

# *Algorithmen und Datenstrukturen*

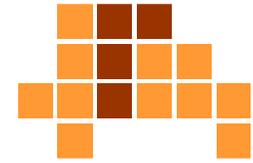
## *Bereichsbäume*

---

Matthias Teschner  
Graphische Datenverarbeitung  
Institut für Informatik  
Universität Freiburg

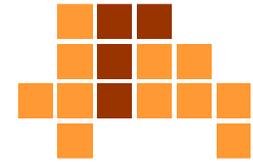
SS 12

# *Überblick*

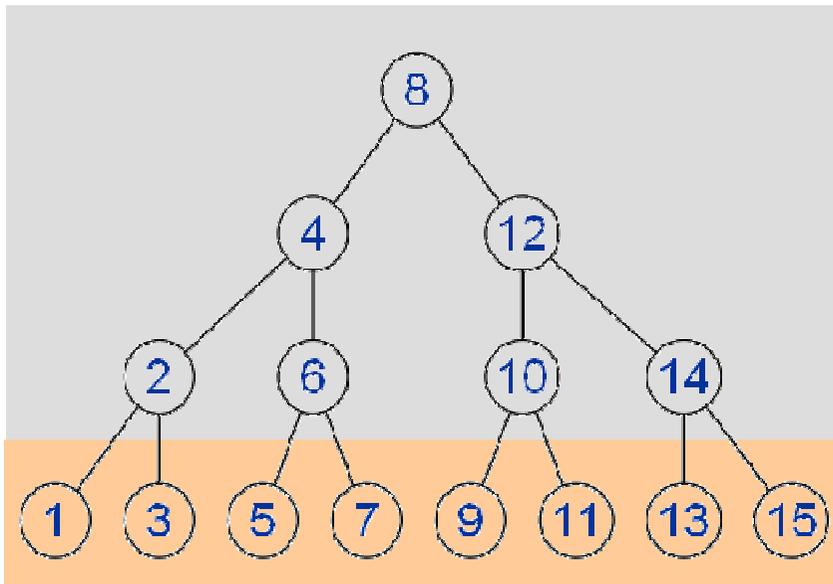


- Einführung
- k-d Baum
- BSP Baum
- R Baum

# Motivation



- Blattsuchbäume
  - Knoten enthalten Hinweise zu Schlüsseln
  - Schlüssel sind in Blättern gespeichert



Innere Knoten enthalten keine Schlüssel.

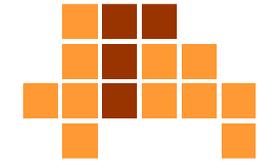
Wert eines Knotens beschreibt ein Kriterium für die Schlüssel in den Blättern.

Bsp: Alle Blätter im linken Teilbaum enthalten Schlüssel, die kleiner als 8 sind. Schlüssel im rechten Teilbaum sind größer gleich 8.

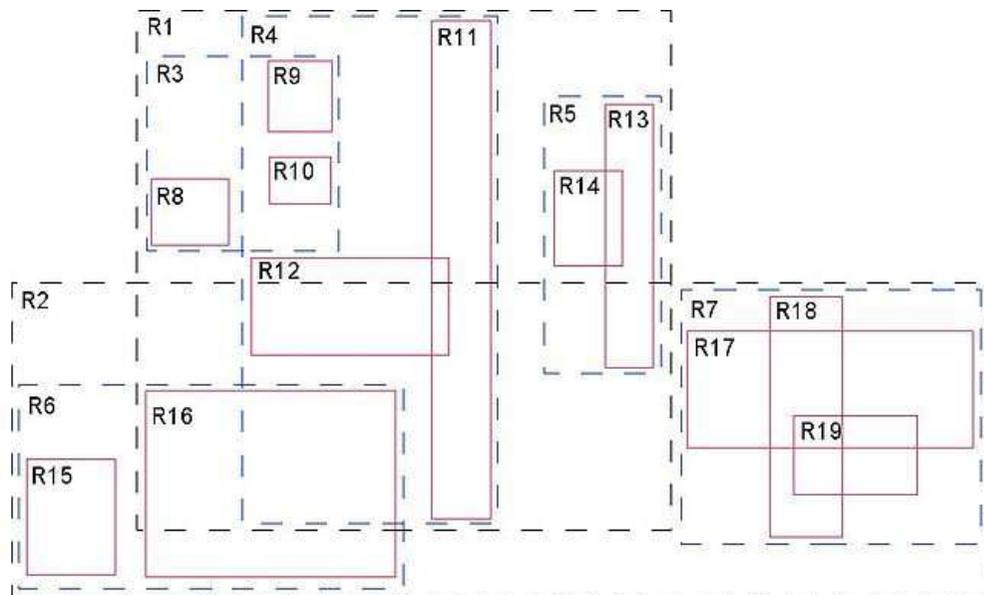
Schlüssel sind in Blättern gespeichert.

1D-Beispiel

# Motivation



- verwalten mehrdimensionale Schlüssel
  - in Datenbanken oder für geometrische Anwendungen



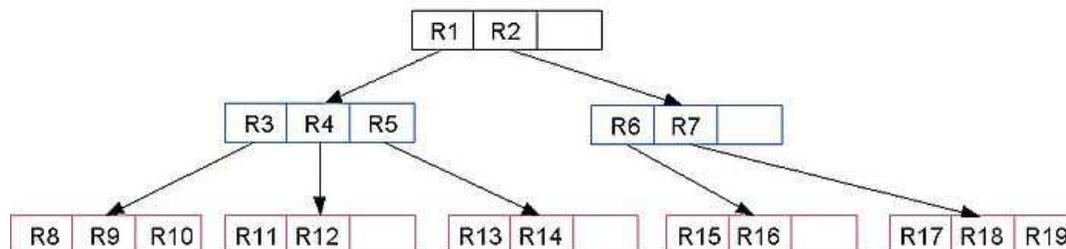
2D-Beispiel

R1-R19 beschreiben 2D-Bereiche, welche die Suche von 2D-Schlüsseln beschleunigen.

R3-55 sind Teilbereiche von R1.  
R6-R7 sind Teilbereiche von R2.  
R8-R10 sind Teilbereiche von R3.  
...

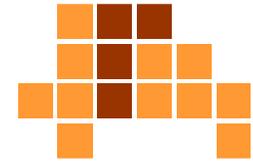
2D-Schlüssel sind in Blättern beispielsweise unter R8-R19 gespeichert.

R1-R19 sind als  $(\min_x, \min_y, \max_x, \max_y)$  repräsentiert.



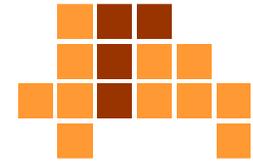
[www.wikipedia.de](http://www.wikipedia.de): R-Baum

# Überblick



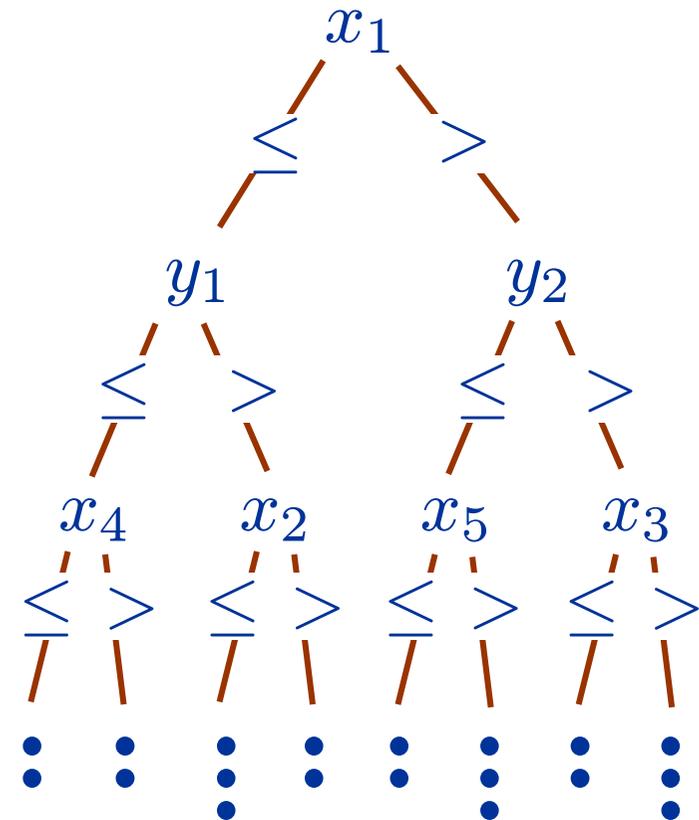
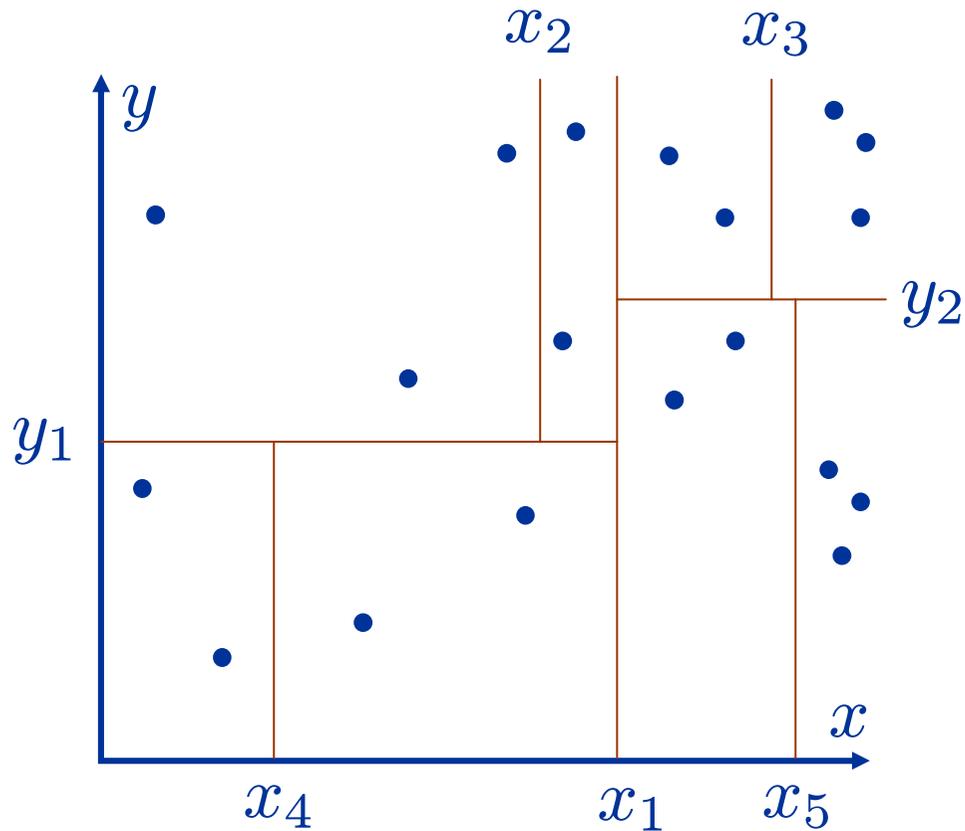
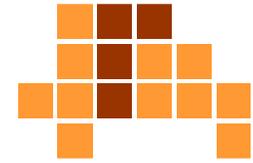
- Einführung
- k-d Baum
- BSP Baum
- R Baum

# Einführung

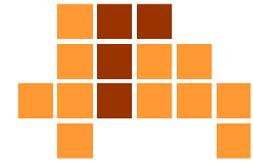


- Blattsuchbaum
  - Knoten beschreiben Eigenschaften von Schlüsseln, die in Blättern gespeichert sind
  - Knoten enthalten keine Schlüssel.  
Alle Schlüssel sind in Blättern gespeichert.
- Schlüssel von beliebiger Dimensionalität
  - $s.key[0], s.key[1], \dots, s.key[k-1]$
- effiziente Bereichsanfrage
  - suche alle Schlüssel  $s$  mit
    - $min_0 \leq s.key[0] \leq max_0$   
 $min_1 \leq s.key[1] \leq max_1$   
...
    - $min_{k-1} \leq s.key[k-1] \leq max_{k-1}$

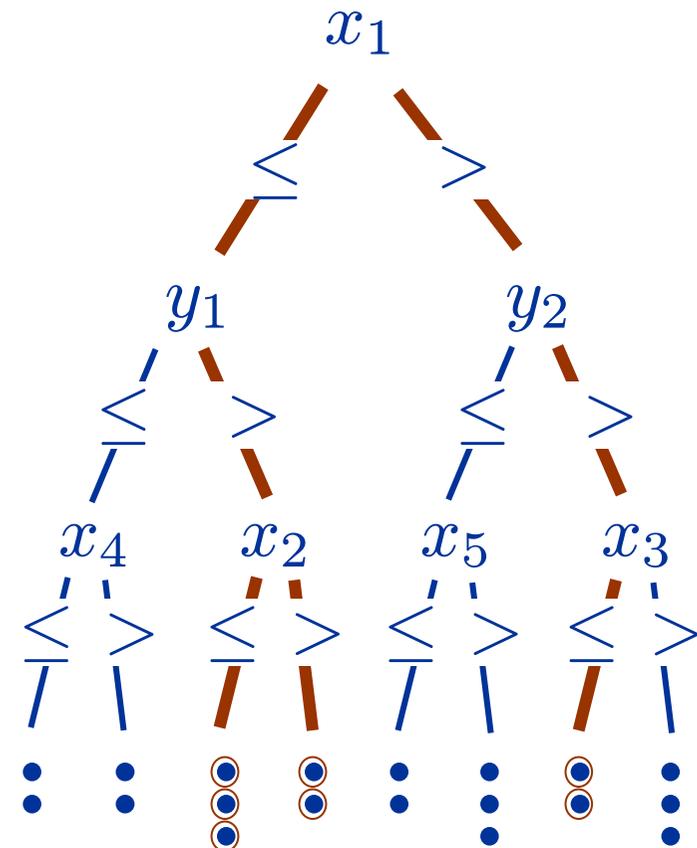
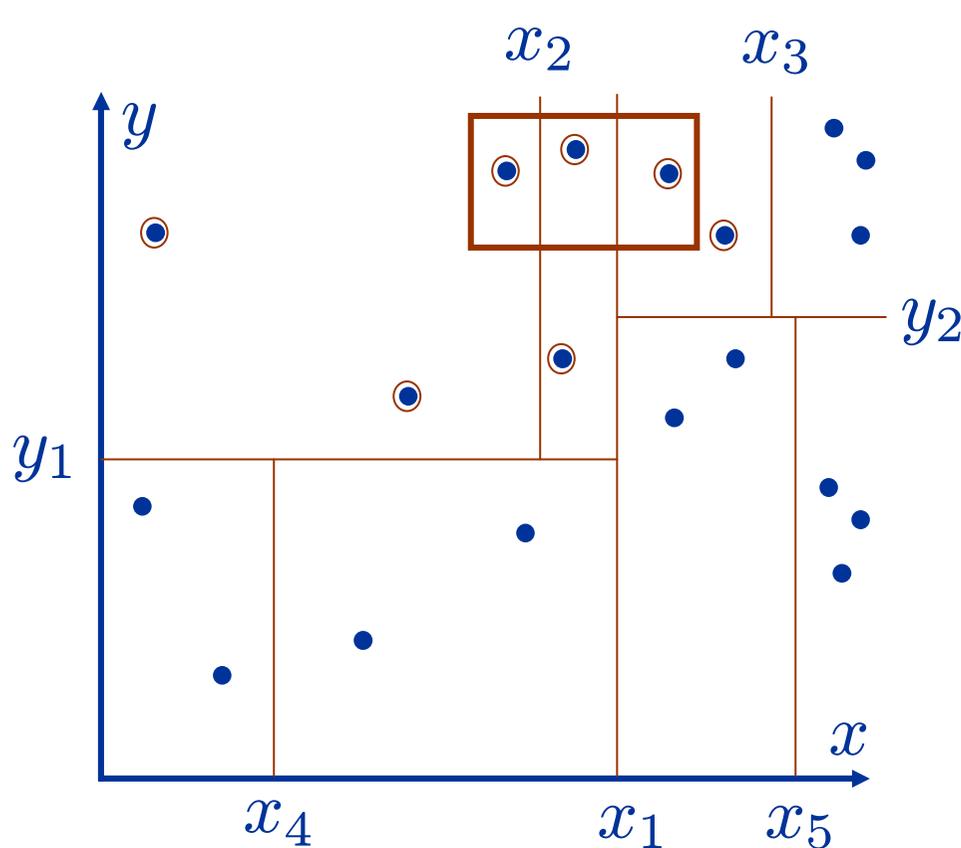
# *k-d Baum (2D)*



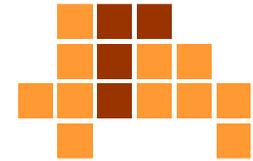
# Bereichsabfrage



- Bereichsintervalle bestimmen zu traversierende Knoten
- Elemente in Blättern (  $\odot$  ) werden gegen Bereich getestet

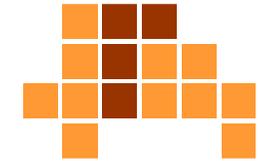


# *k Dimensionen*



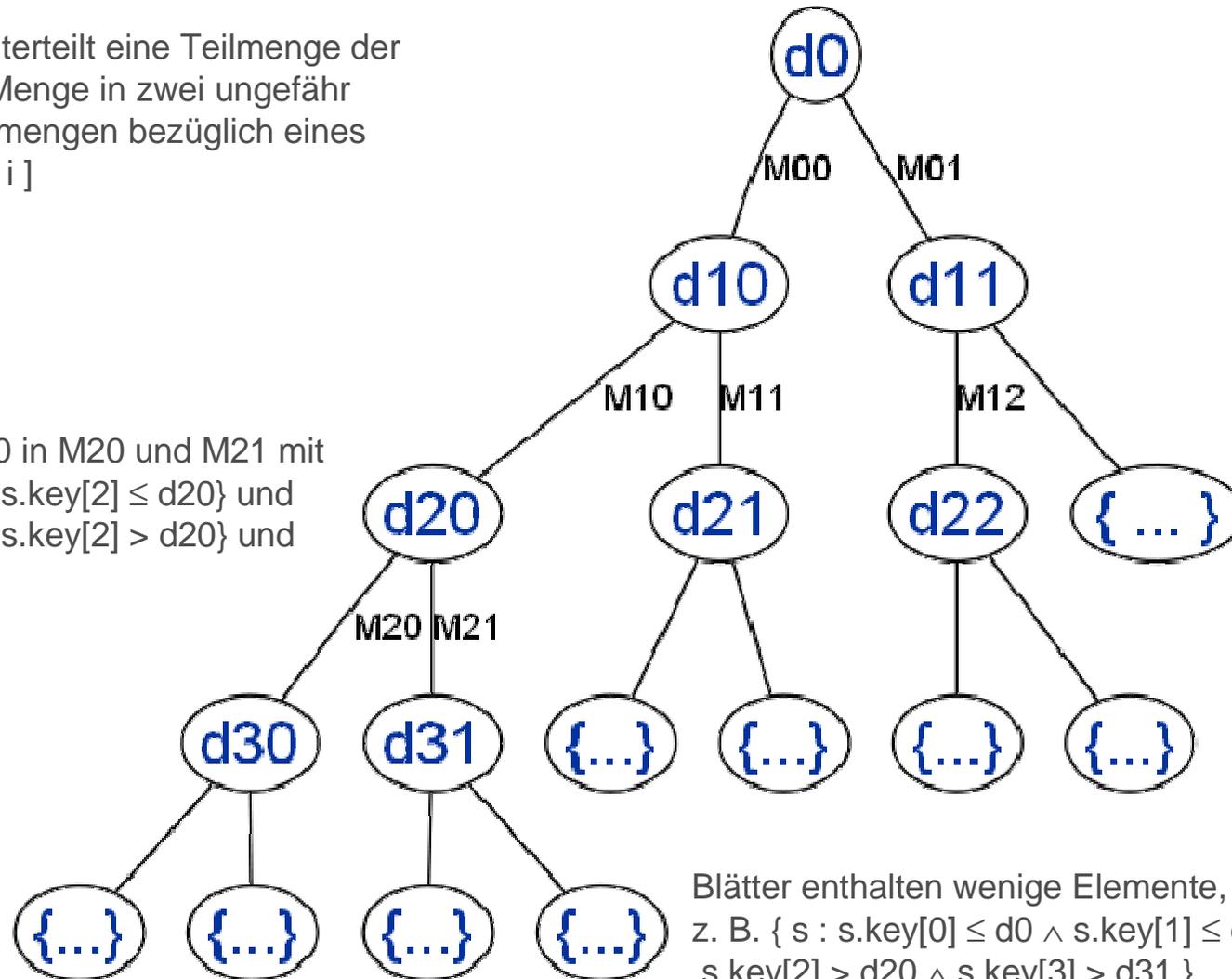
- Schlüssel mit  $k$  Elementen  $s.key[0]$ ,  $s.key[1]$ , ...,  $s.key[k-1]$
- Menge  $M$
- Aufbau des Baums
  - Wahl eines Wertes  $d_0$ , sodass  $M$  in zwei ungefähr gleich große Teilmengen  $M_{00}$ ,  $M_{01}$  geteilt wird mit  $M_{00} = \{ s \in M : s.key[0] \leq d_0 \}$  und  $M_{01} = \{ s \in M : s.key[0] > d_0 \}$
  - Wahl eines Wertes  $d_{10}$ , sodass  $M_{00}$  unterteilt wird:  
 $M_{10} = \{ s \in M_{00} : s.key[1] \leq d_{10} \}$  und  $M_{11} = \{ s \in M_{00} : s.key[1] > d_{10} \}$
  - Wahl eines Wertes  $d_{11}$ , sodass  $M_{01}$  unterteilt wird:  
 $M_{12} = \{ s \in M_{01} : s.key[1] \leq d_{11} \}$  und  $M_{13} = \{ s \in M_{01} : s.key[1] > d_{11} \}$
  - Wahl weiterer Werte  $d$ , sodass die resultierenden Teilmengen weiter unterteilt werden bezüglich  $key[2]$ , ...,  $key[k-1]$ ,  $key[0]$ , ... bis jede resultierende Teilmenge nur wenige Elemente enthält

# *k Dimensionen*

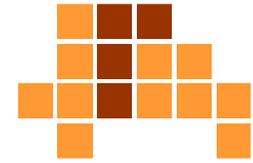


Jeder Knoten unterteilt eine Teilmenge der ursprünglichen Menge in zwei ungefähr gleichgroße Teilmengen bezüglich eines Schlüssels  $\text{key}[i]$

$d_{20}$  unterteilt  $M_{10}$  in  $M_{20}$  und  $M_{21}$  mit  
 $M_{20} = \{s \in M_{10} : s.\text{key}[2] \leq d_{20}\}$  und  
 $M_{21} = \{s \in M_{10} : s.\text{key}[2] > d_{20}\}$  und

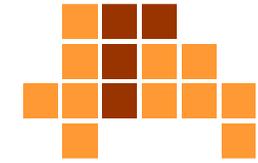


# Anwendung

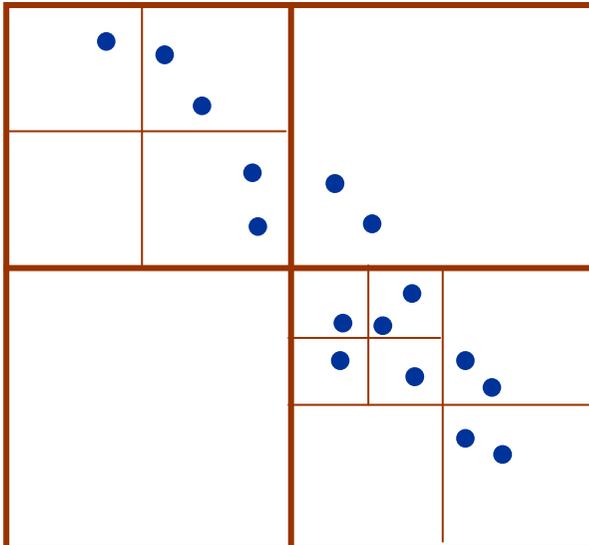


- Suche von k-dimensionalen Punkten in einem vorgegebenen, durch k Intervalle beschriebenen Bereich
- Bereichsanfragen in Datenbanken
  - Datensatz { Name, Postleitzahl, Alter, Einkommen }
  - Suche alle Datensätze, die einem vorgegebenen Postleitzahl-, Alters- und Einkommensbereich entsprechen

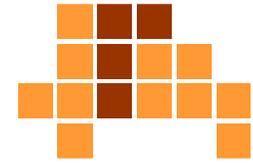
# Quadrees (2D) / Octrees (3D)



- Blattsuchbaum
- Knoten beschreiben Bereiche. Blätter enthalten Schlüssel.
- Wurzelknoten beschreibt Bereich aller Schlüssel.
- Jeder weitere Knoten beschreibt einen Teilbereich. Alle Schlüssel unter diesem Knoten liegen in dem Teilbereich.

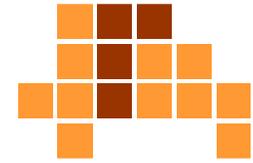


# *k-d Baum / Quadtree / Octree*



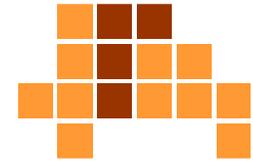
- adaptiv in Bezug auf die Verteilung der Schlüssel
  - große Zellen in Bereichen mit wenig Schlüsseln
  - kleine Zellen in Bereichen mit vielen Schlüsseln
- Einfüge-Operation
  - Zellen mit vielen Schlüsseln können weiter unterteilt werden.

# Überblick

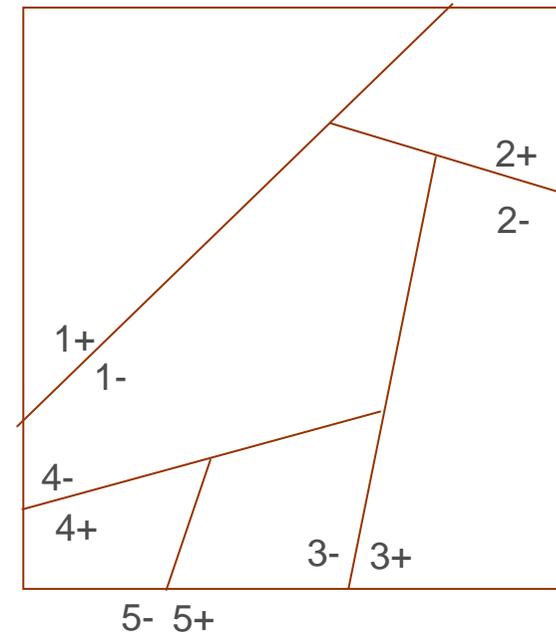


- Einführung
- k-d Baum
- BSP Baum
- R Baum

# Binary Space Partitioning Tree BSP



- Blattsuchbaum
- Raum wird durch beliebig orientierte Hyperebenen unterteilt (Generalisierung eines k-d Baums)
- Ebenen werden in Knoten gespeichert
- Blätter enthalten Schlüssel
- Knoten unterteilen Raum in konvexe Zellen, in welchen die in den Blättern gespeicherten Schlüssel liegen.



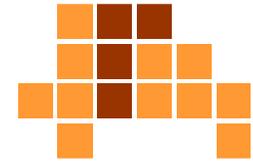
2D-Beispiel:

Gerade 1 unterteilt die Ebene in zwei Teile.

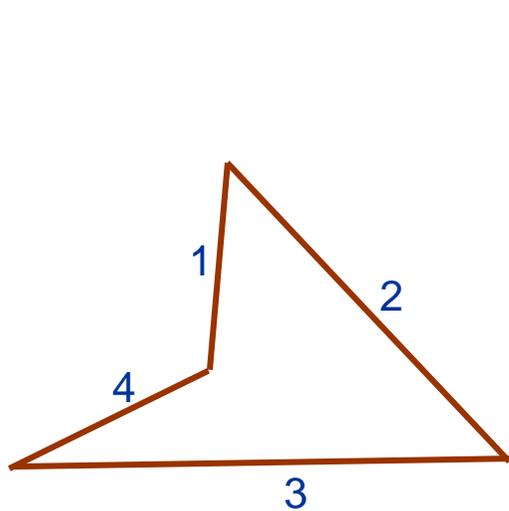
Gerade 2 unterteilt den Bereich 1- weiter in 2+ und 2-.

Gerade 3 unterteilt 2- weiter usw.

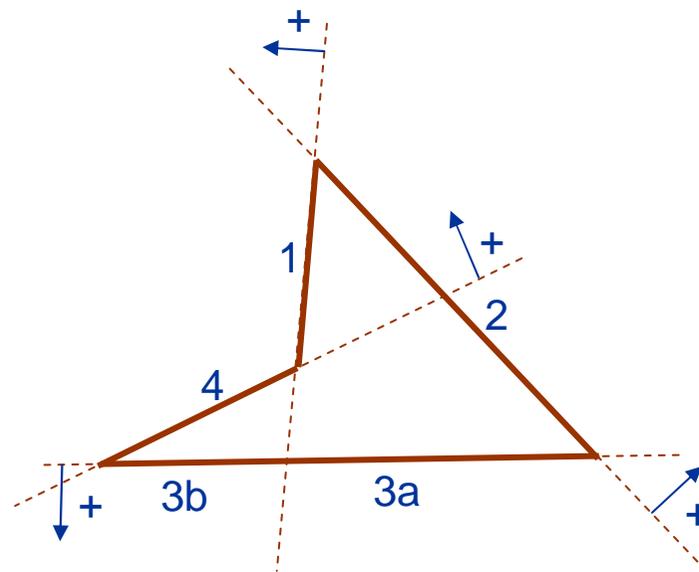
# Anwendung



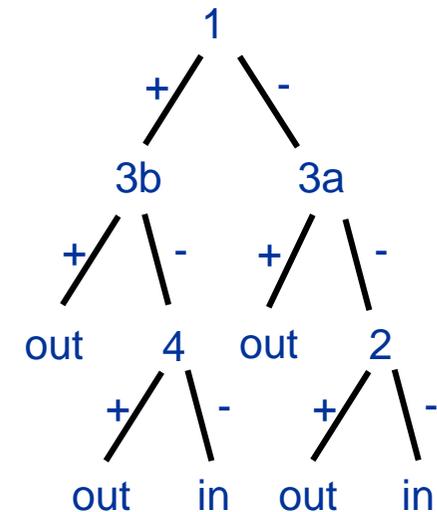
- Test, ob ein Punkt innerhalb oder außerhalb eines geschlossenen Polygons liegt



Szene

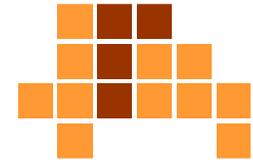


Raumunterteilung

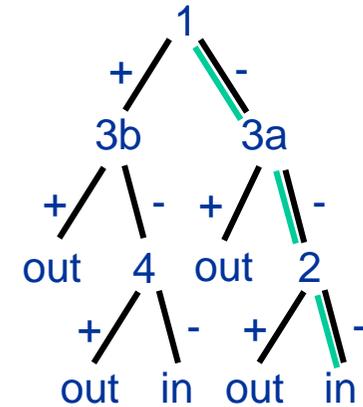
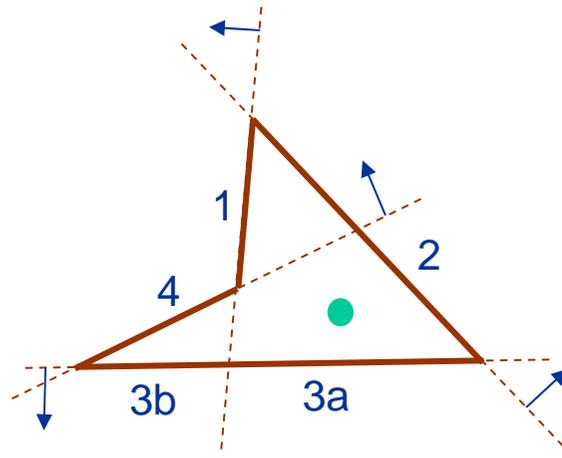


BSP Baum

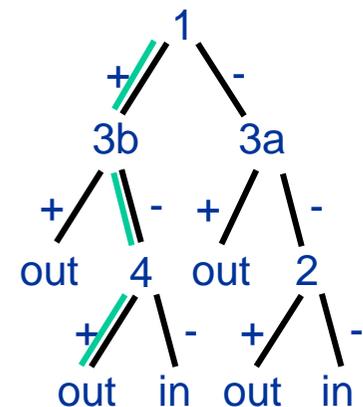
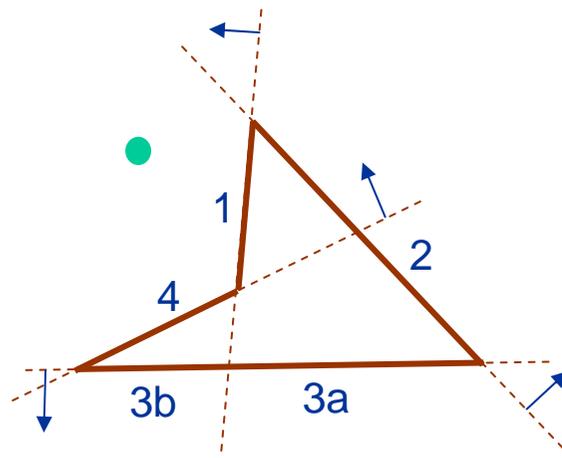
# Anwendung



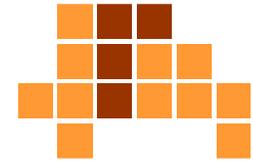
- Anfragepunkt liegt im Polygon



- Anfragepunkt liegt außerhalb des Polygons

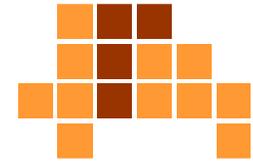


# *Aufbau des BSP Baums*

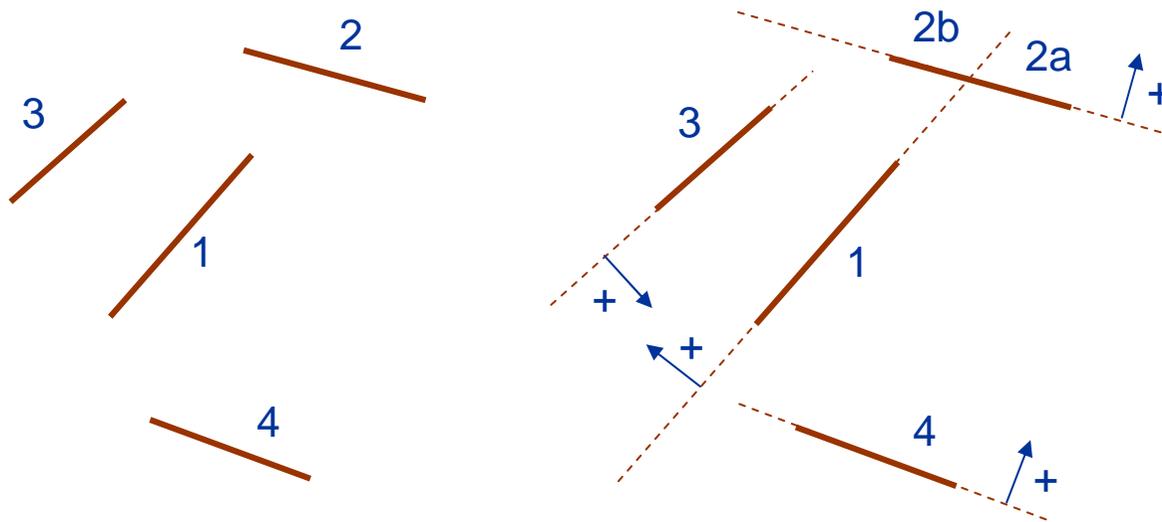


- möglichst geringe Zahl von Knoten
- möglichst geringe Baumhöhe
- schlechtester Fall
  - konvexe Polygone (bezüglich einer Kante liegen alle anderen Kanten immer in der gleichen Halbebene)
  - Generierung von Hilfskanten zur Balancierung des Baums möglich

# Anwendung

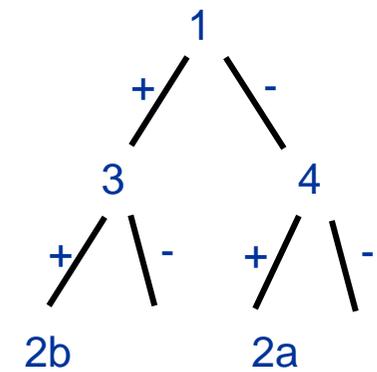


- Lösen des Verdeckungsproblems in 2D



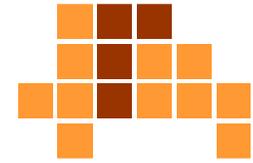
Szene

Raumunterteilung

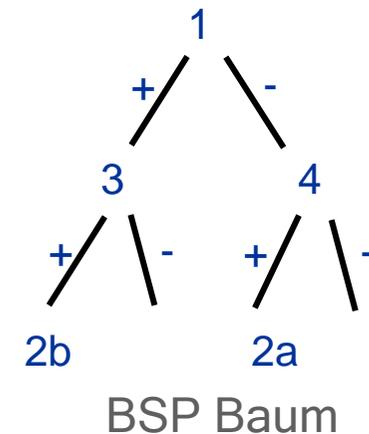
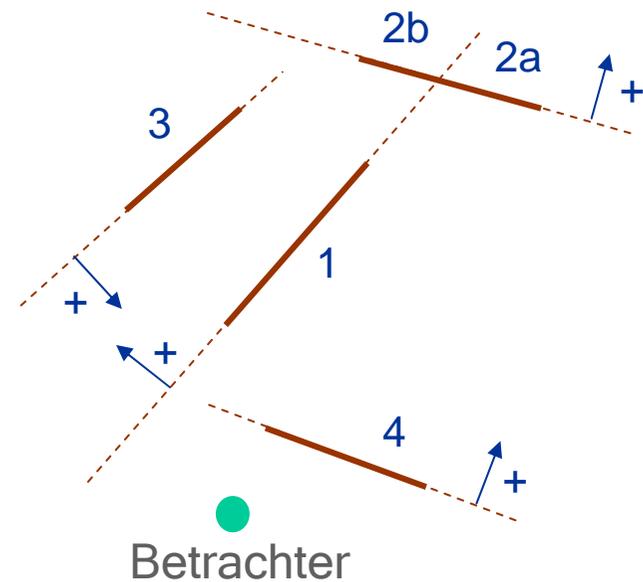


BSP Baum

# Anwendung

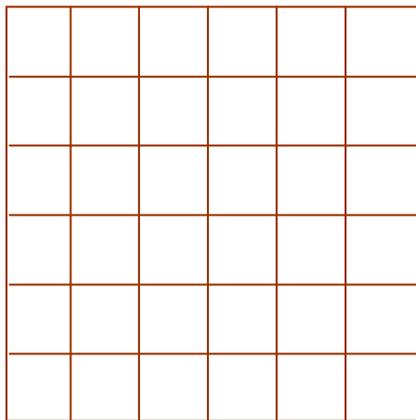
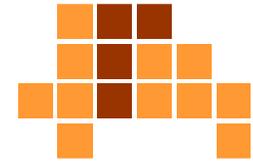


- Betrachterposition ist gegeben
  - zeichne entfernten Teilbaum
  - zeichne Wurzel
  - zeichne Teilbaum des Betrachters
- Regel wird rekursiv angewendet
- ermöglicht back-to-front rendering
- Beispiel: Betrachter ist in 1-
  - zeichne 1+, 1, 1-
  - rekursive Anwendung auf 1+ and 1-
  - Betrachter ist in 3+ → zeichne 3, 2b
  - Betrachter ist in 4- → zeichne 2a, 4

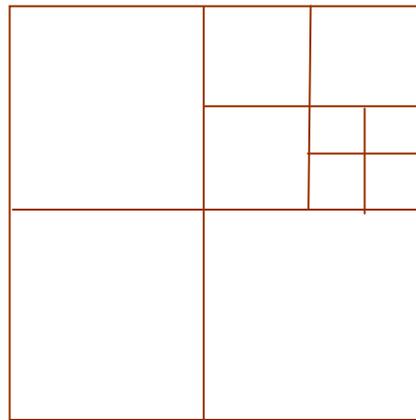


# Zusammenhang

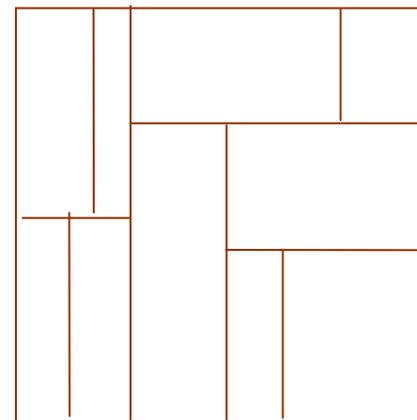
## *k-d Baum, Quadtree, BSP Baum*



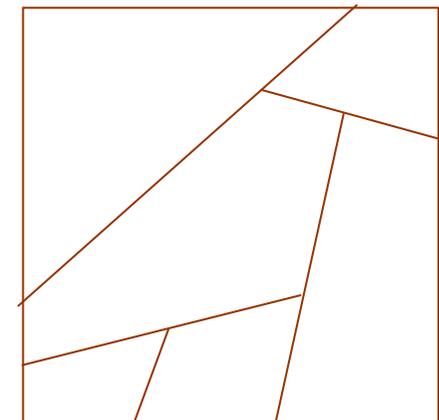
uniformes  
Gitter



Quadtree/Octree



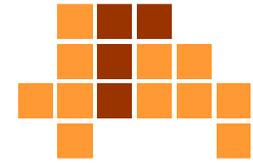
k-d Baum



BSP Baum

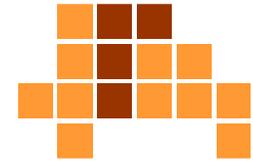
- mehrdimensionaler Raum wird in Bereiche unterteilt
- Knoten beschreiben Bereiche, Blätter enthalten Schlüssel
- unterschiedliche Zahl von Freiheitsgraden
- Bereichsunterteilung ist der Schlüsselverteilung angepasst

# Überblick



- Einführung
- k-d Baum
- BSP Baum
- R Baum

# Anwendung

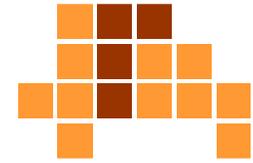


- Simulationsumgebungen, z. B. Robotik
  - Objektflächen werden durch Polygone (Dreiecke) repräsentiert
  - Verfahren zur Kollisionsdetektion stellen fest, ob Objekte sich durchdringen (kollidieren)

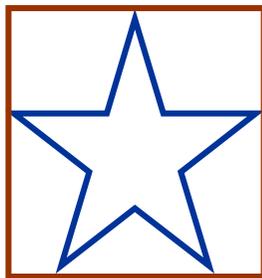


Latombe, Stanford

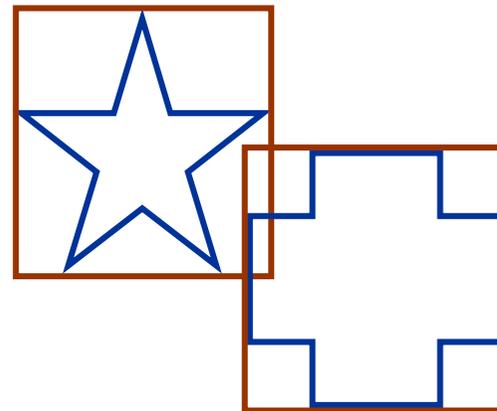
# Motivation



- Kollisionsdetektionsaufwand für  $n$  Polygone ist in  $O(n^2)$
- Bereichsbäume können den Test beschleunigen
- wenn zwei Bereiche nicht überlappen, können die darin enthaltenen Objekte nicht kollidieren

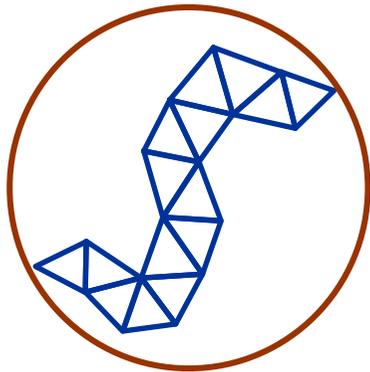
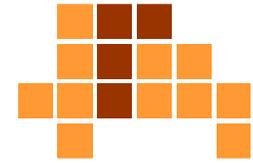


Bereiche überlappen nicht  
→ keine Kollision

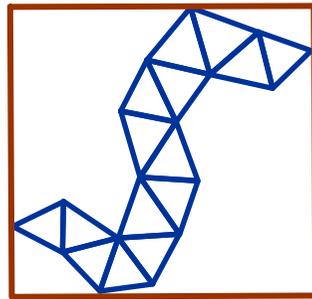


Bereiche überlappen  
→ Kollision möglich

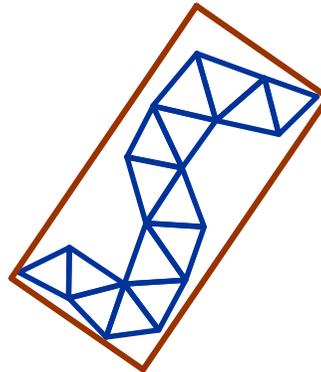
# Bereichsbeschreibung (Typen von Begrenzungsvolumina)



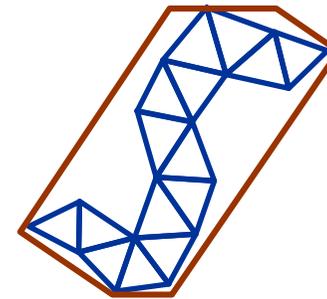
Kugel



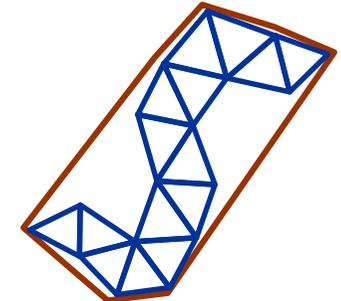
an Hauptachsen  
ausgerichtete Box



an Objekt  
ausgerichtete Box



k-DOP

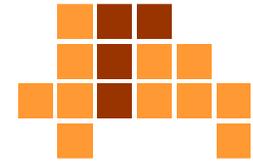


konvexe Hülle

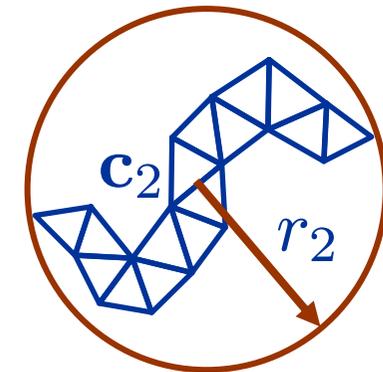
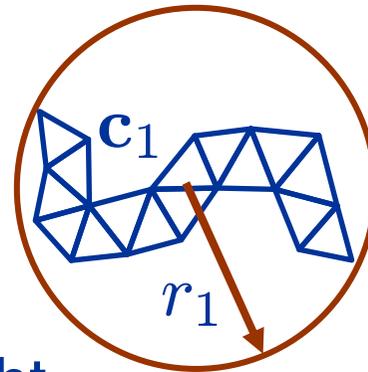
## ■ Auswahlkriterien

- effizienter Überlappungstest
- eng anliegend am Objekt
- effiziente Generierung und Aktualisierung bei Objekttransformation
- speichereffizient

# Kugel



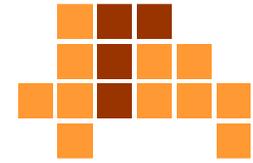
- Kugel sind repräsentiert durch
  - Mittelpunkt  $\mathbf{c}$
  - Radius  $r$



- zwei Kugel überlappen nicht,  
wenn

$$(\mathbf{c}_1 - \mathbf{c}_2)(\mathbf{c}_1 - \mathbf{c}_2) > (r_1 + r_2)^2$$

# Ausgerichtete Quader AABB



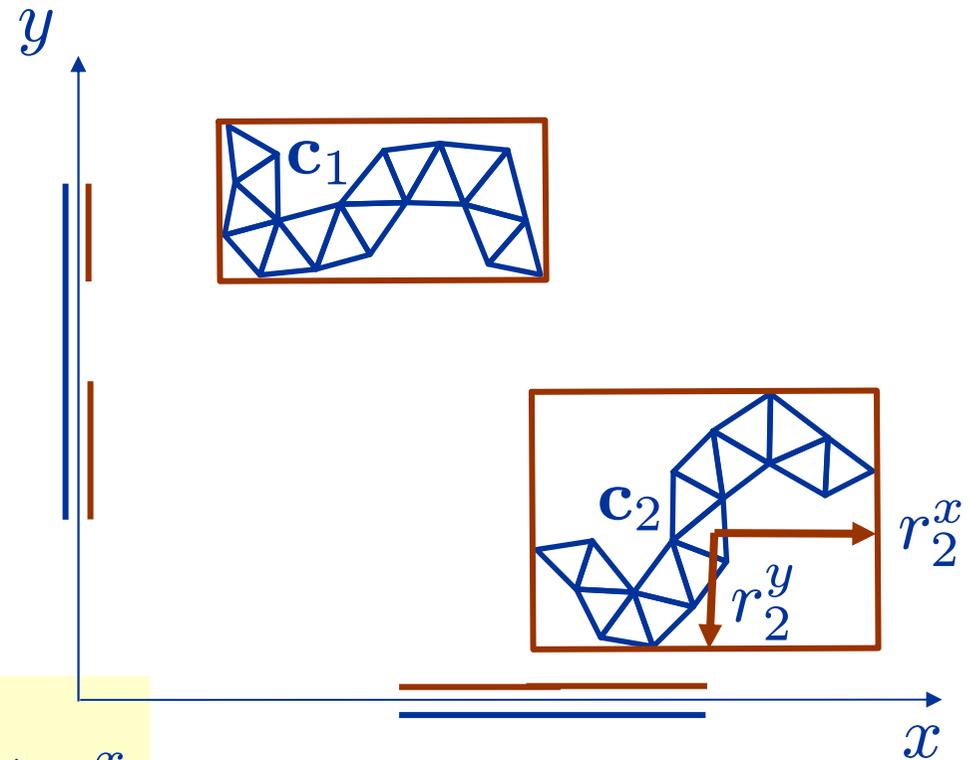
- AABB-Repräsentation

- Mittelpunkt  $\mathbf{c}$
- Radien  $r^x, r^y$

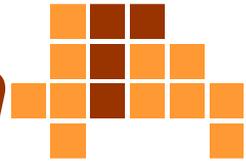
- zwei AABBs in 2D überlappen nicht, wenn

$$\left| (\mathbf{c}_1 - \mathbf{c}_2) \begin{pmatrix} 1 \\ 0 \end{pmatrix} \right| > r_1^x + r_2^x$$

$$\left| (\mathbf{c}_1 - \mathbf{c}_2) \begin{pmatrix} 0 \\ 1 \end{pmatrix} \right| > r_1^y + r_2^y$$



# Optimales Begrenzungsvolumen

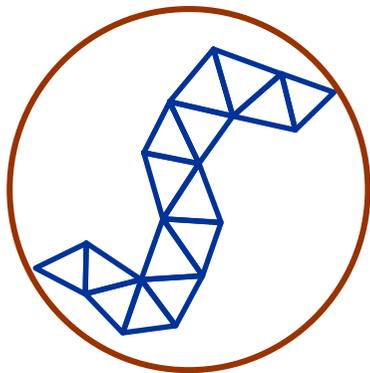


- schwer zu bestimmen

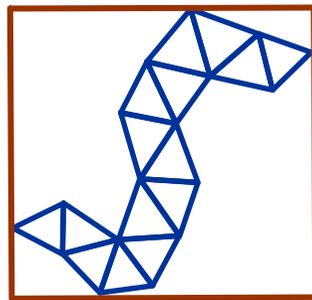
Approximationsqualität



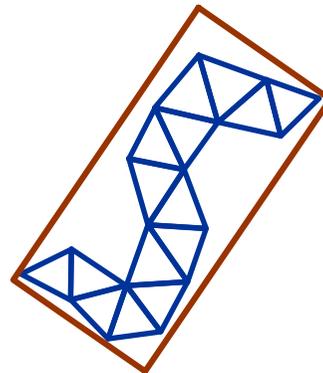
Effizienz des Überlappungs-Tests



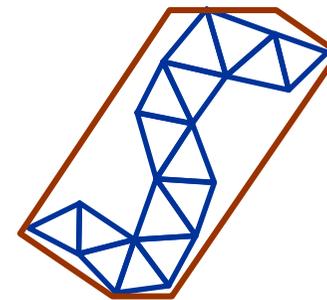
Kugel



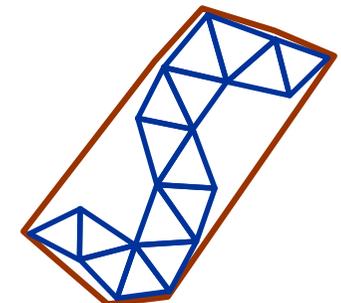
an Hauptachsen  
ausgerichtete Box



an Objekt  
ausgerichtete Box

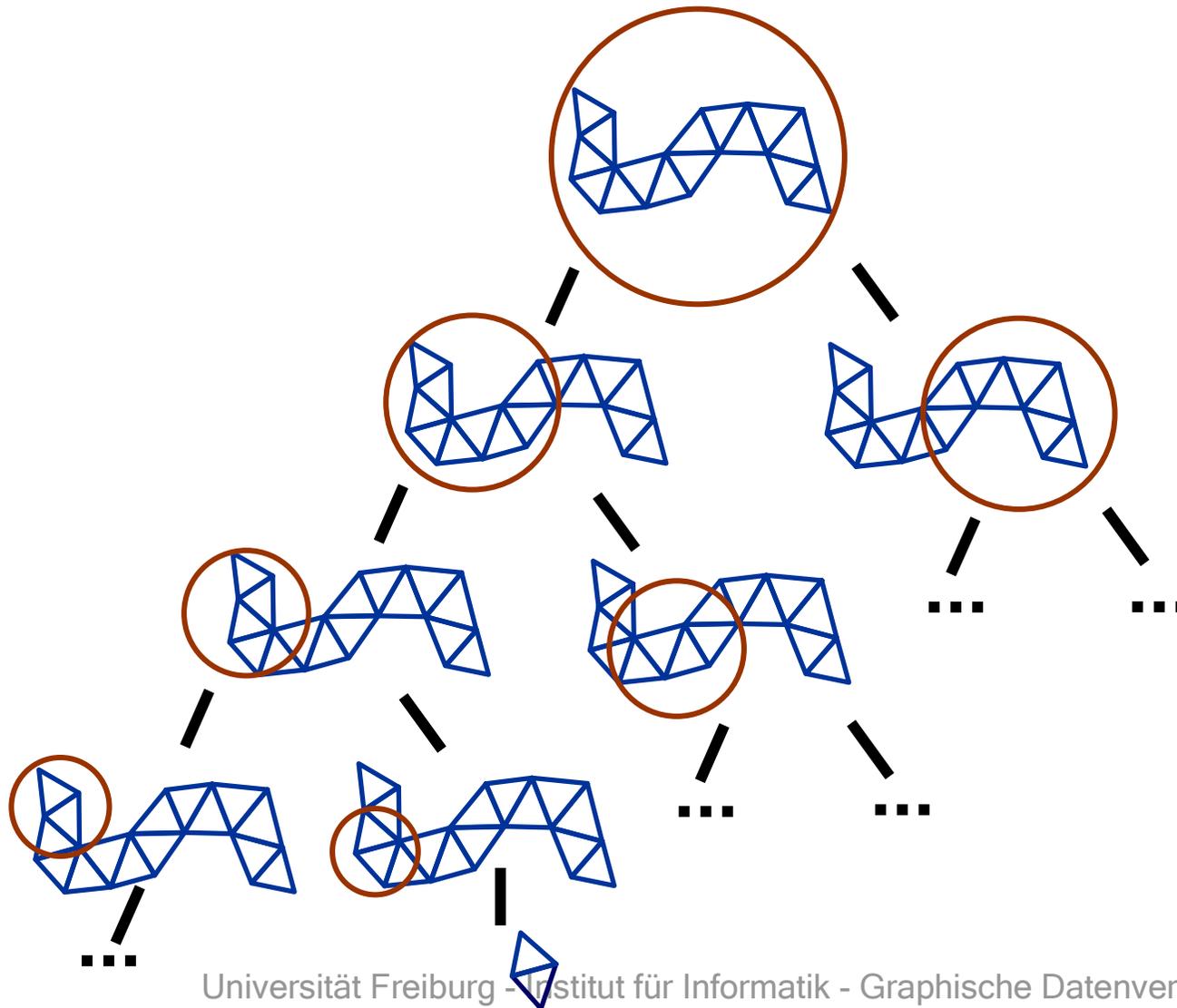


k-DOP

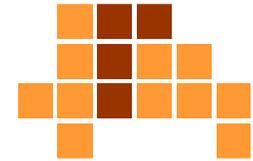


konvexe Hülle

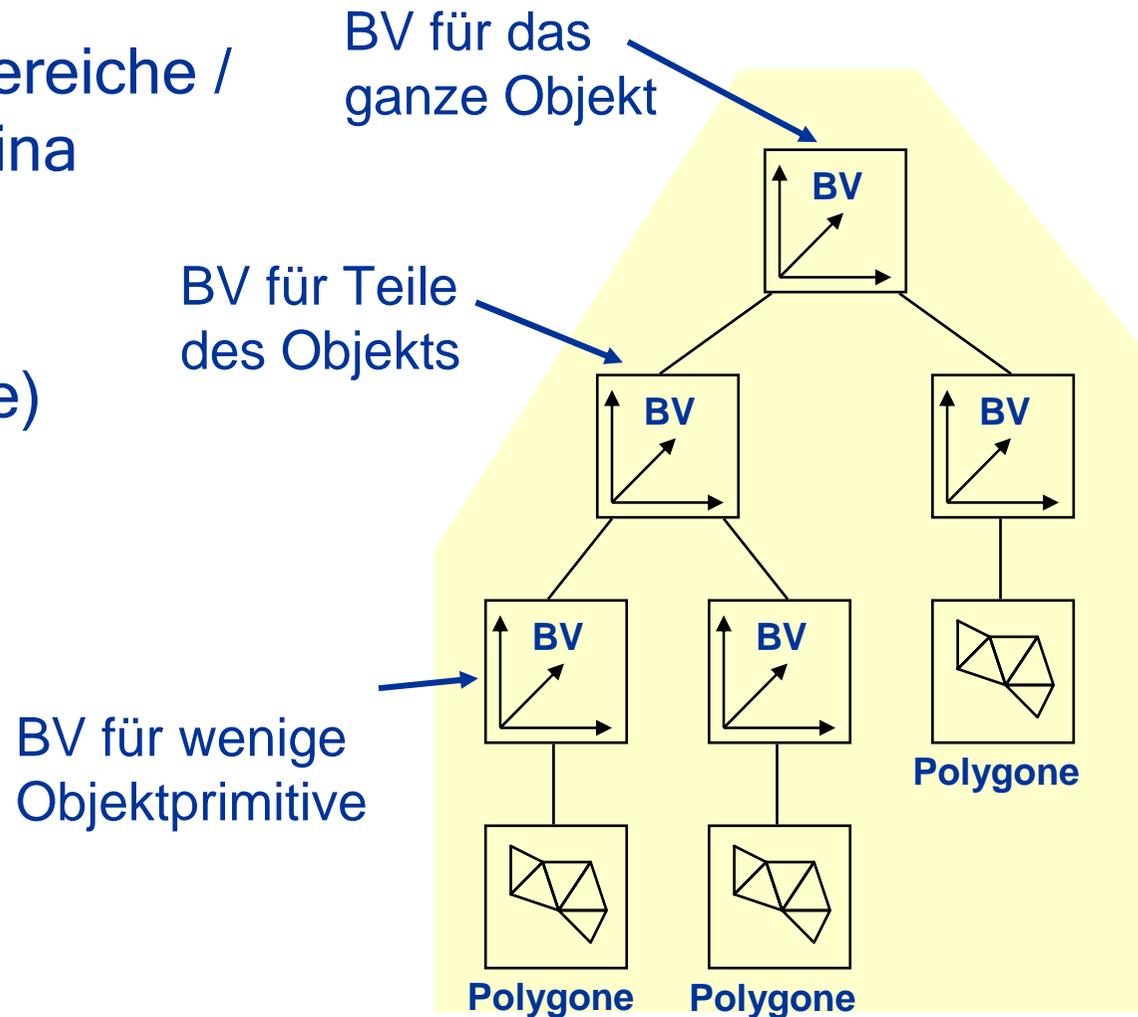
# *Bereichsbaum* *Hierarchie von Begrenzungsvolumina*



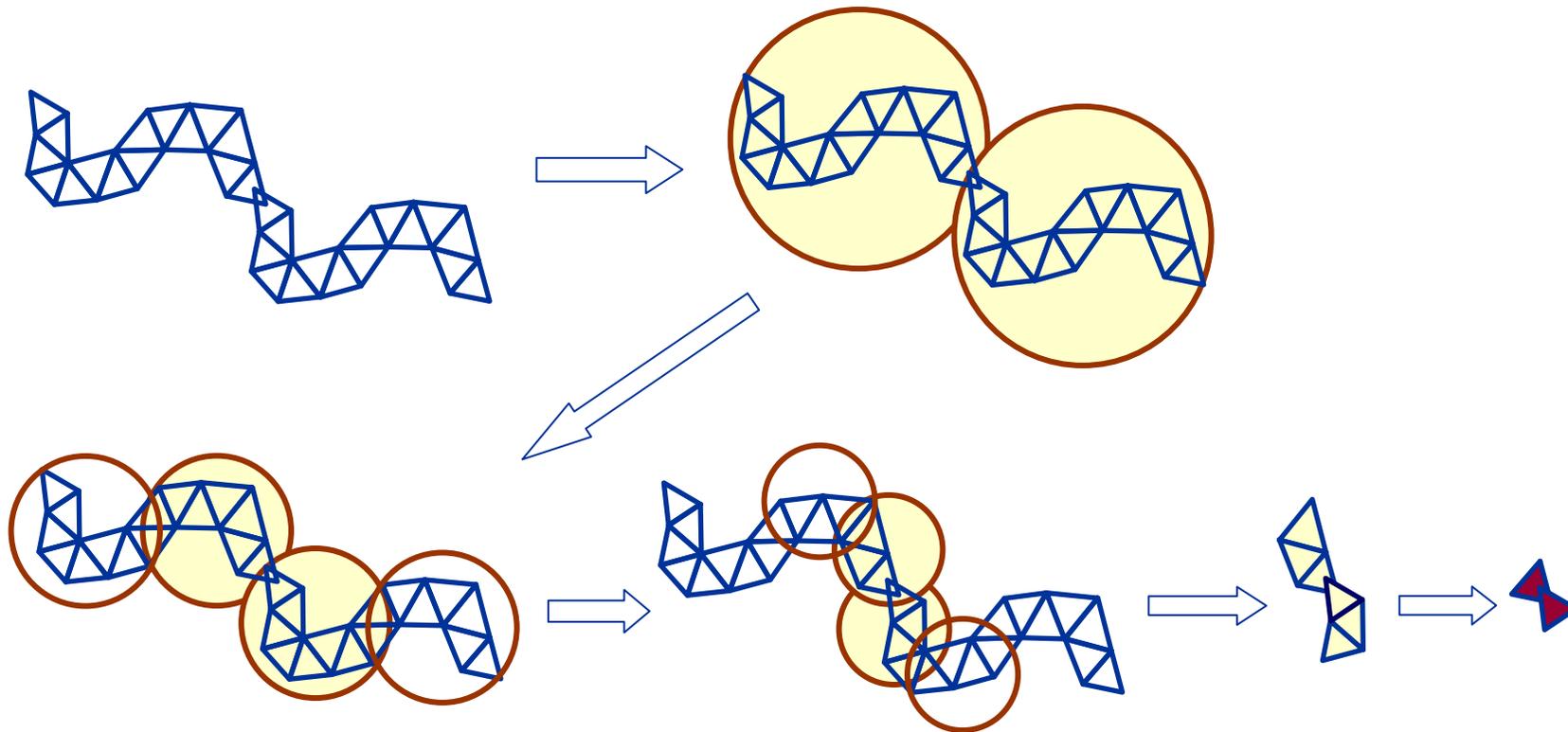
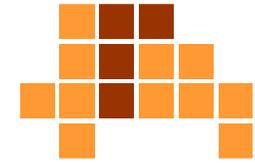
# Datenstruktur



- Knoten enthalten Bereiche / Begrenzungsvolumina
- Blätter enthalten Objektprimitive (Polygone, Dreiecke)

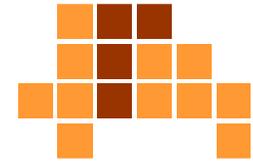


# Überlappungs-Test

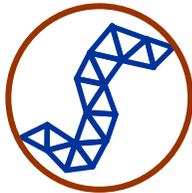


- wenn zwei Bereiche überlappen, werden die Kinder rekursiv getestet
- an Blättern werden Polygone gegen Polygone oder BVs getestet
- effizientes Aussondern von Bereichen, die nicht kollidieren können

# Zusammenfassung



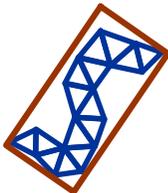
## (1) Begrenzungsvolumen



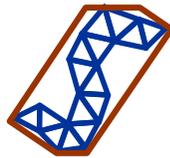
Kugel



AABB

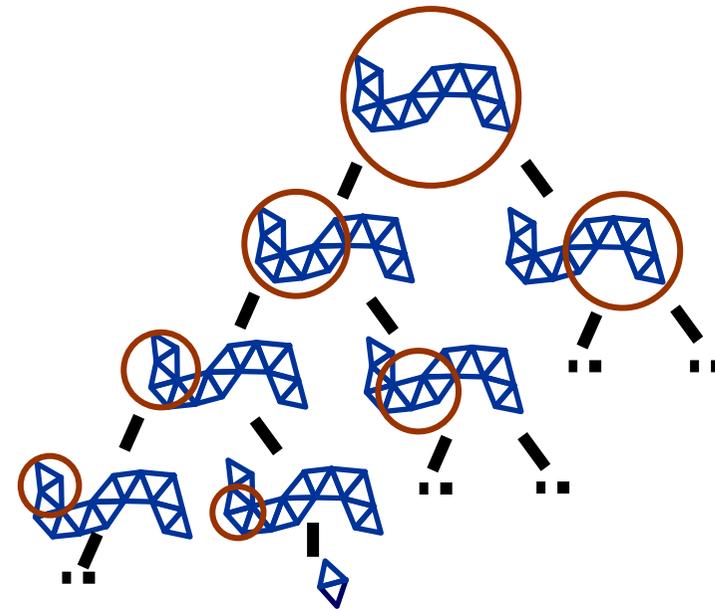


OBB

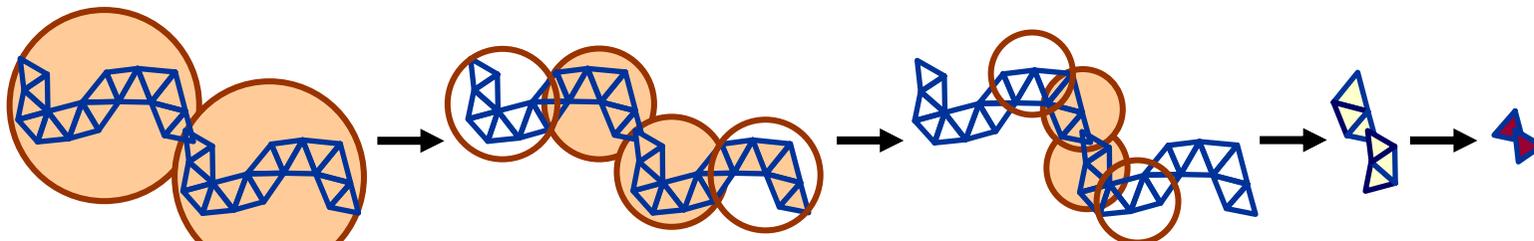


k-DOP

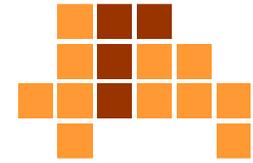
## (2) Bereichsbaum



## (3) Kollisionstest

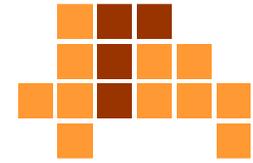


# *Implementierung des Bereichsbaums zur Kollisionsdetektion*



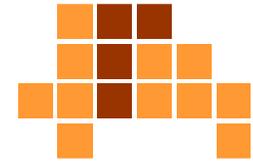
- Ziele
  - balancierter Baum
  - eng anliegende Bereiche / Begrenzungsvolumina
  - minimale Überlappung von Bereichen in einer Ebene des Baums ( $R^*$  - Baum)
- Parameter
  - Typ des Bereichs / Begrenzungsvolumens
  - top-down / bottom-up
  - was wird unterteilt / zusammengeführt: Polygone oder Bereiche
  - wieviel Polygone pro Blatt

# *Implementierung des Begrenzungsvolumens*



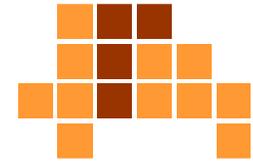
- ausgerichtete Quader (AABB) sind einfach über die Bereichsgrenzen der Polygone bestimmt (xmin, xmax, ymin, ymax)
- aber: AABBs müssen bei Rotation der Polygone neu berechnet werden
- Kugeln sind in der Generierung aufwendiger, können aber bei Rotationen und Translationen des Objekts beibehalten werden.

# Zusammenfassung



- Blattsuchbäume
  - k-d Baum
  - Octree, Quadtree
  - BSP Baum
  - R Baum
- Schlüssel sind in Blättern gespeichert
- erlauben Bereichsanfragen in mehreren Dimensionen
- Anwendungen
  - Datenbanken
  - Rendering
  - Kollisionsdetektion

# *Nächstes Thema*



- Algorithmen / Datenstrukturen
  - Zusammenfassung