac{1}{(t-s)^i} \frac{n!}{(n-i)!} \sum_{k=0}^{i} \begin{pmatrix} i \\ k \end{pmatrix} (-1)^{i-k} \cdot b_k^{n-i}(u).$$

Für die *erste Ableitung* gilt daher

$$q'(u) = \frac{n}{(t-s)} (b_1^{n-1}(u) - b_0^{n-1}(u)).$$

Unterteilung einer Bézier-Kurve in zwei Teilsegmente:
Für $s \leq q \leq t$ gilt die Beziehung

$$q(u) = \begin{cases} \sum_{i=0}^{n} {}_s^q B_i^n(u) \cdot b_0^i(q), & s \leq u \leq q \\ \\ \sum_{i=0}^{n} {}_q^t B_i^n(u) \cdot b_i^{n-i}(q), & q \leq u \leq t \end{cases}.$$

Hierbei bezeichnen ${}_s^q B_i^n(u)$ bzw. ${}_q^t B_i^n(u)$ die Bernstein-Polynome bezüglich der Teilintervalle $[s,q]$ bzw. $[q,t]$.

Darstellung einer Bézier-Kurve:
Bei fortgesetzter Unterteilung konvergiert die Folge der verfeinerten Kontrollpolygone gleichmäßig gegen die Kurve. Bei fortgesetzter Halbierung des Parameterintervalls ist diese Konvergenz exponentiell und liefert ein praktisches Verfahren zur Darstellung einer Bézier-Kurve durch eine *stückweise lineare Approximation.*

allgemeine perspektivische Transformation (3.3.5.7, S.134)
Bei der praktischen Anwendung von Projektionen liegt der Standort des Betrachters beliebig im Objektraum. Eine allgemeine perspektivische Transformation wird durch die Festlegung eines Augpunktes (*Eye*), eines Blickbezugspunktes (*VRef*) und der Angabe einer Oben-Richtung (*ViewUp*) festgelegt.
Die Berechnung der Transformation wird in vier Schritten durchgeführt:

a) Berechnung des Bildschirmkoordinatensystems mit Ursprung *VRef* und den Basisvektoren v'_x, v'_y und v'_z.
v'_z wird durch *VRef* und *Eye* definiert.
v'_x steht senkrecht auf den Vektoren *ViewUp* und v'_z.
Da die $y-$Achse des Bildschirmkoordinatensystems in die entgegengesetzte Richtung des *ViewUp*-Vektors zeigt, bilden die drei Vektoren v'_x, $ViewUp$ und v'_z ein linkshändiges, noch nicht notwendig orthogonales, Koordinatensystem.
Der Vektor *ViewUp* wird daher durch den Vektor v'_y senkrecht zu v'_z und v'_x ersetzt, so daß v'_x, v'_y und v'_z ein rechtshändiges orthogonales Koordinatensystem bilden.
Die Vektoren v'_x, v'_y und v'_z müssen normiert werden!

b) Translation des Bildschirmkoordinatensystems in den Ursprung.

c) Rotation des Bildschirmkoordinatensystems auf das Welt- bzw. Referenzkoordinatensystem:

$$v'_x \to x; \quad v'_y \to y; \quad v'_z \to z$$

d) Perspektivische Transformation mit Abstand $|VRef - Eye|$, wobei der Augpunkt (*Eye*) auf der negativen z-Achse liegt.

Animation (Java3D) (6.2.11.1, S.360)

Bei Java3D die Veränderung des **virtuellen Universums** (inklusive der Szene), ohne dass Aktionen (bzw. Eingabedaten) des Benutzers einen Einfluss auf den Start und den Ablauf der Animation haben. In der Regel werden die Veränderungen durch den Ablauf bestimmter Zeitintervalle initiiert.
Es wird zwischen **Interaktionen** und **Animationen** unterschieden.
Im allgemeinen Sprachgebrauch ist eine Animation die „Erstellung von Bildserien, die bei hinreichend schneller Bildabfolge den Eindruck von kontinuierlicher Bewegung hervorrufen" ([Cla97], S. 337).

Anwendung (6.1, S.323)

Anwendungsprogramm, im Gegensatz zu beispielsweise Betriebssystemen und Bibliotheken.

API (6.1, S.321)

Siehe **Application Programming Interface**.

Application Programming Interface (6.1, S.321)

Ausdruck für die Gesamtheit der Schnittstellen einer Reihe von Klassen oder Prozeduren einer Software-Bibliothek. Wird auch **API** genannt.

B-Spline-Basisfunktionen (4.6.1, S.185)

Gegeben seien $n \leq m \in \mathbb{N}$ sowie eine schwach monotone Folge

$$T = (t_0 = ... = t_n, t_{n+1}, ..., t_m, t_{m+1} = ... = t_{m+n+1})$$

von *Knoten* mit $t_i < t_{i+n+1}, 0 \leq i \leq m$.
Die normalisierten *B-Splines* N_i^n vom Grad n über T sind rekursiv definiert durch

$$N_i^0(u) := \begin{cases} 1, & \text{falls } t_i \leq u < t_{i+1} \\ 0, & \text{sonst} \end{cases}$$

und

$$N_i^r(u) := \frac{u - t_i}{t_{i+r} - t_i} \cdot N_i^{r-1}(u) + \frac{t_{i+1+r} - u}{t_{i+1+r} - t_{i+1}} \cdot N_{i+1}^{r-1}(u)$$

für $1 \leq r \leq n$.
Für $t_{i+r} = t_i$ bzw. $t_{i+r+1} = t_{i+1}$ ist in obiger Gleichung der entsprechende Summand gleich 0 zu setzen.

Eigenschaften der normalisierten B-Splines $N_i^n(u)$:

a) $N_i^n(u)$ besteht stückweise aus Polynomen vom Grad n über T.

b) Die Funktionen N_i^n besitzen einen lokalen Träger, d.h. $N_i^n(u) = 0$ für $u \notin [t_i, t_{i+n+1}]$.

c) Es gilt $N_i^n(u) \geq 0$ für alle $u \in [t_0, t_{m+n+1}]$.

d) Für $u \in [t_0, t_{m+n+1}]$ gilt: $\sum_i N_i^n(u) = 1$.

e) Ist t_j, $j \in \{0, \cdots, m+n+1\}$ ein einfacher Knoten, d.h. $t_{j-1} \neq t_j \neq t_{j+1}$, so ist $N_i^n(t_j)$ mindestens $C^{n-1}-$stetig.

Bei einem Mehrfachknoten $s = t_{j+1} = ... = t_{j+\mu}$ der Multiplizität μ sind die normalisierten B-Splines N_i^n vom Grad n mindestens $C^{n-\mu}$-stetig.

B-Spline-Fläche (4.7.4.3, S.203)

Seien $m < o$ und $n < p$ sowie

$$S = (s_0 = \cdots = s_m, \cdots, s_{o+1} = \cdots = s_{m+o+1})$$

und

$$T = (t_0 = \cdots = t_n, \cdots, t_{p+1} = \cdots = t_{p+n+1})$$

schwach monotone Folgen von Knoten und seien

$$N_i^m(u),\ i = 0, \cdots, o, \quad \text{bzw.} \quad N_j^n(v),\ j = 0, \cdots, p,$$

B-Splines über S und T.
Dann werden analog zu den Bézier-Tensorprodukt-Flächen die Tensorprodukt B-Spline-Flächen vom Grad (m, n) definiert durch

$$q : [s_0, s_{m+o+1}] \times [t_0, t_{n+p+1}] \to \mathbb{R}^3$$

mit

$$q(u, v) \longmapsto \sum_{i=0}^{o} \sum_{j=0}^{p} d_{ij} N_i^m(u) N_j^n(v),\ d_{ij} \in \mathbb{R}^3.$$

Die Punkte d_{ij} heißen *de Boor-Punkte* und bilden das *de Boor-Netz*.

B-Spline-Kurve (4.6.2, S.186)

Gegeben seien $n \leq m \in \mathbb{N}$ sowie eine schwach monotone Folge

$$T = (t_0 = ... = t_n, t_{n+1}, ..., t_m, t_{m+1} = ... = t_{m+n+1})$$

von Knoten mit $t_i < t_{i+n+1}$ und Punkte $d_0, ..., d_m \in \mathbb{R}^3$.
Dann heißt eine Kurve

$$q(u) = \sum_{i=0}^{m} N_i^n(u) \cdot d_i, \qquad d_i \in \mathbb{R}^3,$$

eine *B-Spline-Kurve* (oder einfach ein *B-Spline*) vom Grade n über T.
Der Grad kann dabei vom Benutzer gewählt werden. Häufig beschränkt man sich jedoch auf kubische B-Splines.
Die Punkte $d_0, ..., d_m$ heißen *Kontroll-* oder *de Boor-Punkte* von q. Sie bilden das *Kontroll-* oder *de Boor-Polygon*.

baryzentrische Koordinaten (3.1.2.1, S.104)

Sei $\mathcal{B} = \{p_0; (p_0\vec{p}_1), \cdots, (p_0\vec{p}_n)\}$ ein Koordinatensystem in einem affinen Raum A^n und sei ein Punkt

$$p = \sum_{i=0}^{n} \lambda_i \cdot p_i \quad \text{mit} \quad \sum_{i=0}^{n} \lambda_i = 1$$

als Affinkombination bezüglich dieses Koordinatensystems gegeben.
Dann heißen die Koeffizienten λ_i *baryzentrische Koordinaten* oder *Schwerpunktskoordinaten* von p.

Basiseigenschaft der Bernstein-Polynome (4.8.1.1, S.205)

Wie die Bernstein-Polynome eine Basis für den Raum der reellwertigen univariaten Polynome (Polynome in einer Variablen) über einem Intervall bilden, so bilden die bivariaten Bernstein-Polynome (Polynome in zwei Variablen)

$${}_{RST}B^n_{i,j,k},\ i+j+k=n,$$

eine Basis für den Raum aller reellwertigen bivariaten Polynome vom Grad n.

behavior (Java3D) (6.2.11.2, S.360)

Behavior (Verhalten) ist die Änderung einiger Eigenschaften von **visuellen Objekten** zur Laufzeit. Eigenschaften können beispielsweise die Position, Orientierung, Größe oder Farbe oder Kombinationen davon sein. Ein 'behavior' kann entweder eine **Animation** oder eine **Interaktion** sein.
Siehe auch **Behavior-Klasse**.

Behavior-Klasse (Java3D) (6.2.11.2, S.360)

Sie legt fest, wie sich bestimmte Parameter von **Datenelementen** im **Szenegraphen** zur Laufzeit ändern. Wird beispielsweise die durch ein **TransformGroup**-Datenelement definierte Matrix modifiziert, ändert sich die Position, Orientierung oder Größe. Behavior-Klassen werden bei der Erstellung von **Animationen** sowie von interaktiven Programmen (siehe **Interaktion)** eingesetzt.
Siehe auch **behavior**.

Beleuchtungsmodell (7.7, S.467)

Ein *Beleuchtungsmodell* ist eine Vorschrift zur Berechnung der Farb- und Grauwerte der einzelnen Bildpunkte eines Bildes.
In einem solchen Modell werden die Einflüsse der Lichtquellen (Lage, Größe, Stärke, spektrale Zusammensetzung) sowie der Oberflächenbeschaffenheit (Geometrie, Reflexionseigenschaften) auf die Farbe eines Bildpunktes erfaßt.
Das Modell muß also den Vorgang der Lichtausbreitung in einer definierten Umgebung beschreiben und sich auch (mit vernünftigem Aufwand) berechnen lassen.

Beleuchtungsmodell nach Phong (7.11.2.3, S.499)

Nach dem Phong'schen Modell erhält man dann bei n Lichtquellen die

Leuchtdichte in einem Punkt aus (vgl. **lokales (empirisches) Beleuchtungsmodell**):

$$L_{Phong} = L_{amb} + \sum_{i=1}^{n} (L_{diff,i} + L_{spec,i}).$$

Beleuchtungsstärke (1, S.474)

Das photometrische Gegenstück zur **Bestrahlungsstärke** ist die *Beleuchtungsstärke*. Die Einheit der Beleuchtungsstärke ist das *Lux*, abgekürzt *lx*.

Beleuchtungsverfahren (7.11, S.494)

Beleuchtungsmodelle beschreiben die Zusammenhänge zwischen Raumgeometrie, Lichtquellen und Oberflächenbeschaffenheit zur Berechnung der Leuchtdichte in einem Punkt der darzustellenden Szene.
Beleuchtungsalgorithmen berechnen aus der **Leuchtdichte** einiger ausgewählter Punkte die darzustellende Farbe aller Bildpunkte.
Die Kombination von Beleuchtungsmodell und -algorithmus kennzeichnet spezielle *Beleuchtungsverfahren*.

Bernstein-Polynom bezüglich eines Referenzdreiecks (4.8.1, S.205)

Sind drei affin-unabhängige Punkte $R,S,T \in \mathbb{R}^2$ gegeben und bezeichnen $\rho(U)$, $\sigma(U)$, $\tau(U) \in \mathbb{R}$ die baryzentrischen Koordinaten eines Punktes $U \in \mathbb{R}^2$, d.h.

$$U = \rho(U)\cdot R + \sigma(U)\cdot S + \tau(U)\cdot T \in \mathbb{R}^2, \quad \rho(U)+\sigma(U)+\tau(U) = 1,$$

dann heißen die Polynome

$${}_{RST}B^n_{i,j,k}(U) := \begin{pmatrix} n \\ i,j,k \end{pmatrix} \rho(U)^i \sigma(U)^j \tau(U)^k, \quad i+j+k = n,$$

Bernstein-Polynome bezüglich des Referenzdreiecks $\triangle(R,S,T)$.
Dabei ist

$$\begin{pmatrix} n \\ i,j,k \end{pmatrix} := \frac{n!}{i!\,j!\,k!}.$$

An den Dreiecksrändern, d.h. für $\rho(U) = 0$, $\sigma(U) = 0$ oder $\tau(U) = 0$ entarten die verallgemeinerten Bernsteinpolynome zu den gewöhnlichen Bernsteinpolynomen.

Bernsteinpolynom (4.3.1, S.177)

Die *Bernsteinpolynome* über dem Intervall $[s,t]$ lassen sich aus den binomischen Formeln

$$(t-s)^n = ((t-u)+(u-s))^n = \sum_{i=0}^{n} \begin{pmatrix} n \\ i \end{pmatrix} (u-s)^i (t-u)^{n-i}$$

entwickeln. Die durch $(t-s)^n$ dividierten Summanden

$${}^t_s B^n_i(u) = \frac{1}{(t-s)^n} \begin{pmatrix} n \\ i \end{pmatrix} (u-s)^i (t-u)^{n-i}, \quad i = 0,1,\cdots,n$$

sind Polynome vom Grad n und heißen *Bernsteinpolynome* vom Grad n. Sie bilden eine Basis des Polynomraumes $\mathbb{P}^n$ und besitzen folgende Eigenschaften:

$$\begin{aligned}
\sum_{i=0}^{n} {}_s^t B_i^n(u) &= 1 \text{ für } u \in [s,t] && \text{Partition der 1}\\
{}_s^t B_i^n(u) &\geq 0 \text{ für } u \in [s,t] && \text{Positivität}\\
{}_s^t B_i^n(u) &= \tfrac{u-s}{t-s}\,{}_s^t B_{i-1}^{n-1}(u) + \tfrac{t-u}{t-s}\,{}_s^t B_i^{n-1}(u) && \text{Rekursion}\\
{}_s^t B_i^n(u) &= {}_s^t B_{n-i}^n(t-(u-s)) && \text{Symmetrie.}
\end{aligned}$$

Häufig wird als Parameterintervall das Intervall $[0,1]$ gewählt. In diesem Fall schreibt man statt ${}_0^1 B_i^n$ einfach B_i^n.

Beschleunigungstechniken beim Raytracing (7.12, S.509)

Es gibt die folgenden Möglichkeiten zur Beschleunigung des Raytracing-Verfahrens:

a) Verringerung der durchschnittlichen Kosten der Schnittpunktberechnung zwischen einem Strahl und einem Szeneobjekt

b) Verringerung der Gesamtanzahl der Strahl-Objekt-Schnittpunkttests:

- **Raumunterteilung mit regulären Gittern**
- **Raumunterteilung mit Octrees**
- **Szenenunterteilung mit hierarchischen Bäumen**
- **Mailbox-Technik**
- **Schatten-Caches**
- **adaptive Rekursionstiefenkontrolle**
- **Pixel-Selected Raytracing**

Bestrahlungsstärke (7.8.1, S.470)

Die Größe

$$E_e = I_e \frac{\cos \alpha_2}{R^2}$$

heißt *Bestrahlungsstärke*. Sie ist ein Maß für die pro Fläche auftreffende Strahlungsleistung und nimmt (im Fall punktförmiger Strahler) mit dem Quadrat der Entfernung von der Lichtquelle ab. Dieser Zusammenhang wird als *quadratisches Entfernungsgesetz* bezeichnet.
Die Einheit der Bestrahlungsstärke ist $W\,m^{-2}$.

Bézier-Kurve (4.3.2, S.178)

Die Darstellung von q mit

$$q(u) = \sum_{i=0}^{n} {}_s^t B_i^n(u) \cdot b_i, \quad b_i \in \mathbb{R}^3,$$

heißt *Bézier-Darstellung* im $\mathbb{R}^3$.
Die Punkte $b_i,\ i = 0,\cdots,n$, heißen *Bézier-Punkte* und definieren das *Bézier-Polygon.*
Da

$$\sum_{i=0}^{n} {}_s^t B_i^n(u) = 1,\ \ u \in [s,t],$$

gilt, sind die Bézier-Kurven invariant unter affinen Transformationen.

Zusammenhang zwischen Kurve und Bézier-Polygon:

a) Für $u \in [s,t]$ liegt die Kurve $q(u)$ innerhalb der konvexen Hülle des Bézier-Polygons.

b) Es gilt $q(s) = b_0$, $q(t) = b_n$, d.h. die Kurve verläuft durch Anfangs- und Endpunkt des Polygons.

c) Für die Ableitungen in den Anfangs- und Endpunkten gilt

$$q'(s) = \frac{n}{t-s}(b_1 - b_0),$$

$$q'(t) = \frac{n}{t-s}(b_n - b_{n-1}),$$

d.h. die Kurve besitzt als Tangenten die Anfangs- und Endseiten des Polygons.

d) Für die i–te Ableitung von q in den Randwerten $u = s$ bzw. $u = t$ gilt:

$$q^{(i)}(s) = \frac{1}{(t-s)^i}\frac{n!}{(n-i)!} \cdot \sum_{j=0}^{i} \binom{i}{j} (-1)^{i-j} \cdot b_j$$

bzw.

$$q^{(i)}(t) = \frac{1}{(t-s)^i}\frac{n!}{(n-i)!} \cdot \sum_{j=0}^{i} \binom{i}{j} (-1)^{j} \cdot b_{n-j},$$

d.h. die i-te Ableitung von q im Randwert $u = s$ hängt nur von dem Randpunkt selbst und seinen i unmittelbar benachbarten Bézier-Punkten ab. Analoges gilt im Randwert $u = t$.

Vor- und Nachteile:

+ Das Bézier-Polygon vermittelt eine schnelle Übersicht über den möglichen Kurvenverlauf.
+ Änderungen der Kontrollpunkte erlauben kontrollierte Änderungen des Kurvenverlaufs.
- Der Grad der Kurve ist an die Anzahl der Ecken des Bézier–Polygons gekoppelt, was zu hohen Polynomgraden führt.
- Die Änderung eines Bézier-Punktes wirkt sich auf die gesamte Kurve aus und damit auch auf Bereiche, die eventuell nicht mehr geändert werden sollen.

Bézier-Spline (4.5, S.183)
Definiert man die Segmente eines Splines durch Bézier-Polynome, so spricht man von *Bézier-Splines.* Dabei können verschiedene Stetigkeitsanforderungen an die Knotenpunkte gestellt werden.

Bézier-Spline-Flächen (4.7.4.1, S.200)

Sind $o < r$ und $s < t \in \mathbb{R}$, so definieren wir ein Tensorprodukt-Bézier-Pflaster vom Grad (m,n) durch

$$q(u,v) = \sum_{i=0}^{m}\sum_{j=0}^{n} b_{ij}\,{}_o^rB_i^m(u)\,{}_s^tB_j^n(v), \quad b_{ij} \in \mathbb{R}^3, \quad i = 0\cdots m, \; j = 0\cdots n.$$

Dabei ist $u \in [o,r]$ und $v \in [s,t]$.
Die Koeffizienten b_{ij} heißen *Bézier-Punkte* und bilden das *Bézier-Netz*.

Eigenschaften von Bézier-Pflastern:

a) Die Fläche $q(u,v)$ liegt in der konvexen Hülle des Bézier-Netzes.

b) Die Bézier-Punkte der Randkurven sind die Randpunkte des Bézier-Netzes:

$$\begin{aligned} q(u,s) &= \sum_{i=0}^{m} {}_o^rB_i^m(u)\cdot b_{i0}, \\ q(u,t) &= \sum_{i=0}^{m} {}_o^rB_i^m(u)\cdot b_{in}, \\ q(o,v) &= \sum_{j=0}^{n} {}_s^tB_j^n(v)\cdot b_{0j}, \\ q(r,v) &= \sum_{j=0}^{n} {}_s^tB_j^n(v)\cdot b_{mj}. \end{aligned}$$

c) Das Pflaster verläuft durch die Eckpunkte, d.h.

$$q(o,s) = b_{00},\, q(o,t) = b_{0n},\, q(r,s) = b_{m0},\, q(r,t) = b_{mn}.$$

d) Die Ableitungen in den Randpunkten lassen sich aus den Bézier-Punkten berechnen. Es gilt:

$$\frac{\partial}{\partial u}q(u,v)|_{u=o} = \frac{m}{r-o}\sum_{j=0}^{n} {}_s^tB_j^n(v)\cdot(b_{1j}-b_{0j}).$$

e) Der Twistvektor T_{os} im Eckpunkt mit den Parameterwerten o,s, berechnet sich aus

$$T_{os} = \frac{\partial^2}{\partial u\partial v}q(u,v)|_{u=o,\,v=s} = \frac{mn}{(r-o)(t-s)}(b_{00}-b_{01}+b_{11}-b_{10}).$$

Algorithmen zur Manipulation und Auswertung von Bézier-Kurven übertragen sich direkt auf die Tensorprodukt-Bézier-Pflaster.

Bild (image) (6, S.321)

Ein rechteckiges Feld von **Pixeln**. Es ist entweder im Arbeitsspeicher oder im **Bildspeicher** abgelegt. Der Begriff Bild wird im Zusammenhang mit der Bildgenerierung in OpenGL verwendet.

Bildebene (image plate) (6.2.8.2, S.346)

Das imaginäre Rechteck im **virtuellen Universum**, auf das die Szene projiziert wird.

Bildgenerierung (6.2.3, S.326)
Allgemein: Die „Zusammenfassung einer Klasse von Darstellungsalgorithmen, die auf lokalen Beleuchtungsmodellen und interpolierenden Schattierungsalgorithmen beruhen.” [Cla97], S. 346.
Bei Java3D: Der Vorgang beim Überführen des **Szenegraphen** in ein **gerastertes Bild**.

Bildrechner (graphics display system) (2.4, S.40)
Rechner zur Darstellung komplexer graphischer Objekte. Während bei Vektorgeräten die graphischen Objekte direkt aus der Bilddefinition erzeugt und auf dem Bildschirm dargestellt werden, werden bei **Rastergeräten** die darzustellenden Objekte in Rasterpunkte (Pixel) zerlegt. Die zweidimensionale Pixelmatrix wird im Bildspeicher abgelegt. Während bei Vektorgeräten die zur Aufrechterhaltung eines Bildes notwendige Bildwiederholung durch periodisches Auswerten der Bilddefinition erfolgt, wird bei Rastergeräten die Bildwiederholung durch periodisches Auslesen des Bildspeichers realisiert.

Bildspeicher (Framebuffer/frame buffer) (6.3.3.1, S.377)
Ein „Speicherbereich, der die Farbinformation computergenerierter **Bilder** aufnimmt”. Im Bildspeicher sind für jedes einzelne **Pixel** des Bildes die zugehörigen Farbinformationen abgelegt. Der Bildspeicher enthält zu jedem Pixel entweder die Farbinformation in **Echtfarbdarstellung** oder den **Farbtabellenindex**. Der Bildspeicher „existiert im Allgemeinen auch physikalisch im Ausgabegerät”([Cla97], S. 339).

Bitmap (bitmap) (OpenGL) (6.3.3.1, S.377)
Ein rechteckiges Feld von Bits, das als [visuelles] **Primitiv** oder zum Maskieren (siehe **Maske**) von Bildausschnitten des **Bildspeicher** dient.

Bogenlänge (4.1.1.1, S.171)
Die *Bogenlänge* $s(u)$ einer parametrisierten regulären Kurve $q : [a,b] \to \mathbb{R}^3$ läßt sich gemäß folgender Formel berechnen:

$$s : [a,b] \to [0, s(b)], \quad u \mapsto s(u) := \int_a^u \|q'(t)\| dt.$$

Dabei bezeichnet $\|.\|$ die euklidische Norm im $\mathbb{R}^3$.

BoundingBox-Klasse (Java3D) (6.2.11.4, S.364)
Instanzen von BoundingBox definieren einen quaderförmigen Grenzbereich, dessen Kanten parallel zu den drei Achsen des Koordinatensystems sind. Siehe auch **Bounds-Klasse**.

BoundingSphere-Klasse (Java3D) (6.2.11.4, S.364)
Instanzen von BoundingSphere definieren einen kugelförmigen Grenzbereich. Siehe auch **Bounds-Klasse**.

Bounds-Klasse; bounding volume (Java3D) (6.2.11.4, S.364)
Mit der abstrakten Klasse Bounds (Grenzen) definiert der Entwickler die Grenzen eines dreidimensionalen konvexen und abgeschlossenen Raums (bounding volume, Grenzbereich). Häufig ist der Raum kugel- oder quaderförmig. Dieser Raum ist mit einem **behavior**, mit Licht, Sound oder anderen Leistungsmerkmalen von Java3D verknüpft. Bounds können unter anderem mit einer **BoundingSphere** oder **BoundingBox** definiert werden. Siehe auch **scheduling bound**.

branch graph (Java3D) (6.2.7.8, S.341)
Ein Teilbaum eines **Szenegraphen**, dessen Wurzel ein **BranchGroup**-Datenelement ist.

BranchGroup-Klasse (Java3D) (6.2.7.4, S.336)
Wie **TransformGroup** eine Subklasse von **Group**. BranchGroup-Datenelemente dienen ausschließlich der Gruppierung von weiteren **Datenelementen**. Jedes BranchGroup-Datenelement ist - wie jeder innere Knoten eines **Szenegraphen** - die Wurzel eines Teilgraphen des Szenegraphen. Alle Knoten bzw. Datenelemente in einem solchen Teilgraphen werden logisch gruppiert. Der Teilgraph wird **branch graph** genannt, branch von Verzweigung.
Für die Kinder eines BranchGroup-Datenelements gilt das gleiche wie für TransformGroup-Datenelemente: Ihre Anzahl ist beliebig und alle Kinder sind Instanzen der Subklassen von **Group** oder/und der Klasse **Leaf**. Allerdings dürfen nur BranchGroup-Datenelemente an ein **Locale**-Datenelement angehängt werden.

Brechungsgesetz von Snellius (7.8.3.4, S.478)
Einfallender Strahl, Normale und gebrochener Strahl liegen in einer Ebene. Der Sinus des Einfallwinkels steht zum Sinus des Brechungswinkels in einem konstanten Verhältnis, das nur von der Natur der beiden Medien abhängt:

$$n_1 \, sin\vartheta_1 \; = \; n_2 \, sin\vartheta_2 \Leftrightarrow \frac{sin\,\vartheta_1}{sin\,\vartheta_2} = \frac{n_2}{n_1} = const.$$

n_1 bzw. n_2 sind dabei die **Brechzahlen** (Brechungsindizes) der Medien.

Brechzahl (7.8.3.4, S.478)
Die *Brechzahl* ist definiert als das Verhältnis der Lichtgeschwindigkeit im Vakuum zur Lichtgeschwindigkeit im betreffenden Medium. Der Brechungsindex des Vakuums ist gleich 1.

Bresenham-Algorithmus (2.5.1, S.66)
Effizienter Algorithmus zur Rasterung von Strecken. Der Algorithmus arbeitet ausschließlich mit ganzen Zahlen und verwendet lediglich die Operationen Addition, Subtraktion und Shift.

Candela (1, S.473)

Eine *Candela* ist die Lichtstärke in einer bestimmten Richtung einer Strahlungsquelle, die monochromatische Strahlung der Frequenz $540\,THz$ (entspricht der Wellenlänge von ca. $555\,nm$) aussendet und deren Strahlstärke in dieser Richtung $\frac{1}{683} W\,sr^{-1}$ beträgt.

Capabilities (Java3D) (6.2.11.1, S.360)

Eine spezielle Art von Bit-Variablen. Bei Java3D wird der Zugriff auf die Parameter eines Datenelementes im **Szenegraphen** mit Capabilities kontrolliert. Beispielsweise kann der Wert eines Parameters eines **TransformGroup**-Datenelements nicht verändert werden, bis die zu diesem Parameter zugehörige Capability gesetzt wird.

CIE-Primärvalenzen (7.9.2, S.485)

Um nicht für jedes Primärvalenztripel von neuem die Spektralwertkurven (vgl. Stichwort **Spektralwert**) bestimmen zu müssen, wurde schon 1931 von der *Commision International de l'Éclairage* (CIE) ein Standardsystem von Primärvalenzen vorgeschlagen. Das besondere an diesen CIE-Primärvalenzen ist, daß sie außerhalb der Spektralvalenzkurve liegen. Sie sind also gar nicht darstellbar; man nennt solche Farbvalenzen deshalb *virtuell*.

Client-Server-Architektur (OpenGL) (6.3.4.5, S.383)

„Ein Modell der Betrachtung von Hard- und Softwarearchitekturen, in dem die eine Seite, der „Client", Dienstleistungen nutzt, die die andere Seite, der „Server", anbietet. Diese Form der Modellierung wird hauptsächlich im Kontext von Netzen und in der objektorientierten Programmierung angewendet.
OpenGL ist als ein „Server" konzipiert, der auf einem anderen Rechner ausgeführt werden kann als das ihn benutzende Programm."([Cla97], S. 340)

Clipping (5.1, S.252)

Bei der Ausgabe von Bildern tritt oft das Problem auf, aus der gesamten vorhandenen Bildinformation nur einen Ausschnitt darzustellen. Um ein fehlerfreies Bild zu erhalten, muß die außerhalb des Fensters liegende Bildinformation vor der Bildausgabe abgeschnitten werden (clipping = Klippen). Bildelemente außerhalb des Fensters können zum Überlauf der Koordinatenadressierung des Gerätes führen (Wraparound).

- Der Cohen-Sutherland-Algorithmus (**Cohen-Sutherland-Clipping**) beschreibt das Clippen von Vektoren an rechteckigen Fenstern.
- Der **Weiler-Atherton-Algorithmus** beschreibt das Clippen von Polygonen an beliebigen Fenster-Polygonen.
- Der **w-Clip** löst das dreidimensionale Clippen in homogenen Koordinaten und löst elegant das Wraparound-Problem.

CMG–Modell (7.10.1.1, S.487)

Zur Bezeichnung von Farben für die Plotterausgabe ist ein zum RGB–Würfel komplementäres Modell zweckmäßig, bei dem Weiß im Koordi-

natenursprung liegt. Auf den Koordinatenachsen werden dann die Primärfarben Cyan, Magenta und Gelb abgetragen, Schwarz liegt bei $[1,1,1]$.
Die Umrechnung zwischen beiden Modellen ist einfach:
Von RGB nach CMGe:

$$[C,M,Ge] = [1,1,1] - [R,G,B],$$

und von CMGe nach RGB:

$$[R,G,B] = [1,1,1] - [C,M,Ge].$$

Cohen-Sutherland-Algorithmus (5.1.3, S.253)

Der Cohen-Sutherland-Algorithmus erlaubt das **Clippen** von Vektoren an rechteckigen Fenstern. Im zweidimensionalen Fall wird jedem Vektorendpunkt ein Outcode zugeordnet, mit Hilfe dessen einfach bestimmt werden kann, ob ein Vektor außerhalb oder innerhalb des Fensters liegt oder die Fensterbegrenzung schneidet. Im letzteren Fall müssen Schnittpunktberechnungen mit den Fenstergrenzen durchgeführt werden.

Coons-Pflaster (4.9.1, S.208)

Konstruktion eines viereckigen Pflaster

$$q(u,v),\ (u,v) \in [0,1] \times [0,1] \subset \mathbb{R}^2,$$

dessen vier parametrisierte Randkurven

$$q(u,0), q(u,1),\ u \in [0,1], \quad q(0,v), q(1,v),\ v \in [0,1]$$

gegeben sind:

Coons-Pflaster:

$$Q(u,v) = Q_1(u,v) + Q_2(u,v) - ((1-u),u) \begin{pmatrix} q(0,0) & q(0,1) \\ q(1,0) & q(1,1) \end{pmatrix} \begin{pmatrix} 1-v \\ v \end{pmatrix}$$

mit

$$\begin{aligned} Q_1(u,v) &= (1-v)q(u,0) + vq(u,1), \quad (u,v) \in [0,1] \times [0,1], \\ Q_2(u,v) &= (1-u)q(0,v) + uq(1,v), \quad (u,v) \in [0,1] \times [0,1]. \end{aligned}$$

.
Verallgemeinertes Coons-Pflaster:

Es lassen sich anstelle linearer Bindefunktionen auch andere Paare von Bindefunktionen $f_1(u)$, $f_2(u)$ und $g_1(v), g_2(v)$ verwenden, z.B. kubische Hermite-Polynome. Diese müssen am Rand ihres Definitionsintervalls $[0,1]$ folgende Bedingungen erfüllen:

$$\begin{array}{lllllll} f_1(0) & = & 1 & \text{und} & f_1(1) & = & 0 \\ f_2(0) & = & 0 & \text{und} & f_2(1) & = & 1 \\ g_1(0) & = & 1 & \text{und} & g_1(1) & = & 0 \\ g_2(0) & = & 0 & \text{und} & g_1(1) & = & 1. \end{array}$$

Bezeichnet man die Matrix der Eckpunkte $\begin{pmatrix} q(0,0) & q(0,1) \\ q(1,0) & q(1,1) \end{pmatrix}$ mit A und wählt sonst dieselben Bezeichnungen wie oben, so läßt sich das verallgemeinerte Coons-Pflaster Q schreiben als

$$Q = Q_1 + Q_2 - (f_1, f_2) A \begin{pmatrix} g_1 \\ g_2 \end{pmatrix}.$$

Durch die Wahl der Funktionen f_1, f_2, g_1, g_2 ist die Interpolationsfläche festgelegt.

DAG (Java3D) (6.2.7.5, S.338)
directed acyclic graph, siehe **gerichteter azyklischer Graph**.

Darstellung von Farben (7.9.1, S.480)
Im Kurs werden zwei Möglichkeiten zur Farbdarstellung vorgestellt:

- Farben können als Vektoren eines dreidimensionalen Vektorraumes aufgefaßt werden. Die Vektoren dieses 'Farbraums' heißen *Farbvalenzen*. Die Länge eines Vektors ist ein Maß für die **Leuchtdichte** und heißt *Farbwert*, seine Richtung bestimmt die *Farbart*.
  Mit drei linear unabhängigen Basisvektoren *(Primärvalenzen)* $\mathcal{R}$, $\mathcal{G}$ und $\mathcal{B}$ läßt sich für jede Farbvalenz $\mathcal{F}$ eine Farbgleichung aufstellen:

  $$\mathcal{F} = r\mathcal{R} + g\mathcal{G} + b\mathcal{B}.$$

  Die Werte für r, g und b können nur durch ein Mischexperiment gewonnen werden.
- Für die bildliche Darstellung einer Farbtafel benutzt man häufig eine zweidimensionale Darstellung mit Hilfe von **baryzentrischen Koordinaten** (vgl. GDV I):
  Ist eine Farbvalenz F gegeben durch

  $$\mathcal{F} = R_F\,\mathcal{R} + G_F\,\mathcal{G} + B_F\,\mathcal{B},$$

  so sind die baryzentrischen Koordinaten

  $$r_F = \frac{R_F}{R_F + G_F + B_F}$$

  $$g_F = \frac{G_F}{R_F + G_F + B_F}$$

  $$b_F = \frac{B_F}{R_F + G_F + B_F}.$$

Darstellungsliste (display list) (OpenGL) (6.3.3.2, S.380)
„Eine Zusammenfassung von grafischen Befehlen zu einer Einheit mit eigenem Namen. In OpenGL werden diese Listen auf dem Server abgelegt, so dass sie bei Aufruf im Allgemeinen effizienter zu bearbeiten sind und die Kommunikation zwischen Client und Server reduziert wird" ([Cla97], S. 340). Siehe auch **Client-Server-Architektur**.

Datenelement (Java3D) (6.2.7.3, S.335)
Ein **Szenegraph** besteht aus **Knoten** und **Kanten**. Die Knoten werden bei Java3D durch Instanzen bestimmter Klassen repräsentiert. Da es bei Java3D auch Node-Klassen gibt, verwenden wir den Begriff Datenelement anstelle von Knoten. Auch die Blätter des Szenegraphen bezeichnen wir als Datenelemente.

Datenstrukturen für BReps (7.3, S.455)
Für Boundary-Modelle können wir die folgenden Datenstrukturen verwenden:

- *Polygonorientierte Datenstrukturen:*
  Der Körper besteht aus einer Sammlung von Facetten, welche in einer Tabelle mit Bezeichnern für jede Facette zusammengefaßt werden.
- *Knotenorientierte Datenstrukturen:*
  Knoten werden unabhängig gespeichert und dann den einzelnen Facetten zugeordnet.
- *Kantenorientierte Datenstrukturen:*
  Ein kantenorientiertes Boundary-Modell stellt eine Facette als Zyklus von Kanten dar. Die Knoten einer Facette sind implizit durch die Kanten dargestellt.
  Das wichtigste Beispiel einer kantenorientierten Datenstruktur ist die *Winged-Edge-Datenstruktur:*
  Zusätzlich zu den in einer einfachen kantenorientierten Datenstruktur gespeicherten Beziehungen werden noch Nachbarschaftsbeziehungen zwischen den einzelnen Kanten explizit gespeichert.

diffuse Flächenlichtquelle (7.11.1.2, S.495)
Diese Lichtquelle spielt für **globale Beleuchtungsverfahren**, bei denen alle beleuchteten und reflektierenden Flächen als Sender berücksichtigt werden, eine große Rolle (siehe **Radiosity-Verfahren**). Sie wird durch die Fläche A, die Normale N und die spezifische Lichtausstrahlung $M(\lambda)$ in Lumen/Quadratmeter (lm/m^2) angegeben. Da die Fläche definitionsgemäß diffus strahlt, ist ihre **Leuchtdichte** durch

$$L = \frac{M}{\pi}$$

und die **Lichtstärke** durch

$$dI = \frac{M}{\pi} \cdot dA \cdot cos\vartheta,$$

gegeben, wobei ϑ der Winkel zwischen N und der Richtung zum betrachteten Punkt ist.

display list (OpenGL) (6.3.4.12, S.391)
Siehe **Darstellungsliste**.

Dreiecksflächen in Bernsteinbasis (4.8.2, S.205)

Gegeben sei ein Referenzdreieck $\triangle(R,S,T)$ und ein bivariates Polynom $q:\mathbb{R}^2\to\mathbb{R}^3$. Die Darstellung

$$q(U)=\sum_{i+j+k=n} {}_{RST}B^n_{i,j,k}(U)\cdot b_{i,j,k},\qquad b_{i,j,k}\in\mathbb{R}^3,$$

von q in der Bernstein-Basis bezüglich $\triangle(R,S,T)$ mit Koeffizienten $b_{i,j,k}\in\mathbb{R}^3$ heißt *Bézier-Darstellung* von q bezüglich $\triangle(R,S,T)$.
Die Punkte $b_{i,j,k}$ heißen *Kontroll-* oder *Bézier-Punkte* von q. Sie bilden das *Kontroll-* oder *Bézier-Netz*.

Eigenschaften von Bézier-Dreiecksflächen:

a) Für $U\in\triangle(R,S,T)$ liegt $q(U)$ innerhalb der abgeschlossenen konvexen Hülle des Kontrollpolygons.

b) $q(R)=b_{n,0,0}$, $q(S)=b_{0,n,0}$ und $q(T)=b_{0,0,n}$, d.h. die Fläche verläuft durch die drei Eckpunkte des Bézier-Polygons.

c) Die Fläche ist in den Eckpunkten $q(R)$, $q(S)$ und $q(T)$ tangential an das Kontrollnetz.

d) Die Einschränkung von q auf die Gerade (S,T) ist eine Bézier-Kurve mit den Bézier-Punkten $b_{0,n-k,k}$, d.h. für $U\in(S,T)$ gilt

$$q(U)=\sum_{k=0}^{n} {}_{S}^{T}B^n_k(U)\cdot b_{0,n-k,k}.$$

Analoges gilt für die Geraden (R,S) und (T,R).

e) Die Kurve ist affin invariant, d.h.

$$\Phi\left(\sum_{i+j+k=n} {}_{RST}B^n_{i,j,k}(U)\cdot b_{i,j,k}\right)=\sum_{i+j+k=n} {}_{RST}B^n_{i,j,k}(U)\cdot\Phi(b_{i,j,k}).$$

ebene geometrische Projektion (3.3.5, S.120)

Die *ebenen geometrischen Projektionen* sind dadurch charakterisiert, daß mit Projektionsstrahlen konstanter Richtung, d.h. entlang von Geraden, auf Ebenen projiziert wird.
Diese Projektionen werden in diesem Skript durch eine invertierbare Transformation und anschließende Parallelprojektion entlang einer Koordinatenachse dargestellt.
Zu diesen Projektionen gehören die *Parallelprojektion (rechtwinklige Projektion, schiefwinklige Projektion)* und die *perspektivischen Projektionen*.

Echtfarbdarstellung (true color) (6.3.3.1, S.377)

Eine Farbe kann entweder in Echtfarbdarstellung oder durch einen **Farbtabellenindex** festgelegt werden. Bei der Echtfarbdarstellung erfolgt „eine direkte Umsetzung von Farbkomponenten in darstellbare Werte des Ausgabegerätes. Die Berechnung der Farben erfolgt anhand ihrer Komponenten" ([Cla97], S. 341), z.B. je ein Wert für die Farbkomponenten rot, grün und blau.

Eckpunkt (vertex) (4, S.340)

Die Punkte, mit denen **geometrische Objekte** definiert werden. Beispielsweise der Anfangs- oder Endpunkt einer Strecke oder die Punkte in den „Ecken" des Randes eines Polygons.

Emission (7.8.3.1, S.474)

Die *Emission* wird mit der radiometrischen Größe **Strahldichte** bzw. ihrem photometrischen Gegenstück, der **Leuchtdichte**, beschrieben. Im allgemeinen Fall ist die Strahldichte L von der Emissionsrichtung und von der Wellenlänge λ abhängig. Die beiden Polarwinkel φ und ϑ legen die Abstrahlungsrichtung fest. Die gesamte Emission erhält man durch Integration über den sichtbaren Wellenlängenbereich

$$L'(\vartheta,\varphi) = \int_{\lambda=360nm}^{830nm} L'_\lambda(\lambda,\varphi,\vartheta)\,d\lambda.$$

euklidischer Raum (3.1.2.3, S.105)

Wenn der zum affinen Raum A^n assoziierte Vektorraum V^n ein n-dimensionaler Raum mit Skalarprodukt ist, so heißt A^n *euklidischer Raum.*
(Für V^n gibt es dann eine orthonormale Basis $(e_1,\ldots,e_n)$.)

Evaluator (evaluator) (OpenGL) (6.3.3.2, S.378)

„In OpenGL werden mit Hilfe von Evaluatoren Koordinaten bzw. Parameter von Flächen auf der Basis von Bézierkurven oder -flächen berechnet" ([Cla97], S. 341).

Fächer aus Dreiecken (6.2.10.2, S.357)

Bei Fächern aus Dreiecken wird im Vergleich zu Linienzügen zusätzlich der dritte Eckpunkt automatisch mit dem ersten Eckpunkt verbunden. Wie auch beim **Streifen aus Dreiecken** bilden der erste, zweite und dritte Eckpunkt ein Dreieck. Anschließend wird der vierte Eckpunkt ebenfalls mit dem ersten Eckpunkt verbunden. Der erste, dritte und vierte Eckpunkt bilden ein zweites Dreieck. Analog werden alle weiteren Eckpunkte mit dem ersten Eckpunkt verbunden.

Farbgebung (7.9.1, S.480)

Die *Farbgebung* auf einem Graphikmonitor geschieht auf der Grundlage der additiven Farbmischung. Jeder Bildpunkt des Monitors besteht aus drei dicht beieinanderliegenden Bildpunkten. Aufgrund des beschränkten räumlichen Auflösungsvermögens des Auges werden die drei Farbkomponenten im Auge zu einem einheitlichen Farbreiz gemischt.
Bei üblichen Monitoren setzt sich die Farbe aus einer roten, einer grünen und einer blauen Komponente zusammen (RGB-Monitor).
Für jede Komponente können unterschiedliche Intensitätsstufen gewählt werden. Üblich sind heute 256 Intensitätsstufen.
Es genügen drei Grundfarben, um sämtliche Farbtöne darzustellen (vgl. **Graßmannsches Gesetz**).

Farbmetrik (7.9, S.479)

Farbmetrik ist die Lehre von den Maßbeziehungen der Farben untereinander. In der GDV wird sie aus folgendem Grund benötigt:
Die **Leuchtdichte** wird an jeder Stelle der Szene bestimmt und ein Rasterbild berechnet, das ein wirklichkeitsgetreues Abbild der Szene ist. Da es keine Ausgabegeräte gibt, die direkt mit einer spektralen Leuchtdichteverteilung für jeden Bildpunkt ansprechbar sind, wird üblicherweise ein Bildpunkt eines Ausgabegerätes durch seine Farbe charakterisiert.

Farbmodell (7.10, S.487)

Die unterschiedlichen Farbmodelle lassen sich wie folgt unterteilen:

- *Technisch-physikalische Farbmodelle:*
  - **RGB-Modell**
  - **CMG-Modell**
  - **YIQ-Modell**
  - **YUV-Modell**

  Sie beschreiben eine Farbe als Mischung dreier Primärfarben. Die Unterschiede zwischen den einzelnen Modellen liegen in der Wahl der Primärfarben und der Art der Farbmischung.
  Diese Modelle sind an den Erfordernissen der Ausgabegeräte und Übertragungstechnik orientiert. Sie eignen sich wenig zur direkten Farbdefinition durch den Benutzer.
- *Wahrnehmungsorientierte Farbmodelle:*
  Sie arbeiten mit den Größen Helligkeit, Farbton und Farbsättigung.
  Je nach Art der Beschreibung dieser Größen lassen sich zwei Gruppen von wahrnehmungsorientierten Farbmodellen unterscheiden:

  a) Die Farbe wird in numerisch-symbolischer Form beschrieben:
     - **HLS-System**
     - **H'L'S'-System**
     - HSV-System, H'S'V'-System
     - Ridgway-, Ostwald-, Munsell-, DIN- und Hesselgren- Systeme

  b) Die Farbe wird umgangssprachlich beschrieben:
     - CNS = Color-Naming-System
     - ISCC-NBS-Lexikon
     - ACNS

  Diese Größen müssen zur Ausgabe in ein technisch-physikalisches Modell transformiert werden.
  Im Kurs behandelt wurden die Transformationsalgorithmen
  - RGB nach HLS-/H'L'S'
  - HLS /H'L'S' nach RGB.

Farbtabelle (color look-up table) (6.3.3.1, S.378)

Tabelle, die die **Farbtabellenindizes** in darstellbare Farben des Ausgabegerätes umsetzt: Die Farbtabelle ordnet jedem Farbtabellenindex je einen Wert für jede Farbkomponente (z.B. rot. grün und blau) zu.

Farbtabellenindex (color index) (6.3.3.1, S.377)

Eine Farbe kann entweder in **Echtfarbdarstellung** oder durch einen Farbtabellenindex festgelegt werden. Ein Farbtabellenindex ist ein ganzzahliger Wert. Die durch den Farbtabellenindex beschriebenen Farben werden mit Hilfe einer **Farbtabelle** in die vom Ausgabegerät darstellbaren Farben umgesetzt.

Fenstersystem (window system) (OpenGL) (6.2.4.1, S.326)

Darunter verstehen wir das OpenGL umgebende Betriebs- oder Fenstersystem. Es stellt u.a. **Framebuffer** und z-Buffer bereit, initialisiert und verwaltet den **Zustand** und sorgt für die Bearbeitung von Interaktionen.

Flächenkonstruktion mittels Kurven (4.9, S.207)

Konstruktion von Flächen mittels Interpolation von Kurven:
a) Coons-Pflaster
b) Gordon-Fläche

Formfaktor (2, S.513)

Die Formfaktoren treten in der GDV in der Radiosity-Gleichung des **Radiosity-Verfahrens** auf:
Die Größe $F_{dA_i\text{-}dA_j}$ nennt man den *Formfaktor* zwischen dem Flächenelement dA_i und dem Flächenelement dA_j. Der Formfaktor ist eine dimensionslose Zahl zwischen 0 und 1. Er gibt an, wieviel Lichtstrom des Flächenelements dA_i auf dem Flächenelement dA_j ankommt.
Der Formfaktor $F_{dA_i\text{-}dA_j}$ zwischen den Flächenelementen dA_i und dA_j ist gegeben durch

$$F_{dA_i\text{-}dA_j} = \frac{\cos\vartheta_j \cos\vartheta_i}{\pi R^2} dA_j.$$

Um auch die Verdeckung von Flächen mit zu beschreiben, wird für jedes differentielle Flächenpaar dA_i, dA_j eine Funktion H_{ij} eingeführt, die den Wert 1 hat, falls keine andere Fläche zwischen dA_i und dA_j liegt, andernfalls den Wert 0. Die allgemeine Formel lautet dann

$$F_{dA_i\text{-}dA_j} = H_{ij} \frac{\cos\vartheta_j \cos\vartheta_i}{\pi R^2} dA_j.$$

Durch Integration über die Fläche A_j erhält man den Bruchteil des Lichtstroms der infinitesimalen Fläche dA_i, der auf der endlichen Fläche A_j landet:

$$F_{dA_i\text{-}A_j} = \int_{A_j} H_{ij} \frac{\cos\vartheta_j \cos\vartheta_i}{\pi R^2} dA_j.$$

Den Formfaktor zweier endlicher Flächen bekommt man durch Integration über die Fläche A_i und anschließende Mittelwertbildung (Division durch

A_i):

$$F_{A_i\text{-}A_j} = F_{ij} = \frac{1}{A_i}\int_{A_i}\int_{A_j} H_{ij}\,\frac{\cos\vartheta_j\,\cos\vartheta_i}{\pi R^2}\,dA_j\,dA_i.$$

Eigenschaften der Formfaktoren:

a) Reziprozitätsbeziehung:

$$A_i F_{ij} = A_j F_{ji}.$$

b) Aus Gründen der Energieerhaltung gilt für jede abstrahlende Fläche A_i:

$$\sum_{j=1}^{N} F_{ij} \leq 1.$$

c) Jede konvexe, im Grenzfall ebene, Fläche strahlt kein Licht auf sich selbst ab:

$$F_{ii} = 0.$$

Enthält die Szene nur planare Flächen, so sind bei N Flächen nur $N(N-1)/2$ Formfaktoren zu berechnen.

Als Beispiele im Kurs berechnet werden:

a) Formfaktor zwischen differentieller Fläche und Kreisscheibe,

b) Formfaktor zwischen differentieller Fläche und Polygon.

Im Kurs behandelt werden die folgenden Verfahren zur Berechnung von Formfaktoren:

- **Nusselts Analogon** (Vorbereitung des Hemi-Cube-Verfahrens)
- **Hemi-Cube-Verfahren** (Berechnung von Delta-Formfaktoren)
- **Formfaktorberechnung mit Raytracing**

Formfaktorberechnung mit Raytracing (7.13.3, S.526)

Während beim Hemi-Cube-Verfahren die Lichtquelle die differentielle Fläche darstellt, über die der Hemi-Cube gelegt wird, ist es beim *Verfahren von Wallace* genau umgekehrt: Die Lichtquelle wird als endliche Fläche aufgefaßt und von ihr aus werden die Formfaktoren zu infinitesimalen Flächenelementen berechnet.
Der große Vorteil dieser Vorgehensweise ist, daß die Formfaktoren und damit die Radiosities genau an den gewünschten Stellen berechnet werden.
Die eigentliche Formfaktorberechnung wird durch eine Kombination von **Raytracing** und analytischer Formfaktorberechnung bewerkstelligt. Das Raytracing dient dabei lediglich zur Verdeckungsrechnung.

Fragment (fragment) (OpenGL) (6.3.3.2, S.379)
Ein Fragment ist die Vorstufe eines **Pixels** im Rahmen der **Rendering**-Pipeline. Ein Fragment besteht aus den Informationen des zukünfigen Pixels (x- und y-Wert und der Farbe in **Echtfarbdarstellung** bzw. dem **Farbtabellenindex**, Tiefenwert, Transparentwert und den Texturkoordinaten.)

Framebuffer (6.3.3.1, S.377)
Siehe **Bildspeicher**.

Funktionenraum (4.1.1.2, S.171)
$C[a,b]$ sei die Menge aller stetigen, reellen Funktionen auf dem Intervall $[a,b] \subset \mathbb{R}$. Definiert man für Elemente $f,g \in C[a,b]$ eine Addition und Skalarmultiplikation durch

$$(\alpha f + \beta g)(t) = \alpha f(t) + \beta g(t),$$

mit $\alpha,\ \beta \in \mathbb{R}$, so ist $C[a,b]$ ein reeller Vektorraum.
In diesem Vektorraum heißen n Funktionen $f_1, \cdots, f_n \in C[a,b]$ *linear unabhängig*, falls aus $\sum c_i f_i = 0$ auf $[a,b]$ immer $c_1 = c_2 = \cdots = c_n = 0$ folgt. Dabei ist $f = 0$ auf $[a,b]$, falls $f(t) = 0$ für alle $t \in [a,b]$ gilt.

Geometrie (Java3D) (6.2.2, S.326)
Bei Java3D ist Geometrie die Bezeichnung für ein **geometrisches Objekt**. Geometrien können aus den grundlegenden geometrischen Formen (**Standard-Geometrien** von Java3D) und weiteren Elementen aus Bibliotheken, die Java3D ergänzen, zusammengesetzt werden. Solche Elemente können beispielsweise Spiralen, Segmente aus Kreisen und Kugeln sowie komplexere Formen sein.
Im **Szenegraphen** repräsentieren Leaf-Datenelemente Geometrien.
In der Regel werden mit der Klasse **Shape3D** einfache Geometrien implementiert.

geometrische Stetigkeit (4.4.1, S.182)
Seien $q_1 : [a_1,b_1] \to \mathbb{R}^3$, $q_2 : [a_2,b_2] \to \mathbb{R}^3$ zwei n-mal stetig differenzierbare reguläre Kurven. q_1 und q_2 schließen an der Stelle b_1, a_2 G^n–stetig aneinander, falls es eine zu q_1 äquivalente Kurve $r_1 : [a_0,b_0] \to \mathbb{R}^3$ gibt, so daß r_1 und q_2 an der Stelle b_0, a_2 C^n–stetig aneinanderschließen.

Die G^n–Stetigkeit ist unabhängig von der Parametrisierung. Ferner sieht man aus dieser Definition, daß für reguläre Kurven aus C^n–Stetigkeit stets G^n–Stetigkeit folgt. Für nicht reguläre Kurven braucht dies nicht der Fall zu sein.

geometrisches Objekt (6.2.10.2, S.356)
(Virtuelles) **visuelles Objekt** mit geometrischer Form. Beispiele für geometrische Objekte sind Linien, Polygone und Kugeln. Geometrische Objekte sind neben dem Hintergrund und Effekten wie Nebel die einzigen sichtbaren Elemente der Szene.
Bei Java3D wird ein geometrisches Objekt mit **Geometrie** bezeichnet, bei

bei OpenGL ist damit ein **Primitiv** oder eine Menge von Primitiven gemeint.

Geometry-Klasse (Java3D) (6.2.7.4, S.338)

Eine abstrakte Klasse von Java3D, die die Superklasse von GeometryArray und damit der meisten **Standard-Geometrien** ist.

gerichtet diffuse Reflexion (spekulare Reflexion) (7.8.3.5, S.479)

Da man die beiden idealen Reflexionsarten in der Natur selten antrifft, muß man für alle Oberflächen die richtungsmäßige Verteilung der Größe $\rho_\lambda(\lambda, \varphi_r, \vartheta_r, \varphi_i, \vartheta_i)$ bestimmen. Sehr häufig tritt der Fall auf, daß ρ ein deutliches Maximum in Richtung der spiegelnden Reflexion hat und kleiner wird, je weiter man sich von dieser Richtung entfernt *(spekularer Reflexion)*.

In der GDV ist es üblich, die spekulare Reflexion in einen richtungsunabhängigen, diffusen Anteil (Index d) und einen richtungsabhängigen Anteil (Index s) aufzuspalten.

Im Kurs werden drei Modelle behandelt:

- **Phong-Modell**
- **Modell von Torrance und Sparrow**
- Modell von Cook und Torrance

gerichteter azyklischer Graph (Java3D) (6.2.7.5, S.338)

Eine Datenstruktur aus Knoten und gerichteten Kanten, in der es keine Zyklen gibt. Ein Pfad, der den Kanten in der jeweiligen Richtung folgt, darf nicht mehrmals zum gleichen Knoten führen. Jeder **Szenegraph** ist (ohne Referenzen) ein gerichteter azyklischer Graph (directed acyclic graph).

Glätten von Strecken, Glätten von Polygonkanten (2.5.4, S.74)

Die Schwarz-weiß-Darstellung von glatten Strecken führt, abhängig von der Steigung der Strecke, zu mehr oder weniger unregelmäßigen Treppenstufen. Dieser Effekt (aliasing genannt) kann durch die Verwendung von Graustufen gemildert werden. Entsprechende Verfahren werden als Verfahren zum Glätten (anti-aliasing) bezeichnet. Man unterscheidet Verfahren, die auf Pixelebene arbeiten und Verfahren, die auf Subpixelebene arbeiten (over-sampling).

globale Beleuchtungsverfahren (7.12, S.507)

Im Gegensatz zu den **lokalen Beleuchtungsverfahren** steht bei den globalen Verfahren nicht eine einfache Realisierung, sondern eine möglichst realistische Wiedergabe einer Szene im Vordergrund. Um eine solche photorealistische Darstellung zu erreichen, sollten die physikalischen Vorgänge so exakt wie möglich modelliert werden.

In der GDV werden zwei unterschiedliche Verfahren eingesetzt:

1. **Raytracing,**
2. **Radiosity.**

goniometrische Lichtquelle (7.11.1.5, S.496)

Bei dieser Lichtquelle wird die Ausbreitungscharakteristik in einem Diagramm beschrieben, das die Größe der Lichtstärke als Funktion des Winkels zur Hauptausbreitungsrichtung in einer Tabelle angibt. Zur Ermittlung der Lichtstärke in eine gewünschte Richtung muß unter Umständen zwischen Tabellenwerten interpoliert werden.

Gordon-Pflaster (4.9.2, S.210)

Ausdehnung der Konstruktion von Coons auf Systeme von Flächen:
Ist ein Netz von parametrisierten Kurven

$$q(u_i,v),\ i=0,\cdots,m, \quad \text{und} \quad q(u,v_j),\ j=0,\cdots,n,$$

gegeben, so interpoliert eine Gordon-Fläche R dieses Netz mit Hilfe eines Systems von Bindefunktionen

$$g_i(u),\ i=0,\cdots,m \quad \text{und} \quad h_j(v),\ j=0,\cdots,n:$$

$$R(u,v)=g_1(u,v)+g_2(u,v)-g_{12}(u,v)$$

mit

$$\begin{aligned} g_1(u,v) &= \sum_{i=0}^{m} q(u_i,v)L_i^m(u), \\ g_2(u,v) &= \sum_{j=0}^{n} q(u,v_j)L_j^n(v). \end{aligned}$$

und

$$g_{12}(u,v)=\sum_{i=0}^{m}\sum_{j=0}^{n} q(u_i,v_j)L_i^m(u)L_j^n(v).$$

.

Gouraud–Algorithmus (7.11.3.2, S.501)

Der *Gouraud–Algorithmus* basiert auf dem Scan–line–Algorithmus von Watkins. Es handelt sich dabei um die Interpolation der an den Ecken eines Dreiecks berechneten Leuchtdichten. Diese Werte werden entlang der Kanten des Dreiecks und anschließend entlang einer Rasterzeile interpoliert, was sich in den Vorgang der Rasterung integrieren läßt. Dadurch entsteht ein inkrementelles Verfahren, das sehr effizient realisiert werden kann.
Aufwand:
Der Berechnungsaufwand setzt sich im günstigsten Fall additiv aus einem in der Anzahl der sichtbaren Polygone linearen und einem in der Anzahl der Pixel des Bildschirms linearen Aufwand zusammen.
Bewertung:
Die visuellen Ergebnisse sind befriedigend für diffuse Körper, wenn auch noch die sogenannten **Machband–Effekte** zu beobachten sind.
Die Berücksichtigung eines spekularen Anteils ist nicht sinnvoll. Gouraud-Shading wird meist mit diffuser Reflexion assoziiert.

Graßmannsches Gesetz (7.9.1, S.480)

erstes Graßmannsches Gesetz:
Zwischen je vier Farben besteht immer eine eindeutige lineare Beziehung. Eine Farbe braucht zu ihrer Beschreibung drei voneinander unabhängige Bestimmungsstücke, d.h. die Farbe ist eine dreidimensionale Größe.
zweites Graßmannsches Gesetz:
Gleich aussehende Farben ergeben mit einer dritten Farbe stets gleich aussehende Farbmischungen.

Graderhöhung bei Bézier-Kurven: Corner Cutting (4.3.2.4, S.181)

Werden zur genaueren Approximation einer geometrischen Figur mittels einer Bézier-Kurve mehr Freiheitsgrade benötigt als vorhanden sind, so kann der Grad der Kurve erhöht werden. Bei Graderhöhung um den Grad eins wird dadurch ein weiterer Kontrollpunkt in das Kontrollpolygon eingefügt, ohne daß sich die geometrische Form der Kurve ändert.
Ist eine Bézier-Kurve q vom Grad n mit den Bézier-Punkten $b_0, \cdots, b_n$ gegeben, so ist q als Bézier-Kurve vom Grad $n+1$ darstellbar mit den Bézier-Punkten

$$\begin{aligned} b_0^* &= b_0, \\ b_j^* &= \frac{j}{n+1} \cdot b_{j-1} + (1 - \frac{j}{n+1}) \cdot b_j, \quad j = 1, \cdots, n, \\ b_{n+1}^* &= b_n. \end{aligned}$$

Wegen ihrer geometrischen Bedeutung wird die Graderhöhung auch als “Corner Cutting” bezeichnet.

Group-Klasse (Java3D) (6.2.7.4, S.336)

Group ist eine abstrakte Klasse. Zwei Subklassen von Group sind **BranchGroup** und **TransformGroup**. BranchGroup- und TransformGroup-Datenelemente repräsentieren diese Klassen als Knoten im **Szenegraphen**. Die Hauptaufgabe eines Group-Datenelements ist das Gruppieren von anderen Datenelementen (z.B. **Leaf**- und weiteren BranchGroup- oder TransformGroup-Datenelementen), die Söhne des Group-Datenelements sind.
Die Group-Datenelemente halten den Szenegraphen zusammen.

Grundgesetz der Strahlungsübertragung (7.8.1, S.471)

Das *Grundgesetz der Strahlungsübertragung* beschreibt die differentielle Strahlungsleistung, die ein differentielles Flächenelement dA_1 abstrahlt und die von einem differentiellen Flächenelement dA_2 im Abstand R von dA_1 aufgenommen wird:

$$d^2\varphi_e = L_e \frac{\cos\vartheta_1 \cos\vartheta_2}{R^2} dA_1 dA_2.$$

H’L’S’-System (7.10.2.1, S.491)

Das H’L’S’-System entsteht durch Verschieben von Grün in Richtung Blau. Dadurch liegen Rot, Gelb und Blau gleich weit voneinander entfernt, was der Farbempfindung besser entspricht (vgl. Stichwort **HLS-System**).

Hellempfindlichkeitsfunktion (7.8.2, S.472)

Der Zusammenhang zwischen radiometrischen und photometrischen Größen wird durch die *Hellempfindlichkeitsfunktion* $V(\lambda)$ hergestellt. Die Hellempfindlichkeitsfunktion gibt die relative Empfindlichkeit des Auges in Abhängigkeit von der betrachteten Wellenlänge an. Ist das Auge dunkeladaptiert (*skotopisches Sehen*), so sind nur die sogenannten Stäbchen des Auges für die Rezeption verantwortlich, während beim helladaptierten Auge (*photopisches Sehen*) nur die Farbzapfen beteiligt sind. Deshalb gibt es zwei voneinander abweichende Hellempfindlichkeitsfunktionen, V für photopisches und V' für skotopisches Sehen.

Hemi-Cube-Verfahren (Radiosity) (7.13.2.2, S.523)

Statt der Halbkugel bei **Nusselts Analogon** verwendet man bei diesem Verfahren Halbwürfel (Hemi-Cubes). Jede Seite des Halbwürfels wird mit einem rechteckigen Raster überzogen, den sogenannten Hemi-Cube-Pixeln. Die Projektion der Szene auf eine Fläche des Hemi-Cube stellt somit das gleiche Problem dar, wie die Darstellung einer dreidimensionalen Szene auf einem Raster-Display, weshalb auf die üblichen Scan-Konvertierungsverfahren zurückgegriffen werden kann. Beim Hemi-Cube muß auf fünf Rasterflächen projiziert werden: auf die obere Fläche und auf die vier Seitenflächen. Jedem Hemi-Cube-Pixel q wird ein Deltaformfaktor ΔF_q zugeordnet, der den Beitrag von q zum gesamten Formfaktor beschreibt. Den Formfaktor einer Fläche A_j erhält man, indem man die Deltaformfaktoren derjenigen Hemi-Cube-Pixel, die durch die Projektion von A_j bedeckt werden, aufsummiert.

Durch Verwendung des z-Buffer-Algorithmus für die Scan-Konvertierung der Szene auf die Hemi-Cube-Flächen ist es nun möglich, auch die Verdeckung von Flächen zu berücksichtigen.

HLS-System (7.10.2.1, S.490)

HLS-System = Hue (Farbton) - Lightness (Helligkeit) - Saturation (Sättigung) – System:

Die Farbanordnung entspricht der senkrechten Projektion des RGB–Würfels von Weiß nach Schwarz entlang der Hauptdiagonalen.

Das entstehende regelmäßige Sechseck wird meist durch einen Kreis ersetzt, so daß der *Farbton* (H) als Winkel zwischen 0° und 360° anzugeben ist.

Die *Helligkeit* (L) wird als Wert zwischen 0 und 1 angegeben, wobei 0 Schwarz und 1 Weiß entspricht.

Die *Sättigung* (S) wird als Abstand einer Farbe vom Mittelpunkt des Farbkreises angegeben. Sie beträgt 0 für achromatische Farben und kann als höchsten Wert 1 für die gesättigten Farben auf dem Rand des Farbkreises annehmen.

homogene Koordinaten (3.2.2.1, S.108)

Ist $v \in \mathbb{R}^4$ ein von Null verschiedener Vektor, so definiert v eine Gerade g durch den Ursprung des $\mathbb{R}^4$ mit der Parametrisierung $g(\lambda) = \lambda \cdot v,\ \lambda \in \mathbb{R}$, die wir zur Abkürzung mit $\mathbb{R} \cdot v$ (*projektiver Punkt*) bezeichnen.

Auf diese Weise erhält man eine Abbildung von dem Vektorraum $\mathbb{R}^4$ in den projektiven Raum $P(\mathbb{R}^4)$, genauer

$$\mathbb{R}^4 - \{0\} \to P(\mathbb{R}^4),\ v \longmapsto \mathbb{R} \cdot v.$$

Zwei von Null verschiedene Vektoren v, v' bestimmen genau dann denselben projektiven Punkt, d.h. dieselbe Gerade durch den Ursprung, wenn es ein $\lambda \in \mathbb{R} - \{0\}$ gibt mit $v' = \lambda v$.
Ist $v = (x, y, z, w) \neq 0 \in \mathbb{R}^4$ gegeben, so bezeichnen wir mit

$$[x, y, z, w] = \mathbb{R} \cdot (x, y, z, w)$$

die *homogenen Koordinaten* des projektiven Punktes $\mathbb{R} \cdot v$.
Zur Unterscheidung schreiben wir sie in eckigen Klammern.
Dabei ist zu beachten, daß immer mindestens eine der Koordinaten x, y, z, w von Null verschieden ist und daß $[x, y, z, w] = [x', y', z', w']$ genau dann gilt, wenn es ein $\lambda \neq 0$ gibt mit $x' = \lambda x, y' = \lambda y, z' = \lambda z, w' = \lambda w$.

Hybridmodell (7.6, S.465)

Keiner der drei wichtigsten Zugänge zu Solid Modelling (**Boundary-Darstellung, CSG-Darstellung** oder **Räumliche Zerlegungsmethoden**) ist für alle Anwendungen gleich gut geeignet. Daher werden bei der Implementierung oft mehrere Modelle gleichzeitig verwendet. Dabei muß auf die **Konsistenz** der Daten in den verschiedenen Repräsentationen geachtet werden. Um von einer Repräsentation in eine andere zu wechseln, sind entsprechende Konvertierungsalgorithmen notwendig.
Die CSG-Darstellung ist z.B. für den Anwender am besten als Repräsentationsart geeignet, da Modellierungsvorschriften mittels Boole'scher Operationen direkt angegeben werden können. Zur schnellen Visualisierung sind hingegen BReps besser geeignet, da das Neuauswerten der gesamten CSG-Bäume bei einer Neudarstellung nach affinen Transformationen entfällt. Ein typisches Beispiel eines Hybridmodellierers ist daher ein CSG-BRep-Modellierer. Die CSG-Datenstruktur wird dabei als Modelldatenstruktur verwendet.

ideal diffuse Reflexion (7.8.3.3, S.476)

Im einfachsten Fall ist die reflektierte **Leuchtdichte** unabhängig von der Abstrahlungsrichtung. Man nennt diesen Fall deshalb auch *ideal diffuse* oder *Lambert'sche Reflexion*. Auf dieser Art der Reflexion wird im **Radiosity-Verfahren** der Austauschmechanismus für Licht zwischen den Oberflächen der Objekte modelliert. Die reflektierte Leuchtdichte ist zwar unabhängig von den Winkeln (φ_r, ϑ_r), hängt aber von den Einstrahlungswinkeln (φ_i, ϑ_i) ab:

$$L_{\lambda,r} = \rho_\lambda E_{\lambda,i}(\lambda, \varphi_i, \vartheta_i).$$

Da die Leuchtdichte unabhängig von der Abstrahlrichtung ist, ergibt sich für die spezifische Ausstrahlung (Radiosity)

$$M = \pi L_{\lambda,r}$$

und

$$\rho_\lambda = \frac{M}{\pi E_\lambda} \left[\frac{1}{sr}\right].$$

ideal spiegelnde Reflexion (7.8.3.4, S.477)

Die spiegelnde Reflexion wird durch das aus der Optik bekannte Reflexionsgesetz beschrieben:
Der einfallende und der reflektierte Strahl bilden mit der Normalen der reflektierenden Oberfläche gleiche Winkel. Einfallender Strahl, reflektierter Strahl und Flächennormale liegen in einer Ebene.
Dies ist die strahlenoptische Formulierung des Reflexionsgesetzes. Für elektromagnetische Wellen gilt das Reflexionsgesetz in gleicher Weise.
Ist also die Richtung der einfallenden Strahlung (φ_i, ϑ_i), so wird nur in die Richtung

$$\vartheta_r = \vartheta_i \qquad \text{und} \qquad \varphi_r = \varphi_i + \pi$$

Strahlung reflektiert.
Konventionelles **Raytracing** beruht auf diesem einfachen Reflexionsgesetz.

Immediate Mode (6.3.4.11, S.390)

Bei OpenGL werden die Szenen im Immediate Mode („direkten Modus") dargestellt: Jede Codezeile wird so bald wie möglich ausgeführt. Wenn beispielsweise im Quellcode ein Primitiv definiert wird, können bei der Ausführung des compilierten Programms die ersten Befehle der Definition schon ausgeführt werden, selbst wenn das Ende der Definition noch nicht erreicht ist. Der Immediate Mode ist insbesondere für interaktive Anwendungen geeignet, die ohne spürbare Verzögerung die veränderte Szene darstellen sollen. Der Immediate Mode bietet keine Möglichkeit, einem Primitiv einen Namen oder eine Identifikationsnummer (ID) zuzuordnen, um es auch nach der Definition erneut aufrufen zu können. Um dieses Problem zu umgehen, kann man so genannte **Darstellungslisten** einsetzen.

Inhalt (content) (Java3D), (6.2.7.8, S.341)

Die Gesamtheit der **visuellen Objekte** (inklusive der Eigenschaften der Objekte und der Lichtquellen) und Audio-Objekte eines **virtuellen Universums**.

Inhaltsgraph (content branch graph) (Java3D) (6.2.7.8, S.341)

Der Teil des **Szenegraphen**, der die **visuellen Objekte** (inklusive der Eigenschaften der Objekte und der Lichtquellen) enthält. Die Wurzel eines Inhaltsgraphen ist ein **Locale**-Datenelement. Ein Szenegraph kann mehrere Inhaltsgraphen enthalten.

Interaktion (Java3D) (6.2.11.1, S.360)

Veränderung des **virtuellen Universums** (inklusive der **visuellen Objekte** der Szene) als Reaktion auf Eingabedaten des Benutzers oder in einem Kommando-Antwort-Wechselspiel. Es wird zwischen Interaktionen und **Animationen** unterschieden.

Interpolation mit kubischen Hermite-Polynomen (4.2.4, S.175)

Basisfunktionen sind die *kubische Hermite-Polynome* $H_i^3,\ i = 0,\cdots,3$, auf dem Intervall $[0,1]$:

$$\begin{aligned} H_0^3(u) &= (1-u)^2(1+2u) \\ H_1^3(u) &= u(1-u)^2 \\ H_2^3(u) &= -u^2(1-u) \\ H_3^3(u) &= (3-2u)u^2. \end{aligned}$$

In der *Hermitebasis* gelten für die Koeffizienten $c_0,\cdots,c_3$ eines Polynoms dritten Grades

$$q(u) = c_0H_0^3(u) + c_1H_1^3(u) + c_2H_2^3(u) + c_3H_3^3(u)$$

$$c_0 = q(0), \quad c_1 = q'(0), \quad c_2 = q'(1), \quad c_3 = q(1).$$

Daher können die Koeffizienten geometrisch gedeutet werden:
Sind zwei Punkte P_0, P_1 und zwei Tangentenvektoren m_0 und m_1 der Kurve in diesen Punkten zu interpolieren, so löst das Polynom

$$q(u) = P_0H_0^3(u) + m_0H_1^3(u) + m_1H_2^3(u) + P_1H_3^3(u)$$

die Interpolationsaufgabe.

Interpolation mit Lagrange-Polynomen (4.2.3, S.174)

Bei dieser Interpolationsform wird anstelle der Taylorbasis die sogenannte *Lagrangebasis* verwendet.
Für $n+1$ Knoten $(u_i, P_i), \quad i = 0,\cdots,n$, mit $u_0 < u_1 < \cdots < u_n$ definiert man die Basisfunktionen durch

$$L_i^n(u) = \frac{(u-u_0)(u-u_1)\cdots(u-u_{i-1})(u-u_{i+1})\cdots(u-u_n)}{(u_i-u_0)(u_i-u_1)\cdots(u_i-u_{i-1})(u_i-u_{i+1})\cdots(u_i-u_n)}.$$

Das Interpolationspolynom hat in dieser Basis die Form

$$q(u) = \sum_{i=0}^{n} P_iL_i^n(u) = \sum_{i=0}^{n} P_i \prod_{\substack{j=0 \\ j\neq i}}^{n} \frac{(u-u_j)}{(u_i-u_j)}$$

Für die Basispolynome L_i^n gilt:

$$L_i^n(u_k) = \delta_{ik} = \left\{ \begin{array}{ll} 1 & \text{für} \quad i = k \\ 0 & \text{für} \quad i \neq k \end{array} \right\},$$

wobei δ_{ik} das Kronecker-Symbol ist,
und

$$\sum_{i=0}^{n} L_i^n(u) = 1 \quad u \in [0,1],$$

woraus die affine Invarianz der Lagrangeschen Interpolationskurve folgt.

Vorteil der Lagrangeinterpolation:

Es müssen keine Polynomkoeffizienten berechnet werden, da diese mit den zu interpolierenden Punkten übereinstimmen.

Nachteile:

1. Der Grad der Interpolationsfunktion hängt von der Anzahl der Bestimmungsstücke ab. Entsprechendes gilt für den Rechenaufwand zur numerischen Auswertung.
2. Lokale Änderungen sind nicht möglich.
3. Bei höheren Polynomgraden zeigt die Lagrange-Interpolation eine unerwünschte Welligkeit.
4. Wie bei den Monomen können Kurvensegmente nur mit C^0-Stetigkeit aneinandergefügt werden.

Interpolation mit Monomen (4.2.2, S.173)

Sind $n+1$ Knoten (u_i, P_i) mit paarweise verschiedenen Stützstellen u_i, $i = 0, \cdots, n$, gegeben, so suchen wir ein Polynom

$$q(u) = \sum_{j=0}^{n} c_j u^j, \quad c_j \in \mathbb{R}^3,$$

mit

$$P_i = q(u_i) = \sum_{j=0}^{n} c_j (u_i)^j, \quad i = 0, \cdots, n.$$

Dazu sind die Koeffizienten c_i aus folgendem Gleichungssystem zu bestimmen:

$$\begin{pmatrix} 1 & u_0 & \cdots & u_0^n \\ \cdot & & & \cdot \\ \cdot & & & \cdot \\ \cdot & & & \cdot \\ 1 & u_n & \cdots & u_n^n \end{pmatrix} \begin{pmatrix} c_0 \\ \cdot \\ \cdot \\ \cdot \\ c_n \end{pmatrix} = \begin{pmatrix} P_0 \\ \cdot \\ \cdot \\ \cdot \\ P_n \end{pmatrix}.$$

Da die Stützstellen als paarweise verschieden vorausgesetzt sind, ist die zu dem obigen Gleichungssystem gehörende Matrix (*Vandermondsche Matrix*) invertierbar und die Koeffizienten c_i lassen sich eindeutig bestimmen.

Nachteile dieser Interpolationsmethode:

1. Die Lösung des obigen Gleichungssystems kann für großes n zu numerischen Problemen führen.
2. Wird nur ein Knoten geändert, so muß im allgemeinen das ganze System neu gelöst werden.
3. Kurven in Taylorbasis sind nicht affin invariant.
4. Durch Monominterpolation entstehende Kurvensegmente können in der Regel nur mit C^0-Stetigkeit aneinandergefügt werden.
5. Die Koeffizienten c_j sind nicht unmittelbar geometrisch interpretierbar.

Interpolationsaufgabe (4.2.1, S.172)

Gesucht ist eine Funktion q, die folgende Bedingungen erfüllt:

$$P_i = q(u_i),\ u_i \in \mathbb{R},\ P_i \in \mathbb{R}^3,\ i = 0, \cdots, n,$$

wobei die u_i die Parameterwerte der zugehörigen Punkte P_i darstellen.
Die Punkte P_i heißen *Stützpunkte*, die u_i *Stützstellen* oder Parameterwerte, der Vektor $(u_0, \cdots, u_n)$ heißt *Stützstellenvektor*.
Die Paare (u_i, P_i) legen die Knoten des Interpolationspolynoms fest.

Invarianz bei affinen Abbildungen (4.2.1, S.173)
Gilt für die Basispolynome $f_0, \cdots, f_n$ einer Polynombasis von $\mathbb{P}^n[a,b]$, $a < b, a, b \in \mathbb{R}$, und für alle $u \in [a,b]$ die Bedingung

$$f(u) = \sum_{i=0}^{n} f_i(u) = 1,$$

so gilt für eine affine Transformation Φ und Stützpunkte $P_i,\ i = 1, \cdots, n$:

$$\Phi\left(\sum_{i=0}^{n} P_i f_i(u)\right) = \sum_{i=0}^{n} \Phi(P_i) f_i(u).$$

Anschaulich gesprochen erhält man dieselbe Interpolationskurve, unabhängig davon, ob man zuerst die Stützpunkte affin verschiebt und dann die Kurve berechnet, oder ob man zuerst die Kurve berechnet und dann die einzelnen Kurvenpunkte affin transformiert.

Knoten und Kanten (6.2.7.3, S.336) Ein Szenegraph besteht wie jeder Graph aus Knoten und Kanten und enthält in der Regel zusätzlich **Referenzen**. Die Knoten werden in Java3D als **Datenelemente** bezeichnet.

Knoteneinfügen in B-Splines (4.6.2.4, S.191)
Algorithmus von Boehm:
Gegeben sei eine B-Spline-Kurve $q(u) = \sum_{i=0}^{m} N_i^n(u) \cdot d_i$ vom Grad n über dem Knotenvektor $T = (t_0, ..., t_{m+n+1})$. Nach Einfügen eines zusätzlichen Knotens t, etwa

$$t_l \leq t < t_{l+1},$$

besitzt q eine Darstellung

$$q(u) = \sum_{i=0}^{m+1} N_i^n(u) \cdot d_i^*$$

als B-Spline vom Grad n über dem verfeinerten Knotenvektor

$$T^* = (t_0, ..., t_l, t, t_{l+1}, ..., t_{n+m+1}).$$

Die neuen Kontrollpunkte d_i^* werden wie folgt berechnet:

$$d_i^* = (1 - a_i) \cdot d_{i-1} + a_i \cdot d_i$$

mit

$$a_i = \begin{cases} 1 & \text{für } i \leq l - n \\ \frac{t - t_i}{t_{i+n} - t_i} & \text{für } l - n + 1 \leq i \leq l. \\ 0 & \text{für } l + 1 \leq i \end{cases}$$

konstante Beleuchtung (flat shading) (7.11.3.1, S.500)
Der einfachste Algorithmus zur Beleuchtung polygonal begrenzter Körper belegt ein ganzes Polygon gleichmäßig mit einer Farbe, die aber von der Beleuchtung der Szene abhängt *(flat shading)*. Das bedeutet, daß nur einmal

pro Polygon eine Leuchtdichte zu berechnen ist und die Farbe als konstantes Attribut dieser Fläche mitgeführt werden kann.
Aufwand:
Er hängt stark von der Stelle ab, an der der Algorithmus in die Bildgenerierungspipeline eingefügt wird. Kann er so integriert werden, daß die Berechnung des Beleuchtungsmodells nur für die sichtbaren Polygone durchgeführt wird und die Farbe anschließend als Attribut mitgeführt wird, so ist der Berechnungsaufwand linear abhängig von der Anzahl der sichtbaren Polygone.
Bewertung:
Die resultierende Darstellung der Körper ist nicht realistisch. Obwohl ein Tiefeneindruck entsteht, sind die Polygone deutlich voneinander unterscheidbar.
Neben diesem Nachteil bestehen Einschränkungen in der Auswahl des Beleuchtungsmodells. Durch die konstante Färbung eines Polygons ist die Verwendung von Modellen für die spiegelnde Reflexion nicht sinnvoll.
Anwendungsgebiet:
Entsprechen die Polygone jeweils nur wenigen Pixeln auf dem Bildschirm (sog. Mikropolynome), so kann diese Art der Beleuchtungsberechnung durchaus sinnvoll sein.

konvexe Hülle (3.1.2.2, S.105)

Die *konvexe Hülle* von Punkten ist die kleinstmögliche Menge von Punkten, bei der mit je zwei Punkten auch die Verbindungsstrecke in der Menge liegt:

$$co\{p_0,\cdots,p_n\} = \{p \,|\, p = \sum_{i=0}^{n} \lambda_i p_i,\ \sum_{i=0}^{n} \lambda_i = 1 \text{ und } \ \lambda_i \geq 0,\ i = 0,\cdots,n\}.$$

Beispiele:
Strecke mit den Begrenzungspunkten p_1 und p_2 :
$co\{p_1,p_2\} = \{p|p = \lambda \cdot p_1 + (1-\lambda)\cdot p_2, 0 \leq \lambda \leq 1\}$
Dreieck mit Eckpunkten p_0,p_1,p_2, wobei die p_0,p_1,p_2 drei verschiedene Punkte in der affinen Ebene sind.

Koordinatensystem des affinen Raumes (3.1.1.1, S.102)

Ist ein Koordinatensystem $(o;e_1,\ldots,e_n)$ gegeben, dann heißt für jeden Punkt $p \in A^n$ der Vektor $v = (\vec{op})$ *Ortsvektor* von p.
Die Komponenten von v bzgl. $(e_1,\ldots,e_n)$ heißen *Koordinaten* von p, d.h. p besitzt die Koordinaten $(x_1,\ldots,x_n)$ genau dann, wenn

$$(\vec{op}) = v = x_1e_1 + x_2e_2 + \cdots + x_ne_n.$$

Der Punkt o besitzt die Koordinaten $(0,\ldots,0)$ und heißt *Ursprung* des Koordinatensystems.
Die Punkte p_i mit $(\vec{op_i}) = e_i$ heißen *Einheitspunkte*.
Die Menge der Punkte $(o,p_1,\ldots,p_n)$ eines Koordinatensystems wird als *affine Basis* bezeichnet.

Lambert'scher Strahler (7.8.3.1, S.475)

Lambert'sche Strahler sind Oberflächen, bei denen die **Leuchtdichte** unabhängig von der Betrachtungsrichtung ist. Die Fläche erscheint daher aus

jeder Richtung gleich hell.
Die **Strahlstärke** eines Lambert'schen Strahlers ist proportional zum Cosinus des Winkels zwischen Flächennormaler und Ausstrahlungsrichtung (*Lambert'sches Cosinusgesetz*).

Leaf-Klasse; Leaf-Datenelement (Java3D), (6.2.7.2, S.334)
Leaf ist eine abstrakte Klasse. Leaf-Datenelemente stehen für die **visuellen Objekte** und Audio-Objekte einer Szene. In dem vorliegenden Dokument spielen insbesondere die beiden von Leaf abgeleiteten Klassen **Shape3D** und **Behavior** eine Rolle.

Leuchtdichte (7.8.3.1, S.474)
Die *Leuchtdichte* ist das photometrische Gegenstück zur radiometrischen Größe **Strahldichte.**

Lichtquelle (7.11.1, S.494)
Die wichtigsten Lichtquellen sind:

- **Umgebungslicht (ambientes Licht)**
- **diffuse Flächenlichtquelle**
- **Punktlichtquelle**
- **Richtungslichtquelle**
- **Goniometrische Lichtquelle**
- **Strahler**

Lichtstärke (7.8.2, S.472)
Die *Lichtstärke* I_v ist das photometrische Gegenstück zur **Strahlstärke** I_e. Die Bedeutung der Lichtstärke für die Photometrie wird dadurch deutlich, daß sie eine eigene SI-Basiseinheit bekommt, nämlich die **Candela** (cd).

Locale-Klasse (Java3D) (6.2.7.4, S.336)
Ein zur Locale-Klasse gehörendes Datenelement ist ein Punkt im **virtuellen Universum**. Dieser Punkt hat die Aufgabe einer Landmarke: Die Position der **Geometrien** ist relativ zum Locale-Punkt. Der Locale-Punkt liegt für diese Geometrien also im Ursprung. Locale ist für alle Geometrien der Ursprung, die ein Element des Teilbaums sind, dessen Wurzel das Locale-Datenelement ist.

lokale Beleuchtungsalgorithmen (7.11.3, S.499)
Im Kurs behandelt wurden die folgenden lokalen Beleuchtungsalgorithmen:

- **Konstante Beleuchtung**
- **Gouraud–Algorithmus**
- **Phong–Algorithmus**

Dabei wird davon ausgegangen, daß Objekte in triangulierter Form vorliegen.

Lokale Wirkung der de Boor-Punkte (4.6.2.1, S.187)

Für die B-Splines gilt $N_i^n(u) = 0$, falls $u \notin [t_i, t_{i+n+1}]$. Daher beeinflußt der i-te de Boor-Punkt d_i die Kurve nur über dem Parameterbereich $[t_i, t_{i+n+1}]$. Die Form der Kurve wird über dem Intervall $[t_i, t_{i+n+1}]$ nur von den de Boor-Punkten $d_{i-n}, \cdots, d_{i+n}$ beeinflußt.

lokales (empirisches) Beleuchtungsmodell (7.11.2, S.496)

Lokale Beleuchtungsmodelle beschreiben die in einem Punkt der Objektoberfläche wahrnehmbare **Leuchtdichte**. Dabei wird das Zusammenwirken von einfallendem Licht aus den Lichtquellen und dem Reflexionsverhalten der Objektflächen ausgewertet.
Sekundäre Effekte, wie der Strahlungsaustausch benachbarter Flächen, werden nicht berücksichtigt.
Lokale Beleuchtungsmodelle sollen mit minimalem Aufwand möglichst realitätsnahe Flächengraphik erzeugen. Sie basieren deshalb auch nicht auf strengen physikalischen Gesetzmäßigkeiten.
Neben dem Umgebungslicht sind noch drei Reflexionsarten zu berücksichtigen: **ideal diffus, ideal spiegelnd und gerichtet diffus.**
Im Falle durchsichtiger Objekte kommt unter Umständen noch ein transparenter Anteil hinzu.

Bestimmung der Leuchtdichte für

- **ambientes Licht:**

$$L_{amb} = r_a \cdot \frac{E_a}{\pi} = r_a \cdot L.$$

- **ideal diffus reflektiertes Licht:**
  Die Leuchtdichte hängt vom Einfallswinkel ϑ ab, ist aber unabhängig vom Betrachtungswinkel:

$$L_{diff} = \begin{cases} r_d \cdot L \cdot cos\vartheta, & |\vartheta| < 90° \\ 0 \text{ sonst} \end{cases}$$

$$= \begin{cases} r_d \cdot L \cdot (\vec{N} \cdot \vec{V}_L), \text{ falls } (\vec{N} \cdot \vec{V}_L) > 0 \\ 0 \text{ sonst} \end{cases}$$

- **Gerichtet diffus reflektiertes Licht (Phong-Modell):**
  Phong-Modell:

$$L_{spec} = \begin{cases} r_s \cdot L \cdot \cos^m \gamma, \text{ falls } |\gamma| < 90° \\ 0 \text{ sonst} \end{cases}$$

$$= \begin{cases} r_s \cdot L \cdot (\vec{R} \cdot \vec{E})^m, \text{ falls } (\vec{R} \cdot \vec{E}) > 0 \\ 0 \text{ sonst} \end{cases}$$

  Phong-Modell mit Vermeidung der Berechnung von $\vec{R}$:

$$L'_{spec} = \begin{cases} r_s \cdot L \cdot (\vec{H} \cdot \vec{N})^m, \text{ falls } (\vec{H} \cdot \vec{N}) > 0 \\ 0 \text{ sonst} \end{cases}.$$

Machband–Effekt (7.11.3.2, S.503)

Als Machband–Effekt werden helle oder dunkle Linien bezeichnet, die an Stellen auftreten, an denen sich die Leuchtdichten schnell ändern bzw. diskontinuierlich sind.

Maske (mask) (OpenGL) (6.3.3.1, S.377)

„Zeichenmuster, das zur Auswahl oder zum Ausblenden von Teilen eines **Bildes** dient" ([Cla97], S. 344).
Häufig bestehen Masken aus rechteckigen **Bitmaps**. Eine solche Bitmap wird quasi über einen bestimmten Bereich des **gerasterten** Bildes gelegt. So können beispielsweise die Werte der Bitmap und die Farbwerte oder **Farbindizes** des Bildes mit Operationen verknüpft werden, dazu ein einfaches Beispiel: Alle Bildpunkte (**Fragmente** oder **Pixel)** mit einem Farbtabellenindex, bei dem das höchste Bit 0 ist, wird der Farbtabellenindex für schwarz zugewiesen.

Material-Klasse (Java3D) (6.2.14.1, S.371)

Damit werden bestimmte Eigenschaften von Oberflächen von **geometrischen Objekten** definiert. Eine dieser Eigenschaften ist die Art, wie ambientes, diffuses, punktförmiges und paralleles Licht reflektiert wird und ob die Oberfläche Licht emitiert (z.B. wie bei einer glühenden Herdplatte).

Matrizendarstellung für projektive Abbildungen (3.2.3.1, S.112)

Ist eine projektive Abbildung $f\colon P(\mathbb{R}^4) \to P(\mathbb{R}^4)$ gegeben, so betrachten wir die zugehörige lineare bijektive Abbildung $F\colon \mathbb{R}^4 \to \mathbb{R}^4$. Diese Abbildung läßt sich als 4×4-Matrix A beschreiben, die bis auf einen Skalar $\lambda \neq 0$ festgelegt ist. Solche Matrizen heißen *homogen.*
Wird f durch die Matrix

$$A = \begin{pmatrix} a_{00} & \cdots & a_{03} \\ \vdots & & \vdots \\ a_{30} & \cdots & a_{33} \end{pmatrix}$$

dargestellt, so bildet f den Punkt p mit den homogenen Koordinaten $[x,y,z,w]$ auf den Punkt

$$[x,y,z,w]\cdot A = [a_{00}x+\cdots+a_{30}w,\cdots,a_{03}x+\cdots+a_{33}w]$$

ab.

Modellbildung beim Solid Modelling, (7.1.1, S.449)

In einem zweistufigen Prozeß bildet man zunächst das *mathematische Modell*, dann das *symbolische Modell.*
Das mathematische Modell ist eine Idealisierung des realen Objektes und beinhaltet in der Regel nur einen Teil der Eigenschaften, die das reale Objekt auszeichnen.
Es wird in ein *symbolisches Modell* im sogenannten *Repräsentationsraum* überführt.
Der *Repräsentationsraum* ist abhängig vom *Modellierer* und umfaßt die

Menge derjenigen Objekte, die mit dem Modellierer konstruiert werden können.

Modellierung (modelling) (6.1, S.323)

„Alle Aktivitäten zum Entwurf einer Szene. Dies umfasst die Festlegung der Geometrie, der Materialien, der Lichtverhältnisse und der Blickverhältnisse. Wird in eingeschränktem Sinn auch oft nur für die geometrische Modellierung verwendet." ([Cla97], S.345)

***n*–mal stetig differenzierbare Kurve (4.1.1.1, S.169)**

Eine Kurve heißt *n–mal stetig differenzierbar*, wenn die zugehörige Abbildung q mindestens n-mal stetig differenzierbar (kurz: $C^n - stetig$) ist.

nach der Bogenlänge parametrisiert (4.1.1.1, S.171)

Entspricht für $u \in [a,b]$ der Wert der Bogenlänge $s(u)$ der zugrundeliegenden Intervallänge $u-a$, so nennt man q *nach der Bogenlänge parametrisiert.*

Das ist äquivalent zu der Bedingung $\|q'(u)\| = 1, \quad u \in [a,b]$.

Im allgemeinen läßt sich jede reguläre Kurve nach der Bogenlänge parametrisieren.

Normalenvektor einer Fläche (4.7.1.1, S.194)

Der Vektor

$$n(u,v) := \frac{q_u(u,v) \times q_v(u,v)}{\|q_u(u,v) \times q_v(u,v)\|}$$

heißt *Normalenvektor* im Punkt $q(u,v)$.

Er steht senkrecht auf der Tangentialebene und ist unabhängig von der speziellen Wahl der Parametrisierung.

Nusselts Analogon (Radiosity) (7.13.2.1, S.521)

Wilhelm Nusselt konstruierte eine Halbkugel mit Einheitsradius, in deren Zentrum sich das Flächenelement dA_i befindet. Für die Berechnung von Formfaktoren wird die Oberfläche der Halbkugel in kleine Segmente aufgeteilt. Für jedes dieser Segmente wird der Formfaktor zwischen dem Segment und der Fläche dA_i durch Parallelprojektion auf die Basis berechnet. Nachdem man diese sogenannten *Delta-Formfaktoren* berechnet hat, wird jede Fläche A_j der Szene auf die Halbkugel projiziert, wobei registriert wird, welche Segmente auf der Halbkugel von der Projektion der Fläche A_j bedeckt werden. Der gesamte Formfaktor der Fläche A_j ist dann einfach die Summe der Delta-Formfaktoren der von A_j bedeckten Segmente.

Das Nusseltsche Analogon hat seine Bedeutung darin, daß es die Grundlage zu dem ersten brauchbaren Verfahren zur Berechnung von Formfaktoren ist, nämlich dem **Hemi-Cube-Verfahren.**

Parallelprojektion (3.3.5.1, S.122)

Bei den Parallelprojektionen unterscheidet man zwischen rechtwinkligen und schiefwinkligen Projektionen.

Bei rechtwinkligen Projektionen steht die Projektionsrichtung senkrecht auf

der Projektionsebene, d.h. sie ist parallel zur Normalen der Projektionsebene.
Bei schiefwinkligen Projektionen steht die Projektionsrichtung nicht senkrecht auf der Projektionsebene und bildet mit der Normalen der Projektionsebene einen Winkel.
Bei Parallelprojektionen bleiben Längen auf Geraden parallel zur Projektionsebene erhalten.

Parametertransformation (4.1.1.1, S.170)
Zwei reguläre Kurven $q_1 : [a,b] \to \mathbb{R}^3$, $q_2 : [c,d] \to \mathbb{R}^3$ heißen *äquivalent*, wenn es eine bijektive differenzierbare Abbildung $\varphi : [a,b] \to [c,d]$ mit $\varphi'(u) > 0$ gibt, so daß $q_1 = q_2 \circ \varphi$ gilt.
Wir sagen in diesem Fall auch, daß q_2 durch φ *reparametrisiert* wurde und nennen φ einen richtungserhaltenden Parameterwechsel oder eine *Reparametrisierung* von q_1.

parametrische Stetigkeit (4.4.1, S.182)
Parametrisch stetiger Anschluß (C^n–stetiger Übergang):
Seien $q_1 : [a_1,b_1] \to \mathbb{R}^3$, $q_2 : [a_2,b_2] \to \mathbb{R}^3$ zwei n-mal stetig differenzierbare reguläre Kurven. q_1 und q_2 schließen an der Stelle b_1, a_2 C^n–stetig genau dann aneinander, wenn

$$q_1^{(k)}(b_1) = q_2^{(k)}(a_2) \quad \text{für alle } k = 0, \cdots, n.$$

Parametrisierte Flächen (4.7.1.1, S.193)
Sei $\emptyset \neq D \subset \mathbb{R}^2$ gegeben. Eine Abbildung $q : D \to \mathbb{R}^3$ heißt *parametrisierte Fläche*.

parametrisierte Kurve (4.1.1.1, S.169)
Eine Abbildung $q : \mathbb{R} \supset I = [a,b] \longrightarrow \mathbb{R}^3$ mit $a < b$ heißt *parametrisierte Kurve*.

perspektivische Projektion (3.3.5.4, S.129)
Perspektivische Projektionen sind keine affinen Abbildungen, da sie z.B. Längenverhältnisse nicht invariant lassen: Vom Blickpunkt weit entfernte Objekte werden kleiner dargestellt als Objekte mit kleinem Abstand zum Blickpunkt (Augpunkt, Beobachtungspunkt). Diese Abbildungen sind projektive Abbildungen und lassen sich daher nur im projektiven Raum als Matrizen beschreiben.

Beispiel:
Zweidimensionaler Fall: Perspektivische Projektion eines affinen Punktes $P = (x,y)$ mit Augpunkt $A = (-x_0, 0)$ und Projektionsgerade $x = 0$:
Es ergibt sich die folgende 3×3-Matrix zur Beschreibung dieser Abbildung.

$$[x', y', 1] = [x, y, 1] \begin{bmatrix} 0 & 0 & 1 \\ 0 & x_0 & 0 \\ 0 & 0 & x_0 \end{bmatrix} = [x, y, 1] \begin{bmatrix} 0 & 0 & \frac{1}{x_0} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Wie schon bei den Parallelprojektionen zerlegen wir die Projektion in zwei

Abbildungen

$$\begin{bmatrix} 0 & 0 & \frac{1}{x_0} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & \frac{1}{x_0} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \text{ bzw.}$$
$$P = T_p \cdot P_p.$$

Die Matrix P_p beschreibt die Projektion auf die Gerade $x = 0$. Die Matrix T_p beschreibt die *perspektivische Transformation.*

perspektivische Transformation im dreidimensionalen Raum (3.3.5.6, S.133)

$$[x,y,z,w] \begin{bmatrix} 1 & 0 & 0 & \frac{1}{x_0} \\ 0 & 1 & 0 & \frac{1}{y_0} \\ 0 & 0 & 1 & \frac{1}{z_0} \\ 0 & 0 & 0 & 1 \end{bmatrix} = [x,y,z, \frac{x}{x_0} + \frac{y}{y_0} + \frac{z}{z_0} + w]$$

perspektivische Transformation im zweidimensionalen Raum (3.3.5.5, S.130)

Einpunktperspektive:

Perspektivische Projektion eines affinen Punktes $P = (x,y)$ auf einen Bildpunkt auf der affinen Bildgeraden $x = 0$:

$$[x,y,w] \begin{bmatrix} 1 & 0 & \frac{1}{x_0} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = [x,y, \frac{x}{x_0} + w]$$

Zweipunktperspektive:

$$[x,y,w] \begin{bmatrix} 1 & 0 & \frac{1}{x_0} \\ 0 & 1 & \frac{1}{y_0} \\ 0 & 0 & 1 \end{bmatrix} = [x,y, \frac{x}{x_0} + \frac{y}{y_0} + w]$$

Pfad im Szenegraphen (Java3D) (6.2.7.7, S.339)

Pfad vom **Locale**-Datenelement oder von einem internen Knoten (ohne Blätter) im **Szenegraphen** zu einem Blatt des Szenegraphen.

Phong–Algorithmus (7.11.3.3, S.503)

Das *Phong-Modell* erfaßt auch spiegelnde Effekte. Dazu werden die Normalen benötigt. Deshalb werden nicht die Leuchtdichten, sondern die Normalenvektoren interpoliert. Diese Berechnung kann ebenfalls inkrementell sein und integriert in die Rasterung aufgenommen werden. Allerdings muß jeder Normalenvektor vor Auswertung des Beleuchtungsmodells normiert werden. Nach der Interpolation wird in jedem Punkt die Leuchtdichte berechnet.

Aufwand:

Es entsteht im günstigsten Fall ein in der Anzahl der sichtbaren Polygone

und in der Anzahl der Pixel sich additiv zusammensetzender linearer Aufwand. Die Konstanten sind hier wesentlich größer als bei dem Gouraud–Algorithmus.
Bewertung:
Die Resultate sind im allg. realistischer als bei **konstanter Beleuchtung** oder beim **Gouraud-Algorithmus**, die Mach–Band–Effekte sind reduziert.

Phong-Modell (7.8.3.5, S.479)

Im empirischen *Phong-Modell* wird der richtungsabhängige Anteil der spekularen Reflexion über den Reflexionsgrad berechnet als

$$r_s = r_{s,0} \cos^m \gamma.$$

$r_{s,0}$ ist eine Konstante zwischen 0 und 1. γ ist der Winkel zwischen der Richtung des ideal reflektierten Strahls und der Beobachtungsrichtung. Der Exponent m gibt an, wie schnell das Reflexionsvermögen mit größer werdendem Winkel γ abfällt (vgl. **gerichtet diffuse Reflexion (spekulare Reflexion)**.

Photometrie (7.8.2, S.472)

Im Gegensatz zur Radiometrie beschäftigt sich die Photometrie mit der Messung von elektromagnetischer Strahlung im sichtbaren Bereich von ca. 360 nm bis 830 nm. Der entscheidende Unterschied zur Radiometrie besteht darin, daß nicht nur physikalische Größen, sondern auch die physiologischen Eigenschaften des menschlichen Auges berücksichtigt werden.
Jeder strahlungstechnischen Größe wird eine photometrische Größe zugeordnet, wobei zur Unterscheidung der Index v (für visuell) verwendet wird. Außerdem erhalten die photometrischen Größen neue Einheiten.

photometrische Grundgrößen (7.8.2, S.472)

Durch Gewichtung der strahlungstechnischen Größen $X_{e\lambda}$ mit den Hellempfindlichkeitsfunktionen erhält man die entsprechenden photometrischen Größen $X_{v\lambda}$ und $X'_{v\lambda}$:

$$X_{v\lambda}(\lambda) = Km X_{e\lambda}(\lambda) V(\lambda)$$

und

$$X'_{v\lambda}(\lambda) = Km' X_{e\lambda}(\lambda) V'(\lambda).$$

Km bzw. Km' sind Proportionalitätsfaktoren.

Im Kurs werden die folgenden photometrischen Grundgrößen behandelt:

- Lichtmenge
- Lichtstrom
- **Lichtstärke**
- Spezifische Lichtausstrahlung
- **Beleuchtungsstärke**

- **Leuchtdichte**

Pixel (von picture element) (6.2.10.1, S.350)
Ein einzelnes Bildelement des Rasterausgabegeräts (Displays). Ein Pixel besteht aus je einem x- und y-Wert und drei Werten für die Farbe (bei **Echtfarbdarstellung**) bzw. einen **Farbtabellenindex**-Wert. Jedes Pixel ist einer x- und y-Koordinate zugeordnet. In OpenGL und insbesondere Java3D arbeiten Anwendungsprogramme typischerweise nicht mit Pixeln, sondern mit dreidimensionalen geometrischen Elementen, die schließlich **gerastert** werden. Das Ergebnis sind **Fragmente**. Fragmente können von den Anwendungsprogrammen mit Hilfe von OpenGL oder Java3D manipuliert werden. Anschließend werden die Fragmente in Pixel überführt, die dem **Framebuffer** übergeben werden.

Pixel-Selected-Raytracing (7.12, S.511)
Eine schnelle Previewing-Technik zur Beschleunigung der Visualisierung besteht darin, homogene Regionen in der Bildebene zu detektieren und zu interpolieren. Die Technik basiert auf einer *Divide-and-Conquer*-Strategie.

Pixelmap (pixelmap) (OpenGL) (6.3.3.1, S.377)
Ein rechteckiges Feld von **Pixeln**. Eines der 12 **Primitiven** von OpenGL.

Polynomraum (4.1.1.3, S.172)
Die Menge aller Polynome $\mathbb{P}^n[a,b]$ vom Grad n bildet einen Unterraum von $C[a,b]$ der Dimension $n+1$. Die Monome $1,t,t^2\cdots t^n$ bilden eine Basis, die sogenannte *Taylorbasis (Monombasis)* des Polynomraums. In dieser Basis ist ein reelles Polynom p durch

$$p(t) = c_n t^n + c_{n-1} t^{n-1} + \cdots + c_1 t + c_0$$

mit Taylorkoeffizienten $c_n, \cdots, c_0 \in \mathbb{R}$ gegeben.
Wählt man statt reeller Koeffizienten c_i Vektoren aus dem $\mathbb{R}^3$, so erhält man Polynomkurven im $\mathbb{R}^3$.

Nachteil dieser Darstellung für den interaktiven Entwurf:
Die Taylorkoeffizienten entziehen sich einer unmittelbaren geometrischen Deutung.

Primitiv (OpenGL) (6.3.4.8, S.385)
Bei OpenGL ist Primitiv (das grafische/geometrische/visuelle Primitiv) die Bezeichnung für die einfachsten **geometrischen Objekte**. In OpenGL stehen 12 Primitive zur Verfügung: Punkte, Strecken, Streckenzüge, Dreiecke, Vierecke, Streifen aus Dreiecken, Streifen aus Vierecken, Fächer aus Dreiecken, geschlossene Streckenzüge, konvexe Polygone sowie **Bitmaps** und **Pixelmaps**.

Progressive Refinement (Radiosity) (7.13.2, S.519)
Das Progressive-Refinement-Verfahren beruht darauf, daß man die Ausbreitung des Lichts durch die Szene ausgehend von den primären Lichtquellen

verfolgt. Die Radiosity, die eine Fläche A_j an eine Fläche A_i weitergibt, ist dabei gegeben als

$$B_{j\to i} = \rho_i B_j F_{ij} = \rho_i B_j F_{ji} \frac{A_j}{A_i}.$$

Zu Beginn des Progressive-Refinement-Verfahrens werden die Radiosities B_i mit den Lichtquellentermen E_i initialisiert. Jeder Iterationsschritt besteht darin, die sogenannte unverteilte Radiosity einer Fläche A_j der Szene auf alle anderen zu verteilen.

Vorteile des Verfahrens:

- schnelle Konvergenz,
- zur Durchführung eines Iterationsschrittes genügt die Kenntnis einer Zeile der Formfaktormatrix,
- nach wenigen Iterationsschritten werden im allg. schon sehr gute Ergebnisse erzielt.

Projektion (6.1, S.322)

Die Überführung von dreidimensionalen virtuellen Objekten auf die zweidimensionale **Bildebene**. In der Regel werden zusätzlich die Tiefenwerte der Eckpunkte der Objekte gespeichert.

projektive Abbildung (3.2.3, S.112)

Eine Abbildung f zwischen zwei projektiven Räumen $P(V)$ und $P(W)$ heißt *projektiv,* wenn es eine injektive lineare Abbildung $F : V \to W$ gibt mit $f(\mathbb{R}\cdot v) = \mathbb{R}\cdot F(v)$ für jedes vom Nullvektor verschiedene $v \in V$.
Man schreibt dafür kurz $f = P(F)$.
Zwei lineare Abbildungen $F, F' : V \to W$ definieren dieselbe projektive Abbildung genau dann, wenn es ein $\lambda \neq 0$ gibt mit $F' = \lambda \cdot F$.
Wie bei linearen und affinen Abbildungen sind auch projektive Abbildungen durch ihre Wirkung auf die Basis eines projektiven Raumes festgelegt. Da der reelle projektive dreidimensionale Raum $P(\mathbb{R}^4)$ fünf Basiselemente besitzt, ist eine projektive Abbildung vom $P(\mathbb{R}^4)$ in den $P(\mathbb{R}^4)$ durch die Abbildung von fünf geeigneten Punkten eindeutig festgelegt.

projektive Basis (3.2.2.3, S.110)

Ein $(r+1)$-Tupel $(p_0, \cdots, p_r)$ im projektiven Raum $P(\mathbb{R}^4)$ heißt *projektiv unabhängig,* wenn es linear unabhängige Vektoren $v_0, \cdots, v_r \in \mathbb{R}^4$ gibt mit $p_i = \mathbb{R}\cdot v_i,\ i = 0, \cdots, r$.
Ein Fünftupel $(p_0, \cdots, p_4)$ von Punkten aus $P(\mathbb{R}^4)$ heißt *projektive Basis* von $P(\mathbb{R}^4)$, wenn je vier davon projektiv unabhängig sind.
Im dreidimensionalen projektiven Raum $P(\mathbb{R}^4)$ ist eine *projektive Basis* durch die von den Vektoren

$$v_0 = [1,0,0,0],\ v_1 = [0,1,0,0],\ v_2 = [0,0,1,0],\ v_3 = [0,0,0,1],$$

$$v_4 = [1,1,1,1],$$

definierten Punkte $p_0, \cdots, p_4$ gegeben, also durch fünf projektive Punkte festgelegt.

projektiver Raum (3.2.2, S.108)

Ist ein *reeller affiner Raum A* gegeben, so nennen wir die Menge aller Geraden durch den Ursprung den *reellen projektiven Raum* $P(A)$.
Die Dimension $dim\,P(A)$ des projektiven Raumes $P(A)$ wird $dim\,P(A) = dim(A) - 1$ gesetzt.
Ein projektiver Raum der Dimension eins bzw. zwei heißt *projektive Gerade* bzw. *projektive Ebene.*
Projektive lineare Teilräume des projektiven Raumes $P(A)$ werden ausgehend von einem linearen Teilraum des zu A korrespondierenden Vektorraumes von A analog definiert.

Punktlichtquelle (7.11.1.3, S.495)

Die *Punktlichtquelle* wird meist als in alle Richtungen gleichmäßig sendend (isotrop) angenommen. Neben der Lage im Raum wird sie durch die Lichtstärke $I(\lambda)$ in Candela gekennzeichnet. Im Abstand R erzielt diese Lichtquelle eine **Beleuchtungsstärke**

$$E = \frac{I}{R^2} \cdot cos\vartheta,$$

wobei ϑ der Lichteinfallswinkel (zwischen Flächennormaler und Lichtausbreitungsrichtung) ist. Ist die Punktlichtquelle weit entfernt, so verhält sie sich wie eine Richtungslichtquelle.

Radiometrie (7.8.1, S.468)

Die *Radiometrie* beschäftigt sich mit dem ganzen elektromagnetischen Spektrum. Alle strahlungsphysikalischen Größen bekommen zur Unterscheidung von den entsprechenden **photometrischen Größen** den Index e (für energetisch).

Radiosity-Verfahren (7.13, S.512)

Das Radiosity-Verfahren ist ein globales Beleuchtungsverfahren für ideal diffus reflektierende Oberflächen. Es werden nicht nur die Wechselwirkung der Oberflächen mit den Lichtquellen, sondern auch die Wechselwirkung der Oberflächen untereinander berücksichtigt.
Berechnet wird im Radiosity-Verfahren die Beleuchtungsstärke (gemessen in Lux) einer jeden Fläche.
Anwendungsgebiet:
Lichtverteilung in Gebäuden.
Vorteile:
Betrachtungsunabhängigkeit: Bei diffuser Reflexion ist die Leuchtdichte einer Fläche unabhängig von der Beobachtungsrichtung. Wurde eine Szene mit dem Radiosity-Verfahren berechnet, so ist es möglich, die Szene aus allen Richtungen zu betrachten, ja sogar durch die Szene hindurchzugehen, ohne sie neu berechnen zu müssen.
Vereinfachungen:
Beim Radiosity-Verfahren werden nur die rein diffusen Reflexionen berücksichtigt. Die Oberflächen der Objekte werden durch eine endliche Zahl kleiner, ebener Flächen, sogenannter Patches, beschrieben. Jedes Patch hat

einen einheitlichen Reflexionsgrad und eine einheitliche Radiosity. Die Radiosity einer Fläche j wird dann beschrieben durch:

$$B_j = E_j + \rho_j H$$

mit

$$H = \sum_{i=1}^{N} F_{ij} B_i \frac{A_i}{A_j},$$

wobei N die Anzahl der Flächen in der Szene ist.
Der **Formfaktor** F_{ij} beschreibt den Bruchteil der auf Fläche j ankommenden Strahlungsleistung der Gesamtausstrahlung der Fläche i. In den Formfaktoren ist die gesamte geometrische Information der Szene enthalten.

Aus den beiden obigen Gleichungen erhält man durch Einsetzen die Grundgleichung des Radiosity-Verfahrens, ein lineares Gleichungssystem:

$$B_j = E_j + \rho_j \sum_{i=1}^{N} F_{ij} B_i \frac{A_i}{A_j}.$$

Vor der Lösung des Gleichungssystems müssen die Formfaktoren F_{ij} bestimmt werden, was ungleich aufwendiger ist als die Lösung des Gleichungssystems.

Verfahren zur Lösung des Gleichungssystems:

- Eliminationsverfahren von Gauß
- iteratives Gauß-Seidel-Verfahren
- iteratives **Progressive Refinement**-Verfahren

Farben im Radiosity-Verfahren:
Wenn man die Wellenlängenabhängigkeit der Größen in der Radiosity-Gleichung beachtet, erhält man:

$$B_j(\lambda) = E_j(\lambda) + \rho_j(\lambda) \sum_{i=1}^{N} F_{ji} B_i(\lambda) \qquad \text{für alle Wellenlängen } \lambda.$$

Es ist sinnvoll, die Radiosity-Gleichung für aneinandergrenzende Wellenlängenbänder $\Delta\lambda$ zu lösen. In der GDV wird aber für gewöhnlich die Radiosity-Gleichung nur für die Grundfarben R, G und B eines RGB-Monitors gelöst. Das Ergebnis einer Radiosity-Rechnung liefert einen Radiosity-Wert für jedes Patch der Szene. Durch Skalierung kann dieser Wert in eine darstellbare Farbe umgerechnet werden.

Randrepräsentationen (Boundary Representation) (7.3, S.454)

Ein Körpermodell kann eindeutig durch seine Oberfläche und eine zugehörige topologische Orientierung beschrieben werden. Wegen der topologischen Orientierung ist für jeden Punkt der Oberfläche eindeutig festgelegt, auf welcher Seite das Innere des Objektes liegt.
Boundary Representations (BReps) benutzen diese Tatsache und beschreiben 3D-Objekte durch ihre Oberfläche.

Rastergerät (raster display processor) (2.4, S.40)

Bildrechner zur Erzeugung und Darstellung graphischer Objekte

Neben der zentralen CPU, in der die Bilddefinition erzeugt wird, und dem zentralen Speicher enthält ein Rastergerät die folgenden Hauptkomponenten:

- Die Bilderzeugungskomponente (display processor) erzeugt anhand der Bilddefinition die gerasterte Darstellung des Bildes im Bildspeicher.
- Die Bilddarstellungskomponente (Bilddarstellung) erzeugt durch zeilenweises Schreiben auf dem Videomonitor ein stehendes, flimmerfreies Bild.
- Der Bildspeicher ist das Bindeglied zwischen Bilderzeugungs- und Bilddarstellungskomponente.

Für sämtliche Komponenten gibt es eine Vielzahl unterschiedlicher Realisierungs- und Implementierungsmöglichkeiten.

Rastern; Rasterung; Rasterisierung (6.3.3.2, S.379)

Die Überführung der (in der Regel auf die Bildebene projizierten) **virtuellen Objekte** in **Pixel** oder **Fragmente**. Fragmente sind die Vorstufe von Pixeln. In der Regel werden im z-Buffer die Tiefenwerte der Fragmente abgelegt. „In OpenGL wird die Rasterisierung auf alle grafischen **Primitive**, also auch auf **Bitmaps** und **Pixelmaps**, angewendet." ([Cla97], S. 345)

Raumunterteilung beim Raytracing (7.12, S.509)

In einem Vorverarbeitungsschritt wird der dreidimensionale Objektraum in nichtüberlappende Unterräume unterteilt (sog. Voxel = Volumenelement). Bei der Strahlverfolgung bestimmt dann ein Traversierungsalgorithmus der Reihe nach die vom Strahl durchlaufenen Unterräume. Schnittpunktberechnungen müssen dann nur noch mit denjenigen Objekten durchgeführt werden, die zumindest teilweise in diesen Unterräumen liegen. Wurde ein Schnittpunkt gefunden und ist dieser Schnittpunkt der nächste im aktuellen Voxel, dann kann die Traversierung stoppen.

Raumunterteilung mit regulären Gittern (Raytracing) (7.12, S.510)

Das regelmäßige 3D-Gitter teilt den Objektraum in endlich viele, gleich große, achsenparallele Voxel. Es eignet sich sehr gut für kleine und mittelgroße Szenen mit nicht allzu ungleichmäßig verteilten Szenenprimitiven. Das 3D-Gitter ist leicht zu implementieren und kann mit sehr geringem Rechenaufwand von einem Strahl durchlaufen werden (etwa mit dem Strahltraversierungsalgorithmus von Amanatides und Woo) .

Raumwinkel (7.8.1, S.468)

Den *Raumwinkel* Ω, unter dem eine Fläche A von einem Punkt O aus erscheint, ist gegeben durch:

$$\Omega = \int_A \frac{\cos\alpha}{R^2}\, dA.$$

Er hat die Einheit *sr* (steradiant).
Der volle Raumwinkel ist per Definition die Fläche einer Einheitskugel, hat also die Größe $4\pi sr$.

Raytracing (Strahlverfolgung) (7.12, S.507)

Raytracing simuliert den Prozeß der Lichtausbreitung und arbeitet dabei nach den Gesetzen für ideale Spiegelung und Brechung. Daher ist Raytracing vor allem für Szenen mit hohem spiegelnden und transparenten Flächenanteil gut geeignet.
Die Grundidee besteht darin, Lichtstrahlen auf ihrem Weg von der Quelle bis zum Auge zu verfolgen. Zur Vereinfachung werden beim konventionellen Raytracing nur ideal reflektierte und ideal gebrochene Strahlen weiterverfolgt. Da nur wenige Strahlen das Auge erreichen, kehrt man das Verfahren um (Reziprozität der Reflexion) und sendet durch jedes Pixel des Bildschirms einen vom Augpunkt ausgehenden Strahl in die Szene. Trifft der Sehstrahl auf ein Objekt, so wird das **lokale Beleuchtungsmodell** berechnet. Dann werden zwei neue Strahlen erzeugt, nämlich der reflektierte und der gebrochene (transmittierte) Sehstrahl. Der Leuchtdichtebeitrag dieser beiden Strahlen wird rekursiv berechnet.
Dieser Prozeß bricht ab, wenn eine Lichtquelle getroffen wird, die auf dem Strahl transportierte Energie zu gering wird oder wenn der Sehstrahl die Szene verläßt. Aus praktischen Erwägungen wird man eine Obergrenze für die Rekursionstiefe festlegen. Mit Hilfe dieses Algorithmus wird das Verdeckungsproblem implizit gelöst. Die Schattenberechnung wird durchgeführt, indem man von den Auftreffpunkten des Sehstrahls sogenannte Schattenstrahlen zu den Lichtquellen der Szene sendet. Nur wenn kein undurchsichtiges Objekt zwischen einer Lichtquelle und dem Auftreffpunkt liegt, trägt sie zur Beleuchtung bei.
Zum **Beleuchtungsmodell nach Phong** (vgl. GDV II, Abschnitt 2.5.3 'Lokale Beleuchtungsalgorithmen'), das den lokalen Anteil modelliert, kommen der ideal spiegelnde und der transmittierte Anteil hinzu:

$$L_{ges} = L_{Phong} + r_r L_r + r_t L_t,$$

wobei

L_r	die Leuchtdichte auf dem reflektierten Strahl,
L_t	die Leuchtdichte auf dem transmittierten Strahl,
r_r	der Reflexionsgrad für die Idealreflexion,
r_t	der Reflexionsgrad für die Idealtransmission ist.

rechtwinklige Projektion (3.3.5.2, S.122)

Parallelprojektion entlang der z-Achse auf die Ebene $z = z_0$:

$$[x', y', z', w'] = [x, y, z, w] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & z_0 & 1 \end{bmatrix}.$$

In den meisten Fällen wird man für x_0, y_0 oder z_0 den Wert Null wählen.

Transformation:
Die Lage des Bildschirmkoordinatensystems wird meist so gewählt, daß sein Ursprung mit dem Ursprung des 3D-Koordinatensystems zusammenfällt und die z-Richtung die Oben-Richtung (ViewUp) definiert. Die Projektion der Oben–Richtung auf die Projektionsebene definiert y'.
Seien $p = (p_x, p_y, p_z)$ und $o = (o_x, o_y, o_z)$ die normierte Projektions- bzw. Obenrichtung. Dann läßt sich das rechtshändige Bildschirmkoordinatensystem x', y', z' wie folgt berechnen:

1) $e_{z'} = -p$.
2) Da $e_{x'}$ zu der von p und o aufgespannten Ebene senkrecht ist, gilt
$$e_{x'} = \frac{o \times e'_z}{||o \times e'_z||}.$$
3) $e_{y'} = e_{z'} \times e_{x'}$.

Dann wird die zu projizierende Szene in diesem Koordinatensystem dargestellt (vgl. Abschnitt 3.3.4). Anschließend wird die obige Parallelprojektion entlang der z'–Achse durchgeführt.

Referenz (6.2.7.6, S.339)
Ein Szenegraph besteht aus **Knoten** (**Datenelementen**) und **Kanten** und enthält in der Regel zusätzlich Referenzen. Eine Referenz ist keine Kante und wird als gestrichelter Pfeil dargestellt. Eine Referenz kann nur von einem Leaf- zu einem NodeComponent-Datenelement führen. Zu einem NodeComponent führt *mindestens* eine Referenz.

Reflexion (7.8.3.2, S.475)
Die *Reflexion* von Strahlung wird durch den *spektralen Reflexionsfaktor* ρ beschrieben, der das Verhältnis von reflektierter **Strahldichte** zur einfallenden **Bestrahlungsstärke** E angibt:
$$\rho_\lambda(\lambda, \varphi_r, \vartheta_r, \varphi_i, \vartheta_i) = \frac{L_{\lambda,r}(\lambda, \varphi_r, \vartheta_r)}{E_{\lambda,i}(\lambda, \varphi_i, \vartheta_i)} = \frac{L_{\lambda,r}(\lambda, \varphi_r, \vartheta_r)}{\int L_{\lambda,i}(\lambda, \varphi_i, \vartheta_i)\, cos\vartheta_i\, d\Omega_i}.$$
Der Index i kennzeichnet die Größen der einfallenden, r die Größen der reflektierten Strahlung.
Im allgemeinen werden drei Arten der Reflexion unterschieden:

- **ideal diffuse Reflexion**
- **ideal spiegelnde Reflexion**
- **gerichtet diffuse Reflexion** (spekulare Reflexion)

Reflexionsgrad (4, S.476)
Der *Reflexionsgrad* r gibt das Verhältnis von reflektierter zu einfallender **Bestrahlungsstärke** an und ist deshalb dimensionslos:
$$r_\lambda = \frac{E_{\lambda,r}}{E_{\lambda,i}}, \quad 0 \leq r_\lambda \leq 1.$$

Er wird in der GDV, besonders bei den empirischen Beleuchtungsmodellen, statt des Reflexionsfaktors ρ verwendet (s. Stichwort **Reflexion**).

Regionenorientierter Visibilitäts-Algorithmus (5.3.6.4, S.285)

Regionenorientierte Visibilitätsverfahren zählen zu den linienorientierten Visibilitätsverfahren. In einem ersten Schritt werden die Ränder potentiell sichtbarer Gebiete (Regionen) ermittelt. Anschließend wird die Tatsache ausgenutzt, daß sich die Visibilität eines Objekts nur auf Regionenrändern ändern kann, es also ausreicht, die Visibilitätsverhältnisse auf den Rändern zu verfolgen.

reguläre Fläche (4.7.1.1, S.193)

Eine Fläche heißt *regulär*, wenn q einmal stetig differenzierbar ist und

$$q_u(u,v) := \frac{\partial q(u,v)}{\partial u},\ q_v(u,v) := \frac{\partial q(u,v)}{\partial v} \text{ für alle } (u,v) \in D$$

linear unabhängig sind.
Die Vektoren $q_u(u,v)$ und $q_v(u,v)$ heißen *u-Tangente* bzw. *v-Tangente* von der Stelle (u,v).

reguläre Kurve (4.1.1.1, S.169)

Eine Kurve heißt *regulär*, wenn die zugehörige Abbildung q einmal stetig differenzierbar ist und $q'(u) \neq 0$ für alle $u \in [a,b]$ gilt.
Der *Gradient* $q'(u)$ von q an der Stelle u ist ein Vektor im $\mathbb{R}^3$, der die Tangentenrichtung der Kurve an der Stelle u angibt.

rendern; Rendering (6.3.3.2, S.380)

Allgemein: Die „Zusammenfassung einer Klasse von Darstellungsalgorithmen, die auf lokalen Beleuchtungsmodellen und interpolierenden Schattierungsalgorithmen beruhen.” [Cla97], S. 346.
Bei Java3D: Der Vorgang beim Überführen des **Szenegraphen** in ein ge**rastertes Bild**.

RGB–Modell (7.10.1.1, S.487)

Beim *RGB–Modell* werden die darstellbaren Farben als Punkte eines Einheitswürfels im Ursprung des kartesischen Koordinatensystems beschrieben. Auf den positiven Halbachsen liegen die Primärfarben Rot, Grün und Blau. Grauwerte, darstellbar durch gleichgroße Anteile von R, G und B, liegen auf der Hauptdiagonalen des Einheitswürfels mit Schwarz im Ursprung $[0,0,0]$ und Weiß im Punkt $[1,1,1]$. Eine Farbe wird dann durch Anteile von R, G und B beschrieben, die zu Schwarz addiert werden müssen.
Farbrastersichtgeräte haben R, G, B–Leuchtstoffe, die über individuelle Kathoden angeregt werden, woraus sich die besondere Bedeutung des RGB-Modells ergibt: Alle anderen Farbbeschreibungen müssen vor der Farbausgabe in den äquivalenten Punkt des RGB–Würfels umgerechnet werden.

Richtungslichtquelle (7.11.1.4, S.496)

Die *Richtungslichtquelle* ist durch die Lichtausbreitungsrichtung $\vec{V}_L$ und die spezifische Lichtausstrahlung $M(\lambda)$ gegeben. Alle Lichtstrahlen treffen mit

derselben Richtung $\vec{V}_L$ in der Szene auf, wodurch eine weit entfernte Lichtquelle, wie z.B. die Sonne, modelliert werden kann.

Rotationsmatrizen (3.3.2.3, S.117)

Rotation um die Koordinatenachsen:

1. Rotation um die z-Achse mit dem Winkel α :

$$[x',y',z',w'] = [x,y,z,w] \begin{bmatrix} \cos\alpha & \sin\alpha & 0 & 0 \\ -\sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

2. Rotation um die x-Achse mit dem Winkel α :

$$[x',y',z',w'] = [x,y,z,w] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha & 0 \\ 0 & -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

3. Rotation um die y-Achse mit dem Winkel α :

$$[x',y',z',w'] = [x,y,z,w] \begin{bmatrix} \cos\alpha & 0 & -\sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ \sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Die Drehwinkel sind in dem Rechtssystem x,y,z immer positiv.

Rotation um eine beliebige Achse durch den Ursprung:

Eine Drehung um eine beliebige Achse in Richtung des *normierten* Vektors (x,y,z) mit dem Winkel α kann auf Drehungen um die Koordinatenachsen zurückgeführt werden. Verknüpft man diese Drehungen, d.h. multipliziert die zugehörigen Rotationsmatrizen, so ergibt sich folgende homogene Darstellung:

$$\begin{bmatrix} tx^2+c & txy+sz & txz-sy & 0 \\ txy-sz & ty^2+c & tyz+sx & 0 \\ txz+sy & tyz-sx & tz^2+c & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

wobei $s = \sin\alpha$, $c = \cos\alpha$, $t = 1-\cos\alpha$.

Für kleine Winkel α ($\alpha < 1°$) kann man die Bogenlängen $\sin\alpha$ durch α und $\cos\alpha$ durch 1 approximieren:

$$[x',y',z',w'] = [x,y,z,w] \begin{bmatrix} 1 & z\alpha & -y\alpha & 0 \\ -z\alpha & 1 & x\alpha & 0 \\ y\alpha & -x\alpha & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Rotation um eine Achse, die nicht durch den Ursprung verläuft:

Man verschiebe das Rotationszentrum in den Ursprung, führe die Rotation durch und verschiebe das Rotationszentrum zurück:

$$p' = p \cdot T^{-1} \cdot R \cdot T.$$

Schatten-Caches (Raytracing) (7.12, S.510)

Bei der Verfolgung von Schattenstrahlen beim Raytracing genügt es zu wissen, daß mindestens ein Schnittpunkt des Strahls mit einem undurchsichtigen Objekt existiert. Aus diesem Grund verwaltet jede Lichtquelle einen Schatten-Cache, der das Resultat der Strahlverfolgung des letzten Schattenstrahls speichert. War dieser Schattenstrahl auf seinem Weg von der Lichtquelle zum Objektschnittpunkt auf ein undurchsichtiges Objekt gestoßen, so wird ein Pointer auf dieses schattenwerfende Objekt im Schatten-Cache gespeichert. Ansonsten wird ein Null-Pointer gespeichert. Bevor der nächste Schattenstrahl verfolgt wird, führt man einen Schnittpunkttest mit dem durch den Schatten-Cache referenzierten Objekt durch. Nur wenn dieses Schnittestergebnis negativ ist, wird der Schattenstrahl explizit durch die Szene verfolgt.

scheduling bound (Java3D) (6.2.11.4, S.363)

Wird eine 'Bound' (Grenze) im Zusammenhang mit **Interaktionen**, **Animationen** oder bestimmten anderen Effekten benutzt, spricht man von einer scheduling bound.

Scherungsmatrix (3.3.2.2, S.117)

Bei einer Scherung SH ergibt sich für die affinen Basisvektoren folgende Beziehung:

$$\begin{aligned} SH((1,0,0)) &= (1,s_1,s_3) \\ SH((0,1,0)) &= (s_2,1,s_4) \\ SH((0,0,1)) &= (s_5,s_6,1) \end{aligned} .$$

Man erhält als Scherungsmatrix

$$[x',y',z',w'] = [x,y,z,w] \begin{bmatrix} 1 & s_1 & s_3 & 0 \\ s_2 & 1 & s_4 & 0 \\ s_5 & s_6 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

schiefwinklige Projektion (3.3.5.3, S.127)

Bei den schiefwinkligen Projektionen ist die Projektionsebene meistens parallel zu zwei Koordinatenachsen, die Projektionsrichtung jedoch nicht senkrecht zu der Projektionsebene.

Bei der *Kavalierprojektion* bildet die Projektionsrichtung mit der Normalen der Projektionsebene einen Winkel von 45°.

Daher bleiben die Längen auf Geraden senkrecht zur Projektionsebene bei dieser Projektion unverändert.

Bei der *Kabinettprojektion* werden Längen auf Geraden senkrecht zur Projektionsebene um den Faktor $\frac{1}{2}$ verkürzt. Der Winkel β zwischen Projektionsebene und Projektionsrichtung ist daher $\arctan(2) = 63°$.

Abhängig von α und β ergibt sich die normierte Projektionsrichtung

$$p = (-\cos\alpha \cdot \cos\beta, -\sin\alpha \cdot \cos\beta, -\sin\beta).$$

Stimmt die Projektionsebene mit der x,y-Ebene überein (vgl. Bild 3.22), so lassen sich die Matrizendarstellungen der Kavalier- und Kabinettprojektion

wie folgt bestimmen:
Wie bei den rechtwinkligen Projektionen wird nach einer Transformation eine Parallelprojektion entlang der z'-Achse durchgeführt.
Bei der Transformation werden die Einheitsvektoren e_x und e_y auf sich selbst (die Punkte in der Projektionsebene bleiben fest) und die normierte Projektionsrichtung p auf die z'-Achse abgebildet, d.h. auf ein Vielfaches λ von $e_{z'}$. Damit die Tiefeninformation bei dieser Transformation erhalten bleibt, muß $\lambda > 0$ sein, kann sonst aber beliebig gewählt werden. Wählt man $\lambda = 1$ und trägt die drei Bildvektoren wieder in die Zeilen einer 3×3-Matrix ein, so erhält man die Matrix

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -\cos\alpha\cos\beta & -\sin\alpha\cos\beta & -\sin\beta \end{bmatrix}.$$

Zum Wechsel in das Bildschirmkoordinatensystem wird die Inverse dieser Matrix benötigt. Gemäß Formel 3.23 wird die inverse 3×3-Matrix in eine 4×4-Matrix eingebettet, und es ergibt sich folgende Transformations-Matrix:

$$[x', y', z', w'] = [x, y, z, w] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -\frac{\cos\alpha}{\tan\beta} & \frac{-\sin\alpha}{\tan\beta} & -\frac{1}{\sin\beta} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Die Komposition mit der anschließenden Parallelprojektion entlang der z'-Achse liefert die Matrix

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -\frac{\cos\alpha}{\tan\beta} & \frac{-\sin\alpha}{\tan\beta} & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Shape3D-Klasse (Java3D) (6.2.10.1, S.351)
Mit Shape3D wird die Form einer **Geometrie** festgelegt, z.B. Würfel, Zylinder oder Kugel.

Sichtgraph (view branch graph) (Java3D) (6.2.7.8, S.341)
Der Teil des **Szenegraphen**, in dem die Parameter bezüglich der Sichtweise festgelegt sind. Zwei der Parameter sind der Augpunkt und die Blickrichtung des virtuellen Betrachters in der Szene.

Skalierungsmatrix (3.3.2.1, S.116)
Bei einer Skalierung S ergibt sich für die affinen Basisvektoren folgende Beziehung:

$$\begin{aligned} S((1,0,0)) &= (s_1,0,0) \\ S((0,1,0)) &= (0,s_2,0) \\ S((0,0,1)) &= (0,0,s_3). \end{aligned}$$

Man erhält als Skalierungsmatrix

$$[x',y',z',w'] = [x,y,z,w] \begin{bmatrix} s_1 & 0 & 0 & 0 \\ 0 & s_2 & 0 & 0 \\ 0 & 0 & s_3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Der Sonderfall $s_1 = s_2 = s_3 = s$ bedeutet die gleiche Skalierung für alle Koordinaten.

spektrale Dichte (7.8.1, S.472)

Um die Abhängigkeit von der Frequenz zu erfassen, wird jeder strahlungsphysikalischen Größe X_e eine *spektrale Dichte* $X_{e\lambda}$ zugeordnet:

$$X_{e\lambda} = \frac{dX_e}{d\lambda}.$$

Oft ist es zweckmäßig, mit einer *normierten spektralen Dichtefunktion*

$$\frac{X_{e\lambda}(\lambda)}{X_{e\lambda}(\lambda_0)}$$

zu arbeiten, wobei λ_0 eine zu wählende Bezugsfrequenz ist.

Spektralwert (7.9.2, S.483)

Bestimmung der Farbgleichung bei vorgegebenen Primärvalenzen für einen beliebigen Spektralreiz:
Man zerlegt den sichtbaren Wellenlängenbereich in enge Spektralbänder $\lambda_{k,\Delta\lambda} \ldots \lambda_{l,\Delta\lambda}$ der Bandbreite $\Delta\lambda$ und betrachtet zunächst das Farbempfinden, das ein auf das Spektralband $\lambda_{i,\Delta\lambda}$ beschränkter 'monochromatischer' Spektralreiz erzeugt. Für die zu diesem Farbreiz gehörende Farbvalenz $\mathbf{f}_{\lambda_i}$ kann man eine Farbgleichung aufstellen:

$$\mathbf{f}_{\lambda_i} = \mathbf{r}_{\lambda_i}\mathcal{R} + \mathbf{g}_{\lambda_i}\mathcal{G} + \mathbf{b}_{\lambda_i}\mathcal{B}$$

Die Farbkoeffizienten $\mathbf{r}_{\lambda_i}, \mathbf{g}_{\lambda_i}$ und $\mathbf{b}_{\lambda_i}$ heißen *Spektralwerte* bzgl. der Primärvalenzen $\mathcal{R}$, $\mathcal{G}$, $\mathcal{B}$. Die Spektralwerte können nur mittels Mischexperimenten gewonnen werden. Führt man dieses Experiment für jedes Spektralband innerhalb des sichtbaren Spektralbereichs durch, erhält man die sogenannten *Spektralwertkurven* $\overline{r}$, $\overline{g}$ und $\overline{b}$. Hat man die Spektralwertkurven für ein Primärvalenztripel bestimmt, so kann man daraus die Spektralwertkurven für jedes beliebige Primärvalenztripel berechnen.
Mit Hilfe der Spektralwertkurven für ein gegebenes Primärvalenztripel kann man den Farbreiz, den ein ganzes Wellenlängenspektrum L_λ erzeugt, ermitteln.

spezifische Ausstrahlung (Radiosity) (7.13.1, S.513)

Die der Bestrahlungsstärke entsprechende Sendegröße heißt *spezifische Ausstrahlung* (Radiosity) M_e:

$$d\varphi_{e1} = M_e dA_1.$$

Die Einheit der spezifischen Ausstrahlung ist $W\,m^{-2}$.

Spline (4.4, S.182)

Ein *Spline* ist eine stetige Abbildung q von einer Menge von Intervallen in den $\mathbb{R}^3$. Die Intervalle $[t_i, t_{i+1}],\ i = 0, \cdots, n-1$, werden durch einen *Stützstellenvektor* $T = (t_0, t_1 \cdots, t_n)$ mit $t_0 \le t_1 \le \cdots \le t_n$ definiert. Jedes Intervall $[t_i, t_{i+1}]$ wird dabei auf ein Polynomsegment, das *Spline–Segment*, abgebildet. An den Stützstellen $t_i,\ i = 0, \cdots, n$, stoßen die Spline-Segmente nach Definition zusammen. Dort müssen die Splinesegmente aber nicht unbedingt dieselben Tangentenvektoren besitzen. Sie können sich sowohl in Betrag als auch Richtung unterscheiden.

Spline-Flächen-Interpolation: bikubische Hermite-Int. (4.7.3.2, S.199)

Ein bikubisches Hermite-Pflaster ist gegeben durch

$$q(u,v) = \sum_{i=0}^{3} \sum_{j=0}^{3} H_i^3(u) h_{ij} H_j^3(v),\ 0 \le u, v \le 1,$$

wobei die $H_i^3,\ i = 0, \cdots, 3$ die Hermite-Basisfunktionen sind.
Dabei sind die h_{ij} gegeben durch

$$[h_{ij}] = \begin{bmatrix} q(0,0) & q_v(0,0) & q_v(0,1) & q(0,1) \\ q_u(0,0) & q_{uv}(0,0) & q_{uv}(0,1) & q_u(0,1) \\ q_u(1,0) & q_{uv}(1,0) & q_{uv}(1,1) & q_u(1,1) \\ q(1,0) & q_v(1,0) & q_v(1,1) & q(1,1) \end{bmatrix}.$$

Dabei sind

$$q_u = \frac{\partial q}{\partial u},\ q_v = \frac{\partial q}{\partial v} \quad \text{und} \quad q_{uv} = \frac{\partial^2 q}{\partial u \partial v}.$$

Die gemischten Ableitungen q_{uv} an den Ecken der Pflaster heißen *Twistvektoren*.

Spline-Flächen-Interpolation: Interpolationsaufgabe (4.7.3, S.197)

Gesucht ist eine Funktion q, so daß folgende Bedingungen erfüllt sind:

$$P_{ik} = q(u_i, v_k), \quad P_{ik} \in \mathbb{R}^3,\ u_i, v_k \in \mathbb{R}, \quad i = 0, \cdots, n, \quad k = 0, \cdots, m,$$

wobei die (u_i, v_k) die Parameterwerte der zu interpolierenden Punkte P_{ik} sind.

Spline-Flächen-Interpolation: Monom- und Lagrange (4.7.3.1, S.198)

Monom-Interpolation:
Ist eine Tensorproduktfläche durch

$$q(u,v) = \sum_{i=0}^{m} \underbrace{\sum_{j=0}^{n} c_{ij} G_j^n(v)}_{a_i(v)} F_i^m(u), \quad a \le u \le b,\ c \le v \le d,$$

und eine $(m+1) \times (n+1)$ große Matrix von Stützpunkten $P_{kl} \in \mathbb{R}^3$ und Stützstellen $(u_k, v_l),\ k = 0, \cdots, m,\ l = 0, \cdots, n$, gegeben und soll die Tensorproduktfläche diese Datenpunkte interpolieren, so muß

$$q(u_k, v_l) = \sum_{i=0}^{m} \sum_{j=0}^{n} c_{ij} F_i^m(u_k) G_j^n(v_l) = P_{kl}, \quad k = 0, \cdots, m,\ l = 0, \cdots, n,$$

für alle $(m+1)\times(n+1)$ Parameterpaare (u_l, v_k), $k = 0,\cdots,m$, $l = 0,\cdots,n$, und Datenpunkte P_{kl} erfüllt sein.
In Matrizenschreibweise lauten diese Gleichungen

$$q(u_k, v_l) = (F_0^m(u_k)\cdots F_m^m(u_k)) \begin{pmatrix} c_{00} & \cdots & c_{0n} \\ \cdot & & \cdot \\ \cdot & & \cdot \\ \cdot & & \cdot \\ c_{m0} & \cdots & cmn \end{pmatrix} \begin{pmatrix} G_0^n(v_l) \\ \cdot \\ \cdot \\ \cdot \\ G_n^n(v_l) \end{pmatrix} = P_{kl},$$

$$k = 0,\cdots,m,\ l = 0,\cdots,n.$$

Wir erhalten $(m+1)\cdot(n+1)$ Gleichungen, die wir zusammen in Matrizenform schreiben können:

$$\mathbf{P} = F\mathbf{C}G$$

mit

$$\mathbf{P} = \begin{bmatrix} P_{00} & \cdots & P_{0n} \\ \vdots & & \vdots \\ P_{m0} & \cdots & P_{mn} \end{bmatrix},$$

$$F = \begin{bmatrix} F_0^m(u_0) & \cdots & F_m^m(u_0) \\ \vdots & & \vdots \\ F_0^m(u_m) & \cdots & F_m^m(u_m) \end{bmatrix},$$

$$\mathbf{C} = \begin{bmatrix} c_{00} & \cdots & c_{0n} \\ \vdots & & \vdots \\ c_{m0} & \cdots & c_{mn} \end{bmatrix},$$

$$G = \begin{bmatrix} G_0^n(v_0) & \cdots & G_0^n(v_n) \\ \vdots & & \vdots \\ G_n^n(v_0) & \cdots & G_n^n(v_n) \end{bmatrix}.$$

Sind die Parameterwerte $u_0,\cdots,u_m$ und $v_0,\cdots,v_n$ paarweise verschieden und verwendet man die Monom-Basis, so sind die Matrizen F und G Vandermond'sche Matrizen und daher invertierbar. Die Koeffizientenmatrix $\mathbf{C}$ ist dann gegeben durch

$$\mathbf{C} = F^{-1}\mathbf{P}G^{-1}.$$

Lagrange-Interpolation:
Wird die Lagrange-Basis verwendet, d.h. die Funktionen F_i^m, $i = 0,\cdots,m$, G_j^n, $j = 0,\cdots,n$, sind die Lagrange'schen Basisfunktion zu den Parameterwerten $u_0,\cdots,u_m$ bzw. $v_0,\cdots,v_n$, so gilt

$$F_i^m(u_k) = \delta_{ik} = \begin{cases} 1 & \text{für } i = k \\ 0 & \text{sonst} \end{cases}$$

und

$$G_j^n(v_l) = \delta_{jl} = \begin{cases} 1 & \text{für } j = l \\ 0 & \text{sonst} \end{cases}.$$

Daher sind F und G in diesem Fall Einheitsmatrizen und es gilt $\mathbf{C} = \mathbf{P}$. Wie im Fall von Kurven sind die Koeffizienten direkt durch die zu interpolierenden Punkte gegeben, d.h.

$$q(u,v) = \sum_{i=0}^{m} \sum_{j=0}^{n} P_{ij} L_i^m(u) L_j^n(v).$$

Standard-Geometrie (Java3D) (6.2.10.2, S.355)
Eine bestimmte Gruppe von **geometrischen Objekten** mit grundlegender Form. In Java3D stehen nur 10 Standard-Geometrien zur Verfügung: Punkte, Strecken/Linien, Dreiecke, Vierecke, Streckenzüge/Linienzüge, Streifen aus Dreiecken und Fächer aus Dreiecken. Siehe auch **Geometrie**.

state (OpenGL) (6.3.3.1, S.377)
Siehe **Zustand**.

Status (OpenGL) (6.3.4.10, S.390)
Siehe **Zustand**.

stetig differenzierbare Fläche (4.7.1.1, S.193)
Eine Fläche heißt *n-mal stetig differenzierbar*, wenn die Abbildung q mindestens n-mal stetig differenzierbar ist, d.h. wenn q n-mal stetige partielle Ableitungen besitzt.

Strahldichte (Radiance) (7.8.1, S.471)
Bei flächenförmigen Lichtquellen ist die Strahlstärke nicht nur richtungs-, sondern auch ortsabhängig.
Die auf ein Flächenelement dA_1 bezogene Strahlungsstärke in Richtung ϑ_1 ist zum einen zur Fläche dieses Flächenelements und zum anderen zum Kosinus dieses Winkels proportional. Es gilt

$$dI_{e_1} = L_{e_1} \cos\vartheta_1 dA_1.$$

Die Proportionalitätsgröße L_{e_1} heißt *Strahldichte* (Radiance). Sie hat die Einheit $W\,sr^{-1}\,m^{-2}$.

Strahler (7.11.1.6, S.496)
Beim *Strahler* wird die Lichtausbreitung der Quelle auf einen bestimmten Raumwinkel (Lichtkegel) beschränkt. Der Abfall der Lichtstärke vom größten Wert in Richtung der Symmetrieachse zum Rand wird durch folgendes Gesetz bestimmt:

$$I = I_0 \cdot cos^n \vartheta.$$

Der Exponent n bestimmt dabei die Bündelung des Lichts.

Strahlstärke (7.8.1, S.470)

Strahlt die Punktlichtquelle in alle Richtungen gleich ab, so muß die auf das differentielle Raumwinkelelement $d\Omega_1$ entfallende Strahlungsleistung $d\varphi_{e_1}$ diesem proportional sein. Es gilt also:

$$d\varphi_{e_1} = I_{e_1} d\Omega_1 .$$

Die Proportionalitätsgröße I_e heißt *Strahlstärke* oder auch *Intensität* (Intensity). Die Einheit der Strahlstärke ist Wsr^{-1}. Da natürliche Lichtquellen nicht in alle Richtungen gleich stark abstrahlen, ist die Strahlstärke im allgemeinen eine Funktion der Abstrahlrichtung.

Strahlungsenergie (7.8.1, S.469)

Die wichtigste Größe in der Radiometrie ist die *Strahlungsenergie* (Radiant Energy) Q_e. Von ihr werden alle anderen Größen abgeleitet.

Strahlungsleistung (7.8.1, S.469)

Die *Strahlungsleistung* (Radiant Flux) φ_e ist die zeitliche Ableitung der Strahlungsenergie. Sendet ein Körper im Zeitintervall dt die Strahlungsenergie dQ_e aus, so ist seine Strahlungsleistung durch

$$\varphi_e = \frac{dQ_e}{dt}$$

gegeben. Die Strahlungsleistung wird auch *Strahlungsfluß* genannt und hat die Einheit Watt.
Um aufgenommene Strahlungsleistung von abgegebener unterscheiden zu können, bekommt φ_e noch einen Index 1, wenn es sich um eine Strahlungsquelle handelt und eine 2, falls es ein Strahlungsempfänger ist.

strahlungsphysikalische Grundgrößen (7.8.1, S.468)

Im Kurs werden die folgenden *strahlungsphysikalischen Grundgrößen* behandelt:

- **Raumwinkel**
- **Strahlungsenergie**
- **Strahlungsleistung**
- **Strahlstärke (Intensität)**
- **spezifische Ausstrahlung (Radiosity)**
- **Bestrahlungsstärke**
- **Strahldichte (Radiance)**

Streifen aus Dreiecken (6.2.10.2, S.356)

Bei Streifen aus Dreiecken wird im Vergleich zu Linienzügen zusätzlich der dritte Eckpunkt automatisch mit dem zwei Eckpunkte davor liegenden Eckpunkt verbunden, also mit dem ersten. Der erste, zweite und dritte Eckpunkt bilden ein Dreieck. Anschließend wird der vierte Eckpunkt mit dem zweiten verbunden. Der zweite, dritte und vierte Eckpunkt bilden ein zweites Dreieck. Analog wird mit allen weiteren Eckpunkten verfahren.

Szenegraph (scene graph) (Java3D) (6.2.2, S.326)

Die zentrale Datenstruktur in Java3D. Sie enthält alle Daten zur Beschreibung der Szene und die Daten, die die Sichtweise des virtuellen Beobachters in der Szene spezifizieren.
Der Szenegraph besteht aus genau einem **Sichtgraphen** und in der Regel mindestens einem **Inhaltsgraphen**.
Der Inhaltsgraph besteht unter anderem aus **BranchGroup**-, **TransformGroup**- und **Leaf**-Datenelementen.
In der vorliegenden Kurseinheit enthält der Szenegraph alle Datenelemente, inklusive des **VirtualUniverse**-Datenelements. In der Literatur ist der Szenegraph jedoch manchmal anders definiert. Dann ist das VirtualUniverse-Datenelement nicht Teil des Szenegraphen, sondern der Vater aller Szenegraphen. In diesem Fall ist die Wurzel jedes Szenegraphen ein **Locale**-Datenelement. In einer Java3D-Anwendung lassen sich demzufolge mehrere Szenen definieren - ein Fall, der eher selten ist.
Bei OpenGL stellt der Open Inventor eine dem Szenegraphen von Java3D ähnliche Datenstruktur zur Verfügung.

Tangentialebene (4.7.1.1, S.194)

Ist $q : D \to \mathbb{R}^3$ eine reguläre parametrisierte Fläche, so heißt die von den Vektoren $q_u(u_0, v_0)$ und $q_v(u_0, v_0)$ aufgespannte Ebene *Tangentialebene* $T_{q(u_0,v_0)}$ im Flächenpunkt $q(u_0, v_0)$.

Tensorprodukt-Fläche (4.7.2, S.195)

Ist eine Basis $F_i^m G_j^n,\ i = 0, \cdots, m,\ j = 0, \cdots, n$ eines Tensorproduktraumes $R_1 \otimes R_2$ von Funktionen über $[a,b] \times [c,d] \longmapsto \mathbb{R}$ und Koeffizienten $c_{ij} \in \mathbb{R}^3,\ i = 0, \cdots, m,\ j = 0, \cdots, n$, gegeben, so heißt die bezüglich der Tensorprodukt-Basis dargestellte Funktion

$$q(u,v) = \sum_{i=0}^{m} \sum_{j=0}^{n} c_{ij} F_i^m G_j^n(u,v) = \sum_{i=0}^{m} \sum_{j=0}^{n} c_{ij} F_i^m(u) \cdot G_j^n(v),$$

$(u,v) \in [a,b] \times [c,d]$, *Tensorprodukt-Fläche*.
In Matrizenschreibweise ergibt sich

$$q(u,v) = (F_0^m(u) \cdots F_m^m(u)) \begin{pmatrix} c_{00} & \cdots & c_{0n} \\ \cdot & & \cdot \\ \cdot & & \cdot \\ \cdot & & \cdot \\ c_{m0} & \cdots & c_{mn} \end{pmatrix} \begin{pmatrix} G_0^n(v) \\ \cdot \\ \cdot \\ \cdot \\ G_n^n(v) \end{pmatrix}.$$

Tensorproduktraum (4.7.2, S.194)

Sind $\{F_i^m,\ i = 0, \cdots, m\}$ und $\{G_j^n,\ j = 0, \cdots, n\}$ Basen von zwei Funktionenräumen R_1, R_2 mit reellwertigen univariaten Funktionen (Funktionen in einer Variable) über den Intervallen $[a,b]$ bzw. $[c,d]$, so bilden die bivariaten reellwertigen Funktionen (Funktionen in zwei Variablen)

$$F_i^m G_j^n(u,v) = F_i^m(u) \cdot G_j^n(v),$$

$i = 0, \cdots, m,\ j = 0, \cdots, n,\ (u,v) \in [a,b] \times [c,d]$, eine *Basis des Tensorpro-*

duktraumes $R_1 \otimes R_2$.
Dieser Raum hat die Dimension $(m+1)\cdot(n+1)$.

Transformation (6.2.2, S.326)
Die mathematische Operation, die auf einen **Eckpunkt** oder eine Menge von Eckpunkten angewendet wird. Der Begriff Transformation steht allgemein für geometrische Transformationen von geometrischen Objekten. Übliche Transformationen sind die Translation, Skalierung, Scherung und die Rotation. Jede dieser Transformationen wird durch eine affinen Abbildung definiert. Die affine Abbildung wird als 4×4-Matrix dargestellt und in einem **TransformGroup**-Datenelement gespeichert. Die entsprechende Transformation wird auf alle geometrische Objekte angewendet, die im Teilgraphen liegen, dessen Wurzel das TransformGroup-Datenelement ist.
„Bei OpenGL der Übergang zwischen zwei Koordinatensystemen." ([Cla97], S. 348)

Transformation von Normalen (3.3.3, S.119)
Wird z.B. eine affine Ebene E im $\mathbb{R}^4$ durch die Gleichung

$$x \cdot n^t + d = 0$$

mit Normalenvektor $n = (n_x, n_y, n_z, n_w)$, $x = (x_x, x_y, x_z, x_w)$ und $d \in \mathbb{R}$ beschrieben, so ist zu beachten, daß sich Normalenvektoren nicht mit derselben Matrix A wie die Ortsvektoren transformieren:

$$x \cdot n^t = \underbrace{(x \cdot A)}_{x'} \cdot \underbrace{A^{-1} n^t}_{\left(n \cdot (A^{-1})^t\right)^t}.$$

Transformationen zwischen Primärvalenzsystemen (7.9.2, S.486)
Zunächst drückt man die Primärvalenzen von S' in Koordinaten von S aus:

$$\begin{aligned} P &= p_r \cdot R + p_g \cdot G + p_b \cdot B \\ D &= d_r \cdot R + d_g \cdot G + d_b \cdot B \\ T &= t_r \cdot R + t_g \cdot G + t_b \cdot B \end{aligned}$$

Aus den Koeffizienten erhält man die invertierbare Transformationsmatrix M:

$$S' = MS = \begin{pmatrix} p_r & p_g & p_b \\ d_r & d_g & d_b \\ t_r & t_g & t_b \end{pmatrix} S.$$

TransformGroup-Klasse (Java3D) (6.2.7.4, S.336)
Wie **BranchGroup** eine Subklasse von **Group**. Wie BranchGroup-Datenelemente dienen TransformGroup-Datenelemente der Gruppierung weiterer **Datenelemente**. Jedes TransformGroup-Datenelement ist - wie jeder innere Knoten eines **Szenegraphen** - die Wurzel eines Teilgraphen des Szenegraphen. Alle Knoten bzw. Datenelemente in einem solchen Teilgraphen werden logisch gruppiert. Der Teilgraph wird **branch graph** genannt, branch von Verzweigung.
Zusätzlich werden die geometrischen Daten der gruppierten Datenelemente

geometrisch transformiert, siehe **Transformation**.
Für die Kinder eines TransformGroup-Datenelements gilt das gleiche wie für BranchGroup-Datenelemente: Ihre Anzahl ist beliebig und alle Kinder sind Instanzen der Subklassen von Group oder/und der Klasse **Leaf**.

Translationsmatrix (3.3.1, S.114)
Die Gleichung $p' = T(p) = T(0) + p = (x_0, y_0, z_0) + (x, y, z)$ beschreibt eine Translation T im reellen dreidimensionalen affinen Raum. Die zur Translation gehörende homogene Matrix hat dann die folgende Gestalt:

$$[x', y', z', w'] = [x, y, z, 1] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ x_0 & y_0 & z_0 & 1 \end{bmatrix} \text{ oder}$$

$$p' = p \cdot A.$$

Dabei wurde $w = w_0 = 1$ gewählt.

Umgebungslicht (ambientes Licht) (7.11.1.1, S.495)
Eine ambiente Komponente dient in Beleuchtungsverfahren dazu, alle vom Modell nicht erfaßten, jedoch in Wirklichkeit vorhandenen, indirekten Beleuchtungen zu berücksichtigen. Damit werden auch nicht direkt beleuchtete Szenenteile sichtbar. Das ambiente Licht fällt auf allen Flächen der Szene mit gleicher Stärke ein und wird deshalb durch die Beleuchtungsstärke $E_a(\lambda)$ in Lux (lx) charakterisiert.
Die ambiente **Leuchtdichte** ergibt sich zu

$$L_{amb}(\lambda) = \rho_a(\lambda) E_a(\lambda),$$

wobei ρ_a der ambiente Reflexionsfaktor ist.

Unterraum des affinen Raumes (3.1.2, S.102)
Eine nichtleere Teilmenge $B \subset A^n$ heißt *r-dimensionaler Unterraum* von A^n, falls ein r-dimensionaler Untervektorraum W von V^n existiert, so daß die folgenden beiden Bedingungen erfüllt sind:

(i) $p, q \in B \Rightarrow w = (\vec{pq}) \in W$
(ii) $p \in B, w \in W \Rightarrow$ Es gibt ein $q \in B$ mit $(\vec{pq}) = w$.

Dann ist B selbst ein affiner Raum.
Ein $(n-1)$-dimensionaler Unterraum von A^n heißt *Hyperebene*.

Variation Diminishing Property (4.3.2.3, S.181)
Die Bézier-Kurven im $\mathbb{R}^3$ haben die Eigenschaft, daß eine beliebige Ebene im $\mathbb{R}^3$ die Kurve nicht öfter als das Kontrollpolygon schneidet (*Variation Diminishing Property*).

VirtualUniverse-Klasse (Java3D) (6.2.7.4, S.336)
Ein zur VirtualUniverse-Klasse gehörendes Datenelement repräsentiert ein **virtuelles Universum**. Ein VirtualUniverse-Datenelement kann nur **Locale**-Datenelemente als Kinder haben. Somit ist in einer **Anwendung** die Wurzel aller **Szenegraphen** ein VirtualUniverse-Datenelement.

virtuelles Universum (Java3D) (6.2.7.4, S.336)

Der konzeptionelle Raum, in dem die **visuellen Objekte** existieren. In jeder **Anwendung**, die auf Java3D aufbaut, gibt es ein virtuelles Universum. Das virtuelle Universum wird durch ein **VirtualUniverse**-Datenelement bzw. eine VirtualUniverse-Klasse repräsentiert.

Visibilitätsverfahren (5.3, S.267)

Eine möglichst photorealistische Bilddarstellung setzt die korrekte Ermittlung sichtbarer und die Beseitigung unsichtbarer Bildteile voraus. Hierzu dienen Visibilitätsverfahren (Hidden-line-, Hidden-surface-Algorithmen).
Dem Einsatz des eigentlichen Visibiltätsverfahrens gehen einfache Verarbeitungsschritte voraus:

- Die perspektivische Transformation aller darzustellenden Objekte reduziert die Bestimmung sich gegenseitig verdeckender Punkte auf einen Vergleich ihrer z-Koordinaten.
- Die Rückseiten von Körpern werden, falls keine perspektivische Transformation durchgeführt wurde, anhand ihrer Oberflächen-Normalenvektoren eliminiert.
- Verfahren zur Punkteklassifizierung testen, ob ein Punkt innerhalb oder außerhalb eines geschlossenen Polygons liegt.
- Min/max-Tests testen, ob sich Polygone mit Sicherheit nicht verdecken.

Je nachdem, ob Punkte, Linien oder Flächen auf Verdeckung getestet werden, unterscheidet man

- punktorientierte Visibilitätsverfahren:
  z-Buffer-Algorithmus (Tiefenspeicher) und Varianten
- linienorientierte Visibilitätsverfahren:
  Watkins-Algorithmus
  regionenorientierter Algorithmus
- flächenorientierte Visibilitätsverfahren:
  Warnock-Algorithmus

visuelles Objekt (6.2.11.2, S.361)

Geometrische Objekte, je nach Situation sind zusätzlich auch virtuelle Lichtquellen, der Hintergrund oder andere Elemente gemeint. Die Gesamtheit der visuellen Objekte samt ihren Eigenschaften ergibt den **Inhalt** (content) eines **virtuellen Universums**. Der Ausdruck „visuelles Objekt" statt „Objekt" wird verwendet, um Verwechslungen mit Objekten im objektorientierten Sinn auszuschließen.

w-Clipping (5.2.2, S.260)

Bei der perspektivischen Transformation gibt es endliche Punkte, die auf unendliche Punkte abgebildet werden, und unendliche Punkte, die auf endliche Punkte abgebildet werden. Somit können Liniensegmente auf unterschiedliche Teilsegmente abgebildet werden. Beim w-Clipping wird das Wraparound-Problem in euklidischen 4D-Koordinaten beschrieben und gelöst.

Warnock-Algorithmus (5.3.7.1, S.289)

Der Warnock-Algorithmus ist ein flächenorientiertes Visibilitätsverfahren, bei dem geprüft wird, ob eine Objektfläche

- außerhalb einer Rasterfläche liegt,
- als einziges Objekt über der Rasterfläche liegt,
- der Rasterfläche am nächsten liegt und diese vollständig überdeckt.

Ist eine der Bedingungen erfüllt, liegt die Sichtbarkeit des Objekts fest, andernfalls wird die Rasterfläche weiter unterteilt.

Watkins-Algorithmus (5.3.6.2, S.281)

Der Watkins-Algorithmus ist ein linienorientiertes Visibilitätsverfahren.
Beim Watkins-Verfahren werden ebene Polygone auf gegenseitige Verdekkung getestet. Bestimmt wird die sichtbare Information auf einer Rasterzeile des Bildschirms. Um unnötige Schnittpunktberechnungen zu vermeiden, werden alle Polygonkanten entsprechend der y-Werte ihrer Endpunkte vorsortiert.

Wechsel des Koordinatensystems (3.3.4, S.120)

Sind zwei Koordinatensysteme Ψ und Ψ' mit den Basisvektoren e_1, e_2, e_3, e_4 und e'_1, e'_2, e'_3, e'_4 gegeben und beschreibt die Matrix A die Abbildung, welche die Basisvektoren e_i auf die Basisvektoren $e'_i,\ i = 1, \cdots, 4$ abbildet, so berechnen sich die Koordinaten $[x'_1, x'_2, x'_3, x'_4]$ eines Punktes im Koordinatensystem Ψ' aus den Koordinaten $[x_1, x_2, x_3, x_4]$ desselben Punktes im Koordinatensystem Ψ durch

$$[x'_1, x'_2, x'_3, x'_4] = [x_1, x_2, x_3, x_4] \cdot A^{-1}.$$

Weiler-Atherton-Algorithmus (5.1.4, S.257)

Der Weiler-Atherton-Algorithmus erlaubt das **Clippen** beliebiger Polygone an beliebigen Fensterpolygonen, indem er den zu clippenden Polygonzug sy stematisch verfolgt und die sichtbaren Gebiete des zu clippenden Polygons 'ausstanzt'.

Wendelfläche (4.7.1.1, S.194)

$$q : \mathbb{R}^2 \to \mathbb{R}^3, q(u,v) = (v\cos u, v\sin u, cu) \ \text{ mit } c \in \mathbb{R}$$

YIQ–Modell (7.10.1.2, S.488)

Farben können auch durch Leuchtdichte (Luminanz, Helligkeit) und Farbart (Chrominanz) definiert werden. Ihre Kenngrößen sind der *Farbton* und die *Farbsättigung*. Der Farbton ist mit der dominierenden Wellenlänge des Lichtes gegeben. Die Sättigung ist ein Maß für die spektrale Reinheit.
Aus den RGB-Werten wird die Luminanz als bewertete Summe gebildet:

$$Y = 0,3R + 0,59G + 0,11B.$$

Die Chrominanz wird mit den zwei Differenzen $R-Y$ und $B-Y$ angegeben. Die Differenzen werden bewertet zu den Größen I und Q zusammengefaßt:

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0,30 & 0,59 & 0,11 \\ 0,60 & -0,28 & -0,32 \\ 0,21 & -0,52 & 0,31 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}.$$

YUV–Modell (7.10.1.3, S.490)

Die europäischen Systeme PAL (Deutschland) und SECAM (Frankreich) verwenden wie das **YIQ-Modell** die Helligkeit Y und zwei Farbdifferenzen U und V. Die U- und V-Signale lassen sich durch eine einfache Rotation der Koordinaten im Farbraum in die I- und Q-Komponenten des YIQ-Modells überführen:

$$\begin{aligned} I &= -U\sin(33^\circ) + V\cos(33^\circ) \\ Q &= U\cos(33^\circ) + V\sin(33^\circ). \end{aligned}$$

z-Buffer-Algorithmus (5.3.5.1, S.271)

Der z-Buffer-Algorithmus ist ein punktorientiertes Visibilitätsverfahren.
Beim z-Buffer-Algorithmus werden alle Objekte der Szene nacheinander mit der Bildauflösung abgetastet. Alle Punkte des z-Speichers werden mit dem Wert 'unendlich', alle Punkte des Bildspeichers mit der Hintergrundfarbe vorbelegt; besitzt ein Objektpunkt einen z-Wert kleiner als den im z-Speicher aktuell eingetragenen, so wird dieser in den z-Speicher und seine Farbe in den Bildspeicher eingetragen.
Bei diesem Verfahren entsteht durch die Rasterung Aliasing. Das *A-Buffer-Verfahren* und das *EXACT-Verfahren* bestimmen genauer, welche Anteile eines Pixels von einzelnen Objekten überdeckt werden, und bestimmen aus diesen Anteilen den Farbwert des Bildspeichers wesentlich genauer.

Zentralprojektion (3.2, S.106)

Gegeben sei ein Beobachtungspunkt q (Punkt des Beobachters), eine Gerade g, auf die projiziert wird, und eine dazu nichtparallele Gerade g' als Objekt. Der Beobachtungspunkt q soll dabei auf keiner der beiden Geraden g bzw. g' liegen. Nun wird einem Punkt $p' \in g'$ (Objektpunkt) der Schnittpunkt $p \in g$ der Geraden durch p' und q zugeordnet. Die Punkte auf der Objektgeraden g' werden dabei auf die Gerade g projiziert.

zusammengesetzte Bézier-Pflaster (4.7.4.2, S.202)

Um zwei Pflaster

$$q_1(u,v) = \sum_{i=0}^{m}\sum_{j=0}^{n} b_{ij}{}_{o_1}^{r_1} B_i^m(u)_{s_1}^{t_1} B_j^n(v),$$

$$i = 0,\cdots m,\ j = 0,\cdots n,\ (u,v) \in [o_1,r_1]\times[s_1,t_1],$$

und

$$q_2(u,v) = \sum_{i=0}^{m}\sum_{j=0}^{n} c_{ij}{}_{o_2}^{r_2} B_i^m(u)_{s_2}^{t_2} B_j^n(v),$$

$$i = 0,\cdots m,\ j = 0,\cdots n,\ (u,v) \in [o_2,r_2]\times[s_2,t_2],$$

entlang der Randkurve in v-Richtung aneinanderschließen zu können, müssen erstens die beiden Pflaster dieselbe Randkurve in v-Richtung besitzen, d.h. für die Bézier-Punkte gilt

$$b_{mj} = c_{0j},\ j = 0, \cdots, n,$$

und zweitens müssen entlang der Randkurve, d.h. zu jedem festen v, die Ableitungen in u-Richtung übereinstimmen. Diese stimmen genau dann überein, wenn gilt

$$\frac{1}{(o_1 - r_1)}(b_{mj} - b_{m-1,j}) = \frac{1}{(o_2 - r_2)}(c_{1j} - c_{0j}),\quad j = 0, \cdots, n.$$

Dies bedeutet, daß die Polygonsegmente kollinear sein und zudem ein Längenverhältnis $(o_1 - r_1) : (o_2 - r_2)$ besitzen müssen.

Zusammenhang zwischen Kontrollpolygon und B-Spline (4.6.2.3, S.189)

Ist

$$q(u) = \sum_{i=0}^{m} N_i^n(u) \cdot d_i$$

eine B-Spline-Kurve vom Grad n über T in $\mathbb{R}^3$, dann hängt die Gestalt des Splines q mit dem Kontrollpolygon wie folgt zusammen:

a) Für $t_l \leq u \leq t_{l+1}$ liegt $q(u)$ in der konvexen Hülle der $n+1$ Kontrollpunkte $d_{l-n}, ..., d_l$.

b) Fallen n Kontrollpunkte $d_{l-n+1} = ... = d_l =: d$ zusammen, so gilt $q(t_{l+1}) = d$, d.h. die Kurve verläuft durch diesen n-fachen Kontrollpunkt.

c) Liegen $n+1$ Kontrollpunkte $d_{l-n}, ..., d_l$ auf einer Geraden L, so gilt $q(u) \in L$ für $t_l \leq u \leq t_{l+1}$, d.h. die Kurve hat mit der Geraden L ein Stück gemeinsam.

d) Fallen n Knoten $t_{l+1} = ...t_{l+n} =: t$ zusammen, so ist $q(t) = d_l$ ein Kontrollpunkt, und die Kurve ist dort tangential an das Kontrollpolygon. Insbesondere verläuft die Kurve bei $(n+1)$-fachen Anfangs- und Endknoten durch die Endpunkte des Polygons und ist dort tangential an das Kontrollpolygon.

Zustand (state) (OpenGL) (6.3.3.1, S.377)

„Der Zustand der OpenGL-„Maschine" bzw. der OpenGL-**Zustandsmaschine** steuert die Abarbeitung aller OpenGL-Befehle. Ein Teil des Zustands ist implementierungsabhängig und nicht durch die **Anwendung** beeinflussbar." ([Cla97], S. 348) Der Zustand von OpenGL besteht aus den Werten von mehr als 250 Variablen. Diese Zustandsvariablen werden auch Attribute genannt.

Zustandsmaschine (state machine) (OpenGL) (6.3.4.10, S.390)

„Eine Modell der Informationsverarbeitung" [Cla97], S. 349. Eine Soft- oder/und Hardware, die intern eine oder mehrere Informationen speichert. Die Informationen werden als (interner) **Zustand** des Systems bezeichnet. Die Soft-/Hardware nimmt Eingaben entgegen, interpretiert diese auf Basis des internen Zustands, verändert eventuell abhängig von dem Ergebnis der

Interpretation den eigenen Zustand und erzeugt eventuell Ausgaben. Setzt eine **Anwendung** auf OpenGL auf, kann die Gesamtheit aller OpenGL-Routinen und -Variablen inklusive des Zustandes als eine Zustandsmaschine betrachtet werden.

Zustandsvariable (OpenGL) (6.3.3.2, S.378)
Siehe **Zustand**.

Literaturverzeichnis

[A+85] G. Abraham et al., *Efficient alias–free rendering using bit–masks and look–up tables*, Computer Graphics 19, 1985, 53–59.

[AEW90] Salim S. Abbi-Ezzi, Michael J. Wozny, *Factoring a Homogeneous Transformation for a more Efficient Graphics Pipeline*, Eurographics '90 (C. E. Vandoni, D. A. Duce, Eds.), North-Holland, September 1990, 159–175.

[App67] Arthur Appel, *The Notion of Quantitative Invisibility and the Machine Rendering of Solids*, Proc. ACM Natl. Mtg., 1967, 387.

[ARB99] ARB, *OpenGL Reference Manual, dritte Auflage*, ISBN: 0-201-65765-1, Dezember 1999.

[Bar85] R. E. Barnhill, *Surfaces in computer aided geometric design: A survey with new results*, Computer Aided Design, 1985, 1–17.

[Ben82] S. A. Benton, *Survey of Holographic Stereograms*, Proceedings of SPIE, Vol. 8, 1982, 15–19.

[Bil83] L. M. Bilinov, *Electro–optical and magneto–optical properties of liquid crystals*, Wiley, 1983.

[Bli87] Jim Blinn, *What, Teapots Again?*, IEEE CG&A, Vol. 9, 1987, 61–63.

[Boe80] W. Boehm, *Inserting new knots into a B-spline curve*, Computer Aided Design, Vol. 12, 1980, 50–60.

[Bou99] Dennis J. Bouvier, *Java 3D API Collateral (Tutorial)*, http://java.sun.com/products/java-media/3D/collateral/, 1999.

[BP99] Kirk Brown, Dan Petersen, *Ready-to-Run Java 3D*, ISBN: 0-471-31702-0, 1999.

[Bre65] J. E. Bresenham, *Algorithm for Computer Control of a Digital Plotter*, IBM System J, Vol. 4, 1965, Nr. 1, 25–30.

[Bre77] J. E. Bresenham, *A Linear Algorithm for Incremental Digital Display of Circular Arcs*, CACM, Vol. 20, 1977, 100–106.

[BS93] Bergmann, Schaefer, *Lehrbuch der Experimentalphysik, Bd. 3: Optik*, 9. Auflage, de Gruyter, 1993.

[Car89] L. Carpenter, *The a–buffer, an antialiasing hidden surface method*, Computer Graphics, Vol. 18, 1989, Nr. 3, 103–108.

[Car91] R. M. Carr, *The point of the pen*, BYTE, Vol. 2, 1991, 211–221.

[CB78] M. Cyrus, J. Beck, *Generalized Two- and Three-Dimensional Clipping*, Computers & Graphics, Vol. 3, 1978, 23–28.

[CCC87] R. L. Cook, L. Carpenter , E. Catmull, *The Reyes Image Rendering Architecture*, ACM Computer Graphics, Vol. 21, 1987, Nr. 4, 95–102.

[CCI82] CCIR, *International Radio Consultative Committee, Recommendation 500-2, Kapitel Method for the Subjective Assessment of the Quality of Television Pictures*, 1982.

[CCWG88] M. F. Cohen, S. Chen, J. R. Wallace , D. P. Greenberg, *A Progressive Refinement Approach to Fast Radiosity Image Generation*, ACM Computer Graphics, Vol. 22, 1988, Nr. 4, 75–84.

[CG85] M. F. Cohen, D. P. Greenberg, *The Hemi-Cube: A Radiosity Solution for Complex Environments*, ACM Computer Graphics, Vol. 19, 1985, Nr. 3, 31–40.

[Cha77] S. Chandrasekar, *Liquid crystals*, Cambridge University Press, 1977.

[Cla80] J. Clark, *A VLSI Geometry Processor for Graphics*, Computer IEEE, Vol. 13, 1980, 59–62, 64, 66–68.

[Cla97] Ute Claussen, *Programmieren mit OpenGL*, ISBN: 3-540-57977-X, 1997.

[CLR80] E. Cohen, T. Lyche , R. F. Riesenfeld, *Discrete B-splines and subdivision techniques in computer aided geometric design and computer graphics*, Computer Graphics and Image Processing, Vol. 14, 1980, 87–111.

[Coo86] R. L. Cook, *Stochastic sampling in computer graphics*, ACM Transactions on Graphics, Vol. 18, 1986, Nr. 3, 51–72.

[CP78] I. Carlbom, J. Paciorek, *Geometric Projection and Viewing Transformations*, Computing Surveys, Vol. 1, 1978, Nr. 4, 465–502.

[CW93] M. F. Cohen, J. R. Wallace, *Radiosity and Realistic Image Synthesis*, First Ed., Academic Press, 1993.

[D$^+$85] M. A. Z. Dippé et al., *Antialiasing through stochastic sampling*, Computer Graphics, Vol. 19, 1985, Nr. 3, 69–78.

[dB72] C. de Boor, *On calculating with B-splines*, J. Approx. Theory, Vol. 6, 1972, 50–62.

[dB78] C. de Boor, *A Practical Guide to Splines*, Springer, 1978.

[dB87] C. de Boor, *B-Form Basics, Geometric Modeling, Algorithms and New Trends*, SIAM, 1987, 131–148.

[dC63] P. de Casteljau, *Courbes et surfaces a poles*, Tech. report, A. Citroen, Paris, 1963.

[DeR89] T. D. DeRose, *A coordinate-free approach to geometric programming*, Theory and Practice of Geometric Modeling (W. Strasser, H.-P. Seidel, Eds.), Springer Verlag, 1989, 291–305.

[DIN82] DIN–ISO 7942, *Information Processing – Graphical Kernel System (GKS), Functional Description*, Beuth Verlag GmbH, Berlin, 1982.

[Duf79] T. Duff, *Smoothly Shaded Renderings of Polyhedral Objects on Raster Displays*, ACM Computer Graphics, Vol. 13, 1979, Nr. 2, 270–275.

[EC90] Gershon Elber, Elaine Cohen, *Hidden Curve Removal for Free Form Surfaces*, Computer Graphics (SIGGRAPH '90 Proceedings) (Forest Baskett, Ed.), Vol. 24, August 1990, 95–104.

[Egl90] H. Eglowstein, *Reach out and touch your data*, BYTE, Vol. 7, 1990, 283–290.

[EKP84] G. Enderle, K. Kansy , G. Pfaff, *Computer Graphics Programming, GKS — The Graphics Standard*, Springer, Berlin, Heidelberg, New York, Tokyo, 1984.

[Enc75] J. L. Encarnaçao, *Computer-Graphics: Programmierung und Anwendung von graphischen Systemen*, R. Oldenburg Verlag, München, West Germany, 1975.

[Eve52] R. R. Everett, *The Whirlwind I Computer*, Rev. Electr. Digital Computing, 1952, 70.

[F+83] E. Fiume et al., *A parallel scan conversion algorithm with anti-aliasing for general purpose Supercomputers*, Computer Graphics, Vol. 17, 1983, Nr. 3, 141–150.

[F+85] H. Fuchs et al., *Fast spheres, shadows, textures, transparencies, and image enhancements in Pixel-planes*, Computer Graphics, Vol. 19, 85, Nr. 3, 111–120.

[Far79] G. E. Farin, *Subsplines über Dreiecken*, Ph.D. thesis, Braunschweig, F.R.G., 1979.

[Far80] G. E. Farin, *Bézier polynomials over triangles and the construction of piecewise C^r-polynomials*, Tech. Report TR/91, Dept. Mathematics, Brunel Univ., Uxbridge, Middlessex, 1980.

[Far86] G. E. Farin, *Triangular Bernstein-Bézier patches*, Computer Aided Geometric Design, Vol. 3, 1986, 83–127.

[Fis79] G. Fischer, *Analytische Geometrie*, Friedrich Vieweg & Sohn Verlagsgesellschaft mbH, 1979.

[FvDFH90] J. D. Foley, A. van Dam, S. T. Feiner , J. F. Hughes, *Computer Graphics - Principles and Practice*, Addison–Wesley, 1990.

[Gla95] A. S. Glassner, *Principles of Digital Image Synthesis*, Morgan Kaufmann, 1995.

[GM69] R. Galimberti, U. Montanari, *An algorithm for hidden-line elimination*, Communications of the ACM, Vol. 12, 1969, 206.

[GMSG82] D. Greenberg, A. Marcus, A. H. Schmidt , V. Gort, *The Computer Image: Applications of Computer Graphics*, Addison-Wesley, 1982.

[Gor69] W. J. Gordon, *Distibutive lattices and the approximation of multivariate functions*, Approximation with Special Emphasis on Spline Functions (I. J. Schoenberg, Ed.), Academic Press, 1969, 223–277.

[Gor71] W. J. Gordon, *Blending function methods of bivariate and multivariate interpolation and approximation*, SIAM J. Numer. Anal., Vol. 8, 1971, 158–177.

[Gou71] H. Gouraud, *Continuous Shading of Curved Surfaces*, IEEE Transactions on Computers, Vol. C-20, 1971, Nr. 6, 623–628.

[GR74] W. J. Gordon, R. F. Riesenfeld, *B-spline curves and surfaces*, Computer Aided Geometric Design (R. E. Barnhill, R. F. Riesenfeld, Eds.), Academic Press, 1974.

[Gra53] H. Graßmann, *Zur Theorie der Farbmischung*, Poggendorffs Annalen der Physik, Vol. 89, 1853.

[Gra79] Graphics Standard Committee, *Status Report of the Graphics Standard Committee*, Computer Graphics, Vol. 13, 1979, Nr. 3.

[GTGB84] C. M. Goral, K. E. Torrance, D. P. Greenberg , B. Battaile, *Modeling the Interaction of Light between Diffuse Surfaces*, ACM Computer Graphics, Vol. 18, 1984, Nr. 3, 213–222.

[Hä81] M. Häusing, *Über die Farbkodierung von Informationen auf elektronischen Sichtgeräten*, Fortschritt der VDI-Zeitschriften, 1981, Nr. 10.

[Hai89] E. Haines, *Essential Ray Tracing Algorithms*, An Introduction to Ray Tracing (Andrew S. Glassner, Ed.), Academic Press, 1989, 33–77.

[Hal88] R. A. Hall, *Illumination and Color in Computer Generated Imagery*, Monographs in Visual Communications, Springer-Verlag, 1988.

[Han89] P. Hanrahan, *A Survey of Ray-Surface Intersection Algorithms*, An Introduction to Ray Tracing (Andrew S. Glassner, Ed.), Academic Press, 1989, 79–119.

[Her89] I. Herman, *Projective geometry and computer graphics*, Advances in Computer Graphics IV (Berlin–Heidelberg–NewYork–Tokyo) (M. Grave, M. Roch, Eds.), Springer Verlag, Berlin–Heidelberg–NewYork–Tokyo, 1989.

[Her92] I. Herman, *The Use of Projective Geometry in Computer Graphics*, Lecture Notes in Computer Science, Springer, 1992.

[HG83] R. A. Hall, D. P. Greenberg, *A Testbed for Realistic Image Synthesis*, IEEE Computer Graphics & Applications, 1983, Nr. 11, 10–20.

[HG86] E. A. Haines, D. P. Greenberg, *The Light Buffer: A Shadow-Testing Accelerator*, IEEE Computer Graphics & Applications, 1986, Nr. 9, 6–16.

[Hil00] Francis S. Hill, *Computer Graphics Using OpenGL*, ISBN: 0-023-54856-8, Mai 2000.

[HL89] Hoscheck, Lasser, *Grundlagen der geometrischen Datenverarbeitung*, Teubner, Stuttgart, 1989.

[HLRS85] C. Hornung, W. Lellek, P. Rehwald , W. Strasser, *An area-oriented analytical visibility method for displaying parametrically defined tensor-product surfaces*, Computer Aided Geometric Design, Vol. 2, 1985, Nr. 1-3, 197–205.

[Hor84] C. Hornung, *A Method for Solving the Visibility Problem*, IEEE Comput. Graphics and Appl. (USA), Vol. 4, 1984, 26–33.

[Hug89] A. J. Huges, *Liquid crystal displays in display engineering*, North–Holland, 1989.

[ISO] ISO 8805, *Graphical Kernel System for Three Dimensions (GKS–3D), Functional Description*.

[ISO82] ISO DIS 2382/13, *Data Processing Vocabulary — Computer Graphics*, 1982.

[ISO84] ISO/TC97/SC5/WG2/N305, *PHIGS — Programmers Hierarchical Interface to Graphics Systems*, 1984.

[ISO89] ISO 9592–(1-3), *Programmer's Hierarchical Interactive Graphics System (PHIGS), Part 1–3*, 1989.

[ISO91] ISO 9592–4, *Programmer's Hierarchical Interactive Graphics System (PHIGS), Part 4: Plus Lumière und Surfaces (PHIGS PLUS)*, 1991.

[Jac92] D. Jackél, *Grafik Computer Grundlagen, Architekturen und Konzepte computergrafischer Sichtsysteme*, First Ed., Springer Verlag, 1992.

[JW63] D. B. Judd, G. Wyszecki, *Color in Business, Science and Industry*, Wiley & Sons, 1963.

[Kaj86] J. T. Kajiya, *The Rendering Equation*, Computer Graphics, Vol. 20, 1986, Nr. 4, 143–150.

[Kau82] A. Kaufman, *High-Level Color Naming Standards and Conversion Algorithms*, Tech. report, Center of Computer Graphics, Ben Gurion University of the Negev Beer-Sheva, 1982.

[Kni93] G. Knittel, *PROVEN - Prompt Vector Normalizer*, Proceedings of the 6. IEEE ASIC Conference, Rochester, 1993, 112–115.

[Kor90] K. Kornel, *2d and 3d perspective transforms*, Computer & Graphics, Vol. 14, 1990, Nr. 1, 117–124.

[Kr"91] W. Krüger, *Visualisierung 3-dimensionaler skalarer Datenfelder: Ein "physikalisches" Modell*, Informationstechnik, Vol. 33, 1991, Nr. 2, 72–76.

[Kra89] G. Krammer, *Notes on the Mathematics of the PHIGS Viewing Pipeline*, Computer Graphics Forum, Vol. 8, 1989, Nr. 3, 219–226.

[Kun94] Michael Kunze, *3D-Grafikstandard OpenGL (Einführung in die 3D-Grafiksprache OpenGL)*, c't, Vol. 8, 1994, 198.

[LF80] J. M. Lane, R. F.Riesenfeld, *A theoretical development for the computer generation of piecewise polynomial surfaces*, IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. 2, 1980, 35–46.

[Lou70] P. P. Loutrel, *A solution to the hidden-line problem for computer drawn polyhedra*, IEEE Trans. Comput., Vol. C-19, 1970, 205–213.

[MRC+86] G. W. Meyer, H. E. Rushmeier, M. F. Cohen, D. P. Greenberg , K. E. Torrance, *An Experimental Evaluation of Computer Graphics Imagery*, ACM Transactions on Graphics, Vol. 5, 1986, Nr. 1, 30–50.

[MS68] T. H. Myer, I. E. Sutherland, *On the Design of Display Processors*, CACM, Vol. 11, 1968, Nr. 6, 410–414.

[NB94] X. Ni, M. S. Bloor, *Performance Evaluation of Boundary Data Structures*, IEEE Computer Graphics and Applications, 1994, Nr. 11, 66–77.

[NR72] F. Nake, A. Rosenfeld (Eds.), *Graphics Languages*, North Holland Pub. Co, Amsterdam, 1972.

[NS79] W. M. Newman, R. F. Sproull, *Principles of Interactive Computer Graphics*, McGraw-Hill, New York, NY, 1979.

[Nus28] W. Nusselt, *Graphische Bestimmung des Winkelverhältnisses bei der Wärmestrahlung*, VDI Z., 1928, Nr. 72, 673.

[Par90] B. Pareigis, *Analytische Geometrie und projektive Geometrie für die Computer–Graphik*, Teubner, Stuttgart, 1990.

[PD84] Thomas Porter, Tom Duff, *Compositing digital images*, Computer Graphics (SIGGRAPH '84 Proceedings) (Hank Christiansen, Ed.), Vol. 18, July 1984, 253–259.

[PH90] J. Pöpsel, Ch. Hornung, *Highlight Shading: Lighting and Shading in a PHIGS+/PEX Environment*, Computers & Graphics, Vol. 14, 1990, Nr. 1.

[Pin88] J. Pineda, *A Parallel Algorithm for Polygon Rasterization*, Computer Graphics, Vol. 22, 1988, Nr. 4, 17–20.

[PJ91] A. Pearce, D. Jevans, *Exploiting Shadow Coherence in Ray Tracing*, Proceedings Graphics Interface '91, Nr. 6, 1991, 109–116.

[PP86] M. Penna, R. Patterson, *Projective Geometry and its Application to Computer Graphics*, Prentice–Hall, 1986.

[Pra78] W. K. Pratt, *Digital Image Processing*, John Wiley & Sons, 1978.

[Pra84] H. Prautzsch, *Unterteilungsalgorithmen für multivariate Splines*, Ph.D. thesis, Braunschweig, 1984.

[Pro87] W. E. Proebster, *Peripherie von Informationssystemen*, Springer–Verlag, 1987.

[Pur86] W. Purgathofer, *A Statistical Method for Adaptive Stochastic Sampling*, Eurographics ’86 (A. A. G. Requicha, Ed.), Nr. 8, 1986, 145–152.

[RH84] P. Rehwald, C. Hornung, *An Analytical Visibility Method for Displaying Parametrically Defined Surfaces*, Eurographics ’84 (K. Bo, H. A. Tucker, Eds.), North-Holland, 1984, 137–149.

[Rie73] R. F. Riesenfeld, *Applications of B-Spline Approximation to Geometric Problems of Computer Aided Design*, Ph.D. thesis, Syracuse University, 1973.

[Rog85] D. F. Rogers, *Procedural Elements for Computer Graphics*, McGraw-Hill, 1985.

[RSWS99] Jr. Richard S. Wright, Michael Sweet, *OpenGL SuperBible, zweite Auflage*, ISBN: 1-571-69164-2, Dezember 1999.

[SA99] Mark Segal, Kurt Akeley, *The OpenGL Graphics System: A Specification (Version 1.2.1)*, http://www.sgi.com/software/opengl/manual.html, April 1999.

[Sam90] H. Samet, *Design and Analysis of Spatial Data Structures*, Addison Wesley, 1990.

[Sch20] E. Schrödinger, *Grundlinien einer Theorie der Farbmetrik im Tagessehen*, Annalen der Physik, Vol. 62, 1920.

[Sch76] H. Schaal, *Lineare Algebra und Analytische Geometrie*, Vieweg, Braunschweig, 1976.

[Sch78] G. Schrak, *Graphische Datenverarbeitung*, BI-Wissenschaftsverlag, Mannheim, Wien, Zürich, 1978.

[Sch81] L. L. Schumaker, *Spline Functions: Basic Theory*, Wiley & Sons, New York, 1981.

[Sch91] A. Schilling, *A new simple and efficient antialiasing with Subpixel masks*, Computer Graphics, Vol. 25, 1991, Nr. 4, 133–141.

[Sei89] H.-P. Seidel, *Polynome, Splines und symmetrische rekursive Algorithmen im Computer Aided Geometric Design*, Habilitationsschrift, Tübingen, 1989.

[Sei90] H.-P. Seidel, *Quaternionen in Computergraphik und Robotik*, Informationstechnik **it**, Vol. 32, 1990, Nr. 4, 266–275.

[Sel00] Daniel Selman, *Java 3D Programming*, ISBN: 1-884-77797-X, Juli 2000.

[SGI97] SGI, *The OpenGL Programming Guide (The Official Guide to Learning OpenGL, Version 1.1)*, http://heron.cc.ukans.edu/ebt-bin/nph-dweb/dynaweb/SGI_Developer/OpenGL_PG/, 1997.

[SH74] I. E. Sutherland, G. W. Hodgman, *Reentrant polygon clipping*, Communications of the ACM, Vol. 17, 1974, 32–42.

[Sie81] R. Siegel, *Thermal Radiation Heat Transfer*, Second Ed., Hemisphere Publishing Corporation, 1981.

[Sil92] F. X. Sillion, *Extension of Radiosity Methods for Non-Diffuse Enviroments*, ACM SIGGRAPH Course Notes Global Illumination, 1992.

[SIL93] SILICON GRAPHICS INC., *The Open GL Programming Guide*, Addison Wesley, 1993.

[SK90] P. Slusallek, M. Krämer, *Progressive Refinement Versus Full Matrix*, Tech. report, WSI/GRIS Universität Tübingen, Juni 1990.

[Slu89] P. B. Slusallek, *Flächenmodellierung mit Splines über Dreiecken*, Master's thesis, Tübingen, 1989.

[Smi78] A. R. Smith, *Color Gamut Transform Pairs*, Computer Graphics, Vol. 12, 1978, Nr. 3.

[SRD97] Henry Sowizral, Kevin Rushforth , Michael Deering, *The Java 3D API Specification*, ISBN: 0-201-32576-4, 1997.

[SS68] Robert F. Sproull, Ivan E. Sutherland, *A Clipping Divider*, FJCC, 1968.

[Str74] Wolfgang Straßer, *Schnelle Kurven– und Flächendarstellung auf graphischen Sichtgeräten*, Ph.D. thesis, TU Berlin, 1974.

[Str91] W. Straßer (Ed.), *Visualisierung* it 2, Oldenbourg–Verlag, München-Wien, April 1991.

[SUN00] SUN, *The Java 3D API Specification (Version 1.2)*, http://java.sun.com/products/java-media/3D, April 2000.

[Sut63] I. E. Sutherland, *Sketchpad — A Man-Machine Graphical Communication System*, Spring Joint Comp. Conf. (Baltimore), Spartan Books, Baltimore, 1963.

[The73] R. Theile, *Fernsehtechnik*, Vol. 1, Springer Verlag, 1973.

[TK82] H. J. Tafel, A. Kohl, *Ein– und Ausgabegeräte der Datentechnik*, Hanser–Verlag, München, Wien, 1982.

[WA77] K. Weiler, K. Atherton, *Hidden surface removal using polygon area sorting*, Computer Graphics (SIGGRAPH '77 Proceedings), Vol. 11, 1977, Nr. 2, 214–222.

[War69] J. Warnock, *A Hidden-Surface Algorithm for Computer Generated Half-Tone Pictures*, Tech. Report TR 4–15, NTIS AD-733 671, University of Utah, Computer Science Department, 1969.

[Wat70] G. S. Watkins, *A real-time visible surface algorithm*, Tech. Report UTEC-CS-70-101, Dept. Comput. Sci., Univ. Utah, Salt Lake City, UT, 1970.

[WCG87] J. R. Wallace, M. F. Cohen , D. P. Greenberg, *A Two–Pass Solution to the Rendering Equation: a synthesis of ray tracing and radiosity method*, ACM Computer Graphics, Vol. 21, 1987, Nr. 4, 311–320.

[WEH89] J. R. Wallace, K. Elmquist , E. Haines, *A Ray Tracing Algorithm for Progressive Radiosity*, ACM Computer Graphics, Vol. 23, 1989, Nr. 3, 315–324.

[Whi79] T. Whitted, *An Improved Illumination Model for Shaded Display*, Special SIGGRAPH '79 Issue, Vol. 13, 1979, Nr. 8, 1–14.

[Wil95] Andreas Will, *OpenGL-Programmierung unter Windows 95 und NT*, c't, Vol. 11, 1995, 336ff.

[WND+99] Mason Woo, Jackie Neider, Tom Davis, Dave Shreiner , OpenGL Architectural Review Board, *The OpenGL Programming Guide (The Official Guide to Learning OpenGL Version 1.2), dritte Auflage*, ISBN 0-201-60458-2, August 1999.

[Wys60] G. Wyszecky, *Farbsysteme*, Musterschmidt, Göttingen, 1960.

[YDL84] B. A. Barsky Y.-D. Liang, *A New Concept and Method for Line Clipping*, ACM Trans. Graphics (USA), Vol. 3, 1984, 1–22.