UNIVERSIDAD MICHOACANA DE SAN NICOLÁS DE HIDALGO

DIVISIÓN DE ESTUDIOS DE POSGRADO DE LA FACULTAD DE INGENIERÍA ELÉCTRICA

Efficient Markov Chain Monte Carlo methods using path-space operators and guiding.

Presentado por:

Enrique José Vega Equihua

Tesis para obtener el grado de Maestría en Ciencias en Ingeniería Eléctrica
en la
Universidad Michoacana de San Nicolás de Hidalgo
División de Estudios de Posgrado
Facultad de Ingeniería Eléctrica

Director de tesis:

Dr. Luis Eduardo Gamboa Guzmán

Co-director de tesis:

Dr. Adrien Gruson, École de Technologie Supérieure, Montréal

Morelia, Michoacán. Febrero 2025

Universidad Michoacana de San Nicolás de Hidalgo
Facultad de Ingeniería Eléctrica
División de Estudios de Posgrado

DIVISION POSGRADO

EFFICIENT MARKOV CHAIN MONTE CARLO METHODS USING PATH-SPACE OPERATORS AND GUIDING.

Como resultado del proceso de deliberación efectuada el Viernes 24 de Enero de 2025 a las 11:30 am, los Miembros del Jurado de Examen de Grado de **Enrique José Vega Equihua** aprueban la **Tesis de Maestría en Ciencias en Ingeniería Eléctrica.**

Dr. Jaime Cerda Jacobo
Presidente del Jurado

Dr. Luis Eduardo Gamboa Guzmán
Director de Tesis

Dr. Adrien Gruson
Co-director

Dr. José Antonio Camarena Ibarrola
Vocal

Dr. Luis Valero Elizondo.
Revisor Externo (FISMAT UMSNH)

Dr. J. Aurelio Medina Rios
Jefe de la División de Estudios de Posgrado
de la Facultad de Ingeniería Eléctrica. UMSNH
(Por reconocimiento de firmas)

Morelia, Michoacán, Viernes 24 de enero de 2025.

Resumen

La ecuación de renderizado que modela el transporte de luz, utilizada para generar imágenes por computadora, es resuelta usando el método numérico estocástico llamado integración Monte Carlo, cuya rapidez de convergencia depende de la función de densidad de probabilidad (pdf) usada para obtener las muestras, siendo el caso ideal un muestreo proporcional a la radiancia de los caminos de luz que conforman el espacio muestral. Las formas más simples de obtener muestras son llamadas "path tracing" y "path tracing bidireccional" las cuales son extremadamente ineficientes para explorar caminos de luz *difíciles*, es decir, caminos con ya sea problemas de visibilidad, o con varios vértices especulares.

Este problema es aminorado empleando dos posibles técnicas: exploración global, la cual consiste en almacenar información posicional y direccional de los caminos que ya fueron explorados, con la finalidad de *guiar* los muestreos subsecuentes a zonas de alta radiancia en la escena; y exploración local, un proceso estocástico que usa cadenas de Markov ergódicas las cuales tienen pdfs estacionarias proporcionales a la radiancia transportada por los caminos de luz. Estas cadenas de Markov muestrean caminos de luz "principales" realizando una exploración global del dominio de integración (mutación grande), los cuales serán usados para explorar localmente caminos similares a ellos por medio de la aplicación de pequeñas modificaciones (mutaciones). Los tipos de algoritmos de renderizado que usan cadenas ergódicas de Markov para muestrear son llamados algoritmos de transporte de luz tipo Metropolis.

Los algoritmos de exploración local tienen dos problemas principales: primero, por la naturaleza de las pdfs estacionaria de las cadenas de Markov, los píxeles más brillantes de la imagen renderizada reciben más muestras, causando una convergencia no uniforme de la imagen, es decir, se tiene un problema de estratificación; segundo, no se está usando ninguna información global sobre las zonas con mayor radiancia de la escena, por lo que estos algoritmos carecen de "memoria" pues lo único que hacen es mutar sin almacenar ninguna información.

En esta tesis nos enfocamos en maneras de resolver los dos problemas anteriores, con respecto al primer problema, hicimos mejoras al algoritmo de exploración local estratificado llamado "Stratified Markov Chain Monte Carlo" (SMCMC) mediante la aplicación de operadores del espacio de caminos más complejos, logrando así una convergencia más rápida para escenas con pocos vértices especulares; con respecto al segundo problema, combimanos algoritmos de exploración local con algoritmos de exploración global consiguiendo incrementos sobre el algoritmo de exploración local, tanto de la proporción de aceptación de mutaciones grandes, como de la velocidad de convergencia para escenas con visibilidad complicada pero con pocos materiales especulares y dieléctricos. Todos nuestros resultados fueron obtenidos usando el motor de renderizado "pbrt version 3.0", el cual fue modificado para incluir nuestros algoritmos propuestos. Presentamos nuestros resultados comparando nuestros algoritmos propuestos con algoritmos de la literatura, analizando experimentalmente tanto sus fortalezas como sus desventajas.

Palabras clave— computer graphics, Metropolis Light Transport, stratification, bidirectional path tracing, path guiding

Abstract

The rendering equation that models light transport, utilized to make computer generated images, is solved by using the stochastic numerical method called Monte Carlo integration, whose convergence speed depends of the probability density function (pdf) used to get the samples, being the ideal case, a sampling that is proportional to the radiance of the light paths that make up the space. The simplest ways of obtaining samples are called path tracing and bidirectional path tracing, which are extremely inefficient for exploring *difficult* light paths, that is, paths with either a lot of visibility issues, or with several specular vertices.

This problem is reduced employing two possible techniques: global exploration, which consists in storing positional and directional information provided by the light paths that were already explored, this with the aim of *guiding* the subsequent samplings to high radiance zones in the scene; and local exploration, a stochastic process that uses ergodic Markov Chains with stationary pdfs proportional to the radiance carried by the light paths. These Markov Chains sample "main" light paths by globally exploring the integration domain (large mutation), which will be used to locally explore similar paths by applying small modifications (mutations). The type of rendering algorithms that use ergodic Markov Chains to get their samples are called Metropolis Light Transport algorithms.

Local exploration algorithms have two main problems: firstly, due to the nature of the stationary pdfs of the Markov Chains, the brightest pixels of the rendered image receive more samples, inducing an uneven convergence of the image, that is, a stratification problem; secondly, no global information is being used about the zones with the highest amount of radiance in the scene, or in other words, these algorithms are memory-less because the only thing they do is mutate the current sample without storing any information.

In this thesis we will focus about ways of solving the two previously mentioned problems, with respect of the first problem, we did improvements to the local exploration algorithm called Stratified Markov Chain Monte Carlo (SMCMC) by applying more complex path-space operators, achieving in this way, a faster convergence for scenes with few specular vertices; with respect the second problem, we combined local exploration algorithms with global exploration algorithms achieving an increased acceptance rate of large mutations and also a better convergence speed when compared to local exploration techniques alone for scenes with difficult visibility but few specular and dielectric materials. All of our results were obtained using the rendering engine called "pbrt version 3.0", which was modified to be able to run our proposed algorithms. We present our results by doing comparisons between our proposed methods and methods already found in the literature experimentally analyzing both the strengths and shortcomings of our methods.

Key words— computer graphics, Metropolis Light Transport, stratification, bidirectional path tracing, path guiding

List of Abbreviations

BDPT Bidirectional Path Tracing

BSDF Bidirectional Scattering Distribution Function

DI Direct Illumination

GI Global Illumination

MC Monte Carlo

MCMC Markov Chain Monte Carlo

MCMC-PG Markov Chain Monte Carlo + Path Guiding (at the same time)

MH Metropolis-Hastings

MIS Multiple Importance Sampling

MLT Metropolis Light Transport

MMLT Multiplexed Metropolis Light Transport

MSE Mean Square Error

mpp mutations per pixel

NEE Next Event Estimation

PDF Probability Density Function

PG Path Guiding

PSSMLT Primary Sample Space Metropolis Light Transport

PT Path Tracing

QMC Quasi Monte Carlo

RE Replica Exchange

ReSTIR Reservoir-based Spatio-Temporal Importance Resampling

RJMLT Reversible Jump Metropolis Light Transport

SMAPE Symmetric Mean Absolute Percentage Error

SMCMC Stratified Markov Chain Monte Carlo

spp samples per pixel

List of Symbols

A	Area
A_{image}	Area of the image, i.e., the total number of pixels
$\mathbf{x}$	Point or vertex in the three dimensional space
$\mathbf{X}$	Path composed by several vertices $\mathbf{x}$
u	Canonical random number, i.e., random and uniformly distributed in the range $[0, 1)$
$\mathbf{u}$	Vector of uniformly distributed random numbers u
$\mathbf{U}$	Array of vectors of uniformly distributed random numbers $\mathbf{u}$
$\boldsymbol{\omega}$	Direction vector
$\boldsymbol{\omega}^{\perp}$	Projected direction $\boldsymbol{\omega}^{\perp} = \lvert\mathbf{n_x} \cdot \boldsymbol{\omega}\rvert\boldsymbol{\omega}$ with respect of the normal direction $\mathbf{n_x}$ of $\mathbf{x}$
$\boldsymbol{\omega}_i$	Incoming direction (by convention pointing outwards)
$\boldsymbol{\omega}_o$	Outgoing direction
λ	Wavelength of electromagnetic radiation
Φ	Radiant Flux
E	Irradiance
L	Radiance
S	Any spectral quantity, in general a function, but in our case just three values
S^*	The scalar version of the spectral quantity, can be obtained by any transformation
$L_e(\mathbf{x}, \boldsymbol{\omega})$	Emitted radiance from a light source at $\mathbf{x}$ towards direction $\boldsymbol{\omega}$
$L_i(\mathbf{x}, \boldsymbol{\omega}_i)$	Incoming radiance arriving at vertex $\mathbf{x}$ from direction $\boldsymbol{\omega}_i$
$L_o(\mathbf{x}, \boldsymbol{\omega}_o)$	Outgoing radiance from a vertex $\mathbf{x}$ towards direction $\boldsymbol{\omega}_o$
$\rho(\mathbf{x}, \boldsymbol{\omega}_o, \boldsymbol{\omega}_i)$	Bidirectional Scattering Distribution function: Models the scattering of light arriving at point $\mathbf{x}$ from direction $\boldsymbol{\omega}_i$ towards direction $\boldsymbol{\omega}_o$
I_j	Average measured spectral radiance of pixel j. The value we want to calculate!
R_j	The average measured spectral radiance of pixel j of the reference image
$h_j(\mathbf{r})$	Filter function of all the raster positions $\mathbf{r}$ of pixel j
$W_e(\mathbf{x}, \boldsymbol{\omega})$	Emitted importance from lens vertex $\mathbf{x}$ towards $\boldsymbol{\omega}$
$W_e^j(\mathbf{x}, \boldsymbol{\omega})$	Emitted importance that considers the filter of pixel j
$V(\mathbf{x} \leftrightarrow \mathbf{y})$	Visibility function between two vertices
$G(\mathbf{x} \leftrightarrow \mathbf{y})$	Geometric term of two vertices
$\mathbf{x} \to \mathbf{y}$	Sequence of two vertices
$\mathbf{x} \to \mathbf{y} \to \mathbf{z}$	Sequence of three vertices
$f(\mathbf{x} \to \mathbf{y})$	Equivalent to $f(\mathbf{x}, \boldsymbol{\omega})$ with $\boldsymbol{\omega} = \frac{\mathbf{y}-\mathbf{x}}{\lVert\mathbf{y}-\mathbf{x}\rVert}$
$f(\mathbf{x} \to \mathbf{y} \to \mathbf{z})$	Equivalent to $f(\mathbf{y}, \boldsymbol{\omega}_o, \boldsymbol{\omega}_i)$ with $\boldsymbol{\omega}_o = \frac{\mathbf{y}-\mathbf{z}}{\lVert\mathbf{y}-\mathbf{z}\rVert}, \boldsymbol{\omega}_i = \frac{\mathbf{y}-\mathbf{x}}{\lVert\mathbf{y}-\mathbf{x}\rVert}$
luminance(.)	Returns the luminance of a spectral quantity
Ω	Sample space/Integration domain
$(\Omega, \mathcal{F})$	Measurable space with $\mathcal{F}$ a σ-algebra of Ω
μ	Measure function
$\mathbb{P}$	Probability measure function
$(\Omega, \mathcal{F}, \mathbb{P})$	Probability measure space
X	Random variable
$\mathbb{P}_X$	Induced probability measure function by random variable X
p_f	Normalized function $p_f = \frac{f(\mathbf{x})}{\int_{\Omega} f(\mathbf{x}) d\mu(\mathbf{x})}$
p_X^{μ}	Probability density function (pdf) of random variable X over measure μ
P_X^{μ}	Cumulative density function (cdf) of random variable X over measure μ
p^{ω}	Probability over solid angle domain

$p^{\perp}$	Probability over projected solid angle domain
p^{u}	Probability over primary sample space domain
$E[X]$	Expected value of random variable X
$V[X]$	Variance of random variable X
$\text{supp}(.)$	Support of a function
I	Integral function that wants to be calculated, in rendering $I = I_j$
$I^{(N)}$	Monte Carlo estimator of N samples
$I^{(N_1,\ldots,N_m)}$	Monte Carlo estimator with MIS using m sampling techniques
$w_i(\mathbf{x})$	MIS weight of technique strategy i applied to sample $\mathbf{x}$
$p(\mathbf{x} \to \mathbf{y})$	Markov Chain transition probability density, also denoted as $p(\mathbf{x}, \mathbf{y})$ or $p(\mathbf{y}\|\mathbf{x})$
$T(\mathbf{x} \to \mathbf{y})$	Probability density function to get a new sample $\mathbf{y}$ from $\mathbf{x}$ (a mutation)
$\alpha(\mathbf{x} \to \mathbf{y})$	Acceptance probability density
$\text{MSE}[I, R]$	Mean Squared Error of rendered image I with respect of reference image R
$\text{SMAPE}[I, R]$	Symmetric Mean Absolute Percentage Error of I with respect to R

Contents

Chapter 1

Introduction

Computer graphics is the branch of computer science that specializes in the creation and manipulation of images on a computer [MS16]. The creation of an image implies the use of physical and mathematical models to represent certain aspects of reality into the created image. For example, the behavior of light [PJH16], fluids [Bri15], atoms [RP07], galaxies [Vin+11], or even concepts like emotions [Fus+09] can be visualized (using methods like fMRI) by a computer generated image using the appropriate algorithms and theoretical frameworks, such as optics and electromagnetism.

In concrete, we use the theory of a physics branch that describes a given phenomenon and run a rendering process to synthesize an image that is a tangible representation of the modeled event. Because of this, computer graphics have several applications such as:

- Medicine: Generation of precise medical imaging visualizations that accurately portrays structures of the human body, using the data obtained from body measurements. One example of this are computed tomographies (CT), which create visualizations of internal parts of the body using data gathered by x-rays, which is algorithmically (e.g., Filtered BackProjection [Sch+20]) manipulated by a computer in order to produce the final render.

- Engineering and architecture: Computer aided design (CAD) and manufacturing of engineering designs such as: build modelling and mechatronics modelling (see Figure 1.1a).

- Augmented Reality (AR) and Virtual Reality (VR).

- Movies: Live-action movies use computer graphics to enhance the viewer's experience by adding Computed Generated Imagery (CGI) and Visual Effects (VFX), also there are animated films completely made by computer images. A completely animated film of one and a half hours needs 162000 rendered images, assuming 30 frames per second.

- Videogames: In this form of entertainment, images are generated in real time, with an expected frame rate of 30 frames per second, getting to values as high as 400 frames per second, all of this in a dynamic environment that is continuously changing (see Figures 1.1b and 1.1c).

On top of all the things already said, because the images created using computer graphics can be physically accurate, they can be employed to generate synthetic data for machine learning agents [Sch19].

In this thesis we will focus in the phenomenon of light transport, as it is of utmost importance to have a form of accurately simulating and depicting the images induced by the light, because by doing this, we can approximately recreate the reality we visually perceive. Light transport rendering, is in a way, a "simulated camera" that can take pictures from both our reality and environments conceived by our imagination.

The physical model of light used in this thesis assumes a geometric optics approach in which light behaves as a particle, ignoring its wave like nature. Light will be assumed to travel as particles bouncing around a scene, interacting with objects that will provoke changes in the directions of the particles. This bouncing will be modeled by the *light transport equation*, also known as the *rendering equation* [Kaj86] which is a highly dimensional, recursive, and full of discontinuities integral equation that in almost all the cases has no analytical solution.

(a) Pavillion-Barcelona scene from pbrt 3 [PJH16].

(b) Red Dead Redemption 2 (©2018 Rockstar Games).

(c) Cyberpunk 2077 (©2020 CD Projekt Red)

Figure 1.1: Computer graphics are widely used in engineering and architectural designing (a) and in video games (b,c).

Solving the light transport equation for every pixel of the rendered image in a fast and efficient way is a tremendously difficult problem as most traditional numerical methods struggle with high dimensions and discontinuities. Because of this, the preferred numerical techniques of integration are the Monte Carlo integration methods, which are a family of stochastic numerical methods that consist in taking random samples over the whole integration domain in order to approximate the respective integral.

The final image (render) produced by the Monte Carlo (MC) estimators (usually one for each pixel) has some noise, defined as random *variations* (statistical variance) in color and brightness with respect to the real results (expected values) of the pixels. Images with more variance, i.e, more noise, have a bigger error, and each MC estimator has its own variance. One naive way of reducing the noise of the rendered images is to add more samples, as the error of an MC estimator is reduced with respect of the square root of the number of samples. However, raising the number of samples directly increases the computational effort as each sampled light path has a given computational cost.

An alternative way of reducing the variance of the MC estimator is done by modifying the way of exploring the sample space, as MC estimators that draw samples in a proportional amount to the integrand converge faster, as we would be allocating more time exploring high contribution values of the function, and less computational effort exploring regions of the integration domain that contribute less to the final result. The most energetic paths have higher importance and will contribute the most to the final measured image.

The word "sampling" is almost equivalent in the Monte Carlo integration context to the word "exploring", and the light paths can be seen as the final result of a sequence of samplings to select random vertices, directions or decisions.

Paths can be thought as a topological structure on the scene, with edges that connect each vertex in succession, starting from a light vertex (emitter vertex), and ending with a vertex on the camera. Multiple path topologies are present in the scene and are the soul of the Monte Carlo sampling in light transport research (see Figure 1.2). The integration domain of the rendering equation can then be visualized as the set $\mathcal{P}$ with infinite cardinality, of all possible light carrying paths.

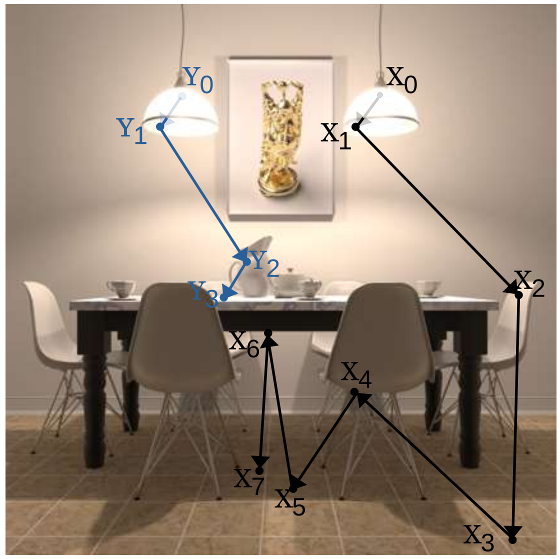


Figure 1.2: The samples of light transport rendering look like a sequence of sampled vertices in the scene which constitute a light path; in this figure there are two light paths $\mathbf{X} = (\mathbf{x}_0, ..., \mathbf{x}_8), \mathbf{Y} = (\mathbf{y}_0, ..., \mathbf{y}_4)$ as an example, noting that there are paths more complicated than others, and that $\mathbf{x}_8, \mathbf{y}_4$ are not shown in the figure, but are lens vertices. The main problem in light transport research is to find ways to sample proportionally to the importance carried by the full net of possible paths in the scene. Image taken from pbrt-v3 [PJH16].

One of the main objectives of light transport research, and also of this thesis is to find efficient ways to sample the network of paths $\mathcal{P}$ of a scene, in order to speed up the convergence of the MC estimator, or in other words, trying to get paths (samples) in a proportional way to the energy of the paths in this complex path network $\mathcal{P}$ that is induced by the materials, objects and lights in the scene.

1.1 Difficult light transport

Some paths in the path-space $\mathcal{P}$ are extremely difficult to explore, as complicated paths are less likely to be found by mere chance, even when we take into account the highly likely directions defined by both the materials and the lights on the scene. Some examples of difficult paths are the following:

- **Paths with visibility issues**: These type of paths are present when we can't connect two vertices because there is an object between the two vertices. A typical example of this would be to put a wall with a small aperture, and a light behind the wall, generating a situation in which most of the light paths can't pass the other side of the wall (see Figure 1.3a).

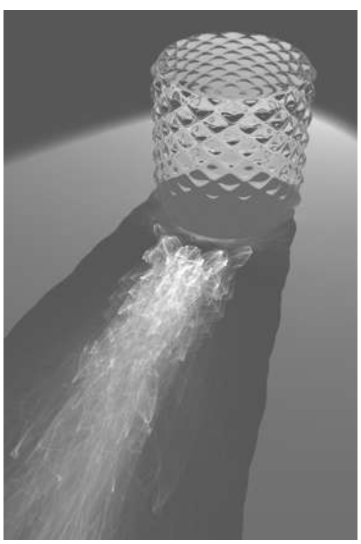
- **Difficult paths induced by the materials on the scene**: Specular materials scatter the light in one (reflection) or at most two (reflections and transmissions) possible outgoing directions, for each incoming direction. Specular and near-specular materials then introduce exploration issues as now the material restricts the possible outgoing directions making the reconnection with another vertex extremely difficult, if not impossible in the perfect specular case. Two examples of difficult paths of these type are:

    1. The *caustic path*: It starts in a light vertex L and then reach a chain of one or more ($^+$) specular S vertices connected to a diffuse D vertex and ending in the camera/eye E, that is, LS^+DE (see Figure 1.3b).

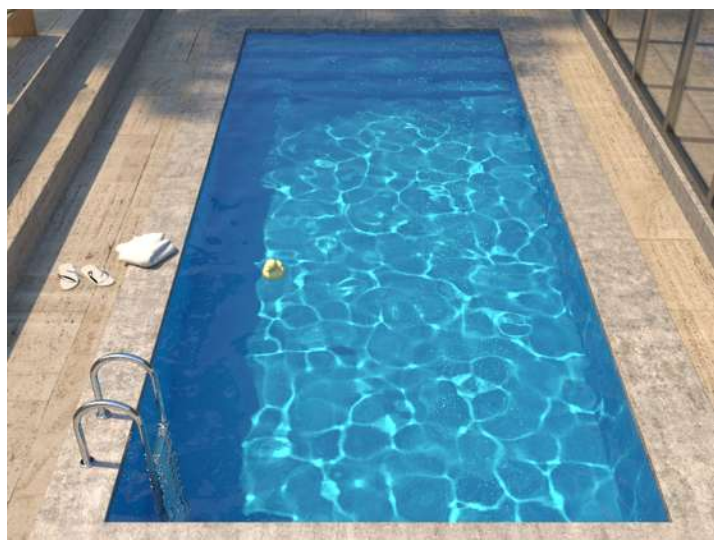
    2. *Specular multi-chain paths*: These type of paths are caustics seen through a specular surface, and are of the form $(D \vee L)DS^+DS^+E$, i.e., a repetition of specular chains with a diffuse vertex between chains DS^+. These type of paths are common in dielectric material such as glasses and water (see Figure 1.3c).

Being able to accurately explore the "integration domain of difficult paths" is a challenging but rewarding task, as we get closer to simulate a wider range of scenes, reducing one step at a time the inability to use more daring lighting scene configurations which was hindering human creativity at the moment of designing scenes.

(a) **Veach ajar** scene with only diffuse materials rendered using one million samples per pixel, with the path integrator of pbrt-v3 [Vea97; PJH16; Bit16]. The light transport on this scene could be made even more difficult adding specular materials instead of diffuse materials, incorporating an extra difficulty to the occlusion problem by reducing the number of possible scattering directions due to the material's nature.



(b) Caustic paths LS^+DE scene [PJH16] being originated by two type of lights, an infinite light (sky), and a point light that causes the light pattern on the floor. Note that a path tracer can't connect the specular glass to the point light.



(c) A pool scene [Vor+14] with a diffuse floor and a specular "lid" (the water), because of this, it has very difficult paths as there are multiple chains of paths of the form DS^+ concatenated (multiple bounce inside the pool, between the water and the diffuse floor), for example, $LS^+DS^+DS^+E$, LS^+DS^+E, or in general any concatenation of that style, these paths are caustics (LS^+D) seen through a specular surface (S^+E).

Figure 1.3: The two main type of difficult paths in rendering, due to visibility and material issues. Note how even a scene with only diffuse materials can be complicated, and also a scene with no visibility issues could be also a complicated scene.

1.2 Problem statement

Standard Monte Carlo integration methods such as path tracing and bidirectional path tracing can't handle that well difficult paths, and have the main disadvantage that they don't use in any way the information of the already sampled paths. They are memory-less, which is a huge waste of information, as it is, of course, a very reasonable intuitive assumption to say that similar paths (even if they are difficult) might have a similar importance.

One way of avoiding this waste of information is to use *global exploration*, which consists in storing the positional and directional light information that each path sample provides, in a data structure, to be able to globally *guide* new samples toward high importance zones in the scene that we already learnt [MGN17].

Another approach is to use *local exploration*, a technique that focuses on exploring similar paths by modifying the current main path, which means that each new sample only depends on the previous sample, defining a Markov Chain process. The MC estimator that gets its samples using a (ergodic) Markov Chain is called a *Markov Chain Monte Carlo* (MCMC) estimator. If the new samples are obtained modifying in a direct way the topology of the current path, this process is called *Path-space Metropolis Light Transport* [VG97]. Eric Veach, the author of this method, called large modifications of the current path *mutations* and small modifications, *perturbations*. On the other hand, if the new samples are obtained by modifying the random numbers that are used to sample the paths (the *primary sample space*), instead of the paths themselves, this technique is called *Primary Sample Space Metroplis Light Transport* [Kel+02]. Kelemen, the author of this technique, called large modifications of the current random number *large mutations* and small modifications, *small mutations*. Notice that in a way, a large mutation is the primary sample space equivalent of a path-space mutation, and a small mutation is the primary sample space equivalent of a path-space perturbation.

Path guiding has the following disadvantages: it is complex to implement, requires a lot of memory, and has a lot of user parameters to tune. Moreover, high contribution paths initially need to be found by mere chance, in order to be added to the guiding pool.

MCMC rendering methods, on the other hand, despite of being easier to implement in the primary sample space, have the disadvantage that they only use the information of the current path, ignoring all the previously learnt paths. Additionally, as the chain needs to be ergodic, *large mutations* are required to find paths globally by pure luck, which is a less efficient process than using path guiding. Finally, they suffer an uneven stratification problem, as the chains can visit any pixel of the image, provoking that brighter pixels (e.g, the door's aperture in Figure 1.3a) receive more samples than darker pixels.

From the previous discussion, we can conclude that two main sources of variance for MCMC methods are its inefficient global exploration (large mutation), and their uneven stratification, as of course, darker pixels might contain several difficult light paths. Thus, solving these two problems will reduce the variance of the MCMC estimator.

An additional advantage of solving the stratification problem of the MCMC algorithms is that it will make the noise less visually apparent to humans, as humans' visual systems are highly sensitive to contrast changes.

1.3 Hypothesis and objectives.

The two hypothesis of this thesis are:

1. A global exploration algorithm combined with a MCMC estimator will improve the efficiency of the MCMC algorithm, and we believe this is the case because large mutation will be now guided by the learnt past paths, instead of just roaming through the whole integration domain led by chance alone.

2. Stratification MCMC algorithms like the technique proposed by Gruson et al. [GWH20] relies on a simple path-space operator. We can further improve the performance of this rendering algorithm by using a more complex path-space operator.

The objective of this thesis is to improve MCMC local exploration through the combination of global exploration methods, and to improve the stratified MCMC integrator proposed by Gruson et al. [GWH20].

1.4 Chapter description

- In chapter 2 we introduce the mathematical basis of the MC estimator and how to accelerate its convergence, alongside with the rendering equation, and the methods of path tracing, bidirectional path tracing and path guiding for global exploration.
- In chapter 3 we delved into the MCMC algorithms applied to rendering, with some of their advantages and disadvantages, and how these disadvantages can be alleviated. It also includes the way in which Gruson et al. [GWH20]. solves the stratification problem on his algorithm called stratified Markov Chain Monte Carlo.
- In chapter 4 we present our proposed algorithms and their results.
- In chapter 5 we present our final conclusions and future work.

Chapter 2

Preliminaries

The studied phenomenon in this thesis will be the transport of *light*, defined as the part of the electromagnetic radiation spectrum that humans can see, ranging approximately from a wavelength of 390 nanometers to a wavelength of 780 nanometers [Hec16]. Visible light, or simply light will be modeled as particles being shot from a source in one or more directions, travelling in a three dimensional world space (*scene*), interacting with objects (*geometric primitives* or *shapes*), each one with distinctive physical properties (*materials*) that cause these light particles to change their trajectories in a different way depending of the type of material. We will only care about the light perceived by an observer (*camera*), that has a set of points (*lens*), which distribute this incoming light into a two dimensional sensor (*film*), ending with a measured projected two dimensional rendered image that was obtained from a physical phenomenon in a three dimensional world.

The equation that models the transported light is called the light transport equation or the rendering equation [Kaj86], and the equation that models how this light is distributed by a lens into a sensor (film) is called the measurement equation [Vea97; PJH16]. Both of these equations as we will see later, are integral equations, which will be solved by a numerical *integrator*.

If we want to obtain photorealistic renders, a physically accurate mathematical model must be used, in our case, the model utilized to study the light transport phenomenon is based on *radiometry* [PJH16], defined as the phenomenological study of the propagation of electromagnetic radiation in the environment. Radiometry is based on the *geometric optics* assumption, that is, that light behaves as a particle flowing through space in straight lines, also known as rays. This unfortunately ignores the wave nature of light, a crucial component to explain phenomena like diffraction, interference, polarization and iridescence.

The radiative transfer model of light transport rendering also requires the following important assumptions:

- *Linearity*: Even if there are multiple light sources on the scene, the behavior of each light is not affected by the presence of the other lights, which means that if ℓ lights are present on a scene, the resulting rendered image is equivalent as if ℓ images with just one light were rendered and then were summed together, i.e., $\text{Image}\left[\bigcup_{i=1}^{\ell} \text{Light}_i\right] = \bigcup_{i=1}^{\ell} \text{Image}\left[\text{Light}_i\right]$. Not meeting this requirement would be equivalent to assume that light interference is happening, which would mean that the emissions of all the lights would be affected by all the other lights present in the scene, phenomenon that occurs at the wave level. We can safely ignore interference without losing any significant macroscopic lighting effects because everyday light is an *incoherent light* [Hec16], which means that all the emitted waves have chaotic and different wave phases. The randomness of incoherent light "cancels out" interference at the macroscopic level.

- *Energy conservation*: When light scatters from an object or a medium, the scattered light can't have more energy than its initial value. Note that the final energy *could* be less than the initial energy because of absorption and heat phenomena. We only care about not creating more energy out of nowhere which is physically impossible.

- *Steady state*: It is assumed that light's radiance distribution does not change over time (it is already at equilibrium), a quite realistic assumption as light can be considered to be an instantaneous phenomenon in non-astronomical distances due to its high speed c.

Both the rendering and measurement equations are integral equations that need to be solved numerically. The first section of this chapter is going to introduce the the rendering equation. After this, in the second section we will present the mathematical theory to solve integrals using Monte Carlo integration. In the third section, we will present ways of applying the Monte Carlo methods to the rendering equation. Finally in the fourth section we will show the chapter's conclusions.

2.1 Physical preliminaries: The light transport equation.

In this work we are going to use the radiative transfer model, that assumes energy conservation, linearity and presumes that light travels as a particle. This section will be divided in three subsection. In the first one we will talk about the physical quantities needed to understand the rendering equation. In the second sub-section because the rendering equation is an integral equation we will introduce the integration domains used in computer graphics. And last but not least, in the third sub-section we will present the measurement and rendering equations.

2.1.1 Physical quantities

In the following list we present the necessary concepts in order to be able to derive the rendering equation [Vea97; PJH16]:

1. *Energy* Q: The capacity of doing work, each light's particle (*photon*) has a particular wavelength λ with an associated energy Q:
$$Q = \frac{h\lambda}{c}, \tag{2.1}$$
where c is the speed of light and h is Planck's constant. The units of energy Q are joules (J).

2. *Radiant Flux* Φ: Rate of transfer of energy through a surface per unit of time:
$$\Phi = \frac{\mathrm{d}Q}{\mathrm{d}t}. \tag{2.2}$$
The *total* emission of the light source is generally described in terms of its flux. Flux units are Watts (W).

3. *Irradiance* E: A measurement of the photons passing through an area per time, i.e., the rate of transfer of energy per area per unit of time:
$$E(\mathbf{x}) = \lim_{\Delta A \to 0} \frac{\Delta\Phi(\mathbf{x})}{\Delta A} = \frac{\mathrm{d}\Phi(\mathbf{x})}{\mathrm{d}A(\mathbf{x})}, \tag{2.3}$$
so consequently the radiant flux over any area is the integral
$$\Phi(\mathbf{x}) = \int_A E(\mathbf{x})\,\mathrm{d}A(\mathbf{x}) \tag{2.4}$$
where the points $\mathbf{x}$ on the integral are located in the area A. The units of irradiance are W/m^2. Note that the irradiance at a point $\mathbf{x}$ changes with respect to the light position, as it's not the same to receive flux in a surface perpendicular to the rays of light $E = \frac{\Phi}{A}$, or receive flux in a surface with some degree of inclination $E = \frac{\Phi|\cos\theta|}{A}$.

4. *Radiance* L: One of the most important concepts in rendering, this is the quantity that is returned by rendering integrators. Radiance measures the change of radiant flux when areas and directions change, it is a quantity that completely encompasses the radiometric behavior of light with directional and positional change:
$$\begin{gathered} L(\mathbf{x},\boldsymbol{\omega}) = \frac{\mathrm{d}E_{\boldsymbol{\omega}}(\mathbf{x})}{\mathrm{d}\boldsymbol{\omega}} = \frac{\mathrm{d}\Phi(\mathbf{x},\boldsymbol{\omega})}{\mathrm{d}\boldsymbol{\omega}\,\mathrm{d}A^{\perp}(\mathbf{x})} = \frac{\mathrm{d}\Phi(\mathbf{x},\boldsymbol{\omega})}{\mathrm{d}\boldsymbol{\omega}^{\perp}\,\mathrm{d}A(\mathbf{x})} = \frac{\mathrm{d}E(\mathbf{x},\boldsymbol{\omega})}{\mathrm{d}\boldsymbol{\omega}^{\perp}} \\ E_{\boldsymbol{\omega}}(\mathbf{x}) \equiv \frac{E(\mathbf{x})}{|\mathbf{n}\cdot\boldsymbol{\omega}|} \equiv E(\mathbf{x},\boldsymbol{\omega}) \quad \mathrm{d}A^{\perp}(\mathbf{x}) \equiv \mathrm{d}A(\mathbf{x})|\mathbf{n}\cdot\boldsymbol{\omega}| \quad \mathrm{d}\boldsymbol{\omega}^{\perp} \equiv \mathrm{d}\boldsymbol{\omega}|\boldsymbol{\omega}\cdot\mathbf{n}|, \end{gathered} \tag{2.5}$$

with units $\frac{W}{m^2 sr}$, where sr are steradians, $E_{\boldsymbol{\omega}}$ is the irradiance at a surface perpendicular to the normalized direction vector $\boldsymbol{\omega}$, and $\mathbf{n}$ is the normal vector of the surface in the current point.

The quantity $A^{\perp}$ is called the *projected area* which is an area with normal direction $\boldsymbol{\omega}$, obtained by performing the vector *projection* $\text{proj}_{\boldsymbol{\omega}}\mathbf{n} = |\mathbf{n}\cdot\boldsymbol{\omega}|\boldsymbol{\omega}$ (see Figure 2.1), which tells us the effective area that is receiving flux. The vector $\boldsymbol{\omega}^{\perp}$ is called the *projected solid angle* and its geometrical interpretation can be given considering the vector projection $\text{proj}_{\mathbf{n}}\boldsymbol{\omega} = |\mathbf{n}\cdot\boldsymbol{\omega}|\mathbf{n}$ which projects the area defined by all the directions $\boldsymbol{\omega}$ into the plane with normal $\mathbf{n}$ (see Figure 2.1), telling us in a way, the amount of effective incoming flux directions that can enter our plane at $\mathbf{x}$.

Now that radiance has been presented, the irradiance E at a point $\mathbf{x}$ located in a surface can be expressed in terms of the radiance as:

$$E(\mathbf{x}) = \int_{\Omega} L_i(\mathbf{x}, \boldsymbol{\omega})|\mathbf{n}\cdot\boldsymbol{\omega}| \,\mathrm{d}\boldsymbol{\omega} = \int_{\Omega} L_i(\mathbf{x}, \boldsymbol{\omega}) \,\mathrm{d}\boldsymbol{\omega}^{\perp}, \tag{2.6}$$

where L_i is the *incident radiance*.

5. *Bidirectional Scattering Distribution Function* (BSDF) ρ: This function describes the amount of light scattered in a point $\mathbf{x}$, from a given incident direction $\boldsymbol{\omega}_i$ to an outgoing direction $\boldsymbol{\omega}_o$ (both pointing outwards by convention, see Figure 2.1):

$$f_{\text{BSDF}}(\mathbf{x}, \boldsymbol{\omega}_o, \boldsymbol{\omega}_i) \equiv \rho(\mathbf{x}, \boldsymbol{\omega}_o, \boldsymbol{\omega}_i) = \frac{\mathrm{d}L_s(\mathbf{x}, \boldsymbol{\omega}_o)}{\mathrm{d}E_i(\mathbf{x})} = \frac{\mathrm{d}L_s(\mathbf{x}, \boldsymbol{\omega}_o)}{L_i(\mathbf{x}, \boldsymbol{\omega}_i)\,\mathrm{d}\boldsymbol{\omega}_i^{\perp}}, \tag{2.7}$$

where $L_s(\mathbf{x}, \boldsymbol{\omega}_o)$ is the exitant scattered radiance at point $\mathbf{x}$ towards direction $\boldsymbol{\omega}_o$ and $L_i(\mathbf{x}, \boldsymbol{\omega}_i)$ is the incident radiance at point $\mathbf{x}$ coming from $\boldsymbol{\omega}_i$, also notice that:

$$\rho(\mathbf{x}, \boldsymbol{\omega}_o, \boldsymbol{\omega}_i)L_i(\mathbf{x}, \boldsymbol{\omega}_i) = \frac{\mathrm{d}L_s(\mathbf{x}, \boldsymbol{\omega}_o)}{L_i(\mathbf{x}, \boldsymbol{\omega}_i)\,\mathrm{d}\boldsymbol{\omega}_i^{\perp}} L_i(\mathbf{x}, \boldsymbol{\omega}_i) = \frac{\mathrm{d}L_s(\mathbf{x}, \boldsymbol{\omega}_o)}{\mathrm{d}\boldsymbol{\omega}_i^{\perp}}, \tag{2.8}$$

so the multiplication of the BSDF times the incident radiance equals the change of the scattered radiance with respect to the projected incoming directions. Integral $\int_{\Omega} \rho(\mathbf{x}, \boldsymbol{\omega}_o, \boldsymbol{\omega}_i)L_i(\mathbf{x}, \boldsymbol{\omega}_i)d\boldsymbol{\omega}_i^{\perp}$ represents the fraction of the incoming radiance scattered to direction $\boldsymbol{\omega}_o$ and its units are the same as radiance. The units of the BSDF are inverse steradians ($1/sr$) and it's a positive function $\rho(\mathbf{x}, \boldsymbol{\omega}_o, \boldsymbol{\omega}_i) > 0\ \forall \boldsymbol{\omega_i}, \boldsymbol{\omega}_o \in \Omega$ whose values are usually obtained experimentally in an optics laboratory.

Each object in the world has a distinctive appearance determined by the material it is made of and each material interacts with the light in a different way, which is described by the BSDF. Materials that can scatter light in every possible direction of the hemisphere are known as *diffuse materials*, materials that can only scatter in one, or at most two directions according to the *law of reflection* $\mathbf{n}\cdot\boldsymbol{\omega}_o = \mathbf{n}\cdot\boldsymbol{\omega}_i$ and the *law of refraction* (*Snell's law*) are called *specular materials* and materials that are in-between these two cases are called *glossy materials*. The first three materials that are implemented in a render system are usually called *matte* (a Lambert diffuse transmitter in all directions), *conductors* (reflective *metals* in just one direction) and *glasses* (materials that reflect and refract in just two possible directions according to Snell's law), alongside their glossy versions, usually called *microfacets*.

There is a subtle point in the previous physical quantities, that can be observed from the beginning looking at the equation of energy 2.1, all the quantities actually depend also of the wavelength, that is, $Q(\lambda), \Phi(\mathbf{x}, \lambda), E(\mathbf{x}, \lambda), L(\mathbf{x}, \boldsymbol{\omega}, \lambda), \rho(\mathbf{x}, \boldsymbol{\omega}_o, \boldsymbol{\omega}_i, \lambda)$. The functions that depend on wavelength define a Spectral Power Distribution (SPD) $S(\lambda)$. According to the trichromatic theory of color, human visual systems perceive color by three types of cells called cones which are receptive to red, green and blue; the rest of colors are generated by a combination of these colors. Photometry is the science that studies how to transform the radiometric quantities $S(\lambda)$ of the visible spectrum into (r, g, b) quantities related to human perception, following the tristimulus theory of color:

$$r = \int_{\lambda} R(\lambda)S(\lambda)\,\mathrm{d}\lambda; \qquad g = \int_{\lambda} G(\lambda)S(\lambda)\,\mathrm{d}\lambda; \qquad b = \int_{\lambda} B(\lambda)S(\lambda)\,\mathrm{d}\lambda, \tag{2.9}$$

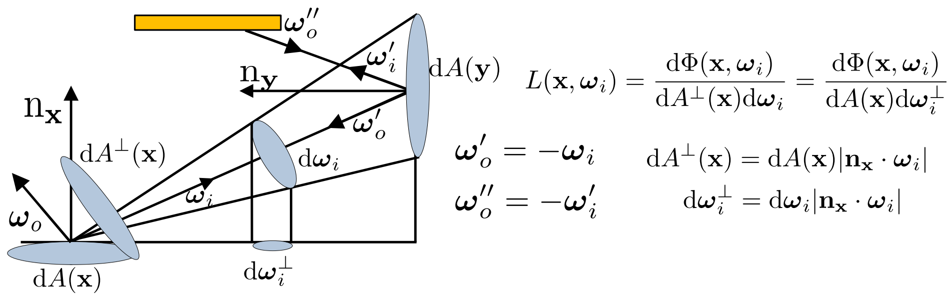


Figure 2.1: In rendering the projected area $A^{\perp}$, and the projected solid angle $\boldsymbol{\omega}^{\perp}$ are necessary because we need to consider that the incoming flux $\Phi(\mathbf{x}, \boldsymbol{\omega}_i)$ reaching the surface point $\mathbf{x}$ from direction $\boldsymbol{\omega}_i$ varies depending of the orientation of the surface at $\mathbf{x}$ with normal $\mathbf{n_x}$. There are two ways of visualizing radiance, as a derivative over $\mathrm{d}A^{\perp}\,\mathrm{d}\boldsymbol{\omega}_i$ which is quite intuitive as this is equivalent to calculating the effective area that is receiving flux, or as a derivative over $\mathrm{d}A\,\mathrm{d}\boldsymbol{\omega}_i^{\perp}$, which is a little less intuitive but it is equivalent to projecting the incoming flux directions over the true (non projected) area of the surface. Notice something quite important, a change of direction $\mathrm{d}\boldsymbol{\omega}_i$ implies a change of area $\mathrm{d}A(\mathbf{y})$.

where $R(\lambda), G(\lambda), B(\lambda)$ are experimentally obtained SPDs, data usually given by the International Commission on Illumination (CIE, short for its initials in french). The type of rendering that uses SPDs is called *spectral rendering*.

Photometry describes a very important quantity called *luminance*, this quantity measures how bright a spectral power distribution is perceived by a human observer, that is, how sensitive are the human eyes to red, green and blue, which is a scalar obtained from the linear combination of the three values on the spectral channels r, g, b obtained from the SPD:

$$\text{luminance}(f) = 0.21267 f_r + 0.71516 f_g + 0.072169 f_b, \tag{2.10}$$

where its specific values can be obtained starting from the CIA-1931 chromaticity values (hue, colorfulness x, y pairs) for the three primary colors $x_R, y_R, x_G, y_G, x_B, y_B$ and the chromaticity of the white point which in this case is the CIE Standard Illuminant D65 (light source of 6500K) x_W, y_W. The four aforementioned pair of chromaticities can be transformed to the CIE-XYZ color space using the relations:

$$X_i = \frac{x_i}{y_i} Y_i \equiv \hat{X}_i Y_i, \quad Y_i = 1 Y_i, \quad Z_i = \frac{1 - x_i - y_i}{y_i} Y_i \equiv \hat{Z}_i Y_i, \quad i \in \{W, R, G, B\}, \tag{2.11}$$

we have several unknown variables Y_i that can be calculated by noting that we want the luminance of the color $(1, 1, 1)$ to map to the unit luminance $Y_W = 1$ and we need all the Y_i to be calibrated with respect of the white point, so we have the system of equations:

$$\begin{pmatrix} \hat{X}_W \\ 1 \\ \hat{Z}_W \end{pmatrix} = \begin{pmatrix} \hat{X}_R & \hat{X}_G & \hat{X}_B \\ 1 & 1 & 1 \\ \hat{Z}_R & \hat{Z}_G & \hat{Z}_B \end{pmatrix} \begin{pmatrix} Y_R \\ Y_G \\ Y_B \end{pmatrix} = \begin{pmatrix} X_R + X_G + X_B \\ Y_R + Y_G + Y_B \\ Z_R + Z_G + Z_B \end{pmatrix}, \tag{2.12}$$

the whole transformation matrix (not just the second row) from RGB to XYZ can be obtained by simply noting

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} \hat{X}_R & \hat{X}_G & \hat{X}_B \\ 1 & 1 & 1 \\ \hat{Z}_R & \hat{Z}_G & \hat{Z}_B \end{pmatrix} \begin{pmatrix} Y_R & 0 & 0 \\ 0 & Y_G & 0 \\ 0 & 0 & Y_B \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix}, \tag{2.13}$$

the reader can easily plug the values on Table 2.1 into Equation 2.11 and solve the system of equations 2.12 to get *similar*[1] luminance coefficients that the ones shown in Equation 2.10 (used by pbrt).

It is quite important to be able to get appropriate scalar quantities from their spectral representation as there are times in rendering that scalars quantities must be used in some functions instead of

[1] There are discrepancies from the 5th decimal place onward, but this might be attributed to the fact that pbrt might have used slightly different constants, but the results of our explanation agree with the International Standard IEC 61966-2-1.

x_W	y_W	x_R	y_R	x_G	y_G	x_B	y_B
0.3127	0.3290	0.64	0.33	0.30	0.60	0.15	0.06

Table 2.1: Chromaticity coordinates for white point and RGB primaries.

the spectral quantities, and luminance is one of the usual scalar quantities widely used, alongside the maximum value, the sum, or any conveniently defined function depending of the three channels. Regardless of the specific scalar transformation, for a given spectral quantity S, its scalar version will be denoted as S^*.

Non spectral rendering is just going to assume that all the operations done by the rendering system are already photometry measurements (instead of radiometry measurements), so all the functions in non spectral rendering are defined as three channel quantities, for example a blue matte material BSDF would be $\rho(\mathbf{x}, \boldsymbol{\omega}_i, \boldsymbol{\omega}_o) = \left(\frac{0}{\pi}, \frac{0}{\pi}, \frac{1}{\pi}\right)$, or the emitted radiance from a diffuse area light source could be $L_e(\mathbf{x}, \boldsymbol{\omega}) = (10, 10, 10)$, where again, this units are just going to be stored in the final EXR image without any processing like the one done in Equation 2.9. All of the following discussion is going to focus in just one channel as the three channels are treated with exactly the same procedures, even the Monte Carlo integration is done using the same samples for the three channels, even though three Monte Carlo estimators could be independently used, one for each channel, making the rendering process unnecessarily three times slower.

2.1.2 Integration domains of the rendering equation

We will briefly explain the integral domains used to solve the rendering equation.

Solid angle domain Consists in integrating over the directions $\boldsymbol{\omega}$ defined by the vectors starting at point $(0, 0, 0)$ and ending in any point on the surface of the unit sphere $S^2 = \{\mathbf{y} \in \mathbb{R}^3 |\ \|\mathbf{y}\|^2 = 1\}$.

$$\mu_\omega(D) = \int_{\boldsymbol{\omega} \in D} \mathrm{d}\mu_\omega(\boldsymbol{\omega}) = \int_{\boldsymbol{\omega} \in D} \mathrm{d}\boldsymbol{\omega} \qquad \mathrm{d}\mu_\omega(\boldsymbol{\omega}) = \mathrm{d}\boldsymbol{\omega} \tag{2.14}$$

Projected solid angle domain This domain considers the current surface vertex $\mathbf{x}$ by multiplying the solid angle domain by $|\mathbf{n_x} \cdot \boldsymbol{\omega}|$:

$$\mu_\omega^\perp(D) = \int_{\boldsymbol{\omega}^\perp \in D} \mathrm{d}\mu_\omega^\perp(\boldsymbol{\omega}^\perp) = \int_{\boldsymbol{\omega} \in D} \mathrm{d}\boldsymbol{\omega}^\perp = \int_{\boldsymbol{\omega} \in D} \mathrm{d}\boldsymbol{\omega} |\mathbf{n_x} \cdot \boldsymbol{\omega}| \qquad \mathrm{d}\mu_\omega^\perp(\boldsymbol{\omega}^\perp) = \mathrm{d}\boldsymbol{\omega}^\perp \tag{2.15}$$

Area domain This domain is defined by all the points $\mathbf{y} \in \mathcal{S}$, with $\mathcal{S}$ the set of all the surface points in the scene

$$\mu_A(D) = \int_{\mathbf{y} \in D} \mathrm{d}\mu_A(\mathbf{y}) = \int_{\mathbf{y} \in D} \mathrm{d}A(\mathbf{y}), \quad \mathrm{d}\mu_A(\mathbf{y}) = \mathrm{d}A(\mathbf{y}) \tag{2.16}$$

The connection between the solid angle domain and the area domain can be understood by introducing the *ray casting function* $t : (\mathbb{R}^3, \Omega) \to \mathbb{R}^3$. This function traces a ray with origin $\mathbf{x}$ and direction $\boldsymbol{\omega}$, and returns the closest surface point $\mathbf{y} \in \mathcal{S} \cup \{\emptyset\}$ if it exists, if this not the case, the ray is said to have escaped the scene $\mathbf{y} = \emptyset$:

$$t(\mathbf{x}, \boldsymbol{\omega}) = \mathbf{y} \text{ with: } \quad \left(\frac{\mathbf{y} - \mathbf{x}}{\|\mathbf{y} - \mathbf{x}\|} = \boldsymbol{\omega} \right) \wedge (\mathbf{y} \in \mathcal{S} \cup \{\emptyset\}) \wedge (V(\mathbf{x} \leftrightarrow \mathbf{y}) = 1), \tag{2.17}$$

where V is the *visibility function* that returns one if there are no objects between the line that connects both points, and zero otherwise. This new point $\mathbf{y}$ is going to be located in the parametric line $\mathbf{R}$:

$$\mathbf{R}(\mathbf{x}, t) = \mathbf{x} + \boldsymbol{\omega} t, t \in [0, \infty), \tag{2.18}$$

the class formed by an origin `Point` and a direction `Vector` which can be evaluated at several points and intersected against the primitives of the `Scene` is a `Ray` object.

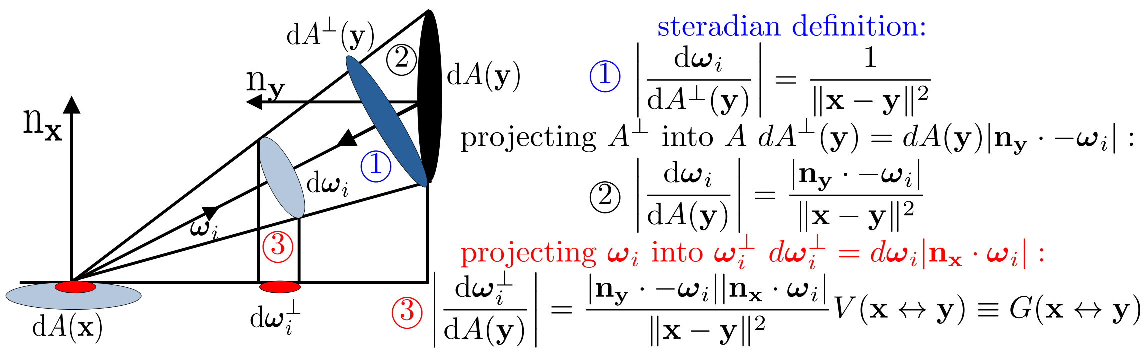


Figure 2.2: The transformation from the solid angle domain and the projected solid angle domain to the area domain. We start from the steradian definition ①, then after projecting $A^{\perp}$ into A with respect of $\mathbf{n_y}$ we obtain the change of variables between solid angle and area ②. After this we can also project $\boldsymbol{\omega}_i$ into $\boldsymbol{\omega}_i^{\perp}$ with respect of $\mathbf{n_x}$ to get the geometric term, that is, the transformation between the projected solid angle domain and the area domain ③.

Having said this, changes in direction $d\boldsymbol{\omega}$ imply changes in areas $dA(\mathbf{x})$, by just considering all the outcomes of the ray casting function $t(x, d\boldsymbol{\omega}) = dA(\mathbf{y})$. The transformation between the solid angle domain and the area domain is given by considering the differential steradian definition:

$$\left|\frac{d\boldsymbol{\omega}_i}{dA^{\perp}(\mathbf{y})}\right| = \frac{1}{\|\mathbf{x}-\mathbf{y}\|^2} \text{ (steradian def.)}; \qquad \left|\frac{d\boldsymbol{\omega}_i}{dA(\mathbf{y})}\right| = \frac{|\mathbf{n_y}\cdot -\boldsymbol{\omega}_i|}{\|\mathbf{x}-\mathbf{y}\|^2}, \tag{2.19}$$

while the transformation between projected solid angle domain and area domain is given by

$$\left|\frac{d\boldsymbol{\omega}_i^{\perp}}{dA(\mathbf{y})}\right| = \left|\frac{d\boldsymbol{\omega}_i}{dA(\mathbf{y})}\right| |\mathbf{n_x}\cdot\boldsymbol{\omega}_i| = \frac{|\mathbf{n_y}\cdot -\boldsymbol{\omega}_i||\mathbf{n_x}\cdot\boldsymbol{\omega}_i|}{r^2} V(\mathbf{x}\leftrightarrow\mathbf{y}) \equiv G(\mathbf{x}\leftrightarrow\mathbf{y}), \tag{2.20}$$

where G is called the *geomteric term*, and V is the visibility function (see Figure 2.2).

Primary sample space domain For completeness we briefly mention this domain, as its proper introduction will be done in section 2.2. In rendering we generate light paths stochastically using uniformly distributed random numbers in the interval $[0, 1)$ to sample directions, positions and decisions, creating in this way a mapping from the vector of random numbers $\mathbf{U}$ used to sample the light path $\mathbf{X}$, that is, $S(\mathbf{U}) = \mathbf{X}$. This hypercube of uniformly distributed random numbers $\boldsymbol{\mathcal{U}} = \cup_{k=1}^{\infty}[0, 1)^k$ is known as the *primary sample space* (PSS) domain [Kel+02].

$$\mu_u(D) = \int_{\mathbf{U}\in D} d\mu_u(\mathbf{U}) = \int_{\mathbf{U}\in D} d\mathbf{U}, \quad d\mu_u(\mathbf{U}) = d\mathbf{U} \tag{2.21}$$

2.1.3 The rendering and the measurement equations

The value for radiance can be calculated recursively, if energy is conserved and light is assumed to have a linear behavior, then for a given system, the difference between the energy going out E_o and coming in E_i must equal the difference between the emitted E_e and absorbed E_a energies [PJH16]. It is also assumed that this holds for the macroscopic level, that is, the radiant flux (rate of transfer of energy through a surface) is conserved:

$$\Phi_o - \Phi_i = \Phi_e - \Phi_a, \tag{2.22}$$

applying energy balance to a surface is equivalent to state that the exitant radiance $L_o(\mathbf{x}, \boldsymbol{\omega}_o)$ at point $\mathbf{x}$ towards direction $\boldsymbol{\omega}_o$ must equal the emitted radiance $L_e(\mathbf{x}, \boldsymbol{\omega}_o)$ at point $\mathbf{x}$ towards direction $\boldsymbol{\omega}_o$ plus the fraction of the incoming radiance $L_i(\mathbf{x}, \Omega)$ at point $\mathbf{x}$ coming from all the possible directions $\boldsymbol{\omega}_i \in \Omega$ that is scattered towards direction $\boldsymbol{\omega}_o$:

$$L_o(\mathbf{x}, \boldsymbol{\omega}_o) = L_e(\mathbf{x}, \boldsymbol{\omega}_o) + \int_{\Omega} \rho(\mathbf{x}, \boldsymbol{\omega}_o, \boldsymbol{\omega}_i) L_i(\mathbf{x}, \boldsymbol{\omega}_i) |\mathbf{n}\cdot\boldsymbol{\omega}_i| \, d\boldsymbol{\omega}_i, \tag{2.23}$$

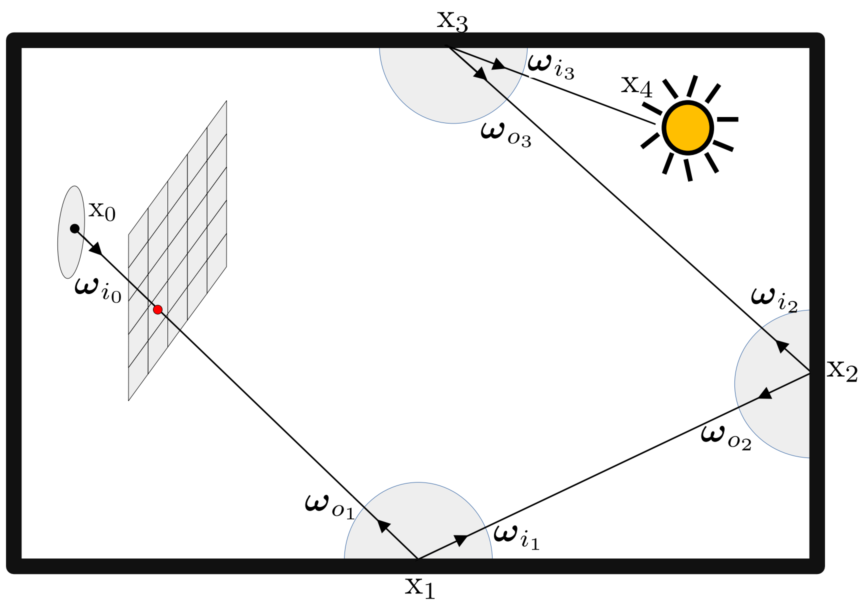


Figure 2.3: In this figure we apply several times the ray casting function starting from a lens point $\mathbf{x}_0$, and a direction $\boldsymbol{\omega}_{i_0}$ that pierces some raster position of some pixel on the film. We invoke the ray casting function several times, e.g., $t(\mathbf{x}_0, \boldsymbol{\omega}_{i_0}), t(\mathbf{x}_1, \boldsymbol{\omega}_{i_1})$,and so on. In this figure we end with the light path $\mathbf{X} = (\mathbf{x}_0, ..., \mathbf{x}_4)$ with $\mathbf{x}_4$ a light vertex. Notice that each vertex $\mathbf{x}_k$ is receiving an incoming radiance $L_{i_k}(\mathbf{x}_k, \boldsymbol{\omega}_{i_k})$, and it's also outputting an outgoing radiance $L_{o_k}(\mathbf{x}_k, \boldsymbol{\omega}_{o_k})$. In particular $L_{i_k}(\mathbf{x}_k, \boldsymbol{\omega}_{i_k}) = L_{o_{k+1}}(\mathbf{x}_{k+1}, -\boldsymbol{\omega}_{i_k})$.

where $\mathbf{n}$ is the normal at point $\mathbf{x}$ and no absorption is assumed. Equation 2.23 is known as the *light transport equation* or the *rendering equation* [Kaj86]. The emittance L_e is a value provided by the scene description given to the renderer, if the emittance of an object is greater than zero (in at least one channel) in any direction this means that the object is a light source.

A recursion can be defined to express the incoming radiance $L_i(\mathbf{x}, \boldsymbol{\omega}_i)$ in terms of the outgoing radiance $L_o(\mathbf{x}', -\boldsymbol{\omega}_i)$ of another point $\mathbf{x}'$ by using the ray casting function defined at Equation 2.17. Then thanks to the absence of a *participating medium*[2], the incident radiance L_i is equivalent to:

$$L_{i_k}(\mathbf{x}_k, \boldsymbol{\omega}_{i_k}) = L_{o_{k+1}}(t(\mathbf{x}_k, \boldsymbol{\omega}_{i_k}), -\boldsymbol{\omega}_{i_k}) = L_{o_{k+1}}(\mathbf{x}_{k+1}, -\boldsymbol{\omega}_{i_k}) = L_{o_{k+1}}(\mathbf{x}_{k+1}, \boldsymbol{\omega}_{o_{k+1}}), \tag{2.24}$$

where the subscript indicates that the incoming radiance at the current point of a process that has been done k times, equals the outgoing radiance of another point one step further in the process, $k+1$ (see Figure 2.3).

Labeling each surface vertex as $\mathbf{x}_0 = \mathbf{x}_{\text{lens}}, \mathbf{x}_1, \mathbf{x}_2, ...$ and expanding Equation 2.23 the rendering equation is the recursive-integral equation:

$$\begin{aligned} &L_{o_1}(\mathbf{x}_1, \boldsymbol{\omega}_{o_1}) = L_e(\mathbf{x}_1, \boldsymbol{\omega}_{o_1}) \\ &+ \int_\Omega \rho(\mathbf{x}_1, \boldsymbol{\omega}_{o_1}, \boldsymbol{\omega}_{i_1})|\mathbf{n}_1 \cdot \boldsymbol{\omega}_{i_1}| \left[L_e(\mathbf{x}_2, \boldsymbol{\omega}_{o_2}) + \int_\Omega \rho(\mathbf{x}_2, \boldsymbol{\omega}_{o_2}, \boldsymbol{\omega}_{i_2}) L_{i_2}(\mathbf{x}_2, \boldsymbol{\omega}_{i_2})|\mathbf{n}_2 \cdot \boldsymbol{\omega}_{i_2}| \,\mathrm{d}\boldsymbol{\omega}_{i_2} \right] \mathrm{d}\boldsymbol{\omega}_{i_1}, \end{aligned} \tag{2.25}$$

where:

$$\begin{aligned} L_{o_k}(\mathbf{x}_k, \boldsymbol{\omega}_{o_k}) &= L_e(\mathbf{x}_k, \boldsymbol{\omega}_{o_k}) + \int_\Omega \rho(\mathbf{x}_k, \boldsymbol{\omega}_{o_k}, \boldsymbol{\omega}_{i_k}) L_{i_k}(\mathbf{x}_k, \boldsymbol{\omega}_{i_k})|\mathbf{n} \cdot \boldsymbol{\omega}_{i_k}| \,\mathrm{d}\boldsymbol{\omega}_{i_k} \\ L_{i_k}(\mathbf{x}_k, \boldsymbol{\omega}_{i_k}) &= L_{o_{k+1}}(t(\mathbf{x}_k, \boldsymbol{\omega}_{i_k}), -\boldsymbol{\omega}_{i_k}) = L_{o_{k+1}}(\mathbf{x}_{k+1}, \boldsymbol{\omega}_{o_{k+1}}). \end{aligned} \tag{2.26}$$

The recursive nature of the light transport equation can be eliminated if only the emittance arriving

[2]Light could be affected not just by the geometric surfaces material but also for phenomena induced by the other points in the "air", such as scattering, absorption and emission, this type of rendering is used for the creation of effects like, fire and smoke. This type of rendering is often called *volumetric rendering*.

at the first intersected point $\mathbf{x}_1$ is considered:

$$L_{o_1}(\mathbf{x}_1, \boldsymbol{\omega}_o) = L_e(\mathbf{x}_1, \boldsymbol{\omega}_o) + \int_\Omega \rho(\mathbf{x}_1, \boldsymbol{\omega}_o, \boldsymbol{\omega}_i) L_D(\mathbf{x}_i, \boldsymbol{\omega}_i) |\mathbf{n} \cdot \boldsymbol{\omega}_i| \, \mathrm{d}\boldsymbol{\omega}_i, \tag{2.27}$$

with:

$$L_D(\mathbf{x}, \boldsymbol{\omega}) \equiv L_e(t(\mathbf{x}, \boldsymbol{\omega}), -\boldsymbol{\omega}), \tag{2.28}$$

this is called the *direct illumination* (DI) version of the rendering equation, as opposed to its recursive version called the *global illumination* (GI) version of the rendering equation. An *integrator* as it was already mentioned is the component of a rendering system that numerically (usually by Monte Carlo integration) solves this equation. Most of the computational efforts of a rendering system are spent solving this equation.

Before proceeding, one notation that is going to be useful is the *three point notation*, introduced by Veach [Vea97] (see Figure 2.4), used for functions that depends on directions defined by sequences of vertices:

$$f(\mathbf{y} \to \mathbf{z}) \equiv f(\mathbf{y}, \boldsymbol{\omega}), f(\mathbf{x} \to \mathbf{x}' \to \mathbf{x}'') \equiv f(\mathbf{x}, \boldsymbol{\omega}_o, \boldsymbol{\omega}_i), \boldsymbol{\omega} = \frac{\mathbf{z} - \mathbf{y}}{\|\mathbf{z} - \mathbf{y}\|}, \boldsymbol{\omega}_o = \frac{\mathbf{x} - \mathbf{x}'}{\|\mathbf{x} - \mathbf{x}'\|}, \boldsymbol{\omega}_i = \frac{\mathbf{x}'' - \mathbf{x}'}{\|\mathbf{x}'' - \mathbf{x}'\|}, \tag{2.29}$$

using this notation, the terms of the rendering equation for a path $\mathbf{X}$ will be denoted as:

$$\begin{aligned} &L_e^k \equiv L_e(\mathbf{x}_k \to \mathbf{x}_{k-1}); \quad f^k \equiv \rho(\mathbf{x}_{k-1} \to \mathbf{x}_k \to \mathbf{x}_{k+1}) |\mathbf{n}_{\mathbf{x}_k} \cdot \boldsymbol{\omega}_{i_k}| \equiv \rho^\perp(\mathbf{x}_{k-1} \to \mathbf{x}_k \to \mathbf{x}_{k+1}); \\ &L_i^k \equiv L_{i_k}(\mathbf{x}_k \to \mathbf{x}_{k+1}). \end{aligned} \tag{2.30}$$

The rendering equation returns the outgoing radiance $L_o(\mathbf{x}, \boldsymbol{\omega})$ emitted by a point $\mathbf{x}$ in some direction $\boldsymbol{\omega}$, but to truly get a usable value with the correct physical units, the way in which the light interacts with the camera's lens and film must be taken into account by adding an *emitted importance function* $W_e(\mathbf{x}, \boldsymbol{\omega})$, which tells us how to weight (more weight equals more importance) each incoming ray arriving at a point in the sensor of the camera (film) $\mathbf{x}$ from some incoming direction $\boldsymbol{\omega}$, then in rendering the value registered by the film of the camera in some region of the space j which corresponds to a pixel of the final image is called the *average measured radiance* I_j of pixel j, and if the film is assumed to be a *linear sensor*[3] [Vea97]:

$$W_e(\mathbf{x}, \boldsymbol{\omega}) \equiv \frac{\mathrm{d}I_j(\mathbf{x}, \boldsymbol{\omega})}{\mathrm{d}\Phi(\mathbf{x}, \boldsymbol{\omega})}; \qquad \mathrm{d}I_j(\mathbf{x}, \boldsymbol{\omega}) = W_e(\mathbf{x}, \boldsymbol{\omega}) L(\mathbf{x}, \boldsymbol{\omega}) \, \mathrm{d}\boldsymbol{\omega}^\perp \, \mathrm{d}A(\mathbf{x}), \tag{2.31}$$

the emitted importance function can be modeled in several ways being one of the most common ones to just assume that all the rays being redirected by the camera lens into all the film positions all have the same importance:[4]

$$W_e(\mathbf{x}, \boldsymbol{\omega}) = \frac{p^{A_{\text{lens}}} p(\boldsymbol{\omega})}{|\cos\theta|} = \frac{p^{A_{\text{lens}}} p^{A_{\text{film}}}}{G(\mathbf{x}_{\text{lens}} \leftrightarrow \mathbf{x}_{\text{film}})} \quad \Rightarrow \quad \int_{A_{\text{lens}}} \int_\Omega W_e(\mathbf{x}_{\text{lens}}, \boldsymbol{\omega}) |\mathbf{n}_{\mathbf{x}_{\text{lens}}} \cdot \boldsymbol{\omega}| \, \mathrm{d}\boldsymbol{\omega} \, \mathrm{d}A(\mathbf{x}_{\text{lens}}) = 1. \tag{2.32}$$

[3]The change in the measured quantity (response) is proportional to the change in the measured incoming light.

[4]For simplified camera models such as the orthographic and perspective (projective cameras) thin less approximations this is true, but for more complex models such as [PJH16] realistic camera, the multiple lenses affect the incoming radiance differently. This is the reason that for an MC forward path tracer estimator that uses a projective camera model the importance function seems as if it was ignored, while if a realistic camera model is being used the importance must be considered.

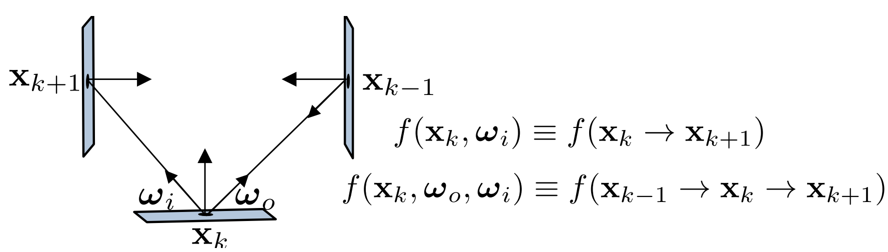


Figure 2.4: The three point formulation of rendering functions. Functions that depends on both points and directions that are related to a succession of vertices can be expressed just with the vertices of the succession with the directions being implicitly understood.

The importance function $W_e(\mathbf{x},\boldsymbol{\omega})$ is still missing one extra factor $\frac{h_j(\mathbf{x}_{\text{film}})}{\int_{A_{\text{film}}} h_j(\mathbf{x})dA(\mathbf{x})} = \frac{h_j}{b_f}$, which is an *antialiasing filter* function, which limits the non-band limited[5] signal to a finite value, losing high definition information in exchange of correctly reconstruct the new signal that contains less information, being then the final importance function for each pixel j:

$$W_e^j(\mathbf{x},\boldsymbol{\omega}) = \frac{p^{A_{\text{lens}}}}{G(\mathbf{x}_{\text{lens}} \leftrightarrow \mathbf{x}_{\text{film}})} \frac{h_j(\mathbf{x}_{\text{film}})}{\int_{A(\mathbf{x})} h_j(\mathbf{x})\,\mathrm{d}A(\mathbf{x})} \Rightarrow \int_{A_{\text{lens}}} \int_{A_{\text{film}}} W_e^j(\mathbf{x}_{\text{l}},\boldsymbol{\omega}) G(\mathbf{x}_{\text{l}} \leftrightarrow \mathbf{x}_{\text{f}})\,\mathrm{d}A(\mathbf{x}_{\text{l}})\,\mathrm{d}A(\mathbf{x}_{\text{f}}) = 1, \tag{2.33}$$

or putting into words, the rays at the center of the pixel have a bigger weight or importance than the rays in the boundary of the filter. The highest value of the antialiasing filter function occurs at the j pixel's center $\mathbf{r}_{j_C} = (r^x_{j_C}, r^y_{j_C})$ and has support

$$\text{supp}(h_j) = \{(x,y) | (r^x_{j_C} - r_x \le x \le r^x_{j_C} + r_x) \wedge (r^y_{j_C} - r_y \le y \le r^y_{j_C} + r_y)\}, \tag{2.34}$$

for some horizontal and vertical radius $r_x, r_y \ge 0.5$. The support of the antialiasing filter function is often called the *footprint of the pixel.*, from now on, the fitering function is assumed to use the radius $r_x = r_y = 0.5$ and always take the value $h_j(\mathbf{x}) = 1 \quad \forall \mathbf{x} \in \text{supp}(h_j)$, filter known as a *box filter*.

The final integral used in rendering to get the radiance values of a pixel is then called the *measurement equation*:

$$\begin{aligned} I_j &\equiv \int_{A_{\text{lens}}} \int_{\Omega} W_e^j(\mathbf{x}_0,\boldsymbol{\omega}) L_{i_0}(\mathbf{x}_0,\boldsymbol{\omega})\,\mathrm{d}\boldsymbol{\omega}^\perp\,\mathrm{d}A(\mathbf{x}_0) = \int_{A_{\text{film}}} \int_{\Omega} W_e(\mathbf{x}_{\text{film}},\boldsymbol{\omega}) L_{i_0}(\mathbf{x}_{\text{film}},\boldsymbol{\omega})\,\mathrm{d}\boldsymbol{\omega}^\perp\,\mathrm{d}A(\mathbf{x}_{\text{film}}) \\ &= \int_{A(\mathbf{x}_0)} \int_{A(\mathbf{x}_1)} W_e^j(\mathbf{x}_0,\boldsymbol{\omega}) L_{o_1}(\mathbf{x}_1,\boldsymbol{\omega}_{o_1}) G(\mathbf{x}_0 \leftrightarrow \mathbf{x}_1)\,\mathrm{d}A(\mathbf{x}_0)\,\mathrm{d}A(\mathbf{x}_1) \end{aligned} \tag{2.35}$$

where $W_e(\mathbf{x}_o,\boldsymbol{\omega})$ is the exitant importance going from the lens point $\mathbf{x}_0$ towards some direction, and $W_e(\mathbf{x}_{\text{film}},\boldsymbol{\omega})$ is the exitant importance for a film point $\mathbf{x}_{\text{film}}$ towards a direction $\boldsymbol{\omega}$, and $\mathbf{x}_1$ is a point already in the scene. To obtain the average measured radiance of a pixel j, an integral over all the raster positions in the film associated with the pixel and all the incoming directions that are being originated from all the possible lens points in the camera must be evaluated.

The value $L_{i_0}(\mathbf{x}_0,\boldsymbol{\omega}_{i_0})$ is then the incoming radiance received at a point $\mathbf{x}_0$ in the lens from a direction $\boldsymbol{\omega}_{i_0}$, this direction $\boldsymbol{\omega}_{i_0}$ pierces a raster position in the film which is related to the final spectrum of the pixel, the rendering pipeline looks like:

$$\text{World} \to \text{Camera} \to \text{Film} \to \text{Raster} \to \text{Pixel}, \tag{2.36}$$

where transforming from world coordinates to camera coordinates implies the use of rotations and translations, transforming from camera coordinates to film coordinates implies a projective transformation (e.g., orthographic, perspective).

2.2 Mathematical preliminaries: Monte Carlo Integration

This section will be divided in four subsections. In the first subsection we will give a quick probability theory review. In the second subsection we will introduce the Monte Carlo integration technique. After this, in the third subsection we will present the technique called Multiple Importance Sampling (MIS) to improve the convergence of the Monte Carlo estimator. Finally, in the fourth subsection, we will introduce an algorithm called Markov Chain Monte Carlo that will help us to increase the efficiency of the Monte Carlo estimator.

2.2.1 Probability preliminaries

The tuple $(\Omega, \mathcal{F})$ is called a *measurable space* if Ω is a non empty set and $\mathcal{F}$ is a *σ-algebra* class[6] of the set Ω. In the context of probability the set Ω is known as the *sample space*, the set of all possible

[5]The maximum frequency needed to reconstruct the signal (the light and its interactions with the scene) is infinite $f_{\max} = \infty$, think of a square wave, for example.

[6]A σ-algebra or σ-field of a non empty set Ω is a class that must satisfy, (i) $\Omega \in \mathcal{F}$, (ii) if $A \in \mathcal{F}$ then the complement set $A^c \in \mathcal{F}$, (iii) if $A.B \in \mathcal{F}$ then $A \cup B \in \mathcal{F}$ and (iv) if $A_1, A_2, ... \in \mathcal{F}$ then $A_1 \cup A_2 \cup ... \in \mathcal{F}$. If only the first three conditions are met, then the class is just an algebra or *field*.

outcomes, and $\mathcal{F}$ is the class of events (outcomes in the sample space, or subsets of the sample space). A set function over the measurable space $(\Omega, \mathcal{F})$ is a *measure* $\mu : \mathcal{F} \to [0, \infty]$ or a *probability measure* $\mathbb{P} : \mathcal{F} \to [0, 1]$ if it meets the following conditions:

Measure conditions

1. $\mu(A) \in [0, \infty]\ \forall A \in \mathcal{F}$
2. $\mu(\emptyset) = 0$
3. If $A_1, A_2, ...$ is a disjoint countable sequence of $\mathcal{F}$-sets and if $\cup_{k=1}^{\infty} A_k \in \mathcal{F}$ then:

$$\mu\left(\bigcup_{k=1}^{\infty} A_k\right) = \sum_{k=1}^{\infty} \mu(A_k). \tag{2.37}$$

Probability measure conditions: Kolmogorov Axioms

1. $0 \leq \mathbb{P}(A) \leq 1\ \forall A \in \mathcal{F}$
2. $\mathbb{P}(\emptyset) = 0, \mathbb{P}(\Omega) = 1$
3. If $A_1, A_2, ...$ is a disjoint $(A_i \cap A_j = \emptyset)$ sequence of $\mathcal{F}$-sets and if $\cup_{k=1}^{\infty} A_k \in \mathcal{F}$ then:

$$\mathbb{P}\left(\bigcup_{k=1}^{\infty} A_k\right) = \sum_{k=1}^{\infty} \mathbb{P}(A_k), \tag{2.38}$$

The tuple $(\Omega, \mathcal{F}, \mu)$ is then called a *measure space* while the tuple $(\Omega, \mathcal{F}, \mathbb{P})$ is called a *probability measure space* [Bil95]. The integral of a measurable function f over the measure space $(\Omega, \mathcal{F}, \mu)$ is given by:

$$\nu(A \in \mathcal{F}) = \int_A f \,\mathrm{d}\mu = \int_{\omega \in A} f(\omega) \,\mathrm{d}\mu(\omega) = \int_{\omega \in A} f(\omega)\mu(\mathrm{d}\omega), \tag{2.39}$$

also the derivative of the set function ν can be obtained which is known as the *Radon-Nykodym derivative*:

$$\frac{\mathrm{d}\nu}{\mathrm{d}\mu}(\omega) = f(\omega). \tag{2.40}$$

A *random variable* X on the probability measure space $(\Omega, \mathcal{F}, \mathbb{P})$ is a *measurable function*[7] $X : (\Omega, \mathcal{F}) \to (H, \xi)$, with $(\Omega, \mathcal{F}), (H, \xi)$ measurable spaces (e.g., $H = \mathbb{R}$). This random variable X induces a new probability measure that describes the probability of the random variable to take certain values $S \subseteq E$ on its measurable mapping:

$$\mathbb{P}_X(S) \equiv \mathbb{P}(X \in S \subseteq H) = \mathbb{P}(X^{-1}(S)) = \mathbb{P}(\{\omega \in \Omega | X(\omega) \in S\}), \tag{2.41}$$

the probability measure $\mathbb{P}_X$ that the random variable induces over the original space is sometimes called the *induced measure* by X. Notice how for a single probability measure space $(\Omega, \mathcal{F}, \mathbb{P})$ multiple different induced measures can be produced by several random variables X, Y, for example if Ω is the interval $[0, 1]$ and the probability measure is just the length of the interval[8], and defining the random variables:

$$X(\omega) = 1\ \forall \omega \in \Omega \quad Y(\omega) = \begin{cases} 1\ \forall \omega \leq p \in [0,1] \\ 0 \text{ elsewhere} \end{cases} \qquad Z(\omega) = \text{length}([0, \omega]), \tag{2.42}$$

it can be seen that $\mathbb{P}(X = 1) = 1$, $\mathbb{P}(Y = 1) = p$, $\mathbb{P}(Y = 0) = 1 - p$ and $\mathbb{P}(Z \in \{\omega' | \omega' < \omega\}) = \omega$, which means that each induced measure for the same probability space is different, because each random variable is describing a different thing about the same probability measure space. We denote the associated *induced* measure of each random variable as $X \sim \mathbb{P}_X$, $Y \sim \mathbb{P}_Y$, $Z \sim \mathbb{P}_Z$, in this example, $Y \sim \text{Bernoulli}(p)$, where Bernoulli$(p)$ is called the Bernoulli distribution and $Z \sim U[0, 1]$ where $U[0, 1]$ is the uniform random distribution in the interval $[0, 1]$.

For a probability measure space $(\Omega, \mathcal{F}, \mathbb{P})$, the probability measure induced by a random variable X with range in a measurable space (H, ξ), can often be expressed as the integral over a function called the *probability density function* (pdf) $p_X : \Omega \to \mathbb{R}^+$. We are going to consider the case in which Ω is some subset of the cartesian power of the real numbers $\Omega = \mathbb{R}^n, \mathbf{x} \in \Omega$ for some $n \in \mathbb{N}$:

$$\mathbb{P}_X(S) = \mathbb{P}(X \in S) = \int_{\mathbf{x} \in S} p_X(\mathbf{x}) d\mu(\mathbf{x}), \qquad \int_{\mathbf{x} \in \Omega} p_X(\mathbf{x}) d\mu(\mathbf{x}) = 1, \qquad \frac{d\mathbb{P}_X}{d\mu}(\mathbf{x}) = p_X(\mathbf{x}), \tag{2.43}$$

[7] A function $X : (\Omega, \mathcal{F}) \to (\Omega', \mathcal{F}')$ from the measurable space $(\Omega, \mathcal{F})$ to another measurable space $(\Omega', \mathcal{F}')$ such that the inverse function satisfies $X^{-1}(F') \in \mathcal{F} \quad \forall F' \in \mathcal{F}'$.

[8] Called Lebesgue measure.

the pdf must always be positive. A function closely related to the pdf is called the *cumulative distribution function*, which is simply the induced probability $\mathbb{P}_X$ evaluated at values smaller than $\mathbf{x}$:

$$[\mathrm{CDF}]_X(\mathbf{x}) = P_X(\mathbf{x}) \equiv \mathbb{P}_X(\{X \leq \mathbf{x}\}) = \int_{\{\mathbf{x}':\mathbf{x}'<\mathbf{x}\}} p_X(\mathbf{x}')\,\mathrm{d}\mu(\mathbf{x}'), \qquad \frac{\mathrm{d}P_X}{\mathrm{d}\mu}(\mathbf{x}) = p_X(\mathbf{x}), \tag{2.44}$$

where $\mathbf{x} = (x_1, x_2, ..., x_n) < \mathbf{y} = (y_1, y_2, ..., y_n)$ if $x_i < y_i$ for all $1 \leq i \leq n$, and where we denote as P_X the cdf function, which is the special case when the induced probability $\mathbb{P}_X$ receives sets of the form $\{X \leq \mathbf{x}\}$ as inputs. Notice how the probability density function p_X describes the induced probability measure $\mathbb{P}_X$, so the distribution of the random variable is going to be denoted as $X \sim p_X$. Any positive normalized function $p_f(\mathbf{x}) \equiv \frac{f(\mathbf{x})}{\int_E f(\mathbf{x})\,\mathrm{d}\mu(\mathbf{x})}$ could be a pdf, in such a way that it has an associated random variable $X \sim p_f$.

We have presented random variables as deterministic entities (measurable set functions), now we will explicitly introduce and explain the random nature of these functions, using the sampling of a population as an example.

In statistics, the sample space Ω is called a population, and each individual $\omega_j \in \Omega$ of this population has a certain measured characteristic e_0^j, this characteristic is a measurement or *realisation* obtained. For a population of N individuals all the realisations $(e_0^1, ..., e_0^N)$ can define a pdf p_e, which can be understood as a plot of the frequencies of the realisations (range of the random variable), with implicit random variable associated $X_0 \sim p_e$, which equals $X_0(\omega_i) = e_0^i$, this example looks too deterministic, as we are assuming that we already know the values of the members of a population, effectively building the pdf from the measured data $(e_0^1, ..., e_0^N) \to X_0 \sim p_e$.

To see the random nature of this process we now suppose that we don't have any of the realisations available, that is, given an individual ω_j whose realisation is unknown, the random variable $X_i \sim p_e$ assigns a realisation to this individual with a probability proportional to p_e, that is, $X_i(\omega_j) = e_i^j$, obtaining for all the individuals of the population a probable but not certain measurement $X_i \sim p_e \to (e_i^1, ..., e_i^N)$, which plotted would resemble p_e. If m random variables were present, they would assign different values to the same individual ω_j: $(e_1^j, e_2^j, ..., e_m^j)$, furthermore, if all the random variables were *independent and identically distributed* (i.i.d.) with $X_i \sim p_e$ then the histogram of the different observations attributed from different random variables to this unique individual would reconstruct p_e, which means that the histogram can be reconstructed from either applying the random variable to all the individuals of the population with unknown measurements, or we could reconstruct p_e by repeating the *experiment* multiple times, applying *different* but i.i.d. random variables to the same individual.

A *random sample of size* N is then defined as a sequence of N independent and identically distributed random variables $\mathbf{X} = (X_i)_{i=1}^N$ [DS11], which is in itself a measurable function $\mathbf{X}(\omega) = (X_i(\omega))_{i=1}^N = (x^i)_{i=1}^N$, with the histogram of $(x^i)_{i=1}^N$ resembling the respective pdf. Each random variable is a 1-sample by itself.

One question that naturally arises is how to obtain samples that are proportional to the pdf p_f of a function $f : \mathbb{R}^n \to \mathbb{R}^+$ which has an associated cdf $F : \Omega \to [0, 1]$. The main method used in computer graphics is called the *inverse transform sampling method* [Ros22], which consists in defining the random variable as:

$$X_i = F^{-1}(U_i), \qquad (\mathbf{x} = F^{-1}(\mathbf{u})) \longleftrightarrow (F(\mathbf{x}) = \mathbf{u}), \tag{2.45}$$

with U_i a uniform distributed random variable in the interval $[0, 1]$. It can be shown that if the random variable X_i is defined in this way, then the cdf induced by this random variable $P_{X_i}(x)$ is the same than the cdf F of the function f, implying that $X_i \sim p_f$:

$$\begin{aligned} P_{X_i}(x) &= \mathbb{P}(\{X_i \leq x\}) = \mathbb{P}(\{F^{-1}(U_i) \leq x\}) = \mathbb{P}(\{F(F^{-1}(U_i)) \leq F(x)\}) \\ &= \mathbb{P}(\{U_i \leq F(x)\}) = F(x), \end{aligned} \tag{2.46}$$

where the facts that all cdf functions are monotonically increasing ($a \leq b \to F(a) \leq F(b)$) and that $\mathbb{P}(\{U_i \leq a\}) = a$ because U_i is uniformly distributed in the interval $[0, 1]$ were used.

We are ready now to talk about the relationship between the primary sample space domain and the area, solid angle, and projected solid angle domains. We will also talk about how a change of measures implies also a change of the pdf functions.

Notice that the probability density function p is the Radon-Nykodym derivative of the cumulative density function P (see Equation 2.44), which in turn is dependent of the measure used. The probability density function that is proportional to a function f with respect to a measure μ will be denoted as p_f^μ. This is very important because a simple change of variables integral:

$$I = \int_A f(\mathbf{x})\,\mathrm{d}\mu(\mathbf{x}) = \int_{T(A)} f(T(\mathbf{x}))\left|\frac{\mathrm{d}\mu(\mathbf{x})}{\mathrm{d}T(\mathbf{x})}\right|\mathrm{d}T(\mathbf{x}), \tag{2.47}$$

completely changes the measure from μ to T which means that the corresponding probabilities must also change from $p^\mu = \frac{\mathrm{d}P}{\mathrm{d}\mu}(\mathbf{x})$ to $p^T = \frac{\mathrm{d}P}{\mathrm{d}T}(\mathbf{x})$. Also notice that a change of measure $\left|\frac{\mathrm{d}\mu(\mathbf{x})}{\mathrm{d}T(\mathbf{x})}\right|$ factor called the *jacobian determinant* is required when we are changing the integration measures. The jacobian determinant that relates the area domain with the solid angle domain (see Equation 2.19), and the jacobian determinant that relates the area domain with the projected solid angle domain (see Equation 2.20) were already derived. Now we will proceed to introduce the jacobian determinant that relates the primary sample domain with both the area domain and the solid angle domain.

Uniform distributed random numbers on the interval $[0,1]$ are called *canonical random numbers*, and they define one of the most important relationships used in this thesis: any sample $\mathbf{x}$ drawn from a probability density distribution p_f that was generated by a canonical random number $\mathbf{u}$ used by the inversion sampling method are related by Equation 2.45, and they define a change of coordinates between the samples and the canonical random numbers that were used to generate them:

$$\left|\frac{\mathrm{d}\mathbf{u}}{\mathrm{d}\mathbf{x}}\right| = \left|\frac{\mathrm{d}F(\mathbf{x})}{\mathrm{d}\mathbf{x}}\right| = p_f(\mathbf{x}), \qquad \left|\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}\mathbf{u}}\right| = \frac{1}{p_f(\mathbf{x})}, \tag{2.48}$$

as most sampling processes in rendering systems use the inversion sampling method to obtain either directions $\boldsymbol{\omega}$ (e.g., material sampling) or points $\mathbf{x}$ (e.g., shape area sampling, like a lens or an area light) we have:

$$\begin{aligned}
&(P_f^\omega)^{-1}(\mathbf{u}_k) = \boldsymbol{\omega}_k \leftrightarrow \mathbf{u}_k = P_f^\omega(\boldsymbol{\omega}_k) && \left|\frac{\mathrm{d}\mathbf{u}_k}{\mathrm{d}\boldsymbol{\omega}_k}\right| = \left|\frac{\mathrm{d}P_f^\omega(\boldsymbol{\omega}_k)}{\mathrm{d}\boldsymbol{\omega}_k}\right| = p_f^\omega(\boldsymbol{\omega}_k) && \left|\frac{\mathrm{d}\boldsymbol{\omega}_k}{\mathrm{d}\mathbf{u}_k}\right| = \frac{1}{p_f^\omega(\boldsymbol{\omega}_k)}\\
&(P_f^A)^{-1}(\mathbf{u}_k) = \boldsymbol{x}_k \leftrightarrow \mathbf{u}_k = P_f^A(\boldsymbol{x}_k) && \left|\frac{\mathrm{d}\mathbf{u}_k}{\mathrm{d}A(\boldsymbol{x}_k)}\right| = \left|\frac{\mathrm{d}P_f^A(\boldsymbol{x}_k)}{\mathrm{d}A(\boldsymbol{x}_k)}\right| = p_f^A(\boldsymbol{x}_k) && \left|\frac{\mathrm{d}A(\boldsymbol{x}_k)}{\mathrm{d}\mathbf{u}_k}\right| = \frac{1}{p_f^A(\boldsymbol{x}_k)}
\end{aligned} \tag{2.49}$$

the integral over the sample space can be transformed to an integral over the space of all possible canonical random numbers, a domain of integration that will be quite useful.

There is still one important thing that we need to address about the integration domains in rendering. Some samplings in rendering only make sense in one particular measure, regardless of the measure used in Equation 2.47, for example the area measure p^A for sampling a point in an area light or a lens, or the solid angle measure to sample directions according to a material p^ω. If we call the measure of integral 2.47 as $\mu = \mu_1$, and the measure used to get a sampling as μ_2 (e.g., area or solid angle), we can convert p^{μ_2} into the appropriate measure p^{μ_1} which corresponds to the measure of the integral by using the chain rule

$$p^{\mu_1}(\mathbf{x}) = \frac{\mathrm{d}P^{\mu_1}}{\mathrm{d}\mu_1}(\mathbf{x}) = \frac{\mathrm{d}P^{\mu_2}}{\mathrm{d}\mu_2}(\mathbf{y})\frac{\mathrm{d}\mu_2}{\mathrm{d}\mu_1} = p^{\mu_2}(\mathbf{y})\frac{\mathrm{d}\mu_2}{\mathrm{d}\mu_1}, \quad p^{\mu_2}(\mathbf{y}) = \frac{\mathrm{d}P^{\mu_2}}{\mathrm{d}\mu_2}(\mathbf{y}) = \frac{\mathrm{d}P^{\mu_1}}{\mathrm{d}\mu_1}(\mathbf{x})\frac{\mathrm{d}\mu_1}{\mathrm{d}\mu_2} = p^{\mu_1}(\mathbf{x})\frac{\mathrm{d}\mu_1}{\mathrm{d}\mu_2}, \tag{2.50}$$

it is then useful to also denote the cdf's as P^μ as a reminder that they might have their own measure, independent of the appropriate measure of integral 2.47, which of course, might need to be transformed using the chain rule 2.50. If we denote the probabilities in the solid angle domain as p^ω, in the area domain as p^A, in the projected solid angle domain as $p^{\omega^\perp} \equiv p^\perp$, and in the canonical random number domain as p^u, and we applied the chain rule we obtain the relationships

$$\begin{aligned}
p^\perp(\boldsymbol{\omega}) &= \frac{\mathrm{d}P^\omega}{\mathrm{d}\mu_\omega}(\boldsymbol{\omega})\frac{\mathrm{d}\mu_\omega}{\mathrm{d}\mu_{\omega^\perp}} = \frac{p^\omega(\boldsymbol{\omega})}{|\mathbf{n}_\mathbf{x}\cdot\boldsymbol{\omega}|}, \quad & p^A(\mathbf{x}') &= \frac{\mathrm{d}P^\omega}{d\mu_\omega}(\boldsymbol{\omega})\frac{\mathrm{d}\mu_\omega}{\mathrm{d}\mu_A} = p^\omega(\boldsymbol{\omega})\frac{|\mathbf{n}_{\mathbf{x}'}\cdot -\boldsymbol{\omega}|}{\|\mathbf{x}-\mathbf{x}'\|^2}.\\
p^u(\mathbf{u}) &= \frac{\mathrm{d}P^\omega}{d\mu_\omega}(\boldsymbol{\omega})\frac{\mathrm{d}\mu_\omega}{\mathrm{d}\mu_u} = \frac{p^\omega(\boldsymbol{\omega})}{p^\omega(\boldsymbol{\omega})} = 1 & p^u(\mathbf{u}) &= \frac{\mathrm{d}P^A}{d\mu_A}(\boldsymbol{x})\frac{\mathrm{d}\mu_A}{\mathrm{d}\mu_u} = \frac{p^A(\mathbf{x})}{p^A(\mathbf{x})} = 1
\end{aligned} \tag{2.51}$$

Now that the random nature of random variables is already explained, two important concept regarding random variables will be introduced. The *expected value* of a random variable $E[X]$ and the *law of the unconscious statistician* which consist in obtaining the expected value of the composition of a random variable with some function f, i.e., $E[f \circ X] = E[f(X)]$ are:

$$E[X] = \int_{\Omega} X \, \mathrm{d}P_X = \int_{\omega \in \Omega} \omega p_X(\omega) \, \mathrm{d}\mu(\omega), \qquad E[f(X)] = \int_{\Omega} f(X) \, \mathrm{d}P_X = \int_{\omega \in \Omega} f(\omega) p_X(\omega) \, \mathrm{d}\mu(\omega), \tag{2.52}$$

where the Radon-Nikodym of the cdf function P_X was used. The *variance* V of a random variable is defined as:

$$V[X] = E[(X - E[X])^2] = E[X^2] - E[X]^2. \tag{2.53}$$

2.2.2 Monte Carlo integration

In rendering an integral:

$$I = \int_{\Omega} f(\mathbf{x}) \, \mathrm{d}\mu(\mathbf{x}) \tag{2.54}$$

needs to be solved, but the rendering integrals are highly dimensional, recursive and full of discontinuities due to abrupt changes in the elements of the scene (e.g., shapes, shadows, textures, materials, etc.). Because of these reasons a method that is robust to discontinuities and computationally efficient in highly dimensional integrals is needed. The suitable numerical method to solve these problems is called the *Monte Carlo integration* method defined as the *estimator* (of the integral):

$$I = \int_{\Omega} f(\mathbf{x}) \, \mathrm{d}\mu(\mathbf{x}) \approx I^{(N)} \equiv \frac{1}{N} \sum_{i=1}^{N} \frac{f(\mathbf{x}_i)}{p^{\mu}(\mathbf{x}_i)} \equiv \frac{1}{N} \sum_{i=1}^{N} S(\mathbf{x}_i), \tag{2.55}$$

where $\mathbf{x}_i \in \Omega$ is called the realisation of the random variable $X_i \sim p$ which was drawn from an arbitrary pdf p, with the obvious condition that $p(\mathbf{x}_i) > 0 \; \forall \mathbf{x}_i \in \Omega$, that is, samples must be taken over the whole *support of the function* f[9]:

$$\operatorname{supp}(f) = \{\mathbf{x} \in \operatorname{dom}(f) | f(\mathbf{x}) \neq 0\}; \qquad \operatorname{supp}(f) = \operatorname{supp}(p). \tag{2.56}$$

The proof that the Monte Carlo (MC) estimator expected value is the correct solution I is done applying the law of the unconscious statistician to a function $S(X_i) = f(X_i)/p(X_i)$ [Vea97], the expected value of a Monte Carlo integrator is:

$$E[I^{(N)}] = \frac{1}{N} \sum_{i=1}^{N} E\left[\frac{f(X_i)}{p(X_i)}\right] = \frac{1}{N} \sum_{i=1}^{N} \int_{\Omega} \frac{f(\mathbf{x})}{p(\mathbf{x})} p(\mathbf{x}) \, \mathrm{d}\mu(\mathbf{x}) = \int_{\Omega} f(\mathbf{x}) \, \mathrm{d}\mu(\mathbf{x}) = I, \tag{2.57}$$

therefore the expected value of a Monte Carlo integrator is the desired integral, in general, if the expected value of an estimator $I^{(N)}$ is the correct solution I, then the estimator is said to be *unbiased*:

$$E[I^{(N)}] = I, \tag{2.58}$$

furthermore if all the X_i are i.i.d. random variables, this obviously implies that all the $S(X_i) = f(X_i)/p^{\mu}(X_i)$ are also i.i.d, and by the *strong law of large numbers* we have

$$\overline{S_N} = S(X_1) + ... + S(X_N); \quad \lim_{N \to \infty} \frac{\overline{S_N}}{N} = E\left[\frac{f(X_1)}{p^{\mu}(X_1)}\right] = E[S(X_1)] = E[S(X_2)] = ... = E[S(X_N)] = I, \tag{2.59}$$

then it follows that the family of Monte Carlo estimators $(I^{(i)})_{i \geq 1}$ converges to the correct solution I when increasing the number of samples. In general, if the family of estimators $(I^{(i)})_{i \geq 1}$ converges to I, the estimator is said to be *consistent*:

$$\lim_{N \to \infty} I^{(N)} = I, \tag{2.60}$$

notice that the difference between consistency and unbiasedness is subtle, and might cause confusions, to see the difference consider the following examples for different estimators:

[9] If the integration is over a domain $A \subseteq \operatorname{supp}(f)$ then a new function f' could be defined as equal to f for all elements in A and zero for all elements outside.

- *Consistent and unbiased*:

$$I_{1,N} = \frac{1}{N}\sum_{i=1}^{N}\frac{f(\mathbf{x}_i)}{p(\mathbf{x}_i)}, \tag{2.61}$$

this is an unbiased and consistent MC estimator with i.i.d. random samples in the interval of integration. The rendering technique path tracing belongs to this category.

- *Inconsistent and unbiased*:

$$I_{2,N} = \frac{f(\mathbf{x}_N)}{p(\mathbf{x}_N)}, \tag{2.62}$$

this estimator only preserves the last value $\mathbf{x}_N$ without considering any of the other sampled values $\mathbf{x_i}, i < N$, notice how the expected value $E\left[I_{2,N}\right] = \int_\Omega \frac{f(\mathbf{x})}{p(\mathbf{x})} p(\mathbf{x}) d\mu(\mathbf{x}) = I$ is the correct result even though it is clearly not consistent at all. Inconsistency does not imply biasedness.

- *Consistent and biased*:

$$I_{3,N} = \frac{1}{N + 10000000}\sum_{i=1}^{N}\frac{f(\mathbf{x}_i)}{p(\mathbf{x}_i)}, \tag{2.63}$$

the expected value of this estimator is clearly biased, but when it tends to infinity it gives the correct result as:

$$\lim_{N\to\infty}\frac{N}{N + 10000000} = \lim_{N\to\infty}\frac{1}{1 + \frac{10000000}{N}} = 1, \tag{2.64}$$

so in the limit to infinity $N \approx N + 10000000$ and therefore the estimator becomes the unbiased and consistent MC estimator. The rendering technique called photon mapping belongs to this category.

- *Inconsistent and biased*:

$$I_{4,N} = f(\mathbf{x}_0) \text{ for some } \mathbf{x}_0 \in \Omega, \tag{2.65}$$

this hypothetic estimator is obviously useless, but despite of this example there are valid biased and inconsistent rendering techniques such as rasterization and radiosity.

2.2.3 Convergence and variance of MC estimators

The expected value is useful to know the unbiasedness of an estimator, but it doesn't tell anything about the rate (speed) of convergence. If the rate of convergence wants to be analyzed, the variance must be used:

$$V[I^{(N)}] = V\left[\frac{1}{N}\sum_{i=1}^{N}\frac{f(X_i)}{p(X_i)}\right] = \frac{1}{N^2}\sum_{i=1}^{N}V\left[\frac{f(X_i)}{p(X_i)}\right] = \frac{1}{N^2}\sum_{i=1}^{N}V[S(X_i)] = \frac{1}{N}V[S(X_1)], \tag{2.66}$$

where the fact that all the random variables X_i are i.i.d. was used which implies $\text{Cov}(X_i, X_j) = 0$ and $V[S(X_1)] = ... = V[S(X_N)]$. Then the standard deviation ($\sqrt{V[I^{(N)}]}$) of the Monte Carlo integrator of N samples $I^{(N)}$ is proportional to the inverse squared root of the number of samples N:

$$\sqrt{V[I^{(N)}]} \propto \frac{1}{\sqrt{N}} \quad \to \quad \sqrt{V[I^{(N)}]} = O\left(\frac{1}{\sqrt{N}}\right), \tag{2.67}$$

a reduction in variance is closely related to a reduction in error, as the variance is a measurement of the deviation from the expected value, which is the integral's desired value, less variance means a closer value to the expected value, this is the reason why the graph of the estimated error versus the number of samples looks like a straight line on a logarithmic scale (for both axis), with the variance acting as a slope modifier.

The natural question that follows this discussion is how to accelerate the convergence, and therefore, reduce the error of the estimator with respect of the number of samples, that is, reduce the variance given by Equation 2.67. We will end this sub-section by briefly discussing several variation reduction methods, with special emphasis to the error reduction techniques *Impotance sampling* (IS) and *Multiple Importance* (MIS). The next sub-section we will present a special type of importance sampling known as *Markov Chain Monte Carlo* (MCMC).

Stratified sampling Strategy used if the samples are independent. Stratification is done by dividing the integration domain Λ into n non-overlapping regions $\Lambda_1, \Lambda_2, ..., \Lambda_n$, with $\Lambda = \bigcup_{i=1}^{N} \Lambda_i$ and where each region is called a *stratum* [PJH16]. In this way we are making sure that each region receives the same amount of samples. The simplest form of stratification appears to be deterministic, as we are intentionally dividing the integration domain in sub-regions defined by our own criteria, e.g., a $N \times M$ grid. This apparently deterministic process is actually an stochastic process, defined as the conditional and sequential stochastic process composed by two random processes, the random process of choosing a stratum (in this case with probability $\frac{1}{NM}$), followed by the stochastic process of selecting a sample given the stratum selection. It has been proved that stratification can *not* increase the variance of Equation 2.67 [Vea97], and it actually always reduces the variance of the MC estimator, except in the case in which the integrated function has the same expected value over all the stratum [PJH16]. Of course stratifications might be done in more advanced ways than just simply defining a $N \times M$ grid, for example, the techniques called latin hypercube sampling and orthogonal array sampling are an example of this [Vea97].

Quasi Monte Carlo (QMC) This variance reduction technique is also used for independent samples. The *local discrepancy* $\mathcal{D}(J, \mathbf{U})$ of a sequence of N samples $\mathbf{U} = (u_i)_{i=1}^{N}$ taken from the s dimensional interval $\boldsymbol{\mathcal{U}}_s = [0,1)^s$, with $J \subseteq \boldsymbol{\mathcal{U}}_s$ a sub-interval is defined as [DSH08]:

$$\mathcal{D}(J, \mathbf{U}) = \left| \frac{A(J, \mathbf{U})}{N} - V(J) \right| \tag{2.68}$$

where $A(J, \mathbf{U})$ is a function that counts the number of samples $u_i \in J$, and $V(J)$ is the volume[10] of the sub-interval J. The *star-discrepancy* is a common way of measuring the global discrepancy of a sequence of N samples $\mathbf{U}$:

$$\mathcal{D}^*(\mathbf{U}) = \sup_J |\mathcal{D}(J, \mathbf{U})|, \tag{2.69}$$

a sequence that has a low star-discrepancy, that is, $\mathcal{D}^*(\mathbf{U}) = O\left(\frac{(\log N)^s}{N}\right)$ is a *low discrepancy sequence* [Vea97]. The MC estimator that uses low discrepancy sequences (e.g., the Sobol sequence and the Halton sequence) is called a *Quasi Monte Carlo estimator* an it has a standard deviation upper bound of $O\left(\frac{(\log N)^s}{N}\right)$ [Vea97]. This upper bound seems terrible, but in practice the analysis of QMC applied to computer graphics is more complex, it can be observed experimentally that in rendering QMC is indeed more efficient than MC in several situations [PJH16], and in top of this, QMC methods can have several algorithmic enhancements that makes them faster for some type of scenes. For a mathematical discussion of QMC applied to rendering we recommend Eric Veach's thesis [Vea97] and the survey of QMC methods applied to rendering by Keller [Kel13].

Adaptive sampling This type of sampling analyzes the information, in particular, the variance of the samples that have been taken so far in order to allocate more computational time in zones of high variance [Vea97]. Notice that this approach changes the pdf on the fly, which induces bias. This is an important remark to make, in rendering we can't naively change the distribution of the MC estimator on the fly while running the rendering process. If we want unbiased renders the pdfs used should be defined a priori before starting the rendering. The simplest solution to be able to use adaptive sampling while preserving unbiasedness is to add an initialization stage whose main objective will be to estimate high variance zones in the scene. After this, we will define a group of fixed pdfs that consider the estimated variance calculated in the initialization step. The rendering technique path guiding uses unbiased adaptive sampling.

Importance Sampling This is the variance reduction technique that we will use the most in this thesis. *Importance Sampling* (IS) consists in reducing the variance of the MC estimator by selecting a pdf that is similar to the integrand, process known known as *flattening the integral* because the ratio $S(X) = f(X)/p(X)$ tends to a constant. To see why this reduces the variance consider the *ideal MC estimator*

$$I^{(1)} = \frac{f(\mathbf{x}_1)}{p_f(\mathbf{x}_1)} = \frac{f(\mathbf{x}_1)}{\frac{f(\mathbf{x}_1)}{\int_\Omega f(\mathbf{x})\, \mathrm{d}\mu(\mathbf{x})}} = I, \tag{2.70}$$

[10] Lebesgue measure. All of these definitions need measure theory to be more formally defined.

which converges in just one sample. Realistically in rendering the function itself can't be sampled as this would require knowing beforehand the desired integral, but there are ways to make the pdf to resemble at least a part of the function, for this, suppose that the estimated function is a multiplication of functions $f(\mathbf{x}) = g(\mathbf{x})h(\mathbf{x})$, even if the pdf p_f of the perfect estimator can't be calculated, if p_g or p_h can be obtained, this would likely reduce the variance of the estimator:

$$[I^{(N)}]_{p_g} = \frac{\int_\Omega g(\mathbf{x})\,\mathrm{d}\mathbf{x}}{N}\sum_{i=1}^{N} h(\mathbf{x}_i); \qquad [I^{(N)}]_{p_h} = \frac{\int_\Omega h(\mathbf{x})\,\mathrm{d}\mathbf{x}}{N}\sum_{i=1}^{N} g(\mathbf{x}_i), \tag{2.71}$$

the question that remains is, which of the two estimators should be chosen, but it turns out that both can be selected preserving a consistent and unbiased estimator if appropriate weighting is used.

Multiple Importance Sampling The way of combining multiple MC estimators with different pdfs in order to reduce the variance is called *Multiple Importance Sampling (MIS)* [VG95b; Vea97] and is one of the most important ideas that has ever been introduced into computer graphics, the use of MIS in rendering marked a turning point in the efficiency of rendering, almost all of the existent techniques based in Monte Carlo estimators were improved after the introduction of this idea, and even to this day, more than twenty years after it was introduced by Eric Veach it is still used in computer graphics research.

The MIS weighted MC estimator for M techniques each one with N_i samples is defined as:

$$I^{(N_1,\ldots N_M)} = I^{(\mathbf{N})} = \sum_{i=1}^{M}\frac{1}{N_i}\sum_{j=1}^{N_i} w_i(\mathbf{x}_{ij})\frac{f(\mathbf{x}_{ij})}{p_i(\mathbf{x}_{ij})}, \tag{2.72}$$

where $\mathbf{x}_{ij} \in \Omega$ is the j-th realization obtained from technique i, with random variable $X_{ij} \sim p_i$, and w_i is a weighting factor that must satisfy:

$$(i)\ \sum_{i=1}^{M} w_i(\mathbf{x}) = 1 \qquad (ii)\ p_i(\mathbf{x}) = 0 \to w_i(\mathbf{x}) = 0. \tag{2.73}$$

Calculating the expected value of the MIS weighted integrator yields:

$$\begin{aligned} E[I^{(\mathbf{N})}] &= \sum_{i=1}^{M}\frac{1}{N_i}\sum_{j=1}^{N_i} E\left[w_i(X_{ij})\frac{f(X_{ij})}{p_i(X_{ij})}\right] = \sum_{i=1}^{M}\frac{1}{N_i}\sum_{j=1}^{N_i}\int_\Omega w_i(X)\frac{f(X)}{p_i(X)}p_i(X)\,\mathrm{d}X \\ &= \sum_{i=1}^{M}\int_\Omega w_i(X)f(X)\,\mathrm{d}X = \int_\Omega \sum_{i=1}^{M} w_i(X)f(X)\,\mathrm{d}X = \int_\Omega f(X)\,\mathrm{d}X = I, \end{aligned} \tag{2.74}$$

therefore applying MIS does indeed give an unbiased estimator.

It is also important to notice that is sufficient that the union of the techniques equals the support of the integrated function $\bigcup_{i=1}^{M}\operatorname{supp}(p_i) = \operatorname{supp}(f)$, this is possible as a result of condition (ii) of Equation 2.73, to see this suppose that technique p_k is not in the whole support, dividing the integration domain into $\Omega = \Omega_k + \Omega_{\neg k}$, where Ω_k is the support of technique k and $\Omega_{\neg k}$ is the domain not covered by p_k, then the expected value of the MIS estimator would be:

$$\begin{aligned} E[I^{(\mathbf{N})}] &= \sum_{i=1}^{M}\frac{1}{N_i}\sum_{j=1}^{N_i} E\left[w_i(X_{ij})\frac{f(X_{ij})}{p_i(X_{ij})}\right] \\ &= \sum_{i\neq k}\frac{1}{N_i}\sum_{j=1}^{N_i}\int_\Omega w_i(X)\frac{f(X)}{p_i(X)}p_i(X)\,\mathrm{d}X + \frac{1}{N_k}\sum_{j=1}^{N_k}\int_{\Omega_k} w_k(X)\frac{f(X)}{p_k(X)}p_k(X)\,\mathrm{d}X \\ &= \sum_{i=1}^{M}\frac{1}{N_i}\sum_{j=1}^{N_i}\int_\Omega w_i(X)\frac{f(X)}{p_i(X)}p_i(X)\,\mathrm{d}X - \int_{\Omega_{\neg k}} w_k(X)f(X)\,\mathrm{d}X = I - \int_{\Omega_{\neg k}} w_k(X)f(X)\,\mathrm{d}X, \end{aligned} \tag{2.75}$$

where the second integral would have introduced bias if $p_k(\mathbf{x}) = 0 \to w_k(\mathbf{x}) = 0$ wasn't true.

The exact function of each weight might be defined in several ways as long as the conditions 2.73 are satisfied, typical ways of defining these MIS weights are given by:

$$
\begin{aligned}
w_i(\mathbf{x}) &= \frac{(N_i p_i(\mathbf{x}))^\beta}{\sum_{j=1}^{M}(N_j p_j(\mathbf{x}))^\beta} && \beta = 1, 2, ...; && \textit{Power heuristics} \\
w_i(\mathbf{x}) &= \begin{cases} 1 \text{ If } N_i p_i(\mathbf{x}) = N_{\max} p_{\max}(\mathbf{x}) \\ 0 \text{ Else} \end{cases} && && \textit{Maximum heuristic}, \\
w_i(\mathbf{x}) &= \begin{cases} 0 \text{ If } N_i p_i(\mathbf{x}) < \alpha N_{\max} p_{\max}(\mathbf{x}) \\ \frac{N_i p_i(\mathbf{x})}{\sum_{N_j p_j(\mathbf{x}) \in \mathcal{M}_\alpha(\mathbf{x})} N_j p_j(\mathbf{x})} \end{cases} && \alpha \in [0, 1] && \textit{Cut-off heuristics}
\end{aligned}
\tag{2.76}
$$

where

$$
N_{\max} p_{\max}(\mathbf{x}) \equiv \max\{N_j p_j(\mathbf{x}) | 1 \leq j \leq M\} \qquad \mathcal{M}_\alpha(\mathbf{x}) = \{N_k p_k(\mathbf{x}) \geq \alpha N_{\max} p_{\max}(\mathbf{x}) | 0 \leq k \leq M\},
\tag{2.77}
$$

the specific case $\beta = 1$ of the power heuristics is known as the *balance heuristic*, name given due that the weighted contribution of the MIS MC estimator that uses the balance heuristic is $\frac{f(\mathbf{x}) w_i(\mathbf{x})}{N_i p_i(\mathbf{x})} = \frac{f(\mathbf{x})}{\sum_{j=1}^{M} N_j p_j(\mathbf{x})}$, which ends up dividing the values $f(\mathbf{x}_{ij})$ by the same factor regardless of the chosen technique, i.e. it balances every sample obtained with different sampling techniques with a weighted sum. Each weighting heuristic has its own variance reduction analysis which can be found in the PhD dissertation of Eric Veach [Vea97].

One important thing to notice before proceeding is that for any realisation $\mathbf{x}_{ij}$ obtained from the sample $X_{ij} \sim p_i$ is that all the different available techniques probabilities must be computed, as if that sample was drawn from all the other techniques $p_1(x_{ij}), ..., p_M(x_{ij})$, usually assigning a bigger MIS weight w_i to the technique i with the bigger $N_i p_i$. For example if there were $M = 3$ sampling techniques i, j, k, getting the balance heuristic would require calculating $(N_i + N_j + N_k) \times 3$ probabilities.

2.2.4 Markov Chain Monte Carlo estimators

This subsection introduces a special type of importance sampling estimators called the Markov Chain Monte Carlo (MCMC) estimators. These estimators obtain samples distributed according to the pdf p_f of any positive function f, making this technique a quite versatile exploration of the sample space method. This MCMC estimator uses a special type of sequence $\mathbf{X} = (X_n)_{n \geq 0}$ of random variables called an ergodic Markov Chain, which will be explained below, starting by the concept of a Markov Chain.

Given a probability space $(\Omega, \mathcal{F}, \mathbb{P})$, the sequence $\mathbf{X} = (X_n)_{n \geq 0}$ of random variables $X_i : (\Omega, \mathcal{F}) \to (S, \mathcal{S})$ is a discrete time *Markov Chain* if it satisfies the *Markov property* [Nor97; MT09; Ros06]:

$$
\mathbb{P}(X_n = s_n | X_{n-1} = s_{n-1}, ..., X_1 = s_1, X_0 = s_0) = \mathbb{P}(X_n = s_n | X_{n-1} = s_{n-1}),
\tag{2.78}
$$

that is, the *state* $s_n \in S$ of the random variable $X_n \sim \mathbb{P}_n$ just depends on the state s_{n-1} of the random variable that is before in the sequence $X_{n-1} \sim \mathbb{P}_{n-1}$, where each $\mathbb{P}_i$ is the induced probability measure by the random variable X_i, or the random variable probability measure.

A Markov Chain only needs two things to fully define all the probability measures of each random variable in the sequence, an initial probability measure $\mathbb{P}_0$ and transition probability measures $(P(s, A))_{s \in S, A \in \mathcal{S}}$ for all elements of the state $s \in S$ such that all the elements of the set $\{P(s, .) | s \in S\}$ are probability measures.

Notice that this time there exists a correlation between random variables, so they are not i.i.d. anymore, which is a big problem as the MC estimator requires this restriction for consistency. This problem is solved by obtaining a special type of Markov Chains that have the property that from some $n \geq 0$, all the random variables have the same probability measure $\mathbb{P}_\pi$, called the *stationary distribution* of the Markov Chain, that is, $X_i \sim \mathbb{P}_\pi \ \forall i \geq n$, making all the random variables $(X_i \sim \mathbb{P}_\pi)_{i \geq n}$ identically distributed. The other problem is that they are not independent random variables, but if the Markov Chain is further required to be an ergodic Markov chain, there exists a version of the strong law of large numbers for ergodic Markov Chains [MT09] that guarantees the consistency of the MC estimator that uses the samples of an ergodic Markov Chain.

Markov Chains can have countable and uncountable state sets, the explanation of ergodicity will be done first in the countable case as it's easier to understand and then will be quickly generalized to uncountable states that are the type of states required in rendering.

Given a non empty countable set $S = I$, an initial probability distribution $(\kappa i)_{i\in I}$ and transition probabilities $(p_{ij})_{i,j\in I}$ with the typical restrictions for probability functions $\sum_{i\in I} \kappa i = 1$ and $\sum_{j\in I} p_{ij} = 1$ for all the states i, there exists a probability measure space $(\Omega, \mathcal{F}, \mathbb{P})$ and random variables defined in $(\Omega, \mathcal{F}, \mathbb{P})$ such that:

$$\mathbb{P}(X_0 = i_0, ..., X_N = i_N) = \kappa_{i_0} p_{i_0 i_1} p_{i_1 i_2} ... p_{i_{N-1} i_N}, \tag{2.79}$$

for all $N \in \mathbb{N}$ and $i_0, ..., i_N \in I$ [Ros06]. The matrix $P = (p_{ij})_{i,j\in I}$ is a *probability transition matrix* which tells the probability $p_{ij} = \mathbb{P}(X_{n+1} = j|X_n = i)$ of going from a state i to a state j, then because of this, all the random variables distributions are determined by the transition matrix P and the initial distribution row vector κ:

$$\begin{aligned} X_0 &\sim \kappa = \mathbb{P}_0 \\ X_1 &\sim \kappa P = \mathbb{P}_0 P = \mathbb{P}_1 \\ &\vdots \\ X_k &\sim \kappa P^k = \mathbb{P}_{k-1} P = \mathbb{P}_k, \end{aligned} \tag{2.80}$$

which makes intuitive sense as the state i of the next random variable is obtained by the expression $(\mathbb{P}_k)_i = \sum_{j=1} \kappa_j^{(0)} P_{ji}^k$ which in words is equivalent to saying: "The probability of reaching state i at time k equals the probabilities of starting at state j times the probability of going from j to i in k steps for all possible states $j \in I$".

A discrete time with countable states Markov Chain might be of several types:

1. *Irreducible Markov Chain*: We say that a state j is *reachable* from state i, denoted as $i \to j$ if $\mathbb{P}(X_{k+n} = j|X_k = i) > 0$ for some $n \geq 1$. Any two states i, j are said to be *communicating states* if $i \to j$ *and* $j \to i$, denoted as $i \leftrightarrow j$. A Markov Chain in which every pair of states $i, j \in I$ are communicating states is called an irreducible Markov Chain.

2. *Aperiodic Markov Chain*: The *period of a state* i is defined as the greatest common divisor $d = gcd\{n \geq 1 : p_{ii}^n > 0\}$, which means that the state can only be reached after multiples of d. If all the states of the Markov Chain have period 1, then the chain is said to be an aperiodic Markov Chain.

3. *Positive recurrent Markov Chain*: If the expected time of return $m_i = E[\inf\{n \geq 1 : X_n = i\}]$ to any of the states is finite the chain will be called positive recurrent.

4. *Ergodic Markov Chain*; If the discrete time countable state Markov Chain is irreducible, aperiodic and positive recurrent, the chain is called an ergodic Markov Chain [Ros23].

5. *Reversible Markov Chain*: A Markov chain is said to be a reversible Markov Chain if there exists some distribution $\mathbb{P}_\pi$ that follows the *detailed balance* condition:

$$(\mathbb{P}_\pi)_i p_{ij} = (\mathbb{P}_\pi)_j p_{ji} \quad \forall i, j \in I, \tag{2.81}$$

which means that the probability of arriving at state i and leaving to state j is the same that the probability of arriving at state j and leaving to state i

If the discrete time Markov Chain with countable states is irreducible and positive recurrent [Ros06] there exists a unique distribution $\mathbb{P}_\pi$ that satisfies:

$$\mathbb{P}_\pi P = \mathbb{P}_\pi, \tag{2.82}$$

which means that the probability of arriving to a state from some other state won't change with respect of the time anymore, this distribution is known as the *stationary distribution*. If we assume the existence of the stationary distribution, then for an ergodic Markov Chain we have the *Markov Chain limiting theorem* [Ros06] which states that the sequence of state distributions $(\mathbb{P}_k)_{k\geq 0}$ converges to the stationary distribution:

$$\lim_{k\to\infty} \kappa P^k = \lim_{k\to\infty} \mathbb{P}_k = \mathbb{P}_\pi. \tag{2.83}$$

If a Markov Chain is reversible with respect to $\mathbb{P}_\pi$ (Equation 2.81), this implies that $\mathbb{P}_\pi$ is a stationary distribution of the chain [Ros06], meaning that $\mathbb{P}_\pi$ does indeed exist. Therefore if an ergodic chain is modeled after some transition probability matrix P that meets the detailed balance condition for some distribution $\mathbb{P}_\pi$, it can be ensured that the initial distribution κ will converge to the stationary distribution $\mathbb{P}_\pi$.

This discussion can be extended to uncountable state sets $S = R$ [Ros06; Vea97], specifically over the real numbers, which is the case used in rendering. Consider any initial probability measure $\nu(A \in \mathcal{R})$ in the measurable space $(R, \mathcal{R})$, where $\mathcal{R}$ is a σ-algebra over R, and the non countable equivalent of the transition probabilities $(P(x, A))_{x \in R, A \in \mathcal{R}} = \mathbb{P}(X_{i+1} \in A | X_i = x)$, where for each fixed $x \in R$, $P(x, .)$ is a probability measure on $(R, \mathcal{R})$, and for each fixed $A \in \mathcal{R}$, $P(x, A)$ is a measurable function of $x \in R$. This probability transitions can often be represented in terms of a probability density function $p(x, y) = p(y|x) = p(x \to y)$ between elements of the state:

$$P(x, A) = \int_{y \in A} p(x, y) \, \mathrm{d}\mu(y), \tag{2.84}$$

which as it was already mentioned will change depending on the measure used and is denoted as $p^\mu(x, y)$, fact that will be quite important in Markov Chains applied to rendering.

The general state, or continuous state Markov Chain probability is then given by [MT09]:

$$\begin{aligned} \mathbb{P}(X_0 \in A_0, X_1 \in A_1, ..., X_N \in A_N) = \int_{x_0 \in A_0} \nu(\mathrm{d}x_0) \int_{x_1 \in A_1} P(x_0, \mathrm{d}x_1)... \\ ... \int_{x_{N-1} \in A_{N-1}} P(x_{N-2}, \mathrm{d}x_{N-1}) P(x_{N-1}, A_N), \end{aligned} \tag{2.85}$$

notice the use of measure theory notation $P(x, \mathrm{d}y)$ which roughly means the transition probability from a state x to some differential set. In the same manner that in the discrete case probabilities after $n+1$ steps could also be defined for the continuous case:

$$P_{n+1}(x, A) = \int_R P(x, \mathrm{d}z) P_n(z, A), \qquad P_1(x, A) = P(x, A), \tag{2.86}$$

and in the same way that the discrete case (Equation 2.80), each random variable has its own probability measure function:

$$\begin{aligned} X_0 &\sim (\nu(A))_{A \in \mathcal{R}} = (P_0(A))_{A \in \mathcal{R}} \\ X_1 &\sim \int_R P_0(\mathrm{d}x) P(x, A) = \int_{x \in R} \int_{y \in A} p_0(x) p(x, y) \, \mathrm{d}\mu(x) \, \mathrm{d}\mu(y) = (P_1(A))_{A \in \mathcal{R}} \\ &\vdots \\ X_i &\sim \int_R P_{i-1}(\mathrm{d}x) P(x, A) = \int_{x \in R} \int_{y \in A} p_{i-1}(x) p(x, y) \, \mathrm{d}\mu(x) \, \mathrm{d}\mu(y) = (P_i(A))_{A \in \mathcal{R}}, \end{aligned} \tag{2.87}$$

where clearly each random variable just depends of the distribution of the random variable before and the transition probability measures, thus

$$p_i(x) = \int_R p_{i-1}(y) p(y, x) \, \mathrm{d}\mu(y). \tag{2.88}$$

Some probability transitions $(P(x, A))_{x \in R, A \in \mathcal{R}}$ have a continuous stationary distribution, also called an *invariant measure* [MT09] defined as:

$$\mathbb{P}_\pi(A) = \int_R \mathbb{P}_\pi(\mathrm{d}w) P(w, A) = \int_R \pi(w) P(w, A) \, \mathrm{d}\mu(w) = \int_{w \in R} \int_{x \in A} \pi(w) p(w, x) \, \mathrm{d}\mu(w) \, \mathrm{d}\mu(x). \tag{2.89}$$

where π is the pdf of the probability distribution measure $\mathbb{P}_\pi$.

For a discrete time with uncountable state Markov chain, the initial distribution ν converges to the stationary distribution, assuming it exists, if the Markov Chain is a positive Harris recurrent and aperiodic chain [MT09], these type of chains are called *ergodic Markov Chains*. This in a very simplified way that ignores some important details[11] is: A Harris recurrent Markov Chain basically means that all the subsets $A \in \mathcal{R}$ must be able to be visited from any state $s \in R$ an infinite amount of times for an infinite time, aperiodic in simplified terms means that every set $A \in \mathcal{R}$ is not confined to be visited only in multiple of some integer bigger than one from any initial state, and a positive chain means that it is irreducible (every set $A \in \mathcal{R}$ can be visited in a finite amount of time from any initial state) and a stationary distribution exists.

The continuous detailed balance for a probability measure Π and for some transition probabilities $(P(x, A))_{x\in R, A\in\mathcal{R}}$ is defined as [MT09]:

$$\mathbb{P}_\pi(\mathrm{d}x)P(x, \mathrm{d}y) = \mathbb{P}_\pi(\mathrm{d}y)P(y, \mathrm{d}x) \tag{2.90}$$

and noting that Equation 2.90 is met if we have the continuous detailed balance for pdfs:

$$\pi(x)p(x, y) = \pi(y)p(y, x) \quad \rightarrow \quad \mathbb{P}_\pi(\mathrm{d}x)P(x, \mathrm{d}y) = \mathbb{P}_\pi(\mathrm{d}y)P(y, \mathrm{d}x), \tag{2.91}$$

this detailed balance in the same way that the discrete state case is a sufficient condition for the existence of a stationary distribution, furthermore if the probability transitions define an aperiodic positive Harris recurrent chain then any initial probability measure $(\nu(x))_{x\in R}$ converges to the stationary probability measure $\mathbb{P}_\pi$.

The essence of the *Metropolis-Hastings algorithm* [Met+53; Has70], is the following:

- Construct a probability density between all the states $K(\mathbf{x} \to \mathbf{y}) = K(\mathbf{y}|\mathbf{x}) = p(x, y)$ in such a way that all the states are reachable from all the other states in an aperiodic and finite amount of time, that is, the probability density of the n-th random variable $p_n(\mathbf{x}) > 0$ for some n aperiodic and finite, and besides of that it must be guaranteed that if the chain was run an infinite amount of time all of the states stay always reachable.

- Use any positive function $f(\mathbf{x})$, which will be called the *importance function* of the Markov Chain, with an associated pdf p_f that will be the stationary *target distribution* $p_f(\mathbf{x}) = \pi(\mathbf{x})$ of the chain. This can be achieved by making this target function satisfy the continuous detailed balance condition 2.91 using our already defined ergodic transition density $p(x, y)$, ensuring the ergodicity of the Markov chain.

The Metropolis-Hastings (MH) algorithm transition density K is defined by two independent (from each other) random processes: the random process of selecting a proposal state (mutation) $\mathbf{y}$ from the current state $\mathbf{x}$ with ergodic probability density $T(\mathbf{x} \to \mathbf{y})$, and the random process of accepting (or rejecting) this proposed state with ergodic probability density $\alpha(\mathbf{x} \to \mathbf{y})$:

$$X_{k+1} = \begin{cases} X'_{k+1} \text{ with probability } \alpha(X_k \to X'_{k+1}) \\ X_k \text{ otherwise} \end{cases} \quad ; \quad K(\mathbf{x} \to \mathbf{y}) \equiv T(\mathbf{x} \to \mathbf{y})\alpha(\mathbf{x} \to \mathbf{y}), \tag{2.92}$$

then the detailed balance applied to this newly defined Kernel is:

$$\frac{f(\mathbf{x})}{\int_\Omega f(\mathbf{x}')\,\mathrm{d}\mu(\mathbf{x}')}\alpha(\mathbf{x} \to \mathbf{y})T(\mathbf{x} \to \mathbf{y}) = \frac{f(\mathbf{y})}{\int_\Omega f(\mathbf{x}')\,\mathrm{d}\mu(\mathbf{x}')}\alpha(\mathbf{y} \to \mathbf{x})T(\mathbf{y} \to \mathbf{x}). \tag{2.93}$$

From Equation 2.93 we can realize that the normalization factors cancel out, being this the reason that any non normalized positive function f could be used as a stationary distribution in the MH algorithm. Also notice that the acceptance stochastic process was introduced because without that process it wouldn't be clear how to satisfy the detailed balance condition. If $p_{i-1} = p_f$ already, the stationary property holds [Vea97]:

$$\begin{aligned} p_i(\mathbf{x}) &= p_{i-1}(\mathbf{x}) \overbrace{\left[1 - \int_\Omega T(\mathbf{x} \to \mathbf{y})\alpha(\mathbf{x} \to \mathbf{y})\,\mathrm{d}\mu(\mathbf{y})\right]}^{\text{P of staying at } \mathbf{x} \text{ (Reject proposals)}} + \overbrace{\int_\Omega p_{i-1}(\mathbf{y})T(\mathbf{y} \to \mathbf{x})\alpha(\mathbf{y} \to \mathbf{x})\,\mathrm{d}\mu(\mathbf{y})}^{\text{P of arriving at } \mathbf{x} \text{ from all } \mathbf{y} \text{ (Accept proposals)}} \\ &= p_{i-1}(\mathbf{x}) + \int_\Omega \left[p_{i-1}(\mathbf{y})T(\mathbf{y} \to \mathbf{x})\alpha(\mathbf{y} \to \mathbf{x}) - p_{i-1}(\mathbf{x})T(\mathbf{x} \to \mathbf{y})\alpha(\mathbf{x} \to \mathbf{y})\right]\mathrm{d}\mu(\mathbf{y}) = p_{i-1}(\mathbf{x}) \end{aligned} \tag{2.94}$$

[11] If a math rigorous book on these topic wants to be checked, Meynn and Tweedie [MT09] is highly recommended.

In order for the initial distribution to converge as fast as possible to the target function, the values of the acceptance distribution $\alpha(\mathbf{y} \to \mathbf{x})$ must be as large as possible, therefore making $\alpha(\mathbf{y} \to \mathbf{x}) = 1$, the acceptance probability density of the Metropolis-Hastings (MH) algorithm is:

$$\alpha(\mathbf{x} \to \mathbf{y}) = \min\left(1, \frac{f(\mathbf{y})T(\mathbf{y} \to \mathbf{x})}{f(\mathbf{x})T(\mathbf{x} \to \mathbf{y})}\right). \tag{2.95}$$

The random process due to the transition probability density T is called a *mutation*. An example of two type of mutations are:

1. *Small mutations*: Which modify the current state $\mathbf{x}$ in just a small amount. This type of mutation is quite useful to explore close regions of the space which will probably contain similar contributions. These mutations by themselves do not meet the ergodicity requirement as the chain could get trapped in an isolated region of the sampling space forever. A typical small mutation is $(X_{i+1}|X_i = \mathbf{x}_i) \sim \mathcal{N}(\mathbf{x}_i, \sigma)$, but we will also use the *Hachisuka small mutation* [HJ11]:

Algorithm 1 Hachisuka small mutation

Input: Random number u to be mutated.
Output: Mutated random number v.

```
ξ ← rand()                 ▷ Unifromly distributed random number in [0, 1)
v ← u
add ← False
if ξ < 0.5 then
    add ← True
    ξ ← ξ × 2.0
else
    ξ ← 2.0 × (ξ − 0.5)
end if
dv ← ξ^2                   ▷ Special case θ = 1 of the paper's [HJ11] formula ξ^{1/θ+1}
if add then
    v ← v + dv
    if v > 1 then
        v ← v − 1
    end if
else
    v ← v − dv
    if v < 0 then
        v ← v + 1
    end if
end if
return v
```

Notice that the Hachisuka mutation can induce changes of even 1 full unit, unlike the normal distributed mutation in which almost all the mutated samples fall in the range $[-3\sigma, 3\sigma]$.

2. *Large mutation*: This type of mutation is essential to ensure ergodicity, it takes a sample in a far place from the current state. A typical large mutation just ignores the current state, and takes a completely new random sample from any place of the whole state space $(X_{i+1}|X_i = \mathbf{x}_i) \sim U(R)$.

The pseudo code of algorithm that accounts for both type of mutations[12] is given in algorithm 2.

If the Metropolis Hastings algorithm is used and the initial distribution has already converged to the stationary distribution $\pi(\mathbf{x}) = p_f(\mathbf{x}) = \frac{f(\mathbf{x})}{\int_{\mathbf{x}\in\Omega} f(\mathbf{x})\,\mathrm{d}\mu(\mathbf{x})} = \frac{f(\mathbf{x})}{b}$, then the samples of a Monte Carlo estimator can be obtained using the samples generated by the Markov Chain. The Monte Carlo

[12]The mathematical justification of using two type of mutations at the same time can be explained by Equation 3.33, assuming as if this single chain was two simultaneous chains with the same state, one chain for large mutations and one chain for small mutations.

Algorithm 2 Metropolis-Hastings Algorithm

Input: Target distribution $\pi(\mathbf{x})$, proposal distributions $T_L(\mathbf{x} \to \mathbf{y})$, $T_s(\mathbf{x} \to \mathbf{y})$ initial value $\mathbf{x}_0$
Output: Sequence of samples $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$ approximating $\pi(\mathbf{x})$

```
Initialize x ← x_0
for i = 1 to n do
    Sample u_1 ~ Uniform(0,1)
    if u_1 ≤ p_large then
        T_i ← T_L
    else
        T_i ← T_s
    end if
    Sample y ~ T_i(x → y)
    Compute acceptance ratio α_i = min(1, π(y)T_i(y→x) / (π(x)T_i(x→y)))
    Sample u_2 ~ Uniform(0, 1)
    if u_2 ≤ α_i then
        Accept the proposal: x ← y
    end if
    Set x_i ← x
end for
```

estimator that uses samples from a Markov Chain that is carefully crafted to converge to the stationary distribution is called a *Markov Chain Monte Carlo* (MCMC) estimator, with stationary distribution p_f. For example, the MCMC estimator of a function f with MCMC pdf p_g would be then:

$$I = \int f(\mathbf{x})\,\mathrm{d}\mu(\mathbf{x}) \approx I^{(N)} = \frac{1}{N}\sum_{i=1}^{N} \frac{f(\mathbf{x}_i)}{p_{\mathrm{MCMC}}(\mathbf{x}_i)} = \frac{\int_\Omega g(\mathbf{x})\,\mathrm{d}\mu(\mathbf{x})}{N}\sum_{i=1}^{N}\frac{f(\mathbf{x}_i)}{g(\mathbf{x}_i)}. \tag{2.96}$$

In section 2.1 we said that the rendering equation is a three channel function f, but in order to be able to apply the MH algorithm we need a scalar stationary pdf for the Markov chain. Thus, we will use as stationary distribution π the scalar function f^*, which as we already mentioned returns a value that was obtained using the three RGB values of f. We will use the luminance defined in Equation 2.10 as our scalar function. With these consideration we have the MCMC estimator used in rendering

$$I^{(N)} = \frac{\int_\Omega f^*(\mathbf{x})\,\mathrm{d}\mu(\mathbf{x})}{N}\sum_{i=1}^{N}\frac{f(\mathbf{x}_i)}{f^*(\mathbf{x}_i)} = \frac{b}{N}\sum_{i=1}^{N}\frac{f(\mathbf{x}_i)}{f^*(\mathbf{x}_i)}. \tag{2.97}$$

The Metropolis-Hastings algorithm applied to rendering is called *Metropolis Light Transport* (MLT) and it is extremely useful as it allows us to explore proportionally to the luminance of the measurement equation, spending more time in zones of high radiance. This algorithm is also extremely useful for the exploration of difficult light paths by using small mutations.

Notice a subtlety, we actually don't have one measurement equation! We have A_{image} measurement equations, one for each pixel. The target distribution of the Markov process is actually the sum [GWH20]:

$$\pi(\mathbf{x}) = \sum_{j=1}^{A_{\mathrm{image}}} h_j(\mathbf{x}) f_j^*(\mathbf{x}), \tag{2.98}$$

where the function h can be simply interpreted as an indicator function, that returns one if the sample $\mathbf{x}$ belongs to pixel j, which in rendering it's actually a box filter. More generally, function h_j is the filter of pixel j, but for simplicity we will just treat it as the indicator function that splats the respective sample into the respective pixel, that is, $f(\mathbf{X}_k)$ for the MCMC process with a box filter is implied to be $f_j(\mathbf{X}_k)$, where the pixel j is selected by the raster position of the sample $\mathbf{X}_k$. The fact that the target distribution of the Markov Chain is the sum of all the integrands of all the measurement equation implies that the Markov Chain is not confined to a single pixel, it can visit every possible pixel of the image, which in a positive note, indicates that only one normalization constant b is required for the whole film. This unfortunately results in a lack of stratification in the MCMC estimator.

There is still one important thing to address, the MCMC estimator that chooses some arbitrary initial random value $\mathbf{X}_0$ is a consistent but biased estimator, due to the fact that the Markov Chain needs some unknown time to converge to the stationary distribution, so each of the initial samples would technically have an incorrect probability density function on the MCMC estimator, the bias caused by the non initial convergence of the Markov Chain is called *start up bias*. The naive way of solving this is to run the chain some time before starting the algorithm, in a process known as *burn in* hoping that it ends up converging to the stationary distribution, this method is not that good because we don't have any guarantee of this happening in the arbitrary chosen amount of time. In the original Metropolis Light Transport paper [VG97], the unbiasedness was addressed by sampling the initial state of the chain proportional to a known pdf $\mathbf{X}_0 \sim p_o$, and assigning a weight $W_0 = f^*(\mathbf{X}_0)/p_0(\mathbf{X}_0)$ to the first sample to account for the discrepancy of pdfs, effectively exchanging bias by statistical noise. It was proven that if each MCMC sample k was divided by $f^*(\mathbf{X}_k)/W_k$, instead of $f^*(\mathbf{X}_k)/b$, and if each weight was made equal to the starting weight, that is, $W_0 = W_1 = ... = W_N = W$ then this newly defined estimator would be unbiased

$$I_j = E\left[\frac{1}{N}\sum_{i=1}^{N} W_i \frac{f(\mathbf{X}_i)}{f^*(\mathbf{X}_i)}\right] = E\left[\frac{W}{N}\sum_{i=1}^{N} \frac{f(\mathbf{X}_i)}{f^*(\mathbf{X}_i)}\right], \tag{2.99}$$

this weighting scheme makes the estimator unbiased but inconsistent, which means that a single Markov Chain could never give the correct solution at all! If we want to use start-up bias elimination we need to average the run of several independent Markov Chains to get the correct result, which is in fact done in rendering by running multiple Markov Chains in a parallelized manner, the exact number of chains is defined by the user as a parameter of the algorithm, for example, pbrt uses 100 chains as a default. The *average* of the values returned by these chains is the correct solution, and if we had C chains then the average would be

$$I_j \approx \sum_{k=1}^{C} \frac{N_k}{N}\left[\frac{W_{C_k}}{N_k}\sum_{i=1}^{N_k} \frac{f(\mathbf{X}_i)}{f^*(\mathbf{X}_i)}\right]; \qquad N = \sum_{i=1}^{C} N_i, \tag{2.100}$$

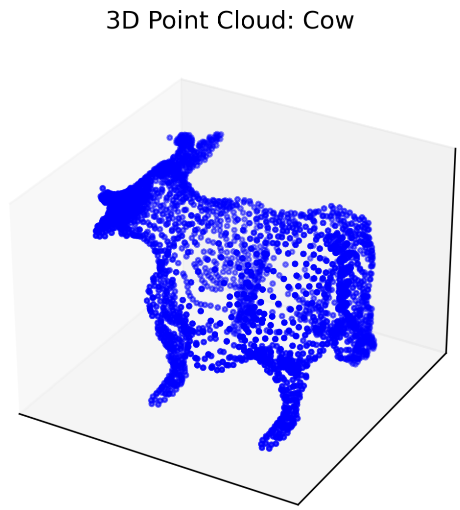
where each MCMC sampler was weighted in a proportional way with respect of the number of samples, that is, N_k/N. Notice that both the biased and unbiased MCMC algorithms are parallelizable by simply running a several Markov Chain on each available thread.

2.2.5 Examples of MCMC sampling

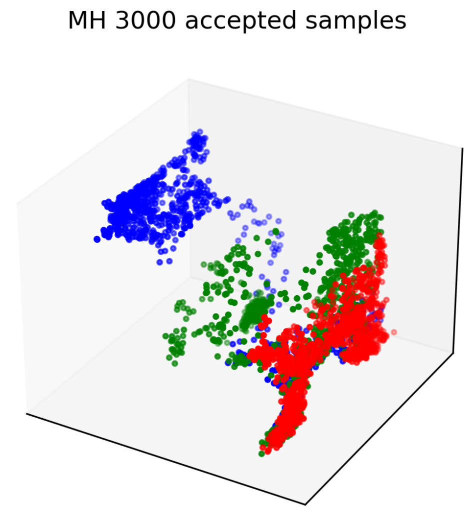
Cloud of points The MH algorithm has an incredible exploration capacity due to the fact that any positive function could be used as a target distribution, for example, a cloud of M points $(\mathbf{y}_i)_{i=1}^{M}$ defined by a polygon file formal (ply) with any shape could be used, by defining some arbitrary target function like $f(\mathbf{x}) = e^{-100D(\mathbf{x})}$, where

$$D(\mathbf{x}) = \min_{1\leq j\leq M}\{d(\mathbf{x}, \mathbf{y}_j)\}, \quad d(x,y) = \sqrt{(y_1-x_1)^2 + (y_2-x_2)^2 + (y_3-x_3)^2}, \quad \mathbf{x} = (x_1, x_2, x_3) \tag{2.101}$$

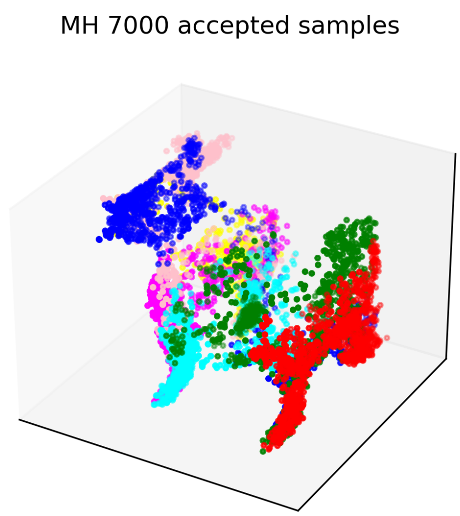
for an example run with 10 million samples using only normal distributed with $\sigma = 0.01$ small mutations, see Figure 2.5.



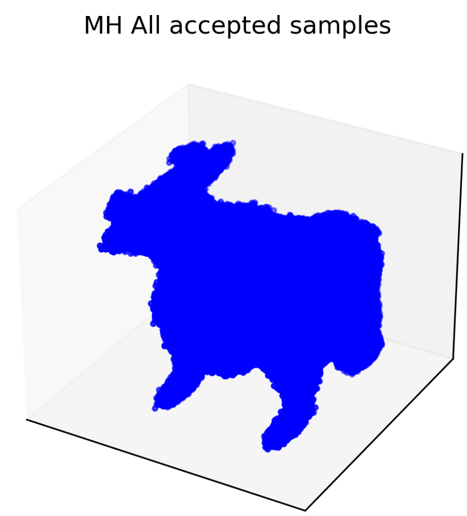
(a) Original cloud of points



(b) Cloud of points obtained using the Metropolis-Hastings algorithm. We are visualizing 3000 accepted samples, samples $1-1000$ are green, $1001-2000$ are red, and $2001-3000$ are blue.



(c) 7000 accepted MCMC mutations, 1000 of each color.

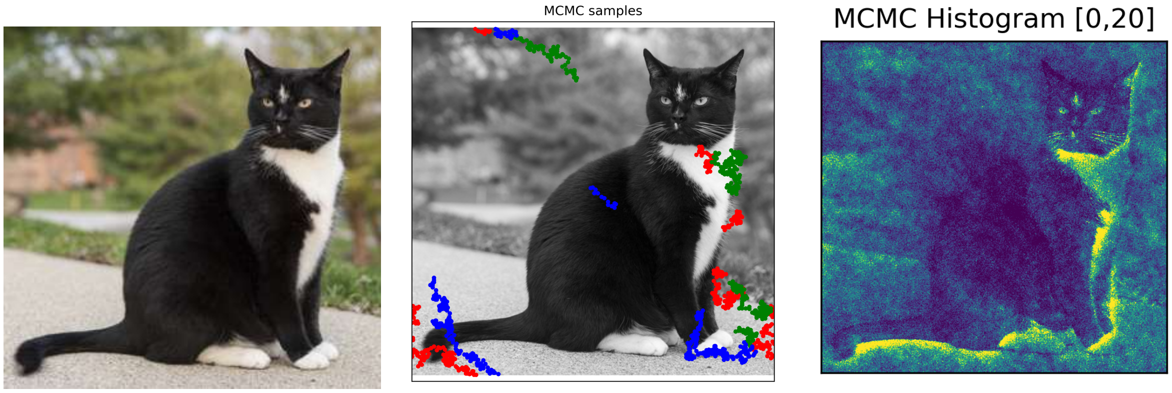


(d) The full result of the algorithm that was run 10 million samples.

Figure 2.5: The Metropolis-Hastings algorithm is quite powerful for exploring a state space using arbitrary target distributions as its limiting distribution (in this example, Equation 2.101). We can see the correlation on each accumulation of colors. This was obtained running 10 million samples using only small normal distributed mutations with $\sigma = 0.01$ in order to visualize better the samples' correlation.

Image This algorithm is extremely versatile, another example of this can be seen by defining as target function the grayscaled version (in the range 0-255) of an RGB image that was transformed using the typical grayscale conversion provided by the python library of image processing called Pillow[13] $\text{Gray}(r, g, b) = 0.299r + 0.587g + 0.114b$. Our small mutations consists in choosing a new pixel p' like $p' = p + \xi$ with $\xi \sim U[-5, 4]$ (if we go out of range, we simply go to the other side of the image, for example $p'_x = -2 \rightarrow p'_x = \text{WIDTH} - 2$), and we only perform a large mutation (choosing any arbitrary pixel) with probability $p_{\text{large}} = 0.01$ (see Figure 2.6).

[13] This library uses a luminance function to transform from RGB to grayscale. But this luminace was defined using different perceptual parameters than our luminance (used by pbrt) given in Equation 2.10.



(a) Reference image taken from Wikipedia, the image that we used was re-scaled to $500 \times 480 = 240000$ pixels.

(b) Visualization of the MCMC process. The accepted samples 1-400 are red, 401-800 are green and 801-1200 are blue.

(c) The histogram of the MH algorithm that was run 1 million samples, with values clamped in the range 0-20 for visualization purposes.

Figure 2.6: MH algorithm run 1 million iterations with importance function equal to the grayscale image version of (a). We use large mutations with probability $p_{\text{large}} = 0.01$ that can select any pixel p of the image. Small mutations were uniformly distributed as $p' = p + \xi$ with $\xi \sim U[-5, 4]$.

Integration We will integrate a toy 1 dimensional function just to illustrate the process that is done in rendering. We define our "rendering equation" $f(x)$ over the integration domain $[0, 1]$, which is going to be divided in N_b "pixels" (bins), in such a way that the full function $f(x)$ gets partitioned in multiple sub-integrals $I_j = \int_{j\Delta x}^{(j+1)\Delta x} f(x)\,\mathrm{d}x = \int_{[0,1]} h_j(x) f(x)\,\mathrm{d}x$, with $\Delta x = \frac{1}{N_b}$, and where $b = \int_0^1 f(x)\,\mathrm{d}x$ is a normalization constant that *all* the pixels (bins) are going to share, and $h_j(x)$ is a function that equals one for values of x inside the j pixel, and zero for values outside. The explicit value of our function is:

$$f(x) = \begin{cases} (x - 1/2)^2 \text{ if } 0 \leq x \leq 1 \\ 0 \text{ otherwise} \end{cases} \qquad b = \int_0^1 f(x)\,\mathrm{d}x = \frac{1}{12} \tag{2.102}$$

with analytical values:

$$I_j = \int_{j\Delta x}^{(j+1)\Delta x} f(x)\,\mathrm{d}x = \frac{((j+1)\Delta x - 1/2)^3 - (j\Delta x - 1/2)^3}{3} \tag{2.103}$$

where of course in rendering I_j are the values of the image to be integrated that we don't know, and the common b among all the pixels must be estimated in some way. We present the python pseudocode (see code 2.1) for both the biased (Equation 2.97) and unbiased (Equation 2.100) MH algorithm for $N_b = 20$ hypothetical pixels. We apply the two mutations $\text{mutate}_1(X) = u, , \text{mutate}_2(X) = X + 0.1(u - 0.5), u \sim U[0, 1]$ with probability of choosing mutate_1 of 0.1. We sample the initial value of our chains uniformly in the interval $[0, 1)$, therefore our Veach weight for the unbiased but inconsistent case is $W = \frac{f(X_0)}{p_0(X_0)} = f(X_0)$. We show the analytical values of the integrals of all the bins (pixels) in table 2.2, the biased but consistent estimated values in table 2.3, and the unbiased but inconsistent estimated values in table 2.4.

```python
np.random.seed(0) #For replicability of numbers
def f(x):
    if x<0.0 or x>1.0:
        return 0.0
    return (x-0.5)**2
bf=1.0/12.0 #it can be analytically calculated

def mh(f,N,nBins,rang,useW):
    X0=rand()
    w=f(X0) #Veach weight W because p0=1
    Xc=X0 #Xc=Xcurrent
    #Xp=Xpropossed
    fc=f(Xc)
```

```
    diff=float(rang[1]-rang[0])
    hist=np.array([0]*nBins)
    pL=0.1 #plarge
    for i in range(N):
        if rand()<pL:
            Xp=rand()*diff
        else:
            Xp=Xc+0.1*(rand()-0.5)
        a=min(1.0,f(Xp)/f(Xc))
        if rand()<a:
            Xc=Xp
        inde=find_bin_index(Xc,nBins,rang)
        hist[inde]+=1
    if useW:
        return hist*w
    else:
        return hist

NSamplesSingle=10000000 #For biased and consistent
NsamplesMultiple=100000 #For unbiased and inconsistent
noBins=20 #Pixels
x_range=[0.0,1.0] #Integration domain
#Run and scale biased and consistent algorithm
single_run_hist=mh(f,NsamplesSingle,noBins,x_range,False)
result_single=bf*single_run_hist/NsamplesSingle

#Run inconsistent but biased algorithm averaging 100 runs (chains)
average_hist=[]
Nchains=100
for i in range(Nchains):
    #Scaled by W Veach Weight (True parameter)
    hist=mh(f,NsamplesMultiple,noBins,x_range,True)
    if i==0:
        average_hist=hist
    else:
        average_hist+=hist
#Unbiased and inconsistent
result_multiple=average_hist/(Nchains*NsamplesMultiple)
```

Code 2.1: MH algorithn python code to integrate all the pixles with Markov Chains that can visit all of the pixels, and use a single shared (by all the pixels) normalization constant b. We show both a biased and consistent Markov Chain integration to calculate the integrals, and the average of several unbiased but inconsistent Markov Chains to calculate the integrals.

R_0	R_1	R_2	R_3	R_4
1.12916667e-02	9.04166667e-03	7.04166667e-03	5.29166667e-03	3.79166667e-03
R_5	R_6	R_7	R_8	R_9
2.54166667e-03	1.54166667e-03	7.91666667e-04	2.91666667e-04	4.16666667e-05
R_{10}	R_{11}	R_{12}	R_{13}	R_{14}
4.16666667e-05	2.91666667e-04	7.91666667e-04	1.54166667e-03	2.54166667e-03
R_{15}	R_{16}	R_{17}	R_{18}	R_{19}
3.79166667e-03	5.29166667e-03	7.04166667e-03	9.04166667e-03	1.12916667e-02

Table 2.2: Each element of this table has the full analytical reference values R_j given by 2.103.

$I_0^{(B)}$	$I_1^{(B)}$	$I_2^{(B)}$	$I_3^{(B)}$	$I_4^{(B)}$
1.13105333e-02	9.01770833e-03	7.02047500e-03	5.26150000e-03	3.77003333e-03
$I_5^{(B)}$	$I_6^{(B)}$	$I_7^{(B)}$	$I_8^{(B)}$	$I_9^{(B)}$
2.53970000e-03	1.53298333e-03	7.94833333e-04	2.92458333e-04	4.11666667e-05
$I_{10}^{(B)}$	$I_{11}^{(B)}$	$I_{12}^{(B)}$	$I_{13}^{(B)}$	$I_{14}^{(B)}$
4.06083333e-05	2.90933333e-04	7.96458333e-04	1.55994167e-03	2.56150833e-03
$I_{15}^{(B)}$	$I_{16}^{(B)}$	$I_{17}^{(B)}$	$I_{18}^{(B)}$	$I_{19}^{(B)}$
3.82582500e-03	5.32210833e-03	7.05846667e-03	9.03633333e-03	1.12597583e-02

Table 2.3: Each element of this table contains the numerical result $I_j^{(B)}$ of the biased and consistent MH algorithm with ten million samples in the whole "film". The MSE error ($\frac{1}{20}\sum_{i=0}^{19}(R_i - I_i^{(B)})^2$) of the "image" is $3.510919e-10$.

$I_0^{(U)}$	$I_1^{(U)}$	$I_2^{(U)}$	$I_3^{(U)}$	$I_4^{(U)}$
1.13522513e-02	9.12647601e-03	7.12383121e-03	5.40401907e-03	3.89083937e-03
$I_5^{(U)}$	$I_6^{(U)}$	$I_7^{(U)}$	$I_8^{(U)}$	$I_9^{(U)}$
2.62807005e-03	1.58580172e-03	8.22511872e-04	3.02967360e-04	4.13633971e-05
$I_{10}^{(U)}$	$I_{11}^{(U)}$	$I_{12}^{(U)}$	$I_{13}^{(U)}$	$I_{14}^{(U)}$
4.03813509e-05	2.94669544e-04	8.11246943e-04	1.59930128e-03	2.60228143e-03
$I_{15}^{(U)}$	$I_{16}^{(U)}$	$I_{17}^{(U)}$	$I_{18}^{(U)}$	$I_{19}^{(U)}$
3.88608290e-03	5.45628123e-03	7.22476966e-03	9.30263634e-03	1.15907172e-02

Table 2.4: Each element of this table contains the numerical result $I_j^{(U)}$ of the unbiased and inconsistent algorithm with 100000 samples per chain and 100 chains. The MSE error ($\frac{1}{20}\sum_{i=0}^{19}(R_i - I_i^{(U)})^2$) of the "image" is $1.425152e-08$

Note that for the biased and consistent case we are scaling by a factor $c_B = \frac{b}{N}$, or its unbiased but inconsistent version that averages C Markov Chains, each one scaled by a factor of $c_{U_i} = \frac{W_{C_i}}{N}$, and as all the chains have the same amount of samples we can then scale the chains by $\frac{1}{NC}$ (where each chain is implied to already have the constant W_{C_i}).

As we already discussed, the normalization integral is shared among all the pixels,

$$b = \sum_{j=0}^{N_b-1} \int_{A_j} h_j(x) f(x)\, \mathrm{d}\mu(x) \equiv \int_A h_j f(x)\, \mathrm{d}\mu(x), \tag{2.104}$$

and its physical meaning is just the average measurement over the whole image, which is an inherent property of the film itself regardless of the amount of pixels (refinement of the intervals). This constant could be denoted as $< b >$ to explicitly remind us of the normalization factor with respect of the amount of pixels, but we will still call this average value b respecting the literature's notation. In our example $b = 1/12$ regardless of the amount of bins, and it's usually calculated with a uniform MC (not MCMC) estimator over the whole integration domain (all the pixels, or bins).

In our example, we used the analytical value for b, but in rendering we don't have this information, so a way of calculating this value is by using a uniformly distributed MC estimator that takes N_{init} bootstrap samples across all the pixels:

$$b \approx \frac{1}{N_{\text{init}}} \sum_{i=0}^{N_{\text{init}}-1} f(x_i) h_j(x_i), \tag{2.105}$$

each of the samples can be saved in a initialization bootstrap array

$$\mathcal{B} = [(f(x_0), x_0), (f(x_1), x_1), ..., (f(x_{N_{\text{init}}-1}), x_{N_{\text{init}}-1})] \tag{2.106}$$

which is going to be used as the initial distribution $X_0 \sim p_0$ to eliminate start up bias, sampling the initial element x_0 of the Markov Chain proportionally to $\mathcal{B}$

$$p_0(x) = \frac{f(x)}{\frac{1}{N_{\text{init}}}\text{sum}(\mathcal{B})} = \frac{f(x)}{b}, \qquad \text{sum}(\mathcal{B}) = \sum_{i=0}^{N_{\text{init}}-1} f(x_i) = \sum_{i=0}^{N_{\text{init}}-1} \mathcal{B}[i][0], \tag{2.107}$$

then the start-up weight would be

$$W(x_0) = \frac{f(x_0)}{p_0(x_0)} = b = E[W], \tag{2.108}$$

which means that the unbiased and inconsistent case for this election of p_0 looks as the biased and consistent algorithm. In rendering due to processes that use clever samplings, it seems like the probabilities are not being taken into account when in reality things are often getting cancelled out, see equations 2.108, 3.14 and 3.27.

There is still one subtlety that its explanation often gets overlooked, in rendering we need to multiply everything by the amount of pixels, i.e., A_{image} (in our case N_b), and this is because if we don't account for this factor the reconstruction of the "image" would be wrong. To see this consider the "image":

$$I = [I_0, I_1, ..., I_{N_b-1}] = \left[\int_{0\Delta x}^{1\Delta x} f(x)\,\mathrm{d}x, \int_{1\Delta x}^{2\Delta x} f(x)\,\mathrm{d}x, ..., \int_{(N_b-1)\Delta x}^{N_b\Delta x} f(x)\,\mathrm{d}x\right], \quad b = \frac{\sum_{i=0}^{N_b-1} I_i}{N_b}, \tag{2.109}$$

for a MCMC estimator, the *expected* amount of samples of a bin i equals $N\frac{I_i}{\sum_{k=0}^{N_b-1} I_k}$, then the final reconstructed image would be

$$\underbrace{\frac{\sum_{i=0}^{N_b-1} I_i}{N N_b}}_{b/N}\left[N\frac{I_0}{\sum_{i=0}^{N_b-1} I_i}, ..., N\frac{I_{N_b-1}}{\sum_{i=0}^{N_b-1} I_i}\right] = \left[\frac{I_0}{N_b}, ..., \frac{I_{N_b-1}}{N_b}\right] = \frac{I}{N_b} \tag{2.110}$$

which is missing a factor N_b to appropriately reconstruct the image, i.e., to cancel out the N_b in the denominator. This is the reason that renderers multiply the final value by N_b/N, where N_b is usually denoted as A_{image}.

2.3 Monte Carlo integration in computer graphics

An integrator solves the measurement equation 2.35 using a Monte Carlo estimator sampling points in the lens with probability $p_{A_{\text{lens}}}$ and in the film with probability $p_{A_{\text{film}}}$. If we assume that we are working with a projective camera with importance function W_e^j defined as in Equation 2.32 we have:

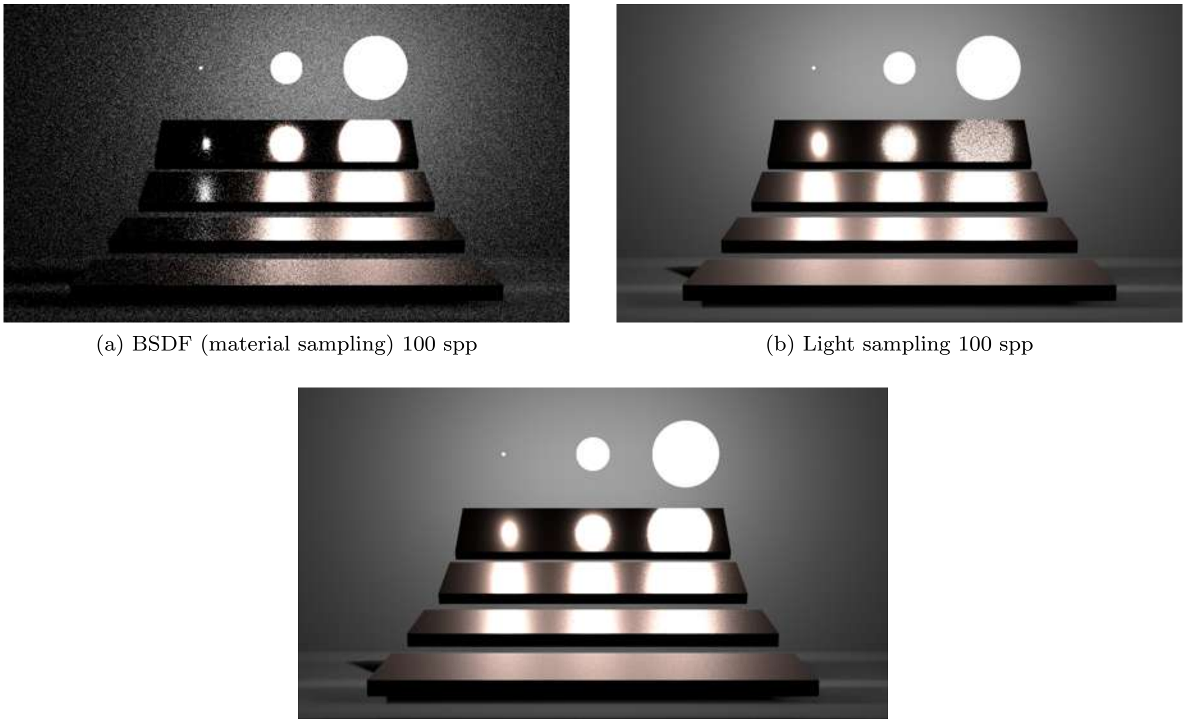
$$I_j \approx \frac{1}{N}\sum_{k=1}^{N} \frac{W_e^j(\mathbf{x}_{0_k}, \boldsymbol{\omega}_{i_{0_k}}) L_{i_0}(\mathbf{x}_{0_k}, \boldsymbol{\omega}_{i_{0_k}}) G(\mathbf{x}_{0_k} \leftrightarrow \mathbf{x}_{\text{film}_k})}{p_{A_{\text{lens}}} p_{A_{\text{film}}}} = \frac{1}{N}\sum_{k=1}^{N} \frac{h_j(\mathbf{x}_{\text{film}_k}) L_{i_0}(\mathbf{x}_{0_k}, \boldsymbol{\omega}_{i_{0_k}})}{b_f p_{A_{\text{film}}}}, \tag{2.111}$$

where usually the filter is normalized with respect of raster space coordinates b_r, instead of film space coordinates b_f, which are related by $\frac{h_j}{b_r} = \frac{h_j}{b_f}\frac{A_{\text{film}}}{A_{\text{image}}} = \frac{h_j}{b_f p_{A_{\text{film}}}}\frac{1}{A_{\text{image}}}$, and for $N = A_{\text{image}}\text{spp}$ we have then that if we evaluate the filter function normalized with respect of the raster coordinates the MC estimator is

$$I_j \approx \frac{1}{\text{spp}}\sum_{k=1}^{N} \frac{h_j(\mathbf{x}_{\text{film}_k})}{b_r} L_{i_0}(\mathbf{x}_{0_k}, \boldsymbol{\omega}_{i_{0_k}}), \tag{2.112}$$

or its slightly biased but with less variance versions:

$$\begin{aligned}
&\text{(Uniform film sampling)} \quad I_j \approx \frac{\frac{1}{N}\sum_{i=1}^{N} \frac{h_j(\mathbf{x}_i) L_o(\mathbf{x}_i, \boldsymbol{\omega}_i)}{A_{\text{lens}} p_{A_{\text{lens}}} p_{A_{\text{film}}}}}{\frac{1}{N}\sum_{i=1}^{N} \frac{h_j(\mathbf{x}_i)}{p_{A_{\text{film}}}}} = \frac{\sum_{i=1}^{N} h_j(\mathbf{x}_i) L_o(\mathbf{x}_i, \boldsymbol{\omega}_i)}{\sum_{i=1}^{N} h_j(\mathbf{x}_i)} \\
&\text{(Pixel Filter Importance Sampling)} \quad I_j \approx \frac{\sum_{i=1}^{N} \frac{h_j(\mathbf{x}_i) L_o(\mathbf{x}_i, \boldsymbol{\omega}_i)}{p_{h_j}(\mathbf{x}_i)}}{\sum_{i=1}^{N} \frac{h_j(\mathbf{x}_i)}{p_{h_j}(\mathbf{x}_i)}}.
\end{aligned} \tag{2.113}$$



(a) BSDF (material sampling) 100 spp

(b) Light sampling 100 spp

(c) MIS sampling 100 spp

Figure 2.7: Veach-MIS scene [Vea97; Bit16]. From top to bottom the roughness (amount of non-specularity) of the tables is increasing. Almost specular materials (top table) have problems when doing only light sampling as they scatter light in less directions. Material sampling has more problems when the light sources are small as they are less likely to get hit. MIS sampling combines the best of both techniques.

We will start our discussion of MC applied to rendering by presenting the application of MIS to direct illumination, then we will follow with more complex techniques.

The direct illumination integral of the rendering equation is defined by limiting the number of bounces in the scene to just one:

$$L_D(\mathbf{x},\boldsymbol{\omega}_o) = \int_\Omega \rho(\mathbf{x},\boldsymbol{\omega}_o,\boldsymbol{\omega}_i) L_e(t(\mathbf{x},\boldsymbol{\omega}_i),-\boldsymbol{\omega}_i)|\mathbf{n}\cdot\boldsymbol{\omega}_i|\,\mathrm{d}\boldsymbol{\omega}_i, \tag{2.114}$$

notice that the variable $\boldsymbol{\omega}_i$ could be sampled in two ways, with respect of the BSDF material function ρ or with respect of the emittance (light) function L_e, having two available sampling techniques. We will apply then the MIS MC estimator of one sample for both techniques $N_L = N_\rho = 1, \mathbf{N} = (1,1)$ which is:

$$I^{(\mathbf{N})} = w_L(\boldsymbol{\omega}_L)\frac{\rho(\mathbf{x},\boldsymbol{\omega}_o,\boldsymbol{\omega}_L)L_e(t(\mathbf{x},\boldsymbol{\omega}_L),-\boldsymbol{\omega}_L)|\mathbf{n}\cdot\boldsymbol{\omega}_L|}{p_L^\omega(\boldsymbol{\omega}_L)} + w_\rho(\boldsymbol{\omega}_\rho)\frac{\rho(\mathbf{x},\boldsymbol{\omega}_o,\boldsymbol{\omega}_\rho)L_e(t(\mathbf{x},\boldsymbol{\omega}_\rho),-\boldsymbol{\omega}_\rho)|\mathbf{n}\cdot\boldsymbol{\omega}_\rho|}{p_\rho^\omega(\boldsymbol{\omega}_\rho)}, \tag{2.115}$$

see Figures 2.7 and 2.8 for more details.

If we want N Direct Illumination samples per pixel, we simply randomly choose N lens points and N raster position as samples for the MC estimator of the measurement equation 2.111, and for each initial pair $(\mathbf{x}_{0_k},\boldsymbol{\omega}_{i_{0_k}})$ we apply Equation 2.115 to estimate $L_{i_0}(\mathbf{x}_{0_k},\boldsymbol{\omega}_{i_{0_k}})$.

As a side note, the first integrator implemented by a beginner is a Direct Illumination estimator, with only material sampling, a camera with only just one point (called a *pinhole camera*), and with samples obtained just in the center of the pixel. The usual next step for this is to implement light sampling and MIS sampling. After this, pixel sampling (not only the center) should be implemented and also lens sampling (not just a camera point).

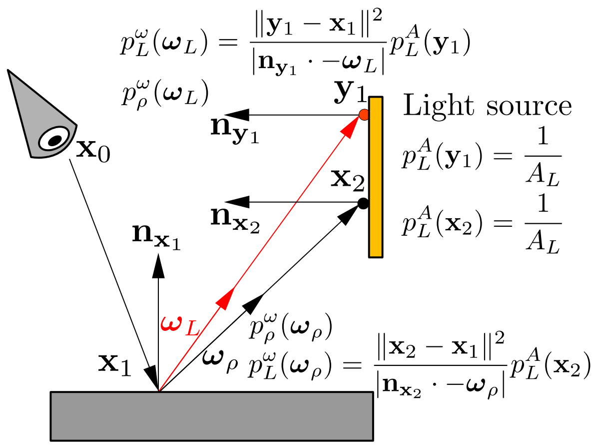


Figure 2.8: MIS of one material sample and one light sample at point $\mathbf{x}_1$ requires the calculation of 4 probability densities. Light point $\mathbf{y}_1$ was obtained sampling a point with an area pdf $p_L^A(\mathbf{y}_1)$, and it's related to direction $\boldsymbol{\omega}_L$ by $t(\mathbf{x}_1, \boldsymbol{\omega}_L) = \mathbf{y}_1$. The area pdf $p_L^A(\mathbf{y}_1)$ must be transformed to the solid angle measure $p_L^\omega(\boldsymbol{\omega}_L)$. Direction $\boldsymbol{\omega}_\rho$ is sampled with the material solid angle pdf $p_\rho^\omega(\boldsymbol{\omega}_\rho)$. To apply MIS we have to consider the "real" pdf values $p_L^\omega(\boldsymbol{\omega}_L), p_\rho^\omega(\boldsymbol{\omega}_\rho)$, and the "hypothetical" pdf values $p_L^\omega(\boldsymbol{\omega}_\rho), p_\rho^\omega(\boldsymbol{\omega}_L)$. Notice that because we are integrating over the solid angle domain all the probability measures must be solid angle pdfs p^ω

2.3.1 Path tracing

The straightforward method of solving the rendering equation consists in stochastically choosing two initial points, a lens point $\mathbf{x}_0$ and a raster position $\mathbf{x}_r$, which define an initial direction $\boldsymbol{\omega}_{i_0} = \frac{\mathbf{x}_r - \mathbf{x}_0}{\|\mathbf{x}_r - \mathbf{x}_0\|}$, and then apply several nested MC estimators (one for each recursion call), that is

$$L_{o_1}(\mathbf{x}_1, \boldsymbol{\omega}_{o_1}) \approx L_e(\mathbf{x}_1, \boldsymbol{\omega}_{o_1}) + \frac{1}{N} \sum_{k=1}^{N} \rho^\perp(\mathbf{x}_1, \boldsymbol{\omega}_{o_1}, \boldsymbol{\omega}_{i_{1_k}}) \left[L_e(\mathbf{x}_{2_k}, \boldsymbol{\omega}_{o_{2_k}}) + \frac{1}{N} \sum_{l=1}^{N} \rho^\perp(\mathbf{x}_{2_k}, \boldsymbol{\omega}_{o_{2_k}}, \boldsymbol{\omega}_{i_{2_l}}) L_{i_2} \right], \tag{2.116}$$

however, this is extremely inefficient as each nested integrator is being invoked several times, yielding an exponential increase in the number of samples with respect to the depth of the recursion. The path tracing algorithm fixes this by considering all the MC estimators to be of just one sample $N = 1$, effectively increasing the exploration efficiency in the image space and focusing just in a concrete chain of vertices (path) for a single pair of lens-raster samples, instead of using a complex network of multiple branching paths.

The *path tracing* algorithm applies the ray casting function $t(\mathbf{x}_k, \boldsymbol{\omega}_{i_k}) = \mathbf{x}_{k+1}$ beginning from the pair $(\mathbf{x}_0, \boldsymbol{\omega}_{i_0})$, where each incoming direction $\boldsymbol{\omega}_{i_k}$ is obtained from a sampling process that depends of the outgoing direction $\boldsymbol{\omega}_{o_k}$ of the current vertex $\mathbf{x}_k$, that is, $(\boldsymbol{\omega}_{i_k} | \boldsymbol{\omega}_{o_k}) \sim p_\rho(\boldsymbol{\omega}_{i_k} | \boldsymbol{\omega}_{o_k})$, then the following points $\mathbf{x}_1, ..., \mathbf{x}_l \in \mathcal{S} \cup \{\emptyset\}$ are obtained in the iterative process:

$$\mathbf{x}_l = \overbrace{t(\ldots t(t(t(}^{l-1 \text{ times}} \mathbf{x}_0, \boldsymbol{\omega}_{i_0}), \boldsymbol{\omega}_{i_1}), \boldsymbol{\omega}_{i_2}), ..., \boldsymbol{\omega}_{i_{l-1}}) \quad p(\boldsymbol{\omega}_{i_{l-1}} | \boldsymbol{\omega}_{o_{l-1}}, \boldsymbol{\omega}_{o_{l-2}}, ..., \boldsymbol{\omega}_{o_1}) \tag{2.117}$$

process that is done until either the light escapes, the path is terminated by Russian roulette[14], or a maximum depth has been reached. From this process each pair of initial samples of lens and film points produce a chain of points known as the *camera path* $\mathbf{X}$:

$$\mathbf{X} = (\mathbf{x}_0, \mathbf{x}_1, ..., \mathbf{x}_k); \mathcal{E}(\mathbf{X}) = (\mathbf{x}_1 - \mathbf{x}_0, \mathbf{x}_2 - \mathbf{x}_1, ..., \mathbf{x}_{k-1} - \mathbf{x}_k); \hat{\mathcal{E}}(\mathbf{X}) = \left(\frac{\mathbf{x_1} - \mathbf{x_0}}{\|\mathbf{x}_1 - \mathbf{x}_0\|}, ..., \frac{\mathbf{x}_{k-1} - \mathbf{x_k}}{\|\mathbf{x}_{k-1} - \mathbf{x}_k\|} \right) \tag{2.118}$$

where $\mathcal{E}(\mathbf{X})$ is the sequence of edge vectors, that when normalized corresponds to the sequence of incoming directions at each vertex $\hat{\mathcal{E}}(\mathbf{X}) = (\boldsymbol{\omega}_{i_0}, ..., \boldsymbol{\omega}_{i_{l-1}})$.

[14] Stochastic process that ends the path according to a probability density p_R, which is related to the length and energy of the path, we don't want to spend too much time on paths with a low radiance.

The radiance carried by some path $\mathbf{X}$ can be estimated as multiple one sample MC estimators for each level of each incoming radiance L_{i_k} and using the definitions given in Equation 2.30 we get:

$$\begin{gathered} L_{o_1}(\mathbf{x}_1, \boldsymbol{\omega}) = L_e^1 + f_s^1 \left[L_e^2 + f_s^2 \left[L_e^3 + f_s^3 \left[L_e^4 + f_s^4 L_i^4 \right] \right] \right] = L_e^1 + f_s^1 L_e^2 + f_s^1 f_s^2 L_e^3 + f_s^1 f_s^2 f_s^3 L_e^4 + ... \\ L_i^k = L_e^{k+1} + f_s^{k+1} L_i^{k+1}; \qquad f_s^k \equiv f^k / p_{\rho_k}; \qquad p_{\rho_k} \equiv p_\rho(\boldsymbol{\omega}_{i_k} | \boldsymbol{\omega}_{o_k}, ..., \boldsymbol{\omega}_{o_1}) \end{gathered} \tag{2.119}$$

this way of estimating a path's radiance is known as *implicit path tracing*. As it was already said, a geometric surface is an emitter if it emits radiance L_e in at least one direction. A common but not necessary assumption is that surfaces that are emiters can't reflect light, that is, $\rho(\mathbf{x}, \boldsymbol{\omega}_o, \boldsymbol{\omega}_i) = 0$ if $\mathbf{x}$ belongs to an emitter. Under this assumption only the paths that reached a light have a radiance different than black $(0, 0, 0)$, and the implicit path tracer would reduce to:

$$L_o(\mathbf{x}_1, \boldsymbol{\omega}) = f_s^1 f_s^2 f_s^3 ... f_s^{k-1} L_e^k, \tag{2.120}$$

for a path $\mathbf{X} = (\mathbf{x}_0, .., \mathbf{x}_k)$, the quantity $\beta^{k-1} = f_s^1 f_s^2 f_s^3 ... f_s^{k-1}$ is called the light path's *throughput*.

The effectiveness of the estimation of the radiance for the path $\mathbf{X}$ can drastically increase if direct illumination at each vertex is estimated sampling points in the light (see Figure 2.9) obtaining in this way a non black radiance with a higher chance. To do this each vertex $\mathbf{x}_k$ incoming radiance (direct illumination) is estimated using MIS with one sample of material ρ, ending in point $\mathbf{x}'_{k+1}$ and one sample of light L_e, ending in the light point $\mathbf{y}_k$:

$$L_D^k = w_L(\boldsymbol{\omega}_{L_k} | \boldsymbol{\omega}_{o_k}) \frac{\rho^\perp(\mathbf{x}_{k-1} \to \mathbf{x}_k \to \mathbf{y}_k)}{p_L(\boldsymbol{\omega}_{L_k} | \boldsymbol{\omega}_{o_k})} + w_\rho(\boldsymbol{\omega}_{\rho_k} | \boldsymbol{\omega}_{o_k}) \frac{\rho^\perp(\mathbf{x}_{k-1} \to \mathbf{x}_k \to \mathbf{x}'_{k+1})}{p_\rho(\boldsymbol{\omega}_{\rho_k} | \boldsymbol{\omega}_{o_k})}, \tag{2.121}$$

so the path itself $\mathbf{X}$ requires light points $(\mathbf{y}_1, ...)$ and material points $(\mathbf{x}'_1, ...)$, a path tracer could use the points in the main path $\mathbf{X}$ as MIS bsdf sample points, such that $(\mathbf{x}'_2 = \mathbf{x}_2, \mathbf{x}'_3 = \mathbf{x}_3, ...)$ (as Mitsuba does) or could use and independent point from the main path (as pbrt does). Then the complete estimator with MIS would be:

$$\begin{aligned} L_o(\mathbf{x}_0, \boldsymbol{\omega}) &= L_e^1 + f_s^1 L_e^2 + f_s^1 f_s^2 L_e^3 + f_s^1 f_s^2 f_s^3 L_e^4 + f_s^1 f_s^2 f_s^3 f_s^4 L_e^5 + ... \\ &= L_e^1 + L_D^1 + f_s^1 L_D^2 + f_s^1 f_s^2 L_D^3 + f_s^1 f_s^2 f_s^3 L_D^4 + ... \end{aligned} \tag{2.122}$$

estimating the direct illumination L_D of each vertex of the path to estimate the radiance in a more accurate way is called *explicit path tracing* or *next even estimation (NEE)*. As a side note notice that if a point in an area light is being sampled, the probability is given in the area measure p_L^A which must be transformed to the solid angle measure p_L^ω. See algorithm 3 for a complete overview of both implicit and explicit path tracing.

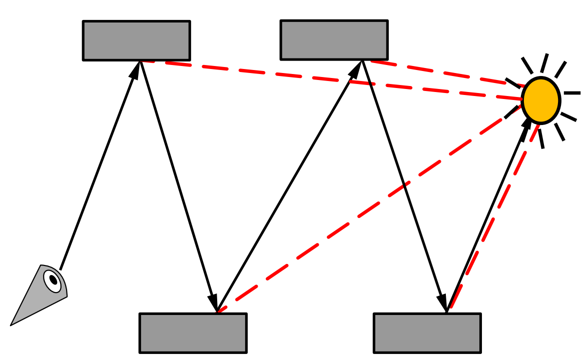


Figure 2.9: A path tracer with next event estimation, the incoming radiance at each vertex is calculated by both a material sample (black rays) and a light sample (red dotted lines). The material rays will also be used to keep bouncing in the scene. For each vertex two rays are being used, but we could also use three rays, two for the MIS sampling and the third one to keep bouncing in the scene, i.e. not reusing one of the rays.

Algorithm 3 Path tracing.

Input: Ray `r` with origin at some lens point $\mathtt{r}_{\mathbf{o}} = \mathbf{x}_0$ and initial direction $\mathtt{r}_{\boldsymbol{\omega}} = \boldsymbol{\omega}_{i_o}$, `maxDepth`, `useNEE`.

Output: Spectral radiance $L_{i_0}(\mathbf{x}_0, \boldsymbol{\omega}_{i_o})$

$L_i \leftarrow (0,0,0)$, $\beta \leftarrow (1,1,1)$ ▷ Radiance and throughput
`bounces` $\leftarrow 0$; `isSpecular` $\leftarrow$ `false`
while `true` **do**
  `foundIntersection` $\leftarrow$ `Intersect(r,isect)`; ▷ `isect` stores all the data of $t(\mathtt{r}_{\mathbf{x}}, \mathtt{r}_{\boldsymbol{\omega}})$
  if $\neg$ `foundIntersection` **then**
    if $\neg$ `useNEE` $\vee$ `isSpecular` $\vee$ `bounces==0` **then**
      $L_i \leftarrow L_i +$ `InfiniteLight(r)`; ▷ Radiance from infinite light sources if present.
    end if
    `break`
  end if
  if $\neg$ `useNEE` $\vee$ `isSpecular` $\vee$ `bounces==0` **then**
    $L_i \leftarrow L_i + \beta *$ `isect.Le(`$-r_{\boldsymbol{\omega}}$`)` ▷ `Le` returns the emitted radiance by the surface
  end if
  if `bounces` $\geq$ `maxDepth` **then**
    `break`
  end if
  if `NEE` $\wedge \neg$ `isSpecular` **then**
    $L_i \leftarrow L_i + \beta *$ `EstimateDirectMIS(isect)` ▷ Estimate direct illumination using MIS 2.121.
  end if
  $(f_s, \boldsymbol{\omega}_i) \leftarrow$ `isect.bsdf.Samplef()` ▷ f_s as in 2.30, $\boldsymbol{\omega}_i$ sampled direction.
  $\beta \leftarrow \beta * f_s$; `isSpecular` $\leftarrow$ `CheckSpecular(isect)`
  `r` $\leftarrow$ `isect.spawnRay(`$\boldsymbol{\omega}_i$`)`
  `bounces` $\leftarrow$ `bounces` $+ 1$
end while
return L_i

2.3.2 Bidirectional path tracing

Implicit path tracing explores the scene in a very inefficient way as we are effectively expecting to reach emitting vertices by mere chance, thing that is extremely difficult in a lot of scenes, as paths end either when the rays escape the scene, lose energy, are terminated by Russian roulette, or if a maximum depth (if it exists) is reached. Explicit path tracing improves this by using the information of the lights in the scene to try to estimate with MIS the incident “direct illumination” at each vertex by making explicit connections with light vertices, which in scenes with difficult light transport is not enough as these explicit connections can’t be made either because of visibility issues or due that the material does not allow that direction.

One way to improve the exploration of the space of light carrying paths is done by deterministically connecting two different kind of paths: a *camera path* $\mathbf{Z} = (\mathbf{z}_0, \mathbf{z}_1, ...)$ which is generated starting from a point $\mathbf{z}_0$ at the camera’s lens, and a *light path* $\mathbf{Y} = (\mathbf{y}_0, \mathbf{y}_1, ...)$ which is generated starting from an emitter point $\mathbf{y}_0$. Using both the light and the camera path several light paths $\mathbf{X}_{s,t}$ of length $k = s + t - 1$ can be generated:

$$\begin{aligned}\mathbf{X}_{s,t} &= \{\mathbf{y}_0, \mathbf{y}_1, ..., \mathbf{y}_{s-2}, \mathbf{y}_{s-1}, \mathbf{z}_{t-1}, \mathbf{z}_{t-2}, \mathbf{z}_{t-3}, ..., \mathbf{z}_1, \mathbf{z}_0\} \\ &\equiv \{\mathbf{x}_0 = \mathbf{y}_0, \mathbf{x}_1, ..., \mathbf{x}_{s+t-2}, \mathbf{x}_{s+t-1} = \mathbf{z}_0\},\end{aligned} \tag{2.123}$$

where $\mathbf{X}_{s,t}$ is obtained by deterministically connecting the vertex s of the light path $\mathbf{Y}$ with vertex t of the camera path $\mathbf{X}$, process known as an (s,t) *technique*. Each of these paths can be used in an MC estimator [VG95a] by noting that the measurement equation 2.35 can be recursively expanded into

$$\begin{aligned}I_j &= \sum_{k=1}^{\infty} \int_{\mathcal{P}_{k+1}} L_e(\mathbf{x}_0 \to \mathbf{x}_1) G(\mathbf{x}_0 \leftrightarrow \mathbf{x}_1) \left(\prod_{i=1}^{k-1} \rho(\mathbf{x}_i \to \mathbf{x}_{i+1}) G(\mathbf{x}_i \leftrightarrow \mathbf{x}_{i+1}) \right) \\ &\qquad \times W_e^j(\mathbf{x}_{k-1} \to \mathbf{x}_k) \, \mathrm{d}A(\mathbf{x}_0) \ldots \mathrm{d}A(\mathbf{x}_k) \\ &= \int_{\mathcal{P}_2} L_e(\mathbf{x}_0 \to \mathbf{x}_1) G(\mathbf{x}_0 \leftrightarrow \mathbf{x}_1) W_e^j(\mathbf{x}_0 \to \mathbf{x}_1) \, \mathrm{d}A(\mathbf{x}_0) \, \mathrm{d}A(\mathbf{x}_1) \\ &+ \int_{\mathcal{P}_3} L_e(\mathbf{x}_0 \to \mathbf{x}_1) G(\mathbf{x}_0 \leftrightarrow \mathbf{x}_1) \rho(\mathbf{x}_0 \to \mathbf{x}_1 \to \mathbf{x}_2) G(\mathbf{x}_1 \leftrightarrow \mathbf{x}_2) W_e^j(\mathbf{x}_1 \to \mathbf{x}_2) \, \mathrm{d}A(\mathbf{x}_0) \, \mathrm{d}A(\mathbf{x}_1) \, \mathrm{d}A(\mathbf{x}_2) \\ &\;\vdots\end{aligned} \tag{2.124}$$

where each $\mathcal{P}_k$ correspond to the space of paths with k vertices, or equivalently, of length $k-1$, which means that the integral over all the possible lengths can be divided in integrals of “fixed” path length. The above integral can be abbreviated as

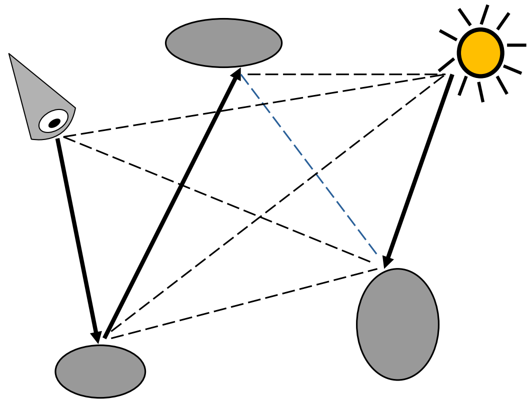
$$\begin{gathered}I_j = \int_{\mathcal{P}} f_j(\mathbf{X}) \, \mathrm{d}\mu_{A_p}(\mathbf{X}), \quad \mathrm{d}\mu_{A_p}(\mathbf{x}_0, ..., \mathbf{x}_k) = \mathrm{d}A(\mathbf{x}_0) ... \, \mathrm{d}A(\mathbf{x}_k); \quad \mu_{A_p}(D) = \int_D \mathrm{d}A(\mathbf{x}_0) ... \, \mathrm{d}A(\mathbf{x}_k), \\ f_j(\mathbf{X}) = L_e(\mathbf{x}_0 \to \mathbf{x}_1) G(\mathbf{x}_0 \leftrightarrow \mathbf{x}_1) W_e^j(\mathbf{x}_{k-1} \to \mathbf{x}_k) \left(\prod_{i=1}^{k-1} \rho(\mathbf{x}_i \to \mathbf{x}_{i+1}) G(\mathbf{x}_i \leftrightarrow \mathbf{x}_{i+1}) \right) \equiv h_j(\mathbf{X}) f(\mathbf{X}).\end{gathered} \tag{2.125}$$

with $\mathcal{P} = \bigcup_{k \geq 2} \mathcal{P}_k$ the set of all possible paths $\mathbf{X}$ in the scene of all possible lengths, and h_j the filter function of pixel j.

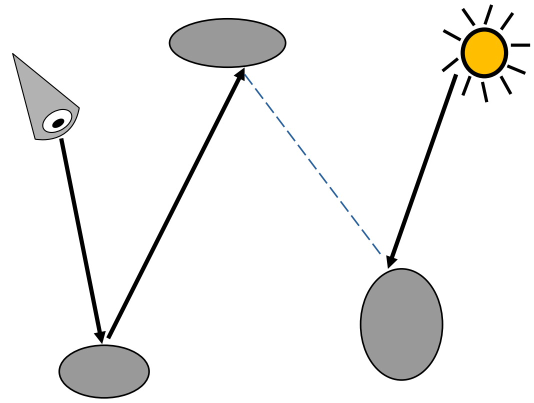
Then as Equation 2.124 is composed by the sum of several integrals for paths with several lengths, the MC estimator of one sample can be defined as the sum of several MC estimators of one sample one for each integral with different path length:

$$I_j \approx \sum_{l \geq 0} \frac{f_j(\mathbf{X}_l)}{p^{A_p}(\mathbf{X}_l)}, \tag{2.126}$$

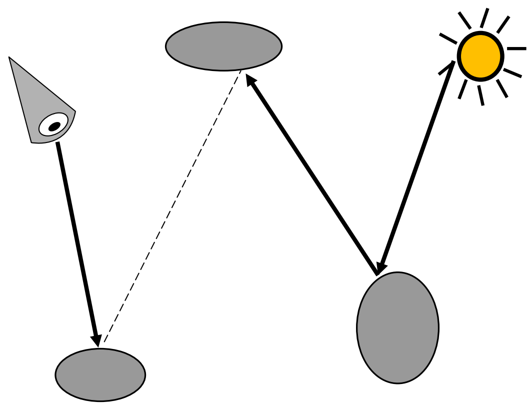
where $l = k$ corresponds to paths with length $k+1$, usually called the *depth of the path* D_k, the depth D_0 would only return a positive radiance for emitters directly seen by the camera. Notice something interesting, an implicit path tracer is equivalent to considering the techniques $(s,t) \in \{(0,t) | t \geq 2\}$ and an explicit path tracer is almost (it uses MIS between implicit $s = 0$ and explicit $s = 1$ paths) equivalent to considering the techniques $(s,t) \in \{(1,t) | t \geq 2\} \cup \{(0,2)\}$.



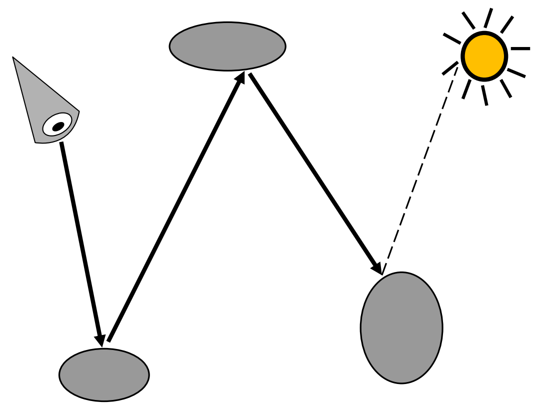
(a) All the camera vertices are connected to all the light vertices, in this example there are 3 camera vertices and 2 light vertices



(b) Isolated technique $t = 3, s = 2$ from 2.10a



(c) The isolated technique $t = 3, s = 2$ imagining as if it was generated using $t = 2, s = 3$. Even if this was not the case.



(d) The isolated technique $t = 3, s = 2$ imagining as if it was generated using $t = 4, s = 1$. Even if this was not the case.

Figure 2.10: BDPT connects all the camera vertices with all the light vertices 2.10a. Note that MIS is applied by supposing that all the deterministically connected paths of BDPT were generated using a different s, t technique. To apply MIS we need to calculate forward and backward probabilities.

A path with the same length could have been generated in multiple ways, for example, for all the paths of length k (D_{k-1}) it follows that $s + t - 1 = k$, then all the elements of the set of tuple pairs

$$\mathcal{T}_k = \{(k+1, 0), (k, 1), (k-1, 2), ..., (1, k), (0, k+1)\} \tag{2.127}$$

would produce a path with the same length, thus using a different tuple pair (technique) a MIS estimator of one sample per technique for some given depth D_k would be:

$$(I_j)_{D_k} \approx \sum_{(s,t)\in\mathcal{T}_{k+1}} w_{s,t}(\mathbf{X}_{s,t}) \frac{f_j(\mathbf{X}_{s,t})}{p_{s,t}^{A_P}(\mathbf{X}_{s,t})}, \tag{2.128}$$

or considering all the possible lengths the full measurement for one sample is then:

$$I_j = \sum_{k\geq 0} (I_j)_{D_k} \approx \sum_{s\geq 0}\sum_{t\geq 0} w_{s,t}(\mathbf{X}_{s,t}) \frac{f_j(\mathbf{X}_{s,t})}{p_{s,t}^{A_P}(\mathbf{X}_{s,t})} = \sum_{s\geq 0}\sum_{t\geq 0} C_{s,t} = \sum_{s\geq 0}\sum_{t\geq 0} w_{s,t} C^*_{s,t}, \tag{2.129}$$

where the probability in product area measure $p_{s,t}^{A_p}(\mathbf{X}_{s,t})$ is just the multiplication of probabilities:

$$p_{s,t}^{A_p}(\mathbf{X}_{s,t}) = \left(\prod_{i=0}^{s-1} P^A(\mathbf{y}_i)\right)\left(\prod_{i=0}^{t-1} P^A(\mathbf{z}_i)\right). \tag{2.130}$$

If the path has a fixed path length k, then it is enough to only choose one of the two indices, as the other is unequivocally determined by the other choice, that is, $s = s(t), t = t(s)$, in this case the path $\mathbf{X}_{s,t}$ can be denoted as just $\mathbf{X}_s = \mathbf{X}_{s,k+1-s}$, notation that is sometimes used, alongside with $w_s = w_{s,k+1-s}, p_s = p_{s,k+1-s}$, etc. The MIS weight of a given path $\mathbf{X}_{s,t}$, with path length $k = s+t-1$ is then

$$w_s(\mathbf{X}_{s,t}) = \frac{p_s(\mathbf{X}_{s,t})}{\sum_{i=0}^{s+t} p_i(\mathbf{X}_{s,t})} = \left(\sum_{i=0}^{s-1} \frac{p_i(\mathbf{X}_{s,t})}{p_s(\mathbf{X}_{s,t})} + 1 + \sum_{i=s+1}^{s+t} \frac{p_i(\mathbf{X}_{s,t})}{p_s(\mathbf{X}_{s,t})} \right)^{-1}, \tag{2.131}$$

Because the previous algorithm deterministically connects vertices in a forward traced camera path and a backwards traced light path, using opposite tracing directions, this algorithm is called *bidirectional path tracing* (bdpt or BPT) [VG95a; LW98] and can be summarized as the algorithm below

Algorithm 4 Bidirectional Path tracing.

Input: $\mathbf{r}_0, \mathbf{z}_0$ ▷ Initial raster position and lens position.
Output: BDPT estimation of one sample 2.129.

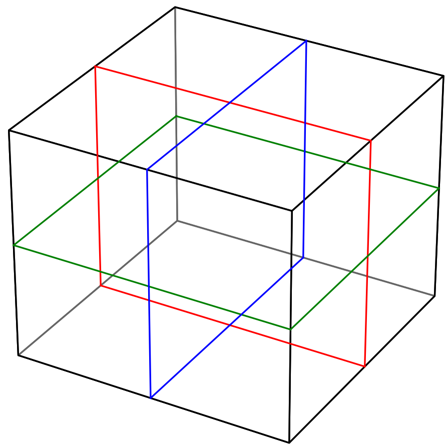
(`nCamera`, $\mathbf{Z}$) ← `TraceCameraPath(`$\mathbf{z}_0, \mathbf{r}_0$`)`
(`nLight`, $\mathbf{Y}$) ← `TraceLightPath()`
`Lpath` ← `(0,0,0)`
for `t`=0;`t`≤`nCamera`; `t++` **do**
  for `s`=0;`s`≤`nLight`; `s++` **do**
    (`Ltechnique`,`newRaster`) ← `ConnectBDPT(s,t,`$\mathbf{Y}, \mathbf{Z}$`)` ▷ Returns $C_{s,t}$ of Equation 2.129.
    if `t`≤ 1 **then**
      `film.AddSample(Ltechnique,newRaster)` ▷ Estimating the light image
    else
      `Lpath` ← `Lpath + Ltechnique`
    end if
  end for
end for
`film.AddSample(Lpath,pixelRaster)` ▷ Estimating the eye image

2.3.3 Global exploration

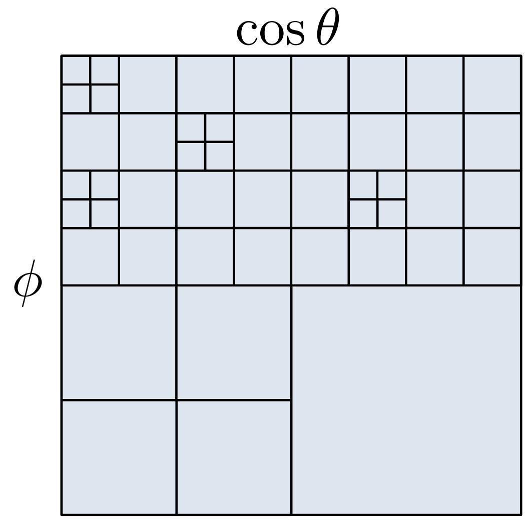
Ideally, we want to *sample paths* proportionally to the distribution of light carrying paths, but algorithms such as PT and BDPT explore the paths without knowing anything about this distribution, the only information that we have available before starting the rendering process is the location of the emitters in the scene, which is used by the NEE PT integrator by sampling points in the light, and by the BDPT integrator by sampling light paths. The problem with the previous methods is that they don't remember any of the paths that were previously sampled, they only care about storing the measurements of each pixel, which is a huge waste of information; imagine that by mere chance we had arrived to a very difficult path, we would like to try similar paths to this path that we should already know. We want to "learn from experience" and guide our sampling processes with our previous knowledge. The process of exploring proportionally to the distribution of the whole network of paths induced by the lights and objects in the scene is called global exploration, and this could be achieved by storing positional and directional information from the paths we have already sampled.

One possible way of storing this information is to use a binary tree, more precisely, a k-d tree, to divide the scene in subregions, where each depth of the three splits in half a different axis. The most refined divisions (the leaves) will then correspond to a sub-region $S \subset \mathcal{S}$ of the physical space of the scene where all the points $\mathbf{x} \in S$ will contribute directional information about the radiance of this zone (see Figure 2.11a). The radiance of each region will be then stored in a quadtree (see Figure 2.11b) located at the corresponding leaf in the binary tree, this quadtree will splat all the incoming radiance $L_i(\mathbf{x}, \boldsymbol{\omega}), \forall \mathbf{x} \in S, \boldsymbol{\omega} \in S^2$ on its nodes, where each node of the quadtree is a division of the directional space.

The three dimensional k-d tree that stores quadtrees on its leaves that will splat radiance according to appropriate divisions of the space of direction is called a *spatio-directional tree* and the algorithm



(a) The space of the scene is divided by a k-d tree, i.e., a binary tree in which each depth of the tree divides a specific axis in half, alternating between axes ($X \rightarrow Y \rightarrow Z \rightarrow X \rightarrow Y \rightarrow Z$....).



(b) Quadtree. Each level of the quadtree divides in four the space of directions. $-1 \leq \cos\theta \leq 1; 0 \leq \phi \leq 2\pi$.

Figure 2.11: A spatio-directional tree is composed of a binary tree that divides the scene, each of the leaves of the binary tree has a quadtree that divides the energy domain.

that uses this type of trees is called *path guiding* [MGN17]. In order to completely understand this algorithm three things must be explained: how to register the radiance information into the quadtrees; how to get samples from the quadtree; and how to determine the splittings and refinements of both the binary tree and the quadtrees.

The algorithm needs two different spatio-directional trees, a building spatio-directional tree B_{T_B} that is going to store the newly acquired information, and a sampling spatio-directional tree B_{T_S} that is going to be used to sample new directions. This could be implemented either by having two quadtrees on each leaf of a single k-d tree, and using the appropriate quadtree for building or sampling, or it could be done by having two independent k-d trees with just one quadtree per leaf.

The PG algorithm works in iterations where each iteration uses the double of samples than the previous iteration, that is, iteration 1 uses one sample, iteration 3 uses four samples, and in general, iteration k uses 2^{k-1} samples, so the total amount of computational effort measured in samples is $1+2+4+...+2^{it-1} = 2^{it}-1$, and where each iteration uses a different and increasingly better trained sampling tree. The doubling amount of samples per each iteration can be explained by considering a leaf node on the k-d tree that was divided, not being a leaf anymore, but which generated two new leaves, the volume encompassed by these two new leaves is half the volume that the old (now divided) leaf had, so we expect by doubling the amount of samples that the two new leaves get the same amount of samples than the parent ex-leaf had.

Each iteration will update a new building tree and will sample a different sampling tree:

$$
\begin{aligned}
B^1_{T_S} &= \emptyset & B^1_{T_B} &= B^1_{T_B} \\
B^2_{T_S} &= B^1_{T_B} & B^2_{T_B} &= \texttt{Clean}(\texttt{Refine}(B^1_{T_B})) \\
&\vdots \\
B^{it}_{T_S} &= B^{it-1}_{T_B} & B^{it}_{T_B} &= \texttt{Clean}(\texttt{Refine}(B^{it-1}_{T_B})),
\end{aligned}
\tag{2.132}
$$

where in the first iteration there is no information in the tree yet, so it has to be recollected using just the usual path tracing; `Refine` means that the topology of the spatio-directional tree, both the kd-tree and the quadtrees, must be appropriately changed and `Clean` means that the current tree information must be erased, so the new iteration can splat new information into the tree. The first building binary tree $B^1_{T_B}$ has a user defined refinement, for example, it can be started as a full empty binary tree with

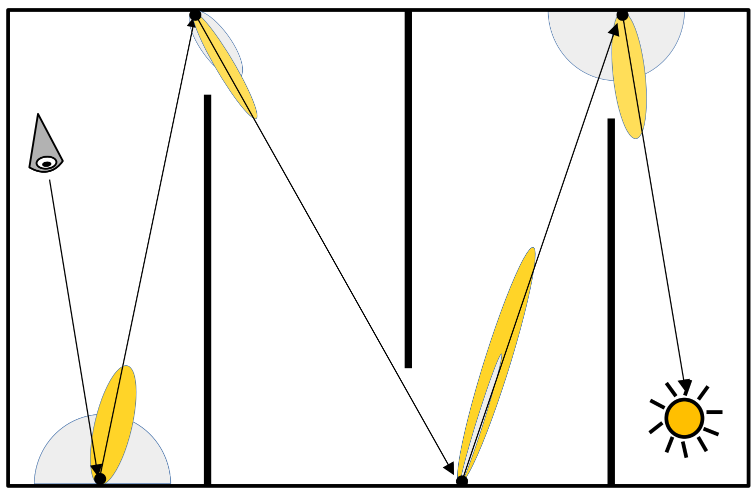


Figure 2.12: Path guiding uses the learnt information of the spatio-directional tree (yellow) combined with the BSDF distribution (gray). Note that the guiding could give a non allowed direction by the material.

depth five.

The incoming radiance $L_i(\mathbf{x},\boldsymbol{\omega})$ is going to be splatted by locating the leaf of the spatial tree that encloses the point using a function like $\mathtt{FindPosition}(\mathbf{x})$ which returns a quadtree, next we will find the appropriate leaf of the returned quadtree with a function of the type

$$\mathtt{FindDirection}(\mathtt{FindPosition}(\mathbf{x}),\boldsymbol{\omega}), \tag{2.133}$$

so the scalar value related to the radiance that is going to be splatted is $\frac{L_i^r(\mathbf{x},\boldsymbol{\omega})+L_i^g(\mathbf{x},\boldsymbol{\omega})+L_i^b(\mathbf{x},\boldsymbol{\omega})}{3}$, using a function like

$$\mathtt{Splat}(\mathtt{FindDirection}(\mathtt{FindPosition}(\mathbf{x}),\boldsymbol{\omega}),L_i(\mathbf{x},\boldsymbol{\omega})). \tag{2.134}$$

the number of splattings done on each binary tree leaf must be counted.

The incoming radiance is going to be splatted for all the vertices in the camera path $\mathbf{X}$ of a unidirectional path tracer without NEE[15], the full path throughput for a path with $q+1$ vertices in which $\mathbf{x}_q$ is an emitter vertex is:

$$\begin{aligned}
L_{i_0}(\mathbf{x}_0,\boldsymbol{\omega}_{i_0}) &= L_{o_1}(\mathbf{x}_1,\boldsymbol{\omega}_{o_1}) = f_s^1 f_s^2 f_s^3 ... f_s^{q-1} L_e^q \\
L_{i_1}(\mathbf{x}_1,\boldsymbol{\omega}_{i_1}) &= L_{o_2}(\mathbf{x}_2,\boldsymbol{\omega}_{o_2}) = f_s^2 f_s^3 ... f_s^{q-1} L_e^q \\
&\vdots \\
L_{i_{q-2}}(\mathbf{x}_{q-2},\boldsymbol{\omega}_{i_{q-2}}) &= L_{o_{q-1}}(\mathbf{x}_{q-1},\boldsymbol{\omega}_{o_{q-1}}) = f_s^{q-1} L_e^q \\
L_{i_{q-1}}(\mathbf{x}_{q-1},\boldsymbol{\omega}_{i_{q-1}}) &= L_{o_q}(\mathbf{x}_q,\boldsymbol{\omega}_{o_q}) = L_e^q,
\end{aligned} \tag{2.135}$$

so for each vertex $\mathbf{x}_k \in \mathbf{X}$, the incoming radiance $L_{i_k}(\mathbf{x}_k,\boldsymbol{\omega}_{i_k})$ is splatted. This is done by temporarily storing the throughput of each vertex $\mathrm{thr}(\mathbf{x}_k) = f_s^1 f_s^2 ... f_s^k$ and dividing the total throughput of the whole path by the partial throughput on each vertex:

$$L_{i_k}(\mathbf{x}_k,\boldsymbol{\omega}_{i_k}) = \frac{\mathrm{thr}(\mathbf{X})}{\mathrm{thr}(\mathbf{x}_k)} L_e^q = \frac{f_s^1 f_s^2 f_s^3 ... f_s^{q-1}}{f_s^1 f_s^2 f_s^3 ... f_s^k} L_e^q = f_s^{k+1} f_s^{k+2} ... f_s^{q-1} L_e^q, \tag{2.136}$$

these splats are just valid for the first iteration as there is only BSDF (ρ) sampling, but the iterations that follow use both the tree sampling and BSDF sampling.

Then for the next iterations the path tracing algorithm for some path $\mathbf{X}$, given a vertex $\mathbf{x}_k$, the vertex $\mathbf{x}_{k+1}$ is going to be sampled with some probability q_{Tr} from the spatio-temporal tree or with some probability $q_\rho = 1 - q_{\mathrm{Tr}}$ from the material function (see Figure 2.12), this decision can be put

[15]Path guiding can use NEE, but even without NEE the improvements with respect to path tracing are tremendous.

as a balance heuristic MIS MC estimator of the sampled direction $\boldsymbol{\omega}_{i_k}$ that is using only one sample, which was obtained either by material sampling p_ρ or by tree sampling p_{Tr}:

$$w_c(\boldsymbol{\omega}_{i_k})\frac{f(\mathbf{x}_k,\boldsymbol{\omega}_{o_k},\boldsymbol{\omega}_{i_k})}{q_c p_c(\boldsymbol{\omega}_{i_k})} = \frac{q_c p_c(\boldsymbol{\omega}_{i_k})}{q_c p_c(\boldsymbol{\omega}_{i_k}) + q_{\neg c} p_{\neg c}(\boldsymbol{\omega}_{i_k})}\frac{f(\mathbf{x}_k,\boldsymbol{\omega}_{o_k},\boldsymbol{\omega}_{i_k})}{q_c p_c(\boldsymbol{\omega}_{i_k})} = \frac{f(\mathbf{x}_k,\boldsymbol{\omega}_{o_k},\boldsymbol{\omega}_{i_k})}{q_c p_c(\boldsymbol{\omega}_{i_k}) + q_{\neg c} p_{\neg c}(\boldsymbol{\omega}_{i_k})}, \tag{2.137}$$

where p_c is the probability distribution of the chosen technique (tree sampling or material sampling), $p_{\neg c}$ is the pdf of the other not chosen technique. Notice something quite important, the MIS estimator appears to only use just one sample of one of the techniques, and we have previously said that at least one sample from each technique was necessary, but this is not a contradiction at all, as the deterministic use of two samples (one per each technique) is being replaced by the stochastic process of choosing one of the two techniques with probability $0 < q_c < 1$.

For the exact sampling and refinement algorithm of the SD trees see algorithm 11, 12 and 13 in appendix A. For the general overview of the full PG technique, see algorithm 5.

Algorithm 5 Path guiding.

Input: `camera, scene, sampler, film, maxIterations`
Output: Rendered image.

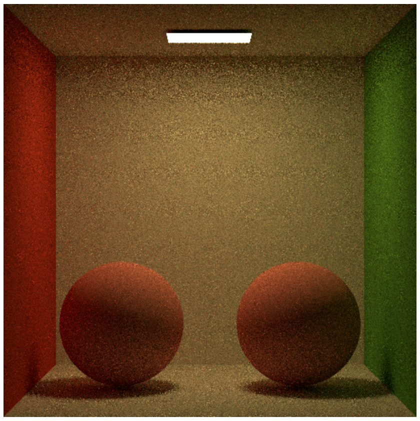
```
B_{T_S} ← ∅, B_{T_B} ← InitializeBinaryAndQuadTrees()      ▷ Sampling and building trees.
it ← 0
samplesPerIteration ← 1
while it<maxIterations do
    for each pixel in pixels do
        for i ← 1 to samplesPerIteration do
            ray ← GenerateCameraRay(pixel,sampler,rasterPosition)
            L_i ← L(ray,B_{T_S},B_{T_B})      ▷ Path tracer that uses material and tree guiding 2.137
            if it==maxIterations−1 then
                AddSample(film,L_i,filterWeight,cameraWeight,rasterPosition)
            end if
        end for
    end for
    B_{T_S} ← Copy(B_{T_B}); B_{T_B} ← Refine(B_{T_B}); B_{T_B} ← Clean(B_{T_B})
    it ← it+1
    samplesPerIteration ← samplesPerIteration×2
end while
```

One interesting artifact that can be observed in the first iterations of the path guiding algorithm is that the spatial division of the scene induced by the spatio-directional tree can be observed in the rendered image, see Figure 2.13.

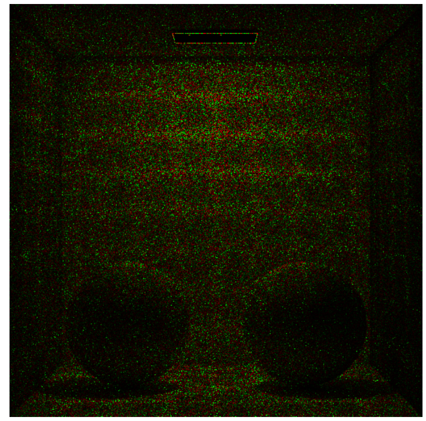
Path guiding is a highly competitive algorithm, as it can be verified in Figure 2.14 but it has difficulties with near specular surfaces, and it reduces to the usual path tracing for specular surfaces as it can be seen in Figure 2.15.

It is a good moment to discuss a common misunderstanding that might arise, when we say that for some scenes BDPT is more *efficient* than PT, what we actually mean is that it converges faster (for these scenes) to the correct solution. For example, a BDPT sample connects all the vertices of the light path with all the vertices of the camera path, being each sample quite computationally expensive when compared to a single sample of PT. The efficiency of BDPT arises from its capacity of sampling paths which are not properly explored by PT (e.g., indirect diffuse lighting). This increased exploration capacity for a lot of scenes improve the overall efficiency (over simpler techniques such as PT) despite of the main drawback of BDPT which is its increased correlation, which in turn, increases the variance of the estimator[16]. Regardless of this main drawback, for a lot of scenes, BDPT will converge faster to the solution than PT, i.e., it is more efficient, even if it appears to do more work. This apparent extra work in the end saves more time overall. In general, a more computationally expensive algorithm could approximate the solution faster than a less expensive algorithm by exploring (sampling) the integration domain more efficiently.

[16] Correlation can be mitigated by probabilistic connection methods, see Matrix Bidirectional Path Tracing [Cha+18] or Popov 2015 [Pop+15] at a cost of added complexity.



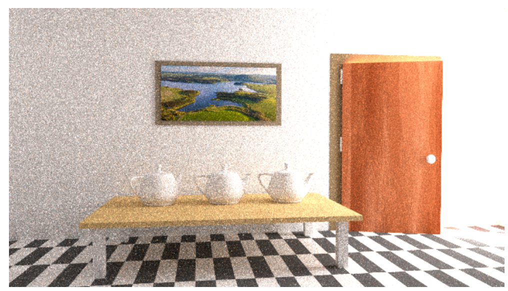
(a) Image rendered using path guiding,



(b) Red green image.

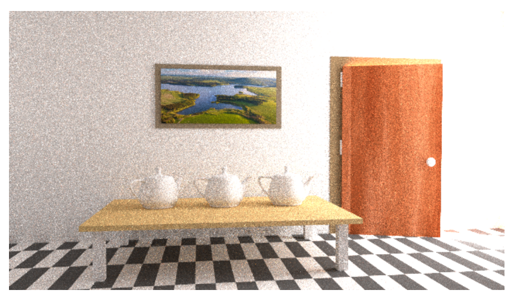
Figure 2.13: Notice that the spatial division induced by the spatio-directional tree can be seen on the first iterations of the algorithm as a grid.

(a) Implicit path tracing 1024spp. SMAPE=0.276738, rMSE=0.628789, t=330seconds



(b) Explicit path tracing (NEE) 1024spp. Errors, SMAPE=0.101068, rMSE=0.227485, time=222 seconds

(c) BDPT 128spp. Errors SMAPE=0.129869, rMSE=0.315636, time=253 seconds



(d) Path Guiding: 1024 total computational samples, only 512 splatted samples. Errors SMAPE=0.087597, rMSE=0.294157, time=317 seconds

Figure 2.14: Note how for this difficult scene path guiding without using NEE, competes with NEE. Also observe that BDPT is not that impressive on this scene as a lot of light vertices behind the wall can't be reconnected with the camera vertices, causing an even worse convergence that NEE PT!. This can be explained as NEE PT doesn't spend too much time sampling non connectable light vertices in the other side of the wall. In this case their speed of convergence to the solution (efficiency) from more time to less time (worse to better) is: path< BDPT < NEE path < PG. The slowness of BDPT is easily explained by the fact that most vertices from the light path remain on the other side of the wall.

(a) Implicit path tracing 1024spp. SMAPE=0.0653636, rMSE=0.078361, t=752seconds

(b) Explicit path tracing (NEE) 1024spp, SMAPE=0.0711859, rMSE=0.0796059, time=627 seconds

(c) BDPT 128spp. SMAPE=0.0457663, rMSE=0.145876, time=522 seconds

(d) Path Guiding: 1024 total computational samples, only 512 splatted samples. SMAPE=0.067522, rMSE=0.22794, time=512 seconds

Figure 2.15: Note how Path guiding struggles with the near specular vertices.

2.4 Chapter conclusions

In this chapter we presented all the mathematical and physical theory used in this thesis. We started by obtaining the rendering equation [Kaj86] through energy conservation, and then we obtained the measurement equation [PJH16; Vea97], a generalized version of the rendering equation that accounts for all the distinct lens positions of the camera and all the raster positions of the pixel returning a three channeled (spectral) value for the A_{image} pixels of the final render.

The measurement and the rendering equations are recursive, high dimensional, full of discontinuities integral equations that must be solved numerically using Monte Carlo (MC) integration techniques. We presented the possible integration domains of these integral equations, alongside the most crucial probability definitions, such as: probability measure, random variable, measure, pdf, cdf, expected value and variance. These definitions are useful because estimating all the A_{image} MC estimators of each measurement equation I_j, one for each pixel, require taking samples from the integration domain Ω proportional to some pdf function. The better we explore the integration domain by taking samples in an intelligent way, the lower the variance of the MC estimator will be, which it is equivalent to increasing the efficiency of the MC estimator, that is, a faster convergence.

We also discussed that there are several ways of reducing the variance but we focused on Importance Sampling (IS) which consists in taking samples proportional, at least to some extent, to the integrand. We further improved IS by combining different Importance Sampling strategies in a process know as Multiple Importance Function (MIS) [VG95b], giving us more flexibility to explore the sample space in different ways, and choosing the most contributing technique according to each region.

We also talked about a very special case of MC estimators called Markov Chain Monte Carlo (MCMC) estimators that take correlated samples in a proportional way to any positive function f, with its normalized version p_f acting as a stationary (target) probability density of the Markov Process of an ergodic Markov Chain. To be able to use the function f as target distribution of the Markov Process we used the Metropolis-Hastings (MH) algorithm [Met+53; Has70] that defines the transition probability density of the Markov Process as the multiplication of the densities of two separated random process, the transition probability density T of modifying the current sample into another times the acceptance probability density α that will tell us if we accept these proposed modifications. This extra stochastic acceptance process allow us to define a detailed balance over our already (by construction) ergodic Markov Chain so we can guarantee that the stationary probability is p_f, effectively being able to sample any positive function f.

We then presented all the previous theory applied to rendering. We started by a MC estimator that takes samples with IS of material as Path Tracing (PT). After this, we introduced an improved version of PT that uses MIS of light and material sampling on each vertex of the path of light traced, algorithm known as Next Event Estimation (NEE) Path Tracing. Then we showed that we could go even further by considering full paths starting from light vertices (light path) and connecting all the vertices of this path with all the vertices of a path that started in a lens point (camera path) exploring in a more efficient way the sample space, because this algorithm considers all the connections between two different type of paths that run on opposite direction, it is called Bidirectional Path Tracing (BDPT) [VG95a].

A problem with all the rendering techniques that were discussed in this chapter was that we are not taking advantage of the information used by the previously sampled paths, so we implemented in pbrt version 3.0 the global exploration algorithm called Path Guiding (PG) [MGN17], which stores positional and directional information about the radiance of the already explored paths, in a spatio-directional kd tree with directional quadtrees stored on its leaves.

We verified that for difficult scenes even algorithms like BDPT fall short, so we need to rely on methods that use the previously sampled light paths in some way, such as PG, and we could also observe that every scene has a different type of light transport that we must carefully consider before starting the algorithm, For example rendering the scene in Figure 2.14 with BDPT would be unwise as most of the light vertices of the light paths would be behind the door.

In the next chapter we will proceed to talk about MCMC estimators (local exploration) applied to rendering.

Chapter 3

Metropolis Light Transport

Metropolis light transport [VG97; Vea97] is a rendering technique that uses a Markov Chain Monte Carlo estimator that gets samples from several ergodic Markov Chains run in parallel. Each of these chains will generate a sequence of paths with positive radiance $(\mathbf{X}_i)_{i\geq 0}$. The target function of these chains will usually be given by a scalar transformation (e.g., the luminance, the sum, the maximum component, etc) of the path's radiance, promoting in this way an exploration proportional to the measurement equation of light transport. Each of these chains can visit all the raster positions in all the pixels of the image.

This algorithm was implemented applying both, large modifications (called mutations in path-space and large mutations in Primary Sample Space) and small changes (called perturbations in path-space and small mutations in Primary Sample Space) to a path $\mathbf{X}_{s,t}$ generated by a bidirectional path tracer:

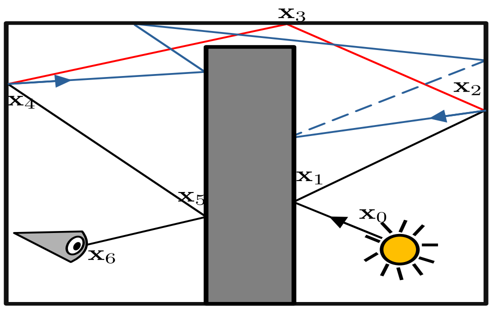
1. **Global exploration (mutations)**: This type of path modification to the path $\mathbf{X}_{s,t}$ returns a very different path, and it's called a *bidirectional mutation*. We are *globally* exploring the path-space in order to meet the ergodicity requirement. A special type of mutation is called *lens subpath mutation* which replace a lens subpath (abbreviated as the regular expression $(L|D)S^*E$ [Vea97]) with another completely different lens subpath, with the objective of getting a proposal with a high likelihood of being in a different pixel, effectively promoting the exploration of paths in the image space, improving in this way the stratification of the samples on the image space.

2. **Local exploration (perturbations)**: They modify the current path $\mathbf{X}_{s,t}$ in a small amount by just slightly perturbing directions and vertex positions.

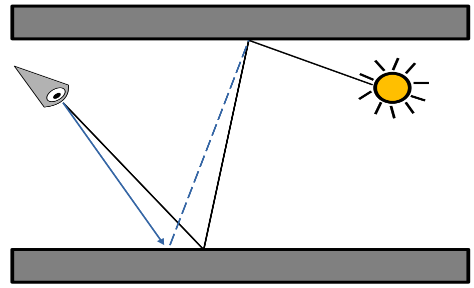
Despite that mutations are essential to meet the requirement of ergodicity and promote the stratification of the samples of the chain across the image plane. They are usually rejected in regions with difficult paths (e.g., caustic paths and paths with difficult visibility). This is because large modifications applied to a difficult path will tend to return a mutated path too far out of the zone of high radiance encompassed by the difficult paths similar to the original path that was mutated. For example, in Figure 3.1a, a lot of bidirectional mutations would result in black paths as they couldn't be reconnected due to occlusions.

To increase the acceptance rate of the proposed paths, small mutations, also known as perturbations, are introduced, which aim to avoid extreme changes in the path inducing a significant increase in the acceptance rate of proposed paths. The three perturbations proposed by Veach were the lens perturbation which consists in perturbing the direction of the ray shot by the point on the lens (see Figure 3.1b), the caustic perturbation, that is done in exactly the same way that the lens perturbation but perturbing the initial direction of the vertex on the light (see Figure 3.1c), and the multi chain perturbation (see Figure 3.1d).

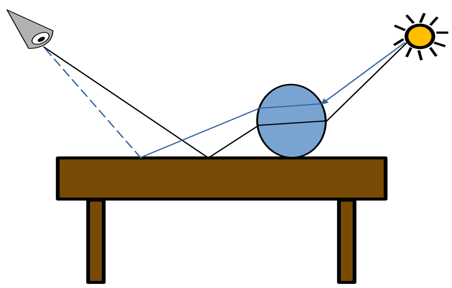
It is important to remark that small mutations are the core of the Metropolis Light Transport, which is a *local exploration* algorithm, meaning that a difficult path needs to be found by mere chance, and only after we have found this difficult path is that we can locally explore similar paths. Large mutation are also very important to guarantee the ergodicity of the chain, and hence, to find the difficult paths by mere chance, so one problem of interest is to develop a method that in some way can guide the large mutations to regions with a high concentration of light paths, using a global exploration technique such as path guiding.



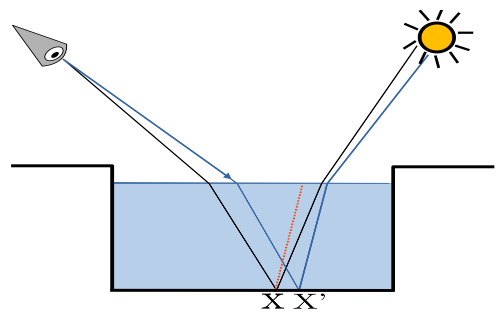
(a) Bidirectional mutations randomly select, according to some pdf, any possible sub-path of $\mathbf{X} = (\mathbf{x}_0, ..., \mathbf{x}_6)$ (including itself), deleting it, resulting in two (possibly empty) unconnected paths, in this example the red line $(\mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4)$. It then replaces the deleted path with two new sampled paths (one for each side of the deletion), of variable length (in this example, one vertex from end point $\mathbf{x}_2$ and three vertices sampled from end point $\mathbf{x}_4$) that will be re-connected.



(b) A lens perturbation only modifies the direction coming from the camera. The only difference between a lens perturbation and a lens mutation is that the lens mutation is more abrupt, drastically changing pixels.



(c) Caustic perturbation. The blue egg is a specular refractive material. This type of perturbation is analogous to the caustic perturbation but starting from the light point instead of the camera point.



(d) Multichain perturbation. A specular transmissive pool (top) and its diffuse floor (bottom). A lens perturbation is performed obtaining a point $\mathbf{x}'$, then the point $\mathbf{x}$ in the original path is perturbed and this direction is used on $\mathbf{x}'$ (the red dotted line is paralel to the line coming out of $\mathbf{x}'$)

Figure 3.1: Small mutations, or perturbations are the core of the MLT algorithm, as they can locally explore around similar paths of a difficult path. The black path is the current path, the blue path is the proposed perturbed path and the dashed line indicates a deterministic connection between vertices.

Another important problem is to develop efficient techniques for the Metropolis algorithms that increase the stratification of samples in the image space, as we ideally would like the samples to be uniformly distributed in the image space, like is the case in PT and BDPT (for the case $t > 1$). Due to the luminance target function of the MLT algorithm, brighter pixels will receive more samples on average, which is a tremendously bad behavior as darker pixels may also have difficult paths that we want to explore (e.g. regions with shadows).

Despite that the mutations and perturbations of path-space MLT are highly customizable, they have the problem that are extremely difficult to implement, and are quite specialized (see Figure 3.1d) as they are created having specific types of paths in mind. This algorithm can't be easily combined with new rendering techniques and is not generalizable, meaning that if new types of light transport are introduced, such as subsurface scattering, volumetric rendering, among others. We would need to design a new type of mutation every time a new type of transport is introduced, a quite cumbersome task.

Another big issue of path-space MLT is that even the large mutations are highly correlated, as they are often reusing parts of the original path, which does indeed make the large mutation more efficient as there are less visibility checks, but with the drawback that this hinders the global exploration of the large step, which is precisely the thing we want to avoid [Ots+17].

In this thesis we will not use path-space domain MLT, but it is useful to mention it as it is more intuitive to speak in terms of the path-space first. We highly encourage the reader to check the implementation of path-space MLT, found in the Mitsuba [Jak10] renderer, as it is one of the most beautifully crafted algorithms in the realm of computer simulations. We will proceed now to talk about the type of Metropolis Light Transport used in this thesis.

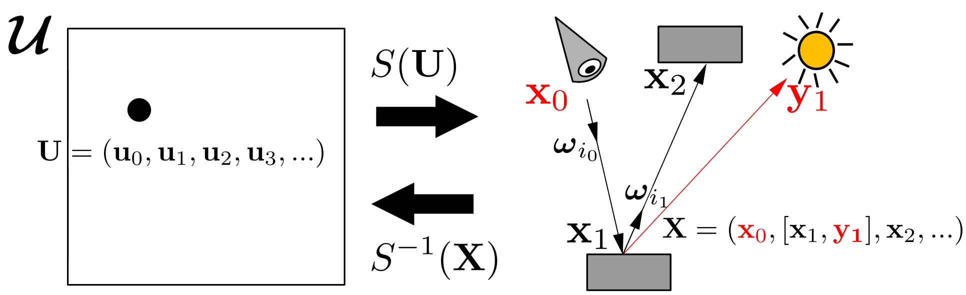
3.1 Primary Sample Space Metropolis Light Transport

In this section we will now present an algorithm that is more general, as it can be applied to any kind of paths without any special kind of distinctions, making the algorithm robust, easy to implement, and above all, with a comparable efficiency to path-space MLT. Furthermore, this algorithm will propose completely new paths as as large mutation making the range of exploration of the large mutations bigger.

In rendering systems, paths are obtained applying the inverse transform sampling method that uses canonical random numbers, so the path-space $\mathcal{P}$ can unequivocally be mapped to the hypercube of random numbers $\boldsymbol{\mathcal{U}} = \cup_{k=1}^{\infty}[0,1]^k$, domain known as the *primary sample space* (PSS) domain [Kel+02]. Using a transformation $S(\mathbf{U}) = \mathbf{X}$, or more precisely, using Equation 2.45 we have

$$S(\mathbf{U}) = \mathbf{X} = \Big(\mathbf{x}_0 = (P_{f_0}^{\mu_0}(\mathbf{u}_0))^{-1}, \mathbf{x}_1 = (P_{f_1}^{\mu_1}(\mathbf{u}_1, \mathbf{x}_0))^{-1}, ..., \mathbf{x}_k = (P_{f_k}^{\mu_k}(\mathbf{u}_k, \mathbf{x}_{k-1}))^{-1}\Big) = (P_S^{\mu}(\mathbf{U}))^{-1}, \tag{3.1}$$

where each f_i denotes the pdf function that was used on each vertex $\mathbf{x}_i \in \mathbf{X}$, μ_i represents their measures (e.g. solid angle, or area sampling), and the functions $P_{f_k}^{\mu_k}$ are the cdf functions of the pdfs which have a dependency on the previous vertex[1], so P_S^{μ} is just a way of abbreviating these dependencies. For example, in Figure 3.2, a vector of random numbers $\mathbf{U}$ generates the NEE path traced path $\mathbf{X}$. Note that the samplings are with respect of different pdfs $(p_{\rho_{\mathbf{x}_0}}, p_{\rho_{\mathbf{x}_1}}, p_{A_{\text{lens}}}, p_{A_{\text{light}}})$ and two different measures (p^{ω}, p^A). The sequence of random numbers used for sampling uniquely determines the traced path of a given integrator, which could be PT, NEE-PT, BDPT, or even a new technique.



$$\mathbf{x}_0 = (P_{\text{A}_{\text{lens}}}^{A}(\mathbf{u}_0))^{-1} \quad \boldsymbol{\omega}_{i_0} = (P_{\rho_{\mathbf{x}_0}}^{\omega}(\mathbf{u}_1))^{-1} \quad \mathbf{y}_1 = (P_{\text{A}_{\text{light}}}^{A}(\mathbf{u}_2))^{-1} \quad \boldsymbol{\omega}_{i_1} = (P_{\rho_{\mathbf{x}_1}}^{\omega}(\mathbf{u}_3))^{-1}$$

$$\mathbf{x}_1 = t(\mathbf{x}_0, \boldsymbol{\omega}_{i_0}) \qquad \mathbf{x}_2 = t(\mathbf{x}_1, \boldsymbol{\omega}_{i_1})$$

$$\mathbf{u}_0 = P_{\text{A}_{\text{lens}}}^{A}(\mathbf{x}_0) \quad \mathbf{u}_1 = P_{\rho_{\mathbf{x}_0}}^{\omega}(\boldsymbol{\omega}_{i_0}) \quad \mathbf{u}_2 = P_{\text{A}_{\text{light}}}^{A}(\mathbf{y}_1) \quad \mathbf{u}_3 = P_{\rho_{\mathbf{x}_1}}^{\omega}(\boldsymbol{\omega}_{i_1})$$

Figure 3.2: The *sequence* of random numbers $\mathbf{U}$ used for sampling vertices and directions determines in a unique way the traced path $\mathbf{X}$. In this example we are showing a PT with NEE. The directions $\boldsymbol{\omega}_{i_0}, \boldsymbol{\omega}_{i_1}$ were sampled using the inverse transform method (the inverse cdf P^{-1} in the solid angle domain) of the material pdfs $p_{\rho_{\mathbf{x}_0}}^{\omega}, p_{\rho_{\mathbf{x}_1}}^{\omega}$ at their respective vertices, while the area points $\mathbf{x}_0, \mathbf{y}_1$ (red vertices) were obtained by using the inverse transform method (on the area domain) of the lens $p_{A_{\text{lens}}}^{A}, p_{A_{\text{light}}}^{A}$. Notice that the red vertices indicate an area domain sampling, and the black vertices were obtained by a solid angle sampling followed by the ray casting function t to retrieve the vertex. The random numbers $\mathbf{u}$ can be retrieved by simply evaluating the appropriate cdf functions of the points or directions.

[1]Technically on all the previous vertices or directions, depending of the situation.

Then with Equation 2.49, applied to each and every vertex we obtain the jacobian determinant

$$\left|\frac{dS(\mathbf{U})}{d\mathbf{U}}\right| = \begin{vmatrix} \frac{dP^{-1}_{f_0}}{d\mathbf{u}_0} & 0 & \dots & 0 & 0 \\ 0 & \frac{dP^{-1}_{f_1}}{d\mathbf{u}_1} & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \dots & 0 \\ 0 & \dots & \dots & \dots & \frac{dP^{-1}_{f_k}}{d\mathbf{u}_k} \end{vmatrix} = \prod_{j=0}^{k} \frac{1}{p_{f_j}(P^{-1}_{f_j}(\mathbf{u}_j))} \equiv \frac{1}{p_S(S(\mathbf{U}))} = \frac{1}{p_S(\mathbf{X})}, \tag{3.2}$$

which is vital in order to make the change of integration measures 2.47 of the measurement integral in path-space to the domain of canonical random numbers

$$I_j = \int_{\mathcal{P}} f(\mathbf{X})h_j(\mathbf{X})\,\mathrm{d}\mu_1(\mathbf{X}) = \int_{\boldsymbol{\mathcal{U}}} \underbrace{\frac{f(S(\mathbf{U}))}{p_S(S(\mathbf{U}))}}_{C(S(\mathbf{U}))} h_j(S(\mathbf{U}))\,\mathrm{d}\mu_2(\mathbf{U}) = \int_{\boldsymbol{\mathcal{U}}} \hat{C}(\mathbf{U})\hat{h}_j(\mathbf{U})\,\mathrm{d}\mu_2(\mathbf{U}) \tag{3.3}$$

where $\hat{C}(\mathbf{U}) = C(\mathbf{X}) = C(S(\mathbf{U}))$ and $\hat{h}_j(\mathbf{U}) = h_j(S(\mathbf{U}))$ is just a way to represent that every variable $\mathbf{U}_0$ is tied to a specific $\mathbf{X}_0$ through the transformation $S(\mathbf{U}_0) = \mathbf{X}_0$, so the evaluation of the function is preserved as the transformation of the coordinates themselves account for the change. For easiness of notation we will often type $\hat{C}(\mathbf{U}) = C(\mathbf{U})$, where the different domains of the function in a way tells us that this is essentially a different function. The *uniformly distributed* MC estimator over the measure of random numbers is equivalent to the MC estimator in path-space that uses the probability of sampling the paths as a distribution

$$I_j = \int_{\boldsymbol{\mathcal{U}}} \frac{f_j(S(\mathbf{U}))}{p_S(S(\mathbf{U}))}\,\mathrm{d}\mu_2(\mathbf{U}) \approx I^{(N)} = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{f_j(S(\mathbf{U}_i))}{p_S(S(\mathbf{U}_i))}\right)\frac{1}{p^u(\mathbf{U}_i)} = \frac{1}{N}\sum_{i=1}^{N}\frac{f_j(S(\mathbf{U}_i))}{p_S(S(\mathbf{U}_i))}, \tag{3.4}$$

where the probability of getting some $\mathbf{U}_i \in \boldsymbol{\mathcal{U}}$ is of course $p^u(\mathbf{U}) = 1 = \frac{1}{1^{\mathrm{size}(\mathbf{U})}}$.

The advantages of this reformulation are not evident in PT and BDPT, as this only looks as a change of name, but this is not the case for MCMC algorithms, because manipulating the random numbers themselves is way easier as we don't have to care about the specifics of the structures of the paths. The MCMC algorithm that manipulates the random numbers in the primary sample space instead of the vertices in the path-space is known as *Primary Sample Space Metropolis Light Transport* (PSSMLT) [Kel+02] and has two type of mutations

1. **Small step mutation**: This mutation is the PSS equivalent to path-space perturbations. This mutation creates a proposed random number vector $\mathbf{V}$ that is very close component to component to the current random number vector $\mathbf{U}$ (see Figure 3.3), this mutation could be defined in several ways, being one of the most common to just use a normal distribution with standard deviation σ and mean around the value to be mutated:

$$\mathbf{V} = (v_0 \sim \mathcal{N}(u_0, \sigma), v_1 \sim \mathcal{N}(u_1, \sigma), ...) \tag{3.5}$$

The probability of a small mutation in a Markov Chain is p_{small}, and if only small and large mutations are done by the chain then this probability is $p_{\mathrm{small}} = 1 - p_{\mathrm{large}}$. The choice of the standard deviation is usually defined by the user, being a common value in rendering $\sigma = 0.01$. It is also possible to dynamically modify σ on every small mutation step so σ can be as large as possible while retaining a high acceptance rate of the proposed mutations [HJ11].

2. **Large step mutations**: The mutation that ensures the chain to be ergodic, the PSS analogous to the bidirectional mutation. This mutation creates a whole new random number without even considering the current random number, that is

$$\mathbf{V} = (v_0 \sim U[0,1), v_1 \sim U[0,1), ...). \tag{3.6}$$

All the paths obtained by PSS large mutation are completely different than the path that is being "mutated". Unfortunately despite that the large mutation has a wider range of exploration, it will still cause correlation when the path has a very high radiance, as the Markov process will tend to reject paths with a lower radiance contribution.

A large mutation in a Markov Chain has an occurrence probability equal to p_{large}, which means that the expected number of large mutations for a Markov Chain of N samples is Np_{large}. The value p_{large} is usually a user given parameter, being a common value in rendering $p_{\text{large}} = 0.3$, but an optimal value could be approximated in a preprocessing step [ZS13].

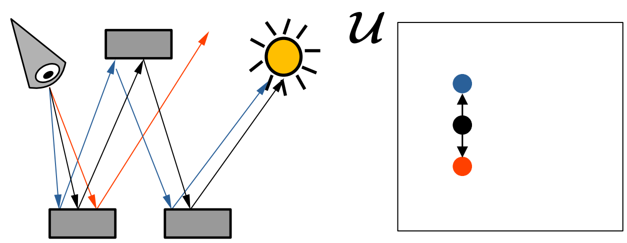


Figure 3.3: Small mutations in PSS. Note the main disadvantage of small mutations is that sometimes small changes could make the mutated path to drastically change (red path), effect that is intensified for longer chains of vertices.

Note that the transition probability density function for both of these mutations is symmetric $T(\mathbf{U} \to \mathbf{V}) = T(\mathbf{V} \to \mathbf{U})$, so the calculation of these factors can be ignored when calculating the acceptance pdf.

The MCMC estimator in primary sample space for samples obtained with probability in path-space $p_{\text{MCMC}} = \frac{I(\mathbf{X})}{b}$ and probability in primary sample space $p^u_{\text{MCMC}} = \frac{I(S(\mathbf{X}))}{bp_S(S(\mathbf{X}))}$ is

$$I_N = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{f_j(S(\mathbf{U}_i))}{p_S(S(\mathbf{U}_i))}\right)\frac{1}{p^u_{\text{MCMC}}(\mathbf{U}_i)} = \frac{b}{N}\sum_{i=1}^{N}\left(\frac{f_j(S(\mathbf{U}_i))}{p_S(S(\mathbf{U}_i))}\right)\Big/\left(\frac{I(S(\mathbf{U}_i))}{p_S(S(\mathbf{U}_i))}\right) \tag{3.7}$$

where I is the target scalar function, and b is the normalization constant (without including the filtering function)

$$b = \int_{\mathcal{P}} I(\mathbf{X})\, \mathrm{d}\mu_{A_p}(\mathbf{X}) = \int_{\mathcal{U}} \frac{I(S(\mathbf{U}))}{p_S(S(\mathbf{U}))}\, \mathrm{d}\mu_u(\mathbf{U}) = \int_{\mathcal{U}} I(\mathbf{U})\, \mathrm{d}\mu_u(\mathbf{U}), \tag{3.8}$$

note how the scalar function in primary sample space domain needs the probability density of generating the path as part of the function, and $I(\mathbf{U})$ is implied to denote $\hat{I}(\mathbf{U})$. The acceptance probabilities in both path-space and primary sample space are:

$$\alpha^{\mathcal{P}}(\mathbf{X} \to \mathbf{Y}) = \min\left(1, \frac{I(\mathbf{Y})T^{\mathcal{P}}(\mathbf{Y} \to \mathbf{X})}{I(\mathbf{X})T^{\mathcal{P}}(\mathbf{X} \to \mathbf{Y})}\right); \alpha^u(\mathbf{U} \to \mathbf{V}) = \min\left(1, \frac{T^u(\mathbf{V} \to \mathbf{U})\frac{I(S(\mathbf{V}))}{p_S(S(\mathbf{V}))}}{T^u(\mathbf{U} \to \mathbf{V})\frac{I(S(\mathbf{U}))}{p_S(S(\mathbf{U}))}}\right). \tag{3.9}$$

The importance function for a forward unidirectional path tracer in PSS is simply the radiance carried by the path divided by its probability of being generated in path-space[2]

$$I(\mathbf{U}) = \frac{f^*(\mathbf{X})}{p_S(\mathbf{X})}; \qquad S(\mathbf{U}) = \mathbf{X}, \tag{3.10}$$

for the case of a BDPT integrator, several paths are obtained by connecting all the vertices of the camera path with all the vertices of the light path, (see Figure 3.4), sampling in this way different lengths *and* different s,t strategies, then the importance function that takes into account all of the paths generated by a BDPT [Kel+02] sample is

$$I(\mathbf{U}) = \max\left\{ w_{s,t}(\mathbf{X}_{s,t})\frac{f^*(\mathbf{X}_{s,t})}{p^{A_p}_{s,t}(\mathbf{X}_{s,t})} \middle| 0 \le s \le l, 0 \le t \le c \right\}, \quad S(\mathbf{U}) = \begin{cases} \mathbf{Y} = (\mathbf{y}_0, ..., \mathbf{y}_{l-1}) \\ \mathbf{Z} = (\mathbf{z}_0, ..., \mathbf{z}_{c-1}), \end{cases} \tag{3.11}$$

which is preferred over just the sum of each individual contributions as this would favor longer paths because they have more possible s,t combinations [Kel+02].

[2]The importance function could consider the filter $I(\mathbf{X}) = \sum_{k=1}^{A_{\text{image}}} I^{(k)}(\mathbf{X}) = \sum_{k=1}^{A_{\text{image}}} h_k(\mathbf{X})f^*(\mathbf{X})$

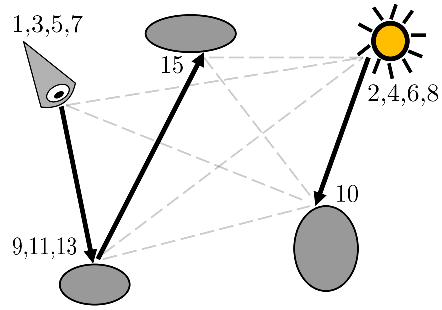


Figure 3.4: The random numbers used by a BDPT integrator. Note that it is good practice to separate the random numbers of the camera path and light path in two different arrays, in this case denoted as the odd and even numbers.

In summary, the main advantages of this algorithm are: its simplicity while retaining the efficiency of its path-space counter part and the increased exploration of the state space by the large mutations. On the other hand, its disadvantages are:

- It treats everything as a black box that receives random numbers and outputs a result, losing the control of the inner processes at the path-space level.
- A small mutation could result in a very different path, phenomenon that is intensified in a "ripple" effect, in which the longer the path, the more different its mutation will be.
- It is full of many parameters defined by the user, such as the number of chains, the large step probability, the size and type of small mutation, the number of initialization samples, and so forth.
- The correlation provoked by the nature of the MCMC algorithms increases the variance of the estimator, the more correlated the samples the bigger the variance [Kel+02] (see Figure 3.5).
- If it is combined with a BDPT integrator, it is computationally expensive, due to the fact that each step of the Markov Chain must perform all the possible connections between camera sub-paths and light sub-paths to estimate the whole sum 2.129. To mitigate this we will present in the next subsection a way to avoid using all the BDPT connections in each iteration, increasing in this way the efficiency of the algorithm

Figure 3.5: This image was rendered using adaptive MCMC [HJ11]. This algorithm automatically adjusts the size of the small mutations to reach an acceptance rate of the small mutations defined by the user, in this example 0.9. An acceptance rate this high makes the size of the small step very small, increasing the correlation of the samples. This increased correlation is very noticeable in the image and it is a cause of variance.

3.1.1 Implementation

3.1.1.1 A general overview

We will present the pseudocode and then explain the important details

Algorithm 6 PSSMLT

Input: `Camera`, `Sampler`
Output: Rendered image.
1: $\mathcal{B} \leftarrow$`GetInitialSamples(`$N_{\texttt{init}}$`)` ▷ N_{init} number of initialization samples defined by the user
2: $b \leftarrow$`ApproximateIntegrationConstant(`$\mathcal{B}$`)`
3: $N \leftarrow \texttt{mpp}A_{\text{image}}$ ▷ Total MCMC samples equal mutatione per pixel times amount of pixels.
4: **for** `k` $= 0$; `k` $< N_{\text{chains}}$; `k++` **do**
5: $N_c \leftarrow N/N_{\text{chains}}$
6: `rngIndex` $\leftarrow$ `Sample(`$\mathcal{B}$`)` ▷ $\mathbf{U}_0$ for current chain.
7: `sampler` $\leftarrow$ `MLTSampler(mpp,rngIndex,`σ`,pLarge)`
8: // `L_PSSMLT` returns importance 3.11 and the splatting radiance 2.129.
9: `(Lcurrent,Icurrent,rasterCurrent)` $\leftarrow$ `L_PSSMLT(sampler)`
10: **for** `i` $= 0$;`i`$< N_c$; `i++` **do**
11: `sampler.StartIteration()` ▷ Chooses between large or small step, with prob. `pLarge`.
12: `(Lproposed,Iproposed,rasterProposed)`$\leftarrow$ `L_PSSMLT(sampler)`
13: $\alpha \leftarrow \min\left(1, \frac{\texttt{Iproposed}}{\texttt{Icurrent}}\right)$
14: `ProposedWeight`$\leftarrow \frac{\alpha+\texttt{isLargeStep}}{\texttt{Iproposed}/b+\texttt{pLarge}}$ `CurrentWeight` $\leftarrow \frac{(1-\alpha)}{\texttt{Icurrent}/b+\texttt{pLarge}}$
15: **if** `Iproposed`$>$`0` **then**
16: `AddSplat(film,rasterProposed,LProposed*ProposedWeight)`
17: **end if**
18: **if** `Icurrent`$>$`0` **then**
19: `AddSplat(film,rasterCurrent,LCurrent*currentWeight)`
20: **end if**
21: **if** `rand()`$< \alpha$ **then**
22: `(Lcurrent,Icurrent,rasterCurrent)` $\leftarrow$ `(Lproposed,Iproposed,rasterProposed)`
23: `sampler.Accept()` ▷ Keeps the new state.
24: **else**
25: `sampler.Reject()` ▷ Restores the previous state using the back ups.
26: **end if**
27: **end for**
28: **end for**
29: `ScaleFilm(b/mpp)`

We will start by getting an initialization array $\mathcal{B}$ (see line 1)

$$\mathcal{B} = [(I(\mathbf{U}_0), \mathbf{U}_0), (I(\mathbf{U}_1), \mathbf{U}_1), ..., (I(\mathbf{U}_{N_{\text{init}}-1}), \mathbf{U}_{N_{\text{init}}-1})]; \qquad \hat{b} = \frac{\text{sum}(\mathcal{B})}{N_{\text{init}}} \tag{3.12}$$

with this array we can obtain the approximation $\hat{b}$ to the normalization constant b, as all the elements of the array are random samples obtained uniformly across the whole film

$$b = \int_{\mathcal{U}} I(\mathbf{U})\, \mathrm{d}\mu_u(\mathbf{U}) \approx \hat{b} = \frac{1}{N_{\text{init}}} \sum_{i=1}^{N_{\text{init}}} \frac{I(S(\mathbf{U}_i))}{p_S(S(\mathbf{U}_i))} = \frac{1}{N_{\text{init}}} \sum_{i=1}^{N_{\text{init}}} I(\mathbf{U}_i), \tag{3.13}$$

for simplicity we will refer to the uniform MC approximation in PSS $\hat{b}$ as b. We will also use array $\mathcal{B}$ to do a continuous sampling (the state space is continuous) in order to get the first state $\mathbf{U}_0$ of each chain (see line 6), this continuous array sampling of $\mathcal{B}$ follows then the initial distribution

$$p_0(\mathbf{U}) = \frac{I(\mathbf{U}_0)}{b}; \qquad W = \frac{I(\mathbf{U}_0)}{p_0(\mathbf{U}_0)} = b, \tag{3.14}$$

where W is the start-up weight to make the algorithm unbiased. As it was already discussed in the preliminaries, the use of the weight W requires us to use several Markov Chains, which are going to be run in parallel. Note something quite important, because several Markov chains that can visit all the pixels are run in parallel, the splatting into the film should be done taking into account thread safety in order to avoid *race conditions*[3].

The estimator with the start-up weight is then:

$$I_j \approx \frac{bA_{\text{image}}}{N} \sum_{i=1}^{N} \frac{f_j(\mathbf{U}_i)}{I(\mathbf{U}_i)}, \tag{3.15}$$

where importance I for the BDPT case is defined as the maximum MIS weighted strategy 3.11, and the radiance splatting f is the whole bidirectional MIS weighted sum 2.129. For the forward unidirectional path tracer the splatting and the importance are just the radiance and its scalar value.

The factors presented in algorithm 6 (line 14) can be obtained by making the following two improvements:

1. **Not completely discarding a rejected proposed sample that has a non zero contribution** [Vea97]: The random variable $\mathbf{U}_{k+1}|\mathbf{U}_k$ could either be $\mathbf{U}_{k+1} = \mathbf{U}_k$ if the mutation was rejected, or being equal to the proposed mutation random variable $\mathbf{U}_{k+1} = \mathbf{U}_m$. To avoid throwing away the information about the proposal, the random variable of the stochastic process of acceptance is replaced by its mean, which means to splat both the proposed mutation (even if it was rejected) and the current sample again (even if the proposal was accepted), with the weights α and $(1-\alpha)$ respectively. This process is known as *mean value substitution* and splits the MCMC estimator in two terms:

$$I_j \approx \frac{bA_{\text{image}}}{N} \left[\sum_{i=1}^{N} \alpha \frac{f_j(\mathbf{U}_{i_p})}{I(\mathbf{U}_{i_p})} + \sum_{i=1}^{N} (1-\alpha) \frac{f_j(\mathbf{U}_{i_c})}{I(\mathbf{U}_{i_c})} \right], \tag{3.16}$$

   where $\mathbf{U}_{i_c}$ is the current vector of random numbers and $\mathbf{U}_{i_p}$ is the proposed vector of random numbers.

2. **Applying MIS to only the large mutations of the MCMC process** [Kel+02]: To do this we use the large mutations of the MCMC estimator as two type of samples at the same time, as the sample of a MCMC estimator with pdf $p_1 = I(\mathbf{U})/b$ and $N_1 = N$ samples, and also as the sample of a uniform MC estimator with pdf $p_2(\mathbf{U}) = 1$ and an expected number of samples $N_2 = Np_{\text{large}}$. Using the balance heuristic

$$m_i(\mathbf{U}) = \frac{N_i p_i(\mathbf{U})}{Np_1(\mathbf{U}) + Np_{\text{large}}p_2(\mathbf{U})}; \qquad m(\mathbf{U}) = \frac{1}{(I(\mathbf{U})/b + p_{\text{large}})N}, \tag{3.17}$$

   where $m(\mathbf{U})$ is the weight that remains after cancelling the numerator with the pdf and the amount of samples of each estimator. Note the subtle difference for the MIS weight of this case, usually when we use an MIS estimator a different sample is generated for each technique, but here we are using the same exact sample for the two different techniques, as if the same sample was obtained in two different ways, so to speak.

   To obtain the specific scaling, we first note that we can scale all the current samples of the MCMC process in a Veach style:

$$\frac{(1-\alpha)f_j(\mathbf{U}_c)}{(f^*(\mathbf{U}_c)/b + p_{\text{large}})N} \text{ (Current MCMC sample)}, \tag{3.18}$$

   and finally, the contribution of a large mutation can either come from a proposal of the MCMC estimator or from its own independent uniform MC estimator:

$$\frac{(\alpha+1)f_j(\mathbf{U}_l)}{(f^*(\mathbf{U}_l)/b + p_{\text{large}})N} \text{ (Large proposal + Uniform MC)} \quad \frac{\alpha f_j(\mathbf{U}_s)}{(f^*(\mathbf{U}_s)/b + p_{\text{large}})N} \text{ (Small proposal)} \tag{3.19}$$

[3]For example, if both threads do the operation on a global variable $a_i = a_i + c$, this could potentially induce an incorrect calculation.

3.1.1.2 Primary Sample Space sampler

To complete the general algorithm, the last thing that needs to be explained is how to update the sampler on each iteration.

A Markov Chain in primary sample space can basically be understood as a sampler, which contains a uniform random number generator (RNG), and an array[4] that stores the current random numbers with information about the last time (iteration) in which they were modified-

```
struct PrimarySample {
public:
    double value; //canonical random number
    double value_backup; //needed if the proposal is rejected
    int64_t lastModificationIteration;//last time this number changed
    int64_t modifyBackup;//back up of the time of last modification
};
//The sampler itself is the Markov Chain
class PSSMLTSampler {
public:
    std::vector<PrimarySample> U;//array of random numbers
    RNG rng: //canonical random number generator
    int64_t currentIteration;
    int64_t lastLargeStepIteration;//last time a large step was accepted
    int sampleIndex;//current element of the U array
    bool isLargeStep=true;//if false a samll mutation
    float largeStepProb;
    //Methods
    void mutate();//mutate U
    void accept();//accept proposal
    void reject();//reject proposal
    void start();//start sampleIndex at 0 and choose kind of mutation
    void get1D();//get random number mutating the current one
};
```

One important thing to point out is that the mutation $\mathbf{V}_k = \text{mut}(\mathbf{U}_k)$, with associated path $S(\mathbf{V}_k) = \mathbf{Y}_k$ might use a totally different amount of random numbers than the path $S(\mathbf{U}_k) = \mathbf{X}_k$, because one of the paths might have ended sooner for causes such as termination by Russian roulette, the ray escaping the scene, or for any other reason. For example, suppose that the path $\mathbf{X}$ used $m+1$ random numbers, and the path $\mathbf{Y}$ used $n+1 < m+1$ random numbers

$$\begin{aligned} \mathbf{U}_k &= (u_0^k, u_1^k, u_2^k, ..., u_n^k, u_{n+1}^k ..., u_m^k) \\ \mathbf{V}_k &= (v_0^k, v_1^k, ..., v_n^k), \end{aligned} \tag{3.20}$$

the way in which a Markov Chain is usually implemented is that each random number is mutated only when it is called by the function `get1D()`, which in turn, is called when the Markov Chain is being used to trace paths in the scene, therefore when the smaller path is traced there will be non mutated random numbers on the current chain, but this is a problem, because all the vectors of random numbers in primary sample space are in theory infinite, so *every* random number should always be mutated. The worst possible solution to this would be to just iterate the whole array of random numbers every time to perform their respective mutations, which it is only possible if the integrator has a maximum depth m_d, even if we use Russian Roulette, as we would simply iterate over the finite amount of random numbers defined by the maximum depth.

A more efficient approach to keep a consistent mutation scheme on all the random numbers of the chain is to keep track of the current iteration `currentTime` of the chain, the last time that each and every element of the array of random numbers was modified `lastModificationIteration`, and the last large mutation time of the chain `lastLargeStepIteration` (stored by the sampler), then with this values we can perform *lazy mutations*, which consist in lazily (efficiently) modifying the random number only when it is used by the sampler:

[4] If several types of paths obtained in a different way are present in the integration process, e.g., the camera paths and the light paths of BDPT, it is recommended to have one array of random numbers that will handle each type of path separately.

Algorithm 7 Lazy evaluation

Input: Primary sample u that is going to be lazily mutated

```
if u.lastModificationIteration < lastLargeStepIteration then
    u.value ← rand()
    u.lastModificationIteration ← lastLargeStepIteration
end if
if isLargeStep then
    u.backUp()                                        ▷ In case of rejected proposal
    u.value ← rand()
    u.lastModificationIteration ← currentIteration
else
    nSmall ← currentIteration − u.largeModificationIteration
    i ← 0
    while i < nSmall − 1 do
        u.value ← smallMutate(u.value)
        u.lastModificationIteration++
        i++
    end while
    u.backUp()
    u.value ← smallMutate(u.value)
    u.lastModificationIteration++
end if
```

This sampler, reiterating, is the MCMC chain itself, and the lazy evaluation algorithm constitutes the soul of the MCMC sampling process in PSS.

The next section we will talk about how to improve PSSMLT, using just one s, t stategy out of all the possible ones on each iteration of the chain.

3.2 Multiplexed Metropolis Light Transport

The efficiency of the PSSMLT algorithm is greatly improved if we limit each chain to just explore paths of the same length k (depth $d = k - 1$), this means that each chain will estimate only one integral of the sum of integrals 2.124. Because of this we need to use at least one chain per each available path length, to get a correct final render with paths of all the lengths.

After fixing the path length k of the chain, we can define the extended state space $(\mathbf{U}, T_s) \equiv \mathbf{U}_{s,t}$ corresponding to a specific s, t technique (see Figure 3.6), where $T_s = s$ is used to indicate the strategy parameter (the "temperature") of the target distribution. The chain will be able to jump between extended states on each iteration, efficiently using just one (s, t) path on each iteration, instead of using all the possible reconnection strategies (s, t) like in PSSMLT. The process of jumping between extended state spaces is called *simulated tempering* [HKD14], and has an importance function

$$I(\mathbf{U}) = \sum_{s=0}^{d+1} I^{(s)}(\mathbf{U}) = \sum_{s=0}^{d+1} f^*(\mathbf{U}, T_s, d) \tag{3.21}$$

with as always, s is choosing the number of light vertices, corresponding to strategy $(s, k+1-s)$, and d is the fixed depth of the chain. Notice that the $d+3$ possible techniques range from $s = 0$ (forward path tracing) to $s = d+2$ (light tracing), but we will choose to ignore light tracing, as we are working with a pinhole camera, only using $d+2$ possible techniques.

The target distribution for each s, t technique is defined as $f^*(\mathbf{X}, T_s, d) = w_s(\mathbf{X}) \frac{f^*(\mathbf{X})}{p_s(\mathbf{X})} p_s(\mathbf{X})$ with transformation to PSS

$$f^*(\mathbf{X}, T_s, d)\,\mathrm{d}\mu(\mathbf{X}) = w_s(\mathbf{X}) \frac{f^*(\mathbf{X})}{p_s(\mathbf{X})} p_s(\mathbf{X})\,\mathrm{d}\mu(\mathbf{X}) = w_s(S(\mathbf{U})) \frac{f^*(S(\mathbf{U}))}{p_s(S(\mathbf{U}))}\,\mathrm{d}\mathbf{U} = f^*(\mathbf{U}, T_s, d)\,\mathrm{d}\mathbf{U}, \tag{3.22}$$

and using a symmetric mutation we have that the acceptance rate is,

$$\alpha\left((\mathbf{U}, s) \to (\mathbf{V}, s')\right) = \min\left(1, \frac{w_{s'}(S_{s'}(\mathbf{V})) f^*_{s'}(\mathbf{V})}{w_{s'}(S_s(\mathbf{U})) f^*_s(\mathbf{U})}\right), \tag{3.23}$$

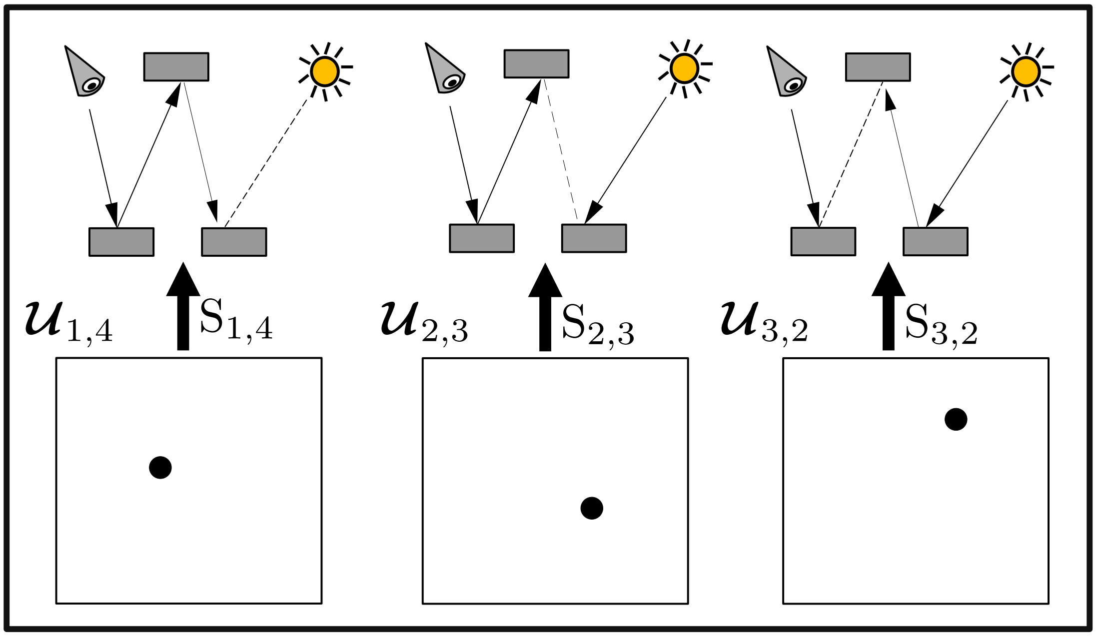


Figure 3.6: MMLT divides each fixed depth, into several $\mathbf{U}$, corresponding to each possible (s,t) technique. In this image, we can see paths of depth 3 (length 4). Note that each point is in a different position in each primary sample domain, this was made in this way to indicate that usually $S_{i,j}(\mathbf{U}_{i,j}) \neq S_{k,l}(\mathbf{U}_{i,j})$.

where $S_s = S_{s,k+1-s}$ is just the transformation from PSS to path-space that "knows" the (s,t) strategy being used. The name of the algorithm that chooses a different path length for each chain, and uses an extended exploration state is called *Multiplexed Metropolis Light Transport* (MMLT) [HKD14].

The initialization process and the splatted weights of the MMLT algorithm are just a little different from PSSMLT. MMLT calculates m_d+1 different normalization constants, where m_d is the maximum depth of the paths present in the image[5], then the normalization for each depth is

$$b_d = \int \sum_{s=0}^{d+1} f^*(\mathbf{U}, T_s, d) d\mathbf{U} \approx \frac{1}{N_{\text{init}}} \sum_{i=0}^{N_{\text{init}}-1} f^*(\mathbf{U}_i, T_{s_i}, d); \qquad 0 \leq d \leq m_d, \tag{3.24}$$

where each random sample of this estimator is of the form $(\mathbf{U}_i, t_i)$, that is, the usual vector of random numbers with an extra discrete index, this extended state sample can be abbreviated as $\mathbf{U}_i$ remembering that the selection of the discrete state is part of $\mathbf{U}_i$. Note that each b_d is an average value over the film, so it is invariant to resolution changes, and it is also an average over the $d+2$ possible connection strategies. With all of these constants we can create an array

$$B = [b_0, b_1, ..., b_{m_d}] \qquad b = \text{sum}(B) \qquad p_d[i] = \frac{b_i}{b}, \tag{3.25}$$

where p_d is the *discrete* pdf associated with the stochastic process of choosing the length of the path.

The initialization process of the MMLT algorithm can be done by creating an array $\mathcal{B}$ of all the

[5] This is a disadvantage of MMLT, we need to assume a maximum depth to get an unbiased algorithm, unlike PSSMLT. We could add extra depths on the fly, by allowing termination by Russian Roulette and using lazy mutations [HKD14], but this would induce bias as we need the information given by both the normalization constants themselves and the *amount* of normalization constants to perform the stochastic process of choosing a depth. A simple analogy of this problem would be to perform a rendering process without knowing how many pixels we are calculating beforehand.

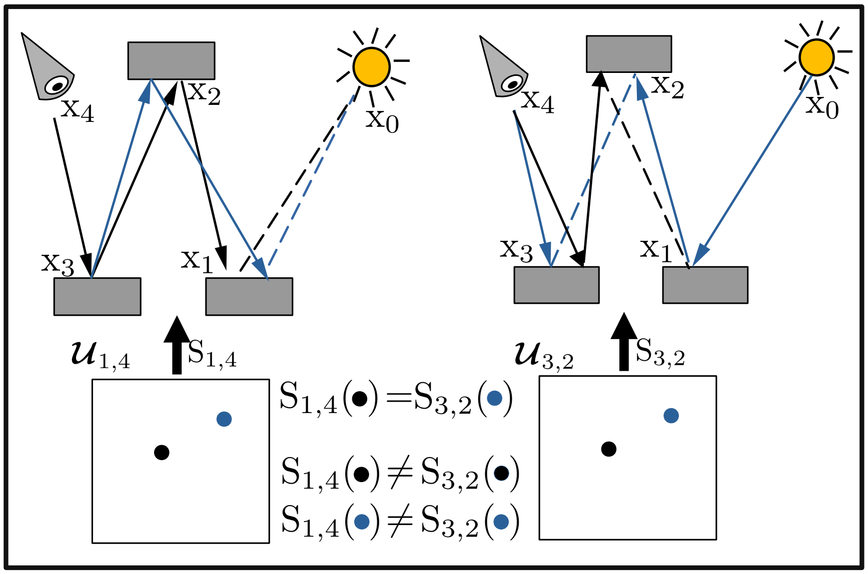


Figure 3.7: The same vector in two different extended spaces induces different paths, which usually, differ greatly than waht is shown in this figure, decreasing the acceptance rate of the chain. Note that the black path is the "correct" $\mathbf{X}$ for the $(1,4)$-technique, and that the blue path is the same "correct"" $\mathbf{X}$ path, for the $(3,2)$-technique.

initialization samples, like in Equation 2.106

$$\begin{aligned}\mathcal{B} = [&[f^*(\mathbf{U}_{0,d_0}, d_0), f^*(\mathbf{U}_{0,d_1}, d_1), ..., f^*(\mathbf{U}_{0,d_{m_d}}, d_{m_d})]\\ &[f^*(\mathbf{U}_{1,d_0}, d_0), f^*(\mathbf{U}_{1,d_1}, d_1), ..., f^*(\mathbf{U}_{1,d_{m_d}}, d_{m_d})]\\ &\vdots\\ &[f^*(\mathbf{U}_{N_{\text{init}}-1,d_0}, d_0), f^*(\mathbf{U}_{N_{\text{init}}-1,d_1}, d_1), ..., f^*(\mathbf{U}_{N_{\text{init}}-1,d_{m_d}}, d_{m_d})]]\end{aligned} \qquad b = \frac{\text{sum}[\mathcal{B}]}{N_{\text{init}}}, \tag{3.26}$$

where if we sample an index of this array we can cleverly sample the depth d, the initial vector of random numbers $\mathbf{U}$, and the initial discrete state t at the same time. The sampled index i_0 of this arrays gives us all the desired information, as the index itself will be the RNG seed used to generate the sample $(\mathbf{U}_0, T_s)$ of the chain, and it also implicitly tells us the sampled depth $d = i_0 \mod m_d + 1$. The initial pdf of this array sampling is then

$$p_0(\mathbf{U}, d) = \frac{f^*(\mathbf{U}, T_s, d)}{b} = \frac{b_d}{b}\frac{f^*(\mathbf{U}, T_s, d)}{b_d} = p_{c_d} p_0(\mathbf{U}|d), \quad W = b, \tag{3.27}$$

which gives a combined MMLT estimator that takes into account all the chains of all the depths:

$$I_j \approx \frac{A_{\text{image}} b}{N}\left[\sum_{k=1}^{N} \frac{(d(\mathbf{U}_{k_p})+2)\alpha f_j(\mathbf{U}_{k_p}, T_{s_{k_p}})}{f^*(\mathbf{U}_{k_p}, T_{s_{k_p}})} + \sum_{k=1}^{N} \frac{(d(\mathbf{U}_{k_c})+2)(1-\alpha) f_j(\mathbf{U}_{k_c}, T_{s_{k_c}})}{f^*(\mathbf{U}_{k_c}, T_{s_{k_c}})}\right], \tag{3.28}$$

where $d(\mathbf{U})$ is the corresponding depth of the chain that is generating the sample $\mathbf{U}$, and T_{s_k} is the (s,t) strategy chosen by $\mathbf{U}$. The factor $d(\mathbf{U})+2$ is due to the fact that for a fixed depth d, there are $d+2$ integration domains as we are sampling the sum 3.21, so analogously to Equation 2.110, we are here visiting $d+2$ "bins" per depth. The original paper used a biased but consistent chain, with depth sampling $p_d = \frac{b_d}{b}$, which makes the original MMLT estimator:

$$I_j \approx \frac{bA_{\text{im}}}{N}\sum_{k=1}^{N}\left[\frac{(d(\mathbf{U}_{k_p})+2)(\alpha + \text{isLarge}) f_j(\mathbf{U}_{k_p}, T_{s_{k_p}}, d_{k_p})}{b_{d_k}[f^*(\mathbf{U}_{k_p}, T_{s_{k_p}}, d_{k_p})/b(\mathbf{U}_{k_p}) + p_{\text{L}}]} + \frac{(d(\mathbf{U}_{k_c})+2)(1-\alpha) f_j(\mathbf{U}_{k_c}, T_{s_{k_c}}, d_{k_c})}{b_{d_k}[f^*(\mathbf{U}_{k_c}, T_{s_{k_c}}, d_{k_c})/b(\mathbf{U}_{k_c}) + p_{\text{L}}]}\right] \tag{3.29}$$

where $b(\mathbf{U})$ is the normalization constant for the depth corresponding to the chain that generated the sample $\mathbf{U}$, $d_{k_c} = d(\mathbf{U}_{k_c}), d_{k_p} = d(\mathbf{U}_{k_p})$, and $b_{d_k} = b(\mathbf{U}_{k_p}) = b(\mathbf{U}_{k_c})$.

With respect of the coding, then the differences with respect to PSSMLT are:

1. The initialization array $\mathcal{B}$ is going to have N_{init} samples for all the depths, and the normalization constant b will be calculated as in Equation 3.26.

2. This initialization array will be used to sample both the initial state and the fixed depth that each chain will have.

3. Instead of using a function `L_PSSMLT`, a function `L_MMLT` will be used which return as importance Equation 3.22. The change of technique index will also be subjected to both large mutations and small mutations.

4. The usual splatting weight of PSSMLT will have an extra $d+2$ factor.

Despite of the performance improvement with respect to PSSMLT, one of the main disadvantages of MMLT is that a random vector $\mathbf{U}$ usually gives a completely different path (see Figure 3.7) if the technique index is changed

$$\mathbf{X} = S_{(s,t)}(\mathbf{U}) \neq S_{(s',t')}(\mathbf{U}) = \mathbf{Y}, \tag{3.30}$$

which induces a reduction in the acceptance rate of the mutations that change techniques. This can be fixed by defining a new type of mutation called *reversible jump mutation* which consists in finding a new vector of random numbers $\mathbf{V}$ that satisfies

$$S_{(s,t)}(\mathbf{U}) = S_{(s',t')}(\mathbf{V}) = \mathbf{X}, \tag{3.31}$$

the name of reversible comes from the fact that in order to find this vector an inversion $S^{-1}_{(s',t')}(S_{(s,t)}(\mathbf{U}))$ must be done, which is done by transforming directions and points into random numbers. The algorithm that performs reversible jumps mutations is called *Reversible Jump Metropolis Light Transport* [Bit+17].

Another technique that shares similarities with the ability that MMLT has of jumping between states is called *serial tempering*, or *replica exchange* [KKK09] , which consists in generating several Markov Chains C_i, each one with a different importance function $I^{C_i}(\mathbf{U}^{C_i})$, where $\mathbf{U}^{C_i}$ is the current state of the chain, then we can define a global vector $\hat{\mathbf{U}} = (\mathbf{U}^{C_1}, ..., \mathbf{U}^{C_N})$ that encapsulates the current global state of all the chains, obtaining in this way a global importance function

$$\hat{I}(\hat{\mathbf{U}}) = \prod_{k=1}^{C_N} I^{C_k}(\mathbf{U}^{C_k}), \tag{3.32}$$

which allows us to change the state between two chains C_i and C_j, with index $j > i$ as:

$$r' = \frac{I^{C_1}(\mathbf{U}^{C_1})I^{C_2}(\mathbf{U}^{C_2})...I^{C_i}(\mathbf{U}^{C_j})...I^{C_j}(\mathbf{U}^{C_i})...I^{C_N}(\mathbf{U}^{C_N})}{I^{C_1}(\mathbf{U}^{C_1})I^{C_2}(\mathbf{U}^{C_2})...I^{C_i}(\mathbf{U}^{C_i})...I^{C_j}(\mathbf{U}^{C_j})...I^{C_N}(\mathbf{U}^{C_N})} = \frac{I^{C_i}(\mathbf{U}^{C_j})I^{C_j}(\mathbf{U}^{C_i})}{I^{C_i}(\mathbf{U}^{C_i})I^{C_j}(\mathbf{U}^{C_j})}. \tag{3.33}$$

then the exchanging rate is $r = \min(1, r')$. Exchanging the states between two chains helps the chain to explore several domains of the space using different importance functions, which obviously improves the exploration as each chain explores the same domain in a fundamentally different way . One special case arises from considering one chain C_i as two chains: one large mutation chain with importance $I^{C_i,L}$, and one small mutation chain with importance $I^{C_i,s}$, as both of these chains have the same state their exchange will always be accepted, this corresponds to the stochastic process of choosing a mutation.

3.3 Stratification

The MCMC based rendering algorithms have a *stratification* problem in the raster space coordinates, as brighter pixels will receive more samples. This is because each pixel receives samples proportional to the value I_j of the pixel, making the algorithm much less efficient in exploring potentially difficult paths in darker pixels. Causing an uneven convergence of the image, which is visually unappealing as the human eye is highly sensible to contrast differences.

One of the first ways in which this problem was solved was done by doing the following thought experiment: If we had all the measurements I_j of the image beforehand we could use this measurements in a new importance function defined as $I(\mathbf{U}) = \frac{f^*(\mathbf{U})}{I_j}$, where I_j is the measurement (that we want to calculate ...) of the j pixel that encloses $\mathbf{U}$. This clearly would induce stratification on the image as we can intuitively see that if a pixel was too bright, the ratio of the importance function would be compensated by this high brightness in the denominator, as opposed to a dark pixel whose value would tend to increase (or at least to not decrease too much) this ratio[6]. But this importance function is merely hypothetical as we obviously don't have the final rendered image yet, so one of the first stratification algorithm consisted in estimating the value of the average measurement of the pixels, by running several sets of Markov Chains in multiple stages (iterations), where in each iteration a new group of Markov chains with a higher degree of stratification is obtained by estimating an increasing refined image of measurements with all the samples of all the previous stages [HH10].

(a) `Veach-ajar` reference. The light is behind the wall coming inside the room from the door's aperture.

(b) Heat map showing the amount of samples received by each pixel. The samples are clamped in the range $0 - 300$.

Figure 3.8: The `Veach-ajar` scene rendered using the MMLT integrator. We can see that more samples go to regions near the door.

Another technique of stratification is called *Stratified MCMC Light Transport* [GWH20], this algorithm instead of using M Markov chains (samplers) that can visit any raster position of the final image, it uses A_{image} chains, one chain per pixel that is confined to visit just some predefined neighbor of the pixel, this pixel's neighborhood is called a *tile*[7], the tile neighborhood of a pixel $j = (j_x, j_y)$ is defined as the pixel itself, and its left, right, up, and bottom pixel neighbors:

$$\mathcal{N}_j = \{(j_x, j_y), (j_x - 1, j_y), (j_x + 1, j_y), (j_x, j_y - 1), (j_x, j_y + 1)\}, \tag{3.34}$$

defining a "t-shaped" tile with five pixels, which means that every pixel $j \in \mathcal{N}_s$ in the tile centered at s receives a radiance estimation $\hat{\mathcal{I}}_s(j)$ from the s Markov chain alone of $N = \text{mpp}$ samples $(\mathbf{U}_1^s, \mathbf{U}_2^s, \mathbf{U}_3^s, ...\mathbf{U}_N^s)$ is

$$\hat{\mathcal{I}}_s(j) = \frac{1}{N} \sum_{i=1}^{N} \begin{cases} b_s \text{ if } S(\mathbf{U}_i^s) \text{ is in } j \\ 0 \text{ else} \end{cases} \qquad \pi_s(\mathbf{U}) = \frac{\sum_{j \in \mathcal{N}_s} h_j(S(\mathbf{U})) f^*(S(\mathbf{U}))}{b_s} \equiv \frac{T_s(\mathbf{U})}{b_s} \tag{3.35}$$

where the unscaled intensity $G_s(j)$ of the pixel j in tile s is defined as:

$$G_s(j) = \frac{1}{N} \sum_{i=1}^{N} \begin{cases} 1 \text{ if } S(\mathbf{U}_i^s) \text{ is in } j \\ 0 \text{ else} \end{cases} \qquad \hat{\mathcal{I}}_s(j) = b_s G_s(j). \tag{3.36}$$

As the tiles are overlapping, each pixel j is receiving multiple contribution from several tiles, so the set of tiles that can contribute radiance to pixel j is denoted as $\mathbb{N}(j)$, hence the final estimation of the radiance of pixel j by several Markov Chains is

$$\hat{\mathcal{I}}(j) = \frac{1}{|\mathbb{N}(j)|} \sum_{s \in \mathbb{N}(j)} G_s(j) b_s, \tag{3.37}$$

[6]For a formal mathematical proof of several stratification theorems in MCMC rendering, see the work by Zirr and Dachsbacher [ZD20]

[7]It could be defined in several different ways (shapes), even using non contiguous pixels.

furthermore, all the pairs of tiles that contain pixel j will be denoted as $\mathbb{M}(j) = \{(s,t) | s \in \mathbb{N}_j, t \in \mathbb{N}_j, s \neq t\}$.

All the previous discussion is under the assumption that the chains are already initialized to an initial random number vector $\mathbf{U}_0$, there are two possible ways to initialize these chains:

1. Naively run all the chains a certain amount of initial samples per pixel, hoping that the chain gets initialized with a non zero radiance value. In the case that some of the chains don't get initialized we simply start the SMCMC algorithm for all the initialized tiles, and for the not initialized ones, instead of advancing the Markov Chain like in the initialized tiles case, we simply keep trying to obtain a non zero importance of the tile. Notice that this way of initialization induces start-up bias as the chains are far from having converged to their stationary distributions, which makes the algorithm biased but consistent.

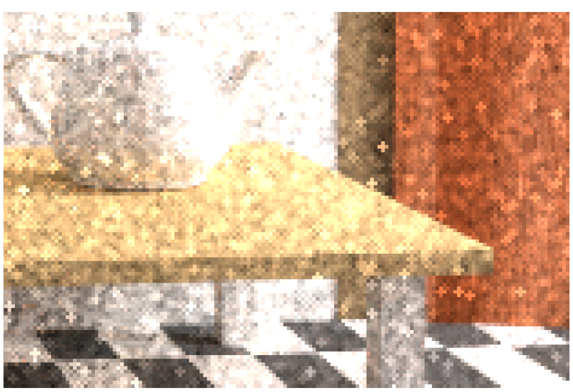
2. Run an amount t_h of global chains equal to the number of available threads in the computing system. Each of these global chains are going to explore the whole image for a burn in period of 100k iterations per chain to get the global chains closer to their global stationary distribution. After this, the t_h chains are run for some small amount of iterations, for example, 4 mutations per pixel ($4A_{\text{image}}/t_h$ iterations per chain). The global Markov Chain will give its current state of the chain to each non initialized tile of each pixel that it visits. In the case in which the tile is already initialized, the global chain will exchange its state with the Markov Chain of the tile using Replica Exchange.

Given that all the chains s are confined within their corresponding tile, there would be A_{image} normalization constants b_s, instead of just one for the whole image as usual, which at first it might seem counter-intuitive, as we would need to estimate an integral for each pixel, but we can take advantage of the overlapping between tiles, noting that the same pixel must converge to the same solution regardless of tile, thus, we are able to treat this problem as the optimization problem of minimizing the difference between overlapping regions, and the difference between the calculated value $\hat{\mathcal{I}}$ and an estimation $\hat{I}$ of the real value, defining in this way the loss function

$$\mathcal{L} = \alpha \left\| \sum_j \hat{\mathcal{I}}(j) - \hat{I}(j) \right\|^2 + \sum_j \sum_{(s,t)\in\mathbb{M}_j} \left\| \hat{\mathcal{I}}_s(j) - \hat{\mathcal{I}}_t(j) \right\|^2 \tag{3.38}$$

where the estimation of the true value $\hat{I}(j)$ can easily be calculated by using the large step proposals of the Markov chains. One way of solving the minimization of the loss function is to use a Jacobi solver to iteratively update the b_s^n constants, with initialization $b_s^0 = \hat{I}(s)$. The process of reconstructing the image by solving the optimization problem of the loss function $\mathcal{L}$ is called *tile aligning* (see Figure 3.9).

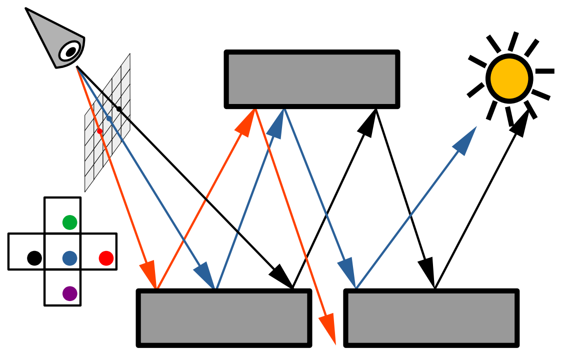
Notice something important, the Markov Chain of the tile s is producing the usual chain of samples $(\mathbf{U}_1^s, \mathbf{U}_2^s, \mathbf{U}_3^s, ...\mathbf{U}_N^s)$, so for each sample one out of the five pixels of the tile must be chosen randomly



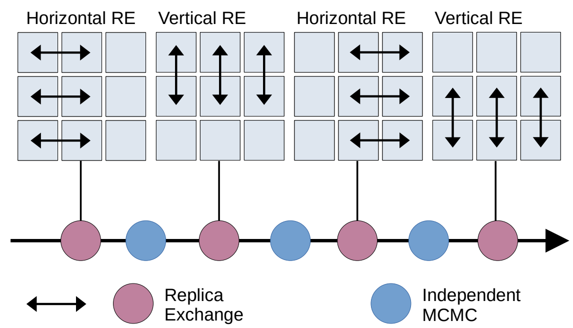
(a) Image without reconstruction, we can notice the t-shaped tiles form in the image.

(b) Reconstructed image, tile aligning.

Figure 3.9: Tile aligning zoom in of render.



(a) Random replay shift map consists in repeating the same random number of the main path **U** but in all the offsets pixels. Shift maps in general are an essential part of gradient domain rendering.



(b) SMCMC can be improved by alternating between RE steps and independent MCMC steps.

Figure 3.10: Replica Exchange and shift mapping are the soul of SMCMC. Diagrams adapted from the SMCMC paper [GWH20] to this thesis' style.

on each iteration as a base pixel or main pixel from which the radiance is going to be estimated. This estimated radiance must be multiplied by five as the probability of choosing that main pixel is $p_m = \frac{1}{5}$. There is a way to improve this by estimating the radiance of the five pixels at the same time per iteration. This can be achieved by randomly selecting on each iteration any pixel in the tile as the main pixel p, and using the current random number $\mathbf{U}$ of the chain we will obtain a main path $\mathbf{X} = S(\mathbf{U})$. This main path can *deterministically* be mapped into all the offset pixels with the transformation $T_{pq}(\mathbf{X})$, which maps the main path $\mathbf{X}$ at pixel p, into an offset path $\mathbf{Y}$ in the pixel q, this transformation is known as a *shift map* [Hua+19; GWH20] (see Figure 3.10a).

Using shift maps the measurement I_q^p on the neighboring pixel q of p can be calculated as

$$I_q^p = \int h_q(T_{pq}(\mathbf{x}))f(T_{pq}(\mathbf{x}))\left|\frac{\mathrm{d}T_{pq}(\mathbf{x})}{\mathrm{d}\mu(\mathbf{x})}\right|\mathrm{d}\mu(\mathbf{x}), \tag{3.39}$$

where $h_q(T_{pq}(\mathbf{x})) = h_p(\mathbf{x})$ if the displacement between pixels was done in integers. As the tile doesn't have anymore just one radiance estimation for one random chosen pixel, then the importance function of the tile is defined as just the sum of the estimated importance on each pixel of the tile.

The shift map used by SMCMC is a *random vector replay shift*, that consists in repeating the same random number of the main path $\mathbf{U}$ but in all the offsets pixels $T_{\text{main,off}_i}(\mathbf{U}) = \mathbf{V}_{\text{off}_i} = \mathbf{U}$, this transformation has the jacobian

$$\left|\frac{\mathrm{d}\mathbf{Y}}{\mathrm{d}\mathbf{X}}\right| = \left|\frac{\mathrm{d}\mathbf{Y}}{\mathrm{d}\mathbf{V}}\right|\left|\frac{\mathrm{d}\mathbf{V}}{\mathrm{d}\mathbf{U}}\right|\left|\frac{\mathrm{d}\mathbf{U}}{\mathrm{d}\mathbf{X}}\right| = \left(\frac{1}{p(T_{pq}(\mathbf{X}))}\right)(1)\,(p(\mathbf{X})) = \frac{p(\mathbf{X})}{p(T_{pq}(\mathbf{X}))}, \tag{3.40}$$

then the contribution of a MC estimator of the offset path would be

$$\frac{f(T_{pq}(\mathbf{x}))}{p(\mathbf{X})}\left|\frac{\mathrm{d}T_{pq}(\mathbf{x})}{\mathrm{d}\mu(\mathbf{x})}\right| = \frac{f(T_{pq}(\mathbf{x}))}{p(\mathbf{X})}\frac{p(\mathbf{X})}{p(T_{pq}(\mathbf{X}))} = \frac{f(T_{pq}(\mathbf{x}))}{p(T_{pq}(\mathbf{X}))}, \tag{3.41}$$

which ends up being equivalent as if we had obtained the path in the offset. Notice that the MC samples for the offset pixel q are being obtained transforming paths over the main pixel p to the offset with the function T_{pq}!

This random replay can be further improved by applying RE between both horizontal tile neighbors, and vertical tile neighbors, in a repeating alternating process (see Figure 3.10b), with probability of accepting the swap for chains s and t with states $\mathbf{U}^s, \mathbf{U}^t$ equal to:

$$r(\mathbf{U}^s, \mathbf{U}^t) = \min\left(1, \frac{T_s(\mathbf{U}^t)T_t(\mathbf{U}^s)}{T_s(\mathbf{U}^s)T_t(\mathbf{U}^t)}\right). \tag{3.42}$$

The SMCMC algorithm presented by Gruson et al. [GWH20] had four different combinations of techniques as it had only the Random Replay (RR) shift, and the possibility of performing RE:

- (RE:Off, Sh:Off): We will sometimes call it "None", as in no shift is being applied.
- (RE:On, Sh:Off): We will sometimes call it RE + None.
- (RE:Off, Sh:RR): We will sometimes call it RR.
- (RE:On, Sh:RR): We will sometimes call it RE + RR.

An example of a comparison between some of these technique is given in Figure 3.11

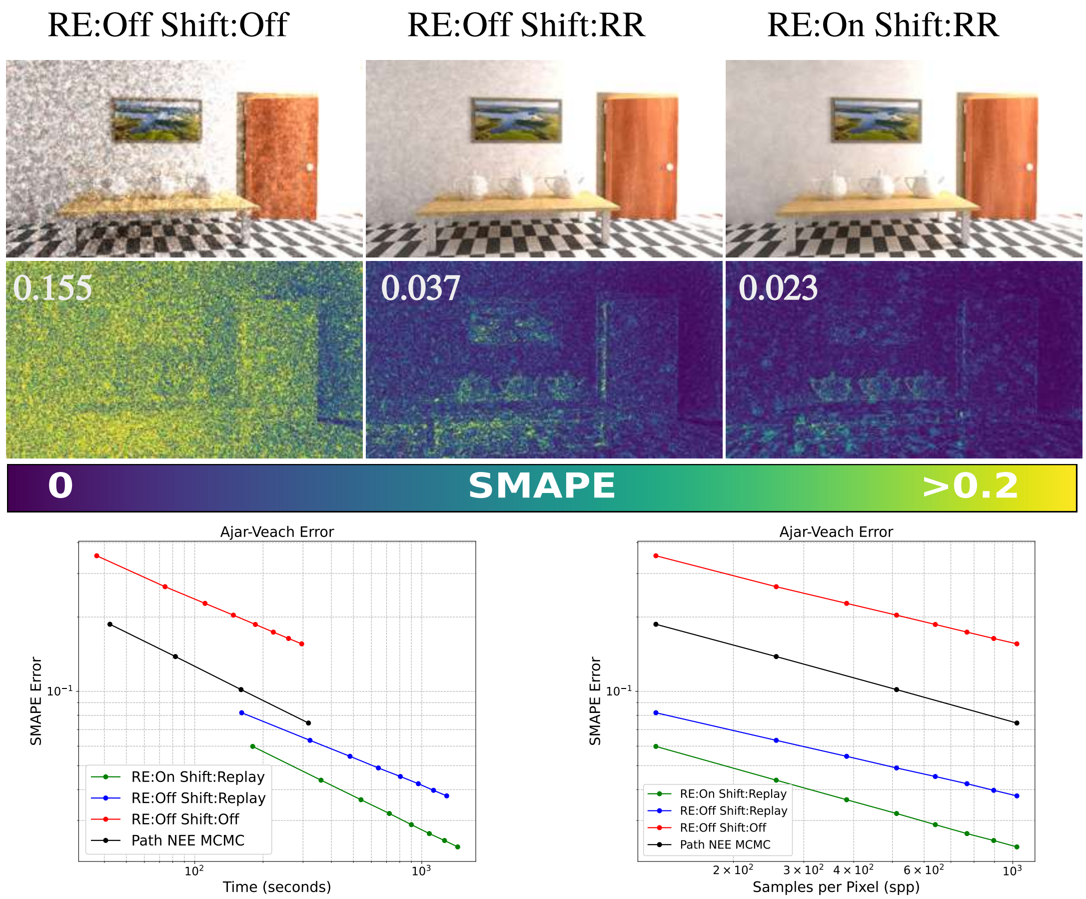


Figure 3.11: SMAPE error of SMCMC. We will present the results of our modifications to SMCMC in the next chapter like this. The three images of the first row are the resulting renders. The three images of the second row are a heat map of the SMAPE errors of each pixel, with a maximum clamp in this case of 0.2 (values ≥ 0.2 appear as yellow). The third row include the logarithmic plots of the SMAPE error plotted against time and against samples per pixel. We can observe that all the lines are straight, typical of consistent and unbiased MC estimators. When we say that an algorithm is more time-efficient than another algorithm we mean that it converges in less time to the correct solution, and analogously, a more sample-efficient algorithm converges in less samples to the correct solution. We can see that the less efficient line (the red one) corresponds to the worst technique (RE:Off, Sh:Off), while the green line corresponds with the most efficient technique (RE:On, Sh:RR) because its error is less in the same amount of time compared with the other techniques. To be more precise we will call the algorithm with the least error when considering time as the most *accurate* algorithm.

3.4 Chapter conclusions

In this chapter we presented the relevant Metropolis Light Transport theory that we will use to design our experiments.

We started by discussing path-space MLT [VG97], which has the disadvantage that it is hard to implement, and consequently, difficult to integrate and combine with other algorithms, and it also has a very correlated large step mutation, the bidirectional mutation, which reduces the global exploration in exchange of computational efficiency and the chance of a better acceptance probability.

We later talked about PSSMLT [Kel+02], a Metroplis-Hasting type algorithm that acts over the random numbers that were used to sample the paths in a usual (bidirectional) path tracer, this algorithm has the advantages that it is easy to implement while retaining efficiency, and that its large step mutation generates unrelated paths to the current path increasing the global exploration range in exchange of more computational effort, and a very likely reduction of the acceptance rate. Regarding its disadvantages, we are losing control over the path-space treating every mutation as a black box and for the case of a PSSMLT that uses a BDPT integrator each step of the Markov chain is slow due to the fact that all the MIS-weighted BDPT connections must be used on each iteration.

After this, MMLT [HKD14] was introduced, dividing the PSS in multiple spaces for each different BDPT (s, t) technique limiting all the Markov Chain to have a fixed length, with the main advantage being that now each iteration of the chain uses a single MIS-weighted BDPT connection and not all of them. Despite this improvement, one disadvantage was that when the BDPT (s, t) technique is changed by the chain, the probability of accepting this new path will likely be low as the new path will be topologically different. We briefly talked about how this could be avoided by defining a reversible jump mutation [Bit+17] to get random numbers that map to the same topology.

Last, but not least, we addressed the problem of stratification, by introducing the Stratified Markov chain Monte Carlo algorithm [GWH20] that uses one Markov chain per pixel, in order to get a more even convergence of the image space.

In the next chapter we will present our proposals and experiments.

Chapter 4

Experiments

Light transport research is usually done by adding the necessary functionalities and algorithms into the code of a well-established rendering system, such as Mitsuba [Jak10] or pbrt [PJH16], in order to be able to test out the research algorithms, methods and hypothesis. In this thesis we obtained all of our results using pbrt version 3.0 [PJH16], which we modified and extended with the implementations needed to run our experiments.

We want all of our resulting estimated renders to be *unbiased*, which means that the stochastic expected value of the estimator is the real solution, and we also want our results to be *consistent*, which means that we are guaranteed to find the actual solution if we increase the number of samples.

To check the correctness (unbiasedness and consistency) of all the images I we obtained with our implemented experiments we need to obtain a *reference image* R. Image R should represent a fully converged render, and synthesizing it is extremely costly depending on the available hardware.

All of the reference images R in this thesis were obtained using standard unbiased and consistent integrators already implemented in the rendering system (in our case pbrt 3.0 [PJH16]), using a varying amount of samples depending of the difficulty of the scene. To obtain the references we used one node of the computer cluster of the Electrical Engineering Postgraduate Division of the Universidad Michoacana de San Nicolás de Hidalgo. This node with 100 GB of RAM has 128 threads (S:C:T=2:28:2, i.e., 2 sockets, 28 cores per socket, 2 threads per core), and each socket has a speed range of 1.2-1.7 GHz. The NEE-PT integrator of pbrt 3.0 with 50000 samples per pixel using this node takes between 4 and 6 hours, depending of the complexity of the scene, even spending about 4 days for a single reference.

All the algorithms, implementations and experiments we did were generated using a local AMD Ryzen 5 5500U CPU, with 6 cores and 12 threads, and a speed range 0.4-4 GHz. The RAM has a capacity of 16 GB.

We tested the correctness of our resultant image I by doing the following two things.

1. **Consistency check**: We obtained the graph of the logarithmic amount of samples as a function of the logarithmic error (for some given error metric), that for a consistent and unbiased MC estimator should be linear (see Figure 4.1).

   To calculate the error of pixel j we considered the scalar $I_j^* = \sqrt{r_j^2 + g_j^2 + b_j^2}$, where (r_j, g_j, b_j) are its RGB values. In this thesis we used most of the time the symmetric mean absolute percentage error (SMAPE) and just few times the mean square error (MSE), defined for images as:

$$\text{SMAPE}[I] = \frac{1}{A_{\text{image}}} \sum_j^{A_{\text{image}}} \frac{|I_j^* - R_j^*|}{|I_j^*| + |R_j^*| + 1 \times 10^{-8}}; \quad \text{MSE}[I] = \frac{1}{A_{\text{image}}} \sum_j^{A_{\text{image}}} (I_j^* - R_j^*)^2, \quad (4.1)$$

   with I_j^* the scalar value of pixel j of the tested method image, R_j^* the scalar value of pixel j of the reference image, and A_{image} are the total amount of pixels.

   We used more the SMAPE error because this type of error punishes more severely differences in contrast rather than differences in brightness. To see this consider the scalar value $(R_j^* = 10, R_k^* = 100)$ of two hypothetical pixels j, k, and an estimator that returns values $(I_j^* = 9, I_k^* = 90)$, the SMAPE error is the same $(1/19, 10/190)$ while the MSE error $(1, 100)$ is not. The SMAPE error only cares about the proportional difference to the brightness, and not about the brightness itself, which makes it in a way, a more localized error metric than the usual global MSE error metric. The tracking of this localized metric is useful in MCMC algorithms due to the fact that most of the MCMC samples go to brighter regions, decreasing the error in brighter pixels but increasing it in darker pixels [GWH20].

2. **Unbiasedness check**: The histogram of the differences $I - R$ for all the pixels should have a zero mean, which implies that there are around the same amount of pixels brighter than the reference $I - R > 0$ than there are pixels darker than the reference $I - R < 0$ (see Figure 4.2).

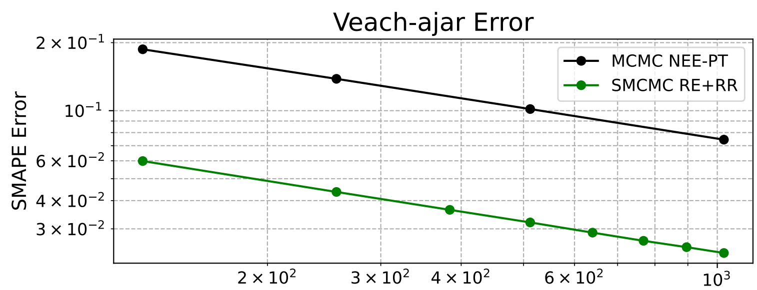
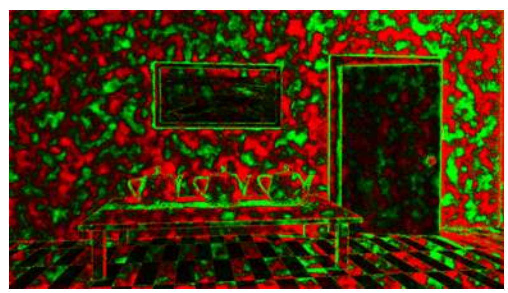
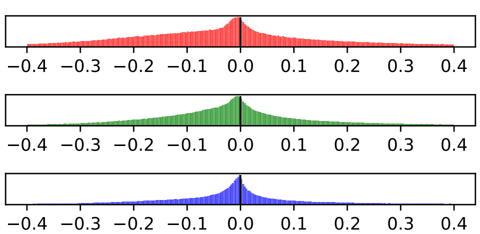


Figure 4.1: A Monte Carlo estimator with a logarithmic scaling in both axes will be a straight line with a varying slope that depends on the efficiency of the algorithm. The green line is a more efficient algorithm for this scene.

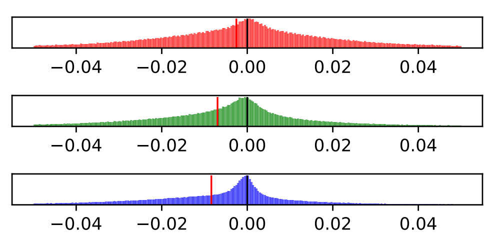
(a) Unbiased render: All the regions of the image have the same amount of red samples than green samples.



(b) Biased render: There are zones where red or green spots accumulates. In this case notice the edges.



(c) Histogram of unbiased render with mean values for each channel (-1.9×10^{-4}, -2.5×10^{-4}, -2.4×10^{-4})



(d) Histogram of biased render with mean values for each channel (-2.5×10^{-3}, -6.9×10^{-3}, -8.3×10^{-3}). The calculated mean (red line) does not correspond with the zero valued desired mean.

Figure 4.2: The red green distribution of an image is obtained by painting green the positive pixels $I - R > 0$, red the negative pixels $I - R < 0$, and black the approximately equal pixels $I = R$. The mean of the histogram that is induced by this difference distribution should always be zero. We didn't show the histograms in the subsequent experiments because we always discarded biased experiments.

Scenes We present the list of scenes that we used for our experiments, alongside their images. All of our scenes were taken from Benedikt Bitterli's rendering resources [Bit16]. We modified the materials and some of the shapes, and adapted them to the rendering system pbrt version 3.0 [PJH16].

- `Veach-ajar`: Scene with only diffuse materials, but with difficult light transport because the room is only being illuminated by the light coming from the door that was left ajar (see Figure 4.3).
- `Bedroom`: Scene with the light entering the bedroom through open windows, but with tanslucent curtains that almost entirely block the window. This scene has some dielectric materials like the chandelier in the roof, or the glasses and vases in the dressers. Dielectrics induce small islands of difficult transport in the path-space of light paths, that are complicated to sample (see Figure 4.4).
- `Kitchen`: Scene with light coming from both inside (microwave/radio region on the table) and outside. Light coming from outside has to first pass through a glass (dielectric) window. This scene has some dielectric materials like the glass cups at the dining table, the glass of the stove's oven, and the glass of the microwave (see Figure 4.5).
- `Staircase`: Scene with several lights coming from several places, but with the slight difficulty of having an almost specular floor (see Figure 4.6).

Figure 4.3: `Veach-ajar` scene. It has only diffuse materials, but it is still a complicated scene due to the fact that the light is on the other side of the wall. Image rendered using one million spp of Path Tracing with Next Event Estimation (NEE-PT).

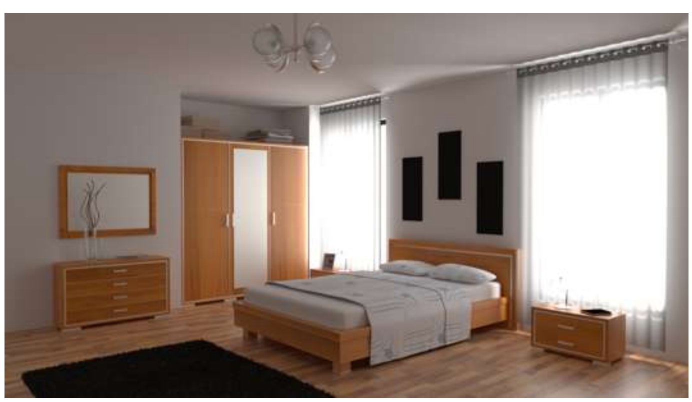


Figure 4.4: `Bedroom` scene. The light comes from outside the open windows, and passes through the translucent curtains. There are some dielectric materials in the scene (see the dressers and the chandelier). Image rendered using 600000 spp of Path Tracing with Next Event Estimation (NEE-PT).

Figure 4.5: `Kitchen` scene. The light comes from both inside (near the radio area on the table) and outside. The windows are closed. The glass cups, the oven's glass and the microwave's glass are dielectric materials. Image rendered using 900000 spp of Path Tracing with Next Event Estimation (NEE-PT).

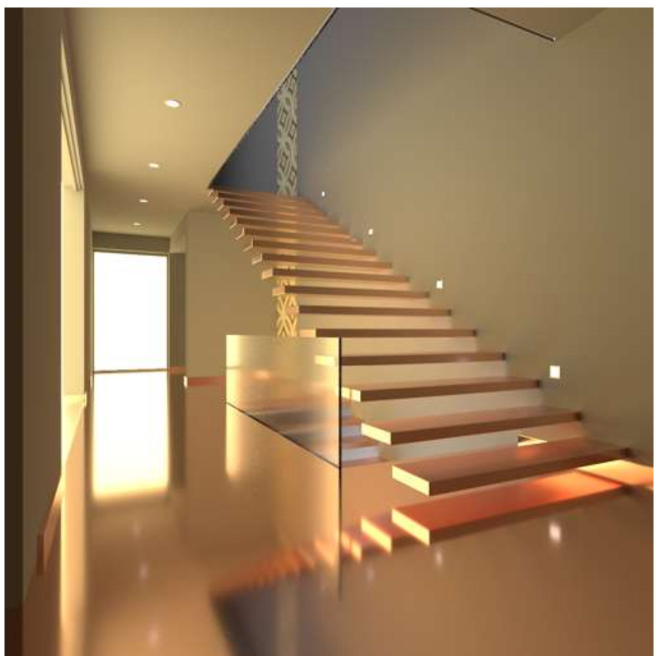


Figure 4.6: `Staircase` scene. A well illuminated scene. The slight difficulty is that the floor is near specular, and it also has a glass. Image rendered using 300000 spp of Path Tracing with Next Event Estimation (NEE-PT).

Chapter outline The two main experiments of this chapter are the combination of local exploration PSSMLT with global exploration and the addition of more complex path-space operator to SMCMC. In particular, section 4.1 shows results of combining MCMC and Path Guiding:

- **Pre-trained MCMC + PG**: In an initialization process we train a path guiding SD tree for a certain amount of samples. After this we will use this pre-trained tree in order to aid the paths of the MCMC algorithm.
- **Online learning MCMC + PG**: The combination of local and global exploration training the tree with the paths traced by the Markov Chain itself.

while section 4.2 shows the results of adding more complex path-space operators to SMCMC:

- **SMCMC BDPT**: The addition of a BDPT integrator to the usual NEE-PT of SMCMC.
- **SMCMC Random Strategy BDPT**: Selecting stochastically one of the (s, t) technique per each light pat depth of BDPT in order to avoid doing several visibility test. We will denote this technique as "rBDPT", that is, random (strategy) BDPT.
- **SMCMC MMLT**: Applying an MMLT style mutation by introducing two stochastic processes per each iteration of the Markov Chain, a stochastic selection of the depth and a stochastic selection of the (s, t) BDPT technique.
- **Guiding**: Using a path guiding tree in the SMCMC algorithm.

4.1 Global exploration

As we have already discussed, PSSMLT uses large mutations to globally explore the path-space, hoping to find a high radiance carrying light path by mere chance, which if it's stochastically accepted, will be subsequently used to explore similar paths by applying small mutations. We propose to use path guiding to increase the efficiency of the global exploration made by the large mutations of the MCMC algorithm. We believe that if the exploration of the path-space domain is done with the aid of a SD tree, this would increase both the probability of finding high importance paths and the acceptance rate of large mutations. We hypothesize that this increased acceptance rate of large mutations will improve the speed of convergence of the MCMC algorithm.

Experimental setup Path guiding has several user defined parameters. This algorithm has had several improvement over the years [Vor+19], but we used the original paper [MGN17] parameters:

- We chose spatio-directional tree samples (instead of material samples) with probability $q_{\mathrm{Tr}} = 0.5$. In modern versions of the algorithm q_{Tr} is obtained using an stochastic-gradient-based Adam optimizer.
- On each iteration k we refined the kd-tree if more than $c\sqrt{2^k}$ samples were registered in the leaf. We, like in the original paper, used $c = 12000$.
- The ratio of the total flux of a directional quadtree Φ to the total flux of a node Φ_n, should always be less than certain value ρ, that is, $\Phi_n/\Phi < \rho$. If this was not the case we performed a subdivision of the leaf. Like in the original paper, we used $\rho = 0.01$
- The directional and the spatial splatting of the samples into the tree could be done applying filters, but like in the original paper, we just splatted into the nearest (to the position or the direction) node.
- We did *not* using Next Event Estimation (explicit path tracing).
- We only stored the rendered image of the last iteration. In modern improvements of the algorithms, the images of all the iterations could be combined using a variance based weighting scheme.

We always used the above hyperparameters in all our guiding trees, regardless of the type of experiment.

Even without all the modern optimizations this algorithm is highly competitive and it fits our purposes. We believe this combination should improve global exploration.

Pre-trained distribution To simplify the analysis of the combination MCMC-PG, we did a pre-trained version of our proposed technique. We defined some amount of samples per pixel spp, and we simply used the computational effort of the 12.5% (1/8) of these samples to train the tree, i.e., $N_{\text{train}} = A_{\text{image}}\text{spp}/8$. These N_{train} samples were exclusively used for training, and they didn't contribute anything to the final rendered image. The other 87.5% (7/8) of the computational effort $N_{\text{MCMC}} = 7A_{\text{image}}\text{spp}/8$ was allocated to the MCMC algorithm that used this pre-trained tree (with N_{train} samples). For example, if we wanted to use a total budget of around 4096 spp, we pre-trained our tree with 512 spp. The remaining 3584 were used to generate our render. We also used an arbitrary number of N_{init} samples for the MCMC bootstrap process.

The acceptance rate α of the MCMC algorithm must be modified accordingly. Remembering Equation 2.137, we have that the estimated value of a path guided traced path $\mathbf{X} = (\mathbf{x}_0, \mathbf{x}_1, ..., \mathbf{x}_k)$ is

$$\frac{f_{\text{PG}}(\mathbf{X})}{p_{\text{PG}}(\mathbf{X})} = f_{\text{PG}}(S(\mathbf{U})) = \left(\prod_{j=1}^{k-1} w_c(\boldsymbol{\omega}_{i_j}) \frac{f(\mathbf{x}_j, \boldsymbol{\omega}_{o_j}, \boldsymbol{\omega}_{i_j})}{q_c p_c(\boldsymbol{\omega}_{i_j})}\right) L_e(\mathbf{x}_k, -\boldsymbol{\omega}_{i_{k-1}}) = \text{Thr}(\mathbf{X}) L_e(\mathbf{x}_k, -\boldsymbol{\omega}_{i_{k-1}}), \tag{4.2}$$

where q_c is the probability of choosing the selected technique (either material sampling or tree sampling), and w_c is its MIS weight. The acceptance rate of our MCMC-PG algorithm considering $\mathbf{V} = \text{mut}(\mathbf{U})$ becomes then

$$\alpha(\mathbf{U} \to \mathbf{V}) = \min\left(1, \frac{f_{\text{PG}}(S(\mathbf{V})) I_{\text{PG}}(S(\mathbf{U}))}{f_{\text{PG}}(S(\mathbf{U})) I_{\text{PG}}(S(\mathbf{V}))}\right). \tag{4.3}$$

for a simple overview see algorithm 8.

Algorithm 8 Pre-trained MCMC-PG.

Input: spp
Output: Rendered image.
1: $N_{\text{train}} \leftarrow A_{\text{image}}\text{spp}/8$
2: $N_{\text{MCMC}} \leftarrow 7A_{\text{image}}\text{spp}/8$
3: TrainTrees(N_{train},B_{T_S}, B_{T_B}) ▷ Train the tree like in PG algorithm 5.
4: $(\mathcal{B}, b) \leftarrow$ BootstrapStage(N_{init}) ▷ Array $\mathcal{B}$ to sample initial state $\mathbf{U}_0$
5: MCMC(N_{MCMC}, B_{T_S}, B_{T_B}) ▷ Run the Markov Chains tracing the paths with the trained tree.

The acceptance rate of the MCMC mutations can be found in Table 4.1 and we can conclude that for the scenes in which guiding is efficient, the large mutation acceptance rate did increase. Scenes with more (almost) specular materials such as `Kitchen` and `Staircase` suffered a reduction in the acceptance rate because MCMC handles better this type of vertices due to the fact that path guiding might suggest impossible scattering directions more often for these vertices.

	`Veach-ajar`		`Bedroom`	
Acc. rate	**MCMC**	**MCMC-PG (P)**	**MCMC**	**MCMC-PG (P)**
Large	0.003	**0.06**	0.054	**0.085**
Small	0.31	**0.35**	**0.53**	0.52
	`Kitchen`		`Staircase`	
Acc. rate	**MCMC**	**MCMC-PG (P)**	**MCMC**	**MCMC-PG (P)**
Large	**0.039**	0.029	**0.15**	0.12
Small	**0.65**	0.63	**0.73**	0.69

Table 4.1: The acceptance rate of the large mutation (global exploration) increases when the path guiding algorithm is efficient, which is when there are not (almost) specular materials and caustic paths. For the `Kitchen` and `Staircase` scene the acceptance rate was decreased when we used MCMC-PG because they have many (almost) specular vertices. This decrease happened due to the fact that MCMC alone is better at exploring these types of paths because path guiding often suggests directions that are not possible scattering directions, reducing the acceptance rate compared with MCMC alone. Another reason for this observed decrease is that the SD tree was only pre-trained using 512 spp, so it could have been trained further.

In Figures 4.7, 4.8 and 4.9, we show the error results for this experiment. We can conclude that it does indeed increase a little bit the efficiency of MCMC for scenes where there are a small amount of (almost) specular materials. But for scenes like `Staircase`, with a high amount of almost specular vertices the efficiency of MCMC was reduced. Unfortunately, the MCMC-PG combination does not improve over standalone path guiding for scenes in which PG is more efficient than MCMC.

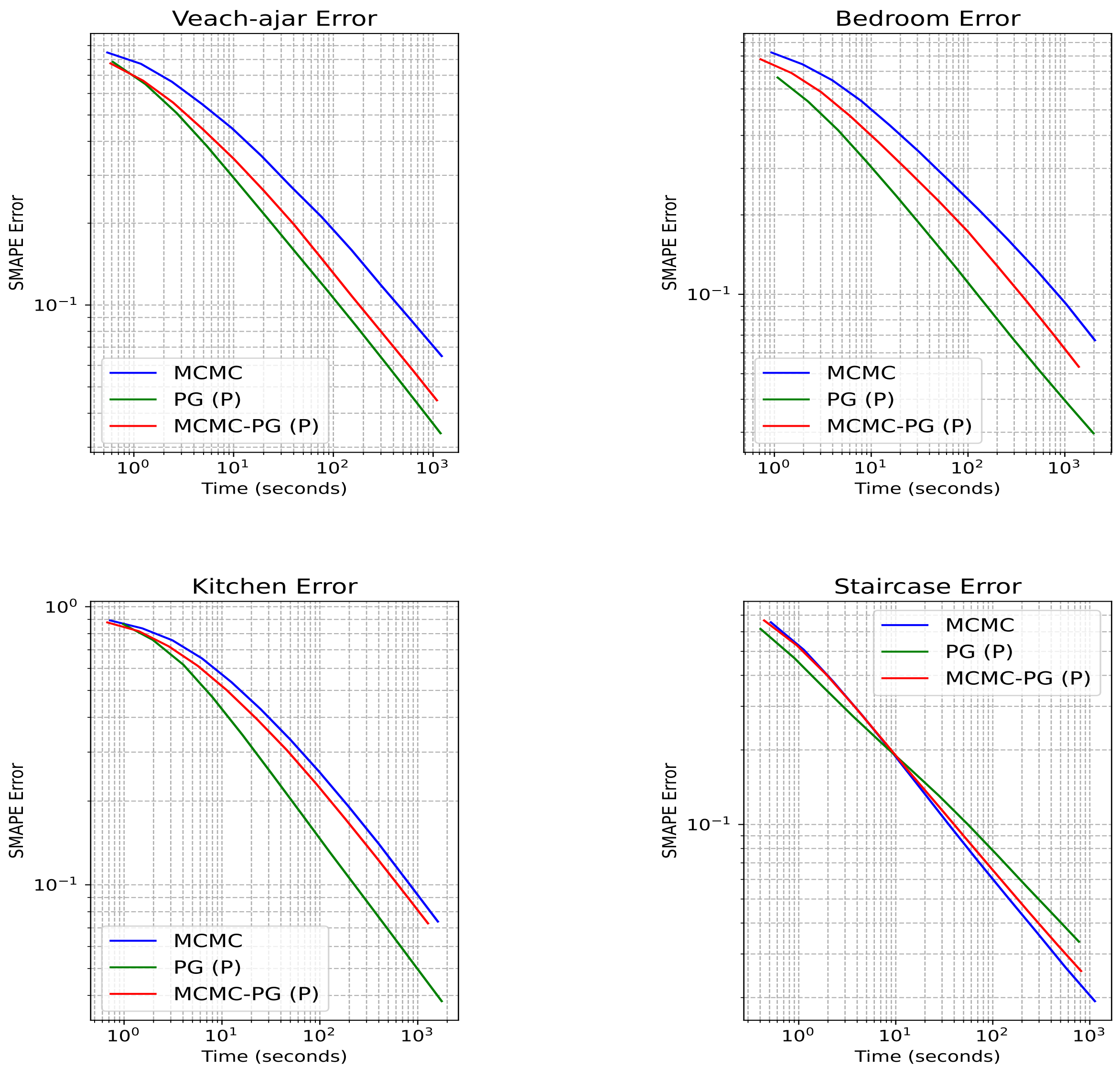


Figure 4.7: The pre-training (P) time is not being taking into account for these plots. MCMC-PG (P) is more efficient than MCMC alone, for scenes with a small amount of (almost) specular vertices (e.g., `Veach-ajar`, `Bedroom` and `Kitchen`), but for the same type of scenes, less efficient than PG (P) alone. For scenes with several (almost) specular materials MCMC-PG (P) is more efficient than PG (P), but less efficient than MCMC. We used a pre-trained tree (instead of an online-trained tree) with also 512 spp of pre-training for the PG (P) algorithm, in order to make a fair comparison between techniques.

Notice that the beginning of the above plots does not look like a straight line because the variance is too high when there a too few samples. Also observe that in the fourth graph (`Staircase` plot) PG (P) appears to be better for the first few seconds until the plots intersect in the range $10 - 20$ seconds, this can be explained because the high variance of the initial process and due to the fact that `Staircase` has a big region of the image that is light directly seen by the camera without needing a single bounce (the "white door") and PG (P) calculates this region perfectly as it is a stratified algorithm that only returns the radiance value of a light directly seen by the camera, while both MCMC and MCMC-PG (P) have a bigger error in this region as each sample has to be distributed in the whole film, so we need more samples to reduce this very bright radiance zone which induces a relatively high error. In a way, for this scene, it is as if PG (P) alone knew already the answer for that region of the scene while the other two algorithms not.

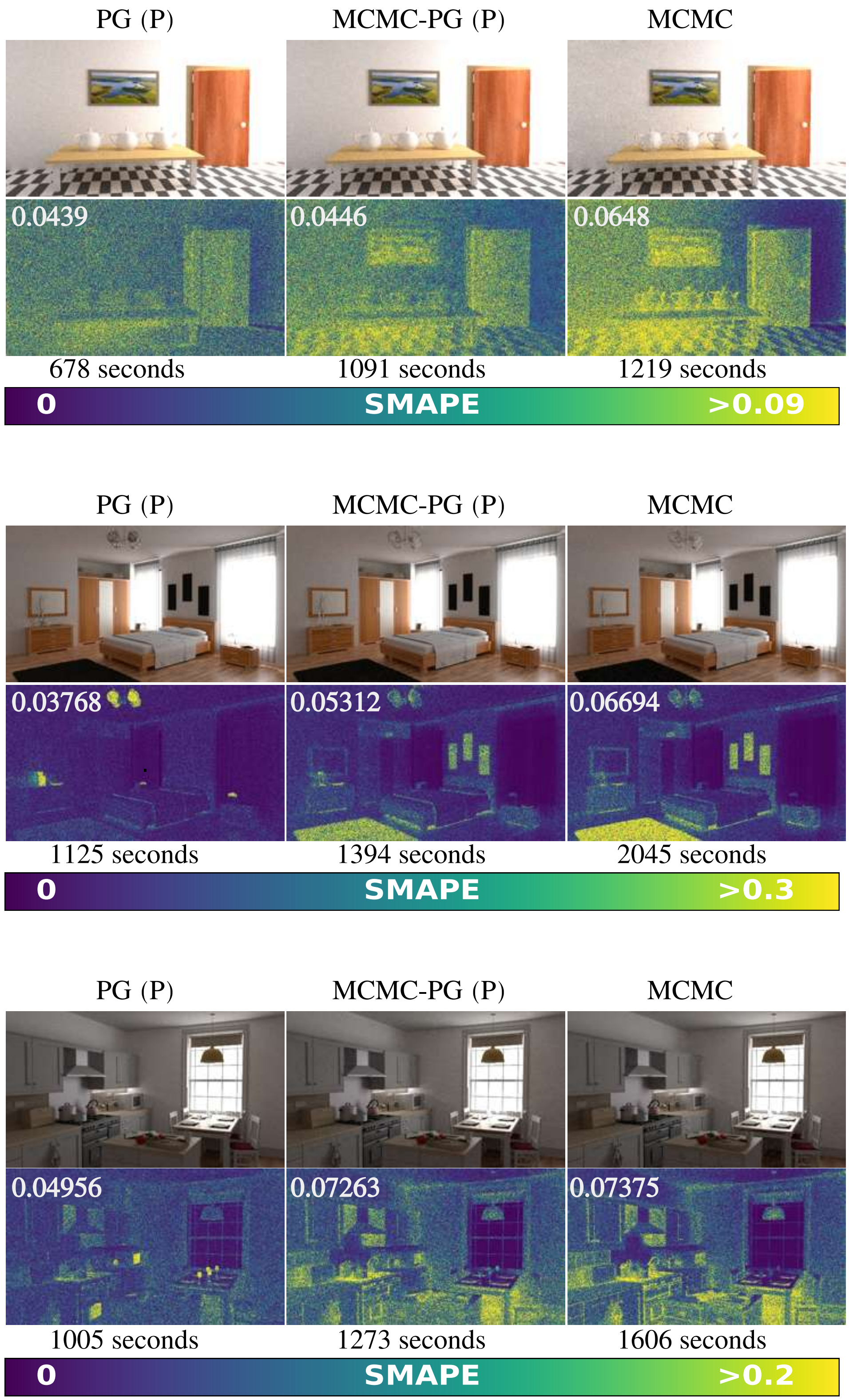


Figure 4.8: MCMC-PG (P) explores more efficiently than MCMC regions in which most of the guided direction by the SD tree are allowed by the scattering directions defined by the materials, but PG (P) is still more efficient for this type of regions. MCMC-PG (P) is also more efficient than PG (P) for exploring (almost) specular materials, but less efficient than MCMC.

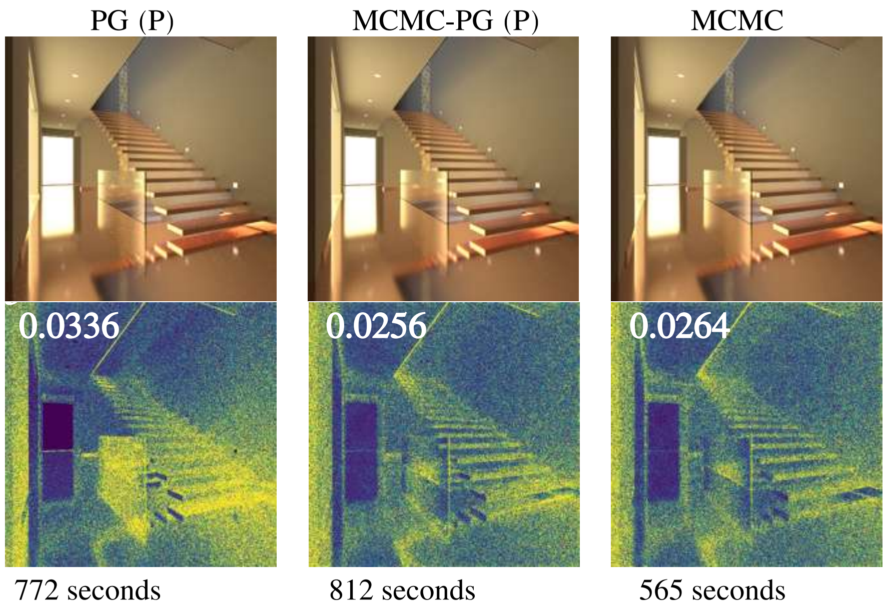


Figure 4.9: Notice how the glass railing has more problems with PG (P) alone. The floor despite being near specular it can be appropriately rendered by PG (P) and MCMC (PG) because the light transport on the floor is simple in this scene. A path tracer (without NEE) is able to render it fairly quick, which means that lights are reached by mere chance very often.

Online learning Now we want to train the trees with the MCMC proposed mutations (regardless if they were accepted or rejected). The training could have been done using only large proposals, which could be treated as "independent" samples from the MCMC process. It could also have been done using all the proposals, even if they were small. We used both type of mutations to train our trees.

Our technique simply run the usual MCMC algorithm, but the path tracing function used by our MCMC-PG technique used the sampling tree B_{T_S} (if it was already built) to trace the paths. The radiance (without considering the MCMC acceptance rate α given in Equation 4.3) at each vertex of this sampled path was used to train the building tree B_{T_B}, which was refined later.

In the same way that in the path guiding algorithm, MCMC-PG changes distribution on each iteration, as we are effectively sampling a different tree, in both topology and information. Because of this, like in PG, we used all the iterations except the last one to progressively train the tree, and the last one to get the final rendered image.

Each time we updated the trees of our MCMC-PG algorithm we also stored global information about the scene in order to test out the hypothesis that the combination MCMC-PG would make MCMC algorithm more efficient.

For more details, see algorithm 9 in page 87 after our results for this experiment. Our results are given in the figures 4.10, 4.11, 4.12, 4.13 and 4.14 and can be summarized as:

- MCMC-PG does improve the efficiency of MCMC, over scenes (`Veach-ajar` and `Kitchen`) with not too many dielectrics and specular vertices, but it's less efficient than PG alone for this case.
- MCMC methods explore in a better way near-specular vertices than PG, which has problems due to the refinement level of the scene world space. We were able to verify that the combination MCMC-PG improves over PG for the `Staircase` scene, but it is still hindering the better technique, in this case, MCMC.

Fom the acceptance ratio shown in Table 4.2 we can conclude that the only scenes in which the performance of MCMC-PG was better than MCMC were scenes in which MCMC-PG improved the large mutations acceptance rate. We can also conclude from Table 4.3 that if the trees have a better training, the acceptance of the large mutations is increased, even if MCMC-PG is not very efficient like in the `Staircase` case.

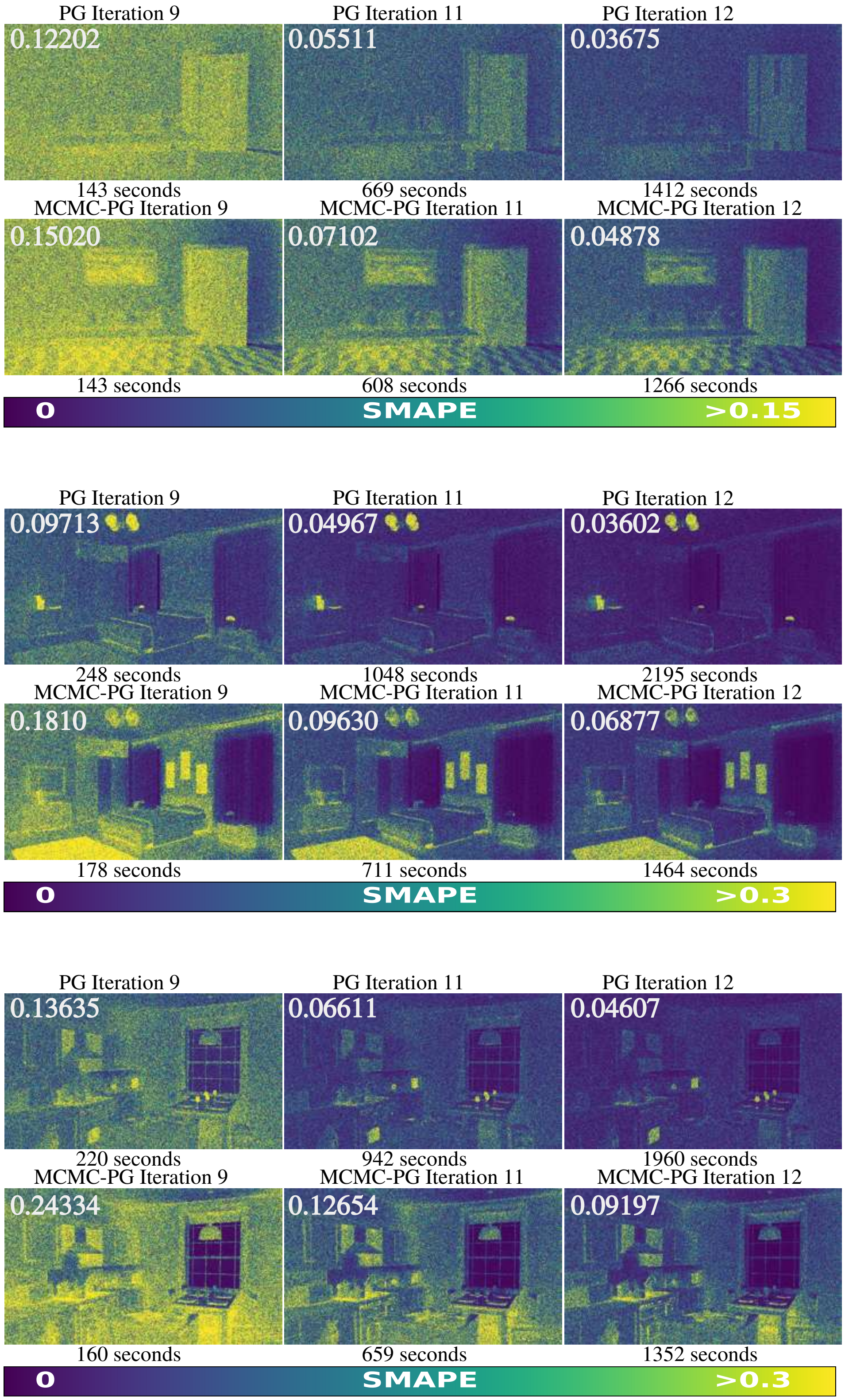


Figure 4.10: Each iteration the SD tree for both PG and MCMC-PG is progressively trained by all the previous iterations. For example, iteration 12 returns a render with $2^{11} = 2048$ samples with the previous 11 iterations used as training. MCMC-PG is more efficient than PG for near specular materials, but less efficient than PG for the rest of the regions.

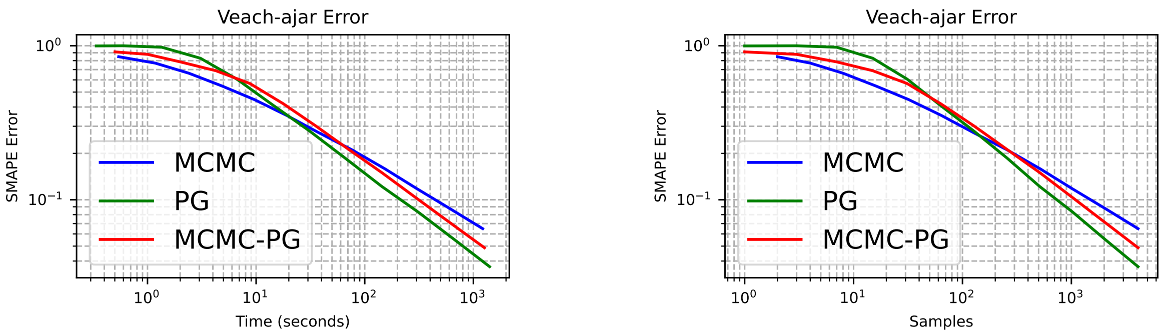


Figure 4.11: `Veach-ajar` was the only scene in which we observed a significant improvement of MCMC-PG over MCMC, but it is still less accurate than just applying PG.

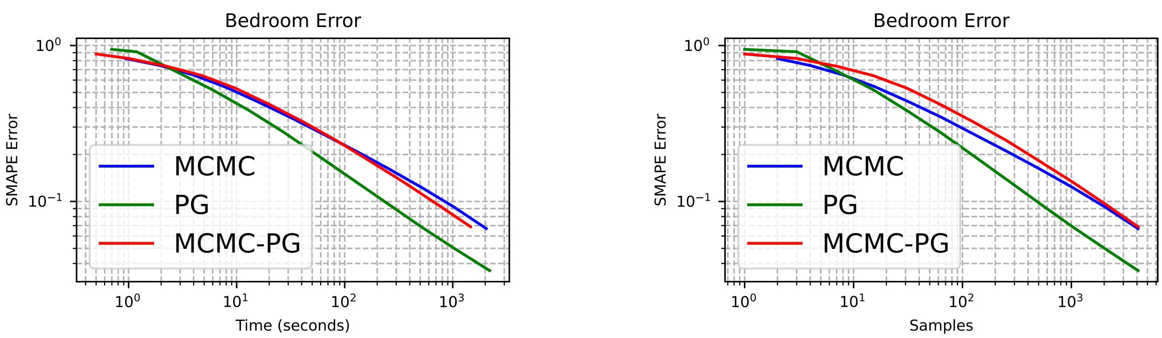


Figure 4.12: In `Bedroom` the accuracy of MCMC-PG with respect to time, is just slightly better than the accuracy of MCMC alone.

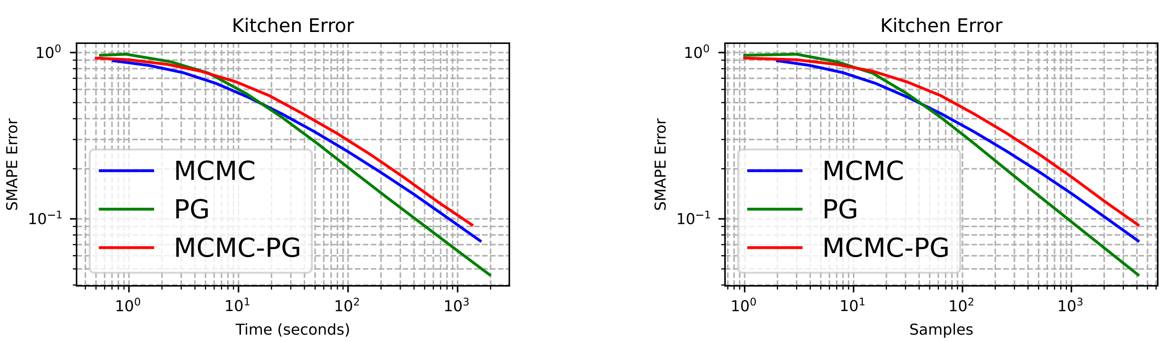


Figure 4.13: `Kitchen` has several dielectric near-specular materials. The combination MCMC-PG just adds an extra computational cost, with no improvement over MC. We already saw in the pre-trained case for this scene that the large mutation acceptance rate was reduced.

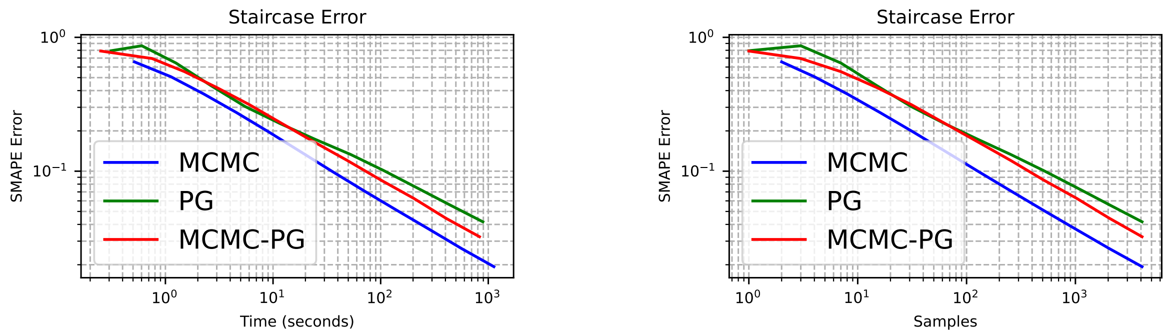


Figure 4.14: In `Staircase` the combination MCMC-PG improves the exploration of near-specular regions with respect of PG (P), but it is still less accurate than just applying MCMC alone.

For all the previous plots we can observe several line intersections, i.e., one algorithm appears to perform better first, but eventually becomes worse after an intersection point of the plots. This can be explained by the way these algorithms work, we used *online training* which means that the sampling pdfs changed on every iteration, effectively leaving us with a "different" estimator for each segment of the plot. In the first iterations we had poorly online trained PG and MCMC-PG trees, and after a few iterations the plots of the algorithms that used path guiding became way better.

`Veach-ajar`			
Acceptance rate	**MCMC**	**NEE**	**MCMC-PG**
Large mutations	0.003	0.039	**0.089**
Small mutations	0.31	**0.46**	0.4
`Bedroom`			
Acceptance rate	**MCMC**	**NEE**	**MCMC-PG**
Large mutations	0.054	0.033	**0.09**
Small mutations	**0.53**	0.51	0.51
`Kitchen`			
Acceptance rate	**MCMC**	**NEE**	**MCMC-PG**
Large mutations	**0.039**	0.018	0.033
Small mutations	**0.65**	0.59	0.62
`Staircase`			
Acceptance rate	**MCMC**	**NEE**	**MCMC-PG**
Large mutations	**0.15**	0.11	0.12
Small mutations	**0.73**	0.67	0.68

Table 4.2: We saw an improvement over MCMC in the `Veach-ajar` and `Bedroom` scenes, and they are the only scenes in which the large acceptance rate is improved by MCMC-PG.

Scene	**Iteration 5**	**Iteration 7**	**Iteration 9**	**Iteration 12**	**Pre-train**
`Veach-ajar`	0.022	0.043	0.060	0.089	0.060
`Bedroom`	0.069	0.077	0.084	0.093	0.085
`Kitchen`	0.026	0.029	0.031	0.033	0.029
`Staircase`	0.109	0.119	0.122	0.127	0.128

Table 4.3: Ratio of accepted large mutations on each iteration of MCMC-PG. The acceptance rate does increase with the training of the spatio-directional tree, regardless of the scene.

Because the results only improved the efficiency marginally with respect to MCMC, we propose to combine guiding with Stratified MCMC, as we believe that the random replay shift and the stratification induced by the algorithms might improve the training of the tree while still using a Markov Chain.

We will proceed first to introduce several experiments we did with SMCMC, in order to improve its efficiency, like adding BDPT and MMLT path operators, ending with our SMCMC-PG experiment.

Algorithm 9 Online MCMC + PG.

Input: spp
Output: Rendered image.

$B_{T_S} \leftarrow \emptyset$, $B_{T_B} \leftarrow$ InitializeBinaryAndQuadTrees()
it $\leftarrow$ `0`
samplesPerIteration $\leftarrow 1$ maxIterations $\leftarrow \log_2$(spp)
while it<maxIterations **do**
    $(b, \mathcal{B}) \leftarrow$ BootstrapStage(N_{init},it)   ▷ Use different seeds on each iteration and the trees.
    for `k` $=$ `0`; `k` $< N_{\text{chains}}$; `k++` **do**
        $N_c \leftarrow N/N_{\text{chains}}$
        rngIndex$\leftarrow$Sample($\mathcal{B}$)   ▷ Get different seed on each different iteration.
        sampler $\leftarrow$ MLTSampler(mpp,rngIndex,σ, p_L) ;
        $(L_c, p_c) \leftarrow$ getRadiance(sampler)   ▷ Using *and* training trees, p_c raster.
        for `i` $=$ `0`;`i`$< N_c$; `i++` **do**
            sampler.StartIteration()   ▷ Chooses between large or small step, with prob. p_L.
            $(L_p, p_p)\leftarrow$ getRadiance(sampler)   ▷ Using *and* training trees.
            $\alpha \leftarrow \min(1, I_p/I_c)$
            $W_p \leftarrow \frac{\alpha}{I_p}$ $W_c \leftarrow \frac{1-\alpha}{I_c}$
            if $I_p > 0$ **and** it $=$ maxIterations $- 1$ **then**
                AddContributionToImage($p_p, W_p L_p$)
            end if
            if it $=$ maxIterations $- 1$ **then**   ▷ Only the most refined tree contributes to the image.
                AddContributionToImage($p_c, W_c L_c$)
            end if
            if rand()$< \alpha$ **then**
                $(L_c, p_c) \leftarrow (L_p, p_p)$
                sampler.Accept()
            else
                sampler.Reject()   ▷ Restores the previous state using the back ups.
            end if
        end for
    end for
    updateSDTrees()
    if it $=$ maxIterations $- 1$ **then**
        ScaleImage(b/samplesPerIteration)
    end if
    it $\leftarrow$ it+1
    samplesPerIteration $\leftarrow$ samplesPerIteration$\times 2$
end while

4.2 Stratification

The objective of this section is to improve the SMCMC technique proposed by Gruson et al [GWH20], and also analyze if the combination SMCMC-PG improves the convergence speed over both PG and SMCMC. We believe that the introduction of more complex path-space operators than just using NEE-PT with random replay will improve the convergence speed of SMCMC, because we would be exploring more efficiently the path-space.

We use the following abbreviations: Sh = Shift, RR = Random Replay, RE = Replica Exchange. The original paper uses the two main techniques (RE:On, Sh:RR), which we sometimes call RE + RR and (RE:Off, Sh:RR).

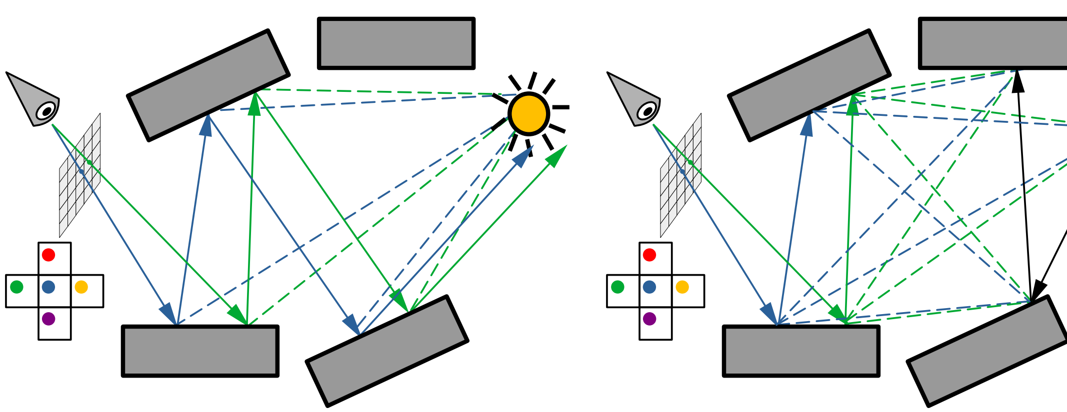
Experimental setup The small mutations that we used are the Hachisuka small mutations (see Algorithm 1). The tile solver that we used is the iterative solver with 200 iterations. $p_{\text{large}} = 0.3$ and the initialization was done using global MCMC chains with RE.

4.2.1 BDPT shift

We think that the introduction of a BDPT styled integration, which we call a "BDPT shif", or a SMCMC-BDPT technique, will improve the convergence speed of the SMCMC algorithm due to the increased exploration of the path-space and that we can take advantage of the fact that the 5 pixels of the tile can share the same light path, reducing in this way the computational cost of this BDPT integration.

To verify if this is true, we modified the random replay shift to account for light connections, in a BDPT-style, effectively replacing NEE (see Figure 4.15a) in the offset paths that were obtained with random replay with deterministic connection with a common light path (see Figure 4.15b). We then assigned a full BDPT sample (the sum given by Equation 2.129) to each pixel in the tile, but we ignored both the cases $t = 0$ (light tracing) and $t = 1$ (direct connection of the camera), as we don't want to drastically change of pixels, this was done, of course, considering an adjusted BDPT-MIS weight that only accounts for $t \geq 2$.

The results of this application are shown in the next three pages, the error plots of our proposed BDPT shift are presented in Figure 4.16 as the blue (no RE) and red (with RE) plots in all the different scenes. The conclusion here is that this proposed "BDPT shift" does indeed improve the efficiency of the random replay (RR) shift with RE of the original paper. For example, using a BDPT integrator for the `Veach-ajar` is notoriously inefficient (see the magenta plot of the `Veach-ajar` scene at Figure 4.16) as no vertex can be connected due to the wall's occlusion. However the combination



(a) Random replay shift map with a NEE path tracer.

(b) Random replay shift with a BDPT styled path tracer, a "BDPT" shift.

Figure 4.15: The Random Replay shift of the original paper [GWH20] used NEE-PT (a), we propose to use the "BDPT" shift (b) with a shared light sub-path used by all the pixels of the tile. Note that both diagrams show only two out of five pixels, but the process is done on all five pixels.

of SMCMC + BDPT improves the efficiency of the estimation as we are taking advantage of both the local exploration of the MCMC process with the compact way of exploring the path-space with BDPT.

Our BDPT shift is strong for scenes with visibility issues with respect of the light (see Figure 4.17), and it works well with scenes with some specularity (see figures 4.18 and 4.19), with the inconvenience that RE affects a little the efficiency of paths that have near-specular dielectric material. Unfortunately, if the amount of near-specular and specular materials is too high (see Figure 4.20) our technique is less efficient, and it can't handle images where light transport with $t = 1$ is important, see the `Veach-egg` scene (Figure 4.21).

Implementing SMCMC with a BDPT shift that includes the technique $t = 1$ is more complex, as each vertex of the light path when it is connected to the camera lens, could end in any pixel of the image. This would cause for all the pixels to receive ($t = 1$)-samples from all the Markov Chains (tiles), with the issue that each ($t = 1$)-sample received would have to be treated differently depending of the Markov Chain (tile sampler) that generated it, due to the fact that each tile has a different normalization constant. But this is quite complicated, if we naively store for each pixel the $t = 1$ contributions from each Markov Chain (tile sampler), we would need to store an array of size A_{image} for each pixel, this would induce RAM usage on the order of terabytes, not to mention the computational effort required for this hypothetic tile aligning of the BDPT light image.

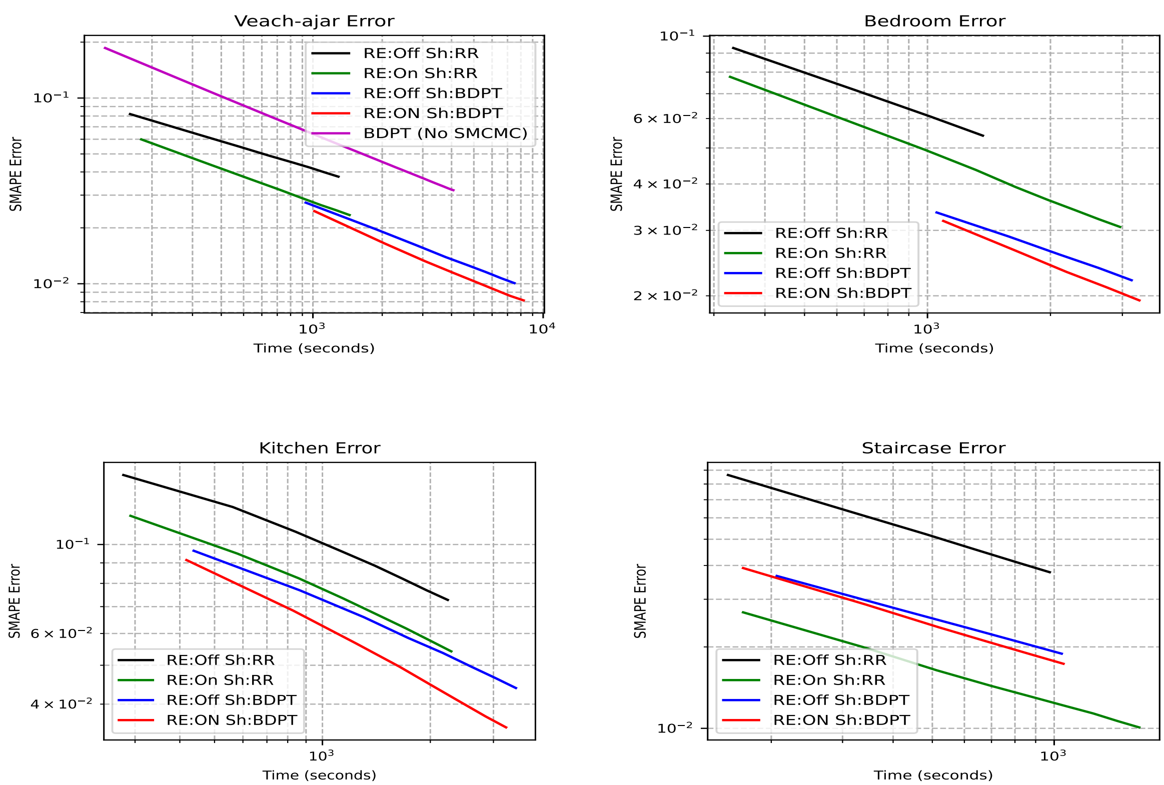


Figure 4.16: The (SMCMC) BDPT shift improves the accuracy of the algorithm except in the `Staircase` case. This scene converges rapidily with a simple (without NEE) path tracer which means that a BDPT shift is too computationally expensive for something that PT does very efficiently. On top of that, the high amount of near specular vertices (a floor and a glass railing, both with roughness of 0.01, i.e., almost specular) hinders the ability of connecting BDPT vertices. As a side note, we included the BDPT (not SMCMC) magenta plot for the `Veach-ajar` scene to show that even if BDPT alone is very inefficient for scenes like `Veach-ajar` our SMCMC BDPT shift takes advantage of the MCMC process improving considerably the efficiency of the SMCMC algorithm in scenes with difficult light transport due to visibility issues alone.

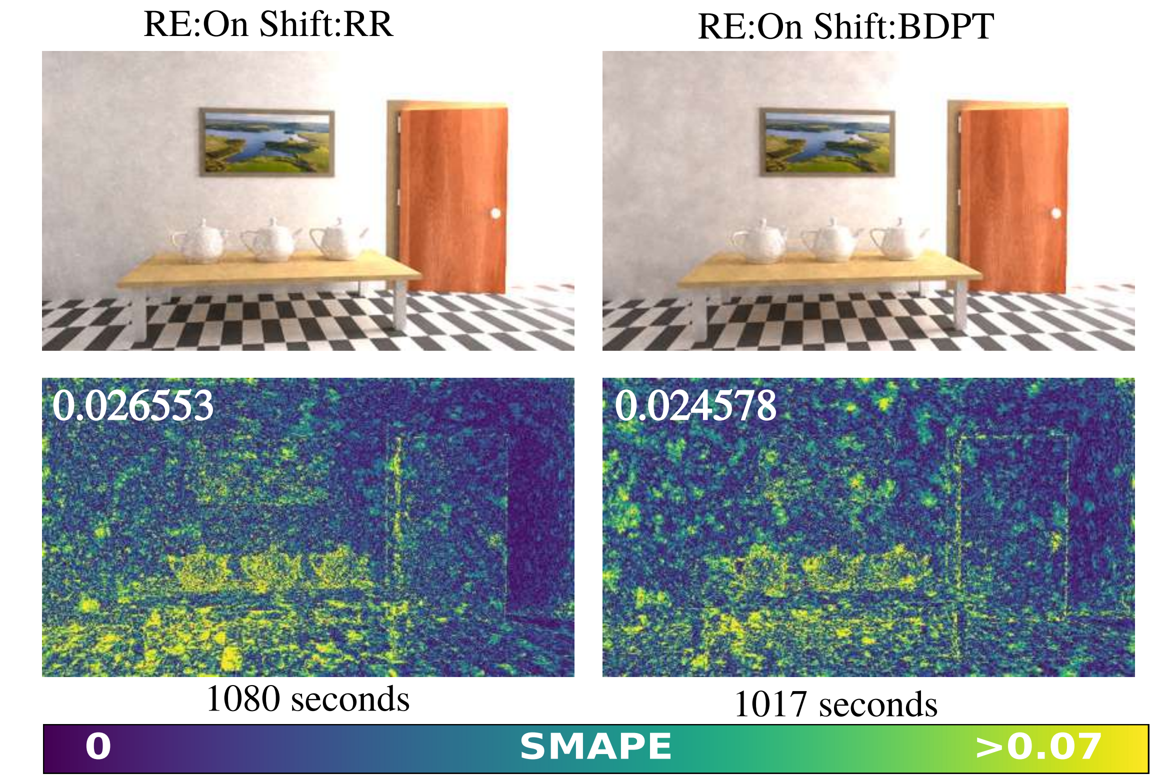


Figure 4.17: Our technique shines with scenes with visibility issues and far from specular materials, that is, diffuse materials and glossy materials with an almost diffuse roughness, like in `Veach-ajar`.

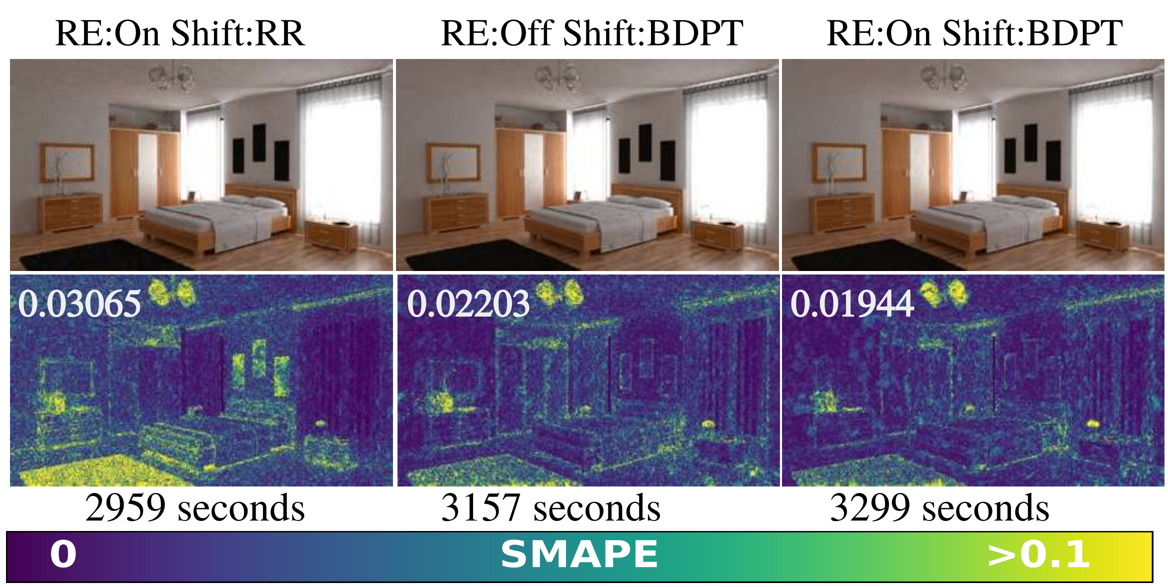


Figure 4.18: We can observe that in the `Bedroom` scene the BDPT shift works better than (RE:On, Sh:RR). It is strong for paths with vertices that are not nearly specular, and works for the dielectric materials (see the chandelier and the glasses on the dressers). BDPT+RE hinders a little bit the dielectric materials, even if it is globally better for the whole image.

Figure 4.19: The BDPT shift works best for parts of the image with visibility issues but that can be reconnected in several directions, i.e., far from specular, even when they are dielectric. Also notice that RE enhances this effect both positively (it improves the far specular with visibility issues areas) and negatively (it is less efficient for near specular glasses).

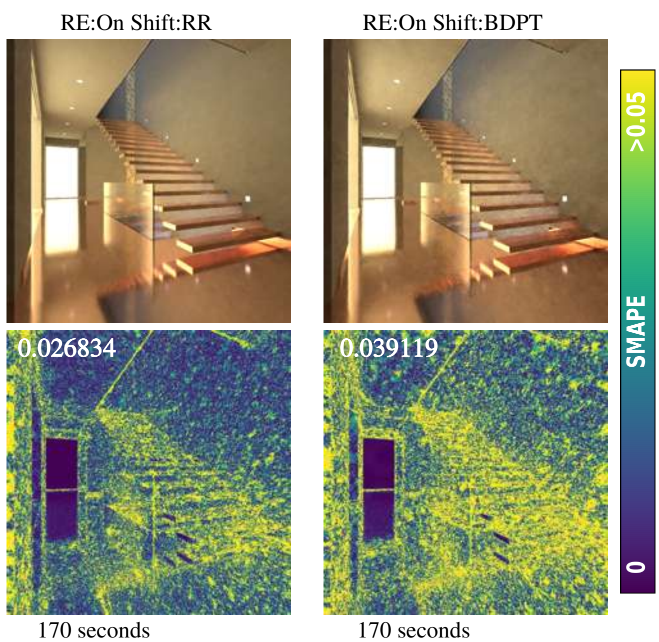
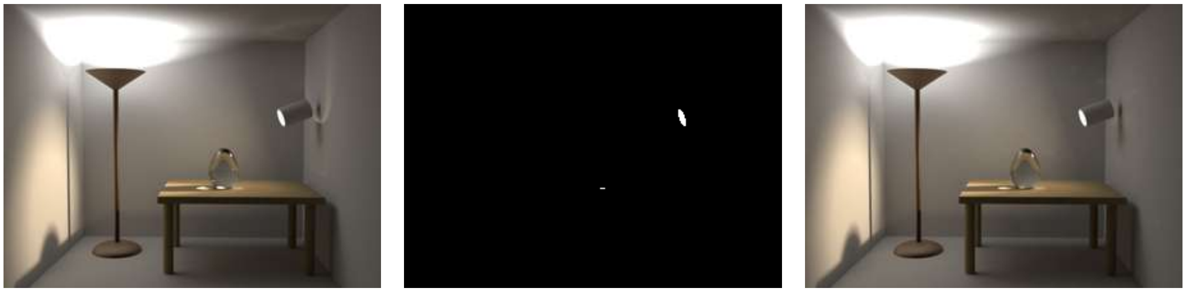


Figure 4.20: In the `Staircase` scene we see that BDPT does not work better, due to the fact that the almost specular floor is making vertex connections impossible.

4.2.2 Random strategy BDPT shift

We believe that we can further improve the efficiency of our BDPT shift, by reducing the amount of expensive deterministic connections that BDPT has to perform from $k+1$ per each depth D_k to only one, stochastically selected, reducing in this way the number of visibility checks which are very computationally expensive.

In a BDPT integrator, we estimate all the integrals $(I_j)_{D_k}$ (Equation 2.128). Inspired by Matrix Bidirectional Path Tracing (M-BDPT) [Cha+18], we denote the contribution of the (s,t) technique to



(a) `Veach-egg` scene, reference. (b) Technique ($s = 6, t = 1$). (c) SMCMC Sh:BDPT, RE:On. 256mpp

Figure 4.21: The paths $t = 1$ are fundamental to get a quick convergence in the upper right corner region of this scene.

the whole integral as

$$I_j^{s,t} \equiv E\left[\frac{1}{N}\sum_{i=1}^{N} w_{s,t}(\mathbf{X}_i)\frac{f_j(\mathbf{X}_i)}{p_{s,t}(\mathbf{X}_i)}\right], \tag{4.4}$$

where all the $\mathbf{X}_i$ have s light vertices and t camera vertices. We can calculate the integral $(I_j)_{D_k}$ over the paths of some fixed length $k+1$ as:

$$(I_j)_{D_k} = \sum_{(s,t)\in\mathcal{T}_{k+1}} I_j^{s,t}, \tag{4.5}$$

where $\mathcal{T}_{k+1}$ was already defined in Equation 2.127. This means that we can perform an MC estimator over the BPDPT-MIS weighted techniques to estimate the integral.

We propose instead of using all the possible (s,t) BDPT strategies, to just randomly select one (s,t) strategy per each integral $(I_j)_{D_k}$ of the paths of a given length $k+1$ with probability $(p_{\text{tech}})_{D_k} = \frac{1}{k+1}$ (because $t \geq 2$), or in other words, we are choosing the strategy with a uniform probability. We do this stochastic process by simply using $m_d + 1$ extra random numbers for the Markov Chain. We will abbreviate this shift map as "rBDPT"

This does improve the efficiency as it can be seen on Figure 4.22, but it has the drawback that for dielectric materials rBDPT is worse than BDPT (see Figures 4.23 and 4.24).

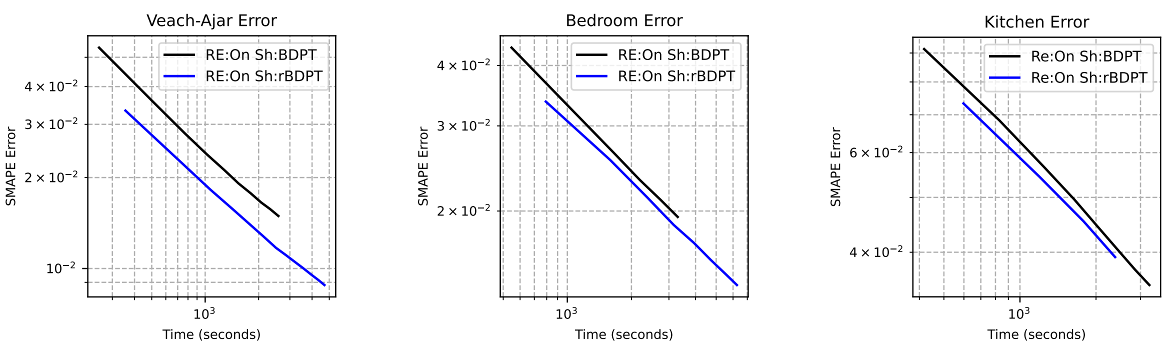


Figure 4.22: Note that for all the three scenes the random strategy BDPT (rBDPT) shift is more efficient. The samples of RE rBDPT were taken in batches of 128mpp while the RE BDPT samples in batches of 32mpp.

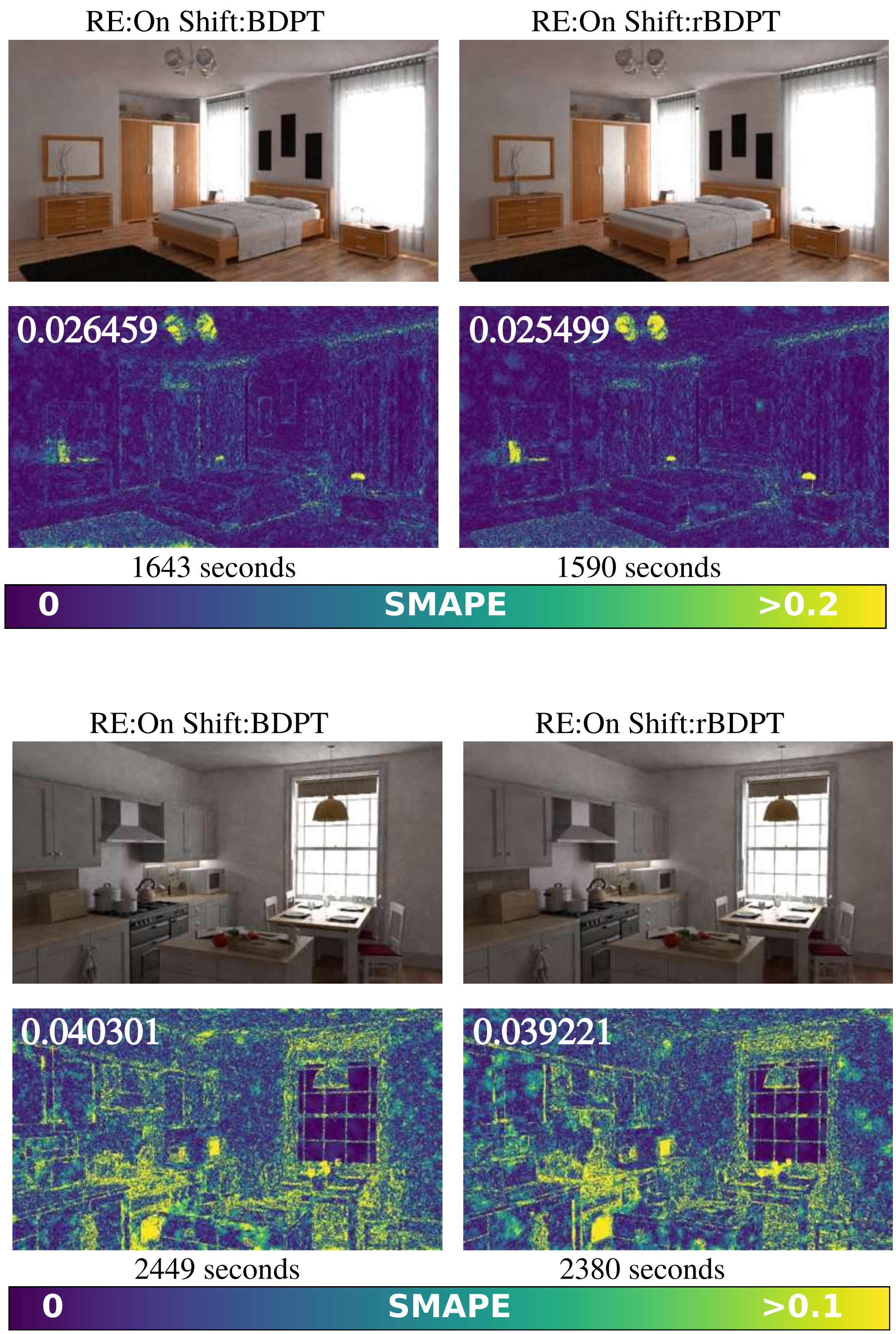


Figure 4.23: rBDPT handles better occluded but diffuse areas, but it doesn't handle well dielectric specular materials because they induce paths in which just few BDPT-MIS weighted (s, t) techniques get most of the radiance.

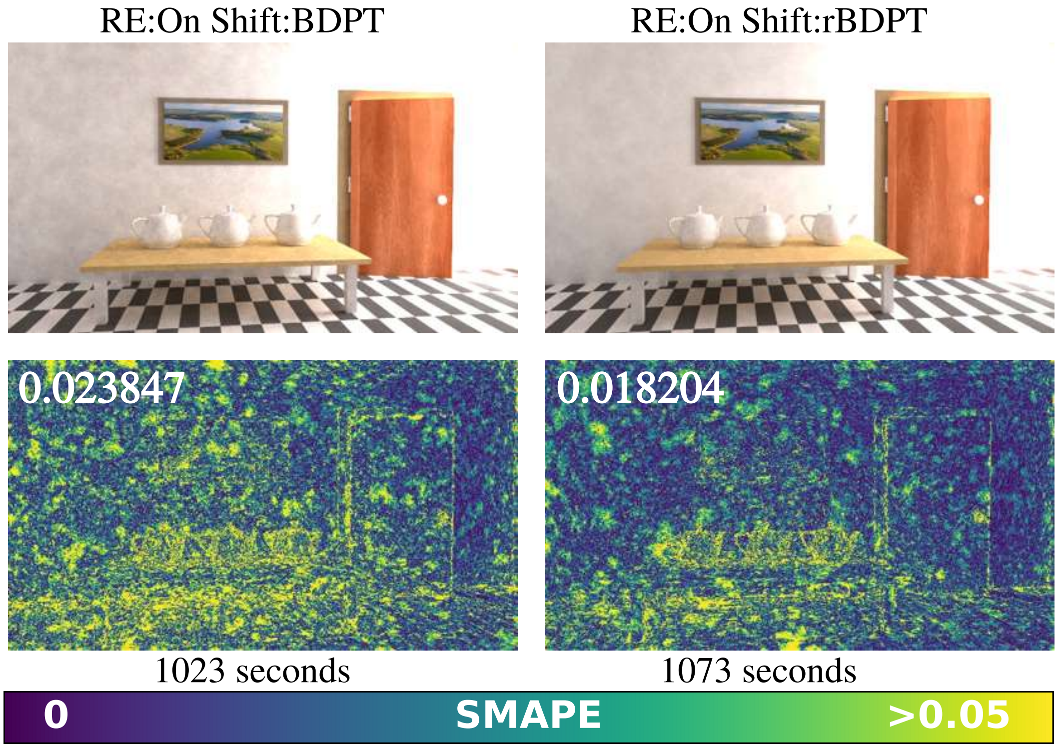


Figure 4.24: All the materials in this scene are diffuse so rBDPT has no problem with the uniform selection of techniques as the BDPT-MIS weights tend to not prefer one specific (s, t) technique.

The decreased efficiency provoked by dielectric materials can easily be explained because these materials induce paths were the MIS-BDPT weights are concentrated in just a few techniques (e.g., in `Veach-egg` the upper right corner has a lot of paths in which $t = 1$ is the technique that contributes the most), so the uniform sampling hinders the technique for this case. In general dielectric glass materials receive a high contribution from $t = 1$ paths and also from long camera paths, i.e., for paths of some depth D_k, we have that the camera paths in which $t \to k + 1$ often have a high contribution.

This shift is other enhancement to scenes with visibility issues but still has problems with specular and near-specular materials.

4.2.3 MMLT shift

We think we can improve ths efficiency of the SMCMC estimator, reducing even more the amount of reconnections done per each iteration, selecting only one depth per iteration with probability p_d, and then only one (s, t)-technique with probability p_{tech}, decreasing the number of connections of the rBDPT shift from $m_d + 1$ to just 1 per pixel in an MMLT-style [HKD14]. We believe this idea will improve the efficiency of the SMCMC algorithm because the exploration induced by the stationary distribution of the target function of the Markov Chain will perform more iterations in less time than both the BDPT shift and the rBDPT shift, increasing the number of tried (but not necessarily accepted) large proposals per compute time. We will call this an “MMLT shift” (see Figure 4.25).

Our MMLT shift has the disadvantage that we are not fixing the depth of the chain as it was suggested but not required in the multiple bounce optimization of the MMLT paper [HKD14]. We can’t do this because each tile has its own normalization constant, therefore we would need $m_d + 1$ Markov Chains per tile, a very inefficient idea for scenes with a high maximum depth m_d. This has the clear disadvantage that the Markov Chain will spend more time in “brighter” depths, but it solves the normalization problem and we don’t need to use $m_d + 1$ Markov Chains anymore.

To be able to apply our MMLT shift we need to define our pdf’s p_d and p_{tech}. For the depth pdf p_d we will simply choose the depth uniformly as $p_d = \frac{1}{m_d+1}$. Once we have chosen some depth d_i stochastically with probability p_d, we need to select a technique k out of the possible (s, t) techniques, with some probability $p_{\text{tech}}(k|d_i)$. If we consider the probability to be uniformly the same across all

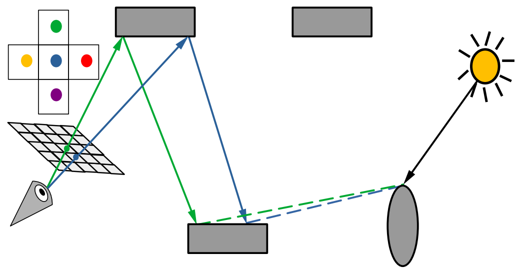


Figure 4.25: Random replay with MMLT-style reconnections, an "MMLT shift". Instead of performing all the connections between all the vertices of the light path with all the vertices of the camera path, we choose one of the techniques in the same style that MMLT.

the techniques (with $t \geq 2$), we have $p_{\text{tech}}(k|d_i) = \frac{1}{d_i+1}$. This *uniform* selected p_{tech} is going to be denoted as a "uMMLT" shift.

For scenes that lack dielectric materials (which have high contributions from $t = 1$ and long t values), we will select long light paths more frequently. This heuristic makes intuitive sense as the Markov Chain of the tile will generate more often short paths for all the five pixels while spending more computational effort in the shared light path. Reducing both the computational cost and the variance of the paths, as they would be a little more topologically similar from each other because they're sharing the same light path.

We can choose our long light paths proportionally to any appropriately defined pdf we like. In our case we will use a "cubic" pdf, defined as

$$p_{\text{tech}}(k|d_i) = \frac{\left[1^3, 2^3, 3^3, ..., (d_i+1)^3\right]}{\sum_{l=1}^{d_i+1} l^3} \tag{4.6}$$

where each index of the array (starting from zero) corresponds to the selection of an s light vertices. For example, the position with 3^3 corresponds to the selection of the technique $s = 2$. Notice that we could have defined it in a linear way l^1 (lMMLT), quadratic l^2 (qMMLT), or any other power or form that selects longer light paths. The heuristic that we will use for our comparisons is the cubic heuristic (cMMLT), as its efficiency is similar to lMMLT and qMMLT for scenes without dielectrics, but a little more efficient than the rest for the `Veach-ajar` scene (see Figure 4.26).

Before proceeding to show our results, we must remind the reader that Equation 4.6 must be included in the MIS-BDPT weight given by Equation 2.131, which is easily done by performing the

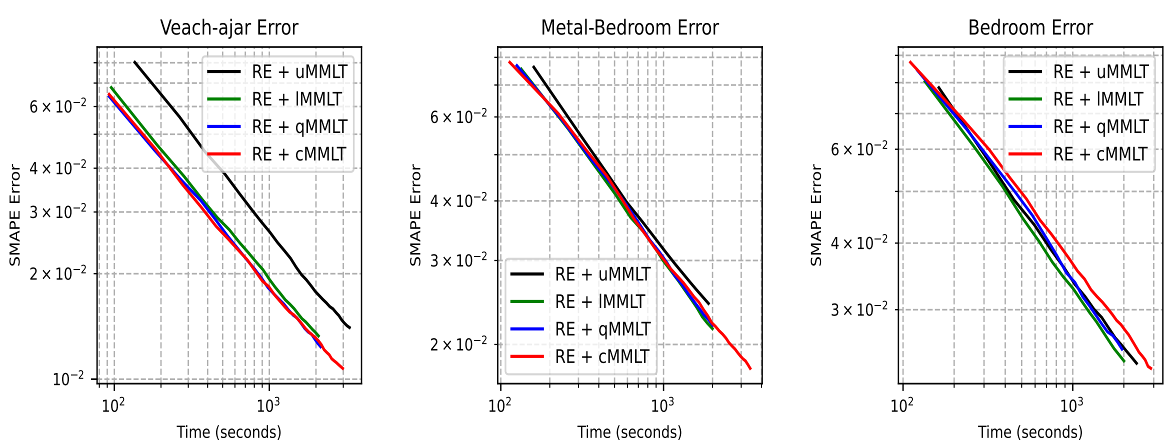


Figure 4.26: For scenes that lack dielectrics (`Veach-ajar` and `Metal-Bedroom`, the version of `Bedroom` without dielectrics) the long light path heuristic works. We will use cMMLT for our comparisons as it is similar to lMMLT and qMMLT in efficiency, but a little better in the `Veach-ajar` scene.

appropriate multiplication. This would look like $p_j(\mathbf{X}_{s,t})p_{\text{tech}}(j|d_i)$ for all possible techniques j of some length $k = s + t - 1$.

For scenes without dielectric materials, like `Veach-ajar` and `Metal-Bedroom` (which simply replaces the dielectrics with almost specular (roughness of 0.01) conductor materials in the `Bedroom` scene), this technique does improve the efficiency (see Figure 4.27).

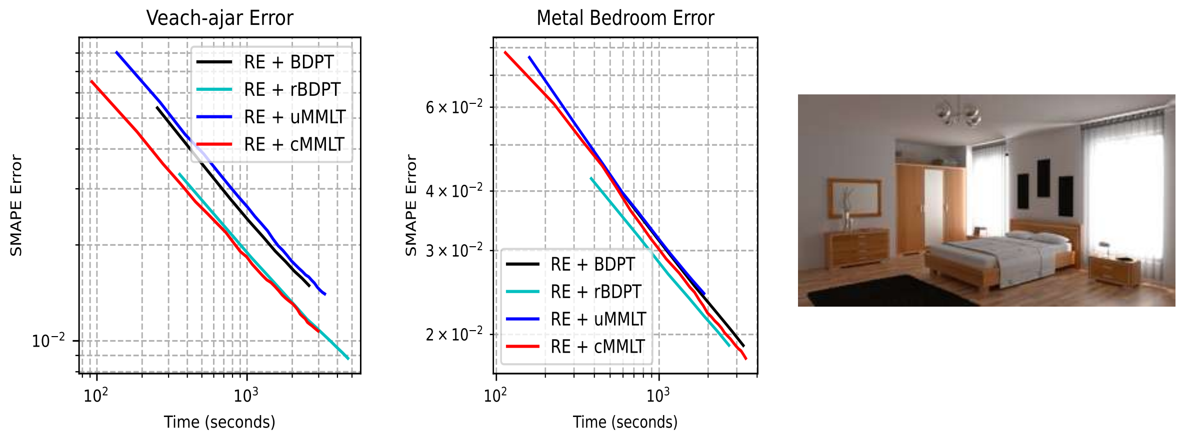


Figure 4.27: These scenes lack dielectrics. We can see that for both scenes cMMLT improves over (RE:On, Sh:BDPT). We are also showing the `Metal-Bedroom` reference.

Unfortunately, dielectric materials are also not handled well by either uMMLT nor cMMLT. Additionally, regions of the scene that are too bright decrease the efficiency as we are taking randomly uniform depth samples per each iteration which makes the acceptance of other depths more unlikely (see Figure 4.28).

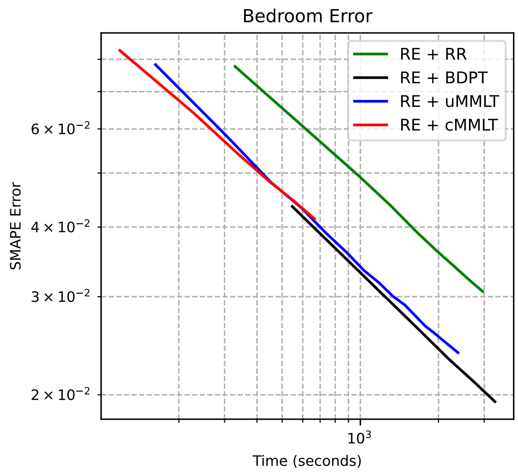
(a) **`Bedroom`** rendered using RE + cMMLT, 768 mpp.

(b) **`Kitchen`** rendered using RE + cMMLT, 640 mpp, with Hachisuka small mutation.

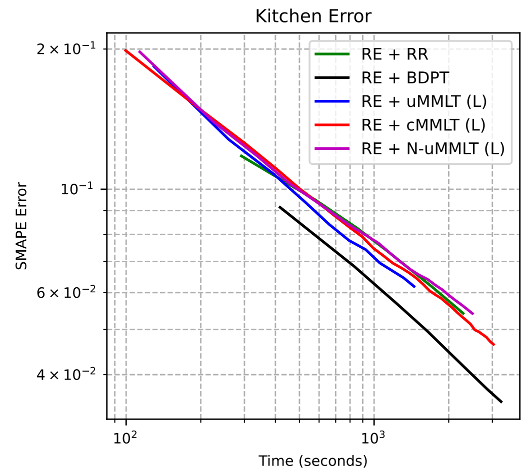
(c) **`Kitchen`** rendered using RE + **N**-cMMLT, 640 mpp, with **N**ormal distributed small mutation ($\sigma = 0.01$).

Figure 4.28: The two problems of this technique. We can see that dielectrics are hard to render by observing the dark grayish glasses on the dressers and the chandelier of (a) and also the dark grayish glasses on the table of (b) and (c), when they should have a clear tonality typical of a glass. We can also notice the brightness exploration problem by looking at the extremely brighter than usual window in Figure (b). The brightness problem can be reduced using smaller mutations (c), as in the `Kitchen` example, $m_d = 13 \to p_d = 1/14 \approx 0.071$, which means that that the Hachisuka mutation is too long. If we want to use the Hachisuka mutation with the cMMLT shift we need to spend up to 20 minutes properly initializing the chains in order to avoid those bright artifacts in the window.

Therefore, for scenes with a high amount of caustic paths and dielectrics, we should avoid the cMMLT shift. The uMMLT also has problems with bright regions in the scene, but in a less intense way than cMMLT. These problems can be reduced by forcing that in the initialization stage, that at least 70 percent of the tiles are initialized (Long initialization), or by just reducing the size of the steps of the small mutation (normal mutation with $\sigma = 0.01$). We present the error plots for these scenes in Figure 4.29.



(a) The MMLT algorithms behave better than the RE + RR of the SMCMC paper, but they have problems with the dielectic regions.



(b) To render Re + cMMLT and avoid the brightness artifacts in the window 4.28b we rendered all of our MMLT experiments with long initialization.

Figure 4.29: The shift that uses normal distributed small mutations and a uniform technique selection is denoted as N-uMMLT. Note that in both scenes, uMMLT is better than RE + RR, but RE BDPT handles in a better way the dielectric paths and has no issues with bright regions of the scene. Also notice that in this case (scenes with dielectrics) uMMLT is better than cMMLT.

In the next pages we will present the SMAPE error images (see Figures 4.30, 4.31, 4.32 and 4.33) and also in Table 4.4 the statistics of the acceptance rate of both large and small mutations of all our current techniques.

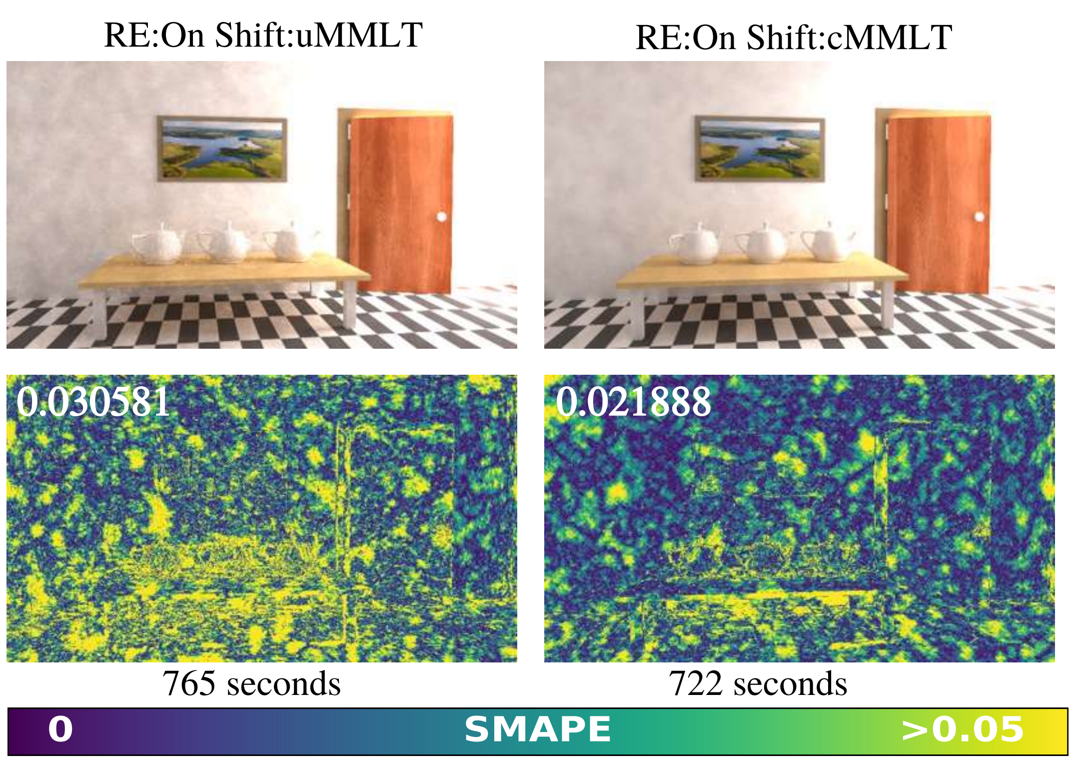


Figure 4.30: In scenes with hard difficult transport due to visibility issues but without dielectric materials like the `Veach-ajar` scene, our long light sub-path heuristic cMMLT is very useful.

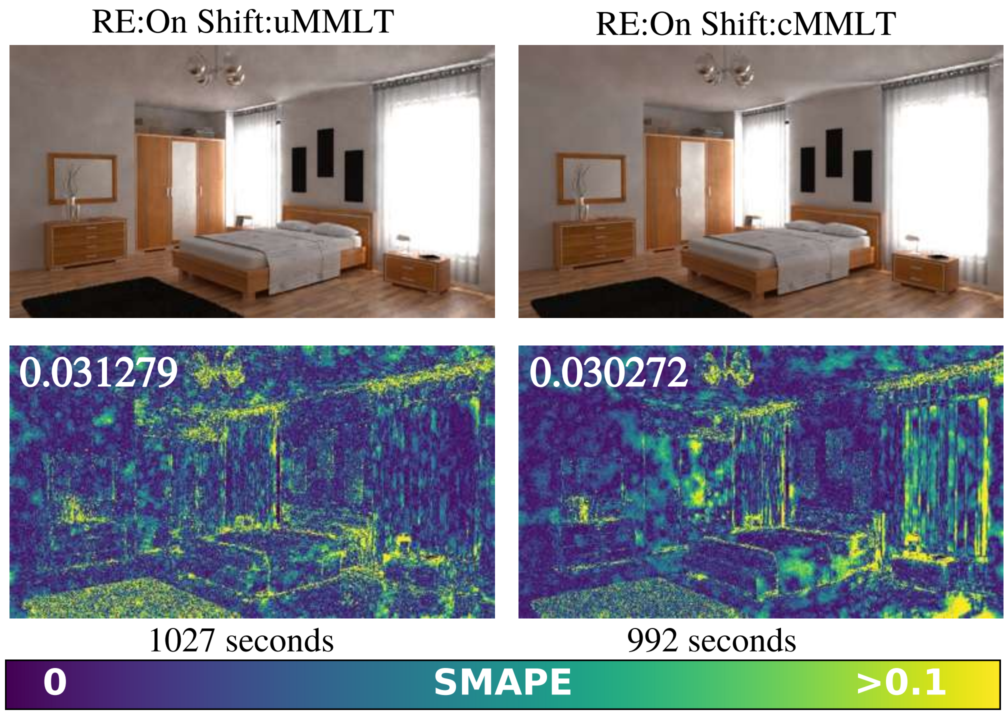


Figure 4.31: When we remove the difficult dielectric paths like in the `Metal-Bedroom` scene, we can observe that our cMMLT is still better than just applying the uniform technique selection of uMMLT.

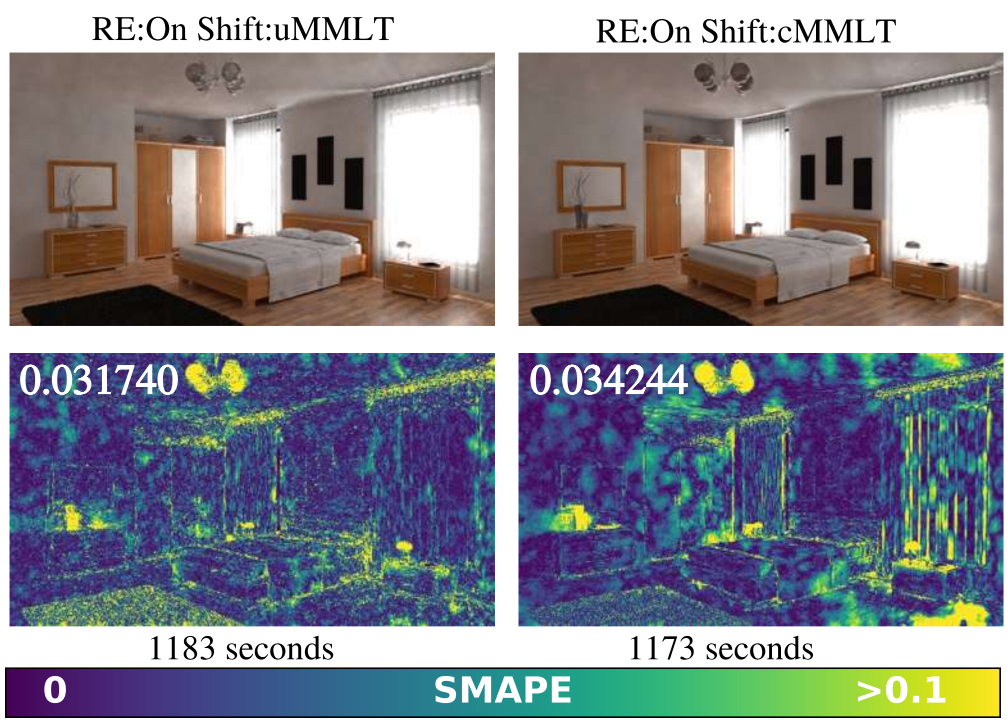


Figure 4.32: The cMMLT heuristic has problems when the scene has dielectric materials. For this type of materials, the BDPT shift is more efficient.

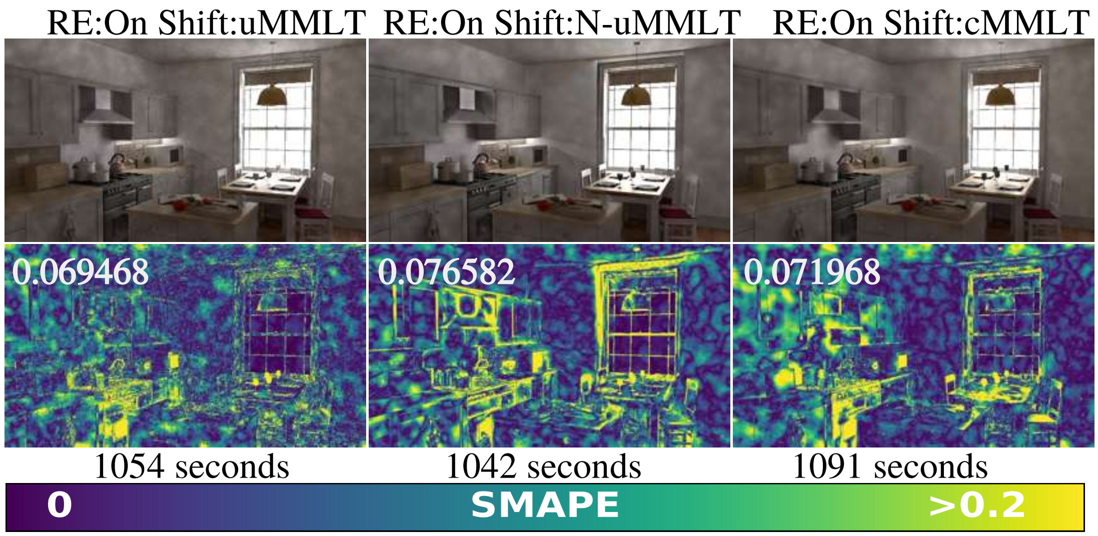


Figure 4.33: This scene has both regions with a high brightness and dielectric materials making the cMMLT very inefficient. Even the uMMLT struggles with this type of scene so we just recommend to use the BDPT shift for this type of scenes.

Veach-ajar					
Acceptance rate	**RR**	**BDPT**	**rBDPT**	**uMMLT**	**cMMLT**
Large mutations	0.085	**0.13**	0.09	0.012	0.012
Small mutations	0.49	0.48	0.5	0.51	**0.53**
Bedroom					
Acceptance rate	**RR**	**BDPT**	**rBDPT**	**uMMLT**	**cMMLT**
Large mutations	0.13	**0.27**	0.20	0.11	0.038
Small mutations	0.46	**0.57**	0.55	0.42	0.51
Kitchen					
Acceptance rate	**RR**	**BDPT**	**rBDPT**	**uMMLT**	**cMMLT**
Large mutations	0.12	**0.19**	0.14	0.0033	0.0042
Small mutations	0.44	**0.50**	**0.50**	0.033	0.12

Table 4.4: All of this ratios were obtained using the above techniques combined with RE. Our BDPT shift is always the most accepted shift for proposed large mutations, and it is also highly competitive with the acceptance small mutations. Regarding uMMLT and cMMLT, they don't behave too well in scenes full of dielectrics.

4.2.4 Guided shift.

This shift consists in using the path guiding algorithm to trace the paths of the random replay process of the SMCMC algorithm. We think that this combination might improve the training of the path guiding tree because we have both stratification and many additional training vertices for the tree due to random replay. Consequently with a better trained tree we think that the efficiency of the combination SMCMC-PG (with implicit random replay) might improve the efficiency of SMCMC. Our implementation is quite straightforward as it can be seen in algorithm 10.

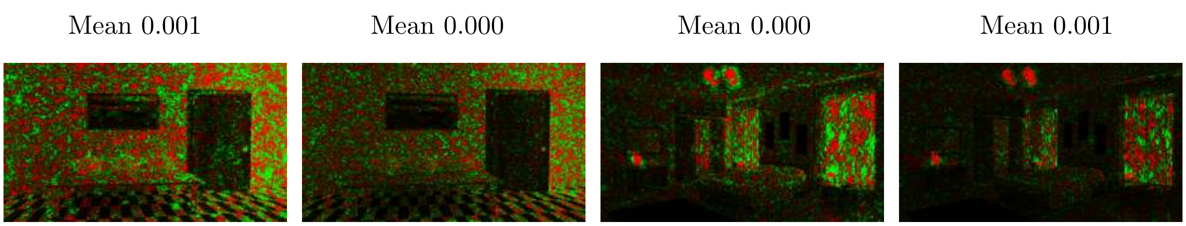
On each iteration of the algorithm we have that the sampling distribution is different as we are using different trees. This implies that the plot of the full rendering process won't be a straight line, as each iteration has a different pdf. Thus, for a *quick* check of unbiasedness we have verified, using the difference error $I - R$, that our method is unbiased (see Figures 4.34 and 4.35).

The error plots are given in Figure 4.36. Despite that this combination produces unbiased renders, its convergence speed is not competitive.

Algorithm 10 Path guided SMCMC.

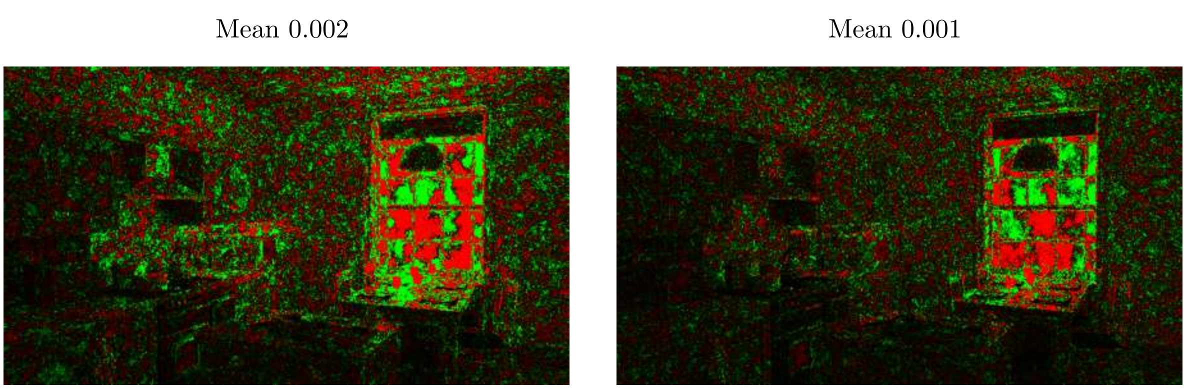
Output: Sequence of progressively rendered images.

samplesPerIteration ← 32; ▷ This is just some arbitrary number.
Images← { }
for $i = 1$ to maxIterations **do**
    initializeTiles() ▷ Either naively or with global MCMC chains with RE.
    for tile **in** tiles **do**
        for $j = 1; j \leq$ samplesPerIteration; $j++$ **do**
            MCMCmutation(tile) ▷ Trains and uses path guiding trees for tracing paths.
        end for
    end for
    currentImage ← reconstructTiles()
    Images.push_back(currentImage)
    updateSDTrees()
    cleanTiles() ▷ To avoid more bias, as each iteration used different trees.
    cleanSamplers() ▷ Just the statistics, as $\mathbf{U}_0$ must be something anyway.
    samplesPerIteration ← 2 samplesPerIteration
end for
return Images



(a) PG Iteration 4 (256 mpp) (b) PG Iteration 6 (1024 mpp) (c) PG Iteration 4 (256 mpp) (d) PG Iteration 6 (1024 mpp)

Figure 4.34: There is around the same amount of green regions than red ones. But near specular dielectrics are not properly sampled.



(a) PG Iteration 4 (256 mpp) (b) PG Iteration 6 (1024 mpp)

Figure 4.35: PG has problems with near specular samplings and this method inherits this problem.

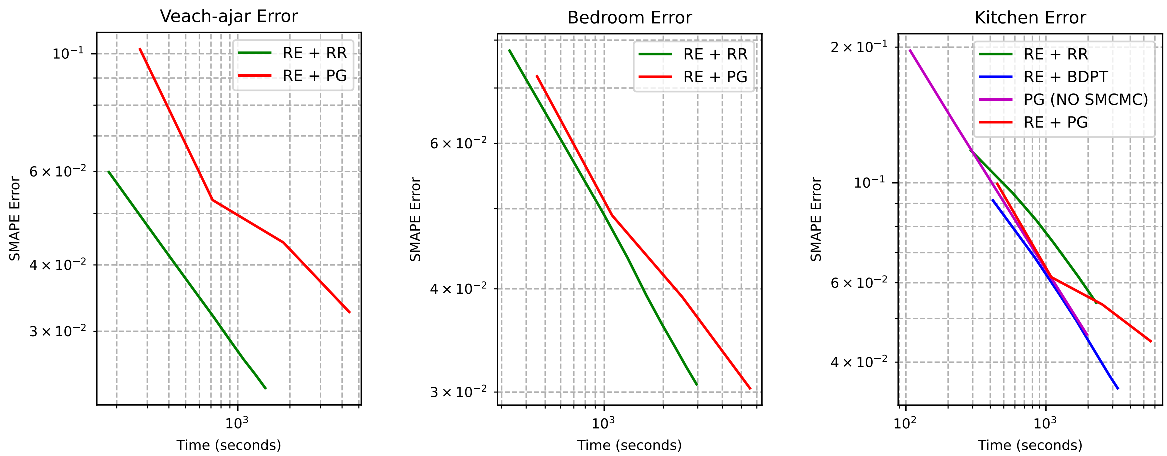
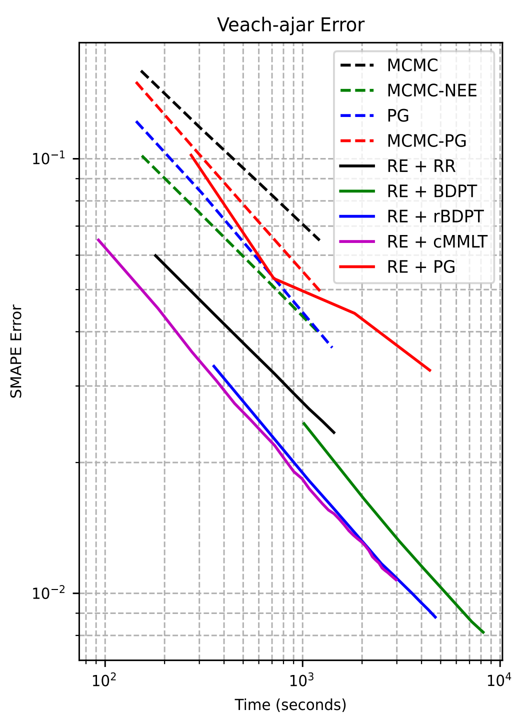


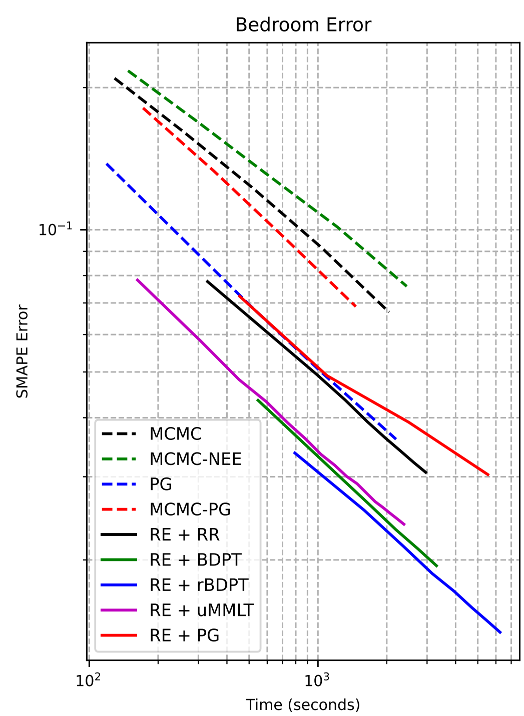
Figure 4.36: The red plot is the error graph of the random replay shift that uses path guiding. Unfortunately its speed of convergence is not competitive when compared with other techniques.

4.3 Chapter conclusions

In this chapter we combined MCMC with PG, and also did several modifications to the path operators of the SMCMC algorithm. We show a summary of all our error plots in figures 4.37 and 4.38.



(a) For the `Veach-ajar` scene without dielectrics the best shift is cMMLT almost tied with rBDPT. For this scene all the SMCMC techniques (solid lines) are superior to non-SMCMC techniques (solid lines).



(b) For the `Bedroom` scene despite that rBDPT seems to be the best globally, it has a slower convergence than just BDPT in the dielectric parts, so we recommend to use the BDPT shift.

Figure 4.37: The dashed plots correspond to all the techniques other than SMCMC, while the solid plots correspond to the SMCMC techniques.

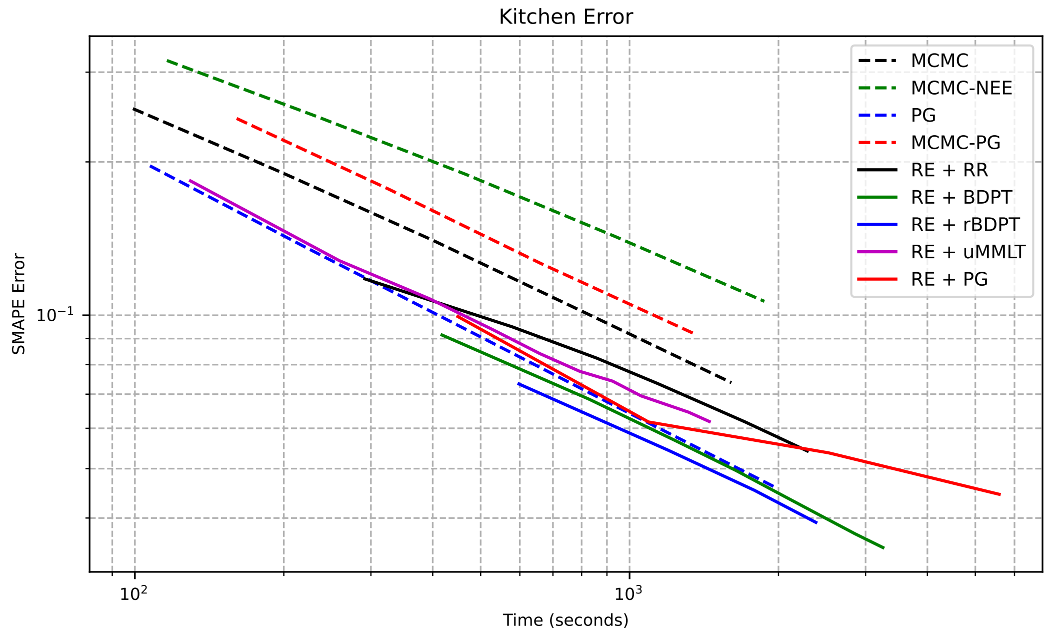


Figure 4.38: The dashed plots correspond to all the techniques other than SMCMC. For this scene the best shift globally is rBDPT but we recommend BDPT as it takes care of dielectric regions more appropriately. Notice how uMMLT is slow because this scene has the two worst problems for the MMLT shift techniques, dielectric materials and very bright regions.

For the MCMC-PG combination we can conclude:

- For scenes with visibility issues but with few specular vertices MCMC-PG does improve the large step acceptance rate, and in general, the convergence speed of the MCMC algorithm. However, it is still less efficient than PG alone for this case.
- For scenes with many specular vertices, PG is slow, and the combination MCMC-PG improves the integration speed a bit with respect of PG, but it is still slower than using MCMC alone.
- In the MCMC-PG method, the more trained the tree, the better the ratio of large accepted mutation becomes regardless of the scene, even for scenes in which this method is not as accurate as MCMC.

Regarding the path operator changes done to SMCMC, we can conclude:

- Our BDPT shift outperforms the original Replica Exchange + Random Replay of the SMCMC paper for all the scenes that don't have large regions of specular materials, for example, the almost specular floor of `Staircase`.
- The random strategy selection BDPT shift, rBDPT, increases the convergence spped of the BDPT shift, but it has problems with regions where dielectric materials are present. Even if it appears to be the best estimator globally, the parts of the scene with dielectrics converge slower than just using the BDPT shift.
- The MMLT style shift is also better than the BDPT shift for scenes where there are no dielectric materials present and there are not very bright pixels.
- The use of a guiding SD tree progressively trained on each iteration of the SMCMC algorithm is just too computationally expensive and slower for scenes with few dielectric and few specular vertices. For scenes with dielectric materials, combined with visibility issues (like `Kitchen`) the extra computational effort is more rewarded, but it is still an overall inefficient technique.

Chapter 5

Conclusions

In this chapter we present a summary of all the techniques employed in this thesis, with their respective disadvantages and advantages, and how our proposed algorithms affected the performance of these techniques. We also present possible future research ideas.

5.1 General conclusions

Throughout this thesis we used several rendering techniques in order to solve the measurement equation of light transport. The most straightforward techniques were Path Tracing (PT), Next Event Estimation Path Tracing (NEE-PT) and Bidirectional Path Tracing (BDPT), which take advantage of the distribution generated by the materials and the light in the scene.

We could verify that the selection of the most efficient integrator depends heavily in the scene used, as there are integrators that even if they are computationally more expensive per sample, like BDPT, they would produce several useless vertices (e.g., the light sub-paths of `Veach-ajar` and the NEE connections in the `Bedroom` scene), wasting valuable computation time. Therefore, the selection of the integrator is an important decision made by the user.

In order to improve the efficiency of previous techniques we used both global exploration (Path Guiding [MGN17]) and local exploration (PSSMLT [Kel+02] and SMCMC [GWH20]) algorithms. We could see that the path guiding algorithm was less efficient for scenes in which a very localized positional and directional information, like in near specular vertices and caustic paths were required. But it was incredibly efficient for scenes in which high radiance path islands of the path-space were not present, i.e., for materials with a wide range of scattering directions. On the other hand, MCMC algorithms were better at exploring similar paths for these specular and caustic regions, but with the disadvantage that their large step exploration is just driven by mere chance, and the stratification of the samples across the rendered image is uneven. To reduce the stratification issue we used the SMCMC algorithm which is in general, more efficient when considering localized error metrics such as the SMAPE error.

Our contribution begins by combining path guiding (global exploration) with PSSMLT (MCMC) and SMCMC (local exploration). We conclude that our MCMC-PG mix (without NEE) increases the efficiency of MCMC (without NEE) and also the acceptance rate of large mutations, only in scenes without caustic paths and specular materials. Regarding our naive combination of SMCMC-PG, despite being unbiased, it showed to be inefficient when compared to the other SMCMC techniques we proposed.

We modified the SMCMC algorithm by changing its NEE-PT way of obtaining its paths for a more complex BDPT-style path-space operator, achieving an increased performance of the efficiency of the paper's original techniques, for scenes in which paths sampled only from the lights are not important, and with few specular regions. We could observe that for the special case in which several BDPT light vertices are thrown away due to visibility issues (e.g., `Veach-ajar`), but with far from specular materials, even if the BDPT integrator alone is less efficient, our proposed method to combine SMCMC-BDPT improves the efficiency of this special type of scenes, even over the paper's presented techniques [GWH20]. For this type of scenes we were able to improve upon our SMCMC-BDPT by proposing another two mixes that we called SMMC-rBDPT, SMCMC-MMLT, being the best of the two the SMCMC-cMMMLT mix, which prefers sampling long light subpaths and short camera subpaths in an MMLT style for all the tiles.

5.2 Future work

In this section we will present potential research ideas for both the combination of path guiding with MCMC and SMCMC.

5.2.1 MCMC-PG ideas

Modern enhancements of path guiding use a *continuous* distribution to spatially and directionally subdivide the scene [DWL23], thus, we believe that the MCMC-PG combination would benefit from using this new continuous path guiding technique, because the acceptance rate of MCMC algorithms is highly sensitive to abrupt changes (enhanced by the discrete subdivisions).

One thing we noticed in our MCMC-PG implementation is that the training of the tree was more inefficient because of the correlation of MCMC algorithms. Brighter regions of the scene receive more samples of the chain, causing the samples to agglomerate in fewer leaves of the spatial kd-tree. Furthermore, small mutations accumulate the samples in fewer directional quadtree's leaves also hindering the uniform training of the tree. We saved our SD trees in raw uncompressed binary files and noticed the file size of MCMC-PG was smaller than PG when compared iteration by iteration, i.e., MCMC-PG had less nodes. To fix the training problem of the online training MCMC-PG algorithm we suggest as a possibility to use MCMC-PG but with a bidirectional path tracer, taking advantage of the heavy computational effort done in connecting several vertices in just one MCMC iteration to train the tree.

5.2.2 SMCMC ideas

Our introduction of BDPT to SMCMC was lacking a way to get all the paths sampled from the light. We leave for future work, investigate a way to efficiently add this type of paths, being the most naive solution, to just estimate the light image with either only the large steps mutations of the Markov Chain or with an independent chain specialized in just this type of paths.

Another idea is to add a more sophisticated sampling process over the (s,t) techniques to our MMLT shift in SMCMC, as we only tested polynomial distributions (uniform uMMLT, linear lMMLT, quadratic qMMLT and cubic cMMLT) to favor the proposal of longer light subpaths. One possible way of doing this is to learn statistics of the paths on the fly for each tile in a preprocessing step, akin to the learning done on each iteration of path guiding, but in this case we would be learning the distribution of the different (s,t)-BDPT techniques.

In our SMCMC MMLT shift, we chose the depth of our chains randomly, effectively hurting the exploration of less bright depths, specially in zones where lights were visible from the camera. It is still an open problem how to efficiently improve the exploration of the depth space. A naive way is to define few Markov Chains per each tile, for example, 3, handling each one a specific range of depths.

In recent years, a new technique to solve the rendering equation has emerged. This new rendering paradigm is called Reservoir-based Spatio-Temporal Importance Resampling (ReSTIR) [Bit+20; Ouy+21]. This algorithm was first conceived for Direct Illumination. In its Direct Illumination version, it samples M light vertices, then for all the pixels, one possible light vertex out of the M light vertices is chosen using the mathematical technique called weighted reservoir sampling [PJH23]. Each pixel's reservoir stores a weight, the number of samplings that it has performed, and the current chosen light vertex. Then the information of spatially neighboring reservoirs is combined, that is, we are using information about the paths that were sampled by the neighbors (see Figure 5.1). This algorithm has been extended to global illumination. It is then an active research area to study ways in which we could combine ReSTIR sampling with MCMC algorithms and PG. For example there is a paper that combines MCMC mutations with ReSTIR in order to decorrelate the ReSTIR samples [Saw+24]. An interesting idea would be to mix the Stratified MCMC algorithm proposed by Gruson et al. [GWH20], and maybe exchange or even combine the shift mapping process with a ReSTIR-like sampling.

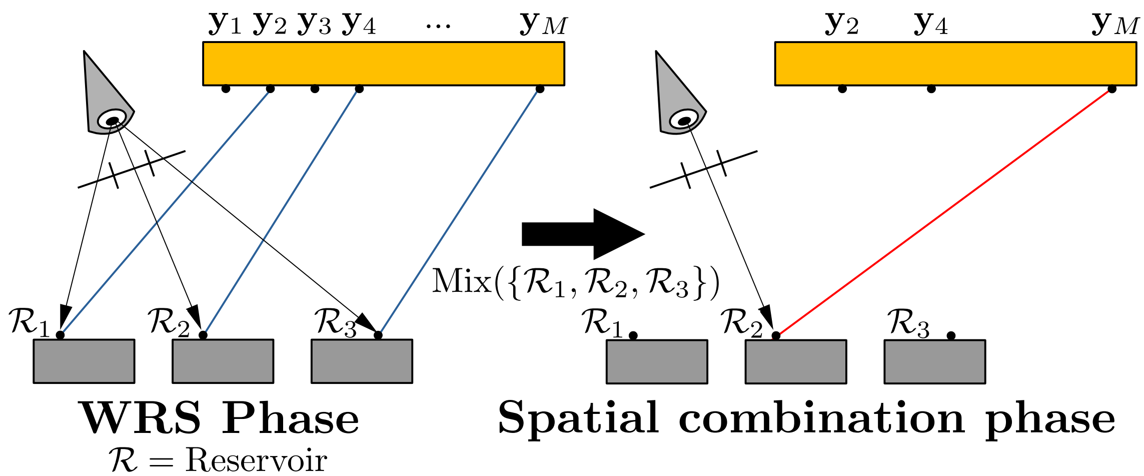


Figure 5.1: ReSTIR DI. Each pixel i has a reservoir $\mathcal{R}_i$ which gets a light sample (blue lines) using weighted reservoir sampling (WRS). Then for all the pixels (in this image, only $\mathcal{R}_2$), the reservoir of the pixel is going to sample another light point using the information about its neighboring reservoirs (red line). We could also use temporal information from previous frames, using a similar procedure. Due to the fact that this algorithm is highly parallelizable and can use the information of previous frames of an animated sequence of renders, it has been used in the rendering of videogames. For example, Cyberpunk 2077 (see Figure 1.1c) has a ReSTIR rendering option available [Wym+23].

Appendix A

Path guiding supplementary algorithms

These algorithms were given taken from the supplementary documents of the Path Guiding paper [MGN17].

Algorithm 11 Refine the Binary Tree Post Training Iteration

procedure REFINEBINARYTREE(k)
  for all leaf in leafNodes **do**
    if NumSamples(leaf) > c $\cdot \sqrt{2^k}$ **then**
      Subdivide(leaf)
    end if
  end for
end procedure
procedure SUBDIVIDE(leaf)
  numSamples ← NumSamples(leaf)
  children ← SplitAlternatingAxis(leaf)
  SetNumSamples(children, numSamples / 2)
end procedure

Algorithm 12 Refine a Quadtree Post Training Iteration

procedure REFINEQUADTREE($quadtree$)
  $\Phi \leftarrow$ FLUX($quadtree$)
  for each node in quadtree **do**
    if FLUX($node$)$/\Phi \leq \rho$ **then**
      PRUNECHILDREN($node$)
    else if ISLEAF($node$) **then**
      SUBDIVIDE($node$)
    end if
  end for
end procedure
procedure SUBDIVIDE($leaf$)
  $\Phi_n \leftarrow$ FLUX($leaf$)
  $children \leftarrow$ SPLITQUADTREE($leaf$)
  SETFLUX($children, \Phi_n/4$)
end procedure

where c and ρ are constants defined by the user, the paper recommends $c = 12000$ and $\rho = 0.01$; the `Refine` function at equation 2.132 implies the refinement of the binary tree and also the refinement of all the quad trees on its leaves, when a binary tree is subdivided the quadtree is deleted from the node that is not a leaf anymore and copied two times into the two new nodes.

Algorithm 13 Sampling and Computing PDF from a Quadtree

```
procedure SampleQuadtree(node)
    if IsLeaf(node) then
        return UniformRandomPositionIn(node)
    else
        child ← SampleChildByEnergy(node)
        return SampleQuadtree(child)
    end if
end procedure
procedure PdfQuadtree(node, ω)
    if IsLeaf(node) then
        return 1/(4π)   ▷ The domain is between cos θ ∈ [−1, 1] and φ ∈ [0, 2π] , i.e. (2)(2π) = 4π
    else
        child ← GetChild(node, ω)
        α ← 4· Flux(child) / Flux(node) ▷ 4 because continuous integral of an array of 4 elements.
        return α· PdfQuadtree(child, ω)
    end if
end procedure
```

Index

Bibliography

[Met+53] Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. “Equation of State Calculations by Fast Computing Machines”. In: *The Journal of Chemical Physics* 21.6 (June 1953), pp. 1087–1092. ISSN: 0021-9606. DOI: 10.1063/1.1699114.

[Has70] W. K. Hastings. “Monte Carlo Sampling Methods Using Markov Chains and Their Applications”. In: *Biometrika* 57 (1970), pp. 97–109.

[Kaj86] James T. Kajiya. “The rendering equation”. In: *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’86. New York, NY, USA: Association for Computing Machinery, 1986, pp. 143–150. ISBN: 0897911962. DOI: 10.1145/15922.15902.

[Bil95] Patrick Billingsley. *Probability and measure*. 3rd edition. John Wiley & Sons, 1995. ISBN: 0-471-00710-2.

[VG95a] Eric Veach and Leonidas Guibas. “Bidirectional Estimators for Light Transport”. In: *Photorealistic Rendering Techniques*. Ed. by Georgios Sakas, Stefan Müller, and Peter Shirley. Berlin, Heidelberg: Springer Berlin Heidelberg, 1995, pp. 145–167. ISBN: 978-3-642-87825-1.

[VG95b] Eric Veach and Leonidas J. Guibas. “Optimally combining sampling techniques for Monte Carlo rendering”. In: *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’95. New York, NY, USA: Association for Computing Machinery, 1995, pp. 419–428. ISBN: 0897917014. DOI: 10.1145/218380.218498.

[Nor97] J. R. Norris. *Markov Chains*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 1997. ISBN: 978-0-521-63396-3.

[Vea97] Eric Veach. “Robust Monte Carlo Methods for Light Transport Simulation”. PhD thesis. Stanford University, 1997.

[VG97] Eric Veach and Leonidas J. Guibas. “Metropolis Light Transport”. In: *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*. Addison-Wesley, Aug. 1997, pp. 65–76.

[LW98] Eric Lafortune and Yves Willems. “Bi-Directional Path Tracing”. In: *Proceedings of Third International Conference on Computational Graphics and Visualization Techniques (Compugraphics’* 93 (Jan. 1998).

[Kel+02] Csaba Kelemen, László Szirmay-Kalos, György Antal, and Ferenc Csonka. “A Simple and Robust Mutation Strategy for the Metropolis Light Transport Algorithm”. In: *Computer Graphics Forum* 21.3 (2002), pp. 531–540. DOI: 10.1111/1467-8659.t01-1-00703.

[Ros06] Jeffery S. Rosenthal. *A first look at rigorous probability theory*. 2nd edition. World Scientific Publishing, 2006. ISBN: 978-9812703712.

[RP07] M. Rafat and P. L. A. Popelier. “Visualization and integration of quantum topological atoms by spatial discretization into finite elements”. In: *Journal of Computational Chemistry* 28.16 (2007), pp. 2602–2617. DOI: 10.1002/jcc.20767.

[DSH08] Ishaan L. Dalal, Deian Stefan, and Jared Harwayne-Gidansky. “Low discrepancy sequences for Monte Carlo simulations on reconfigurable platforms”. In: *2008 International Conference on Application-Specific Systems, Architectures and Processors*. 2008, pp. 108–113. DOI: 10.1109/ASAP.2008.4580163.

[Fus+09] P Fusar-Poli, A Placentino, F Carletti, P Landi, P Allen, S Surguladze, F Benedetti, M Abbamonte, R Gasparotti, F Barale, J Perez, P McGuire, and P Politi. "Functional atlas of emotional faces processing: a voxel-based meta-analysis of 105 functional magnetic resonance imaging studies". In: *Journal of psychiatry and neuroscience* 34.6 (Nov. 2009), pp. 418–432.

[KKK09] Shinya Kitaoka, Yoshifumi Kitamura, and Fumio Kishino. "Replica Exchange Light Transport". In: *Computer Graphics Forum* 28.8 (2009), pp. 2330–2342. DOI: 10.1111/j.1467-8659.2009.01540.x.

[MT09] Sean Meyn and Richard L. Tweedie. *Markov Chains and Stochastic Stability*. 2nd edition. Cambridge University Press, 2009. ISBN: 978-0521731829.

[HH10] Jared Hoberock and John C. Hart. "Arbitrary Importance Functions for Metropolis Light Transport". In: *Computer Graphics Forum* 29.6 (2010), pp. 1993–2003. DOI: 10.1111/j.1467-8659.2010.01713.x.

[Jak10] Wenzel Jakob. *Mitsuba renderer*. http://www.mitsuba-renderer.org. 2010.

[DS11] Morris H. DeGroot and Mark J. Schervish. *Probability and statistics*. 4th edition. Pearson, 2011. ISBN: 978-0321500465.

[HJ11] Toshiya Hachisuka and Henrik Wann Jensen. "Robust adaptive photon tracing using photon path visibility". In: *ACM Trans. Graph.* 30.5 (Oct. 2011). ISSN: 0730-0301. DOI: 10.1145/2019627.2019633.

[Vin+11] F H Vincent, T Paumard, E Gourgoulhon, and G Perrin. "GYOTO: a new general relativistic ray-tracing code". In: *Classical and Quantum Gravity* 28.22 (Oct. 2011), p. 225011. DOI: 10.1088/0264-9381/28/22/225011.

[Kel13] Alexander Keller. "Quasi-Monte Carlo Image Synthesis in a Nutshell". In: *Monte Carlo and Quasi-Monte Carlo Methods 2012*. Ed. by Josef Dick, Frances Y. Kuo, Gareth W. Peters, and Ian H. Sloan. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 213–249. ISBN: 978-3-642-41095-6.

[ZS13] Károly Zsolnai and László Szirmay-Kalos. "Automatic Parameter Control for Metropolis Light Transport ". In: *Eurographics 2013 - Short Papers*. The Eurographics Association, 2013. DOI: 10.2312/conf/EG2013/short/053-056.

[HKD14] Toshiya Hachisuka, Anton S. Kaplanyan, and Carsten Dachsbacher. "Multiplexed metropolis light transport". In: *ACM Trans. Graph.* 33.4 (July 2014). ISSN: 0730-0301. DOI: 10.1145/2601097.2601138.

[Vor+14] Jiří Vorba, Ondřej Karlík, Martin Šik, Tobias Ritschel, and Jaroslav Křivánek. "Online Learning of Parametric Mixture Models for Light Transport Simulation". In: *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2014)* 33.4 (Aug. 2014).

[Bri15] Robert Bridson. *Fluid Simulation for Computer Graphics*. 2nd edition. CRC Press, 2015. ISBN: 978-1482232837.

[Pop+15] Stefan Popov, Ravi Ramamoorthi, Frédo Durand, and George Drettakis. "Probabilistic Connections for Bidirectional Path Tracing". In: *Computer Graphics Forum (Proceedings of the Eurographics Symposium on Rendering)* 34.4 (2015).

[Bit16] Benedikt Bitterli. *Rendering resources*. https://benedikt-bitterli.me/resources/. 2016.

[Hec16] Eugene Hecht. *Optics*. 5th edition. Pearson, 2016. ISBN: 978-1292096933.

[MS16] Steve Marschner and Peter Shirley. *Fundamentals of computer graphics*. 4th edition. CRC Press, 2016. ISBN: 978-1-4822-2941-7.

[PJH16] Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically Based Rendering: From Theory to Implementation (3rd ed.)* 3rd. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., Oct. 2016, p. 1266. ISBN: 9780128006450.

[Bit+17] Benedikt Bitterli, Wenzel Jakob, Jan Novák, and Wojciech Jarosz. "Reversible Jump Metropolis Light Transport Using Inverse Mappings". In: *ACM Trans. Graph.* 37.1 (Oct. 2017). ISSN: 0730-0301. DOI: 10.1145/3132704.

[MGN17] Thomas Müller, Markus Gross, and Jan Novák. "Practical Path Guiding for Efficient Light-Transport Simulation". In: *Comput. Graph. Forum* 36.4 (July 2017), pp. 91–100. ISSN: 0167-7055. DOI: 10.1111/cgf.13227.

[Ots+17] Hisanari Otsu, Anton S. Kaplanyan, Johannes Hanika, Carsten Dachsbacher, and Toshiya Hachisuka. "Fusing state spaces for markov chain Monte Carlo rendering". In: *ACM Trans. Graph.* 36.4 (July 2017). ISSN: 0730-0301. DOI: 10.1145/3072959.3073691.

[Cha+18] Chakravarty Reddy Alla Chaitanya, Laurent Belcour, Toshiya Hachisuka, Simon Premoze, Jacopo Pantaleoni, and Derek Nowrouzezahrai. "Matrix Bidirectional Path Tracing". In: *Eurographics Symposium on Rendering - Experimental Ideas & Implementations.* Ed. by Wenzel Jakob and Toshiya Hachisuka. The Eurographics Association, 2018. ISBN: 978-3-03868-068-0. DOI: 10.2312/sre.20181169.

[Hua+19] Binh-Son Hua, Adrien Gruson, Victor Petitjean, Matthias Zwicker, Derek Nowrouzezahrai, Elmar Eisemann, and Toshiya Hachisuka. "A Survey on Gradient-Domain Rendering". In: *Computer Graphics Forum* 38.2 (2019), pp. 455–472. DOI: 10.1111/cgf.13652.

[Sch19] Dominik Schraml. "Physically based synthetic image generation for machine learning: a review of pertinent literature". In: *Photonics and Education in Measurement Science 2019.* Ed. by Maik Rosenberger, Paul-Gerald Dittrich, and Bernhard Zagar. Vol. 11144. International Society for Optics and Photonics. SPIE, 2019, 111440J. DOI: 10.1117/12.2533485.

[Vor+19] Jiří Vorba, Johannes Hanika, Sebastian Herholz, Thomas Müller, Jaroslav Křivánek, and Alexander Keller. "Path Guiding in Production". In: *ACM SIGGRAPH 2019 Courses.* SIGGRAPH '19. Los Angeles, California: ACM, 2019, 18:1–18:77. ISBN: 978-1-4503-6307-5. DOI: 10.1145/3305366.3328091.

[Bit+20] Benedikt Bitterli, Chris Wyman, Matt Pharr, Peter Shirley, Aaron Lefohn, and Wojciech Jarosz. "Spatiotemporal reservoir resampling for real-time ray tracing with dynamic direct lighting". In: *ACM Trans. Graph.* 39.4 (Aug. 2020). ISSN: 0730-0301. DOI: 10.1145/3386569.3392481.

[GWH20] Adrien Gruson, Rex West, and Toshiya Hachisuka. "Stratified Markov Chain Monte Carlo Light Transport". In: *Computer Graphics Forum* 39.2 (2020), pp. 351–362. DOI: 10.1111/cgf.13935.

[Sch+20] R. Schofield, L. King, U. Tayal, I. Castellano, J. Stirrup, F. Pontana, J. Earls, and E. Nicol. "Image reconstruction: Part 1 – understanding filtered back projection, noise and image acquisition". In: *Journal of Cardiovascular Computed Tomography* 14.3 (2020), pp. 219–225. ISSN: 1934-5925. DOI: 10.1016/j.jcct.2019.04.008.

[ZD20] Tobias Zirr and Carsten Dachsbacher. "Path differential-informed stratified MCMC and adaptive forward path sampling". In: *ACM Trans. Graph.* 39.6 (Nov. 2020). ISSN: 0730-0301. DOI: 10.1145/3414685.3417856.

[Ouy+21] Y. Ouyang, S. Liu, M. Kettunen, M. Pharr, and J. Pantaleoni. "ReSTIR GI: Path Resampling for Real-Time Path Tracing". In: *Computer Graphics Forum* 40.8 (2021), pp. 17–29. DOI: 10.1111/cgf.14378.

[Ros22] Sheldon M. Ross. *Simulation.* 6th edition. Academic Press, 2022. ISBN: 978-0323857390.

[DWL23] Honghao Dong, Guoping Wang, and Sheng Li. "Neural Parametric Mixtures for Path Guiding". In: *SIGGRAPH '23 Conference Proceedings.* Los Angeles, CA, USA, 2023. DOI: 10.1145/3588432.3591533.

[PJH23] Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically Based Rendering: From Theory to Implementation.* 4th. The MIT Press., Oct. 2023, p. 1312. ISBN: 978-0262048026.

[Ros23] Sheldon M. Ross. *Introduction to Probability Models.* 13th edition. Academic Press, 2023. ISBN: 978-0443187612.

[Wym+23] Chris Wyman, Markus Kettunen, Daqi Lin, Benedikt Bitterli, Cem Yuksel, Wojciech Jarosz, and Pawel Kozlowski. "A Gentle Introduction to ReSTIR Path Reuse in Real-Time". In: *ACM SIGGRAPH 2023 Courses.* SIGGRAPH '23. Los Angeles, California: Association for Computing Machinery, 2023. ISBN: 9798400701450. DOI: 10.1145/3587423.3595511.

[Saw+24] Rohan Sawhney, Daqi Lin, Markus Kettunen, Benedikt Bitterli, Ravi Ramamoorthi, Chris Wyman, and Matt Pharr. "Decorrelating ReSTIR Samplers via MCMC Mutations". In: *ACM Trans. Graph.* 43.1 (Jan. 2024). ISSN: 0730-0301. DOI: 10.1145/3629166.

Anti-plagiarism report and pledge of originality

Enrique José Vega Equihua

Efficient Markov Chain Monte Carlo methods using path-space operators and guiding_pdf

Universidad Michoacana de San Nicolás de Hidalgo

Detalles del documento

Identificador de la entrega

trn:oid:::3117:431478289

Fecha de entrega

18 feb 2025, 1:44 p.m. GMT-6

Fecha de descarga

18 feb 2025, 1:50 p.m. GMT-6

Nombre de archivo

Efficient Markov Chain Monte Carlo methods using path-space operators and guiding..pdf

Tamaño de archivo

82.3 MB

113 Páginas

49,449 Palabras

234,432 Caracteres

Formato de Declaración de Originalidad y Uso de Inteligencia Artificial

Coordinación General de Estudios de Posgrado
Universidad Michoacana de San Nicolás de Hidalgo

A quien corresponda,

Por este medio, quien abajo firma, bajo protesta de decir verdad, declara lo siguiente:

- Que presenta para revisión de originalidad el manuscrito cuyos detalles se especifican abajo.
- Que todas las fuentes consultadas para la elaboración del manuscrito están debidamente identificadas dentro del cuerpo del texto, e incluidas en la lista de referencias.
- Que, en caso de haber usado un sistema de inteligencia artificial, en cualquier etapa del desarrollo de su trabajo, lo ha especificado en la tabla que se encuentra en este documento.
- Que conoce la normativa de la Universidad Michoacana de San Nicolás de Hidalgo, en particular los Incisos IX y XII del artículo 85, y los artículos 88 y 101 del Estatuto Universitario de la UMSNH, además del transitorio tercero del Reglamento General para los Estudios de Posgrado de la UMSNH.

Datos del manuscrito que se presenta a revisión		
Programa educativo	Maestría en Ciencias en Ingeniería Eléctrica	
Título del trabajo	Efficient Markov Chain Monte Carlo methods using path-space operators and guiding	
	Nombre	**Correo electrónico**
Autor/es	Enrique José Vega Equihua	1234657k@umich.mx
Director	Luis Eduardo Gamboa Guzmán	luis.gamboa@umich.mx
Codirector	Adrien Gruson	adrien.gruson@etsmtl.ca
Coordinador del programa	Norberto García Barriga	norberto.garcia@umich.mx

Uso de Inteligencia Artificial		
Rubro	**Uso (sí/no)**	**Descripción**
Asistencia en la redacción	No	
Traducción al español	No	
Traducción a otra lengua	No	
Revisión y corrección de estilo	No	

Formato de Declaración de Originalidad y Uso de Inteligencia Artificial

Coordinación General de Estudios de Posgrado
Universidad Michoacana de San Nicolás de Hidalgo

Uso de Inteligencia Artificial		
Rubro	**Uso (sí/no)**	**Descripción**
Análisis de datos	No	
Búsqueda y organización de información	No	
Formateo de las referencias bibliográficas	No	
Generación de contenido multimedia	No	
Otro	No	

Datos del solicitante	
Nombre y firma	Enrique José Vega Equihua
Lugar y fecha	Morelia Michoacán Febrero 2025