Efficient Markov Chain Monte Carlo Methods for Global Illumination Algorithms

Robert Kraus

Master Thesis

Advisor: dr Andrzej Łukaszewski

University of Wrocław
Department of Mathematics and Computer Science
Institute of Computer Science
September 30, 2016
(the slightly refined after-defense version)

Abstract

The point of this thesis is to discuss the usage of algorithms based on ray tracing combined with Monte Carlo and Markov Chain Monte Carlo integration methods for the purpose of solving global illumination problem. The main advantage of MCMC methods is their ability to share information about light found during computations between a number of pixels or let's say ability to focus sampling efforts on important domain regions. The spatial inter-pixel coherency in path space is exploited so the cost of rendering is amortized over all image pixels. We aim to deliver clarified description and refined variants of presented techniques that will help with any programming/engineering work and scientific research, both academic and industrial, involved with these techniques.

The structure of this document is organized in the following manner. In the first chapter we bring an introduction to Computer Graphics and the two major topics, the Global Illumination and Light Transport Simulation. The second chapter describes some of the path space properties. The third chapter brings a theoretical background for Monte Carlo integration along with analysis of its properties in the context of global illumination. The fourth chapter is an introduction to Markov Chains and their role in Monte Carlo integration. In the fifth chapter we present clarified Energy Redistribution Path Tracing algorithm along with experiments and suggestions for further improvements.

Key Words and Related Fields

computer graphics, computational graphics, computational physics, physically based rendering, physically valid rendering, physically accurate rendering, realistic image synthesis, light transport simulation, global illumination, ray tracing, rendering systems, numerical integration, Monte Carlo methods, Markov Chain Monte Carlo methods, Markov Chains.

Efektywne Metody Markov Chain Monte Carlo Dla Algorytmow Globalnego Oswietlenia

Streszczenie

Celem tej pracy jest omowienie algorytmow sledzenia promieni w polaczeniu z metodami Monte Carlo oraz metodami Markov Chain Monte Carlo dla rozwiazywania problemu globalnego oswietlenia. Glowna zaleta metod MCMC jest ich zdolnosc do wspoldzielenia informacji zwiazanej z oswietleniem pomiedzy pikselami lub innymi slowy ich zdolnosc do skupienia wysilku zwiazanego z probkowaniem w najbardziej istotnych obszarach dziedziny calkowania. Koherencja przestrzenna, pomiedzy pikselami, jest wykorzystana w przestrzeni sciezek, dzieki czemu koszt renderingu jest zamortyzowany wzgledem wszystkich pikseli obrazu. Naszym celem jest dostarczenie klarownego opisu i udoskonalonych wersji prezentowanych technik renderingu, ktore moga pomoc w przyszlych pracach inzynieryjnych/programistycznych oraz badaniach naukowych, na poziome akademickim oraz przemyslowym, zwiazanych z tymi technikami.

Struktura dokumentu zorganizowana jest w nastepujacy sposob. Pierwszy rozdzial jest wprowadzeniem do grafiki komputerowej oraz dwoch waznych zagadnien, globalnego oswietlenia i symulacji transportu swiatla. Rozdzial drugi przedstawia niektore wlasciwosci przestrzeni sciezek. Rozdzial trzeci dostarcza teoretycznych podstaw metod Monte Carlo wraz z analiza ich wlasciwosci w kontekscie globalnego oswietlenia. Rozdzial czwarty jest wprowadzeniem do lancuchow Markova oraz ich roli w calkowaniu metodami Monte Carlo. W rozdziale piatym prezentujemy przejrzysty i uporzadkowany opis algorytmu Energy Redistribution Path Tracing wraz z eksperymentami oraz sugestiami dotyczacymi jego udoskonalenia.

Slowa kluczowe

grafika komputerowa, grafika obliczeniowa, fizyka obliczeniowa, rendering bazowany na fizyce, rendering poprawny fizycznie, synteza fotorealistycznego obrazu, symulacja transportu swiatla, globalne oswietlenie, sledzenie promieni, systemy renderingu, calkowanie numeryczne, metody Monte Carlo, metody Markov Chain Monte Carlo. lancuchy Markova.

Contents

1 Introduction

1.1 A Brief History of Rendering in Computer Graphics

Rendering (or forward rendering) is a process in which a picture or a synthetic photograph gets created by performing computational processes on a given digital description of a scene. There is a lot of freedom in terms of geometrical information, a scene can be represented in the form of triangles, polygons, surfels or voxels. Usually these geometric objects are associated with some kind of appearance information, here we can employ shading functions, BxDF functions, color coefficients, textures, etc.

The before mentioned computational processes handle the whole magic behind the image creation, their principles vary wildly and the goals that they try to achieve are also strongly different. Our expectations from a rendered image may be different depending on the context, in some cases we want to get perfect photographic quality of a picture, but on the other hand we may not require a perfectly matching photographic results, instead we want a strongly photographic impression with slightly artistic or unnatural appearance. This may be desired in film production, short movies or some artwork. In some cases we would like to achieve high or constant framerate in order to employ a rendering method into a real-time environment, an obvious example here are computer games. Sometimes we would like to perform offline rendering technique that pushes it hard to the shortest rendering time that is possible to achieve through sharing lighting information between pixels and frames. As we have figured out the challenges we can now think of methods that will let us solve them.

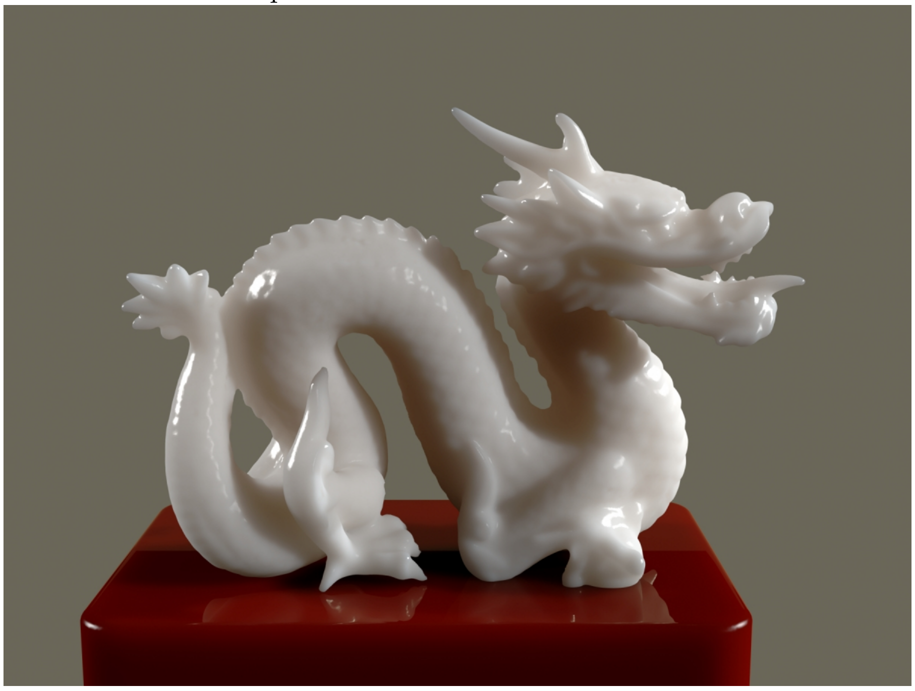
Traditionally, the 3D rendering was performed by employing the so called rasterization algorithms. They rely on discretizing each triangle directly into pixels. Rasterization often works together with shading methods that are appearance-oriented, that means they strive to recreate the visual phenomena on an object surface instead of simulating physical phenomena that will in turn lead to a visual appearance creation. This of course has advantages and disadvantages. The main advantage is the high performance, rendering techniques designed to work with rasteraztion are usually very fast and when accelerated by the hardware they can easily bring high framerates that have critical value for games. On the other hand these methods suffer from hardships in the context of available visual effects and the overall image quality. They may struggle to produce some of photographic or cinematic visual effects in an accurate and consistent manner as the time goes by and scene content changes due to motion and destruction. We refer to these method as appearance-based having in mind that they are often called ad-hoc shading (or lighting) techniques or simply hacks. The latter two pretty much explain their selective nature along with inability to capture the global context of shading and lighting. In some cases it may be hard or even impossible to combine a few such techniques to achieve a complete result that shows added visual richness of all employed methods.

The appearance-based rendering techniques may be very useful in some use cases, as we discussed, but there are situations in which they fail badly. When materials are very complex, when geometry complexity is very high, when scenes are animated, when complex camera models are employed and when photo-realistic quality is desired, these methods may be insufficient. Fortunately they have their counterparts, the physically-based rendering techniques. They employ physical principles and models of phenomena involved with light travel along with numerical methods to bring effective and efficient computational models and algorithms. Examples of computational models are density functions describing material optical properties that can accurately reproduce microscopic phenomena at the macroscopic level and light elements carrying density of some measurable radiometric quantity. Examples of computational algorithms are Monte Carlo methods and Markov Chain Monte Carlo methods. Key concepts that often appear in these methods are light ray, light beam, light path, photon (particle). Some people think that these techniques are brute-force solutions. This comes from the fact that they are usually more computationally demanding. Let's say they are expensive in terms of CPU or GPU cycles, they are often involved with non-trivial memory access patterns which leads to low memory performance through cache eviction, they may also suffer from necessity for clever buffering or caching for temporary results when multi-threading is used. On the other hand, they are very intelligent, they do not apply physical models straightly, meaning they do not process billions of billions of rays or photons in order to reproduce the nature of light in the complete sense. Instead they are based on clever observations and assumptions that bring simplicity. The most fundamental thing here is a concept of steady-state (or stationary) distribution, many physical phenomena that we will want to model or re-create will be expressed by some distribution, often it will be some kind of density function. For a nearly comprehensive coverage of the topic along with rich bibliography we refer reader to the highly valued work, [7] Physically Based Rendering.

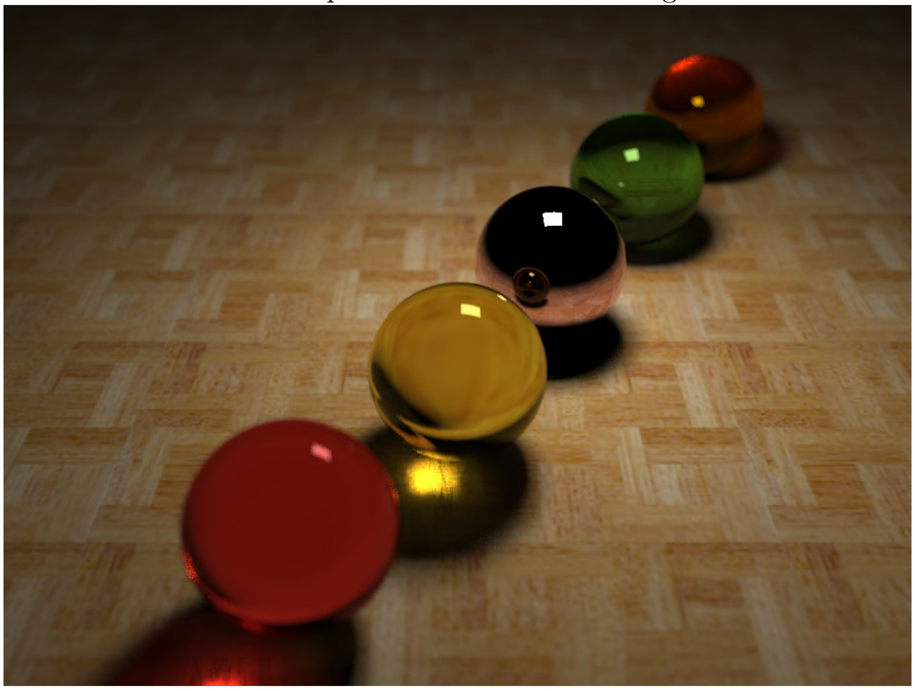
In three-dimensional computer graphics, the rendering process refers to making an image from a given geometric data, including vertices, triangles, polygons, meshes. If an output image is desired to have photo-realistic quality, then rendering is performed by light transfer. Methods that create such synthetic photographs by measuring light transport are collectively known as global illumination algorithms. In a very real sense, the process of global illumination is a physically based computer simulation in which virtual light is traced or propagated through a virtual scene and recorded on a virtual film plane. Optically correct mathematical models of light transport are computationally extremely expensive. From a point of view of a rendering system there is no need to process complete information about light flowing through a virtual scene.

The most versatile global illumination algorithms currently available are based on ray tracing and numerical integration.

An example of scene with translucent materials



An example of scene with caustic light



An example of scene with refractive materials

An example of interor rendering

1.2 Introduction to Global Illumination and Light Transport

1.2.1 The Global Illumination

The illumination in which perceived surface get its appearance from light striking it directly from some light source or set of light emitting objects is called direct illumination. More formally we can say that the direct illumination terms describes lighting that passes through only one reflection or transmission along the way. Majority of the brightness in an image comes exactly from this kind of lighting, but astonishingly majority of visual richness in an image is brought by the indirect illumination. The term 'indirect' simply means that light travelled through at least two reflections before reaching an eye or a camera. So the global illumination is the one in which both direct and indirect parts are being taken into account, and it guarantees that any observable surface will show appearance that is dependent on the entire scene characteristics instead of a local properties of a single material. This kind of lighting brings us all the natural visual effects along with photographic or cinematic quality, hence it is so important for us.

Within the time of last 40 years researchers developed a lot of algorithms based on different principles. Important examples are ray tracing, unidirectional path tracing [6], bidirectional path tracing [10], metropolis light transport [1], energy redistribution path tracing [2], photon mapping [9], radiosity, instant radiosity, many-lights and lightcuts [11], point-based color bleeding [12].

1.2.2 The Light Transport Simulation

The nature of light is very complex and not all of its properties play a role in image creation, therefore applying complete set of physical laws that govern light flow through a scene is definitely not the best solution. As stated previously, they key concept is a steady-state distribution of some measurable quantity. We have to consider two media, space and time, let's say that light flows through volumes or between surfaces and does so over the time. To construct distributions expressed by density functions we often forget about time, we do that by computing our quantities in a 'per unit of time' manner, thus we are usually interested in power instead of energy. The same thing applies to the measurements over space, we construct density functions by defining quantities 'per unit solid angle', 'per unit surface area' or 'per unit of volume'. All that gives us the ability to define quantities that are in some scene area-less, volume-less and time-less. Such quantities allow us to express meaningful values on points and rays (lines), this in turn provides us with the ability to draw a finite and relatively small set of discrete samples, these samples are processed by some magic which in turn leads to the final solution. To put it in other words, what we do is not exactly light transport simulation in its very strict meaning, we do not reconstruct every single event that occurs during light travel, how waves or particles are scattering and interact with matter at the atomic or quantum level. Instead, we try to analytically define stationary distributions of all the relevant properties of light transport and we aim to reconstruct them numerically. So by employing numerical integration over space or space-time we determine what will be on a camera film in terms of color and brightness for every pixel.

2 The Structure and Properties of the Path Space Fabric

The path space depth. This is the quantity that determines how long are the longest paths that contribute to our global illumination solution. Theoretically, for a perfect solution we should set this parameter to infinity. In practice, it often makes no sense because extremely long paths contribute very little to the image, let's say that values such as 5, 10, 20 (or let's say less than 50) are often good enough.

The maximal ray trace depth. It is a threshold that a ray tracing engine must not exceed. It often has the same value as the path space depth.

The lighting deepness. There is a bit of trickery when it comes to defining a ray depth. Light reflected off a mirror forms an imaginary image on the mirror's surface. When dealing with a set of mirrors, we observe virtual images within already virtual images. Here comes a problem, assuming that the lighting deepness is expected to be N, we can observe that every bounce off a mirror within a consecutive/unbroken chain of mirrors drops one lighting layer. In other words, as the number of off-mirror deflections grows the lighting quality gets more shallow. Assuming that the expected lighting deepness equals 4, after 3 off-mirror bounces we will perceive our scene with direct lighting only. Well, we could increase the maximal ray trace depth to some level to overcome this issue, but this would lead to excessive rendering times, as some rays would be constructing very long light paths, rays mentioned here are those not involved with mirrors seen straightly from the camera. To resolve this problem we introduce concepts of physical and virtual ray depth. The physical ray depth is simply the total number of deflections performed, and this is the quantity that is checked against the maximal ray trace depth. The virtual ray depth is the total number of deflections performed since the last primordiality state, its a state in which ray has been shot from a camera or bounced off a first mirror chain.

Now let's take a look at different path types with respect to a basic type system known as Heckbert's notation. Let's recall this notation, the D symbol denotes highly diffusive node, the G symbol denotes glossy node, the S symbol denotes a generally specular or mirror-like node, the T symbol denotes a transparent surface, the L symbol denotes a light source and the E symbol denotes an eye. These come together with the set of operators, $+$ denoting 1 or more repetitions, $*$ denoting 0 or more repetitions. Let's see some examples.

- EDD^+L : a path contributing to the color bleeding effect
- $ED(D|G)^+L$: a more general form of a path contributing to the color bleeding effect
- $ESDD^+L$: a path contributing to the color bleeding seen through a mirror
- ES^+DD^+L : a path contributing to the color bleeding seen through a chain of mirrors
- $ETTDD^+L$: a path contributing to the color bleeding seen through a glass
- $EDSL$: a path contributing to the caustic effect created by a single curved mirror
- EDS^+L : a path contributing to the caustic effect created by a chain of mirrors (flat or curved)
- $EDTTL$: a path contributing to the caustic effect made by a single transparent object
- $ED(TT)^+L$: a path contributing to the caustic effect made by multiple transmissions/refractions
- $ED(T|S)^+L$: a path creating the caustic effect through any mixture of mirrors and glasses
- $ETDTL$: a path shows matte surface that is observed and illuminated through a glass
- $ESTDTD^*L$: the same as above, but seen through a mirror and involved with complete bleeding
- EL : a path with no reflections, light seen directly by an eye

3 Monte Carlo Integration for Global Illumination Setting

This section gives an overview of sampling ideas in the context of basic (conventional) Monte Carlo methods, leading up to a construction of standard Monte Carlo path tracer. We present the concepts of spectrum-valued functions, ray paths, path space, absolute and relative path density, random variables (estimators), importance sampling, and implicit and explicit light paths. We also introduce a hierarchy of type classes related with different scattering modes (light flow events).

3.1 General Idea of Monte Carlo Integration

Consider the problem of integrating the function f over some domain Ω:

$$\int_{\Omega} f(\bar{x})\,\mathrm{d}\mu(\bar{x}).$$

We place a bar over the x to indicate that it may be a vector rather than just a scalar quantity. Monte Carlo integration solves this integral by creating a random variable X_f with expected value equal to the integral:

$$E[X_f] = \int_{\Omega} f(\bar{x})\,\mathrm{d}\mu(\bar{x}).$$

X_f is constructed starting with a sampling procedure Sp which generates samples from Ω according to some probability distribution, p. To complete X_f, a sample location $\bar{x}$ is chosen using Sp, and $X_f(\bar{x})$ is evaluated

$$X_f(\bar{x}) = \frac{f(\bar{x})}{p(\bar{x})}.$$

This expression forms an unbiased estimator of the integral, which may have a high variance. The usual way to reduce the variance is to average a number of samples taken from X_f. The mean of samples taken from X_f can be thought as of new estimator

$$\overline{X}_f(\{\bar{x}_i\}_{i=1}^{n}) = \frac{1}{n}\sum_{i=1}^{n} X_f(\bar{x}_i).$$

This new estimator has the same expected value as the X_f, but its variance gets lower and lower as the number of samples taken from X_f grows up,

$$\lim_{n\to\infty} \overline{X}_f(\{\bar{x}_i\}_{i=1}^{n}) = \int_{\Omega} f(\bar{x})\,\mathrm{d}\mu(\bar{x}).$$

3.2 Behavioural Properties of Monte Carlo Estimators

Consistency. An estimator is weakly consistent if it satisfies the weak law of large numbers, which states that the sample average converges in probability towards the expected value,

$$\overline{X}_f(\{\bar{x}_i\}_{i=1}^n) \xrightarrow{p} \mu \quad \text{when} \quad n \to \infty.$$

That is to say that for any positive number ε,

$$\lim_{n\to\infty} Pr(|\overline{X}_f(\{\bar{x}_i\}_{i=1}^n) - \mu| < \varepsilon) = 1.$$

Interpreting this result, the weak law essentially states that for any non-zero margin specified, no matter how small, with a sufficiently large sample there will be a very high probability that the average of the observations will be close to the expected value, that is, within the margin. Convergence in probability is also called weak convergence of random variables. This version is called the weak law because random variables may converge weakly (in probability) as above without converging strongly (almost surely) as below.

An estimator is strongly consistent if it satisfies the strong law of large numbers, which states that the sample average converges almost surely to the expected value

$$\overline{X}_f(\{\bar{x}_i\}_{i=1}^n) \xrightarrow{a.s.} \mu.$$

That is,

$$Pr(\lim_{n\to\infty} \overline{X}_f(\{\bar{x}_i\}_{i=1}^n) = \mu) = 1 \quad \text{when} \quad n \to \infty.$$

This law justifies the intuitive interpretation of the expected value of a random variable as the "long-term average when sampling repeatedly". Almost sure convergence is also called strong convergence of random variables. This version is called the strong law because random variables which converge strongly (almost surely) are guaranteed to converge weakly (in probability). The strong law implies the weak law.

Biased versus Unbiased. Let θ be an estimated quantity, and let $\hat{\theta}$ be an estimator of θ. The bias of $\hat{\theta}$ is defined as

$$B(\hat{\theta}) = E[\hat{\theta}] - \theta.$$

It is the distance between the average of the collection of estimates, and the single quantity being estimated. It also is the expected value of the error, since

$$E[\hat{\theta}] - \theta = E[\hat{\theta} - \theta].$$

The estimator $\hat{\theta}$ is a biased estimator of θ if and only if $B(\hat{\theta}) \neq 0$, and analogously the estimator $\hat{\theta}$ is an unbiased estimator of θ if and only if $B(\hat{\theta}) = 0$. Note that bias is a property of the estimator, not of the estimate. Often, people refer to a "biased estimate" or an "unbiased estimate", but they really are talking about an "estimate from a biased estimator" or an "estimate from an unbiased estimator".

3.3 Ray Paths and Path Space

In a geometrical sense a path is a chain of points striked by rays during scattering events. For a path of length n we have nodes x_i $(0 \leq i \leq n)$ represented by points in $\mathbb{R}^3$. For all paths the node x_0 refers to the eye point (view point or camera location), and for all paths in which light flow occurs the node x_n refers to a point on a light source. All middle points $(0 < i < n)$ refer to the light flow events, such as reflections and transmissions. However, we can have many different types of flow events in a single node, and because of nature of algorithms we will usually have to use only one specified type of flow event at each node. To denote a path of length n we will use the following notation

$$\bar{x}^{[n]} = x_0 : x_1 : \ldots : x_n.$$

Let τ_i be the type of the node x_i, to denote a type of the path $\bar{x}^{[n]}$ we will use similar notation as we used to denote the path itself.

$$\tilde{\tau}^{[n]} = \tau_0 : \tau_1 : \ldots : \tau_n.$$

The type of the path $\bar{x}^{[n]}$ is simply a chain of types related with the nodes. The fact that $\bar{x}^{[n]}$ is the path of the type $\tilde{\tau}^{[n]}$ is expressed as follows

$$\bar{x}^{[n]} :: \tilde{\tau}^{[n]}.$$

There are a few ways to define node types and path types. We begin with description of different classes of node types. The first way to define a type of a single node is to classify its scattering mode as reflection (R) or transmission (T). We will call this class of node types as node types of the first kind, letting $\Lambda_1 = \{R, T\}$ be the set of node types of the first kind. The second way to define a type of a single node is to classify its scattering mode as diffuse (D), glossy (G) or specular (S), calling this class of types as node types of the second kind, and letting $\Lambda_2 = \{D, G, S\}$ be the set of types. Finally, the third way is to mix the two ways just described into one, which results in most detailed class of node types. We have types like diffuse reflection (D_r), glossy reflection (G_r), specular reflection (S_r), diffuse transmission (D_t), glossy transmission (G_t) and specular transmission (S_t), letting $\Lambda_3 = \{D_r, G_r, S_r, D_t, G_t, S_t\}$ be the complete set of node types of the third kind. We also introduce two additional classes, Λ_α and Λ_β. The class Λ_α is a set of types, for each of which the direct and indirect illumination can not (or should not) be computed separately. For example, in a case of mirror-like glossy reflection (imagine a tiny solid angle around a perfect specular reflection direction) it is very likely that a random direction generated towards a randomly chosen light source will be far away from perfect specular reflection direction. On the other hand, there are types for which the direct and indirect illumination treated separately can be desired, for example a perfect diffuse reflection or a glossy reflection with a wide solid angle. These types belong to the Λ_β class. Both the Λ_α and Λ_β classes are subsets of Λ_3 .

As said before path type is a chain of node types. It is very important that there is no need to keep all node types in this chain in a single class type. This gives us freedom in describing path types. There will be two types associated with a single light path. The strong type $\tau_\bullet^{[n]} = \tau_1^\bullet : \tau_1^\bullet : \ldots : \tau_n^\bullet$ and the weak type $\tau_\circ^{[n]} = \tau_1^\circ : \tau_1^\circ : \ldots : \tau_n^\circ$. The strong type refers to concrete scattering modes chosen by a sampling procedure to generate and evaluate a path. The weak type refers to how we look at a path.

Let $\mathcal{S}$ be a scene (a part of an instance of global illumination problem), and let $\mathcal{C}$ be a camera. Let $\mathcal{P}^{\infty}(\mathcal{S},\mathcal{C})$ be a set of all possible light paths in $\mathcal{S}$ we want to consider. Most important paths are those connecting camera to a light source (directly or through multiple scattering events), such that for each i between 0 and $n-1$ nodes x_i and x_{i+1} are mutually visible. We will refer to this kind of paths as valid paths. These paths can either transport light energy or not, due to materials properties (for example reflectivity). In an ideal case we would like to integrate f only over valid paths, but it is usually hard or even impossible to design a sampling procedure that generates only valid paths. Invalid paths are paths that do not connect the camera with a light source. Such a path occurs when a sampling procedure fails to connect the camera with a light source. This may happen if a sampling procedure has not made a connection yet, but it decided to terminate scattering events. It may also happen when created path can not transport light because of an obstacle that makes two consecutive nodes mutually invisible. Note that invalid paths affect only the variance, but still lead to the same final solution at expense of weaker convergence. However, we do not need to include complete set of invalid paths, and we can try to design a sampling procedure that generates invalid paths rarely.

Let $\mathcal{P}^{(i)}(\mathcal{S},\mathcal{C})$ be the subset of $\mathcal{P}^{\infty}(\mathcal{S},\mathcal{C})$ that contains only paths of length i. Let $\mathcal{P}^{(i)}_{\tau,\text{-}}(\mathcal{S},\mathcal{C})$ be the subset of $\mathcal{P}^{(i)}(\mathcal{S},\mathcal{C})$ that contains only paths of type τ. Following equalities describe subdivision of $\mathcal{P}^{(i)}(\mathcal{S},\mathcal{C})$ into disjoint sets with respect to path length (i), path type (τ) and wavelength (λ).

$$\mathcal{P}^{\infty}(\mathcal{S},\mathcal{C}) = \bigcup_{i=1}^{\infty} \mathcal{P}^{[i]}(\mathcal{S},\mathcal{C}) = \bigcup_{i=1}^{\infty} \mathcal{P}^{(i)}(\mathcal{S},\mathcal{C})$$

$$\mathcal{P}^{[k]}(\mathcal{S},\mathcal{C}) = \bigcup_{i=1}^{k} \mathcal{P}^{(i)}(\mathcal{S},\mathcal{C})$$

$$\mathcal{P}^{(i)}(\mathcal{S},\mathcal{C}) = \bigcup_{\tau \in EH^{i-1}(H+L)} \mathcal{P}^{(i)}_{\tau,\text{-}}(\mathcal{S},\mathcal{C})$$

$$\mathcal{P}^{(i)}_{\tau,\text{-}}(\mathcal{S},\mathcal{C}) = \bigcup_{\lambda \in [\lambda_{min},\lambda_{max}]} \mathcal{P}^{(i)}_{\tau,\lambda}(\mathcal{S},\mathcal{C})$$

Light paths connecting the eye point to a light source can be understood as light streams that transfer light energy. The value of each light path refers to the steady-state distribution of light, i.e. fixed amount of light energy per unit time.

3.4 Energy and Spectrum-Valued Functions

In the context of global illumination the value of f refers to energy. However, energy can be represented by a spectrum of light intensity, referring to the range of colors (i.e. real-valued function of wavelength of light) instead by a scalar value. Due to that, f is the functional of the type $\Omega \to (\mathbb{R} \to \mathbb{R})$ and it may not be clear how to integrate such functional.

If we ignore wavelength dependency then we can easily compute the total energy. Instead of integrating spectrum-valued f over the Ω, we can integrate $\int\limits_{\lambda_{min}}^{\lambda_{max}} f(\bar{x})(\lambda)\,\mathrm{d}\lambda$ over the Ω. In the case of continuous spectrum $[\lambda_{min}, \lambda_{max}]$ the total energy e_t can be computed as follows

$$e_t = \int\limits_{\Omega} \int\limits_{\lambda_{min}}^{\lambda_{max}} f(\bar{x})(\lambda)\,\mathrm{d}\lambda\,\mathrm{d}\mu(\bar{x}),$$

and in the case of discrete spectrum $\{\lambda_1, \lambda_2, \ldots, \lambda_n\}$ it can be computed as follows

$$e_t = \int\limits_{\Omega} \sum_{i=1}^{n} f(\bar{x})(\lambda_i)\,\mathrm{d}\mu(\bar{x}).$$

The concept of total energy will be useful for some algorithmic reasons, but it is insufficient for image synthesis. We need the information about the colors. We can define integral of f as follows

$$\int_{\Omega} f(\bar{x})\,\mathrm{d}\mu(\bar{x}) \overset{def}{=} \lambda \xrightarrow{s} \int_{\Omega} f^{\lambda}(\bar{x})\,\mathrm{d}\mu(\bar{x}) \quad \text{where} \quad f^{\lambda}(\bar{x}) = f(\bar{x})(\lambda).$$

In computer graphics we usually deal with discrete spectrum rather than continuous one, therefore we can assume that $\Lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_n\}$. Under this assumption we can replace function s with a vector $\vec{s} = [s(\lambda_1), s(\lambda_2), \ldots, s(\lambda_n)]$, now f becomes the function of the type $\Omega \to \mathbb{R}^n$,

$$f(\bar{x}) = \left[f^{\lambda_1}(\bar{x}), f^{\lambda_2}(\bar{x}), \ldots, f^{\lambda_n}(\bar{x})\right].$$

The definition of f just described lead us to the integral of f given by

$$\int_{\Omega} f(\bar{x})\,\mathrm{d}\mu(\bar{x}) = \left[\int_{\Omega} f^{\lambda_1}(\bar{x})\,\mathrm{d}\mu(\bar{x}),\ \int_{\Omega} f^{\lambda_2}(\bar{x})\,\mathrm{d}\mu(\bar{x}),\ (\ldots),\ \int_{\Omega} f^{\lambda_n}(\bar{x})\,\mathrm{d}\mu(\bar{x})\right],$$

the estimator the integral of f given by

$$X_f(\bar{x}) = \left[\frac{f^{\lambda_1}(\bar{x})}{p^{\lambda_1}(\bar{x})},\ \frac{f^{\lambda_2}(\bar{x})}{p^{\lambda_2}(\bar{x})},\ (\ldots),\ \frac{f^{\lambda_n}(\bar{x})}{p^{\lambda_n}(\bar{x})}\right],$$

the variance reduction by averaging a number of samples taken from X_f given by

$$\frac{1}{n}\sum_{i=1}^{n} X_f(\bar{x}_i) = \left[\frac{1}{n}\sum_{i=1}^{n}\frac{f^{\lambda_1}(\bar{x}_i)}{p^{\lambda_1}(\bar{x}_i)},\ \frac{1}{n}\sum_{i=1}^{n}\frac{f^{\lambda_2}(\bar{x}_i)}{p^{\lambda_2}(\bar{x}_i)},\ (\ldots),\ \frac{1}{n}\sum_{i=1}^{n}\frac{f^{\lambda_n}(\bar{x}_i)}{p^{\lambda_n}(\bar{x}_i)}\right],$$

and the total energy given by

$$e_t(f(\bar{x})) = \sum_{i=1}^{n} f^{\lambda_i}(\bar{x}) \qquad e_t(X_f(\bar{x})) = \sum_{i=1}^{n}\frac{f^{\lambda_i}(\bar{x})}{p^{\lambda_i}(\bar{x})}.$$

Using formulation just described we can draw a single sample $\bar{x}$, and then we can compute each f^{λ_i} using $\bar{x}$ as the sample point. This implies that

$$\forall_{\bar{x}\in\Omega}\, \forall_{i\in\{1,2,\ldots,n\}}\, \forall_{k\in\{1,2,\ldots,n\}}\, p^{\lambda_i}(\bar{x}) = p^{\lambda_k}(\bar{x}).$$

It means that we are using a single probability density function p for all elements of the spectrum, which obviously has a serious drawback - in a general case p is unable to be the best importance function for each element of the spectrum. However, this has also a great advantage. In the context of ray tracing, taking samples from the path space is extremely expensive in comparison to the evaluation cost of the functions f, f^{λ_1}, f^{λ_2}, etc. Therefore usage of the single sample $\bar{x}$ to compute these functions compensates impossibility of designing an independent probability density function for each wavelength.

Let $\vec{\varrho} = [\varrho_1, \varrho_2, (\ldots), \varrho_n]$ be a vector that represents the percentage/fractional energy distribution over the spectrum of wavelengths (Λ). The $\vec{\varrho}$ must satisfy following conditions

- $\sum_{i=1}^{n} \varrho_i = 1$
- $\forall_{i\in\{1,2,\ldots,n\}}\, \varrho_i \in [0,1]$
- $\forall_{i\in\{1,2,\ldots,n\}}\, \varrho_i \cdot e_t = f^{\lambda_i}$.

In the context of the total energy and the fractional energy distribution vector we introduce the $\circledast$-notation, which states that for any spectrum-valued function g, $(\vec{\varrho} \circledast e_t)$ denotes the value of g separated into its total energy e_t and its fractional energy distribution $\vec{\varrho}$.

3.5 Types of Stochastic Noise

The (Markov Chain) Monte Carlo based rendering can suffer from a few types of noise. We present our own noise classification.

Spatial noise. A noise that occurs on surfaces and in volumes due to undersampling. It is perfectly noticeable kind of noise when it comes to observing a surface on which different points or regions suffer error coming from estimator variance. In the context of global illumination mixed with Monte Carlo methods this translates to weak samplers causing high start-up variance and insufficient number of samples. The appearance of this noise is usually grain-like or grit-like, but when Energy Redistribution Path Tracing is employed it gets more smooth but splotch-like look.

Chromatic noise. This kind of noise shows its influence when a scene contains diffractive, dispersive or goniochromatic materials. When such materials are present light flow gets more complicated and it may be sometimes impossible to sample light paths by tracing multi-chromatic rays. In such cases single-chromatic rays take over. Even if we are able to trace multi-chromatic rays, the BxDF functions may have strongly different shapes expressing the underlying reflection distributions. Consequently, any sampling distribution will cause high variance on at least one of spectrum elements.

Temporal noise. Imagine a single frame that was rendered with sampling rate good enough to achieve perceptually pleasant quality. Let's say that noise is reduced to a level in which all color gradients are very smooth and all the details are perfectly clear. Now let's imagine a sequence of such frames that build a movie. Each frame perceived independently will bring that impression of nearly perfect image quality. However, corresponding or similar pixels (in terms of screen space position) will expose varying level of noise that will lead to an annoying flickering or blinking. That is to say that when rendering an animation, the variance reduction requirements are stronger and the sampling rate must be increased or some other trickery must be applied.

Focal noise. When a camera model used by a rendering engine employs any kind of optical system that is able to re-construct focal blur (also known as depth of field) then the complexity of rendering equations increases as the dimensionality of integration problem grows. Focal noise shows up on surfaces being out of focus, the larger the distance between an object and focal plane the higher variance will damage an image.

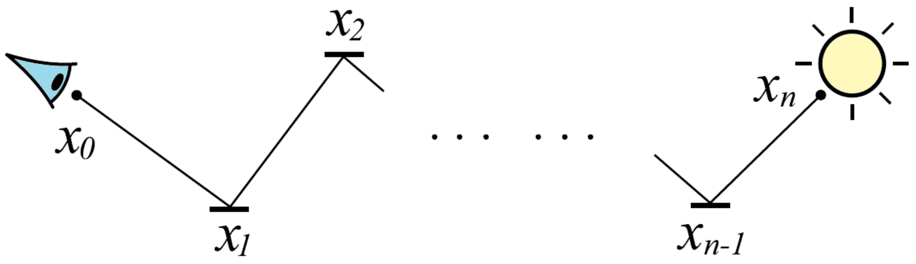
Screen-space noise. This is a noise that comes from estimation errors varying across nearby pixels. Its nature is complex because its a mixture of spatial noise, chromatic noise, temporal noise and focal noise. The appearance characteristics is usually grainy and gritty but may be splotchy as well.

Light-space noise. When path space is sampled very well with respect to geometry but is poorly sampled in terms of light source set then we experience light-space noise. The appearance of this kind of noise is quite tricky because it not grainy, gritty or splotchy. A scene suffering from this noise looks like it were rendered with a small set of point lights, so it may contain banding and large patches.

3.6 Monte Carlo Path Tracing

A path tracer samples the rendering equation by means of ray paths that connect the eye point to a light source through a number of scattering events (reflections or refractions). To build a path, a path tracer sends out a ray from the eye point into the scene. The path tracer then extends the ray through a number of scattering events to produce an eye subpath, using a probabilistic sampling function to choose the outgoing direction at intersection points.

Since the ray paths created by a path tracer are Monte Carlo samples of the rendering equation, the path tracer evaluates them in such a way that the expected value of the paths that contribute to a given pixel is equal to the pixel brightness. To see how this is done, consider the path in figure below that connects the eye point to a light source:



To form an unbiased estimate of the light reaching the eye along direction $x_1 \rightarrow x_0$, the MC sampler in a path tracer multiplies the pertinent terms of the rendering equation together (i.e. $f_*(...)$, $cos(...)$ and $L_e(...)$), and divides by the probability that the path was generated by the sampler.

The path tracer may connect the eye subpath to a light source in one of two ways. First, p_d may happen to choose a direction that hits a light source. We will refer to this kind of path as an implicit path. Second, the path tracer may connect the eye subpath directly to a point on a light source. We will refer to paths created in this way as explicit paths. Explicit paths may lead to the lower variance because they estimate the direct illumination by sampling over the area of all light sources in the scene. Unfortunately, some scattering modes can not be combined with sampling over the area of light sources, e.g. perfect specular reflections, so we need implicit paths too. The MC sampler draws samples from the following equation

$$\begin{aligned} L(\Psi_k \leftarrow x_k) \overset{\text{def}}{=} & \, L_e(\Psi_k \leftarrow x_k) \\ & + \int_{\Omega(x_k)} f_*(\Psi_k \leftrightarrow \Theta_k) L(x_k \leftarrow \Theta_k) \cos(N_k, \Theta_k) \, \mathrm{d}\omega(\Theta_k). \end{aligned}$$

We can rewrite this equation by splitting the integral with respect to scattering modes.

$$
\begin{aligned}
L(\Psi_k \leftarrow x_k) \stackrel{\text{def}}{=} & \, L_e(\Psi_k \leftarrow x_k) \\
& + \sum_{\tau_k^\bullet \in \Lambda_\beta} \int_{\Omega(x_k, \Psi_k, di)} f_*[\tau_k^\bullet](\Psi_k \leftrightarrow \Theta_k) L(x_k \leftarrow \Theta_k) \cos(N_k, \Theta_k) \, \mathrm{d}\omega(\Theta_k) \\
& + \sum_{\tau_k^\bullet \in \Lambda_\beta} \int_{\Omega(x_k, \Psi_k, ii)} f_*[\tau_k^\bullet](\Psi_k \leftrightarrow \Theta_k) L(x_k \leftarrow \Theta_k) \cos(N_k, \Theta_k) \, \mathrm{d}\omega(\Theta_k) \\
& + \sum_{\tau_k^\bullet \in \Lambda_\alpha} \int_{\Omega(x_k, \Psi_k, ti)} f_*[\tau_k^\bullet](\Psi_k \leftrightarrow \Theta_k) L(x_k \leftarrow \Theta_k) \cos(N_k, \Theta_k) \, \mathrm{d}\omega(\Theta_k).
\end{aligned}
$$

Finally, in the case of direct illumination we can switch to the integration over the surface area of light sources as follows

$$
\begin{aligned}
L(\Psi_k \leftarrow x_k) \stackrel{\text{def}}{=} & \, L_e(\Psi_k \leftarrow x_k) \\
& + \sum_{\tau_k^\bullet \in \Lambda_\beta} \int_{A_e} f_*[\tau_k^\bullet](\Psi_k \leftrightarrow \Theta_k) L(x_k \leftarrow \Theta_k) G_V(x_k, x_{k+1}) \, \mathrm{d}A(x_{k+1}) \\
& + \sum_{\tau_k^\bullet \in \Lambda_\beta} \int_{\Omega(x_k, \Psi_k, ii)} f_*[\tau_k^\bullet](\Psi_k \leftrightarrow \Theta_k) L(x_k \leftarrow \Theta_k) \cos(N_k^\Theta, \Theta_k) \, \mathrm{d}\omega(\Theta_k) \\
& + \sum_{\tau_k^\bullet \in \Lambda_\alpha} \int_{\Omega(x_k, \Psi_k, ti)} f_*[\tau_k^\bullet](\Psi_k \leftrightarrow \Theta_k) L(x_k \leftarrow \Theta_k) \cos(N_k^\Theta, \Theta_k) \, \mathrm{d}\omega(\Theta_k),
\end{aligned}
$$

where

$$
V(x_k, x_{k+1}) = \begin{cases} 1 & \text{if } x_k \text{ and } x_{k+1} \text{ are mutually visible} \\ 0 & \text{if } x_k \text{ and } x_{k+1} \text{ are not mutually visible} \end{cases}
$$

$$
G(x_k, x_{k+1}) = \frac{|\cos(N_k^\Theta, \Theta_k) \cos(N_{k+1}^\Psi, \Psi_{k+1})|}{||x_k - x_{k+1}||^2}
$$

$$
G_V(x_k, x_{k+1}) = G(x_k, x_{k+1}) \cdot V(x_k, x_{k+1}).
$$

Now we will introduce few expressions referring to partial evaluation of the estimator. The term Υ_Θ denotes the value of the estimator, limited to the subpath $x_a : x_{a+1} : \ldots : x_{b-1} : x_b$, under assumption that at each node from this subpath sampling over the (hemi)sphere of directions has been performed,

$$
\Upsilon_\Theta(\bar{x}^{[n]}, a, b) = \prod_{k=a}^{b} \frac{f_*[\tau_k^\bullet](\Psi_k \leftrightarrow \Theta_k) |\cos(N_k, \Theta_k)|}{p_s(x_k) p_m(\tau_k^\bullet) \delta_\beta(p_{ii}, \tau_k^\bullet) p_d(\Psi_k \rightarrow \Theta_k)}.
$$

The term $\Upsilon_E(\bar{x}^{[n]})$ refers to terminating the $\bar{x}^{[n]}$ as an explicit path, it includes the value of the estimator of the direct illumination with respect to the sampling over the surface area of all light sources,

$$
\Upsilon_E(\bar{x}^{[n]}) = L_e(\Psi_n \leftarrow x_n) \frac{f_*[\tau_{n-1}^\bullet](\Psi_{n-1} \leftrightarrow \Theta_{n-1}) G_V(x_{n-1}, x_n)}{p_s(x_{n-1}) p_m(\tau_{n-1}^\bullet) p_{di}(\tau_{n-1}^\bullet) p_l(l_k) p_a(x_n)}.
$$

The term $\Upsilon_I(\bar{x}^{[n]})$ refers to terminating the $\bar{x}^{[n]}$ as an implicit path,

$$\Upsilon_I(\bar{x}^{[n]}) = \begin{cases} \dfrac{L_e(\Psi_n \leftarrow x_n)}{p_t(x_n)} & \text{if } \tau_{n-1} \in \Lambda_\alpha \\ 0 & \text{if } \tau_{n-1} \in \Lambda_\beta \end{cases}$$

Using the terms Υ_Θ, Υ_I and Υ_E, the complete estimator can be defined as follows

$$X_f(\bar{x}^{[n]}) = \begin{cases} \Upsilon_I(\bar{x}^{[n]})\Upsilon_\Theta(\bar{x}^{[n]}, 1, n-1) & \text{if } \bar{x}^{[n]} \text{ is an implicit path} \\ \Upsilon_E(\bar{x}^{[n]})\Upsilon_\Theta(\bar{x}^{[n]}, 1, n-2) & \text{if } \bar{x}^{[n]} \text{ is an explicit path,} \end{cases}$$

where

- $L_e(\Psi_k \leftarrow x_k)$ is the emitted radiance leaving the node x_k along the direction Ψ_k
- $f_*[\tau_i^\bullet](\Psi_i \leftrightarrow \Theta_i)$ is the B*DF function value limited to the scattering mode $\tau_i^\bullet$ at the node x_i
- $p_e(x_i)$ is the probability that the MC sampler decided to estimate by light emission, which causes path termination at the node x_i
- $p_s(\tau_i^\bullet)$ is the probability that the MC sampler decided to perform scattering events
- $p_m(\tau_i^\bullet)$ is the probability that the scattering mode $\tau_i^\bullet$ was chosen by the MC sampler
- $p_d(\Psi_i \rightarrow \Theta_i)$ is the probability of generating the outgoing direction Θ_i for the given incoming direction Ψ_i .
- p_{di} is the probability that the MC sampler decided to estimate direct illumination, which causes path termination
- p_{ii} is the probability that the MC sampler decided to estimate indirect illumination, which causes path extension
- $p_l(l_k)$ is the probability that light source l_k was chosen by the MC sampler
- $p_a(x_n)$ is the probability that point x_n was chosen on the light source with respect to surface area.

Now we propose a set of formulas for computing the terms related to probability,

$$p_e(x_i) = \begin{cases} \zeta_e + (1-\zeta_e)\cdot\zeta_a & \text{if } x_i \text{ is emittive} \\ \zeta_a & \text{if } x_i \text{ is not emittive} \end{cases} \tag{1}$$

$$p_s(x_i) = \begin{cases} (1-\zeta_e)\cdot(1-\zeta_a) & \text{if } x_i \text{ is emittive} \\ (1-\zeta_a) & \text{if } x_i \text{ is not emittive} \end{cases} \tag{2}$$

$$p_m(\tau_i^\bullet) = \frac{val(\tau_i^\bullet)}{(1-\zeta_a)} \tag{3}$$

where $\zeta_e \in [0,1]$ stands for an emission coefficient and $\zeta_a \in [0,1]$ stands for an absorption coefficient.

Graphical representation of possible sampling selections.

$$\begin{tikzcd}
 & & L(\Psi_k \leftarrow x_k) \arrow[dll] \arrow[dr] & & & & \\
L_e(\Psi_k \leftarrow x_k) & & & \int_{\Omega(x_k)} \arrow[dl] \arrow[dr] & & & \\
 & \Lambda_\beta & \mathcal{T} \arrow[dl] \arrow[dr] & & \mathcal{T} \arrow[dr] & & \Lambda_\alpha \\
 & \boldsymbol{di} & & \boldsymbol{ii} & & \boldsymbol{ti} &
\end{tikzcd}$$

Absolute Path Density. The product of all of the terms related to probability in a ray path (p_e, p_s, p_m, p_d, p_{di}, p_{ii}, p_l, p_a) can be thought of as the absolute path density in path space with respect to the given MC sampler.
Relative Path Density. If we ignore some of the terms used in the definition of absolute path density then we obtain the relative path density in path space with respect to the given MC sampler. This concept will be useful in the next section, in the context of path mutations. According to mutation strategies, the relative path density is defined as the product of p_m, p_d, p_l and p_a.

3.7 Construction of Monte Carlo Path Tracing algorithm

In our Foxy engine we generally implemented two kinds of MCPT, both based on the knowledge presented in previous sections.

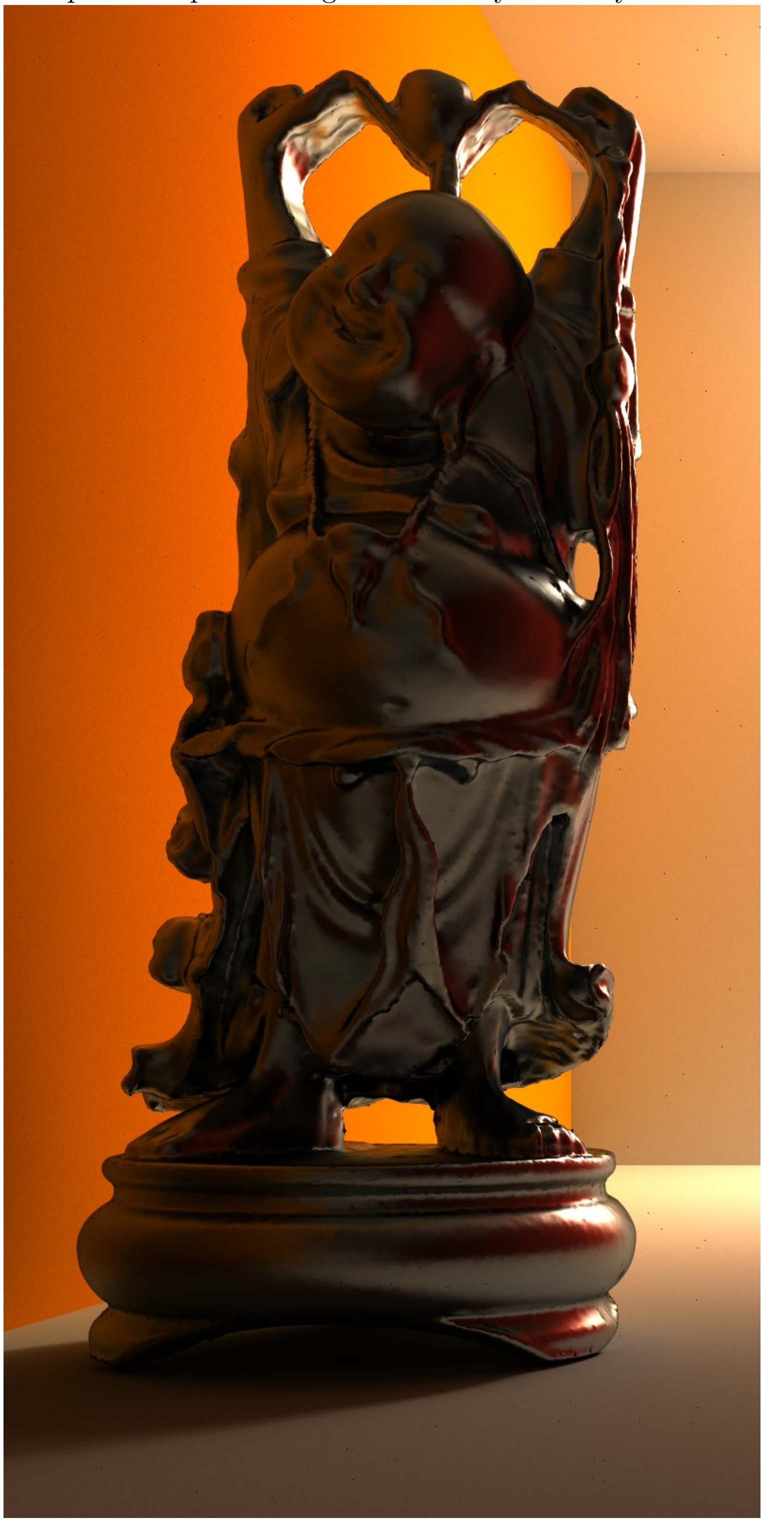
Untangled MCPT. This variant of the path tracer works under assumption that each valid path has only one connection with a light source. So we start at an eye point, we continue to extend a path through a chain of scattering events, at each node we apply selection between extension and termination. In a case of termination the explicit light sampling is performed. This sampler may also hit a light source by an accident. So it can construct both, implicit and explicit paths. This is the sampler that we used as an initial sampler for ERPT.

Entangled MCPT. This variant of the path tracer works in a slightly different manner, at each internal node of a path it performs both, implicit and explicit sampling. So it rather produces a tree containing many entangled paths. It is a better stand-alone technique, compared to the Untangled MCPT, but is less suitable for being an initial sampler for ERPT.

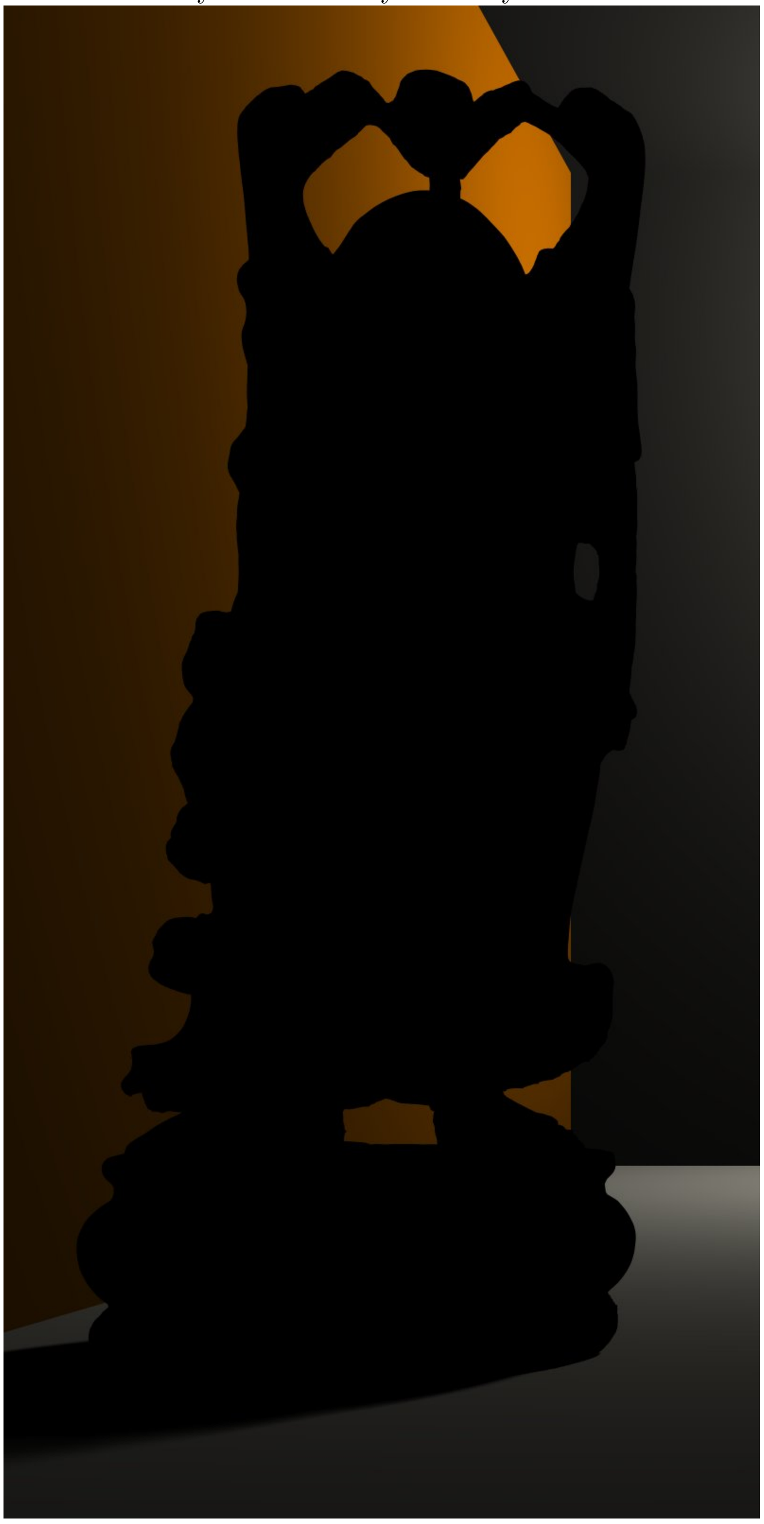
3.8 Empirical Depth-Based Analysis of MC Integration in Path Space

Let's have a look at what is inside each lighting layer, keep in mind that the k^{th} lighting layer is the path space subset made of paths having length equal to $k+1$. Our Foxy engine renders lighting layers into distinct buffers, this allows us to see what is there, how much light information is stored in each layer. Notice that the Buddha model is illuminated from the back side, so the 1^{st} lighting layer shows nearly black silhouette. Longer paths start to connect observable side of the Buddha model with the light source so we can see different lighting effects on his shiny metallic surface.

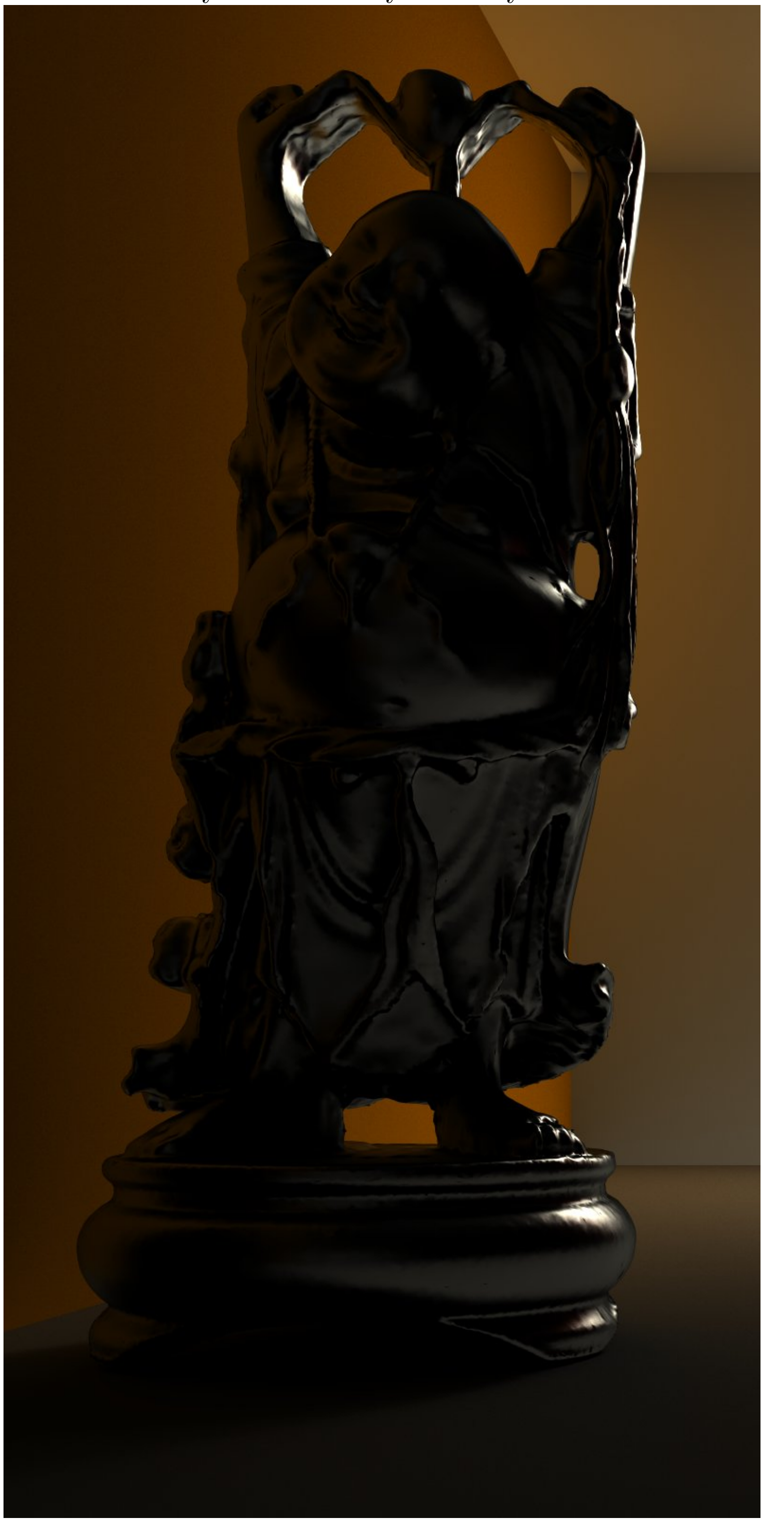
complete composite image rendered by the Foxy's MCPT



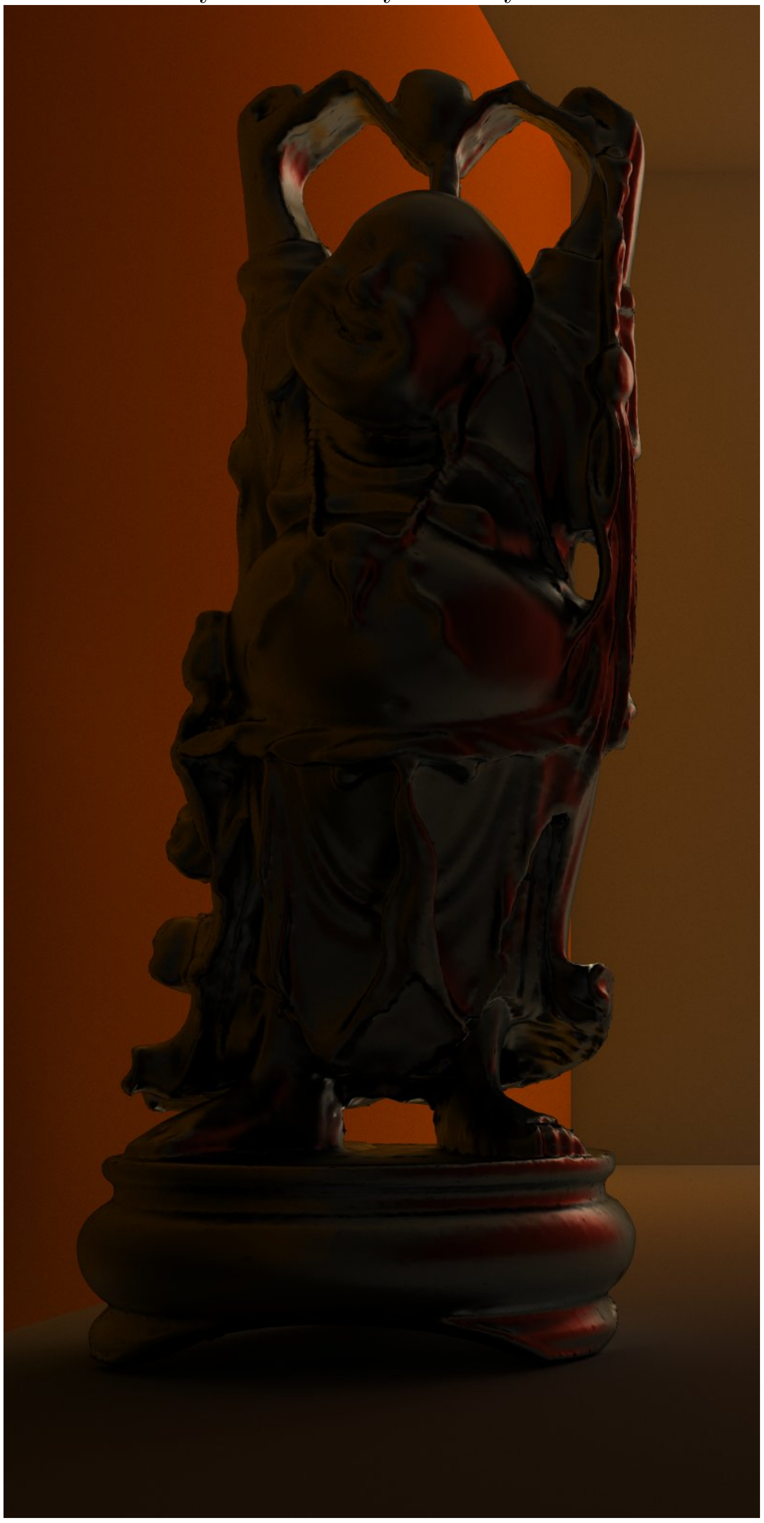
1st layer rendered by the Foxy's MCPT



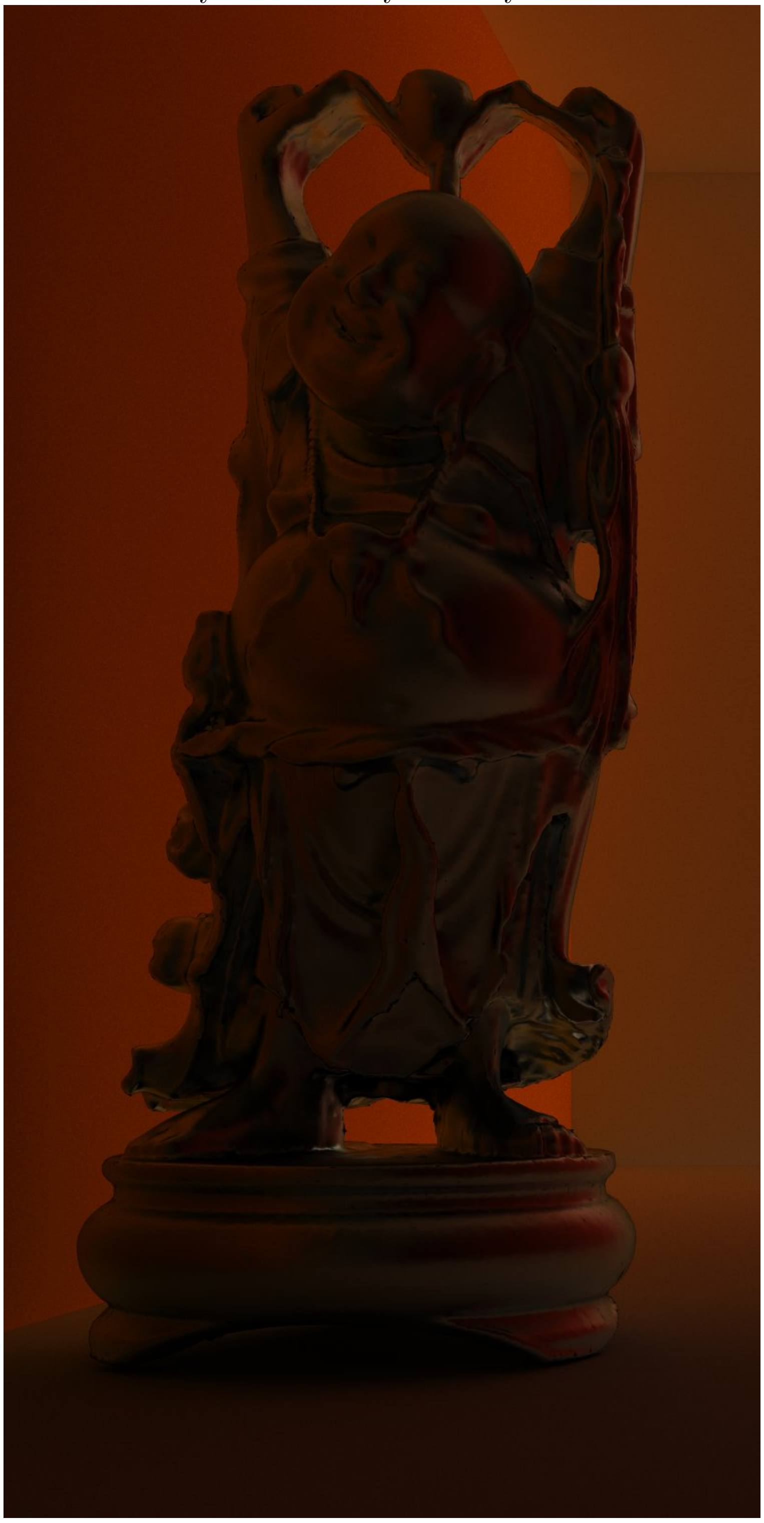
2nd layer rendered by the Foxy's MCPT



3rd layer rendered by the Foxy's MCPT



4th layer rendered by the Foxy's MCPT



The story about lighting deepness and noise in depth-wise layers. When looking at the pictures rendered by Monte Carlo based algorithms we usually perceive some kind of noise. We can say that if an image is not noise-free then our lighting distribution in a scene is undersampled. An important observation can be made here, the variance varies across the path space depth layers, hence the noise appearance varies as well. On some layers noise will be more destructive. Notice that since all layers sum up to a complete image, noises from all the layers sum up to strong noise when at least one layer introduces strong noise.

The very popular strategy employed to reduce variance is importance sampling. It aims to sample more important paths (with higher power content) with higher probability. This is generally a very good technique unless our ability to recognize importance is poor. Well, unfortunately it usually is very poor, comprehensive importance sampling methods exist but are time-consuming while importance sampling methods based on partial information can be misleading. To understand the topic it is good to pay attention to terms that are under the integral symbol inside our rendering equations. There is roughly $BxDF() \times cos() \times L()$ expression, we can easily construct importance samplers based on the $BxDF() \times cos()$ part of the expression, but taking into account $L()$ is a difficult task. So our depth-based distribution of paths is not a perfect solution, but is fairly simple to implement and quite effort-less in terms of computational resources. We believe that it may bring an improvement in some cases and we leave this as a subject for further research. When a majority of lighting in a scene comes from longer paths (strong indirect lighting) we may see benefits.

Let's think about the depth-wise path distribution. But before we do that let's mention some facts. If some lighting layer contains strong noise the final image will be strongly noise-affected. The first lighting layer, made of purely direct lighting, converges rapidly. The deeper the layer the slower its convergence, this is a consequence of the dimensionality growing with lighting depth, it can be understood as a high variance at low sampling rate within a highly multi-dimensional space. Well, it looks like we escaped from the curse of dimensionality by applying Monte Carlo methods instead of classical numerical quadrature, but the issue still exists just in a different form. Another way to look at it is related to floating-point numbers, notice that since precision of such numbers is limited we can express not only discrete but also finite number of rays outgoing from a single point. This in turn translates to an observation which states that deeper lighting layers have more light paths immersed in them, moreover, the number of light paths grows roughly exponentially with the lighting depth. Of course from a point of view of set theory each lighting layer has the same cardinal number, roughly meaning that the number of path in each lighting layer is the same. But from a point of view of numerical integration the information in lighting layers differ in terms of complexity. The limited precision of floating-point number is mentioned here to give us an intuition about the samplable structure of path space and the number of samplable paths that varies across lighting layers.

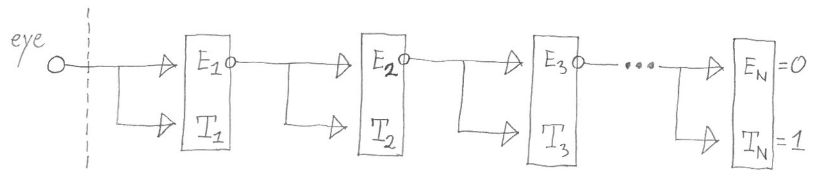
As we discussed, deeper lighting layers contain more complex information about light transport, this increased complexity comes from larger number of light paths. Usually we rate path importance by its absolute radiance with relation to a complete set of paths constructing light transport distribution, but we should ask ourselves a question - when a sampler draws a light path, how does depth-based distribution look like? Or in other words - how big is the probability of drawing a light path from a certain depth layer? If we focus Unidirectional Path Tracing we can control the depth-based distribution by manipulating the termination probability of a path. We experimented with three strategies. Right after the description we provide derivations, luckily, after some time spent on re-expressing and unrolling relevant formulas these distributions has shown nice recursive patterns that we managed to notice and consequently we were able to derive closed form and elegant probability densities.

The BxDF-based distribution employment. This probably the most common method which directly correlates the termination probability with a total reflectivity of a material. This method is an absolute minimum of having an importance sampling applied into MC rendering. This method has a natural tendency to generate less samples in deeper layers. By running MCPT (with explicit sampling employed) with roughly 1000 paths per pixel we can easily see that the first layer is in a perfectly converged state, the second layer will usually start to show some gritty noise and in deeper layers the noise influence grows rapidly.

The equidistribution employment. In this method our goal is to draw equal number of samples within every lighting layer. Well, we tried this distribution just because of curiosity, it is fairly simple and computationally cheap. But it did not bring perceivable changes.

The expodistribution employment. In this method our goal is to draw more samples in deeper layers, forcing the growth of samples to be as close as possible to some exponential distribution. The experiments with this distribution were motivated by the observation telling us that the number of samplable paths grows exponentially with respect to the ordinal number of path space layer. Well, it shows improvement as we expected, but at the same time is too aggressive, it generates too small number of samples in the first layers. We leave this as a subject for future work. We have impression that some kind of re-scaling or weighting constructed on the top of exponentially fading probabilities would help. Also it would be excellent to combine that distribution with classical importance sampling techniques.

The derivation of the equidistribution [1].



The equidistribution derivation:

[1] Notation

- N – the path space depth
- c_{QD} – the equidistribution constant
- E_k – the probability that determines a path extension
- T_k – the probability that determines a path termination

[2] Derivation

$$T_1 = c_{QD} = 1/N \qquad T_{k+1} = 1 \Big/ N \times \prod_{i=1}^{k} E_i$$

$$\forall_{2 \leq k \leq N} \left[\left(\prod_{i=1}^{k-1} E_i \right) \times T_k = c_{QD} \right] \qquad E_k = (1 - T_k)$$

Let $\mathbb{D}(k) = N \times \prod_{i=1}^{k} E_i$, so $T_{k+1} = 1/\mathbb{D}(k)$ and $E_{k+1} = 1 - T_{k+1} = 1 - 1/\mathbb{D}(k) = \frac{\mathbb{D}(k) - 1}{\mathbb{D}(k)}$

$$\mathbb{D}(k+1) = N \times \prod_{i=1}^{k+1} E_i = \left(N \times \prod_{i=1}^{k} E_i \right) \times E_{k+1} = \mathbb{D}(k) \times \frac{\mathbb{D}(k) - 1}{\mathbb{D}(k)} = \mathbb{D}(k) - 1$$

$$\left. \begin{aligned} T_{k+1} &= 1/\mathbb{D}(k) = 1/(\mathbb{D}(k-1) - 1) \\ T_k &= 1/\mathbb{D}(k-1) \\ T_1 &= 1/N \end{aligned} \right\} \quad T_{1+a} = 1 \Big/ \left(N - \sum_{i=1}^{a} 1 \right)$$

$$\Big\downarrow {\scriptstyle k := 1+a,\ a = k-1}$$

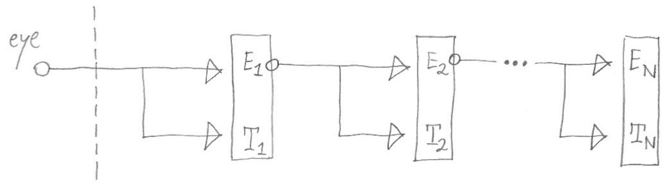
$$T_k = 1 \Big/ \left(N - \sum_{i=1}^{k-1} 1 \right) = 1 \Big/ (N - (k-1)) = 1 \Big/ (N - k + 1)$$

[3] Solution

$$\underline{\underline{T_k = 1/(N - k + 1)}}$$

[1] We are computer graphics enthusiasts that are capable of making fancy handcrafts, drafts and drawings.

The derivation of the expodistribution [2].



The expodistribution derivation:

[1] Notation

- □ N – the path space depth
- □ E_k – the probability that determines a path extension
- □ T_k – the probability that determines a path termination
- □ a_1 – the geometric sequence scaling factor
- □ q – the geometric sequence ratio

[2] Derivation

$$T_1 = a_1 \cdot q^{N-1} \qquad T_k = a_1 \cdot q^{N-k} \Big/ \prod_{i=1}^{k-1} E_i$$

$$\forall_{2 \le k \le N} \left[\prod_{i=1}^{k-1} E_i \times T_k = a_1 \cdot q^{N-k} \right]$$

$$\sum_{k=1}^{N} a_k = \sum_{k=1}^{N} a_1 \cdot q^{k-1} = \frac{a_1(1-q^N)}{1-q} = 1$$

Let $q = 1/2$ so

$$\frac{a_1\left(1-\frac{1}{2^N}\right)}{1-\frac{1}{2}} = 1$$

$$\downarrow$$

$$a_1 = \frac{1}{2} \cdot \frac{2^N}{(2^N-1)}$$

Let $\mathbb{D}(k) = \prod_{i=1}^{k} E_i = \prod_{i=1}^{k-1} E_i \times \left(1 - \frac{a_1 \cdot q^{N-k}}{\prod_{i=1}^{k-1} E_i}\right) = \mathbb{D}(k-1) \times \left(1 - \frac{a_1 \cdot q^{N-k}}{\mathbb{D}(k-1)}\right)$

$$\downarrow$$

$$= \mathbb{D}(k-1) - a_1 \cdot q^{N-k}$$

$$\downarrow \boxed{\mathbb{D}(1) = 1}$$

$$= \mathbb{D}(k) = 1 - \sum_{i=2}^{k} a_1 \cdot q^{N-i}$$

$$T_k = a_1 \cdot q^{N-k} \Big/ \mathbb{D}(k-1)$$

$$\downarrow$$

$$= a_1 \cdot q^{N-k} \Big/ \left(1 - \sum_{i=2}^{k-1} a_1 \cdot q^{N-i}\right)$$

[3] Solution

$$T_k = a_1 \cdot q^{N-k} \Big/ \left(1 - \sum_{i=2}^{k-1} a_1 \cdot q^{N-i}\right)$$

[2]We are computer graphics enthusiasts that are capable of making fancy handcrafts, drafts and drawings.

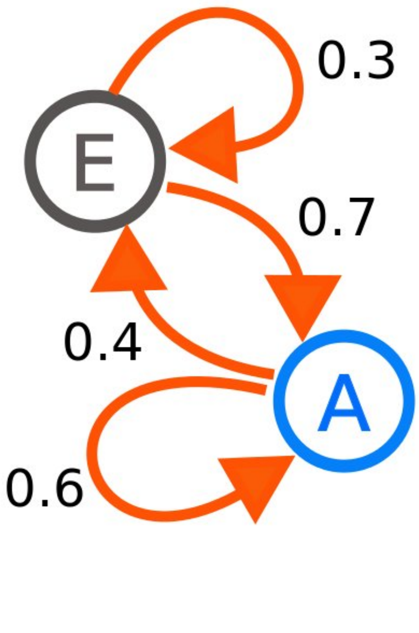
4 Markov Chain Monte Carlo Methods

This section gives an overview of sampling ideas leading up to Energy Redistribution sampling. We give a brief overview of Markov Chain Monte Carlo integration, and present the concepts of correlated integrals, energy flow, and general and detailed balance. Finally, Metropolis sampling is reviewed and compared against Energy Redistribution sampling.

4.1 Introduction

The Markov Chain is a kind of random walk over a set of space locations. Generally, between every two locations A and B we define probabilities, $T(A \rightarrow B)$ - the probability of movement from A to B, and $T(B \rightarrow A)$ - the probability of movement from B to A. When these movements or let's say their transition probabilities are designed carefully, the unique stationary distribution π for the set of space locations will exist. That distribution expresses the steady-state probability of being at certain location. This concept may look completely useless in the context of integration and global illumination, but we will explain why this is a wrong impression. We also strongly recommend to read the [4], it brings a good example of Markov Chain employed to reconstruct/copy an image. To employ this into integration problems such us integral equations for light transport simulation we go in reverse direction. We know how does our best sampling function look like, it is the Holy Grail of Monte Carlo Rendering, the $\frac{f}{\int f}$. We will treat it as the steady-state distribution of visiting paths in the path space and we will try to figure out the movements along with their probabilities that will lead to our desired distribution. Before the promised derivation let's look at an example of simple 2-state Markov Chain.

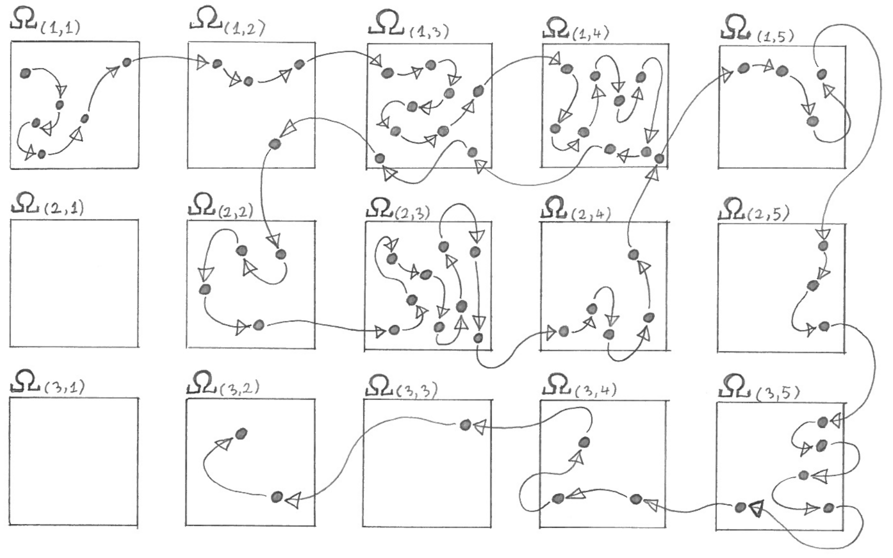
An example of a simple Markov Chain. [3]



[3]The image found on Wikipedia.

Graphical representation of a multi-domain Markov Chain [4].

An example of Markov Chain: 64 steps over 15 domains/pixels



notice that:

[1] some domains are visited more times
[2] some domains are visited less often
[3] there are domains that are not visited at all

[4] We are computer graphics enthusiasts that are capable of making fancy handcrafts, drafts and drawings.

A Markov Chain that would satisfy the detailed balance rule supports the following equality:

$$\pi(x) \times T(x \to y) = \pi(y) \times T(y \to x).$$

Now let's apply our Holy Grail of Monte Carlo rendering, let $\pi(x) = \frac{f(x)}{\int f}$:

$$\frac{f(x)}{\int f} \times T(x \to y) = \frac{f(y)}{\int f} \times T(y \to x).$$

Now we can notice that the $\frac{1}{\int f}$ terms cancel out, this is the step where the wizardry does the job and the proof that our Markov Chain is able to converge to our desired distribution without an explicit knowledge about it:

$$f(x) \times T(x \to y) = f(y) \times T(y \to x).$$

Now it looks like we are stuck, we have one equation but two unknown values, $T(x \to y)$ and $T(y \to x)$, but we will employ more wizardry, we will express the unknown values as a product of yet another unknown values. So let $T(x \to y) = TT(x \to y) \times a(x \to y)$, where TT is our tentative transition, we can design it on our own with a lot of freedom, we have:

$$f(x) \times TT(x \to y) \times a(x \to y) = f(y) \times TT(y \to x) \times a(y \to x).$$

After a little reorganization the equation directly expresses the value of $a(x \to y)$, we end up with:

$$a(x \to y) = \frac{f(y) \times TT(y \to x)}{f(x) \times TT(x \to y)} \times a(y \to x).$$

But we still have two unknown values, we call them acceptance probabilities. We have one equation and two variables, this simply means that the equation may have more than one solution. The formal derivation is beyond the scope of this thesis, but the solution that is often used looks like that:

$$a(x \to y) = min\left[1, \frac{f(y) \times TT(y \to x)}{f(x) \times TT(x \to y)}\right].$$

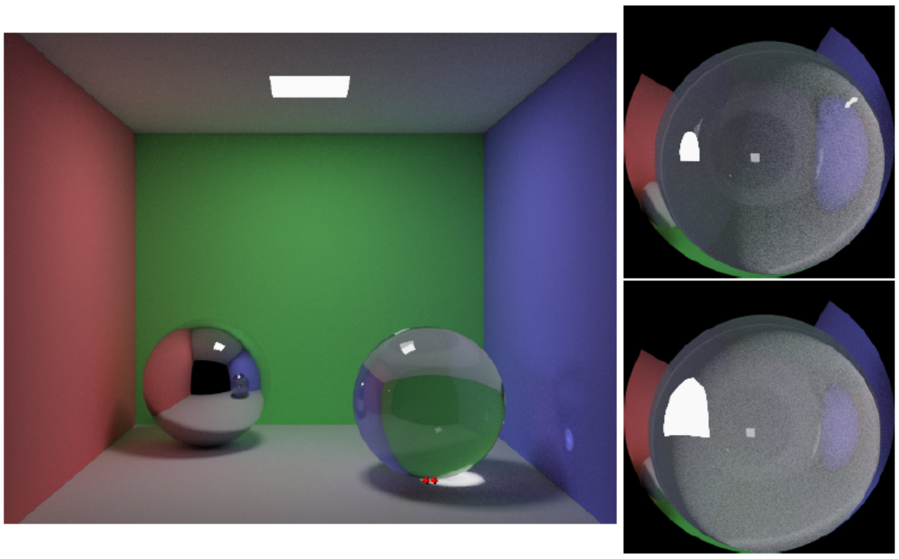
That's it! This formula brings not only an acceptance probability that allows our Markov Chains to work, but it also guarantees the highest acceptance rate across possible solutions.

For some it still may be a bit confusing because in computer graphics we are interested in pixels, we want to have brightness and color in all pixels and here we are talking about some awkward walking or jumping between paths through some network of strange movements with some probabilities. In our opinion a good way to think about it is to assume that each path has a box attached to it. Every time when we visit a path we throw a single coin into the box. Of course there is also a N_{coins} - the total number of coins that we have thrown into the boxes. If we divide the number of coins by the N_{coins} for each box, we will get approximation of the steady-state probability of being in each path. Yet again, what about pixels? Nobody wants to store all the boxes in memory. Now imagine, that we can perform grouping, we can think about groups of paths and the probability of visiting a certain group. The probability of visiting a group is simply a sum of all the probabilities attached to paths being members of a group. Now notice that every pixel is illuminated by a set of paths, these paths form a group with well-defined steady-state probability that is directly proportional to the desired pixel brightness. So our pixels act as bins capturing the probability that is proportional to the incoming radiance. So we can think that instead of throwing coins into boxes attached to paths, we can directly throw coins into pixels. We believe that this delivers a good intuition about the work performed by Markov Chains in the context of image synthesis.

4.2 Correlated Integrals

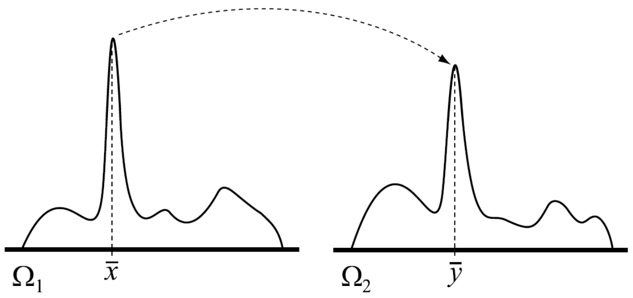
Some of integration problems involve the estimation of not just one, but a large number of integrals. Photorealistic image synthesis is a particularly pertinent example. In the context of Monte Carlo ray tracing each pixel of an image is an integral that is evaluated using Monte Carlo integration. When a standard path tracer is used as a Monte Carlo ray tracer, each pixel is solved by n statistically independent MC samples. Notice that, since samples within a domain of a single integral are statistically independent, samples from different integral domains are also statistically independent. Strictly speaking, each MC sample is generated from scratch, despite the fact that when a high contribution sample (light path) is found by a MC path tracer, it would be better to exploit this information to generate new samples by performing exploration of nearby paths. The most successful correlated integral solutions tend to exploit the correlation between integrals to reduce variance, in order to achieve faster convergence. In fact, the correlation between pixel integrands is the implied basis for many of the global illumination algorithms currently in use, including irradiance caching [Ward et al. 1988], photon mapping [9] and Metropolis Light Transport [Veach and Guibas 1997]. Irradiance caching and photon mapping take adavantage of inter-pixel correlation by caching incident light values, which are later used to approximate parts of the pixel integrals that are difficult to evaluate independently. MLT leverages the correlation between pixel integrals in a different way, using mutation strategies to share integrand information between pixels. We present a similar approach, utilizing path mutations to spread the energy of initial Monte Carlo pixel estimates over the image plane.

A fish-eye view from two points [5].



4.3 Energy Flow

One way to coordinate sampling efforts between correlated integrals is to use a process of energy flow. (By energy, we simply mean the value of a real-valued function. For a color-valued function, such as an image, energy refers to the luminance.) Energy flow allows a sampling procedure to perform a directed search between similar points in the domains of correlated integrals. To see why this can be useful, consider two correlated integrals, I_1 and I_2, with domains Ω_1 and Ω_2. Suppose that in the process of sampling, a high contribution point $\bar{x}$ is found in Ω_1 (i.e. $X_f(\bar{x})$ is large). Since I_1 and I_2 are correlated, it is likely that a high contribution point $\bar{y}$ will exist in a location similar to $\bar{x}$ in Ω_2. Energy flow establishes a connection between points $\bar{x}$ and $\bar{y}$ and transfers some of the energy at $\bar{x}$ to $\bar{y}$. Figure 2 shows this graphically. Often, energy flow can be more efficient than standard Monte Carlo sampling because the cost of finding high contribution points is amortized over multiple integrals.



The expected energy flow. In practice, energy flow is created by perturbing or "mutating" a source point, $\bar{x}$, to produce a destination point, $\bar{y}$. (Imagine laying a pipe from $\bar{x}$ to $\bar{y}$ along which energy can flow.) Some of the energy at $\bar{x}$ is then transferred to $\bar{y}$. Let

[5]The image comes from original publication [2].

$T(\bar{x} \to \bar{y})$ be the transition probability from $\bar{x}$ to $\bar{y}$, that is, the probability that $\bar{y}$ is chosen as the destination point given that $\bar{x}$ is the source point. In this situation, the expected flow from $\bar{x}$ to $\bar{y}$ is

$$E[\phi(\bar{x} \to \bar{y})] = E[X_f(\bar{x})p(\bar{x})T(\bar{x} \to \bar{y})q(\bar{x} \to \bar{y})]$$

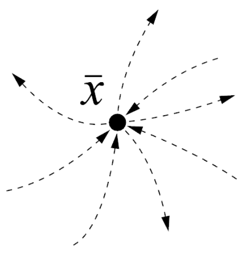
where $\phi(\bar{x} \to \bar{y})$ denotes the energy flow from $\bar{x}$ to $\bar{y}$, $E[\cdot]$ is the expected value, $X_f(\bar{x})p(\bar{x})$ is the expected energy located at $\bar{x}$ from an initial Monte Carlo estimate, and $q(\bar{x} \to \bar{y})$ is the percentage of energy at $\bar{x}$ that flows to $\bar{y}$ once a connection has been established.

General and detailed balance. Astonishingly, energy flow can occur without biasing the integral estimates, as long as certain conditions on the flow amount are met. In particular, the integral estimates will remain unbiased as long as the expected flow of energy out of any point $\bar{x}$ equals the expected flow back in. We will refer to this property as general balance. More formally, we say that general balance holds if

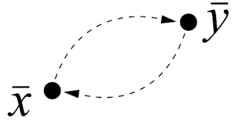
$$\forall_{\bar{x}} \quad E\left[\int \phi(\bar{x} \to \bar{y}) \, \mathrm{d}\mu(\bar{y})\right] = E\left[\int \phi(\bar{y} \to \bar{x}) \, \mathrm{d}\mu(\bar{y})\right]$$

An even stronger constraint that guarantees unbiased-ness is called detailed balance. Detailed balance requires that the expected flow between any two points be equal. In other words,

$$\forall_{\bar{x}}\forall_{\bar{y}} \quad E[\phi(\bar{x} \to \bar{y})] = E[\phi(\bar{y} \to \bar{x})]$$



General balance



Detailed balance

4.4 Review of Metropolis Sampling

We will now shortly review characteristics of Metropolis sampling, pointing out main differences between MLT sampling and ERPT sampling. For more details about the MLT we suggest reading the original work [1]. Generally, classic Metropolis sampling relies on creating a single very long Markov Chain with stationary distribution that is proportional to brightness distribution over pixels. This approach has a few limitations and weaknesses.

Stratification. The stratification is a property that determines pixel coverage quality in our image. Let's recall the MCPT algorithm for a while, in this technique we can easily control the stratification because we are able to manually set the number of MC samples for each pixel. Often, the simplest choice gets employed, which is equal number of MC samples across all the image pixels. While it may not be the optimal solution it guarantees that every pixel will be assigned a reasonably high number of MC samples. In MLT this is not the case, a long Markov Chain can store a lot of information in some pixels, resulting in nearly converged solution, but it can leave a lot of pixels that are sampled poorly or pixels that are not sampled at all. This tells us that the MLT cannot work successfully with short Markov Chains, by definition it should be run with settings promoting very long Markov Chains.

Start-up bias. Despite the fact that Markov Chain eventually converges to a desired distribution first samples within a Markov Chain are generated with some arbitrary start-up distribution that often does not match that desired distribution. This means that a prefix sub-chain of some usually unpredictable length will bring an error to our image. We have to put additional effort in order to overcome the issue. In contrast to the MLT, the ERPT does not have that issue because initial MC samples bring correct estimation with potentially high variance and Markov Chains are only used to reduce that variance.

Importance sampling. In its classic formulation Metropolis Light Transport is unable to take advantage of importance sampling, at least not with the same level of freedom as seen in MCPT. In ERPT we have both benefits, the importance sampling for initial MC samples and Markov Chains bringing the higher order of convergence.

Ergodicity. The Metropolis Light Transport requires comprehensive set of mutations, this roughly means that no matter where a Markov Chain starts, moves or stays it must always be able to reach any point within a domain (let's say a path in path space) in limited number of steps. In other words, there must be no region in a domain from which Markov Chain cannot escape and all regions have to be always approachable for a Markov Chain. Skipping the formal language again, mutation set must guarantee that the unique stationary distribution exists. This issue forces us to deal with more complexity when designing a mutation set. The ERPT does not suffer from this problem because initial MC samples ensure ergodicity, therefore our requirements for properties that mutations must satisfy are relaxed.

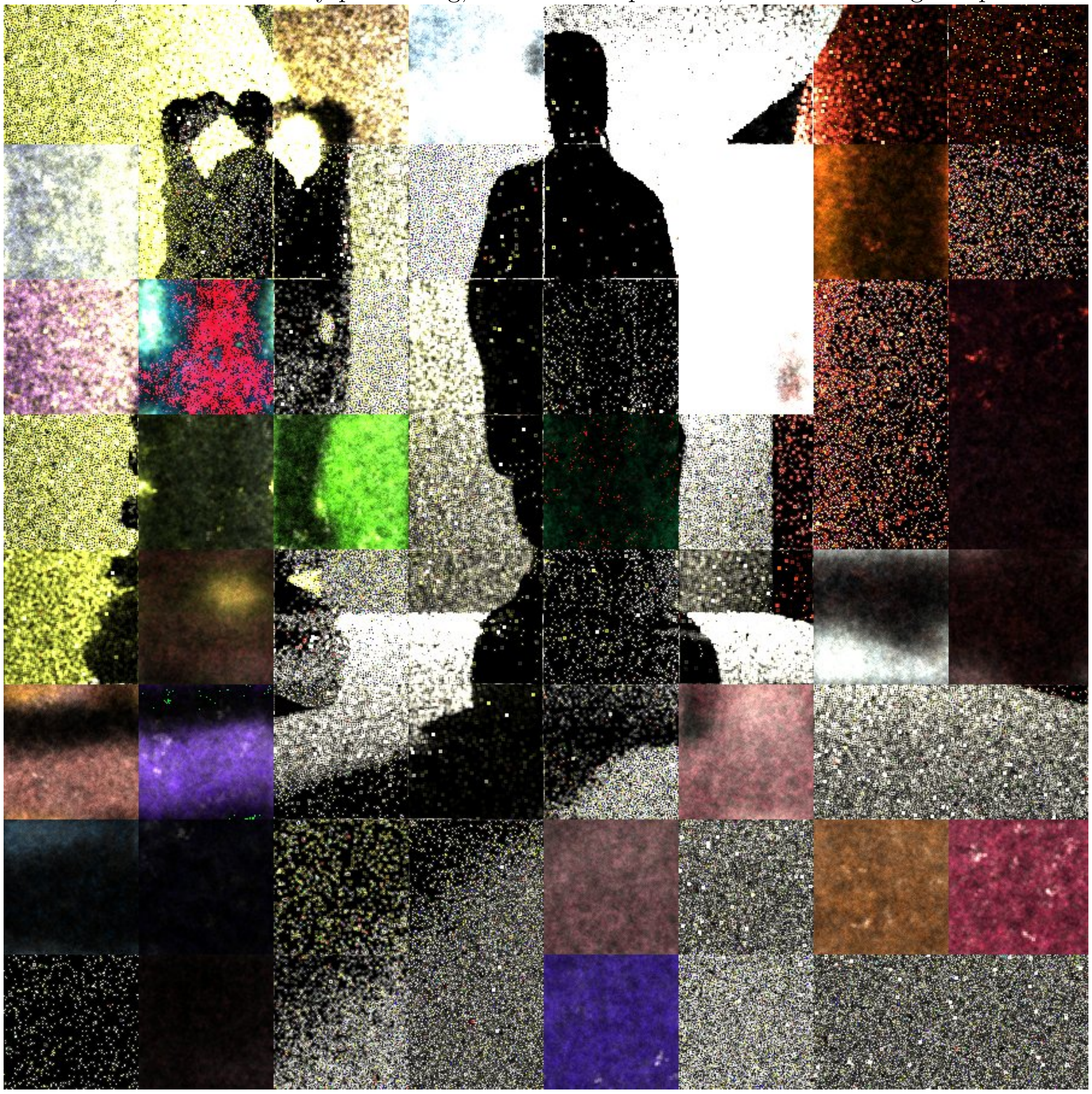
5 Energy Redistribution Path Tracing

This algorithm was introduced by David Cline, Justin Talbot and Parris Egbert in their article [2] ”Energy Redistribution Path Tracing”. The original paper was very informative in the context of some elements of the algorithm, but some elements were difficult to understand or to figure out when making an implementation. Later, Christopher Batty published his technical report [3] Implementing Energy Redistribution Path Tracing. His work gets into some details better than the original paper, however it sill suffers from some mistakes and rendered images show visual corruptions. Many people were trying to understand the algorithm and to implement it correctly, these efforts shown that it is a challenging, error-prone and time-consuming task.

The situation mentioned above gave us a motivation to understand this tricky technique, to analyse properties of mutation schemes and to bring a friendly description that will be helpful during any engineering or research work involved with this algorithm. Below we mention a list of properties that in our opinion make the ERPT an interesting research subject.

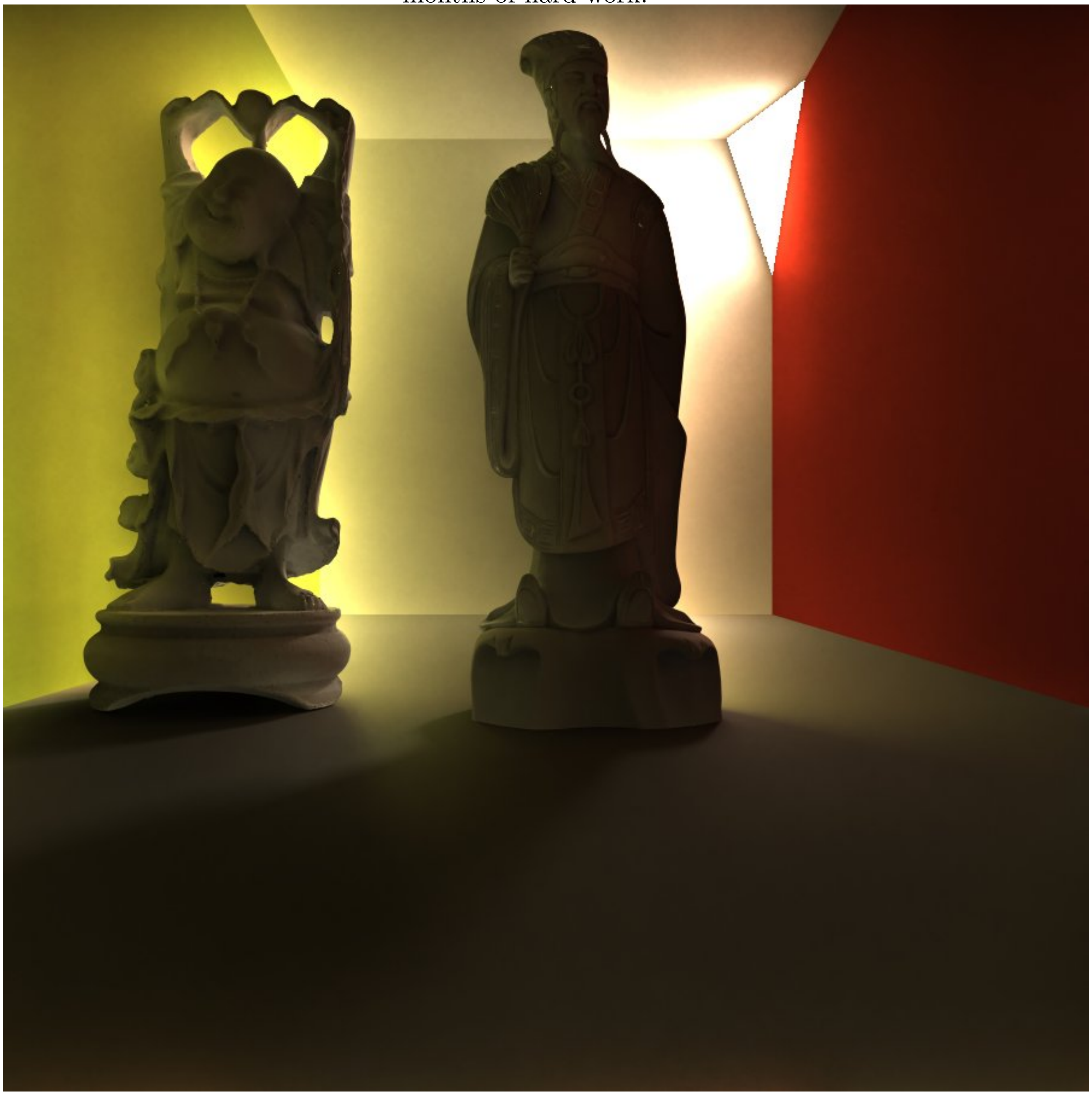
- **locality**, mutations can be easily controlled in a local manner
- **predictability**, predictable behaviour in a relatively small number of iterations
- **offline-suitability**, suitable for offline rendering within space-time of paths
- **simplicity**, mutation schemes that are sufficient to achieve good results are fairly simple
- **flexibility**, a lot of freedom in mutation design
- **mixing-vulnerability**, brings benefits of both the MC and MCMC worlds

The first ERPT-based image rendered by FlowlightFox. [6].
Well, that was not very promising, nor it was impressive, but we did not give up.



[6]Models taken from Stanford repository.

The first high quality ERPT-based image rendered by FlowlightFox, achieved after many months of hard work.[7]



It is important to note that this image does not use any biased refinement techniques. It shows quite difficult back-side lighting scenario with perceivable direct and indirect shadows. All the edges, corners, cavities, wrinkles and tiny details are modelled through real geometry. The matte-like surface appearance is achieved by employing Lambert and Oren-Nayar models.

[7] Models taken from Stanford repository.

5.1 A Quick Analysis of Deposition Constant

The Metropolis Light Transport suffers from start-up bias because the initial path X_0 for the mutation chain is not sampled with a distribution proportional to f. The Energy Redistribution Path Tracing algorithm overcomes this problem somehow and samples the X_0 with a distribution proportional to f. This is achieved by creating many mutation chains per MC sample, all with equal length N. The Monte Carlo Path Tracing is used to generate the initial path X_0 for these mutation chains. When using MCPT, the path X_0 is sampled with probability $p(X_0)$, thus the probability of having X_0 as the initial path of a chain is proportional to $p(X_0)$. However, this number should be proportional to $f(X_0)$. Hence, the number of mutation chains starting at X_0 is off by a factor of $\frac{f(X_0)}{p(X_0)}$. This is resolved by starting multiple chains per initial path X_0. When on average, the number of chains for X_0 is proportional to $\frac{f(X_0)}{p(X_0)}$, the initial mutation chain paths referred to as X_0 are sampled proportional to f and start-up bias is eliminated. This is realized by making the number of mutation chains $numChains(X_0)$ per path X_0 equal to:

$$numChains(X_0) = floor\left[U(0,1) + \frac{f(X_0)}{p(X_0)} \times \frac{1}{N \cdot q_{ed}}\right]$$

In this equation, the $U(0,1)$ is a uniform random number between 0 and 1 and the q_{ed} is the deposition energy quantum (called e_d in the original work [2]), it is the amount of energy that is contributed to the image by each mutation in a chain. To see why ERPT does not require an unbiased estimate of c (the MLT's equivalent of q_{ed}), let us compute the expected number of contributions to the image plane for MLT and ERPT. For MLT, the total number of contributions to the image plane always equals N. For an ERPT sample with initial path X_0, the average number of contributions equals $\frac{1}{q_{ed}} \cdot \frac{f(X_0)}{p(X_0)}$. Therefore, the expected number of contributions per ERPT sample equals

$$\int_\Omega \frac{1}{q_{ed}} \cdot \frac{f(x)}{p(x)} \cdot p(x) \mathrm{d}\Omega(x) = \frac{1}{q_{ed}} \int_\Omega f(x) \mathrm{d}\Omega(x) = \frac{c}{q_{ed}}$$

Compared to MLT, the expected number of contributions per ERPT sample is off by a factor of $\frac{c}{q_{ed}} \cdot \frac{1}{N}$. So, to keep the ERPT estimator unbiased, instead of contributing $\frac{c}{N}$ per mutation, as in MLT, an ERPT mutation should contribute $\frac{c}{N} \cdot \frac{q_{ed}}{c} \cdot \frac{N}{1} = q_{ed}$. Hence, the ERPT algorithm does not require an explicit estimate of c. The energy quantum q_{ed} may be chosen freely without introducing bias. However, it does influence its performance. The number of mutation chains per ERPT sample is inversely proportional to both N and q_{ed}. The expected number M of mutation chains per ERPT sample can be regulated by using $q_{ed} = \frac{c}{N \times M}$. This again requires an estimate of c. However, this estimate does not need to be very accurate and may even be biased, as it only influences the performance of the ERPT algorithm. In practice, the ERPT algorithm is not very sensitive to the value of q_{ed}. We provide this analysis based on the work in [8].

5.2 Core Control Parameters

In its basic form the algorithm is based on a few parameters that govern the workflow. The algorithm creates and processes a lot of Markov Chains so parameters will mainly express their properties. The first parameter is the deposition quantum. It is a small fraction of energy that well be recorded or deposited in the image after each single Markov Chain step. Keep in mind that the energy word here is used rather non-formally, it should be rather replaced with radiance or power, but for the sake of simplicity and harmony with the original publication [2], let it be energy. The second parameter, the chain length, determines how many steps a single Markov Chain will survive. The third one, the expected number of mutation per initial MC sample, determines how many Markov Chains in average will be spawned from a single Monte Carlo sample. This parameter is referred to as a desired number of mutations per correlated integral in the original article [2], but we consider it a little bit misleading. Often it is interpreted as a desired number of mutations per pixel and it works this way quite well, the only concern we had with this interpretation is that it is not fully independent of other parameters. Notice that when number of initial MC samples grows the number of mutations per MC sample than can be physically achieved shrinks. If the number of initial MC samples per pixel will be grater than the desired number of mutations then obviously some of these MC samples will get no Markov Chains at all, they will be just forgotten. To avoid these issues we define the desired number of mutations as a desired number of mutations per initial MC sample. Of course these parameters are in addition to the regular set of parameters present in the start-up Monte Carlo sampler, the lighting deepness, the maximal ray trace depth, number of Monte Carlo samples per pixel.

5.3 Algorithm Outline and Correction

For complementary set of algorithm elements we refer the reader to [2] and [3]. Here we will only recall the routines that are subject of our interest within this document. Let's start with the explicit formulation of the deposition quantum estimation, it's fairly simple and can easily be adopted to work with tiles by limiting pixel ranges.

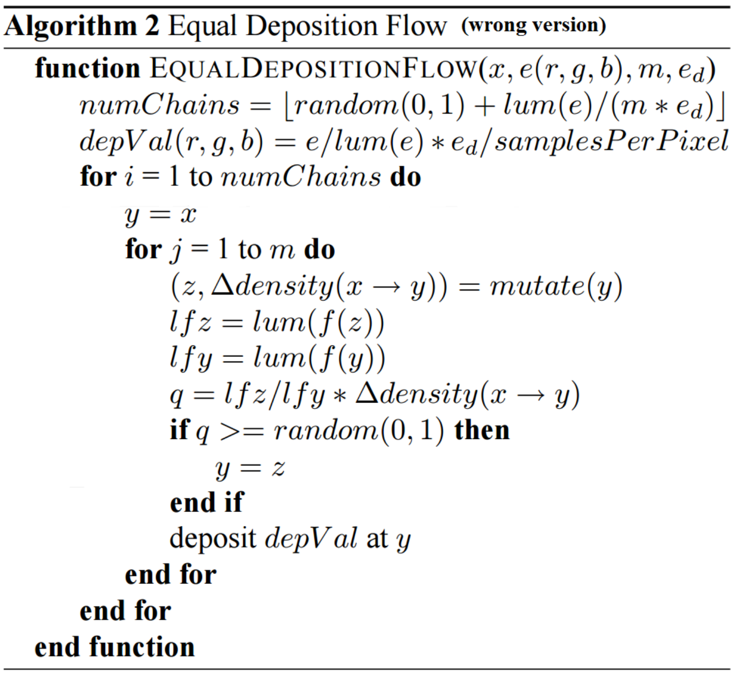
The Deposition Quantum Estimation algorithm.

Algorithm 1 Deposition Energy Estimation

```
function ComputeEd
    $energy = (0, 0, 0)$
    $count = 0$
    for each pixel in image do
        create a path $x$ in the current pixel.
        $X_f(x) = f(x)/p(x)$
        $energy = energy + X_f(x)$
        $count++$
    end for
    $e_d = luminance(energy)/(count * mutations)$
end function
```

Christopher Batty in his report [3] described the Equal Deposition Flow routine that is responsible for spawning Markov Chains and for accumulation of deposition quantum with correct spectral-distribution of energy. Unfortunately, he made a mistake that is easily perceivable as splotches having undesired color flavours. Due to his description of the Equal Deposition Flow a red-coloured path can redistribute its energy over path space regions that are made of blue-coloured paths, or let's say that a reddish surface can trade its energy with a blueish surface. This leads to an undesired color blending. Below we remind his version of the algorithm and we deliver the refined version. Our modification is based on a principle stating the no matter what is going on we should always preserve the spectral-distribution of the current state of Markov Chain, or let's be more specific, we should preserve the color of the so far accepted light path. Renderings created by the FlowlighFox show that our change is good.

The flawed version of the Equal Deposition Flow algorithm.



Algorithm 2 Equal Deposition Flow **(wrong version)**

```
function EQUALDEPOSITIONFLOW($x, e(r, g, b), m, e_d$)
    $numChains = \lfloor random(0, 1) + lum(e)/(m * e_d) \rfloor$
    $depVal(r, g, b) = e/lum(e) * e_d/samplesPerPixel$
    for $i$ = 1 to $numChains$ do
        $y = x$
        for $j$ = 1 to $m$ do
            $(z, \Delta density(x \rightarrow y)) = mutate(y)$
            $lfz = lum(f(z))$
            $lfy = lum(f(y))$
            $q = lfz/lfy * \Delta density(x \rightarrow y)$
            if $q >= random(0, 1)$ then
                $y = z$
            end if
            deposit $depVal$ at $y$
        end for
    end for
end function
```

The refined version for the Equal Deposition Flow algorithm.

Algorithm 2 Equal Deposition Flow **(correct version)**

```
function EQUALDEPOSITIONFLOW($x, e(r, g, b), m, e_d$)
    $numChains = \lfloor random(0, 1) + lum(e)/(m * e_d) \rfloor$
    for $i$ = 1 to $numChains$ do
        $y = x$
        for $j$ = 1 to $m$ do
            $(z, \Delta density(x \to y)) = mutate(y)$
            $lfz = lum(f(z))$
            $lfy = lum(f(y))$
            $q = lfz/lfy * \Delta density(x \to y)$
            if $q >= random(0, 1)$ then
                $y = z$
            end if
            $depVal(r, g, b) = f(y)/lum(f(y)) * e_d/samplesPerPixel$
            deposit $depVal$ at $y$
        end for
    end for
end function
```

5.4 Implementing the Mutations

When it comes to implementing mutation schemes, we must be sure what to calculate and how. The probably most mysterious part of the algorithm's set of formulas that needs to be computed is the tentative transition density division, or briefly $TTDD$. Christopher Batty in his report calls it $\Delta density(x \to y)$. Let's recall how it is defined:

$$TTDD(x \to y) = \frac{T(y \to x)}{T(x \to y)}.$$

Keep in mind that the $T(x \to y)$ and $T(y \to x)$ are relative probability densities of performing a tentative step from one path to another. Also notice that the term $T(x \to y)$ is a multiplication of partial probabilities $T_0(x \to y)$, $T_1(x \to y)$, and so on. Assuming that a sampled path has length equal to n, we have:

$$T(x \to y) = \prod_{k=0}^{n} T_k(x \to y).$$

The migration over film plane. In our implementation we perform screen-space migration over pixels by walking randomly within a square of $K \times K$ pixels. We employ uniform sampling with respect to the square surface area and when walking near the edges of the square we wrap around the square to maintain equal probability of jumping into any pixel from any pixel within the square. Consequently, we have:

$$\frac{T_0(y \to x)}{T_0(x \to y)} = 1.$$

The hard re-connection. This type of mutation event occurs when we connect the y_k node to the x_{k+1} node. This is a completely deterministic operation, so we consider it as a Dirac-impulse and we have:

$$T_k(x \to y) = 1,$$

and if both transitions, $(x \to y)$ and $(y \to x)$, are guaranteed to perform hard re-connection, we have:

$$\frac{T_k(y \to x)}{T_k(x \to y)} = 1.$$

Extension and termination through the soft re-connection. When we generate a ray within a solid angle surrounding the re-connection direction we say that it is the soft re-connection. When performed on non-emitting surfaces it just acts as a path extension, but when performed in the context of light sources it acts as path termination. Assuming that the uniform distribution was chosen and that the solid angles are equally sized, we have:

$$\frac{T_k(y \to x)}{T_k(x \to y)} = 1.$$

Otherwise, with different solid angles (let's call it caps) and distributions we have to take into account probability densities involved with rays generated within the solid angles constructed around the re-connection vectors, finally we have:

$$\frac{T_k(y \to x)}{T_k(x \to y)} = \frac{p_{cap}(y_k, x_{k+1})}{p_{cap}(x_k, y_{k+1})}.$$

For complete information about sampling of spherical caps that we used in the context of soft re-connection routine we send the reader to [5].

Extension through the plain BxDF. In this case we simply generate a sample based on a local BxDF function, very similar to what would be done in a typical MCPT. Assuming that for our x_k and y_k this is the only choice, we have:

$$\frac{T_k(y \to x)}{T_k(x \to y)} = \frac{p_d(y_k)}{p_d(x_k)},$$

often the cosine-lobe distribution is used, having in mind that the probability density for this distribution equals $\frac{cos(\theta_{x_k})}{\pi}$, we end up with:

$$\frac{T_k(y \to x)}{T_k(x \to y)} = \frac{cos(\theta_{y_k})}{cos(\theta_{x_k})}.$$

Mixtures. As we can see, often many terms cancel out, so our formulas get simplified. But it should be explicitly stated that if for a single path we employ a set of strategies, s_1, s_2, ..., s_k, then the complete mixture transition density is given by:

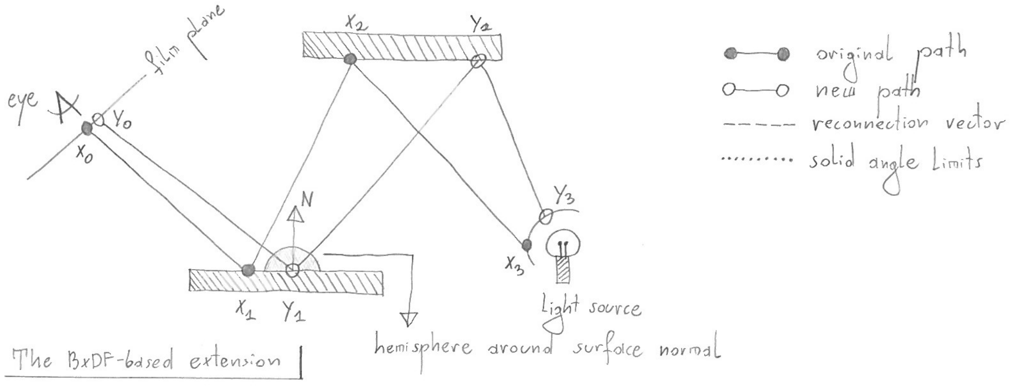
$$T(x \to y) = \sum_{i=1}^{k} p(s_i) \cdot T^{s_i}(x \to y),$$

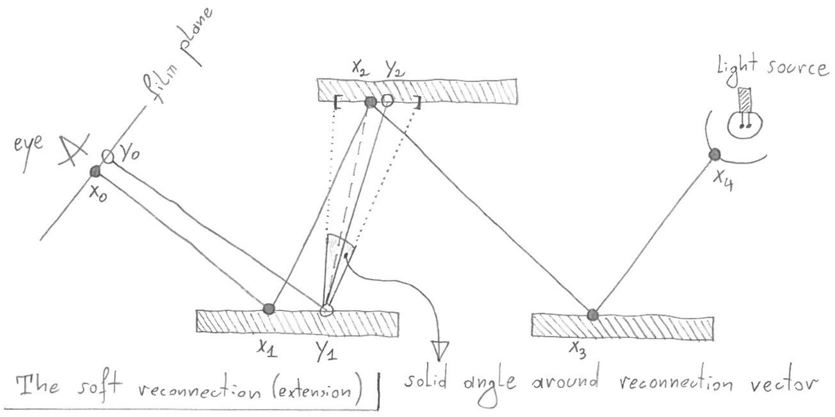
where $p(s_i)$ is the probability of selecting the s_i strategy, and keep in mind that all these probabilities $p(s_i)$ have to sum to 1, so $\sum_{i=1}^{k} p(s_i) = 1$.

There is also another one bit added to the complexity if we want to avoid corner and cavity traps. Later an illustration will be shown, it will mention some issues that affect blow-like mutations described in the next section. To fight against these traps we suggest to perform a random selection between the hard re-connection and BxDF extension, this translates into geometric sequence exponentially fading. So the average or expected number of rays traced per mutation will grow only slightly, it would require some calculations to say it precisely but straight from the top of our heads we feel like the growth should be about 1 or 2. Also we have to reflect this change in the transition probability, constructing a mixture sampling on the top of the two selectable operations.

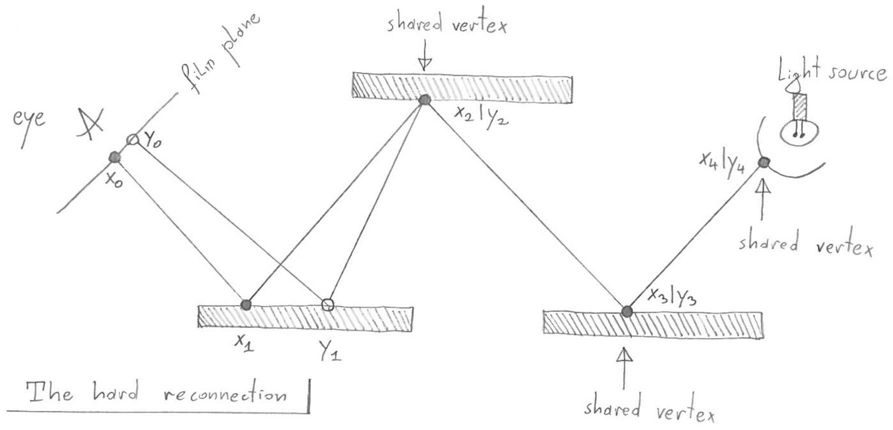
Finally we would like to remind that our ERPT mutations are closed to the path length and the weak path type (defined in the third chapter). Also for each initial MC sample all the Markov Chains must have the same length, keep in mind that the Markov Chain length have to be also preserved across all the paths having the same signature with respect to weak type and path length. These are needed to support the detailed balance rule from which the Equal Deposition Flow was derived.

Illustration presenting the discussed mutating operations [8].





film plane
eye
y_0
x_0
x_2 y_2
light source
x_4
x_1
y_1
x_3
The soft reconnection (extension)
solid angle around reconnection vector



[8]We are computer graphics enthusiasts that are capable of making fancy handcrafts, drafts and drawings.

5.5 Designing Good Mutations

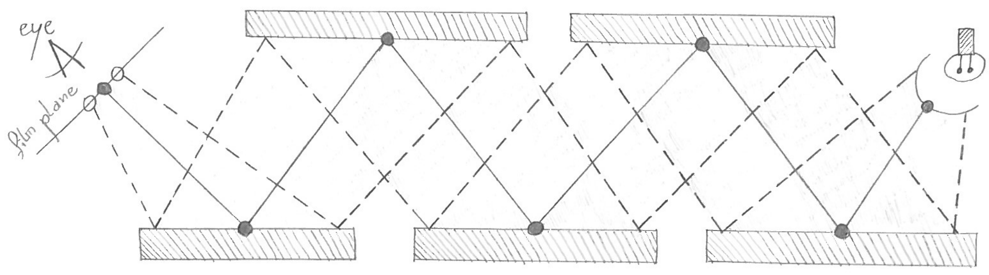
First of all, for the sake of clarity and simplicity in this document, we will work with uni-directional mutations only.

A mutation explores path space around an original path. Let's say a mutation performs walking over nearby paths. Assuming that all mutations start from an eye we have two ways of creating a mutated path. One way is to expand a path sub-set surrounding an original path, it is achieved by generating mutation directions within wide solid angles. The second way is to extend a paths within that sub-set, an extension is about making the mutation prefix path longer. Extending the mutation prefix path may result in a complete reconstruction of an original path. What is the best choice? Should mutations be narrow or rather wide? Should they be short—shallow or rather long—deep? As always it depends on what do we want to achieve. The long—deep mutation prefix paths lead to a more rich information about the lighting space because more vertices get modified, shorter mutation prefix paths result in an undesired appearance as if a scene was rendered with a small set of point lights. Long mutation prefix paths are required for playing with caustic paths as a mutation has to go through all the specular vertices, eventually reaching the light source. Long mutation prefix paths are more time-consuming as they require more rays to be traced. Wider mutations allow a Markov Chain to cover vast part of the path space and screen space quickly, with smaller number of steps, this means that to achieve a good screen space and path space coverage it is sufficient to run many short chains, which in turn improves stratification over screen space. Unfortunately everything has its price, wider mutations struggle to capture small or high frequency details/features and high energy paths covering tiny parts of the path space. Narrow mutations have contrary properties, they are good in capturing complex details/features and irreplaceable in exploring tiny path space regions containing high energy paths. The worst combination of just mentioned properties is a wide and long mutation, this kind of mutation is nearly worthless as it deteriorates into plain Monte Carlo sampler.

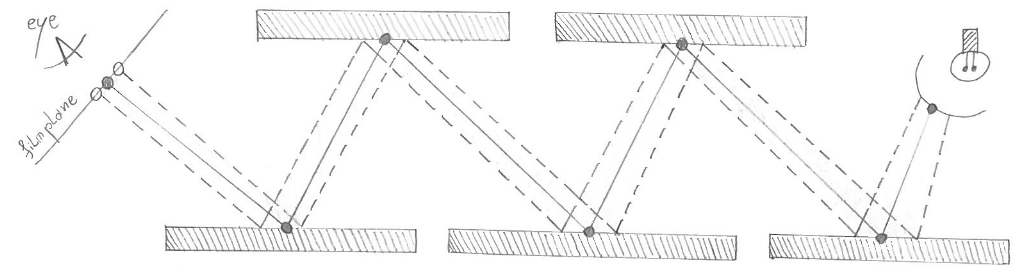
Taking into account the previously mentioned mutation properties we can introduce two major mutation types that will not strictly refer to certain types of light paths, such as caustic or bleeding paths. The first mutation type is a blow-like mutation which is focused on expansion with a tendency to short mutation prefix paths. It may by either wide or narrow, but tends to re-connect to an original path as soon as possible. The second mutation type is a drill-like mutation which is focused on drilling the space surrounding an original path through narrow solid angles. If an original path is a caustic path then the drilling is continued until a light source will be striked, otherwise the mutation can terminate somewhere in the middle. Obviously it is a number one choice for caustic paths, but can be also a good choice for bleeding paths flowing through some difficult geometry.

We consider three types of path extension during mutation events. The first one is a free extension, it is just about performing a BxDF-based deflection. The second one is a strict reconnection, it is about connecting a trailing/pending vertex of a new (mutated) path to corresponding (with the same ordinal number) vertex of an original path. This kind of extension also terminates a path. The third one is a non-strict reconnection which is similar to strict reconnection, the only difference is that we treat the reconnection direction as a deflection dominant, then we generate a random direction within a small solid angle surrounding that dominant. Eventually we will either hit the same surface with some spread around the original vertex or we will hit completely different surface—object.

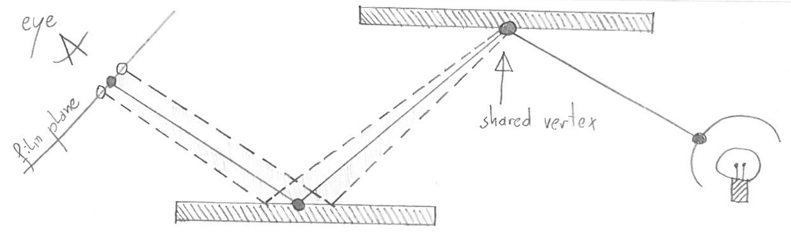
Graphical representation of blows and drills with respect to narrow and wide variants [9].



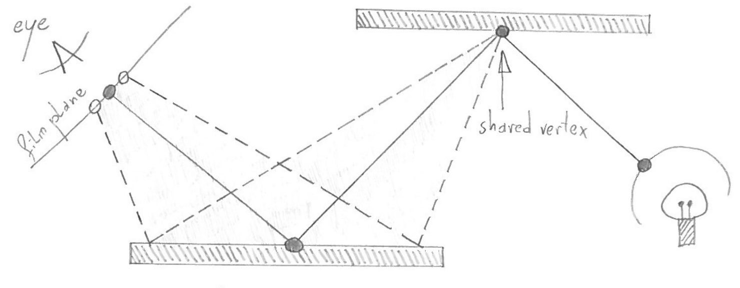
The wide drill-Like mutation



The narrow drill-Like mutation



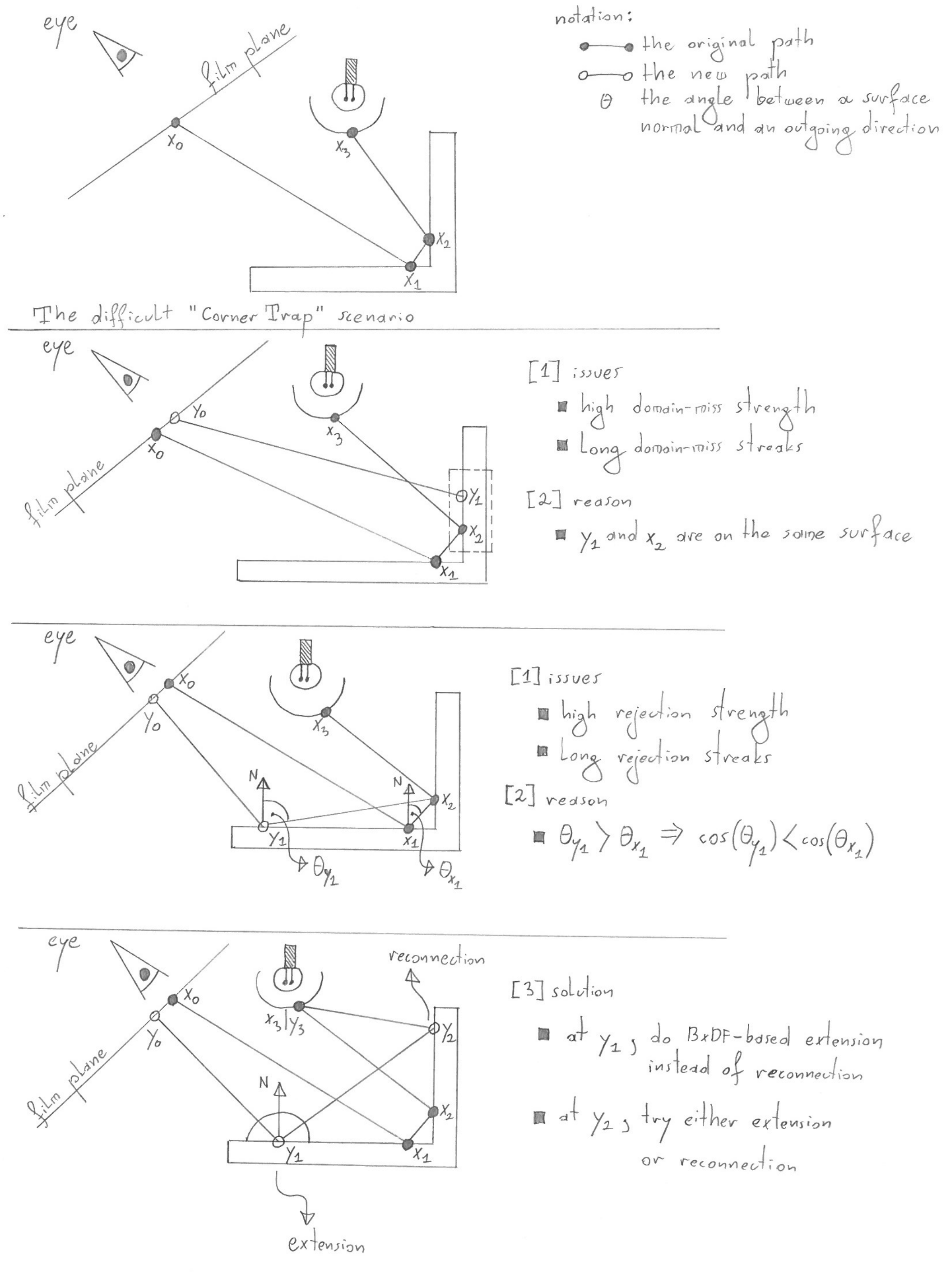
The narrow blow-Like mutation



The wide blow-Like mutation

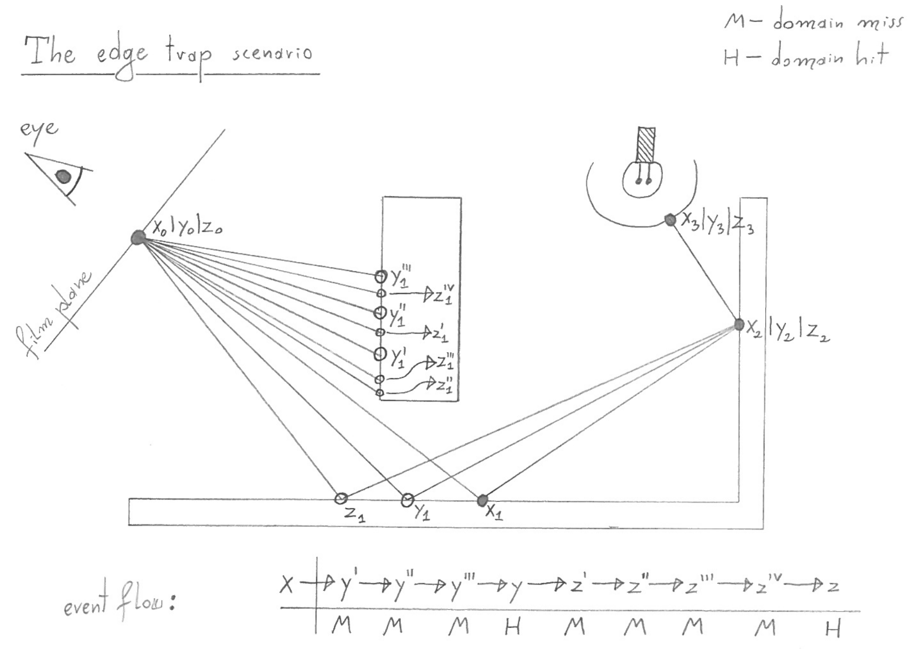
[9]We are computer graphics enthusiasts that are capable of making fancy handcrafts, drafts and drawings.

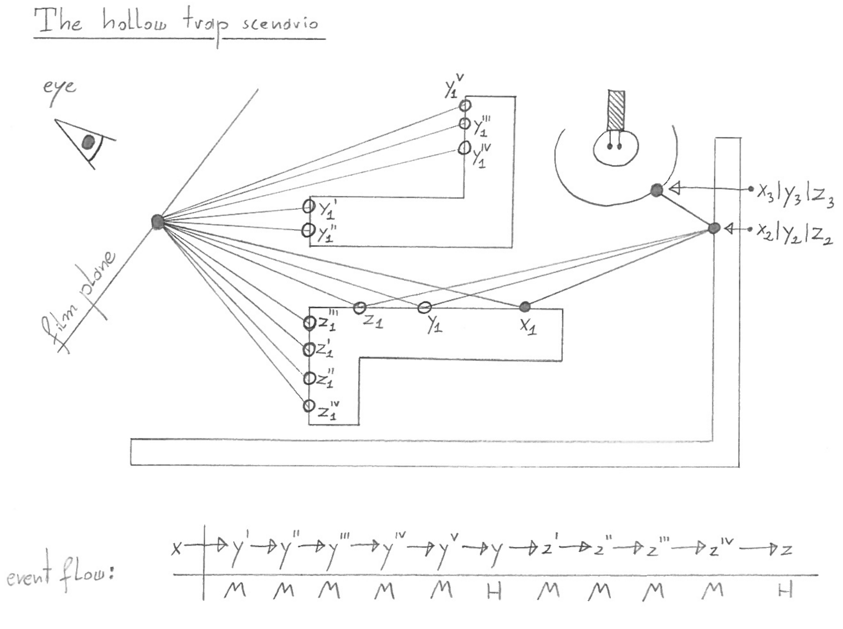
Graphical representation of the Corner Trap [10].


eye
film plane
x_0
x_1
x_2
x_3
notation:
the original path
the new path
θ the angle between a surface normal and an outgoing direction
The difficult "Corner Trap" scenario
y_0
y_1
[1] issues
■ high domain-miss strength
■ Long domain-miss streaks
[2] reason
■ y_1 and x_2 are on the same surface
[1] issues
■ high rejection strength
■ Long rejection streaks
[2] reason
■ $\theta_{y_1} > \theta_{x_1} \Rightarrow \cos(\theta_{y_1}) < \cos(\theta_{x_1})$
N
θ_{y_1}
θ_{x_1}
reconnection
$x_3 | y_3$
y_2
[3] solution
■ at y_1, do BxDF-based extension instead of reconnection
■ at y_2, try either extension or reconnection
extension

[10]We are computer graphics enthusiasts that are capable of making fancy handcrafts, drafts and drawings.

Graphical representation of the Edge Trap and Hollow Trap [11].





[11]We are computer graphics enthusiasts that are capable of making fancy handcrafts, drafts and drawings.

Now we will introduce quality norms that quantify empirically measurable mutation properties. An obvious way of rating the quality of an algorithm is rating the visual appearance in the context of time it takes to render it. Can we do any better than that? Can we have some more precise, detailed, formal or numerical? Fortunately yes, we noticed that there is a set of events and a set of properties that can be constructed on the top of that events, these properties can be measured during rendering runtime and they provide very meaningful information about how good our mutations are for a given scene and lighting conditions. We noticed like four events, the mutation acceptance, the mutation rejection, the domain hit and the domain miss. Now we will define them.

The mutation acceptance event occurs when a Markov Chain, during its tentative transition step, decided that a new path generated by a successful mutation is accepted and our chain moves to it. This is the most desired event across all the events.

The mutation rejection event occurs when a Markov Chain, during its tentative transition step, decided that a new path generated by a successful mutation gets rejected and it stays at the original path in path space. When this event shows up we are generally a bit unhappy but it is not the worst thing that can happen, it depends on a few things, we will discuss it later. This event has also an additional advantage, despite the fact that a sample got rejected, we still can accumulate to both locations by replacing the deposition quantum with expected values at both locations, the new and the original.

The domain hit event occurs when a mutation does not fail to create a new path. In other words, a path that got created by a mutation routine is valid in our path space and we are are allowed to make a decision, we can either accept it or reject it. A path may be invalid for many reasons, re-connection step may strike an obstacle along the way, path extension may fail to hit a light source, path extension may hit a material that does not have the same scattering mode, a mutation ray can fly away into a void. This is absolutely desired event and we definitely aim to be in a situation in which all mutations end up with domain hits because this means that our time spent on tracing mutation rays was not wasted.

The domain miss event occurs when a mutation fail to create a valid path. This can be interpreted as an instant rejection with a little bit more pain because of computing resources wasted on tracing ray for a path that can not be constructed. That is the biggest fail as there is no chance for Markov Chain to leave its current location and the accumulation will always go fully into the original location. We can not accumulate expected values at both locations. It is also likely in some cases that this event will appear in a form of long consecutive chain of failing mutations which leads to a highly visible defects that are hard to recover from.

We can count these events and build up some metrics or norms on the top of them. We introduce the concepts such as strength and streak. The strength is simply the ratio of number of events of some type to the total number of mutations performed. The streak is the number of uninterrupted consecutive events of the same type. We will be usually interested in average streak, the median could also be useful but is more difficult to compute. These two concepts can be mixed with any type of event and this is what brings us measurable information about a MCMC-based algorithm performance. So let's answer the question - what properties a good set of mutations should have? Obviously, it should have high acceptance strength, low rejection strength and low domain miss strength. This information however is still far from being comprehensive and judicial. Even if rejection strength is quite high, it does not mean that Markov Chains perform poorly. This is the situation in which streaks play their role. If rejection strength is high but the average rejection streak is low an employed set of mutations can be considered good. To understand that think about the following example.

Imagine a Markov Chain that performs 200 steps and imagine that half of steps will face rejection events and another half will face acceptance events. The lacing pattern or let's say the order in which these events are interleaved is important. Let's consider two cases, in the first case our mutation chain starts with rejection, hits the acceptance in the second step and continues to alternate between rejection and acceptance all the way until the end, so it looks like $RARARARARA(RA)^*$. In this case rejection strength will be equal to 0.5 (or let's say 50 %) and the average rejection streak will be 1. And this is not that bad, this scenario has a chance to spread information over 100 pixels in a very balanced way because each of these pixels may be visited not more that 2 times. In the second case we want to consider a situation in which a Markov Chain starts with the acceptance event and generally shoots 100 uninterrupted acceptance events followed by 100 uninterrupted rejection events. So a symbolic word for that situation would look like $AAAAAA(A)^*RRRRRR(R)^*$. The first sequence of acceptance events will have a chance to cover 100 different pixels which is awesome but the second sequence ruins it all because it will accumulate 100 times into the same pixel. So we do not have that nice accumulation balance and this kind of event will create extremely bright spot or speckle on our image. Usually it is a dramatic defect, a deal-breaker in terms of high quality rendering. Let's look at how it is expressed by our norms, the rejection strength is exactly the same, but the average rejection streak equals 100 instead of 1. This shows us that the ordinal distribution of events is very important and has significant impact on the image quality. Also we can easily notice that average streaks must be as low as possible for events such as rejections and domain misses. The two cases that we just presented do not cover the whole set of possibilities. There are other lacing patterns that are undesired and destructive, e.g. $(AR^6AR^9AR^3AR^7)^+$. For this example some other measurable norm should be established, so we could detect it during rendering. Generally, the statistical norms just described explain why in the original paper [2] they needed to use the so called Consecutive Sample Filtering. This biased filter was successfully removing speckles, sparkles, fireflies, bright spots, hot spots or however we want to call them, at the expense of bias in estimates. But since that routine is using only a single threshold it is unable to solve cases such as $(AR^6AR^9AR^3AR^7)^+$.

5.6 Optical Type System as a Tool for User-Controllable Transition Schemes

Here we would like to describe the complementary optical type system with respect to all important events and properties that can be used to a construction of optimised transition schemes. Here we define 16 atomic types:

- eDr : external diffuse reflection
- $eWGr$: external wide glossy reflection
- $eNGr$: external narrow glossy reflection
- eSr : external specular reflection
- eDt : external diffuse transmission
- $eWGt$: external wide glossy transmission
- $eNGt$: external narrow glossy transmission
- eSt : external specular transmission
- iDr : internal diffuse reflection
- $iWGr$: internal wide glossy reflection
- $iNGr$: internal narrow glossy reflection
- iSr : internal specular reflection
- iDt : internal diffuse transmission
- $iWGt$: internal wide glossy transmission
- $iNGt$: internal narrow glossy transmission
- iSt : internal specular transmission

These types can be mixed up with the concepts of impulse deflection and scattered deflection to bring even larger set of types. By using these types to construct a hierarchy of strong and weak path types we can perform a path space partitioning which in turn may be used to design intelligent mutation strategies through relaxing the type-oriented constraints. We mention it as an avenue for future exploration.

5.7 MC Sample-Set Entropy

Here we would like to introduce a concept of MC sample-set entropy along with an explanation of its role in achieving good MC-MCMC sample trade-off. Imagine that we generated some set of initial MC samples and we chose values for all required parameters. Having these things fixed we can introduce the concept of a mutation volume. Pick a single MC sample from our set of initial MC samples and imagine running an infinite number of Markov Chains started from this sample. These chains will cover all paths reachable from the initial MC sample, the complete set of these reachable paths is what we call mutation volume. Now notice that depending on the distribution of initial samples mutation volumes may bring smaller or larger coverage of the path space, also they may be mutually overlapping. A coarse-grain observation here is that we want to achieve good path space coverage with not too much overlapping. So the entropy in this context would express how rich and unique is the information given by the set of initial MC samples.

5.8 Trade-Off Between Number of MC Samples and MCMC Samples

One of the ERPT properties that have not been explored is the relationship between the number of initial MC samples and the desired number of MCMC samples per MC sample. Here we would like to discuss the topic.

In our experiments we were suffering from the light-space noise when using just 1 initial MC sample per pixel. Values between 4 and 8 seemed to bring quite good results. In general sense we think that this trade-off is dependent on our earlier mentioned depth-wise distribution. Taking into account MC sample-set entropy we may prefer a good stratification over strong importance sampling. By stratification here we mean both, the stratification within each lighting depth-wise layer and the stratification over these lighting layers. So again we meet depth-wise distribution.

5.9 Workflow, Work-splitting Schemes and Accumulation Styles

The Energy Redistribution Path Tracing is quite challenging for multi-threading, like all other algorithms that involve Markov Chains. The reason is very obvious, a Markov Chain that starts from some pixel walks over an image plane and accumulates information into randomly visited pixels. We consider two work-split schemes, pane-based and tile-based.

In pane-based scheme we simply create an image buffer for every thread. Assuming that our screen space has $N \times M$ pixels and we have T threads running at the same time, we have $N \times M \times T$ pixels in total that will have to be stored in memory.

In tile-based scheme we split the screen into tiles and each thread gets its tile to render. Assuming that our screen space has $N \times M$ pixels and we have T threads, we could expect that memory consumption will be proportional to $N \times M$ pixels. There is an issue, if we just split the rendering into tiles then we will end up with highly perceivable tile edges. Unfortunately, experiments show that regardless of the number of initial MC samples and number of mutations they will persist. To overcome the issue we introduce flanged tiles with fringed deposition. The concept is very simple each tile has a flange that can be understood as an extension of an original tile. The initial Monte Carlo samples can only be generated from pixels that belong to an original tile, meaning that all Markov Chains start from original tile. The key thing is that chains can accumulate not only into original tile but also into its flange. This strategy results in flange-flange and flange-tile overlapping that in turn leads to a blending with less hard edges. Unfortunately, this solution is also imperfect and edges may be still noticeable as a lattice pattern with low frequency. To resolve the issue we can force flanges to be large enough to cover the entire screen space. Well, obviously we get back to the pane-based memory consumption but the situation is not hopeless, we will return to this topic later. One may ask - why should I bother myself with flanged tiles then? The answer is that they bring another benefit, the varying deposition quantum. Notice that tiles cover different regions of a screen space and a scene. Some of them may be lighter and some of them may be darker. This observation leads to a conclusion that calculating a single deposition quantum from an entire screen is generally sub-optimal. Tiles with larger total amount of radiance will enlarge the deposition quantum. Larger deposition quantum means that darker regions of an image will get smaller number of Markov Chains redistributing the energy of initial Monte Carlo samples. So darker tiles will remain undersampled, suffering from higher variance and lower image details. From a technical point of view this translates to a desired number of mutations being preserved within an entire screen space but being non-preserved within some of tiles. With flanged tiles we can calculate a distinct deposition quantum for each of tiles, consequently we will maintain the desired number of mutations within each tile.

Now let's return to the topic of memory management in the context of accumulation. When rendering algorithms are determining pixel values or calculating any kind of partial information that will contribute to the final pixel value, memory may be accessed in many different patterns, memory requirements in terms of size may be enormous, memory may suffer from synchronization issues when many threads are employed. To overcome all the challenges and to service each scenario as efficient as possible we introduce a few accumulation styles that impact memory consumption requirements along with its computational properties.

AccumulationStyle::Blasting. This is the most simple way of accumulation, in which each thread has access to the same memory resource, let's say that this resource is shared between threads, and each of these threads can access this shared memory (e.g. by performing read—write operations) simultaneously without any synchronization mechanisms. This method offers low memory consumption and fast memory access but assumes that threads are working on disjoint sub-sets of the memory resource. In other words, the assumption states that the original memory resource is shattered into elements exclusively accessible by only one thread. The Monte Carlo Path Tracing is a good example of an algorithm that can successfully exploit this method.

AccumulationStyle::Buffering. This is a method in which the original memory resource is duplicated, copied multiple-times or may be split into many overlapping elements. The key thing is that all that copies and elements are represented by independent memory allocations that eventually must be flushed into the original memory allocation. This method may lead to an excessive memory consumption but keeps threads safe in terms of memory access collisions without necessity for any synchronization mechanisms. Also, an important property is that buffers are by-definition large or let's say large enough to survive the entire thread lifetime. Consequently, the flush operation is performed only when a thread finishes its job, and it is guaranteed that there will be no need for flush operations somewhere in the middle. The Energy Redistribution Path Tracing is a good example of an algorithm that can make use of this method, but preferably only when it draws a single framebuffer.

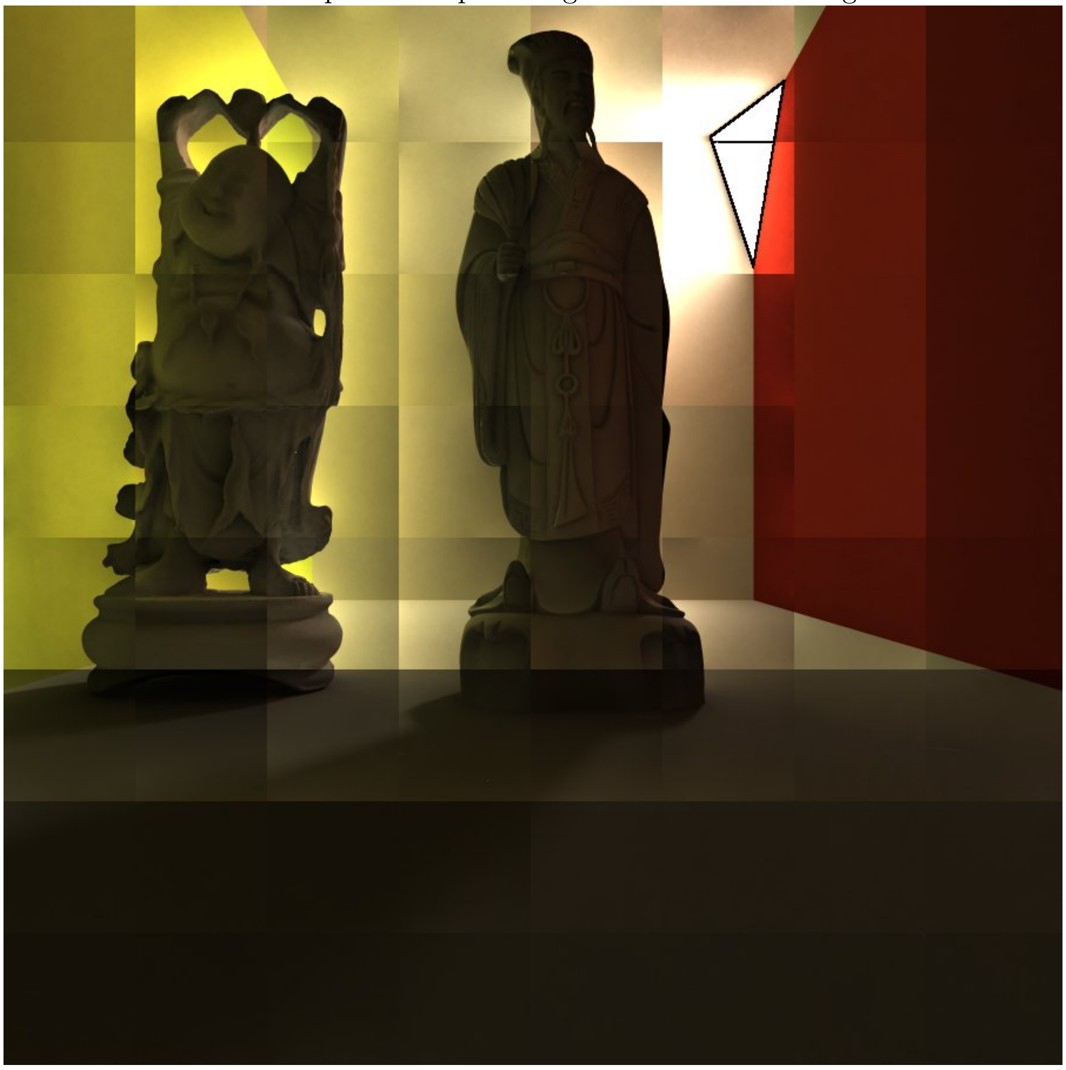
AccumulationStyle::Caching. This method is pretty much similar to buffering but with one important exception - cache is not large enough to survive the entire thread lifetime, or let's say it may survive but it is not guaranteed. So during a runtime of a single thread we can expect/experience many flush operations. When excessive memory consumption is not acceptable we can employ relatively small buffers (called caches) that will provide us with a good balance between low memory consumption and low flush frequency. Unfortunately there are trade-offs, smaller caches imply higher flushing frequency while lower flushing frequency works in the opposite direction - implies larger caches. Well, we have to live it out. The Energy Redistribution Path Tracing is a good example of an algorithm that can make use of this method, especially when it draws multiple-framebuffers, e.g. framebuffers corresponding to distinct layers of path-space. As an implementation of this method we propose the so called micro-tiling algorithm, which requires only one KeyLock (mutex-like object) per framebuffer.

A quick analysis telling us why buffering does not always work well for us. Markov Chain Monte Carlo algorithms are challenging in terms of memory transactions because they start drawing samples at some pixel and as the time passes they migrate between pixels. Consequently, they perform accumulate operations over many pixels in unpredictable patterns, possibly any pixel can be visited by a Markov Chain starting from a certain pixel. We are nearly unable to make any assumptions about set of visited pixels nor we have any 'a priori' knowledge. So let's imagine that we want to render an image with the following properties:

- full HD resolution (that is 1920×1080),
- the PathSpace depth equals 8,
- a single pixel is 12-bytes large (3 channels x 4 bytes-per-channel),
- we employ 12 threads in our rendering process.

This gives us $12 \times 1920 \times 1080 \times 8 \times 12$ bytes, which translates to approximately 2278 [MegaByte] or roughly 2.22 [GigaByte]. Set the PathSpace depth to 16 and use 24 threads, you will end up with 8.88 [GB]. It is a trap leading to an enormous memory allocation explosion.

An example of sharp tile-edges in ERPT rendering



6 Conclusion

Now we will do a summary for this compact document, we will mention what we wanted to achieve, what was done and what was not, finally leaving the reader with suggestions for future work. One of the goals was to create a complete and correct implementation of discussed algorithms on the ground of our own rendering engine, the FlowlightFox gsl.Engine, or simply the Foxy. The term gsl.Engine stands for Global Shading-and-Lighting Engine. The vast majority of time spent on that thesis went into the design and development of that rendering engine along with many experiments and a painful debugging that was not completed during the timespan of this work. Yet we are happy that this quite big and complex system having approximately 1.1 MB large code (that is a bit over 10^6 of characters) has many working features, tools and instruments, that allowed us to perform our experiments and gave us a lesson about how hard it can be to develop such system. We were struggling a lot with floating-point imprecisions, multi-threading, memory consumption and corruptions.

We presented quite large amount of theoretical background for light transport and Mote Carlo methods that we used as a base for construction of algorithms that we implemented in the Foxy. Although they look a bit overcomplicated we believe they bring a bit of the well-grounded comprehensiveness. Based on our experiences with the Foxy and its instruments allowing us to look into different parts of path space we prepared a Monte Carlo noise classification. We made an insight into distinct lighting depth-wise layers to see how much information is stored, and to see how the noise appearance and convergence properties vary across these lighting layers. We proposed and derived two depth-based distributions, the equidistribution which appeared to be too weak to bring noticeable improvement for convergence in deeper layers, and the expodistribution which appeared to be too aggressive for first few layers, leaving them simply undersampled for a very long time. We suggest to construct a distribution that is somewhere in the middle between the two, or some fine-tuning of the expodistribution. We would also like to explore well-known importance sampling techniques combined with our depth-wise distributions. We leave it as a subject for future work. As for Markov Chains and Energy Redistribution Path Tracing, we tried to bring a reader-friendly descriptions, derivations and intuitions. We believe that our work will bring a bit of clarity where things seem to be overcomplicated and difficult to understand or figure out. We also believe that our work will uncover the true and beautiful simplicity of MCMC-based algorithms that is hidden behind the loads of mathematical symbols and beefy formulas. We feel like we delivered an added value to the topic as we made a few refinements, corrections and improvement suggestions for the ERPT. We brought a clarified and flawless interpretation of the parameter named desired number of mutations per correlated integrand, it is presented as independent of all other parameters so a user does not have to pay attention if he set up all the values fairly good and will not struggle to find a flawless balance. We also presented the refined and by our best knowledge correct version of the Equal Deposition Flow routine. This was followed by the invention of statistical quality norms that are measurable at runtime, they are good quantifiable expression of visual corruptions that are by-products or side-effects of ERPT. We mentioned how they explain the necessity of employing the consecutive sample filtering presented in the original publication [2] bringing the ERPT to the world. We would like to continue this topic, aiming to construct norms that will be able to detect all the relevant lacing patterns with respect to every single mutation strategy. It would be also great to build up an automatic fine-tuning of the ERPT parameters and automatic selection of mutation strategies based on that statistical norms, our high quality seen in rendering was achieved by gathering the statistics and performing manual tuning based on that gathered information, we agree that this is not comfortable to a user. We presented a different view on the mutation strategies. Instead of distinguishing between the lens and caustic mutation strategies, we constructed all our mutations as an eye-side manner. We focused on whether they are long—deep or short—shallow, and whether they are wide or narrow. We discussed how to apply them to different lighting situations, unfortunately we did not manage to examine our unidirectional caustic-oriented mutations, we would like to see how they perform in combination with importance sampling techniques, e.g. [9] photon map based importance sampling. We addressed the issue of corner and cavity traps and suggested a simple solution to that problem. We also introduced an idea of using multiple deposition energy quanta with different values and to our knowledge it is the first time of mentioning such idea. However more experiments for its claimed strengths would be useful along with the formal proof of preserved properties of the algorithm. We are happy to say that we managed to theoretically (by our observations and derivations) as well as practically (by our implementation) create the ERPT version that can handle variety of diffuse surfaces modelled by Lambert and Oren-Nayar models that are involved with complex edges and cavities. We remind that for the purpose of our experiments the texturing system was turned off and all the surface details, shapes, features, cavities and edges on the Asian statues were modelled purely by high resolution geometry, this is a proof that the Foxy successfully resolves the addressed ERPT-related issues while bringing high quality nearly photo-realistic images and preserving the original unbiased nature of the ERPT.

7 Bibliography

[1] Eric Veach, Leonidas J. Guibas. Metropolis Light Transport. Computer Science Department, Stanford University.

[2] David Cline, Justin Talbot, Parris Egbert. Energy Redistribution Path Tracing. Brigham Young University.

[3] Christopher Batty. Implementing Energy Redistribution Path Tracing. Department of Computer Science, The University of British Columbia.

[4] David Cline and Parris Egbert, 2005. A practical introduction metropolis light transport. Brigham Young University.

[5] Philip Dutre. Global Illumination Compendium. Department of Computer Science, Katholieke Universiteit Leuven.

[6] Kajiya J. T. 1986. The Rendering Equation. In SIGGRAPH 1986.

[7] Matt Pharr, Greg Humphreys. Physically Based Rendering, From Theory To Implementation.

[8] Dietger van Antwerpen. Unbiased physically based rendering on the GPU. Delft University of Technology.

[9] Henrik wann Jensen, 1996. Global illumination using photon mapping. In Rendering Techniques, 21-30.

[10] Eric P. Lafortune, Yves D. Willems. Bi-Directional Path Tracing. COMPUGRAPHICS 1993.

[11] Bruce Walter, Sebastian Fernandez, Adam Arbree, Kavita Bala, Michael Donikian, Donald P. Greenberg. Lightcuts: a scalable approach to illumination. SIGGRAPH 2005.

[12] Per H. Christensen. Point-Based Approximate Color Bleeding. Technical Report #08-01, Pixar Animation Studios, 2008.