Efficient computation of global illumination based on adaptive density estimation

submitted by

Wong Kam Wah

for the degree of Doctor of Philosophy

at the University of Hong Kong

in September 2001

Efficient computation of global illumination based on adaptive density estimation

by

Wong Kam Wah

B.Sc.(Computer Studies) H.K.U. 1990

M.Phil. H.K.U. 1996

A thesis submitted in partial fulfilment of the requirements for

the Degree of Doctor of Philosophy

at the University of Hong Kong.

September 2001

Declaration

I declare that this thesis represents my own work, except where due acknowledgement is made, and that it has not been previously included in a thesis, dissertation or report submitted to this University or to any other institution for a degree, diploma or other qualification.

....

Wong Kam Wah

Acknowledgements

I would like to thank my advisor, Dr. Wang Wenping, for taking me on as his student and allowing me to pursue my research interests, and for his support during the course of this work.

Abstract of thesis entitled

Efficient computation of global illumination based on adaptive density estimation

submitted by

Wong Kam Wah

for the degree of Doctor of Philosophy

at the University of Hong Kong

in September 2001

This thesis presents a new method of density estimation for the global illumination framework. In the new method, to represent an illumination function the appropriate number of terms that should be used in a series of orthogonal basis functions is determined adaptively and automatically. Moreover, a surface subdivision scheme is combined with the estimator to increase the accuracy of estimation. We show that our new method is more efficient than the other similar adaptive meshing approaches.

This thesis also presents an empirical comparison of different density estimation methods that are applied in the global illumination framework, or proposed in the statistics literature but have not yet been applied by computer graphics researchers. The comparison investigated three estimators (kernel estimator, wavelet thresholding estimator, and our new estimator) in four different aspects: image quality, L_2 error, running time, and memory and storage requirement. We show that each estimator has its own shortcomings, but our method is more practical if computational resources are limited.

Lastly, this thesis presents a simple load balancing strategy that can be applied to the parallel implementation of our new estimator. The strategy is based on the fact that in our new estimator, polynomial update requires less global information. Therefore, these tasks can be passed to other idle processors for evaluation without introducing too much communication overhead. We implemented the parallel algorithm on a cluster of PC, and an improvement of the speedup factor was obtained.

Chapter 1

Introduction

For solving global illumination, the classical hierarchical radiosity method [12, 28] cannot easily be extended to handle complex optical effects, such as non-uniform luminaries or light scattered from a non-diffuse surface. Walter et al [35] proposed a density estimation framework to solve this problem. Energy-carrying particles are emitted from each light source, traced through the environment, hit and reflected from surfaces until they are absorbed probabilistically. Illumination on a surface can be estimated from the density of particle-hit points on the surface.

The density estimation framework has several advantages over the classical hierarchical radiosity method. First, most complex optical effects can easily be simulated. Secondly, the huge memory requirement in the classical hierarchical radiosity method can be avoided [29]. Lastly, since particles interact with the raw input geometry only, the particle tracing stage and surfaces' illumination estimation are independent of each other. As a result, the error in the surfaces' illumination estimation is not propagated [25, 35].

The illumination on a surface is proportional to the density of the particle-hit points on the surface. Regions with more particle-hits should have brighter illumination. Heckbert [14] first noted that reconstructing illumination from particle-hits is a density estimation problem [27]. Several techniques are given in the statistics literature for solving the density estimation problem [27]. Some of these techniques have already been applied by computer graphics researchers; for example, the adaptive meshing methods [14, 32], the kernel methods [3, 35], and the orthogonal series estimator [8].

According to our knowledge, after the density estimation framework was introduced, there has been no research work that investigating the effect of various density estimation methods on this framework. Obviously the performance of the density estimator used determines the effectiveness of this framework.

We have developed a new density estimation method for this framework, presented in Chapter 3. Our method is based on the orthogonal series estimator [27]. In the orthogonal series estimator, surface illumination is represented by a finite number of terms in a set of orthogonal basis functions. The main contribution of the new method is that the appropriate number of terms that should be used in the orthogonal series is determined adaptively and automatically by the algorithm. Furthermore, in order to avoid using an overly large number of terms, an adaptive surface subdivision scheme is incorporated in the new method. Therefore, the new method combines the advantage of orthogonal series estimator approach and adaptive meshing approach. We show that our new method is more efficient than the other similar adaptive meshing approaches.

In Chapter 4, we compared the performance of this new method to the other existing density estimators that do not belong to the adaptive meshing category. We compared our method with the kernel method and the wavelet thresholding method. Our results show that each estimator has its own shortcomings in different aspects (image quality, L_2 error, running time, and memory and storage requirement), but our new estimator is more practical if computational resources are limited.

We have also developed a parallel algorithm for the new method. In the new estimator, the computational effort is mainly composed of two activities: the ray shooting tasks and the polynomial update tasks. As the latter one requires local information only, polynomial update tasks can be passed to the other idle processors without introducing too much communication overhead. This property leads to a load balancing strategy for the parallel implementation of our new estimator. We present the details of this parallel algorithm in Chapter 5. Finally, we give the conclusions in Chapter 6.

Chapter 2

Background and related work

2.1 Global illumination

In computer graphics terminology, rendering means producing a colored image from a description of a synthetic scene model. To compute the color, a set of rules are used for the computation, which take into account the set of light sources in the scene and the surface properties; for example, whether a surface is reflective or not. This set of rules are called the illumination model.

There are several illumination models. Some of them try to ignore certain physical properties during computation in order to gain efficiency. Some models try to handle as many physical properties as possible, in order to produce so called photo-realistic images.

The global illumination model is one of the photo-realistic models which tries to compute the energy emitted from light sources, reflected, or refracted from surfaces in the scene, and eventually arriving the viewpoint of an observer. Following the notation of [14], if we denote L as a light source, D a diffuse surface, S a specular surface, and E the viewpoint of a viewer, the global illumination model computes all the possible light paths from light sources to the viewer. These paths can be expressed in the form of a regular expression $L(D|S)^*E$.

Note that classical global illumination algorithms compute certain possible light paths and ignore the others. For example, recursive ray tracing computes $LD(S)^*E$. The classical radiosity algorithm assumes that all surfaces in the scene are perfectly diffuse, therefore the algorithm computes the paths $L(D)^*E$. Chen et al [3] divide all possible light paths into four classes: (i) direct illumination path LDS^*E; (ii) caustic path LS^+DS^*E; (iii) highlight path LS^*E; and (iv) radiosity path $L(S|D)^*DS^*DS^*E$. They propose a multi-pass algorithm which uses a combination of ray-tracing and radiosity to handle different light paths. Now, the multi-pass approach has become the major tool to compute global illumination.

In this thesis we study the first phase of the multi-pass approach, called view-independent global illumination. In view-independent global illumination, no view point is specified. The color of different surfaces is computed and stored on the surfaces using some representation methods, such as texture, piecewise constant functions, piecewise higher order functions, etc. Written in the form of a regular expression, the view-independent global illumination computes the light paths $L(D|S)^*D$. View-independent global illumination has the limitation that the lighting information is stored on diffuse surfaces only. Therefore, other lighting effects such as specular highlights are not handled. However, the results of view-independent global illumination allow for view-independent applications, such as virtual walkthroughs. The rapid growth of the Internet and the idea of Virtual Reality seeks a demand for efficient view-independent global illumination algorithm. In addition, the results can also be passed to the later phases of a multi-pass algorithm.

2.2 Density estimation framework

The first view-independent global illumination algorithm is the classical radiosity methods [12, 28]. It assumes that all surfaces in a scene are perfectly diffuse, and it computes all light paths $L(D)^*D$. Several works have been proposed to incorporate the specular surfaces in the computation [16]; however, as reported by [34], these methods are neither robust nor efficient. Recently, Jensen proposed the photon

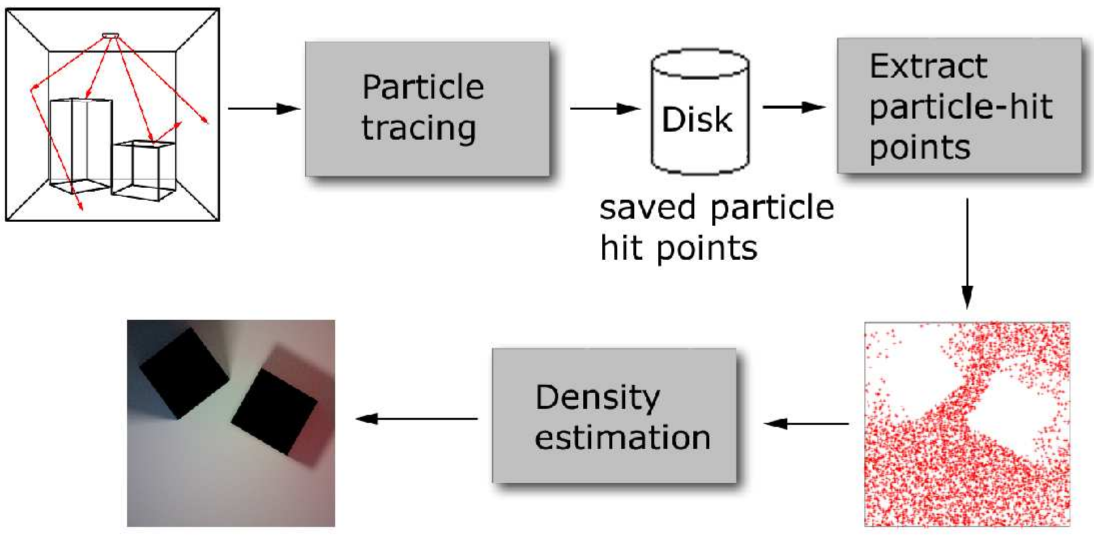


Figure 2.1: The density estimation framework.

mapping method to tackle the problem [17], which makes view-independent global illumination available for industry. Sillion and Puech [26] provide detailed background materials on this topic.

In this thesis we are interested in one of the approaches called density estimation. Walter et al [35] proposed a density estimation framework to solve this problem (Figure 2.1). Energy-carrying particles are emitted from each light source, traced through the environment, hit and reflected from surfaces until they are absorbed probabilistically. The surface hit points are stored in secondary storage devices. Illumination on a surface can then be estimated from the density of particle-hit points on the surface using the kernel density estimation method.

The density estimation framework has several advantages over the classical hierarchical radiosity method. As the particle tracing stage and surfaces' illumination estimation are independent of each other, most complex optical effects can be simulated. Moreover, the error in the surfaces' illumination estimation is not propagated. More complex surface characteristics are also supported. Curve surfaces are easily incorporated, whereas in the classical radiosity method they are required to be tessellated into a large number of triangles.

There are two remarks for this framework. The second step, i.e., storing the particle hit points in secondary storage, is only necessary when the kernel method is used for density estimation. Other density estimation methods can also be used to replace the kernel method in the framework, and most of these methods do not require to store the particle hit points. In Chapter 4 of this thesis we present an empirical comparison on different density estimation methods when they are applied to this framework.

The second remark is that this density estimation framework is a probabilistic technique, which is sometimes called Monte Carlo technique. The algorithm is simpler than the deterministic one (for example, the classical radiosity method), and most of the complex visual effects can be simulated. However, as the Monte Carlo approach usually requires a large number of samples in order to have a reliable result, it is usually not efficient. Veach [33] provided a detailed description on applying Monte Carlo methods for light transport simulation. Note that improving the efficiency of the density estimation framework is one of the most challenging problems.

2.3 Density estimation

The core part of density estimation framework is the density estimation stage. The density estimation problem can be stated as follow: derive a density function $\hat{f}(x)$ that approximates an unknown probability density function $f(x)$ from which independent samples $X_1, X_2, \ldots, X_n$ are drawn. Researchers in the computer graphics community [14, 25] have realized that in a particle-tracing radiosity method the illumination reconstruction process is a density estimation problem. In the density estimation framework, the particles-hit points are the sample points $\{X_i\}$, and the illumination function is a scaled probability density function $f(x)$. In this section, we review some of the commonly used techniques for solving the density estimation

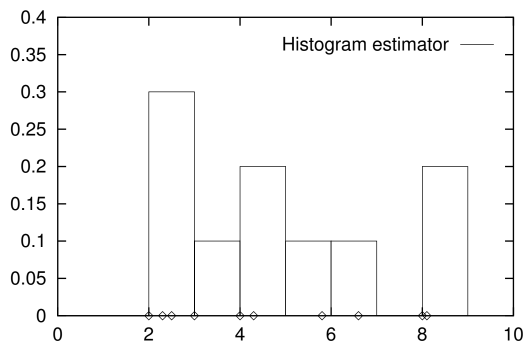


Figure 2.2: A histogram estimator.

problem that are related to our new method.

2.3.1 Histogram

The histogram is the oldest and most widely used density estimator. Suppose that the domain $[a, b]$ of a variable x is divided into sub-intervals $[a + mh, a + (m + 1)h)$, each with width h. Let n denote the number of samples drawn from $[a, b]$. The histogram estimator is defined by

$$\hat{f}(x) = \sum_{i=1}^{n} \frac{1}{nh} * (no.\ of\ X_i\ in\ the\ same\ bin\ of\ x)\ ,$$

Figure 2.2 shows an example of the histogram estimator for the samples {2, 2.3, 2.5, 3, 4, 4.3, 5.8, 6.6, 8, 8.1}. The "meshing" technique in computer graphics can be viewed as a two-dimensional implementation of the histogram method: surfaces in a virtual scene are subdivided into smaller size meshes called "patches", and the illumination of a patch is assumed to be piecewise constant. The approximate illumination of a patch is proportional to the number of particle-hits on the patch. More sophisticated "adaptive meshing" approaches, such as the one in [32], can also be viewed as a variation of the histogram method. In that case it allows sub-intervals to have different width, and the number of sub-intervals is changed adaptively.

2.3.2 Orthogonal series estimator

Meshing approaches assume the surfaces illumination to be piecewise constant. Surfaces are subdivided into smaller patches in order to represent illumination variations. In contrast, orthogonal series approaches represent surface illumination by a series of higher order functions.

Feda [8] proposed a Monte Carlo radiosity algorithm based on the orthogonal series estimator. Let $\{\phi_i, i \geq 0\}$ be a complete set of orthonormal basis functions on an interval I. Suppose that density function $f(x)$ is represented by:

$$f(x) = \sum_{i=0}^{\infty} a_i \phi_i(x) \ , \qquad x \in I \ .$$

where $a_i = \int_I f(x)\phi_i(x)dx$. An orthogonal estimator is then given by

$$\hat{f}(x) = \sum_{i=0}^{m} \hat{a}_i \phi_i(x) \ , \tag{2.1}$$

where $\hat{a}_i$ is an estimation of a_i. Since $\{X_i\}$ are drawn from the probability density function f,

$$a_i = \int_I f(x)\phi_i(x)dx = E[\phi_i(X)] \approx \frac{1}{n}\sum_{j=1}^{n} \phi_i(X_j) \equiv \hat{a}_i \ .$$

Note that only a finite number of terms (m in equation (2.1)) are used in the estimation.

2.4 Problem of existing approaches

Meshing approaches need to subdivide surfaces into smaller patches in order to represent illumination variations. Deciding how fine a surface should be subdivided is not an easy task. Adaptive meshing approaches try to overcome this problem by making the decision automatically during the particle tracing. However, as constant illumination is assumed over patches, it usually takes time before a surface is subdivided into sufficient small patches. Therefore, the convergence rates of these approaches are low. Moreover, to represent smooth illumination, a surface has to be subdivided into a large number of patches. Therefore, the memory requirement of these approaches is also high.

On the other hand, Feda's orthogonal series algorithm uses higher order basis functions to avoid the problem of constant illumination. However, the most important parameter (m in equation (2.1)) must be specified by the user. Just like deciding how to subdivide a surface in the histogram method, choosing an appropriate value of m for each surface is not an easy task. Moreover, when an illumination discontinuity exists on a surfaces, the orthogonal series estimator cannot give a good result without surface subdivision. This is one of the main problems to be addressed by our new method.

Chapter 3

The new adaptive density estimation method

3.1 Overview

The new method is an extension of Feda's algorithm [8]. Two modifications are added to the orthogonal series estimator. First, the new method determines automatically for each surface the appropriate number of terms that should be used in the orthogonal series. Second, when an illumination discontinuity entails an unacceptably large number of terms, the new algorithm subdivides the surface adaptively to better capture the illumination discontinuities. Hence the new method combines in a natural manner the advantage of orthogonal series estimator for modeling smooth variation of illumination and that of adaptive subdivision for modeling illumination discontinuities.

3.2 Choosing basis functions

We choose Legendre polynomials as the basis functions for quadrilateral surfaces. The one-dimensional Legendre polynomials are generated by the following recursive formula [31]:

$$\begin{aligned} P_0(x) &= 1 \\ P_1(x) &= x \\ (n+1)P_{n+1}(x) &= (2n+1)\ x\ P_n(x) - n\ P_{n-1}(x)\ , \end{aligned}$$

where P_n is a polynomial of degree n. The normalized Legendre polynomials are

$$\hat{P}_n(x) = \sqrt{n+\frac{1}{2}}\ P_n(x)\ ,$$

These polynomials are orthonormal over the domain $[-1, 1]$, i.e.,

$$\int_{-1}^{1} \hat{P}_n(x)\hat{P}_m(x)dx = \delta_{n,m}\ ,$$

where δ is the function

$$\delta_{n,m} = \begin{cases} 1\ , & \text{if } m = n; \\ 0\ , & \text{otherwise.} \end{cases}$$

For quadrilateral surface with parametric domain $D = \{(x, y) : -1 \le x \le 1, -1 \le y \le 1\}$, a two-dimensional basis set $\{\hat{Q}_{n,k}(x, y)\}$ can be generated by multiplying two one-dimensional polynomials in the two variables [31]:

$$\hat{Q}_{n,k}(x, y) = \hat{P}_{n-k}(x)\hat{P}_k(y)\ , \quad 0 \le k \le n\ .$$

It is easy to prove that $\{\hat{Q}_{n,k}(x, y)\}$ are orthonormal over the domain D:

$$\begin{aligned} &\int_{-1}^{1}\int_{-1}^{1} \hat{Q}_{n,k}(x, y)\hat{Q}_{m,j}(x, y)dxdy \\ &= \int_{-1}^{1}\int_{-1}^{1} \hat{P}_{n-k}(x)\hat{P}_k(y)\hat{P}_{m-j}(x)\hat{P}_j(y)dxdy \\ &= \int_{-1}^{1} \hat{P}_{n-k}(x)\hat{P}_{m-j}(x)dx \int_{-1}^{1} \hat{P}_k(y)\hat{P}_j(y)dy \\ &= \delta_{n,m}\delta_{k,j} \end{aligned}$$

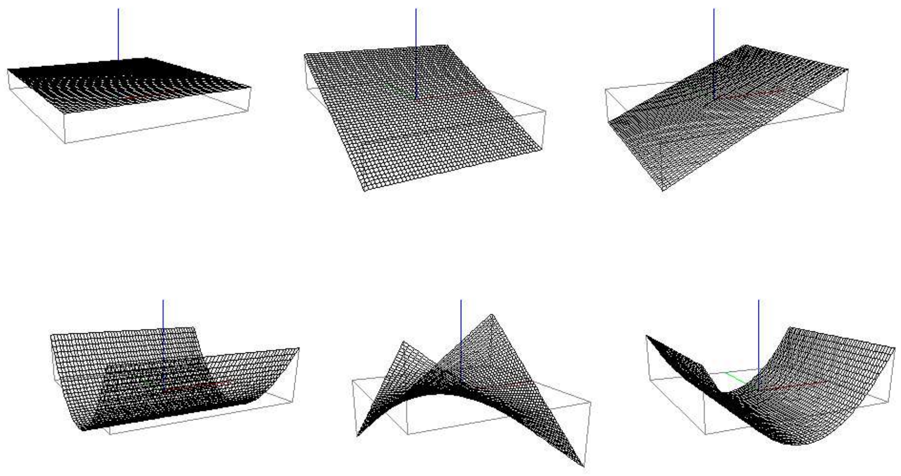


Figure 3.1: The first six terms of two-dimensional Legendre basis functions.

For example, the first 6 terms in the two-dimensional Legendre basis are:

$$\begin{array}{lll}
\hat{Q}_{0,0}(x,y) = \hat{P}_0(x)\hat{P}_0(y) & \hat{Q}_{1,0}(x,y) = \hat{P}_1(x)\hat{P}_0(y) & \hat{Q}_{2,0}(x,y) = \hat{P}_2(x)\hat{P}_0(y) \\
 & \hat{Q}_{1,1}(x,y) = \hat{P}_0(x)\hat{P}_1(y) & \hat{Q}_{2,1}(x,y) = \hat{P}_1(x)\hat{P}_1(y) \\
 & & \hat{Q}_{2,2}(x,y) = \hat{P}_0(x)\hat{P}_2(y)
\end{array}$$

For illustration purposes, figure 3.1 shows the three-dimensional plotting of these 6 terms.

Legendre polynomials are chosen as the orthogonal basis functions for the following reason: Fourier series and Legendre series are two common choices in the orthogonal series estimator for a finite domain. However, Hall [11] points out that Fourier series has the problem of "edge effects", referring to the large bias towards the endpoints of the domain. Legendre polynomials do not suffer from this deficiency. His paper concludes that Legendre series is better than Fourier series in the density estimator method.

Note that the new method is independent of the choice of basis functions, and that other basis functions can also be used. It might be worthwhile to study the influence of different basis functions on the method, however that is out of the scope of this thesis.

3.3 Error of an orthogonal series estimator

In order to find the most appropriate number of terms m in equation (2.1), we start from the formula of the error of an orthogonal series estimator. The performance of a density estimator is usually measured by the integrated-square error (ISE), defined by

$$ISE = \int_I [\hat{f}(x) - f(x)]^2 dx \ .$$

Note that $\hat{f}(x)$ in the formula depends on the samples, therefore the ISE is only concerned with the available samples $\{X_1, \ldots, X_n\}$. It is also appropriate to analyze the average error over all possible sample sets. The mean-integrated-square error (MISE) is therefore also of interest:

$$MISE = E\{ \int_I [\hat{f}(x) - f(x)]^2 dx \ \} \ ,$$

where the E operator concerns the expected performance of $\hat{f}(x)$ over all possible samples $\{X_1, \ldots, X_n\}$.

If an orthogonal series estimator is used, the formula can be written as:

$$\begin{aligned} MISE &= E\{ \int_I [\sum_{i=0}^{m} (\hat{a}_i - a_i)\phi_i(x) - \sum_{i=m+1}^{\infty} a_i \phi_i(x)]^2 dx \ \} \\ &= E[\sum_{i=0}^{m} (\hat{a}_i - a_i)^2 + \sum_{i=m+1}^{\infty} {a_i}^2] \\ &= \sum_{i=0}^{m} Var(\hat{a}_i) + \sum_{i=m+1}^{\infty} a_i^2 \ , \end{aligned} \tag{3.1}$$

where $Var(\hat{a}_i)$ is the variance of $\hat{a}_i$.

Equation (3.1) shows the well-known tradeoff between variance and bias in the density estimation problem [27]. The first term in equation (3.1) is the error introduced by the variance of the estimator (due to insufficient number of samples), and

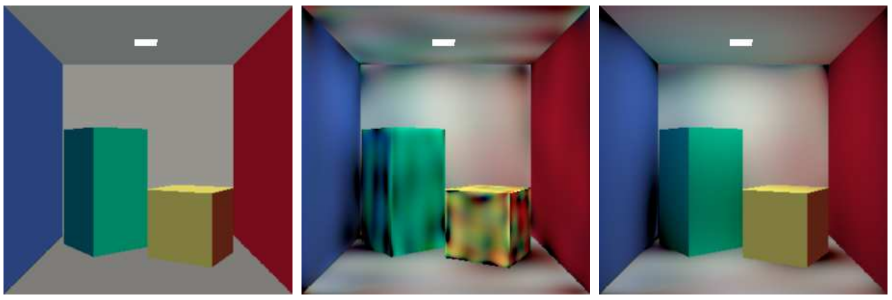


Figure 3.2: Inappropriately chosen values of m. From left to right: all surfaces use one single term, 45 terms, and the number of terms decided interactively by the user. The same number of particles (10^4) are traced for the three images.

the second term is the bias of the estimator (due to truncating the infinite series). If $n \to \infty$, $Var(\hat{a}_i) \to 0$; in this case bigger m leads to a smaller bias (the second term in equation (3.1)), and therefore leads to smaller error. However, if n is finite and relative small, the variance of $\hat{a}_i$ contributes significantly to the error. Therefore, it is necessary to set an appropriate value of m so that we may have balance between the first and the second terms of equation (3.1) to minimize $MISE$.

Figure 3.2 shows the result of inappropriately chosen values of m. In the left image all surfaces use only the first term in the series (i.e., the constant term), and the shading becomes too flat because truncating the infinite series introduced a large bias in the estimator. In the middle image all surfaces use 45 terms. For those surfaces with few particle-hits, there are too many terms in the series, and the variance in the coefficients $\hat{a}_i$ cause visual artifacts. In the right image, we pick values of m interactively for each surface. For those surfaces with few particle-hits or have smooth illumination, we choose a smaller m. For the other surfaces we choose a larger m. The resulting image has fewer visual artifacts.

Equation (3.1) can be further expanded as

$$MISE = \frac{1}{n-1} \sum_{i=0}^{m} \left[\, 2\hat{d}_i - (n+1)\hat{a}_i^2 \,\right] + \sum_{i=0}^{\infty} a_i^2 \;, \tag{3.2}$$

where $\hat{d}_i = \frac{1}{n}\sum_{j=1}^{n} {\phi_i}^2(X_j)$. Details of the expansion of (3.2) can be found in [20, 5]. Our aim is to choose a number m which minimizes the MISE, or equivalently, minimizes the function

$$J(m) = \frac{1}{n-1}\sum_{i=0}^{m} \left[\ 2\hat{d}_i - (n+1)\hat{a}_i^2\ \right],$$

since the second term on right hand side of (3.2) is independent of m.

In the next section we will present some observations on the characteristics of $J(m)$ in several typical radiosity setups. Then we will present a new orthogonal series estimator based the observations.

3.3.1 Characteristics of $J(m)$

We investigate the behaviour of the function $J(m)$, using several simple scenes shown in Figure 3.13. In the first three scenes, the receiver contains a smooth illumination, while in the last two scenes, the receiver's illumination contains discontinuities. We plot the function $J(m)$ against m, for m between [0..60], when the surface captures 400, 1600, and 6400 particles, respectively. The charts are shown in the middle column of Figure 3.13. The right column of Figure 3.13 plots the computed coefficients $\hat{a}_i$ against i.

From the charts, we have the following observations:

- the pattern of the function $J(m)$ falls into two categories: (i) for smooth illumination function, an optimal m, denoted as m^*, exists which minimizes $J(m)$. The function $J(m)$ increases (or remains unchange) for all $m \geq m^*$ (as shown in the middle column of the first three rows in Figure 3.13); (ii) for discontinue illumination function, there is no such "optimal" m. The function $J(m)$ decreases as m increases. Therefore, to reduce the error, we would have to use a very large value of m (as shown in the middle column of the last two rows in Figure 3.13).

- For smooth illumination functions, when the number of particle hits, denoted by n, is small (e.g. 400), the value of m^* is not "stable". When more particles are captured, m^* may shift to a larger value. However, the values of m^* for $n = 1600$ and $n = 6400$ are more or less the same. We believe that $n \geq 2000$ is sufficiently large for the value of m^* to become stable.

- The function $J(m)$ is not necessary monotonic, and may have multiple peaks in some situations. This is illustrated by the first scene shown in Figure 3.13, when $n = 400$. Since the illumination is symmetric on the receiver, there is no linear variation over the entire surface. Theoretically the coefficients for the linear terms ($m = 1, 2$) should be zero, and including these terms in the series does no harm. However, as variance exists, the computed coefficients $\hat{a}_1$ and $\hat{a}_2$ are non-zero, and including these terms will even increase the error. Including later terms in the series (e.g. the quadratic terms when $m = 3, 4, 5$) reduces the error.

We also study the function $J(m)$ for two more general scenes. The first scene is the Cornell box with one light source and 18 surfaces, and the second scene is an office scene with 1 light source and more than 300 surfaces, as shown in Figure 3.3. We observe that the function $J(m)$ in those scenes exhibits similar behaviour as the one in the simple scenes in Figure 3.13. Moreover, for all of the scenes we studied, the value of m^* for most surfaces are clustering within the range [0..30], and only few surfaces (those with illumination discontinuities) require a value of m^* outside this range. Within this range, the values of m^* seem to be distributed in a rather random manner, and are, in fact, strongly dependent on the smoothness of illumination distribution on a surface. This observation suggests that it would be an extremely difficult and tedious task for a human user to select by inspection the optimal value of m for each surface even in a small scene. This problem acutely demands an automatic method for selecting the best value of m. Based on established results in statistics, we propose a new algorithm that solves the above problem satisfactorily.

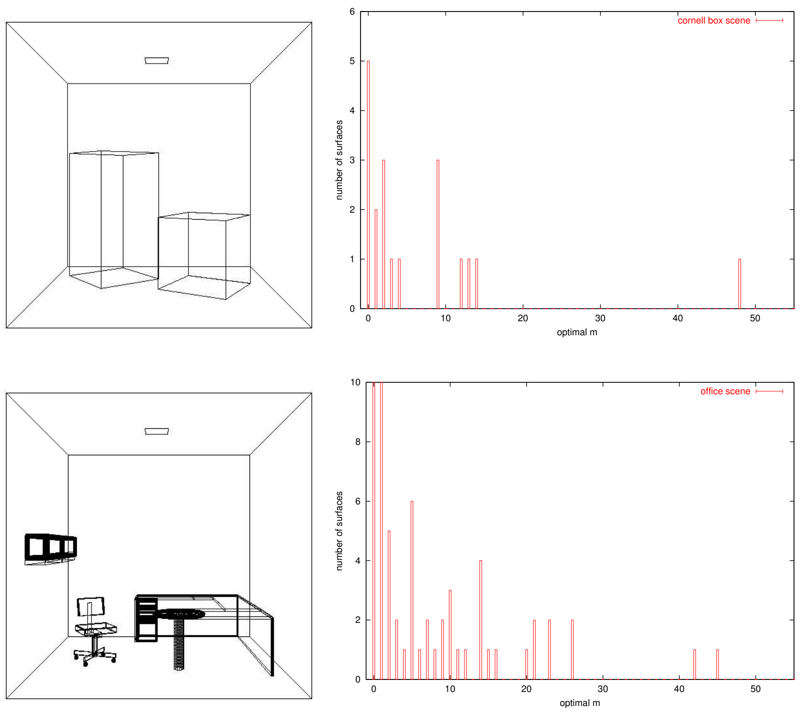


Figure 3.3: Distribution of optimal m in the Cornell box scene (top) and the office scene (bottom).

3.4 The new method

3.4.1 Automatic determination of values of m

Since $J(m)$ may have multiple peaks, to avoid being trapped at a local minimum one cannot simply increase m incrementally until $J(m+1) > J(m)$. Rather, the following strategy is adopted. Based on our extensive testing, we decide $M = 30$ to be an appropriate upper bound on the number of terms that will be used in the orthogonal estimator for any surface. $M = 30$ is appropriate in the sense that it is large enough to allow enough terms to be used for modeling smooth illumination accurately and, at the same time, not too big to compromise computational efficiency; after all, if the optimal value of m exceeds $M = 30$, it is normally the case where there is strong illumination discontinuity, and in this case it should be more efficient

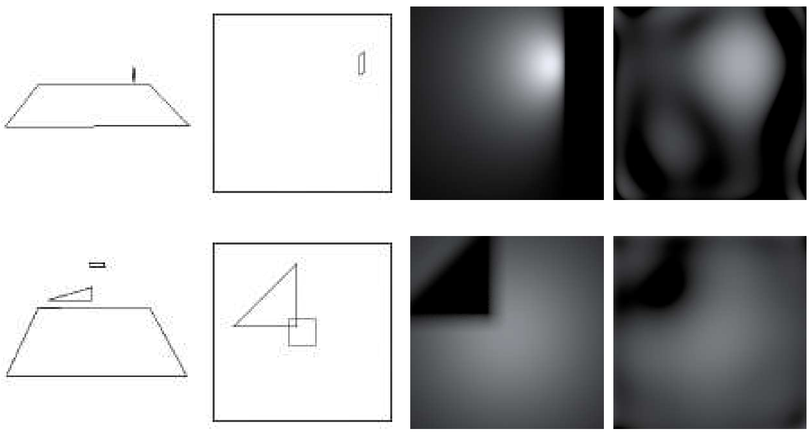


Figure 3.4: Results without surface subdivision. From left to right: the front view, top view, analytical solution, and adaptive orthogonal function approximation. In both cases $m = M = 45$ is reached.

to subdivide the surface rather than using excessively many terms in the estimating series. With M having been fixed, we choose a value m^* of m that minimizes $J(m)$ for all $m \leq M$. We reiterate that the value $M = 30$ may not be large enough for surfaces with illumination discontinuities. This situation will be handled in our method by using adaptive surface subdivision.

3.4.2 Adaptive surface subdivision

When an illumination discontinuity exists on a relatively large surface, the estimated function based on orthogonal series alone cannot fit the illumination function very well unless an impractically large number of terms are used. Figure 3.4 shows two simple scenes illustrating this situation. The images show that even though 45 terms are used, the illumination still fails to produce a good approximation. Moreover, in the region with low gradient the visual artifact is even worse. It is because if too many number of terms in the series are used, the coefficients in the high order terms will have large variance. As the high order polynomials have several peaks in the basis functions, this will cause some noticeable oscillations appear in the images.

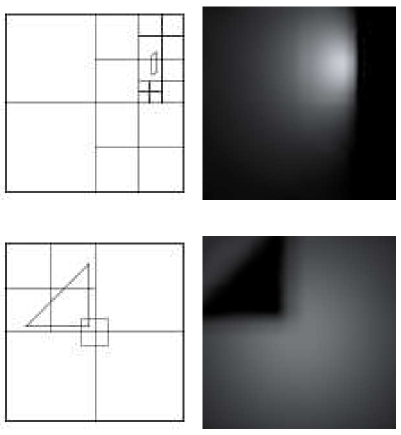


Figure 3.5: The results of the new estimator with surface subdivision.

To overcome this problem, a surface subdivision scheme is added to the estimator. When an appropriate number of terms determined by the algorithm reaches a predefined value M, the algorithm subdivides the surface. After subdivision, each subsurface will have its own density estimation function, and the late-coming particles hit on the surface will be captured by a sub-surface. Note that the subdivided surfaces cannot discard the $\{\hat{a}_i\}$ it has computed so far, because those $\{\hat{a}_i\}$ correspond to the contribution of energy carried by the previous particles captured by the surface before. Therefore, this estimated function should be added in an appropriate manner to the sub-surfaces' illumination estimation at the rendering stage.

There is one problem if surface subdivision is added to the estimator: the estimated functions of adjacent sub-surfaces may not be guaranteed to be consistent along subdivision boundaries. As a result, visual artifacts will appear in the final image. An interpolation technique is used to alleviate the problem: at a region close to the subdivision boundary, the estimated functions of the sub-surfaces will be interpolated by a blending function. Figure 3.6 illustrates the process in 1D. Note that interpolation is applied only at the image-rendering stage as a tools to remove visual artifact. It will not have any effect on the density estimator. Figure 3.5 shows the results of the new estimator with surface subdivision, with the same setups as shown

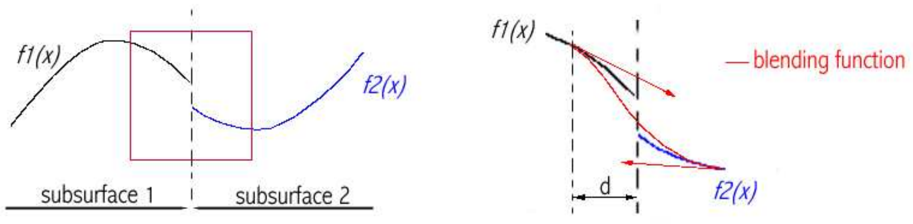


Figure 3.6: The interpolation of two functions near a subdivision boundary. Left: the two estimated functions do not agree across the subdivision boundary. Right: zoom view of the red rectangular region in the left image. A blending function is used to interpolate the two functions in the region with distance $\leq d$ from the boundary (d is a pre-defined constant).

in Figure 3.4.

3.5 Comparison with Tobler et al approach

Tobler et al [32] give an adaptive meshing algorithm for the particle-tracing radiosity method. Their algorithm allows the surface to have higher order illumination function terms rather than only constant ones. Despite the resemblance of the new method to their method, these two methods have the following differences:

- In their method the number of terms used for each surface is specified by the user, whereas in the new method it is determined automatically. This makes the new method more practical for complex scenes.

- Their method focuses on adaptive meshing. At each level (the “current level”), the algorithm uses a “preview level” to keep track of particle-hits in a finer surface resolution. By comparing the illumination function of the “preview level” and the “current level”, the algorithm decides whether the surface should be subdivided or not. In contrast, the new method emphasizes determining the suitable number of terms used in the series for each surface. Thus no “preview level" is necessary, and a surface is subdivided only when the number of terms determined is larger than acceptable.

- In their method, after a surface is subdivided, the sub-surfaces use the same number of terms in the series as their parent surface. In the new method, the number of terms used by each of the sub-surfaces may differ, and is adaptively determined. This is a proper treatment, because when a surface is subdivided, the sub-surfaces will tend to have smoother illumination than their parent; consequently, they should use a smaller number of terms than their parent.

3.6 Experiments and results

Four different methods have been implemented for comparison purpose: (1) *AM-C*: Tobler et al [32] adaptive meshing method with constant illumination; (2) *AM-H*: same as (1), but with higher order function on each surface (the number of terms m is defined by the user); (3) *FOSE*: Feda [8] orthogonal series estimator with a user defined m and without surface subdivision; (4) *NEW*: the new method. These methods are applied to the Cornell-box scene, and the running time, memory used, and L_2 *error* (MISE) of two surfaces against the number of particles traced are plotted. The surfaces chosen for error analysis are the floor, which is rough and has large variation in illumination, and the right red wall which is smooth with small variation in illumination. For the methods *AM-H* and *FOSE* 45 terms for the floor and 15 terms for the red wall are used. A reference solution is generated by shooting 10^8 particles to the scene, where each surface has a texture of size 200×200 to capture particle hits.

The results are shown in figures 3.14 and 3.15. A sequence of images are generated, as shown in figure 3.17, by increasing the number of particles traced. The images and the statistics results show that:

- *error:* the *FOSE* method gives the largest error. The error will be bounded from below by some value and cannot be improved anymore if surface subdivision is not used, as the degree is limited. Adaptive meshing methods (*AM-C* and *AM-H*) produce smaller error due to surface subdivision, but *AM-C* uses a lot of memory to store the meshes structure in order to achieve this small error level. The *AM-H* method avoids the storage problem, but that the user has to choose a suitable m for each surface makes this method impractical. The new method generates smaller error than the other three methods (if we ignore the case when there are too few particles traced, e.g. less than 10000), and it does not require the user to specify the number m for each surface.

- *running time:* as shown in figure 3.15, the *AM-H* method takes the longest time to trace the same number of particles. That is because for every particle-hit the method has to evaluate a high order illumination function for both the "preview level" and the "current level". The new method takes longer time than *AM-C* and *FOSE* because it has to evaluate the function $H(m)$ for all $m \leq M$, in return for a more accurate illumination estimation.

In summary, to achieve the same error level, the new method needs to shoot fewer particles and runs faster. Figure 3.16 plots the L_2 error against the running time used in the four methods. These plots show the superiority of the new method.

3.7 Extension to general cases

3.7.1 Triangles

So far we have only dealt with quadrilateral surfaces. However, in a virtual scene most of the objects are made up of triangles. Moreover, most modeling packages in computer graphics industry tessellate all the input primitives into triangles. However, the basis functions given in section 3.2 are not orthogonal over a triangular domain. In this section we describe the method given in [31] and [19] to construct an orthonormal basis set which can be used on triangular domain.

The method starts with the one-dimensional Jacobi polynomials. With arbitrary

$\alpha, \beta > -1$, the one-dimensional Jacobi polynomials are generated by the following recursive formula:

$$\begin{aligned}
J_0^{(\alpha,\beta)}(x) &= 1 \ , \\
J_1^{(\alpha,\beta)}(x) &= \frac{1}{2}[(\alpha+\beta+2)\ x + (\alpha-\beta)] \ , \\
J_{n+1}^{(\alpha,\beta)}(x) &= \frac{(2n+\alpha+\beta+1)(2n+\alpha+\beta+2)}{2(n+1)(n+\alpha+\beta+1)}\ x\ J_n^{(\alpha,\beta)}(x) \\
&+ \frac{(2n+\alpha+\beta+1)(\alpha^2-\beta^2)}{2(n+1)(n+\alpha+\beta+1)(2n+\alpha+\beta)}\ J_n^{(\alpha,\beta)}(x) \\
&- \frac{(n+\alpha)(n+\beta)(2n+\alpha+\beta+2)}{(n+1)(n+\alpha+\beta+1)(2n+\alpha+\beta)}\ J_{n-1}^{(\alpha,\beta)}(x) \ .
\end{aligned}$$

The normalized Jacobi polynomials are:

$$\hat{J}_n^{(\alpha,\beta)}(x) = \sqrt{\frac{n!(\alpha+\beta+n+1)!(\alpha+\beta+2n+1)}{2^{\alpha+\beta+1}(\alpha+n)!(\beta+n)!(\alpha+\beta+n+1)}}\ J_n^{(\alpha,\beta)}(x) \ .$$

The Jacobi polynomials are orthogonal over the domain $[-1, 1]$ with respect to weight function $(1-x)^\alpha(1+x)^\beta$. Therefore,

$$\int_{-1}^{1} (1-x)^\alpha(1+x)^\beta\ \hat{J}_n^{(\alpha,\beta)}(x)\ \hat{J}_m^{(\alpha,\beta)}(x)\ dx = \delta_{n,m} \ .$$

Note that Legendre polynomials are special case of Jacobi polynomials, with both α and β are set to zero.

Now, with arbitrary $\alpha, \beta, \gamma > -1$, define

$$A_{n,k}(x,y) = A_{n,k}^{(\alpha,\beta,\gamma)}(x,y) = (1-x)^k\ \hat{J}_{n-k}^{(2k+\beta+\gamma+1,\alpha)}(2x-1)\ \hat{J}_k^{(\gamma,\beta)}(\frac{2y}{1-x}-1) \ .$$

It can be shown that the polynomials $\{A_{n,k}(x,y)\}$ are orthogonal over the triangular domain $T = \{(x,y) :\ 0 \le x, y \le 1,\ 0 \le x+y \le 1\}$ with respect to the weight function $x^\alpha y^\beta(1-x-y)^\gamma$:

$$\begin{aligned}
I = &\int_{x=0}^{1}\int_{y=0}^{1-x} x^\alpha y^\beta (1-x-y)^\gamma A_{n,k}(x,y) A_{m,s}(x,y)\ dxdy \\
= &\int_{x=0}^{1} (1-x)^{k+s} x^\alpha\ \hat{J}_{n-k}^{(2k+\beta+\gamma+1,\alpha)}(2x-1)\ \hat{J}_{m-s}^{(2s+\beta+\gamma+1,\alpha)}(2x-1) \ * \\
&\quad \left[\int_{y=0}^{1-x} y^\beta(1-x-y)^\gamma\ \hat{J}_k^{(\gamma,\beta)}(\frac{2y}{1-x}-1)\ \hat{J}_s^{(\gamma,\beta)}(\frac{2y}{1-x}-1)\ dy\right] dx \ .
\end{aligned}$$

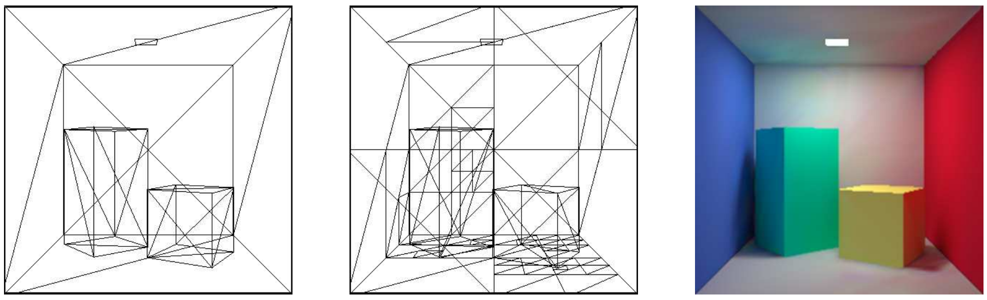


Figure 3.7: Cornell box made up with triangles. From left to right: surfaces before subdivision, after subdivision, and the final color image.

Let $\frac{2y}{1-x} - 1 = t$. Then

$$I = \int_{x=0}^{1} (1-x)^{k+s} x^{\alpha}\ \hat{J}_{n-k}^{(2k+\beta+\gamma+1,\alpha)}(2x-1)\ \hat{J}_{m-s}^{(2s+\beta+\gamma+1,\alpha)}(2x-1)\ * \\ \left[(\frac{1-x}{2})^{\beta+\gamma+1} \int_{t=-1}^{1} (1+t)^{\beta}(1-t)^{\gamma}\ \hat{J}_{k}^{(\gamma,\beta)}(t)\ \hat{J}_{s}^{(\gamma,\beta)}(t)\ dt \right] dx\ .$$

The term inside the square brackets is zero if $k \neq s$. If $k = s$,

$$I = \frac{1}{2^{\beta+\gamma+1}} \int_{x=0}^{1} (1-x)^{2k+\beta+\gamma+1} x^{\alpha}\ \hat{J}_{n-k}^{(2k+\beta+\gamma+1,\alpha)}(2x-1)\ \hat{J}_{m-k}^{(2k+\beta+\gamma+1,\alpha)}(2x-1)\ dx\ .$$

Let $2x - 1 = t$. Then

$$\begin{aligned} I &= \frac{1}{2^{2k+2\gamma+2\beta+\alpha+3}} \int_{t=-1}^{1} (1-t)^{2k+\beta+\gamma+1}(1+t)^{\alpha}\ \hat{J}_{n-k}^{(2k+\beta+\gamma+1,\alpha)}(t)\ \hat{J}_{m-k}^{(2k+\beta+\gamma+1,\alpha)}(t)\ dt \\ &= \frac{1}{2^{2k+2\gamma+2\beta+\alpha+3}}\ \delta_{n,m}\ . \end{aligned} \tag{3.3}$$

To simplify the computation, we pick $\alpha = \beta = \gamma = 0$ so that the weight function become 1. Also, from equation (3.3), we know that $A_{n,k}(x,y)$ can be normalized by multiplying the term $\sqrt{2^{2k+3}}$. As a result, the orthonormal basis functions we used for triangle are

$$\hat{A}_{n,k}(x,y) = \sqrt{2^{2k+3}}\ (1-x)^{k}\ \hat{J}_{n-k}^{(2k+1,0)}(2x-1)\ \hat{J}_{k}^{(0,0)}(\frac{2y}{1-x} - 1)\ .$$

Figure 3.7 shows the result of the Cornell box when the surfaces are made up of triangles. It can be seen that the color picture generated is almost the same as the one with quadrilateral surfaces.

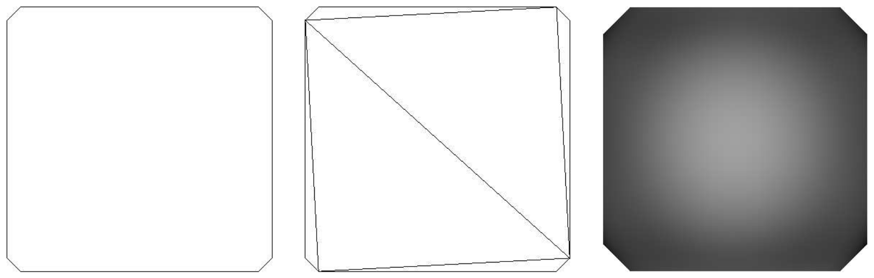


Figure 3.8: Tessellate an arbitrary planar surface. From left to right: surface before subdivision, after subdivision, and the final color image.

3.7.2 Arbitrary planar surfaces

We may tessellate arbitrary planar surfaces into triangles before passing them to the algorithm. Figure 3.8 shows an octagon before subdivision, after tessellated into triangles, and the color image produced by our algorithm. It should be noticed that tessellation may sometimes produce thin and ill-shape triangles, as shown in figure 3.8. This will usually result in poor illumination continuity along the subdivision boundaries. A better subdivision scheme, such as the one proposed in [1], can be applied to produce fewer ill-shape triangles.

3.7.3 Bicubic surfaces

A bicubic surface is defined by the parametric representation

$$Q(u,v) = \sum_{i=0}^{3} \sum_{j=0}^{3} p_{ij} b_i(u) b_j(v) \ ,$$

where the p_{ij} are the control points, and the $b_i(u) * b_j(v)$ are the blending functions. As the basis functions we used is defined on parametric space, our algorithm can be directly applied to bicubic surfaces. It treats bicubic surfaces and quadrilateral surfaces equally. Figure 3.9 shows a simple scene containing a Bezier surface and the image produced by our algorithm.

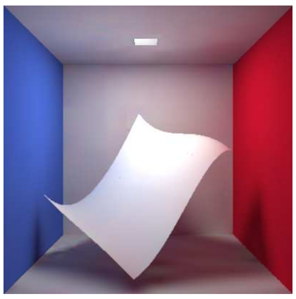


Figure 3.9: Simple scene contains a bezier surface.

3.7.4 Quadric surfaces

If constant illumination is used, quadric surfaces, such as cylinders, cones and spheres in scene are usually needed to be tessellated into a large number of triangles or quadrilaterals. This will increase the ray shooting time because the number of input primitive increases. Moreover, tessellation also introduces color inconsistencies along the subdivision boundaries.

On the other hand, quadric surfaces can also be formulated using parametric representation $Q(u, v)$. Therefore our algorithm can also be directly applied to quadric surfaces. Figure 3.10 shows a simple scene containing several quadric surfaces. It shows that using higher order basis functions, our algorithm produces smoother illumination on quadric surfaces than their tessellated counterpart.

However, it should be noticed that in order to apply our algorithm, the parametric representation of the quadric surfaces should have uniform area distribution. This is similar to the problem of texture mapping: we want to have a parametric representation such that mapping a texture space $[u, v]$ onto a quadric surface will result in less distortion. If the representation does not have uniform area distribution, distortion of estimated illumination will occur, and visual artifacts will appear in the final image. For example, in the middle image of figure 3.10, we used a parametric representation as shown in figure 3.11. Figure 3.11 also shows the area distribution of the corresponding parametric representation. As the differential area at the poles of

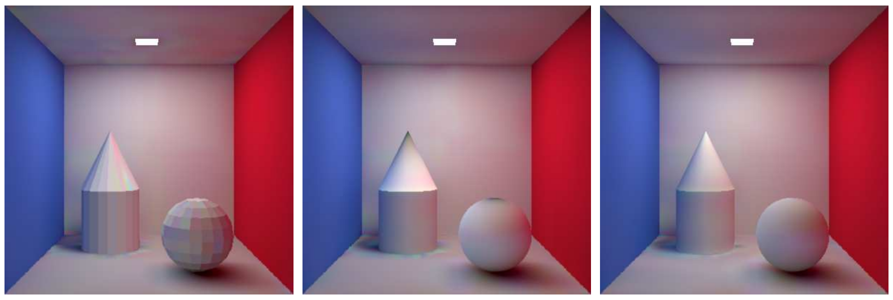


Figure 3.10: Simple scene contains several quadric surfaces. In the left image quadric surfaces are tessellated, and in the middle image each quadric surface is treated as a single surface. In the right image a correct parametric representation is used.

sphere and the apex of cone approaches zero, nearly no particle hit is recorded near these locations, and the estimated illumination becomes dark. Figure 3.12 shows the parametric representation we used to correct the distortion, and the corrected result is shown in the right image of figure 3.10.

3.8 A practical example

Figure 3.18 shows a synthesized image of the graphics laboratory at the University of Hong Kong generated by the new method. The model contains 16000 triangles. In total 10^7 particles are shot from the light sources, and roughly 10^8 particles are traced. The computation takes 5.5 hours on a SGI MIPS R10000 195MHZ processor, with 512MB main memory. This example shows that the new method is more suitable than the other three methods for complex scenes, because it uses less memory, produces a better rendering quality, and there is no need to specify the parameter m for each surface.

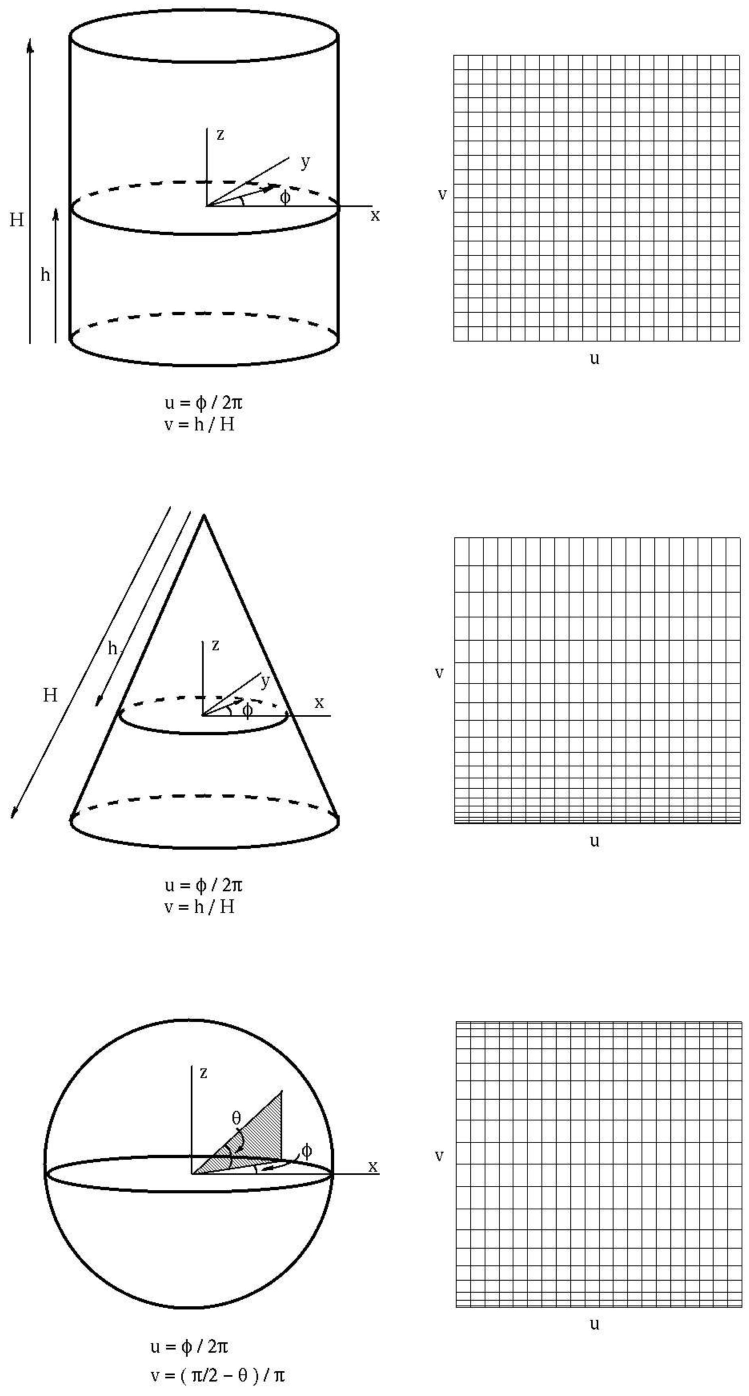


Figure 3.11: Parametric representation of quadric surfaces we used before correction.

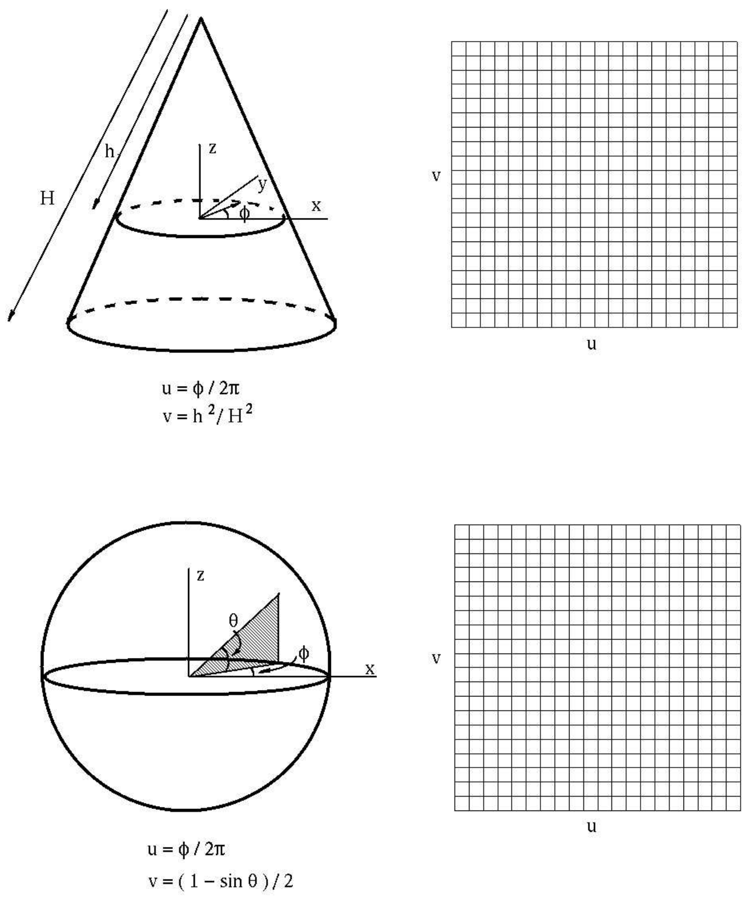


Figure 3.12: Parametric representation of quadric surfaces we used to correct the artifacts in the middle image of figure 3.10.

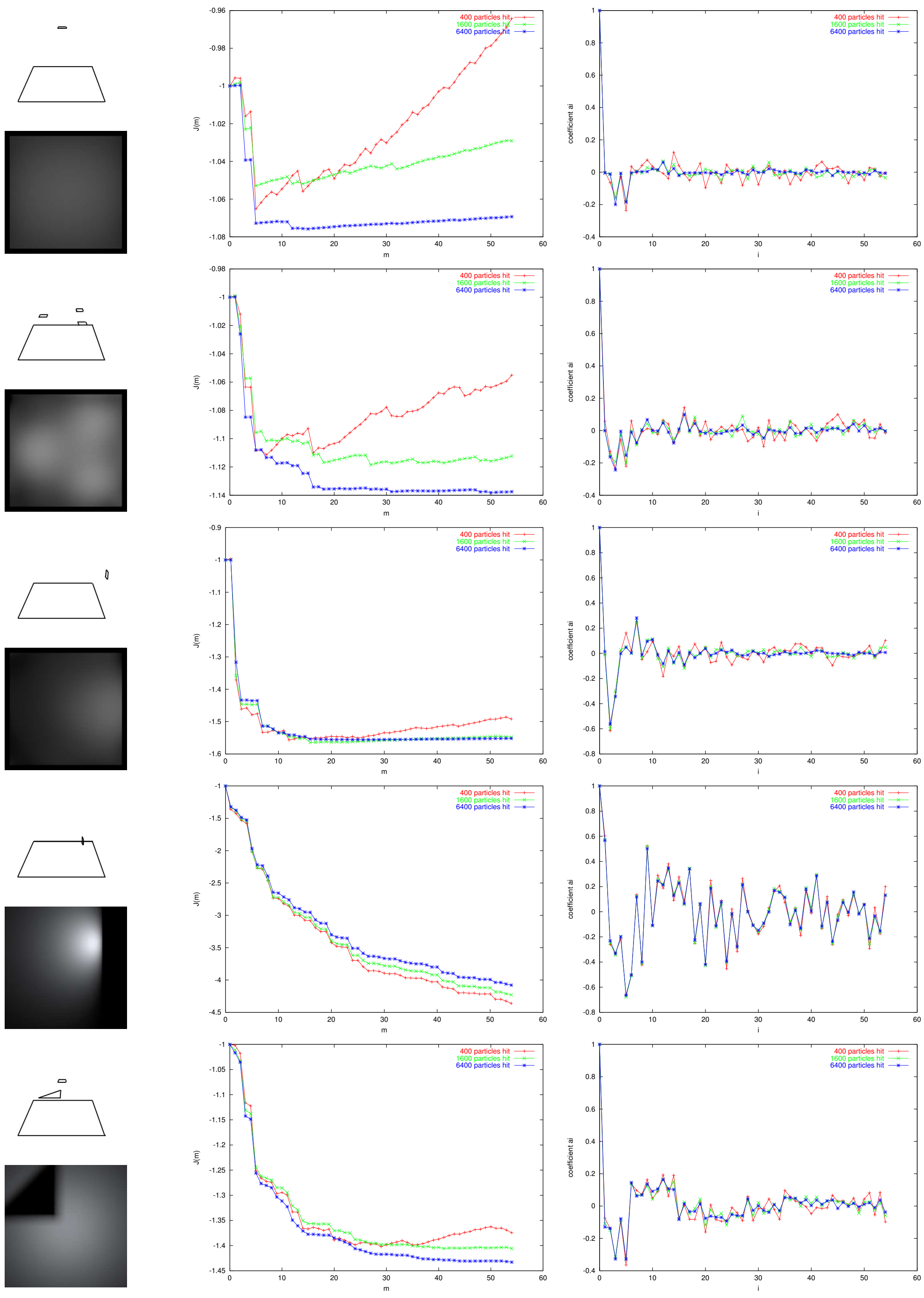


Figure 3.13: The behaviour of $J(m)$. From left to right: the scene setup, $J(m)$ against m, and the coefficients $\hat{a}_i$ against i.

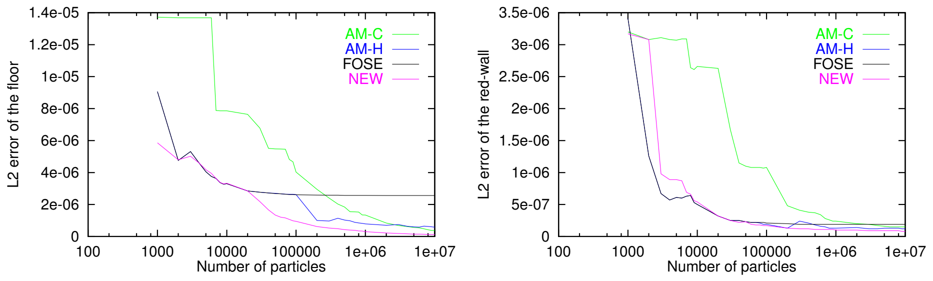


Figure 3.14: The L_2 error of the methods.

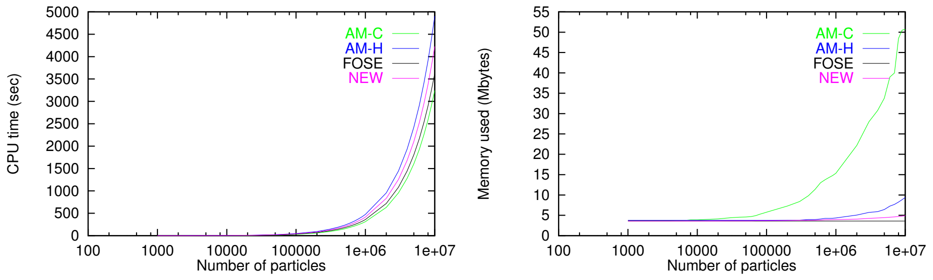


Figure 3.15: CPU time and memory used by different methods.

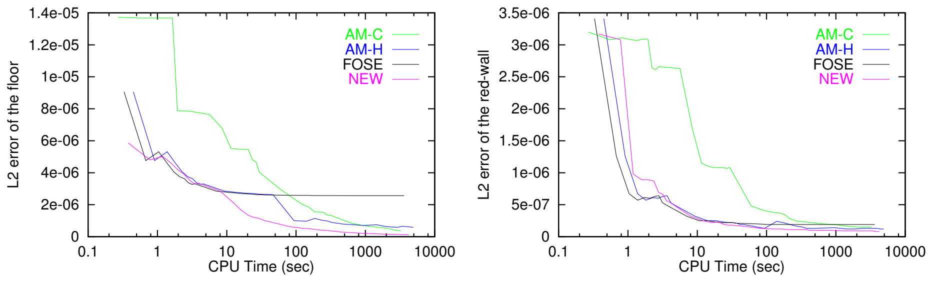


Figure 3.16: L_2 error against running time.

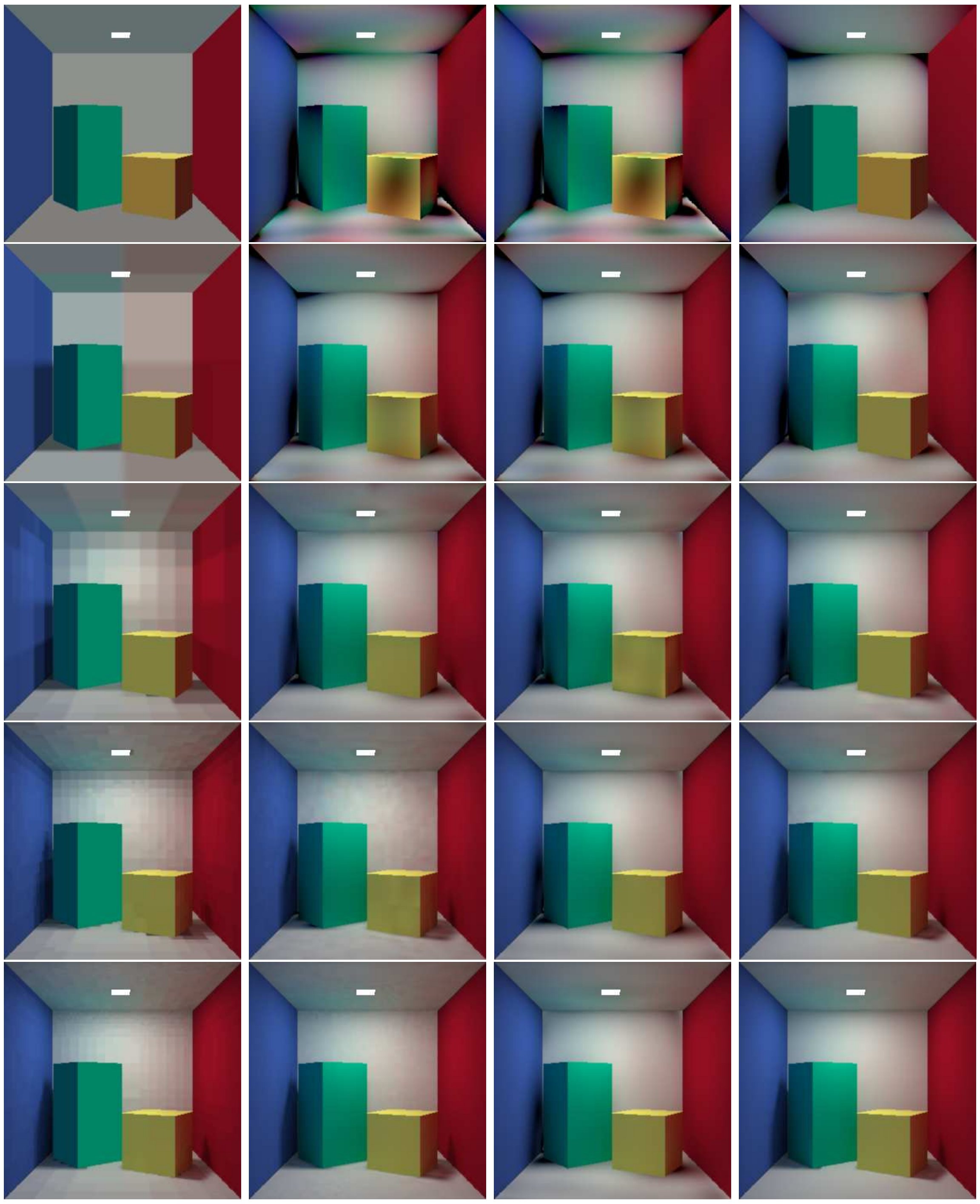


Figure 3.17: Final rendered images. From left to right: *AM-C, AM-H, FOSE, NEW*. From top to bottom: 10^3, 10^4, 10^5, 10^6, and 10^7 particles are traced.

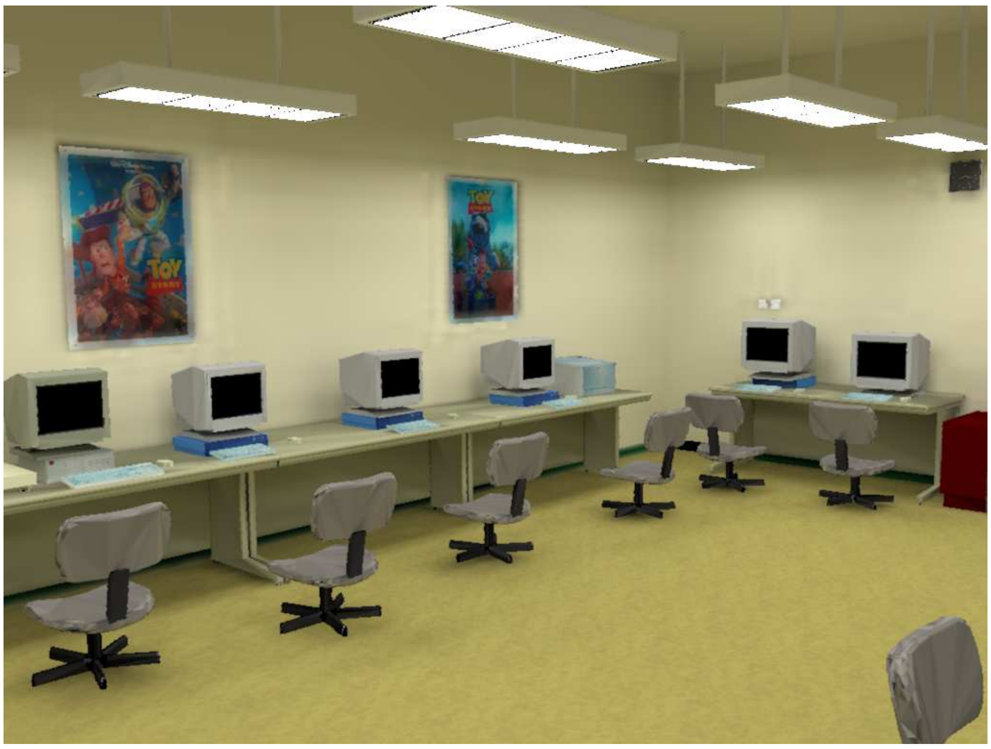


Figure 3.18: A complex scene rendered by the new method.

Chapter 4

Empirical comparison of different density estimators

In Walter et al [35] original density estimation framework, they proposed to use the kernel method for the density estimation. They also suggested a heuristic procedure to choose the tuning parameter (the *kernel width*). After their paper there have been some new results on density estimation published in the statistics literature. For example, the wavelet thresholding method [7] has gained certain attention in the recent years. However, there has been no results showing which method is more appropriate for the global illumination problem.

In Chapter 3 we showed that our new density estimation method outperforms the other similar adaptive meshing approaches. In this chapter we compare the method with other approaches that do not belong to the adaptive meshing category. We compare three density estimation methods (our new method, kernel method, and wavelet thresholding method), both qualitatively (by comparing the rendered images) and quantitatively (by measuring the error from reference images). We also compare the running time and the memory requirement of these methods. We conclude, at the end of this chapter, that each estimator has its own shortcomings in some aspects, but our method is more practical if computational resources are limited.

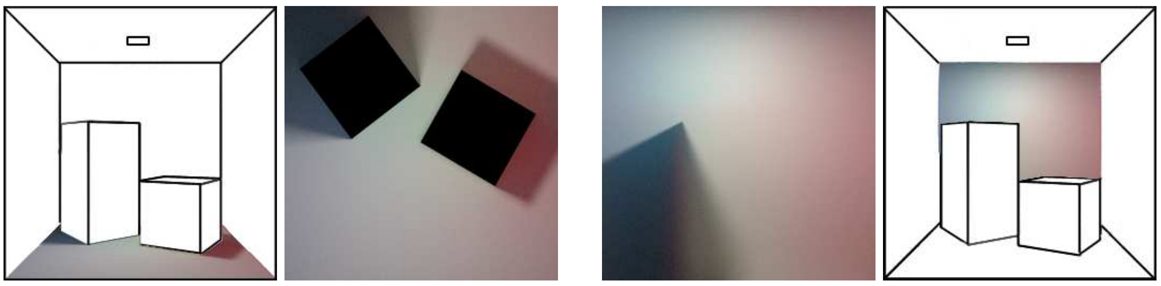


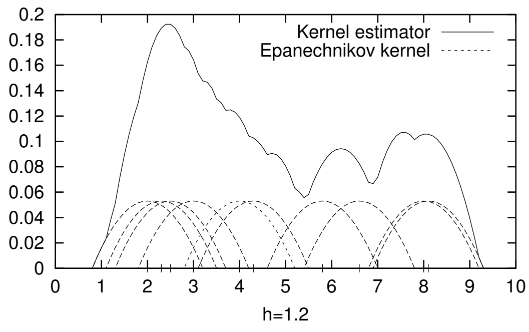
Figure 4.1: Reference images of the two surfaces chosen for error analysis.

4.1 Experiment setup

Our experiments were done on an Intel Pentium III 600MHz PC with 512MB main memory. We follow the framework described in Chapter 2. The Cornell box, as shown in figure 2.1, is used as the input scene. Up to 1 million particles (for each color channel) are shot from the light source. As particles may reflect from the surfaces they hit, totally 6.6 million particled are traced, generating 3.6 million particle-hit points.

We wanted all methods under comparison using the same set of particle-hit points. Therefore, we stored the particle-hit points on each surface are stored in secondary storage (used totally 47MB disk space). The file is stored in the binary format. Each entry in the file corresponds to one particle-hit, which contains one integer representing the *id* of the surface hit, two floating values representing the parametric coordinates of the hit point, and one character representing the color channel (R for red, G for green, and B for blue). We trace and estimate the three color channels independently.

Based on the particle-hit points, we perform density estimation on the surfaces. Two surfaces are chosen for the experiment: the floor, which contains two dark area with sharp shadow boundaries; and the facing wall, which has smooth illumination over the surface, but has a soft shadow at one of its corners. Figure 4.1 shows the reference images of these two surfaces. Details of the three density estimation methods



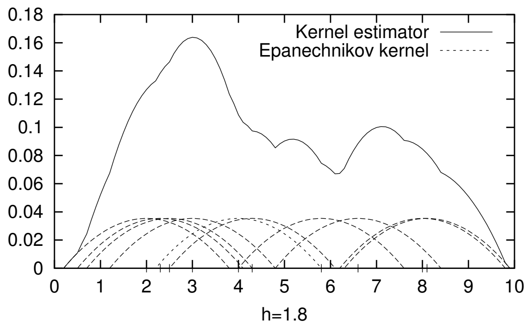


Figure 4.2: Kernel estimators with different kernel widths.

are given in the next section. We estimate the illumination at 256x256 grid points on the surface, and render a color image for each of the results (figure 4.4 and 4.5).

The reference images are generated by the Monte Carlo Path Tracing method [18], using 1024 samples for each pixel. The size of the reference images is also 256x256 pixels, so that we can measure the error of the estimated images pixel-by-pixel. To measure the performance of an estimator quantitatively, we used the *integrated-square-error ISE* (also called L_2 *error*):

$$ISE = \int [\hat{f}(x) - f(x)]^2 dx$$

where $f(x)$ is the unknown function and $\hat{f}(x)$ is the estimated function.

4.2 Other density estimators being compared

4.2.1 Kernel estimator

The kernel estimator is the most commonly used method for density estimation, and there are vast theories and results available in the statistics literature. Let $K(x)$ be a kernel function satisfying the condition $\int K(x)dx = 1$. The scaled kernel $K_h(x)$ with kernel width h (also called the *bandwidth*) is defined as $K_h(x) = \frac{1}{h}K(\frac{x}{h})$. With sample size n, the estimated function is defined as

$$\hat{f}_h(x) = \frac{1}{n}\sum_{i=1}^{n} K_h(x - X_i) \tag{4.1}$$

We can think that, for each sample point X_i, a kernel is placed centered at the sample point. The estimated function is the sum of these n kernels. There are different choices for the kernel function. Figure 4.2 shows examples of kernel estimators with different kernel widths. The kernel function used is the Epanechnikov kernel as suggested in [35], defined as

$$K(x) = \begin{cases} \frac{2}{\pi}(1 - |x|^2) & \text{if } |x| \leq 1 \\ 0 & \text{otherwise.} \end{cases}$$

From figure 4.2 we see that the choice of h affects the resulting estimation, and choosing a suitable h is not an easy task [25]. The kernel width should be narrow enough to capture the details in density distribution, and should be wide enough to avoid spurious fine structures.

Heuristic kernel width

Walter et al [35] suggested a heuristic choice for the kernel width. In their paper, h_i for surface i is chosen as:

$$h_i = \sqrt{\frac{CA_i}{n_i \pi}} \ ,$$

where A_i is the area of surface i, n_i the number of particles hit on it, and C is a user-specified parameter. The value C is equals to the average number of particle-hits that the kernel should cover. In [35], C is suggested to be a value between 4,000 and 16,000.

Automatic bandwidth selection

Instead of heuristic choice, several methods [4] have been proposed to select a bandwidth automatically. Most of them try to minimize the L_2 error. Our experiments follow Sheather and Jones' approach [24], since it is the most popular "plug-in selector" used in this area.

It is well known [27] that the L_2 error of any density estimator $\hat{f}$ is equal to:

$$ISE = bias(\hat{f})^2 + variance(\hat{f}) \ , \tag{4.2}$$

The bias and variance of a kernel estimator with kernel K and width h are given by [27]

$$bias(\hat{f}_h)^2 \approx \frac{1}{4} h^4 \sigma_2^2 \int f''(x)^2 dx$$
$$variance(\hat{f}_h) \approx \frac{1}{nh} \int K(t)^2 dt \ ,$$

where $\sigma_2 = \int t^2 K(t) dt$.

Substituting the above formulas into (4.2), an optimal h that minimizes the ISE is given by

$$h_{opt} = \left[\frac{\int K(t)^2 dt}{n \ \sigma_2^2 \ \int f''(x)^2 dx} \right]^{1/5} . \tag{4.3}$$

However, the term $\int f''(x)^2 dx$ is unknown. This term can be estimated, using the sample points again, with the formula

$$\int f''(x)^2 dx \approx \hat{\beta}(h_0) \equiv \frac{1}{n^2 {h_0}^5} \sum_i \sum_j L^{IV}(\frac{X_i - X_j}{h_0}) \ ,$$

where L^{IV} is the fourth derivative of another kernel function L, and h_0 is the kernel width for L (note that L is not necessary to be equal to K). The kernel width h can then be set to

$$h = \left[\frac{\int K(t)^2 dt}{n \ \sigma_2^2 \ \hat{\beta}(h_0)} \right]^{1/5} .$$

To avoid choosing h_0 arbitrarily, one possible way is to use the same bandwidth h to estimate $\int f''(x)^2 dx$. Therefore, we solve the following equation for h

$$h = \left[\frac{\int K(t)^2 dt}{n \ \sigma_2^2 \ \hat{\beta}(h)} \right]^{1/5} .$$

However, a good bandwidth for estimating $f(x)$ may not be an appropriate bandwidth for estimating $\int f''(x)^2 dx$. The "plug-in selectors" tries to find a reasonable relationship between h_0 and h_{opt}. Sheather and Jones [24] suggested the following formulas

$$h_{SJ} = \left[\frac{\int K(t)^2 dt}{n \ \sigma_2^2 \ \hat{S}(\hat{\alpha}(h_{SJ}))} \right]^{1/5} ,$$

where

$$\begin{aligned}
\hat{\alpha}(h) &= 1.357\left[\frac{\hat{S}(a)}{\hat{T}(b)}\right]^{1/7} h^{5/7} \ , \\
\hat{S}(h) &= \frac{1}{n(n-1)h^5}\sum_i\sum_j L^{IV}(\frac{X_i - X_j}{h}) \ , \\
\hat{T}(h) &= \frac{-1}{n(n-1)h^7}\sum_i\sum_j L^{VI}(\frac{X_i - X_j}{h}) \ , \\
a &= 0.920\hat{\lambda}n^{-1/7} \ , \\
b &= 0.912\hat{\lambda}n^{-1/9} \ , \\
\hat{\lambda} &= \text{the sample interquartile range.}
\end{aligned}$$

In our experiments, we compare the performance of both the Walter's kernel width with $C = 4,000$ (denoted by h_W), as well as the Sheather and Jones' kernel width (denoted by h_{SJ}).

It should be noticed that the optimality of equation (4.3) is valid only if $f(x)$ has the second derivative. This formula fails if f has illumination discontinuities. In this case, the kernel estimator tends oversmooth the function, as shown in our experiment results in section 4.3.

4.2.2 Wavelet thresholding estimator

Applying wavelets to statistical problems has gained a large attention in the recent years. In particular, a method called wavelet thresholding estimator is proposed to solve the density estimation problem. We present a summary of this method in this section. Details about basic wavelet theory can be found in [2], and details of the wavelet thresholding estimator can be found in [7, 13].

The wavelet estimator is also an orthogonal series estimator, and the function being estimated is projected on the wavelet space

$$f(x) = \sum_k \alpha_{J,k}\varphi_{J,k}(x) + \sum_{j\geq J}\sum_k \beta_{j,k}\psi_{j,k}(x) \ ,$$

where φ is the father wavelet, ψ is the mother wavelet, and

$$\begin{aligned}\varphi_{j,k}(x) &= 2^{j/2}\varphi(2^j x - k) \ ,\\ \psi_{j,k}(x) &= 2^{j/2}\psi(2^j x - k) \ ,\\ \alpha_{j,k} &= \int f(x)\varphi_{j,k}(x)dx \ ,\\ \beta_{j,k} &= \int f(x)\psi_{j,k}(x)dx \ .\end{aligned}$$

In order to understand the method, we start from the *linear wavelet density estimator*

$$\hat{f}_{j_0}(x) = \sum_k \hat{\alpha}_{J,k}\varphi_{J,k}(x) + \sum_{j \geq J}^{j_0} \sum_k \hat{\beta}_{j,k}\psi_{j,k}(x) \ , \tag{4.4}$$

where $\hat{\alpha}_{j,k} = \frac{1}{n}\sum_{i=1}^{n} \varphi_{j,k}(X_i)$ and $\hat{\beta}_{j,k} = \frac{1}{n}\sum_{i=1}^{n} \psi_{j,k}(X_i)$.

Note that in practice we will not compute $\hat{\beta}_{j,k}$ directly. Instead we compute the estimator as

$$\hat{f}_{j_0}(x) = \sum_k \hat{\alpha}_{j_0+1,k}\varphi_{j_0+1,k}(x) \ , \tag{4.5}$$

which is equivalent to (4.4) and the detail coefficients $\hat{\beta}_{j,k}$ can be found out, if necessary, by applying the discrete wavelet transform [13].

The resolution level j_0 is the tuning parameter in this method. In [13] it is shown that the L_2 error of a wavelet estimator is bounded by

$$ISE \leq C_1 \frac{2^{j_0}}{n} + C_2 \frac{1}{2^{j_0 s}} \ , \tag{4.6}$$

where n is the sample size, $s+1$ is the regularity of the wavelet, C_1 and C_2 are two constants that depend on the function $f(x)$ under estimated and the wavelet basis, but do not depend on j_0. The optimal j_0 is given by

$$2^{j_0} \simeq n^{\frac{1}{2s+1}} \ .$$

Studies [7, 13] show that this linear estimator cannot perform well when the function has discontinuities. A better *non-linear wavelet estimator* can be used

$$\hat{f}_{j_1}(x) = \sum_k \hat{\alpha}_{j_0,k}\varphi_{j_0,k}(x) + \sum_{j \geq j_0}^{j_1} \sum_k \beta^{*}{}_{j,k}\psi_{j,k}(x) \ , \tag{4.7}$$

where

$$2^{j_0} \simeq n^{\frac{1}{2s+1}} \ ,$$
$$2^{j_1} \simeq \frac{n}{logn} \ ,$$
$$\hat{\beta^*}_{j,k}(x) = \begin{cases} \hat{\beta}_{j,k}(x) & \text{if } \left|\hat{\beta}_{j,k}(x)\right| > \lambda_{j,k}, \\ 0 & \text{otherwise} \end{cases} \ ,$$

and $\{\lambda_{j,k}\}$ is a set of properly chosen threshold values.

Even though the mathematical proof of this result is tedious, the intuitive idea behind this estimator is simple: we set j_0 in such a way that the balance between bias and variance is about right (equation (4.6)). Then we try to include some detail coefficients $\hat{\beta}_{j,k}$ (in higher level j, $j_0 \leq j \leq j_1$) that are not random noise, but represent some important information about the function. Due to the localization property of wavelets, those coefficients that are *significantly different from zero* represent some non-smoothness of the function at the corresponding location. So we include the coefficients $\{\hat{\beta}_{j,k} : \left|\hat{\beta}_{j,k}\right| \geq \lambda_{j,k}\}$. For those coefficients that are *close to zero* (indicated by $\left|\hat{\beta}_{j,k}\right| < \lambda_{j,k}$), we treat them as noise due to the stochastic nature of the samples, and discard those coefficients in order to remove the noise.

There are many methods suggested for the choice of $\{\lambda_{j,k}\}$, and in [6] the "universal threshold" is suggested

$$\lambda_{j,k} = \lambda = \hat{\sigma}\sqrt{2log(n)} \ , \tag{4.8}$$

where $\hat{\sigma}$ is an estimation of the noise magnitude of the wavelet coefficients at the finest level j_1, and is suggested to be computed by the median absolute deviation, divided by .6745

$$\hat{\sigma} = \frac{1}{0.6745} \ MEDIAN[\left|\hat{\beta}_{j_1} - MEDIAN(\hat{\beta}_{j_1})\right|] \ . \tag{4.9}$$

Therefore, the wavelet thresholding density estimation can be summarized in the following steps:

1. For each particle-hit, update the coefficients $\hat{\alpha}_{j_1+1,k}$.

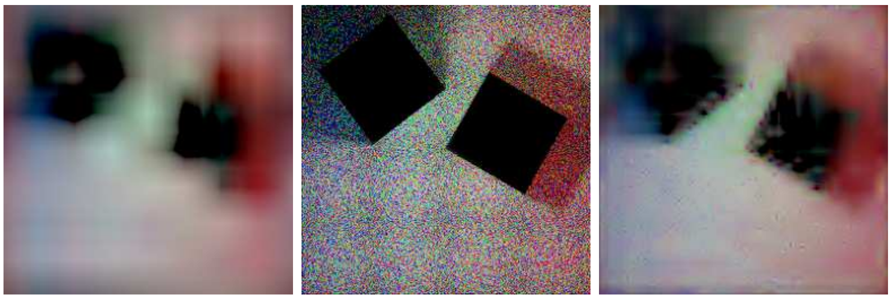


Figure 4.3: Non-linear wavelet thresholding estimator.

2. After all sample points are processed, apply discrete wavelet transform to obtain the coefficients $\hat{\alpha}_{j_0,k}$ and $\hat{\beta}_{j,k}$, $j_0 \le j \le j_1$.

3. Compute the "universal threshold" by formula (4.8) and (4.9).

4. Threshold the detail coefficients $\hat{\beta}_{j,k}$, $j_0 \le j \le j_1$.

5. Apply inverse discrete wavelet transform to obtain the coefficients $\hat{\alpha}^*_{j_1+1,k}$ (where superscript $*$ means thresholded coefficients).

6. Given any position x, compute $\hat{f}(x)$ by the formula:

$$\hat{f}_{j_1}(x) = \sum_k \hat{\alpha}^*_{j_1+1,k} \varphi_{j_1+1,k}(x)$$

Figure 4.3 shows an example of wavelet thresholding estimator. The left image of figure 4.3 includes the coefficients $\hat{\alpha}_{j_0,k}$ only (in equation (4.7)), and discards all detail coefficients $\hat{\beta}_{j,k}$, $j_0 \le j \le j_1$. The middle image includes all detail coefficients. The right image includes those coefficients that pass the threshold test. This figure illustrates the idea why the wavelet thresholding estimator works.

In our experiments, we use Daubechies-6 [2] as the wavelet basis function.

4.3 Results

We compared the three density estimators in four different aspects: visual quality, L_2 error, running time, and memory requirement. Results are shown in figure 4.4

and 4.5.

4.3.1 Visual quality

It can be seen that the kernel estimator produces images with the best visual appearance. The pictures contain less noise than the other two estimators. As mentioned in section 4.2.1, the kernel estimator tends to oversmooth the function when the function has discontinuities. From the images of floor we see that the sharp shadow boundaries are blurred. This situation is even more obvious in Walter's heuristic kernel width method, since their kernel width is usually larger than the h_{SJ}, results in a further oversmoothed image. In the wavelet estimator, even though most of the noise is suppressed by thresholding, the images still contain small spikes. However, the method tries to preserve the sharp shadow boundaries on the floor. In our new estimator, as the variance is suppressed by formula (3.3), high frequency noise does not appear in the final images. Bias is reduced by surface subdivision, but the mismatch of estimated functions along surface subdivision boundaries produces another kinds of visual artifacts as mentioned in section 3.4.2 and [36].

4.3.2 L_2 error

Since the kernel estimator oversmooth the function, they produce the largest L_2 error compare to the other two estimators. The other two estimators achieve a better L_2 error than the kernel estimator, even though they produce some visual artifacts. This situation is more obvious on the floor than on the wall, because the sharp shadow boundaries on the floor are strongly biased by the kernel estimator. It is interesting to notice that the best L_2 error does not mean the best visual image, because the human visual system is more sensitive to noise than bias.

4.3.3 Running time

Even though the kernel estimator produces the best visual quality, the long running time of this method makes it less attractive. From equation (4.1) we see that for each position x, the kernel estimator needs to evaluate the kernel function on *every* samples. Moreover, when the sample size is large, the automatic bandwidth selection takes several hours to compute h_{SJ}. On the other hands, the two orthogonal series estimators need only to evaluate the coefficients ($\hat{a}_i, \hat{\alpha}_{j_0,k}$ *and* $\hat{\beta}_{j,k}$) once. After that, for each position x, the estimator $\hat{f}(x)$ (equation (2.1) and (4.7)) is independent of the sample size n. This makes the two estimators run much faster than the kernel estimator.

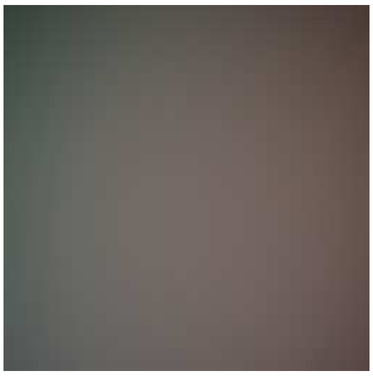
4.3.4 Memory and storage requirement

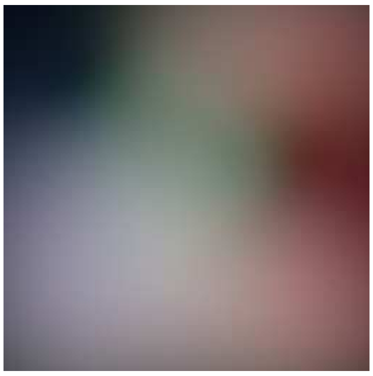
In order to evaluate equation (4.1), the kernel method needs to read and store the n sample points in main memory. On the contrary, the two orthogonal series estimators need to store the coefficients only. The sample points are read from the disk, the coefficients are updated, and then the sample points can be discarded. There is no need to store the sample points in main memory. Moreover, it is worth mentioning that in these two methods the sample points are even not necessary to be stored in secondary storage. The particle tracing stage and the coefficients updating stage can be done at the same time. This is an obvious advantage over the kernel estimator.

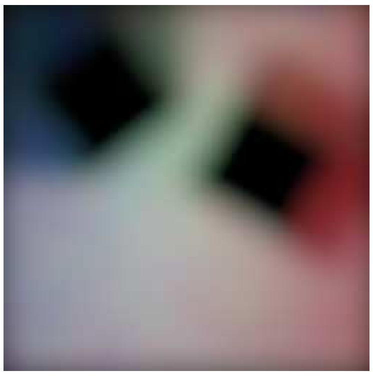
4.4 Which one is the best?

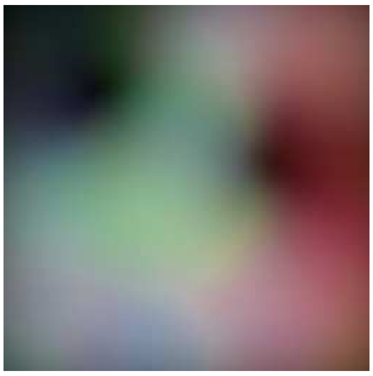
The results show that wavelet thresholding estimator can adapt to local illumination discontinuities, and achieve a better L_2 error than kernel estimator. This observation is consistent with the theoretical results given in the statistics literature. However, even though the method is suitable for data analysis, it may not be suitable for visual applications such as the global illumination problem, as annoying random noise exists in the final images. The kernel estimator usually gives an oversmoothed result, and sharp shadow boundaries are not preserved. However, the visual quality of this estimator is the best because less noise appears in the final images. When the sample size is large, however, the long running time and the large memory and storage requirement of kernel estimator makes it less attractive.

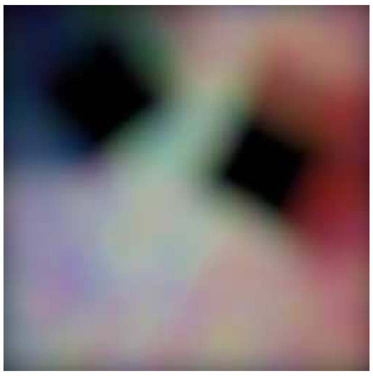
When the sample size is small, our new estimator may not give a good result because surfaces have not been subdivided yet. When the sample size is large, however, the results produced by our method are comparable to the other estimators. We conclude that if the running time and resources (main memory and secondary storage) is not the major concern, the kernel estimator gives a robust result with better image quality. If computational resources are limited, the new method we proposed is a more practical method.

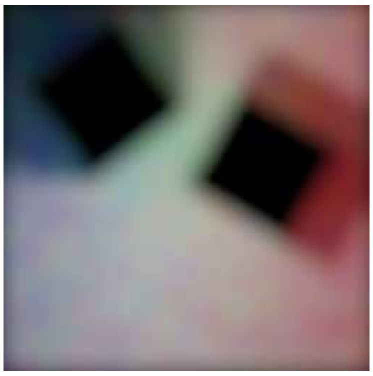

Walter kernel h_W=0.91
ISE=0.666 time=166s

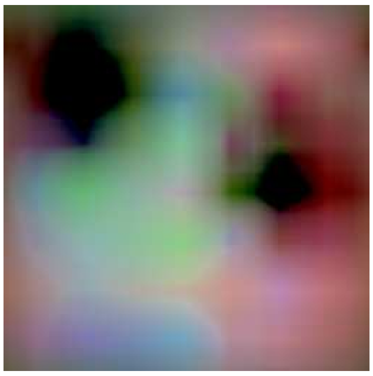

Walter kernel h_W=0.29
ISE=0.215 time=578s

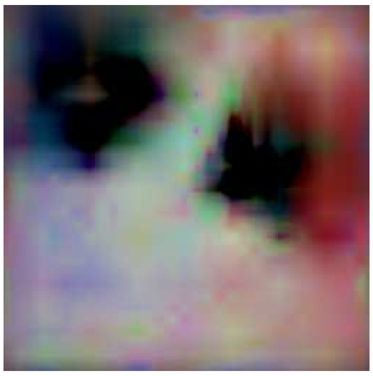

Walter kernel h_W=0.093
ISE=0.058 time=1400s

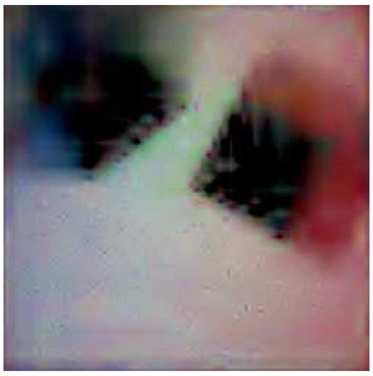

SJ kernel h_{SJ}=0.21
ISE=0.148 time=65s

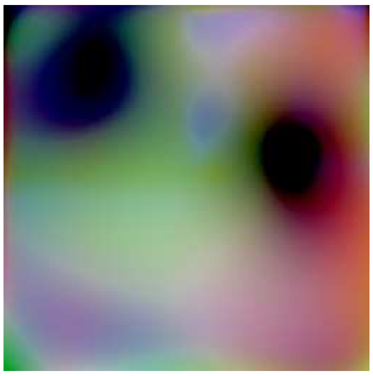

SJ kernel h_{SJ}=0.099
ISE=0.064 time=114s

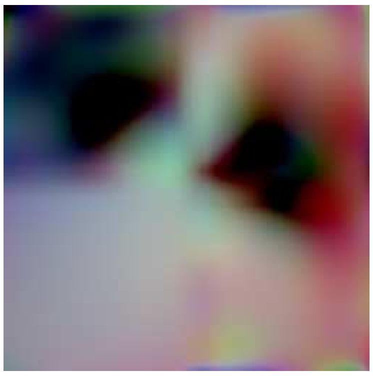

SJ kernel h_{SJ}=0.062
ISE=0.040 time=1430s


Wavelet thresholding
ISE=0.146 time=12s


Wavelet thresholding
ISE=0.064 time=13s


Wavelet thresholding
ISE=0.034 time=23s


New estimator
ISE=0.156 time=8s


New estimator
ISE=0.058 time=14s

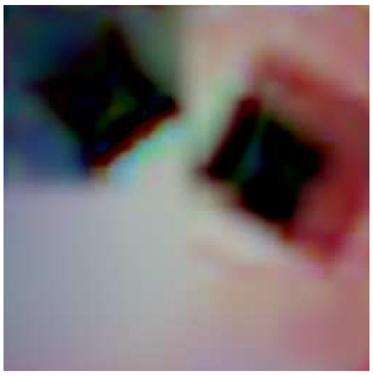
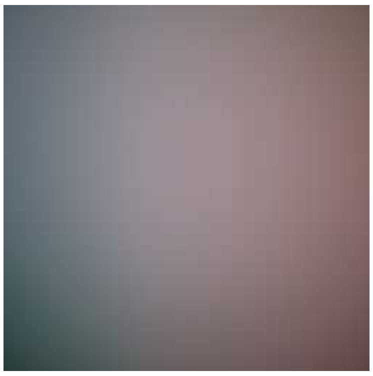
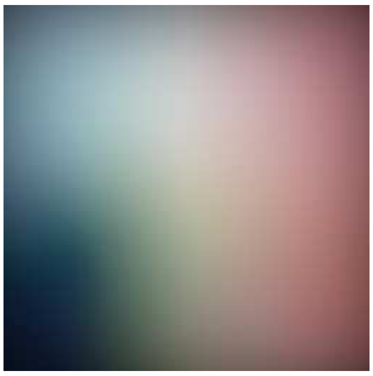
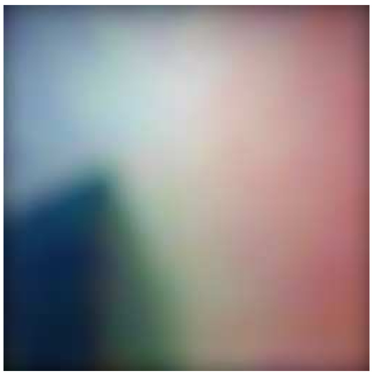
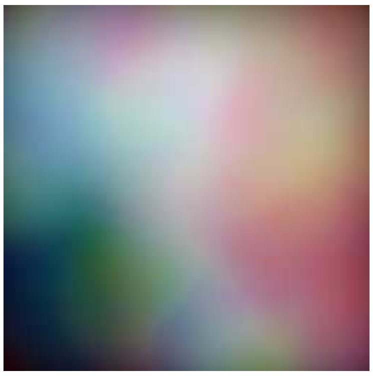

New estimator
ISE=0.034 time=45s

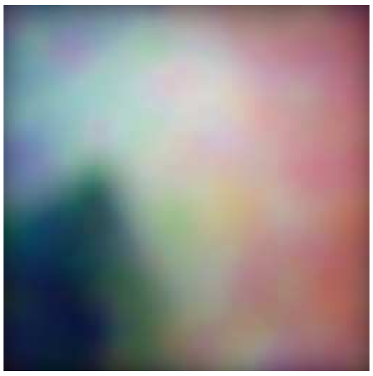
Figure 4.4: Estimated illumination of the floor. Sample size: 1548 (left column), 14846 (middle column) and 148525 (right column).

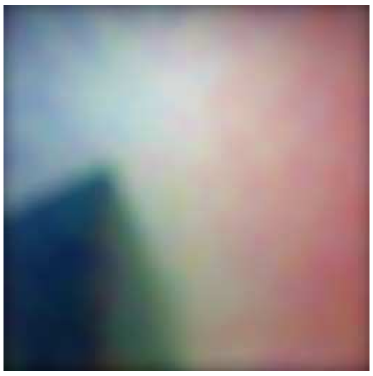

Walter kernel h_W=0.746
ISE=0.313 time=299s

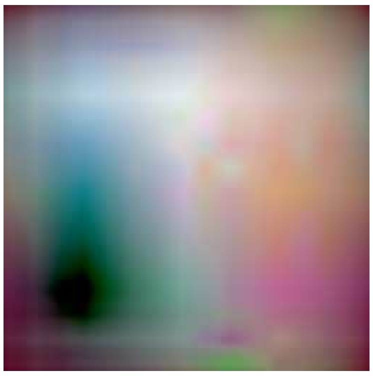

Walter kernel h_W=0.238
ISE=0.031 time=807s

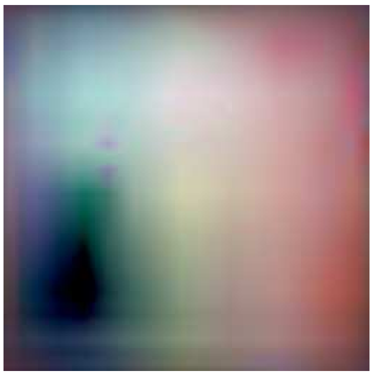

Walter kernel h_W=0.075
ISE=0.013 time=2331s

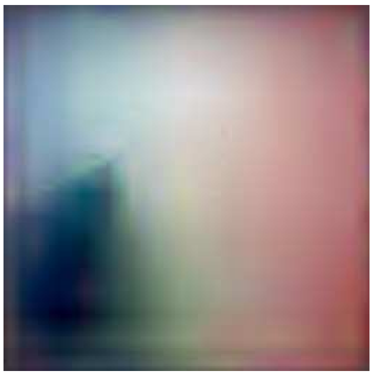

SJ kernel h_{SJ}=0.176
ISE=0.027 time=58s

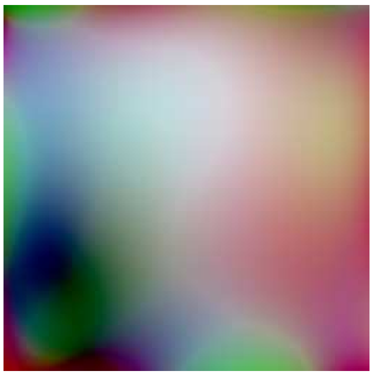

SJ kernel h_{SJ}=0.089
ISE=0.016 time=223s

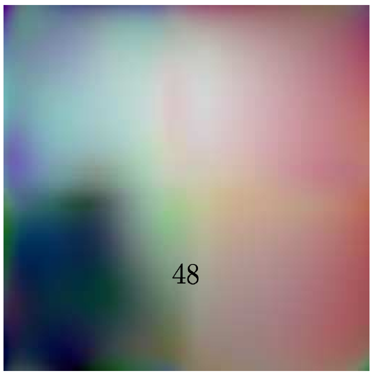

SJ kernel h_{SJ}=0.056
ISE=0.012 time=2456s


Wavelet thresholding
ISE=0.042 time=12s


Wavelet thresholding
ISE=0.025 time=14s


Wavelet thresholding
ISE=0.015 time=26s


New estimator
ISE=0.037 time=5s


New estimator
ISE=0.014 time=14s

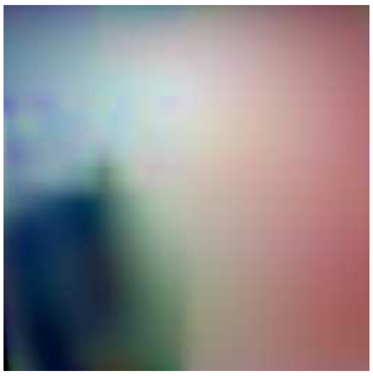

New estimator
ISE=0.011 time=48s

Figure 4.5: Estimated illumination of the facing wall. Sample size: 2290 (left column), 22523 (middle column) and 226175 (right column).

Chapter 5

Parallel implementation issues

Based on today's technology, a single-processor computer does not have the capability to achieve realistic rendering at interactive rate. Even though the processor speed will become faster in the future, the complexity of simulated scenes demand for highly visual quality will increase as well. On the other hand, even more and more powerful multi-processor machines or multi-computer clusters are commercially available. How to make use of this parallel processing power is another major challenge when designing new graphics algorithms. In this chapter we investigate a parallel implementation of the new density estimator we described in Chapter 3.

5.1 Reviews of related works

For solving the global illumination problem, many research works have been done on parallel ray tracing [15, 37]. Most of these methods make use of the ray-coherence properties of the primary rays or shadow rays to design a load balancing method. However, these techniques are not suitable for our method because the nature of ray tracing is different from the density estimation framework. In general there is no coherence between rays in the density estimation framework. Even in the case when quasi-random number is used to govern the direction of the primary rays, those rays diverge in all directions, therefore it reduces the potential of caching the so called "candidate hit-objects".

Different parallel radiosity algorithms are also proposed in [30, 23, 9, 22]. These algorithms are based on hierarchical radiosity, which is also different from the density estimation framework. In hierarchical radiosity, the computation is driven by a pair of sending surface and receiving surface. With the sending surface and receiving surface being known, different static and dynamic load balancing strategies can be applied. However, these kinds of load balancing strategies cannot be directly applied to the density estimation framework.

The first implementation of a parallel random walk Monte Carlo radiosity algorithm is given in [10]. However, no load balancing solution is proposed in that paper. Zareski et al [38] gives a parallel implementation of the original density estimation framework. However, as mentioned in Chapter 2, the original density estimation framework used secondary storage to store the particle hit points, and the density estimation phase is performed after the ray shooting phase. As a consequence, the algorithm in [38] is designed on a shared memory parallel platform and achieve a satisfactory parallel efficiency. Our new estimator does not store particle hit points in secondary storage, therefore the method in [38] cannot be directly applied to our method.

5.2 A parallel implementation of the new density estimator

5.2.1 The parallel model

There are several parallel platforms currently existing, such as SMP, MPP, or cluster of workstations. As a cluster of workstations is the most scalable and affordable platform, our algorithm is designed for this platform. There is no shared memory among the processors, and communication between processors are based on message-passing.

We further assume that a scene is complex enough so that it is infeasible to duplicate the whole scene on each processor. In this case, the scene should be subdivided

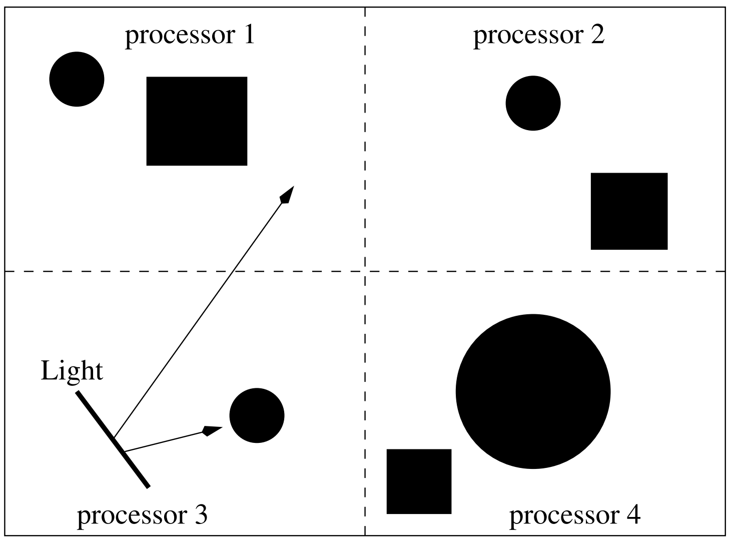


Figure 5.1: Ray shooting in a distributed scene.

and distributed among the processors. Each processor will hold a partial scene only, as shown in figure 5.1. A ray originating from one processor may hit some local objects in this processor, and the radiosity value of the hit object can be updated. A ray may also hit nothing in its local scene and enter another region held in another processor. In this case, the ray-shooting task should be transferred to the neighbor processor and continue there. As the number of light sources and scene geometry in each partial scene is different, load imbalance becomes the major problem in this approach.

5.2.2 Combining data-driven and demand-driven

In [21], a parallel ray-tracing algorithm was described, which motivated our research work reported in this chapter. Their method combines the "data-driven" tasks and "demand-driven" tasks to improve the load balancing among processors. In the "data-driven" approach, the scene is distributed among processors and each processor holds a partial scene. Each processor is assigned a block of the final image, and ray tracing is performed on individual processors. If a ray enter into a neighbor region without hitting any local objects, the ray is transferred to the neighbor processor. However, load imbalance may happen in this approach. To overcome this problem, their method makes use of coherence between primary rays and shadow rays. Since primary rays have the same origin and almost the same direction, they are likely to hit the same objects. It is the same cases in the shadow rays. Therefore, if the possible hit objects are sent and cached on some other processors, the primary rays or shadow rays can be traced by that processor. The tracing of primary rays and tracing of shadow rays are therefore converted as "demand-driven" tasks, and passed to other idle processors for the computation. A master processor is responsible for scheduling these "demand-driven" tasks. Their paper shows that by combining the "data-driven" and "demand-driven" approaches, the load balancing among processors can be improved significantly.

Unlike the method given in [21], there is no coherence between rays that we can exploit in the density estimation framework. Therefore, the demand-driven tasks must be defined from another aspect. Our method makes use of the special computation pattern of an orthogonal series estimator to improve the load balancing. In our new estimator, M higher order polynomial functions are needed to be evaluated whenever a ray hits a surface. We found that these steps are computation intensive, and may contribute up to 20% of the total execution time. Since only little of scene information is needed to evaluate these polynomials, these tasks are suitable to be migrated to another processor. We treat these polynomial update tasks as the "demand-driven" tasks. In such a way, imbalance workload among processors can be improved. The next two sections describe our algorithm in details.

5.2.3 Data-driven in density estimation framework

Our algorithm proceeds as followed: when the program starts, a master process reads the scene and partitions the scene into smaller sub-scenes, based on the binary space partitioning (BSP) method. At the same time a global BSP tree is built. A leave node of the global BSP tree holds the ID of a partial scene and also the ID of

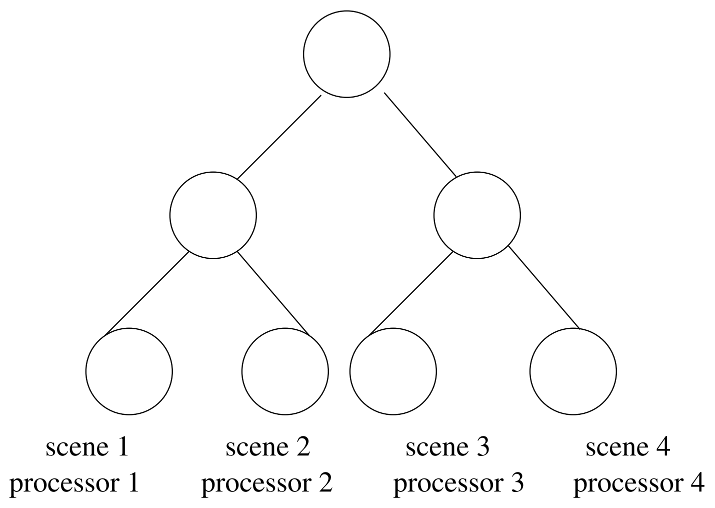


Figure 5.2: A global BSP tree.

processor which will process this partial scene (figure 5.2). When it is finished, the global BSP tree is broadcast to all slave processors, and each slave processor reads the partial scene that it is assigned to.

Afterward, each processor starts the ray shooting process under the density estimation framework. The number of rays it should shoot depends on the emittance of the light sources in its local sub-scene. Each time when there is no request from the master, a random ray is generated from the light sources in the sub-scene and the ray is traced. If the ray hits a local object, the object's radiosity value is updated. Otherwise, if the ray enter a neighbor region, the global BSP tree is traversed to find out which processor is holding the neighbor region. A "ray-shoot" request is then passed to the master, and the master will pass the request to the neighbor processor later.

For each slave processor, a "ray-shoot" request may be received from the master processor. This means that a ray originated from another processor has entered the sub-scene region held by this processor. This ray is then traced on this processor.

If no more rays need to be traced, the processor sends a "finish" signal to the master and wait. The master will either send a "finish" signal back when all processors are done, or send some “ray-shoot” requests to this processor. In the latter case, the slave process needs to process the incoming requests and wait again until the former happens. Figure 5.3 shows the pseudocode of a slave process.

We adopt the master-slave approach in our algorithm and require all slaves to send their requests to the master. This will introduce more communication overhead. However, we still adopt this master-slave approach because determining whether the whole rendering process has been finished is not a trivial task, and using a master-slave approach will simplify the situation. When the master has received the “finish” signal from all the slaves, it can notify the slaves that the whole process has been done. However, if a slave is allowed to send a “ray-shoot” request directly to the target processor after the target processor has signal the master that it has been finished, then the master will have no way to know whether the ray shooting process has stopped or not.

In order to reduce the communication overhead, several short messages are queued in a buffer before being sent. When the buffer is full (in our implementation the buffer size is 20), these messages are then sent as a longer message. It is because sending a long message at one time has less communication overhead than sending many small messages one-by-one. Figure 5.4 shows a logical diagram of a master process.

5.2.4 Demand-driven in density estimation framework

Load imbalance is usually a major problem in the “data-driven” approach mentioned above. As proposed in [21], we can add “demand-driven” tasks to balance the workload among processors. Unlike the method given in [21], there is no coherence between rays that we can exploit in the density estimation framework. Therefore, the demand-driven tasks must be chosen from another aspect. In our approach, the high-order polynomial update is treated as the demand-driven task.

In our new density estimation method, apart from the ray-object intersection tests during ray shooting, most computational time is spent on evaluating the high order polynomials. Whenever a ray hits an object at X_n, the M polynomial coefficients

```
SlaveMainLoop() {
    do {
        Test and get request from master;
        if (has request)
            if (request == "ray-shoot")
                setup the ray;
                ShootRay(ray);
        else if (need to generate local ray)
                ray := generate a local ray;
                ShootRay(ray);
    } until (no need to generate local ray);
    send "finish" signal to master;
    while (not done) {
        Test and get request from master;
        if (has request)
            if (request == "ray-shoot")
                setup the ray;
                ShootRay(ray);
            else if (request == "finish")
                done := true;
    }
}

ShootRay( ray ) {
    while (ray not absorbed) {
        obj := closest object hit by the ray;
        if (obj exists)
            update the object's coefficients;
            compute the reflected ray;
        else if (ray enter neighbor region)
            send "ray-shoot" request to master;
    }
}
```

Figure 5.3: A slave process.

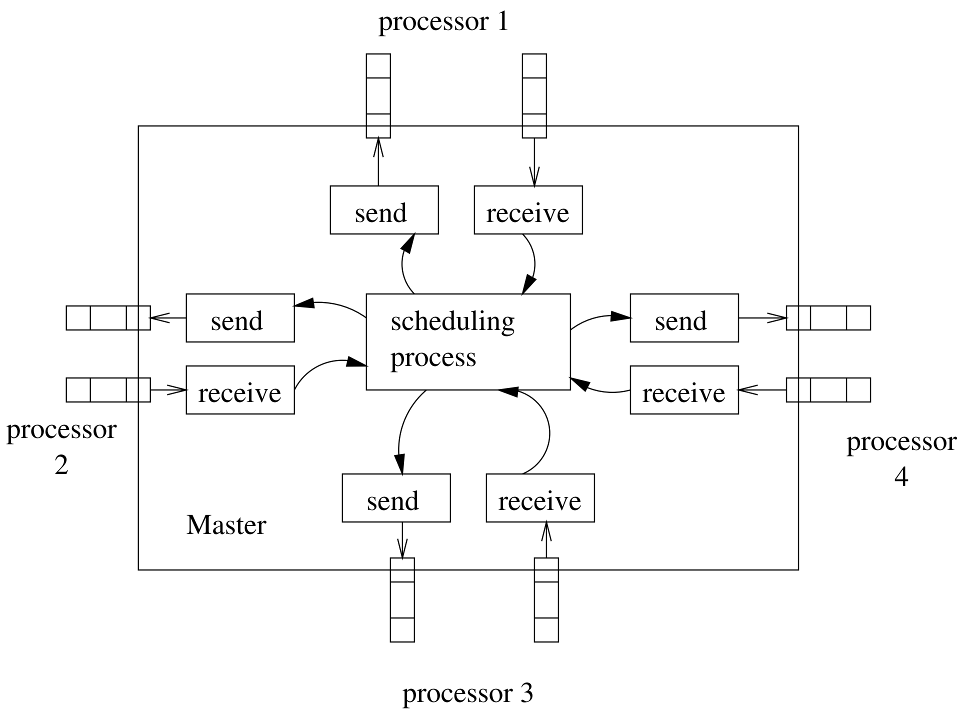


Figure 5.4: A master process.

stored in the object are needed to be updated

$$a_i = a_i + \phi_i(X_n) \qquad i \in [0..M] \ .$$

We found that the total time spent on evaluating these polynomials $\phi_i(X)$ is substantial. Figure 5.6 shows a diagram of computational pattern of a direct parallel implementation of our new density estimation method. This figure shows the existing of load imbalance. It also shows that the polynomial update tasks contribute a large portion of the total running time.

However, only little information is required to evaluate the polynomials. Basically, only the parametric coordinates (u, v) of the hit point on the object is needed. Therefore, these evaluating tasks will be a good choice for being migrated to other idle processors. In our algorithm, whenever a processor sends a "finish" signal to the master, the master will notify the other working processors that there is at least one idle processor. Afterward, for every particle hit, the polynomial updates in all the other slaves are postponed and a "poly-eval" request is passed to the master as a demand-driven task. The slaves can continue tracing the ray or generate other rays without the need to wait for the result of the polynomial updates.

After receiving a "poly-eval" request, the master processor determines which processor is available to evaluate the polynomials. It can, for example, determine from the length of the out-going buffer queue which processor has fewer tasks to send. This "poly-eval" request is then sent to that processor. After the polynomials are evaluated, the resulting coefficients are then passed back to the processor issuing this request. The issuing processor will update the hit object's polynomial coefficients when it receives the results. Figure 5.5 shows the pseudocode of a slave process when "demand-driven" tasks are added, and figure 5.7 shows the computational pattern of the new algorithm.

```
SlaveMainLoop() {
    do {
        Test and get request from master;
        if (has request)
            if (request == "ray-shoot")
                setup the ray;
                ShootRay(ray);
            else if (request == "someone-idle")
                deferPolyEval := true;
            else if (request == "poly-eval-finish")
                update the object's coefficients;
        else if (need to generate local ray)
            ray := generate a local ray;
            ShootRay(ray);
    } until (no need to generate local ray);
    send "finish" signal to master;
    while (not done) {
        Test and get request from master;
        if (has request)
            if (request == "ray-shoot")
                setup the ray;
                ShootRay(ray);
            else if (request == "poly-eval")
                evaluate the polynomials;
                send back the results with "poly-eval-finish" signal;
            else if (request == "poly-eval-finish")
                update the object's coefficients;
            else if (request == "finish")
                done := true;
    }
}

ShootRay( ray ) {
    while (ray not absorbed) {
        obj := closest object hit;
        if (obj exists)
            if (deferPolyEval) send "poly-eval" request to master;
            else               update the object's coefficients;
            compute the reflected ray;
        else if (ray enter neighbor region)
            send "ray-shoot" request to master;
    }

}
```

Figure 5.5: A slave process with demand-driven tasks.

5.3 Results and discussions

We have implemented our algorithm on a cluster of PC with each node is based on a Pentium III processor. Our program is implemented using C programming language and the Message-Passing-Interface (MPI) library. Note that non-blocking message passing library calls, such as MPI_Isend() and MPI_Irecv(), are used in the implementation.

The scene we used is shown in figure 5.8. It contains 36 walls, 15 light sources and 66 chairs. Each chair contains 92 polygons and 4 spheres. In total there are 6123 polygons and 264 spheres. The scene is divided into 4 sub-scenes automatically by the algorithm based on the binary-space-partitioning method.

The total number of primary rays shot from the light sources is 100,000. The execution time is shown in table 5.1. From table 5.1 we found that roughly 20% of the execution time is used to evaluate the polynomial basis functions. When the demand-driven algorithm is used, the algorithm runs about 13% faster than that one without demand-driven tasks. Figure 5.7 shows the running time of the processors after demand-driven tasks are added. From this figure, we found that most of the polynomial update tasks are transferred from the busy processors to some idle processors.

When no demand-driven tasks are added, the speedup factor is 0.78 for 4 slaves. When there are demand-driven tasks, the speedup factor increased to 0.9. This improvement is not very impressive, as the imbalance in ray shooting processes dominates the running time. However, it should be noticed that our method is straight forward and can be easily combined with other existing load balancing techniques. Therefore, our strategy can be viewed as a complementary method to the other existing load balancing techniques.

	Processor	Rays shot from light sources	Time for ray shooting	Time for polynomial update	Total time
sequential	1	100,000	37.14 (62%)	13.4 (22%)	60.32
no demand-driven	1	25,000	10.81	3.29	17.75
	2	15,000	4.52	2.02	7.72
	3	30,000	10.14	4.08	17.87
	4	30,000	11.32	4.12	19.13
	program finish				19.25
	parallel efficiency				78%
with demand-driven	1	25,000	10.87	1.86	16.38
	2	15,000	4.35	6.85	12.38
	3	30,000	10.06	2.64	16.35
	4	30,000	11.06	1.99	16.74
	program finish				16.85
	parallel efficiency				90%

Table 5.1: CPU time (in sec) used.

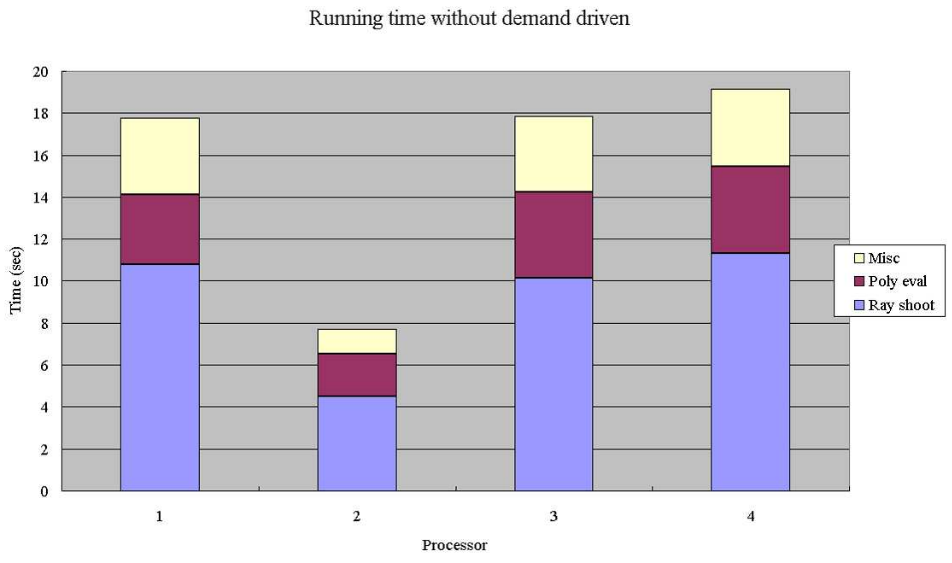


Figure 5.6: Computational pattern of a parallel orthogonal series estimation.

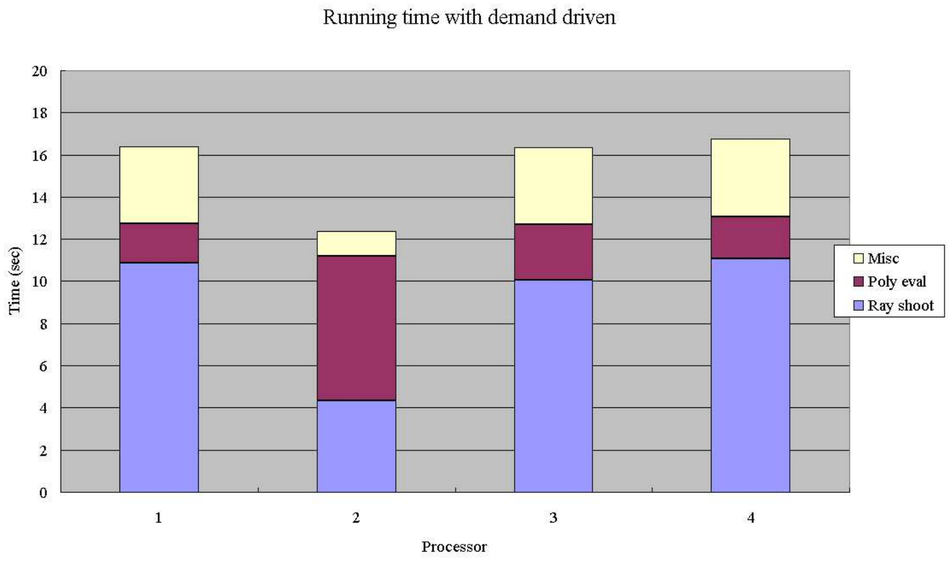
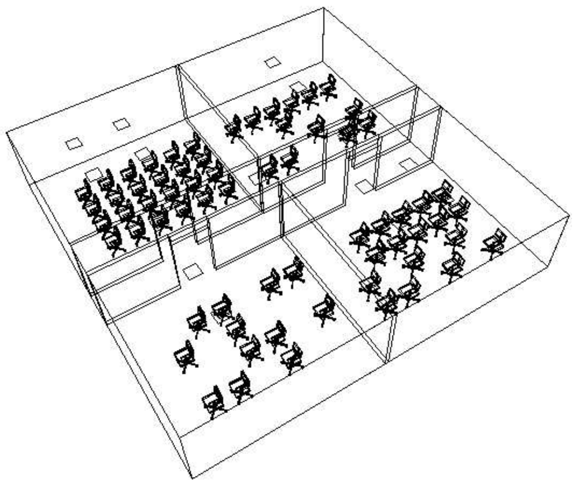


Figure 5.7: Computational pattern of a parallel orthogonal series estimation with "demand-driven" tasks added.



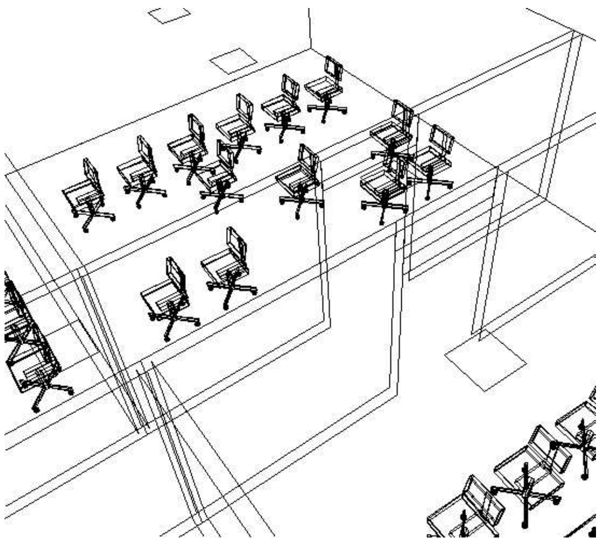


Figure 5.8: Wireframe of the scene used. Top: full view. Bottom: zoom view.

Chapter 6

Conclusions

A new method is proposed for estimating the illumination of a surface in the density estimation framework. The method adds two modifications to the standard orthogonal series estimator. First, the method determines adaptively and automatically for each surface the appropriate number of terms that should be used in the series. Second, when an illumination discontinuity exists on a surface, this method subdivides the surface to improve accuracy. As a result, to achieve the same error level, the new method uses less memory, requires fewer particles, and runs faster.

We also compared our new method with the kernel method and the wavelet thresholding method. We showed that each estimators has its own shortcomings in some aspects, but our method is more practical if computational resources are limited. We also proposed a load balancing strategy for the parallel implementation of our new estimator. The strategy is based on the special computational pattern of the new estimator, and it is simple and straight forward so that it can be easily combined with other existing load balancing techniques.

There is still lot of work to be done with this method. One major problem of the density estimation framework is that tiny surfaces may not get enough particle-hits even though a large number of particles have been traced. The chairs in figure 3.18 shows the problem. One way to solve this problem is to trace more particles, as shown in figure 6.1, but this method increases the computational cost as well. A more sophisticated method has to be developed to solve this problem. Moreover, a better method has to be developed to rectify the inconsistent shading value across subdivision boundaries. Lastly, there are many kinds of orthogonal basis functions. Different sets of basis functions have different properties. It is worthwhile to study other orthogonal series to see which is the most suitable one for the density estimation framework.

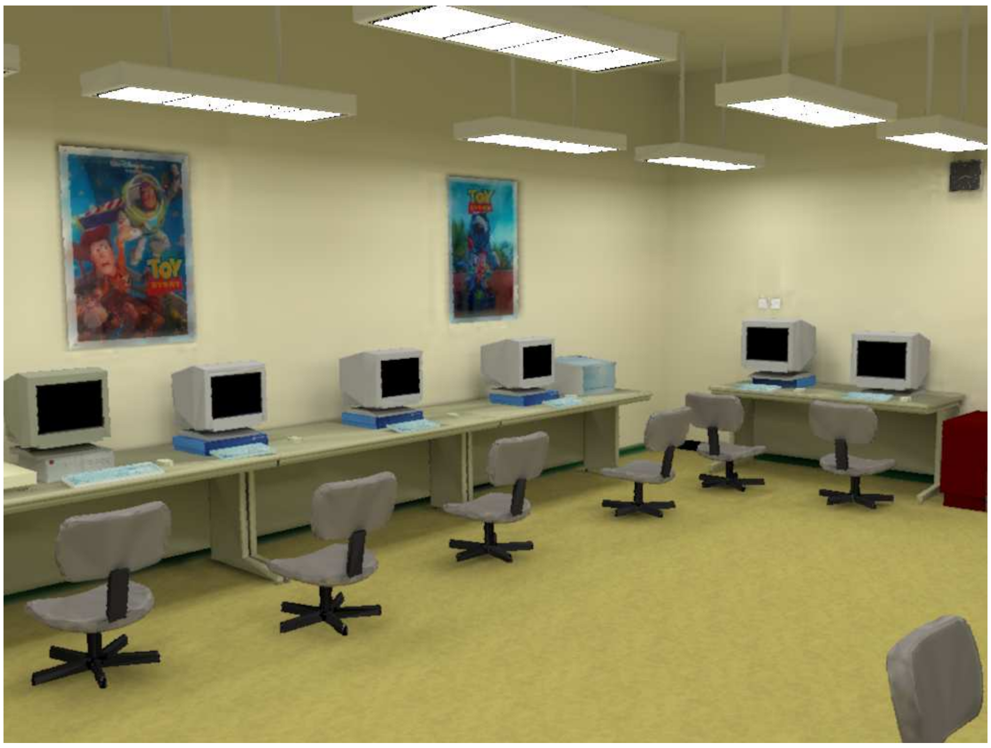


Figure 6.1: The same scene as shown in figure 3.18, when five times the number of particles have been traced.

Contents

List of Figures

Bibliography

[1] D. R. Baum, S. Mann, K. P. Smith, and J. M. Winget. Making radiosity usable: Auomatic preprocessing and meshing techniques for the generation of accurate radiosity solutions. In *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, pages 51–60, July 1991.

[2] C. S. Burrus, R. A. Gopinath, and H. Guo. *Introduction to Wavelets and Wavelet Transforms - A Primer.* Prentice-Hall, 1998.

[3] Shenchang Eric Chen, Holly E. Rushmeier, Gavin Miller, and Douglass Turner. A progressive multi-pass method for global illumination. In *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, pages 165–174, July 1991.

[4] S. T. Chiu. A comparative review of bandwidth selection for kernel density estimation. *Statistica Sinica*, 6:129–145, 1996.

[5] P. J. Diggle and P. Hall. The selection of terms in an orthogonal series density estimator. *Journal of American Statistical Association*, 81:230–233, 1986.

[6] D. L. Donoho and I. M. Johnstone. Ideal spatial adaptation by wavelet shrinkage. *Biometrika*, 81(3):425–455, 1994.

[7] D. L. Donoho, I. M. Johnstone, G. Kerkyacharian, and D. Picard. Density estimation by wavelet thresholding. *The Annals of Statistics*, 24(2):508–539, 1996.

[8] Martin Feda. A Monte Carlo approach for Galerkin radiosity. *The Visual Computer*, 12(8):390–405, 1996.

[9] Chen-Chin Feng and Shi-Nine Yang. A parallel hierarchical radiosity algorithm for complex scenes. In *IEEE Parallel Rendering Symposium*, pages 71–78, October 1997.

[10] P. Guitton, J. Roman, and G. Subrenat. Implementation results and analysis of a parallel progressive radiosity. In *IEEE Parallel Rendering Symposium*, pages 31–38, October 1995.

[11] P. Hall. Comparison of two orthogonal series methods of estimating a density and its derivatives on an interval. *Journal of Multivariate Analysis*, 12:432–449, 1982.

[12] Pat Hanrahan, David Salzman, and Larry Aupperle. A rapid hierarchical radiosity algorithm. In *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, pages 197–206, July 1991.

[13] W. Härdle, G. Kerkyacharian, D. Pickard, and A. Tsybakov. *Wavelets, Approximation, and Statistical Applications.* Lecture Notes in Statistics 129. Springer-Verlag, 1998.

[14] Paul S. Heckbert. Adaptive radiosity textures for bidirectional ray tracing. In *Computer Graphics (SIGGRAPH '90 Proceedings)*, volume 24, pages 145–154, August 1990.

[15] A. Heirich and J. Arvo. Scalable monte carlo image synthesis. In *Parallel Computing*, volume 3, pages 845–860, 1997.

[16] David S. Immel, Michael F. Cohen, and Donald P. Greenberg. A radiosity method for non-diffuse environments. In *Computer Graphics (ACM SIGGRAPH '86 Proceedings)*, pages 133–142, August 1986.

[17] H. W. Jensen. *Realistic Image Synthesis Using Photon Mapping.* A. K. Peters, Ltd, 2001.

[18] J. T. Kajiya. The rendering equation. In *Computer Graphics (ACM SIGGRAPH '86 Proceedings)*, pages 143–150, August 1986.

[19] Tom Koornwinder. Two-variable analogues of the classical orthogonal polynomials. In *Theory and Application of Special Functions.* Academic Press, 1975.

[20] R. A. Kronmal and M. E. Tarter. The estimation of probability densities and cumulatives by fourier series methods. *Journal of American Statistical Association*, 63:925–952, 1968.

[21] E. Reinhard and F. W. Jansen. Rendering large scenes using parallel ray tracing. In *First Eurographics Workshop of Parallel Graphics and Visualization*, pages 67–80, September 1996.

[22] L. Renambot, B. Arnaldi, T. Priol, and X. Pueyo. Towards efficient parallel radiosity for dsm-based parallel computers using virtual interfaces. In *IEEE Parallel Rendering Symposium*, pages 79–86, October 1997.

[23] J. Richard and J. P. Singh. Parallel hierarchical computation of specular radiosity. In *IEEE Parallel Rendering Symposium*, pages 59–70, October 1997.

[24] S. J. Sheather and M. C. Jones. A reliable data-based bandwidth selection method for kernel density estimation. *J. Roy. Statist. Soc. Ser. B*, 53:683–690, 1991.

[25] Peter Shirley, Bretton Wade, Philip Hubbard, David Zareski, Bruce Walter, and Donald P. Greenberg. Global illumination via density estimation. In *Eurographics Rendering Workshop 1995*, June 1995.

[26] F. X. Sillion and C. Puech. *Radiosity and Global Illumination.* Morgan Kaufmann, 1994.

[27] B. W. Silverman. *Density Estimation for Statistics and Data Analysis.* Chapman & Hall, 1986.

[28] Brian Smits, James Arvo, and Donald Greenberg. A clustering algorithm for radiosity in complex environments. In *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 435–442, July 1994.

[29] M. Stamminger, H. Schirmacher, P. Slusallek, and Hans-Peter Seidel. Getting rid of links in hierarchical radiosity. *Computer Graphics Forum*, 17(3):165–174, 1998.

[30] W. Sturzlinger, G. Schaufler, and J. Volkert. Load balancing for a parallel radiosity algorithm. In *IEEE Parallel Rendering Symposium*, pages 39–46, October 1995.

[31] P. K. Suetin. *Orthogonal Polynomials in Two Variables.* Gordon & Breach, 1999.

[32] Robert F. Tobler, Alexander Wilkie, Martin Feda, and Werner Purgathofer. A hierarchical subdivision algorithm for stochastic radiosity methods. In *Eurographics Rendering Workshop 1997*, pages 193–204, June 1997.

[33] Eric Veach. *Robust Monte Carlo Methods for Light Transport Simulation.* Ph.D. dissertation, Stanford University, December 1997.

[34] John R. Wallace, Michael F. Cohen, and Donald P. Greenberg. A two-pass solution to the rendering equation: a synthesis of ray tracing and radiosity methods. In *Computer Graphics (ACM SIGGRAPH '87 Proceedings)*, pages 311–320, 1987.

[35] Bruce Walter, Philip M. Hubbard, Peter Shirley, and Donald F. Greenberg. Global illumination using local linear density estimation. *ACM Transactions on Graphics*, 16(3):217–259, July 1997.

[36] K. W. Wong. A new adaptive density estimator for particle-tracing radiosity. In *Pacific Graphics 2000*, pages 62–70, October 2000.

[37] H-J. Yoon, S. Eun, and J. W. Cho. An image parallel ray tracing using static load balancing and data prefetching. In *First Eurographics Workshop of Parallel Graphics and Visualization*, pages 53–66, September 1996.

[38] D. Zareski, B. Wade, P. Hubbard, and P. Shirley. Efficient parallel global illumination using density estimation. In *IEEE Parallel Rendering Symposium*, pages 47–54, October 1995.