VOXEL METHOD
FOR REALISTIC RENDERING

By

Łukasz Piwowar

SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
AT
UNIVERSITY OF WROCŁAW
WROCŁAW, POLAND
SEPTEMBER 2008

UNIVERSITY OF WROCŁAW
DEPARTMENT OF
MATHEMATICS AND COMPUTER SCIENCE

The undersigned hereby certify that they have read and recommend to the Faculty of Graduate Studies for acceptance a thesis entitled "**Voxel method for realistic rendering**" by **Łukasz Piwowar** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy**.

Dated: September 2008

External Examiner: _____
Examiner

Research Supervisor: _____
Rémy Malgouyres

Examing Committee: _____
first reader

_____
second reader

UNIVERSITY OF WROCłAW

Date: **September 2008**

Author: **Łukasz Piwowar**

Title: **Voxel method for realistic rendering**

Department: **Mathematics and Computer Science**

Degree: **Ph.D.** Convocation: **October** Year: **2008**

Permission is herewith granted to University of Wrocław to circulate and to copy for non-commercial purposes, at its discretion, the above title upon the request of individuals or institutions.

Signature of Author

For my parents who gave me the rod - not the fish, and for my wife.

Table of Contents

Table of Contents **v**

Abstract **vii**

Acknowledgements **ix**

1 Introduction **1**

2 Global Illumination **4**
2.1 Radiometric quantities 5
2.2 Monte Carlo methods 7
2.2.1 Path tracing 7
2.2.2 Bi-directional path tracing 8
2.2.3 Photon Mapping 8
2.3 Classic Radiosity Method 11
2.3.1 Solid angle 11
2.3.2 Continuous illumination equation 12
2.3.3 Radiosity 12
2.3.4 Progressive Refinement Radiosity 14

3 Voxel Global Illumination **15**
3.1 Voxel Approximation 15
3.2 Discretization 17
3.3 Optimal Complexity for Visibility 19
3.4 Shooting Energy from Light Source Voxels, discrete version 21
3.5 Shooting Energy from Light Sources, raytracing (continuous) version 22
3.6 Cache method 24

4 Rendering **26**
4.1 Direct voxel solution view 26
4.2 Texture based method 27
4.2.1 Storing textures 30
4.3 Instant Radiosity 32
4.3.1 Implementation details 32
4.4 Lightcuts 38
4.4.1 Implementation details 39
4.5 General rendering process 41

5 Results **43**
5.1 Experimental results 43
5.1.1 Correctness test 44
5.1.2 Parameters adjusting 46

Bibliography **53**

Abstract

In [22] (Malgouyres, 2002) and [5] (Chatelier and Malgouyres, 2006), a new approach to global illumination is initiated, and a discretization of the diffuse illumination based on voxel approximation of surfaces by voxels is proposed. What is interesting about this method is that visibility is determined in linear time with respect to the amount of rays. Moreover, it directly provides a voxel-based irradiance lookup octree. However, the method presented in [5] has two weaknesses. First, solid angle sampling is the same for each bounce, and in particular, direction sampling from light sources is insufficient while the cost of direction sampling after one or two bounces is very expensive. Second, no reconstruction process is presented and direct display of the voxel solution requires many voxels, which also increases the runtime. In this thesis, we address these two shortcomings by proposing an iterated cached coarse global illumination solution, in which direction sampling decreases after each bounce, followed by a reconstruction phase based on lightcuts [32]. It results in a competitive accurate multi-bounce global illumination method with a voxel irradiance cache octree.

In [33] (Zrour et al.), a distributed memory version of the propagation method is proposed for scenes with high number of voxels, which allows to scale a method using several threads/computers. The results of our thesis open prospects of a GPU implementation of these parallel algorithms.

We describe a development process of this new global illumination method based on voxels discretization.

We start from direct visualization, through texture based method to the combining linear voxel method with instant radiosity and finally lightcuts method to obtain fast and accurate solution to light propagation.

The method is extended and its overall time complexity has been improved. Many real world scene tests have been made to prove its practical usage.

Acknowledgements

I would like to thank both my advisors Rémy Malgouyres and Krzysztof Loryś for friendly and patient advising, Andrzej Łukaszewski for valuable graphics conversations and Jurek Tomasik for valuable non graphics conversations.

Wrocław, Poland
September 1, 2008

Łukasz Piwowar

Chapter 1

Introduction

Computer graphics is one of the most growing and interesting research fields in Computer Science. It covers a broad range of topics including image-based modeling and rendering, photo realistic and non-photo realistic image synthesis, curve and surface modeling, interactive 3D user interfaces, image editing, surface reconstruction and modeling, and many more. We focus on its one aspect - photo realistic rendering. Using voxel global illumination combined with other modern methods we try to add a small but a significant improvement to that area.

Comprehensive multi-bounce methods for global illumination have been extensively studied, and finding the right balance of speed vs accuracy is not an easy task. The most widely used approach consists in the step of coarse computation of a global illumination solution followed by a step of reconstruction by gathering the light by ray-tracing to provide good quality images.

The photon mapping method described in [13] [14] [15] (Jensen 1996, Jensen 1997, Jensen 2001) is a good example. This method traces random rays from light sources, and at each intersection, traces a new random ray, the ray walk is terminated by Russian roulette with a probability that depends on the BRDF. Then, a second

phase consists in a viewpoint dependant ray-tracing for computing good quality images. Instant radiosity (Keller, 1997) was the first of such methods ever introduced. Other methods for final gather steps have been used by [11] (Granier and Drettakis, 2004), and [2] (Arikan et al., 2005). In all these methods, determining visibility by ray-object intersection is a very important time cost factor. Attempts at getting rid of visibility problems altogether have been made by [8] (Dachsbacher et al., 2007), however at a cost of increased memory and reintroducing hierarchical radiosity drawbacks of difficult refinement and meshing [27] (Sillion and Puech, 1994). For accelerating the gather step, photon splatting [20] (Lavignotte and Paulin, 2003), [7] (Dachsbacher and Stamminger, 2006) can be used, but it neglects some occlusions for indirect light and uses rough approximations for speedup.

In [22] (Malgouyres, 2002) and [5] (Chatelier and Malgouyres, 2006), a completely new approach to global illumination is initiated, and a discretization of the diffuse illumination based on voxel approximation of surfaces by voxels is proposed. What is interesting about this method is that visibility is determined in linear time with respect to the amount of rays. Moreover, it directly provides a voxel-based irradiance lookup octree. However, the method presented in [5] has two weaknesses. First, solid angle sampling is the same for each bounce, and in particular, direction sampling from light sources is insufficient while the cost of direction sampling after one or two bounces is very expensive. Second, no reconstruction process is presented and direct display of the voxel solution requires many voxels, which also increases the runtime. In this thesis, we address these two shortcomings by proposing an iterated cached coarse global illumination solution, in which direction sampling decreases after each bounce, followed by a reconstruction phase based on lightcuts [32] (Walter et al.,

2005). It results in a competitive accurate multi-bounce global illumination method with a voxel irradiance cache octree.

In [33] (Zrour et al.), a distributed memory version of the propagation method is proposed for scenes with high number of voxels, which allows to scale a method using several threads/computers. The results of our thesis open prospects of a GPU implementation of these parallel algorithms.

Thus, in this thesis, we propose a competitive alternative to the irradiance-caching and the photon mapping methods to compute indirect lighting. We then describe the process of the method's development, from the theoretical idea of light propagation over voxels and direct visualization (at that stage, the method was unpractical because of the runtime and memory cost), through lightmaps pre-computation (which allows realtime walkthroughs - although with low quality or high storage cost), instant radiosity that provides good quality images with still high runtime, to the lightcuts solution with a low number of shadow rays, which is fully comparable to current state of the art methods in the area. Perspectives set out in conclusion promise a useful method for fast comprehensive multi-bounce global illumination in complex scenes for photo-realistic and physically based rendering.

In the first two chapters we give an overview of main global illumination methods and basic notions. From chapter 3, we describe in detail the voxel-based global illumination method and its extensions. In Chapter 4, we characterize a reconstruction step: how to visualize the computed voxel solution to obtain good quality images. We also summarize the general rendering process. In Chapter 5, we present results and comparison with other methods, and we try to check influence of parameters adjustment on the final image. Finally, we set-out perspectives for future work.

Chapter 2

Global Illumination

The main problem of physically-based rendering is to compute interactions of the light with the three dimensional scene. We need to do that if we want to produce an image indistinguishable from a photograph.

The input data that is commonly used in computer graphics is a scene description that includes camera, triangle meshes and a light setup. The rendering equation introduced by James Kajiya [1] in year 1986, describes light transport from one surface point to another as the sum of emitted radiance and reflected radiance. Global illumination methods attempt to provide an efficient and accurate numerical solution to this equation. Global illumination is able to simulate such effects as: color bleeding (strong color influence mainly reflected light from diffuse surfaces), caustics (focused light reflected of transmitted by one or more specular surfaces). Other possible subtle effects are subsurface scattering (useful for skin rendering) and volumetric effects (fog, dust and other participating media).

Global illumination equation [27] is:

$$\underbrace{L(x,\overrightarrow{\omega_o})}_{\text{total radiance}} = \underbrace{L_e(x,\overrightarrow{\omega_o})}_{\text{emitted radiance}} + \underbrace{\int_\Omega p(x,\overrightarrow{\omega_o},\overrightarrow{\omega_i})L_i(x,\overrightarrow{\omega_i})\cos\Theta_i d\sigma_{\overrightarrow{\omega_i}}}_{\text{reflected radiance}} \tag{2.0.1}$$

This equation describes interactions of light with the surfaces. An algorithm which solves this problem is expected to model all types of light paths that is $L[D|S]^*E$, where L, D, S, E mean respectively light source, diffuse reflection or refraction, specular reflection or refraction and eye.

2.1 Radiometric quantities

Basic notions of the radiosity:

- by D we denote a set of discrete directions in space
- *Radiance* is the amount of energy travelling at some point in a specified direction, per unit time, per unit area perpendicular to the direction of travel, per unit solid angle $L(x,\omega)$ is radiance at x in direction ω expressed in units: $\frac{W}{m^2*r}$ that is power per unit area per unit solid angle, and θ,ϕ are proper directions in spherical coordinates.
- $Irradiance$ is the total power arriving at a surface, per unit area on the surface. $I(x,\omega)$ is irradiance at x in direction ω
  $I(x,\omega) = d\Phi/dA = L(x,\omega)\cos\theta d\omega$
- *Photons* are packets of energy which travel in a straight line in vacuum with velocity c (300.000 m/s), in computer graphics it mostly defines the ray and its associated radiance.

- The flux is the rate of energy flowing through a surface per unit time (watts), also a flow of photons per unit time.

- *Radiosity* (B) is the total power leaving a point on a surface, per unit area on the surface.
  $B(x) = \int_\Omega L_0(x, \theta, \phi) \cos\theta d\omega$

- $BRDF$ - Bidirectional reflectance distribution function - is a 4-dimensional function that defines how light is reflected from the surface.
  $\rho_{bd}(x, \theta_0, \phi_0, \theta_i, \phi_i) = \frac{L_0(x,\theta_o,\phi_o)}{L_i(x,\theta_i,\phi_i)\cos\theta_i d\omega}$

2.2 Monte Carlo methods

The general Monte Carlo method [23] approximates the value of the function, by randomly sampling its domain, and aggregating results. It is like guessing the value, based on the known values in random samples. The name is a reference to a famous casino in Monaco where roulette was treated as random number generator.

In computer graphics Monte Carlo method is mostly used as a sampling technique for estimating high-dimensional integrals over complex domains.

Sample of the pixel estimate:

$$q = \frac{1}{N} \sum_{i=1}^{N} f_k \tag{2.2.1}$$

2.2.1 Path tracing

Pure Monte Carlo path tracing [1] is one of the simplest method of global illumination. For each pixel of the image, a ray is sent into the scene. When it hits a triangle it stores its color and its emission, then bounces off and continues on its path. Finally all stored information is sent back and affects a final value. At each step light that is returned to the pixel along the path is scaled by certain factors like reflection factor and color of the material. We repeat whole process in each pixel, adding some randomness and averaging results, then image is slowly converged to the solution.

This strategy is simple, but usually blind for light emitters. Unless a ray hits an emitter on its path, it will return only zero.

For outdoor scenes most rays will eventually hit the sky (probably after several bounces), and find non zero values. For indoor scenes with small emitters many rays may be needed to find them, which makes the method slow.

Emitter sampling may be a simple improvement of this method. We know the position of all emitters from scene description, and we can get profit from that information. At each step we send an extra ray to some random emitter. We have to be careful not to sum incoming light twice, thus, we do not consider direct emitters hits any more.

Monte Carlo path tracing is considered unbiased, which means that it does not introduce consistent error (the only error is in the effects of the randomness).

2.2.2 Bi-directional path tracing

Bi-directional path tracing is a hybrid two-pass algorithm introduced by Lafortune [26]. The particles are traced simulatively from a selected light source and the viewpoint (first eye-pixel intersection). Then shadow rays are used to connect each intersection point in the light path with each intersection point in the eye path, their contributions are added to the power of the computed pixel. Bidirectional path tracing is an effective and unbiased rendering algorithm for many kinds of indoor scenes, with or without strong indirect lighting. The main weakness of the algorithm is selection of the light sources which may not contribute rendered portion of the scene. It is not suitable for outdoor scenes, or scenes where light and eye paths cannot be connected effectively (for example scenes with separated geometry like dark room near to the bright one, separated by ajar doors).

2.2.3 Photon Mapping

Russian Roulette [18] is a standard Monte Carlo technique. It can help us to absorb photons at given probability, and reduce efforts spent on evaluating unimportant ray paths. Lets say that $r \in [0, 1]$ is a random variable

$p \in [0, 1]$ is probability that another radiance estimate L_n is made:

$$L_n = \begin{cases} r < p \text{ then } \frac{L}{p} \\ \text{otherwise } 0 \end{cases}$$

Photon mapping [16] is a biased Monte Carlo technique for light transport. Light is transported along a large number of paths from the light sources in the scene (photons), and a data structure is built to record the distribution of illumination (the photon map). The photons might hit different kinds of surfaces in the scene, specular, diffuse, or a mixed one. When the photon hits a diffuse surface, the photon's energy and incident direction are computed and stored in the kd-tree [4] [3] structure. If the photon hits a purely specular surface, the photon is reflected. When it hits a partially diffuse specular surface, Russian roulette method is used to resolve if the photon should be stored (and absorbed) or reflected. While storing the photon, the energy, intersection point, and incoming direction, of the photon are stored in a photon map. During scene rendering, the photon map is used to estimate the amount of illumination at any point in the scene.

Using photon maps directly at rendering time to approximate all illumination at the point being shaded, makes resulting image blotchy, thus, it can be only used in case of quick approximation to the lighting in the scene. For final rendering we should use final gather technique.

Final gather is a caching technique to improve approximation of the local irradiance. While computing the color of a primary ray instead of taking the direct photon map value, we shoot series of rays at random angles into the scene to calculate the averaged light energy. It allows us to compute global illumination effects (like color bleeding) efficiently. To compute global illumination rendering using n bounces, we

start by emitting photons from light sources for the $n-1$ bounces. Then, the last bounce is simulated by tracing a beam of rays from the surface (final gathering), at each ray-surface intersection the photon map is used to compute the radiance. The main benefit is that we compute only one bounce, and add it to previously cached information about $n-1$ bounces from the photon map. Final gathering for good results is using typically 200 to 5000 rays per gather [15].

Photon mapping renders stored photons using classic ray tracing, but with techniques similar to Monte Carlo ray tracing. Photons propagate $flux$ and rays gather radiance. If we intersect a specular object after an eye path, we use shadow rays but also send recursively reflected ray, in case of diffuse surface we compute radiance in the eye direction, from the stored photons.

To compute the reflected *radiance* at a point x in the outgoing direction ω, we do the following:
The reflected radiance:

$$L_r(x,\omega) = \int_{\Omega_x} f_r(x,\omega',\omega)\frac{d^2\phi(x,\omega')}{dA_i} \tag{2.2.2}$$

is approximated as:

$$L_r(x,\omega) \approx \sum_{p=1}^{n} f_r(x,\omega',\omega)\frac{\Delta\phi_p(x,\omega')}{\pi r^2} \tag{2.2.3}$$

where r is the distance to the farthest photon p, ω' is direction of incoming radiance, ω is direction of outgoing radiance, f_r is bidirectional reflectance distribution function (BRDF), dA_i is a differential solid angle and πr^2 is its approximation.

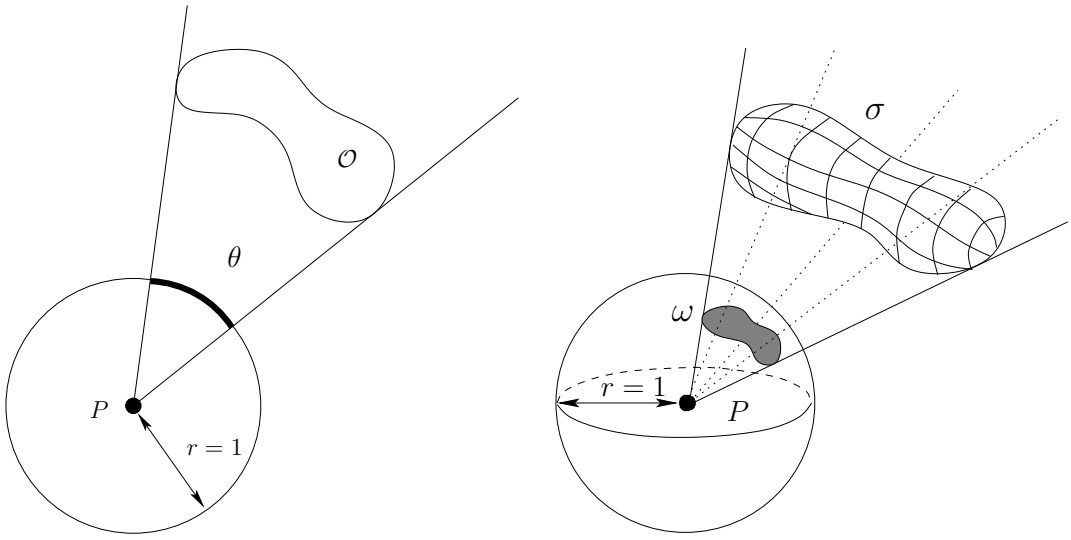
Because photons are stored in a balanced kd-tree (structure that is both compact and efficient) the total computation time for photon mapping is:

$c_1 * V * \log_2 V + c_2 * F * K + c_3 * F * \log_2 V$, where c_1 is a constant factor for sorting photons during the construction of the tree, c_2 a the factor for processing single found photon, c_3 is a factor required during searching over the kd-tree, V is the total number of photons, F is the number of searches for K nearest photons.

The time it takes to locate F photons in a tree with V photons is $\mathcal{O}(F \log_2(V))$. In practice the search is much more efficient since the photons are located in the same parts of the tree. [13]

2.3 Classic Radiosity Method

2.3.1 Solid angle



(a) 2d (b) 3d

Figure 2.1: Solid angle

A is a solid angle which is a 3D generalization of the angle between two lines. A solid angle is used to measure the portion of space occupied by an object as seen from a point [27].

2.3.2 Continuous illumination equation

Let x and y denote two points in a scene. Let $B(x)$ denote the radiosity at the point x (i.e the amount of the energy leaving x per area). Let $V(x, y)$ denote the visibility function, defined to be equal to 1 if y is visible from x in the scene and 0 otherwise. Let r denote the distance between two points x and y.

Let $\theta(x, y)$ denote the angle between the vector $x\vec{y}$ and the normal vector $\vec{n}$ at the point x. If that angle is less than or equal to $\frac{\pi}{2}$, there is no interaction between x and y, and we define it by convention $\cos\theta$ to 0.

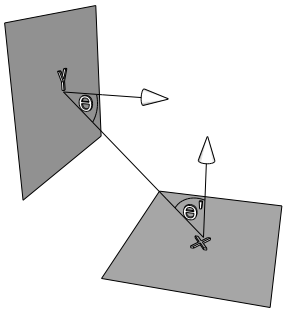


Figure 2.2: Geometric relation

The continuous diffuse illumination equation is as follows:

$$B(x) = E(x) + \rho(x) \int B(y) \frac{\cos\,\theta\,\cos\,\theta'}{\pi\, r^2} V(x, y)\, dy \tag{2.3.1}$$

The continuous diffuse illumination equation in the general case is not analytically solvable. Numerical computations are used to get an approximation. We use a discretization method from [22] to solve the equation.

2.3.3 Radiosity

The radiosity method computes light paths of the form $LD * E$, paths which start at a light source and make multiple diffuse bounces before reaching the eye.

A classical approach to the radiosity method is to compute radiosity values in patches and then interpolate them during visualization stage by Gouraud [10] shading. A patch is a piece of scene, usually a triangle or a quad with area not larger than a fixed limit. Subdividing the scene into patches is the first step of the classic radiosity.

The form factor between two patches P_i and P_j is:

$$F_{ij} = \frac{1}{A_i} \int_{x \in P_i} \int_{y \in P_j} \frac{\cos\theta \cos\theta'}{\pi r^2} V(x,y) dxdy \tag{2.3.2}$$

F_{ij} is the proportion of the total power leaving patch P_i that is received by patch P_j. We can use it to compute the energy arriving at i. The form factor depends only on the geometry, and has the following properties:

1. reciprocity: $A_i F_{ij} = A_j F_{ji}$,
2. additivity: $A_{i(j \cup k)} F_{ij} = F_{ij} + F_{ik}$
3. sum to 1: $\sum_{j=1}^{N} F_{ij} = 1$

The scene is divided into a set of patches. All form factors are computed (this part is view-independent), there are many ways to compute that. We compute the matrix M (see 2.3.4), and solve linear system in form of:

$$B_i = E_i + \rho_i \sum_{j}^{n} B_j F_{ij}, i = 1, 2, ..., n \tag{2.3.3}$$

where B_i is the radiosity, E_i is the emission, and p_i is the reflectivity factor of patch i.

F_{ij} is the form factor from the patch i to the patch j which represents the fraction of energy leaving patch i and reaching the patch j ($F_{ii} = 0$).

The linear system above can be rewritten in a matrical form:

$$\underbrace{\begin{pmatrix} 1-\rho_1 F_{11} & -\rho_1 F_{12} & \dots & -\rho_1 F_{1n} \\ -\rho_2 F_{21} & 1-\rho_2 F_{22} & \dots & -\rho_2 F_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ -\rho_n F_{n1} & -\rho_n F_{n2} & \dots & 1-\rho_n F_{nn} \end{pmatrix}}_{\text{base matrix M}} \begin{pmatrix} B_1 \\ B_2 \\ \dots \\ B_n \end{pmatrix} \begin{pmatrix} E_1 \\ E_2 \\ \dots \\ E_n \end{pmatrix} \tag{2.3.4}$$

Directly solving the linear system by gaussian elimination is expensive in time and memory, time complexity is $\mathcal{O}(n^3)$.

The linear system is sparse and can be solved by Gauss-Seidel iterative method [30]:

$$B_i(k) = E_i + \rho \sum B_j(k-1) F_{i,j} \tag{2.3.5}$$

Gauss-Seidel iterations are series of light energy gathering processes. The radiosity of the current row patch is gathered and updated based on the current estimate of the radiosity of every other patch in the environment. This classic radiosity method is referred to as the gathering method.

2.3.4 Progressive Refinement Radiosity

The basic approach of progressive refinement radiosity [6] is to identify the brightest patch in the scene and shoot (distribute) its energy to the other patches visible from it. This is equivalent to computing only those rows of the form-factor matrix A that correspond to the brightest patches. In practice, this approach results in a fast convergence to the solution without computing all the rows of the matrix. The main advantage is that we do not need to compute all A_{ij}. This radiosity method is often referred to as the shooting method.

Chapter 3

Voxel Global Illumination

3.1 Voxel Approximation

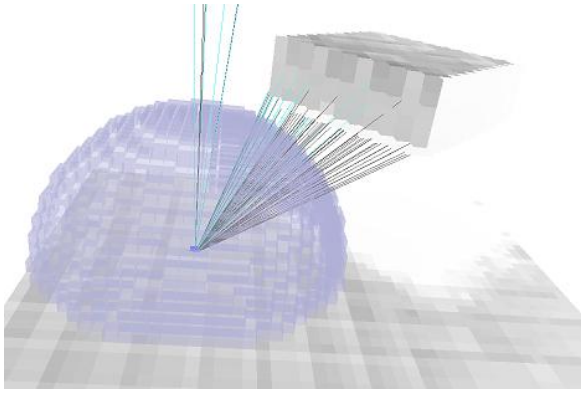


Figure 3.1: Voxels half sphere

By a change of variable, using the correspondence between visible points from the point x and points on a sphere S centered at x, and the continuous diffuse illumination equation is equivalent to

$$B(x) = E(x) + \rho(x) \int_S B(y) \frac{\cos\,\theta}{\pi} \mathrm{d}\overrightarrow{\sigma}$$

In [22], we simply discretize this equation by approximating the surfaces by a discrete surface, and also sampling the sphere to estimate the integral, by introducing

a finite set of directions.

$$B(x) = E(x) + \rho(x) \sum_{\overrightarrow{\sigma} \in D} B(I(x, \overrightarrow{\sigma})) \frac{\cos\theta(x, I(x, \overrightarrow{\sigma}))}{\pi} \Delta\Omega(\overrightarrow{\sigma}) \tag{3.1.1}$$

where:

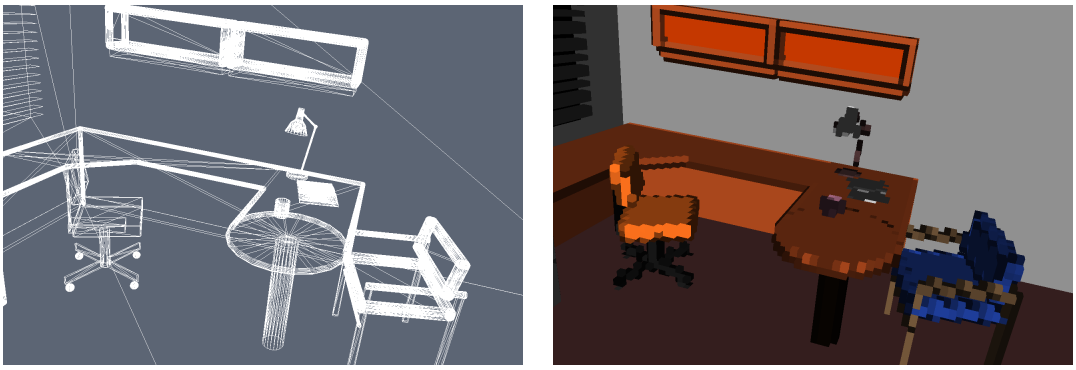
- x is now a voxel (unit or fixed size cube);
- D is a set of discrete directions in space;
- $I(x, \overrightarrow{\sigma})$ is the first point y viewed from x in the direction of $\overrightarrow{\sigma}$ (as in a ray-object intersection);
- to quantify how much of an object is seen from a point, the term $\Delta\Omega(\overrightarrow{\sigma})$ is the fraction of a solid angle associated to the direction $\overrightarrow{\sigma}$.

This discrete equation is a large linear system with unknowns $B(x)$ but the $I(x, \overrightarrow{\sigma})$ depends on visibility and occlusions.

The way to obtain a discrete set of voxels from a continuous surface is described in [21] for an implicit surface, and in [22] for a mesh.

In [5], a solution of the linear system is obtained with optimal complexity. We use a slightly improved version of this method, by using bucket (or radix) sort (complexity $\mathcal{O}(n)$) instead of quicksort. Thus, final complexity is $\mathcal{O}(N \times I \times D)$, where N is the number of voxels of the discrete surface, D is the number of directions used for sampling the sphere, and I is the number of iteration, or number of successive reflections used for indirect lighting. We emphasize that **the complexity is linear (with a small coefficient) with respect to the total number of rays traced**. We now explain these existing developments in detail.

3.2 Discretization



(a) original scene (b) discretized voxel set

Figure 3.2: Discretization process

Discretization consists in approximating meshes by a set of voxels. We project each triangle into XY,YZ and XZ planes, but the order of the projections depends on the values of the triangle normal vector coordinates. For each triangle, we proceed with the following steps:

- compute the coordinates of the triangle normal vector (x, y, z) to choose the plane orthogonal projection that gives us maximal area of the projected triangle.
- sort the three coordinate values (x, y, z), we do that in constant time
- start the discretization from the projection of minimal absolute coordinate value (x, y, z) to gain maximal area of the projected triangle. Additionally, we project only in case that triangle coordinates are less than or equal to 0.25. That value was selected experimentally by testing many 3D objects. In a previous solution this value was equal to $1/\sqrt{3}$. But that seems to be too much (there were small holes in discrete objects).

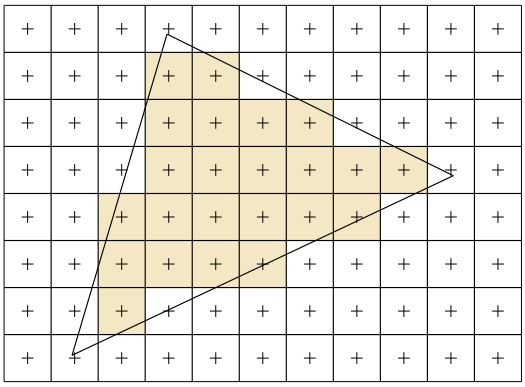


Figure 3.3: Single triangle discretization process

Using a triangle rasterization algorithm, we compute the position (x, y, z) of each voxel. We insert each voxel only in case its space position (x, y, z) is still empty. Previous sorting of normal vector coordinates absolute values ensures that voxels from the biggest projection will take precedence over the voxels from the next projections.

Projected triangle rasterization algorithm works in 3 steps:

- compute the bounding box of the triangle projected for specified plane XY,YZ or XZ (minimal rectangle with integer coordinates that covers all triangle points).
- for each integer couple of points (x, y) in the bounding box area, we compute the intersection of the ray (perpendicular to the triangle) with the proper plane (that include the triangle).
- compute the position (x, y, z) of each voxel and store an interpolated normal vector at each voxel.

3.3 Optimal Complexity for Visibility

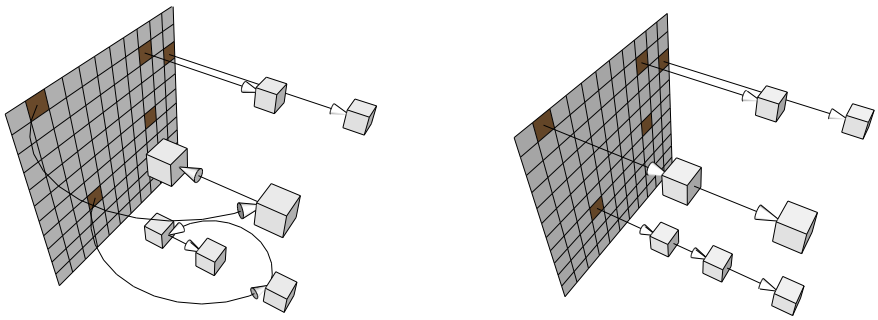
To solve Equation (3.1.1), a converging iterative method similar to the one used with classical patch-based radiosity can be used: it usually relies on Jacobi or Gauss-Seidel relaxation. If we consider the linear equation under its form $B = E + M.B$, where B is a vector of elements and M a matrix of factors, some properties of M ensure the sequence $B_{n+1} = E + M.B_n$ converges toward a limit, which is a solution to the discrete linear system. Roughly speaking, this is a transcription of light gathering, each iteration going a step further in light re-emission. Convergence is expected since light is progressively absorbed. Technically, each iteration consists in propagating packets of energy between mutually visible voxels.

Given a direction vector $(a, b, c) \in \mathbb{Z}^3$ with $a \geq b \geq c$, a notion of a $3D$ line has been proposed [9], as the set of points $(x, y, z) \in \mathbb{Z}^3$ such as

$$\mu \leq cx - az < \mu + \omega \text{ and } \mu' \leq bx - ay < \mu' + \omega'$$

where $\mu, \mu', \omega, \omega'$ are integers. Other cases can be deduced by symmetry. It is noteworthy that under this definition, a $3D$ discrete line represents the intersection between two discrete planes, each being the orthogonal extrusion of a $2D$ discrete line included in one of the coordinate planes. The connectivity is related to the thickness ω and ω' of these $2D$ lines. If the two $2D$ projections are naïve (resp. standard), the $3D$ line is called naïve (resp. standard).

Let us denote by $\mathbb{Z}^3_*$ the set $\mathbb{Z}^3 \backslash \{(0, 0, 0)\}$. Given an integer vector $\overrightarrow{v} \in \mathbb{Z}^3_*$, the set $\mathbb{Z}^3$ can be partitioned into $3D$ discrete lines, whose direction vector is $\overrightarrow{v}$ (see Figure 3.5). Moreover, given a voxel $x \in \mathbb{Z}^3$, finding out which $3D$ discrete line in the partition the point x belongs to can be done in constant time.



(a) Unsorted lists of voxels (b) Sorted lists of voxels

Figure 3.4: The lexicographic on voxels coincides with linear order on rays

Now, solving our linear system given by Equation (3.1.1) by the Gauss-Seidel relaxation amounts to transferring energy from one voxel x to the first voxel y visible from x in a direction σ, for some finite set of sample directions σ. We can assume w.l.o.g that σ has integer coordinates. Now, if we consider some fixed direction σ and a partition of the voxel space $\mathbb{Z}^3$ into discrete lines parallel to σ, then the voxels of the discretized surface (say mesh or implicit surface) that lie on the same discrete line can be arranged in an ordered list. In this ordered list, the first visible voxel is the next voxel on the list (see Figure 3.4). So, once the lists are sorted, we can propagate the energy in linear time $\mathcal{O}(N)$.

Now, the idea is that by going over the set of all surface voxels in a lexicographic order (lexicographic orders are precomputed by radix sort), we can build all the lists in linear time $\mathcal{O}(N)$ (see Figure 3.5). We do this for each of the D sample directions and for each of the I Gauss-Seidel iteration, and we get a numerical solution of Equation (3.1.1) in time $\mathcal{O}(N \times I \times D)$.

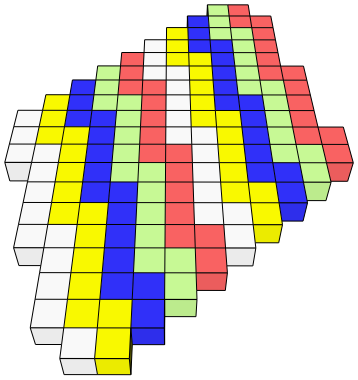


Figure 3.5: Lists of voxels obtained by intersection with discrete lines

3.4 Shooting Energy from Light Source Voxels, discrete version

The optimal complexity method described above works very well, but there is a drawback: it does not allow to do different sampling for directions for different voxels. Now, light source voxels (non-zero emittance) emit much more power than voxels reflecting indirect light. So, we should send more rays, or more discrete lines from light source voxels.

Here we present a method, combining shooting energy from light sources and optimal complexity propagation, which allows us to use a finer sampling for rays from light sources. Since there are few light source voxels, we retain the advantage of optimal complexity for indirect lighting, while avoiding rough approximation by insufficient sampling of the energy emitted by light sources.

In shooting energy from light sources, we must preserve the quantitative accuracy of the method. To do so, voxels emit some energy in proportion to their contribution to surface area, as defined below.

Each connected component χ of the continuous surface (e.g mesh) separates the

space into two connected components, C and $\overline{C}$ (Jordan theorem), where $\overline{C}$ is the component which contains the viewpoint. We consider O the set of all the voxels $(i, j, k) \in Z^3$ which belong to C (i.e. voxels inside an object or outside the scene). The discretization of the surface is the boundary F of O, which is the set of all voxels in O which are $6-$adjacent to a voxel in the complement $\overline{O}$ of O.

For a voxel $x \in F$, the element of area $\Delta A(x)$ is the sum, for all $6-$neighbours y of x in $\overline{O}$, of the dot product $\overrightarrow{xy}.\overrightarrow{N}$, where $\overrightarrow{N}$ is the normal vector at the point x.

$$\Delta A(x) = \sum_{y \in N_6(x) \cap \overline{O}} \overrightarrow{xy}.\overrightarrow{N}$$

We take into consideration a sample of directions from a light source voxel y. Each sample direction $\overrightarrow{\sigma}$ is associated with a portion of solid angle $\Delta\Omega(\overrightarrow{\sigma})$. Through this portion of solid angle, the voxel x emits a power equal to :

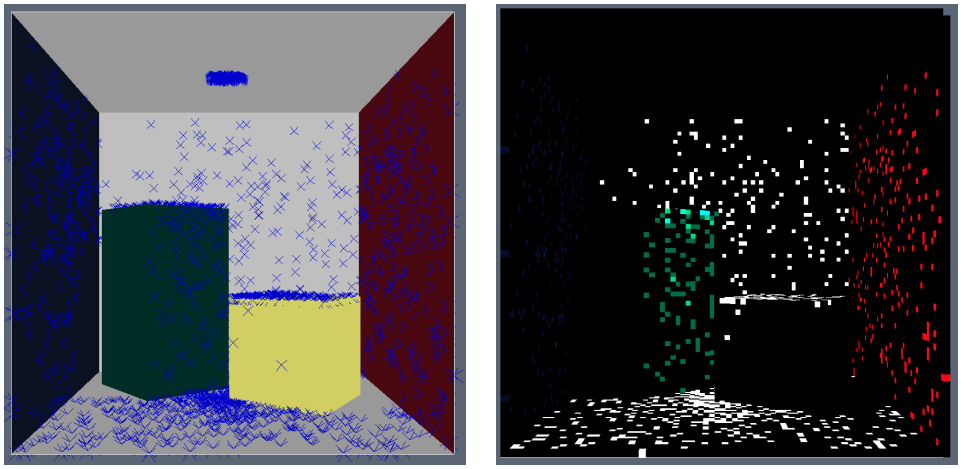
$$\frac{1}{2\pi} E(y) \Delta A(y) \Delta\Omega(\overrightarrow{\sigma}).$$

This energy is transmitted to the intersection voxel, first voxel met when going over a discrete line form y in the (integer) direction $\overrightarrow{\sigma}$.

To compute the intersection voxel, we store all surface voxels in an octree, which allows us to use some accelerated discrete ray-object technique similar to the ones used in discrete ray-tracing ([29]). The energy, thus, sent to most voxels is then stored and used as input of our optimal complexity propagation method.

3.5 Shooting Energy from Light Sources, raytracing (continuous) version

Although the discrete shooting version improves initial light propagation, it does suffer from scale problem. If we consider small light sources (like most existing in the



(a) Direct photons shooting (b) Initial voxel power

Figure 3.6: Photons shooting and voxels initialization

real world) in a vast space, the voxel size needs to be small enough to approximate light source shapes, thus, many voxels are required. If we choose too big a voxel size, the light leaking through the walls appears. A natural solution to this problem is the use of the raytracing instead of discrete shooting in this step.

We proceed as follows:

- choose number of initial photons on light sources with the random normal distribution that depends on light power and area.
- the energy of each sample is equal to $\frac{\text{overall energy}}{\text{number of samples}}$
- shoot from each initial photon n rays into n directions, with energy equal to E/n
- store given energy for each intersection point and create a set of indirect photons

The set of indirect photons is used as an initial voxel setup (initial voxel power is equal to the sum of the power of the photons inside the voxel). We propagate

using standard linear voxel method of 3.3. At the end we combine initial photons with indirect light propagated by the voxel method into set of VPL's (Virtual Point Lights), and use it as input data for rendering methods (see Instant Radiosity 4.3 or Lightcuts 4.4).

3.6 Cache method

After each reflection of the diffuse surface propagated energy amount is reduced in a significant way. We can use that property by using less directions at each intersection level. That is why we introduce the cache method technique. We propagate only energy that was created during last shooting phase, summing all energy in global buffer after each iteration. We split computations into sets of independent i to $i+1$ intersection levels, and we use different (usually divided by two after each step) number of directions at each level.

We do the following:

- shoot photons from direct light sources by raytracing at path length one (only first intersections).
- sum the amount of energy in respective voxels and use it as initial linear method values, we store that into *currentCache*.
- for each iteration

  propagate *currentCache* and store new values into *CachePlus*

  add energy from *CachePlus* to *GlobalCache*

  move energy from *CachePlus* to *currentCache*

  clear *currentCache*

- combine energy from $GlobalCache$ (converting voxels into indirect paths length 2+ virtual point lights) and we add photons as direct paths of the length 1 virtual point lights.

In that case overall complexity is $\mathcal{O}(N \times D)$. Experimental results show that this kind of propagation produces very similar results to the full propagation (the same number of directions at each iteration step).

Chapter 4

Rendering

4.1 Direct voxel solution view

Figure 4.1: Direct voxel view, computed in 24 hours

The first display approach (see [22] and [5]) consisted in visualization voxels solution directly, by interpolating values between voxels by averaging voxels values into nearby vertices and then using standard Gouraud interpolation and z-buffer to

visualize final image. First attempt was fully software based (custom raytracer with simple gouraud shading).

The drawback is that voxels are visible and we have to use a huge number of voxels to achieve acceptable quality, which makes the runtime go through the roof.

4.2 Texture based method

A second approach to display is lightmaps, which allows real-time animation of a static scene through z-buffer (using standard graphics card) or real-time raytracing.

The radiosity method computes a value for each voxel x of the discretized scene, which corresponds to the quantity of light emitted by each voxel. Before the final rendering we gather these values back from the voxels to visualize the scene. Gathering of the color values into the vertices (even with trilinear interpolation between the voxels) is not enough. In that case we lose most information about the light between the vertices. We can adaptively divide the scene into more faces but that would increase the scene complexity.

The solution is lightmaps. We create bitmaps that are able to store values from all the voxels. Similarly to the discretization step for each triangle, we proceed with the following steps:

- compute the coordinates of the triangle normal vector
- sort the projections by coordinate values (x, y, z)
- compute lightmap (see below) for the projection with minimal coordinates (to gain a maximal area of the projected triangle)

- compute the bounding box of the projected triangle (minimal rectangle with integer coordinates that covers all triangle points)
- extend that rectangle by 2 pixels in each direction
- compute the minimal power of 2 that is not less than the texture rectangle's width and height - it is the size of our final square texture

For each pixel of the texture we compute a ray - plane (containing the projected triangle) intersection to compute the third coordinate of the voxel. If the voxel exists at the proper position we store its color values into the texture otherwise we store a pink color (255,128,255) to mark an empty voxel (we use that color, because it is not often used). For each vertex of the triangle we calculate proper u, v coordinates over our texture.

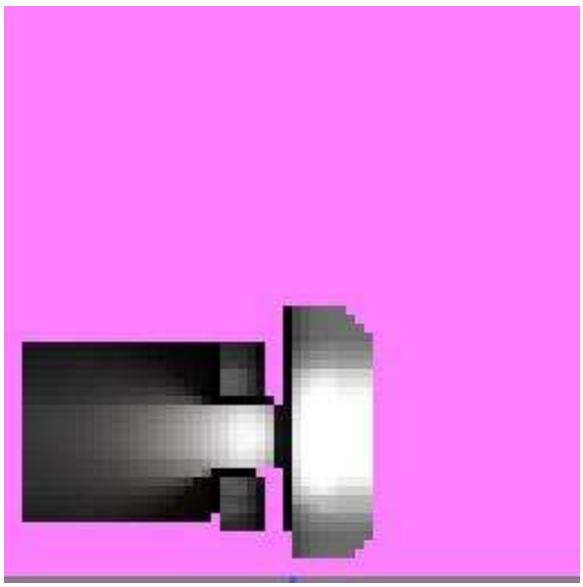


Figure 4.2: Sample lightmap after the computing color values

We, thus, obtain a bitmap composed of some significant pixels, which correspond

to voxel radiosity values, and some pink pixels which correspond to no voxels (this is due to the fact that the mesh is not planar and is not isometric to a bitmap plane). Now, during the display process, some graphics library will perform an interpolation between pixels of the bitmap. It may happen that the library uses some pink values in the process, generating some artefact. We overcome this difficulty by replacing each pink pixel color by the color of the nearest significant pixel for the city-block distance. We only need to modify values of pixels close to significant pixels (at city-block distance less than 4).

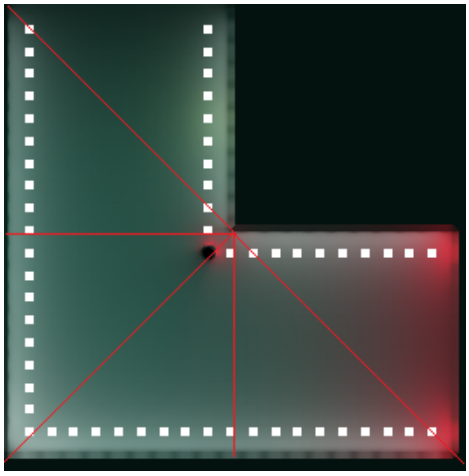


Figure 4.3: Lightmap of the common faces

Then we exchange pink color into black one. At this step we check if the texture consists of only one color. If so we decrease its size into 8x8 and set u,v texture coordinates for each vertices into 0.5 (we do not need to create large textures containing only one color).

In order to store all the triangular maps into a square texture, we pack the triangles into a single plane (see figure 4.3). In order to save the memory, we pack each configuration only once (details in next section).

4.2.1 Storing textures

As the final output, we create 512×512 textures that are well suitable for hardware graphics cards. We have sets of textures of different size greater than or equal to 8x8. We need to assume the following: each texture will be stored only once (two of the same textures will be stored at the same position), textures placement will be near to the optimal (we will not waste much space). We can complete that task by using a quadtree structure. A quadtree is a tree structure in which each node has less than 5 children. This is well suited to guarantee that textures with the same size will occupy similar positions in the final texture. We traverse the tree looking for a free position for our texture. Then we place our texture and recompute u, v coordinates for each vertex of the triangle for the final texture. Additionally, we compute a hash code for each stored texture, that allows us not to store the same texture more than once. First we check if the texture does not exist already in the tree and then we store it. The hash code helps us to compare the textures, if hash codes are different, textures are different too, otherwise we have to compare the textures pixel by pixel what is computationally expensive. The hash code is just a 64 bit long profile of the texture, hash codes should be different for different textures. In our case it is calculated as a sum of the colors *xor* position *xor* the number of non-zero red/green/blue values. It can be replaced by any function that is easy to compute and gives different values for different data in most of the cases.

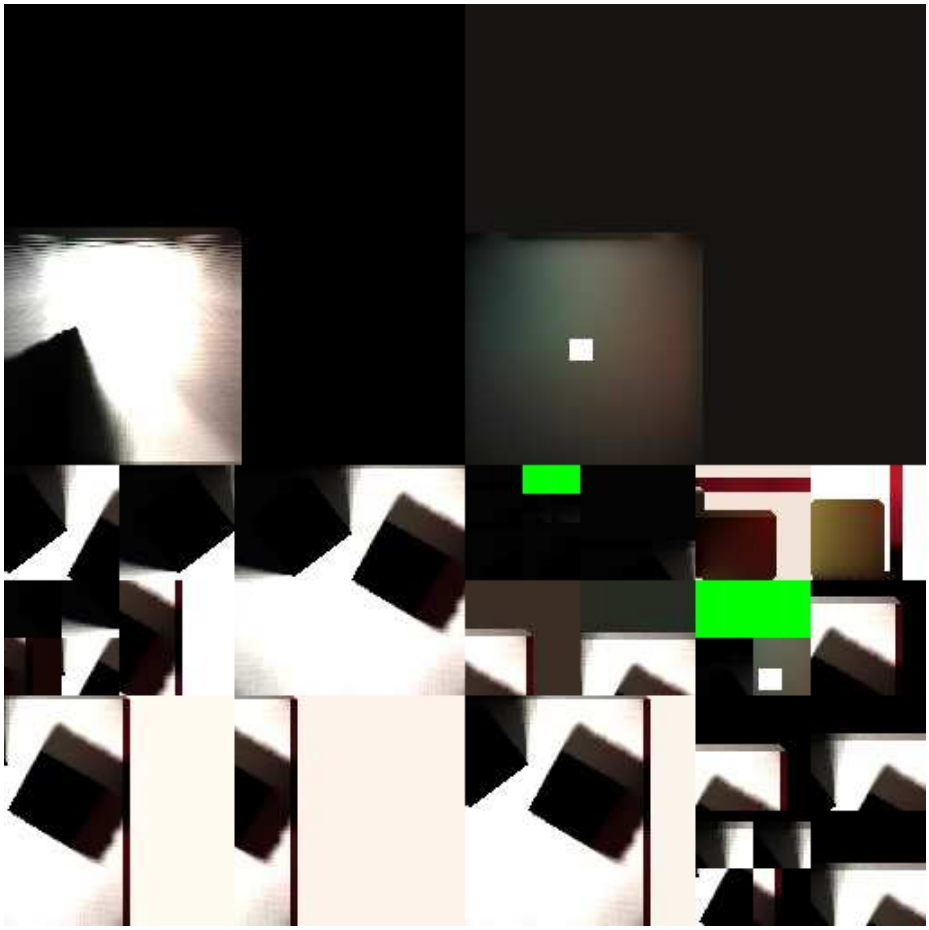


Figure 4.4: One of two lightmaps for "The Cornell Box" test scene

4.3 Instant Radiosity

Introduced by Alexander Keller in [17] Instant Radiosity is a sparse method for the radiance approximation. It converts lighting problem into a set of virtual point lights, by casting a number of photons from light sources and storing its direction, position and power (attenuated by the diffuse component of that surface), also the mean reflectivity p of the scene is calculated. $p * N$ (where N is the number of the original points) continue on to a next bounce (will survive after one reflection), this process is repeated until all of the paths are completed (and photons absorbed). Then for each virtual point light the scene is rendered using that point as the only light source, results from all the lights are accumulated and divided by the number of the lights. The architecture of this method allows to use graphics card hardware acceleration, by composing the final image in the accumulation buffer. This method is not very accurate, because it relies on accumulation buffer depth (8-bit seems to be not enough except simple preview scenes). The original method works with diffuse surfaces and uses low-discrepancy points Quasi-Monte Carlo Integration. The basic instant radiosity method introduces consistent errors, thus, limiting the minimal light distance (light-intensity is divided by the square of the distance towards the light source, for a distance less than 1, the formula for the distance attenuation gives a higher light intensity than at the original VPL position. This problem is solved by bias compensation described in paper [19] "Illumination in the Presence of Weak Singularities".

4.3.1 Implementation details

Initially to compute the final image we use raytracing and use the voxels as a light sources (or $\mathbb{VPL}\sim$) in the way inspired by the instant radiosity method in [17]. Of

course, the principle must be substantially modified to be adapted to voxels. The intensity of the voxels is proportional to their radiosity as obtained in the output of the method in [5]. However, to take into account the nature of radiosity (power per unit of steradian per unit of area) we select the voxels *randomly* according to a probability proportional to their *area*, as defined below, and multiplied by a solid angle.

Each connected component χ of the continuous surface (e.g mesh) separates the space into two connected components, C and $\overline{C}$ (Jordan theorem), where $\overline{C}$ is the component which contains the viewpoint. We consider O the set of all the voxels $(i, j, k) \in Z^3$ which are contained in C (i.e. voxels inside an object or outside the scene). The discretization of the surface is the boundary F of O, which is the set of all voxels in O which are 6−adjacent to (i.e. have a common face with each other) a voxel in the complement $\overline{O}$ of O.

For a voxel $x \in F$, the element of area $\Delta A(x)$ is the sum, for all 6−neighbors y of x in $\overline{O}$, of the dot product $\overrightarrow{xy}.\overrightarrow{N}$, where $\overrightarrow{N}$ is the normal vector at the point x.

$$\Delta A(x) = \sum_{y \in N_6(x) \cap \overline{O}} \overrightarrow{xy}.\overrightarrow{N}$$

So, we make a raytracing phase by tracing rays from the viewpoint through the pixels. For each pixel, we compute the ray-object intersection point I that we must shade. In order to shade the point I, we use the sample voxels y, randomly selected with probability proportional to $\Delta A(y)$, as point light sources with intensity, or *energy contribution*:

$$C_y(I) = B(y) \frac{\cos\,\theta(I, y) \cos\,\theta(y, I)}{||I - y||^2} V(I, y)$$

In fact, we select two random samples sets of voxels: one for light sources and one for non-light sources. The reason for this is that the light sources (non-zero emittance) emit much more power and must be sampled more finely. We denote by $\mathcal{L}$ the set of all voxels with non-zero emittance (light source voxels) and $\mathcal{NL}$ the set of all non-light source voxels. We select a set $V_d \subset \mathcal{L}$ of light source sample voxels. and a set $V_i \subset \mathcal{NL}$ of non-light source sample voxels. We denote $n_d = |V_d|$ the number of light source sample voxels and $n_i = |V_i|$ the number of non-light source sample voxels, The total power of the pixel (or equivalently at I) is computed as :

$$\frac{1}{2\pi}\left(\frac{TAD}{n_d}.\sum_{y\in V_d} C_y(I) + \frac{TAI}{n_i}.\sum_{y\in V_i} C_y(I)\right)$$

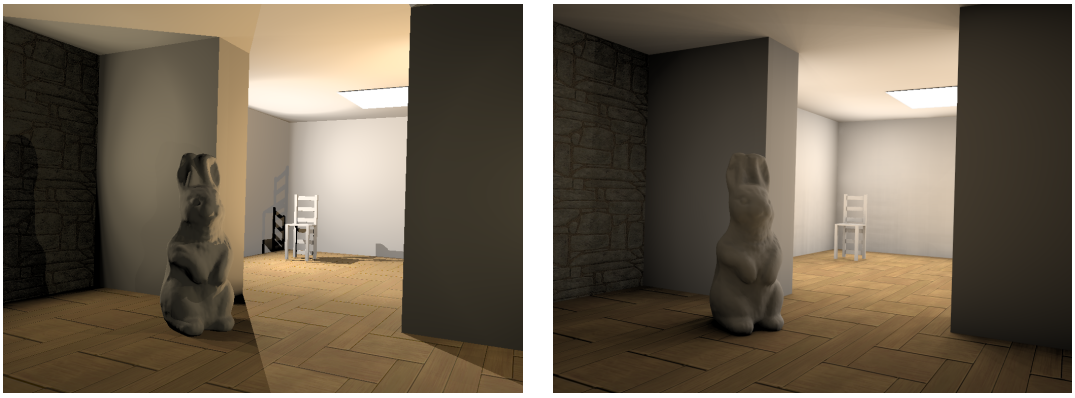
where:

$$TAD = \sum_{x\in\mathcal{L}} \Delta A(y)$$

$$TAI = \sum_{x\in\mathcal{NL}} \Delta A(y)$$

are the total areas of light source surfaces $\mathcal{L}$ and non-light source surfaces $\mathcal{NL}$.

We use progressive raytracing that allows the user to get approximate results after a few seconds. For each frame, part of all the Virtual Point Lights (VPL) is taken into account. Thus, at each instant the preview image is an approximation of the energy of the final solution. The output converges to the final solution and computation can be stopped by the user or automatically (by using a difference threshold between successive frames). Since we use the same set of $VPLs$ for all the pixels (and possibly for all viewpoints in the same static scene), no noise is noticeable (the only problem is hard shadow edges which disappear after a sufficient number of $VPLs$ is taken into account).



(a) 16 VPL (b) 480 VPL

(c) 1856 VPL

Figure 4.5: Bunny test scene, progressive rendering after 1, 30 and 116 frames

To create a proper VPL order, we select the voxels randomly according to a probability proportional to their *area*. To do this, we place all the voxels in a table, compute a random number between 0 to the sum of the areas ΔA, and use a binary search to select the right voxel. Conformly to theory of statistics, we do not reject a voxel after selection if selected more then once.

In the raytracing step, at each frame, for each pixel we compute the contribution of the next 16 of the $VPLs$ to the final picture.

We add features like anti-aliasing and depth of field without additional cost, by

Figure 4.6: The Sponza Atrium test scene

slightly modifying the ray's position and/or direction. For both effects we use stratified sampling computed once for each frame and used by each pixel.

This method generated proper images, but its complexity is dependent on $\mathcal{O}(n)$, where n is a number of virtual point lights. At this point the method works and generates accurate images, but computation time was still far above comparing to the other standard techniques (because of number of VPL's to be computed for each pixel).

We improved that by using the Lightcuts.

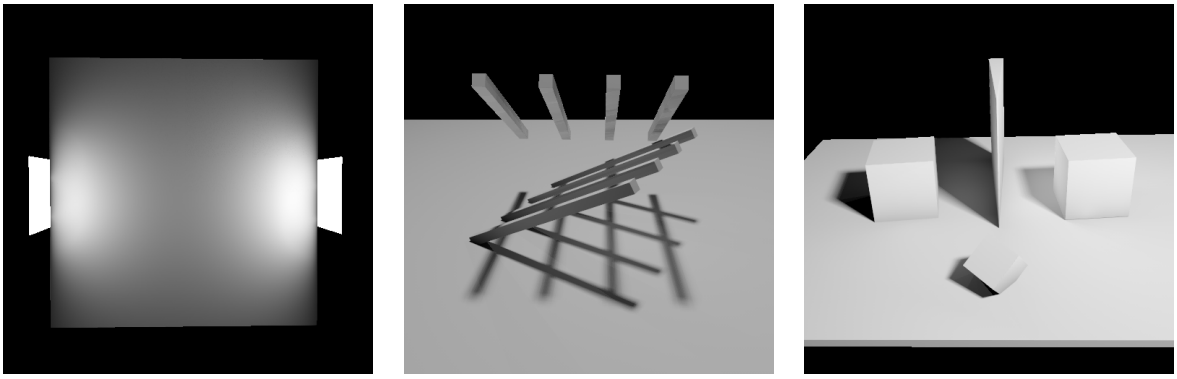


Figure 4.7: Three of the Global Illumination Test Scenes ([28]) (emission test, shadow test, and geometric accuracy)

Table 4.1: Test scenes statistics

Scene	**Voxels**	**Light path length**	**Shooting discrete sphere radius**	**Linear method discrete sphere radius**	**Image size**	**Total time (seconds)**
Bunny1	146355	5	35	22	1024x768	558
Bunny30	146355	5	35	22	1024x768	2486
Bunny116	146355	5	35	22	1024x768	7742
Sponza	236754	7	35	26	1024x768	29185

Table 4.2: Statistics for different kinds of rays

Scene	**Voxel shooting time**	**Linear method propagation time**	**Ray-tracing display time**	**number of discrete rays (shooting)**	**number of discrete rays (linear method)**	**number of continous rays (raytracing)**
Bunny1	93	404	52	70.5e6	807.6e6	9.45e6
Bunny30	93	404	1980	70.5e6	807.6e6	297e6
Bunny116	93	404	7236	70.5e6	807.6e6	1 133e6
Sponza	274	960	27755	211.4e6	1 181e6	2 030e6
Emission	5	50	112	3.7e6	284e6	8.7e6
Shadow	4	23	119	3.2e6	141e6	29.3e6
Geometric	11	20	176	8.1e6	109e6	43.5e6

4.4 Lightcuts

The lightcuts method introduced in 2005 by a paper "Lightcuts: A Scalable Approach to Illumination" [32] and "Implementing lightcuts" [31], enables to render massive (thousands to millions) amounts of lights in reasonable (sublinear) time. The only disadvantage is a small (determined by user - usually less than 2%) pixel potential error value. All types of light (like direct, indirect, sun, sky light and enviroment light maps) are converted into a sets of point lights. Then common lightcut tree is created which unifies illumination and enables transparent tradeoffs between adequate components. Most previous works in that area like: "A rapid hierarchical radiosity algorithm" [12], "A Light Hierarchy for Fast Rendering of Scenes with Many Lights" [25], accelerate the processing of individual lights but scale linearly with the number of lights, contrary to the lightcuts method which scales sub-linearly.
Given a set of point light sources S, the radiance at a surface point x viewed from direction ω is

$$L_S(x,\omega) = \sum_{i \in S} \underbrace{M_i(x,\omega)}_{\text{material}} \underbrace{G_i(x)}_{\text{geometric}} \underbrace{V_i(x)}_{\text{visibility}} \underbrace{I_i}_{\text{intensity}} \tag{4.4.1}$$

A cluster is a set of point light sources which are approximated by a single representative light source:

$$\begin{aligned} L_C(x,\omega) &= \sum_{i \in C} M_i(x,\omega) G_i(x) V_i(x) I_i \\ &\approx M_j(x,\omega) G_j(x) V_j(x) \sum_{i \in C} I_i \end{aligned} \tag{4.4.2}$$

4.4.1 Implementation details

We divide direct and indirect VPL's to create separate trees, and combine them in root node as described in the paper. Assuming that number of virtual point lights may be hundred of thousands, optimal tree creation time is desired. To achieve $\mathcal{O}(n \log n)$ tree creation time complexity we use technique described in [24], that consists in using of bsp-tree (binary search partition tree) and minheap structures. Opposite to the Instant Radiosity method we do not order voxels stochastically computing its projected area, we consider all of them (we treat them as a set of points over the scene that we use to compute exact radiosity value). First we randomly select positions on the light sources (a number of samples at each light source depends on its total power and area). This way we sample two sets of the initial points. One of them is used to compute direct lights sampling, and is used for creation of the direct light lightcuts tree. The second one is used as initial photons positions. Separating a number of samples in each case helps to better fit parameters in case of different scenes (a scene fully lit by a direct light vs a scene lit mostly by an indirect light). From each of such positions, we shoot a number of photons in accordance with the cosine weight over the half sphere (in the direction of the normal vector). At each intersection point we store photon position and its energy scaled by the material diffuse factor. For each stored photon we search nearby voxel with a similar normal vector, if found we add photon energy to that voxel. At this point we have a set of direct VPL's for direct light computation, and a set of non-zero emission voxels (light propagated at path length 1). We do continue propagation of this set of voxels by the linear voxel method as described previously. Finally we gather all the voxels as an indirect VPL's set. Because of ordered voxels positions this set is well suitable for indirect

light approximation, it is better placed then disordered set of photons in the original shooting method (in that case many photons may be near to each other, while some places may stay empty at all).

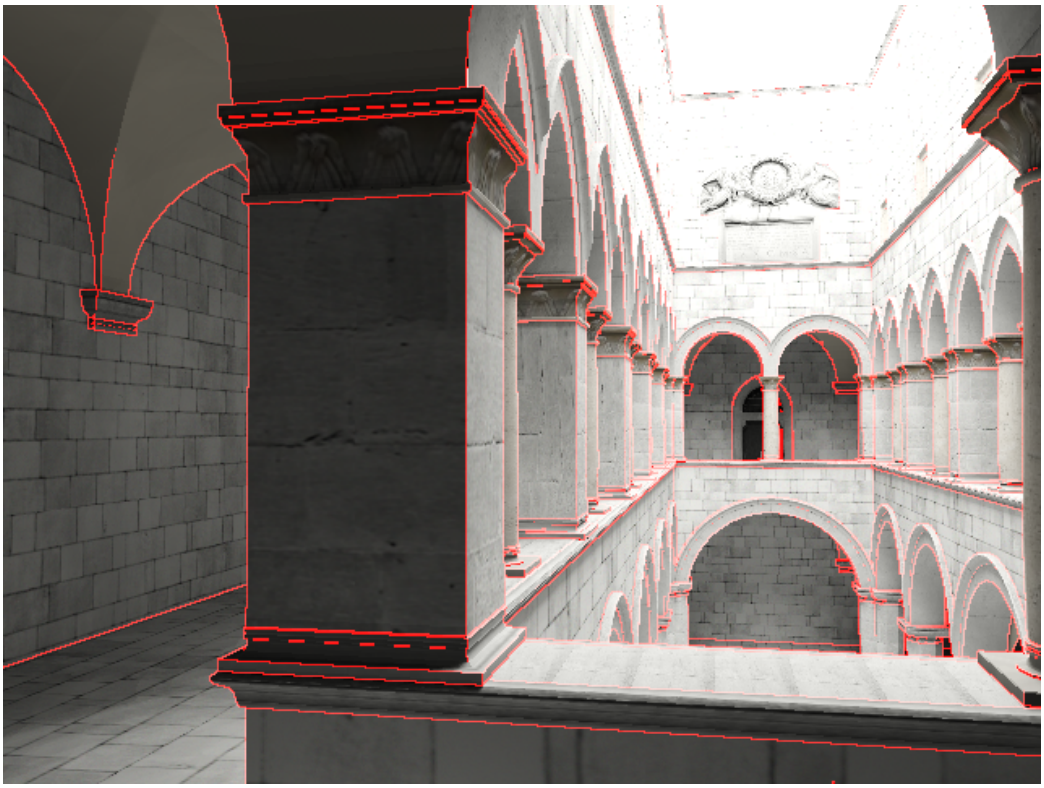


Figure 4.8: Adaptive antialiasing

To improve final results we use adaptive anti-aliasing, for red pixels (as seen in the picture 4.8) we compute additional 15 samples and average the result. Samples are computed using quasi-random 4x4 stratified sampling (the same set of samples is used for each pixel). Aliased pixels selection is based on their potential difference in material, ratio of the length to the eye or the normal vector to its neighbor pixels.

4.5 General rendering process

We start by discretization of the input scene into a set of the voxels. Then we shoot photons from light sources and store their positions and energy. We do not reflect photons at the surfaces, we store all of them. The photons are converted into initial voxels values (usually we sum energy from several photons for a single voxel), and then energy is propagated by the linear voxel method in several iterations. At each iteration, a number of directions is divided by two. All voxels are converted into an indirect virtual point light set. A number of points is stochastically selected (according to the power and their area) on the light sources and converted into a direct virtual point light set. Both sets are used to create respective trees for lightcuts, then are connected into a root node. Lightcuts algorithm is used to obtain the final image.

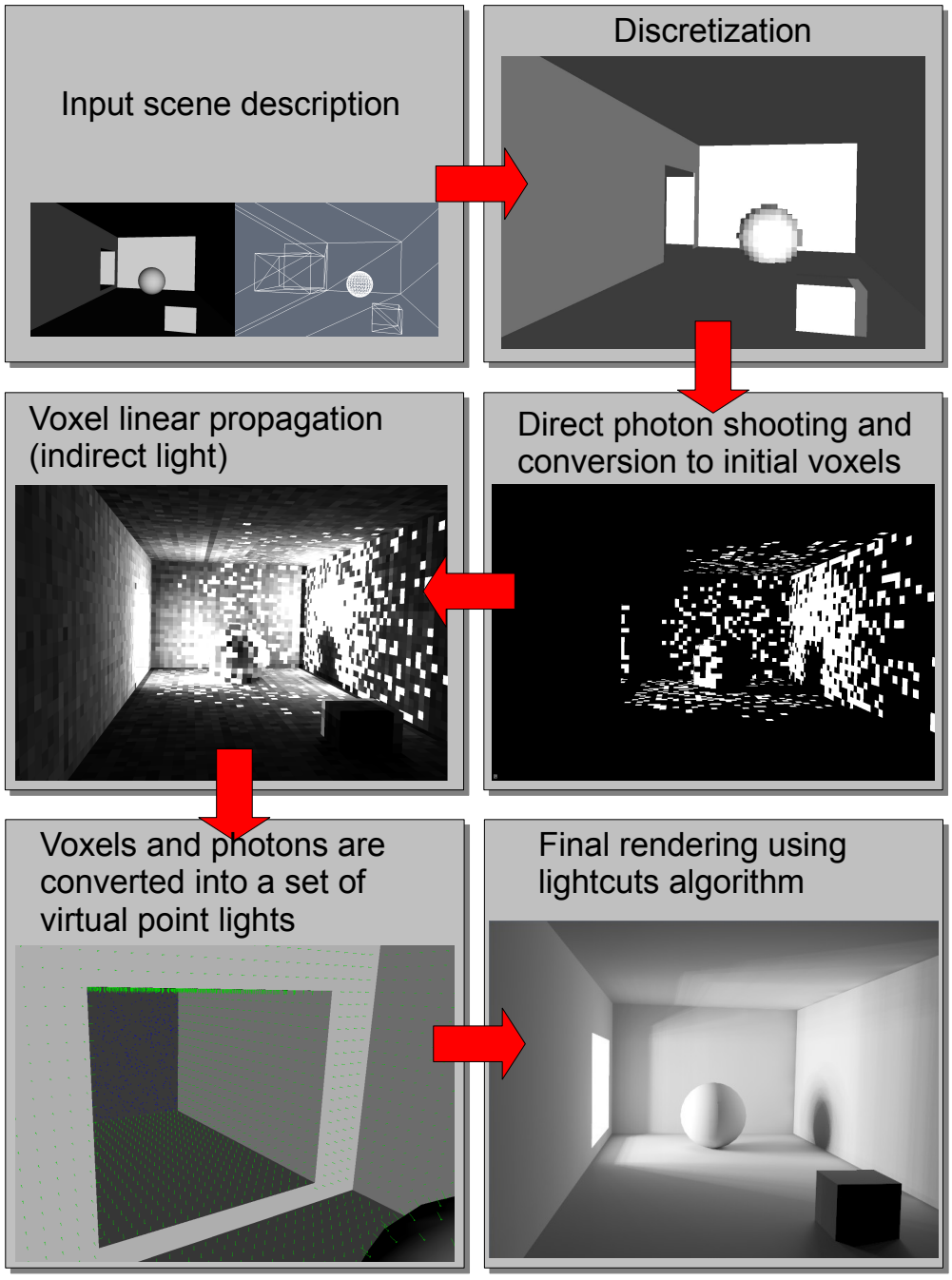


Figure 4.9: Rendering flow process

Chapter 5

Results

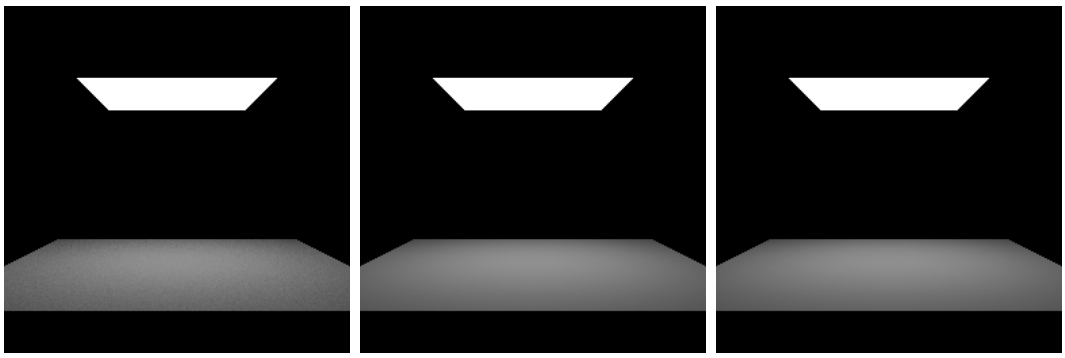
All the tests were computed on PC with 2GB ram, Intel Core 2 Duo 6300 (1.86GHz) using single thread, on Windows Vista.

5.1 Experimental results

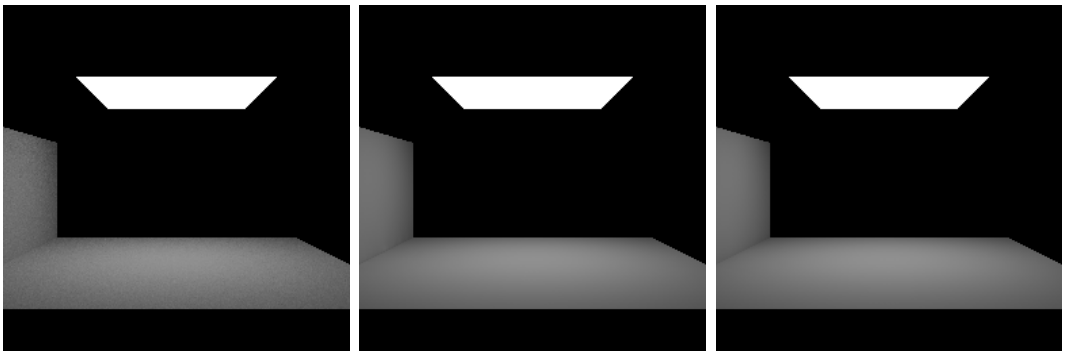
In this chapter we test the method practically, and compare it to the other methods.

5.1.1 Correctness test

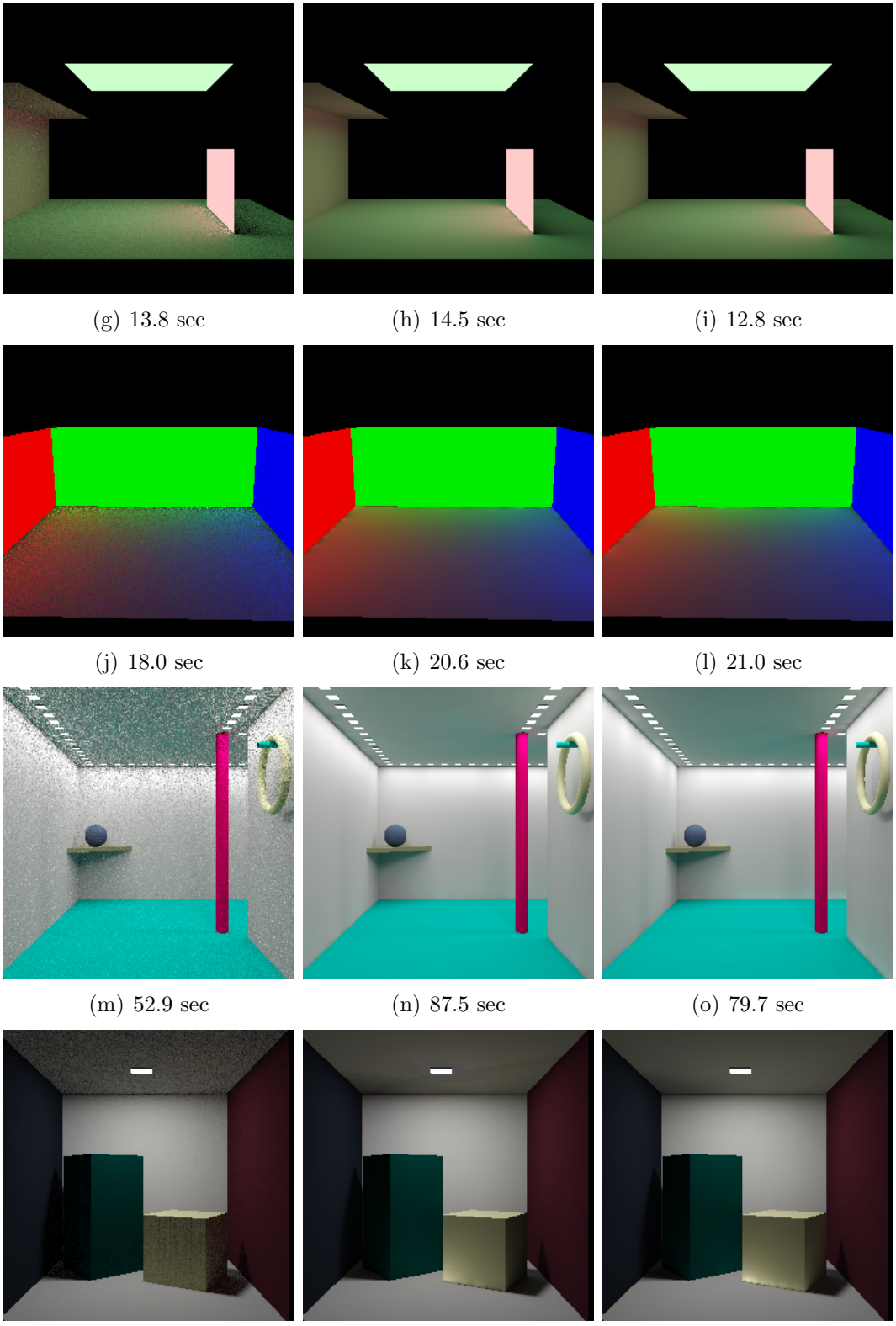
This test compares the method with a reference solution (path tracing). The left column is Monte Carlo path-tracing (50 samples per pixel), the middle one is photon lightcuts, and the right one is the voxel lightcuts method. All pictures were computed using 1600 direct photons, 1600 indirect photons, 128 directions in linear voxel propagation, and 2 iterations, discretization resolution was 64. Results show potential superiority of the voxel method over the photon method in scenes with strong influence of indirect light. We compute path-tracing at constant rate samples per pixel (50), thus, the time is sometimes better, but the noise is strongly visible. This test shows that the method is fully accurate.



(a) 11.4 sec (b) 9.1 sec (c) 6.6 sec



(d) 13.5 sec (e) 11.2 sec (f) 9.0 sec



(g) 13.8 sec (h) 14.5 sec (i) 12.8 sec

(j) 18.0 sec (k) 20.6 sec (l) 21.0 sec

(m) 52.9 sec (n) 87.5 sec (o) 79.7 sec

(p) 33.1 sec (q) 61.4 sec (r) 63.7 sec

Figure 5.1: Comparison with other methods

5.1.2 Parameters adjusting

This test was created to compare different parameters adjustment. The columns represent the numbers 32, 64, 128, 256 which are numbers of voxels directions (in the linear voxel method) and the rows the number of initial photons used to shoot from light sources. Sufficient number of initial photons (rows in pictures) is more important than sufficient number of directions, which means we can propagate indirect light roughly, but we need a good starting base (initial emission) in voxels. Separating direct lighting from indirect photons used to initialize voxels emission values before the propagation allowed us to save computation power for similar results of the full solution. Dividing the number of voxel directions after each step does not introduce visible errors, and allows us to reduce the propagation time. Finally voxels contrary to photons, guarantee a proper indirect virtual point lights placement and density (there is, at most, one VPL's per voxel). In some cases the photons density is not easily predictable and a lot of VPL's may be wasted while at the same time in other parts of the scene density may be too low.

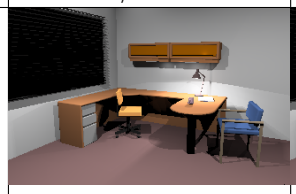
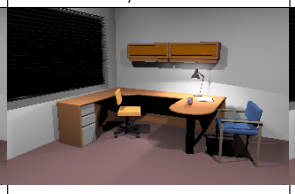
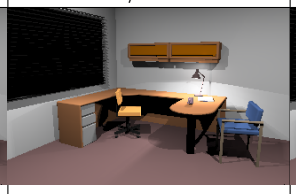
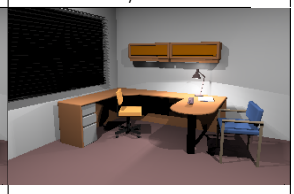
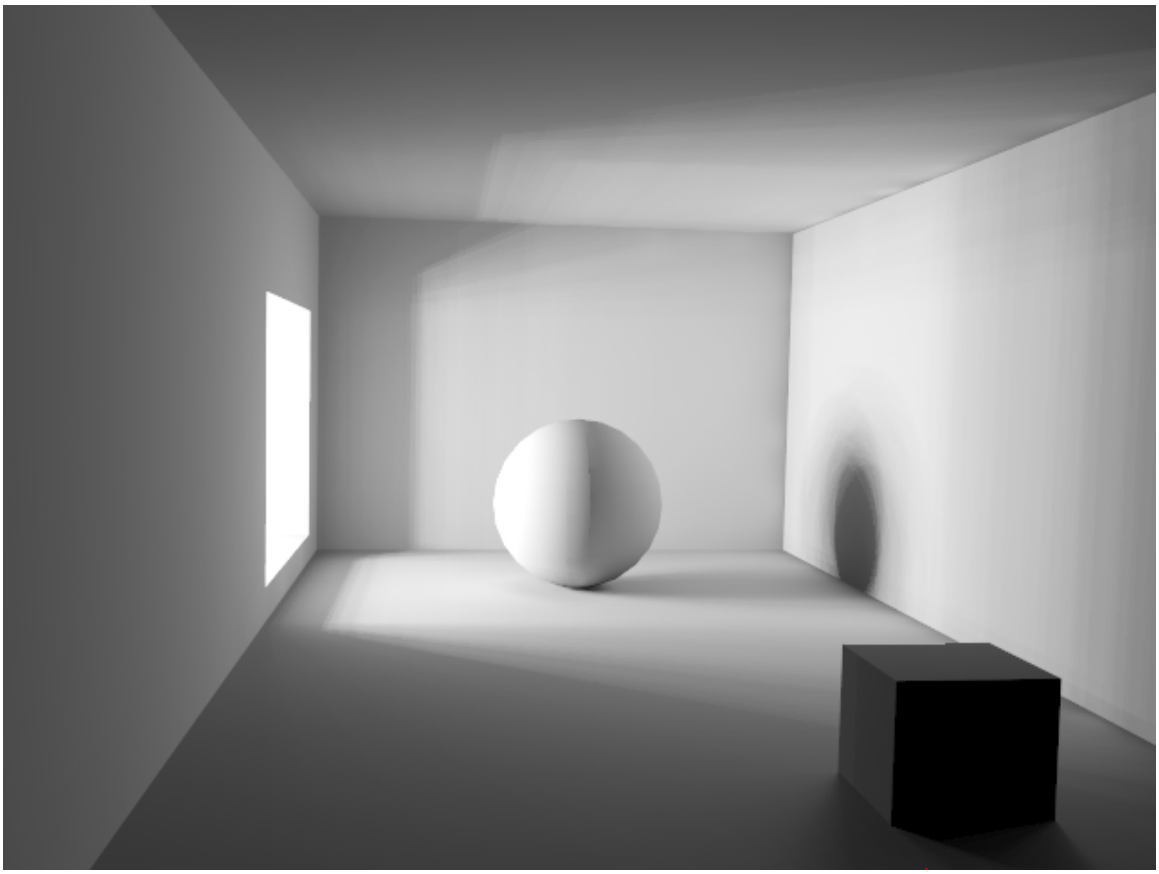
	32	64	128	256
32				
	2721/51.7 sec	9194/53.1 sec	18848/57.6 sec	20987/58.5 sec
64				
	4301/51.6 sec	10532/53.7 sec	19542/57.4 sec	21015/57.9 sec
128				
	6707/51.3 sec	14891/55.2 sec	20291/57.8 sec	21143/58.5 sec
256				
	6527/40.3 sec	17254/44.5 sec	20458/46.3 sec	21217/47.7 sec
512				
	8399/41.4 sec	17056/44.6 sec	20953/47.0 sec	21243/46.9 sec
1024				
	11531/54.5 sec	18875/57.8 sec	21023/59.3 sec	21306/59.6 sec

Figure 5.2: Number of lights and computation time



Width	640
Height	480
Antialiasing	off
Discretisation resolution	64
Number of voxels	15503
Number of iterations	4
Voxel directions	4555
Average lightcut cut size	303
Primary photons	800
Direct photons	80
Continuous rays	108110125
Discrete rays	112002401
Rays/sec (continuous)	208733.0
Rays/sec (linear)	5773319.6
Propagation time	19.8
Propagation time (linear only)	19.4
Raytracing time	517.9
Overall time	537.8

Width	640
Height	480
Antialiasing	off
Discretisation resolution	127
Number of voxels	108455
Number of iterations	2
Voxel directions	146
Average lightcut cut size	212
Primary photons	650
Direct photons	200
Continuous rays	87913202
Discrete rays	26637785
Rays/sec (continuous)	140816.2
Rays/sec (linear)	5122650.9
Propagation time	6.1
Propagation time (linear only)	5.2
Raytracing time	624.3
Overall time	630.5

Width	640
Height	480
Antialiasing	off
Discretisation resolution	64
Number of voxels	20858
Number of iterations	2
Voxel directions	425
Average lightcut cut size	100
Primary photons	4000
Direct photons	10
Continuous rays	41947952
Discrete rays	14944716
Rays/sec (continuous)	72487.4
Rays/sec (linear)	5977886.4
Propagation time	3.5
Propagation time (linear only)	2.5
Raytracing time	578.6
Overall time	582.2

Width	640
Height	480
Antialiasing	on
Discretisation resolution	64
Number of voxels	22615
Number of iterations	4
Voxel directions	256
Average lightcut cut size	294
Primary photons	4000
Direct photons	256
Continuous rays	184834570
Discrete rays	10146637
Rays/sec (continuous)	68986.83
Rays/sec (linear)	5172795.48
Propagation time	4.15
Propagation time (linear only)	1.96
Raytracing time	2679.27
Overall time	2684.43

Conclusion

Our experimental results show that voxel based Global Illumination can be competitive for exact simulation and has a potential for the future for real-time animation of static scenes. We prove herby that its very good complexity can be used practically.

The voxel based method is accurate for precise multi-bounce global illumination, and provides very large numbers of rays per second for discrete rays, which makes the technique promising, a number of discrete rays per second is about 5 million, at the same time a number of continuous rays per second is about 200-400 thousand for a sample scene.

In the future, we are going to develop a method for faster reconstruction step based directly on the voxels values. We are also going to use the lightcuts method to compute initial power in voxels to compare it to photon shooting. Finally, we could find a method for fast animation which would enable adding an object into the scene without recomputing the whole solution by the use of anti-radiance.

Bibliography

[1] *The rendering equation*, New York, NY, USA, ACM, 1986.

[2] Okan Arikan, David A. Forsyth, and James F. O'Brien, *Fast and detailed approximate global illumination by irradiance decomposition*, Proceedings of ACM SIGGRAPH 2005, ACM Press, 2005.

[3] J. L. Bentley and J. H. Friedman, *Data structures for range searching*, ACM Computing Surveys **11** (1979), no. 4, 397–409.

[4] J.L. Bentley, *Multidimensional binary search trees used for associative searching*, CACM **18** (1975), no. 9, 509–517.

[5] Pierre Y. Chatelier and Remy Malgouyres, *A low-complexity discrete radiosity method*, Computers & Graphics **30** (2006), 37–45.

[6] Michael F. Cohen, Shenchang Eric Chen, John R. Wallace, and Donald P. Greenberg, *A progressive refinement approach to fast radiosity image generation*, SIGGRAPH Comput. Graph. **22** (1988), no. 4, 75–84.

[7] Carsten Dachsbacher and Marc Stamminger, *Splatting indirect illumination*, SI3D '06: Proceedings of the 2006 symposium on Interactive 3D graphics and games (New York, NY, USA), ACM Press, 2006, pp. 93–100.

[8] Carsten Dachsbacher, Marc Stamminger, George Drettakis, and Frédo Durand, *Implicit visibility and antiradiance for interactive global illumination*, August 2007.

[9] Isabelle Debled-Rennesson, *étude et reconnaissance des droites et plans discrets*, Ph.D. thesis, Université Louis Pasteur, Strasbourg, 1995, PhD thesis.

[10] Henri Gouraud, *Continuous shading of curved surfaces*, (1998), 87–93.

[11] Xavier Granier and George Drettakis, *A final reconstruction approach for a unified global illumination algorithm*, ACM Trans. Graph. **23** (2004), no. 2, 163–189.

[12] Pat Hanrahan, David Salzman, and Larry Aupperle, *A rapid hierarchical radiosity algorithm*, Computer Graphics **25** (1991), no. 4, 197–206.

[13] Henrik Wann Jensen, *Global Illumination Using Photon Maps*, Rendering Techniques '96 (Proceedings of the Seventh Eurographics Workshop on Rendering) (New York, NY), Springer-Verlag/Wien, 1996, pp. 21–30.

[14] ———, *Rendering caustics on non-Lambertian surfaces*, Computer Graphics Forum **16** (1997), no. 1, 57–64.

[15] ———, *Realistic image synthesis using photon mapping*, A. K. Peters, Ltd., Natick, MA, USA, 2001.

[16] Henrik Wann Jensen and Niels Jørgen Christensen, *Photon maps in bidirectional monte carlo ray tracing of complex objects*, Computers & Graphics **19** (1995), no. 2, 215–224.

[17] Alexander Keller, *Instant radiosity*, Proceedings of the ACM SIGGRAPH Conference (SIGGRAPH-97) (New York), ACM Press, August 3–8 1997, pp. 49–56.

[18] David B. Kirk and James Arvo, *Unbiased sampling techniques for image synthesis*, Computer Graphics (SIGGRAPH '91 Proceedings), vol. 25, July 1991, pp. 153–156.

[19] T. Kollig and A. Keller, *Illumination in the presence of weak singularities*, vol. Monte Carlo and Quasi-Monte Carlo Methods 2004, pp. 245–257, Springer Verlag, 2004.

[20] Fabien Lavignotte and Mathias Paulin, *Scalable photon splatting for global illumination*, Proceedings of GRAPHITE, 2003, pp. 1–11.

[21] R. Malgouyres, *Une dfinition des surfaces de z^3*, Ph.D. thesis, Universit Clermont 1, 1994.

[22] Rémy Malgouyres, *A discrete radiosity method*, DGCI (Achille J.-P. Braquelaire, Jacques-Olivier Lachaud, and Anne Vialard, eds.), Lecture Notes in Computer Science, vol. 2301, Springer, 2002, pp. 428–438.

[23] N. Metropolis and S. Ulam, *The monte carlo method*, J. Am. Statistical Assoc. **44** (1949), 335–341.

[24] M. Miksik, *Implementing lightcuts*, In Central European Seminar on Computer Graphics for students, 2007.

[25] Eric Paquette, Pierre Poulin, and George Drettakis, *A light hierarchy for fast rendering of scenes with many lights*, Computer Graphics Forum **17** (1998), no. 3, ??–??

[26] H. P. Santo (ed.), *Bi-directional Path Tracing*, Alvor, Portugal, 1993.

[27] François Sillion and Claude Puech, *Radiosity and global illumination*, Morgan Kaufmann Publishers, San Francisco, 1994.

[28] B. Smits and H. Jensen, *Global illumination test scenes*, Tech. report, University of Utah, 2000, Technical report.

[29] Nilo Stolte and René Caubet, *Discrete ray-tracing of huge voxel spaces*, Comput. Graph. Forum **14** (1995), no. 3, 383–394.

[30] G. Strang, *Linear algebra and its applications*, HBJ, San Diego, 1988.

[31] Bruce Walter, Sebastian Fernandez, Adam Arbree, Kavita Bala, Michael Donikian, and Donald P. Greenberg, *Implementing lightcuts*, SIGGRAPH '05: ACM SIGGRAPH 2005 Sketches (New York, NY, USA), ACM Press, 2005, p. 55.

[32] ———, *Lightcuts: a scalable approach to illumination*, SIGGRAPH '05: ACM SIGGRAPH 2005 Papers (New York, NY, USA), ACM Press, 2005, pp. 1098–1107.

[33] Rita Zrour, Fabien Feschet, and Rémy Malgouyres, *Parallelization of a discrete radiosity method using scene division*, OTM Conferences (2) (Robert Meersman and Zahir Tari, eds.), Lecture Notes in Computer Science, vol. 4276, Springer, 2006, pp. 1213–1222.