UNIVERSITAT DE BARCELONA

Facultat de Matemàtiques
i Informàtica

DOBLE GRAU DE MATEMÀTIQUES I ENGINYERIA INFORMÀTICA

Treball final de grau

A Gaussian Process-based Approach to Rendering

Autor: Masa Zsanett Gulyás

Director: Dr. Ricardo Marques
Realitzat a: Departament de Matemàtiques i Informàtica

Barcelona, 24 de gener de 2023

Contents

Abstract iii

Resum iii

Resumen iv

Acknowledgements v

1 Introduction 1
- 1.1 Background . . . 2
- 1.2 Objectives . . . 2
- 1.3 Planning . . . 3

2 Theoretical Foundations 5
- 2.1 Monte Carlo and variance reduction techniques . . . 5
  - 2.1.1 Importance Sampling . . . 6
- 2.2 Bayesian Monte Carlo . . . 7
  - 2.2.1 Bayesian Regression . . . 8
  - 2.2.2 Bayesian Monte Carlo quadrature equations . . . 10
- 2.3 Physically Based Rendering . . . 11

3 Bayesian Monte Carlo for Physically Based Rendering 13
- 3.1 Matching of the Rendering Equation with the Bayesian Quadrature Equations . . . 13
- 3.2 Implementation . . . 14
  - 3.2.1 Workbench . . . 14
  - 3.2.2 Renderer . . . 15
  - 3.2.3 Classical Monte Carlo . . . 16
  - 3.2.4 Bayesian Monte Carlo . . . 18
  - 3.2.5 Specifying a more informed prior mean function in the Bayesian Monte Carlo estimate . . . 23

3.2.6 Experiments performed with the renderer to validate the integrators . . . 25
3.2.7 Implementation of error measure and image comparison . . 27

4 Results and Discussion **31**
4.1 Performance of Monte Carlo vs Bayesian Monte Carlo 31
4.2 Improving Bayesian Monte Carlo for Rendering: Run Time and Error Analysis 33
4.3 Use of a Constant Prior in Bayesian Monte Carlo 33
4.4 Discussion of Results and Future Work 35

5 Conclusion **37**

Abstract

Many physically-based image rendering algorithms use the illumination integral to determine the color of each pixel in the rendered image. This integral has a component that can be sampled but has no known analytical expression, so it cannot be computed directly and must be evaluated with approximation methods. Among these we can find the Monte Carlo (MC) and the Bayesian Monte Carlo (BMC) integration methods. MC integration consists in defining a random variable such that its expected value is the solution to the integral, and then repeatedly sampling that random variable to estimate the true value. In contrast, BMC models the function to be integrated using a Gaussian Process, which allows for the incorporation of prior information. While MC is conceptually simple and straightforward to implement, it has a slower convergence rate compared to BMC. BMC, on the other hand, allows for better estimates with the same number of samples, even without prior information, by taking into account all available information about the samples, in particular the covariance of the sample locations. In this thesis, I implemented the MC and the BMC algorithms for integration, and compared their performances in two settings: for the estimation of a single integral with a known true value and for image rendering using the root-mean-squared error (RMSE). My results showed that the error of BMC converged much faster compared to MC in both settings, mirroring the existing literature on the topic. In addition, I experimented with the use of a constant prior in the BMC method, and found promising results for single integral estimation, although further work is needed to successfully apply this finding to image rendering.

2020 Mathematics Subject Classification. 60G15, 11K45

Resum

Molts algoritmes de renderització d'imatges basades en la física utilitzen l'integral d'il·luminació per determinar el color de cada píxel en la imatge renderitzada. Aquesta integral té un component que es pot mostrejar, però que no té una expressió analítica coneguda, per la qual cosa no es pot calcular directament i s'ha d'avaluar amb mètodes d'aproximació. Entre aquests es troben els mètodes d'integració de Monte Carlo (MC) i de Bayesian Monte Carlo (BMC). La integració MC consisteix a definir una variable aleatòria, el valor esperat de la qual és la solució a l'integral, i després mostrejar aquesta variable aleatòria repetidament per obtenir una estimació del valor real. D'altra banda, BMC fa servir un procés gaussià per modelar la funció a integrar i permet la introducció d'informació prèvia sobre la funció. MC és conceptualment simple i fàcil d'implementar en comparació amb BMC, però té una taxa de convergència notablement més alta. BMC permet aconseguir millors estimacions amb el mateix nombre de mostres fins i tot sense informació prèvia, ja que té en compte tota la informació disponible sobre les mostres, en particular la covariància de les posicions de les mostres. En aquesta tesi, he implementat els algoritmes MC i BMC per a la integració i he comparat el seu rendiment en dos entorns: per a l'estimació d'una única integral amb un valor real conegut i per a la renderització d'imatges fent servir l'error quadràtic mitjà (RMSE). Els meus resultats van mostrar que l'error de BMC va convergir molt més ràpid en comparació amb MC en tots dos entorns, reflectint la literatura existent sobre el tema. A més, he experimentat amb l'ús d'una funció prèvia constant en el mètode BMC i he obtingut resultats prometedors per a l'estimació d'una única integral, encara que es necessita més treball per aplicar aquesta troballa amb èxit a la renderització d'imatges.

Resumen

Muchos algoritmos de renderizado de imágenes basadas en la física utilizan la integral de iluminación para determinar el color de cada píxel en la imagen renderizada. Esta integral tiene un componente que se puede muestrear, pero que no tiene una expresión analítica conocida, por lo que no se puede calcular directamente y se tiene que evaluar con métodos de aproximación. Entre estos se encuentran los métodos de integración de Monte Carlo (MC) y de Monte Carlo Bayesiano (BMC). La integración MC consiste en definir una variable aleatoria cuyo valor esperado es la solución a la integral, y luego muestrear esa variable aleatoria repetidamente para obtener una estimación del verdadero valor. Por otro lado, BMC utiliza un Proceso Gaussiano para modelar la función a integrar, y permite la introducción de información previa sobre la función. MC es conceptualmente simple y fácil de implementar en comparación con BMC, pero tiene una tasa de convergencia notablemente más alta. BMC permite obtener mejores estimaciones con el mismo número de muestras incluso sin información previa, ya que tiene en cuenta toda la información disponible sobre las muestras, en particular la covarianza de las posiciones de las muestras. En esta tesis, he implementado los algoritmos MC y BMC para la integración, y he comparado sus rendimientos en dos entornos: para la estimación de una única integral con un valor verdadero conocido y para el renderizado de imágenes utilizando el error cuadrático medio (RMSE). Mis resultados muestran que el error de BMC converge mucho más rápido en comparación con MC en ambos entornos, reflejando la literatura existente sobre el tema. Además, he experimentado con el uso de una función previa constante en el método BMC, y he obtenido resultados prometedores para la estimación de una única integral, aunque se necesita más trabajo para aplicar este hallazgo con éxito al renderizado de imágenes.

Acknowledgements

I would like to express my deepest gratitude to my advisor, Ricardo Marques, for his patience, dedication and help in achieving my goals with this thesis. He kept me on track with weekly meetings and gave me an awesome masterclass on Gaussian Processes, a topic that I had struggled to understand at first but now I find fascinating. Thank you for your continued support!

I would also like to acknowledge the Universitat de Barcelona for the resources provided throughout my years here. I learned a lot from many excellent lecturers, and I specifically want to acknowledge Anna Puig for her engaging Computer Graphics lectures, which sparked my initial interest in the field.

Finally, I want to give my heartfelt gratitude to Rubén for his constant encourgament, support, and his very motivating motivational speeches, which kept me going when I felt overwhelmed by university and work. Thank you also for teaching me all your cool LaTeX tricks.

Chapter 1

Introduction

The synthesis or *rendering* of photorealistic images from a virtual model is a central focus in the field of Computer Graphics. Creating realistic images involves simulating the way light interacts with the surfaces in the virtual scene and determining how much light reaches the virtual camera from the visible parts of the scene. The amount of light that reaches the virtual camera from a point in the scene can be obtained using the *illumination integral*, first formulated by Kajiya [7]. The illumination integral, as presented in this thesis 2.16, comprises terms whose analytical expressions are not known, making it intractable to compute analytically. Therefore, its value must be approximated using alternative methods.

Monte Carlo methods are a broad category of mathematical and computational techniques that rely on statistical sampling to simulate various phenomena or determine the values of functions. Monte Carlo methods can be used to evaluate an integral by defining a random variable whose expected value is the solution to the integral. By drawing samples of this random variable and averaging them, an estimate of the expected value is obtained, which serves as an approximation to the integral's real solution. It is worth noting that Monte Carlo integration should only be used when it is not feasible or possible to compute the integral analytically. An example of this is the illumination integral mentioned above.

Although Monte Carlo methods have been used in computer rendering since the 1980s, such as in the works of Arvo [1], Cook [2], and Kajiya [7], they have been criticised for for their dependence on the specific sampling distribution chosen to retrieve samples and for discarding information about the proximity of the samples [12], which leads to a relatively slow convergence rate to the real result.

In 2002, Ghahramani and Rasmussen [5] introduced a Bayesian approach to integral estimation that incorporates prior knowledge and explicitly takes into consideration the proximity of sample locations to achieve more accurate results. This method, known as Bayesian Monte Carlo, resembles traditional Monte Carlo

methods in its use of sampling for integral estimation. The method uses a Gaussian process to model the unknown function being integrated.

The main emphasis of this thesis is on the application of Bayesian Monte Carlo for image rendering. However, the traditional Monte Carlo method is also discussed and is used as reference with the results obtained from Bayesian Monte Carlo.

1.1 Background

As the main sources in this thesis I used the following two textbooks: *Advanced Global Illumination* [3] and *Efficient quadrature rules for illumination integrals* [11]. In *Advanced Global Illumination*, there is extensive background on the rendering equation and how it is derived from the physics of light transport, as well as a detailed explanation on Monte Carlo methods and variance reduction techniques, both of which I cover in Section 2.3 and 2.1, respectively. On the other hand, *Efficient quadrature rules for illumination integrals* covers Bayesian Monte Carlo Methods extensively, which I talk about in detail in Section 2.2. These books draw on the research of many authors also cited here. The rendering equation was proposed for the first time by Kajiya [7], in a publication where he suggests the use of a Monte Carlo method for solving it, and presents a form of variance reduction called Hierachical sampling. Later, Ghahramani and Rasmussen [5] proposed the Bayesian Monte Carlo method for evaluating integrals as an alternative to classical Monte Carlo methods. They show in this publication that Bayesian Monte Carlo outperforms classical Monte Carlo despite any importance sampling methods applied to the latter.

The use of Bayesian Monte Carlo for physically-based rendering remains an active area of research. In [10], a more general Bayesian Monte Carlo solution is proposed, which addresses non-diffuse BRDFs whereas until then only the perfect diffuse case had been considered. Additionally, a fast method for determining hyperparameters is proposed, avoiding the need to learn them for each individual BRDF. In [8], a theoretical and experimental analysis of Gaussian Processes as a tool for approximating the incident radiance function on a sphere is performed. The publication also presents a method for efficient computation and rotation of spherical harmonics coefficients.

1.2 Objectives

The main objective of this thesis is to design, implement, and evaluate a Bayesian Monte Carlo algorithm for use in an image rendering application. Specifically, the

following goals will be accomplished:

1. Create a Python script for prototyping and displaying the error of integral estimators as the number of samples increases by evaluating the integral of a specific function over the unit hemisphere.

2. Develop a basic rendering application using Python by incorporating different integrator classes to compute the value of each pixel, based on an existing prototype.

3. Create a classical Monte Carlo algorithm, visualize its error convergence relative to the number of samples, and develop a "Monte Carlo Integrator" for the rendering application.

4. Design and implement a Bayesian Monte Carlo algorithm, visualize its error convergence relative to the number of samples, and develop a "Bayesian Monte Carlo Integrator" for the rendering application.

5. Optimize the performance of the Bayesian Monte Carlo Integrator for faster execution.

6. Investigate the use of a constant prior in the Bayesian Monte Carlo method and its impact on reducing the estimation error.

7. Conduct a comprehensive comparison of the developed methods by evaluating both their estimations of a known integral over the unit hemisphere and the resulting complete rendered images.

1.3 Planning

The planning is presented in the Gantt chart 1. The image is in the Appendix due to its size.

Chapter 2

Theoretical Foundations

2.1 Monte Carlo and variance reduction techniques

Monte Carlo methods are a class of statistical methods that use random sampling to generate approximate solutions to a broad set of problems. In this thesis, we are interested particularly in Monte Carlo integration, that is, using random sampling to approximate an integral. Monte Carlo methods are generally used in cases where obtaining the integral of a function deterministically is not possible or computationally unfeasible. Examples of these cases are functions in higher dimensions or functions that can be sampled but do not have a known analytical formula. In this section, we will go through the Monte Carlo method for integration in detail and prove that its convergence rate is $\frac{1}{\sqrt{N}}$ where N is the number of samples.

If the integral we want to solve using Monte Carlo is denoted by

$$I = \int_a^b f(x)\, dx, \tag{2.1}$$

we can consider the estimator

$$\hat{I} = \frac{1}{N} \sum_{i=1}^{N} \frac{f(x_i)}{p(x_i)}, \tag{2.2}$$

where x_i are randomly selected samples over the domain of the integral with probability distribution $p(x)$. It can be proved that $\hat{I}$ is an unbiased estimator of I by showing that its expected value is equal to I:

Proof.

$$\begin{aligned} E\left[\hat{I}\right] &= E\left[\frac{1}{N}\sum_{i=1}^{N}\frac{f(x_i)}{p(x_i)}\right] = \frac{1}{N}\sum_{i=1}^{N}E\left[\frac{f(x_i)}{p(x_i)}\right] \\ &= \frac{1}{N}N\int\frac{f(x)}{p(x)}p(x)\,dx = \int f(x)\,dx = I. \end{aligned} \tag{2.3}$$

□

Since $\hat{I}$ is unbiased, there is no systemic error in the estimation, but we still need to consider random error due to variance. The variance of $\hat{I}$ is

$$\begin{aligned} V\left[\hat{I}\right] &= V\left[\frac{1}{N}\sum_{i=1}^{N}\frac{f(x_i)}{p(x_i)}\right] = \frac{1}{N^2}\sum_{i=1}^{N}V\left[\frac{f(x_i)}{p(x_i)}\right] \\ &= \frac{1}{N^2}NV\left[\frac{f(x_i)}{p(x_i)}\right] = \frac{1}{N}\left(E\left[\left(\frac{f(x)}{p(x)}\right)^2\right] - E\left[E\left[\frac{f(x)}{p(x)}\right]^2\right]\right) \\ &= \frac{1}{N}\left(\int_a^b\frac{f(x)^2}{p(x)}\,dx\; - I^2\right) := \sigma^2. \end{aligned} \tag{2.4}$$

This shows that the variance σ^2 decreases linearly in inverse proportion to N. The estimator's error, defined as the standard deviation σ, decreases in inverse proportion to $\sqrt{N}$. This is considered to be a slow convergence rate [3, p. 48]; to decrease the error by half, four times as many samples are needed. To improve the convergence rate of Monte Carlo integration, various variance reduction techniques have been developed, such as importance sampling.

2.1.1 Importance Sampling

We can see in 2.4 that aside from the number of samples, the probability distribution from which the samples are taken affects the variance of $\hat{I}$. Importance sampling consists of choosing a $p(x)$ that will hopefully reduce variance based on information that might be available about $f(x)$.

It can be proved that the optimal choice of probability distribution function, which will yield zero variance, is $p(x) = \frac{f(x)}{I}$.

Proof.

$$
\begin{aligned}
V\left[\hat{I}\right] &= \frac{1}{N}\left(\int_a^b \frac{f(x)^2}{p(x)}\,dx \; - I^2\right) = \frac{1}{N}\left(\int_a^b \frac{f(x)^2}{\frac{f(x)}{I}}\,dx \; - I^2\right) \\
&= \frac{1}{N}\left(I\int_a^b f(x)\,dx \; - I^2\right) = \frac{1}{N}\left(I^2 - I^2\right) = 0.
\end{aligned}
\tag{2.5}
$$

□

However the integral I is the result we want to estimate with the Monte Carlo method in the first place, so clearly it is not possible to obtain the optimal $p(x)$. What we can deduce from the formula $p(x) = \frac{f(x)}{I}$ is that a good probability distribution function will closely mimic the shape of the function we are trying to approximate. This way, we can use any information available of the shape of $f(x)$ to choose a $p(x)$ that will yield the least amount of variance.

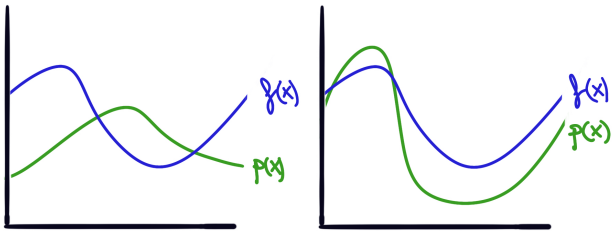


Figure 2.1: Two examples of possible probability density functions $p(x)$. Clearly, the $p(x)$ on the right is better for sampling $f(x)$ because it mimics the shape of $f(x)$, and so it will generate samples more frequently in the segments where $f(x)$ has larger values.

2.2 Bayesian Monte Carlo

In his 1987 paper [12], statistician A. O'Hagan presented two main criticisms to Monte Carlo integration: 1) the result depends not only on the samples and their values, but also on the probability distribution they were sampled from, which violates the Likelihood Principle from a Bayesian point of view, and 2) Monte Carlo does not consider the distance between samples, leading to redundant information in close samples and not enough weight given to isolated samples. A short example that O'Hagan proposes to demonstrate the second objection is as follows:

Suppose we have three samples x_1, x_2 and x_3 such that $x_1 = x_2$. Our estimation would be

$$\hat{I} = \frac{1}{3}\frac{f(x_1)}{p(x_1)} + \frac{1}{3}\frac{f(x_2)}{p(x_2)} + \frac{1}{3}\frac{f(x_3)}{p(x_3)} = \frac{2}{3}\frac{f(x_1)}{p(x_1)} + \frac{1}{3}\frac{f(x_3)}{p(x_3)}. \tag{2.6}$$

It is clearly unreasonable to give twice as much weight to the exact same sample if it happens to be repeated. In practice, it is almost impossible to get the exact same sample twice, but the distance between samples is still valuable information that is discarded in Monte Carlo integration.

In 2002, Ghahramani and Rasmussen reformulated the Monte Carlo integration problem into a Bayesian inference problem [5]. This new algorithm avoids the inconsistencies of classical Monte Carlo, and yields better approximations using the same number of samples. Instead of attempting to approximate the integral directly, Bayesian Monte Carlo first finds an estimate of the unknown function $f(x)$ that can then be either integrated analytically or approximated better than the original function. Below we are going to describe Bayesian Regression in general terms and then delinate the Bayesian Monte Carlo Quadrature equations derived by Ghahramani [5].

2.2.1 Bayesian Regression

Bayesian Regression is an approach for regression that allows approximating a function $f(x)$ using a Gaussian Process given a set of samples of $f(x)$ and allows for the introduction of prior information into the estimation.

Definition 2.1 ([13], Definition 2.1)**.** *A Gaussian Process (GP) is a collection of random variables such that each finite sub-collection of them has a joint Gaussian distribution, which is a normal multivariate distribution.*

Let $f\colon \mathbb{R}^n \to \mathbb{R}$ be the function we want to approximate. Its mean and covariance functions $\bar{f}(x)$ and $k(x, x')$ [13, Equation 2.13] are

$$\bar{f}(x) = \mathbb{E}[f(x)], \quad \text{and} \quad k(x, x') = \mathbb{E}[(f(x) - \bar{f}(x))(f(x') - \bar{f}(x'))].$$

A GP used to approximate $f(x)$ can be characterized solely by its mean and covariance functions [13, Section 2.2]. We denote the Gaussian process of the function $f(x)$ by $\mathcal{GP}(\bar{f}(x), k(x, x'))$ and we write $f(x) \sim \mathcal{GP}(\bar{f}(x), k(x, x'))$.

The random variables of the GP represent the values of the function $f(x)$ for all $x \in \mathrm{Dom}(f)$, where $\mathrm{Dom}(f)$ is usually $\mathbb{R}^n$. To approximate f, we obtain a set of samples $D = \{((x_1, y_1), \ldots, (x_n, y_n))\}$. In this section, we will denote as f_i the value of f on the sample position x_i, i.e., $f(x_i)$.

The way in which we define a Gaussian process implies a *consistency* requirement [13, Section 2.2]. This requisite is fulfilled automatically if we impose that $\text{cov}(Y_p, Y_q) = k(x_p, x_q)$ [11, 13]. From now on, we will impose this condition on all the Gaussian processes we define.

There is a problem with the covariance $k(x, x')$ that we defined previously in 2.2.1, and it is that we cannot compute it because we do not know $f(x)$. Consequently, we must approximate it using a chosen *covariance function*. Some commonly used covariance functions are

- Squared exponential covariance function, defined as

$$k(x, x') = \sigma_f^2 \text{exp}\left(-\frac{1}{2}\frac{\|x - x'\|^2}{l^2}\right), \tag{2.7}$$

  where σ_f^2 and l are hyperparameters, i.e., parameters that we should select based on our observations.

- Sobolev covariance function, defined as

$$k(x, x') = \frac{2^{2s-1}}{s} - \|x - x'\|^{2s-2} \tag{2.8}$$

  where $\|\cdot\|$ is the Euclidean norm and s is the smoothnes paramater.

To improve numerical conditioning on the GP model, we can assume the observations are affected by an i.i.d. Gaussian noise σ_n, modifying the covariance $\text{cov}(Y_p, Y_q) = k(x_p, x_q) + \sigma_n^2 \delta_{pq}$, where $\delta_{pq} = 1$ if $p = q$ and $\delta_{pq} = 0$. We do this because in realistic scenarios, the observed values are affected by several sources of noise, such as the numerical errors introduced by representing numbers in computers.

Prior and posterior of Gaussian Processes. We have seen that Gaussian Processes allow us to model unknown functions $f(x)$ using Gaussian Random Variables for each possible function value $f(x)$. After preprocessing our data and building a Gaussian Process with this information, called **prior** Gaussian Process, our objective is to add the information of the, possibly noisy, samples $D = \{(x_i, Y_i)\}_{i=1}^n$, where $Y_i = f(x_i) + \epsilon_i$, to this prior GP to generate a new Gaussian Process, called the **posterior** Gaussian Process. From now on, we consider that the noise on the observations Y_i ϵ_i is given by i.i.d. Gaussian noise with zero mean and fixed variance σ_n^2 for all i. We condition the prior GP by applying Bayes' rule. According to [13], the resulting posterior process is also a GP with mean and covariance

$$\begin{aligned} \mathbb{E}[f(x)|D] &= \bar{f}(x) + k(x)Q^{-1}(Y - \bar{F}), \\ \text{cov}[f(x), f(x')|D] &= k(x, x') - k(x)^t Q^{-1} k(x'), \end{aligned} \tag{2.9}$$

respectively, where

$$\begin{aligned} k(x) &= (k(x_1, x), \ldots, k(x_n, x))^t, \\ K_{i,j} &= k(x_i, x_j) \text{ with } (i, j) \in [1, n]^2, \\ Q &= (K + {\sigma_n}^2 I_n), \\ Y &= (Y_1, \ldots, Y_n)^t, \\ \bar{F} &= (\bar{f}(x_1), \ldots, \bar{f}(x_n))^t, \end{aligned} \tag{2.10}$$

and where I_n is the identity matrix of dimension n. Note that Y is the vector containing the observations and $\bar{F}$ contains the expected values for the samples from the prior GP.

2.2.2 Bayesian Monte Carlo quadrature equations

Recall that our problem was to estimate the integral (2.1) $I = \int_{\mathbb{R}^d} f(x)dx$. In this chapter, we consider a more general integral of the form

$$I = \int_{\mathbb{R}^d} f(x)p(x)dx, \tag{2.11}$$

where $p(x)$ is known and $f(x)$ is unknown except for a set of samples D. Note that the previous Equation (2.1) is only a special case of Equation (2.11) where $p(x) = 1$. As we saw previously in Subsection 2.2.1, we can compute a posterior GP model of $f(x)$ by using the Bayes' rule over a prior GP given D. Using this posterior GP model to approximate I is known as the *Bayesian Monte Carlo* (BMC) method, which we explore in this section.

Let D be our usual set of samples. Using the previous posterior Gaussian Process for $f(x)$, we compute the *posterior estimate of* I, $\hat{I} = \mathbb{E}(I|D)$ by integrating both sides of the first equation in (2.9), obtaining

$$\hat{I} = \bar{I} + z^t Q^{-1}(Y - \bar{F}), \tag{2.12}$$

where

$$\begin{aligned} \bar{I} &= \int \bar{f}(x)p(x)dx, \\ z &= \int k(x)p(x)dx, \\ Q &= \begin{bmatrix} k(x_1, x_1) & \ldots & k(x_1, x_n) \\ \vdots & \ddots & \vdots \\ k(x_n, x_1) & \ldots & k(x_n, x_n) \end{bmatrix}. \end{aligned} \tag{2.13}$$

according to [4]. It can be shown that the variance of the previous integral estimate (2.12) follows a Gaussian distribution of mean $\hat{I}$ and variance given by

$$\mathrm{Var}(I|D) = \hat{V} - z^t Q^{-1} z, \tag{2.14}$$

where

$$\hat{V} = \iint k(x, x')p(x)p(x')dxdx', \tag{2.15}$$

also according to [4].

The previous variance (2.14) does not depend on the values of the observations Y, as defined in (2.10), in contrast with Monte Carlo integration. In addition, BMC methods are flexible enough to consider arbitrary sampling methods, not depending on the specific pdf from which sample locations are drawn. In particular, sample sets that minimize (2.14) are called *optimal sample sets*, and their use is one of the reasons why BCM outperforms several state-of-the-art methods [11].

2.3 Physically Based Rendering

Figure 2.2: Example of a photorealistic rendering [6].

Physically based rendering is an approach to realistic image synthesis that models how light behaves in the physical world. Image synthesis or rendering means generating images from a model of a virtual three-dimensional scene. Usually this model will include information about objects and light sources that are in the scene, and about the viewpoint from which the scene is observed.

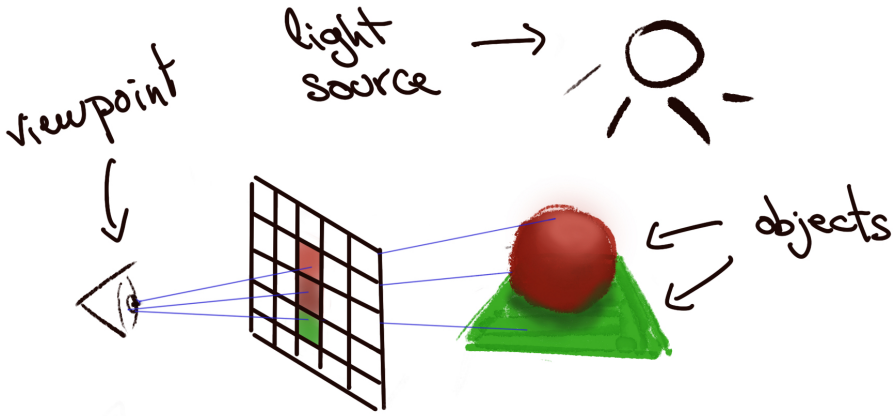


Figure 2.3: Image rendering from a virtual scene.

The aim of realistic image synthesis is to make these images mirror reality as closely as possible. To achieve this, the underlying physical processes regarding the behavior of light must be simulated with maximum accuracy. In particular, given a specific point on the scene, called *shading point*, we compute its color as the exitant radiance value $L_o(\omega_o)$ in the direction of the viewer ω_o. The exitant radiance $L_o(\omega_o)$ at a given shading point in the direction ω_o is computed using the *illumination integral*. This integral is given by [11]

$$L_o(\omega_o) = \int_\Omega L_i(\omega_i)\rho(\omega_i, \omega_0)(\omega_i \cdot n)d\Omega(\omega_i), \tag{2.16}$$

where $L_i(\omega_i)$ is the incident radiance in direction ω_i, Ω is the hemisphere where we evaluate the integral, ρ is the BRDF, and n is the unitary normal vector at the shading point in the surface.

The BRDF, or emphbidirectional reflectance distribution function, defines how light is reflected at an opaque surface, and is given by

$$\rho(\omega_i, \omega_o) = \frac{dL_o(\omega_o)}{dE_i\omega_i} = \frac{dL_o(\omega_o)}{L_i(\omega_i)\cos\theta_i d\omega_i}, \tag{2.17}$$

where E_i is the incident radiant power on the surface per unit surface area and θ_i is the angle between w_i and the surface normal.

On the other hand, radiance is a physical quantity describes how much *radiant power* arrives at, or leaves from, a point on a surface per unit solid angle per unit projected area. Radiant power quantifies the energy that is emitted, arrives to, or flows through a surface per unit of time. For further information on the physical formalization of these concepts, please refer to the textbook [3].

Chapter 3

Bayesian Monte Carlo for Physically Based Rendering

3.1 Matching of the Rendering Equation with the Bayesian Quadrature Equations

Recall the illumination integral (2.16)

$$L_o(\omega_o) = \int_\Omega L_i(\omega_i)\rho(\omega_i, \omega_0)(\omega_i \cdot n)d\Omega(\omega_i),$$

where $L_i(\omega_i)$ is the incident radiance from the direction ω_i, $\rho(\omega_i, \omega_0)$ is the BRDF of the material, and n is the normal vector to the surface.

Recall the Bayesian Monte Carlo method, which attempted to solve the integral (2.11)

$$I = \int_{\mathbb{R}^d} f(x)p(x)dx,$$

where $p(x)$ has a known analytical expression and $f(x)$ does not, but it can be evaluated on any given location x.

Normally, the BRDF has a known analytical expression, so the product $\rho(\omega_i, \omega_0)(\omega_i \cdot n)$ of the illumination integral is considered to be the known part of the Bayesian Monte Carlo method, $p(x)$, and the incident radiance $L_i(\omega_i)$ is the unknown part $f(x)$. Thus, given a set of samples $D = \{L_i(\omega_j) : j \in [1, \ldots, n]\}$ we can approximate the value of $L_o(\omega_o)$ using BMC.

In my implementation, for simplicity, I considered $f(x) = L_i(\omega_i)\rho(\omega_i, \omega_0)(\omega_i \cdot n)$ and $p(x) = 1$, and my samples were of the form $D = \{L_i(\omega_j)\rho(\omega_j, \omega_0)(\omega_j \cdot n) : j \in [1, \ldots, n]\}$.

3.2 Implementation

In this thesis, I implemented the classical Monte Carlo and Bayesian Monte Carlo methods for rendering. The Monte Carlo and Bayesian Monte Carlo estimators were first developed and tested in a prototyping environment which I called *Workbench* (more details in 3.2.1). Once the estimator was validated in the Workbench, it was applied in a python-based rendering environment (more details in 3.2.2), after some minor modifications. In Bayesian Monte Carlo, I made some adjustments known to decrease run time, and finally experimented with adding a constant prior to reduce the error.

3.2.1 Workbench

For each integration method, I fist developed a version in the *Workbench*, a script for estimating the integral of a given function over the unit hemisphere 3.1. Testing an integrator on the Workbench means estimating only one specific integral over the hemisphere instead of the hundreds or thousands required for image rendering. This approach is more convenient for prototyping, fine-tuning and validating new estimators than a direct application in a rendering environment. The Workbench evaluates the error of the methods by comparing their estimations to the real value of the integral (i.e. the reference or ground truth value). This means that in the Workbench we must choose functions the integral of which we can obtain analytically or using another approximation method. In Figure 3.1, a flow diagram of the Workbench application can be observed. The *AppWorkbench* is configured with the function to integrate $f(x)$ and its previously (analytically) computed integral over the unit hemisphere. The methods to compare and a set of sample numbers *n_samples* to use are also specified. For each *n_samples*, the samples are generated accordingly. These samples are used to obtain an approximation of the integral of $f(x)$ with each method. The results are compared against the pre-computed ground truth and the absolute differenced for each method and number of samples is graphed with *pyplot*. The *AppWorkbench* also allows generating each batch of samples n times and averaging the error in order to obtain more stable results.

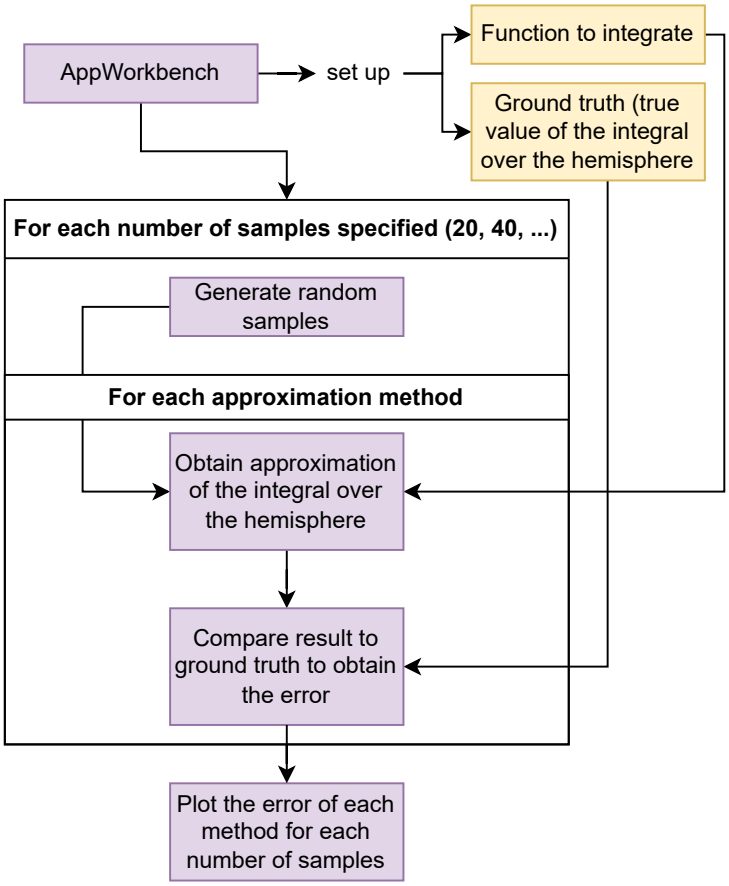


Figure 3.1: Workbench application flow diagram.

3.2.2 Renderer

After having tested a method in the Workbench, it can then be modified slightly for image rendering. For each method, I implemented an *Integrator* class, which is initialized from the renderer application 3.2 to compute the color of each pixel. The renderer application sets up the virtual scene to be rendered, an iterates over all pixels using the Integrator to obtain their values. To evaluate the error of rendered images, they were compared to a reference image 3.13 obtained with the Monte Carlo method 2.1 using 3000 samples per pixel. Figure 3.2 shows a flow diagram of the rendering application. The *AppRenderer* initializes the integrator of the method we want to use for approximating the integral (Monte Carlo, Bayesian Monte Carlo). It initializes the virtual scene and assigns it to the integrator, and then it calls the integrator's pre-render and render methods. In those methods, the integrator iterates over each pixel, traces a ray to the closest hit on the scene,

and approximates the illumination integral to obtain the pixel's color. Finally, the rendered image is saved locally and displayed.

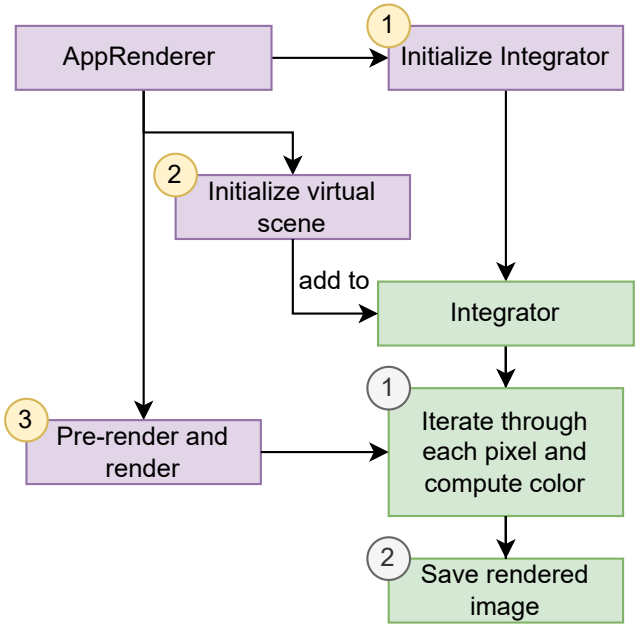


Figure 3.2: Renderer application flow diagram.

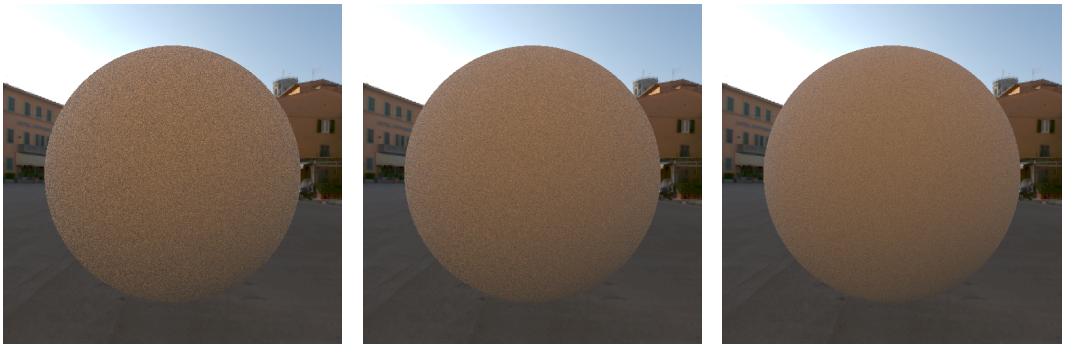
3.2.3 Classical Monte Carlo

Before attempting to use Bayesian Monte Carlo for computing the illumination integral, I implemented a classical Monte Carlo integrator 2.1. This method is less advanced than Bayesian Monte Carlo and it allowed me to get a better understanding on probabilistic methods for image rendering.

To approximate an integral using a Monte Carlo method, first we have to generate samples over the hemisphere. In our implementation 3.4, this is done by obtaining two samples from a uniform distribution, $u1, u2 \sim \mathcal{U}(0,1)$, and then converting them to the *zenith angle* and *azimuth angle*, respectively.

The conversion to hemispherical coordinates was done using the *Lambert cylindrical projection*, which is the most common hemispherical projection used in computer graphics [11, p.19]. A $(x,y) \in (0,1) \times (0,1)$ can be projected onto the unit hemisphere using the Lambert cylindrical projection as follows:

$$\begin{aligned} \phi &= 2\pi x, \\ \theta &= \arccos(1-2y), \end{aligned} \tag{3.1}$$



(a) 20 spp (b) 40 spp (c) 80 spp

Figure 3.3: An example of images rendered with classical Monte Carlo varying numbers of samples per pixel (spp). Note that, as the number of spp increases, the visible noise resulting from estimation error is progressively reduced. The RMSE of each image is 0.02202, 0.01561 and 0.01274, for (a), (b) and (c), respectively. More details regarding the computation of the RMSE are given in Sec. 3.2.7, and a precise error comparison with other methods can be found in the results section.

where (ϕ, θ) are the hemispherical coordinates. This results in a uniformly distributed random direction on the unit hemisphere, which must be rotated according to the surface normal of the object which color we are computing.

After obtaining the hemispherical sample positions $\omega_1, \omega_2, \ldots, \omega_n$, we evaluate the integrand $f(\omega_j) = L_i(\omega_j)\rho(\omega_j)\cos(\omega_j)$ and the probability distribution $\text{pdf}(\omega_j)$ on each. To constitute the integrand, we obtain the BRDF ρ from the properties of the object, and the incident radiance $L_i(\omega_j)$ by tracing another ray in the direction ω_j and obtaining the BRDF of the object if there is another hit or the background color otherwise. As for $\text{pdf}(\omega)$, since our chosen PDF is uniform, the result will be the same for all ω_j: $p(\omega_j) = \frac{1}{2\pi}$.

Finally, when we have the values over $f(\omega)$ and $p(\omega)$ of all samples, we can compute the Monte Carlo estimation, for each color channel separately, using the estimator $\hat{I} = \frac{1}{N}\sum_{i=1}^{N}\frac{f\omega_j)}{p(\omega_j)}$ (2.2).

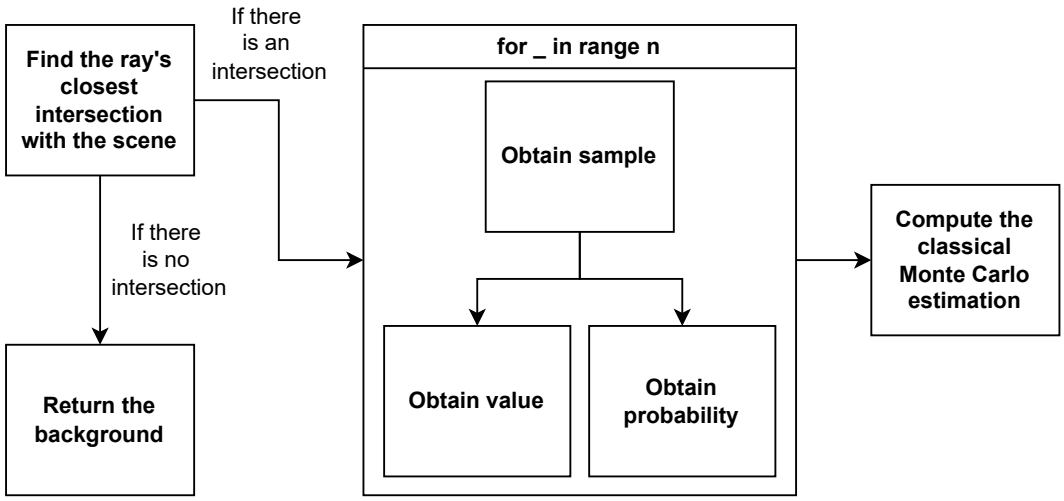


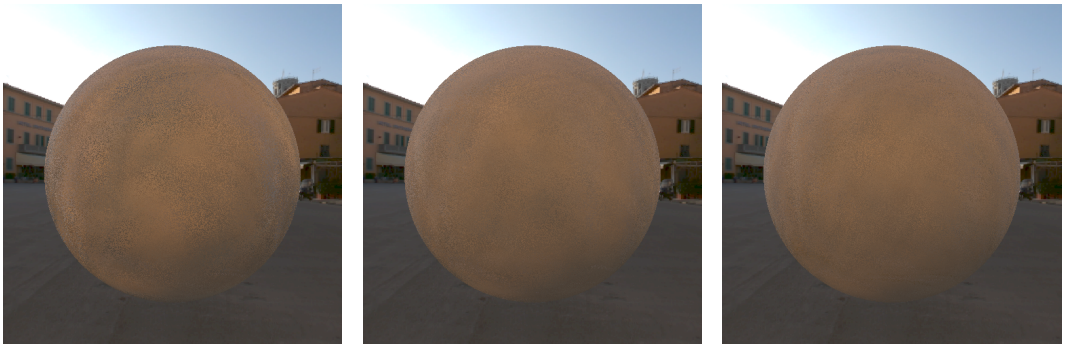
Figure 3.4: Classical Monte Carlo integrator flow diagram.

3.2.4 Bayesian Monte Carlo

Having tested the Monte Carlo integrator, the next step was the Bayesian Monte Carlo method. Before the integrator 3.6, I implemented a *GP* (Gaussian Process) class that, given a set of sample positions and values, returns a posterior estimate of $f(\omega)$ at a given hemispherical direction ω. Recalling the Bayesian Monte Carlo quadrature equations, we have $\hat{I} = \bar{I} + z^t Q^{-1}(Y - \bar{F})$ (2.12), where $\bar{I} = \int \bar{f}(\omega)p(\omega)d\omega$ and $z = \int k(\omega)p(\omega)d\omega$ (2.13). Assuming no prior information (i.e. $\bar{f}(\omega) = 0$), the BMC quadrature equation (2.12) can be rewritten as $\hat{I} = z^t Q^{-1} Y$.

Therefore, the *GP* class must compute Q^{-1} and z^t to return the estimate $\hat{I}$. I used a Sobolev covariance function with smoothness parameter $s = 1.4$ as found to be optimal by Marques [9]. I added a noise of 0.01 to the diagonal of Q for accommodating the observations to the assumptions of the GP model and avoiding numerical instability while inverting Q. For z^t, since in general we do not have an analytic expression to evaluate this vector of integrals, I computed it using the classical Monte Carlo method with 10000 samples and a uniform PDF.

This implementation gave better results with the same number of samples when compared to Monte Carlo both in the Workbench and in the rendering application in terms of RMSE. However, the execution time was far greater, which was especially noticeable in the renderer where the estimation has to be computed for each pixel. This is due to two very costly operations that the algorithm above has to compute for each estimation: matrix inversion and integration of the co-



(a) 20 spp (b) 40 spp (c) 80 spp

Figure 3.5: An example of images rendered with Bayesian Monte Carlo using 4 GPs and varying numbers of samples per pixel (spp). Note that, as the number of spp increases, the visible noise resulting from estimation error is progressively reduced. The RMSE of each image is 0.01623, 0.01161 and 0.01040, for (a), (b) and (c), respectively. More details regarding the computation of the RMSE are given in Sec. 3.2.7, and a precise error comparison with other methods can be found in the results section.

variance function over the hemisphere, that are of complexity $\mathcal{O}(n^3)$ and $\mathcal{O}(m)$ where n is the number of samples of the set D and m is the number of samples taken from the Sobolev covariance function, respectively.

Luckily, there is a technique known to greatly decrease runtime for rendering [10]. It consists of taking a fixed set of sample locations that are re-used for each pixel, and applying a rotation that aligns them to the new surface normal. If $\vec{n_0}$ is the *up* vector on the unit hemisphere where the original samples were generated, and $\vec{n}$ is the new surface normal, then

$$\begin{aligned} &\vec{a} = \vec{n_0} \times \vec{n} = \begin{bmatrix} a_0 & a_1 & a_2 \end{bmatrix} \text{ is the rotation axis,} \\ &\alpha = \arccos n_0 \cdot n \text{ is the rotation angle, and} \end{aligned} \tag{3.2}$$

$$\begin{bmatrix} c + a_0^2(1-c) & a_0a_1(1-c) - a_2s & a_0a_2(1-c) + a_1s \\ a_1a_0(1-c) + a_2s & c + a_1^2(1-c) & a_1a_2(1-c) - a_0s \\ a_2a_0(1-c) - a_1s & a_2a_1(1-c) + a_0s & c + a_2^2(1-c) \end{bmatrix}, \tag{3.3}$$

where $c = \cos(\alpha)$ and $s = \sin(\alpha)$, is the rotation matrix that aligns n_0 to n.

Since the covariance matrix is invariant to rotation [10], Q^{-1} and z^t do not need to be recomputed. This gives way to a very significant decrease in time complexity, making the integrator's runtime similar to that of the Monte Carlo integrator while reducing the error significantly.

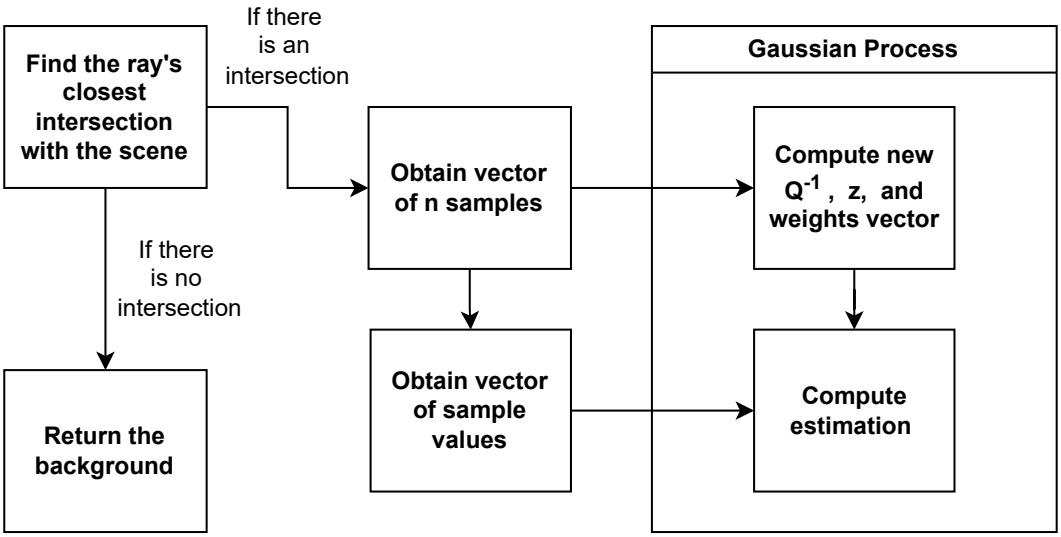


Figure 3.6: Bayesian Monte Carlo integrator flow diagram.

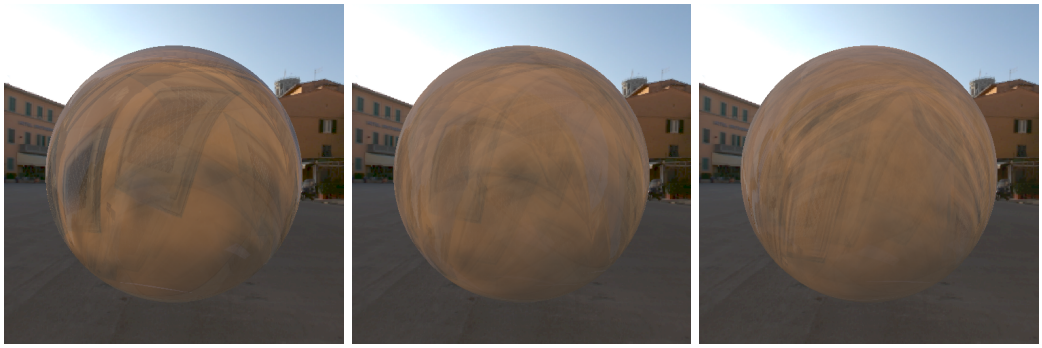
However, this technique has a small downside. The use of the same sample locations over and over introduces a coherence in the approximation error for neighboring pixels. This creates low frequency patterns or artefacts in the image that are visible to the human eye.

To decorrelate the sample directions across neighboring pixels, I have performed an additional random rotation on the azimuth. Considering a random number $\gamma \sim \mathcal{U}(0, 2\pi)$, the matrix

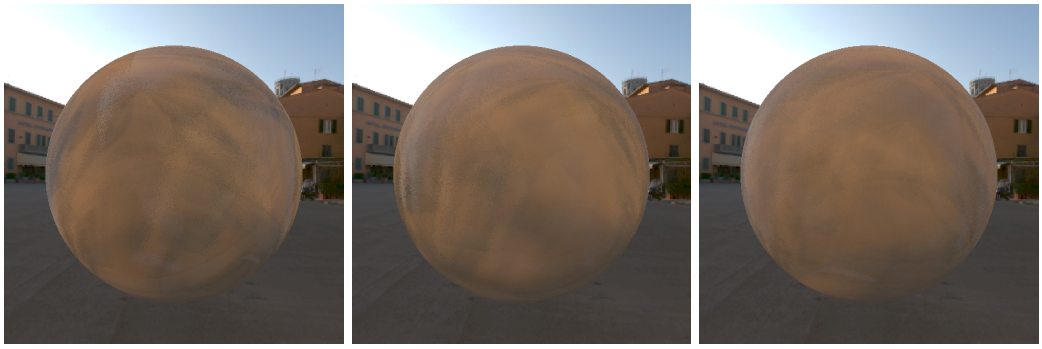
$$\begin{bmatrix} \cos\gamma & 0 & -\sin\gamma \\ 0 & 1 & 0 \\ \sin\gamma & 0 & -\cos\gamma \end{bmatrix}, \tag{3.4}$$

performs a random rotation on the azimuth (assuming y is the vertical axis).

As shown in 3.7, the random rotation of the sample set helps mitigate the problem of low frequency artefacts due to error coherence across neighboring pixels. However, in some cases, the pattern might still be visible despite the rotation, because the sample set is always the same. To cope with this problem, we can use more than a single sample set to render the final image. In practice, this can be achieved using a small number of Gaussian processes, each with its own sample set. When estimating the color of a given pixel, the GP to use is selected randomly 3.8. As it can be seen in 3.5, using 4 GPs with a random rotation on the azimuth blurs the error patterns very effectively.



(a) 1 GP without random rotation.



(b) 1 GP with random rotation.

Figure 3.7: An example of images rendered with Bayesian Monte Carlo with 20, 40 and 80 spp from left to right. The images on top have been generated with 1 GP without applying a random rotation on the azimuth. The images on the bottom have been generated with 1 GP and with an added random rotation. It can be observed that the artifacts visible on the top images are somewhat faded with the addition of a random rotation. Please also compare with Figure 3.5, where additional GPs were used to further decorrelate the noise.

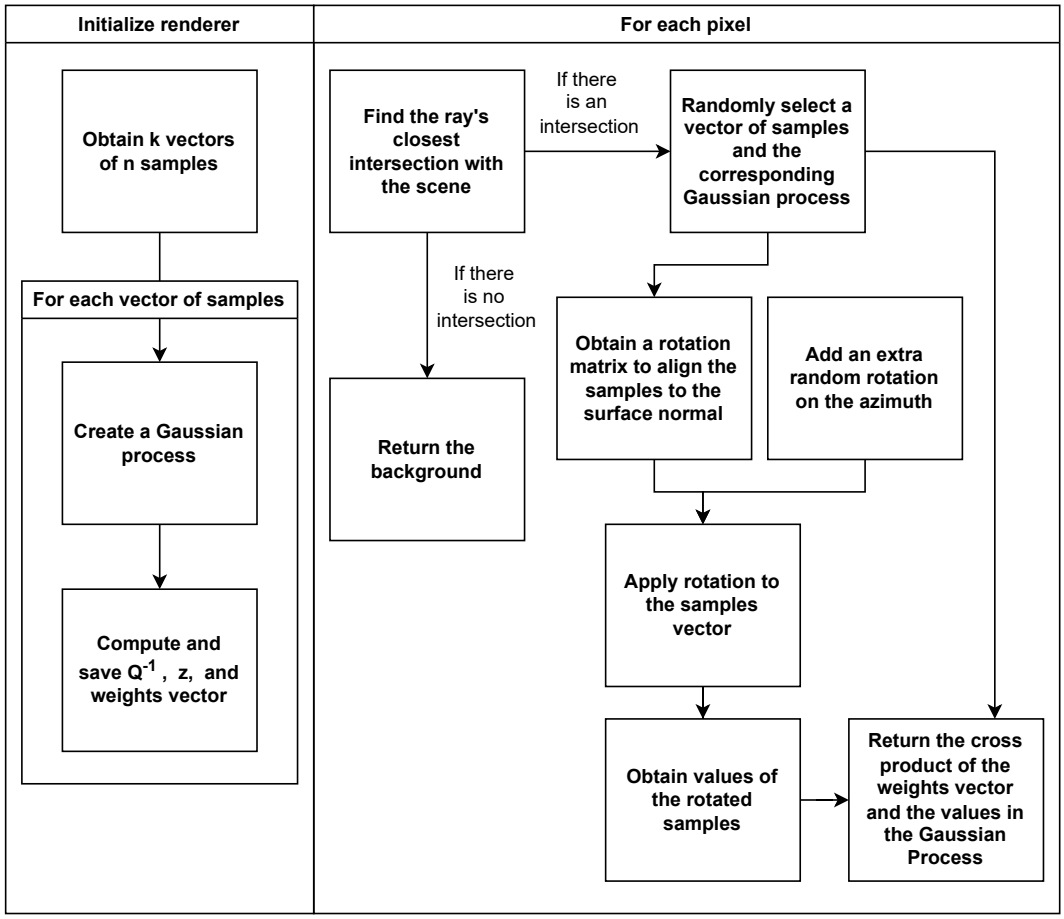
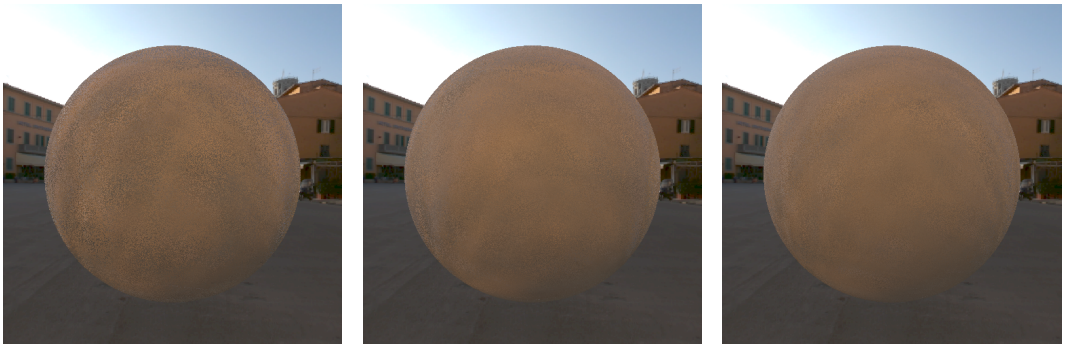


Figure 3.8: Improved Bayesian Monte Carlo integrator flow diagram.

3.2.5 Specifying a more informed prior mean function in the Bayesian Monte Carlo estimate



(a) 20 spp (b) 40 spp (c) 80 spp

Figure 3.9: An example of images rendered with BMCP using 4 GPs and varying numbers of samples per pixel (spp). Note that, as the number of spp increases, the visible noise resulting from estimation error is progressively reduced. The RMSE of each image is 0.01747, 0.01096 and 0.01022, for (a), (b) and (c), respectively. More details regarding the computation of the RMSE are given in Sec. 3.2.7, and a precise error comparison with other methods can be found in the results section.

In addition to implementing Bayesian Monte Carlo for rendering, we tried to test the following hypothesis: *The RMSE of a rendered image using the same number of samples can be improved by leveraging the estimates for adjacent pixels in the form of a constant prior*.

Recall the BMC quadrature equation 2.12, $\hat{I} = \bar{I} + z^t Q^{-1}(Y - \bar{F})$, where $\bar{I}$ is the integral of the prior of the Gaussian Process $\bar{f}(x)$, and $\bar{F}$ is the vector of samples of $\bar{f}(x)$ evaluated at the same locations as Y. If we assume the prior to be constant, $\bar{f} = c$, then we have that $Y = [c, \ldots, c]$ is a constant vector and $\bar{I}$ can be easily evaluated over the hemisphere.

Our main concern with using a constant prior was whether it would prove to be suitable for approximating integrals of non-constant functions, since neither the incident radiance nor the BDRF will be constant in most situations.

For this reason, we first designed a smaller-scale experiment in the Workbench. We wanted to see if the error of the estimation could be improved for non-constant functions the integrals of which we know analytically and for which we can find a constant that has the same definite integral over the hemisphere.

Recall that the function we wish to integrate has the form $f(\omega) = L_i(\omega)\rho(\omega)\cos\omega$. To simplify the experiment, we assigned $\rho(\omega) = 1$ and we chose the functions $L_i(\omega) = \cos\omega$ and $L_i(\omega) = \cos^2\omega$. These integrals can be computed analytically

over the hemisphere:

$$\begin{aligned} I_1 &= \int_\Omega \cos^2\omega d\omega = \int_0^{2\pi} d\Phi \int_0^{\frac{\pi}{2}} \cos^2\theta \cdot \sin\theta d\theta = \frac{2\pi}{3}, \\ I_2 &= \int_\Omega \cos\omega^2 d\omega = \int_0^{2\pi} d\phi \int_0^{\frac{\pi}{2}} \cos^3\theta \cdot \sin\theta d\theta = \frac{2\pi}{4}. \end{aligned} \tag{3.5}$$

In addition, we have that the integral of a constant c over the hemisphere can be computed as

$$I = \int_\Omega c d\omega = c \int_0^{2\pi} d\Phi \int_0^{\frac{\pi}{2}} d\theta = 2\pi c. \tag{3.6}$$

This way we can impose $I_1 = \int_\Omega c_1 d\omega$ and $I_2 = \int_\Omega c_2 d\omega$ and obtain $c_1 = \frac{1}{3}$ and $c_2 = \frac{1}{2}$ such that their integrals over the hemisphere correspond to those of the functions $\cos\omega$ and $\cos^2\omega$, respectively.

By assigning c_1 and c_2 as priors for I_1 and I_2, both times we obtained an improvement in the workbench, as shown in the results section.

Having obtained promising results in the Workbench, I implemented the addition of a constant prior in rendering. This time I had to add a pre-rendering step, where the values of every second pixel using Bayesian Monte Carlo are computed without a prior 3.10. In the rendering step, the remaining pixels are computed using a combination of the colors of the adjacent pre-rendered pixels as prior. Unfortunately, as is shown in the results section, the error measured with the rendered images did not display a significant improvement over not using any prior in Bayesian Monte Carlo 4.2. The possible reasons for this and ideas for improvement are detailed in the *further work* section.

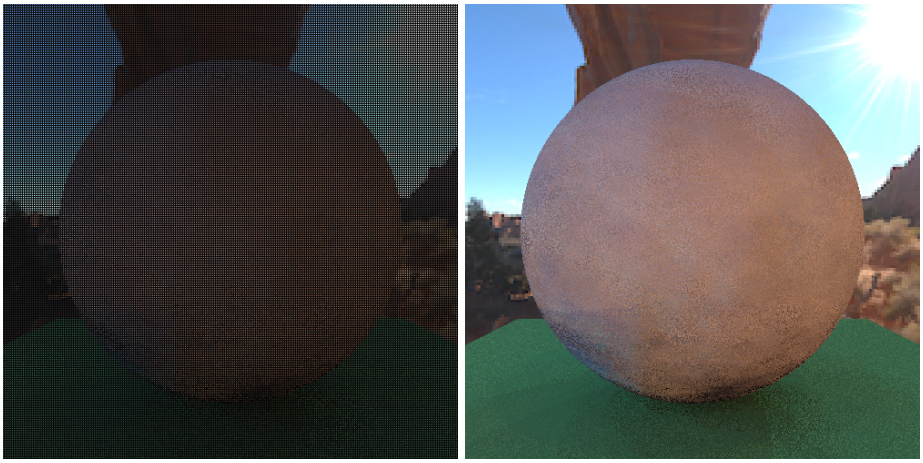


Figure 3.10: Left: Pre-render of every second pixel with Bayesian Monte Carlo using no prior information. Right: After rendering the remaining pixels using a blend of the color of adjacent pixels.

3.2.6 Experiments performed with the renderer to validate the integrators

After implementing each method, I performed a series of tests in tightly controlled conditions so as to validate the correction of the different implementations of the CMC, BMC and BMCP estimators. Note that the goal of these experiments is not performing a comparison between the methods (provided later in 4), but instead to ensure their convergence towards the correct result.

Test 1: $f(\omega) = \cos\omega$

Consider the illumination integral

$$L_r(\omega_o) = \int_\Omega L_i(\omega_i) \cdot K_d \cdot \cos\theta_i \, d\omega_i. \tag{3.7}$$

Taking $K_d = 1$ and $L_i = 1$ constants, we can now evaluate the illumination integral 3.7 over the hemisphere and obtain a constant value:

$$L_r(W_o) = \int_\Omega \cos\theta_i \, d\omega_i = \int_0^{2\pi} d\phi \int_0^{\frac{\pi}{2}} \cos\theta \cdot \sin\theta d\theta = \pi. \tag{3.8}$$

By setting the $K_d = 1$ and $L_i = 1$ manually before rendering, we can make sure that the illumination integral we are approximating is going to be 3.8 in every pixel in which there is a hit with an object. Knowing this, I generated a very small

test scene 3.11 in order to check that the values in pixels where there is a hit are indeed approximating π.

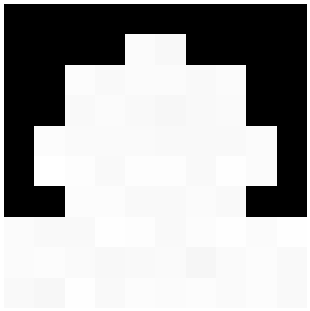


Figure 3.11: Test scene of a sphere on a plane of size 10x10.

Test 2: $f(\omega) = \cos^2 \omega$**.**

Another value pair we considered to validate the methods was $K_d = 1$ and $L_i = \cos\theta_i$ where the incident radiance varies over the hemisphere. In this case the integral evaluates to

$$L_r(W_o) = \int_\Omega \cos^2\theta_i \, d\omega_i = \int_0^{2\pi} d\phi \int_0^{\frac{\pi}{2}} \cos^2\theta \cdot \sin\theta d\theta = \frac{2\pi}{3}. \tag{3.9}$$

Like in the previous test, I tested that all values are around $\frac{2\pi}{3}$ in a simple scene for all the implemented integrators.

Test 3: Rotation Test

I used this test to validate that I was taking into account the changing orientation of the surface normal in each impact with the scene. For this purpose, I rendered the default scene with the environment map using only one sample: the surface normal. If we are obtaining the incident radiance with the correctly rotated surface normal, this test has to result in a sort of distorted mirror of the environment map where its elements can be well discerned 3.12. If we tried obtaining the incident radiance without rotating the normal, we would not have a clearly recognizable reflection of the environment map on the surface of the objects.

The tests described in this section were performed in all three methods, and they yielded the correct results.

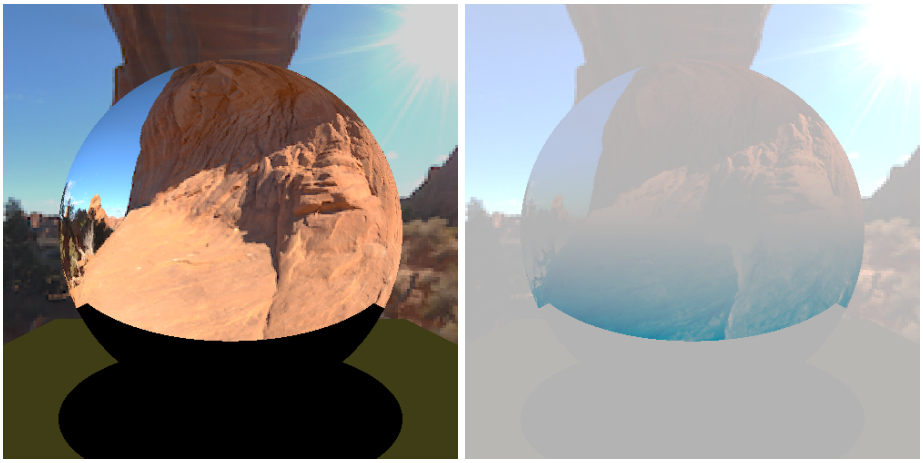


Figure 3.12: Rotation tests. Right: expected result. Left: result without normal rotation

3.2.7 Implementation of error measure and image comparison

To obtain a numerical measure of the error of an image, we used the *root-mean-square error* (RMSE) of the image and a previously computed reference image with a large number of samples. In our case, we used images generated with classical Monte Carlo using 3000 samples 3.13 as reference for all error estimates.

root-mean-square error method to compute the difference between a result image we have obtained and a reference image:

Algorithm 1 Mean squared error method to compute the difference between a result image and a reference image.

$sum \leftarrow 0$
for $i \leftarrow 1$ to n **do**
  for $j \leftarrow 1$ to m **do**
    $sum \leftarrow sum + norm(reference[i,j] - result[i,j])$
  end for
end for
return $sum/(n*m)$

In order to reduce the chance factor, I programmed a short script to compute the root-mean-square error and average run time of n renderings of the same scene. I used the average mean squared error and run time of 20 renderings of the same scene for all results presented in this thesis.

In addition to a numerical error measure, we found it helpful to have a graphic

Figure 3.13: Reference images computed with classical Monte Carlo using 3000 samples per pixel. I used the environment map on the left (*Arch*) during most of the development stage, and then later switched to the environment map on the right (*Pisa*) for the final experiments, as it presents less variation in color, which allows for a better evaluation of the methods.

representation of the error. I computed the "error images" by finding the difference, pixel by pixel, of the norm of all three color channels. In this script, I included the options to either normalise the difference or not, as normalising yields better images, but when comparing two "error images" it is more useful to have absolute errors. I also included the options to either represent the errors in grayscale or using a colormap (from matplotlib) for more aesthetic / visually pleasing results, with the latter having a longer run time 3.14.

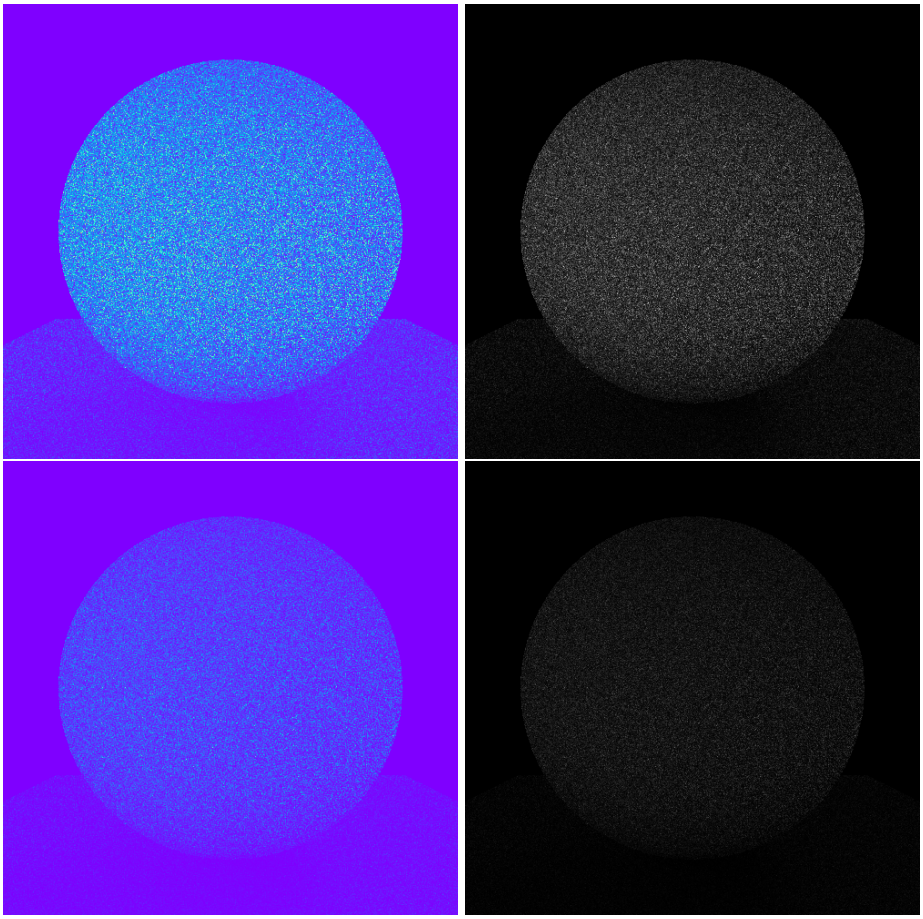


Figure 3.14: Error of Monte Carlo with 30 samples compared to a reference image of Monte Carlo with 3000 samples. Left: Using a colormap (rainbow). Right: No colormap. Top: Normalized. Bottom: Absolute error.

Chapter 4

Results and Discussion

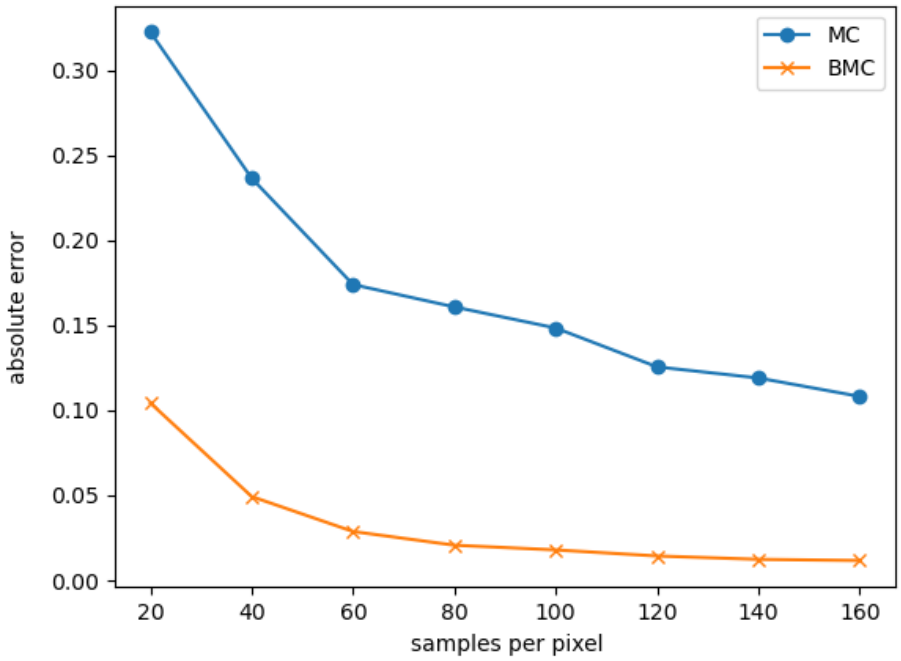
In this section I present the results for the Monte Carlo (MC) and the Bayesian Monte Carlo (BMC) methods. The figures and tables related to rendering show the mean errors obtained from various methods across multiple identical experiments. To account for computational constraints and the reduction of error variability with increased sample size, the number of experiments varied based on the sample size. Specifically, for sample sizes of 20, 40, 60, and 80, the experiments were repeated 80, 40, 26, and 20 times respectively. Additionally, the figures related to workbench present the mean errors obtained by averaging the error of all estimations over 300 runs for each sample size.

4.1 Performance of Monte Carlo vs Bayesian Monte Carlo

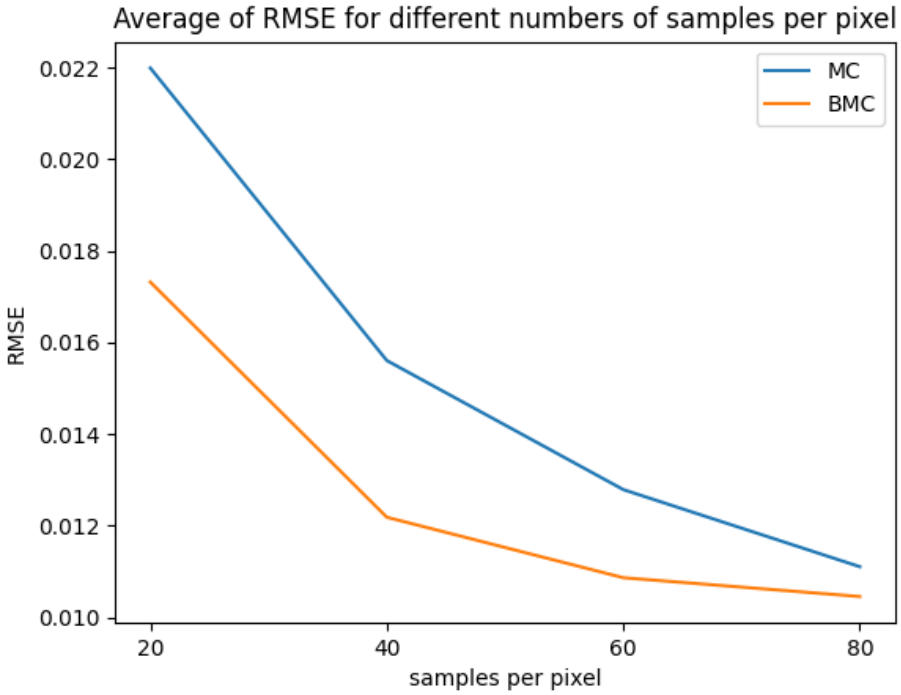
In my experiments, in line with previous research, I found that BMC converges faster than MC. This is because BMC includes uses the covariance of sample locations for the estimation, while MC discards this information.

I evaluated the integral of $\cos^3(x)$ over the unit hemisphere in the workbench, the known value of which is $\frac{2\pi}{4}$. Figure 4.1 (a) shows the error of both methods over different numbers of samples, calculated over 300 trials with different samples used each time.

In image rendering, the results were similar. I generated images using 20, 40, 60, and 80 samples per pixel (spp) with both integrators and used the Root Mean Squared Error (RMSE) to compare them to a reference image. Note that to reach the same RMSE value as BMC with 40 samples, MC requires almost 80 samples. This is a significant difference in rendering, where obtaining sample values is costly. The results can also be visually seen by comparing the two images generated with the same number of samples per pixel using both methods, as shown in Figure 4.1 (b).



(a) Workbench: Error of estimation of the integral of $\cos^3(x)$ over the unit hemisphere.



(b) Rendering: RMSE of rendered images using different numbers of spp.

Figure 4.1: MC vs BMC for integral estimation and image rendering.

4.2 Improving Bayesian Monte Carlo for Rendering: Run Time and Error Analysis

The techniques described in Subsection 3.2.4 greatly reduced execution time of Bayesian Monte Carlo. As an example, the average time to render an image of 500x500 pixels using 40 samples per pixel (spp) with the improvements was in the range of 440-590 seconds, while without them, the same image did not finish rendering in over 24 hours. This is because the naive implementation computes the Bayesian Monte Carlo quadrature equations 2.2.2 for every pixel. By reusing sample-location sets, the number of times these computations are executed is reduced to the number of Gaussian Processes chosen (4-5 GPs are typically sufficient to decorrelate the noise).

The additional measures taken to decorrelate the error, also described in 3.2.4, did not alter the root mean square error (RMSE), as can be seen in Table 4.1. These results demonstrate that the value of these techniques lies in redistributing the noise in a way that makes it appear more random, which makes it less noticeable to the human eye, as shown in Figure 3.7.

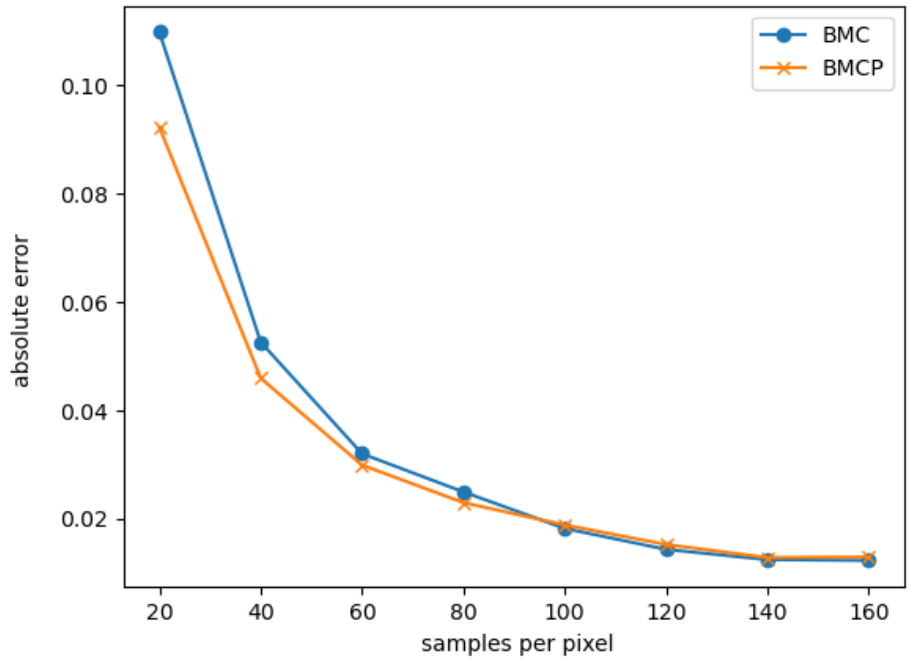
Method	20 spp	40 spp	60 spp
BMC 1 GP (no rotation)	0.01741	0.01287	0.01179
BMC 1 GP	0.01751	0.01239	0.01089
BMC 4 GP	0.01732	0.01218	0.01086

Table 4.1: Average RMSE of images rendered with 20, 40, and 60 spp. The runtime of the naive Bayesian Monte Carlo was too high to be included in this experiment.

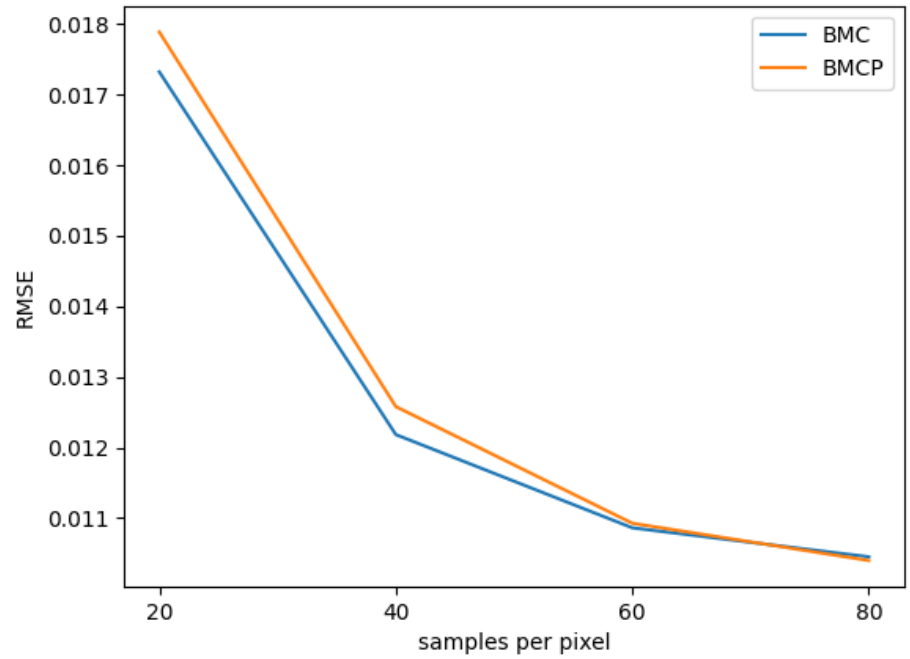
4.3 Use of a Constant Prior in Bayesian Monte Carlo

The use of a constant prior in the workbench resulted in a small improvement when approximating the integral of the function $cos^2(x)$ over the unit hemisphere, as compared to not using any prior 4.2 (a). However, in image rendering, the use of a constant prior taken from a combination of adjacent pixel colors slightly delayed convergence to the correct result, as shown in Figure 4.2 (b).

Our choice of a constant prior relies on the assumption that adjacent pixels will have similar colors. However, this is usually not true for pixels on the edges or borders of objects. In these cases, using the average of adjacent pixels as a prior can lead to worse estimates than using no prior at all. Figure 4.3 illustrates that the error is larger on the borders when using BMCP compared to BMC.



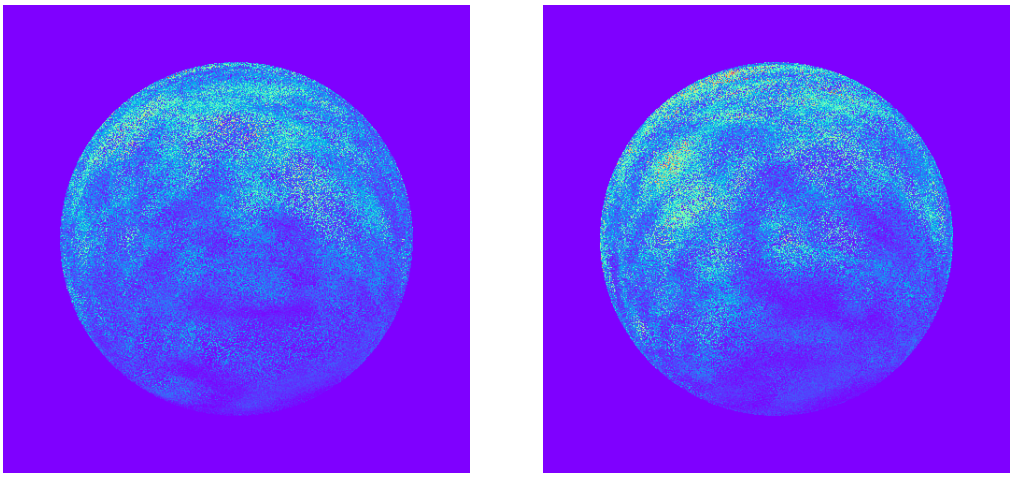
(a) Workbench: Error of estimation of the integral of $\cos^2(x)$ over the unit hemisphere.



(b) Rendering: RMSE of rendered images using different numbers of spp.

Figure 4.2: BMC vs BMCP for integral estimation and image rendering.

4.4 Discussion of Results and Future Work



(a) (b)

Figure 4.3: Error, pixel by pixel, rendered with 40 spp. Subfigure (a) was rendered with BMC and (b) with BMCP. Note that in (b) the error is visibly more concentrated on the borders of the object.

The results of using Bayesian Monte Carlo for image rendering were positive and consistent with existing research on the topic. Regarding the incorporation of prior knowledge, it was demonstrated in the workbench that the use of an appropriate constant prior can improve estimations of non-constant functions. However, it is still uncertain if a constant prior can be applied to improve image rendering as well. The simple prior used in this study was not sufficient to improve the RMSE in rendering. An important part of the error when using a constant prior seems to be concentrated on the edges of the objects, which could be addressed in future work by assigning weights to the adjacent pixels based on their likelihood of having similar colors to the pixel we are computing. For instance, colors are more likely to be similar when the surface normals are similar and the points are closer together in the virtual scene.

Chapter 5

Conclusion

In this thesis, I successfully implemented the Bayesian Monte Carlo (BMC) method for image rendering and compared it to the classical Monte Carlo (MC) method. I also attempted to improve the efficiency of performing image synthesis with the BMC method by incorporating prior information. Using a Python-based script, I estimated the value of different integrals over the unit hemisphere to validate and compare the methods before applying them to image rendering. The performance of the methods for rendering was evaluated using the root-mean-squared error with respect to a a reference image generated with a large number of samples using Monte Carlo.

My results were consistent with previous research, in that I found that the BMC method outperformed MC using the same information (samples). However, another important advantage of BMC over MC which I could not fully exploit in this thesis is the possibility of easily incorporating prior information in the integral estimate. My experimentation with adding a constant mean prior function enhanced the BMC estimate in the prototyping phase. This shows that prior information can affect the BMC estimate in a relevant way, and thus opens doors for future research in this direction. However, more work has to be done to export this solution to rendering, where my implementation still needs some tuning. My results in rendering with a constant prior show that, if inaccurate, prior information can lead to worse predictions than no prior at all. Although my choice for the prior in image rendering in this thesis was not ideal, I have some ideas to improve it for future work involving taking into account the likelihood of an adjacent pixel having a similar color to the pixel being computed.

Bibliography

[1] J. Arvo and D. Kirk. "Particle transport and image synthesis". In: *Proceedings of the 17th annual conference on Computer graphics and interactive techniques*. 1990, pp. 63–66.

[2] R. L. Cook. "Stochastic sampling in computer graphics". In: *ACM Transactions on Graphics (TOG)* 5.1 (1986), pp. 51–72.

[3] P. Dutré, K. Bala, and P. Bekaert. *Advanced Global Illumination*. 2nd ed. Wellesley, Massachusetts: A K Peters, Ltd., 2006.

[4] Z. Ghahramani and C. Rasmussen. "Bayesian Monte Carlo". In: *Advances in Neural Information Processing Systems*. Ed. by S. Becker, S. Thrun, and K. Obermayer. Vol. 15. MIT Press, 2002.

[5] Z. Ghahramani and C. Rasmussen. "Bayesian monte carlo". In: *Advances in neural information processing systems* 15 (2002).

[6] Jay-Artist. *The White Room Cycles*. [Online; accessed january 19, 2023]. 2012. URL: `https://blendswap.com/blend/5014`.

[7] J. T. Kajiya. "The rendering equation". In: *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*. 1986, pp. 143–150.

[8] R. Marques, C. Bouville, and K. Bouatouch. "Gaussian Process for Radiance Functions on the Sphere". In: *Computer Graphics Forum*. Vol. 41. 6. Wiley Online Library. 2022, pp. 67–81.

[9] R. Marques, C. Bouville, and K. Bouatouch. "Optimal sample weights for hemispherical integral quadratures". In: *Computer Graphics Forum*. Vol. 38. 1. Wiley Online Library. 2019, pp. 59–72.

[10] R. Marques, C. Bouville, M. Ribardiere, L. P. Santos, and K. Bouatouch. "A spherical Gaussian framework for Bayesian Monte Carlo rendering of glossy surfaces". In: *IEEE Transactions on Visualization and Computer Graphics* 19.10 (2013), pp. 1619–1632.

[11] R. Marques, C. Bouville, L. P. Santos, and K. Bouatouch. “Efficient quadrature rules for illumination integrals: From quasi Monte Carlo to Bayesian Monte Carlo”. In: *Synthesis Lectures on Computer Graphics and Animation* 7.2 (2015), pp. 1–92.

[12] A. O’Hagan. “Monte Carlo is fundamentally unsound”. In: *The Statistician* (1987), pp. 247–249.

[13] C. K. Williams and C. E. Rasmussen. *Gaussian processes for machine learning*. Vol. 2. 3. MIT press Cambridge, MA, 2006.

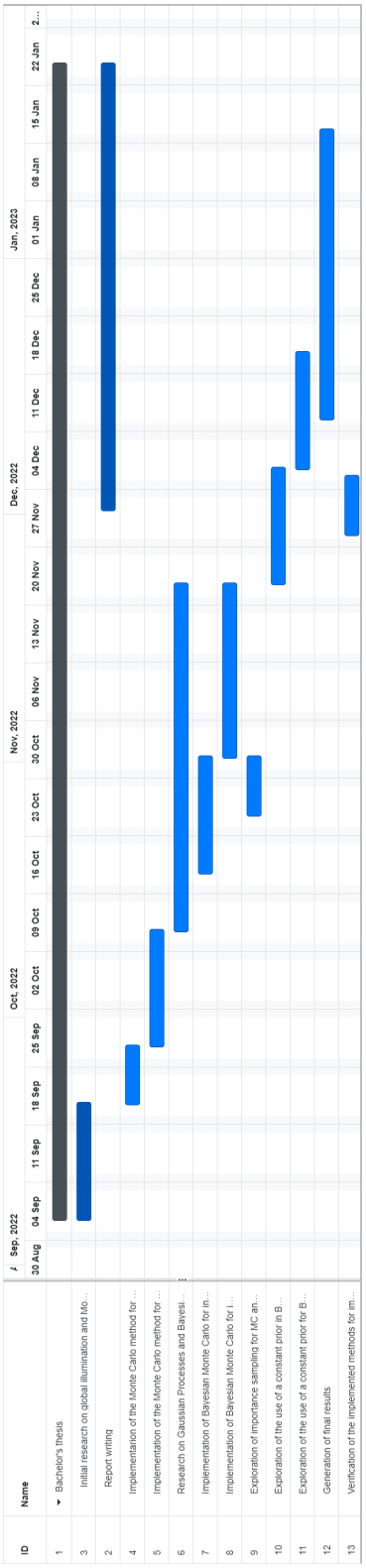


Figure 1: Gantt Chart of the work developed during the thesis.