REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

Ministère de l'enseignement supérieur et de la recherche scientifique

Université Mohamed Khider Biskra

Faculté des Sciences Exactes et des Sciences de la Nature et de la Vie

Département d'informatique

N° d'ordre :………………………

Série : …………………………….

Thèse

Présentée en vue de l'obtention du diplôme de

docteur 3ème cycle en Informatique

Option : Techniques d'image et d'intelligence artificielle

Toward an Efficient Approach for Selecting VPLs in Global Illumination

Par :

Djihane BABAHENINI

Soutenue le : 05 / 10/2017

Devant le jury :

Djedi Noureddine	Professeur	Université de Biskra	Président
Babahenini Mohamed Chaouki	Professeur	Université de Biskra	Rapporteur
Bouatouch Kadi	Professeur	Université de Rennes 1	Co-rapporteur
Ait Aoudia Samy	Professeur	ESI. Oued Smar Alger	Examinateur
Cherif Foudil	Professeur	Université de Biskra	Examinateur

To my Mother...

To the memory of my father...

To Mohamed...

Acknowledgements

First, I want to thank almighty ALLAH for giving me the will, patience and health to develop this work although the circumstances surrounding us I would like to express my profound gratitude to my supervisors Prof. Babahenini Mohamed Chaouki and Prof. Kadi Bouatouch, for their involvement in this research work and for the support they have given me, their patience, their availability and the relevance of their advice that have been of invaluable assistance throughout this work. Also, I would like to thank Adrien Gruson for his help. I would especially like to thank Kadi for having dedicated me of the time throughout those years, as well as for his advice and encouragement, for his distinguished reception in the IRISA laboratory and his supervision during my stay in France.
I would like to extend my warmest thanks to Adrien Gruson for his kindness and patience, without forgetting to thank the members of the IRISA laboratory and especially FRVSense group for their support during my research in their laboratory.

Special thank the director of the LESIA laboratory, Prof. Cherif Foudil for his support during all my years of study.

I would like to thank Prof. Ait ouadia Sami, a professor at the ESI. Oued Smar. Algiers, Prof. Djedi Nouredinne, Prof. Cherif Foudil for having accepted to judge this work.

Finally, I thank my mother, my husband Mohamed and my family for encouraging me during these many years, and to have always been available when I needed it.

Résumé

En informatique graphique, le rendu nécessite le calcul de la contribution des sources lumineuses (éclairage direct) et de tous les objets de la scène à travers des réflexions, des réfractions et des diffusions multiples dans des milieux participants tels que la fumée, la poussière, les nuages (éclairage indirect).
L'éclairage indirect est une tâche très coûteuse, principalement à cause de calcul de la visibilité (entre les rayons lumineux et les objets de la scène). Pour accélérer l'éclairage indirect, il est nécessaire d'exploiter les performances élevées du GPU (Graphics Processing Unit).

Le but de cette thèse est d'utiliser le GPU pour calculer tous les objets visibles à partir des sources lumineuses. À cette fin, nous mettons une caméra à 360 degrés composée d'un DPRSM (Dual Paraboloid Reflective Shadow Map). Pour chaque pixel de cette caméra, nous calculons le point visible, sa position 3D, sa normale et sa couleur. Chaque point visible agit comme une source de lumière ponctuelle qui joue le rôle de source de lumière secondaire, appelée VPL (Virtual Point Light). Pendant le rendu, tout point de la scène reçoit un rayon en raison de l'éclairage direct et de l'éclairage indirect. Ce dernier est la contribution des VPLs.
Le calcul des contributions de tous les VPL prend beaucoup de temps de calcul. Une meilleure solution est de choisir un petit sous-ensemble de VPL par importance à l'aide d'une méthode de transformée inverse (appelée IT) basée sur le calcul d'une CDF (Cumulative Distribution Function). Ensuite, nous calculons la contribution de ce petit sous-ensemble de VPL aux points visibles de la caméra à travers les pixels. La façon dont la CDF est calculée est cruciale pour la qualité de l'image de rendu. Nous proposons deux méthodes pour calculer une CDF efficace ainsi qu'une méthode MIS (Multiple Importance Sampling) combinant une méthode de transformée inverse avec une approche de tracé de chemin distribué (gathering).

mots clés: Visibilité, éclairage indirect, Voxelisation, Méthodes de transformée inverse, VPL.

ملخص

في مجال الرسومات الحاسوبية، تتطلب إضاءة مشهد ثلاثي الأبعاد حساب كمية الضوء المنبعثة من المصادر الضوئية المختلفة وهذا النوع من الإضاءة يطلق عليها مصطلح الإضاءة المباشرة. لكن لجعل المشهد ثلاثي الأبعاد يبدو وكأنه حقيقي فإنه لا يكفي حساب الإضاءة المنبعثة من المصادر الضوئية المباشرة فحسب، ويرجع ذلك إلى أن الأجسام المكونة للمشهد الثلاثي الأبعاد تعكس كمية من الضوء الذي يساهم بدوره في إضاءة أجسام أخرى. إذا فإن حساب انعكاس، انكسار وانتشار الضوء في مشهد ثلاثي الأبعاد يسمح بتجسيد ومحاكاة مجموعة من الظواهر الأكثر تعقيدا ومثال ذلك: الدخان، الغبار والسحاب... الخ. ويسمى هذا النوع من الإضاءة بالإضاءة غير المباشرة. إن حساب الإضاءة غير المباشرة يستغرق مدة طويلة من الزمن تصل إلى ساعات أو حتى أيام، حيث أن معظم هذا الوقت المستغرق يتم استهلاكه في تحديد الرؤية بين نقطتين من كل جسم داخل المشهد الثلاثي الأبعاد. لتسريع عملية حساب الإضاءة غير المباشرة فإننا نقوم بالبرمجة باستخدام وحدة معالجة الرسومات الموجودة داخل الحاسوب. الهدف من هذه الأطروحة هو حساب الإضاءة غير المباشرة باستخدام وحدة معالجة الرسومات لجميع الأجسام المرئية. ومن أجل تحقيق هذه الغاية فإننا قمنا بوضع كاميرا تبصر الأشياء عبر ٣٦٠ درجة. ولكل بيكسل من هذه الكاميرا نحسب نقطة ضوئية افتراضية تكون معرفة بموقعها في المشهد، النظامي، اللون. كل نقطة ضوئية تلعب دور مصدر ضوء ثانوي. يستغرق حساب إضاءة جميع النقط الضوئية الافتراضية وقتا طويلا جدا. ويتمثل أحد أفضل الحلول في اختيار مجموعة فرعية صغيرة منهم بطريقة عشوائية استنادا إلى دالة التوزيع التراكمي. ثم نحسب مساهمة هذه المجموعة الفرعية الصغيرة من النقط الضوئية الافتراضية إلى النقاط المرئية من الكاميرا. تظهر النتائج نوعية عالية للصورة المقدمة. حيث نقترح طريقتين لحساب كفاءة النقط الضوئية الافتراضية.

الكلمات المفتاحية: الرؤية، الإضاءة غير المباشرة،التكعيب، أساليب التحويل العكسي، نقطة ضوئية افتراضية.

Abstract

In computer graphics, rendering requires computing the contribution of the light sources (direct lighting) and that of all the objects within the scene through multiple reflections, refractions and scattering within participating media such as smoke, dust, clouds (indirect lighting).
Indirect lighting is a very time-consuming task, mainly due to visibility computation (between light rays and the scene's objects). To speed up indirect lighting, one way is to exploit the high performance of the GPU (Graphics Processing Unit).

The aim of this thesis is to use the GPU for computing all the objects visible from the light sources. To this end, we place a 360-degree camera consisting of a DPRSM (Dual Paraboloid Reflective Shadow Map). For each pixel of this camera, we compute the visible point, its 3D position, normal and color. Each visible point acts as a point light source which plays the role of secondary light source, called VPL (Virtual Point Light). During rendering, any point in the scene is assigned a radiance due to direct lighting and indirect lighting. This latter is the contributions of the VPLs.
Computing the contributions of all the VPLs is very time-consuming. One better solution is to choose by importance a small subset of VPLs using an inverse transform method (called IT method) based on CDF (Cumulative Distribution Function). Then we compute the contribution of this small subset of VPLs to the points visible from the camera through the pixels. The way the CDF is computed is crucial for the quality of the rendered image. We propose two methods for computing an efficient CDF as well as an MIS (Multiple Importance Sampling) method combining an IT method with a gathering approach.

keywords: Visibility, Indirect lighting, Voxelization, Inverse Transform methods, Virtual Point Light.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Computer graphics domain is concerned with the creation and management of different interactions between the light and the objects of a 3D scene. Despite its appearance in the early 1950s, its applications are very limited. Currently, computer graphics invades different fields of research and it has an important goal in research. The realistic rendering of 3D scenes, which is nowadays strongly sought by the industries of cinema, architecture, multimedia, video games, etc.

The global illumination techniques in recent years are known a considerable improvement, both visually and in terms of computation time. A final realization of images photo-realistic with the techniques of global illumination, the exact calculation of the distribution of light in the 3D scene is the most important element to be taken into account. Indeed, the equation of rendering that has been introduced in 1886 makes it possible to model the distribution of the light in a scene. Or, an exact solution of this equation is impossible since there is an infinity of incident lights. Lighting simulation and the search for a solution to the problem of global illumination still is a very active research in computer graphics. Global illumination aims to simulate different lighting effects in a $3D$ scene. Several approaches exist, which are based on tracing rays or photons, such as: bidirectional path tracing[LW93], gathering, Metropolis-Hasting ([MRR$^+$53]; [Has70]), photon tracing[Jen96], and methods based on Virtual Point Light (VPL)([Kel97];[RGK$^+$08]; [REH$^+$11]). Many methods have been proposed in the literature to compute indirect lighting, some of them make use of VPLs. The use of VPLs can be an efficient way to compute global illumination. The VPLs are computed as follows. One classical camera is

placed at each point light source. Rendering from this camera allows computing a GBuffer containing for each pixel of this camera: the 3D position of the point visible to the point light source through the pixel, its normal, flux and color. This GBuffer is called RSM (Reflective Shadow Map). Note that a classical Shadow Map contains only the z coordinate (depth) of the visible point through a pixel, and an RSM is an extension of a Shadow Map. Each visible point stored in an RSM is called VPL. Each VPL acts as a secondary light source that could contribute to any point in the scene (indirect illumination).
Previous studies have shown that the critical step in global illumination computation is to determine visibility. This latter can be computed using algorithms based on Shadow Maps ([RGK+08]; [REH+11]). The authors of these latter papers have reported a good approximation of visibility when using VPLs, but it is very expensive (in terms of memory storage) to associate a Shadow Map with each VPL. Once the VPLs have been computed, for efficiency purpose, the radiance of a pixel of the scene camera is computed by summing the contributions (to the point visible through the pixel of the scene camera) of a small set of VPLs (rather than all the VPLs) selected randomly using an inverse transform method (IT) requiring the computation of a cumulative distribution function (CDF). The way the CDF is computed is crucial for the quality of the rendered image.

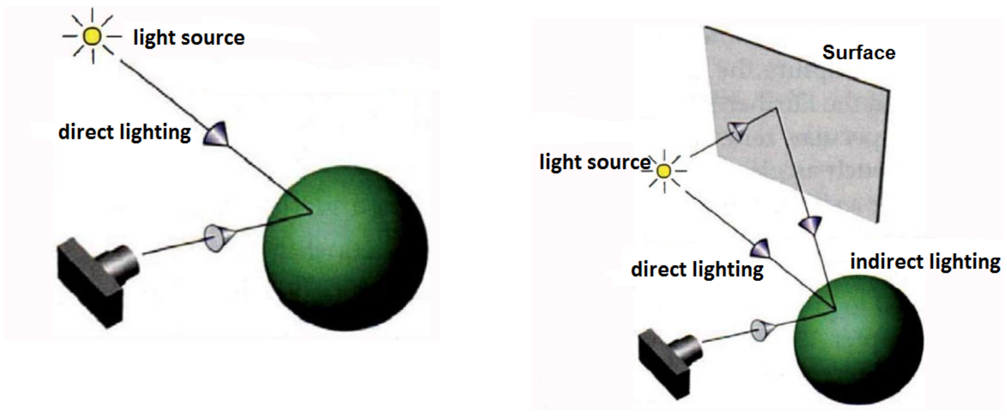


FIGURE 1.1: Direct vs Indirect illumination.

Our proposed methods can be used in several application domains, such as Virtual Reality and Augmented Reality, to improve realism. We could also apply our methods in Multimedia and Video games applications to minimize the rendering time.
In this manuscript, we are interested in global illumination methods based on

VPLs. The VPLs are constructed by placing two PRSM (Paraboloid Reflective Shadow Map), each having a field of view of 180 degrees around a point light source. This way a visibility of 360 degrees is assigned to a point light source. Note that a classical RSM has a field of view of only 90 degrees. From now on, the set of two PRSMs is called DPRSM (Dual Paraboloid Reflective Shadow Map).
The goal of this thesis is to propose a technique that uses the inverse transform method to select the more contributive VPLs in the scene, to do this we have to go through the implementation of the techniques of view adaptive imperfect shadow maps (global CDF) that is presented by [REH+11], and distributed ray tracing (gathering approach).

1.1 Motivation

The motivations of this PhD thesis is to provide new rendering techniques for selecting the VPLs that contribute more to the final image. We have focused on two main research axis:

- implementation of the methods for efficiently computing a CDF (used to select randomly a small set of VPLs contributing to the radiance of a point visible from the viewpoint) as well as an MIS (Multiple Importance Sampling) method combining an IT method (Inverse Transform) with a gathering approach aiming at improving the quality of the rendered image.
- visibility is computed using a voxel-based approach. We consider only single bounce indirect lighting and diffuse objects.

1.2 Contributions

Our main contributions are:

- implementation details on DPRSM and PRSM;

- two PRSMs (called DPRSM) are placed around each point light source to compute VPLs, and one PRSM is assigned to each VPL for visibility purpose;
- we show that DPRSM outperforms classical RSM when rendering with VPLs. To cover a $360 degree$ field of view a DPRSM requires two rendering passes while a classical RSM requires six.
- use of Dual Paraboloid Reflective Shadow Maps (DPRSM): when randomly selecting a small set of VPLs, each of the two paraboloid reflective shadow maps (PRSM) of this DPRSM is randomly selected at a time according to a Russian roulette [AK90];
- novel methods for computing a CDF;
- an MIS (Multiple Importance Sampling) method combining an inverse transform method (for computing CDF) with a gathering approach.

1.3 Organization of the dissertation

This thesis is divided into three main parts:

1. **Part 1- Background on global illumination:** we will introduce in the chapter2 the fundamental formulation and equations to formalize the light transport function. In the chapter 3 we will present the Monte Carlo integration and the methods that used Monte Carlo integration for approximating a solution to the rendering equation.
2. **Part 2- State of the art on voxel and VPL based methods:** in this part we will present voxel-based methods for approximating the visibility computation during the rendering step (4). Where we will describe in the chapter 5 the diffirent methods for VPL-based rendering.
3. **Part 3- Contributions and results:** we will show an overview of our methods for computing the CDF. In chapter 6 we describe how to select one PRSM using Russian roulette. Our inverse transform methods (local CDF and gathering-based global CDF) aiming at selecting the more contributive VPLs are presented in chapter 7. As well as we will detail in this chapter

our proposed combined technique based on the MIS principle, with some experimental results.

Part I

Background on global illumination

Chapter 2

Mathematical fundamentals on global illumination

2.1 Introduction

In this chapter, we will present the basic radiometric quantities used in the global illumination field, as well as all mathematical equations to approximate the calculation of the lighting. Light is a set of photons, each of them has a certain energy. These photons are distributed in the electromagnetic spectrum. The visible part in the electromagnetic spectrum is ranged between ultraviolet and infrared radiation with $\lambda \in [380, 780]$ nanometers (nm) where λ represents the wavelength (see figure 2.1).
The following formula shows the energetic light power Φ (measured in Watts):

$$\Phi = \int_{380nm}^{780nm} \Phi(\lambda) d\lambda \tag{2.1}$$

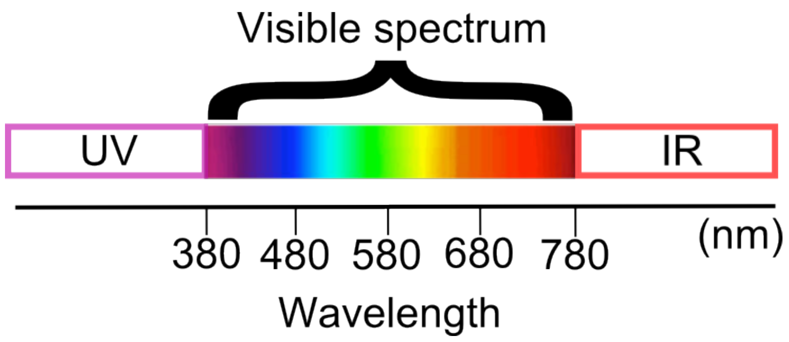


FIGURE 2.1: The electromagnetic spectrum.

The techniques for computing the illumination in computer graphics are based on the light distribution over a 3D surface.

2.2 Solid Angle

The solid angle describes the area taken by an object projected onto the unit sphere centred at a point x. In the global illumination domain, the solid angle represents a set of directions around a sphere. From now on, the solid angle is denoted by ω. It can be computed by the ratio between the projected area A and the radius square r of the sphere as in the following equation 2.2

$$\omega = \frac{A}{r^2} \tag{2.2}$$

We can approximate the solid angle around a point x onto a small area by 2.3:

$$d\omega = \frac{Acos\theta}{r^2} \tag{2.3}$$

Where $cos\theta$ is the angle between the axis r and the surface normal N.
The figure below 2.2 illustrates an example of solid angle: Moreover, the solid angle can also be represented in spherical coordinates system as. This formula is popularly used in the rendering domain because we integrate around a sphere (hemisphere). The formula is described as follows 2.4:

$$d\omega = sin\theta d\theta d\varphi \tag{2.4}$$

The following figure illustrates the solid angle in the spherical coordinates system 2.3:

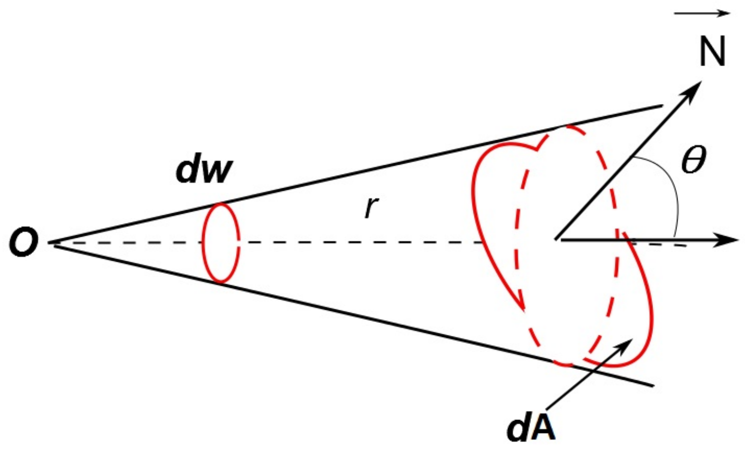


FIGURE 2.2: Illustration of the solid angle: $d\omega$ is a differential angle solid projected over a differential area dA.

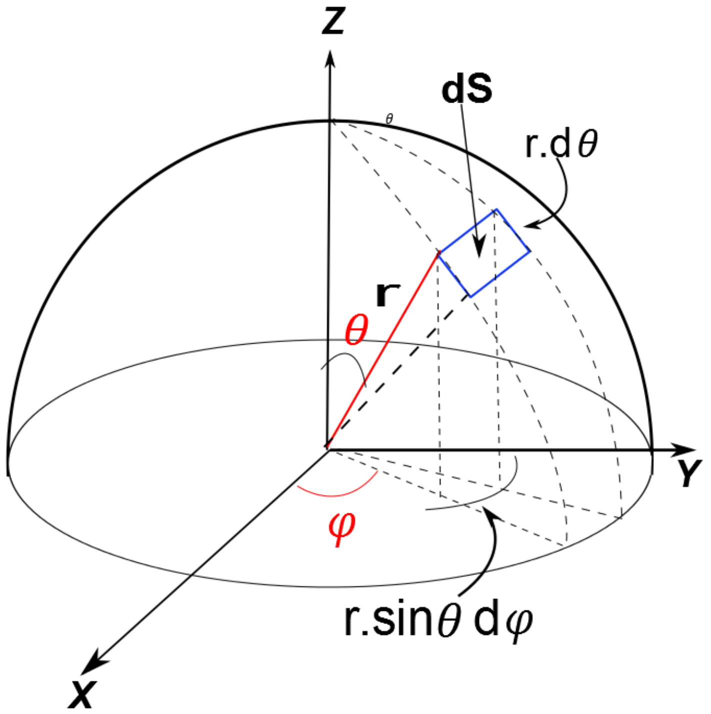


FIGURE 2.3: Illustration of the solid angle: $d\omega$ is a differential angle solid projected over a differential area dA.

2.3 Radiometric quantities

Radiometry domain is an energetic system that allows quantifying the energy of all types of radiations. While the photometry domain concerns only the visible radiations which define in $\lambda \in [380, 780]$.

The Radiometric quantities are physical measurements that characterised the different light phenomena. In computer graphics domain, we use these quantities to define the rendering equation.

1. **Radiant flux (light power):** denoted as Φ and measured in Watts (W). It expresses how much radiant energy Q is propagated from, through, or to a surface by a unit of time t:
$$\Phi = \frac{dQ}{dt} \tag{2.5}$$

2. **Irradiance:** it corresponds to the radiant flux Φ received per a unit of surface area A_1. It denoted as E and measured in $W.m^{-2}$:
$$E(x \leftarrow \omega) = \frac{d\Phi}{dA_1(x)} \tag{2.6}$$

3. **Radiant Exitance**: is the radiant flux leaving a surface area A_2. It called also Radiosity. Notated by B and measured in $W.m^{-2}$:
$$B(x \rightarrow \omega) = \frac{d\Phi}{dA_2(x)} \tag{2.7}$$

4. **Radiance:** is the radiant flux Φ per unit of solid angle ω per unit of surface area A:
$$L(x \rightarrow \omega) = \frac{d\Phi}{d\omega dA(x) cos\theta} \tag{2.8}$$
Where θ is the angle between the normal of the surface area A and the direction ω. The luminance is measured in $W.m^{-2}.sr^{-2}$

5. **Intensity:** it is the radiant flux Φ emitted by a point light source per unit of solid angle ω. It is measured in $W.sr^{-1}$:
$$I(x \rightarrow \omega) = \frac{d\Phi}{d\omega} \tag{2.9}$$

2.4 Bidirectional Reflectance Distribution Function

When the light reaches a surface, it can be reflected, or transmitted. A part of this energy can be absorbed. The function which defines the energy distribution through a surface is called BSSRDF(Bidirectional surface scattering reflectance distribution function). For simplification, the BSSRDF is replaced with the BRDF. This end expresses the ratio between the reflected radiance in a given direction and the irradiance. The BRDF considers only the light arriving at a point x of a surface and the light reflected from this same point. It used to describe the properties of the diffuse reflection, the rough specular or the ideal specular (glossy) materials. When the light is coming on a surface, it reflected the energy. The amount of the radiance reflected from the point x is given by:

$$dL_o(x \rightarrow \omega_o) = f_r(x, \omega_i, \omega_o) dE(x \leftarrow \omega_i) \tag{2.10}$$

For a single light wavelength, the BRDF is given by the following formula (equation2.11):

$$f_r(x, \omega_i, \omega_o) = \frac{radiance}{irradiance} = \frac{dL_o(x \rightarrow \omega_o)}{dE(x \leftarrow \omega_i)} = \frac{dL_o(x \rightarrow \omega_o)}{L_i(x \leftarrow \omega_i) cos\theta_i d\omega_i} \tag{2.11}$$

Where θ_i is the angle between the normal of the surface and the direction ω_i with $cos\theta_i$ is the dot product of the normal N and the direction ω_i. (see figure 2.4 for more details).
The figure below (2.4) illustrates the BRDF function: The BRDF has two proper-

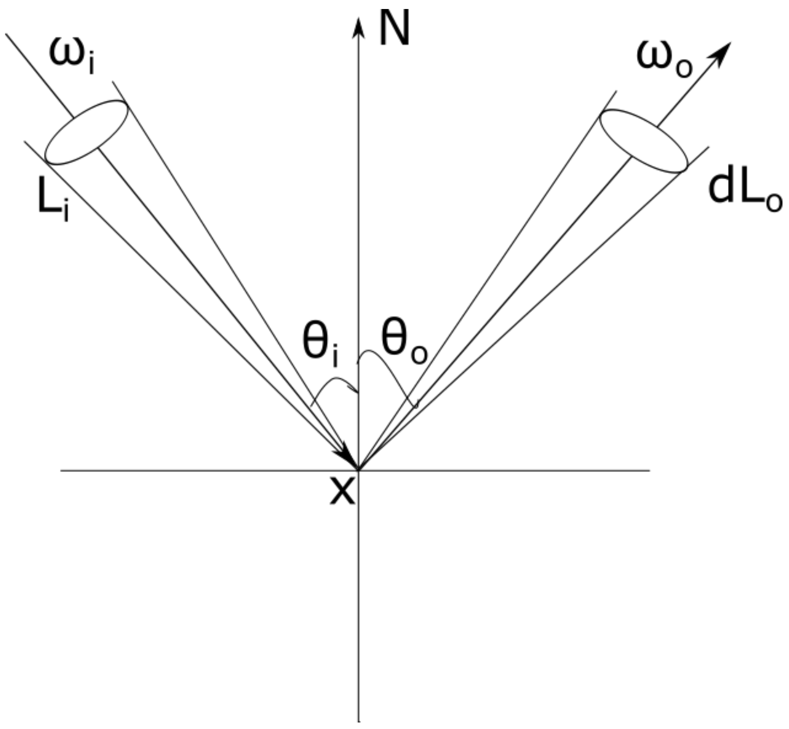


FIGURE 2.4: Illustration of the BRDF function: the relation between the incoming radiance L_i and the outgoing radiance L_o.

ties which are: reciprocity and energy conservation. The reciprocity property also called Helmholtz reciprocity is given by:

$$fr(x, \omega_i, \omega_o) = fr(x, \omega_o, \omega_i) \tag{2.12}$$

This property means that the BRDF at a point x coming from the direction ω_i and reflected into direction ω_o is the same as the BRDF at a point x coming from the direction ω_o and reflected into direction ω_i.
The energy conservation property expresses that for all the directions ω_i and ω_o over the hemisphere Ω_+, the total energy quantity that is reflected can not be higher than the incident quantity energy. The conservation energy property is given by:

$$\int_{\Omega_+} f_r(x, \omega_i, \omega_o) cos\theta_i d\omega_i \leqslant 1 \tag{2.13}$$

2.4.1 Diffuse surfaces

The diffuse surfaces (Lambertian surfaces) are the surfaces that uniformly reflect the energy in all the directions over the hemisphere (see figure 2.5).

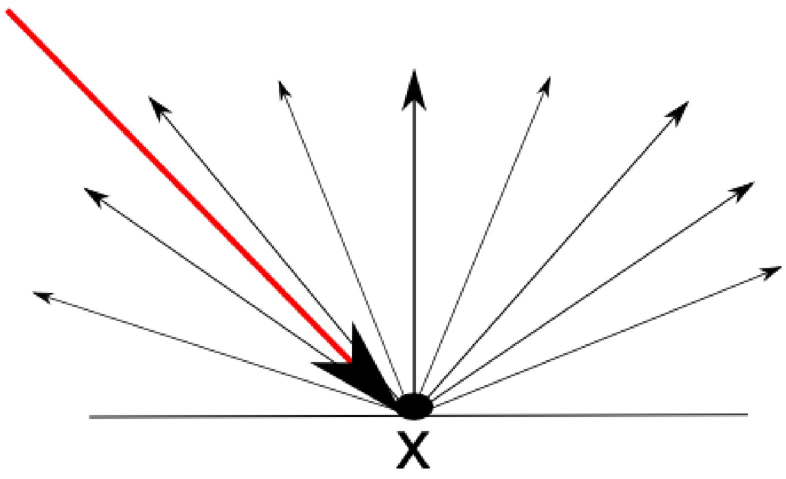


FIGURE 2.5

Let ρ_d is the diffuse albedo of a surface. It represents the fraction of the incident irradiance reflected into the radiosity B:

$$\rho_d = \frac{B_o(x \rightarrow \omega_o)}{E_o(x \leftarrow \omega_i)} \tag{2.14}$$

For a perfect Lambertian surface, the BRDF can be define as:

$$fr(x, \omega_i, \omega_o) = \frac{\rho_d}{\pi} = k_d \tag{2.15}$$

2.5 Rendering equation

In this section, we present the basic rendering equation that is physically modelling the lighting distribution problem in a $3D$ scene.
In the field of computer graphics, the generation of images requires an accurate calculation of the light distribution in a 3D scene, the calculation of a point color in a 3D model requires the integration of the whole incident illumination. Obviously. Kajiya [Kaj86] has been introduced in 1986 the rendering equation that is used to determine the outgoing radiance at a point x in a surface A. However, it is difficult to solve this recursive integral analytically. For that, there are many techniques to approximate the solution of the rendering equation.
The light transport does not depends on only to the self-emittance at a point x but to its reflections and refractions properties. Thus, the light distribution can be expressed in the direction domain or in the area domain. In the following, we will illustrate how the rendering equation is represented in each domain. For more details, please refer to the Physically Based Rendering Techniques (PBRT) book [HP04].

2.5.1 Direction domain

Using the definition of the BRDF (2.11), we can deduce the outgoing radiance dL_o at a point x in a differential direction $d\omega$ as:

$$dL_o(x \to \omega_o) = fr(x, \omega_i, \omega_o) L_i(x \leftarrow \omega_i) cos\theta_i d\omega_i \tag{2.16}$$

To find the radiance in all the direction domain, we integrate the equation 2.16:

$$L_o(x \to \omega_o) = \int_{H^2} fr(x, \omega_i, \omega_o) L_i(x \leftarrow \omega_i) cos\theta_i d\omega_i \tag{2.17}$$

To generate the formula, we add the self emittance L_e at a point x 2.18:

$$\underbrace{L_o(x \rightarrow \omega_o)}_{total \quad \text{radiance}} = \underbrace{L_e(x \rightarrow \omega_o)}_{emitted \quad \text{radiance}} + \underbrace{\int_{H^2} fr(x, \omega_i, \omega_o) L_i(x \leftarrow \omega_i) cos\theta_i d\omega_i}_{reflected \quad \text{radiance}} \tag{2.18}$$

Where $\omega \in H^2$ and H in the hemisphere around the point x
After some interactions, the outgoing radiance can be an incident radiance of another point. Therefore, the integral that represents the expression of the light transport will be recursively.

2.5.2 Area domain

Our goal is now to transform the light transport equation represents in the direction domain to another one represents in the area domain. We integer the light transport over all the surfaces.
The following figure 2.6 represents the geometry of the light transport in the area domain: To transform the rendering equation to an area domain, it is necessary

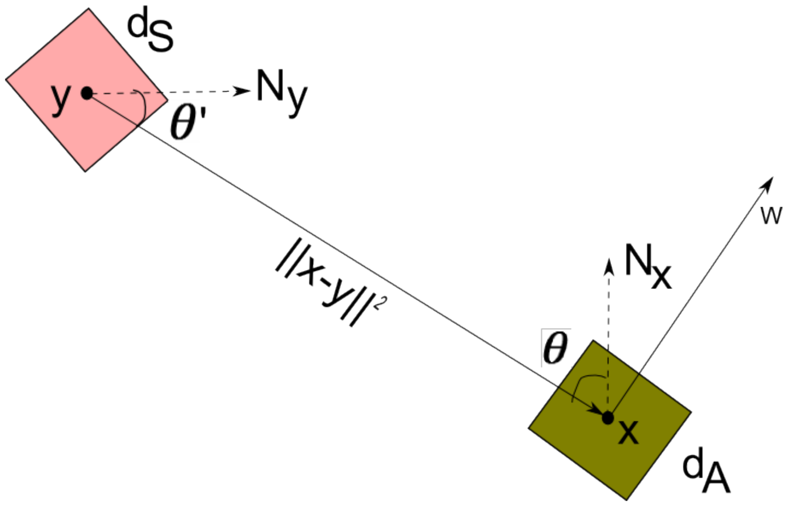


Figure 2.6: Illustration of the light transport geometry.

to reformulate the solid angle $d\omega$ by a differential surfaces unit dA as follows 2.19:

$$d\omega = \frac{cos\theta ds}{\|\mathbf{x} - \mathbf{y}\|^2} \tag{2.19}$$

Let us call $V(x, y)$ the visibility function between x and y (visibility $= 1$ if the point x is visible from point y, and 0 otherwise). $G(x, y)$ is the geometry term which defines the relation between the orientation and the distance between the points on the different surfaces. Where:

$$G(x, y) = \frac{cos\theta cos\theta'}{\|\mathbf{x} - \mathbf{y}\|^2} \tag{2.20}$$

The equation 2.18 becomes:

$$L_o(x \to \omega_o) = L_e(x \to \omega_o) + \int_A fr(x, \omega_i, \omega_o) L_i(x \leftarrow \omega_i) V(x, y) G(x, y) d_S \tag{2.21}$$

Equation 2.21 expresses the global illumination model in the area domain. It describes the light transport mechanism between areas. In this thesis, we use both global illumination formulation in direction domain and in area domain.
There are several approaches to resolve the light transport equation (in direction domain or in area domain). In this thesis, we are interested in the Monte Carlo rendering techniques.

2.6 Conclusion

The rendering equation consists in calculating the distribution of the light at different points in a scene, it is difficult to solve it by the analytical methods, for this reason, there are several methods that have been developed to simplify the resolution of the rendering equation. The approximate method consists in discretizing the scene in small surfaces called patch, this discretization offers an advantage of transforming the rendering integral into a system of equations, then calculating the radiance in each patch. The main problem of this method is that it allows simulating only the diffuse effects. The most methods used today to solve the rendering equation are Monte Carlo methods which are based on probabilistic sampling. In the next chapter, we will describe in detail the Monte Carlo integration and the different rendering techniques that used Monte Carlo integration to approximate the global illumination effects.

Chapter 3

Theory behind Monte Carlo integration

3.1 Introduction

In the previous chapter, we have introduced some basic mathematical concepts and formulas which are used in the global illumination domain. As we have shown, there are several approaches to solve the light transport integral. It is practically impossible to find an analytic solution to the rendering equation, especially for the complex geometry scenes. In this section, we will demonstrate an efficient way to solve the rendering equation by using the Monte Carlo integration. Moreover, this integration to the global illumination problem can be a robust solution because it is based on random sampling. It generates realistic rendering scenes and it is considered as the most popular solution to the global illumination problem.

3.2 Probabilistic concepts

To simulate the light distribution, it is necessary to resolve the rendering integration. In practice, it is impossible to find an analytical solution. For this reasons, the most efficient solution is the use of Monte Carlo estimator. In this section, we define how we can use the Monte Carlo integration to resolve the rendering equation. We also show some algorithms and methods that used Monte Carlo methods in computer graphics domain.

3.2.1 Discrete random variable

A discrete random variable X is a variable that takes a countable random values. In a domain Ω, we note $X(\Omega) = \{x_1, x_2, ..., x_k, ...\}$ where x_k is the outcome of the discrete random variable X. Each variable x_k has a probability mass function p, where $p(x_k) \in [0; 1]$. If $X(\Omega)$ takes a finite elements then the probabilities sum of all the variable x_k is equal to 1.

$$p(X = x_k) \geq 1 \tag{3.1}$$

and

$$\sum_{x_k \in X(\Omega)} p(X = x_k) = 1 \tag{3.2}$$

A Cumulative Distribution Function (CDF) of a discrete random variable X is the probability gives to a variable which takes a value less than or equal to x_k

$$CDF(X) = P(X) = p(X \leq x_k) \tag{3.3}$$

for all i = 1,...,k

$$P(X) = \sum_{x_i \leq x_k} p(x_i) \tag{3.4}$$

3.2.2 Continuous random variables

A continuous random variable X is a variable which takes its values in uncountably infinite of outcomes x_k from its domain. Let us define the Probability Distribution Function of a continuous random variable $X \in [a; b]$, the probability affected to a such variable x in the small interval $[a; b]$. It is noted by $p(x)dx$.
The Cumulative Distribution Function (CDF) of a continuous random variable, is expressed by:

$$CDF(X) = P(X) = P(x \in [a; b]) = \int_a^b p(x)dx \tag{3.5}$$

3.2.3 Expected value

We can define the expected value as the mean of a probability function. Moreover, we have two expected value types according to the random variable.
The expected value of a discrete random variable X is defined as:

$$E(X) = \sum_{i=1}^{k} x_i p(x_i) \tag{3.6}$$

For a continuous random variable $X \in \Omega$, its expected value is given by:

$$E(X) = \int_{\Omega} x_i p(x_i) dx_i \tag{3.7}$$

Moreover, we can express the expected value of a random variable function as:

$$E(f(x)) = \int_{\Omega} f(x) p(x) dx_i \tag{3.8}$$

3.2.4 Variance

The variance is the deviation (the difference) of a random variable from its mean. The variance of a random variable X is defined as:

$$V[X] = \sigma^2 = E[(X - E[X])^2] \tag{3.9}$$

In other words:

$$V[X] = \sigma^2 = E[X^2] - E[X]^2 \tag{3.10}$$

3.3 Monte Carlo integration

The Monte Carlo (MC) methods are the methods that based on a stochastic sampling. In computer graphics, the Monte Carlo integration is used to approximate the rendering equation. Where it is very difficult to provide an exact analytical solution.
We consider the integral I that defined a function $f(x)$ over a domain Ω. The goal

of Monte Carlo integration is to evaluate the integral I:

$$I = \int_{\Omega} f(x)dx \tag{3.11}$$

Monte Carlo approach approximates the integral I by selecting N random samples with a certain pdf. The Monte Carlo estimator is given by:

$$\hat{I_N} = \frac{1}{N}\sum_{i=1}^{N}\frac{f(x_i)}{p(x_i)} \tag{3.12}$$

We have:

$$\lim_{N\longrightarrow+\infty}\hat{I_N} = \lim_{N\longrightarrow+\infty}\frac{1}{N}\sum_{i=1}^{N}f(x_i) \approx I_N \tag{3.13}$$

We can rewrite the equation 2.17 as:

$$L_o(x \to \omega_o) = \int_{H^2}\frac{fr(x,\omega_i,\omega_o)L_i(x \leftarrow \omega_i)cos\theta_i d\omega_i}{p(x)}p(x) \tag{3.14}$$

In other way:

$$L_o(x \to \omega_o) = \int_{H^2} g(x)p(x) \tag{3.15}$$

Where:

$$g(x) = \frac{fr(x,\omega_i,\omega_o)L_i(x \leftarrow \omega_i)cos\theta_i d\omega_i}{p(x)} \tag{3.16}$$

In this case, the rendering equation is approximated 2.17 by using the Monte Carlo model as:

$$L_o(x \to \omega_o) \approx \frac{1}{N}\sum_{i=1}^{N} g(x_i)p(x_i) \tag{3.17}$$

3.4 Inverse transform method

To compute the indirect lighting using Monte Carlo integration, it is necessary to choose some random points in a certain domain. There are existed different methods to select random points such as inverse transform, rejection, and metropolis methods. In this section, we describe the inverse transform method that will be used in our work to generate the VPLs.

The goal of this method is to generate random variables which correspond to a certain pdf. For that, we should compute the Cumulative Distribution Function

(CDF).
Furthermore, we define the CDF function for two kinds of distributions, discrete and continuous distributions. We define the CDF and the PDF functions in the discrete domain as:

$$CDF(i) = \sum_{j=1}^{i} PDF(i) \tag{3.18}$$

And:

$$PDF(i) = CDF(i+1) - CDF(i) \tag{3.19}$$

As the PDF distribution, the CDF function is distributed between 0 and 1 (see figure 3.1 for more details).

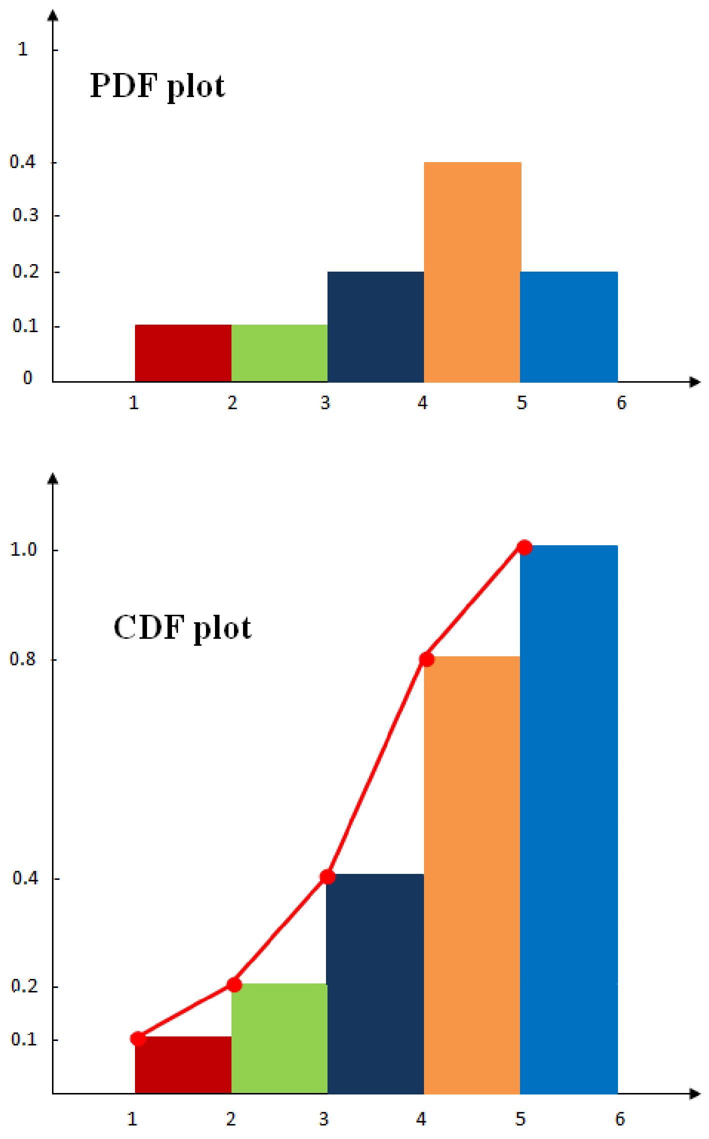


FIGURE 3.1: Example of the PDF and CDF distribution in the discrete domain.

For continuous distribution, the PDF and the CDF functions are described as follows:
Let us call $p(x)$ the probability distribution function of the variable x, defined over the interval $[x_{min}, x_{max}]$, we can sample a random variable x which is distributed a from according to p from a set of uniform random numbers $\xi_i \in [0, 1]$.
The inverse is produced in two phases:

1. Integrate the PDF p(x) to obtain the Cumulative Distribution Function $P(x) = \int_{x_{min}}^{x} p(x)dx$.
2. inverse $P(x)$ by using a uniform random numbers ξ_i. $x_i = P^{-1}\xi_i$.
   Where P^{-1} is the inverse function of the CDF P.
   See figure 3.2

In the following, we show an example for sampling variables using the inverse transform method:

3.4.1 Uniform sampling of triangles

1. **CDF computation:**
   First, we compute the barycentric of the triangle as: $x = \alpha A + \beta B + \gamma C$, where $\alpha + \beta + \gamma = 1$.
   We get: $\alpha = 1 - \beta - \gamma$. So the barycentric point x is expressed by two unknown variables β and γ $x = (1 - \beta - \gamma)A + \beta B + \gamma C$ where $\beta + \gamma = 1$
   Integrating the constant 1 across the triangle gives:

$$\int_{\gamma=0}^{1} \int_{\beta=0}^{1-\gamma} d\beta d\gamma = 0.5 \tag{3.20}$$

   In order that the PDF becomes uniform, so our pdf is:

$$p(\beta, \gamma) = 2 \tag{3.21}$$

   Since β depends on γ (or γ depends on β), we use the marginal density for γ, $p_G(\gamma)$:

$$p_G(\gamma) = \int_{\gamma=0}^{1-\gamma} 2d\beta = 2 - 2\gamma \tag{3.22}$$

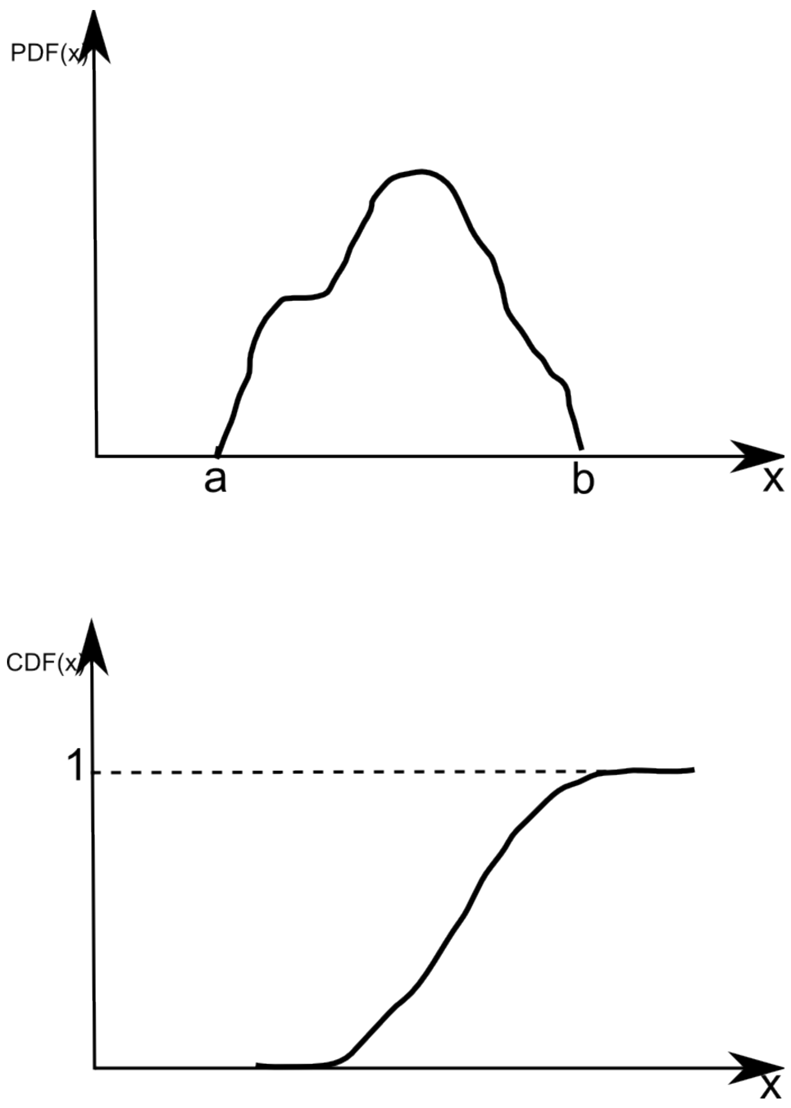


FIGURE 3.2: Shapes of PDF and CDF functions. First row: the PDF function in the contiuous distribution. Second row: the CDF function that is the integration of the PDF function.

and then, From $p_G(\gamma)$, we find $p(\beta|\gamma)$

$$p(\beta|\gamma) = \frac{p(\gamma, \beta)}{p_G(\gamma)} \tag{3.23}$$

$$p(\beta|\gamma) = \frac{2}{(2 - 2\gamma)} = \frac{1}{(1 - \gamma)} \tag{3.24}$$

2. **Transform the CDF:**

   To find γ we look at the cummulative pdf for γ:

We generate the first uniform random variable ξ_1:

$$\xi_1 = P_G(\gamma) = \int_{\gamma=0}^{\gamma} p_G(\gamma) d\gamma \tag{3.25}$$

$$\xi_1 = P_G(\gamma) = \int_{\gamma=0}^{\gamma} 2 - 2\gamma d\gamma = 2\gamma - \gamma^2 \tag{3.26}$$

Solving for γ we get:

$$\gamma = 1 - \sqrt{1 - \xi_1} \tag{3.27}$$

Then, we generate a second random variable ξ_2 to find β:

$$\xi_2 = P(\beta|\gamma) = \int_{\beta=0}^{\beta} p(\beta|\gamma) d\beta \tag{3.28}$$

$$\xi_2 = \int_{\beta=0}^{\beta} \frac{1}{1-\gamma} d\beta = \frac{\beta}{1-\gamma} \tag{3.29}$$

Solving for β we get:

$$\beta = \xi_2 \sqrt{1 - \xi_1} \tag{3.30}$$

Thus given a set of random numbers ξ_1 and ξ_2, we warp these to a set of barycentric coordinates sampling a triangle:

$$(\beta, \gamma) = (\xi_2 \sqrt{1 - \xi_1}, 1 - \sqrt{1 - \xi_1}) \tag{3.31}$$

3.5 Importance sampling

It is a Monte Carlo approximation method where the expected value of the light transport equation is approximated by a mean value (see equation 3.17). When using the importance sampling technique, the distribution will be non-uniform. The importance sampling method (IS) allows choosing the best PDF distribution. In other words, it is necessary to choose the distribution where its shape will be very similar to the integrand shape in order to reduce the variance value.
The IS technique is a variance reduction technique that can efficiently approximate the light transport integral which leads to converge the solution between the distribution and the integrand. (see figure 3.3).

Now, lets go back to the equation 2.17. To approximate this integral using im-

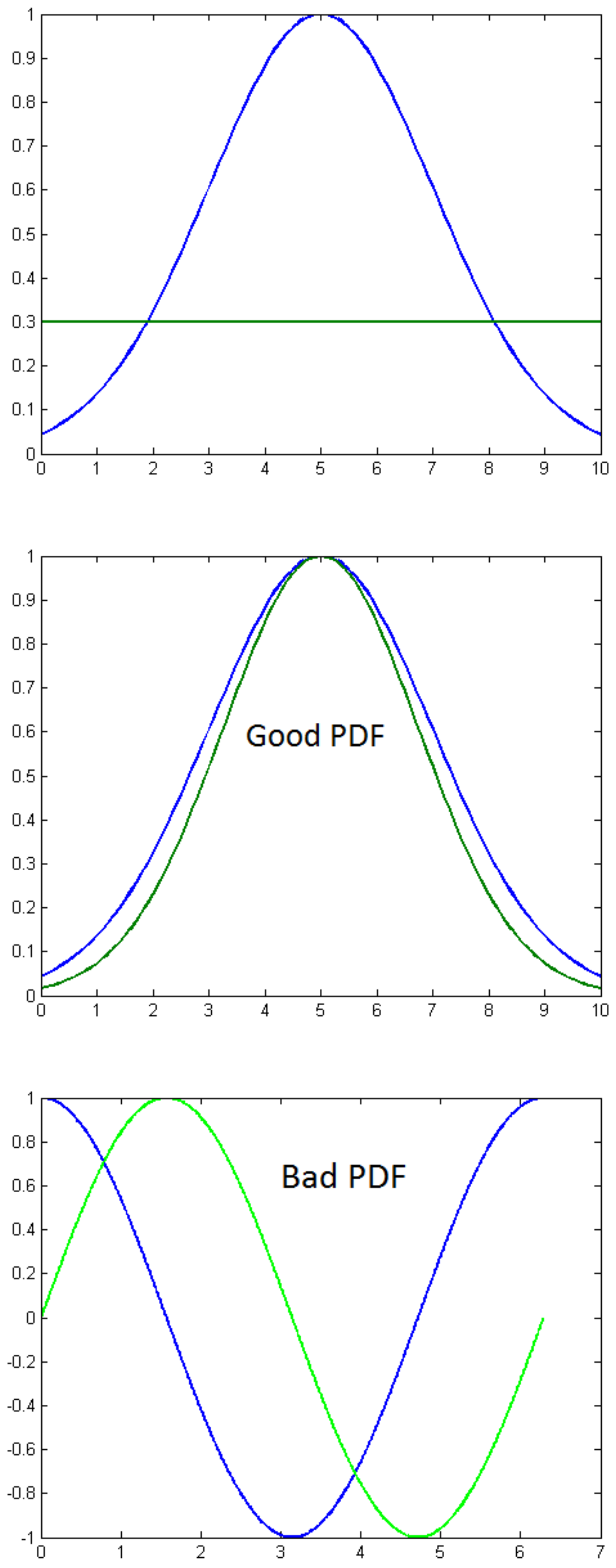


Figure 3.3: different Integrand and PDF plots: first row: uniform distribution with constant PDF (in green), second row: PDF (in green) is closer to the integrand (in blue) than in the third row.

portance sampling method, we should find the PDF that minimizes the variance. We put:

$$p(\omega_i) = \frac{cos\theta}{\pi}[\text{Dut03}] \tag{3.32}$$

The CDF of a such distribution is:

$$P(\omega_i) = \int_{\Omega} p(\omega_i) d\omega_i \tag{3.33}$$

Because we integrate the rendering equation over the hemisphere, we can formulate ω_i by θ and φ angles:

$$P(\omega_i) = \int_0^{2\pi} \int_0^{\frac{\pi}{2}} \frac{cos\theta}{\pi} sin\theta d\theta d\varphi \tag{3.34}$$

$$P(\omega_i) = \frac{1}{\pi} \int_0^{2\pi} \int_0^{\frac{\pi}{2}} cos\theta sin\theta d\theta d\varphi \tag{3.35}$$

We use the inverse transform method (see section 3.4) , we obtain:

$$P(\omega_i) = \varphi \frac{1}{2\pi} sin^2\theta = \frac{\varphi}{2\pi}(1 - cos^2\theta) \tag{3.36}$$

We choose two uniform random variables ξ_1 and ξ_2, where:

$$\xi_1 = cos^2\theta \Rightarrow \theta = arccos\sqrt{\xi_1} \tag{3.37}$$

And

$$\xi_2 = \frac{\varphi}{2\pi} \Rightarrow \varphi = 2\pi\xi_2 \tag{3.38}$$

3.6 Multiple Importance Sampling

In some cases, we need to approximate the light transport integral by using importance sampling technique but for more than one strategy. Veach [Vea97] has been proposed a Multiple Importance Sampling (MIS) technique that allows combining two (or more) rendering strategies, in order to get an efficient estimator with a low variance as much as possible.
In this section, we describe the MIS technique and we show how we apply it in the rendering domain. Veach [Vea97] has been introduced a formulation which

permet to approximate the Monte Carlo integration as follows:

$$F = \frac{1}{n} \sum_{i=1}^{k} \sum_{j=1}^{n} \omega_i(x_{ij}) \frac{f(x_{ij})}{p_i(x_{ij})} \tag{3.39}$$

Where F is the MIS estimator, k is the number of strategies (estimators), n is the number of samples (number of directions for example), $f(x_{ij})$ is the contribution of the light path, $p_i(x_{ij})$ is the probability associated to the estimator $f(x_{ij})$ and $\omega_i(x_{ij})$ is the weight of each sample.
The weights $\omega_i(x_{ij})$ are used to determine which strategy is more robust and efficient than the others. The balance heuristic function allows to compute the $\omega_i(x_{ij})$ as follows:

$$\omega_i(x_{ij}) = \frac{p_i(x_{ij})}{\sum_{k=1}^{nbStrategy} p_k(x_{ij})} \tag{3.40}$$

3.7 Conclusion

In this chapter, entitled theory behind Monte Carlo integration, we have described the Monte Carlo algorithm. This algorithm, consisting mainly of two passes:

- **first pass:** consists of searching the good pdf that converges the solution of the global illumination (in general by using importance sampling technique).
- **second phase:** consists of transforming the light transport problem from the continuous domain to the discrete one.

After the detailed description of the technique chosen (Monte Carlo integration). We describe in the next part, we will cite the different related work concerning the computation visibility which is the main cost step in the rendering process it is now time to implement it.

Part II

State of the art on voxel and VPL based methods

Chapter 4

Voxel-based models

4.1 Introduction

In this chapter, we define the voxelization method which is used to compute the visibility during the rendering process. We show in section 4.2 the principal idea of the voxelization. Then we present in section 4.3 the ray marching algorithm that is used on our work and that allows us to browse in the voxelized scene. In addition, we see in section 4.4 some existing methods that used the voxelization in the global illumination domain.

4.2 Voxelization

The voxelization is the process of transforming a set of triangles that represent the scene to a set of voxels. In other words, it describes the passage from a continuous representation to a discrete one. The voxelization is the best approximation to the continuous geometry. It can be used to minimize the ray object intersection when computing the global illumination.
The 3D scene is inserted into an Axis Aligned Bounding Box (AABB). Then, the scene is subdivided into a grid of 3D cells. Each cell in the three-dimensional regular grid contains some information about the scene. When the cells contain the presence of the geometry information, we call this voxelization type as binary

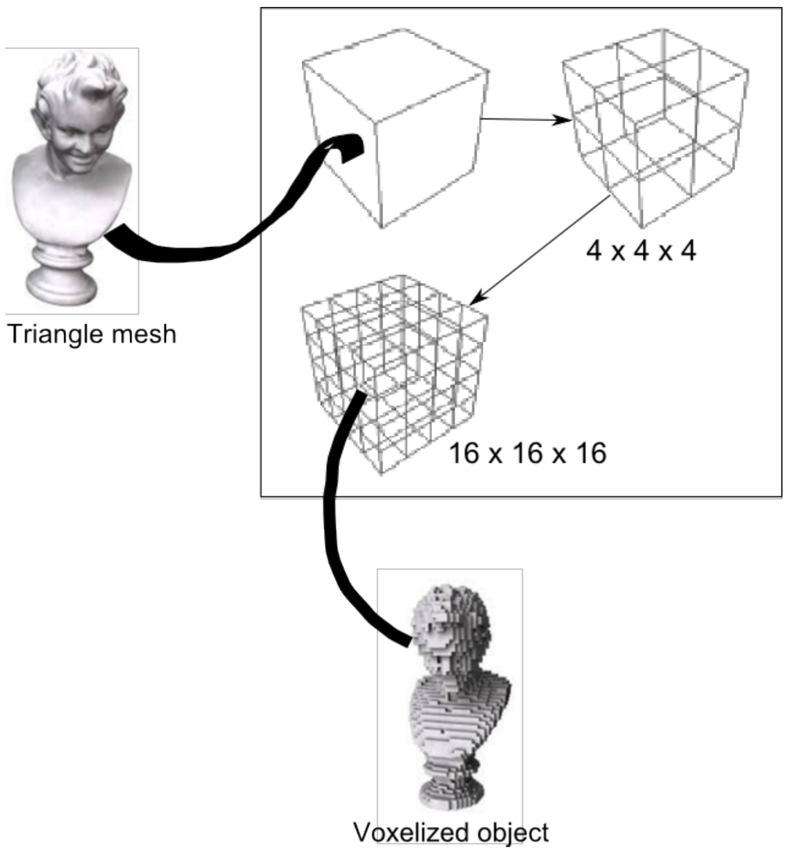


FIGURE 4.1: Transformation of a triangle mesh to a voxelized representation.

voxelization. In the other case, the multi valued voxelization, when the cell contains other information, such as normal, position or material. The figure below (figure 4.1) illustrates the voxelization process:

4.3 Ray marching

After create and subdivide the bounding box that encompasses the entire scene. The second step consists of browsing the 3D volume (bounding box) in order to determine the next cell and verify if the object geometry is partially or totally within this cell or not. Using the ray marching algorithm has the advantage that it is not necessary to compute the ray object analytically as in the classical ray tracing. We present in the algorithm 1 the most robust ray marching algorithm (Amanatide's algorithm) [AW+87]:

Algorithm 1 amanatide(3D_texture_image grid)

```
1: o = origin_ray(); // return the ray origin
2: v0 = first_voxel(grid); // determine the first voxel that contains the the ray origin
3: if outside(o, grid) == true then
4:     // the ray origin is outside of the grid
5:     entry = search_entryPoint() // search the entry point
6: end if
7: step(stepX, stepY, stepZ) = (1, 1, 1);
8: t = (tx, ty, tz); // values of t corresponding to the points resulting from the intersection between the ray and 3 faces of the initial voxel
9: tDelta = (tDeltaX, tDeltaY, tDeltaZ); // distance travelled by the ray between two successive faces perpendicular to the x, y and z faces respectively
10: min = minimum(tx, ty, tz);
11: switch(min)
12: case tx :
13: X += stepX ;
14: tx += tDeltax ;
15: break ;
16: case ty
17: Y += stepY ;
18: ty += tDeltay ;
19: break ;
20: case ty
21: Z += stepZ ;
22: tz += tDeltaz ;
23: break ;
```

4.4 Voxel-based methods

Usually directly using triangles (modelling the geometry of a scene) in real-time rendering scenario can be not efficient. To reduce the intersection computation, a scene can be subdivided into voxels. Several methods have been proposed ([CNLE09]; [THGM11]; [CG12]). Hu et al [HHZ+14] have presented a new ray tracing method, programmable on the modern GPU, which uses an A-Buffer and

a grid voxelization to represent the scene geometry.
In this section, we present some related work techniques that used the voxelization in the rendering domain.

4.4.1 Voxel path tracing

It is the first method presented in voxel-based global illumination [THGM11] that allows calculating indirect illumination using a scene voxelization.The goal of such a representation is to accelerate the process of the ray-object intersection. The authors proposed a simple ray marching for computing intersection between the rays and the volume data and projecting the outgoing point of the hit voxel onto a Reflective Shadow Maps to determine the visibility from the point light source. Figure 4.2 illustrates the principle idea of the voxel path tracing: This method

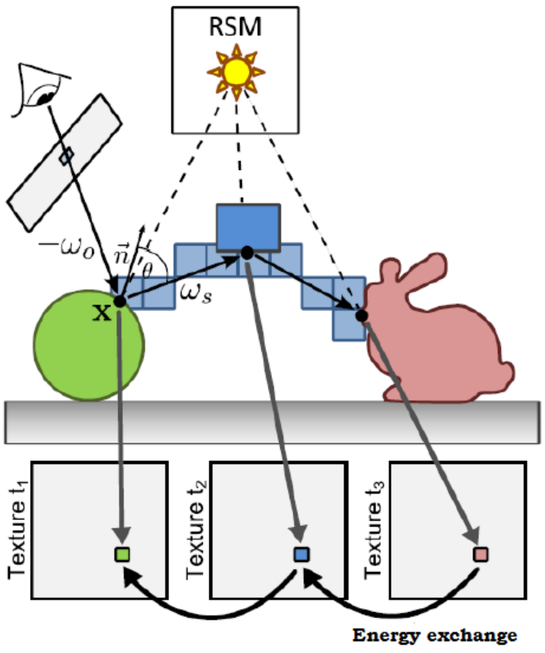


FIGURE 4.2: Voxel path tracing. [THGM11]

allows multi bounces rendering with diffuse and specular materials.

4.4.2 Voxel Cone Tracing

Voxel Cone Tracing (VCT) is a method very similar to the ray tracing method. It is used to accelerate the indirect lighting computation by tracing cones and pre-filtered voxel-based representation during the approximation of the indirect lighting. Crassin et al's [CNS$^+$11] have been proposed a method that simulates both ambient occlusion [KL05] and indirect lighting (with diffuse and specular materials) by using the sparse voxel octree structure. The sparse voxel octree structure is a mipmap hierarchical structure [Wil83] that is used to store the incoming radiance. Once the sparse voxel octree is constructed, the incoming radiance is estimated by the use of pre-filtred representation. The incoming radiance is then determined by performing a ray marching algorithm (to determine the visibility) is performed along each cone and by sampling the pre-filtred data structure in the mip level that corresponds to the selected cone diameter. The determination of the mip level is due using this formula:

$$mip_{level} = log_2(d_{circle}) \tag{4.1}$$

Where d is the diameter of the correspondent intersection circle in the cone tracing.
Crassin and Green [CG12] have been presented a new voxelization method using the GPU hardware rasterizer. This voxelization process is done as follows: first, each triangle in the 3D scene is projected along its dominant axis using an orthographic projection. The dominant axis is an x, y, or z axis which provides the maximum projected area. Then the corresponding voxel position is written into a 3D texture. For each mip level, the voxels are stored in a 3D texture at the corresponding resolution. The advantage of this method compared to the one proposed by Crassin et al [CNS$^+$11] is that the voxels are constructed according to the mip-map levels, and only the visible voxels will be inserted on the octree (3D texture).

4.4.3 Layered Reflective Shadow Maps

The Layered reflective shadow maps (LRSM) method proposed by Sugihara et al [SRS14] is inspired by voxel cone tracing [CG12]. It is an efficient method that uses voxel cone tracing [CG12] to simulate both diffuse and specular indirect lighting. In this paper, the authors propose to divide the RSM into n layers in order to speed up the rendering time and they used it combined to voxel cone tracing technique for computing the indirect lighting. The LRSM method uses voxels only for computing the visibility, where the VCT method [CNS$^+$11] stores more information (example normal, position,... etc) at each voxel, which leads to a large memory consumption. The LRSM algorithm is composed of three main steps that are summarized in the following figure (figure 4.3).

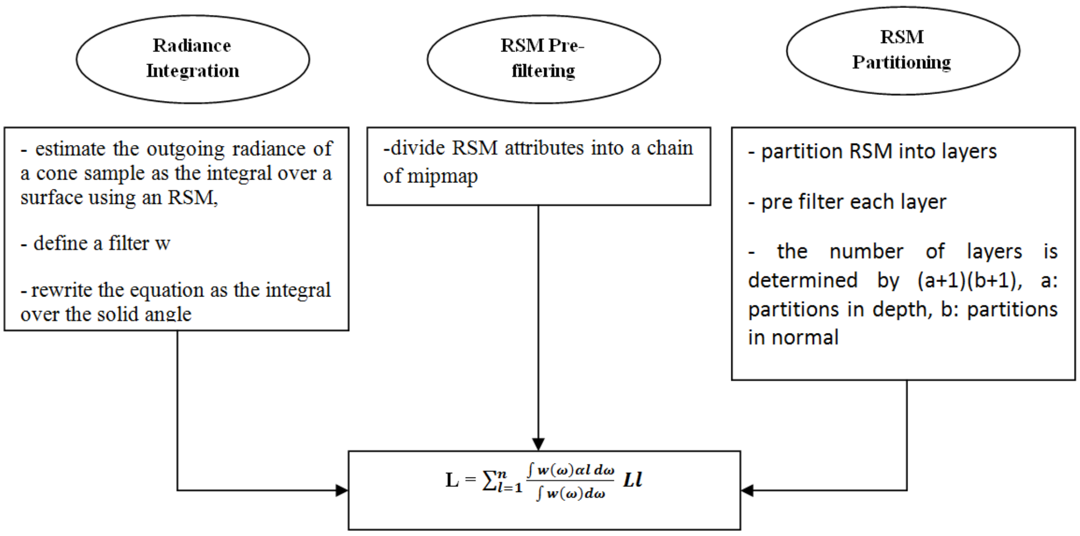

Radiance Integration
RSM Pre-filtering
RSM Partitioning
- estimate the outgoing radiance of a cone sample as the integral over a surface using an RSM,
- define a filter w
- rewrite the equation as the integral over the solid angle
-divide RSM attributes into a chain of mipmap
- partition RSM into layers
- pre filter each layer
- the number of layers is determined by (a+1)(b+1), a: partitions in depth, b: partitions in normal
$L = \sum_{l=1}^{n} \frac{\int w(\omega)\alpha l \, d\omega}{\int w(\omega) d\omega} Ll$

FIGURE 4.3: Layered Refelective Shadow Maps steps.

4.5 Conclusion

In this chapter, we have shown the motivation of using the voxelization in the rendering domain. The voxelization that consists of transforming a continuous geometry representation to a discrete one allows accelerating the visibility computation during the rendering process. When we simulate the indirect lighting with VPL, it should place a shadow map for each VPLs to compute the visibility term. But it is very costly. For this reason, we use the voxelization and the ray marching algorithm to efficiently compute the visibility. In the next chapter, we present some existing methods which simulate the indirect lighting by the use of the VPLs and which called VPL-based rendering methods.

Chapter 5

VPL-based rendering methods

5.1 Introduction

State-of-the-art solutions [RDGK12] to algorithms, that are used for realistic image synthesis, rely on path tracing, photon mapping, and radiosity methods. Each of them can perform efficiently in terms of time rendering or image photo-realism. We will focus our relative work on VPL [Kel97] and path tracing on the GPU. For the other techniques, the reader can refer to Ritschel's et al. state of the art [RDGK12].

5.2 Generating VPL

In the literature, there are several methods to approximate global illumination using Monte Carlo estimator and Importance sampling [ARBJ03] using a Probability Density Function (PDF). The main advantage of Importance Sampling is to minimize the variance error when the PDF is closer to the integrand. In off-line rendering, VPL generation can be performed using rejection sampling [GS10] or more complex sampling techniques based on Monte Carlo Markov Chain (MCMC) [Gil05]. When real-time is targeted, an approach based on Instant Radiosity and Shadow Mapping [Wil78], so-called Reflective Shadow Maps [DS05], has been proposed to approximate one bounce indirect lighting. It considers each pixel in the shadow maps as a secondary point light source defined by its world space coordinates, its normal and its flux, information that allows evaluating the contribution of each VPL. Computing the contribution of all the VPLs, stored in a Reflective Shadow Map, is time-consuming. This is why only a subset of VPLs is used to compute the indirect radiance of a pixel (see figure 5.1). Dachsbacher et al [DS05]

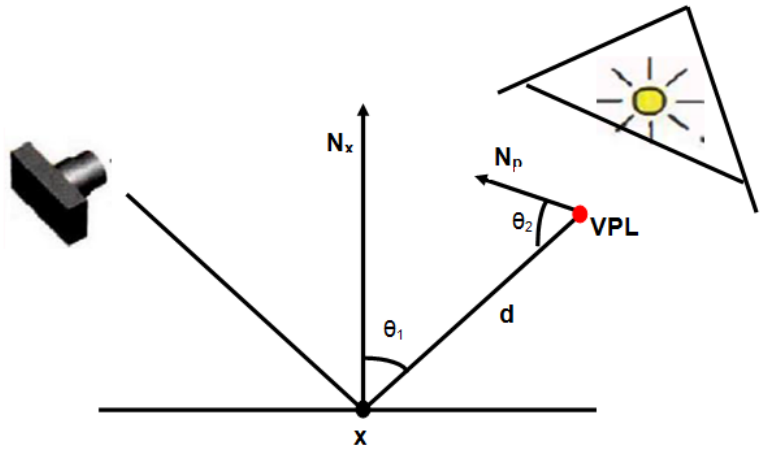


FIGURE 5.1: Reflective Shadow Maps scheme

propose two solutions to avoid the contribution evaluation of all the RSM pixels; which are:

1. **first solution:** for one visible point from the view camera, only a random subset of RSM pixels are used.
2. **second solution: screen space interpolation** the author suggested a strong assumption: if two points in the scene are close to each other, their projection on the screen will be close to each other too [DS05]. This solution has been realized in two passes
   - first pass: the authors proposed to render the screen (from the camera view) in a low resolution and to evaluate the indirect illumination for all the RSM pixels.
   - second pass: in this step, the scene is rendered in a full resolution. If the samples normal (the three or four surrounding low resolution samples) are similar to the pixels normal and if its world space position is close to pixel's position then a bilinear interpolation of the indirect lighting is performed.

The vpl contribution is calculated using equation 5.1:

$$contrib_{VPL} = \frac{1}{\pi}.gFactor.kd_x.fluxVPL \tag{5.1}$$

Where $gFactor$ represents the attinuation factor:

$$gFactor = \frac{cos\theta_1 cos\theta_2}{d^2} \tag{5.2}$$

And $fluxVPL$ defines the color reflected by the VPL from some bounces N:

$$fluxVPL = \frac{4\pi Ikd_v}{N} \tag{5.3}$$

Then the indirect illumination is approximated by summing the contributions of all the VPLs as in equation 5.4

$$indirect = \sum_{v=0}^{nbVPL} contrib_v \tag{5.4}$$

5.3 Evaluating visibility

The determination of the visibility term is the most expensive operation, especially in real-time rendering. Instant Radiosity [Kel97] is a popular technique that calculates indirect lighting due to a set of Virtual Point Lights (VPLs). Unlike the inverse transform method, Barák et al. [BBH13] exploit the performances of Metropolis-Hastings algorithm [Seg07] to determine the VPLs and use a small number of them to render the scene. Hedman et al [HKL16] have proposed a temporally coherent technique that allows sampling the VPLs in large scenes and enable frame to frame distribution for minimizing the VPL flickering.

5.3.1 Imperfect Shadow Maps (ISM)

Ritschel et al. [RGK+08], uses the observation that an approximate visibility term for VPLs is sufficient. The authors proposed a technique, called ”Imperfect Shadow Map”, which represents the scene surfaces by a set of points and splat them in parallel in the different shadow maps. Figure 5.2 illustrates an overview of the ISM technique:

The main steps of the ISM method are described in the figure 5.3. First, we have as input, a 3D surface geometry. The ISM algorithm consists of:

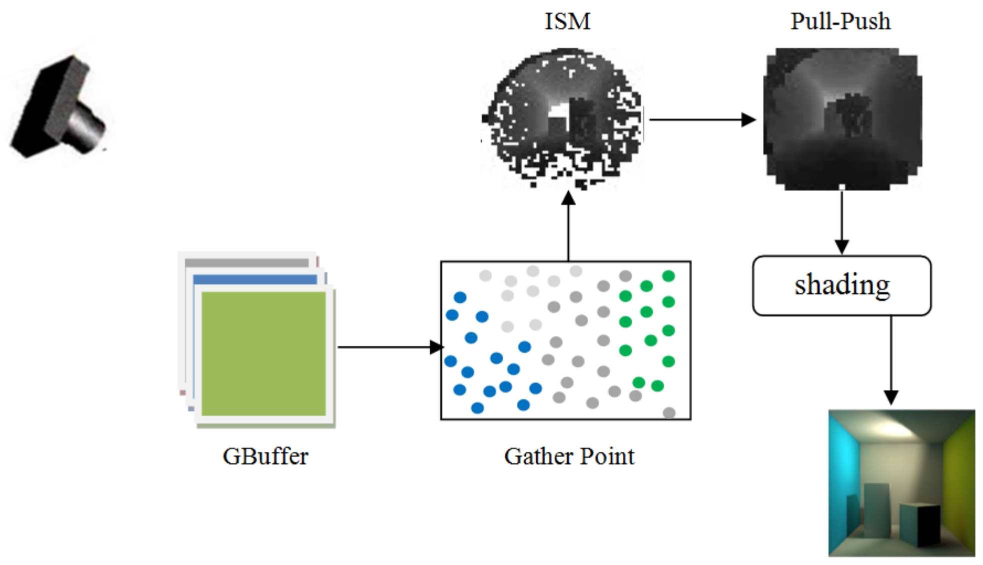


FIGURE 5.2: ISM method overview.

1. **scene preprocessing:** in this step, we randomly select some triangles and we calculate its barycentric points. These points are then used to compute the visibility because it is very easier to evaluate the visibility point to point (barycentric point to VPL) than to evaluate it triangle to point.

2. **ISM creation:** the scene is rendered from the point light source, then we select a uniform random number of VPLs. We create an ISM for each VPL that contains the depth value information of some gather points (points that are visible from the view camera). The ISM is imperfect because we can find that some VPLs are not reached by the gather points. For this reason, we apply the pull-push technique to fill in holes.

3. **shading:** after creating ISMs, we use it to compute the indirect lighting. We render the scene from the view camera. We perform the visibility test: we project the gather point in ISMs, and we test if its depth is less than the depth stored in ISM then the gather point is visible. In this case, we compute the VPL contribution to this gather point. The final contribution that defines the indirect lighting is computed by summing the contribution of all the visible points.

The disadvantage of this technique is that the scene representation is not adaptative and may be not optimal for a large scene. This limitation has been solved by Ritschel et al. [REH+11].

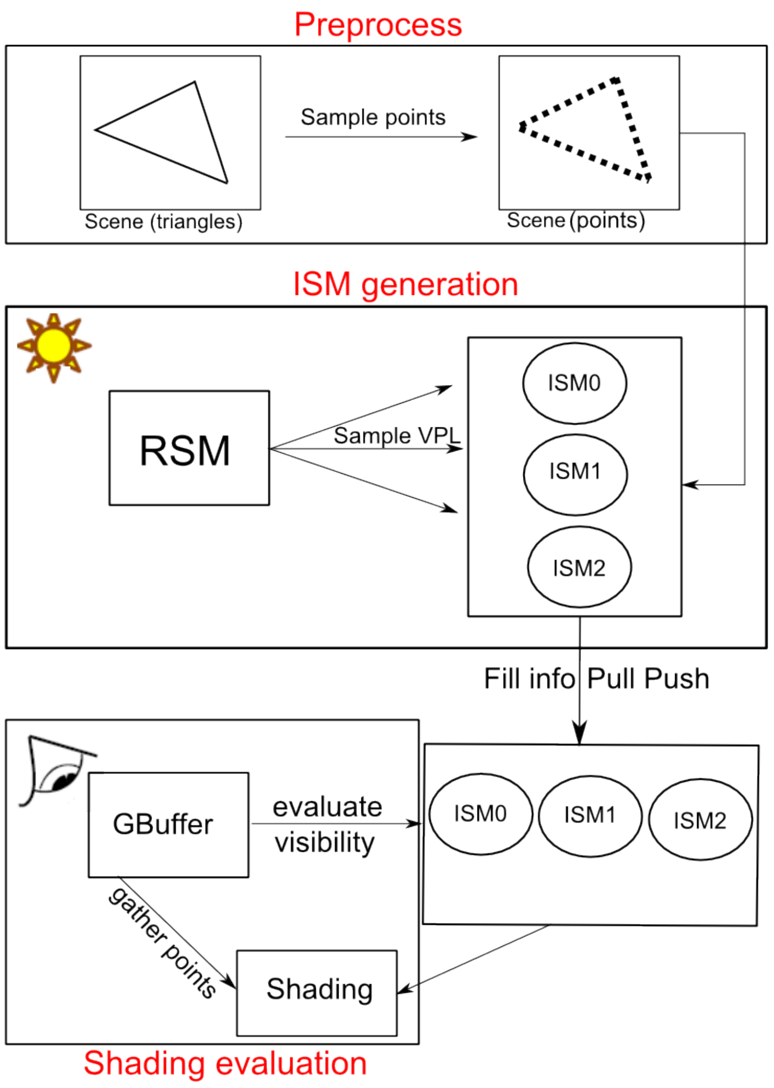


FIGURE 5.3: Defferent steps of ISM technique: preprocess, ISM generation and shading.

5.3.2 View Adaptive Imperfect Shadow Maps

Moreover, the authors propose a new method to choose the VPLs which contribute much to the final image by creating a data structure, called Bidirectional Reflective Shadow Maps, based on a Cumulative Distribution Function (CDF) (see figure 5.4). The method uses the inverse transform method which requires the computation of a CDF.
The goal of the Bidirectional Reflective Shadow Maps is to avoid the artifacts due to the VPLs creation. The main general steps of this method are:

- render scene from view camera;

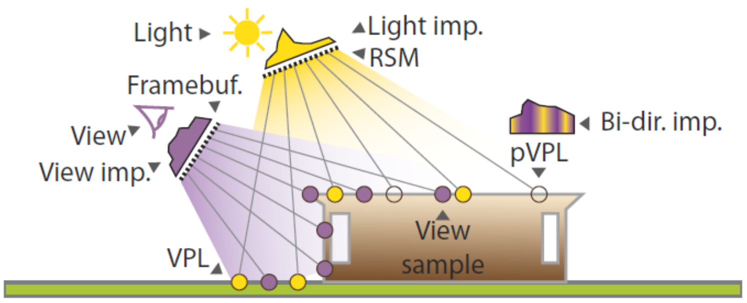


FIGURE 5.4: Bidirectional Reflective Shadow Maps [REH+11]

- render scene from light point;
- select a random gather point in the scene;
- for a potential VPL in RSM, test visibility of that VPL with all view samples;
- define non-uniform VPL sampling with Cumulative Distribution Function (CDF), to select VPLs which have a strong influence on view samples;

5.3.3 Rich VPLs

A Rich-VPL method [SHD15] handles glossy reflections with multiple primary light sources in the scene. Dammertz et al. [DKL10] have proposed a progressive method to simulate indirect lighting. It combines and exploits the advantages of three methods: Virtual Point Lights, caustics, and specular gathering, to be able to render a large variety of global illumination effects.

5.4 Clustering

Many lights methods rely on clustering to reduce the time needed to compute the contributions of the VPLs, such as the method proposed by Olsson et al. [OBS+15]. Dong et al. [DGR+09] use clustering to compute visibility. This clustering idea, in global illumination, has been wildly used [Seg07, HL15]. Hašan et al [HKWB09] have introduced a Virtual Spherical Lights (VSL) method to resolve the singularity problem due to the VPLs. The lightcut methods [WFA+05, WABG06] try to avoid the VPL flickering.

A real time based Instant Radiosity method has been proposed by Novák et al[NED11] to approximate bias compensation and avoid the artifacts of the VPLs. Nabata et al [NIDN16] have proposed a method to estimate more precisely the error due to VPL clustering.

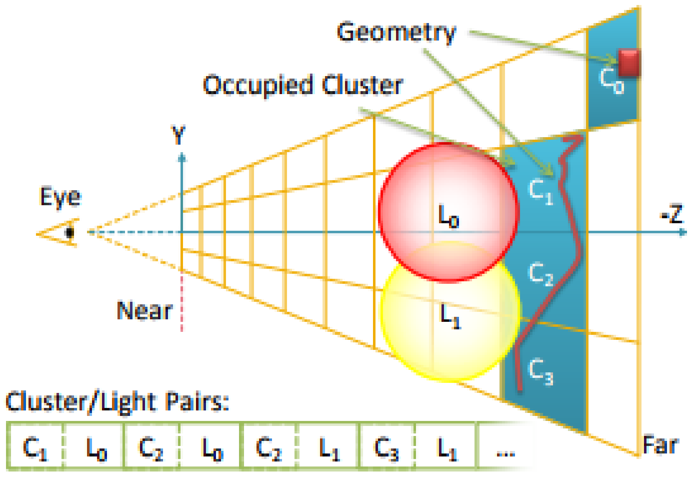


FIGURE 5.5: Illustration of the depth subdivision into clusters [OBS+15].

5.5 Conclusion

In this chapter, we have presented some VPL-based methods for the indirect illumination contribution. In reality, despite the fact that the VPL-based methods give a good approximation whether for diffuse or specular materials. But there are limited by the artifacts (flickering) due to the VPLs sampling, especially when we generate a small number of VPLs. So, to cover this problem, we have shown different methods used to clamp the VPLs and resolve the singularity problem due to the VPLs. In the next chapter, we will present our new VPL-based techniques to allow a robust and efficient way for selecting the more contributive VPLs.

Part III

Contributions and results

Chapter 6

Dual Paraboloid Reflective Shadow Maps for VPL-based rendering

6.1 Introduction

Computer graphics is concerned with the creation and management of different interactions between light and the objects of a 3D scene. Direct and indirect lighting are computed using global illumination techniques. In the past decade, these latter have known a considerable improvement, both visually and in terms of computation time.

One popular global illumination rendering approach is based on virtual point light (VPL), a technique which operates in two passes. First, a set of VPL is generated by tracing light rays from a point light source. The points (called VPL) resulting from the intersection of these light rays and the scene are stored in a data structure called RSM (Reflective Shadow Map). They act as secondary light sources. Second, during the rendering step each point visible from the camera (called gather point or visible point) gathers the contribution of a small set of VPLs (selected randomly) after evaluation of visibility between these VPLs and the gather points. Visibility evaluation is the most costly part in VPL-based techniques. To tackle visibility, several approaches exist on the GPU using some approximations. One popular approach is to evaluate visibility based on Shadow Maps [RGK+08] that contain approximate visibility information. The authors showed that this type of

approximation is good enough to compute image with perceptual differences.
Another important feature for VPL-based technique is the quality of the small VPL set selected for each gather point in the rendering step. A good VPL set needs to approximate indirect lighting for every scene's point so that the resulting rendered images are noise-free. Usually, VPLs are generated following a probability density function (PDF) that usually is proportional to their contributions to the camera pixels (gather points). However, for efficiency reason, an approximation of the VPLs contribution is used to compute the PDF.
For example, only a subset of gather points (corresponding to camera pixels) is considered for evaluating the PDF, and the visibility term is not evaluated (that means that the gather points of the subset are considered as visible to all the VPLs). In our approach, we do compute visibility using a paraboloid RSM.
The VPL-based rendering methods provide good results compared to other rendering techniques and require only a few seconds, especially for diffuse BRDF.
The main objective of this work is to provide implementation details regarding the construction, the implementation and the use of DPRSM and PRSM within the framework of VPL-based rendering.
The main contributions of this chapter are:

- implementation details on DPRSM and PRSM;
- two PRSMs (called DPRSM) are placed around each point light source to compute VPLs, and one PRSM is assigned to each VPL for visibility purpose;
- we show that DPRSM outperforms classical RSM when rendering with VPLs. To cover a 360 degree field of view a DPRSM requires two rendering passes while a classical RSM requires six.

6.2 System overview

In this section, we describe our contributions (figure 7.1) which are: DPRSM construction and Selection, our global CDF method which relies on paraboloid shadow maps associated with each VPL to speed up the visibility computation.
The scene is represented by a set of triangles. First, we render the scene from the camera viewpoint to generate the position, normal and color of all visible points (GBuffer in Figure 7.1). We consider only point light sources. We create a DPRSM

(stored in a texture) for each point light source: a texel contains the position, normal, color of the point V visible to the point light source (V represents a VPL). Then, we render the scene by sampling a subset of VPLs from the DPRSM by importance and computing their contributions to the points visible to the camera. For that, we create a single PSM at each VPL to speed up visibility needed by the rendering process (module create paraboloid VPL in figure 7.1). Recall that a CDF is computed and used by an inverse transform-based rendering method (for more details see [BGBB17]). We exploit the advantages of the PSM structure associated with each VPL, to efficiently compute the visibility between a gather point (stored in the GBuffer) and a VPL stored in the DPRSM. We detail in the following subsection (subsection 6.2.1) the construction of a DPRSM as well as the computation of the visibility term between the selected VPLs and each gather point, which is the main important step in our proposal.

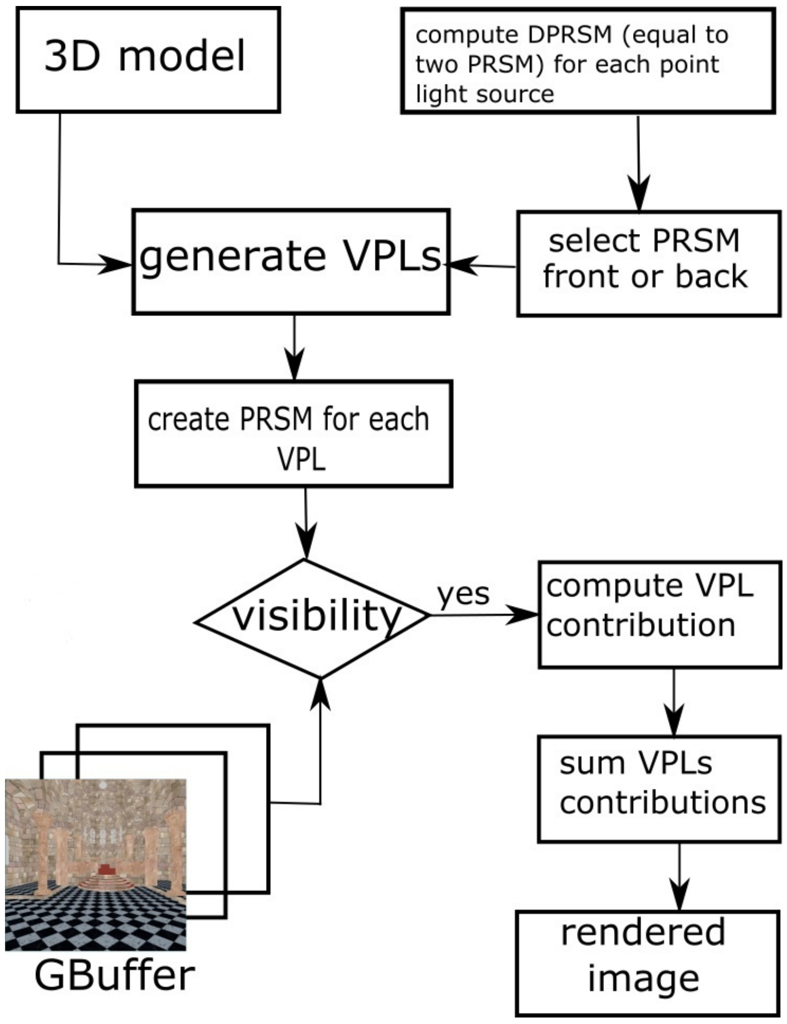


Figure 6.1: Overview of our global CDF method using the DPRSM at the point light source and a PSM at each selected VPL to evaluate visibility.

6.2.1 Dual Paraboloid Reflective Shadow Maps (DPRSM)

The scene is represented by a set of triangles. First, we construct a DPRSM at the point light source, each PRSM texel corresponds to a VPL. Then, we project the triangles of the 3D scene onto the DPRSM. As described by Gascuel et al's [GHFP08], a PRSM uses a non linear projection. This is why, each triangle is transformed into a single curved triangle, then projected onto the PRSM surface, each pixel of a PRSM contains the depth, normal and color of a point of the projected triangle.
In the following subsections, we detail the DPRSM construction, the selection of one PRSM to evaluate visibility and the computation of the visibility between a gather point and a VPL.

6.2.1.1 Construction

In this section, we show how to construct a DPRSM for a point light source. We create at the point light source position (oriented toward the z-axis of the associated $3D$ coordinates system) two paraboloids (each corresponding to a hemisphere) which are called front and back faces, each one represents a PRSM. One hemisphere is created to cover 180 degree field of view. The two hemispheres are put back-to-back to cover all the scene parts (360 degree field of view). We show how to compute the projection of a point of the 3D scene onto a PRSM.

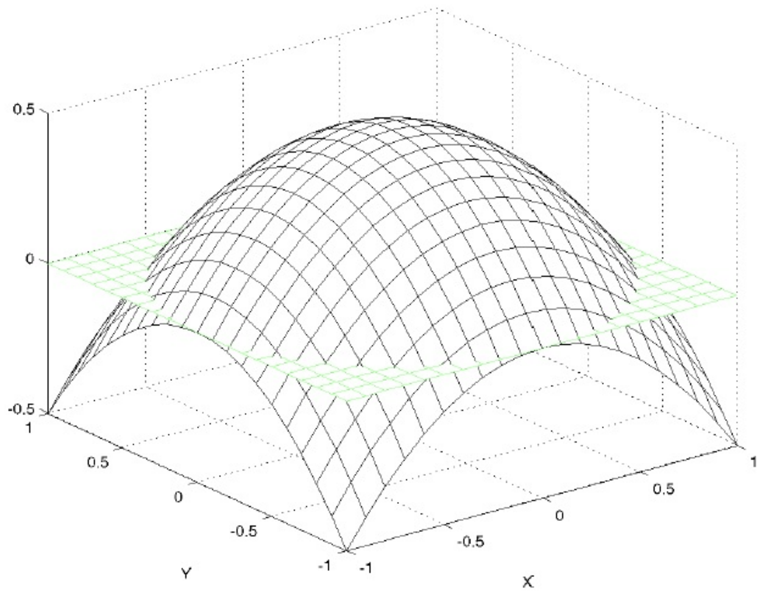


FIGURE 6.2: Dual Paraboloid Shadow Maps [BAS02]

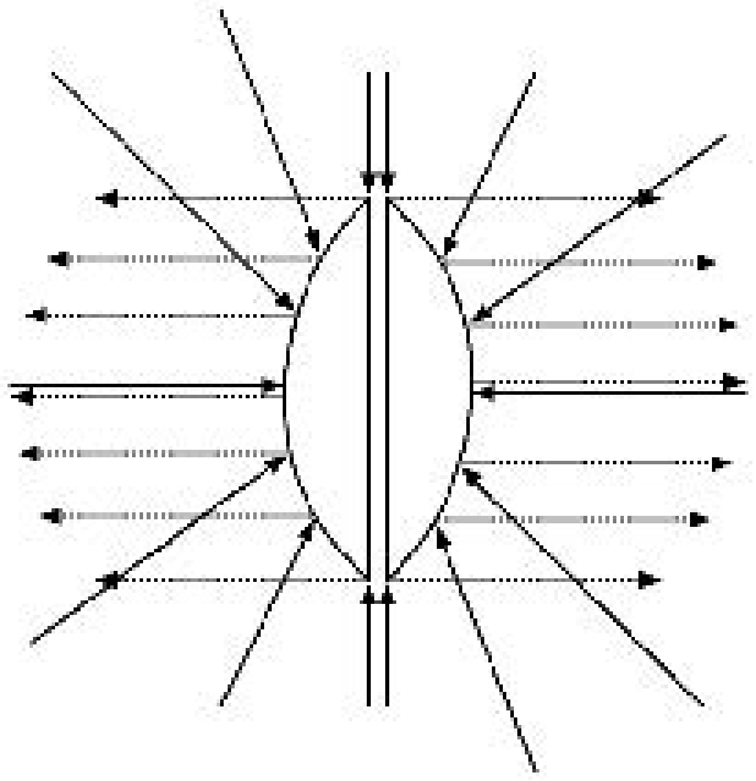


FIGURE 6.3: DPSM: using two paraboloids to capture the complete environment [BAS02]

As described by Heidrich and Seidel [HS98] the image seen by an orthographic camera facing a reflecting paraboloid

$$P = f(x,y) = \frac{1}{2} - \frac{1}{2}(x^2+y^2), x^2+y^2 \leqslant 1 \tag{6.1}$$

contains all information about the hemisphere centered at $(0,0,0)$ and oriented towards the camera $(0,0,1)$. This function is plotted in Figure 6.2. Since the paraboloid acts like a lens, all reflected rays originate from the focal point $(0,0,0)$ of the paraboloid.

In order to capture the complete environment (360 degree), two paraboloids attached back-to-back can be used, as shown in Figure 6.3. Each paraboloid captures rays from one hemisphere and reflects it to one of the two main directions d_0 and d_1 (see below).

To project a 3D point (of the scene) onto a paraboloid (3D-to-2D mapping), we have to find the point $P = (x,y,z) = P(x,y,f(x,y))$ on the paraboloid that reflects a given direction $\vec{V}$ towards the direction $d_0 = (0,0,1)$ (or $d_1 = (0,0,-1)$ for the opposite hemisphere).

The normal vector at the paraboloid surface is calculated by the cross product of the tangents for the x and y coordinates, computed as partial derivatives V_x and V_y:

$$V_x = \frac{\partial P}{\partial x} = (1, 0, -x) \tag{6.2}$$

$$V_y = \frac{\partial P}{\partial y} = (1, 0, -y) \tag{6.3}$$

We compute the normal vector at point P on the paraboloid as:

$$\vec{N_p} = V_x \times V_y \tag{6.4}$$

$$\vec{N_p} = (1, 0, -x) \times (0, 1, -y) = (x, y, 1) \tag{6.5}$$

Since the paraboloid is perfectly reflecting we simply calculate the halfway vector $\vec{H}$ which is equal to $\vec{N_P}$ up to some scaling factor. Using $\vec{H}$ and Equation 6.5 we can now formulate the 2D mapping of $\vec{V}$:

$$\vec{d_0} + \vec{V} = \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = k.\vec{N_P} = k.\begin{pmatrix} X/Z \\ Y/Z \\ 1 \end{pmatrix} = k.\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{6.6}$$

To sum up, given a 3D point of the scene, its projection onto the paraboloid is $P = (x, y, f(x, y))$, where x and y are computed using equation 6.6.

We describe in algorithm 2 and 3 the pseudocodes for the vertex and fragment programs that generate a DPRSM on the GPU.

Algorithm 2 ***vector3*** *vertexGenerateDPRSM(scene s, vector3 p, vector3 n, vector3 c)*

```
1: // Pseudocode of the Vertex Shader
2: matrix4 MVP = M_light. M_model. p; // matrix for transforming a point of the scene to the dual paraboloid coordinate system
3: for each point (vertex) in the scene do
4:     p = getValue(s, position); // retrieve the position of each point in the 3D scene
5:     n = normalize(getValue(s, normal)); // retrieve the normal of each point in the 3D scene
6:     c = getValue(s, color); // retrieve the color of each point in the 3D scene
7:     newPos = MVP * p; //transform the position of each point p to the dual paraboloid coordinate system by using a model view projection (MVP) matrix
8: end for
9: return newPos;
```

Algorithm 3 *GenerateDPRSM(scene s, float depth, texture2D positionDPRSM, texture2D normalDPRSM, texture2D colorDPRSM)*

```
1: // Pseudocode of the Fragment Shader
2: // Output: 3 textures representing the DPRSM: positionDPRSM (position of VPL), normalDPRSM (normal of VPL), colorDPRSM (color of VPL)
3: p = GBuffer (position);
4: n = GBuffer (normal);
5: c = GBuffer (color);
6: vertex = vertexGenerateDPRSM(s, p, n, c);
7: // coords.x and coords.y are the x and y coordinates of the projected point in the DPRSM coordinate system, computed using equation 6.6
8: for each vertex in the scene do
9:     depth = position.z; // the depth of each point in the scene is equal to the third coordinate z
10:    invDepth = 1 - position.z;
11:    if depth >= 0.0 then
12:        //compute coordinates in front praboloid
13:        coords.x = (position.x / depth) * 0.5 + 0.5;
14:        coords.y = (position.y / depth) * 0.5 + 0.5;
15:    else
16:        //compute coordinates in back praboloid
17:        coords.x = (position.x / invDepth) * 0.5 + 0.5;
18:        coords.y = (position.y / invDepth) * 0.5 + 0.5;
19:    end if
20:    // calculate the partial derivative for x and y (see equation 6.2 and 6.3
21:    dx = partialDerivative(depth);
22:    dy = partialDerivative(depth);
23:    normal = vectProduct(dx, dy);
24:    positionDPRSM = (positionTex, coord.x, coord.y); //get the position from DPRSM (positionTex) with the new texture coordinates (coords.x and coords.y)
25:    normalDPRSM = (normalTex, coord.x, coord.y); //get the normal from DPRSM (normalTex) with the new texture coordinates (coords.x and coords.y)
26:    colorDPRSM = (colorTex, coord.x, coord.y); //get the color from DPRSM (colorTex) with the new texture coordinates (coords.x and coords.y)
27: end for
```

The main difficulty of the DPRSM generation method lies in how to convert a simple triangle to a curved one. Because when we create a DPRSM, we perform a non-linear projection (see equation 6.1), unlike for classical RSM. The paraboloid center represents the light source position and its direction indicates which paraboloid will be used. To generate a DPRSM, we have inspired by the method proposed by Gascual et al's [GHFP08]. To compute the contribution of a VPL (stored in the DPRSM) we store at each texel of the DPRSM, the normal and color of the projected points rather than the depth value only [BGBB17]. Moreover, we compute visibility by creating a PSM at each selected VPL.

6.2.1.2 DPRSM selection

Once the DPRSM has been created, for each visible point (from the view camera) a PRSM (front or back face) is selected. Then a VPL is randomly sampled from the selected face using a CDF. Next, the contribution of the sampled VPL is computed. We repeat this process (iteration) until a subset of VPLs have been sampled. The contributions of all the sampled VPLs are summed to give the indirect radiance of the visible point. We could uniformly select one of the two faces of the DPRSM as shown in algorithm 4.

Algorithm 4 ***face*** *selectUniformParaboloid(FACE face, random psi)*

```
1: if psi < 0.5 then
2:     return front_face;
3: else
4:     return back_face;
5: end if
```

Figure 6.4 depicts the uniform selection at the DPRSM. Two paraboloids, front and back, are created at the point light source. According to the value of the uniform random variable ψ, a number of VPLs are selected from the front texture (red circles in figure 6.4) or from the back texture (green circles in figure 6.4).

In fact, this uniform selection (of a face of the DPRSM) is not efficient, because the importance of the two paraboloids is not always the same and depends on the scene and the position of the light source. We propose to resort to a Russian roulette [AK90]. We compute the probability bF (respectively bB) of choosing a

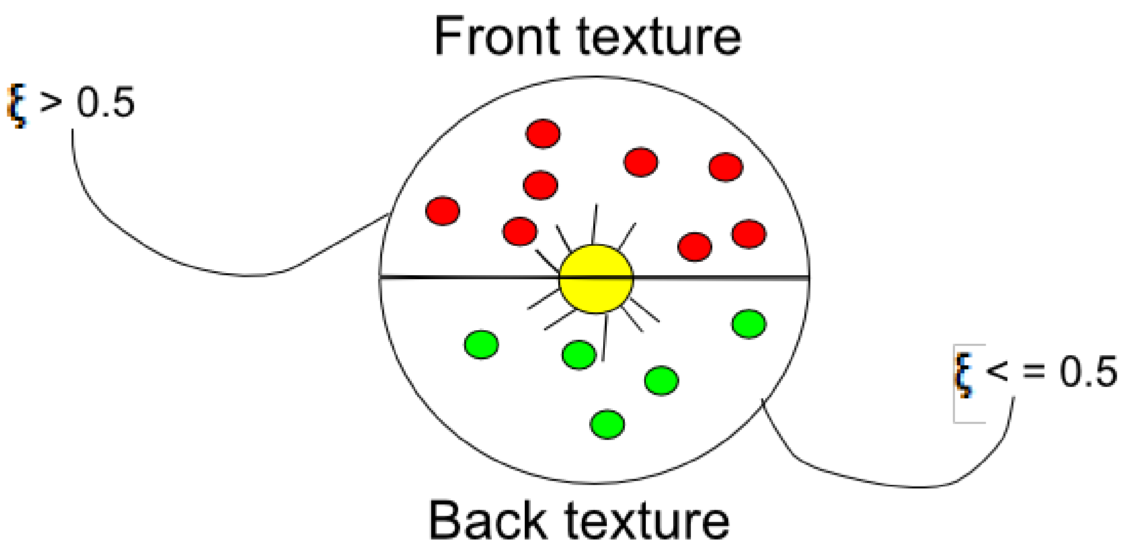


FIGURE 6.4: uniform selection in DPRSM.

front PRSM (respectively a back PRSM) of the DPRSM by computing the sum of the average contributions vF and vB of all the VPLs stored in a PRSM (front or back) [BGBB17]. The PRSM selection probabilities are the normalized average contributions of all the VPLs: bF = vF/(vF+vB) and bB = vB/(vF+vB). Our selection paraboloid method is given by algorithm 5:

Algorithm 5 *face selectParaboloid(FACE face, random psi)*

```
1: vF = sumContrib(front); // average contribution sum for front face
2: vB = sumContrib(back); //average contribution sum for back face
3: bF = vF/(vF+vB);
4: bB = vB/(vF+vB);
5: if psi < bF then
6:     return front_face;
7: else
8:     return back_face;
9: end if
```

6.2.1.3 Visibility Computation

After generating a DPRSM at a point light source and placing a camera into the 3D scene, we create a PSM (at each selected VPL) containing the depth value of the scene's points visible from the VPL. Each VPL is considered as a secondary light source. The VPLs are select using the method presented in [BGBB17] (see subsection 6.2.2).

To compute the indirect lighting, we determine the visibility of each gather point (point visible from the view camera) from the selected VPL. For that, we project each visible gather point GP onto the PSM assigned to the VPL and we test if GP is visible or not from the VPL (line 19 in algorithm 6). If that is the case, we compute the contribution of the selected VPL to GP. Algorithm 6 describes the rendering process as well as the visibility computation from a selected VPL. Note that a PSM (Paraboloid Shadow Map) is constructed in a similar way to that of a PRSM, the difference lies in the data stored: only a depth (or z coordinate) for PSM, while depth, normal, color, position for a PRSM.

Algorithm 6 *2D_texture_image* VPL_Rendering(scene s, float depth)

```
1: gBuffer = GenerateGBuffer(); // a buffer containing the position, normal,
   color of the visible points from the camera view
2: dprsm = GenerateDPRSM(s, depth); // front and back buffers containing
   position, normal, color of the visible points from the light source
3: for each gather point $(i, k)$ from the gBuffer do
4:   for j = 1 to #NBVPL do
5:     // NBVPL: small number of VPLs
6:     psi = sample_uniform_random_variable(); // psi ranging from 0 to 1
7:     if selectParaboloid(face, psi) == front_face then
8:       //see algorithm 5
9:       VPL = sampleFromFront(); // select the VPL of coordinates from the
         front face
10:    else
11:      if selectParaboloid(face, psi) == back_face then
12:        //see algorithm 5
13:        VPL = sampleFromBack(); // select the VPL coordinates from the
           back face
14:      end if
15:    end if
16:    prsm = GeneratePRSMatVPL(); // front buffer at the selected VPL con-
       taining position, normal, color of the visible points from the selected VPL

17:    p = projectToPRSM(GP, prsm); // project each gather point to the PRSM
       of the VPL
18:    // function visible allows to test if the projected gather point p is visible
       from the selected VPL or not
19:    if visible(p, prsm) == true then
20:      //point p is visible from the PRSM of the selected VPL
21:      image[i][k] += computeContribution(VPL) / bF //bF: probability of
         selecting the front face
22:    end if
23:  end for
24: end for
25: return image; // rendered image
```

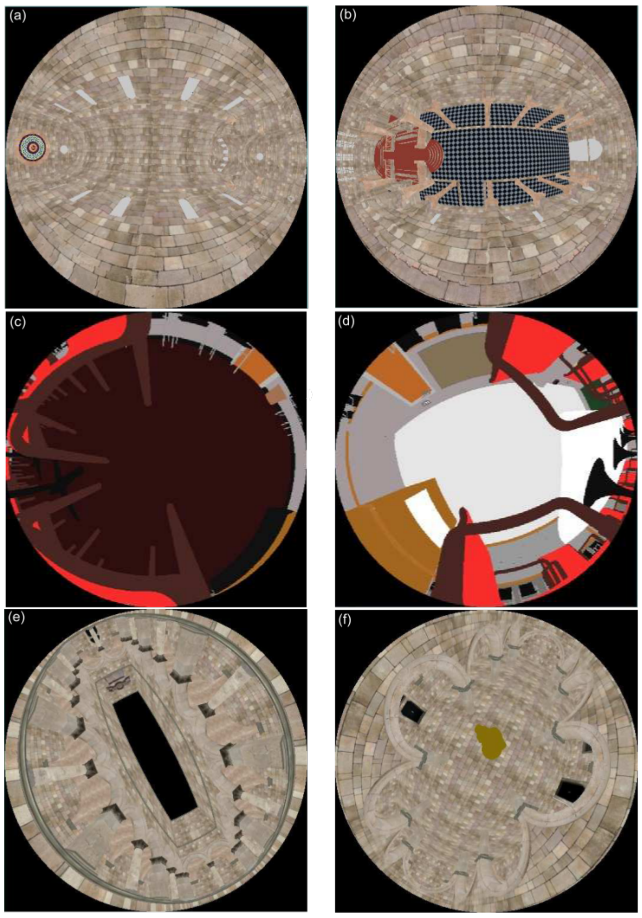


FIGURE 6.5: Figure (a), (c) and (e) show the color buffer of the front face corresponds to all the triangles of the Sibenik scene, the Conference scene and the Sponza scene respectively. Figure (b), (d), and (f) correspond to the back face color buffer for the four scenes respectively.

6.2.2 VPL-based indirect lighting method

In this section, we summarize the global CDF method (for more details see [BGBB17]) that is used to determine the VPLs which contribute more to the final image.

1. As input, we have a 3D scene. To render it, we place a camera and a point light source;
2. we associate a DPRSM with the point light source;
3. we compute the average contribution of all the VPLs to a small subset of gather points selected randomly (a gather point is a visible point from the view camera);

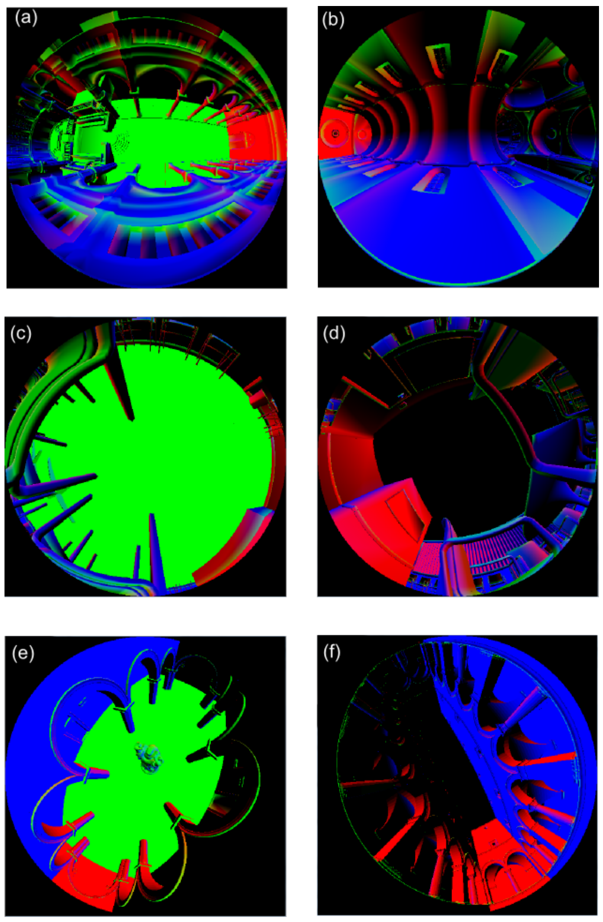


FIGURE 6.6: Figure (a), (c) and (e) show the normal buffer of the front face corresponds to all the triangles of the Sibenik scene, the Conference scene and the Sponza scene respectively. Figure (b), (d), and (f) correspond to the back face normal buffer for the four scenes respectively.

4. we store these average contributions in a linear array (also stored in 1D texture) that is used to compute discrete probabilities assigned to the VPLs;

5. from these discrete probabilities, we compute discrete CDF values stored in a linear array;

6. to determine the more contributive VPLs (mcVPL), we use the inverse transform method in the discrete domain. In other words, we generate a uniform random variable (ranging from 0 to 1), then we perform a binary search in the CDF array, to determine the CDF index k that represents the column and the row. This pair (row, column) represents the mcVPL coordinates.

As mentioned in figure 6.8, our goal is to compute the indirect lighting L_r in direction ω_r and at a point x. Equation 6.7 represents the rendering equation

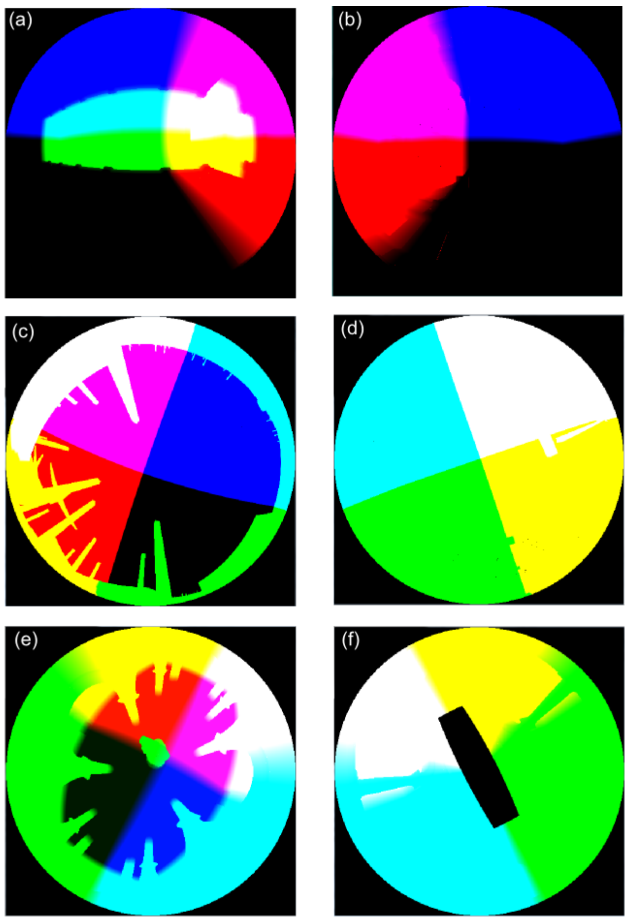


FIGURE 6.7: Figure (a), (c) and (e) show the position buffer of the front face corresponds to all the triangles of the Sibenik scene, the Conference scene and the Sponza scene respectively. Figure (b), (d), and (f) correspond to the back face position buffer for the four scenes respectively.

proposed by Kajiya [Kaj86]:

$$L_r(x, \omega_r) = \int_{\Omega} f_r(x, \omega_i, \omega_r) L_i(x, \omega_i) cos\theta_i d\omega_i \tag{6.7}$$

Where $f_r(x, \omega_i, \omega_r)$ is the BRDF of the point x, ω_i in the incident direction, and $cos\theta_i$ is the angle between the normal at the point x and the incoming direction ω_i.

We approximate this equation using Monte Carlo approach as:

$$L_r(x, \omega_r) = \frac{1}{N} \sum_{i=1}^{N} \frac{f_r(x, \omega_i, \omega_r) L_i(x, \omega_i) cos\theta_i}{p(\omega_i)} \tag{6.8}$$

Where N corresponds to the number of samples (in our method the samples are the VPLs) and p is the probability density function.

We propose to use the VPLs for approximating the indirect lighting. Our goal is to select the VPLs that contribute more to the final image. So we need to find the VPL as well as its PDF.

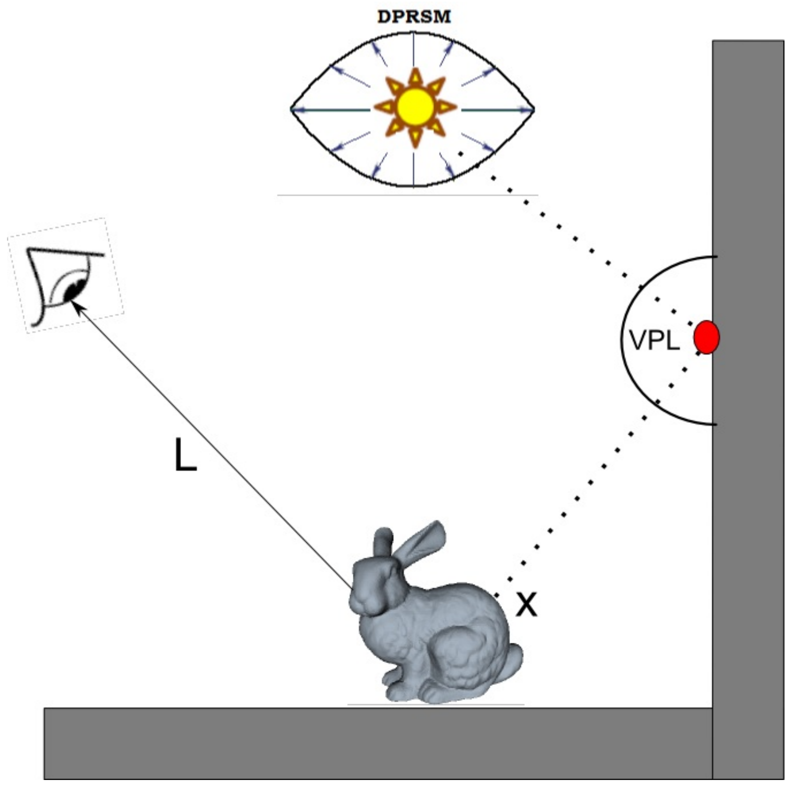


FIGURE 6.8: Scene representation example.

To speed up the rendering process, we propose to generate the mcVPL by the global CDF method with a DPRSM placed at the point light source, but unlike in [BGBB17], we associate a PSM at each VPL. At the rendering step, we determine the visibility term (gather point - mcVPL) by projecting each gather point onto the shadow map of the selected VPL, without using voxelization. If the gather point is visible then we compute the contribution of such a VPL. We assign a PRSM to each VPL to speed up visibility from each VPL. We also assign a classical RSM to each VPL and we show that parabolic RSM is more efficient than classical RSM in terms of image quality and resolution.

6.3 Results and discussion

The results shown in this section have been obtained with a computer equipped with Intel i7 930 @ 2.80 GHZ, 8GO RAM running 64 bits. For the GPU, we have used NVIDIA Geforce GTX 780 OC 6GB. We used three test scenes: Sibenik, Crytek Sponza and Conference room (see table 6.1). The three scenes, used in this chapter, are available in [McG11], they contain only diffuse objects.
We present some results obtained with our global CDF-based rendering method to validate the quality and the efficiency of using a DPRSM at each point light source and a PSM at each VPL. To speed up the rendering process, we used only one bounce indirect lighting. We compare our images, generated by our method, to the reference image calculated with the method described in [REH$^+$11].
Table 6.1 shows the color buffer corresponding to all the triangles of the Sibenik, Crytek sponza and Conference room scenes. The scenes contain thousands of triangles. All the scenes' objects have diffuse BRDF because our proposed global CDF-based rendering method runs only for diffuse objects.

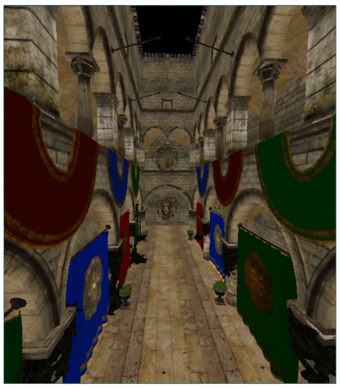
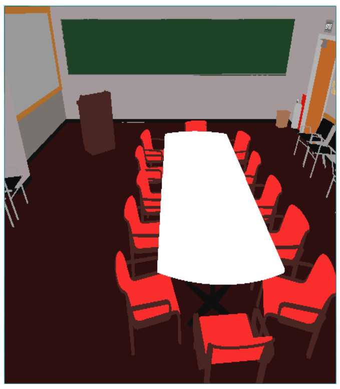
Sibenik 75,284 triangles	Crytek Sponza 262,267 triangles	Conference 331,179 triangles
		

Table 6.1: The number of triangles and the color buffer of the GBuffer which contains the diffuse color corresponding to all the triangles of the Sibenik, Crytek Sponza with bunny object and Conference room respectively.

Figure 6.9 shows a comparison between our global CDF method and the reference one proposed by Ritschel et al's [REH$^+$11]. To generate a reference image, for all the scenes, we used a large number of gather points ($1k$ gather points) for computing the average contribution of all the VPLs (used to compute a PDF then a CDF) that will be used to compute the most contributive VPL with all its parameters (its position, normal, color and PDF). Furthermore, for the reference images we used a large number of VPLs (10 k VPLs). To compute the visibility term of a reference image, we applied the voxelization method proposed by Crassin et al's [CG12], with a grid resolution of 128^3. On the other hand, our images generated with our global CDF-based rendering method have been obtained with only 800 VPLs rather than 10 k. We created a Paraboloid Shadow Map with a resolution of 512×512 at each selected VPL to compute visibility. As we mentioned in figure 6.9, each image obtained with our method look similar to the corresponding reference image. But our images look more illuminated than the reference ones because our method makes uses a DPRSM which covers a $360 fieldofviewwhilein$[REH+11]$aRSMcoversonlyamaximumof 90$.

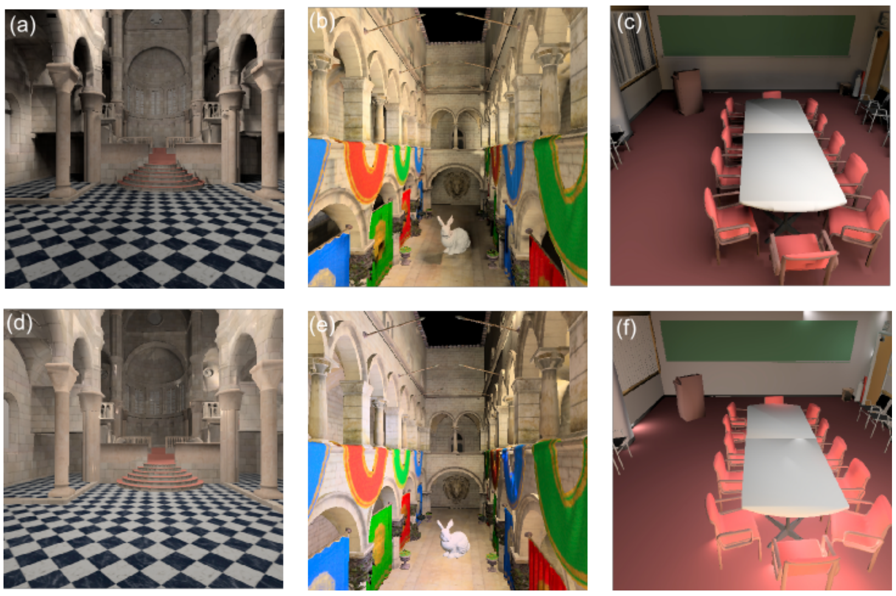


FIGURE 6.9: The Sibenik, Crytek Sponza, and Conference room scenes have been rendered using 800 randomly selected VPLs per gather point. Image (a), (b), and (c) are the reference images for the three scenes generated with the global CDF-based method [REH+11], the contribution of each gather point is computed using a large number of VPLs (10k VPLs). Images (d), (e), and (f) have been computed using our global CDF method.

After comparing our images with the reference ones visually, we validate our approach by computing the mathematical RMSE metric. Table 6.2 gives the RMSE value between our images generated by our global CDF method and the reference images computed using the method proposed by Ritschel et al's [REH+11]. The obtained results of table 6.2 prove that our images are very similar to the reference with a small error of 0.0163 for the Sibenik scene, 0.0319 for the Crytek Sponza scene and 0.011 for the Conference room scene.

Scene	RMSE
Sibenik	0.0163
Crytek Sponza	0.0319
Conference	0.011

TABLE 6.2: RMSE error between our images and the reference for the Sibenik, Crytek Sponza and Conference room scenes.

Scene	Sibenik	Crytek Sponza	Conference
RSM	0.011	0.02	0.018
Cube maps	0.26	0.28	0.21
DPRSM	0.15	0.16	0.11

TABLE 6.3: Rendering time comparison for Sibenik, Crytek Sponza and Conference room with a 512×512 resolution when using a classical RSM, a cube map and a DPRSM.

Table 6.3 shows the rendering time for the Sibenik, Crytek Sponza and Conference room scenes when using a classical RSM (with a 70-degree field of view), a DPRSM, and a cube map at the point light source. The resolution of all the scenes is 512×512. We observe that the cube maps projection is the slowest technique because it generates the results with 6 rendering passes. Furthermore, the classical RSM is faster than the DPRSM but it is not efficient because the view space is limited by the field of view (70-degree). On the other hand, the DPRSM projection allows us to cover all the $3D$ space.

In figure 6.10 we show the rendering time measured in milliseconds as a function of the number of gather points for both our global CDF method and the reference method [REH+11]. Figure 6.10 demonstrates that the rendering time increases with the number of the gather points. Moreover, we observe that for different numbers of gather points (2k, 5k, 10k, 20k, 30k), the rendering time of our global CDF method is lower than the one of the reference method. To obtain good quality images, it is necessary to randomly select a large number of gather points. In addition, we show that when we choose a small number of gather points (for 2k and 5k gather points), we obtain approximately the same rendering time.

Table 6.4 summarizes the rendering time of our global CDF-based method (with

Scene	Resolution	Time (ms)
Sibenik	512×512	0.15
	1024×1024	0.63
	2048×2048	1.79
Crytek Sponza	512×512	0.16
	1024×1024	0.5
	2048×2048	1.85
Conference room	512×512	0.11
	1024×1024	0.46
	2048×2048	1.69

Table 6.4: Rendering time with different scene resolutions for the Sibenik, Crytek Sponza and Conference room scenes.

DPRSM and PSM) when only 800 VPLs for each gather point. We changed the resolution of the paraboloid shadow map associated with each selected VPL, then we computed the rendering time. We observe that the paraboloid shadow map resolution influences to the rendering time. The higher the resolution, the higher the image quality, but the performance decreases and the rendering time gets higher.

6.4 Conclusion

We present in this chapter, the goal of using a DPRSM at the point light source during the rendering pass. we have provided implementation details of DPRSM and PSM used in VPL-based rendering methods as an approximation of the solution to the global illumination problem. We have presented a VPL-based rendering method that allows improving the selection of the most contributive VPLs stored in a Dual Paraboloid RSM (that we called DPRSM). To speed up the rendering time, we have created a paraboloid shadow map at each selected VPL, and we have projected each gather point to this paraboloid shadow map to evaluate visibility from a VPL. We have shown that our method is more efficient in terms of rendering time than when using voxelization when computing visibility from a VPL. But it consumes much memory space. This projection type allows us to provide a large field of view to cover the total space. It generates results in two rendering passes. We created two paraboloid front and back at the point light source according to its z-axis orientation, we transformed each triangle to a single curved triangle and we projected it in the paraboloid space. To select one paraboloid between the front or back faces, we have proposed to use the Russian roulette algorithm. In the next chapter, we will show how we use the DPRSM structure to select the VPLs which contribute more to the final image.

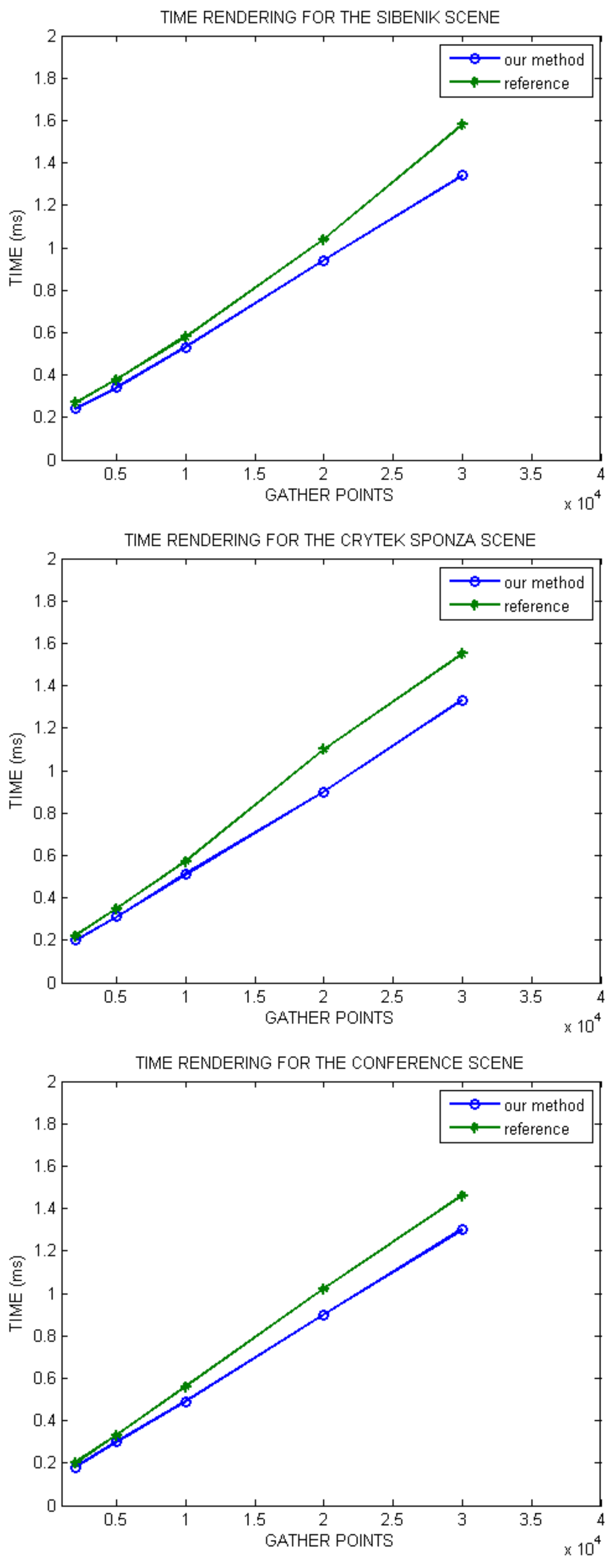


FIGURE 6.10: Plots representing the rendering time in milliseconds of our global CDF method and the reference method proposed by Ritschel et al's [REH+11] for the Sibenik, Crytek Sponza and the Conference room scenes respectively.

Chapter 7

Efficient Inverse Transform methods for VPL Selection in Global Illumination

7.1 Introduction

One popular approach used to approximate the indirect lighting is called virtual point light (VPL) [Kel97] technique which is a two-pass approach. First, a set of VPL is generated by tracing some light path from the light source and store their vertex as a secondary light source. Second, each visible point from the camera gathers the VPL contribution by evaluating the visibility.
Visibility evaluation is the most costly part in VPL techniques. Several approaches exist on the GPU using some scenes approximation. Imperfect Shadow Maps approach [RGK+08] evaluate it using Shadow Maps that contains approximate visibility information. The authors showed that this type of approximation is good enough to compute image with perceptual differences.
Another Shadow Maps type called Dual Paraboloid Shadow Maps (DPSM) first introduced by Heidrich and Seidel [HS98] is used to approximate the visibility term. The main advantage of this projection type consists in giving a large field of view to cover all the scene parts.
The quality of the VPL set is very important. Thus, a good VPL set needs to approximate the light transport for every pixel to produce low noise images. Usually, VPLs are generated following a probability density function (PDF) that usually

is proportional to their contributions to the camera pixels. However, for efficiency reasons, the approximation of the VPL contribution is used to compute the PDF. For example, only a subset of camera pixel is considered or visibility term is not evaluated. In our approach, we show how to compute this information using the advantages of a DPRSM to approximate the visibility information.
Our VPL-based method gives good results compared to other rendering techniques in only a few seconds, especially for diffuse BRDF.
In this chapter, we summarize our contributions (figure 7.1): the CDF computation, use of MIS combining inverse transform-based rendering and a gathering-based rendering.

7.2 System Overview

First, to reduce the cost of the ray-object intersection, the scene is spatially subdivided into voxels according to the method proposed in [CG12]. Visibility computation is sped up using this voxelization. The scene is rendered from the camera viewpoint. We create a GBuffer that contains the position, normal and color of all the visible points (so-called gather points). Then, we build a Dual Paraboloid RSM (DPRSM) at each point light source. We propose two methods for computing the CDF: local CDF, and gathering-based global CDF. Recall that a CDF is used by an inverse transform method to randomly select a subset of VPLs (stored in the DPRSM) to compute the radiance of a gather point as the sum of the contributions of the selected VPLs. To improve the resulting rendered images we use an MIS approach combining an inverse transform method (based on local CDF or gathering-based global CDF) and a gathering-based rendering method.
Below, we summarize the state of the art method for computing a global CDF [REH+11], while the main parts of our method (figure 7.1) are detailed in the following sections.

7.2.1 Gathering-based method

In the following subsection, we will see the based gathering approach in the scene voxelization.
In the first time from the camera view, we generate a GBuffer which contains all the visible points (gather points). Then we create a dual paraboloid RSM (front

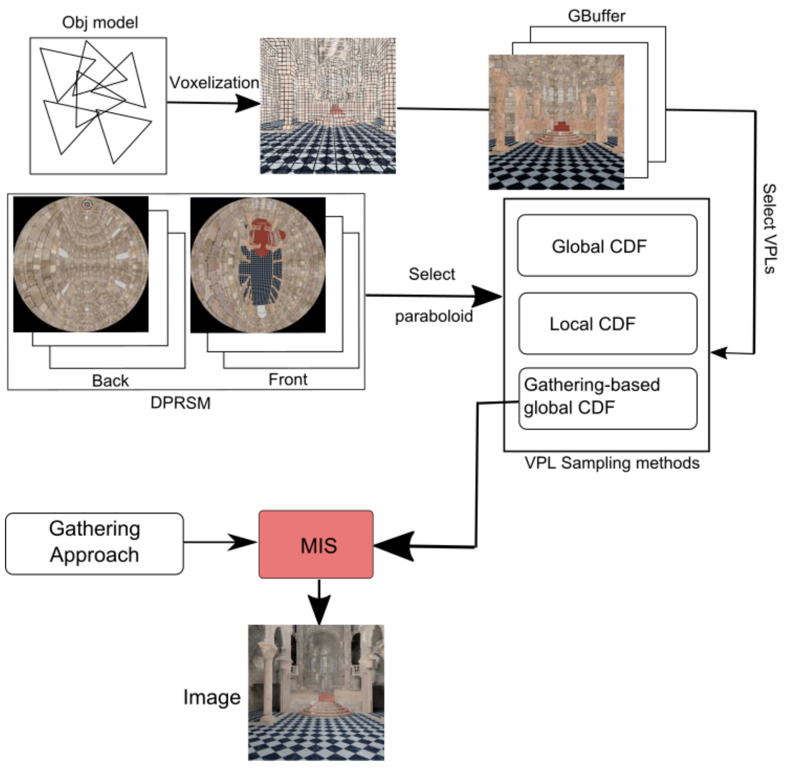


FIGURE 7.1: Overview of our inverse transform methods and MIS for computing the indirect illumination at each gather point using different types of CDF.

and back) at the light source to store the VPLs. our goal is to compute the VPLs contributions, for that, we shoot n rays from each gather point and for each direction, we search for the first intersection point using the Ray Marching algorithm. We project the outgoing point into the two paraboloid RSM if the point is visible in one of them we calculate its contribution to the gather point. The detail of the method in the algorithm below (algorithm 1):

Algorithm 7 gathering()

```
1: gBuffer = GenerateGBuffer(position, normal, color); // position, normal, color
   of the visible point
2: rsm = GenerateDualParaboloidRSM(positionRSM, normalRSM, colorRSM);
   // front and back hemisphere at point light source
3: for  i = 1 to GP do
4:    // GP: number of gather point
5:    for  i = 1 to NBDIR do
6:       // NBDIR: number of random directions in the hemisphere
7:       dir = randomDirection(GP);
8:       its = rayMarching(GP, dir);
9:       if its != -1 then
10:         // intersection found
11:         outgoing = computeOutgoing();
12:         uvVPL = projectPraboloid(outgoing, rsm);
13:         // project the outgoing point into the RSM and return its UV coordi-
            nates
14:         if visible(GP, vpl) then
15:            contrib = computeContribution();
16:         end if
17:      end if
18:   end for
19: end for
```

The main disadvantage of this algorithm is that we should compute the contribution of all the VPLs which are visible by the gather points (there is no Importance Sampling selection), and in this case, the Monte Carlo estimator will not be efficient.

7.2.2 Computing a global CDF

The global CDF method refers to the method of [REH+11] which consists in computing a CDF from just a small number of a selected gather points, say points visible to the viewpoint through pixels (Figure 7.2). To compute a CDF, the method computes the average contributions (V_i) of all the VPLs to a small subset of gather points selected randomly. Then these average contributions are stored in a linear array (also stored in 1D texture) and then used to compute discrete probabilities (A_i) assigned to the VPLs. From these discrete probabilities, discrete CDF values (B_i) are computed and stored in a linear array.

7.2.3 Determining the most contributive VPLs

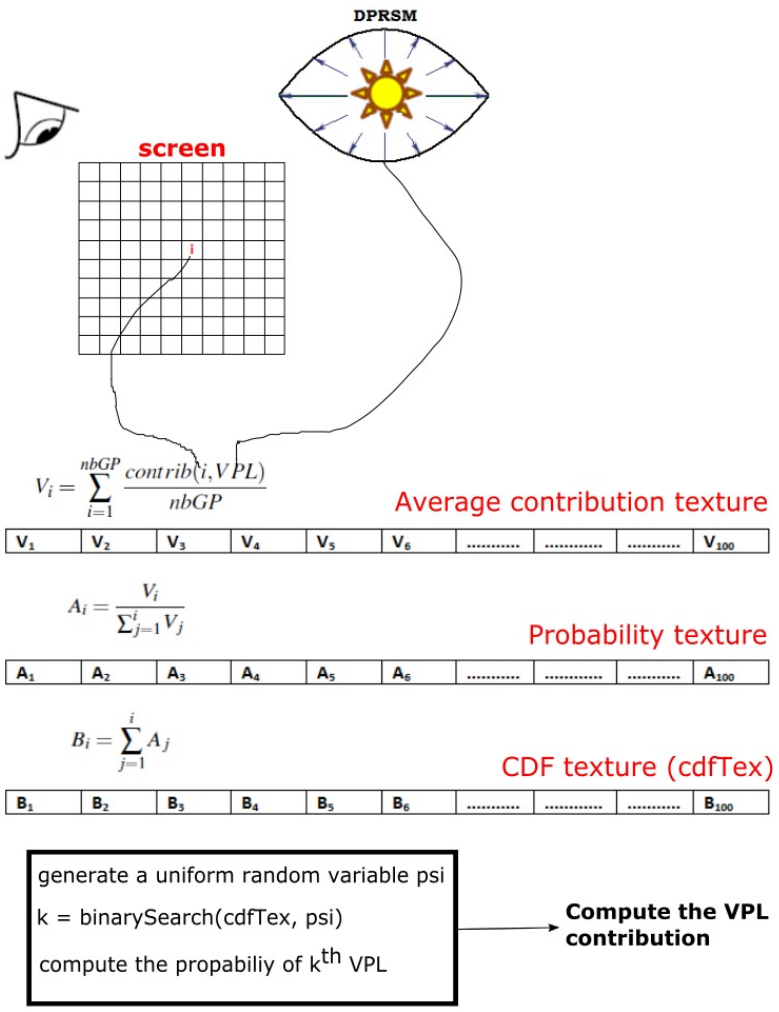
To determine the more contributive VPL (mcVPL), the inverse transform method is used in the discrete domain. In other words, a uniform random variable (ranging from 0 to 1) is generated, then a binary search is performed in the CDF array, to determine the CDF index k that represents the column $c = [\frac{k}{N}]$ and the row $r = modulo(k, c)$. The (r, c) pair represents the mcVPL coordinates in a PRSM. In the rendering step, the contribution of each mcVPL is divided by its probability.

7.3 VPL Sampling methods

In this section we propose two methods for computing a CDF used in an inverse transform approach to sample VPLs from a DPRSM. From now on, they will be called *local* and *gathering-based*. Note that the objective is to sample the most contributive VPLs based on an importance function which is the CDF. Before detailing these two methods we show in the following subsection how to compute the radiance $L(x)$ at a point x due to a certain number of randomly selected VPLs.

7.3.1 Computing the contribution of a VPL to a gather point

Let us see now how to compute the radiance $L(x)$ at a point x resulting from the contributions of a certain number of randomly selected VPLs. Let us assume that a VPL is a very small surface with normal N and flux ϕ_v (emittance in this case, say flux emitted per unit surface). We can consider this VPL as a point light source which has an emittance ϕ_v and an intensity I_v (flux emitted per unit solid



DPRSM
screen
i
$V_i = \sum_{i=1}^{nbGP} \frac{contrib(i,VPL)}{nbGP}$
Average contribution texture
V1 V2 V3 V4 V5 V6 .... V100
$A_i = \frac{V_i}{\sum_{j=1}^{i} V_j}$
Probability texture
A1 A2 A3 A4 A5 A6 .... A100
$B_i = \sum_{j=1}^{i} A_j$
CDF texture (cdfTex)
B1 B2 B3 B4 B5 B6 .... B100
generate a uniform random variable psi
k = binarySearch(cdfTex, psi)
compute the propabiliy of kth VPL
Compute the VPL contribution

FIGURE 7.2: Overview of global CDF method: illustration of the computation of a discrete CDF to generate mcVPLs.

angle). The VPL intensity I_v is expressed as [DS05]:

$$I_v = \phi_v \frac{cos\theta_1}{\pi} \tag{7.1}$$

where θ_1 is the angle between the normal N at the VPL v and the lighting direction from the VPL. All the used notations are given in figure 7.3.
For a diffuse surface (here a VPL v) there is a relation between its emittance ϕ_v and its radiance L' [Dut03]:

$$\phi_v = L^{'}\pi \tag{7.2}$$

The radiance $L^{'}$ of the VPL v due to a point light source is given by:

$$L^{'}(v) = I_s \frac{cos\alpha}{d_1^2} f_r^d(v) \tag{7.3}$$

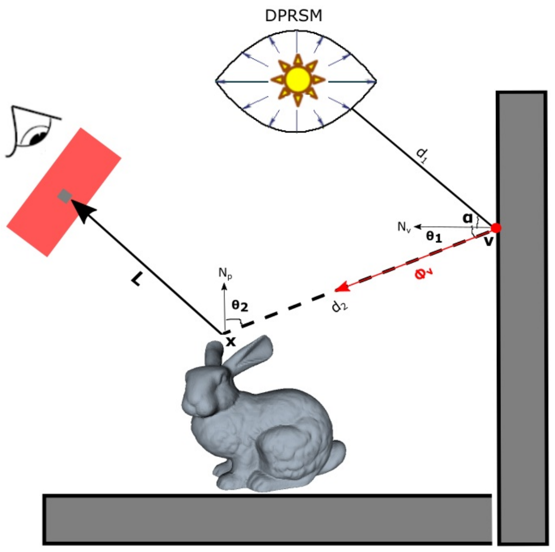


Figure 7.3: Computation of indirect lighting due to VPLs.

where I_s is the intensity (flux emitted per unit solid angle) of the point light source and $f_r^d(v)$ the diffuse BRDF of the surface containing the VPL v. Finally, given a VPL v, we compute its radiance L' using equation 7.3, then its emittance ϕ_v (equation 7.2) and its intensity I_v (equation 7.1). Using the Monte Carlo integration, we perform these computations for all the VPLs to calculate the radiance $L(x)$ at a gather point x due N randomly selected VPLs as follows [GS10]:

$$L(x) = \sum_{v=1}^{N} I_v f_r^d(x) \frac{cos\theta_2}{d_2^2} \frac{1}{p_v} \tag{7.4}$$

where p_v is the pdf (probability density function, corresponding to the used CDF) of the accepted VPL v, I_v the intensity of VPL v, and $f_r^d(x)$ the BRDF at the visible point x (see figure 7.3 for the other notations).
Note that the radiance of the gather point x due to a VPL (not randomly selected) is given by:

$$L(x) = I_v f_r^d(x) \frac{cos\theta_2}{d_2^2} \tag{7.5}$$

7.3.2 Stratified CDF

In this section we will investigate how to improve the VPL selection with global CDF method and by stratification, we propose to divide the screen into four regions, then we compute the average contribution of a gather points subset in each region. Moreover, we stratified the image plane to ensure that all gather points are tested, but we compute only one CDF, so in one rendering pass we generate one VPL for each paraboloid (front or back). The following figure illustrates the algorithm of this proposed method:

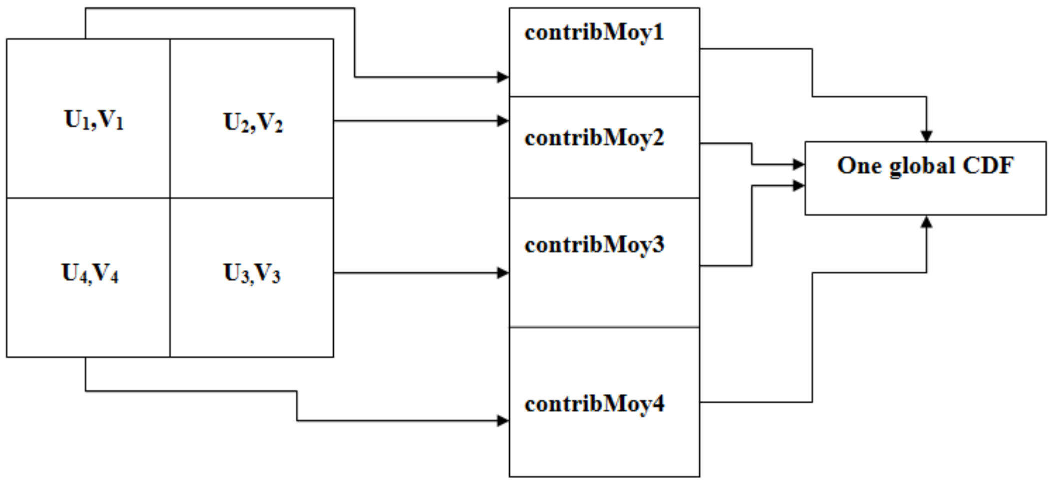


FIGURE 7.4: Overview of Stratified CDF: we compute the average contribution of each screen quarter, then we merge it to generate one CDF.

7.3.3 Local CDF

In this section, we describe our first method (that we call local CDF) which uses N CDFs unlike a method which uses a single CDF also called global CDF. As detailed in subsection 7.2.2 let us summarize how a global CDF is computed using the method presented in [REH+11]. Once an RSM (in our methods we use a DPRSM) is computed, a subset S_{GP} of gather points GP is randomly selected (corresponding to a subset of pixels). Then, for each VPL within the DPRSM, its contribution to each GP of S_{GP} is computed using equation 7.5. Then each VPL v is assigned an average contribution which is equal to the sum of its contributions to the GP of the subset S_{GP} divided by the cardinal of S_{GP}. These VPL average contributions help build a CDF which will be used to sample VPLs from the constructed DPRSM (see subsection 7.2.2 for more details). The way the $GPs \in S_{GP}$ are distributed over the image plane is crucial for an efficient calculation of a CDF (efficient importance sampling). To better distribute these GP, we propose to subdivide the image plane into N regions. For each region i, we perform a uniform sampling to get a subset of GPs, called S^i_{GP}, used to compute a local CDF (called CDF_i). Once all the local

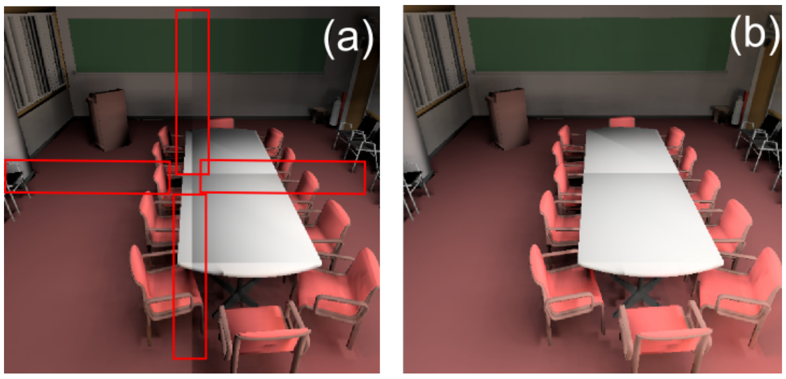


FIGURE 7.5: Illustration of the discontinuity problem when we use one CDF for each gather point. In this example, the image plane is subdivided into 4 regions, (a) local CDF method with only one CDF for each GP of a region, we see discontinuity at the boundaries of regions; (b) local CDF method with 4 CDF per GP, the discontinuities have disappeared.

CDF_i are computed, to render an image we compute the radiance of all the GPs within a region i using only CDF_i. Even though this approach seems interesting, it is source of artifacts consisting of discontinuity at the region boundaries as shown in figure 7.5. To overcome this problem we proceed as follows. In the rendering step, to compute the radiance of each GP of the image plane, all the N CDF_i are sampled at the same time, then the contributions of the N selected VPLs are computed and assigned to the GP. This process is repeated N_{NG} times. Thus, the radiance of each GP requires the sampling of $N_{NG} \times N$ VPLs.

The algorithm 8 summarizes the rendering step when we divide the screen to 4 regions and using one CDF for each gather point:

Algorithm 8 *2D_texture_image* renderingWithDivide(contribution contrib1, contribution contrib2, contribution contrib3, contribution contrib4)

```
1: gBuffer = GenerateGBuffer(); //  a buffer containing the position, normal,
   color of the visible points from the camera view
2: dprsm = GenerateDualParaboloidRSM(); // front and back buffers containing
   position, normal, color of the visible points from the light source
3: for each gather point $(i, k)$ from the gBuffer do
4:    for  j = 1 to #NBVPL do
5:       // NBVPL: small number of VPLs
6:       psi = sample_uniform_random_variable(); // psi ranging from 0 to 1
7:       find_region = determine_screen_part(); // function returns the local region
         of the screen
8:       if find_region == region1 then
9:          pdf1 = pdf_region(region1); // pdf of the VPL in the first region
10:         contrib1 = computeContribution(uvVPL) / pdf1; // the contribution
            of the first region
11:      else
12:         if find_region == region2 then
13:            pdf2 = pdf_region(region2); // pdf of the VPL in the second region
14:            contrib2 = computeContribution(uvVPL) / pdf2; // the contribution
               of the selected VPL in the second region
15:         else
16:            if find_region == region3 then
17:               pdf3 = pdf_region(region3); // pdf of the VPL in the third region
18:               contrib3 = computeContribution(uvVPL) / pdf3; // the contribu-
                  tion of of the selected VPL in the third region
19:            else
20:               if find_region == region4 then
21:                  pdf4 = pdf_region(region4); // pdf of the VPL in the last region

22:                  contrib4 = computeContribution(uvVPL) / pdf4; // the contri-
                     bution of the selected VPL in the third region
23:               end if
24:            end if
25:         end if
26:      end if
27:   end for
28: end for
29: finalContrib = contrib1 + contrib2 + contrib3 + contrib4;
30: return finalContrib;
```

Recall that, the VPLs are sampled from the front or the back DPRSM using the Russian roulette technique [AK90] (see algorithm 9).

Algorithm 9 *face selectParaboloid(FACE face, random psi)*

```
1: vF = sumContrib(front); // average contribution sum for front face
2: vB = sumContrib(back); //average contribution sum for back face
3: bF = vF/(vF+vB);
4: bB = vB/(vF+vB);
5: if psi < bF then
6:     return front_face;
7: else
8:     return back_face;
9: end if
```

The final contribution is then divided by the probability of choosing a front PRSM (bF) or by the probability of choosing a back PRSM (bB) according to the Russian roulette algorithm (see algorithm 10).

Algorithm 10 *2D_texture_image* final_rendering(contribution finalContrib)

```
1: finalContrib = renderingWithDivide(contrib1, contrib2, contrib3, contrib4)
2: if selectParaboloid(face, psi) == front_face then
3:     //see algorithm 9
4:     uvVPL = sampleFromFront(); // select the VPL of coordinates uvVPL from
       the front face
5:     image[i][k] += finalContrib / bF //bF: probability of selecting the front face
6: else
7:     if selectParaboloid(face, psi) == back_face then
8:         //see algorithm 9
9:         uvVPL = sampleFromBack(); // select the VPL of coordinates uvVPL
           from the back face
10:        image[i][k] += finalContrib / bB //bB: probability of selecting the back
           face
11:    end if
12: end if
13: return image; // rendered image
```

7.3.4 Gathering-based global CDF

We describe in this section our second method for computing a global CDF used in an inverse transform approach. We call this method: Gathering-Based Global CDF (called GBG from now on). This method is global because it does not need a subdivision of the image plane into regions. Our objective is to compute a more efficient global CDF than the one proposed by Ritschel et al. [REH+11]. Our approach differs from this method in the way the VPL average contributions (see subsection 7.2.2 for more details) are computed. Indeed, rather than computing these contributions for a small subset of GPs, our approach computes them for all the GPs. Our GBG method works as follows (Algorithm 11). At each GP, a hemisphere is placed above it, then a set of rays are randomly (according to a pdf) traced from the GP. For each ray, the first intersection point P is computed using Ray Marching through the voxel-based subdivision of the scene [CG12] (line 7). Then we project P onto the shadow map DPRSM (line 11). If P is visible from the DPRSM then it corresponds to a VPL. In this case we compute its contribution (see equation 7.5) to the current GP which is stored in a GBuffer. This contribution is updated when considering the rest of the GPs. This process is repeated for all the GPs. The result is the total average contributions of all the VPLs stored in the DPRSM. The computed average contributions (line 24) are stored in a texture. As the method is based on ray sampling through a GP hemisphere, it may happen that some VPLs are not reached by the sampled rays, consequently there contribution is null, which corresponds to holes in the associated texture. To fill the holes, we propose to use a 3×3 median filter as a reconstruction filter. Figure 7.6 illustrates an example of 3×3 median filter.

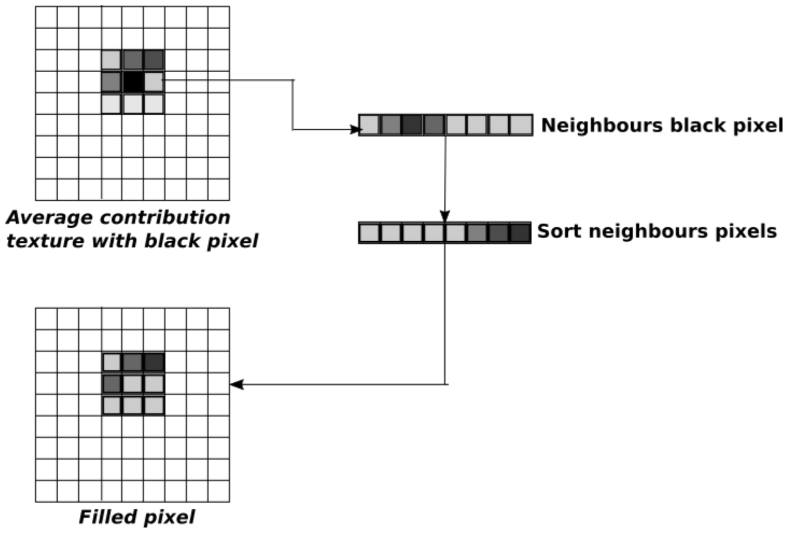


FIGURE 7.6: Example of median filter

The computation of an average contribution of all the VPLs is described by algorithm 11.

Note that the average contributions is a luminance that is determined by converting an RGB color into a scalar value called luminance using the following formula:

$$Luminance = 0.299 * R + 0.587 * G + 0.114 * B \tag{7.6}$$

Algorithm 11 *2D_texture_image* compute_average_contribution()

```
1: gBuffer = GenerateGBuffer(); // a buffer containing the position, normal,
   color of the visible points from the camera view
2: dprsm = GenerateDualParaboloidRSM(); // front and back buffers containing
   position, normal, color of the visible points from the light source
3: for each gather point $(i, k)$ from the gBuffer do
4:   for j = 1 to #NBDIR do
5:     // NBDIR: number of random directions in the hemisphere
6:     dir = randomDirection();
7:     its = rayMarching(dir);
8:     if its == 1 then
9:       // 1 if intersection point found by Ray Marching
10:      P = computeOutgoing(); // intersection point P
11:      uvVPL = projectPraboloid(P, dprsm);
12:      // project the outgoing point P into the DPRSM and return its UV
         coordinates in the variable $uvVPL$
13:      if visible(uvVPL, front) then
14:        // if the VPL belongs to the front PRSM
15:        nbVPLFront = nbVPLFront + 1; // nbVPLFront number of the vis-
           ible VPLs in the front face
16:      else
17:        if visible(uvVPL, back) then
18:          // if the VPL belongs to the back PRSM
19:          nbVPLBack = nbVPLBack + 1; // nbVPLBack number of the
             visible VPLs in the back face
20:        end if
21:      end if
22:    end if
23:  end for
24:  average_contrib[i]k] += computeContribution(uvVPL) / (nbVPLFront +
     nbVPLBack); // computeContribution() uses equation 7.5
25: end for
26: return average_contrib; // image of average contributions
```

Given the VPL average contributions, we compute a CDF (called GBG) according to Ritschel et al's method (see subsection 7.2.2). We render the scene using this CDF as follows (algorithm 12). First from each gather point visible from the view camera, we randomly sample a small number of VPLs that are selected from the DPRSM. In using a Russian roulette, we sample the VPL from the front PRSM (line 9) or from the back PRSM (line 14). As we use the Russian roulette technique, the final contribution is divided by the propability of selecting the front or the back face (line 10 and line 15).

Algorithm 12 *2D_texture_image* renderingWithCDF()

```
1: gBuffer = GenerateGBuffer(); // a buffer containing the position, normal, color of the visible points from the camera view
2: dprsm = GenerateDualParaboloidRSM(); // front and back buffers containing position, normal, color of the visible points from the light source
3: for each gather point $(i, k)$ from the gBuffer do
4:    for j = 1 to #NBVPL do
5:       // NBVPL: small number of VPLs
6:       psi = sample_uniform_random_variable(); // psi ranging from 0 to 1
7:       if selectParaboloid(face, psi) == front_face then
8:          //see algorithm 9
9:          uvVPL = sampleFromFront(); // select the VPL of coordinates uvVPL from the front face
10:         image[i][k] += computeContribution(uvVPL) / bF //bF: probability of selecting the front face
11:      else
12:         if selectParaboloid(face, psi) == back_face then
13:            //see algorithm 9
14:            uvVPL = sampleFromBack(); // select the VPL of coordinates uvVPL from the back face
15:            image[i][k] += computeContribution(uvVPL) / bB //bB: probability of selecting the back face
16:            // computeContribution() uses equation 7.4
17:         end if
18:      end if
19:   end for
20: end for
21: return image; // rendered image
```

7.3.4.1 Average contribution texture

To generate mcVPLs, we compute a discrete CDF, so we need to know the average contribution of all the VPLs in the dual paraboloid RSM.
From each gather point, we compute the contribution of the visible VPL as in (algorithm 1), but we can find that the same VPL contributes to two (or more) gather points. In this case, we do the sum of contributions.

The final result can contain some holes because not all the VPLs have a contribution, for that we must fill in the holes before computing CDF. The figure below depicts the overview of the average contribution texture:

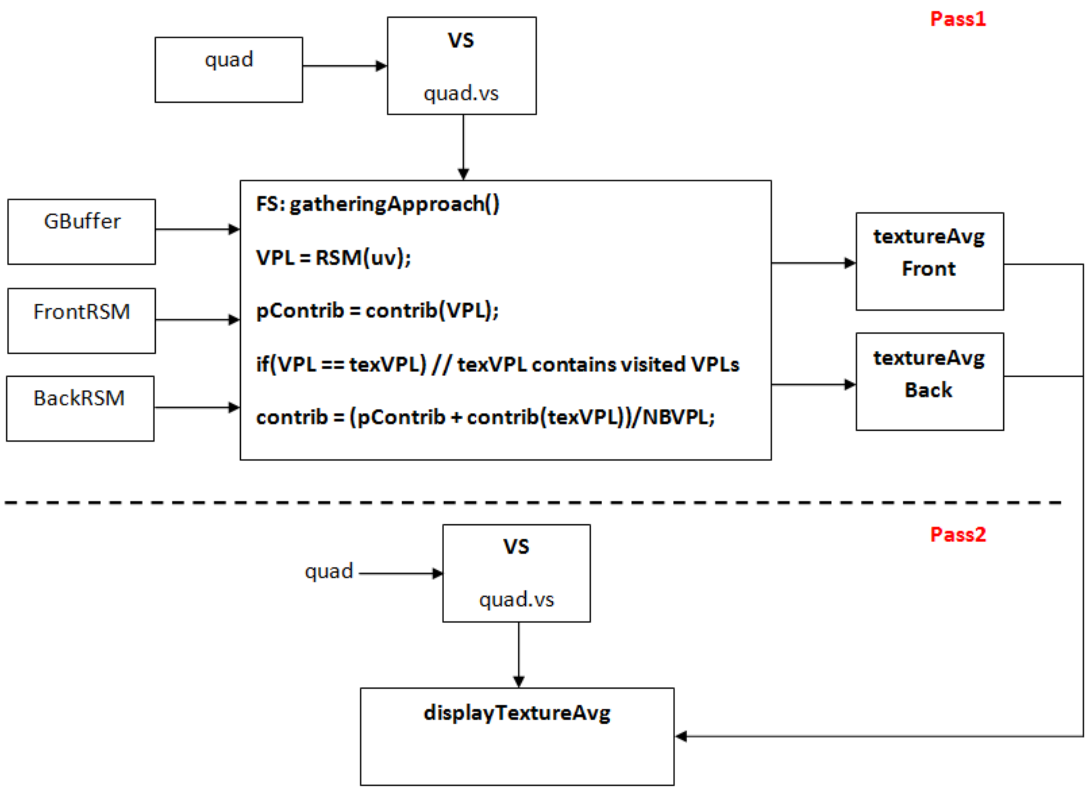


FIGURE 7.7: Algorithm overview of the average contribution texture on GPU.

7.3.4.2 Implementation details

As mentioned earlier, our goal is to compute the average contribution texture. The idea is based on the gathering approach where the directions shoot from each gather points are randomly chosen.When a given VPL contributes to two or more gather points, it is necessary to update the average contribution texture. In this section, we describe how we can update a texture using the GLSL. The following figure shows an example of scene:

For each gather point, we have a fragment shader and in GLSL we have two types of textures:

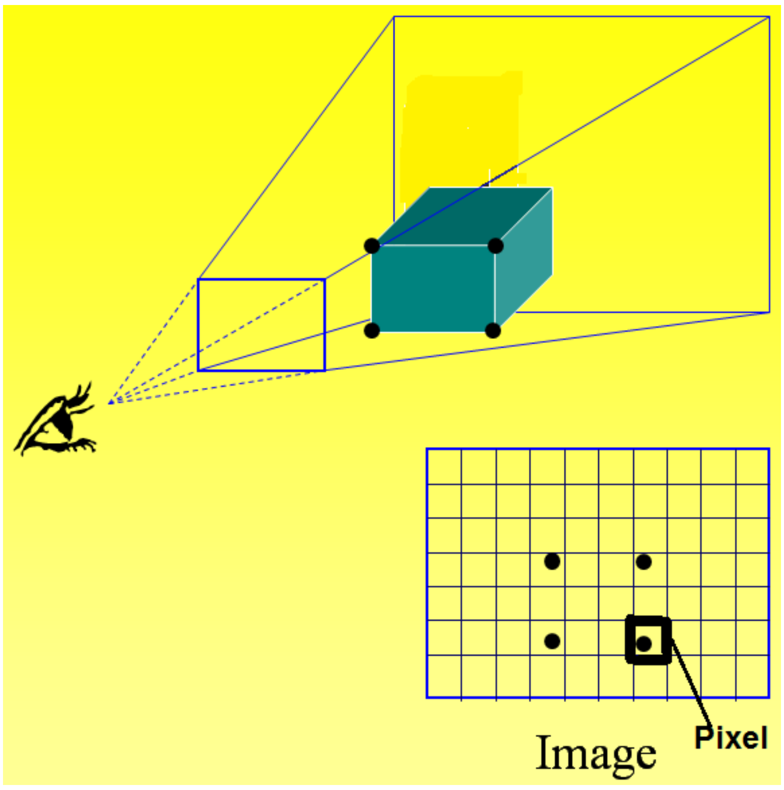


FIGURE 7.8: Scene representation.

1. sampler2D : access to a texture and get back its value;

   ```
   uniform sampler2D tex;
   value = texture(tex, UV);
   ```

   where UV is the texel coordinates of the current gather point

   We can use this texture for just reading the information stored in the UV coordinates.

2. out: this texture type corresponds to an input in the Frame Buffer Object (FBO) which can contain three kinds of textures: color, depth, and stencil. Figure 3 illustrates the FBO components: With this texture, we can write a value and we address it with GL_COLOR_ATTACHMENT

   On the fragment shader, we write:

   ```
   layout(location = 0) out float contribMoy;
   ```

   Which design the first input in the FBO

   We put in this texture a value that has the same data type (float in this case) to define the average contribution of the current gather point:

   ```
   contribMoy = computeContribution();
   ```

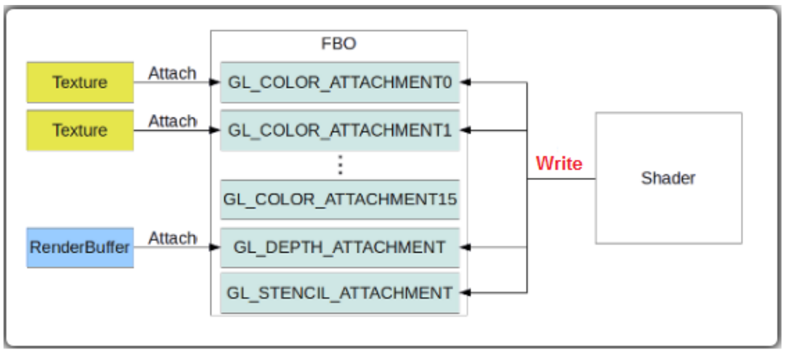


FIGURE 7.9: FBO components.

So we store the average contribution in the UV coordinates of the gather point. For the second gather point, we create another fragment shader and we execute the same operations. We do it for all the gather points

We obtain the following result But we should display the contributions from the

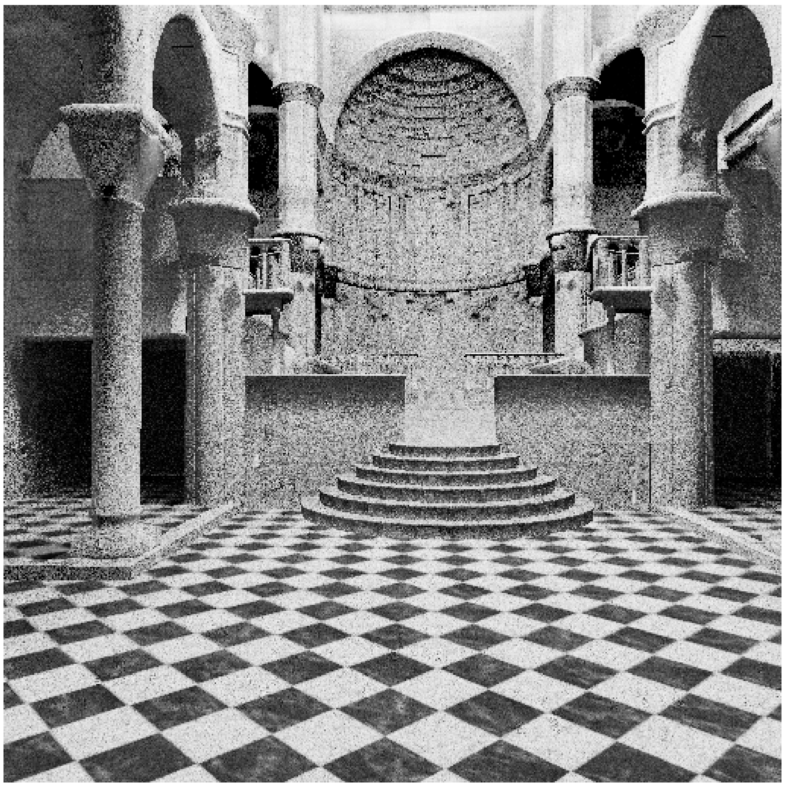


FIGURE 7.10: Back average contribution texture.

light of view according to the position of the VPLs (front or back). To overcome this problem we transform the gather points in either front or back texture.

In the fragment shader and in the case of the back texture we have:

```
vec2 UV: the interpolated coordinates of such gather point
vec3 coordUV = (depthMVPBiasFront * vec4(UV, 1.0, 1.0)).xyz;
```

We multiply all the gather points coordinates by the transform matrix associated with the front texture. We obtain the result below:

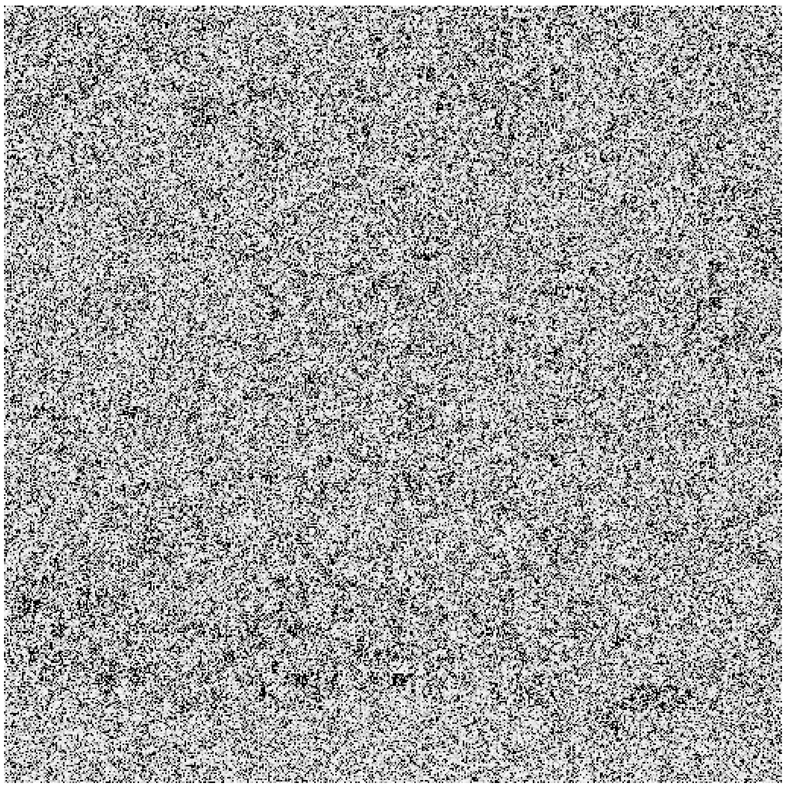


FIGURE 7.11: Back average contribution texture (from the back texture).

7.3.4.3 Filling in holes

When we look to the last result (in figure 5) we can find some little holes, so before computing the CDF, we must fill in holes. For that, we propose to use a reconstruction filter (median filter). The following scheme shows how to give a value to each black pixel.

The following figure shows the comparison between the two back texture before and after filtering:

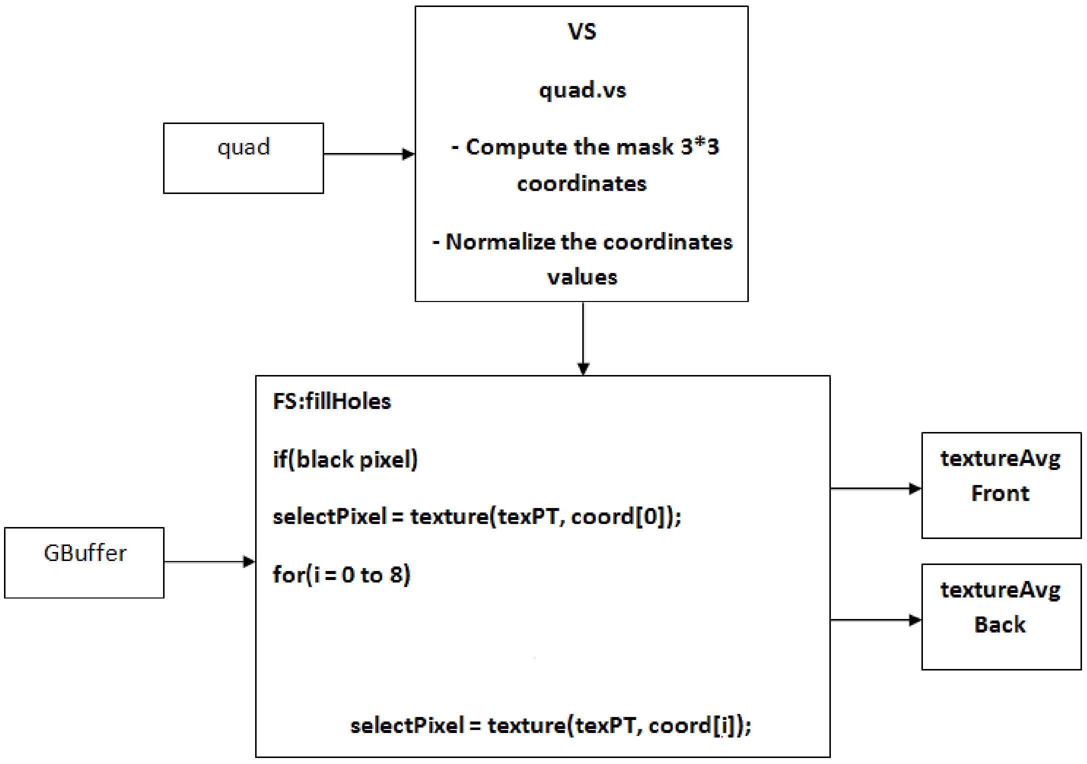

quad
VS
quad.vs
- Compute the mask 3*3 coordinates
- Normalize the coordinates values
GBuffer
FS:fillHoles
if(black pixel)
selectPixel = texture(texPT, coord[0]);
for(i = 0 to 8)
selectPixel = texture(texPT, coord[i]);
textureAvg Front
textureAvg Back

FIGURE 7.12: Median filter method overview.

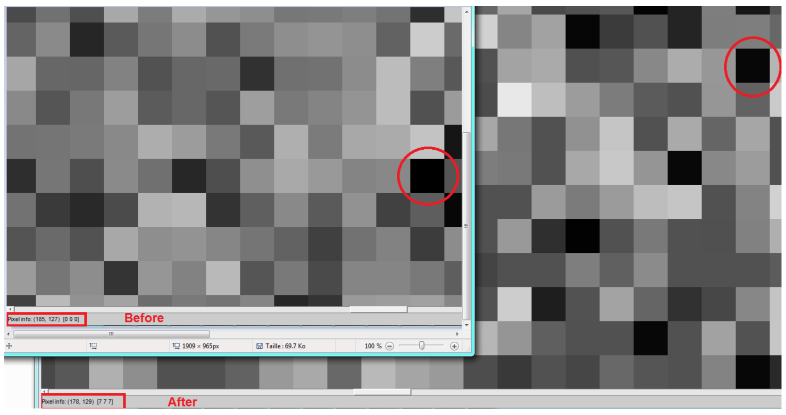


FIGURE 7.13: Median filter result.

Figure 7.14 illustrates the CPU and GPU pass of the Gathering-based global CDF. Pass 1 (on GPU) consists of generating the two maps the average contribution map and the VPL cumulation map. The average contribution map which contains the average contribution of the VPL to the gather points. During the generation of the average contribution texture, we can find that some VPLs reached several times, thus, in the second map (VPL cumulation), we update the occurrences of the reached VPLs.

In the second pass (on CPU) we retrieve the average contribution and the accumulation VPLs maps to the CPU in order to compute our gathering-based global CDF. Then, we find that the VPL contribution texture contains holes because some VPLs are not reached, we fill in holes using the median filter (see subsection 7.3.4.3 for more details) and we compute the CDF by using the inverse transform method as proposed by Ritschel et al's [REH+11] (third pass on CPU). After computing our GBG CDF, we send the selected VPLs according to this GBG CDF to the GPU (fourth pass), and we compute the contributions of these VPLs (see algorithm 12).

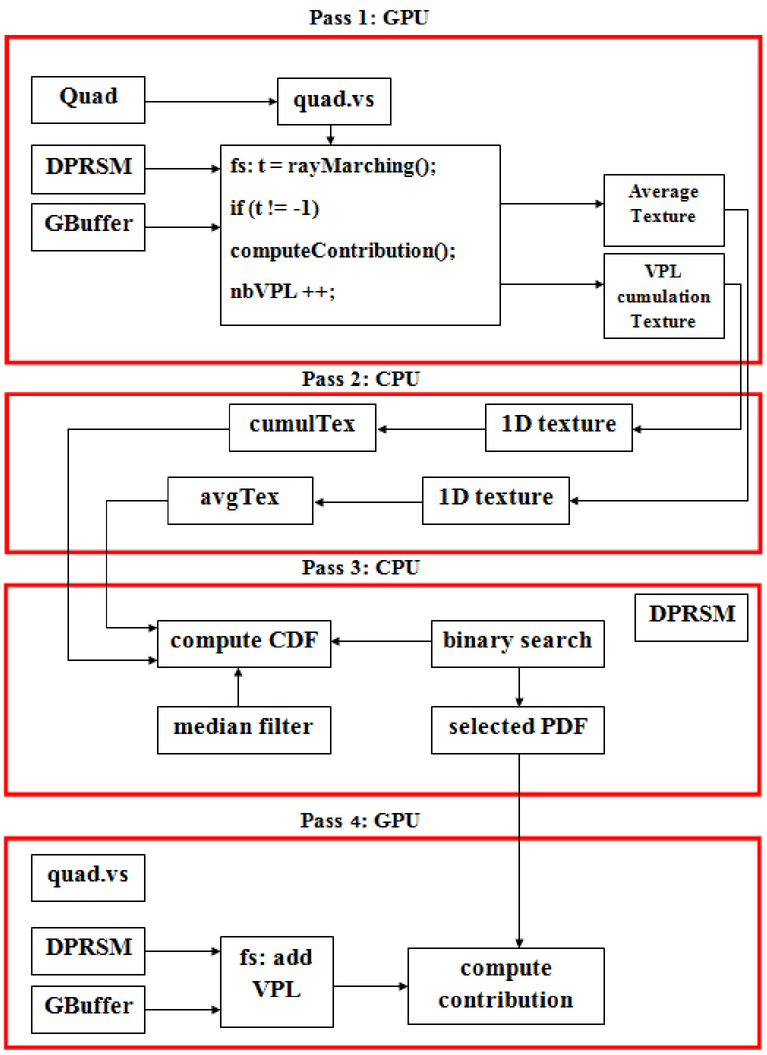


FIGURE 7.14: Different passes on CPU and GPU for computing our gathering-based global CDF.

7.4 A Multiple Importance Sampling Approach

In this section, our objective is to improve a CDF-based rendering method (inverse transform) by combining it with a gathering-based rendering method. We propose *Multiple Importance Sampling* (MIS) to carry out this combination of two estimators.

7.4.1 Background on MIS

Let us compute two Monte Carlo estimators of the integral of $f(x)$, one with a sampling distribution (PDF) $p_1(x)$ and the other with PDF $p_2(x)$.
In our case we have two rendering strategies:

1. Gathering approach: Monte Carlo method sampling a hemisphere placed above a GP and using a cosine PDF;

2. Inverse transform method: using a CDF computed with any approach local or global.

The MIS strategy consists in combining the two Monte Carlo estimators as described by veach [Vea97]:

$$F = \frac{1}{n_1} \sum \omega_1(X_{1,j}) \frac{f(X_{1,j})}{p_1(X_{1,j})} + \frac{1}{n_2} \sum \omega_2(X_{2,j}) \frac{f(X_{2,j})}{p_2(X_{2,j})} \tag{7.7}$$

We use a weighting function defined by Veach [Vea97] to combine the two estimators. The set of weights given by this function allows to generate samples $X_{1,j}$ or $X_{2,j}$ to reduce the variance.

Veach proposes two balance heuristic weights associated with each strategy:

$$\omega_1(X_{1,j}) = \frac{p_1(X_{1,j})}{p_1(X_{1,j}) + p_2(X_{1,j})} \tag{7.8}$$

$$\omega_2(X_{2,j}) = \frac{p_2(X_{2,j})}{p_1(X_{2,j}) + p_2(X_{2,j})}, \tag{7.9}$$

$X_{1,j}$ and $X_{2,j}$ are the samples of the random variable x generated with the PDF p_1 and p_2 respectively. In our case these samples are pairs (θ, ϕ) (elevation angle,

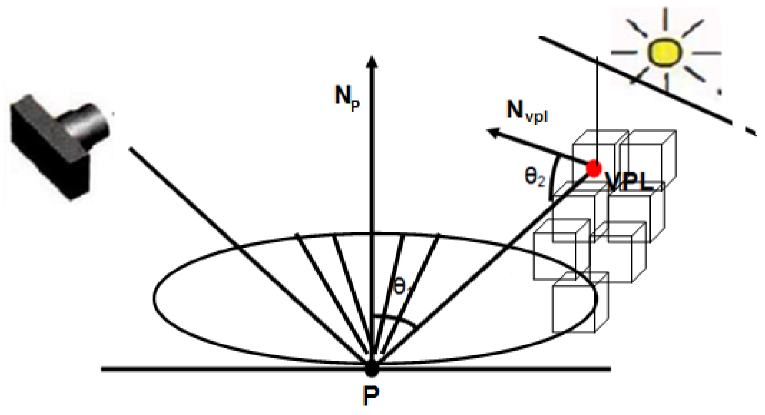


FIGURE 7.15: Representation of MIS technique in a 3D scene.

azimuthal angle) representing a direction of a ray. Note that, when using p_2 a VPL is associated with each sample direction. The first distribution (PDF) $p_1(X_{1,j})$ is used to sample a hemisphere above a GP used by the gathering-based rendering method

$$p_1(X_{1,j}) = \frac{cos\theta sin\theta}{\pi}, \tag{7.10}$$

where θ is the polar angle formed by the sample ray and the normal at the GP, while the second distribution $p_2(X_{2,j})$ is used to sample VPLs according to one inverse transform method (local or GBG).

$$p_2(X_{2,j}) = \alpha_{ij} \tag{7.11}$$

where α_{ij} is the PDF values associated with the chosen CDF (see section 7.3.4) Figure 7.15 shows how we can run the MIS technique in a 3D voxelized scene:

7.4.2 Description of our MIS method

We describe in this section our general algorithm that uses the MIS principle for rendering. We show how to combine the two estimators (gathering estimator and gathering-based global CDF estimator). To weight the contributions, we use the balance heuristic method as described in section section 7.4.1. Our main goal is to compute the PDF of each strategy when we generate samples from the other strategy ($p_1(X_{2,j})$ and $p_2(X_{1,j})$).
Our algorithm consists of four Parts:

1. Run a gathering method to generate the average contribution texture that will be used to compute the CDF: result = gathering-based global CDF (GBG CDF). See section 7.3.4.

2. Run a classical gathering-based rendering: for each gather point and for each incident direction, we compute the incident radiance due to a VPL (VPL corresponding to the projection into the the DPRSM of the first ray-scene intersection point). The result is a contribution (contrib_gathering) to all the gather points weighted by ω_1 (see algorithm 14).

3. For each gather point of the previous step, select N VPLs using the gathering-based global CDF: the result is a contribution (contrib_CDF) for all the gather points weighted by ω_2 brought by the N selected VPLs (see algorithm 15).

4. Sum the two contributions contrib_CDF and contrib_gathering (using equation 7.7).

We give in figure 7.16 an overview of the two methods combined by the MIS technique. We start by sampling the camera view for the gathering approach, when we sample the light source for our proposed inverse transform method.

Algorithm 13 summarizes our proposed method.

To apply the MIS technique, we have to find the VPL PDF values of the j samples

Algorithm 13 *2D_texture_image* MIS()

```
1: gBuffer = GenerateGBuffer(position, normal, color); // position, normal, color
   of the visible point
2: dprsm = GenerateDualParaboloidRSM(); // front and back buffers containing
   position, normal, color of the visible points from the light source
3: // pdf1 = $p_1(X_{1,(i,k)})$, pdf2 = $p_2(X_{1,(i,k)})$
4: contrib_gathering = MISdensityGathering(pdf1, pdf2);
5: // pdf3 = $p_2(X_{2,(i,k)})$, pdf4 = $p_1(X_{2,(i,k)})$
6: contrib_CDF = MISdensityVPL(pdf3, pdf4);
7: // Final contribution
8: for each gather point $(i, k)$ from the gBuffer do
9:    image[i][k] = (contrib_gathering[i][k] + contrib_CDF[i][k]) / NBDIR; //
      $NBDIR$: number of incident directions shooted from each gather point $j$
10: end for
11: return image; // rendered image with MIS and stored in a 2D_texture
```

from the gathering strategy (noted $p_2(X_{1,(i,k)})$) and the gathering distribution of

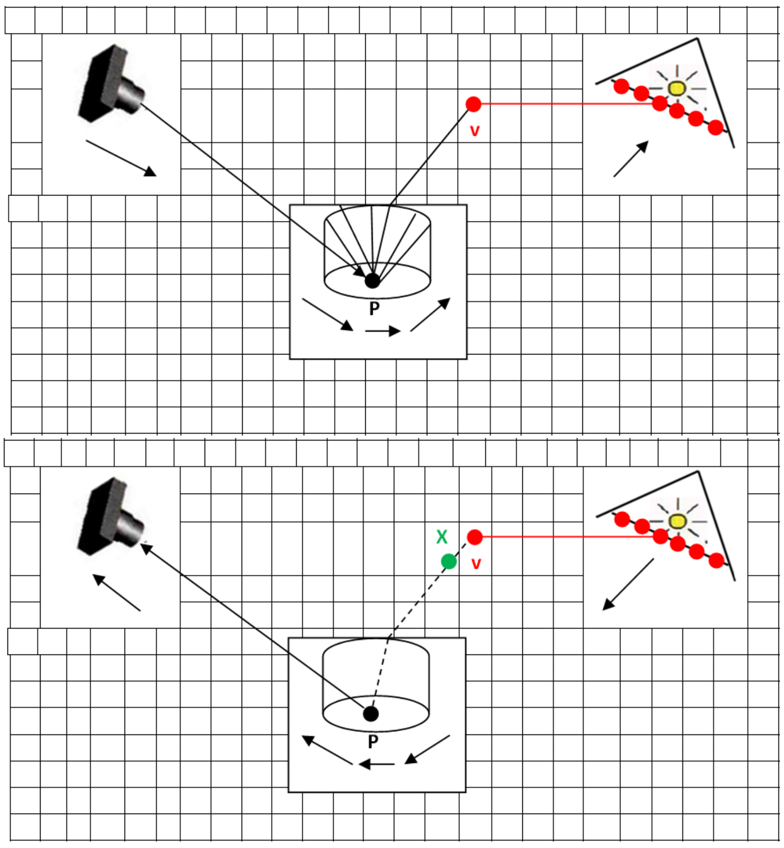


FIGURE 7.16: Overview of our MIS technique using the two strategies.

j samples from the gathering-based global CDF strategy when we compute the VPL contribution (noted $p_1(X_{2,(i,k)})$)
The method of computing $p_2(X_{1,(i,k)})$ is described in detail by Algorithm 14. To find the VPLs of the j samples from the gathering strategy, we start by shooting N random rays. According to [Dut03], the direction is generated from the hemisphere proportionally to cosine-weighted solid angle as follows: The spherical coordinates in the hemisphere are:

$$\varphi = 2\pi r_1 \tag{7.12}$$

$$\theta = acos(\sqrt{r_2}) \tag{7.13}$$

Where r_1 and r_2 are a uniform random variables ranged between $[0, 1]$
The x, y and z coordinates of the random direction are:

$$x = cos(2\pi r_1)\sqrt{1 - r_2} \tag{7.14}$$

$$y = sin(2\pi r_1)\sqrt{1 - r_2} \tag{7.15}$$

$$z = \sqrt{r_2} \tag{7.16}$$

And the PDF of selecting a such random direction is given by

$$PDF(dir) = \frac{cos(\theta)}{\pi} \tag{7.17}$$

Then, for each random direction, we forward the rays (generated randomly) in the uniform 3D grid that contains all the scene geometry until we find the first intersection. In this case, we project the outgoing point (outgoing in the algorithm 14) to the DPRSM. After that, we test if outgoing is visible in one face of the DPRSM (front or back) and we compute its contribution.

Algorithm 14 *2D_texture_image* MISdensityGathering(PDF pdf1, PDF pdf2)

```
gBuffer = GenerateGBuffer(position, normal, color); // generate the position, normal, color of the visible point from the view camera
dprsm = GenerateDualParaboloidRSM(); // front and back buffers containing position, normal, color of the visible points from the light source
for each gather point $(i, k)$ from the gBuffer do
  for j = 1 to #NBDIR do
    // NBDIR: number of random directions in the hemisphere
    dir = randomDirectionPropToCosine(x, y, z);
    pdf1 = dir.z / pi;
    its = rayMarching(dir);
    if its == 1 then
      // intersection found
      outgoing = computeOutgoing();
      uvVPL = projectPraboloid(outgoing, dprsm);
      // project the outgoing point into the DPRSM and return the UV coordinates of the associated VPL
      if visible(uvVPL) then
        pdf2 = getPDF(uvVPL); // retrieve the pdf value associated with the VPL
        w1 = balanceHeuristic(pdf1, pdf2); // see equation 7.8
        contrib_gathering[i][k] += $(w1 \times computeContribution(uvVPL))$ / pdf1; // computeContribution() uses equation 7.4
      end if
    end if
  end for
end for
return contrib_gathering; // image generated with a gathering-based rendering method
```

The gathering PDF $p_1(X_{2,(i,k)})$ of the (i, k) sample, generated from the gathering-based global CDF, is computed as:

$$p_1(X_{2,(i,k)}) = p_2(X_{2,(i,k)}) \times p_{pv} \tag{7.18}$$

Where $p_2(X_{2,(i,k)})$ is the probability of selecting a VPL according to the gathering-based global CDF method, p_{pv} is the probability of connecting the gather point p to the selected VPL v. As this connection is deterministic its distribution (pdf) is equal to 1.
Algorithm 15 illustrates the method of computing $p_1(X_{2,(i,k)})$.

To compute $p_1(X_{2,(i,k)})$ we search for the angle θ between the normal $\vec{N}$ of the gather point and the direction $\vec{pv}$ formed by the gather point and the selected VPL.
We have:

$$cos\theta = \langle \vec{N}, \vec{pv} \rangle \tag{7.19}$$

$$\theta = acos(\langle \vec{N}, \vec{pv} \rangle) \tag{7.20}$$

Algorithm 15 *2D_texture_image* MISdensityVPL(PDF pdf3, PDF pdf4)

```
1: gBuffer = GenerateGBuffer(position, normal, color); // generate the position, normal, color of the visible point from the view camera
2: rsm = GenerateDualParaboloidRSM(); // front and back buffers containing position, normal, color of the visible points from the light source
3: for each gather point $(i, k)$ from the gBuffer do
4:    for j = 1 to #NBVPL do
5:       vpl = getVPL(); // selected VPL according to a global CDF
6:       pdf3 = getPDF(vpl); // pdf $p_2(X_{2,(i,k)})$ of the selected VPL determined by VPL method
7:       gp = pointFromGBuffer(gBuffer); // gp = gather point
8:       distance = length(vpl - gp);
9:       dir = normalize(distance);
10:      outgoing = rayMarching(gp, dir);
11:      if (outgoing - distance) ¡ psi then
12:         // visible gather point
13:         contrib = computeContribution(); // computeContribution() uses equation 7.4
14:         theta = computeAngle(normal, dir);
15:         pdf4 = $(cos(theta) \times sin(theta))/pi$;
16:         w2 = balanceHeuristic(pdf4, pdf3); // see equation 7.9
17:         contrib_CDF[i][k] += $(w2 \times computeContribution(vpl))/pdf3$; // computeContribution() uses 7.4
18:      end if
19:   end for
20: end for
21: return contrib_CDF; // image generated with a GBG CDF-based method
```

So we can compute $p_1(X_{2,(i,k)})$ as:

$$p_1(X_{2,(i,k)}) = \frac{cos\theta sin\theta}{\pi} \tag{7.21}$$

7.5 Results and evaluation

In this section, we show some results obtained with our two CDF-based methods (local CDF and gathering-based global CDF) as well as our MIS method. We validate our results in terms of computation speed, and objective and perceptual qualities. Our test scenes contain only static diffuse objects. We have used four scenes: Sibenik , Conference room , Sponza Buddha and Crytek Sponza scenes. We have placed one point light source in these scenes and considered single bounce indirect illumination. Our methods run on the GPU (NVIDIA Geforce GTX 780 OC 6GB) and on the CPU (Intel i7 930 @ 2.80 GHZ, 8GO RAM running 64 bits) using OpenGL 4.3.

In the following, we describe the specifications of each scene. The four scenes used in this chapter are available in [McG11]. Their common characteristic is that they contain only diffuse BRDF because our proposed methods run for diffuse objects only. The Conference room scene contains $331,179$ triangles without textures. While the Sibenik scene is a closed scene containing $75,284$ triangles with textured objects. On the other hand, the Sponza Buddha and the Crytek Sponza scene is an open scene.

We illustrate in figure 7.17 the voxelization-based method (to speed up visibility calculation) by Crassin and Green [CG12] for Sibenik, Conference, Sponza Buddha and Crytek Sponza scenes. To capture the scene details, we choose a voxel grid with a resolution 128^3.

The images, generated with our method, have been compared to reference images computed with Ritschel et al's global CDF method (see section 7.2.2). These reference images have been generated using a global CDF computed with a large number ($4k$) of gather points (used to compute the average contributions of the VPLs) and a large number of VPLs ($10k$ VPLs) used to compute the radiance of each gather point during rendering. The resolution of the computed images is 512×512.

When rendering using a local CDF or gathering-based global CDF (GBG CDF),

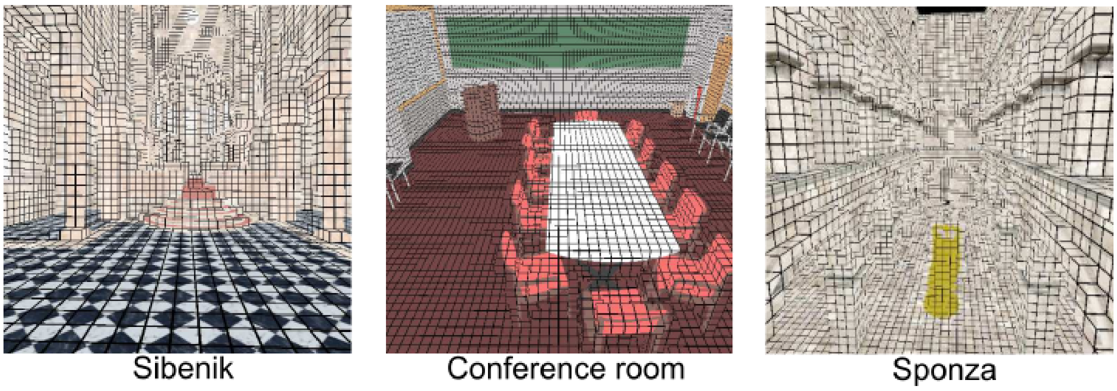


FIGURE 7.17: a, b and c show the voxelization-based approach of the Sibenik, the Conference scene and the SponzaBuddha scene respectively. We use a voxel grid with a resolution equal to 128^3

800 VPLs are selected randomly according to the used CDF. One DPRSM is computed for the point light source placed in the scene. Recall that a DPRSM consists of two PRSMs: front and back. Each PRSM is assigned three buffers: position buffer, normal buffer and color buffer.

Figure 7.18 provides results obtained with our local CDF, our GBG CDF and our MIS methods together with the reference image of the four test scenes. Regarding the local CDF approach, we have subdivided an image into four regions (say $N = 4$, see section 7.3.3). Images (b), (f) and (j) provide better results than those obtained with the global CDF-based method [REH+11] (images (a), (e) and (i)) since some regions of the image are better shaded while they look dark when using the global CDF-based method. This can be explained by the fact that with a local CDF the gahtering points (used to build a CDF) are uniformly distributed over the image (similar to stratification).
For the same reasons, our GBG CDF method gives better results (images (c) (g) and (k)) compared to the global CDF method [REH+11] (images (a), (e) and (i)). This is due to the fact the global CDF-based method assumes that a gather point is visible from all the VPLs when computing the CDF, which is a strong assumption. Rather, our GBG CDF computes visibility through Ray Marching in a voxel grid.
Table 7.1 gives some rendering times in milliseconds for four methods: global CDF, Local CDF, gathering-based CDF and MIS. Our GBG CDF and MIS methods are faster than the global CDF-based method. While generating better results, our local CDF-based method is slower than the global CDF-based method because it repeats four times (one for each region) the process of computing a CDF using a

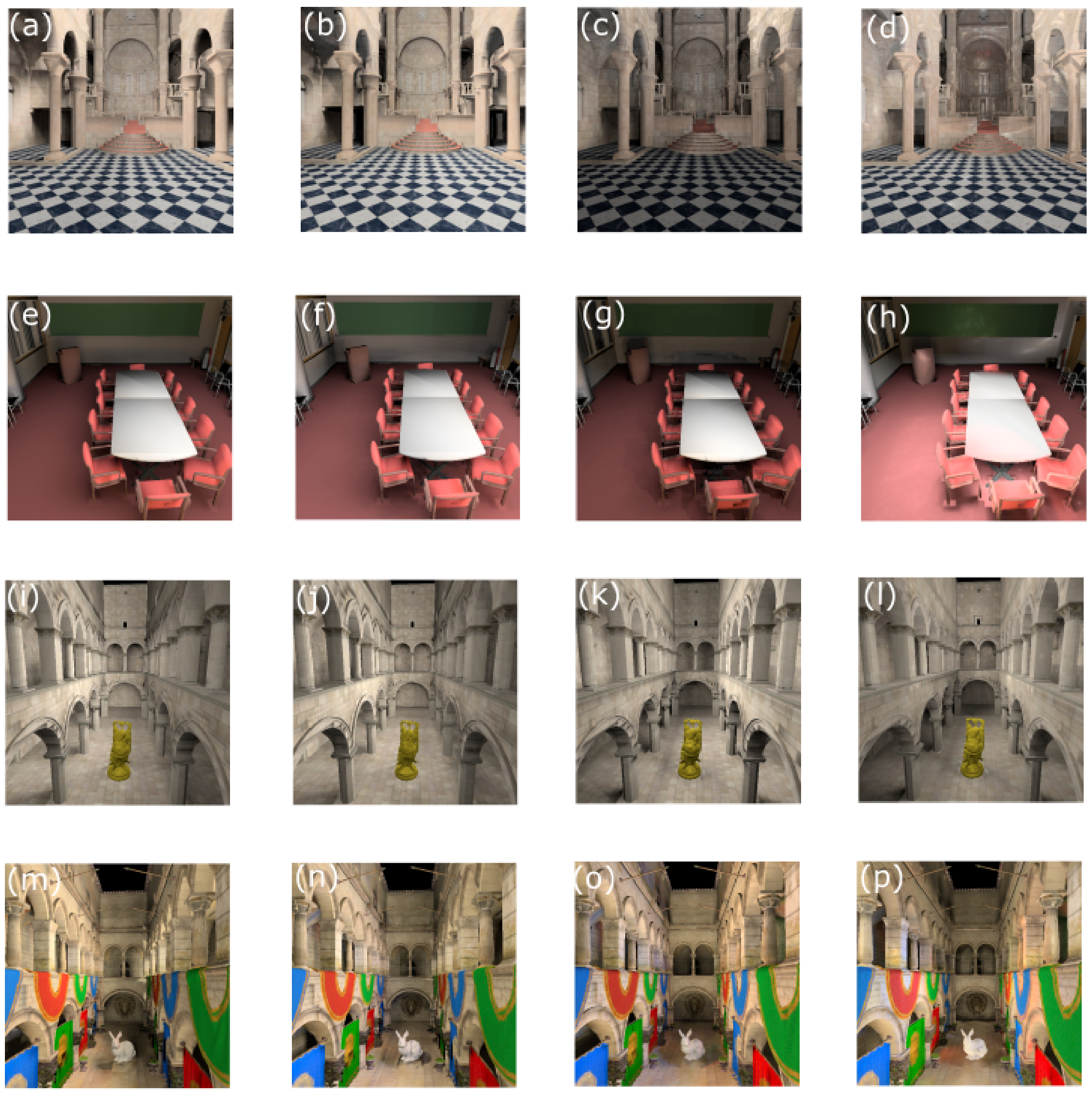


FIGURE 7.18: The Sibenik, Conference, Sponza Buddha and Crytek Sponza scenes have been rendered using 800 randomly selected VPLs per gather point and 800 directions for the MIS method. Image (a), (e), and (i) are the reference images for the four scenes generated with the global CDF-based method [REH+11], the contribution of each gather point is computed using a large number of VPLs (10k VPLs). Images (b), (f), and (j) have been computed using our local CDF. Images (c), (g) and (k) have been generated with our GBG CDF method. Our MIS method provides the images (d), (h), and (l).

method similar to that of the global CDF-based method.

Scene	Sibenik	Conference	Sponza Buddha
Global CDF	600 ms	560 ms	270 ms
Local CDF	630 ms	640 ms	410 ms
GBG CDF	350 ms	340 ms	190 ms
MIS method	460 ms	460 ms	230 ms

TABLE 7.1: Time rendering of our local CDF, GBG CDF, and MIS methods compared to the global CDF method. This time rendering is computed in milliseconds for the four scenes using the same number of VPLs (800 VPLs for each method).

Table7.2 summarizes the time for generating the GBuffer containing all the information (position, normal, color) of each visible point (gather point). It also provides the time for generating the DPRSM and the visibility time that is computed using a Ray Marching algorithm. Note that the shadow maps resolution is 512×512. The time is computed in milliseconds. We observe that the Ray Marching algorithm (visibility computation) is the most expensive step for each scene (Sibenik $79.86ms$, Conference $78.71ms$, and Sponza Buddha $80.27ms$). Furthermore, the DPRSM generation step takes a few milliseconds for the Conference ($4.54ms$) and the Sponza Buddha ($5.64ms$) scenes but it takes $33.85ms$ for the Sibenik scene because the Sibenik scene contains a large number of points stored in the DPRSM. For the same reasons, we found that the GBuffer time generation is higher for the Sibenik scene ($15.23ms$) compared to the Conference scene ($0.82ms$) and the Sponza Buddha scene ($1.06ms$).

Scene	Sibenik	Conference	Sponza Buddha
GBuffer	15.23 ms	0.82 ms	1.06 ms
DPRSM	33.85 ms	4.54 ms	5.64 ms
Ray Marching	79.86 ms	78.71 ms	80.27 ms

TABLE 7.2: Time for generating GBuffer, DPRSM, Ray Marching for Sibenik, Conference, and Sponza scenes in millisconds. The Shadow Map resolution is 512×512

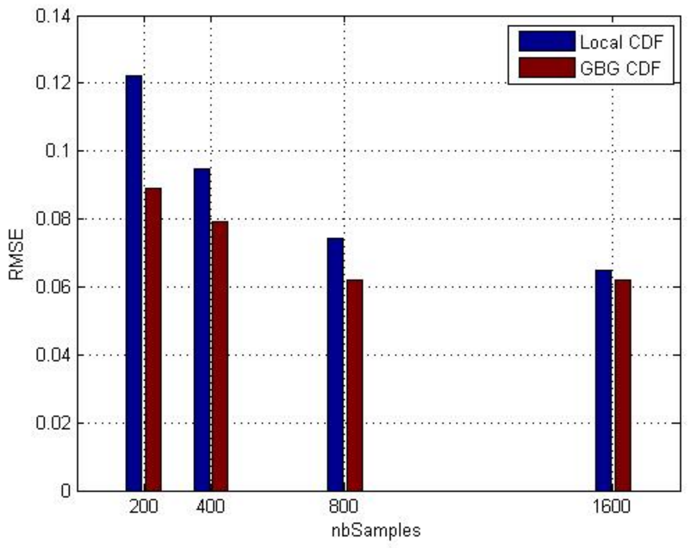


FIGURE 7.19: RMSE results for the Conference scene as a function of the number of VPLs used for rendering.

Now we evaluate our local-based CDF and gatheirng-based global CDF methods by using the RMSE metric and the HDR-VDP-2 [MKRH11] metric which is a perceptual metric applied to HDR images (High Dynamic Range). Note that all the images generated by our method are HDR images and the reference images are generated with the global CDF-based method as explained above. Figure 7.19 gives the RMSE for the local CDF-based and gathering-based CDF methods as a function of the number of VPLs selected during the rendering step. We can notice that the RMSE is small for the two methods and decreases when the number of VPLs increases. An interesting result is that, for the gathering-based CDF method, the RMSE reaches its smaller value for a number of 800 VPLs, which means that this number is sufficient for getting good quality results.
Figure 7.20 shows the perceptual difference between the images, generated with our local CDF-based and gathering-based CDF methods, and the reference images. The perceptual differences are evaluated using the HDR-VDP-2 [MKRH11] perceptual metric applied to HDR images. We observe that most of the image of the test scenes is assigned a low error given by the HDR-VDP-2 metric (blue and green parts in figures a, c, e, g, i, k). Therefore, our local CDF and our GBG CDF images closely resemble the reference image. However, there exist small regions with an non negligible error (a, c, e, g, i, k). To reduce these high values (red parts), we can increase the number of VPLs during rendering.
Figure 7.21 shows results obtained with the gathering-based CDF and MIS methods for the Sibenik , Conference, Sponza Buddha and Crytek Sponza scenes. We

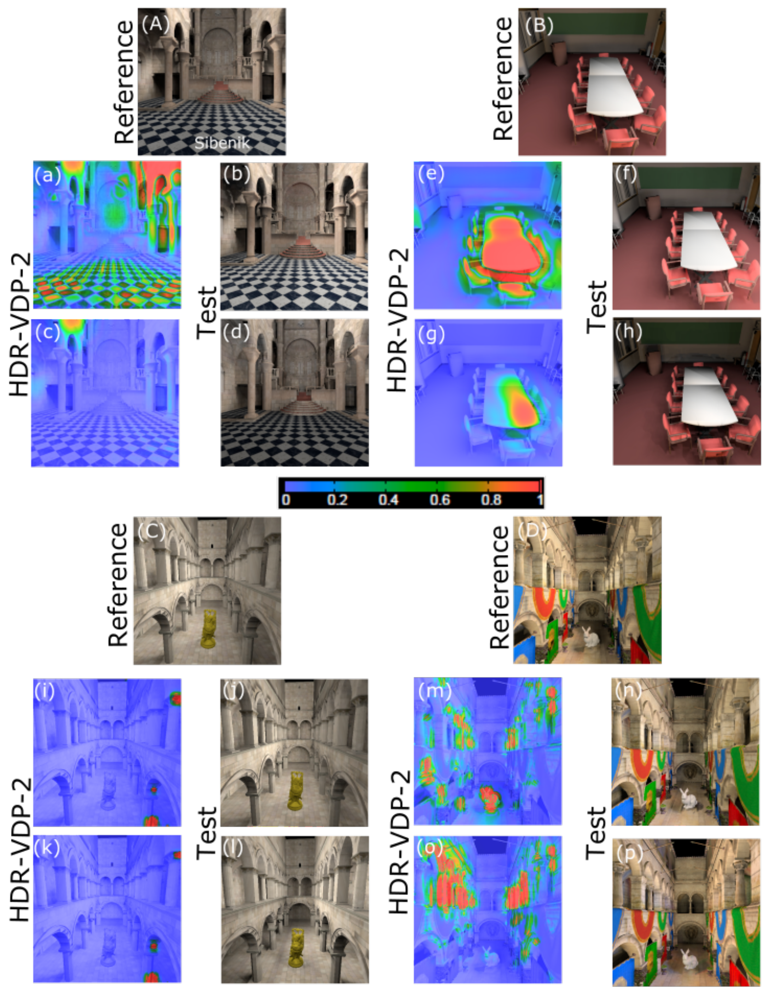


Figure 7.20: HDR-VDP-2 metric between the reference images and those obtained with our methods for the four test scenes. The first column represents the reference image for the Sibenik (A), Conference room (B), Sponza Buddha (C) and Crytek Sponza (D); the second column shows the HDR-VDP-2 images and the third one shows our results. Images (b),(f), (j) and (n) have been generated with our local CDF method. Images (d), (h), (l) and (p) have been generated with our GBG CDF. Images (a), (e), (i) and (m) provide the HDR-VDP-2 metric between the reference images (image (A),(B), (C), (D)) and the images (b), (f), (j) and (n) respectively. Images (c), (g) , (k) and (o) represent the HDR-VDP-2 metric between the reference images (image (A),(B), (C), (D)) and the images (d), (h), (l) and (p) respectively.

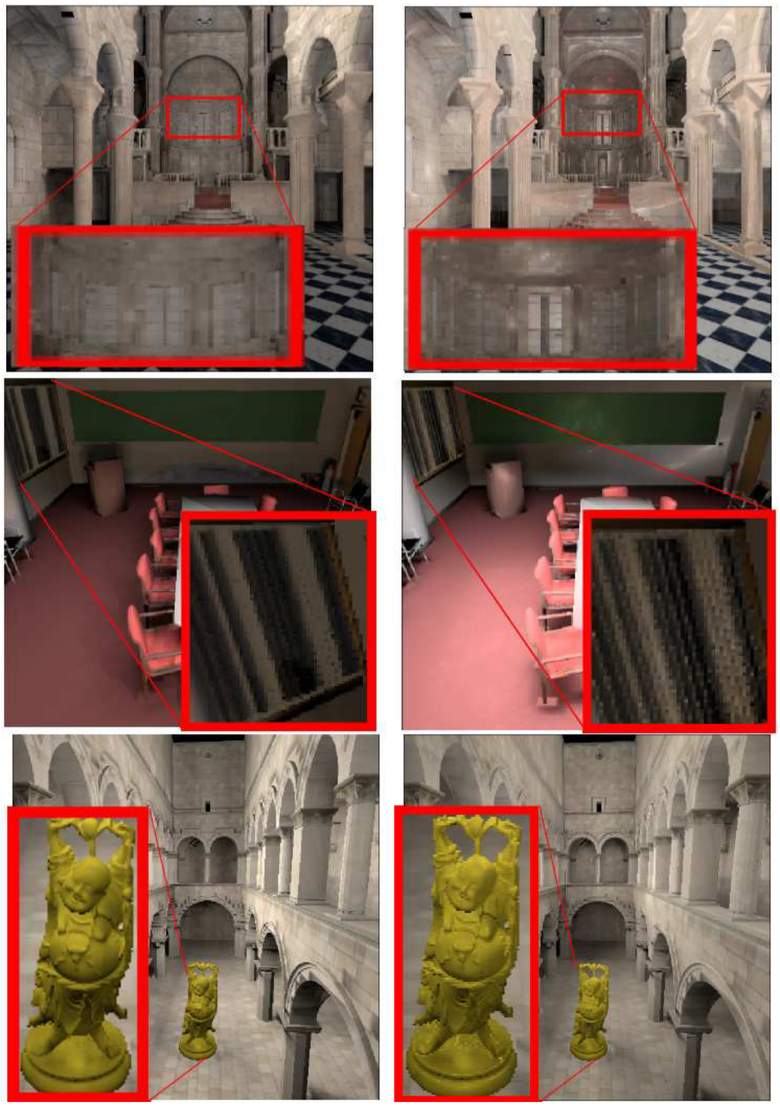


FIGURE 7.21: Comparison of our GBG CDF method without MIS (left column) and our GBG CDF method with MIS (right column).

have used the same number of samples (a sample is: a VPL for the gathering-based CDF approach, and a direction for the MIS) to generate the images of figure 7.21.

7.6 Conclusion

In this chapter, we have demonstrated in detail our inverse transform method for selecting the more contributive VPLs (local CDF and gathering-based global CDF) as well as our MIS method that combined our gathering-based global CDF with the gathering approach. Our methods improve the selection of the more contributive VPLs which used the Dual Paraboloid RSM to store the VPLs. We prove that our algorithms are efficient and that we can automatically evaluate the contribution of the selected VPL with its PDF value. To approximate the visibility term, we used the voxelization which allows accelerating the visibility. We have focused on the diffuse surfaces only.

Chapter 8

Conclusion

In this thesis, we focus on the VPL-based methods to approximate a solution to the global illumination problem. We have presented three VPL-based rendering methods: local CDF, GBG CDF, and MIS. These methods allow improving the selection of the most contributive VPLs stored in a Dual Paraboloid RSM (that we called DPRSM).
To achieve our goal, we have proposed several new algorithms such as:

- **DPRSM:** the VPLs are stored in both PRSMs and a set of VPLs is selected from each PRSM (front or back). In the selection process of VPLs, a PRSM is randomly selected according to the Russian roulette technique.
- **local CDF:** this method consists of subdividing the image plane into 4 parts, choosing gather point GP (points visible from the camera) in each part (stratum) and calculation of associated CDF. Thus computing 4 local CDFs, each associated with its stratum. With this technique, given a GP p belonging to the stratum S, if a VPL is generated from the CDF associated with S to calculate the indirect illumination in p, then a discontinuity is obtained between the images associated with the strata. To avoid this discontinuity, for each GP p, we generate 4 VPLs from the 4 local CDFs and their contributions to p are calculated. To avoid creating a shadow map for each VPL, we proposed to calculate visibility using voxelization.
- **gathering-based global CDF:** in this method, path tracing PT (gathering) is done (it is used to send several rays in each GP), and during the PT we save the contributions sum of the selected VPLs and the number of

GPs to which they contribute, in order to calculate the average contributions. At the end of this PT, some VPLs contributed well to the GPs and others not (which we have called NcVPL for Non-contributive VPL). In this case, it would be necessary to interpolate the contributions of these NcVPLs with a simple average filter. Then calculate a CDF from these average contributions.

- **MIS:** a robust and efficient MIS method that combines our "Gathering-based global CDF" method with the gathering approach, in order to obtain more realistic images.

Our proposed techniques have several advantages by taking into account:

- **Visibility:** we accelerate the visibility step by representing the scene with set of voxels;
- **Locality:** we divide the screen into N regions and we compute the CDF [REH+11] for each region (our local CDF method);
- **Camera importance:** when we compute the CDF, we evaluate the importance of each direction shooted from a gather point (our gathering-based global CDF).

Our methods consider only one bounce indirect illumination because we use a DPRSM placed at the point light source. We use the Russian roulette technique to select one face between the front and the back faces of the DPRSM. All the results are concerned with only indirect illumination. For visualizing our results on LDR displays, our HDR images have been tone-mapped using Reinhard's operator [RSSF02]. Our results show that our local CDF-based method, that divides the screen into N regions, generates good images in term of quality but requires more computing time compared to our other methods. This is why we have proposed our gather-based global CDF to lower the time rendering.
Furthermore, we have applied the HDR-VDP-2 metric to show the perceptual differences between our HDR images and the reference images obtained with the global CDF method [REH+11] with a high number of sample VPLs.

We have focused on the diffuse surfaces only. So, as future work, it would be worth to adapt our algorithms to handle glossy surfaces and caustics. The shown results have been generated from the static scenes. How to extend our methods to dynamic scenes. This is left for future work.

Bibliography

[AK90] James Arvo and David Kirk. Particle transport and image synthesis. *ACM SIGGRAPH Computer Graphics*, 24(4):63–66, 1990.

[ARBJ03] Sameer Agarwal, Ravi Ramamoorthi, Serge Belongie, and Henrik Wann Jensen. Structured importance sampling of environment maps. *ACM Transactions on Graphics (TOG)*, 22(3):605–612, 2003.

[AW$^+$87] John Amanatides, Andrew Woo, et al. A fast voxel traversal algorithm for ray tracing. In *Eurographics*, volume 87, pages 3–10, 1987.

[BAS02] Stefan Brabec, Thomas Annen, and Hans-Peter Seidel. Shadow mapping for hemispherical and omnidirectional light sources. In *Advances in Modelling, Animation and Rendering*, pages 397–407. Springer, 2002.

[BBH13] Tomá Barák, Jirí Bittner, and Vlastimil Havran. Temporally coherent adaptive sampling for imperfect shadow maps. In *Computer Graphics Forum*, volume 32, pages 87–96. Wiley Online Library, 2013.

[BGBB17] Djihane Babahenini, Adrien Gruson, Mohamed Chaouki Babahenini, and Kadi Bouatouch. Efficient inverse transform methods for vpl selection in global illumination. *Multimedia Tools and Applications*, pages 1–25, 2017.

[CG12] Cyril Crassin and Simon Green. Octree-based sparse voxelization using the gpu hardware rasterizer. *OpenGL Insights*, pages 303–318, 2012.

[CNLE09] Cyril Crassin, Fabrice Neyret, Sylvain Lefebvre, and Elmar Eisemann. Gigavoxels: Ray-guided streaming for efficient and detailed voxel rendering. In *Proceedings of the 2009 symposium on Interactive 3D graphics and games*, pages 15–22. ACM, 2009.

[CNS+11] Cyril Crassin, Fabrice Neyret, Miguel Sainz, Simon Green, and Elmar Eisemann. Interactive indirect illumination using voxel cone tracing. In *Computer Graphics Forum*, volume 30, pages 1921–1930. Wiley Online Library, 2011.

[DGR+09] Zhao Dong, Thorsten Grosch, Tobias Ritschel, Jan Kautz, and Hans-Peter Seidel. Real-time indirect illumination with clustered visibility. In *VMV*, pages 187–196, 2009.

[DKL10] Holger Dammertz, Alexander Keller, and Hendrik PA Lensch. Progressive point-light-based global illumination. In *Computer Graphics Forum*, volume 29, pages 2504–2515. Wiley Online Library, 2010.

[DS05] Carsten Dachsbacher and Marc Stamminger. Reflective shadow maps. In *Proceedings of the 2005 symposium on Interactive 3D graphics and games*, pages 203–231. ACM, 2005.

[Dut03] Philip Dutré. Global illumination compendium. *Computer Graphics, Department of Computer Science Katholieke Universiteit Leuven*, 6, 2003.

[GHFP08] Jean-Dominique Gascuel, Nicolas Holzschuch, Gabriel Fournier, and Bernard Peroche. Fast non-linear projections using graphics hardware. In *Proceedings of the 2008 symposium on Interactive 3D graphics and games*, pages 107–114. ACM, 2008.

[Gil05] Walter R Gilks. Markov chain monte carlo. *Encyclopedia of Biostatistics*, 2005.

[GS10] Iliyan Georgiev and Philipp Slusallek. Simple and robust iterative importance sampling of virtual point lights. *Proceedings of Eurographics (short papers)*, 4, 2010.

[Has70] W Keith Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.

[HHZ+14] Wei Hu, Yangyu Huang, Fan Zhang, Guodong Yuan, and Wei Li. Ray tracing via gpu rasterization. *The Visual Computer*, 30(6-8):697–706, 2014.

[HKL16] Peter Hedman, Tero Karras, and Jaakko Lehtinen. Sequential Monte Carlo Instant Radiosity. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. ACM, 2016.

[HKWB09] Miloš Hašan, Jaroslav Křivánek, Bruce Walter, and Kavita Bala. Virtual spherical lights for many-light rendering of glossy scenes. In *ACM Transactions on Graphics (TOG)*, volume 28, page 143. ACM, 2009.

[HL15] Binh-Son Hua and Kok-Lim Low. Guided path tracing using clustered virtual point lights. In *SIGGRAPH Asia 2015 Posters*, page 43. ACM, 2015.

[HP04] Greg Humphreys and Matt Pharr. *Physically Based Rendering*. Morgan Kaufmann, 2004.

[HS98] Wolfgang Heidrich and Hans-Peter Seidel. View-independent environment maps. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, pages 39–ff. ACM, 1998.

[Jen96] Henrik Wann Jensen. Global illumination using photon maps. In *Rendering Techniques 96*, pages 21–30. Springer, 1996.

[Kaj86] James T Kajiya. The rendering equation. In *ACM Siggraph Computer Graphics*, volume 20, pages 143–150. ACM, 1986.

[Kel97] Alexander Keller. Instant radiosity. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 49–56. ACM Press/Addison-Wesley Publishing Co., 1997.

[KL05] Janne Kontkanen and Samuli Laine. Ambient occlusion fields. In *Proceedings of the 2005 symposium on Interactive 3D graphics and games*, pages 41–48. ACM, 2005.

[LW93] Eric P Lafortune and Yves D Willems. Bi-directional path tracing. 1993.

[McG11] Morgan McGuire. Computer graphics archive, August 2011.

[MKRH11] Rafat Mantiuk, Kil Joong Kim, Allan G Rempel, and Wolfgang Heidrich. Hdr-vdp-2: a calibrated visual metric for visibility and quality predictions in all luminance conditions. In *ACM Transactions on Graphics (TOG)*, volume 30, page 40. ACM, 2011.

[MRR+53] Nicholas Metropolis, Arianna W Rosenbluth, Marshall N Rosenbluth, Augusta H Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6):1087–1092, 1953.

[NED11] Jan Novák, Thomas Engelhardt, and Carsten Dachsbacher. Screen-space bias compensation for interactive high-quality global illumination with virtual point lights. In *Symposium on Interactive 3D Graphics and Games*, pages 119–124. ACM, 2011.

[NIDN16] Kosuke Nabata, Kei Iwasaki, Yoshinori Dobashi, and Tomoyuki Nishita. An error estimation framework for many-light rendering. In *Computer Graphics Forum*, volume 35, pages 431–439. Wiley Online Library, 2016.

[OBS+15] Ola Olsson, Markus Billeter, Erik Sintorn, Viktor Kämpe, and Ulf Assarsson. More efficient virtual shadow maps for many lights. *IEEE transactions on visualization and computer graphics*, 21(6):701–713, 2015.

[RDGK12] Tobias Ritschel, Carsten Dachsbacher, Thorsten Grosch, and Jan Kautz. The state of the art in interactive global illumination. In *Computer Graphics Forum*, volume 31, pages 160–188. Wiley Online Library, 2012.

[REH+11] Tobias Ritschel, Elmar Eisemann, Inwoo Ha, James DK Kim, and Hans-Peter Seidel. Making imperfect shadow maps view-adaptive: High-quality global illumination in large dynamic scenes. In *Computer Graphics Forum*, volume 30, pages 2258–2269. Wiley Online Library, 2011.

[RGK+08] Tobias Ritschel, Thorsten Grosch, Min H Kim, H-P Seidel, Carsten Dachsbacher, and Jan Kautz. Imperfect shadow maps for efficient computation of indirect illumination. *ACM Transactions on Graphics (TOG)*, 27(5):129, 2008.

[RSSF02] Erik Reinhard, Michael Stark, Peter Shirley, and James Ferwerda. Photographic tone reproduction for digital images. *ACM Transactions on Graphics (TOG)*, 21(3):267–276, 2002.

[Seg07] Benjamin Segovia. Interactive light transport with virtual point lights. *These de doctorat en informatique, Université Lyon*, 1, 2007.

[SHD15] Florian Simon, Johannes Hanika, and Carsten Dachsbacher. Rich-vpls for improving the versatility of many-light methods. In *Computer Graphics Forum*, volume 34, pages 575–584. Wiley Online Library, 2015.

[SRS14] Masamichi Sugihara, Randall Rauwendaal, and Marco Salvi. Layered reflective shadow maps for voxel-based indirect illumination. In *High Performance Graphics*, pages 117–125, 2014.

[THGM11] Sinje Thiedemann, Niklas Henrich, Thorsten Grosch, and Stefan Müller. Voxel-based global illumination. In *Symposium on Interactive 3D Graphics and Games*, pages 103–110. ACM, 2011.

[Vea97] Eric Veach. *Robust monte carlo methods for light transport simulation.* PhD thesis, Stanford University, 1997.

[WABG06] Bruce Walter, Adam Arbree, Kavita Bala, and Donald P Greenberg. Multidimensional lightcuts. In *ACM Transactions on Graphics (TOG)*, volume 25, pages 1081–1088. ACM, 2006.

[WFA$^+$05] Bruce Walter, Sebastian Fernandez, Adam Arbree, Kavita Bala, Michael Donikian, and Donald P Greenberg. Lightcuts: a scalable approach to illumination. In *ACM Transactions on Graphics (TOG)*, volume 24, pages 1098–1107. ACM, 2005.

[Wil78] Lance Williams. Casting curved shadows on curved surfaces. In *ACM Siggraph Computer Graphics*, volume 12, pages 270–274. ACM, 1978.

[Wil83] Lance Williams. Pyramidal parametrics. In *Acm siggraph computer graphics*, volume 17, pages 1–11. ACM, 1983.