PRACTICAL PHOTON MAPPING IN HARDWARE

Joshua Eli Steinhurst

A dissertation submitted to the faculty of the University of North Carolina at Chapel Hill in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science.

Chapel Hill
2007

Approved by:

Dr. Anselmo Lastra

Dr. Gary Bishop

Dr. Leonard McMillian

Dr. Steven Molnar

Dr. Jan Prins

ABSTRACT

JOSHUA ELI STEINHURST: Practical Photon Mapping in Hardware
(Under the direction of Dr. Anselmo Lastra)

Photon mapping is a popular global illumination algorithm that can reproduce a wide range of visual effects including indirect illumination, color bleeding and caustics on complex diffuse, glossy, and specular surfaces modeled using arbitrary geometric primitives. However, the large amount of computation and tremendous amount of memory bandwidth, terabytes per second, required makes photon mapping prohibitively expensive for interactive applications.

In this dissertation I present three techniques that work together to reduce the bandwidth requirements of photon mapping by over an order of magnitude. These are combined in a hardware architecture that can provide interactive performance on moderately-sized indirectly-illuminated scenes using a pre-computed photon map.

1. The computations of the naive photon map algorithm are efficiently reordered, generating exactly the same image, but with an order of magnitude less bandwidth due to an easily cacheable sequence of memory accesses.

2. The irradiance caching algorithm is modified to allow fine-grain parallel execution by removing the sequential dependency between pixels. The bandwidth requirements of scenes with diffuse surfaces and low geometric complexity is reduced by an additional 40% or more.

3. Generating final gather rays in proportion to both the incident radiance and the reflectance functions requires fewer final gather rays for images of the same quality.

Combined Importance Sampling is simple to implement, cheap to compute, compatible with query reordering, and can reduce bandwidth requirements by an order of magnitude.

Functional simulation of a practical and scalable hardware architecture based on these three techniques shows that an implementation that would fit within a host workstation will achieve interactive rates. This architecture is therefore a candidate for the next generation of graphics hardware.

To Barbie

ACKNOWLEDGMENTS

This dissertation, and the research that it documents, could never have been accomplished without the selfless assistance of others. Intellectually, I am indebted to all my instructors, classmates and colleagues who have together prepared me for this task. Since arriving in Chapel Hill, Dr. Anselmo Lastra has been my advisor, guiding me with patience and understanding through the process of finding and pursuing my topic to completion. I have learned a lot from him about the conduct and presentation of research. Additionally, I have received more grammar lessons "then" shall be mentioned here.

My committee, Dr. Gary Bishop, Dr. Leonard McMillan, Dr. Steven Molnar and Dr. Jan Prins, have provided me with thoughtful suggestions at every step and have graciously accommodated changes to my timeline. Much of the direct inspiration for this research arose from the discussions and work of the Advanced Research in Graphics Hardware (ARGH!) group. In particular, Justin Hensley and Greg Coombe have served as co-authors, collaborators and informed sounding boards throughout the entire process.

During my stay in Chapel Hill the talented computer science department staff provided a pleasurable and productive working environment. This is also true of ITS who supported the computational research cluster on which the dissertation experiments were executed.

Enjoying food and shelter as I do, I am grateful to the funding sources that have permitted me to do this research: the state of North Carolina, the National Science Foundation, the NVIDIA Corporation, and the Paul Hardin Dissertation Fellowship.

Directly assisting with the production of this document were Barbra Gregory, who

transformed my rough diagrams into polished images worthy of publication, and Thorsten Scheuermann, who assisted with editing.

Throughout my life I have been supported, encouraged, and taught by my parents, family, and friends. Without their backing I would not be the person that I am, and this dissertation would certainly not have been attempted, much less completed. Mom, Dad, Dan, Sarah, Ben, Bubbie, Zayde, the rest of my wonderful family, my life long friends from Montpelier, my college and graduate school friends: you have kept my spirits hight and priorities straight. Thank you.

More important than anything else I did in Chapel Hill, even this dissertation, was finding my financée, Barbie. She has supported me in so many ways, that I don't have the words (much less the space) to express my gratitude and love.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

CHAPTER 1

INTRODUCTION

> "Computer graphics is now a mature discipline. Both hardware and software are available that facilitate the production of graphical images as diverse as line drawings and realistic renderings of natural objects. A decade ago the hardware and software to generate these graphical images cost hundreds of thousands of dollars. Today, excellent facilities are available for expenditures in the tens of thousands of dollars . . . "
>
> — *Procedural Elements For Computer Graphics* (Rogers, 1985, xi)

Not only has the cost of graphics hardware fallen by an order of magnitude each of the last three decades, but the expectations of graphics consumers has grown correspondingly. The original applications that inspired and funded early computer graphics research, such as scientific visualization, Computer Aided Design (CAD), and flight simulators, continue to flourish. However, the primary driver of the computer graphics industry, and therefore research, is now entertainment applications such as movies and video games, which have insatiable demands for higher quality imagery.

Not content with wireframe or plastic looking objects, movie audiences now expect photo-realistic scenery, characters, and special effects. In addition to the careful work of whole teams of artists and the sheer increase in computer speed, this has been achieved by the increasing sophistication of computer graphics rendering algorithms. One of the many areas of improvement has been the more faithful simulation of the physics of light. Shadows, indirect lighting, and reflections are all natural effects that we expect to see in an image when appropriate. However, this kind of simulation can be very expensive. Each of the 24 images per second in a movie often requires several hours of computation.

In 2005, domestic video game sales overtook movie box office receipts (Crandall and Sidak, 2006; MPAA, 2006). Much of this success can be attributed to the increasing sophistication of consumer graphics hardware available at reasonable prices. This has allowed game developers to produce interactive experiences that are visually closer to movies. There is a crucial difference that explains the still obvious gap in quality. Movies require the generation of approximately one quarter of a million images, but only once. Stored on film or disk, the images are simply displayed to viewers. Interactive video games, however, must produce dozens of unique images every second for each and every player. Even with the most powerful commodity graphics hardware available, only a simplistic light transport algorithm can typically be used. This dissertation presents a novel architecture that can interactively generate images using an advanced light transport algorithm, photon mapping. This will allow for more photo-realistic imagery in interactive applications such as video games.

1.1 Realistic image synthesis

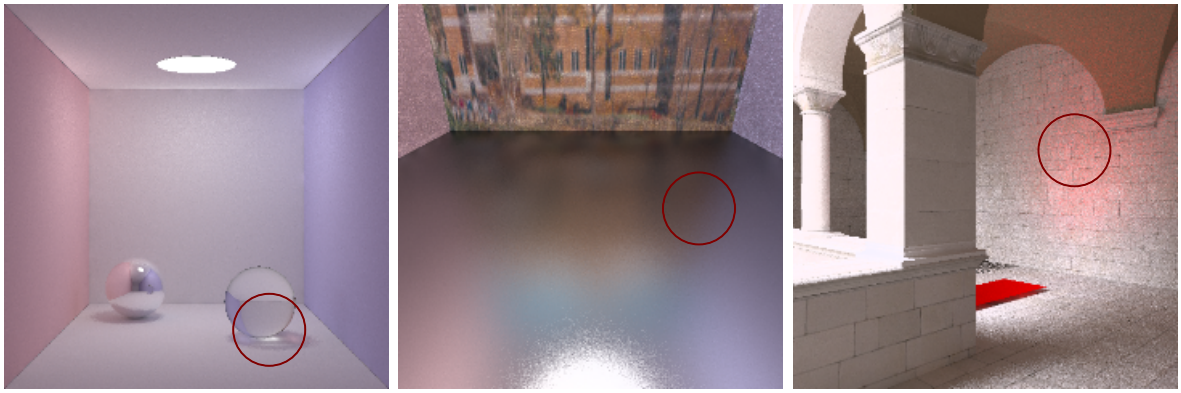
Modelling the interaction of light and the objects in a scene is the essence of realistic image synthesis. The detail with which we can specify the geometry of a scene has improved remarkable over the years. Commodity graphics hardware allows the interactive viewing of all but the largest of scenes. It is the more accurate simulation of light transport that will differentiate the next generation of graphics hardware.

Local illumination models assume that light is emitted by a source, reflects off a single surface and is captured on the viewing plane forming an image. Although sophisticated physically-based and directly acquired models of reflection are now in use, the framework in which they are used in commodity hardware is relatively simple. The illumination at any point in the scene is independent of the rest of the scene. This separability enables a high degree of parallelism that is well exploited in existing hardware.

Real world illumination at a single point, however, is dependent on the entire scene. One object may cast a shadow on another, light may bounce indirectly off of multiple objects, there may be highly specular mirrors, caustics may form, etc. These effects can be painstakingly added one by one to an image generated with local illumination models using single purpose rendering algorithms such as shadow volumes (Heidmann, 1991), environment maps (Blinn and Newell, 1976; Greene, 1986), pre-computed radiosity textures (Cohen and Wallace, 1993), or even a highly specific glittering gem effect (Gosselin, 2004).

However, these algorithms are hard to combine or generalize and often fail to capture essential parts of the underlying physics: the shadow maps may cause aliasing; environment maps are generally incorrect; the sparkling gem effect may fail when the gem is submerged in water; and the pre-computed radiosity textures have to be recomputed if the scene changes significantly. A more accurate simulation will require less programmer and artist time for each scene. It will therefore be more economical as processing power becomes more plentiful and cheap relative to the salaries of skilled labour. Generic global illumination models, used at least partially in many photo-realistic renderings, create the visual effects directly by simulating, with varying accuracy, the physics of light transport between objects in a scene.

The very benefit of global illumination, the correct simulation of multiple interactions of light with the scene, also explains why generic global illumination algorithms are so rarely used in interactive applications. The illumination at every point in the scene depends not only on the light sources but also recursively on every other visible point in the scene, because they *may* be reflecting light. This recursion is almost always expensive to compute, and global illumination algorithms can be distinguished primarily by how they break the recursion. Photon mapping, the algorithm used in this dissertation, terminates the recursion very early while still supporting effects arising from multiple reflections.



(a) The light passing through the glass sphere is refracted, forming a caustic on the floor.

(b) A highly glossy, but not specular, floor reflects a blurred image of a textured wall.

(c) Indirect lighting coming through the archway reflects off a glossy red polygon onto the wall.

Figure 1.1: Photon mapping is a powerful and efficient global illumination algorithm. These images demonstrate some particular effects handled well by photon mapping. This dissertation presents a hardware architecture capable of rendering these images at interactive rates.

1.2 Photon mapping

Photon mapping (Jensen, 1996a; Jensen, 2001) is a popular and robust global illumination algorithm. It can reproduce a wide range of visual effects including indirect illumination, color bleeding, and caustics on complex diffuse, glossy and specular surfaces represented using arbitrary geometric primitives, as demonstrated in Figure 1.1. The algorithm generates an image in two phases.

The first phases breaks the energy of the light sources into discrete packets called *photons*[1]. The photons are shot into space and allowed to probabilistically reflect, refract and be absorbed by surfaces. At every intersection with the scene, the photons, their locations, energy, and direction of travel, are recorded in the *photon map*.

The actual image is generated in the second phase. Although the process has multiple steps (see Chapter 2 for details) it uses the photon map as an estimate of the amount

[1]The term photon is somewhat misleading because they do not represent actual physical photons. The most apparent restriction is that only a subset of ray optics is supported. The name is however highly suggestive of the initial algorithm and has become entrenched.

and quality of light arriving at any point in the scene from all directions. To estimate the incident radiance, a photon gather is performed. This is essentially a k-Nearest-Neighbors (kNN) search, finding the k photons closest to the point of interest from the map. The photon gather is used to terminate, at an early stage, the recursive evaluation of light transport.

The realism of the photon mapping comes at a cost, in terms of both computation and memory bandwidth. Using the naive photon map algorithm on the Sponza scene, as described in Section 1.4 and shown in Figure 1.3, 145 million floating point operations are required for a single 512×512 image. This computational cost is significant, but manageable given the tremendous increase in semiconductor capabilities. The memory bandwidth requirements, 367 Gigabytes (GB) per image, is a more significant problem because off-chip bandwidth has not improved at the same rate as computational power.

This cost is very significant for interactive applications that must generate a new image dozens of times a second. At thirty images a second, the memory bandwidth requirement reaches 11 Terabytes (TB) per second. In current semiconductor technology this is a prohibitively large bandwidth for a desktop-sized machine. Improvements are therefore necessary if photon mapping is to be made practical for interactive use.

1.3 Summary of original contributions

This dissertation presents several novel techniques to dramatically reduce the bandwidth cost of photon mapping. These techniques are then combined into a practical hardware architecture that can support interactive applications. Taken together, these contributions support my thesis statement:

> Scenes with *complex illumination* can be rendered using the photon mapping algorithm at *interactive rates* by a *hardware architecture* which can *feasibly* be implemented using semiconductor technology available in 2010.

Specifically, the architecture combines the following techniques to reduce the bandwidth requirements of photon mapping:

Low bandwidth photon gathers using reordering: Photon-gather reordering produces the same image as the naive photon mapping algorithm, but reorders the computations such that the memory accesses are more coherent (Steinhurst et al., 2005). A 128 KB cache is then able to exploit this coherence and reduce the bandwidth to main memory by over an order of magnitude.

Tiled irradiance caching with pre-computed radius for split-sphere heuristic: Irradiance caching is commonly used to reduce the number of final gathers required (Ward et al., 1988). However, the conventional irradiance caching algorithm creates a sequential dependency between the pixels of an image preventing either efficient parallel execution or the application of photon-gather reordering. The image is divided into tiles each with a separate irradiance cache. To remove the dependency within each tile, I present a new technique for calculating the split-sphere heuristic. Together these technqiues provide a means to apply irradiance caching to a finely-grained parallel photon mapper without introducing significant additional error to the generated images.

Resource constrained importance sampling: Global importance sampling, which generates final gather rays in proportion to both the incident radiance and the reflectance function, requires fewer final gather rays to compute images of the same quality. This in turn reduces the number of photon gathers and the amount of bandwidth required. I present *combined importance sampling*, which, compared to existing schemes, requires limited bandwidth, computation and intermediate storage (Steinhurst and Lastra, 2006).

1.4 Scenes of interest

The intended use of the photon mapping architecture is for interactive applications such as video games and architectural walkthroughs. To evaluate the proposed system I have used variations on two scenes with the following qualities:

- Varied surface reflectance properties: diffuse, glossy and specular. This is important to fully test the combined importance sampler. It also allows a more complete evaluation of gather reordering when considerable coherence is already present in the gather requests.

- A camera position that views less than half of the scene. This exposes a potential weakness of bi-directional global illumination algorithms such as photon mapping, in that effort may be wasted because not all of the photon map is used.

- Varied illumination complexity. The light in scenes with high illumination complexity must navigate through several reflections before entering the camera. In scenes such as these, the indirect illumination is very important and is a good case for photon mapping. In simpler scenes most of the light is due to direct illumination and the indirect illumination calculated by photon mapping is subtle.

These requirements will result in a fair evaluation of photon mapping, irradiance caching, importance sampling, and query reordering, thereby making the architecture simulation representative of the desired workload. I have chosen two basic scenes that, with some variations, meet these criteria and sample the space of potential uses:

Global illumination demonstration There are certain global illumination phenomena, which can be subtle, for which the photon map algorithm is particularly good, such as caustics and color bleeding. This scene would exaggerate these effects with camera position and scene composition to closely examine the effects. I have chosen

Figure 1.2: Several variations of this global illumination test scene, a slightly modified Cornell box, are used in this dissertation. Note that the scene is a simple box with a single area light, a mirrored sphere and a glass sphere, and colored walls. This scene exhibits indirect illumination as color bleeding, reflections, and a bright focused caustic under the glass sphere. This image using the naive photon mapping algorithm requires 50 gigabytes of memory bandwidth and 37 billion floating point operations.

a variation of the standard Cornell box, Figure 1.2, traditionally used for global illumination evaluation. The geometry is very simple, a handful of rectangles, but it meets the other criteria. In later chapters I will use further variations, for example adding extra objects for complexity or changing the surface reflectance. Alternatively, the light may be replaced by a spot light aimed at the ceiling, eliminating any direct lighting, thus emphasizing the requirements for a quality computation of the indirect illumination.

Architectural walkthrough Global illumination is a necessity for complex scenes with indirect illumination. However, architectural models with individual rooms connected by doors, windows, and archways can waste computation in bi-directional algorithms such as photon mapping. I have used the Sponza Atrium model, provided by Marko Dabrovic of RNA Studios, as a test scene for exploration. The

Figure 1.3: The Sponza atrium, modeled by Marko Dabrovic of RNA Studios and used with permission, is a large complex structure with a central atrium and corridors that overlook the atrium. The sky is modeled as an area light the size of the top opening, all illumination in the corridors is indirect. The naive photon mapping algorithm generates 367 gigabytes of traffic to the photon map memory and uses 209 billion floating point operations.

geometry is complicated and is composed of different regions with small openings onto a shared atrium. The light is modeled as coming from the sky, a large area light. The light spilling through the archway is all indirect. Without global illumination this scene would be mostly black. Minor variations such as the introduction of small glossy reflectors are used in evaluating the architecture.

1.5 Dissertation summary

This dissertation is divided into three parts: background, individual techniques, and the combined architecture. Chapter 2 provides an overview of the evolution of graphic hardware architectures, global illumination algorithms (including a detailed explanation of photon mapping) and the criteria of the target implementation technology.

Inspired by techniques commonly used in high performance computing I explore two

methods for reducing the bandwidth costs of photon mapping without changing the resulting image. Chapter 3 show that a highly effective, if impractically expensive, reordering technique is able to reduce the required memory bandwidth by four orders of magnitude. A more practical reordering that is amenable to hardware implementation reduces the requirements by one order of magnitude. An additional reordering that further improves upon this is discussed. Instead of modifying the algorithm, Chapter 4 examines using different data structures to store the photon map. This approach is found to have negligible effects on bandwidth compared to photon gather reordering.

Irradiance caching and importance sampling are both well known techniques to reduce the number of final gather rays, and hence photon gathers, required. In Chapters 5 and 6, I present modifications to make these approaches more amenable to parallel hardware implementation while maintaining image quality. The effectiveness of these techniques depends greatly on the scene and can vary from a 50% reduction in memory bandwidth to an order of magnitude. Both techniques can be made compatible with photon gather reordering, thus we need not choose between one and the other.

Finally, these three techniques are then combined in an architecture, presented and evaluated in Chapter 7. The architecture has been functionally simulated and found to be practical, scalable, and efficient. Two boards containing a total of 8 replications of a custom chip will be able to render variations of the Cornell box scene and Sponza model at 30-128 frames per second.

CHAPTER 2

BACKGROUND

2.1 The evolution of graphics hardware

Since the earliest days of squiggles on oscilloscopes, the architects of computer graphics hardware have sought to achieve the most realistic image generation possible at interactive rates while constrained by the available technology. Kurt Akeley has categorized graphics hardware into generations (Akeley and Hanrahan, 2001). These generations are determined on the basis of functionality rather than technological considerations such as size or cost. Each generation was therefore dominant in different years depending on the portion of the market, i.e. price range, considered: research, high-end, and consumer-level commodity parts.

The first generation to be sold commercially generated wireframe images. Based on oscilloscope and plotter technology, the display devices were vector based and did not require prohibitively large amounts of memory. The major difference between the architectures in the first generation was the algorithm used to solve the *visibility* problem, the removal of the surfaces that are behind other objects from the point of view of the camera.

The focus of second generation architectures was on reducing the computational cost of the visibility calculations. As multi-megabyte memories became affordable, raster display devices quickly began to dominate. Raster displays store an array in memory, called a frame buffer, that records the color that should be displayed for every pixel of the image. Although this introduces resolution dependent artifacts into generated images, it

decouples image generation and display. This decoupling allows for the simple z-buffer visibility algorithm (Catmull, 1974). Although memory was slightly less expensive than it had been, it was still difficult to justify the inclusion of much more than the frame buffer. When these machines were built the gap between memory access time and computation speed was not as significant as it would soon become. This change will have a large impact on system design.

By the third generation, semiconductor technology improvements permitted interactive systems to both use more computation per frame and have significantly larger memories. The additional computation allocated to rasterization allowed application programmers to use more polygons, allowing for more geometrically detailed models. The lighting models also became more complex, although still restricted to local illumination. With the additional memory, system designers were able to incorporate texture mapping, a technique capable of adding significant information to the generated image without increasing the detail of the geometric model. Texture mapping not only requires large memories, but generates a large amount of memory traffic requiring high bandwidth interconnects. The speed of the texture memories became a technological problem and required careful, and often expensive, engineering.

The RealityEngine was a successful high-end third generation machine (Akeley, 1993). Through extensive replication of texture memories high performance was achieved without the use of caches. Although never built due to business issues, the NEON architecture is highly indicative of the design of consumer systems of the late 1990's (McCormack et al., 1998). Due to the low target selling price, the designers of the NEON were unable to justify the use of expensive replication. Instead, the design emphasis was on maximizing the power of a very small texture cache. This was performed by a careful reordering of the rendering process.

The vast majority of commercially available products today fall into the fourth generation. Rasterization and visibility are no longer the central performance limitations

for most users. Consequently, system designers are able to devote their new resources to adding programmability throughout the pipeline. Computationally expensive vertex and pixel shaders provide sophisticated local illumination effects and can be pushed to efficiently perform *limited* global illumination such as Whitted-style ray tracing and direct visualization photon mapping (Purcell et al., 2002; Purcell et al., 2003).

An early example of the fourth generation was the UNC PixelFlow project (Molnar et al., 1994; Eyles et al., 1997). The shading calculations were decoupled from rasterization, reducing the impact of complex shading models.The sort-last ordering combined with a large number of SIMD processing elements enabled massive parallelism throughout the machine. The NVIDIA GeForce6800 is a recent commodity graphics processor (Montrym and Moreton, 2005) made possible by the tremendous increase in computational capability of semiconductor technology. As memory access speeds and total bandwidth have lagged behind the growth in computation, efficient use of the memory cache and bus have become even more crucial in order to achieve acceptable performance.

The fifth generation of graphics hardware is still in the research phase and there is uncertainty about what algorithms and architectures will succeed commercially. It seems clear however that global illumination support is the key feature targeted by system designers seeking to improve the quality of image generation (Chalmers et al., 2002; Dutré et al., 2003). Section 2.5 reviews the previously reported global illumination architectures. In this dissertation I examine in detail one specific algorithm, photon mapping, and design a complete architecture to evaluate its fitness as the algorithm of choice for the fifth generation of graphics hardware.

The chapter begins by laying out the semiconductor technology targeted during the design and evaluation of my architecture. The design choices are driven by the continuing trend towards tremendous amounts of computation capability while only modest gains in memory bandwidth are predicted. Before presenting the details of photon mapping and the difficulties it presents a system designer, it is necessary to review the terminology

and notation of light transport simulation as well as other established algorithms for generating images with global illumination.

2.2 Target implementation technology

The goal of this dissertation is to present a photon mapping architecture that can be implemented in the next three years using no more than one custom designed chip. Furthermore, the system should fit within a single host workstation, using one or more expansion cards, to reduce system complexity and cost. The computational power and memory bandwidth that the architecture and algorithms can require are therefore constrained. Although semiconductor technology has improved both resources at a staggering rate, the growing gap between computational performance and memory bandwidth is a severe challenge.

A suitable measure of the computational cost of a global illumination algorithm is the number of floating point operations required, as they are significantly more expensive than integer or bit-vector arithmetic. A single FLoating point OPeration is commonly abbreviated as a FLOP. The rate of computation is then measured in FLOPS, floating point operations per second. Frequently the need to discuss the total number of floating point operations that a process requires arises, without regard to execution time, in which case the abbreviation FLOPs, with a small s, is used as the plural of FLOP.

Figure 2.1 shows the growth of computation in traditional consumer graphics processors as measured in floating point operations per second. This growth in computation matches the transistor density trend data provided by the International Technology Roadmap (ITRS, 2006). Because fourth-generation graphics algorithms are easily cast in the stream processing paradigm (Venkatasubramanian, 2003), graphic hardware architectures are able to increase their performance on nearly the same curve as the underlying semiconductor technology. It is therefore expected that commodity graphics hardware

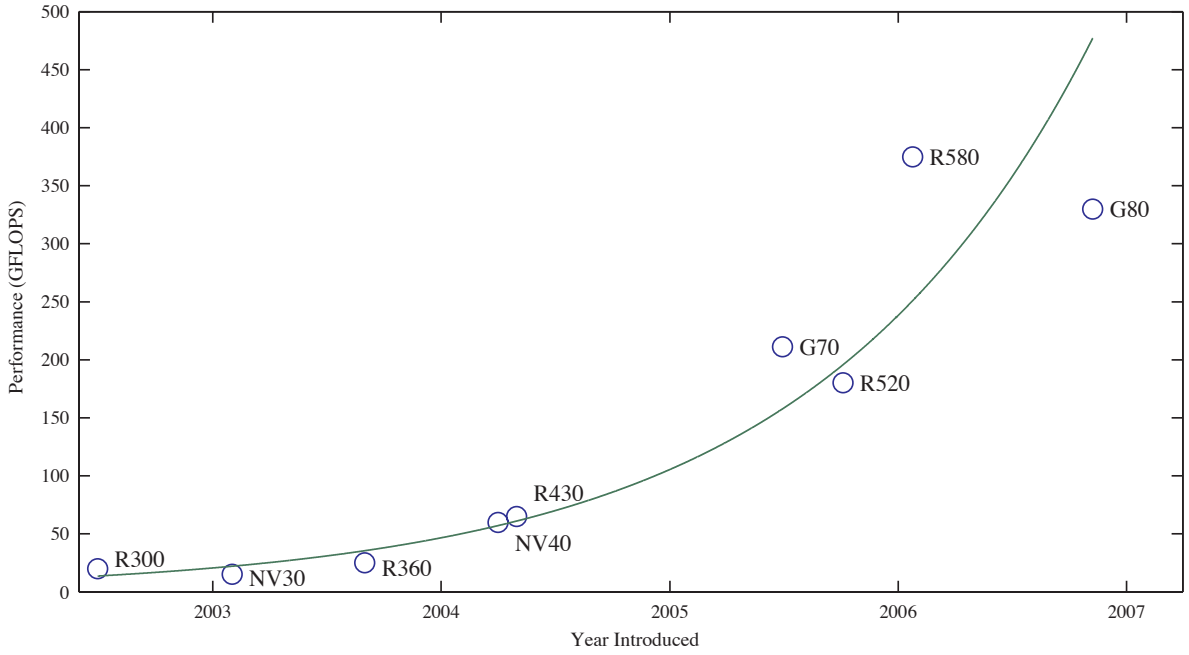


Figure 2.1: Growth in GPU computation. The exponential growth of transistor density has enabled commodity graphics hardware to perform an increasing number of floating point operations per second. *This data, along with that presented in Figure 2.2, was originally collected by Lastra and extended to include more recent products using published manufacturer specifications (Lastra, 2006; ATI, 2007; NVIDIA, 2007).*

will continue to increase the amount of computation that can be performed on a single chip. A design that requires no more than 500 GFLOPS will not present a problem by 2010, the timeline envisioned for the architecture in this dissertation.

It is instructive to compare this exponential growth in performance of graphics hardware to that of traditional CPUs. A CPU typically allocates a large portion of its transistor budget to both large memory caches and complex control logic, leaving relatively little for the actual arithmetic and logical functions. The recent trend towards multiple cores on a single chip is an attempt to take advantage of the additional computational power while constrained by inherent lack of parallelism in individual workloads.

Unlike computation, memory bandwidth has been growing at a linear rate as shown in Figure 2.2. This growing gap remains despite a significant investment in inter-connect technologies. Algorithms and architectures must therefore minimize their requests for

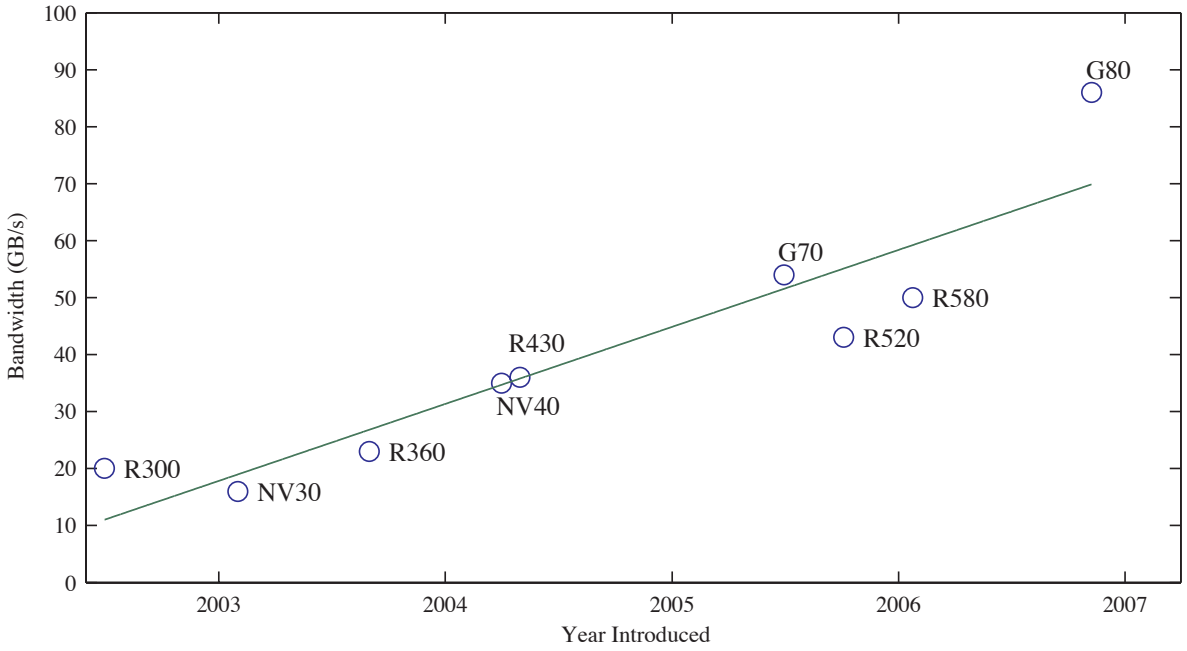


Figure 2.2: Growth in GPU bandwidth. The growth in bandwidth available on commodity graphics hardware, while impressive, has been linear. The increasing gap between computation and bandwidth must be directly addressed by any global illumination hardware architecture.

data from memory. Global illumination algorithms typically reference the same data, from large data structures that can not be kept locally, multiple times. Care must be taken to reduce the required bandwidth either by using effective caches or by reordering the computations to reduce redundant accesses. A design that requires no more than 90 GB per second for each chip will be feasible in 2010.

2.3 Realistic image synthesis

Realistic image synthesis as introduced in Chapter 1 is the computational art of simulating the science of optics: the generation, propagation and detection of light in a scene. There are highly accurate physical models of light, such as photonics, electromagnetic, wave, and quantum, that explain almost all observed visual effects including diffraction and polarization. However, these models are currently too computationally intensive

for general use in computer graphics. This dissertation, and the majority of computer graphics systems, utilizes the *ray optics* model of light propagation (also known as *geometric optics*) which would be familiar to early optical researchers such as Johannes Kepler, Willebrord Snell and Isaac Newton.

The principal restrictions of ray optics are that light travels in straight lines, at an infinite speed, and is not influenced by external factors such as electromagnetic fields or gravity. The rays of light are redirected at the boundaries of objects either according to simple geometric rules, such as Snell's law of refraction, or by more complicated functions described in Section 2.3.2. Although ray optics is a simple model, it is capable of expressing the majority of visual effects seen in the world, such as reflection, refraction, and image formation. There are several excellent texts outlining the algorithmic application of ray optics to computer graphics, such as (Shirley, 2000) and (Glassner, 1995).

In this section we apply radiometry, the measurement of light energy transfer, to ray optics to discuss a single equation from which all standard global illumination algorithms can be derived. Monte Carlo integration is presented as a technique commonly used to solve this equation while generating images. There are several well-writen introductions to these topics, in particular those of Cohen (Cohen and Wallace, 1993, Chapter 2), Jensen (Jensen, 2001, Chapter 2) and Dutré (Dutré et al., 2003, Chapter 2) upon all of which this section draws.

2.3.1 Radiometry

Physically based global illumination algorithms are expressed in terms of radiometric units, which are used to quantify light. These terms are listed in Table 2.1 and briefly described here. The basic unit of light is a single photon, which carries a certain amount of *radiant energy*, expressed in Joules (J) and denoted as Q. The total amount of energy arriving or departing an entire surface via a set of photons at any instant in time is

Symbol	Quantity	Unit
Q	Radiant Energy	J
Φ	Radiant Power (Flux)	W
$E(x)$	Irradiance (Incoming)	W/m^2
$B(x)$	Radiosity (Outgoing)	W/m^2
$L(x, \vec{\omega})$	Radiance	$W/m^2/sr$

Table 2.1: Common radiometric symbols, terms and units. x is a specific surface point. Radiance may be either incoming or outgoing depending on the direction of $\vec{\omega}$.

measured in Watts (W) (Joules per second) and referred to as *radiant power* or *flux*.

$$\Phi = \frac{dQ}{dt}$$

A light source, such as a household light bulb, can be described as emitting a certain number of Watts[1]. In a similar fashion, the rate at which photons arrive on an office desk from all directions is a measure of flux.

To describe the movement of light energy over time at a specific point, x the flux is differentiated over an area and is measured in Watts per square meter. This unit is referred to as *radiosity*, $B(x)$, if the light is being emitted and *irradiance*, $E(x)$, is the light is incident. The term irradiance is an unfortunate historical artifact as it is *not* the opposite of radiance, defined shortly.

$$E(x), B(X) = \frac{d\Phi}{dm^2}$$

Radiosity does not distinguish the direction that light was traveling when measured and is therefore unsuitable for making critical measurements such as the amount of light arriving at a camera from a particular angle.

[1]Light bulbs are labeled by the electrical energy they consume over time, also measured in Watts. Because light bulbs are inefficient, the majority of energy is dissipated as heat, the radiant power of a light bulb will be significantly smaller than the electrical usage.

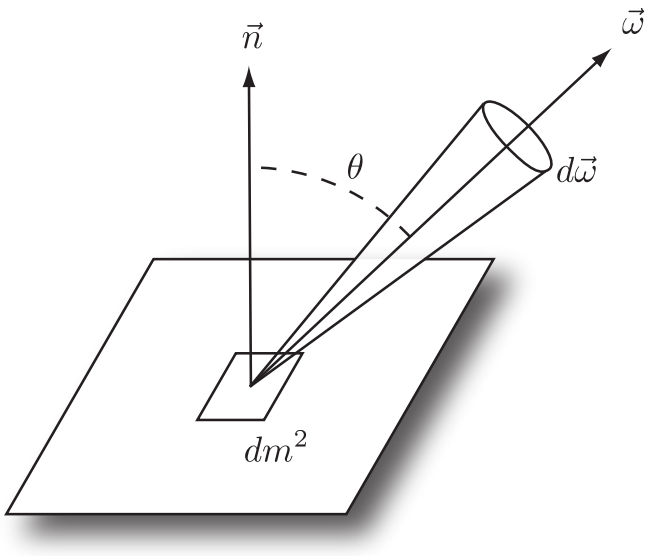


Figure 2.3: Radiance is the total radiant flux arriving across, or departing from, a differential area from the range of directions in a differential solid angle around $\vec{\omega}$. Depending on the direction of light transport, radiance is categorized as incident or exitant. Radiance is the physical unit measured by image sensors such as eyes and cameras.

Radiance, $L(x, \vec{\omega})$, is the flux across a differential area from a specific differential direction (solid angle) and is measured in Watts per square meter per steradian (Figure 2.3).

$$L_i(x, \vec{\omega}) = \frac{d^2\Phi}{dm^2 d\omega}$$

Radiance may be either incident, $L_i(x, \vec{\omega})$, emitted, $L_e(x, \vec{\omega})$, or reflected, $L_r(x, \vec{\omega})$. The exitant radiance is the sum of the emitted and reflected radiance, $L_o(x, \vec{\omega}) = L_e(x, \vec{\omega}) + L_r(x, \vec{\omega})$. Image sensors, such as our eyes, cameras, and virtual cameras, are modeled as sensors of radiance. Radiance also plays a critical role in the modeling of reflection and refraction on non-diffuse surfaces.

In this dissertation it is assumed that the space between objects is empty. This assumption can be combined with the property of ray optics that states that light travels

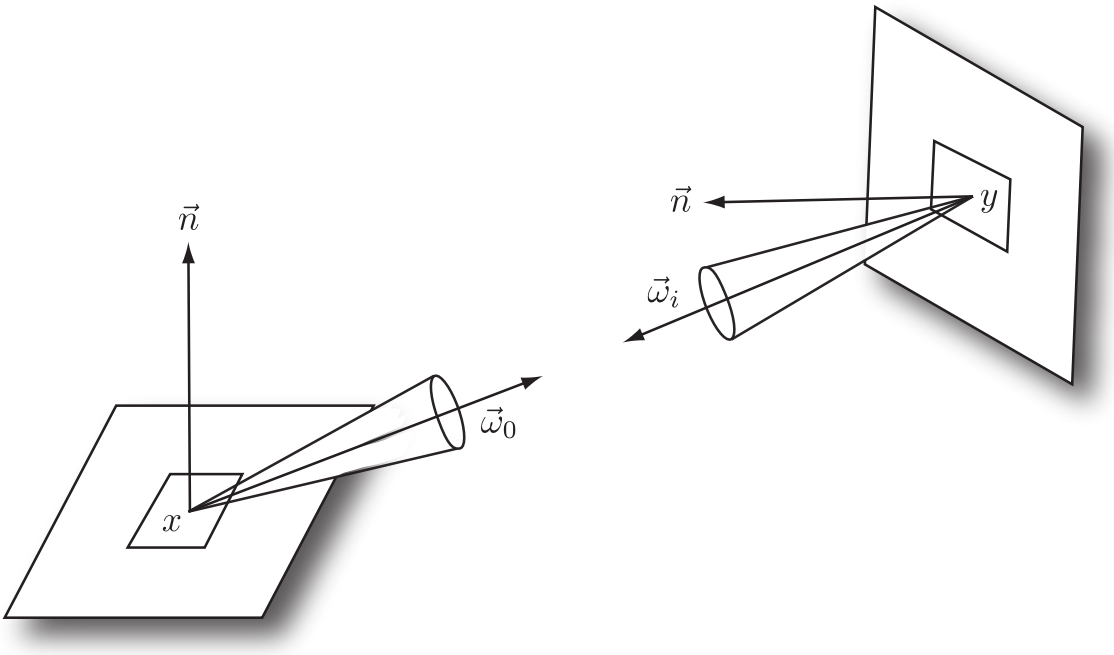


Figure 2.4: The invariance of radiance. The exitant radiance leaving a point in the direction of another is exactly equal to the radiance arriving at second point from the direction of the first.

in a straight line to form an important property of radiance. As shown in Figure 2.4, the exitant radiance at point x in the direction $\vec{\omega}_o$ towards point y is equal to the radiance that arrives at y from the direction, $\vec{\omega}_i$ of x.

$$L_o(x, \vec{\omega}_o) = L_i(y, \vec{\omega}_i)$$

This critical property allows us to compute the incident radiance at a point if we already know the exitant radiance at other points in the scene. This observation forms the basis of the rendering equation.

2.3.2 The rendering equation

An ideal pinhole camera (Figure 2.5) measures the flux arriving on the image plane at the location p for each pixel, potentially sampled in order to improve image quality or

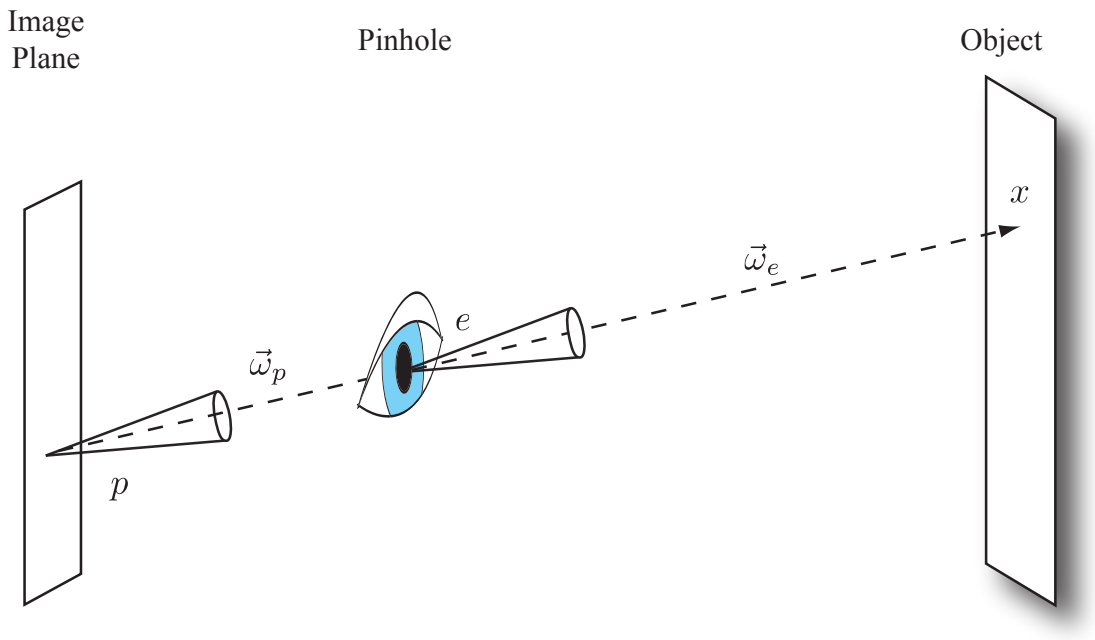


Figure 2.5: Image formation with an ideal pinhole camera. For every pixel p in the image, the radiance arriving at the eye-point e from x in the direction $\vec{\omega}_p$ must be computed. $L_i(e, \vec{\omega}_p) = L_o(x, -\vec{\omega}_p)$

enable additional visual effects. Because the pinhole allows only one ray of light to reach any point on the image plane, it is the incident radiance, $L_i(p, \vec{\omega}_p)$, that is measured. By the invariance of radiance, this is the same radiance passing through the pinhole, at position e, along the same ray $\vec{\omega}_p$. The position e is commonly referred to as the camera location or eye-point. The ray originating at e in the direction $\vec{\omega}_p$ is called an eye-ray.

Each eye-ray must be intersected with the scene to find the first intersection point, x, Although the majority of graphics hardware uses rasterization to resolve visibility, ray-casting is now a practical approach, even for interactive applications (Section 2.5). Once the intersection has been found, the invariance of radiance can be used to determine the incident radiance at the eye, and hence on the image plane, in terms of the exitant

Pre-condition: e is the location of the eye (camera)
Pre-condition: Any per-scene or per-image pre-procesing has been performed
  for each pixel (u, v) of the image **do**
    for each sample **do**
      image$(u, v) \leftarrow 0$
      // Form the eye-ray $(e, \vec{\omega}_p)$
      $p \leftarrow$ sampled location on the image plane of pixel (u, v)
      $\vec{\omega}_p \leftarrow (e - p)$
      // Use ray-casting to find the closest intersection in the scene
      $x \leftarrow$ intersect_with_scene$(e, \vec{\omega}_p)$
      // Compute $L_i(e, \vec{\omega}_p)$ based on $L_o(x, -\vec{\omega}_p)$
      image$(u, v) \leftarrow$ compute_exitant_radiance$(x, -\vec{\omega}_p)$
    end for
    image$(u, v) \leftarrow$ image(u, v)/(# samples)
  end for
Post-condition: image = the generated image

Algorithm 2.1: The pixel-driven rendering algorithm, adapted from (Dutré et al., 2003). The quality of a generated image depends on the exact implementation of the method compute_exitant_radiance().

radiance at x back along $\vec{\omega}_p$.

$$L_i(p, \vec{\omega}_p) = L_i(e, \vec{\omega}_p) = L_o(x, -\vec{\omega}_p)$$

Algorithm 2.1 provides a high-level view of this process, deferring for the moment how the exitant radiance, L_o, is computed.

The exitant radiance at x towards the eye, or any other point in the scene viewed from a particular direction, is the sum of the reflected, refracted and locally emitted radiance. The emitted radiance, L_e, is computed directly from the description of the object containing x. For most surfaces, the value is uniformly zero. The amount of light that is reflected and refracted towards the viewer, however is more complicated. It depends on both the light that arrives on the surface, from all directions, and the specific material properties of the surface. When a ray of light lands on an object at a particular point, z, from a particular direction, $\vec{\omega}_i$, the energy will either be absorbed, or emitted

from a potentially different point, z', in a potentially different direction, $\vec{\omega}_o$. When restricted to reflection[2] this is known as the Bidirectional Scattering Surface Reflectance Distribution Function (BSSRDF) and has eight dimensions (Nicodemus et al., 1977).

Nicodemus also introduced a simplification of the BSSRDF where reflected light is assumed to be emitted from the same location where it landed ($z' = z$). The Bidirectional Reflectance Distribution Function (BRDF) is denoted as

$$f_r(z, \vec{\omega_i}, \vec{\omega_o}) = \frac{dL_r(z, \vec{\omega}_o)}{L_i(z, \vec{\omega}_i)(\vec{\omega_i} \bullet \vec{n}) d\vec{\omega}_i}$$

where $\vec{\omega}_o$ is towards the viewer, $\vec{\omega}_i$ is the direction from which the incoming light is arriving, $\vec{n}$ is the local surface normal at z, L_r is the reflected radiance towards the viewer, and $(\vec{\omega_i} \bullet \vec{n})$ represents the foreshortening due to the two surfaces not necessarily being parallel. Although this formulation precludes the representation of some desired visual effects, such as the sub-surface scattering seen in white marble statues, it can be computed without additional expensive pre-computation.

The perfectly specular BRDF, an ideal mirror, reflects the incoming light energy only along the reflected angle and is a special case. The BRDF of a more general surface will reflect some portion of the incoming light to the entire hemisphere of visible directions. Purely diffuse surfaces reflect the light evenly, while glossy surfaces exhibit a behavior between that of ideal specular and diffuse reflectors. Some examples of common BRDFs are illustrated in Figure 2.6. Global illumination algorithms differ in the BRDFs that they efficiently support.

The concepts of incident, reflected and emitted radiance and the BRDF are united in the rendering equation (Kajiya, 1986) using an integral with a domain of the hemisphere,

[2]Refraction can be analyzed in a symmetric fashion to reflection. For clarity, refraction is not considered throughout this section.

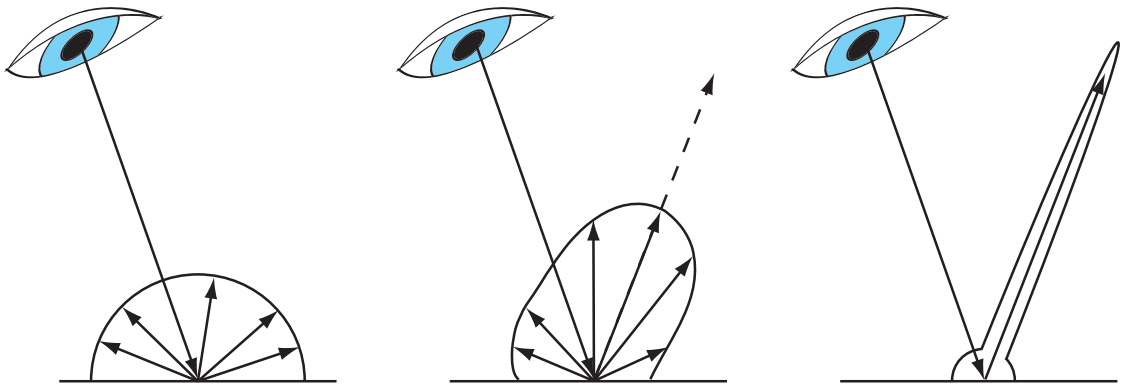


Figure 2.6: Examples of common BRDF classifications. The purely diffuse BRDF reflects light from all directions equally towards the viewer, while the highly specular surface, an ideal mirror, reflects only the light from a single direction. In between, glossy surfaces reflect light from all directions, but weigh those that come near the angle of specular reflection more heavily.

Ω, of incoming radiance at z and shown in Equation 2.1.

$$L_r(z, \vec{\omega}_o) = L_e(z, \vec{\omega}_o) + \int_{\Omega} f_r(z, \vec{\omega}_i, \vec{\omega}_o) L_i(z, \vec{\omega}_i)(\vec{\omega}_i \bullet \vec{n}) d\vec{\omega}_i \tag{2.1}$$

There are several interesting characteristics of the rendering equation. The first is that it is *recursive*, the light that reflects from z is dependent on the light that arrives at z from other surfaces, but that is in turn dependent on how much light lands on those other surfaces. Every global illumination algorithm based on the rendering equation must find a way to terminate this recursion in order to guarantee that the algorithm will halt. Secondly, the key operation while rendering an image is a *continuous integration*. In all but the most trivial of scenes, the integral can not be solved analytically. Both of these challenges must be addressed, in an implementation of compute_exitant_radiance(), before the rendering equation can be used to generate an image.

2.3.3 Numerical integration

The integrand of the rendering equation, shown in Equation 2.1, is the complicated product of three separate functions. The incident radiance function, $L_i(x, \vec{\omega})$ is generally unknown[3] and exhibits strong discontinuities because of changes in visibility. For these reasons, analytical integration can not be applied to the rendering equation. The approach of this section is inspired by (Veach, 1997).

When an integral can not be solved analytically in closed form, Equation 2.2, numerical integration can be used to find an estimate, $\hat{\Theta}$, of the true value Θ (Hamming, 1973). The integral is replaced by the weighted sum of N discrete point evaluations of the integrand, $g(s)$, as shown in Equation 2.3. There are an entire family of numerical integration estimators that differ in their selection of the points S_i at which $g(s)$ is evaluated and the weights W_i which describe how they are combined. The correctness and efficiency of these techniques depends on the exact function being integrated. In the case of computer graphics and the rendering equation, many techniques are not suitable.

$$\Theta = \int_{\Omega} g(s) d\mu(s) \tag{2.2}$$

$$\hat{\Theta} = \sum_{i=1}^{N} W_i g(S_i) \tag{2.3}$$

The rendering equation, $g(s) = f_r(x, \vec{\omega}_i, \vec{\omega}_o) L_i(x, \vec{\omega}_i)(\vec{\omega}_i \bullet \vec{n})$ is the product of three functions, with x and $\vec{\omega}_o$ fixed. The domain of integration, Ω, is the entire hemisphere of directions from which light arrives at x. Each sample S_i is a single direction light may be arriving from. $d\mu(s)$ is the differential solid angle and Θ is the reflected radiance reflected for which we are solving.

As a point of notation, statisticians customarily refer to each value S_i as an obser-

[3]Indeed, if $L_i(x, \vec{\omega})$ were known for all x and $\vec{\omega}$, then an evaluation $L_i(e, \vec{\omega}_p)$ for all eye-rays would produce the required measurements on the image plane for the final image with no further work. This is closely related to lightfields and lumigraphs (Gortler et al., 1996; Levoy and Hanrahan, 1996).

vation and the entire vector $< S_1, S_2, S_3, \cdots S_N >$ as a sample. In computer graphics it is standard to refer to each observation S_i as a *sample* and call N the sample size. The whole vector of samples is informally referred to using a variety of terms such as "sample locations" or in the case where Ω is the hemisphere, "sample directions". In this dissertation, the computer graphics terms are used exclusively.

A simple numerical integration technique used in introductory Calculus courses is the Riemann Sum. The samples S_i are spread uniformly throughout the domain Ω. Each sample is given a weight $W_i = \frac{1}{N}$. The Riemann Sum is a deterministic numerical integration technique because the sample locations S_i and weights W_i are fixed ahead of time.

Other deterministic integration methods include the trapezoid rule and Simpson's rule. Although beneficial in other domains, these methods are rarely useful in computer graphics because unless the integrand is smooth, a very large sample size will be required to achieve good results (Veach, 1997). Even more troublesome, extending these techniques to more than one dimension is very expensive. If N samples are sufficient for a given one-dimensional problem, N^d samples would be required for a d-dimensional problem. This is known as the "curse of dimensionality".

2.3.4 Monte Carlo integration

In contrast to these deterministic methods, Monte Carlo integration is stochastic. The samples, S_i, are chosen randomly from the region of integration, Ω. The generic Monte Carlo estimator of $g(s)$, G_N, is shown in Equation 2.4 (Rubinstein, 1981). Since all the weights in Monte Carlo estimators have a common term of $\frac{1}{N}$, this has been factored out of the summation.

$$G_N = \frac{1}{N} \sum_{i=1}^{N} W_i g(S_i) \tag{2.4}$$

If $W_i = 1$ and every point in Ω has the same probability of being sampled by S_i then this is called the Blind Monte Carlo technique. Although Blind Monte Carlo is very straightforward, samples are chosen from the domain randomly but uniformly, it takes a large value of N to have much confidence in the estimate. Importance sampling is one technique to improve the performance of the blind Monte Carlo estimator, and is discussed in detail in Chapter 6.

To apply Monte Carlo integration to the rendering equation, Equation 2.2, the position x and the viewing angle $\vec{\omega}_o$ are fixed. Instead of integrating over the continuous hemisphere Ω, the hemisphere is discretely sampled and the results summed as in Equation 2.5.

$$L_r(x, \vec{\omega}_o) = L_e(x, \vec{\omega}_o) + \frac{1}{N} \sum_{i=1}^{N} f_r(x, S_i, \vec{\omega}_o) L_i(x, S_i)(S_i \bullet \vec{n}) \tag{2.5}$$

2.4 Global Illumination algorithms

There are a wide variety of global illumination algorithms, only a few of which are be discussed here. They differ from each other in how they solve the rendering equation. Dutré et al. presents a well integrated theoretical framework (Dutré et al., 2003). Pharr and Humphreys provide a comprehensive practitioners guide with full implementations (Pharr and Humphreys, 2004). *Path tracing* is by far the largest class of global illumination algorithms. As these form the basis of photon mapping, they will be explored in some depth in this section. Finite-element methods, such as radiosity, use a fundamentally different approach to solving the rendering equation, the advantages and disadvantages of which are explored at the end of this section.

2.4.1 Path tracing

Path tracing algorithms establish light transport paths between lights sources and those points in the scene where we wish to compute the incident radiance. Often, these points are directly on the image plane, corresponding to the pixels of the rendered image. This approach yields the pixel-driven rendering algorithm presented in Algorithm 2.1. The path tracing methods differ in their implementation of compute_exitant_radiance$(x, \vec{\omega}_o)$ and any preprocessing that may be required before image generation begins.

2.4.1.1 Whitted-style ray tracing

Perhaps the best known path tracing algorithm is *ray tracing*, introduced to computer graphics by Whitted (Whitted, 1980). Requiring no pre-processing and being fully deterministic, the method is simple to implement (Algorithm 2.2). The desired exitant radiance is computed as the sum of three terms: the locally emitted light, the reflected light due to direct illumination from the scene-defined light sources, and the light reflected along the angle of perfect reflection or refraction. The integral in the rendering equation is replaced with one recursive evaluation of compute_exitant_radiance computing indirect illumination.

The algorithm is therefore only correct for perfectly specular surfaces, such as chrome spheres. More general materials, including those with a diffuse and glossy BRDF, are not handled correctly and the image generated will be biased. The recursion must somehow be terminated for an image to eventually be generated. Simple heuristic criteria for limiting the depth of recursion, such as running out of storage or a reaching hard-coded maximum depth, lead to biased results and a large number of shadow rays which contribute little to the final image (Shirley, 2000). These two issues lead to the development of a more sophisticated scheme, *stochastic ray tracing*.

Pre-condition: x is the location under consideration
Pre-condition: $\vec{\omega}_o$ is the direction towards the viewer
  // Compute the locally emitted radiance in direction $\vec{\omega}_o$
  result $\leftarrow$ emitted_illumination$(x, \vec{\omega}_o)$
  // Compute the direct illumination due to the pre-defined light sources
  result $\leftarrow$ result + direct_illumination$(x, \vec{\omega}_o)$
  // Compute the indirect illumination due to specular reflection
  if recursion termination criterion not yet reached **then**
    $\vec{\omega}_i \leftarrow$ compute_reflected_angle$(\vec{n}, \vec{\omega}_o)$
    $y \leftarrow$ intersect_with_scene$(x, \vec{\omega}_i)$
    result $\leftarrow$ result + $f_r(x, \vec{\omega}_i, \vec{\omega}_o) \times$ compute_exitant_radiance$(y, -\vec{\omega}_i)$
  end if
Post-condition: result = the total reflected radiance from x towards $\vec{\omega}_o$

Algorithm 2.2: Whitted-style ray tracing. Various heuristics can be used to terminate recursion. This simple implementation of compute_exitant_radiance$(x, \vec{\omega}_o)$ is valid only for perfectly specular surfaces, such as chrome balls. Nonetheless, it provides impressive results in suitable scenes.

2.4.1.2 Stochastic ray tracing

Cook introduced *distributed ray tracing* as a comprehensive framework for extending Whitted-style ray tracing using stochastic techniques, based on Monte-Carlo integration (Cook et al., 1984). It handles arbitrary material BRDFs, as well as motion blur, depth of field, and other realistic effects. In a straightforward implementation, arbitrary BRDFs are supported by replacing the single reflection sample, along the angle of specular reflection, with N randomly chosen samples using Equation 2.5 to compute the total reflected radiance. These random samples can be chosen uniformly, proportionally to the BRDF, or according to the quasi-Monte-Carlo technique (Shirley, 2000). Even an implementation of this basic technique that allowed only 4 levels of recursion, generating seriously biased images, would require N^4 evaluations of compute_exitant_radiance. As N is in the range of hundreds to thousands, this implementation of distributed ray tracing is highly inefficient, but can be easily adapted to preserve its strengths while reducing the number of computations that contribute little to the final image.

An alternative method (Algorithm 2.3) to terminating the recursion of the rendering

Pre-condition: x is the location under consideration
Pre-condition: $\vec{\omega}_o$ is the direction towards the viewer
  // Compute the locally emitted radiance in direction $\vec{\omega}_o$
  result $\leftarrow$ emitted_illumination$(x, \vec{\omega}_o)$
  // Compute the direct illumination due to the pre-defined light sources
  result $\leftarrow$ result + direct_illumination$(x, \vec{\omega}_o)$
  // Determine if recursion should continue
  $\rho \leftarrow$ probability of absorption
  if $\rho >$ random unit number **then**
    // Compute the indirect illumination due to specular reflection
    $\vec{\omega}_i \leftarrow$ sample_visable_hemisphere$(\vec{n}, \vec{\omega}_o)$
    $y \leftarrow$ intersect_with_scene$(x, \vec{\omega}_i)$
    result $\leftarrow$ result + $f_r(x, \vec{\omega}_i, \vec{\omega}_o) \times$ compute_exitant_radiance$(y, -\vec{\omega}_i)/(1 - \rho)$
  end if
Post-condition: result = the total reflected radiance from x towards $\vec{\omega}_o$

Algorithm 2.3: Stochastic ray tracing. Using Monte Carlo integration to support arbitrary BRDFs and Russian roulette to terminate the recursion, stochastic ray tracing is a simple but robust global illumination algorithm. Because only a single sample is generated and evaluated, many samples per pixel must be generated from the image plane.

equation is to use Russian roulette. Instead of generating N secondary rays at each point, a single ray is created and recursively evaluated. However, this ray is chosen randomly from the hemisphere, not the angle of specular reflection as in Whitted-style ray tracing. At each intersection the ray is probabilistically terminated, depending on how reflective the material BRDF is. To avoid biasing the result, the intermediate reflected radiance must be scaled by the probability of absorption. In exchange for reducing the branching factor of recursion, this method requires hundreds to thousands of samples from each pixel. This implementation of distributed ray tracing focuses its efforts near the top of the recursion tree and is far more efficient. A full range of improvements and optimizations to stochastic ray tracing can be found in (Shirley, 2000).

2.4.1.3 Additional path tracing algorithms

Although more expressive than Whitted-style ray tracing, stochastic ray tracing has the problem that many of directions sampled will contribute little towards the final image unless they are directed to those portions of the scene that either contain light sources or strong reflectors of light sources. This results in a noisy image, especially if any caustics are present. Two alternative path tracing techniques increase efficiency by utilizing the knowledge of the specified light sources.

Bi-directional path tracing, like stochastic ray tracing using Russian roulette, traces multiple rays from the eye deep into the scene, recording all the intersection points (Lafortune and Willems, 1993; Veach and Guibas, 1994). Rather than using shadow rays to compute direct illumination and propagating backwards through the ray, a second ray is traced from a light source, recording all its intersections. Using the visibility function, as determined by ray tracing, and a series of weights, derived from the rendering equation, the light power is transfered from the light ray to eye ray and accumulated at the pixel. Although each eye ray is significantly more expensive to process, in certain scenes it can take far fewer samples per pixel to reduce noise to an acceptable level.

Bi-directional path tracing can be enhanced by improving the generation of sample directions as more information is known. A sophisticated technique that uses this approach is *Metropolis light transport* (Veach, 1997). Starting with standard bi-directional path tracing, those paths that lead to a strong contribution to the image are used as guides indicating which portions of the scene are brightly illuminated. Care is taken to avoid bias while expending as few paths as possible on those portions of the scene not expected to contribute. An example where Metropolis light transport works well is two rooms separated by a keyhole, ordinarily a difficult scene to render without a lot of noise. Once even one path finds the keyhole, it is mutated, generating other paths that also get through the keyhole.

2.4.2 Radiosity

Classical radiosity computes a radiosity value for every surface point in the scene during a view-independent pre-process (Goral et al., 1984). The surfaces in a scene are divided into patches and initialized with the appropriate radiosity if the patch is part of a light source. Using a numerical linear system solver, the energy is allowed to transfer through the system until a steady state is found. This solution is then consulted by compute_exitant_radiance() directly during pixel-driven rendering, Algorithm 2.1. Based on energy balance equations from radiative heat transfer studies, radiosity has been validated to direct physical observation (Goral et al., 1984; Pattanaik et al., 1997). Recall that the radiometric term radiosity describes the total flux leaving a differential point in *all directions*. By not storing the hemispherical distribution of exitant radiance, the standard method can only support scene that are purely diffuse.

A significant amount of research has been published on improving the computational efficiency of radiosity, removing artifacts due to the selection of the patches and extending the technique to non-diffuse surface. These results are well summarized in (Cohen and Wallace, 1993) and (Dutré, 1994, Chapter 5). Unfortunately, some difficulties remain. The surfaces must be parameterizable in order to be divided into patches, adding complexities when arbitrary procedurally generated geometry is used. Additionally, many of the techniques to improve efficiency remove the regularity of operations and make it difficult to take advantage of parallel independent processors with limited communication.

2.4.3 Discussion

This section touched on just a few of the global illumination algorithms currently in use. More in depth examinations of global illumination techniques, and their variations, can be found in many excellent texts, such as those by Dutré (Dutré et al., 2003) and Pharr (Pharr and Humphreys, 2004). The critical criteria for evaluating a global illumi-

nation algorithm for a given use are:

- The restrictions imposed on the scene, both geometric and material properties
- The visual effects that are correctly and efficiently rendered
- The computation required to prepare intermediate data structures
- The ability to parallelize the workload on cost effective architecture

None of the algorithms examined so far in this chapter meet these requirements for the scenes and visual effects described in Chapter 1. In Section 2.6 photon mapping is presented as a possible contender implementation in fifth generation graphics hardware. The next section examines existing fifth generation graphics architectures, focusing on those that are used by or inspired the architecture presented in Chapter 7.

2.5 Fifth generation graphics architectures

As described in Section 2.1, it is expected that fifth generation graphics architectures will concentrate on adding support for global illumination. There is as of yet no commercially available examples of interactive[4] fifth generation architectures, however there are research designs that have been proposed and/or built. Many early hardware global illumination systems are discussed by Chalmers (Chalmers et al., 2002). In this section I discuss a sample of the more recent system architectures intended to be implemented as interactive hardware, that have inspired aspects of the architecture presented in this dissertation.

A series of papers and prototype machines from the University of Saarland define an efficient architecture for interactive ray tracing. The original SaarCOR design required

[4]The RenderDrive line of products by ARTSVPS are commercially produced offline accelerators for high quality ray tracing (ARTVPS, 2007).

a static scene and only supported a limited numbers of secondary rays (Schmittler et al., 2002). A highly efficient FPGA implementation demonstrated that the architecture was able to operate with very limited memory bandwidth by tracing rays in small bundles (Schmittler et al., 2004). More recently, a second implementation with partially dynamic scenes and programmable shaders with more secondary rays has been demonstrated (Woop et al., 2005). The architecture is still however a Whitted-style ray tracing, unable to efficiently handle diffuse inter-reflections, caustics, or other phenomena that the photon mapping algorithm handles well.

As described in Section 2.6, ray casting is one of the two most important computational kernels in photon mapping. The SaarCOR architecture casts coherent rays in order to reduce the required memory bandwidth significantly. The algorithms deployed in the architecture described in Chapter 7 take care to ensure coherency of not only the photon gather locations, but also the rays that are cast using a small instantiation of the SaarCOR.

The GI-Cube architecture provided Whitted-style ray tracing in the context of volume rendering (Dachille and Kaufman, 2000). The regular nature of the volume rendering allowed for a simple packet based routing structure to sort partially traced rays to improve cache coherencey. This provided both inspiration for the hashed photon gather reordering presented in Chapter 3 as well as the overall packet-based routing for the entire architecture presented in Chapter 7.

2.6 Photon mapping

Photon mapping (Jensen, 1996a), is a worthy candidate for interactive rendering using high quality global illumination. A popular and robust global illumination algorithm, it can reproduce a wide range of visual effects including indirect illumination, color bleeding, and caustics on complex diffuse, glossy and specular surfaces represented using arbitrary

geometric primitives, as was demonstrated in Figure 1.1. When these visual effects are present in an image, photon mapping tends to be cheaper than the previously discussed alternatives for images of equivalent variance. It is however important to note that the algorithm is biased. Once the photon map is generated, the final image will not reach zero variance as the number of pixel samples increases to infinity. The algorithm generates an image in two phases.

The first phase breaks the energy of the light sources into discrete packets called *photons*. Although inspired by the physics of light, this use of the word photon is somewhat misleading because they do not represent actual physical photons. The name is however highly suggestive of the initial algorithm and has become entrenched. The photons are shot into space and allowed to probabilistically reflect, refract and be absorbed by surfaces. At every intersection with the scene, the photon, their locations, energy, and direction of travel, are recorded in the *photon map*.

The actual image is generated in the second phase. The photon map is an estimate of the incident radiance at any point in the scene. To consult the photon map, a photon gather is performed. This is essentially a kNN search, using the photon map to find the k photons closest to the point of interest. The photon gather is used to terminate, at an early stage, the recursive evaluation of rendering equation.

The photon map algorithm is most comprehensively described in Jensen's own book with pointers to more recent research (Jensen, 2001). In this section, we examine more closely the two phases of the algorithm and examine the costs to determine what the limiting factor is.

2.6.1 Photon map creation

Photons are generated on the scene-defined light sources. A simple approach is to allocate the photon budget to light sources according to the total flux emitted, randomly choosing

Pre-condition: N_{PM} is the number of photons requested for the indirect photon map
  while #Photons $< N_{PM}$ **do**
    // Determin initial photon position and direction
    $p \leftarrow$ position on a light source
    $\vec{d} \leftarrow$ randomly selected direction from light source
    // Follow photon through scene until absorbed
    caustic_path $\leftarrow$ true
    repeat
      $x \leftarrow$ intersect_with_scene$(p, \vec{d})$
      if x is not a highly specular surface **then**
        caustic_path $\leftarrow$ false
        Record $(x, \vec{d})$ in the indirect map
      end if
      if caustic_path is true **then**
        Record $(x, \vec{d})$ in the caustic map
      end if
      $p \leftarrow$ compute_reflected_angle$(x, -\vec{d})$
    until photon is absorbed
  end while
Post-condition: The indirect and caustic photon maps represent a sparse representation of incident radiance throughout the scene

Algorithm 2.4: Photon map creation. Photons are traced from the light sources through the scene. The intersections are recorded in the indirect photon map. The caustic photon map records only those intersections which were not diffusely reflected from the light. The process continues until enough photons are successfully traced.

the initial direction from the light sources into the scene. Each photon is then traced through the scene (Algorithm 2.4). They are probabilistically reflected, refracted and finally absorbed at diffuse or glossy surfaces. A record, p, is made of each intersection the photon has with the scene. Only those intersection with highly specular surfaces are not stored. It is these records that form the entries in the photon map, storing the location, x_p, attenuated flux, Φ_p, and incident direction, $\vec{\omega}_p$. A second, smaller, photon map is used to further accelerate the rendering of caustic reflection. This caustic photon map records the intersections of only those photons that have not hit a diffuse surface and which are not stored in the indirect photon map. A photon map is a *sparse* representation of the incident radiance, $L_r(x, \vec{\omega})$, for all points in the scene and from all directions.

For typical scenes with indirect illumination, the number of photons shot, N_{PM}, is required to be in the hundreds of thousands or even millions (Jensen, 2001). As there will be a very large number of kNN searches, it is important that these searches be efficient. The standard acceleration data structure is the *kd*-tree (Bentley, 1975). The choice of data structure will be studied in Chapter 4.

For static scenes the photon map can be generated as a single preprocess and reused for multiple viewpoints. Alternatively, time dependent photon mapping adds time as dimension for the kNN gathers, allowing for animations without temporal aliasing (Cammarano and Jensen, 2002). Jensen described a simple three-pass technique that uses an additional importance map, generated by shooting *importans* (Jensen, 1996b). Importans are an analogue of photons; they are shot from the camera into the scene and their distribution is then used to determine when the standard photons should be recorded in the photon map. Additionally, importance sampling can be used to guide the initial directions in which photons are shot using a projection map from the light sources (Jensen and Christensen, 1995).

2.6.2 Image generation

After the photon maps are created, the image is rendered using the pixel-driven algorithm (Algorithm 2.1). The exitant radiance is computed in four steps, outlined in Algorithm 2.5. The direct and emitted illuminations are computed in the same manner as they were for path tracing. The indirect illumination however is computed twice, once for general indirect illumination and once for caustics, using the two separate photon maps.

To compute the caustic illumination using the photon map, a kNN search within the caustic photon map is conducted in a small neighborhood around the point x. Reasonable values for k are 100 or more (Jensen, 2001). As these photons are taken as a sparse

Pre-condition: x is the location under consideration
Pre-condition: $\vec{\omega}_o$ is the direction towards the viewer
// Compute the locally emitted radiance in direction $\vec{\omega}_o$
result $\leftarrow$ emitted_illumination$(x, \vec{\omega}_o)$
// Compute the direct illumination due to the pre-defined light sources
result $\leftarrow$ result + direct_illumination$(x, \vec{\omega}_o)$
// Compute the indirect illumination using the caustic photon map
result $\leftarrow$ result + photon_gather(caustic_map$, x, \vec{\omega}_o)$
// Compute the indirect illumination using the indirect photon map
result $\leftarrow$ result + compute_indirect$(x, \vec{\omega}_o)$
Post-condition: result = the total reflected radiance from x towards $\vec{\omega}_o$

Algorithm 2.5: Generation of an image using the photon map. Direct illumination and self emitted radiance are computed as they were for path tracing. The caustic photon map is queried using a single photon gather. The rest of the indirect illumination is computed using one of the two visualization techniques discussed in this section.

sampling of the incident radiance at the position x, Jensen showed that the rendering equation can be adapted to compute the reflected radiance at x in any direction, Equation 2.6, without further recursion of the rendering equation. (The term A represents the area of the circle defined by the radius required to find all k photons.) This process is referred to as a *photon gather* and is illustrated in Figure 2.7. It is worth noting that this process is similar to Monte Carlo integration, except that the sample directions have been chosen previously, by the selection of photon paths.

$$L_r(x, \vec{\omega}_o) = \sum_{p=1}^{k} f_r(x, \vec{\omega}_p, \vec{\omega}_o) \frac{\triangle \Phi_p(x, \vec{\omega}_p)}{\triangle A} \tag{2.6}$$

The caustic illumination is computed for every eye-ray using a single photon gather at x in the caustic photon map. There are two common techniques for computing the general indirect illumination using photon gathers. The first, direct visualization, uses a second final gather at x in the general indirect photon map. There is no need to recurse through the rendering equation at all, leading to a very efficient algorithm. Unfortunately, direct visualization exhibits strong visual artifacts unless a prohibitively large number of photons are used (Jensen, 2001).

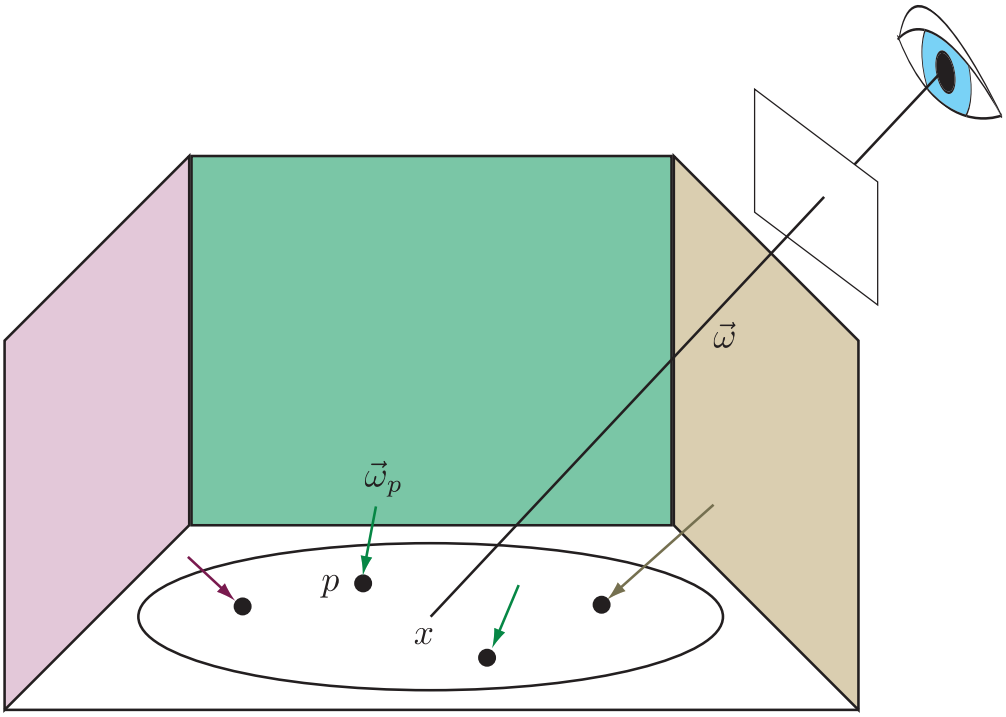


Figure 2.7: Computing reflected radiance using a photon gather. A single photon gather performed at x to estimate the reflected indirect illumination. The k nearest photons in the photon map are located, and are interpreted as a sparse representation of the incident radiance, allowing for the evaluation of the rendering equation without further recursion.

2.6.2.1 Final gather visualization

The final gather visualization, shown in Figure 2.8, estimates the reflected radiance at x by evaluating the rendering equation using a Monte Carlo integration. Instead of a single photon gather, the hemisphere surrounding x is sampled and N rays, $\vec{\omega}_i$, are cast out into the scene as described in Section 2.3.4. At each intersection point, y_i, a standard photon gather is performed. These results are combined using Equation 2.5 to compute the indirect diffuse illumination. Reasonable values for N are 100 or more (Jensen, 2001). Because the artifacts of direct visualization are stark, I use the higher-quality final-gather visualization in this dissertation. The simple modified Cornell box image in Figure 1.2 required only $N = 50$ to achieve acceptable visual quality. The more complicated image of the Sponza scene in Figure 1.3, however, required $N = 200$.

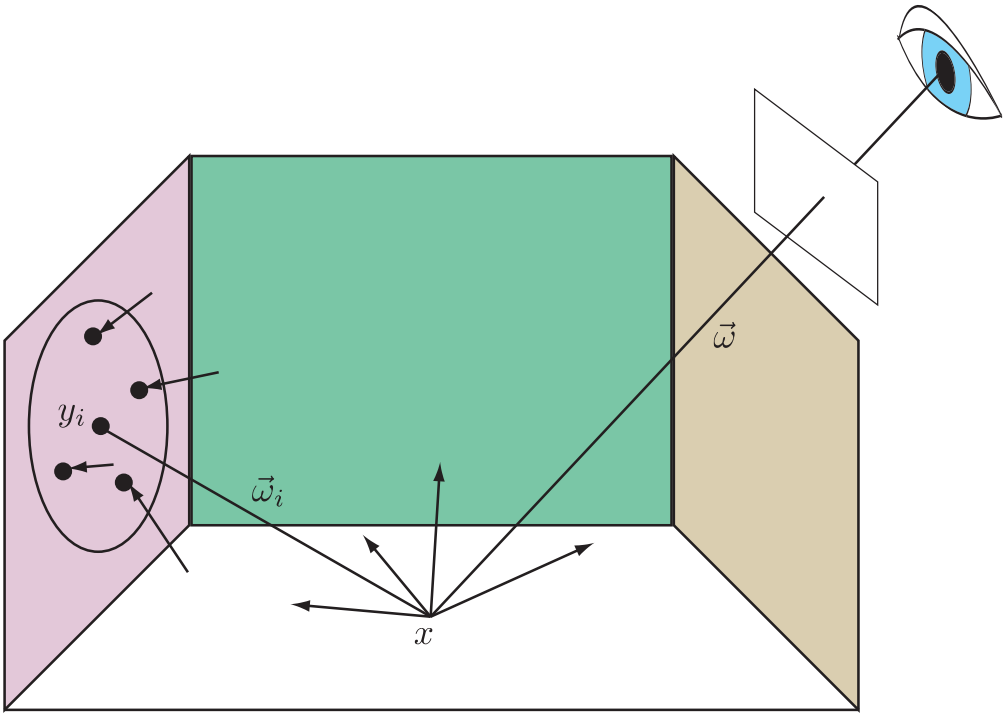


Figure 2.8: Final gather visualization of the photon map. A higher quality estimate of incident radiance is obtained by using Monte Carlo integration to perform a final gather. A photon gather is performed at the end of each secondary ray, $\vec{\omega}_i$.

2.6.3 Computational Requirements

The computational requirements of generating interactive images using the final-gather visualization algorithm are significant. The costs are analyzed in Section 7.3 and summarized in Section 7.3.5. The most significant portion of the computation is the indirect photon gathers. The photon gathers require a kNN search (traversing the photon map, comparing normals and verifying that the photons found are within the allowed radius), evaluating the BRDF for each photon chosen, and finally the accumulation in the frame buffer of the results.

The modified Cornell box image in Figure 1.2 requires 37 GFLOPs to render, or 1.1 TFLOPS at 30 frames per second. The Sponza scene requires 209 GFLOPs per image, or 6.3 TFLOPS at 30 frames per second. This is a lot of computation, far more than

is expected to be available on CPUs in the near future. In Section 2.2 it was estimated that fine-grained graphics applications can easily expect to support 500 GFLOPS of computation by 2010. The workload of the Sponza atrium would need to be split among 13 chip replications to achieve this rate. This number of chips, each with dedicated high-speed memory, is aggressive to include within a single host workstation.

2.6.4 Bandwidth Requirements

The bandwidth requirements of photon mapping are measured as either the total number of bytes that must be transfered from off-chip memory to render an image, or as a rate using the number of bytes per second. The data structures holding the photon map, geometry, surface properties and other required items are too large to fit in embedded memory for the next several years.

The modified Cornell box image in Figure 1.2 requires 50 GB to render, or 1.4 TB/s at 30 frames per second. The Sponza scene requires 367 GB per image, or 11 TB/s at 30 frames per second. These bandwidth rates are very high, with the bulk of the cost due to the kNN searches for the indirect photon gathers. Using the estimate of near future memory bandwidth in Section 2.2, the 11 TB/s would require that the workload be divided among 123 chips each with a replication of the same memory. This is not feasible to house in a single host workstation, and would require either a large cluster or a large specially built chassis.

2.7 Discussion

Photon mapping is a viable candidate for implementation as an interactive global illumination architecture. Although the computational requirements are significant, it is the memory bandwidth that precludes a straight forward implementation from being small enough to install in a single host workstation. It is the photon gathers used by the final

gather visualization to compute indirect illumination that are so expensive. Therefore, this dissertation focuses on the bandwidth requirements of the indirect photon gathers, introducing three techniques that reduce their cost while not significantly increasing the computation required. The results are then utilized by a feasible hardware architecture to support interactive rendering.

CHAPTER 3

PHOTON GATHER REORDERING

Rendering an image with the naive photon mapping algorithm, as described in Section 2.6, generated a tremendous amount of memory traffic. The Sponza scene, for example, required 367 GB for a single image. An interactive system generating thirty frames per second would therefore require at least 11 TB/s of memory bandwidth. Only a small portion of the required bandwidth can be attributed to solving the visibility problem, with ray casting, or performing the local evaluation of surface reflectance, in the form of texture lookups. The large value is due almost entirely to the photon gathers.

Chapters 5 and 6 will describe two techniques to reduce the number of photon gathers that must be performed, and thus indirectly the bandwidth requirements. Together with Chapter 4, this chapter focuses on reducing the bandwidth costs of performing a *particular and fixed* set of photon gathers without any modifications to the generated image. This is achieved by changing the order of operations, allowing the memory cache to become more effective.

After an exploration of the origin of the high bandwidth costs incurred by the naive photon mapping algorithm, the potential reordering schemes are broken into two categories. Four specific techniques are presented and analyzed. Of these, the reordering based on the Hilbert curve reduces off-chip bandwidth by up to four orders of magnitude (to 15 MB for the Cornell scene), but requires 1 GB of intermediate high-speed storage. Alternative techniques are presented that still achieve one order of magnitude improvement for the scenes described in Chapter 1 but only use 1 MB of intermediate storage.

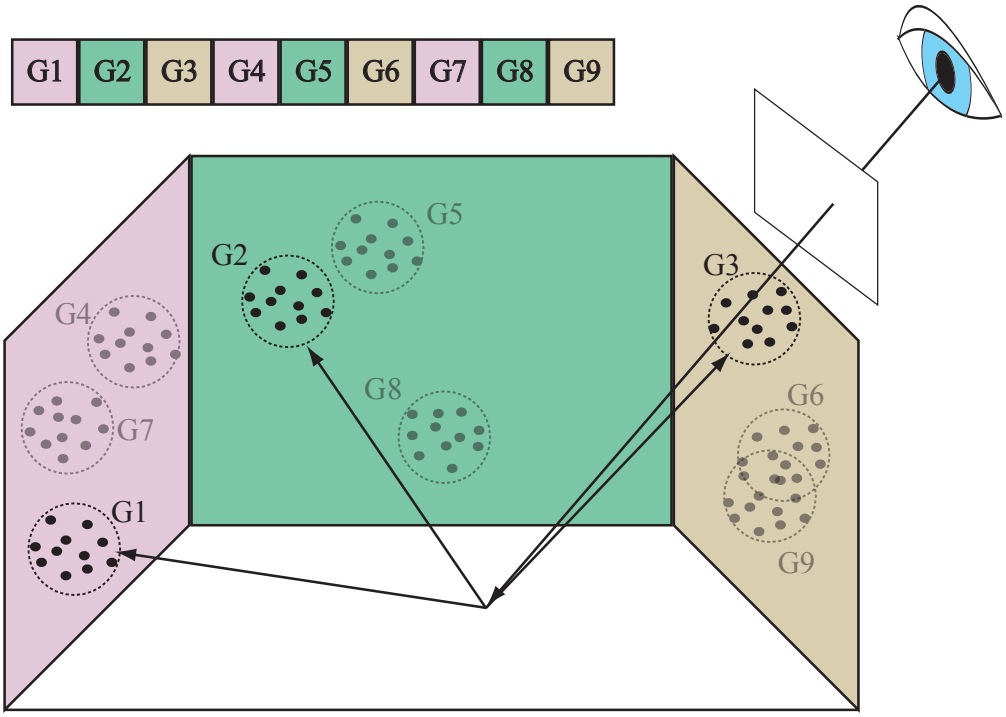


Figure 3.1: In this example scene, each eye ray spawns three final gather rays. The dashed circles show the locations of the photon gathers. The gather locations are numbered in the order they might be generated by the naive algorithm. Although many of the gathers are very close to each other in 3D space, and hence in the photon map as well, those gathers are not performed at similar times. A cache is therefore unable to reduce the bandwidth requirements.

The algorithms presented in this chapter form the basis of the architecture described in Chapter 7.

3.1 Origin of the high costs

The naive implementation of photon mapping, as described in Chapter 2, is illustrated in Figure 3.1. The pixels of the image are evaluated sequentially, in row-major order. A pixel may be sampled multiple times, for higher image quality with less geometric aliasing, with each sample establishing an eye ray into the scene. This eye ray is cast through the scene to find the first intersection, x. If x is on a highly specular object, the

ray is reflected and cast again, until a non-specular object is found. Using the Monte-Carlo integration technique, N_{FG} final gather rays, originating at x, are sampled from the visible hemisphere uniformly. This process is then repeated for every other pixel. Notice that the random sampling of the hemisphere is performed pixel by pixel without regard to prior choices. These final gather rays are then traced through the scene to their own intersection points, y_i, where a photon gather is performed in lieu of a recursive application of the rendering equation.

Memory caches reduce memory traffic by exploiting locality in the memory request stream. To be effective they require both spatial and temporal locality (Patterson and Hennessy, 1996). Using a reasonable data structure (see Chapter 4 for a discussion of the available choices) each individual photon gather need examine only a small portion of the memory holding the photon map. Notice, however, that in the illustration there are many photon gathers on the right wall that need to be performed. They are so close that they will require access to the same portion of the photon map, indeed many of the same photons will be found for each photon gather. The fact that so many photon gathers require the same piece of memory implies that there is strong spatial locality present in the workload and that a cache has the potential to greatly reduce the bandwidth requirements drastically by preventing many photon gathers from requiring any new information from memory.

Unfortunately, these photon gathers that are so close to each other were in fact generated by different pixels and are evaluated at different times. There is therefore very little temporal locality and the cache is unable to provide any substantial reduction in bandwidth requirements. The key insight in this chapter is that the photon gathers are completely independent of each other, despite the sequential order implied by the naive algorithm. By *reordering* the photon gathers, temporal locality can be improved and the potential coherence will become exposed, reducing memory bandwidth dramatically without changing the final image.

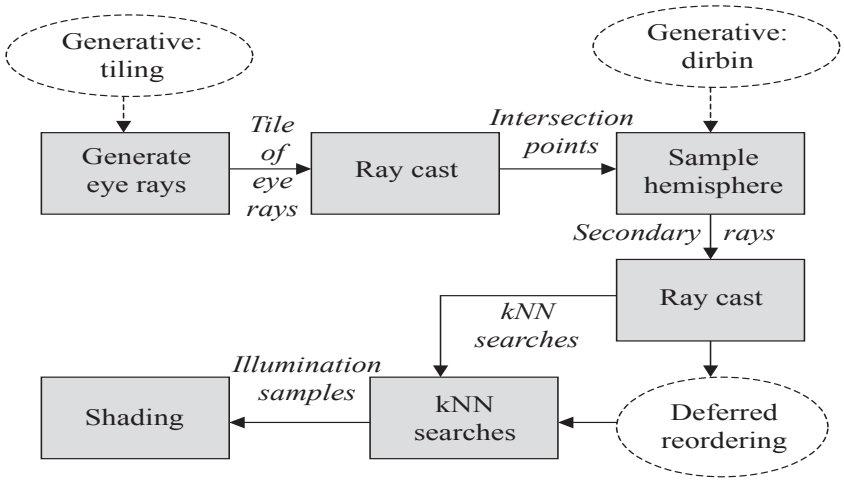


Figure 3.2: After the eye rays for each tile are generated, they are cast into the scene. At each intersection point the hemisphere of directions is sampled, possibly using generative reordering. The final gather (secondary) rays are then intersected with the scene, determining where the photon gathers will take place. This list of photon gathers may be reordered a second time, deferred reordering, before the gathers are processed. The final result for each query is a contribution to an individual pixel. The results are accumulated in the framebuffer after shading.

3.2 Reordering algorithms

The techniques presented in this chapter can be characterized as taking two different approaches for changing the order in which photon gathers are processed. The first, *generative reordering*, modifies the order in which the final gather rays, and hence photon gathers, are generated. *Deferred reordering*, on the other hand, generates a list of photon gathers, Y, and then sorts the list before performing the searches. Figure 3.2 provides a logical overview of how to apply these two approaches to a system which generates images using the photon map.

To compare the reordering algorithms described in this chapter, a publicly available renderer, `pbrt` (Pharr and Humphreys, 2004), was modified by adding a software cache simulator. The renderer notes all accesses to the photon map and provides the stream of memory references to the cache simulator for analysis. The simulator models a fully-associative cache that implements the Least Recently Used (LRU) eviction policy. Fully

associative caches are costly to implement in hardware because of the large number of comparators required. Caches are instead usually implemented with the less optimal set-associative eviction policy. However, it has been shown that, in the absence of a highly regular access pattern, reducing the associativity raises the number of block evictions and fetches by a constant factor (Hill and Smith, 1989). The reduction from a fully-associative cache to an 8-way set-associative cache increases the number of fetches by approximately 39%.

The experiments in this chapter were conducted on the test scenes from Section 1.4, the Cornell box and the Sponza model. At first, the cache is configured as 128 KB in size with 128 byte cache lines. Section 3.5 explores the parameters we chose for the cache. The *kd*-tree data structure for the photon map is used throughout this chapter and the images generated are 512×512 with a single sample per pixel. The number of final gather rays per eye-ray, N_{FG}, varies as noted for each experiment.

3.3 Generative reordering

The naive ordering can be thought of as the base case for reordering. The first set of techniques examine and modify the order in which the final gather ray directions are generated. These rays are determined before ray casting has been performed. As a consequence, the final locations of the photon gathers, denoted y_i in Section 2.6.2, are not yet known. Even without this knowledge, significant increases in temporal locality are possible.

In many scenes the eye rays from neighboring pixels will intersect the scene at points x in close proximity to each other (Wald et al., 2001). The nature of this coherence is that the origins of the final gather rays cast during the Monte Carlo integration will tend to be similar for pixels near to each other in the image.

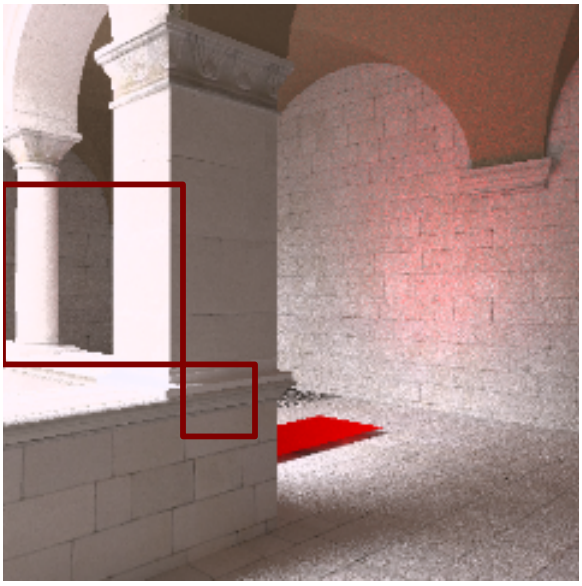


Figure 3.3: A small image tile has the property that its pixels will tend to project to a small number of objects that are close to each other. The origins for the final gather rays will therefore also be closer together than those of large tiles.

3.3.1 Tiled reordering

It would seem that this coherence could be exploited by breaking the screen into tiles. The naive photon map algorithm would then be applied to each tile independently. The pixels of each individual tile are processed in scanline order. The list of associated photon gather sites, $Y^{<a,b>}$, is an enumeration of the photon gather locations for the tile $<a, b>$. A similar technique is commonly used in graphics rasterization hardware to improve texture memory locality (McCormack et al., 1998). As demonstrated in Figure 3.3, image tiles that are of moderate size generally project onto just a small portion of the scene, causing all the gather rays in $Y^{<a,b>}$ to have similar origins. This will be true except in scenes that have significant depth discontinuities, such as a leafy forest.

The problem with the tiled approach by itself is that while the origins, x, of the rays used by the Monte Carlo integration are similar, the directions, $\vec{\omega_i}$, remain spread across the hemisphere. In scenes which consist primarily of relatively open rooms, with or without complex objects or wall surface geometry, the resulting search locations y_i remain

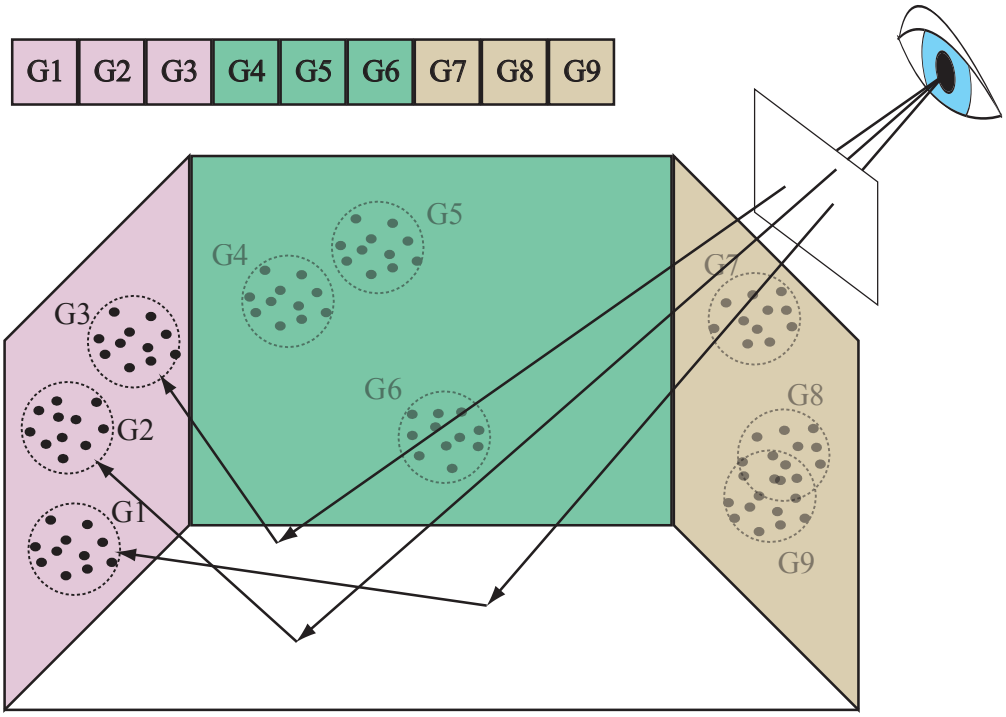


Figure 3.4: Although the division of the image into tiles brings coherence to the origins of the final gather rays, the directions of the rays remains random. To expose the coherence in the location of photon gathers, the tile is processed multiple times. During each pass only those final gather rays with similar directions are generated.

too scattered throughout the scene to improve cache efficiency. This lack of improvement can be seen on the top line of the graphs in Figure 3.5. As the tile sizes varies, there is not a noticeable change in the bandwidth requirements of the naive tiled algorithm.

3.3.2 Tiled direction-binning reordering

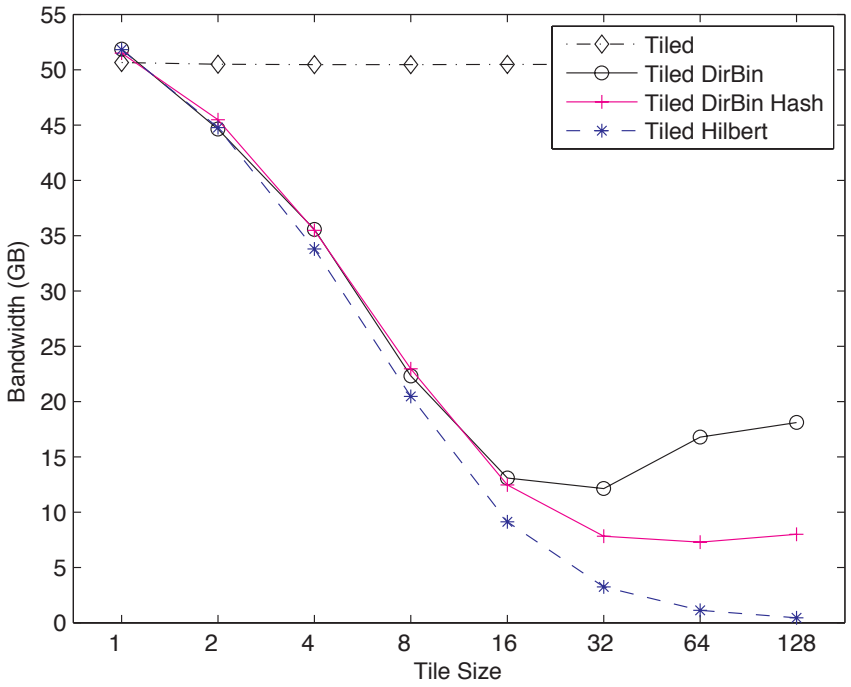
The tiled algorithm can be improved by explicitly grouping the final gather rays by direction, $\vec{\omega_i}$, in addition to the implicit grouping by origin, x. The resulting rays will share both similar origins *and* directions (see Figure 3.4). They will therefore tend to highly coherent and intersect the scene at points y_i near each other (Wald et al., 2001).

This generative ordering can be implemented by performing multiple passes over the tile, after the initial ray casting has located the eye-ray intersections x. Each pass will

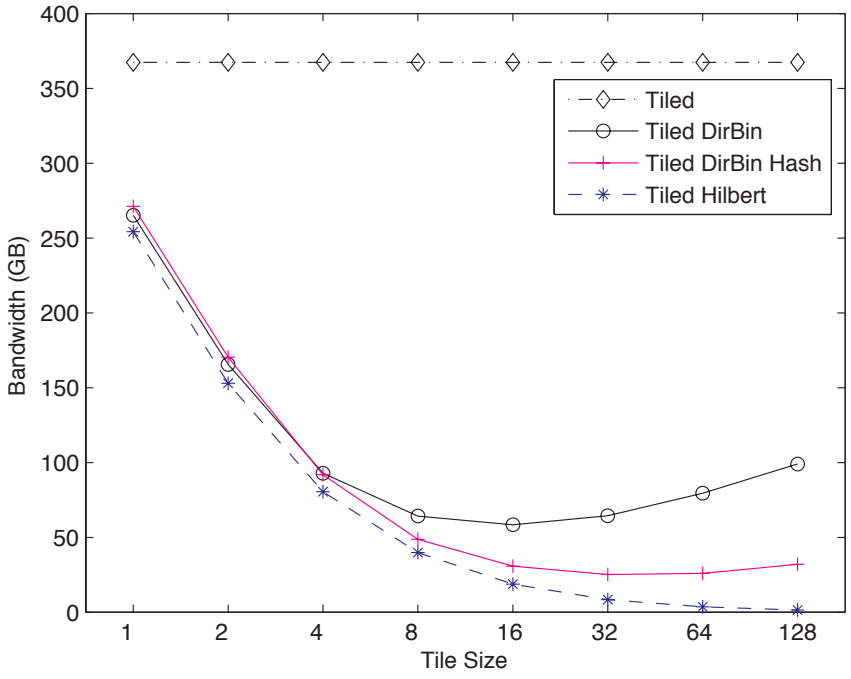
only generate those rays, $\vec{\omega_i}$, that fall within a specified portion of the hemisphere. This is in contrast to the naive algorithm which generates all the rays for a single eye ray before continuing. The hemisphere is divided into bins of equal solid angle. The number of bins is a system parameter; the smaller that each direction bin is, the more coherent the rays generated will be. This benefit is offset by the repeated work incurred during each pass. The correct tradeoff is found experimentally.

The tiled direction-binned reordering algorithm requires less than a third of the naive algorithm's bandwidth for all tile sizes larger than 4×4 (Figure 3.5). The modified Cornell scene is reduced from 50GB per image to 13 GB with a tile size of 16×16, while the Sponza scene is reduced from 367 GB to 58 GB. An interesting feature of the graphs in Figure 3.5 is the knee at tiles of size 16×16. As mentioned above, larger tiles cover multiple surfaces, reducing the coherence of the final gather ray origins, x. The result is that the final gather rays for the tile will spread further out in the scene when the tiles get large. For the test scenes in Chapter 1, this knee occurred at 16×16 and it was determined to use 16×16 bins for all results presented in this chapter.

When generating an image using the photon map, highly specular surfaces such as mirrors and glass objects are treated specially. Instead of using a final gather immediately, the eye ray is reflected and/or refracted as a single ray until it reaches a diffuse or glossy surface. If this occurs to some of the pixels in a tile, then the origins of the final gather rays will be very far apart, eliminating the coherence of photon gather locations. The implementation used in this dissertation tackles this problem by creating a list of those eye rays affected, and delaying all action on them until the final gathers for all other pixels in the tile are handled.



(a) Modified Cornell box, $N_{FG} = 33$



(b) The Sponza atrium, $N_{FG} = 200$

Figure 3.5: The tiled Hilbert curve reordering results in the lowest bandwidth for each tile size. Only tiles of moderate size are practical due to internal storage constraints. 16×16 and 32×32 are both feasible and perform well for the cost effective reorderings of tiled direction binned, both with and without hash reordering.

3.4 Deferred reordering

The generative reordering techniques described in the previous section enumerate a list Y, $Y^{<a,b>}$ if tiling is being used, of photon gather locations. The second set of techniques for improving the locality of the photon gathers consists of reordering the elements $y_i \in Y$ to form Y' before performing the photon gathers. There are several possible deferred reorderings, and they may be matched with any generative reordering; any combination is possible.

This decoupling of the photon-gather-location generation from the actual gather process does however introduce computational and storage overhead. For each photon gather the following must be stored: the search location, the direction from which this point is viewed, a pointer to the local material properties, the destination pixel, and an RGB pixel contribution weight. The system described in this dissertation requires 44 bytes for each deferred photon gather.

If all the photon gathers for a 512×512 image with 100 gathers per pixel, $N_{FG} = 100$, are deferred for reordering, then 1 GB of intermediate storage will be required. If the screen is divided into 16×16 tiles, then 1 MB of intermediate storage will be required. Some deferred techniques do not require the full list Y to be enumerated at any one moment, but instead operate in a streaming fashion over a window of photon gather requests.

3.4.1 Hilbert curve reordering

The tiled direction-binning generative reordering technique is efficient and effective at reducing bandwidth. A deferred technique that waits for the entire list of photon gather requests, T, and performs an near-optimal sorting on that list would provide a measure for just how much reduction in bandwidth is possible.

The Hilbert curve, shown in both two and three dimensions in Figure 3.6, is a space-

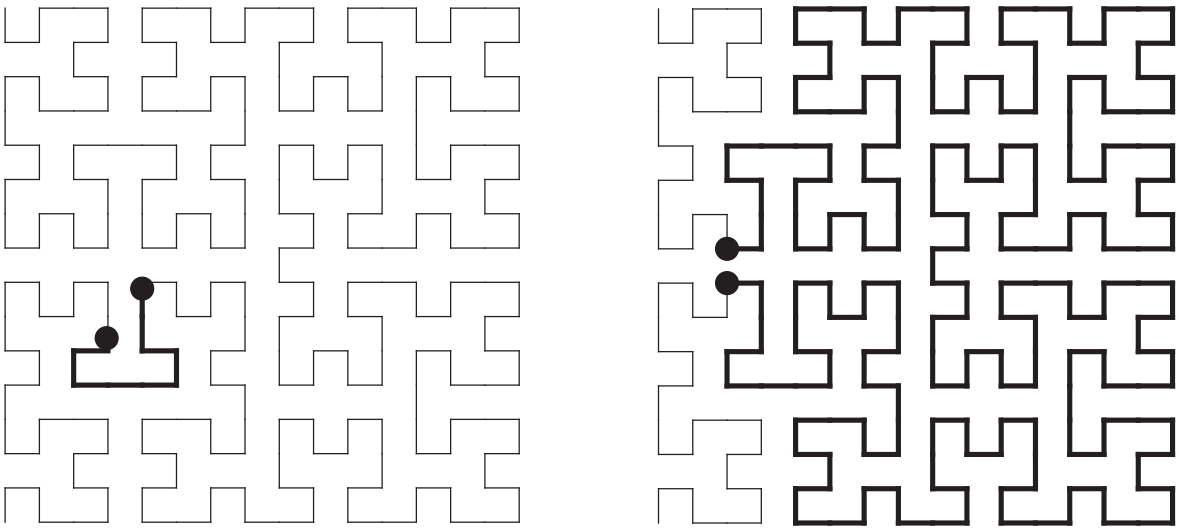


Figure 3.6: The Hilbert curve is a space filling curve with the property that points close to each other in the multidimensional space will tend to be close to each other in the linear order imposed by the Hilbert Curve. Although this property does not always hold, the Hilbert curve is optimal in 2D and believed to be optimal in higher dimensions.

filling fractal. It has the useful property that if two points are close to each other in the multidimensional space, they will tend to be close to each other in the linear order imposed by the Hilbert curve. It can therefore be used to produce a linear mapping of a multidimensional space (Faloutsos and Roseman, 1989; Moon et al., 1996).

As a deferred ordering, the entire set of photon gathers, Y, can be organized along a three-dimensional Hilbert curve, $Y' = \text{Hilbert}(Y)$. This is shown symbolically in Figure 3.7. All the photon gathers in Y are considered to be inserted into an uniform grid so fine that no cell contains more than a single gather. This grid is then traversed in the Hilbert curve order to produce the new order, Y'. As the traversal moves through the grid, the photon gathers are processed in the photon gather until. The system used in these experiments to reorder the gathers was developed to minimize memory bandwidth in the presence of a small cache and is derived from code provided by Liu and Snoeyink (Liu and Snoeyink, 2007).

The Hilbert reordering algorithm, applied over all of the searches in Y, is able to

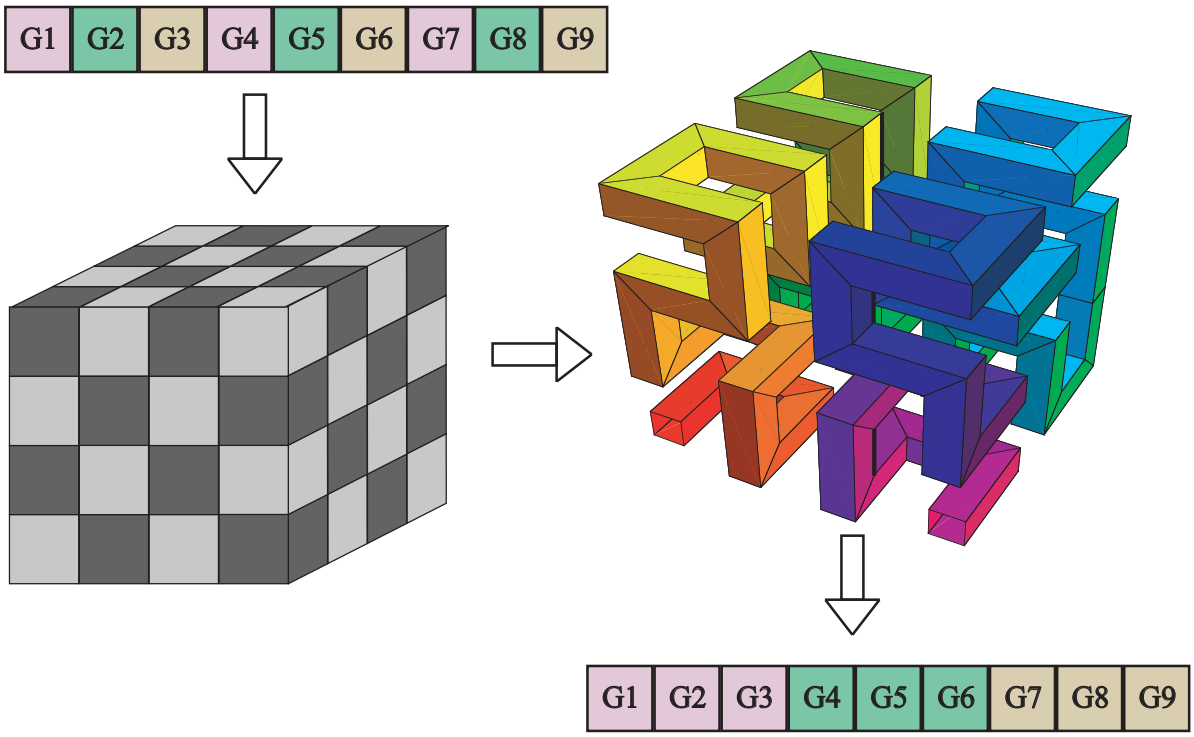


Figure 3.7: The Hilbert curve is used to reorder photon gathers by inserting them into a virtual grid that is then stepped through in the order indicated by the Hilbert curve.

reduce the bandwidth of the Cornell scene from 50 GB to 15 MB, an improvement of four orders of magnitude. This can be considered a nearly optimal result; the average photon is transferred from memory less than twice. This shows that reordering has the potential to dramatically reduce the cost of photon mapping. However, applying this technique to the entire image requires that we store the entire list Y before it is sorted. This does not seem feasible because it would require 1 GB of memory for a system using 16×16 tiles.

3.4.2 Tiled Hilbert reordering

Generating the Hilbert curve ordering for the entire image at once required a significant amount of processing and storage. To reduce this overhead, the reordering can be done on individual screen tiles generated by the screen tiled algorithm, $Y'^{<a,b>} = \text{Hilbert}(Y^{<a,b>})$.

Although this reduces both the computational and storage overhead, it also reduces the effectiveness of the reordering. The bandwidth increases as the number of pixels in the tile decreases, as shown by the bottom curves in Figure 3.5. As the tile size increases, the Hilbert reordering can exploit more coherence in the search locations. For reasonable tile sizes of 8×8 to 32×32, the bandwidth ranges from 20 GB to 3 GB for the Cornell box, and 40 GB to 8 GB for the Sponza model. Notice that unlike the tiled direction-bining algorithm, there is no knee in the graph. This is because rather than attempting to make the final gather rays more coherent, the actual locations of the photon gathers are directly sorted.

As the work by Liu matures, it may prove possible to perform a Hilbert reordering on a tile basis directly in hardware. The additional reduction of bandwidth for the photon gathers would have to exceed the cost. Tiles would have to be large enough so that the difference between the generative techniques and the Hilbert curve technique is large, perhaps 32×32 or 64×64, but small enough to not require additional off chip bandwidth. One possibility is that the Hilbert reordering algorithm could be adapted to the stream processing model.

3.4.3 Hashed reordering

Several authors (Indyk et al., 1997; Gionis et al., 1999) have explored hashing algorithms for kNN searches, the core algorithm underneath a photon gather. This section introduces a reordering algorithm that uses hashing functions with a different approach, one that has low computation costs and manageable memory use. This algorithm is similar to the ray-queue hashing of the GI-Cube (Dachille and Kaufman, 2000), introduced in Section 2.5.

The GI-Cube is a volume rendering architecture that uses ray tracing to create global illumination effects (Dachille and Kaufman, 2000). Rather than tracing each ray indi-

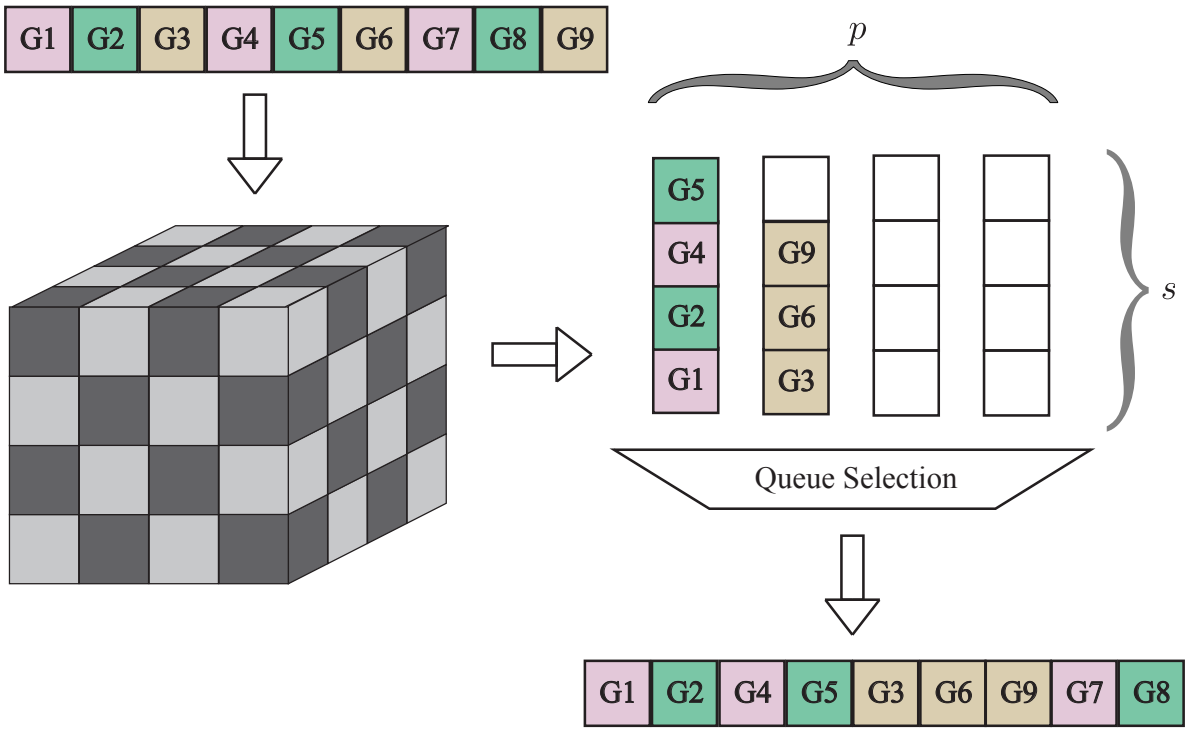


Figure 3.8: The hashed reordering algorithm. As each photon gather request is generated, it is assigned to a queue based on a three-dimensional hashing function. A single queue is serviced until it is exhausted, when the then fullest queue is selected. The reordering is restricted to a window of $p \times s$ photon gather requests.

vidually throughout the entire volume, potentially requiring a great deal of repetitive memory accesses, the GI-Cube traces rays together as long as they remain in the same portion of the scene. The scene is divided into blocks of volumetric data small enough to fit entirely in the cache. Separate queues are established with multiple blocks assigned to each queue. The mapping of blocks to queues is computed by a three-dimensional hashing function.

Only those rays within a single queue are processed at a time. If during ray tracing any ray leaves its current block, the ray is moved to the queue of the block it is entering. The rays in a queue are processed until the queue is empty, when the queue with the most entries is selected as the new active queue. This process achieves a tremendous reduction in memory bandwidth.

This algorithm is adapted to photon gather reordering as illustrated in Figure 3.8. The entire scene is divided into a regular grid, according to system-wide parameters that set the number of intervals along each dimension. Each gather location in Y is compared to this grid to determine which three dimensional block it corresponds to. A three-dimensional hashing function is applied to the block each photon gather request falls in order to assign it to on of the p queues. Each queue is able to hold only s pending photon gather requests, restricting the reordering to a window of $p \times s$ gather requests.

Photon gather requests are extracted from a single queue at a time and passed to the photon gatherer. When the queue becomes empty the then fullest queue is selected and processed. In this fashion the generated list of photon gather requests Y is reordered, $Y' = \text{Hash}(Y)$.

There are three user controllable parameters that must be adjusted: the number of hash buckets, the size of these buckets and the size of the uniform grid. Optimizing these variables is difficult, and is a drawback of the method. In order to reduce the number of structural collisions, it is important to choose a prime number of buckets.

A notable feature of the hashed reordering is that there is no need to wait until the complete list Y has been enumerated. The incoming photon gather requests can instead be processed in a streaming fashion. Because the queues only hold $p \times s$ gather requests, the window of reordering is finite and usually far smaller than $|Y|$. Unfortunately, by operating only on a small window, this algorithm is unable to perform large-scale reordering by itself. It is therefore unsuitable when applied directly to the naive generative ordering.

3.4.4 Tiled direction-binning Hashed reordering

The more coherent the initial list of search locations, the better the hash reordering described in the previous section will perform. The hashed reordering algorithm can

be combined with tiled direction-binning. Each generated list of gather locations for a single tile, $Y^{<i,j>}$, is already highly coherent. With hashed reordering applied to each tile $<M, N>$, $Y' = \text{Hash}(< Y^{<1,1>} \cdots Y^{<M,N>} >)$, the bandwidth requirement of the Cornell box scene is slightly reduced from 13 GB to 12 GB, for a practical tile size of 16×16. The additional reduction is small, as the tiled direction-binning reordering is able to extract most of the available coherence. (The tiled Hilbert reordering required 9 GB.) The Sponza scene experiences a more substantial reduction from 58 GB to 31 GB. This is approximately half the possible difference between tiled direction-binning and the Hilbert curve reordering.

The experiments in this dissertation were run with 17 hash buckets of 128 elements. Increasing the number of buckets or the size of the buckets reduces the number of collisions only marginally. For the Cornell scene we used a uniform grid with 5 intervals in all three dimensions, while in the Sponza scene we used 8.

3.5 Cache parameters

The experiments in this dissertation use the previously described cache simulator, attached to the `pbrt` rendering system. Like Ma and McCool (Ma and McCool, 2002), off-chip memory is assumed to be relatively slow and to be accessed in chunks the size of cache lines. Standard DRAM designs achieve peak bandwidth when the accesses are coherent, long, and aligned to page boundaries. This means that reading individual photons from scattered locations in memory is highly inefficient. The naive bandwidth estimates of 50 GB and 367 GB are generated with 20 bytes reads of single photons scattered across memory. Assuming 20 bytes per photon and standard 128 byte cache lines (Intel, 2003), this means that about six photons fit in a cache line (with some extra room for data structure bookkeeping). Efficient algorithms should make use of several, if not all, of the six photons stored in a cache line. Several of the data structures that are

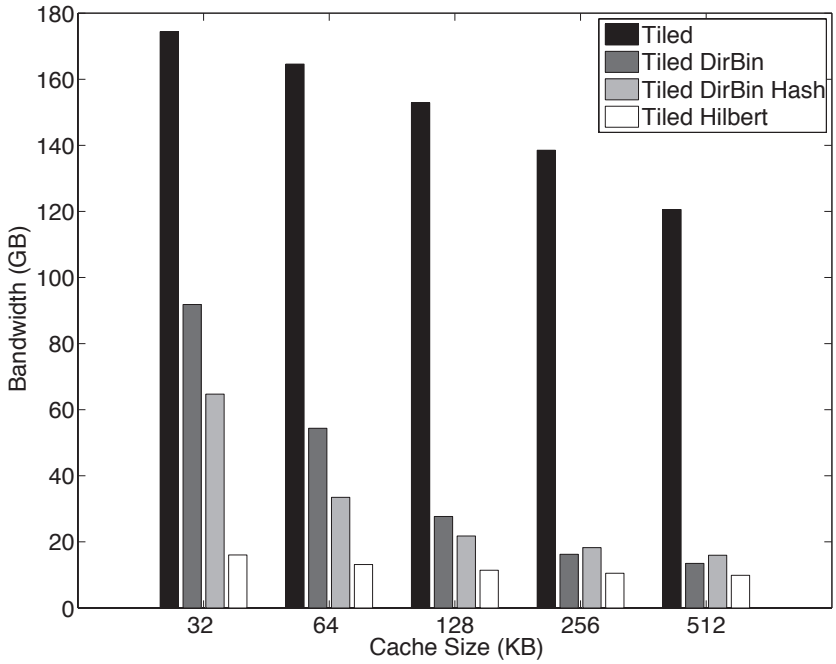


Figure 3.9: The practical reordering algorithms described in this chapter experience diminishing returns on caches larger than 128 KB. The simple tiled algorithm will continue to benefit as the cache size is increased. There is little difference for the tiled Hilbert algorithm, as its working set is less than 32 KB. (Cornell box scene, $N_{FG}=100$, *kd*-tree, 128B cache lines and 16×16 pixel tiles.)

discussed in Chapter 4 are explicitly organized in blocks that are the size of the cache lines in order to maximize the utility of each read. In this chapter cache lines of 128 bytes are used.

For the purpose of comparing the reordering algorithms in this section, we fixed the cache to be 128 KB in size with 128 bytes cache lines. Figure 3.9 formed the basis for our choice of cache size. For the Cornell box scene, increasing cache size beyond 128 KB had no effect when a good ordering algorithm was used. The Sponza scene has a larger working set and is able to make good use of a 256 KB cache, although larger sizes provide little to no benefit.

Reordering algorithm	Bandwidth	
	Cornell	Sponza
Tiled naive	50 GB	357 GB
Tiled direction-binning	13 GB	58 GB
Tiled direction-binning with hashed	12 GB	31 GB
Tiled Hilbert	9 GB	19 GB

Table 3.1: The bandwidth requirements to render a single frame of the modified Cornell box, Figure 1.2 and the Sponza atrium, Figure 1.3, for the various reordering algorithms. The combination of tiled direction-binning generative reordering and hashed deferred reordering is able to reduce the bandwidth requirements by over an order of magnitude; however the requirements for the Sponza Atrium are still more than the 24 GB/image an 8-chip system would provide. (*128 KB cache, 128 byte cache lines,* 16×16 *tiles, kd-tree.*)

3.6 Conclusion

Photon gather reordering reduces the memory bandwidth requirements of final gather rendering using the photon map. Although the bandwidth can be reduced drastically by fully sorting all gather requests for the entire image, this requires too much intermediate computation and storage. The combination of tiled direction-binning generative reordering and the hashed deferred reordering is practical, easy to implement and highly effective, achieving over an order of magnitude reduction in bandwidth requirements.

There are alternative reordering schemes to be investigated. For example, the ray tracing process itself generally uses an acceleration data structure that divides space. There are ways that this information could be harvested and reused to perform the deferred reorder cheaply. The hope is to find a reordering able to more closely approximate the superior results of the tiled Hilbert while remaining as cheap as tiled direction-binning. Additional work extending that of Liu and Snoeyink could allow for the direct application of the Hilbert curve algorithm itself, if the computational and added bandwidth costs can be kept reasonable. The results in this chapter assumed that the conventional data structure, the *kd*-tree was used. In the next chapter the potential benefit of alternative data structures is examined.

CHAPTER 4

DATA STRUCTURES

The memory bandwidth requirements of photon gathering are clearly dependent on the specific data structure that the photon map has been stored in. Recall that each photon gather involves a kNN search which requires finding the k photons closest to the point where the reflected radiance is being calculated. In the extremely inefficient case of a simple unordered list, each photon gather would require consulting every single photon in the photon map. A single gather would therefore require $O(N_P)$ memory accesses, where N_P is the number of photons in the map.

Spatial subdivision data structures can be used to reduce the number of photons that must be examined and therefore the total memory accesses required to perform any one gather. For example, the *kd*-tree data structure, discussed in Section 4.2, was originally suggested by Jensen because only $O(k+\ln(N_P))$ memory accesses are generated (Jensen, 1996a). With the number of photons near or exceeding one million for high quality indirect illumination, this can be a tremendous savings.

The traditional analysis of algorithms, and their data structures, uses the Random Access Memory (RAM) model of computation and concentrates on the arithmetical computation effort (Cormen et al., 2001). Under the RAM model, each basic arithmetic operation costs exactly one unit of time, as do *all* memory accesses. In particular, the RAM model does not correctly model those computer memory systems which have one or more levels of cache. In such hierarchical memory systems, the cost of a single memory access depends on the exact contents of the cache when a request is serviced, and can vary by several orders of magnitude.

Algorithms and their data structures both can be modified to be more efficient in the presence of a hierarchical memory system. In Chapter 3 the photon gathers were reordered to promote cache efficiency. The strong spatial clustering of photon gathers discussed in that chapter will favor data structures that store nearby photons close to each other. In this chapter, four possible data structures for the photon map are considered, the traditional *kd*-tree, *kd*B-tree, uniform grid, and Block Hashing. They have been shown to have a direct impact on memory requirements (Christensen and Batali, 2004; Jensen, 2001; Ma and McCool, 2002; Wald et al., 2004).

The experiments in this chapter were performed using a cache simulator, described in Chapter 3, attached to a modified open-source renderer (Jarosz, 2004). The underlying data structure code was modified to implement the different structures studied while analyzing the stream of memory references generated. The results are reproduced in Figure 4.1 and Table 4.1.

4.1 Uniform grid

Perhaps the simplest of all spatial decomposition data structures, the uniform grid is easy to implement. The spatial extent of the scene is divided into uniformly sized cells along a three-dimensional grid. (The intervals on each dimension need not be equal.) As each photon is inserted into the structure, its thee-dimensional location is compared against the grid and the proper cell selected. The photon is then stored in that cell. During rendering, the kNN search starts by selecting the cell containing the point at which radiance is being estimated, referred to as y_i in final gather visualization, and considers all the photons in that cell. If k photons are not selected, the neighboring cells are considered in order of distance until enough photons are found that match the criteria.

The efficiency of this data structure depends heavily on the distribution of the photons

throughout the scene. Unfortunately, if participating media are not used, photons are stored only on the surfaces of objects (Jensen, 2001). The result is that a large number, sometimes a majority, of cells will in fact be empty while others will have large number of photons. If the grid uses direct addressing, then each cell must be large enough to hold the number of photons in the largest cell. This can be a huge waste of memory if the grid is fine enough to be beneficial. Alternatively the grid could be implemented using indirect addressing. With indirect addressing, the full three-dimensional grid contains only pointers to lists of photons if there are any photons stored in that cell. This limits the waste of memory, but does add a layer of indirection and delay to the memory accesses.

Both approaches were tried in the simulator described in the introduction to this chapter. The direct addressing approach required so much memory that results were not acquired. The indirect approach was too slow, due to the consideration of photons that were not selected. Results were not obtained unless the grid was so fine that even the grid of pointers required too much memory. The uniform grid has however been used in a hardware photon map system that efficiently generated and rendered small photon maps (Purcell et al., 2003). It was made practical in that application by restricting the renderer to direct visualization and restricting the size of individual cells by discarding similar photons. This produces good results for caustics which nearly saturate the rendered pixels, but was insufficient for high quality general indirect illumination.

4.2 *kd*-tree

The *kd*-tree is a binary search tree. Starting from the root node a splitting plane, aligned with one of the three natural axes, is defined. A balanced *kd*-tree selects the splitting plane such that half of the photons under that node are on each side. Finding a single photon therefore requires $O(\ln(N_P))$ memory accesses. Typically, the tree is complete and stored as a priority heap to eliminate the need for pointer overhead.

Jensen selected the balanced *kd*-tree (Bentley, 1975) for his implementation because it enables kNN searches at a cost of $O(k + \ln N_P)$, as analyzed by the RAM model. The *kd*-tree can be difficult to implement in hardware due to the deep chain of indirection required while traversing the tree. This latency can be reduced if the portion of the tree being traversed is able to stay in the cache, or if delay queues are used to allow for the overlapping of requests. The SAARCOR ray tracing architecture provides an efficient strategy and architecture for the traversal of a *kd*-tree (Schmittler et al., 2002). Foley adapted the traversal algorithms directly, to be stack-free, and hence easier to implement in multi-threaded hardware, but with performance reductions (Foley and Sugerman, 2005).

As mentioned above, the RAM model analysis of the balanced *kd*-tree relies on the assumption of a constant memory access time, which is not valid in hierarchical memory systems. A memory request that can not be satisfied by the cache will require a memory fetch of an entire cache line. Photons can be compressed to 20 bytes each (Jensen, 1996a), so a typical 128 byte cache line is able to hold 6 photons with eight bytes available for data structure overhead. If the *kd*-tree is stored in the standard priority heap structure, which eliminates the overhead of storing pointers, only one photon per cache line will typically be considered until the leaves of the tree are encountered. Although the *kd*-tree performs decently for small cache lines, as shown in Figure 4.1, larger cache lines actually require more memory bandwidth.

Wald analyzed the performance of the *kd*-tree and demonstrated an alternative to balancing that achieves a speedup of 1.2 to 3.4 for kNN queries (Wald et al., 2004). This improvement is similar in magnitude to that achieved with the *kd*B-tree.

4.3 *kd*B-tree

The *kd*B-tree (Robinson, 1981) blends the advantages of the *kd*-tree with the classical B-tree (Bayer and McCreight, 1972) by explicitly organizing the photons into cache-sized blocks. The bounding volume of the photon map is recursively divided into mutually exclusive rectangular regions. Each region contains a list of pointers to either another region or to a leaf node. The leaf blocks contain a list of photons that are contained within a bounding volume. The branching factor of the region nodes and the number of photons that can be stored in each leaf node is dependent on the size of a cache line. (In the original paper, the data structure was stored on disk and the size of a disk page was the limiting factor.)

The *kd*B-tree was developed in the database community, which has extensively studied the problem of *k*NN searches (Havran, 2000; Indyk et al., 1997). However, as noted by Ma and McCool (Ma and McCool, 2002), many of these searches are designed for high-dimensional data.

When implemented in the photon map renderer, the *kd*B-tree data structure was found to reduce bandwidth below that of the *kd*-tree only when the cache lines are large. Since each block of data imposes some overhead, as the block size shrinks the overhead begins to dominate. Experiments showed that for the naive ordering, Figure 4.1a, the crossover point was at 256B cache lines, with the *kd*-tree being a better choice for smaller cache lines. Although larger cache lines favor the *kd*B-tree, the absolute bandwidth is actually greater than with small cache lines because of unnecessary photons being brought into the cache without ever being used.

When the photon gathers are already well ordered, as with tiled Hilbert reordering, the *kd*B-tree is actually a poor choice of data structure, Figure 4.1b. This is explained by the observation that the overhead of pointers and region nodes will reduce the number of photons that fit into the cache. When multiple gathers are performed in succession in

the same portion of the photon map, the more compact *kd*-tree data structure is able to make better use of the fixed sized cache.

4.4 Block Hashing

Introduced by Ma and McCool (Ma and McCool, 2002), *block hashing* is an approximate-kNN data structure that stores spatially coherent photons in cache-line sized blocks. It was specifically designed for generating images from a pre-computed photon map. The goal is to reduce memory latency by minimizing the number of dependent memory requests compared to a hierarchical tree traversal, such as the *kd*-tree or *kd*B-tree.

In block hashing, photons are hashed into blocks based on a Locality-Sensitive Hash. Each block is designed to fit within a single cache line, by restricting the number of photons that are placed into leaf blocks, for fast memory access. There are several parameters that must be adjusted for optimum performance, including hash-bucket size and the number of hash tables. The experiments in this chapter follow the heuristics outlined by Ma and McCool to select the parameters.

Block Hashing was designed to reduce the effects of long memory latency, not to decrease memory bandwidth. However, it is competitive in terms of bandwidth with the other data structures. Block Hashing was originally designed with a 256 byte cache line, but the experiments here use 64 to 1024 byte cache lines for a fair comparison, so in the cache line experiments the accuracy parameter was selected in the same spirit as Ma and McCool (Ma and McCool, 2002). Since Block Hashing is an approximate kNN search, an accuracy of $A = 20$ was used to better compare with the exact kNN searches. As this parameter controls the number of blocks that are examined for each search query, it directly impacts bandwidth. Ma and McCool suggest using $A = 10$ to $A = 16$, but these lower values caused a large number of photon blocks to be orphaned at small cache line sizes.

	kd-tree	*kd*B-tree	BH
Naive	23,045	11,755	73,523
Tiled Dir-Binned	3,927	4,397	20,316
Tiled Dir-Binned Hash	2,590	2,870	9,912
Tiled Hilbert	1,215	1,134	2,350
Hilbert	1.6	1.8	3.1

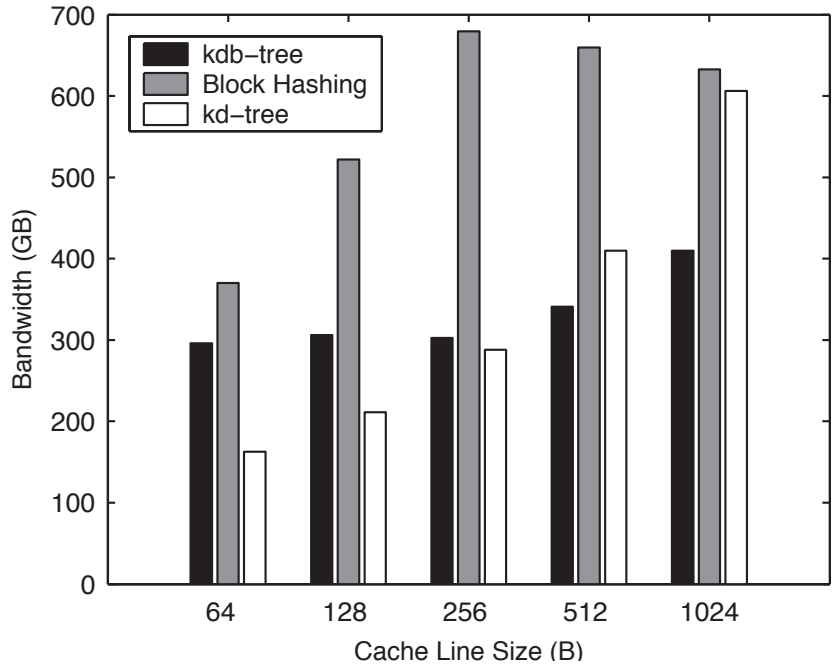
Table 4.1: The average number of memory-block fetches for different combinations of data structures and reorderings. This number is directly related to bandwidth by the size of the cache line. The difference between the data structures is most significant for the naive order, and the effect is negligible for the Hilbert reordering. (Note in particular the excellent performance of the *kd*B-tree when the naive ordering is used.) Regardless of the data structure, memory efficiency is improved more by choosing a better reordering. The block hashing data structure does not reduce bandwidth. It does however reduce latency in the photon gather unit, leading to an implementation with less internal storage. (Cornell box, 16×16 tiles, 128KB cache with 128B cache lines)

4.5 Conclusion

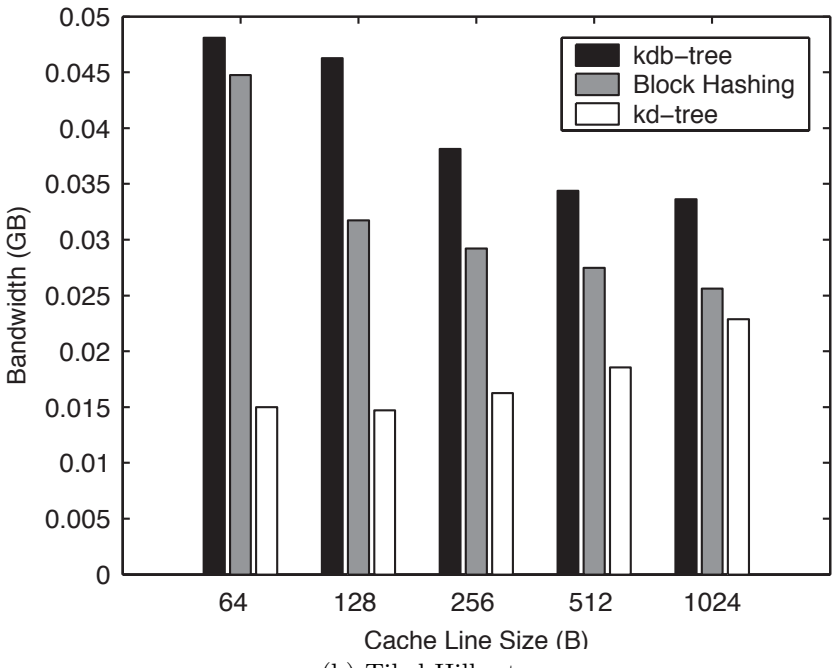
The data structure in which the photon map is stored can make a significant impact on the bandwidth requirements of image generation when using the final gather visualization. A study of three proposed data structures has demonstrated two important results: 1) the choice of data structure is not as important as the ordering of photon gathers; and 2) when an efficient reordering is used, the choice of data structure has a negligible impact.

Table 4.1 shows the interaction between the data structures and reordering algorithms for one specific cache configuration. The block hashing data structure does not reduce bandwidth, although it will reduce latency in the photon gather unit leading to an implementation with less internal storage. The *kd*B-tree reduces the bandwidth requirements by a factor of two compared to the *kd*-tree only for the naive ordering.

With all the other reorderings, the original *kd*-tree actually requires slightly less bandwidth than the other data structures. This is by virtue of having a more compact representation. A larger portion of the photon map can be stored in the cache at any one time. This can be seen clearly in Figure 4.1 where the cache line size is varied. The small



(a) Naive tiled ordering



(b) Tiled Hilbert

Figure 4.1: Interaction between cache line size and bandwidth for various data structures. With an effective reordering algorithm, such as the tiled Hilbert reordering, the data structure choice has little impact on bandwidth regardless of cache line size. Large cache lines actually increase the required bandwidth because unnecessary photons are brought into the cache. Small cache line sizes, however, penalize block oriented data structures that require storage for overhead for each block. (Cornell box, 128KB cache, 16×16 tiles)

cache line sizes penalize those data structures that use the cache line size as a block size and incur overhead per block. Future research could look at alternative implementation strategies that do not limit the block size to the cache line size. Additionally there are other data structures as well as variations of those described in this chapter that should be looked at in more detail. The choice of data structure becomes more complicated when dynamic photon map generation is considered.

Based on these results, the well known and analyzed *kd*-tree is used throughout the rest of this dissertation, including the architecture presented in Chapter 7. The *kd*-tree is simple, well-understood and the chains of indirect memory accesses can be hidden using delay queues as have been implemented in proven architectures.

CHAPTER 5

IRRADIANCE CACHING

In 1988, Ward et al. observed that indirect illumination tends to change slowly over a surface and "should not be recalculated at each pixel, but should instead be averaged over surfaces from a small set of computed values" (Ward et al., 1988). Suppose that the indirect illumination at the point x' in Figure 5.1 has been carefully computed and that the result has been stored in a data structure that can be indexed by location. In many situations it would be a waste to expend effort computing the indirect illumination at a nearby point x. Instead, the data structure should be consulted, finding all the points near x, such as x'. The stored illumination results can then be interpolated, providing an estimate for the indirect illumination at x

This is an alluring tactic considering how expensive it can be to compute indirect illumination using a high quality Monte Carlo integration, such as final gather visualization with hundreds of secondary rays. In photon mapping, each ray not only must be traced through the scene, but also requires a memory bandwidth intensive photon gather. Both of these actions have computational and bandwidth costs, even when using photon gather reordering, as was explored in Chapter 3.

In their paper, Ward et al. described a concrete implementation of this idea, the *irradiance cache*. The irradiance cache algorithm is easy to implement because of an important assumption that is made: the surfaces seen in the image are purely diffuse. Because the reflectance function, $f_r(x, \vec{\omega_o}, \vec{\omega_i})$, does not depend on the angle of incident light, $\vec{\omega_i}$, only a single cumulative value must be stored in each record to represent all indirect illumination, the irradiance, at a single point. Unlike radiosity, not *all* the

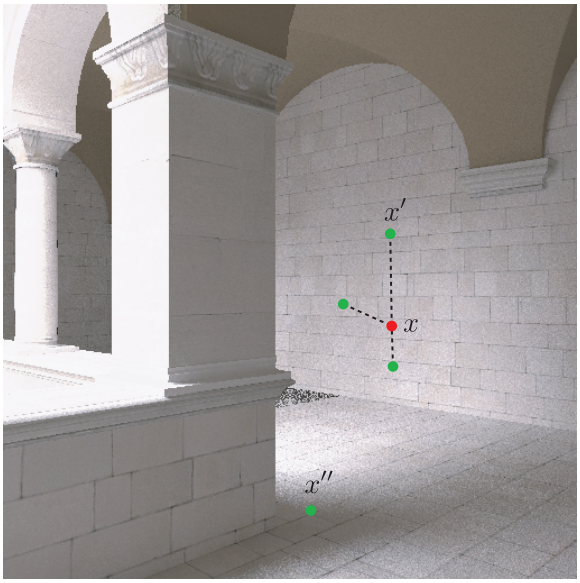


Figure 5.1: As observed by Ward, indirect illumination tends to change slowly across a surface. With the introduction of some bias, the indirect illumination that is expensively calculated at x' can be reused at x. Care must be taken to reduce the use of erroneous values that do not share the same indirect illumination. For example, x'' is shaded by the balcony wall.

surfaces in the scene are required to be diffuse. However, irradiance caching is only able to provide an advantage for those pixels of the scene that project to diffuse surfaces.

This chapter begins with a review of the details of irradiance caching. Two obstacles to the implementation of irradiance caching in a parallel system, such as the architecture described in Chapter 7, are described and novel solutions to each provided. The positive interaction of irradiance caching with photon gather reordering is then explored. Finally, results for the test scenes of Chapter 1 are presented and discussed.

5.1 The Irradiance Cache

The rendering equation, introduced in Chapter 2, provided a means for computing the radiance reflected from a point x towards a viewer from all directions.

$$L_r(x, \vec{\omega}_o) = L_e(x, \vec{\omega}_o) + \int_\Omega f_r(x, \vec{\omega}_i, \vec{\omega}_o) L_i(x, \vec{\omega}_i)(\vec{\omega}_i \bullet \vec{n}) d\vec{\omega}_i \tag{5.1}$$

It is common to separate the computation of the incident radiance, L_i, into two parts: direct illumination, light arriving directly from the original light sources; and indirect illumination, all other light. Throughout this chapter it is assumed the direct illumination has already been calculated through the means described in Chapter 2, as they are typically more efficient. Additionally, the term for emitted radiance, L_e, is also removed for clarity.

The central assumption of the irradiance caching algorithm is that the Bidirectional Reflectance Distribution Function (BRDF) at x is purely diffuse. The consequence is that for all $\vec{\omega}_i$, $f_r(x, \vec{\omega}_i, \vec{\omega}_o)$ is a constant and can be pulled out of the integration and replaced by a constant c:

$$L_r(x, \vec{\omega_o}) = \int_\Omega f_r(x, \vec{\omega_i}, \vec{\omega_o}) L_i(x, \vec{\omega_i})(\vec{\omega_i} \bullet \vec{n}) d\vec{\omega_i} \tag{5.2}$$

$$= c \int_\Omega L_i(x, \vec{\omega_i})(\vec{\omega_i} \bullet \vec{n}) d\vec{\omega_i} \tag{5.3}$$

$$= cE(x) \tag{5.4}$$

A second consequence of the diffuse surface assumption is that the reflected radiance, as computed in Equation 5.3, is not dependent on the viewing direction $\vec{\omega}_o$. The integral portion of that equation is known as the *irradiance* at location x. Denoted as $E(x)$, it has units of watts per square meter. Irradiance measures the total power of the light landing on an area. The irradiance at a point does not depend on what angle it is seen from, and once computed it is no longer necessary to remember the angular distribution of the incident light. Therefore, irradiance requires only a small amount of storage: 3 floating point numbers if the typical RGB spectrum is used. It is for these reasons that the irradiance cache stores irradiance values as opposed to the full incident radiance function.

As discussed in Section 2.3.3, the rendering equation can not usually be evaluated analytically, because L_i can only be evaluated for specific directions, so Monte Carlo

integration is often used. The same situation arises with computing the irradiance at point x, and Monte Carlo integration can be used here as well[1]:

$$E(x) = \int_{\Omega} L_i(x, \vec{\omega})(\vec{\omega} \cdot \vec{n}) d\omega \tag{5.5}$$

$$E(x) = \frac{2\pi}{N} \sum_{j=1}^{N} L_i(x, \vec{\omega_j})(\vec{\omega_j} \cdot \vec{n}) \tag{5.6}$$

Note the close similarity between Equations 5.6 and Equation 2.5 which described the final gather. They are both the weighted averages of the sampled incident radiance $L_i(x, \vec{\omega}_i)$. The irradiance can easily be computed at the same time as a final gather at very little additional cost.

5.1.1 Error Estimate

When the reflected radiance from a particular point x is required, a decision must be made. Either a new irradiance value must be computed, using the expensive Monte Carlo final gather, or the current records in the irradiance cache can be interpolated together to approximate the irradiance. The algorithm presented in the next section uses a heuristic error model to make this determination. Specifically, Ward introduced the split-sphere model, Figure 5.2, to approximate the error, $\epsilon_{x'}(x)$, of using the irradiance stored at x' at x:

$$\epsilon_{x'}(x) = E(x') \left(\frac{4}{\pi} \frac{||x - x'||}{R_{\text{avg}}} + \sqrt{2 - 2\vec{n}_x \cdot \vec{n}_{x'}} \right) \tag{5.7}$$

The split-sphere model is of course only a heuristic because it can not know the true

[1]Equation 5.6 is valid only if the the directions $\vec{\omega_i}$ are sampled uniformly from the hemisphere Ω. In the more general case where importance sampling is used (see Chapter 6) and the samples are drawn from a probability distribution function, the equation must be modified as follows:

$$E(x) = \frac{1}{N} \sum_{j=1}^{N} \frac{L_i(x, \vec{\omega_j})(\vec{\omega_j} \cdot \vec{n})}{p(\vec{\omega_j})}$$

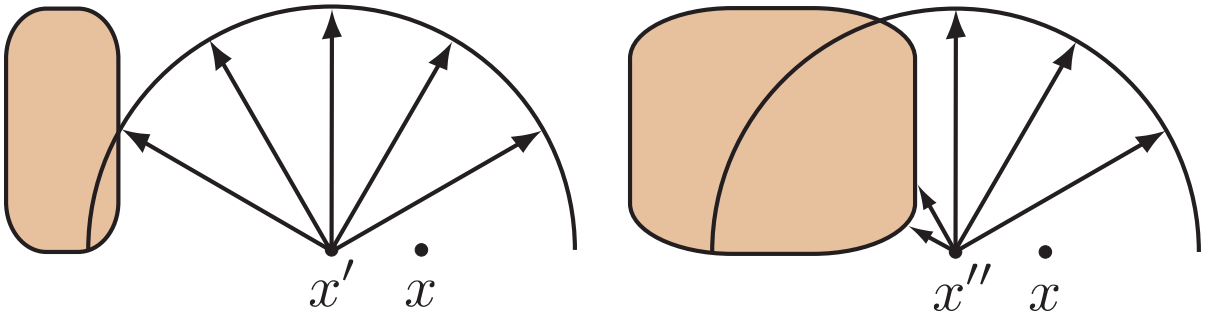


Figure 5.2: Ward's split-sphere model computes an estimate of how likely indirect illumination is to vary at points near x. The assumption is that the farther away the geometry in the scene is, the smaller that changes to visibility will be and hopefully also illumination. Consider for example the change in the amount of additional sky that becomes visible if x'' is moved slightly, compared to x'. This distance is measured by applying the harmonic mean to the length of the final gather rays to compute R_{avg}.

irradiance at x. The error term does however take four important factors into account: 1) change in the normal between the two points; 2) Euclidian distance between x and x_i; 3) irradiance at x'; and 4) how far away the rest of the scene is, R_{avg}. The first two factors convey a sense of the geometric change present in the scene, without relying on an analysis of the actual geometry, which might be complicated or procedural. The last term approximates the rest of the scene as residing on a sphere of radius R centered at x'. The further away the sphere is, the further x can be from x' without introducing significant error. Since the scene is not actually a sphere, an estimate is made by averaging the distance to all surfaces seen during Monte Carlo integration. The harmonic mean is used, as it considers high values to be outliers and concentrates on smaller, i.e. closer, values. This presents a conservative estimation, analyzed in the original paper.

$$R_{\text{avg}} = \frac{N}{\frac{1}{R_1} + \frac{1}{R_2} + \frac{1}{R_3} + \cdots \frac{1}{R_N}} \tag{5.8}$$

Although the split-sphere model is used in this dissertation, there are other error models that could be used. Irradiance gradients, for example, have been shown to produce images of lower variance at the cost of a more complicated formulation (Ward and

Heckbert, 1992). However, initial investigations indicate that it will be more difficult to apply irradiance gradients to a parallel system.

5.1.2 The Algorithm

The irradiance cache must be updated and queried during the rendering of a single image. Like the photon map, it must be efficient to perform a kNN search to find the nearby candidate points x' that have pre-computed values of irradiance. However, unlike the photon map, the irradiance cache must be updated continuously whenever a new value is computed. As a compromise between these two requirements, Ward et al. proposed using the octree data structure (Samet, 1990).

When the pixel-driven rendering algorithm (Algorithm 2.1) is used to generate images using final gather photon map visualization, the use of the irradiance cache requires two straightforward modifications. Before a final gather is performed at any point x, the irradiance cache is queried to find all records, S, that are in the irradiance cache and potentially close enough to x to be useful. Each record $j \in S$ contains: the location x_j; the irradiance at x_j, $E(x_j)$; the surface normal at position x_j, $\vec{n}_{x_j}$; and the calculated value of R_{avg} at x_j.

For every element $j \in S$, the error model is evaluated with respect to x. If the estimated error is greater than a user specified maximum threshold value, $\epsilon_{x_j}(x) > \epsilon_{\max}$, then that record is discarded from S. If an insufficient number of records remains after this process, a final gather uses Equation 5.6 to compute $E(x)$, and the result is stored as a new record in the irradiance cache.

If, however, a sufficient number of records *do* have a low enough estimated error, they can be interpolated to compute $E(x)$, thereby avoiding the high cost of a final gather. Interpolation requires the selection of appropriate weights. The interpolation weight that

point $j \in S$, has at point x is $w_{x_j}(x)$:

$$w_{x_j}(x) = \frac{1}{\epsilon_{x_j}(x)} = \frac{1}{\frac{||x-x_j||}{R_{\text{avg}}} + \sqrt{1 - \vec{n}_x \cdot \vec{n}_{x_j}}} \tag{5.9}$$

The value R_{avg} is the stored value at x_j. The interpolation is then very straightforward, the weights are normalized and the results accumulated for all records in S to compute the estimate of $E(x)$:

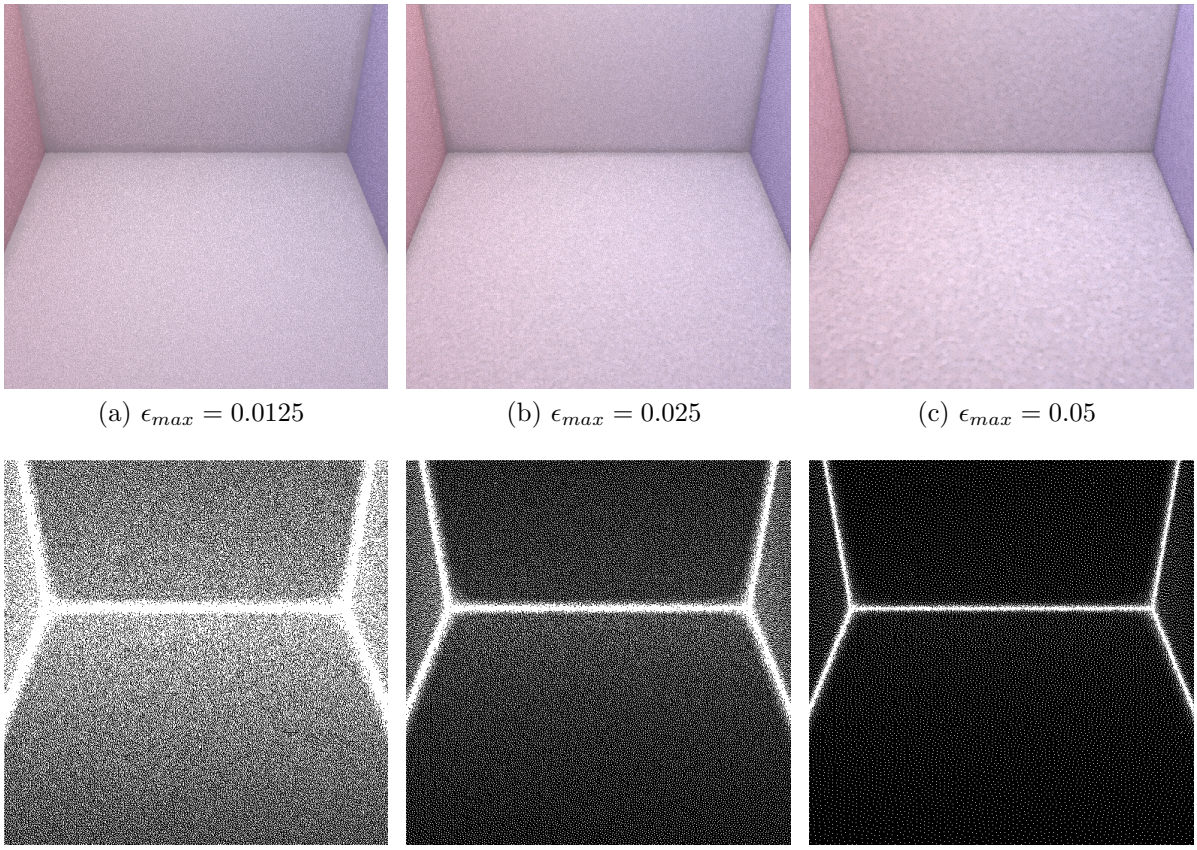
$$E(x) = \frac{\sum_{j \in S} w_{x_j j}(x) E(w_j)}{\sum_{j \in S} w_{x_j j}(x)} \tag{5.10}$$

There are additional heuristic tests that Ward describes in the paper to decrease the likelihood of interpolating across separate surfaces. However, they are only heuristics as the full geometry is not known.

5.1.3 Discussion

The irradiance cache works best for scenes that are predominately diffuse and have low geometric complexity. An extreme example is the empty modified Cornell Box in Figure 5.3. The top row shows the box rendered using the irradiance cache for a variety of user specified maximum errors, ϵ_{max}. The second row is a visualization of the rendering process, the sites in the scene where this irradiance cache could not be used and a new final gather was performed are set to white, while all other pixels are black. Observe that the majority of final gathers are performed near the corners where R_{avg} is small, indicating that indirect light will change rapidly due to close geometry (in this case it is the wall). This can also be seen on the floor. The wall behind the viewer is significantly farther away than the side walls, hence there are more gathers performed on the portions of the floor close to the front wall than at the bottom of the image.

The Sponza atrium shown in Figure 5.4 is also purely diffuse, but has a somewhat more complicated geometric design. In particular, the areas near the ornamentation on



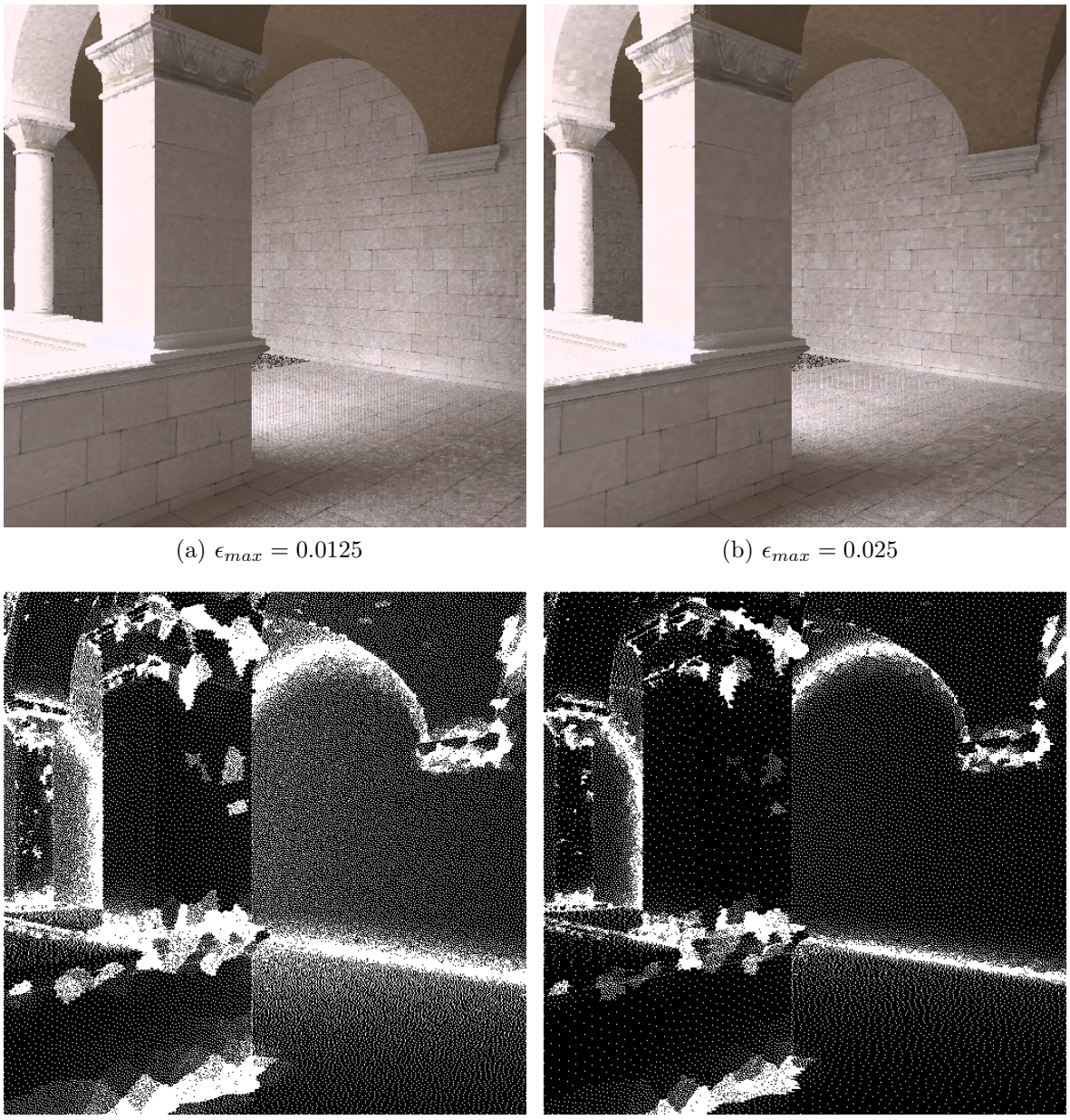
(a) $\epsilon_{max} = 0.0125$ (b) $\epsilon_{max} = 0.025$ (c) $\epsilon_{max} = 0.05$

(d) $\epsilon_{max} = 0.0125$, $R_{IC} = 45\%$ (e) $\epsilon_{max} = 0.025$, $R_{IC} = 76\%$ (f) $\epsilon_{max} = 0.05$, $R_{IC} = 91\%$

Figure 5.3: By increasing the user specified maximum allowed error, more pixels are able to make use of the irradiance cache. The white pixels in the bottom row indicate where a final gather was performed. R_{avg} was calculated directly during the final gathers. Irradiance caching is very effective for this scene, at all values of ϵ_{max} as shown by the large proportion of pixels able to use the irradiance cache, R_{IC}.

the walls, and the columns are almost incapable of using the irradiance cache at all. Fortunately, the flat walls are able to utilize the irradiance cache significantly, presenting an overall reduction in final gathers. This combination of intricate detail and large flat walls is common to architectural walkthroughs.

Some scenes can present greater difficulties to the irradiance caching algorithm than the ones examined so far. In Figure 5.5 the geometry of the modified Cornell box is modified by the inclusion of geometrically modeled bushes with simple diffuse material properties. The maximum error is kept constant as the number of instances of the bush is increased. This is a particularly difficult scene because there are many small gaps in the



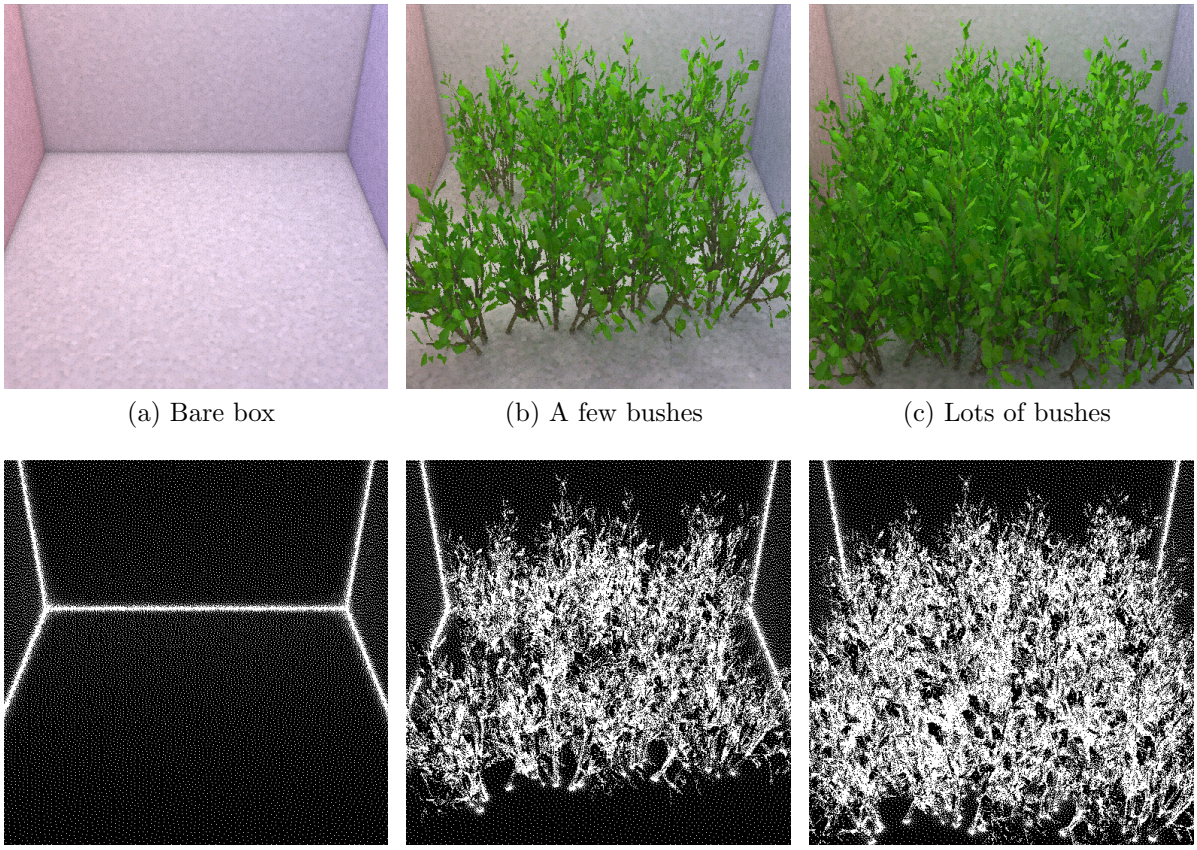
(a) $\epsilon_{max} = 0.0125$

(b) $\epsilon_{max} = 0.025$

(c) $\epsilon_{max} = 0.0125$, $R_{IC} = 66\%$

(d) $\epsilon_{max} = 0.025$, $R_{IC} = 82\%$

Figure 5.4: By increasing the user specified maximum allowed error, more pixels are able to make use of the irradiance cache. The white pixels in the bottom row indicate where a final gather was performed. R_{avg} was calculated directly during the final gathers. Irradiance caching is very effective for this scene, at all values of ϵ_{max}.



(a) Bare box (b) A few bushes (c) Lots of bushes

(d) Bare box, $R_{IC} = 91\%$ (e) A few bushes, $R_{IC} = 69\%$ (f) Lots of bushes, $R_{IC} = 57\%$

Figure 5.5: When the geometry is more complicated, the three tests: Euclidean distance, normal and in front of, will fail much more often. This reduces the proportion of pixels able to use the irradiance cache, R_{IC}. ($\epsilon_{max} = 0.05$) *Bush model by 3DPlants.com.*

foliage and the normals vary extensively. The portion of the image covered by the bushes is unable to make much use of the irradiance cache despite a relatively high maximum error threshold.

The irradiance cache as has been described makes its decisions on a pixel by pixel basis. Each pixel potentially uses all the previously computed values for the entire image. This imposes a sequential dependency between pixels. This dependency breaks the natural parallelism of photon mapping by imposing two separate obstacles. The first is that the image can not be computed as independent tiles, as was suggested for query reordering in Chapter 3. The second is that the computation of R_{avg} must be computed for a new record in the irradiance cache *before* a subsequent pixel can decide to make use

of the result. This also presents a problem when reordering is used.

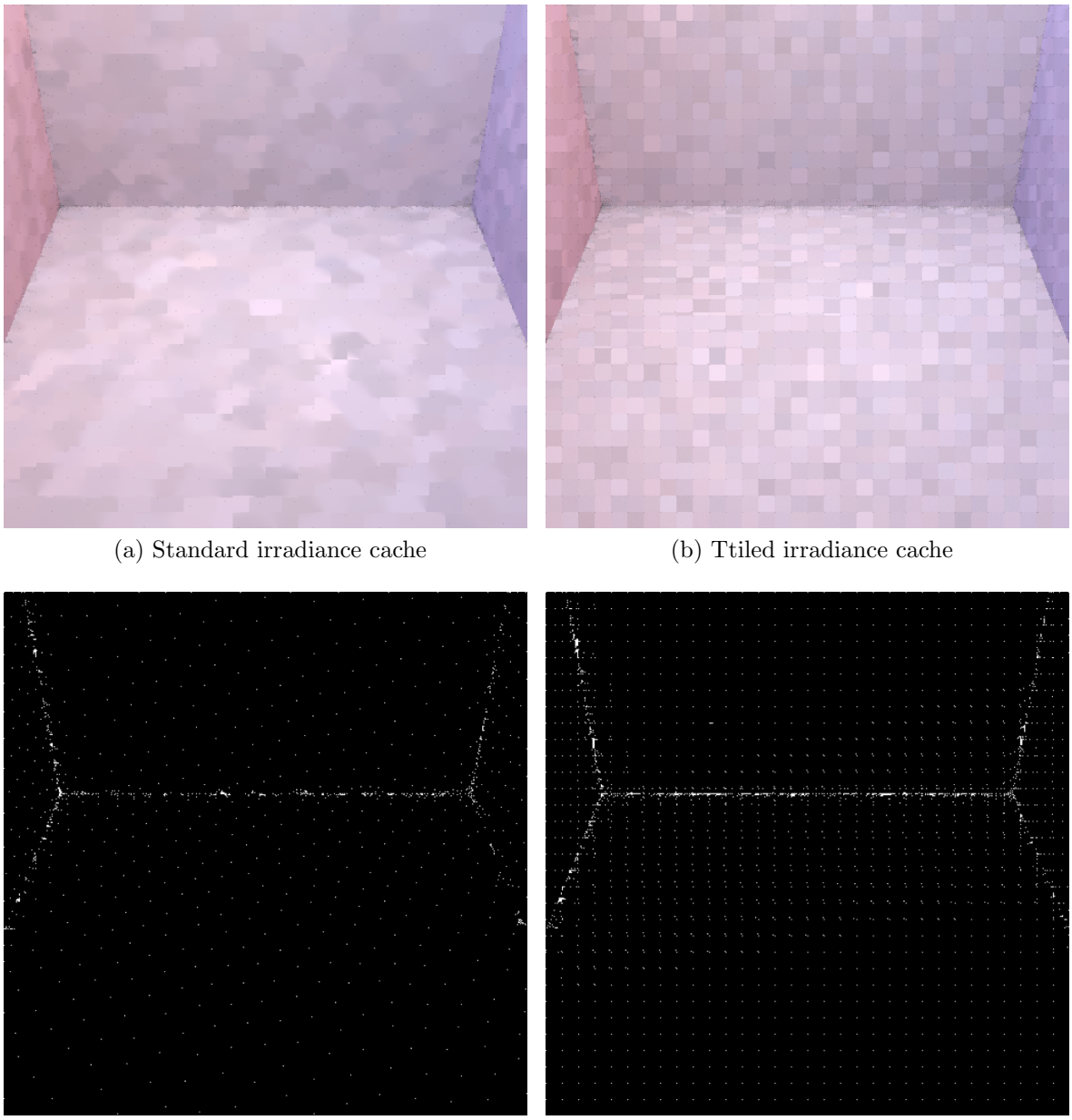
In this chapter I propose two simple techniques that eliminate these two dependencies without significantly reducing the effectiveness of irradiance caching or noticeably changing the generated images. These techniques will allow the architecture described in Chapter 7 to render the image in independent tiles on separate processors and to process all the pixels in a single tile at the same time using photon gather reordering.

5.2 Tiled irradiance caching

The architecture in Chapter 7, like most of those described in Chapter 2, will process the image in small independent tiles. The tiles are processed in parallel on different chips in an unpredictable order with no communication allowed. Instead of a single shared irradiance cache, which would require a large and expensive interconnect, I investigated the impacts of starting each tile with an empty cache. The irradiance caching algorithm is conservative. If it is not able to access all the sites it ordinarily would, it will simply perform more final gathers than it would optimally.

The impact of the reduced size irradiance cache was found to be insignificant, unless the maximum allowed error is so high that the sphere of influence of any one record projects to an area on the screen larger than the size of a tile. Figure 5.6 shows this exaggerated case. In this example, 16×16 tiles were used, each containing 256 pixels, but the maximum error was set so high that the tile boundaries became visible because 2 records were typically computed per tile and used throughout the entire tile. As a result the shape of the irradiance cache artifacts become square instead of the small smooth circles usually expected. This is only a significant problem when $\epsilon_{\max}$ is so large as to provide otherwise unacceptable results. The effect of the independent irradiance caches is negligible for smaller irradiance values.

The key motivation for the separate irradiance caches was the high cost of maintaining



(a) Standard irradiance cache

(b) Ttiled irradiance cache

(c) Standard irradiance cache

(d) Tiled irradiance cache

Figure 5.6: Irradiance caching with a new cache for each 16×16 tile. In this demonstration, the maximum allowed error is set very high, $\epsilon_{max} = 0.3$, which provides the large circular blurring of the indirect illumination in the standard image. This blurring is restricted to the tile size when independent caches are used, giving more jarring square artifacts. In practice, the maximum error is not allowed to be this large and the effects of tiled irradiance caching are negligible.

an accurate shared copy among many independent processors. There is therefore never a copy of an irradiance cache for the whole image. A consequence is that it can not be reused in the next frame; the cache will start empty for each tile for each frame.

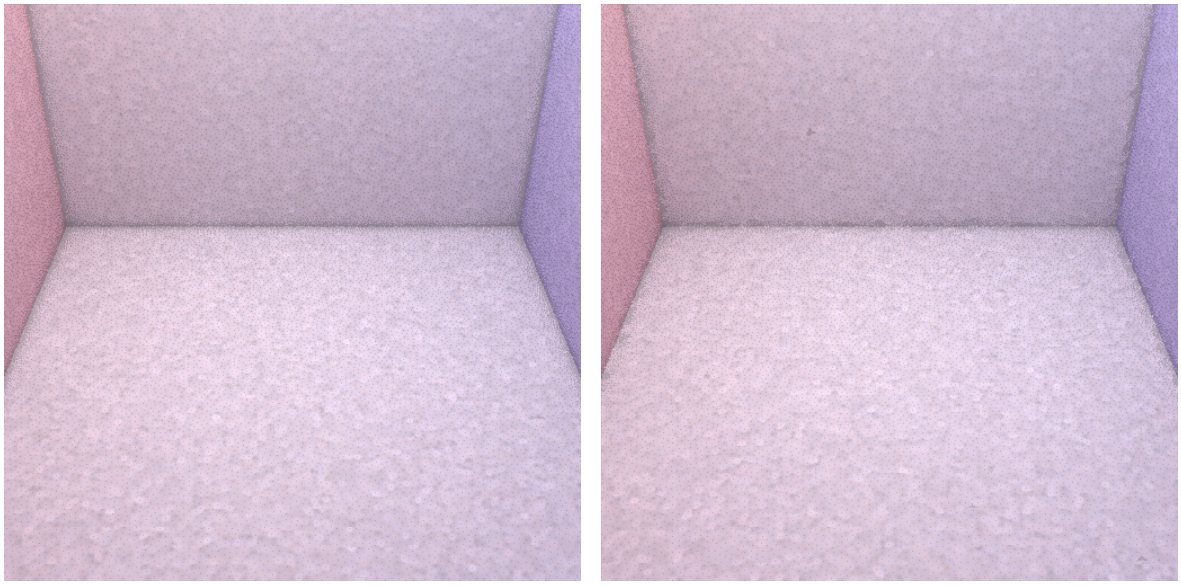
5.3 Pre-computing the harmonic mean distance

The irradiance cache algorithm as previously described requires the computation of the value R_{avg} for each irradiance value before it is inserted as a record into the cache. Typically, this value is computed during a final gather by computing the harmonic mean of the distances to the scene in the directions sampled, see Section 5.1. This value is then used to approximate the error that other points would experience if they used the value of this record instead of recomputing the indirect illumination.

The standard irradiance cache algorithm fully evaluates each pixel, potentially performing this final gather and computing $E(x)$ and $R_{\text{avg}}(x)$, before moving on to the next pixel. However, the reordering schemes that will be used in the proposed architecture, fully described in Chapter 3, requires that all sites that will use final gathers be identified *before any of the final gathers for that tile are performed*. Since the value R_{avg} is typically computed as a byproduct of the final gather, a new method must be found.

As the rendering of a tile begins, an initial pass over the entire tile is performed to determine which pixels will use interpolated values and which must perform full final gathers. Those that can interpolate a value add themselves to a linked list at the pixel performing a final gather. In this manner, when the final gather is eventually performed the resulting irradiance value can be applied to all the pixels that decided to use that irradiance value.

There is a difficulty however. The irradiance cache starts empty for each tile. As it is determined that pixels can not use interpolation and must perform a full final gather, they need to be added to the irradiance cache so that neighboring pixels can choose



(a) Direct Computation of R_{avg} (b) Pre-computing R_{avg} for a subset of photons

Figure 5.7: Pre-computing R_{avg} at a subset of photons. Using pre-computed values of R_{avg} instead of the directly computed values does not noticeable change the results. ($\epsilon_{max} = 0.05$)

to use them. However, this has to happen before any final gathers take place in order to allow for the reordering. Not only is the irradiance value, $E(x)$, not yet computed, but R_{avg} as previously described required the final gather. Inspired by Christensen's approach to storing irradiance directly on the photon's themselves (Christensen, 1999), I propose pre-computing R_{avg} at a subset of the photons before any image generation begins.

During the initial pass over the tile, an extra photon gather at the location x is performed, but only the closest photon to x that has a pre-computed value of R_{avg} is consulted with the result used in order to insert the pixel x into the irradiance cache.

A minor change to the computation of $\epsilon_{x_j}(x)$ must be made. The value of $E(x_j)$ can not be known at the time that it is needed to make the irradiance caching decision. It can however be conservatively removed from the error term. This is conservative because higher values of $E(x)$ would reduce the number of pixels that need to compute a full final gather. Performing unnecessary final gathers will only reduce the observed variance in

the image.

This technique does not noticeably change the observed results as shown in Figure 5.7. This is because, unlike indirect illumination which can change sharply without regard for geometry, at a shadow boundary for example, the distance to the rest of the scene changes smoothly. The cost of the extra photon gather required to find the value of R_{avg} is not significant because only one per pixel is required and they are highly coherent. The upfront cost of pre-computing the values compares favorably to tracing 100 rays to sample the visibility at every pixel before tracing the final gather rays for the final gather.

5.4 Interaction with photon gather reordering

The architecture presented in Chapter 7 will combine photon gather reordering with irradiance caching. The difficulties of implementing them together have been discussed and addressed earlier in this chapter. It is a separate issue to consider the consequences on bandwidth requirements of using these techniques together. This interaction was studied in two ways.

For the first method, the bushy Cornell boxes in Figure 5.5 were studied while varying the parameters for both irradiance caching and photon gather reordering. The addition of progressively more bushes creates high geometric complexity. In particular there are holes that the eye rays can intermittently pass through. This feature of the scenes will cause the irradiance cache heuristics to reject most cache points in the bushy box, causing more final gathers to be computed.

As shown in Table 5.1, query reordering is compatible with and complementary to irradiance caching. In scenes that are very compatible with irradiance caching, such as the bare Cornell box, irradiance caching will provide a greater overall reduction than

	Naive	Tiled DirBin	Tiled Dirbin Hash	Tiled Hilbert
Bare Cornell box				
Without Irradiance Cache	100.0%	12.7%	9.6%	5.6%
With Irradiance Cache	1.8%	1.2%	1.2%	1.1%
Two Sets of Bushes				
Without Irradiance Cache	100.0%	23.0%	11.4%	**6.3%**
With Irradiance Cache	**8.5%**	3.8%	2.9%	2.4%
Many Bushes				
Without Irradiance Cache	100.0%	23.9%	**12.0%**	**6.0%**
With Irradiance Cache	**12.6%**	5.4%	3.7%	2.9%

Table 5.1: Interaction between irradiance caching and query reordering. The bandwidth for each combination of scene, irradiance caching, and query reordering technique is shown as a percentage of the naive ordering without irradiance caching. Query reordering is effective both when irradiance caching is and is not used. Irradiance caching is always more effective than just query reordering for simple scenes, such as the bare Cornell box. However, as the geometric complexity increases, irradiance caching by itself becomes less effective than the query reordering techniques, as highlighted by the bold figures. (Modified Cornell box, *kd*-tree, 128B cachelines, 128KB cache and 16×16 tiles.)

even the best query reordering algorithm. Query reordering does, however, provide the same proportional benefit to irradiance caching as it does without it.

As the scene becomes more complex, however, query reordering is more powerful than irradiance caching alone. Consider the Cornell box with two sets of bushes; the tiled Hilbert reordering is more effective than irradiance caching at reducing bandwidth. For example, in the scene with two sets of bushes, the tiled Hilbert reordering requires 6.3% of the bandwidth compared to 8.5% for irradiance caching. The difference is even starker for the scene with many bushes, as shown in the bold numbers. This reduction is possible without any of the potential quality problems inherent to the irradiance cache algorithm.

In both cases however, the two algorithms can work together to produce superior results. The implementation of query reordering is an orthogonal system-design decision to that of irradiance caching. Unless it is known that irradiance caching will be ineffective

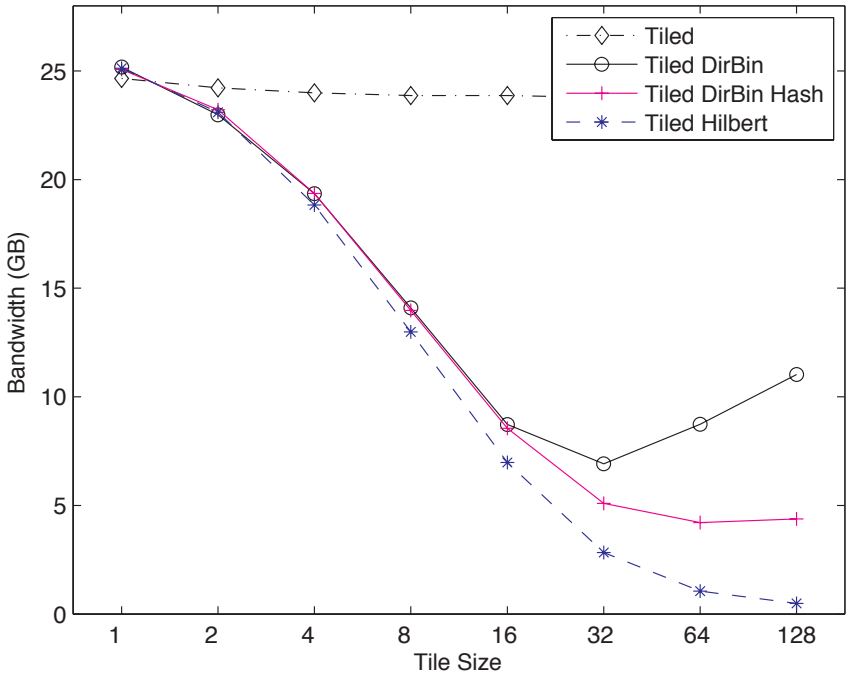
for all desired scenes, the combination of the two methods is the best strategy.

An alternative method for evaluating the interaction is to repeat the experiments of Section 3.2 with irradiance caching enabled. The results are shown in Figure 5.8, and should be compared to those presented in Figure 3.5. The implications are the same; photon gather reordering and irradiance caching are complementary techniques and do not interfere with each other. A sensible system will implement both to reduce memory bandwidth requirements.

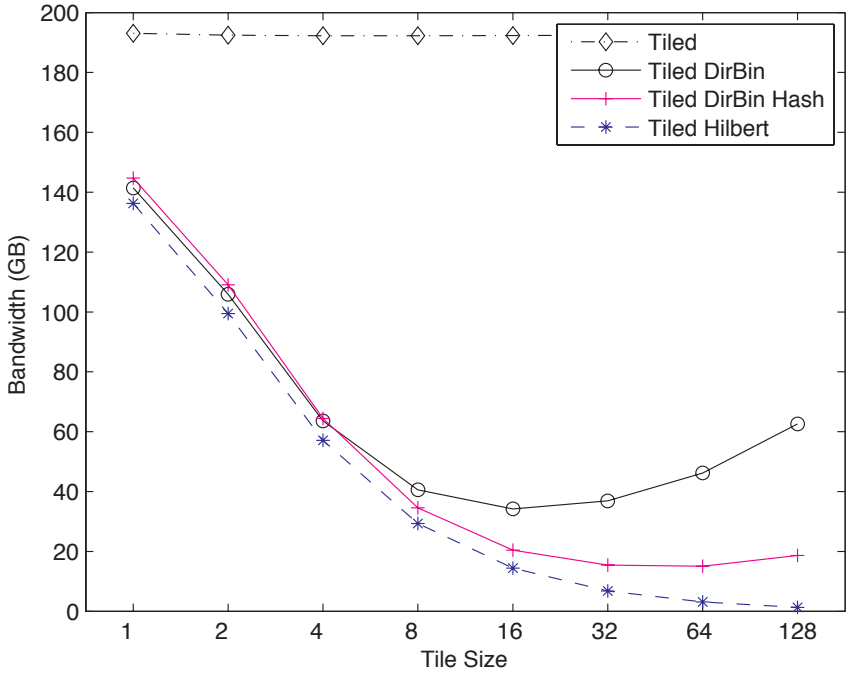
5.5 Possible Extensions

The irradiance cache as defined is restricted to diffuse surfaces. Radiance caching, introduced by Křivánek, permits a process like irradiance caching on glossy surfaces (Křivánek et al., 2005). This is performed by storing an approximation of $L_i(x, \vec{\omega})$ as a set of spherical harmonic coefficients instead of a single irradiance value. When the radiance cache site is then used by another pixel, the spherical harmonic coefficients are evaluated. The spherical harmonics are used to reduce the storage size of the radiance cache. However, by approximating the sparse evaluation of radiance in spherical harmonics some information will be lost. Additionally, finding the coefficients takes computational effort. These drawbacks could be overcome by an implementation that follows the general process described in this chapter.

In order to break the sequential dependency between pixels, it was necessary to decide *before* each final gather which other pixels will use the computed irradiance value. This could be further adopted by sending to the unit performing the final gather the information, such as local surface normal, about all the pixels which wish to make use of the photons found in the final gather. The single kNN search could then be approximately used as the photon gather for all these pixels at once. The computation burden is higher then irradiance caching, but the bandwidth requirements would remain the same.



(a) Modified Cornell Box, $N_{FG} = 33$, $\epsilon_{\max} = 0.25$



(b) The Sponza Scene, $N_{FG} = 200$, $\epsilon_{\max} = 0.00625$

Figure 5.8: This graph shows the same experiment as in Figure 3.5, but with irradiance caching. Reordering remains effective when combined with irradiance caching, although the magnitude of the improvement is smaller. Reordering alone creates a more significant reduction in bandwidth than irradiance caching, even for these two scenes that are highly compatible with irradiance caching.

Configuration	Bandwidth	
	Cornell	Sponza
Tiled naive	50 GB	357 GB
Tiled naive w/ Irradiance caching	24 GB	193 GB
Tiled direction-binning with hashed	12 GB	31 GB
Tiled direction-binning with hashed w/ Irradiance caching	9 GB	21 GB

Table 5.2: The bandwidth requirements to render a single frame of the modified Cornell box, Figure 1.2 and the Sponza atrium, Figure 1.3, with and without irradiance caching and reordering. The combination of tiled direction-binning generative reordering with hashed deferred reordering and irradiance caching brings the bandwidth requirement to just above the maximum that an 8-chip system would provide; 24 GB per image at 30 frames per second. (*128 KB cache, 128 byte cache lines,* 16×16 *tiles, kd-tree.*)

Overall system bandwidth will actually decrease because it will now be possible to use indirect illumination interpolation on glossy surface which were previous ineligible for interpolation. However, intermediate storage needs are increased and the implementation becomes more complex.

Finally, another possible extension to the methods described in this chapter is to build upon Christensen's approach to improving the speed of photon mapping(Christensen, 1999). At the same time that values of R_{avg} are computed on some photons during a pre-process, the incident irradiance at purely diffuse points can be pre-computed and stored as well. The normal photon gathers that take place during a final gather are then replaced with a search for the closest photon with a stored irradiance value. This does not reduce the number of photon gathers, but does reduce the number of photons that must be fully processed at a compromise in image correctness.

5.6 Conclusions

The irradiance cache is an effective technique for reducing the computational and bandwidth requirements of photon mapping. It is only effective for diffuse surface with low geometric complexity. It can successfully be adapted to the organizational structure

required by the architecture presented in this dissertation by two simple modifications: breaking the image into independent tiles with separate irradiance caches and then breaking the sequential dependency between pixels. Irradiance caching is compatible with photon gather reordering, providing significant further reductions in memory bandwidth. However, the requirements are still steep. In the next chapter another technique for reducing not the number of final gathers but the number of final gather rays is explored.

CHAPTER 6

IMPORTANCE SAMPLING

This chapter presents the third technique of the dissertation for reducing the memory bandwidth requirements of generating images using a photon map with final gather visualization. Chapters 3 and 4 examined algorithmic and data structure alternatives that reduced the costs without modifying the photon gathers performed or the resulting images. Chapter 5 presented irradiance caching as a way to reduce the number of final gathers actually performed, by interpolating previously computed results across multiple pixels of the final image. By contrast, this chapter examines a method for reducing the cost of those final gathers that must be performed. This technique will also improve the quality of the generated images, especially for scenes with glossy material properties or uneven indirect illumination, conditions that are particularly difficult for global illumination algorithms.

The Monte Carlo integration technique, introduced in Section 2.3.4 as a means for evaluating the rendering equation, forms the basis of the final gather as used to generate images from a photon map. N_{FG} secondary final gather rays are sampled from the hemisphere of visible directions and the incident radiance is evaluated, forming an estimate of the reflected radiance. The intuitive idea of *importance sampling* is to carefully choose those N_{FG} directions such that they sample the portion of the incoming light that will contribute greatly to the reflected radiance. This will allow for the generation of higher-quality, lower-variance images for the same number of samples. Alternatively, fewer samples are needed to achieve the same image quality as before.

This chapter begins with some background, first revisiting Monte Carlo integration

and examining why it can take extremely high values of N_{FG} to generate images with sufficiently low variance. The concept of importance sampling is then introduced and explained through a one-dimensional example. Multiple importance sampling, which forms the basis of many computer graphics sampling algorithms is also reviewed.

Importance sampling is then applied to the rendering equation, in particular to the context of final gather photon map visualization. I review existing methods that leverage the knowledge of the incident radiance that is stored in the photon map. Existing techniques are reviewed, but they are found to be either too costly or not provide a sufficient reduction in variance. This leads to the presentation of a novel scheme, combined importance sampling. The schemes are compared in terms of variance reduction performance, intermediate storage and computational cost. Combined importance sampling is found to be effective yet requires approximately one fiftieth the computation and little intermediate storage.

Combined importance sampling is then used to generate actual images using varying number of final gather rays and the resulting image quality and bandwidth requirements are compared. In addition to those scenes presented in Chapter 1, scenes that are more difficult for global illumination algorithms, such as those with glossy surfaces and no direct lighting, are tested as they are only practical with importance sampling because the number of final gather rays is otherwise too high for an interactive system. Finally, the implications for the implementation in a parallel system utilizing query reordering and irradiance caching are considered.

6.1 Background

Numerical integration was introduced in Section 2.3.3 as a technique for calculating an estimate, $\hat{\Theta}$, for the integral of a function which can not be solved analytically for its true value, Θ. The integral is replaced by the weighted sum of N discrete point evaluations

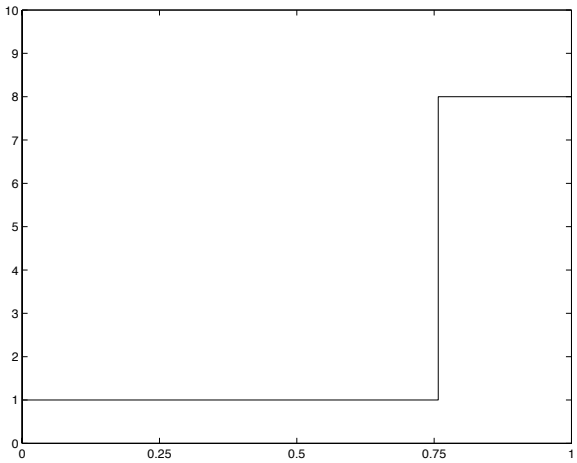


Figure 6.1: This simple piecewise constant function $g(x)$ is used as an example integrand throughout this section. By inspection, $\Theta = \int_0^1 g(x)dx = 2.75$.

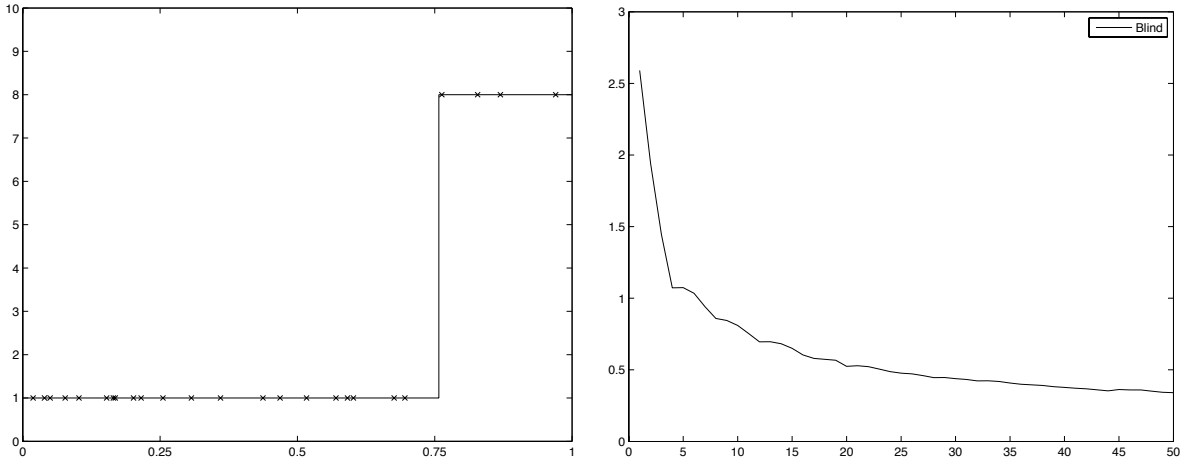
of the integrand, $g(s)$, as shown in Equation 6.2. There are an entire family of numerical integration estimators that differ in their selection of the points, S_i, at which $g(s)$ is evaluated and the weights, W_i, which describe how they are combined.

$$\Theta = \int_\Omega g(s)d\mu(s) \tag{6.1}$$

$$\hat{\Theta} = \sum_{i=1}^{N} W_i g(S_i) \tag{6.2}$$

Before considering the numerical integration of the rendering equation later in this chapter, a simple example is used to explain *why* importance sampling works. Figure 6.1 shows a one dimensional piecewise constant function $g(x)$. This function was chosen as it is simple to analytically integrate, allowing the various estimators introduced in this section to be directly compared to the true value Θ. The region of integration is $\Omega = [0, 1]$ and $d\mu(x)$ is simply dx. The integral is the area under the function, $\Theta = \int_0^1 g(x)dx = 2.75$.

Section 2.3.4 introduced Monte Carlo integration. The samples X_i are chosen randomly from the region of integration, Ω. The generic Monte Carlo estimator F_N is shown in Equation 6.3. Since all the weights in Monte Carlo estimators have a common term



(a) 25 Uniform random samples (b) Expected error of N sample

Figure 6.2: Although the large spike from .75 to 1 contributes most of the value of the integral of $g(x)$, only a few of the uniformly random samples in Figure (a) happen to find it. For these particular samples, blind Monte Carlo integration finds a value for $\hat{\Theta}$ of 2.12, quite far from the correct value of 2.75. Figure (b) repeats this experiment 500 times for each value of N between 1 and 50. Although larger values of N can be expected to give results closer to Θ, the rate of convergence is low, $O(\frac{1}{\sqrt{N}})$. 40 samples is only twice as good as 10.

of $\frac{1}{N}$, this has been factored out of the summation.

$$F_N = \frac{1}{N} \sum_{i=1}^{N} W_i g(X_i) \tag{6.3}$$

If $W_i = 1$ and every point in Ω has the same probability of being sampled by X_i then this is called the blind Monte Carlo estimator. In the simple one-dimensional example, $X_i = \zeta_i$, where ζ_i is uniform random variable from the range $[0, 1]$. In Figure 6.2a, Ω has been sampled 25 times, as marked by the crosses. At each sample point, the value $g(X_i)$ has been evaluated. Blind Monte Carlo integration with this particular set of samples gives an estimated integral of $\hat{\Theta} = 2.12$. This estimate will change for different sample locations.

Although Blind Monte Carlo is very straightforward, you simply need to uniformly pick random samples in the domain, it can take a large value of N to have confidence in the estimate. The rate with which F_N will converge to Θ is $O(\frac{1}{\sqrt{N}})$ (Rubinstein, 1981).

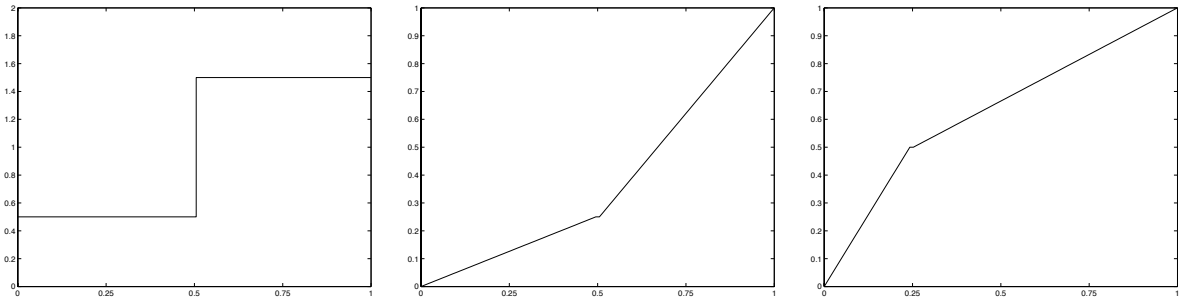
To reduce the error by half, four times as many samples must on average be taken. This is clear in Figure 6.2b where for each sample size N, 500 different trials were run for each sample size and the average standard deviation from the true value of Θ was computed. Quadrupling $N = 10$ to $N = 40$ only reduced the error by one half.

Some regions of a function contribute more to the true value of the integrand than others. In our example, the region $[.75, 1]$ contributes 2, while the larger region $[0, .75]$ contributes only .75. It can be argued that blind Monte Carlo integration in Figure 6.2a wasted a large number of samples on the region $[0, .75]$.

To reduce the variance of the estimate $\hat{\Theta}$, it is logical to focus the samples where $g(x)$ is large, as they will contribute more to the final answer. This is because a small relative variance in the important regions will alter the total estimated value *more* than the same relative variance in those regions which have a small value of $g(x)$. In general, Monte Carlo integration performs best if $f(x)$ is a constant function. However, we are rarely interested in integrating constant functions.

6.1.1 Importance sampling

The goal of importance sampling is to choose the N samples such that it is more likely that a sample X_i will contribute significantly to $\hat{Q} = F_N$. Importance sampling requires prior knowledge. Suppose we were informed, somehow, that $g(x)$ in the range $[0.5, 1]$ will contribute more to the final result than $[0, 0.5]$ will. We would like to put more samples in that part of the domain. This prior knowledge is expressed in terms of a probability density function (p.d.f.) $p(x)$. A potential $p(x)$ for our example function is shown in Figure 6.3a. Making use of this knowledge to construct a new Monte Carlo estimator requires us to do two things. The first is to pick the set of samples, X_i, not uniformly but according to $p(x)$. They should still be random, but no longer uniformly distributed. For the $p(x)$ in Figure 6.3a we would expect many more sample locations above 0.5 than



(a) $p(x)$ (b) $c(x)$ (c) $c^{-1}(x)$

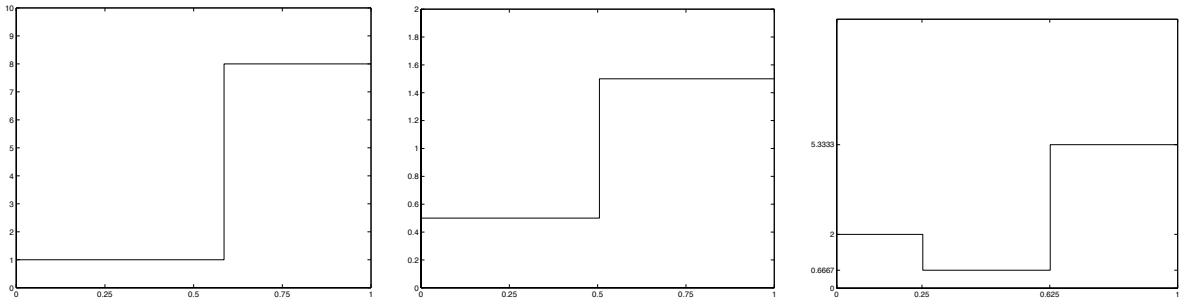
Figure 6.3: Importance sampling chooses the samples $X_i = c^{-1}(\zeta_i)$ where ζ_i is a uniform random variable. For this example probability distribution function $p(x)$, only a quarter of the samples are expected to be below 0.5, as is reflected in the cumulative distribution function $c(x)$. The construction of $c^{-1}(x)$ enables fast selection of samples according to $p(x)$.

below it.

Choosing a sample X_i according to $p(x)$ is easy if $p(x)$ is known. The first step is to find the cumulative distribution function (c.d.f.), $c(x)$, that corresponds to $p(x)$. This is shown in Figure 6.3b. This function is then inverted to form $c^{-1}(x)$, Figure 6.3c. The idea is to sample $c^{-1}(x)$ with uniformly random numbers ζ_i to find the samples X_i that will be used. In our example, only one quarter of the random numbers are mapped to the range $[0, 0.5]$. The other three quarters are instead mapped to the second half of the range of $g(x)$. If ζ_i is a uniform random variable in $[0, 1]$, then let the set of samples $X_i = c^{-1}(\zeta_i)$. This changes the function presented to the Monte Carlo algorithm from $g(x)$ to $g(c^{-1}(x))$.

Simply using the different samples will result in a biased answer unless the weights are also adjusted. By inspection of Figure 6.4a we see that $\int_0^1 g(c^{-1}(x))dx = 3.625$, a significant error from the correct value of 2.75. By setting $W_i = \frac{1}{p(X_i)}$ we can correct for this error. The more likely we were to choose a specific sample, the less we should take its result into account. On the other hand, a sample that was chosen despite a low probability *but* has a high value $g(X_i)$ should count highly in the estimate.

It is easiest to visualize importance sampling by studying what happens if we use



(a) $f(c^{-1}(x))$ (b) $p(x)$ (c) $\frac{f(c^{-1}(x))}{p(x)}$

Figure 6.4: The sampling strategy described in this section generates samples according to $p(x)$. Using these samples directly in the Monte Carlo estimator without adjusting the weights would however lead to a biased result. Setting $W_i = \frac{1}{p(X_i)}$ is equivalent to using blind Monte Carlo integration on the new function $\frac{g(c^{-1}(x))}{p(x)}$.

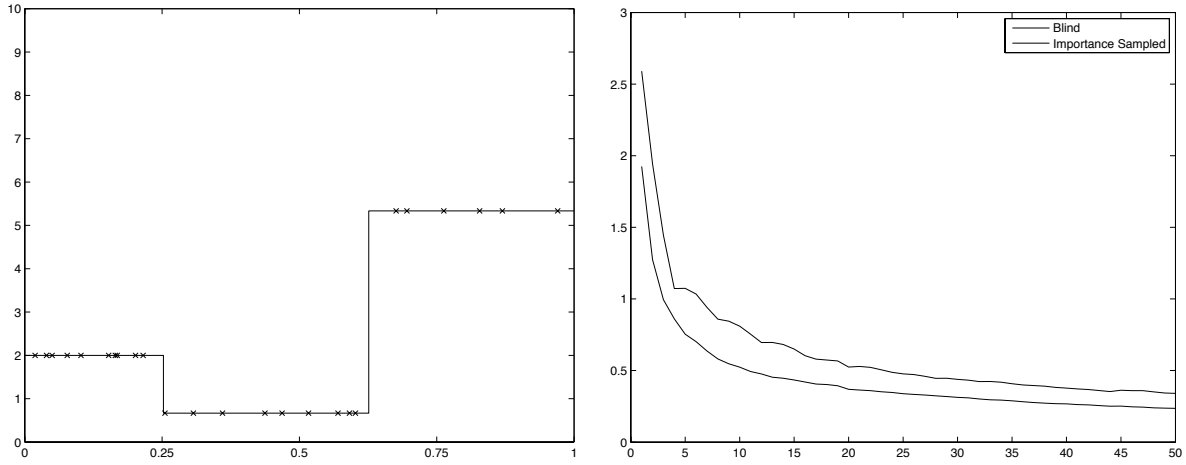
blind Monte Carlo integration of the function $\frac{g(c^{-1}(x))}{p(x)}$, as presented in Figure 6.4c. By inspection it can be verified that $\int_0^1 \frac{g(c^{-1}(x))}{p(x)} dx = 2.75$ as expected. When this new function is sampled using 25 uniform random numbers we get the results shown in Figure 6.5a. The estimated result is closer to the true value Q. Indeed, Figure 6.5b shows that importance sampling with the little information given to us in $p(x)$ requires fewer than half the samples that blind Monte Carlo does for the same expected error.

The expected error of Monte Carlo integration using importance sampling can not be expressed without knowing the specific function and probability distribution function. However, the expected error is derived directly from the definitions:

$$O\left(\sqrt{\int \frac{g^2(X)}{p^2(x)} p(x) dx - \Theta^2}\right) \tag{6.4}$$

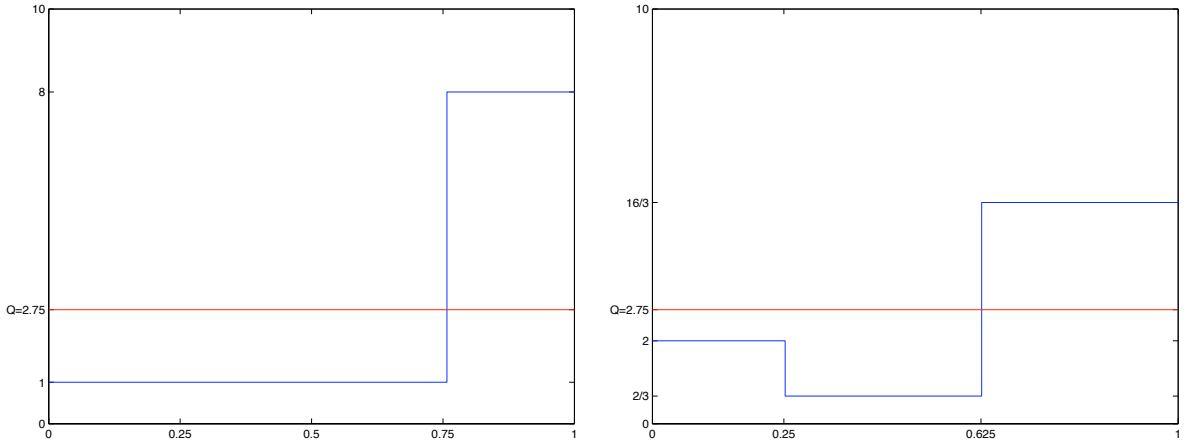
Intuitively we know that the more that $p(x)$ resembles $g(x)$ the lower the error will be. Indeed, inspection of Equation 6.4 shows that the error would be zero if and only if $p(x) = kg(x)$ for some constant k. In practice, even a rough correlations between $p(x)$ and $g(x)$ is good enough to provide a reduction in variance (Veach, 1997).

Figure 6.6 demonstrates graphically why importance sampling works. Recall that



(a) $F_N = 2.32$ (b) Average Standard Deviation

Figure 6.5: Using importance sampling, the same number of samples will result in a smaller error from the true value. In Figure (a), 25 samples computed a estimate of 2.32, closer to 2.75 than the blind Monte Carlo estimator achieved with the same number of samples. Repeating the experiment from Figure 6.2, importance sampling is shown to be significantly better than blind Monte Carlo. Fewer than half the number of samples are required for equivalent results.



(a) $g(x)$ (b) $\frac{g(c^{-1}(x))}{p(x)}$

Figure 6.6: Importance sampling works because it attempts to integrate a function that is closer to being a constant. The average standard deviation of $g(x)$ from 2.75 over the range $[0, 1]$ is 3.03, while it is merely 2.29 for $\frac{g(c^{-1}(x))}{p(x)}$.

blind Monte Carlo works best when the integrand is as close to a constant function as is possible. One way to measure the uniformity of a function is to compute the average standard deviation to the average value throughout the domain. For our example the average value is Θ. The average standard deviation of $g(x)$ from 2.75 is 3.03, while that of the modified function $\frac{g(c^{-1}(x))}{p(x)}$ is 2.29. The function as modified by importance sampling is therefore a more constant value, allowing the blind (standard) Monte Carlo method to perform better.

6.1.2 Multiple importance sampling

For complicated integrands, it is unlikely that there will be known a single p.d.f. $p(x)$ that contains all the prior information that is known about the integrand. Instead, we might be presented with a series of M probability density functions each of which recognize the contribution of a specific important section of Ω. Consider for example the rendering equation, a rough idea of both L_i and f_r might be known, while the dot product is known perfectly. This provides three separate sampling strategies, each expressed as a separate p.d.f., $p_j(x)$. If the budget for generating and evaluating samples allows only N samples, they could be allocated between the M sampling strategies, such that $N = \sum_{j=1}^{M} N_j$, to create a new Monte Carlo estimate:

$$F_{M,N} = \sum_{j=1}^{M} \frac{1}{N_j} \sum_{i=1}^{N_j} W_j(X_{i,j}) \frac{g(X_{i,j})}{p_j(X_{i,j})} \tag{6.5}$$

In this estimate, sample $X_{i,j}$ is the ith sample chosen according to strategy j, using $p_j(x)$.

This approach was studied by Veach, who labeled it multiple importance sampling (Veach and Guibas, 1995). The difficult task is assigning the weights $W_j(X_{i,j})$ to each sample to minimize the error. Veach showed that using a constant weight for each sampling strategy $p_j(x)$, does not lead to better convergence and instead proposed a heuristic, which he proved to be near optimal.

The Balance Heuristic (Equation 6.6) however computes an individual weight for each sample $X_{i,j}$ as generated by sampling strategy j. The weight compares the probability that $X_{i,j}$ was generated by $p_j(x)$ to the probability that the other distributions would have generated the same sample.

$$W_j(X_{i,j}) = \frac{N_j p_j(X_{i,j})}{\sum_k N_k p_k(X_{i,j})} \tag{6.6}$$

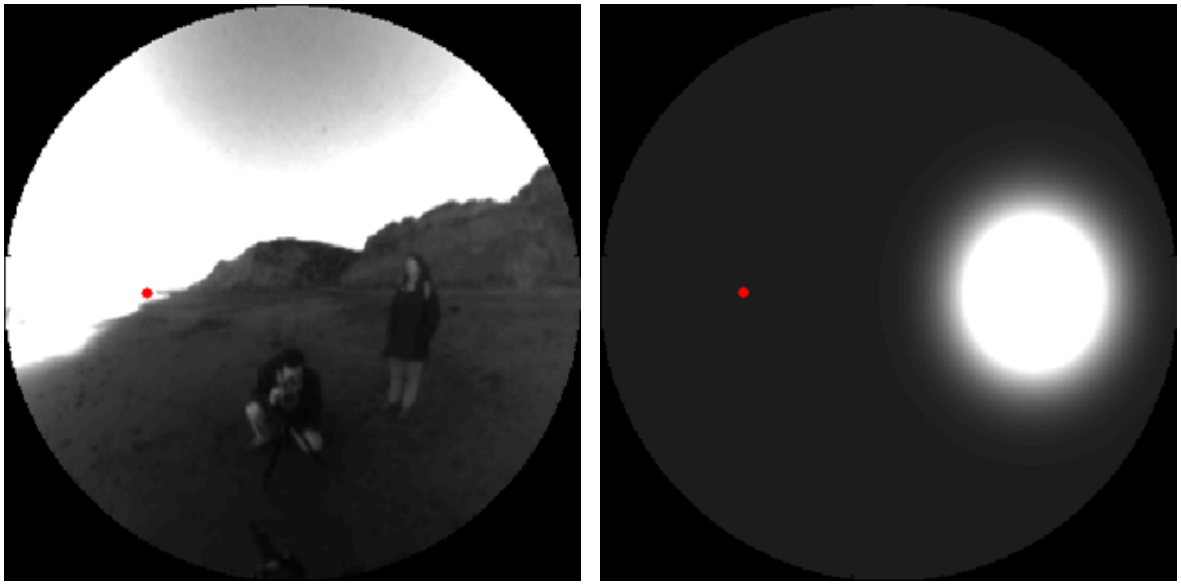
The samples that have a large probability of being selected by one p.d.f. but were in fact selected by another, that only had a low probability, will not have undue influence on the final result. As explored by Veach, this greatly approach reduces erroneously large values from being effecting the estimated. The Balance Heuristic does however require that all M p.d.f.s be evaluated for all N samples. If any of the p.d.f.s are expensive to compute, this can become a large computational burden.

6.2 Sampling the rendering equation

The final gather in photon mapping uses Monte Carlo techniques to numerically evaluate the rendering equation, reproduced from Section 2.3.2 in Equation 6.7. The integral in the rendering equation is the product of three terms: the incident radiance $L_i(x, \vec{\omega}_i)$ (the light arriving at a point x from the direction $\vec{\omega}_i$), the Bi-directional Reflection Distribution Function (BRDF) $f_r(x, \vec{\omega}_i, \vec{\omega}_o)$, and a cosine term (dot product between $\vec{\omega}_o$ and the surface normal.

$$L_r(z, \vec{\omega}_o) = L_e(z, \vec{\omega}_o) + \int_\Omega f_r(z, \vec{\omega}_i, \vec{\omega}_o) L_i(z, \vec{\omega}_i)(\vec{\omega}_i \bullet \vec{n}) d\vec{\omega}_i \tag{6.7}$$

Reducing variance in final-gather global-illumination algorithms is a well established field of research. Szmirmay-Kalos has published a particularly through survey (Szirmay-Kalos, 1999). This chapter focuses on the importance sampling technique just introduced.

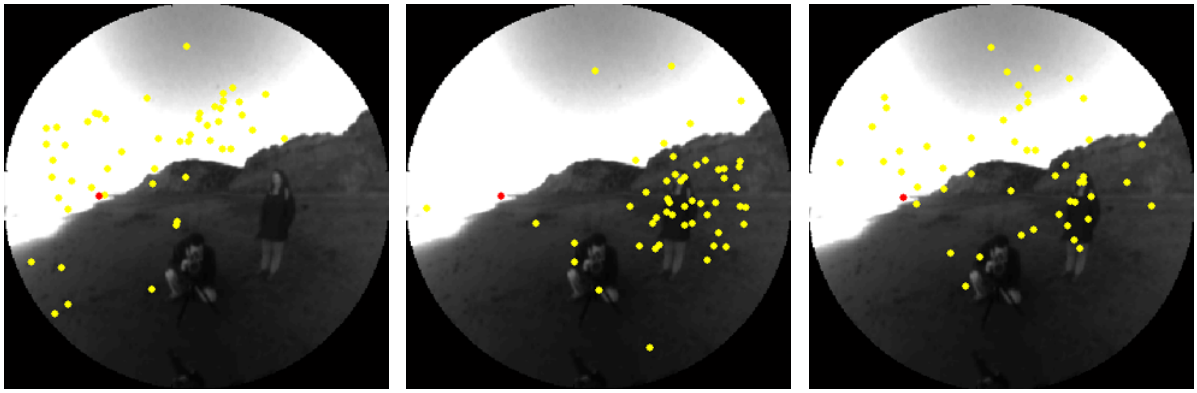


(a) Incident Radiance (b) BRDF

Figure 6.7: The captured incident radiance at a point and the evaluated BRDF for the given viewing direction, represented by the red dot, on a glossy surface. Global importance sampling considers both of these when generating samples in order to maximize the contribution of each sample towards the reflected radiance. (*The beach lightprobe used by permission of Paul Debevec (Debevec, 1998)*)

Typically when importance sampling is applied to the rendering equation, it is applied to only two of the three terms. (The cosine term is easy to incorporate into either the incident radiance or BRDF.) In contrast, *global importance sampling* techniques are able to importance sample according to all aspects of the integrand.

The utility of global importance sampling can be seen in the Figure 6.7. The entire hemisphere of incoming directions was captured by a physical camera. The portion with the sky is very bright in comparison to the rest of the scene. Sampling according to a p.d.f. that captures this knowledge would have us sample that portion of the sky much more than the rest as shown in Figure 6.8a. The BRDF, however, is very glossy and because of the position of the virtual camera, is strong in a different part of the hemisphere, facing the ground this time. Sampling according to a p.d.f. based on the BRDF alone dedicates most of the samples in that region of the sky as shown in Figure 6.8b. Clearly it would be best to take both into account, as demonstrated in Figure 6.8c.



(a) Incident Radiance (b) BRDF (c) Global

Figure 6.8: Sample directions generated with importance sampling according to different p.d.f.s. Global importance sampling will result in lower variance by taking both incident radiance *and* reflectance into account.

6.2.1 Sampling the BRDF

It has long been common practice in computer graphics to generate samples according to the product of the cosine and BRDF terms (Blasi et al., 1994; Dutré, 1994; Lange, 1991). Figure 6.8b shows a representative sampling according to the BRDF shown in Figure 6.7. This requires the computation of a transform, or warp, $T(u, v)$ with a density that matches the value of the BRDF. For example, the Lambertian BRDF can be importance sampled using the transform

$$T(u, v) = (\phi = \arccos(\sqrt{u}), \theta = 2\pi v) \tag{6.8}$$

where u and v are uniform random variables and (ϕ, θ) is the generated sample direction (Shirley, 1992).

BRDFs composed of one or more cosine lobes and a diffuse component, including all those derived from the Phong BRDF (such as Lafortune's) can be sampled analytically using a two step scheme (Lafortune and Willems, 1993). A lobe is chosen and its exponent is used to generate a sample in the coordinate space of the lobe. The sample is then transformed into the shared local coordinate system of x. The cost of computing the

probability of having chosen the sample is directly proportional to the number of lobes. This is because the lobes may overlap, and therefore any particular sample might have been generated by multiple lobes.

The BRDF composed of multiple lobes must be sampled carefully. The naive sampling strategy would randomly choose a lobe for each sample and then a random sample on that lobe. A subsequent sample would likely pick a different lobe, most likely generating a sample with a significantly different direction. This will reduce the amount of coherence that is available, and hence increase memory bandwidth requirements. To maximize the coherence of sampled directions, generate all the samples for each lobe before considering the next lobe. Further discussion of the interaction with photon gather reordering is found in Section 6.5.

More complicated BRDFs, such as those that are fitted to directly acquired data, can not be analytically importance sampled and are difficult to sample efficiently. There are however some recent works tackling this problem. Lawrence has proposed a scheme that factors BRDFs for sampling purposes (Lawrence et al., 2004). He also has proposed a method using a curve approximation of the c.d.f. to reduce the size the multidimensional table of acquired BRDFs (Lawrence et al., 2005).

6.2.2 Sampling the incident radiance function

It can be difficult to importance sample the incident radiance L_i because, as was discussed in Section 2.3.3, it is the very function for which we are solving! In some constrained situations, however, the radiance function at a specific point in the scene is provided as an environment map and assumed to be true for all points nearby (Greene, 1986). Environment maps can be used for importance sampling by constructing a large table to compute the inverse c.d.f. Several authors have shown how to pre-process an environment map with a specific BRDF to enable efficient importance sampling using structured sampling,

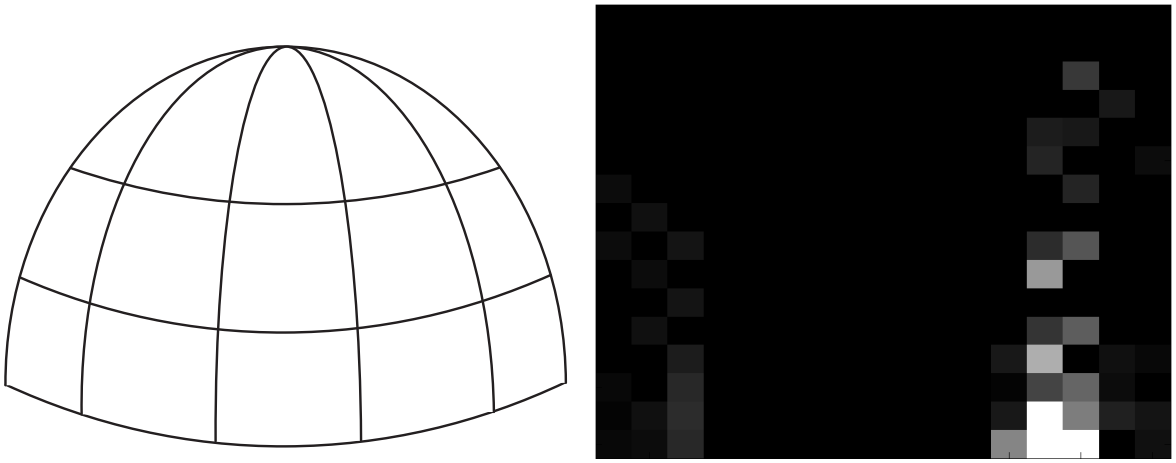
spherical harmonics and wavelets (Agarwal et al., 2003; Ramamoorthi and Hanrahan, 2001; Clarberg et al., 2005). Bidirectional Importance Sampling is a recent two-step algorithm that improves the quality of samples generated according to the product of an environment map and a BRDF (Burke et al., 2005).

Adaptive techniques are an alternative solution to global importance sampling. By storing previously generated samples into a hierarchical data structure, a coarse approximation of the radiance is built up, which is then used to guide future sampling (Bustillo, 1997; Lafortune and Williams, 1995).

In the remainder of this section, three published strategies for importance sampling in the context of photon mapping are presented. By taking advantage of the sparse representation of incident radiance stored in the photon map, they are able to generate sample directions that will likely provide a large contribution to the reflected radiance. The methods differ in how they use the photon map and how and if they also incorporate information from the BRDF. It is found that while the existing methods all have their strengths, they either do not produce samples of sufficiently high quality, or are simply too expensive for use in an interactive system. This motivates my proposed method as presented in Section 6.3.

6.2.3 Jensen's method

Jensen originally described the use of a photon map for importance sampling (Jensen, 1995). Since the photon map is a coarse approximation of incident radiance throughout the scene, it can be used to establish a useful probability distribution function for importance sampling. The original technique suggested by Jensen was simple and effective. Before each final gather is performed, a preprocess is performed, Algorithm 6.1. The photon map is queried at point x for k photons that meet certain criteria for relevance. The direction associated with each photon is projected onto the unit square using the



(a) (b)

Figure 6.9: Jensen's technique divides the hemisphere above x into bins, Figure (a), which are projected onto a unit square. A kNN search is performed, and each bin accumulates the energy carried by the photons that arrive from directions within the bin. By normalizing the energy table, a tabular p.d.f. is obtained, Figure (b). (The (θ, ϕ) mapping has placed those bins facing the sun in Figure 6.7a to the bottom right of the table.) Th p.d.f. is then turned into an inverse c.d.f and sampled, generating samples like those shown in Figure 6.8a.

inverse transformation $T^{-1}(\theta, \phi)$ defined by the BRDF importance sampling. The unit square is divided into a grid of C_J cells. Jensen found that a grid of size 4×8 was often suitable.

Any cells with zero energy are given a small amount; although there doesn't appear to be much light arriving from those directions based on this small sampling, it would cause bias if the cell is assumed to be completely dark. The amount of energy donated to these cells is user specified. The implementation used in this dissertation follows the suggestions in Jensen's paper. The accumulated energy in each cell is then normalized so that they sum to one. The resulting table is a tabular p.d.f., giving a discrete approximation of how important each segment of the hemisphere above x is. Using any well-defined order, such as row-major, the probabilities in the cells are summed together, forming a one-dimensional array storing the c.d.f..

To generate each sample, following Algorithm 6.2, a random number ζ is uniformly

// Find k nearest photons and project their energy into a 2D array of the hemisphere
queryResult $\leftarrow k$NN(x)
for each $P \in$ queryResult **do**
  $[u, v] \leftarrow T^{-1}(P_\theta, P_\phi)$
  energyTable$[u, v] \leftarrow$ energyTable$[u, v] + P_e$
  totalEnergy $\leftarrow$ totalEnergy $+ P_e$
end for
// Donate a small portion of the total energy to any empty cells
energyDonation $\leftarrow 0.05 \times$ totalEnergy$/C_J$
for each cell $[u, v] \in$ energyTable **do**
  if energyTable$[u, v] == 0$ **then**
    energyTable$[u, v] \leftarrow$ energyDonation
    totalEnergy $\leftarrow$ totalEnergy + energyDonation
  end if
end for
// Normalize the energy table, producing the p.d.f.
for each cell $[u, v] \in$ pdfTable **do**
  pdfTable$[u, v] \leftarrow$ energyTable$[u, v]$/totalEnergy
end for
// Construct a 1D c.d.f. array
cdfArray[0] $\leftarrow 0$
for $i = 1$ to $C_J + 1$ **do**
  cdfArray$[i] \leftarrow$ cdfArray$[i - 1]$ + pdfTable$[i]$
end for

Algorithm 6.1: Jensen's method for importance sampling begins with a kNN search in the indirect photon map. These photons are then used to compute a low-resolution 2D table of incident radiance. Only k photons are used, leaving the possibility that a dim region of the hemisphere will appear totally dark. Those cells are given a small amount of energy to prevent bias. The energy table is then normalized to create a 2D p.d.f., the prefix sum of which forms a 1D c.d.f. function that can be directly sampled with a uniform random number from the range $[0, 1]$.

for $i = 1$ to N_{FG} **do**
  // Choose which portion of the hemisphere to sample
  $\zeta_1 \leftarrow \text{RandomUniformInRange}(0, 1)$
  $idx \leftarrow \text{BinarySearch}(\text{cdfTable}, \zeta_1)$
  $[u, v] \leftarrow \text{TranslateIndex}(idx)$ // Which 2D cell does idx correspond to
  // Generate a sampling direction within the portion of the hemisphere $[u, v]$
  $\zeta_2 \leftarrow \text{RandomUniformInRange}(0, 1)$
  $\zeta_3 \leftarrow \text{RandomUniformInRange}(0, 1)$
  $(\theta, \phi) \leftarrow T([u, v], \zeta_2, \zeta_3)$
end for

Algorithm 6.2: Generating the final gather rays using Jensen's method begins by choosing a single cell according to the inverse of the c.d.f.. A random ray within that cell is then generated and given the probability indicated by pdfTable$[u, v]$.

chosen in the range $[0, 1]$. The cell in the c.d.f. containing ζ is selected, and a random direction is generated within the region of the hemisphere represented by that cell. This is performed by applying two additional random uniform numbers to $T(u, v)$, although their range is restricted to that of the cell under question. The probability of having chosen this sample is found directly from the p.d.f. of the cell. In Figure 6.8a the samples were generated with Jensen's technique and clearly favor those portions of the hemisphere that have strong incident radiance. Storage requirements can be reduced by discarding the p.d.f. table after the c.d.f. array is calculated. Any entry of the p.d.f. can be reconstructied directly from the c.d.f. table.

In his paper, Jensen only demonstrated his technique for diffuse surfaces. This appears to be because if the transform is fixed for the entire image and does not depend on viewing direction, then the inverse transform can be performed by a simple table lookup. The technique can only be extended to more complicated BRDFs if $T^{-1}(\theta, \phi)$ can be defined and easily computed. However, this is difficult for acquired BRDFs or those composed of multiple Phong lobes. In fact, is unclear how to handle even a simple BRDF composed of both a diffuse term and a single Phong lobe, because any specific direction (θ, ϕ) could have been generated by either the diffuse term or the Phong lobe. Therefore no inverse function can exist.

The method works well as long as the grid is fine enough to capture the essence of the incident radiance and there are enough photons collected so that the p.d.f. table is generated with sufficiently low variance. It will have trouble however if there are very small but bright regions of the sky. Although the correct general area of the sky will be focused on, many of the generated final gather rays would miss the light source.

6.2.3.1 Resource requirements

Jensen's method requires the intermediate storage of C_J cells to hold the c.d.f. As described above, the p.d.f. can be recovered directly from the c.d.f. as required. Following the original paper, the experiments were performed with a 4×8 grid and each element stored a 32-bit float for a total of 128 bytes. This storage requirement is listed in Table 6.1 where it is compared to the following algorithms.

To present a comparison of the computational cost of the methods in Table 6.2, each operation is assigned a specific cost. Projecting each photon into a specific bin can be performed using precomputed tables. This turns the operation into a 2D integer binary search and is assigned the equivalent of a single FLOP. The energy from each photon must then be added to the cell and a cumulative sum must be kept, at the cost of 2 FLOPs per photon. Each cell of the table then must be: 1) checked to see if an energy donation is required; 2) normalized to generate the p.d.f. table; and 3) accumulated to generate the c.d.f. table. Altogether, this is a cost of 3 FLOPS per cell. To generate each sample, a random number is used in a binary search of the c.d.f., at the cost of approximatly 1 FLOP. Generating a sample within a specific cell requires 5 FLOPS. The probability of the sample is evaluated by subtracting the neighboring entries of the c.d.f. table, at 1 FLOP. The final cost is approximated as $3P + 3C_J + 7N_{FG}$.

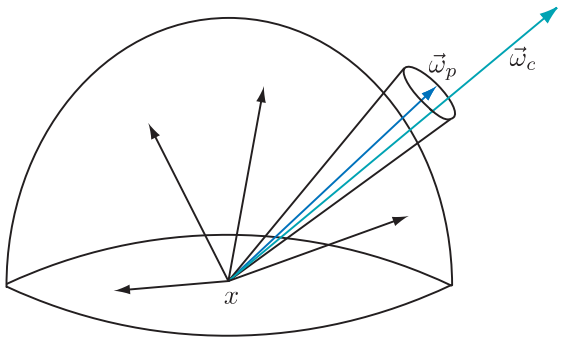


Figure 6.10: Hey and Pharr's methods generate a sample by first performing a kNN search at x, selecting a photon and constructing a cone around the direction, $\vec{\omega}_p$, that the photon arrived from. A direction within that cone, $\vec{\omega}_c$, is then selected randomly.

6.2.4 Hey's method

Hey and Purgathofer described a sophisticated method for importance sampling incident radiance using the photon map (Hey and Purgathofer, 2002). Like Jensen, they begin with a kNN search to sample incident radiance. Also as in Jensen's method, the photons are projected onto a 2D unit grid divided into cells. The paper suggests $C_H = 32 \times 32$ cells. Instead of summing the energy arriving in each cell, as Jensen did, the 3×3 circle of cells around the project direction is identified and each cell incremented by one. This table will be used to estimate the density of photon arriving from any given direction.

However, unlike Jensen's method, this table is not directly used to generate final gather rays; it plays only a secondary role. The complicated details are carefully laid out in their paper. There are two fundamental ways in which their algorithm departs from Jensen's. This first is that the actual list of photons found by the kNN search is kept and used while generating final gather directions.

The list of photons is sampled, with the probability of choosing a particular photon defined by the amount of energy it carried compared to the others. Once a specific photon p is selected, a cone is generated from x along the direction the photon arrived from, $\vec{\omega}_p$. A random direction, $\vec{\omega}_c$, within this cone is then sampled as shown in Figure 6.10.

The angle subtended by the cone is calculated using an estimate of photon density

from the 2D table. This allows the algorithm to tightly mold the generated samples to the incident radiance function. If a photon has many neighbors, its cone will subtend a small solid angle and the samples will stay close to the photon's direction. If, however, the photon is far from its neighbors it will generate samples covering a large portion of the visible hemisphere.

Importance sampling requires not only that the samples X be generated, but also that the probability of having selected X, $p(X)$, be known. Because the cones around each photon may overlap and be arbitrarily large, X could have been generated by multiple photons. Therefore, computing $p(X)$ requires a test to determine if X lies within the cones of *any* of the P photon. This test requires P dot-products per sample.

The second difference between this method and Jensen's is the use of Veach's multiple importance sampling. The number of final gather rays, N_{FG}, is divided with N_B allocated to BRDF sampling, as described in Section 6.2.1, and N_P using the strategy outlined here. This does however impose a significant additional cost, because the probability of having chosen a sample according to *both* methods must be determined for *every* sample. For each of the N_B final gather rays sampled according to the BRDF, the probability of having selected them using the photons must be computed. This can be a costly operation because, as just described, the cone around all P photons must be compared against. For a total of N_{FG} samples, this will require $N_{FG} \times P$ dot products. Unless the computations of the BRDF are very expensive this will be the dominating cost of Hey's method.

6.2.4.1 Resource requirements

Hey's technique requires more storage since it must keep the list of P photons. In addition to the 96 bytes per photon, using Jensen's compact representation, Hey's algorithm must store three 32-bit values for each photon. (The alternative would be recomputing these values before each sample is generated, an expensive strategy.) Additionally, a table of

size $C_H = 32 \times 32$ is stored as 32-bit floating point numbers. The total of 6,256 bytes is presented in Table 6.1. Although this is significantly more storage than Jensen's method requires, as long as P is not excessively large, it is not a prohibitive problem.

During the preprocess stage the photon density table is constructed, reflected radiance for each photon computed, the c.d.f. that will be used to choose each photon is built, and the footprint radius and height for every photon is computed. (See the original paper for more details.) This preprocessing alone requires $P(8 + \text{BRDF}) + 2C_H$ FLOPS.

The computational costs of generating the samples is however more significant. N_B samples are generated according to the BRDF. N_P samples are generated by picking a photon using a binary search in the c.d.f. for 1 FLOP, and then generating a sample in the cone surrounding the photon for 5 FLOPS. For all N_{FG} samples, their probabilities according to *both* the BRDF and all P photons must be computed. This costs $(\text{BRDF})N_B + 6N_P + (1 + 5P + \text{BRDF})N_{FG}$.

For the purposes of a concrete comparison, it is assumed that all three operations involving the BRDF (evaluation, importance sampling, and evaluating the p.d.f) cost 10 FLOPS. The results shown in Table 6.2 are high, over an order of magnitude higher than Jensen's method.

6.2.5 Pharr's method

In an update to his `pbrt` system, Pharr proposed, but did not analyze, a simple approximation to Hey's algorithm (Pharr, 2005). The density estimate was assumed to be constant and a cone subtending a user-specified angle was used for all photons (10 degrees was recommended). This algorithm was not expected to perform as well as Hey's but is cheaper because the photon density and the related measures need not be estimated. However Pharr's method still requires $N_{FG} \times P$ dot products to determine the probability of having chosen the generated samples, with or without multiple importance sampling.

6.2.5.1 Resource requirements

Pharr's method only stores the list of found photons, at 96 bytes per photon using Jensen's compact representation. Pharr's method also avoids all the preprocessing work of Hey's method. Further, during sample generation the photon is chosen randomly without regard to the energy it brought with it, eliminating the need to build a c.d.f. table of the photons. The computational cost is therefore reduced to (BRDF)$N_B + 5N_P +$ $(1 + 5P + \text{BRDF})N_{FG}$.

6.3 Combined importance sampling

The methods of Hey and Pharr improved upon that of Jensen in two ways: 1) they incorporated information about non-diffuse BRDFs into their sampling using multiple importance sampling; and 2) by sampling from within cones around the photons they achieve a tighter fit of the generated samples to the limited knowledge about the incident radiance function. In exchange for these improvements they incurred a large computational cost. In this section I present a novel algorithm that incorporates knowledge of glossy BRDFs without a direct implementation of multiple importance sampling, keeping costs low while performing nearly as well as Hey and Pharr's methods for the scenes described in Chapter 1.

In multiple importance sampling, the samples are generated according to two separate sampling strategies and the balance heuristic required computing a joint probability by consulting all M p.d.f.s. These requirements taken together require that information for both sampling strategies be stored throughout the sampling process and that many of computations be performed. Combined importance sampling, however, integrates the strategies together as a pre-process, leaving a single strategy to generate all N_{FG} samples and a single p.d.f. to evaluate (Steinhurst and Lastra, 2006). The result is the same but

can only be performed when it is possible to directly combine the M p.d.f.s. As will be shown shortly, this approach is simple when the p.d.f.s are expressed in a tabular form.

The balance heuristic (Equation 6.6) can be substituted into the MIS estimator (Equation 6.5) to obtain the estimator used by Hey and Pharr (Equation 6.9).

$$F_{M,N} = \frac{1}{N} \sum_{i=1}^{M} \sum_{j=1}^{N_i} \frac{f(X_{i,j})}{\sum_k \frac{N_i}{N} p_k(X_{i,j})} \tag{6.9}$$

Observe that this equation uses multiple sampling strategies. When restricted to two strategies, BRDF and photon-map-based incident radiance, the denominator can be rewritten as

$$\hat{p}(x) = \frac{N_B}{N} p_B(x) + \frac{N_P}{N} p_P(x) \tag{6.10}$$

where $p_B(x)$ and $p_P(x)$ are the p.d.f.s for selecting according to the BRDF and photon tables respectively.

$$F_N = \frac{1}{N} \sum_{i=1}^{N} \frac{f(X_i)}{\hat{p}(X_i)} \tag{6.11}$$

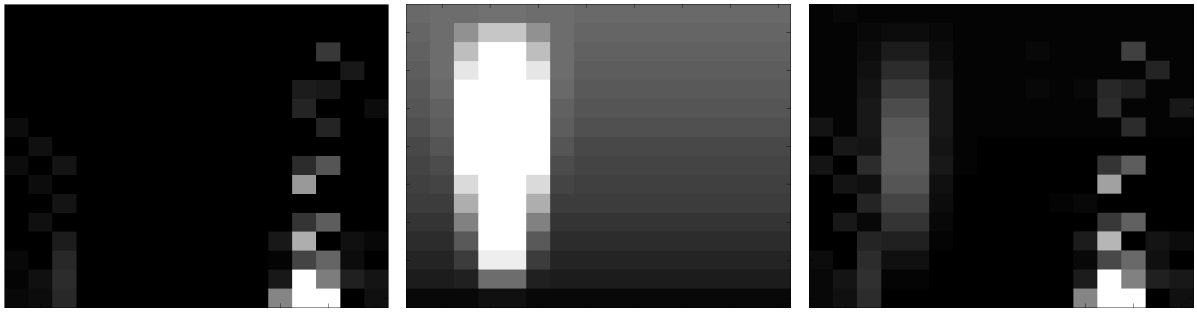
Note that $\hat{p}(x)$ is the weighted average of $p_B(x)$ and $p_P(x)$. This gives the reduced estimator (Equation 6.11) the appearance of the standard importance sample estimator, presented earlier in this chapter. Most importantly, there is a *single* sampling strategy and only a *single* evaluation of $\hat{p}(x)$ is required.

This new strategy can be easily applied to final gather ray generation using the photon map by building upon Jensen's method. The following explanation is outlined in Algorithm 6.3. The starting point is a p.d.f. table constructed based on the incident radiance exactly as was performed in Jensen's preprocessing. The transformations for the diffuse BRDF, $T(u, v)$ and $T^{-1}(\theta, \phi)$, are used, regardless of the true BRDF.

A second p.d.f. table is generated according to the following rule. For each cell, a single ray, $\vec{\omega}_c$, is generated through the middle of the corresponding portion of the hemisphere. The BRDF is then evaluated, $f(x, \vec{\omega}_c, \vec{\omega}_o)$, and stored in the cell. Cells with

Pre-condition: pdfLiTable is computed as described in Algorithm 6.1
// Compute evaluatedBRDFTable
for each cell $[u, v] \in$ evaluatedBRDFTable **do**
$\vec{\omega}_c \leftarrow$ centerDirectionOfCell(u, v)
evaluatedBRDFTable$[u, v] \leftarrow f(x, \vec{\omega}_c, \vec{\omega}_o)$
totalReflectance $\leftarrow$ totalReflectance $+ f(x, \vec{\omega}_c, \vec{\omega}_o)$
end for
// Donate a small portion of the total reflectance to any empty cells
donation $\leftarrow 0.05 \times$ totalReflectance$/C_J$
for each cell $[u, v] \in$ evaluatedBRDFTable **do**
if evaluatedBRDFTable$[u, v] == 0$ **then**
evaluatedBRDFTable$[u, v] \leftarrow$ donation
totalReflectance $\leftarrow$ totalReflectance $+$ donation
end if
end for
// Normalize the reflectance table, producing the p.d.f.
for each cell $[u, v] \in$ pdfBRDFTable **do**
pdfBRDFTable$[u, v] \leftarrow$ evaluatedBRDFTable$[u, v]$/totalReflectance
end for
// Average the two p.d.f. tables
for each cell $[u, v] \in$ pdfCombinedTable **do**
pdfCombinedTable$[u, v] \leftarrow$ (pdfLiTable$[u, v]$ + pdfBRDFTable$[u, v]$)$/2$
end for
// Construct a combined 1D c.d.f. array
cdfArray$[0] \leftarrow 0$
for $i = 1$ to $C_J + 1$ **do**
cdfArray$[i] \leftarrow$ cdfArray$[i - 1]$ + pdfCombinedTable$[i]$
end for

Algorithm 6.3: The preprocessing stage for combined importance sampling begins with that of Jensen's (Algorithm 6.1). A second p.d.f. table is constructed and the results averaged to form a single p.d.f. table to sample from. A visual representation in shown in Figure 6.11.



(a) Incident radiance p.d.f. (b) BRDF p.d.f. (c) Combined p.d.f.

Figure 6.11: Combined importance sampling generates as a preprocess a tabular p.d.f. for both the incident radiance and the BRDF. These two p.d.f.s are then combined by averaging the values in each cell independently. Only one strategy for generating samples is therefore required and $p(X)$ is simple to compute. An example of the samples generated using this p.d.f. is shown in Figure 6.8c.

	Storage	Bytes
Jensen	$4C_J$	128
Combined	$4C_J$	128
Pharr	$96P$	960
Hey	$4C_H + 128P$	6,256

Table 6.1: The intermediate storage cost for each of the global importance sampling techniques. We assume that P photons were gathered for the radiance estimate, and that there were N_B samples generated according to the BRDF and N_P samples generated according to the incident radiance for a total of N samples. Jensen's and the combined technique require storage of tables with C_J cells, while Hey used tables of size C_H.

non-positive values are given a small positive value and finally, the table is normalized in order forming a p.d.f. table.

Following Equation 6.10, the two p.d.f. tables are then directly averaged together to form a single p.d.f. that is used for sampling. This process is shown in Figure 6.11. The combined p.d.f. table is converted into a c.d.f. table as before. Samples are then generated using the same process as in Algorithm 6.1. Only a single operation is required to determine the probability of having generated a particular sample.

6.3.0.2 Resource requirements

The combined importance sampling method has costs similar to that of Jensen's. The storage requirements are exactly the same, as is the cost to generate samples. The preprocess has additional costs however. They mostly arise from the evaluation of the BRDF for each cell; creating the additional p.d.f. table and averaging the two together are both relatively cheap. The total cost formula is $3P + (3 + \text{BRDF})C_J + 6N_{FG}$.

6.3.0.3 Discussion

Combined importance sampling is simple and efficient. It will be shown in the next section that it is effective for the scenes where the BRDF is not too glossy or the incident radiance too sharp. These two limitations both arise from the use of a low-resolution table to express the p.d.f. and are explained using the same rationale as was used for Jensen's technique: samples within a single cell are generated purely randomly, so they can miss a particularly small feature in the integrand. Although it will consequently not be able to capture highly specular surfaces perfectly, these surfaces are often sampled specially in path tracers (Shirley, 2000). Experiments show that a 4×8 table is adequate for *glossy* surfaces. (Those equivalent to a Blinn-Phong cosine lobe of n=16.)

An important benefit of this approach, beyond the reduction in computational cost, is that the BRDF does not have to be sampled, only evaluated. Therefore neither $T(u, v)$ nor $T^{-1}(\theta, \phi)$ need be known, or easy to evaluate. As was discussed in Section 6.2.1, those functions can be difficult to specify and evaluate for acquired reflectance functions.

6.4 Results

The effectiveness of the combined importance sampling is demonstrated in this section in two ways. First, an extensive point-response analysis was performed to carefully compare both the variance and the convergence of the techniques in a controlled experiment. Based

	Computation	FLOPS
Jensen	$3P + 3C_J + 7N_{FG}$	1096
Combined	$3P + C_J(4 + \text{BRDF}) + 7N_{FG}$	1,448
Pharr	$N_B \times \text{BRDF} + 5N_P + N_{FG}(\text{BRDF} + 5P + 1)$	51,850
Hey	$(P(8 + \text{BRDF}) + 2C_H) +$ $(N_B \times \text{BRDF} + 6N_P + N_{FG}(\text{BRDF} + 5P + 1))$	55,748

Table 6.2: The computational cost for each of the global importance sampling techniques. See the text for details on the assumptions behind the cost of operations. The BRDF evaluations are separated out so that the impact of complicated BRDF models is apparent. Hey's algorithm is by far the most expensive in terms of both storage and computation. Both Pharr's and Hey's require $N \times P$ dot products and comparisons to compute the probability of the generated samples. For even modest values of N and P their product gives rise to a dominating cost.

on the positive results of these experiments, and the cost advantages described in the previous section, combined importance sampling was chosen for implementation in the architecture presented in Chapter 7. Images generated by the architecture simulation are therefore presented here to demonstrate the benefit when applied in the proposed context.

6.4.1 Comparison of variance

In order to evaluate these importance sampling techniques with realistic incident radiance, a series of acquired lightprobes was used (Debevec, 1998). Each light probe contains a high dynamic range and low distortion view of one hemisphere. The construction and consultation of the photon map was simulated by choosing pixels of the light probe proportional to their energy and solid angle subtended. This simulation of photon tracing and a kNN search was verified against an open-source photon map implementation (Pharr and Humphreys, 2004). Although the combined importance sampling technique is directly extensible to complicated analytical and acquired BRDFs, this experiment was restricted to a BRDF with a simple combination of pure diffuse and a single modified Phong cosine lobe. This allowed direct comparison of the performance of the various

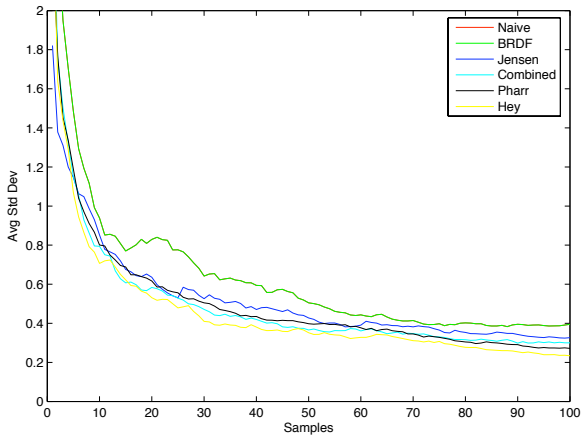
techniques using the photon map to an analytical importance sampling of the BRDF.

One difficulty in comparing these algorithms, is the selection of specific parameters that must be supplied by the user. In most cases the values used were those suggested by the original authors. The most sensitive was the ratio of the number of cells, C_J and C_H, to the number of photons gathered. Jensen's paper found $C_J = 4 \times 8$ cells to be good for a variety of scenes where only 50 photons were gathered. This however works out to fewer than 2 photons on average per bin, which causes a high variability in the estimated p.d.f. Performing a kNN query for 100 photons was found to perform better in these experiments, and was convenient as the photon gathers in the simulator also use 100 photons per search.

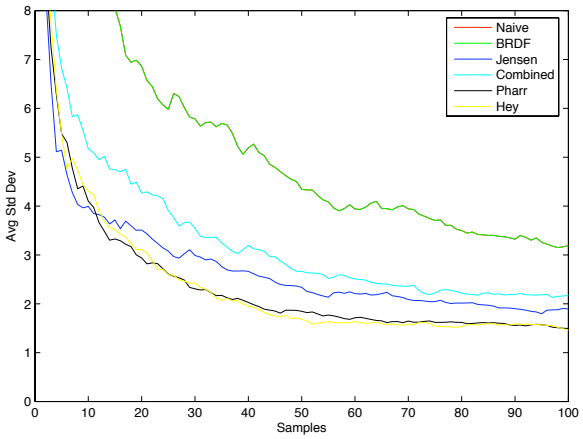
Figure 6.12 shows the average standard variation across 500 trials for the various importance sampling techniques for 2 different lightprobes and 3 different surfaces: pure diffuse, a sharp Phong lobe, and a mixture of those two. The combined importance sampling technique was compared with naive sampling, analytical BRDF sampling, Jensen's method, and both Hey and Pharr's methods. The exponent of the glossy cosine lobe was set to 16, which is quite sharp while remaining glossy and not highly specular.

In the scene illuminated with the beach lightprobe, the viewer was positioned so that the reflected glossy lobe faced the ground away from the sun. In the kitchen scene, the viewer looked down at a glossy table such that the reflection was in the general direction of the bright window. An algorithm is said to perform better than another if fewer samples are required to achieve the same variance. None of the importance sampling techniques exhibited bias.

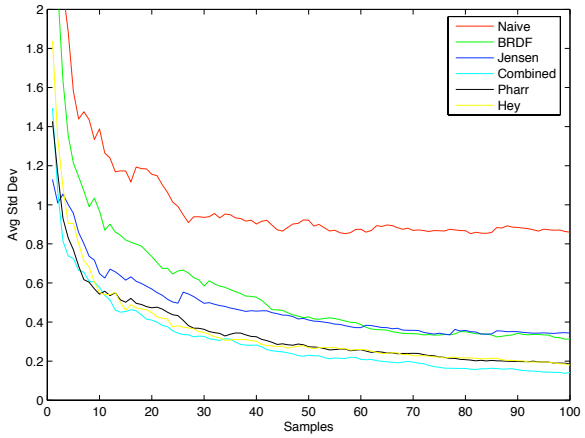
As expected, Hey's algorithm performs best under nearly all conditions because it is able to focus so effectively on areas of high contribution in the photon map while using multiple importance sampling incorporates knowledge from the BRDF. While only approximating Hey, Pharr's method achieves almost similar performance under standard lighting conditions, a room lit by a window as opposed to a pinhole. Its performance



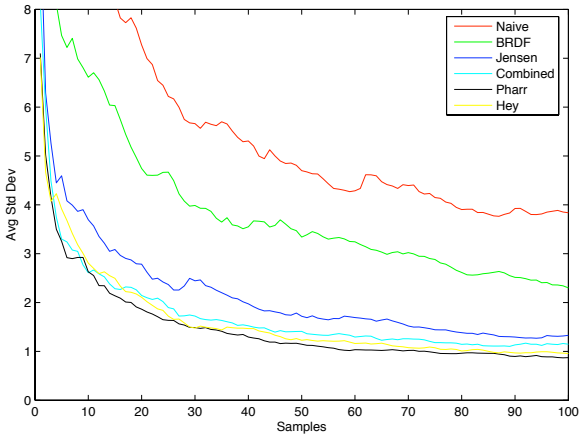
(a) Beach: Pure Diffuse



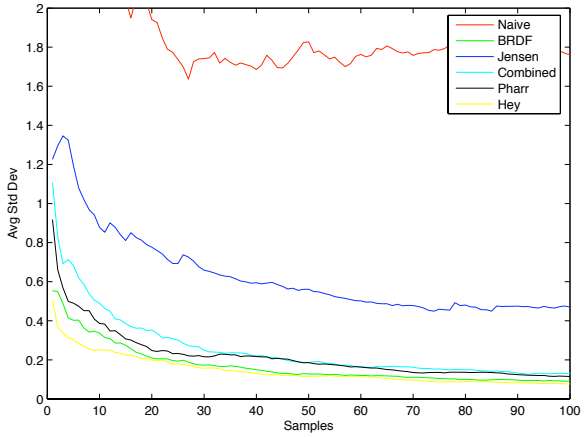
(b) Kitchen: Pure Diffuse



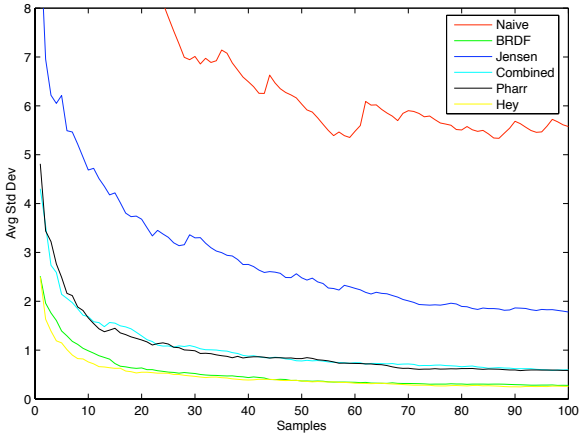
(c) Beach: 1/2 Glossy



(d) Kitchen: 1/2 Glossy



(e) Beach: Pure Glossy



(f) Kitchen: Pure Glossy

Figure 6.12: The convergence rates for the beach and kitchen lightprobes for three different surface BRDFs. The convergence rate was established by averaging the standard deviation from the expected result for 500 trials. Particularly interesting cases are (c) and (d) where information from both the BRDF and the photon map's approximation of incident radiance allows for better convergence than either alone.

would drop if the incident radiance function has higher frequency components. The combined importance sampling technique provides performance comparable to Pharr's and is always much closer to Hey's than to Jensen's while having an implementation cost close to Jensen's.

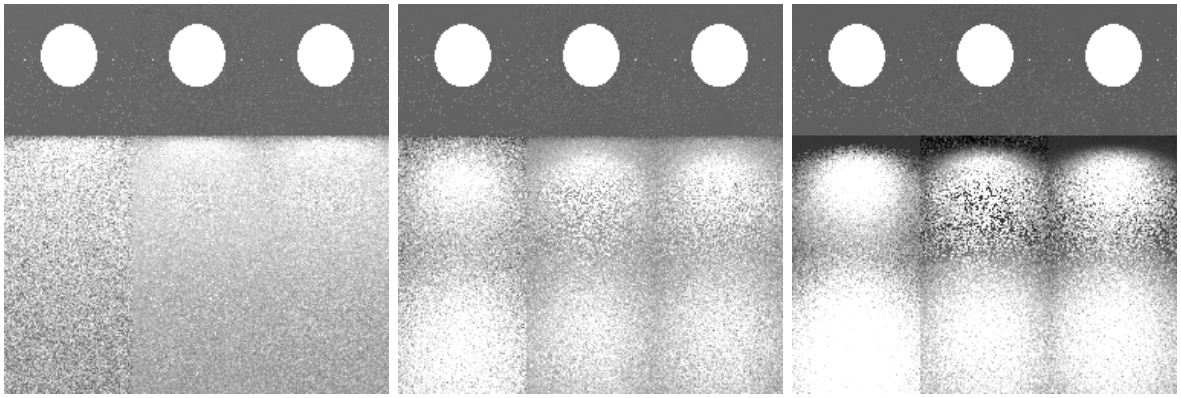
6.4.2 Generated image quality

Combined importance sampling was shown to have competitive performance, significantly cheaper computational cost, and a simple implementation. For these reasons it was selected for inclusion in the architecture presented in this dissertation (see Chapter 7). Comparing the results of the functional simulator allows us to compare the techniques as used to generate actual images.

All three test scenes shown in this section are illuminated almost entirely by indirect light. This will require the importance sampling to perform well to achieve acceptable results. In Chapter 7 it will be shown that importance sampling works well in scenes with less dramatic lighting as well.

The dome scene, used in the experiments in Figure 6.13, is a square surrounded by a dome. Two spot lights are pointed at the dome, one straight above and one in front. The light then reflects towards the square, acting as small area light sources, although purely indirect. This scene was imaged nine times, with varying surface reflectance properties and importance sampling methods. From each generated image a thin strip was removed, with the results grouped by surface reflectance.

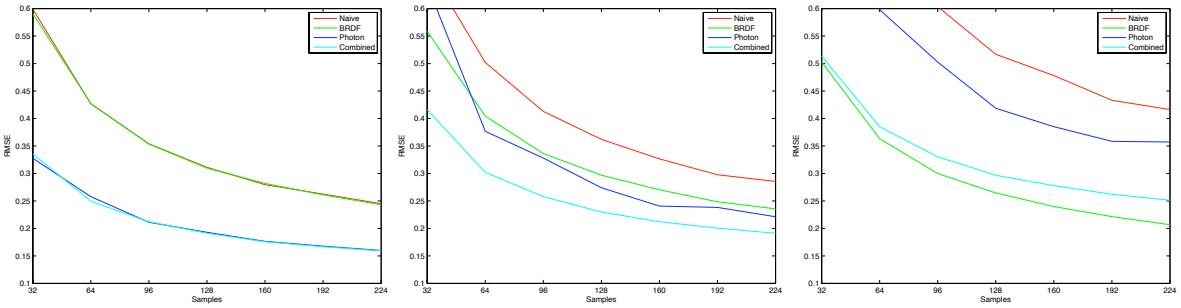
When the square is fully diffuse, the combined importance sampling method is indistinguishable from Jensen's. When the square is fully glossy, the combined importance sampling method performed almost as well as analytically sampling the BRDF. For more general surfaces, that are partially diffuse and glossy, combined importance sampling is the best option. These visual observations are confirmed by examining the variance



(a) Pure Diffuse (128 Gathers) L→R: BRDF, Jensen, Combined

(b) 1/2 Glossy (128 Gathers) L→R: BRDF, Jensen, Combined

(c) Pure Glossy (128 Gathers) L→R: BRDF, Jensen, Combined



(d) Pure Diffuse

(e) 1/2 Glossy

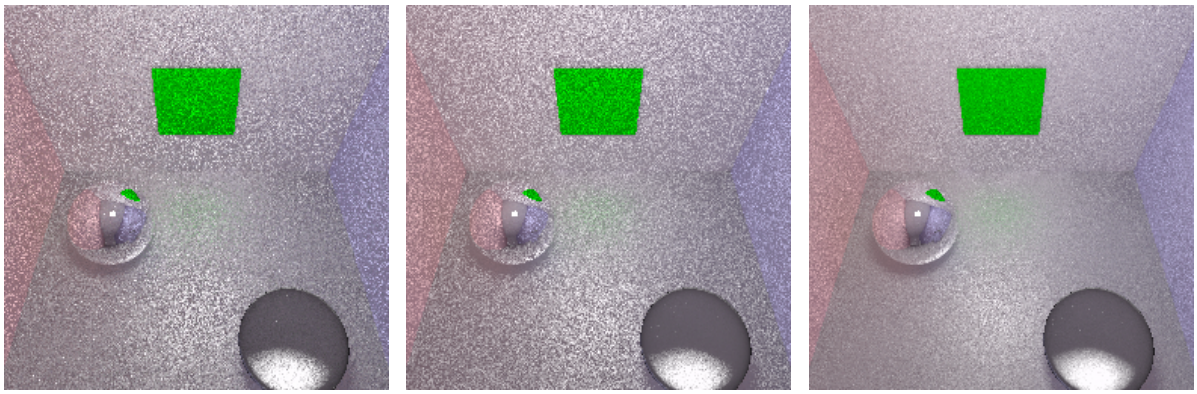
(f) Pure Glossy

Figure 6.13: The nine experiments described in the text are visualized by extracting vertical strips and group into a montage by importance sampling strategy in Figures (a) through (c). Combined importance sampling is visually the best overall choice, which is confirmed by examining the observed variances in Figures (d) through (f).

graphs in Figures 6.13d through 6.13f.

A modified Cornell box with a half-diffuse and half-glossy floor is shown in Figure 6.14. This set of images shows the tradeoffs a designer must make when using importance sampling. An image of equivalent quality can be generated using fewer samples, about one quarter in this case, or a significantly better image can be generated with the same number of final gathers.

Combined importance sampling supports more complicated BRDF functions, as long as they do not contain particularly sharp peaks. In Figure 6.15 the curved foreground object has a reflectance defined by a three lobe Lafortune model fitted to acquired data. The benefits of importance sampling do not rely on the scene containing a glossy surface,

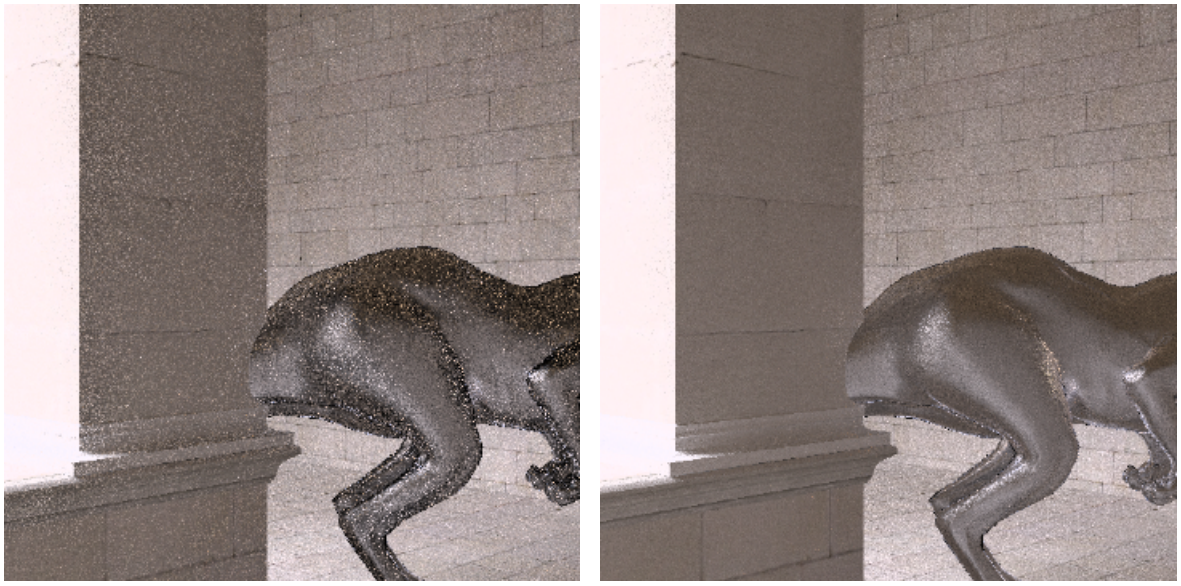


(a) Combined 32 Gathers

(b) Cosine weighted 128 Gathers

(c) Combined 128 Gathers

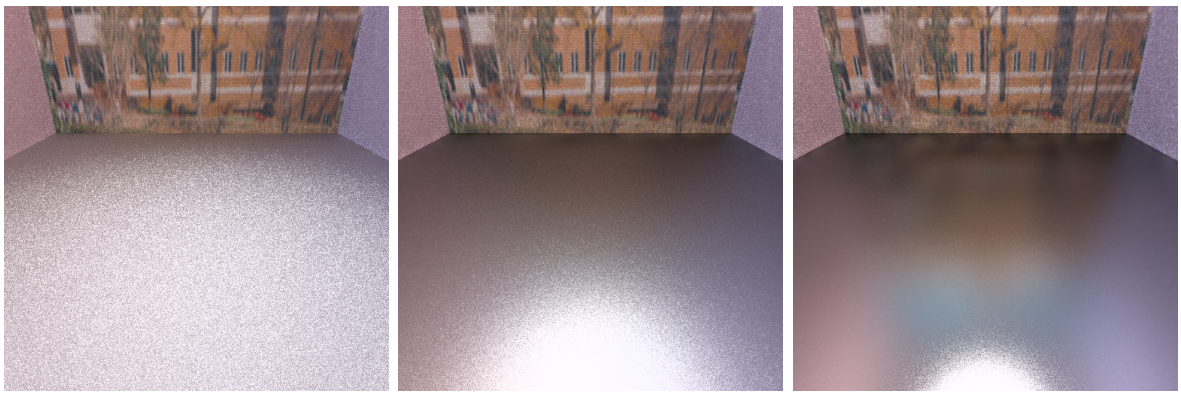
Figure 6.14: The floor of this modified Cornell box is half diffuse and half glossy. The lighting is entirely indirect, a spot light aimed at the ceiling. For low variance images in this scene, final gather rays must be selected on the basis of both incident radiance and the local surface reflectance. The combined importance sampling method produces an image of similar quality using one quarter of the samples. Alternatively, a superior image is generated with the same sample budget.



(a) Cosine weighted sampling 33 Gathers

(b) Combined importance sampling 33 Gathers

Figure 6.15: In this view of the Sponza atrium, the light is purely indirect, the walls are diffuse and the curved object in the foreground has a three lobe Lafortune BRDF fit to acquired data. The curved geometry of the foreground object benefits greatly from importance sampling according to the BRDF. Presented with the diffuse walls, combined importance sampling is the exact same as Jensen's algorithm.



(a) $n = 1$ (b) $n = 16$ (c) $n = 128$

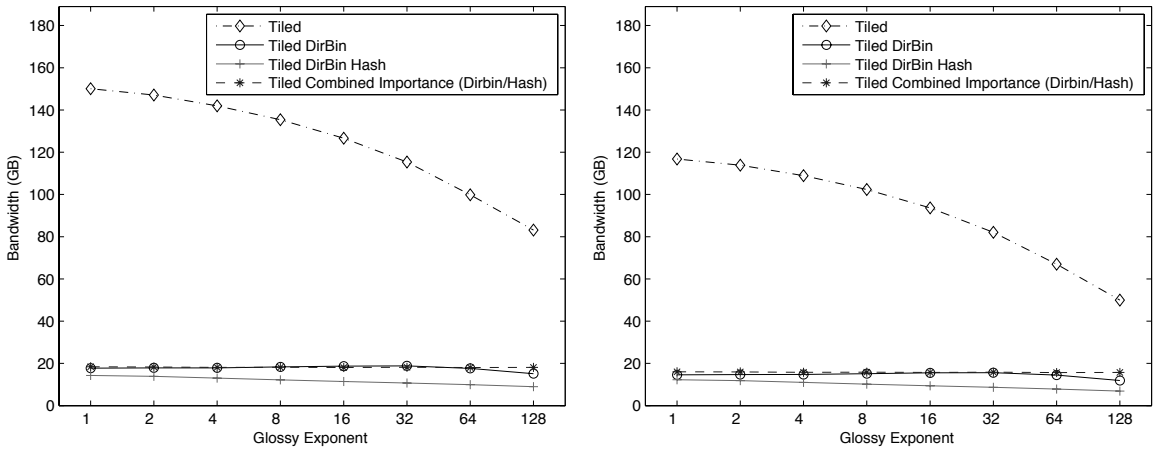
Figure 6.16: The interaction of importance sampling with query reordering was investigated using a modified Cornell box. A glossy floor was created with a BRDF that is a combination of Lambertian and a single Phong cosine lobe of the indicated exponent. During experiments, the camera was zoomed so that the floor filled most of the image.

of course. Presented with a diffuse surface, such as the walls in the same scene, combined importance sampling is the exact same as Jensen's algorithm.

6.5 Interaction with gather reordering

The non-uniform sampling of final gather ray directions that arise from importance sampling not only reduces the variance of the computed reflected radiance but also changes the spatial distribution of the photon gathers that must be performed. This will have a direct impact on bandwidth requirements, and in this section we examine the magnitude of this effect and its interaction with both query reordering and irradiance caching.

To clearly examine the effect of a glossy surface on these other bandwidth reducing techniques, a modified version of the Cornell box scene was used that with a viewpoint focused on a glossy wooden floor. The BRDF in this experiment is composed of two separate terms. The first is a simple uniform Lambertian. The second is a single Phong cosine lobe. The resulting images are shown in Figure 6.16 for several values of n, the glossy exponent of the Phong illumination model.



(a) Without Irradiance Caching (b) With Irradiance Caching

Figure 6.17: As the exponent of the glossy lobe is sharpened the required bandwidth for naive photon mapping is reduced. This effect is minor or non-existant for reordering techniques that are already successfully exploiting coherence. For a fair comparison, the naive and direction-binned techniques are shown here with analytical BRDF sampling. (Modified Cornell Scene, 16×16 tiles, *kd*-tree 128KB cache, 128B cachelines, $\epsilon_{max} = 0.025$)

Surfaces with a higher glossy exponent were found to require less bandwidth for the naive ordering than those that are more diffuse (Figure 6.17a). This is because the sampled final gather rays are clustered in a small portion of the hemisphere, corresponding to the expected contribution.

The effect is less pronounced, or even non-existant, for the higher quality reordering algorithms. When the photon queries have already been reordered, however, there is no reduction in bandwidth as the glossy exponent increases. This is because the reordering have already captured most of the available coherence.The pattern held true both with and without irradiance caching (Figure 6.17b).

Although glossy surfaces require less photon gather bandwidth than uniform diffuse surfaces, the improvement due to importance sampling is minor compared to that obtained from query reordering. This experiment shows that query reordering is compatible with importance sampling and irradiance caching, exploiting the additional potential coherence among the photon gathers.

6.6 Conclusion

Combined importance sampling is a simple global importance sampling algorithm that reduces variance in generated images for scenes demonstrated at a low cost. It is compatible with photon gather reordering and irradiance caching. It therefore can be incorporated into a system which utilizes those techniques to reduce memory bandwidth.

The low cost is due to a preprocessing stage where multiple p.d.f.s are combined together, leaving a single sampling strategy to be used during final gather ray generation. Although combining p.d.f.s is not always possible, it is easy in the application presented in this chapter because the p.d.f. of incident radiance when captured by a kNN search in a photon map is naturally expressed in a table. Not all functions, such as sharp BRDFs, are well represented this way however. If two more general p.d.f.s were expressed in different ways, varying basis functions for example, that could prevent the application of combined importance sampling.

CHAPTER 7

ARCHITECTURE

The most significant impediment to interactive image generation using the photon mapping algorithm is the memory bandwidth requirement of photon gathering. The previous chapters of this dissertation have presented several techniques that were shown to be successful at reducing the memory bandwidth requirements: photon gather reordering, irradiance caching and combined importance sampling. Irradiance caching reduces the number of final gathers that must be performed. Importance sampling reduces the number of final gather rays and subsequent photon gathers that must be generated. Finally, photon gather reordering increases the coherence of the memory request stream generated while servicing those photon gathers that remain to be performed. Together these techniques will reduce the memory requirements of image generation by almost two orders of magnitude.

The algorithms and data structures to accomplish these reductions were designed with care to neither impose significant additional computation nor hide the natural parallelism of the original photon map algorithm. Further, it was shown that the techniques are compatible; not only can they be implemented in a single rendering system but their benefits build on each other. In this chapter, the techniques are combined into a single proposed hardware architecture that could be practically implemented by 2010. An implementation small enough to fit in a standard desktop PC will be shown via simulation to have sufficient performance to support the interactive rendering of scenes with difficult indirect illumination at 30 frames per second.

In the first section of this chapter, the goals of the architecture and the implementation

constraints imposed by current semiconductor technology are revisited. The architecture is then described at a high level before a careful examination of each component. The required bandwidths, computations and internal storage are shown to be well within the expected performance of the targeted implementation technology. A functional simulation of the architecture is presented and used to verify both the correctness of the architecture and evaluate the expected performance under a variety of conditions. Finally, scalability, deadlock and load balancing are addressed along with limitations and potential design alternatives.

7.1 Goals and constraints

It is the goal of this dissertation to present a feasible hardware architecture for interactive image generation using the photon mapping algorithm. Chapter 2 established an overall vision for the system, based on reducing overall system complexity and cost. From this vision, the following goals are obtained: 1) the entire system will be packaged as one or more expansion cards that can be inserted into a standard commodity workstation; 2) no more than one major custom ASIC design will be required; 3) no inter-chip communication will be allowed; and 4) only semiconductor technologies that are expected to be commercially available in the next three years should be used.

A standard commodity workstation will be used as a host. It will handle all the generic tasks such as program loading and execution, input/output, and management of the hardware system. This frees the hardware architecture to focus on the tasks directly related to image generation. This arrangement is typical for both consumer graphics cards, such as the GeForce 6800 (Montrym and Moreton, 2005), and also large high-end research systems, such as PixelFlow (Eyles et al., 1997). Limiting the implementation to a small number of expansion cards reduces the overall system complexity. Whole classes of subsystems such as a power supply, chassis, and host interface do not have to

be designed or tested.

Each custom ASIC that must be designed for the system represents a large development and testing cost. This is particularly true of high performance designs that use recent semiconductor technologies. To limit this cost, the architecture will use only a single custom chip design. Although the development and testing of a design is expensive, the per unit cost is relatively low. Therefore, multiple instances of the chip will be used together for higher performance. This provides greater flexibility in deployment. Models offering different performances can be constructed from the same components at different price points.

There are several advantages to prohibiting inter-chip communication. The first is that the majority of the rendering chip's communication resources can be dedicated to high speed memory links. The second advantage is that the expansion cards do not need to support a bus or routing system between the chips. Instead, a simple tree communication structure with the host will suffice. System testing is also easier, as each chip performs its duties in isolation from the others. Finally, this lack of communication removes one of the largest barriers to system scalability in terms of rendering chips. This last aspect is addressed in more detail in Section 7.4.3.

It will be shown in Section 7.3.4 that the bulk of the computations in this architecture are highly parallel, regular, and exhibit few control flow dependencies, thus permitting deep pipelining. This allows the resources on the chip to be allocated to processing elements instead of control, routing and large memories. These are also the characteristics of commodity GPUs which enable them to perform so much useful computation on a single chip, as compared to a CPU. In Section 2.2 it was shown that it is a conservative estimate that GPUs will be performing 500 GFLOPS by 2010[1].

Even without photon gather reordering, irradiance caching, or combined importance

[1]Recall from Chapter 2 that the total number of FLoating point OPerations to perform an operation is abbreviated as FLOPs, while the rate of FLOPs over time is denoted as FLOPs per second, FLOPS.

sampling, generating 512×512 images of Sponza Atrium at 30 FPS will be shown, in Section 7.4, to require 7.4 TFLOPS. This burden can be handled by less than sixteen chips, which can fit on a small number of expansion boards.

It has long been lamented that the memory bandwidth available to a single chip is increasing at a much slower rate than the computational performance of that same chip (Dally and Poulton, 1998). High end commodity chips such as GPUs are expected in 2010 to have a usable bandwidth of at least 90 GB/s (see Section 2.2). The naive photon mapping algorithm, as originally described in Chapter 2, has been shown to require 11 TB/s to generate 512×512 images of the Sponza atrium at 30 FPS. Dividing that bandwidth across twelve dozen chips is simply not feasible for a workstation sized machine. The techniques presented in this dissertation are therefore required for a feasible architecture.

Because memory bandwidth is the limiting factor, the description and initial analysis of the architecture concentrates on that. Once it has been established how many tiles per second can be performed on a single chip, using the 90 GB/s permitted, it is then verified that the corresponding computation and internal storage for the ray casting, irradiance caching, importance sampling, and the photon gathers themselves are reasonable. During initial discussions, this chapter will assume that the target output is 30FPS for 512×512 images with a single sample per pixel.

Performance is, of course, a function of many variables. Some are set by the users, some fixed by the architecture as described here, and some depend on the exact scene. Table 7.1 defines the notation used throughout the rest of the chapter to discuss these costs. Nominal values are shown with references to where they are discussed. Those values that are observed characteristics of certain scenes, such as the proportion of pixels that can use irradiance caching, are given as an expected range with a conservative value used during calculations unless otherwise noted.

Notation	Description	Nominal Value	Reference
P	Number of pixels in a tile	16×16	Chapter 3
T	Number of tiles in (512×512) image	1024	Chapter 1
F	Number of frames per second	30	Chapter 1
N_{FG}	Number of final gather rays	33-**100**–200	Chapter 2
k	Number of photons searched for	100	Chapter 2
N_S	Number of shadow rays	8	Chapter 2
C_J	Number of importance sampling bins	4×8	Chapter 6
N_{PM}	Number of photons in indirect map	**.5**–6×10^6	Chapter 2
R_{IC}	Ratio of pixels using irradiance cache	0–**60**–95%	Chapter 5
R_{FG}	Ratio of pixels using full final gather	100%-R_{IC}	Chapter 5
L_{IC}	IC records interpolated per pixel	4	Section 7.3.3
C_{B^1}	Partial evaluation of f_r (Stage 1)	12 FLOPs	Section 7.3.2
C_{B^2}	Partial evaluation of f_r (Stage 2)	10 FLOPs	Section 7.3.2
C_{RayCast}	Intersecting ray with scene	330 FLOPs	Section 7.3.2
C_{G}	Gathering the k nearest photons	2,389 FLOPs	Section 7.3.4
S_B	Size of partially evaluated BRDF	13 bytes	Section 7.4

Table 7.1: The notation used throughout the rest of the chapter to discuss the cost of photon mapping in terms of both bandwidth and FLOPs. Nominal values are shown with references to where they are discussed. Those values that are observed characteristics of certain scenes, such as the proportion of pixels that can use irradiance caching, are given as an expected range with a conservative value in bold that is used during calculations unless otherwise noted.

7.2 System overview

Figure 7.1 shows the proposed system at a high level. The host machine contains one or more expansion cards. Each card contains up to four identical and independent custom designed chips, each with its own matching high-speed commodity memory. A typical implementation would provide 256 MB of memory to store a replication of the scene geometry and the photon map. The rendering chips on each card are connected to a shared PCI-Express bridge. The bridge allows for broadcasting information from the host to all chips as well as for individual responses from each chip back to the host. This simple board design does not, however, allow for communication between rendering chips, communication is solely with the host.

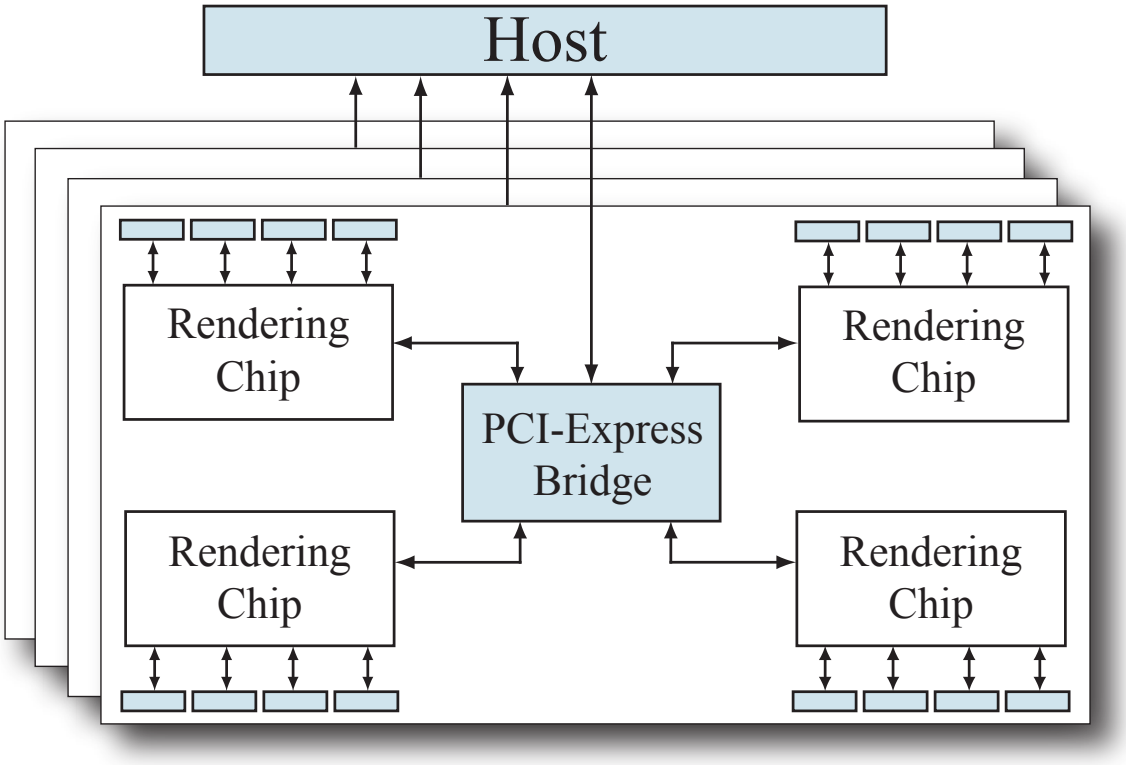


Figure 7.1: A host can have multiple rendering boards for greater performance. In addition to constructing the photon map, the host is responsible for allocating tiles to the rendering chips. Each rendering chip is independent and can not communicate with another. Each chip is provisioned with dedicated memory that holds a replication of the scene geometry and photon map, as broadcast from the host. As the tiles are finished, the resulting image is returned to the host where all tiles are assembled and displayed using a conventional graphics device.

7.2.1 Duties of the host

The duties of the host fall into two categories. Before rendering begins, the host performs all necessary pre-processing and hardware configuration. The scene geometry is broadcast to each rendering chip where it is stored in the memory for use during ray tracing. The host is also responsible for the generation and broadcast of the photon map. Additionally, the computation of R_{avg}, as described in Section 5.3, for irradiance cache calculations is performed by the host.

During interactive rendering, the host divides each image into fixed size tiles and

dynamically assigns the tiles to rendering chips as they become free. When a tile has been completed, the partial image is returned to the host and a new tile is assigned. Each chip maintains a small queue of tiles to prevent stalls waiting for the host. The tiles are then assembled and displayed by the host using a conventional graphics system.

7.2.2 Interconnect bandwidth

The interconnect between the card(s) and the host does not need to have a particularly high bandwidth. For the retained mode operation envisioned for this system, the scene geometry and photon map are broadcast to all chips only infrequently, as the scene changes. During active rendering, the host simply instructs each chip which tile it is to compute. This takes 2 bytes per tile for a total of $2TF = 60$ KB/s. Each tile is then returned to the host as a high dynamic range image, with each pixel represented using 16-bit RGB components. This requires a total bandwidth to the host of $PTF(3\times2 \text{ bytes}) = 45$ MB/s. The current industry standard, PCI-Express 1.1, is able to provide a bandwidth of 4 GB/s to a card, while the new 2.0 specification will obtain up to 8 GB/s (PCISIG, 2007), far exceeding the needs of this architecture.

7.3 The rendering chip

The internal structure of the rendering chip is shown in Figure 7.2. The only communication link provided is to the host via the PCI-Express bridge. Each chip has dedicated off-chip memory divided into four banks to maximize bandwidth while requiring no more pins on the package than currently in common use. Each chip maintains a replication of the entire scene geometry and photon map. This allows a tile to be rendered without communication to other processors. A typical implementation will provide 256 MB of memory.

The on-chip communication between the components is point-to-point, buffered by

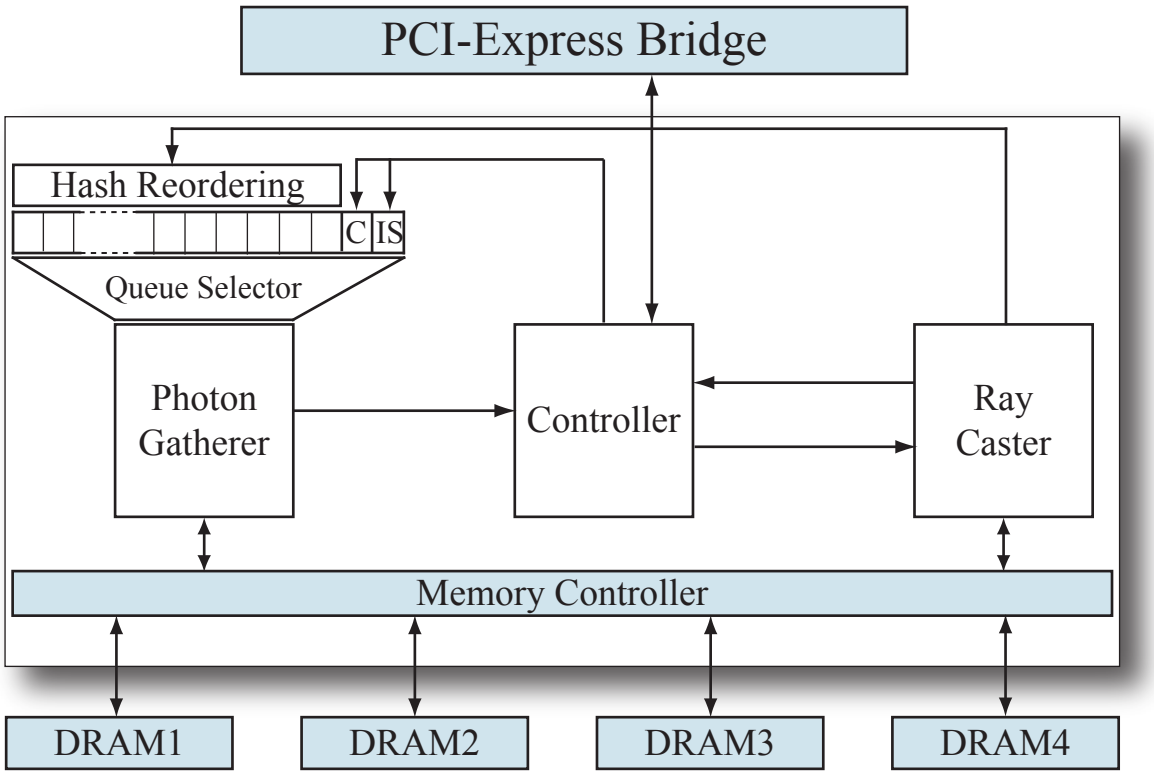


Figure 7.2: Each rendering chip has its own associated commodity DRAM memory large enough to store both the scene and the photon map. The three major portions communicate using packets tagged with the address of the destination pixel, allowing internal reordering and out of order processing.

small queues, eliminating the need for routing devices or a bus structure. Inspired by the GI-Cube architecture (Dachille and Kaufman, 2000) the communication is partitioned into packets. A packet represents a request to the component receiving it; examples are a ray to be traced, an indirect photon gather to be performed, or a partial result to be accumulated for output. Each packet is tagged with the pixel address it will eventually contribute to. The allows for a component to process the incoming stream of requests in the most efficient order.

This section begins by following the life cycle of a tile as it is rendered by the chip, concentrating on the overall structure of the rendering chip and the division of labor between the three main components. Each component is then examined in turn, with a detailed analysis of the computation and internal storage requirements. Further, the

composition and size of the various packet types between each component are presented, allowing for a careful consideration of the inter-component bandwidths.

7.3.1 The rendering of a tile

The controller maintains a small queue of tiles that have been assigned to this chip by the host. When the chip is free to begin a new tile[2], the ray caster is told to generate and trace the eye rays through the scene. The ray tracing is terminated at the first intersection with a surface that is not highly specular. The ray caster partially evaluates the BRDF at the intersection point x as both location and viewing direction are now known. Any required texture lookups, for example, can be performed at this time. This reduces the amount of shading information that must be passed along. Direct illumination is computed at this time using shadow rays generated from x towards the light sources.

This intersection data is then returned to the controller, where it is stored for future use. The controller issues two separate requests to the photon gatherer. The first is for a photon gather at x in the caustic map[3]. The second request is for a special purpose photon gather in the indirect map. This gather serves two purposes. The first is to retrieve a value of R_{avg} for irradiance caching decisions. The second is to retrieve the accumulated table of photon energy used to create the p.d.f for importance sampling.

These requests are sent to the photon gather unit where they are separated into two queues until processed. The nature of the queuing system is discussed in Section 7.3.4, but at some point each of these requests is dequeued and processed. Two packets are returned to the controller, one containing the contribution to the final image due to caustics and the second containing the value R_{avg}, used in irradiance caching decisions,

[2]The proposed architecture actually will have two tiles being processed at the same time to increase utilization. This detail has been removed from this overall description for clarity.

[3]Recall from Chapter 2 that a special photon map is created that is specialized for rendering caustics. Caustics are rendered by performing direct visualization on this *caustic photon map*.

and the accumulated energy table from the indirect gather, used in combined importance sampling.

At the controller, these packets are received and their information placed into the same data structure holding the intersection data. The packets may arrive in any order. Processing begins when they are all accounted for. First, the irradiance caching algorithm from Chapter 5 is applied to each pixel to determine if a final gather must be performed. This determination can be made *before* the final gathers only because the pre-computed values of R_{avg} have been found. Instead of a traditional irradiance cache data structure, those pixels that will interpolate the irradiance results to be computed at other pixels are added with an interpolation weight to a list in the tile data structure, to be consulted when the results arrive later. More details of this process are presented in Section 7.3.3.

For those pixels that *do* require a final gather, the energy table returned by the importance sample photon gather request is converted to a p.d.f., combined with an evaluation of the partially-evaluated BRDF, and stored as part of the intersection results determined earlier, as was described in Chapter 6. The combined p.d.f. and c.d.f. are used to generate the N_{FG} sample directions. With these sample directions now known, the BRDF is evaluated and a weight is calculated for each gather ray. This weight is sent in a packet along with the intersection point and sample direction to the ray caster. Note that the final gather rays are generated using the direction-bin reordering algorithm, see Chapter 3.

The final gather rays are now processed by the ray casting unit. They are traced through the scene until they encounter the first surface that is not purely specular. Once again the BRDF is partially evaluated. The details of these intersections are formed into a packet and sent to the photon gather unit, where standard indirect photon gathers at these secondary intersections, y_i, will be performed.

As the indirect packets arrive from the ray casting unit they are inserted into a set of queues kept in on-chip memory. These queues are used to perform the additional

reordering provided by the hashed technique described in Chapter 3. Details are provided in Section 7.3.4. The queues are drained one a time. For each photon gather request that is part of a final gather two values are computed and returned in a packet to the controller. First is the partial contribution that this gather provides to the destination pixel in the tile. While computing this value, the contribution to the irradiance at that pixel is also calcuated. This will be used by all the pixels dependent on the irradiance cache record for this destination pixel. Finally, when these values are returned to the controller they are accumulated into the respective pixels of the frame buffer, which is sent to the host when the tile is complete.

7.3.2 Ray casting unit

There are three duties of the ray casting unit: 1) generating and tracing the eye rays for each tile; 2) calculating direct illumination using shadow rays; and 3) tracing final gather rays to determine photon gather locations. All three duties use ray casting as the core kernel of operation. The internal structure of this unit can be a slightly restricted implementation of the SaarCOR architecture (Schmittler et al., 2002). That architecture has been validated by an FPGA implementation of a reduced version (Schmittler et al., 2004). The techniques described in their series of publications reduce the computational and bandwidth cost of ray-casting, as discussed shortly.

For each tile, the ray caster samples the image plane P times to generate the eye rays, $(e, \vec{\omega}_e)$. These rays will be highly coherent as they share a common origin and similar directions. Each ray is traced through the scene until it intersects with a surface at a point x that is not purely specular, as per the photon mapping algorithm laid out in Chapter 2. However, even those rays that do reflect off a specular surface will maintain a high degree of coherence (Chalmers et al., 2002). Note that glossy surfaces are treated as non-specular, their reflections will be computed with a final gather instead of a single

	Size	Explanation
Valid	1 bit	Did this eye ray actually intersect the scene
Tag	1 bit	Which of the two current tiles this packet belongs to
(x_o, y_o)	1 byte	Pixel offset within current tile (2×4 bits)
x	6 bytes	Intersection location (3×16 bits)
$\vec{n}$	4 bytes	Surface normal at x (2×16 bits)
$\vec{\omega}_o$	4 bytes	Direction of viewing ray arriving at x (2×16 bits)
$L_d(x, \vec{\omega}_o)$	6 bytes	Direct Illumination, RGB (3×16 bits)
W_{acc}	6 bytes	Accumulated attenuation on path, RGB (3×16 bits)
$f_p(\vec{\omega}_i)$	S_B	Partially evaluated BRDF

Table 7.2: Ray caster to controller packet. $\vec{\omega}_o$ will point from x towards the eye, unless a purely specular reflection or refraction occurred. If that case, the accumulated attenuation along the path from the eye to x is passed as W_{acc}. Each packet requires $(28 + S_B)$ bytes.

reflected ray.

At each intersection point x, the BRDF is partially evaluated by fixing the viewing direction, $\vec{\omega}_o$, as this is now known. This has the effect of reducing the seven dimensional $f(x, \vec{\omega}_i, \vec{\omega}_o)$ reflectance function to $f_p(\vec{\omega}_i)$, a two dimensional function. For almost all BRDF representations, this reduces the amount of data required to represent the reflectance. For example, because x is now fixed, all texture evaluations can be performed and merged into a single resulting RGB color. This partial-evaluation reduces the size of the packet that will be returned to the other units, as well as any storage that may be necessary. The exact size of the partially evaluated reflectance function, S_B, depends on the exact BRDF on the surface. The simulator described in Section 7.4 supports Lambertian and Phong lobe materials, which require only 13 bytes to represent $f_p(\vec{\omega}_i)$. The computational cost of performing the partial evaluation is $C_{B^1} = 12$ FLOPs.

If the eye ray *did* reflect or refract on highly specular surfaces then each of those interactions reduces the amount of light that will eventually reach the eye. To account for this, the reflectance at each specular interaction is computed and the product is returned to the controller as a weight. Generally, this weight will be 1.

For each eye ray that does eventually intersect with the scene, the direct illumination,

$L_d(x, \vec{\omega}_o)$, at the intersection x from the light sources is computed using shadow rays. The number of shadow rays required, N_S, depends on the complexity of the illumination. Area light sources of a large number of point sources can be accommodated using importance sampling (Szirmay-Kalos, 1999). The experiments in this dissertation use $N_S = 8$. Note that the shadow rays for the entire tile are expected to be be highly coherent as they are traced through the scene, because they share similar origins and direction. This reasoning is the same as was used in the discussion of tiled direction-binning photon gather reordering in Chapter 3. The results of tracing all P eye rays and computing the direct illumination are returned to the controller in individual packets. The composition and size of these packets is laid out in Table 7.2.

The third task of the ray caster is to trace the final gather rays that will be generated by the control unit. As was discussed in Chapter 3, tiled direction-binned reordering will tend to generate final gather rays in an order with similar origins and directions. Importance sampling will further increase the expected coherence of these rays, by concentrating them on a smaller portion of the hemisphere (see Chapter 6). Like eye rays, the final gather rays are traced until the first non purely specular intersection. The reflectance is evaluated at each intervening specular surface, with the product reducing the accumulated weight sent by the controller.

Direct illumination is not performed, but the BRDF is partially evaluated as before. Far more final gather rays will be cast than eye rays and shadow rays combined. The resulting intersection points, partially evaluated BRDF and adjusted weights are sent to the photon gatherer as requests for indirect photon gathers. The composition and size of the packets is presented in Table 7.3.

Two tiles are being processed by each rendering chip at any one time. Each packet contains a one bit flag to establish which tile it belongs to. The ray casting of final gather rays from the *older* tile takes precedence over the generation and tracing of eye rays for the newer tile. This preference helps to establish the guarantee of progress made

	Size	Explanation
Valid	1 bit	Did this final gather ray actually intersect the scene
Tag	1 bit	Which of the two current tiles this packet belongs to
(x_o, y_o)	1 byte	Pixel offset within current tile (2×4 bits)
y_i	6 bytes	Intersection location (3×16 bits)
$\vec{n}$	4 bytes	Surface normal at y_i (2×16 bits)
$\vec{\omega}_o$	4 bytes	Direction of viewing ray arriving at y_i (2×16 bits)
W_{acc}	6 bytes	Accumulated reflectance on path, RGB (3×16 bits)
W_{IC}	2 bytes	Irradiance weight, directly transfered (1×16 bits)
$f_p(\vec{\omega}_i)$	S_B	Partially evaluated BRDF

Table 7.3: Ray caster to photon gatherer packet. These packets are similar to those returning to the controller, Table 7.2, except: 1) no direct illumination is performed; 2) instead of the original eye intersection point x, the location of the final gather, y_i, is sent; 3) The value of W_{acc} provided in the request by the controller is updated with any intersected surfaces; and 4) an additional weight is transfered directly from the request, to the transmitted request. The total size per packet is $(24 + S_B)$ bytes.

in Section 7.5.3.

7.3.2.1 Computation

The FPGA implementation of the SaarCOR rendered 1024×768 images of interesting scenes at 13.5 FPS using some portion of the 4 GFLOPS provided by the FPGA chip. A conservative estimate of the average computation required per ray cast, both eye and shadow rays, is $C_{\text{RayCast}} = 330$ FLOPs, although the actual number is certainly lower because they did not utilize 100% of the FPGA resources.

Taking into account both the eye rays and the final gather rays, the entire system must cast $FTP(1 + R_{FG}N_{FG} + N_S) = 385$ million rays per second during the interactive rendering. For the values listed in Table 7.1, this works out to approximately 100 GFLOPS. Considering that multiple rendering chips will be used, only a portion of this computation need be performed by a reduced version of the SaarCOR architecture on each rendering chip (see Section 7.4).

7.3.2.2 Memory bandwidth

The SaarCOR architecture was shown to render approximately 100 million rays per second using only 1 GB/s of bandwidth to the scene description, including bilinear texturing (Schmittler et al., 2004). For 385 million rays per second, this requires a system wide bandwidth of approximately 4 GB/s. Spread across multiple chips, this becomes inconsequential next to the memory bandwidth requirements of photon gathering.

7.3.2.3 Storage

The only internal storage required is the cache used by the SaarCOR architecture to reduce the memory bandwidth requirements of ray tracing. As described in their paper, the cache is a specially designed set of mailboxes managed directly by the traversal and intersection units. The SaarCOR FPGA implementation achieved extremely low external bandwidth to the scene description using only a total of 75 KB of cache, divided among several uses.

7.3.3 Controller

The controller handles a wide range of activities. At a high level it communicates with the host, tracks which tiles are currently being processed, maintains the state of every pixel and generates the final gather rays using both irradiance caching and importance sampling. Central to the controller is a large table of information kept about each pixel of the current tile. A key feature of the design is that requests to other units are sent in packets with the pixel offsets, allowing them to be matched with the proper pixel when they are returned, potentially out of order.

As eye rays are generated, cast through the scene, and have their direct illumination computed, the results are sent to the controller as packets described in Table 7.2. This data is stored in the central table. Immediately two separate requests are sent to the

	Size	Explanation
Tag	1 bit	Which of the two current tiles this packet belongs to
Type	1 bit	Caustic or importance sampling/R_{avg}
(x_o, y_o)	1 byte	Pixel offset within current tile (2×4 bits)
x	6 bytes	Intersection location (3×16 bits)
$\vec{\omega}_o$	4 bytes	Direction of viewing ray arriving at x (2×16 bits)
W_{acc}	6 bytes	Accumulated reflectance on path, RGB (3×16 bits)
$f_p(\vec{\omega}_i)$	S_B	Partially evaluated BRDF at x

Table 7.4: Controller to photon gatherer packet. This packet is used for two types of requests, as indicated by the type bit. The requests for caustic gathers have a total size per packet is $(18 + S_B)$ bytes. The importance-sampling/R_{avg} gathers omit both W_{acc} and $f_p(\vec{\omega}_i)$ for a packet of 12 bytes.

photon gatherer unit. The first is for a caustic photon gather. The request packet is described in Table 7.4, and contains mostly a subset of what was originally received from the ray caster. The second request is for a kNN search to be performed, with the closest value of R_{avg} returned along with the computed energy table to be used for combined importance sampling.

At some later time, the results of these requests will be returned by the photon gatherer, using the packets described in Table 7.6. The RGB color returned by the caustic gather will be directly added to the indicated pixel in the central frame-buffer. As the values of R_{avg} are returned it becomes possible to begin the irradiance caching procedure.

The pixels are considered in the order they are received. The small initially-empty on-chip irradiance cache is consulted to see if there are sufficient useable records currently in the irradiance cache. If there are, the irradiance caching can be used. However, there are no irradiance values stored yet. Instead, a linked list is kept for each pixel that will perform a final gather. All pixels that determine that they will use that pixel's to-be-computed irradiance value are added to the list along with an weight. See Chapter 5 for details.

For those R_{FG} pixels that can not use the irradiance cache, a final gather will be

	Size	Explanation
Tag	1 bit	Which of the two current tiles this packet belongs to
(x_o, y_o)	1 byte	Pixel offset within current tile (2×4 bits)
x	6 bytes	Origin of final gather ray (3×16 bits)
$\vec{\omega}_i$	4 bytes	Direction of final gather ray (2×16 bits)
W_{acc}	6 bytes	Accumulated reflectance on path, RGB (3×16 bits)
W_{IC}	2 bytes	Irradiance weight (1×16 bits)

Table 7.5: Controller to ray caster packet. Each generated final gather ray is sent to the ray caster to be traced throughout the scene. Two weights are included so that upon return to the controller the results can be directly applied to the frame buffer. The total size per packet is 20 bytes.

performed using combined importance sampling. The photon gatherer has at this point returned the information needed to construct the p.d.f. table for incident radiance. Using the technique of Chapter 6 and the partially evaluated BRDF at x, the energy table is prepared into a combined p.d.f. table suitable for generating final gather ray directions.

The controller's next task is to generate, using the reordering algorithms of Chapter 3, the final gather rays and pass them to the ray caster. A total of $PR_{FG}N_{FG}$ final gather rays are generated for each tile using the combined importance sampling technique. In order to apply the tiled direction-binned photon gather reordering, it would be best to loop over the tile once for each bin and generate just those directions that fit the current bin. However, this is difficult when using importance sampling because the bins used by the importance sampling strategy are locally defined to the hemisphere around the normal. Nearby pixels can easily have greatly different normals. Therefore, the importance sampling bins can not be used directly in direction-binned reordering. Instead, the controller generates and stores all $PR_{FG}N_{FG}$ rays, then loops over them multiple times, sending a portion to the ray caster.

The final gather rays are sent to the ray caster using the packets defined in Table 7.5.There are N_{FG} final gather rays in the Monte Carlo integration and each is attributed a different weight, W_{acc}. Because it would be difficult to recreate this weight and expensive to store it until the result of the photon gather is returned the weight

is sent along in the request packet. This weight also accounts for any attenuation that might have occurred during the casting of the eye ray, as reported by the ray caster originally and stored in the central table. Additionally, a single floating point weight, W_{IC}, is sent. It will be used to properly weight the contribution that this final gather ray will have to the irradiance value of this pixel.

The final gather photons will be returned out of order because of the hashed reordering that will be performed in the photon gatherer. Because the packets, described in Table 7.6, contain the pixel offset and the values have been modulated by the weights sent with them through the ray casting, they can be added directly to the frame buffer. If the linked list of pixels dependent on this pixel for an irradiance value is not empty, then the returned irradiance value will be applied to those pixels accordingly.

7.3.3.1 Computation

Compared to the other units, the jobs of the controller are varied and the computations, while not great, are not as regular. It is envisioned that instead of dedicated custom-designed hardware, a general purpose embedded CPU would be used.

Accepting the incoming packets from the ray caster and issuing the initial requests to the photon gatherer requires no computation, simple the copying of data. The cost of octree manipulation is highly variable, as it depends on the qualities of the scene and viewing parameters. The task can be simplified by the recognition that there can be no more than P records inserted, and far fewer if irradiance caching is effective. The octree data structure can therefore be replaced by a simple unordered list for small values of P.

Adding a record to the list requires no computation. With a conservative estimate that all pixels are eventually added to the list, there are on average no more than $\frac{P}{2}$ entries during each lookup. Before a record is selected for use, it must be checked that it is closer than a maximum distance and that the normal does not vary too far from that of the current pixel, for a cost of 5 FLOPs per search record considered. The entire search

process for all P pixels of the tile therefore requires at most $P(5\frac{P}{2})$ FLOPs per tile. More efficient data structures and search algorithms may be used, but this computation is not significant compared to the other operations as it occurs only once per tile.

The combined PDF table is created and sampled as described in Chapter 6, at a cost of $PR_{FG}(3k + C_J(4 + C_{B^2}) + 7N_{FG})$ FLOPs per tile. An additional division and three multiplications are required per final gather ray to incorporate the probability with the previously accumulated reflectance and the $\frac{1}{N_{FG}}$ term from Monte Carlo integration. Reordering the gathers then requires only integer comparisons.

When the partial results are returned from the photon gatherer, the indirect and caustic illumination values are added to the frame buffer with 3 floating point additions, $3P(1 + R_{FG}N_{FG})$. The irradiance values are multiplied by the stored wight in each element of the linked list and then added to the appropriate pixel, $4PR_{FG}L_{IC}$, The computational requirements are summarized in Table 7.7.

7.3.3.2 Storage

The central table in the controller maintains a record for each pixel of the tile. Initially this contains the information returned by the ray caster, less the direct lighting value and the now implicit tile and pixel offset fields, for a total of $(21+S_B)$ bytes per pixel. To this is added the RGB frame-buffer, initialized to direct illumination; it accumulates value until the tile is finished and returned to the host for display. This requires 3×16 bits = 6 bytes per pixel.

As described above, the irradiance cache is implemented with an unordered list instead of an octree, due to the small number of potential records. Each record is embedded in the central tile store and contains the value of R_{avg} (see Chapter 5), which adds 1×16 bits = 2 bytes per pixel.

Experimentally it has been observed that when reasonable values of ϵ_{max} are used for irradiance caching, the linked list of irradiance cache links will have on average $L_{IC} = 4$

links per pixel. The linked list needs to store the offset of the dependent pixel, 1 byte, and the weight, 6 bytes, as well as the link to the next element, 2 bytes, for a storage cost of $9PL_{IC}$ bytes. For the expected values from Table 7.1, this is only about 9 KB.

The temporary storage of the final gather rays during reordering requires $PR_{FG}N_{FG}$ rays each composed of a (θ,ϕ) pair and a probability of having chosen that direction, stored as 5×16 bits = 10 bytes. This will require 33 KB for the expected parameter values. (The value grows to 100 KB if $N_{FG} = 100$ is required for satisfactory image quality.)

Although there will be two tiles in progress at any time, requiring the central table to be duplicated, there will only be a single tile performing final gathers at any one time, so some structures are only needed once. This brings the total internal storage requirements of the controller to $2P(27 + S_B) + 9PL_{IC} + 10PR_{FG}N_{FG} + 2P$.

7.3.4 Photon gatherer

The internal structure of the photon gatherer is shown in Figure 7.3. Inspired by the GI-Cube architecture (Dachille and Kaufman, 2000), the three types of incoming gather requests are sorted into a series of queues. Those arriving from the ray casting unit are for final gathers, and are assigned to a subset of the queues using a hash function as explained in Section 3.4.3. The simulation results presented in this chapter use 17 queues with 128 slots each for the hashed reordering. Note that the queues are able to hold *only a small portion* of the photon gather requests required to render a single tile. Therefore, only a sliding window of requests are reordered. It is worth noting however that any specific gather may take a very long time to be processed if its queue is never the most full. However because of guarantees discussed in Section 7.5.3, at the end of every tile the hash queues will be emptied, ensuring that every request is processed.

The two types of photon gather requests from the controller are handled differently.

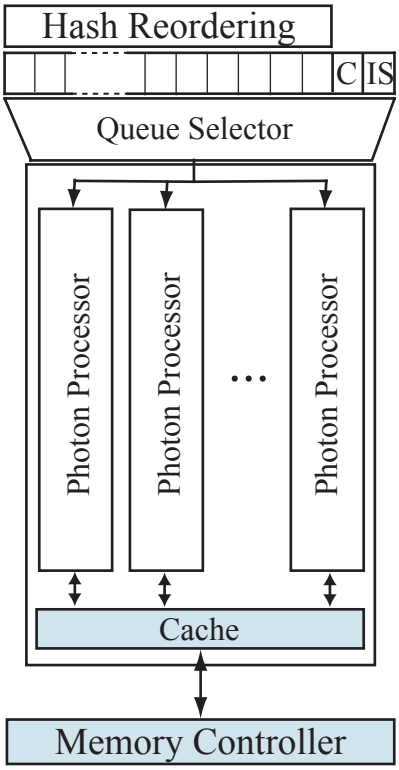


Figure 7.3: The photon gatherer treats a series of queues as hash buckets to perform additional sorting on the gather requests. The caustic photon gathers and importance-sampling/R_{avg} requests are however given their own queues. A single queue is drained until empty before another queue is selected. This serves to improve memory cache efficiency. Several gathers are processed independently. Although a large number of floating point operations are required, they are very regular and simple.

Two separate queues are provided for each of the caustic photon gather requests and the importance-sampling/R_{avg} requests. These requests are given their own queues because the locations of their requests are very coherent; they take place at the intersection points, x, of a single tile. Furthermore, the caustic photon gather requests actually are serviced out of an entirely different photon map, which would severely pollute the cache if mixed with photon gathers from the indirect map.

Multiple photon gather processors, with a shared cache, work in tandem to process the requests stored in a single queue. When there are no more entries in the currently selected queue, the sizes of all queues are compared and the one that has the most requests is selected. In order to eliminate one cause of deadlock the queues holding final

gather requests are given priority over the queues holding both caustic requests and the importance-sample/R_{avg} requests, regardless of how full the queues are (see Section 7.5.3).

Those indirect photon gathers that belong to a final gather are processed as follows. First a kNN search is performed by traversing the indirect photon map, stored as a kd-tree. This traversal is based directly on the technique laid out in the SaarCOR architecture (Schmittler et al., 2002). The direct traversal of the kd-tree is performed using integer operations. This results in a stream of candidates for the kNN search. Each potentially usable photon is checked to see if it is within the maximum radius and if the normal is close enough to the normal at x. If the photon is valid, it is inserted into the priority list maintained using the parallel sorter introduced in the GI-Cube (Dachille and Kaufman, 2000).

When the final k photons have been identified, the energy they carry is reflected to the viewed direction by finishing the evaluation of the BRDF at y_i (begun in the ray casting unit). This result is the amount of light reflected from this particular photon gather location towards the original intersection point x. This value is returned to the controller in two forms after being multiplied with the two weights provided in the request packet, Table 7.3. The first represents the contribution that this value has on the actual pixel performing this final gather. The second is the weight that this reflected value should have on the estimate of the irradiance value at the same pixel. This will be used by those *other* pixels that are interpolating the irradiance of *this* pixel.

Caustic photon gather requests are handled in a similar fashion. However they use the caustic photon map for their kNN search and do not generate partial irradiance values. The third type of photon gather request performs the kNN search in the indirect map and then performs two quick operations. First, an energy table, as described in Chapter 6, is built. Secondly, the closest photon with a pre-computed R_{avg} value is found, this is easy as the kNN search had to order the photons already. The results of all requests are returned to the controller for processing, using the packets described in Table 7.6.

	Size	Explanation
Tag	1 bit	Which of the two current tiles this packet belongs to
Type	2 bits	Final gather, caustic or importance-sampling/R_{avg}
(x_o, y_o)	1 byte	Pixel offset within current tile (2×4 bits)
FP_{part}	6 bytes	Partial contribution to frame-buffer, RGB (3×16 bits)
E_{part}	6 bytes	Partial contribution to irradiance value, RGB (3×16 bits)
R_{avg}	2 bytes	Average distance to scene, (2×16 bits)
p_{L_i}	$2C_J$	Incident radiance energy table for combined importance sampling

Table 7.6: Photon gatherer to controller packet. The packet size varies depending on the type of gather request that has been serviced. A caustic gather omits E_{part}, R_{avg} and p_{L_i} for a packet size of 10 bytes. The gathers that are part of a final gather omit R_{avg} and p_{L_i} for a total of 16. Finally the importance-sampling/R_{avg} gathers omit FP_{part} and E_{part} for a total of $6 + 2C_J$.

7.3.4.1 Computation

To find k photons in a photon map with N_{PM} photons, $O(k + ln(N_{PM}))$ photons must be examined (Jensen, 2001). Experiments showed that a reasonable rule of thumb is that $1.2k$ photons must be examined. 13 floating point operations are required to check against the radius and normal. The photons are then partially sorted to select the k closest photons that are acceptable. The cost of choosing the nearest k=100 photons, is approximately $C_G = 13(1.2k) + 1.2klog_2(1.2k)) = 2,389$ FLOPs.

Both caustic and indirect photon gathers must compute the amount of light reflected back along the viewing ray for each chosen photon. Since the BRDF has already been partially evaluated by the ray caster, only $C_{B^2} = 10$ FLOPs are required for each gather now that the incident radiance direction, $\vec{\omega}_p$, is known. In indirect gathers, the two separate weights are multiplied at the cost of 2 FLOPs. (Caustic gathers only require 1 multiplication as they are not involved with irradiance caching.)

The computational cost for performing an indirect gather, $C_G + k(C_{B_2} + 2)$ FLOPs, becomes significant in the context of the large number of indirect gathers performed, $FTPR_{FG}N_{FG}$. For importance-sampling/R_{avg} gathers, instead of computing the BRDF, the flux carried by each photon is simply accumulated in a table as described in Chapter 6.

(The assignment of each photon to a bin is performed by an inexpensive integer table lookup.)

The regular nature and high performance requirements of the photon gatherers leads to the decision to implement them with dedicated hardware. The *kd*-tree traversal is patterned on the SaarCor's architecture. Because the gathers are expected to be highly coherent, the memory accesses are expected to be fulfilled primarily by the cache, allowing the traversal unit to be highly utilized. The partial BRDF evaluation, however, will be implemented in a simple programmable processing element in order to support a user defined BRDF. The table lookup of photons to energy table locations can be performed very cheaply using integer compares in dedicated hardware.

7.3.4.2 Storage

There are two large sources of storage requirements in the photon gatherer unit. The first is the large set of queues used to reorder the gather requests. There are 19, each of which holds 128 requests. Each request requires the storage of the complete packet sent from either the ray caster or the controller. The larger of these two require $(24 + S_B)$ bytes. The queues therefore use $19 \times 128 \times (24 + S_B)$ bytes, or 88 KB for the nominal value of S_B.

The simulation results presented throughout this chapter use a single shared memory cache in the photon gatherer of 128 KB. Large caches were shown in Chapter 3 to reduce bandwidth requirements for particularly complicated scenes, but 128 KB provided most of the potential benefit for the scenes investigated in this dissertation.

7.3.4.3 External bandwidth

The photon gather unit is, of course, the source of the tremendous bandwidth requirements. Multiple gather units work in unison to drain a single queue at a time. This increases the coherence of the direction-binned importance-sampled gather locations. It

Operation	FLOPs per tile
Ray casting	$P(1 + R_{FG}N_{FG} + N_S)C_{\text{RayCast}}$
Partial evaluation of BRDF	$P(1 + R_{FG}N_{FG})C_{B^1}$
Caustic gather	$P(C_G + k(C_{B^2} + 1))$
IS/R_{avg} gather	$P(C_G + k)$
Indirect gathers	$PR_{FG}N_{FG}(C_G + k(C_{B^2} + 2))$
Irradiance cache decisions	$5P(P/2)$
Compute combined c.d.f.	$PR_{FG}(3k + C_J(C_{B^2} + 4))$
Generate final gather rays	$7PR_{FG}N_{FG}$
Accumulate gathers	$3P(1 + R_{FG}N_{FG})$
Accumulate irradiance links	$4PR_{IC}L_{IC}N_{FG}$

Table 7.7: A summary of the computational cost of using photon mapping to generate a single tile with the architecture laid out in this section in terms of the notation laid out in Table 7.1. The most significant terms are those of ray casting and the indirect gathers.

is particularly important for the two special queues. The caustic requests are in an entirely separate photon map, and would otherwise pollute the cache. The requests stored in the importance sample queue will all generally be very close to each other due to the way eye rays intersect the scene. The bandwidth generated by the architecture is studied in Section 7.4.

7.3.5 Summary of costs

The architecture laid out in this chapter requires significant resources beyond external memory bandwidth: the computational requirements in the various units, the communication rate between each unit and the required on-chip storage must all be feasible with predicted semiconductor technology. In this section these various requirements are summarized for the processing of a single tile. Section 7.4.3 uses these formulas, along with experimental results for external bandwidth, to determine the maximum number of tiles that a single rendering chip can render. That value in turn leads to the number of rendering chips and boards required to display images at 30 frames per second.

Table 7.7 lists the computational requirements, in FLOPs, of rendering a single tile.

Link	Packet Size	Count
Raycaster—Controller	$S_B + 28$ bytes	P
Raycaster—Gatherer	$S_B + 24$ bytes	$PR_{FG}N_{FG}$
Controller—Raycaster	20 bytes	$PR_{FG}N_{FG}$
Controller—Gatherer		
Caustic gathers	$S_B + 18$ bytes	P
IS/R_{avg} gathers	12 bytes	P
Gatherer—Controller		
Caustic gathers	10 bytes	P
IS/R_{avg} gathers	$(6+2C_J)$ bytes	P
Indirect gathers	16 bytes	$PR_{FG}N_{FG}$

Table 7.8: A summary of the inter-unit bandwidths in the architecture laid out in this section to generate a single tile in terms of the notation laid out in Table 7.1. The most significant links are those handling the final gather photon gather requests.

Due to the large number of final gather rays, even when using irradiance caching and importance sampling, the two highest cost operations are ray casting and the indirect photon gathers. Fortunately, these two operations are both highly regular and require no communication. This enables the architecture to take full advantage of the increases in semiconductor performance, as its requirements are more like a GPU than a CPU.

The architecture is designed on the assumption that all rendering units are on the same die. This assumption permits high bandwidth communication between the units at low cost. Table 7.8 summarizes the expected bandwidths between all units. These values are maximums. For example, if an eye ray does not intersect with the scene then final gather rays will not be generated for that one pixel. The links that carry the most data are those that are involved with the final gathers. No attempt has been made to compress the interconnection between units.

The rendering algorithms laid out in this chapter require intermediate data structures which must be stored on-chip if the external memory bandwidth is to be kept to a minimum. Additionally, the memory caches, used to reduce memory bandwidth, must also fit on the die. Table 7.9 enumerates these costs. Each rendering chip is allowed to have two tiles being processed at the same time. The storage implications of this

Location	Purpose	Storage Size
Ray caster	Memory Cache	75 KB
Controller	Irradiance Cache	$2P$
Controller	Irradiance Links	$9PL_{IC}$
Controller	DirBin Lists	$10PR_{FG}N_{FG}$
Controller	Central Table	$2P(27 + S_B)$
Photon gatherer	Hash Queues	$19 \times 128(24 + S_B)$
Photon gatherer	Memory Cache	128 KB

Table 7.9: A summary of the on-chip storage requirements of the architecture laid out in this section in terms of the notation laid out in Table 7.1. The most significant usage are the caches and reordering queues in the photon gatherer, however compared to commodity chips with 4-10MB on on board cache these values are reasonable. Allowing two tiles to be in process at the same time requires that there be two copies of the central table.

are that the central table in the controller must be duplicated. For reasonable values of the parameters, such as those found in Table 7.1, these on-chip storage requirements are reasonable, less than the multiple megabytes of cache used on high end CPUs.

7.4 Simulation and analysis

The architecture presented in Figure 7.2 has been validated and analyzed using a functional simulator built on the `pbrt` rendering system (Pharr and Humphreys, 2004). Although the core libraries and routines were used, the standard top-level rendering loop was replaced with a collection of functional units corresponding to the ray caster, controller and photon gatherer. The pre-processing of the photon map was extended to the calculation and storage of R_{avg} for irradiance caching.

Communication between the units was constrained to the exchange of packets, as defined in Tables 7.2 to 7.4. The internal storage of the units was limited to that presented in Table 7.9. Each unit has been simulated to a different level of detail in order to show the feasibility of the concept and that it has the correct interaction with the other units.

The ray caster is a reduced implementation of the SaarCOR architecture. As that

architecture was validated in their second paper (Schmittler et al., 2004), the performance figures are adopted from their work, and core `pbrt` routines are utilized to perform ray casting. Their performance numbers, in terms of memory bandwidth and computation, are actually conservative as the needs of the rays that are cast in this architecture are strongly coherent (see Section 7.3.2).

The photon gatherer simulation concentrates on providing an accurate simulation of the cache behavior of the memory accesses. The memory bandwidth is measured by intercepting all memory references to the photon map and simulating a cache. (Using a process identical to that of Section 3.2.) The hashed photon gather reordering queues are augmented with the caustic and importance-sampling/R_{avg} queues, as described in Section 7.3.4. A single photon processor was used during simulation.

The simulation of the control unit focuses on the detailed and complex coordinating role that it plays in the architecture. The primary outcome of its simulation is the validation that it has the correct information, when needed, in order to generate the final gather rays. The irradiance cache process was simplified using the reduced data structure described in Section 7.3.3.

After describing the characteristics of the specific scenes used during testing, the measured memory bandwidth requirements are presented along with the calculated computational requirements. It is these two factors that limit performance by establishing an upper bound on the number of tiles per second that a single rendering chip can render for each scene. This information is used to determine the number of rendering chips required to meet the interactive goal of 30 frames per second for the test scenes.

7.4.1 Test scenes

Performance is highly dependent on the scene being rendered, the complexity of the geometry, the complexity of the illumination, and how effective performance enhancing

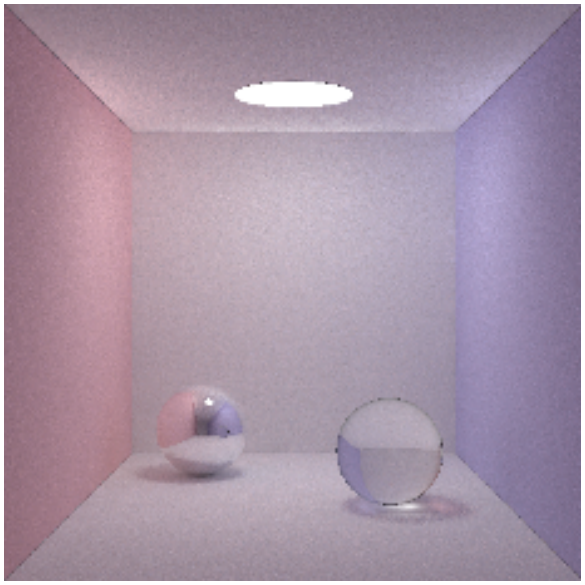
techniques such as irradiance caching, importance sampling and gather reordering can be. The scenes from Chapter 1 provide the basis for a set of experiments in this section to determine the expected performance of some typical implementations of the architecture.

The first scene is the modified Cornell box used throughout the dissertation. As Figure 7.4a demonstrates, an acceptable image can be generated using only 33 final gather rays per pixel without irradiance caching, importance sampling or photon gather reordering. In Figure 7.4b the three techniques are applied. The irradiance cache is highly effective, 62% of the pixels avoid the final gather process completely. Together with photon gather reordering the bandwidth required to generate a single image is reduced from 50 GB to 7.8 GB.

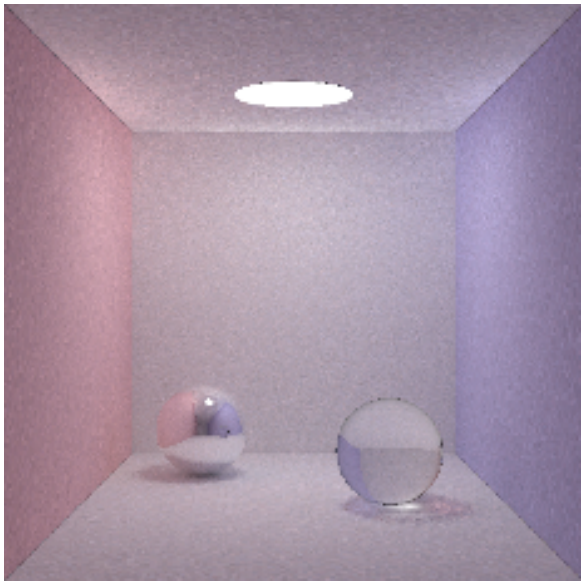
There is a very strong direct illumination component in the Cornell box. Except for points in shadow, more radiance arrives from the overhead light than as the result of reflections. As previously discussed, it is much more difficult to create low-noise images for scenes that have predominately indirect illumination. For the bottom row of Figure 7.4 the overhead light has been replaced with a white diffuse circle, with a spot light located below, targeted perfectly on the circle. The image in Figure 7.4c required 300 final gather rays per pixel and still is not acceptable. In Figure 7.4d however importance sampling allows for fewer samples to produce an image of greater quality with less than one tenth the bandwidth requirements

Figure 7.5 shows a box with a glossy floor and a textured back wall for a set of similar experiments. Because the camera has focused on the glossy floor, irradiance caching is not very effective. Only 8% of the pixels are able to make use of the irradiance cache. Combined importance sampling is, however, quite effective. In both of the cases of direct and indirect illumination the number of samples per pixel can be reduced by a factor of three while improving the quality of the generated images. The bandwidth requirements are also reduced by over an order of magnitude when all three techniques are applied.

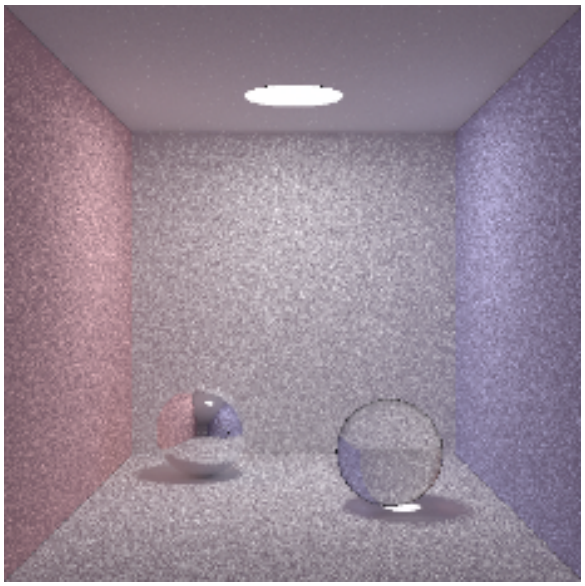
The Sponza atrium, shown in Figure 7.6, naturally has only indirect illumination.



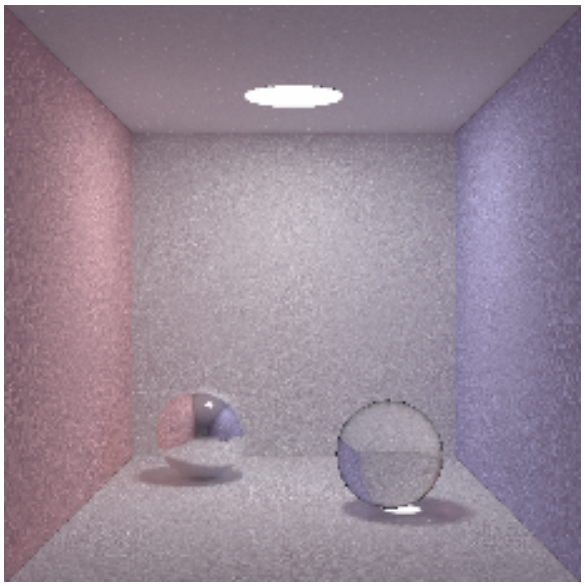
(a) Direct illumination, no optimizations, 50 GB
$N_{FG} = 33$



(b) Direct illumination, all optimizations, 7.8 GB
$N_{FG} = 33$, $\epsilon_{max} = 0.035$ ($R_{IC} = 62\%$)

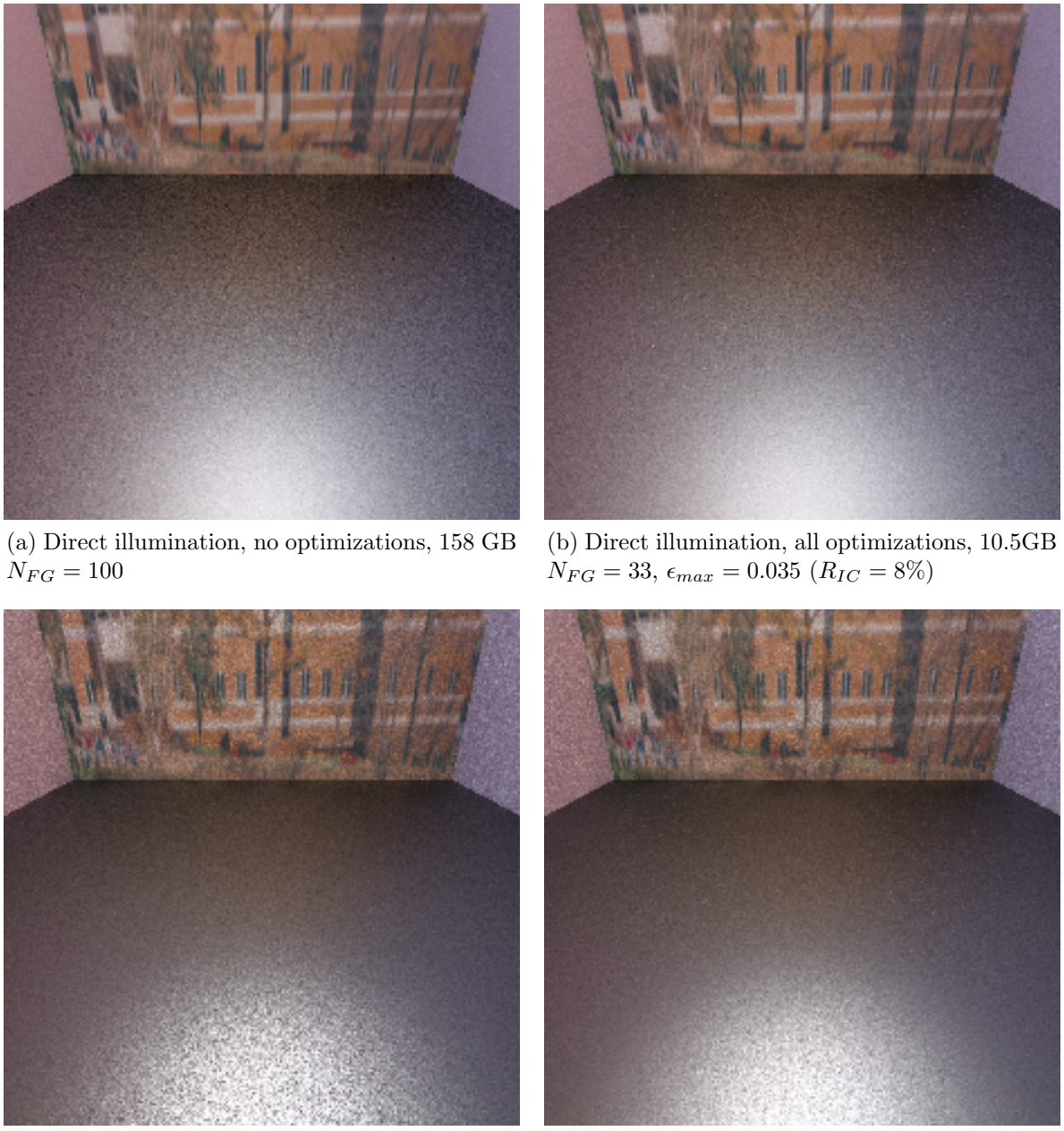


(c) Inirect illumination, no optimizations, 391 GB
$N_{FG} = 300$



(d) Inirect illumination, all optimizations, 23 GB
$N_{FG} = 200$, $\epsilon_{max} = 0.025$ ($R_{IC} = 46\%$)

Figure 7.4: The top row shows the modified Cornell box with direct lighting. In the bottom row, the light has been replaced with a reflected spotlight leaving only indirect illumination, more difficult to render. In both cases the use of irradiance caching, importance sampling and photon gather reordering enables images of higher quality to be generated using substantially less memory bandwidth.



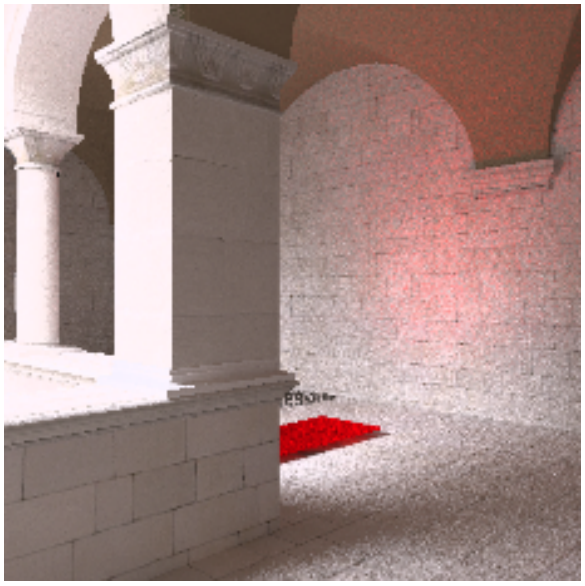
(a) Direct illumination, no optimizations, 158 GB $N_{FG} = 100$

(b) Direct illumination, all optimizations, 10.5GB $N_{FG} = 33$, $\epsilon_{max} = 0.035$ ($R_{IC} = 8\%$)

(c) Inirect illumination, no optimizations, 426 GB $N_{FG} = 300$

(d) Inirect illumination, all optimizations, 22 GB $N_{FG} = 100$, $\epsilon_{max} = 0.025$ ($R_{IC} = 8\%$)

Figure 7.5: When the camera is focused on a glossy reflector, most pixels are unable to use irradiance caching. Combined importance sampling, together with photon gather reordering, however is able to significantly reduce the memory bandwidth requirements.



(a) Direct illumination, no optimizations, 367 GB
$N_{FG} = 200$

(b) Direct illumination, all optimizations, 10.5 GB
$N_{FG} = 100$, $\epsilon_{max} = 0.063$ ($R_{IC} = 36\%$)

Figure 7.6: With the exception of the glossy red reflector, the Sponza atrium is purely diffuse and irradiance caching is an effective technique. For those final gathers that must be performed, combined importance sampling is effective at generating final gather rays that find the central, well-lit atrium. Together the techniques reduce the bandwidth.

Although the surfaces are all purely diffuse, except for the red reflector, combined importance sampling is still able to provide a significant boost by generating more sample rays out towards the central courtyard which receives sunlight. Irradiance caching is fairly effective, and altogether the memory bandwidth requirements are reduced by a factor of forty.

7.4.2 Single chip performance

The performance of a single chip implementation of the architecture is potentially limited by either bandwidth or computational resources. Performance can be measured as the number of tiles of a given scene that can be rendered in one second. Table 7.10 brings the measured memory bandwidth results from the previous section together with the calculated computational requirements of generating each test scene (Table 7.7).

Scene	$\frac{\text{GB}}{\text{image}}$	$\frac{\text{tiles}}{\text{sec}}$	$\frac{\text{GFLOPs}}{\text{image}}$	$\frac{\text{tiles}}{\text{sec}}$
Modified Cornell (Direct)	50	1,843	37	13,838
Modified Cornell (Direct, optimized)	7.8	11,815	16	32,000
Modified Cornell (Indirect)	391	236	312	1,641
Modified Cornell (Indirect, optimized)	23	4,007	115	4,452
Glossy floor (Direct)	158	583	106	4,830
Glossy floor (Direct, optimized)	10.5	9,701	34	15,059
Glossy floor (Indirect)	426	216	312	1,641
Glossy floor (Indirect, optimized)	22	4,189	98	5,224
Sponza atrium	265	348	209	2,450
Sponza atrium (optimized)	24	3,840	69	7,420

Table 7.10: Measured in tiles per second, the maximum performance of a rendering chip varies by test scene and choice of optimizations. A 90 GB/s memory connection is able to render at least 3,800 tiles per second for each test scene when all optimizations are used. In every case the 500 GFLOPS capability of the single chip was able to support the same tile rate as the memory.

There are 1,024 16×16 tiles in each 512×512 image. Therefore, a rendering chip with 90 GB/s of memory bandwidth is able to render at least 3,800 tiles per second for each test scene, when all the optimizations (irradiance caching, importance sampling and photon gather reordering) are used.

The only case where the 500 GFLOPS capability of the single rendering chip is unable to support the performance allowed by the memory is the modified Cornell box with indirect lighting and all optimizations. In this case photon gatherer reordering was particularly well suited to the scene, reducing memory requirements by a factor of twenty.

It is now particularly clear that irradiance caching, importance sampling and photon gather reordering should be used. Not only were the images of higher quality, but performance was significantly better in terms of memory bandwidth and computational cost.

Scene	$\frac{\text{tiles}}{\text{sec}}$	Frames/Sec			
		1 Chip	2 Chips	4 Chips	8 Chips
Modified Cornell (Direct, optimized)	11,815	12	23	**46**	**92**
Modified Cornell (Indirect, optimized)	3,765	4	7	15	**30**
Glossy floor (Direct, optimized)	9,701	9	19	**38**	**76**
Glossy floor (Indirect, optimized)	4,180	4	8	16	**33**
Sponza atrium (optimized)	3,840	4	8	15	**30**

Table 7.11: Based upon the maximum performance, in tiles per second, for each optimized scene in Table 7.10, the system performance for multi-chip systems is presented here. Those configurations achieving at least 30 frames per second are shown in bold.

7.4.3 Expected system performance

The architecture permitted no communication between rendering chips or between tiles. A consequence is that the system scales in terms of rendering chips. The only limiting factor is the number of tiles available for processing, and the host interconnect. As was discussed in Section 7.2.2, the host interconnect has very low bandwidth requirements. The system can certainly scale to the number of rendering chips that can be reasonably inserted into a host workstation. The results of this linear scaling are shown in Table 7.11. When using all the optimizations, all the test scenes can be rendered at 30 frames per second or higher using eight rendering chips configured on two boards with four chips each. Such an implementation can fit inside of a standard desktop workstation.

7.5 Discussion

A desktop-sized implementation of the architecture has been shown to be able to provide the desired interactive performance. Using irradiance caching, importance sampling, and photon gather reordering in combination, off-chip memory bandwidth, computational requirements, on-chip intra-unit bandwidth, and internal storage are all practical on near future semiconductor technology.

In this section the architecture is examined at a higher level, concentrating on concerns

such as load balancing, scalability, avoidance of deadlock and the limitations imposed by design choices along with some potential design alternatives.

7.5.1 Load Balancing

Any parallel system must establish a method for distributing work among its processors. The goal is to ensure that the full potential of a system is being used. There are two levels on which to think about load balancing in this architecture.

The first is the allocation of screen tiles to rendering chips. Images are often highly uneven in their geometric and illumination complexity. The result is that some tiles will take significantly longer than other tiles to compute. However, each tile is so expensive to render that the task allocation can be performed dynamically. A simple, greedy allocation process suffices. If a rendering chip is given a series of relatively easy tiles, it will finish them quickly and be assigned more while other units work harder on fewer tiles. Although there is plenty of bandwidth to the host, latency could be an issue as the CPU might be busy. Each chip therefore maintains a very short list of assigned tiles.

Secondly, within each chip, the load balance question is with regards to the utilization of the three components. The photon gatherer needs enough computational power to fully saturate the available bandwidth to off-chip memory, but no more. The other units need only be as powerful as they need to be to keep the photon gatherer busy. With two tiles being processed at any given time, if the ray caster and controller get ahead of the photon gatherer they can prepare requests for the gatherer as soon as it finishes the previous tile.

7.5.2 Scalability

The architecture is scalable in several dimensions. Performance for a fixed scene can be linearly improved by the addition of extra rendering chips, until communication or task

allocation barriers arise. However, because there is no communication allowed between tiles, the tiles can be processed independently on separate rendering chips, which are also not permitted to communicate with each other. This is a direct advantage of rendering with tiles, reordering with tiles, and using independent irradiance caches per tile with no global sharing of results.

Because of the low bandwidth requirements between the rendering chips and the host, several workstations connected in a local area network to a single host would be able to support a large image, displayed interactively, of scenes where irradiance caching and importance sampling were not particularly effective. Although such a machine would be quite expensive, the implementation would be simple and not require custom cabinets, power supplies, or interconnects.

Using similar a argument, the architecture is predicted to scale to larger image resolutions. The only effect that will be noticed with the increase in the number of tiles to be rendered is that if the field of view is kept constant there should be a mild increase in the number of pixels able to use the irradiance cache, R_{IC}, because the pixels will project closer to each other in the scene.

A consequence of the full replication of the scene and photon map to the private memory bank of each rendering chip is that the scene complexity is restricted to memory available. The implementation proposed in this chapter provided 256 MB, to be shared among geometry, texture and photon maps.

7.5.3 Progress and deadlock

A parallel system is said to be *deadlocked* if any component is waiting for an event that will never occur (Chalmers et al., 2002). In the absence of deadlock there is said to be *progress*. The architecture presented in this chapter presents a few areas of concern where deadlock could potentially occur. First, the potential for deadlock when only one tile

is being processed at any time on a rendering chip is considered. Then, the additional complications of two tiles being processed concurrently is discussed.

The raycaster, controller and photon gatherer are connected by queues of bounded size. If the queue between a unit A and a unit B becomes full, unit A will be unable to perform any additional processing until unit B has partially drained the queue. This will cause deadlock only if the units have a circular dependency. As an example, the photon gatherer sends the results of final gather rays to the controller. If the controller were still generating the final gather rays for this tile and was unable to process the photon gatherer's results, then the photon gatherer would be blocked from performing any more work. This would in turn prevent the photon gatherer from accepting any more final gather requests from the ray caster, which would be unable to absorb final gather rays from the controller.

A simple answer to this problem, and all the others like it in the architecture, is that each unit preferentially handles those requests that are the closest to contributing to the frame buffer. For the photon gatherer this means that requests for caustic and importance-sample gathers have a very low priority. The ray caster will preferentially cast final gather rays over new eye rays, and the controller will process incoming results from the photon gatherer before accepting packets from the ray caster. This approach can be made more efficient in the controller by the addition of a small dedicated piece of hardware to handle the blending of incoming final gather results, leaving the embedded CPU free to generate final gather rays. Further experiments need to be conducted to establish that this preferential treatment does not reduce utilization of upstream components.

If multiple tiles are being processed at the same time on a single rendering chip (the architecture described in Section 7.3 has two tiles in process at a time), deadlock can occur if the newer of the tiles is allowed to block a resource needed by the older tile. This architecture avoids this problem by not starting a new tile until all of the first tile's primary eye rays have been generated, cast, and the caustic and importance

sample/irradiance requests generated and sent to the photon gatherer. This does not reduce performance as long as generating and processing final gather rays remains teh dominate cost.

7.5.4 Limitations

There are several limitations to the overall system design, the specific architecture presented here, and its evaluation. The most serious is the requirement that the photon map be pre-computed by the host and broadcast to each rendering chip. Depending on host speed, this precludes dynamic scenes with moving objects or light sources; only the camera is allowed to move. The cost of generating a photon map is significantly less than that of rendering from it using final gather visualization, so it is possible that an architecture could incorporate this task directly.

Although the architecture is not itself limited to Lambertian and Phong BRDFs, the simulation and performance analysis is. There are two ways that the performance of the system is sensitive to the choice of allowed BRDF. The computation costs of the BRDF can be significant, but it is the cost of finishing the evaluation of the BRDF, C_{B^2} FLOPs, that is particular important as it is part of the photon gather cost for final gathers. Secondly, the storage size of the partially evaluated BRDF, S_B bytes, not only changes the size of the packets emitted by the ray caster, but also the storage that the controller and photon gatherer hold.

One area where the architecture does not scale well is with increases in either scene complexity or photon map size. It is currently required that both of these data structures be fully replicated to the dedicated memory of each rendering chip. Relaxing this would require a complete redesign of the architecture.

Only a functional simulation was constructed. Each unit was written in a high level language, accepts the specified input and emits the described output. Every operation

performed was justified in terms of analyzed computation cost, but was implemented with general purpose libraries. No effort was made to be cycle accurate, so there is no simulated timing data. A logical extension of this research would be a low level simulator, or implementation of key units on a FPGA to ensure validity of the design.

CHAPTER 8

SUMMARY AND CONCLUSION

The growing demand for realistic image synthesis using global illumination is clear. Unfortunately, the interaction of light, in a general way, with multiple surfaces between the light source and the viewer is very costly to simulate, as it involves a potentially unbounded number of computations. This problem is particularly acute for interactive applications such as video games and training simulators. These applications must generate a new image dozens of times a second.

The research presented in this dissertation provides one possible solution to the problem by providing a novel hardware architecture that interactively renders scenes using the photon mapping algorithm. The photon mapping algorithm accurately renders many of the visual effects that are expected in real scenes. However, the resources it requires have precluded an interactive implementation of the full algorithm. In my dissertation, I have addressed this issue by presenting several techniques that reduce the resource requirements. A hardware architecture implementing this techniques was shown to be promising, and presents a potential direction for the next generation of graphics hardware.

8.1 Research Contributions

The research presented in this dissertation has made several contributions. These include novel techniques to dramatically reduce the bandwidth cost of photon mapping. These

techniques were then combined into a feasible hardware architecture, which could be built in the next three years, that supports interactive applications.

Specifically, the research contributions of my work include:

Low bandwidth photon gathers using reordering: I presented photon gather reordering in Chapter 3. This technique generates the exact same images as the standard photon mapping algorithm, but reorders the computations such that the memory accesses become more coherent. The higher locality of reference increases the efficiency of a cache, reducing the bandwidth requirements.

Several reordering algorithms were introduced and compared. While the Hilbert reordering, applied to the entire image, reduced bandwidth requirements by four orders of magnitude, it required a prohibitive amount of intermediate storage. The combination of the tiled direction-binning generative reordering with the hashed deferred reordering is practical, easy to implement and highly effective, achieving over an order of magnitude reduction in bandwidth requirements, 357 GB to 31 GB for the Sponza atrium image.

Tiled irradiance caching with pre-computed radius for split-sphere heuristic: Irradiance caching reduces the number of final gathers by interpolating previously computed values. Although it should only be used on purely diffuse surfaces, it can be highly effective at reducing the computation and memory bandwidth requirements of photon mapping for some scenes. The conventional irradiance caching algorithm, however, imposes a sequential dependency between the pixels of an image, preventing efficient parallel execution.

In Chapter 5, I laid out two solutions that when implemented together allow irradiance caching to be used in a parallel rendering system. The first used a separate irradiance cache for each tile, eliminating communication between processors. The second stored a pre-computed value for Ward's split-sphere heuristic, permitting

the combination of irradiance caching with photon gather reordering. The modified irradiance caching algorithm was able to avoid up to half of all gathers in compatible scenes.

Combined importance sampling: Global importance sampling, which generates final gather rays in proportion to prior knowledge of *both* the incident radiance and the surface reflectance, reduces the number final gather rays required to generate an image. Previously published sampling algorithms maintained the prior knowledge in two separate sampling strategies. Although providing superior sampling, this raises costs.

Combined importance sampling was presented in Chapter 6 as a technique that lowers computational and storage costs by merging multiple probability distribution functions together, leaving only a single sampling strategy. Combined importance sampling was demonstrated to be effective in common scenes while being cheap to compute. For the test scenes shown in Chapter 7, with complex illumination and/or glossy surfaces, the number of final gather rays are reduced by approximately one third while generating higher quality images.

A complete photon mapping architecture: In Chapter 7, these three techniques were combined into a complete hardware architecture, which was then functionally simulated. It was shown that this architecture would be capable of rendering scenes with complex illumination from a static pre-computed photon map. A target implementation, using two expansion boards in a workstation with a total of 8 replications of a custom designed chip, would be expected to render images of the test scenes at rates of at least 30 frames per second. As the scene geometry, textures and photon map are replicated to each chip, they must fit in the dedicated memory, proposed at 256 MB. This architecture was shown to be feasible, for the

expected semiconductor technology of 2010, by measuring and/or calculating the external bandwidth, computation, inter-unit bandwidth and storage requirements.

8.2 Limitations

The techniques and architecture presented in this dissertation are promising, but there are limitations that should be addressed in future research. The first set of limitations address the type of application and scenes that may be efficiently rendered. Because the photon map is pre-computed and broadcast to each rendering chip, the scene and the illumination must remain static. This presents an obstacle to applications where the dynamic motion of objects, not just the viewpoint, is required.

A second limitation on the scenes is the permissible size of the scene representation. The entire scene including material properties and textures must fit in the relatively small, dedicated memories of each rendering chip. This will be a concern for larger environments or scenes highly detailed with geometry or textures.

Scenes with many glossy surfaces or complicated geometry are unable to obtain a significant benefit from irradiance caching. Outdoor natural scenes with leafy plants and glass and chrome office interiors, for example, will require more final gathers. In contrast, importance sampling works best when the surfaces are glossy or the illumination localized to a few regions of the visible hemisphere. The current system relies on the operator to select the number of final gather rays to be generated. An adaptive system would be beneficial, but requires that some computations be fully resolved before determining if others should be computed. It is therefore unclear how to incorporate adaptive sampling with photon gather reordering.

A further limitation of combined importance sampling, as presented in this dissertation, is using a tabulated probability distribution function. High frequencies in the surface reflectance or incident radiance functions are lost or aliased. This reduces the ef-

ficiency of the sampling strategy. Research should be performed on applying the general idea, of merging multiple sampling strategies into one, to p.d.f.s that can represent sharp functions.

The architecture simulation was only performed at the functional level. A cycle-accurate simulation of an actual hardware design would provide not only higher confidence in the architecture but also a fuller sense of the cost of implementation, performance of operation, and correctness of design.

8.3 Future Work

The simulated performance of the architecture presented in Chapter 7 is very promising and suggests a possible direction for fifth-generation graphics hardware architectures. However, the limitations just discussed show that there is more work that can and should be performed to increase the quality and efficiency of realistic image generation. In this section, I provide an brief glimpse of some of the areas I feel should be addressed, both in and outside the context of the architecture.

Improved gather reordering algorithms: There are alternative algorithms for reordering to be investigated. They must decrease memory bandwidth without incurring a significantly higher cost elsewhere in the system. An interesting possibility leverages information already present in the architecture. To accelerate ray casting, the scene geometry is stored in a data structure which subdivides space. This information could be harvested during ray traversal and used to reorder photon gathers.

A second approach is to make direct use of the Hilbert reordering by adapting the cache-efficient algorithm presented by Liu and Snoeyink for hardware implementation (Liu and Snoeyink, 2007). As with all fully deferred reorderings however,

the entire list of photon gather requests must be computed and stored. For large tile sizes this will require more intermediate storage than is practical to keep on-chip, the benefit of streaming the requests to memory must be weighed against the reduction of bandwidth allocated to photon gatherer.

Radiance caching: Radiance caching, like irradiance caching, is a process that reduces the number of final gathers by interpolating previously computed values (Křivánek et al., 2005). However, unlike irradiance caching, it can be used on glossy surfaces, increasing the proportion of scenes that can significantly benefit from the reduction. The tradeoff is an expensive compression step after each final gather to reduce the storage costs until future pixels determine if they can use this value in their interpolation.

When all final gathers are performed in parallel, as is done for a tile in the architecture presented in Chapter 7, an alternative emerges. Each photon gather request could carry a list of all the pixels that intend to interpolate this result, along with their partially evaluated BRDF values. The effect would be the same as having performed a full final gather for each pixel except that the final gather directions are correlated, adding a bias also present in standard irradiance caching. Although the computational requirements will not be reduced, bandwidth to the photon map should be significantly reduced.

Other applications of combined importance sampling: The combined importance sampling technique is notable for its simplicity. Following the example of Jensen it could be implemented in a path-tracing renderer which uses the photon map only for importance sampling (Jensen, 1995).

As used in this dissertation, combined importance sampling requires that each sampling strategy be expressed as a table. For many functions with sharp features, a tiny light source for example, this requires a tradeoff between an inefficient stor-

age mechanism or loss of previously known information, reducing the quality of generated samples. Although the probability distribution functions used in this dissertation were represented as tables, the technique could be applied whenever the p.d.f.s are expressed using the same basis functions. A method that shows promise is to express the probabilities with spherical harmonics, which are linearly fit, and average the components.

Further development of the architecture: The architecture in Chapter 7 has only been functionally simulated. A lower level simulation should be performed in order to directly measure expected performance. This will increase confidence in the completeness of the architecture and suggest improvements.

A potentially serious limitation of the architecture is the current requirement for strict replication of the entire photon map and scene among all rendering chips. This places a hard limit on the complexity of the scene that can be rendered, regardless of the total amount of memory in the machine. While processing each tile, the final gather rays may need to be cast throughout the entire scene and photon gathers performed throughout the entire photon map. This makes it inefficient to use the host as a large virtual memory.

One possible solution, worth further examination, is to relax the rule against communication between rendering chips. As final gather rays leave a region of the scene they could be routed to a rendering chip which holds that portion of the scene. Load balancing will however become an important issue, as a single overloaded rendering chip could slow a large machine dramatially.

The architecture as presented is also limited to static scenes and illumination. Many interactive applications require support for the dynamic motion of objects in a scene. Recently there has been work on supporting interactive ray casting in hardware (Woop et al., 2005). However, dynamic objects will also effect the

illumination in a scene. The photon map must be either recreated from scratch or updated every frame. The overall cost of creating a photon map is not high compared to the cost of image rendering. It should be possible to extend the hardware architecture to handle this task directly, instead of relying on the host.

8.4 Conclusion

It is not yet obvious which fundamental rendering algorithms will emerge in commercial fifth-generation graphics hardware architectures. In this dissertation, I have presented one possible system based on the photon mapping algorithm. The general approaches from high performance computing that I have used, such as conserving memory bandwidth by reordering computations, can be applied to other existing global illumination algorithms. Alternatively, there is a compelling argument for new global illumination algorithms designed from the outset for a finely-grained parallel implementation, recognizing the high cost of memory accesses.

As dedicated architectures for more of these algorithms are developed, it may become obvious that some specialized processing elements, such as the reordering queues and the *kd*-tree traversal units, are common in order to achieve high performance. The trend in current GPUs is towards greater programability. However, the addition of these common processing elements could provide an evolutionary transitional step towards a single high performance architecture supporting multiple global illumination algorithms.

BIBLIOGRAPHY

Agarwal, S., Ramamoorthi, R., Belongie, S., and Jensen, H. W. (2003). Structured importance sampling of environment maps. *ACM Transactions on Graphics*, 22(3):605–612.

Akeley, K. (1993). Reality Engine graphics. In *Proceedings of ACM SIGGRAPH 1993*, ACM Press / ACM SIGGRAPH, pages 109–116. Computer Graphics Proceedings, Annual Conference Series, ACM.

Akeley, K. and Hanrahan, P. (2001). Real-time graphics architectures. `http://graphics.stanford.edu/courses/cs448a-01-fall/`. Accessed August 10, 2006.

ARTVPS (2007). ARTVPS homepage. `http://www.artvps.com/`. Accessed January, 27, 2007.

ATI (2007). ATI homepage. `http://www.ati.com/`. Accessed April 8, 2007.

Bayer, R. and McCreight, E. (1972). Organization and maintenance of large ordered indexes. *Acta Informatica*, 1(3):173–189.

Bentley, J. L. (1975). Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517.

Blasi, P., Saëc, B. L., and Schlick, C. (1994). An importance driven Monte-Carlo solution to the global illumination problem. In *Rendering Techniques '94 (Proceedings of the Fifth Eurographics Workshop on Rendering)*, pages 173–183.

Blinn, J. and Newell, M. (1976). Texture and reflection in computer generated images. *Communications of the ACM*, 19(10):542–547.

Burke, D., Ghosh, A., and Heidrich, W. (2005). Bidirectional importance sampling for direct illumination. In *Proceedings of Eurographics Symposium on Rendering 2005*, pages 147–156.

Bustillo, E. (1997). A neuro-evolutionary unbiased global illumination algorithm. In *Rendering Techniques '97 (Proceedings of the Eigth Eurographics Workshop on Rendering)*, pages 263–274.

Cammarano, M. and Jensen, H. W. (2002). Time dependent photon mapping. In *Proceedings of the 13th Eurographics Workshop on Rendering*, pages 135–144.

Catmull, E. (1974). *A Subdivision Algorithm for Computer Display of Curved Surfaces*. PhD thesis, University of Utah.

Chalmers, A., Davis, T., and Reinhard, E. (2002). *Practical Parallel Rendering*. A.K. Peters.

Christensen, P. H. (1999). Faster photon map global illumination. *Journal of Graphics Tools*, 4(3):1–10.

Christensen, P. H. and Batali, D. (2004). An irradiance atlas for global illumination in complex production scenes. In *Proceedings of Eurographics Symposium on Rendering 2004*, pages 133–141.

Clarberg, P., Jarosz, W., Akenine-Möller, T., and Jensen, H. W. (2005). Wavelet importance sampling: efficiently evaluating products of complex functions. *ACM Transactions on Graphics*, 24(3):1166–1175.

Cohen, M. F. and Wallace, J. R. (1993). *Radiosity and Realistic Image Synthesis*. Academic Press Professional, Boston.

Cook, R. L., Porter, T., and Carpenter, L. (1984). Distributed ray tracing. *Computer Graphics (Proceedings of ACM SIGGRAPH 84)*, 18(3):137–145.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2001). *Introduction to Algorithms*. The MIT Press, Cambridge, MA, second edition.

Crandall, R. W. and Sidak, J. G. (2006). Video Games: Serious business for America's economy. `http://www.theesa.com/files/VideoGames-Final.pdf`. Accessed October 9, 2006.

Dachille, Jr., F. and Kaufman, A. (2000). GI-Cube: an architecture for volumetric global illumination and rendering. In *Proceedings of the SIGGRAPH/Eurographics Workshop on Graphics Hardware*, pages 119–128.

Dally, W. J. and Poulton, J. W. (1998). *Digital Systems Engineering*. Cambridge University Press.

Debevec, P. E. (1998). Rendering synthetic objects into real scenes: Bridging traditional and image-based graphics with global illumination and high dynamic range photography. In *Proceedings of ACM SIGGRAPH 1998*, ACM Press / ACM SIGGRAPH, pages 189–198. Computer Graphics Proceedings, Annual Conference Series, ACM.

Dutré, P. (1994). Importance-driven Monte Carlo light tracing. In *Rendering Techniques '94 (Proceedings of the Fifth Eurographics Workshop on Rendering)*, pages 185–194.

Dutré, P., Bekaert, P., and Bala, K. (2003). *Advanced Global Illumination*. A. K. Peters, Ltd., Boston.

Eyles, J., Molnar, S., Poulton, J., Greer, T., Lastra, A., England, N., and Westover, L. (1997). PixelFlow: the realization. In *HWWS '97: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, pages 57–68.

Faloutsos, C. and Roseman, S. (1989). Fractals for secondary key retrieval. In *PODS '89: Proceedings of the Eighth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 247–252.

Foley, T. and Sugerman, J. (2005). KD-tree acceleration structures for a GPU raytracer. In *HWWS '05: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*, pages 15–22, New York, NY, USA. ACM Press.

Gionis, A., Indyk, P., and Motwani, R. (1999). Similarity search in high dimensions via hashing. In *Proceedings of the International Conference on Very Large Databases*, pages 518–529.

Glassner, A. (1995). *Principles of Digital Image Synthesis*. The Morgan Kaufmann Series in Computer Graphics. Morgan Kaufmann.

Goral, C. M., Torrance, K. E., Greenberg, D. P., and Battaile, B. (1984). Modeling the interaction of light between diffuse surfaces. *Computer Graphics (Proceedings of ACM SIGGRAPH 84)*, 18(3):213–222.

Gortler, S. J., Grzeszczuk, R., Szeliski, R., and Cohen, M. F. (1996). The lumigraph. In *Proceedings of ACM SIGGRAPH 1996*, ACM Press / ACM SIGGRAPH, pages 43–54. Computer Graphics Proceedings, Annual Conference Series, ACM.

Gosselin, D. (2004). Phat Lewt: Drawing a diamond. In *Game Developer's Conference*.

Greene, N. (1986). Environment mapping and other applications of world projections. *IEEE Computer Graphics and Applications*, 6(11):21–29.

Hamming, R. W. (1973). *Numerical Methods for Scientists and Engineers*. McGraw-Hill, Inc., second edition.

Havran, V. (2000). Analysis of cache sensitive representations for binary space partitioning trees. *Informatica*, 23(3):203–210.

Heidmann, T. (1991). Real shadows, real time. *Iris Universe*, pages 23–31.

Hey, H. and Purgathofer, W. (2002). Importance sampling with hemispherical particle footprints. In *Proceedings of the 18th Spring Conference on Computer Graphics*, pages 107–114. ACM Press.

Hill, M. D. and Smith, A. J. (1989). Evaluating associativity in CPU caches. *IEEE Transactions on Computers*, 38(12):1612–1630.

Indyk, P., Motwani, R., Raghavan, P., and Vempala, S. (1997). Locality-preserving hashing in multidimensional spaces. In *STOC '97: Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing*, pages 618–625. ACM Press.

Intel (2003). Microburst Architecture. *IA-32 Intel Architecture Software Developers Manual*, 1:37–40.

ITRS (2006). International technology roadmap for semiconductors: Executive summary. `http://www.itrs.net/links/2006Update/2006UpdateFinal.htm`. Accessed January 27, 2007.

Jarosz, W. (2004). `http://renderedrealities.net/`. Accessed April 2004.

Jensen, H. W. (1995). Importance driven path tracing using the photon map. In *Rendering Techniques '95 (Proceedings of the Sixth Eurographics Workshop on Rendering)*, pages 326–335.

Jensen, H. W. (1996a). Global illumination using photon maps. In *Rendering Techniques '96 (Proceedings of the Seventh Eurographics Workshop on Rendering)*, pages 21–30.

Jensen, H. W. (1996b). *The Photon Map in Global Illumination*. PhD thesis, Technical University of Denmark.

Jensen, H. W. (2001). *Realistic Image synthesis Using Photon Mapping*. AK Peters.

Jensen, H. W. and Christensen, N. J. (1995). Photon maps in bi-directional Monte Carlo ray tracing of complex objects. *Computer & Graphics*, 19(2):215–223.

Kajiya, J. T. (1986). The rendering equation. *Computer Graphics (Proceedings of ACM SIGGRAPH 86)*, 20(4):143–150.

Křivánek, J., Gautron, P., Pattanaik, S., and Bouatouch, K. (2005). Radiance caching for efficient global illumination computation. *IEEE Transactions on Visualization and Computer Graphics*, 11(5):1–11.

Lafortune, E. P. and Willems, Y. D. (1993). Bi-directional path tracing. In Santo, H. P., editor, *Proceedings of Third International Conference on Computational Graphics and Visualization Techniques (Compugraphics '93)*, pages 145–153, Alvor, Portugal.

Lafortune, E. P. and Williams, Y. D. (1995). A 5D tree to reduce the variance of Monte Carlo ray tracing. In *Rendering Techniques '95 (Proceedings of the Sixth Eurographics Workshop on Rendering)*, pages 11–20.

Lange, B. (1991). The simulation of radiant light transfer with stochastic ray-tracing. In *Photorealistic Rendering in Computer Graphics (Proceedings of the Second Eurographics Workshop on Rendering)*.

Lastra, A. (2006). Table of historical GPU performance. `http://www.cs.unc.edu/~lastra/Research/GPU_performance.html`. Accessed August 2, 2006.

Lawrence, J., Rusinkiewicz, S., and Ramamoorthi, R. (2005). Adaptive numerical cumulative distribution functions for efficient importance sampling. In *Proceedings of Eurographics Symposium on Rendering 2005*.

Lawrence, J., Rusinkiwicz, S., and Ramamoorthi, R. (2004). Efficient BRDF importance sampling using a factored representation. *ACM Transactions on Graphics*, 23(3):496–505.

Levoy, M. and Hanrahan, P. (1996). Light field rendering. In *Proceedings of ACM SIGGRAPH 1996*, ACM Press / ACM SIGGRAPH, pages 31–42. Computer Graphics Proceedings, Annual Conference Series, ACM.

Liu, Y. and Snoeyink, J. (2007). A notation for Hilbert curves to support multidimensional spatial indexing. In preparation.

Ma, V. C. H. and McCool, M. D. (2002). Low latency photon mapping using block hashing. In *HWWS '02: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics Hardware*, pages 89–99.

McCormack, J., McNamara, R., Gianos, C., Seiler, L., Jouppi, N. P., and Correll, K. (1998). Neon: a single-chip 3D workstation graphics accelerator. In *HWWS '98: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics Hardware*, pages 123–132.

Molnar, S., Cox, M., Ellsworth, D., and Fuchs, H. (1994). A sorting classification of parallel rendering. *IEEE Computer Graphics and Applications*, 14(4):23–32.

Montrym, J. and Moreton, H. (2005). The GeForce 6800. *IEEE Micro*, 25(2):41–51.

Moon, B., Jagadish, H. V., Faloutsos, C., and Saltz, J. H. (1996). Analysis of the clustering properties of the Hilbert space-filling curve. *Knowledge and Data Engineering*, 13(1):124–141.

MPAA (2006). U.S. theatrical market: 2005 statistics. `http://www.mpaa.org/researchStatistics.asp`. Accessed October 9, 2006.

Nicodemus, F. E., Richmond, J. C., Hsia, J. J., Ginsberg, I. W., and Limperis, T. (1977). Geometric considerations and nomenclature for reflectance. In *Monograph 161*. National Bureau of Standards (US).

NVIDIA (2007). NVIDIA homepage. `http://www.nvidia.com/`. Accessed April 8, 2007.

Pattanaik, S. N., Ferwerda, J. A., Torrance, K. E., and Greenberg, D. P. (1997). Validation of global illumination solutions through CCD camera measurements. In *Proceedings of the Fifth Color Imaging Conference, Society for Imaging Science and Technology*, pages 250–253.

Patterson, D. and Hennessy, J. (1996). *Computer Architecture: a Quantitative Approach*. Morgan Kaufmann, second edition.

PCISIG (2007). PCI-SIG Specifications. `http://www.pcisig.com/specifications/`. Accessed March 28, 2007.

Pharr, M. (2005). Extended photon map implementation. `http://www.pbrt.org/plugins/exphotonmap.pdf`. Accessed May 2005.

Pharr, M. and Humphreys, G. (2004). *Physically Based Rendering from Theory to Implementation*. Morgan Kaufmann.

Purcell, T. J., Buck, I., Mark, W. R., and Hanrahan, P. (2002). Ray tracing on programmable graphics hardware. *ACM Transactions on Graphics*, 21(3):703–712.

Purcell, T. J., Donner, C., Cammarano, M., Jensen, H. W., and Hanrahan, P. (2003). Photon mapping on programmable graphics hardware. In *HWWS '03: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*, pages 41–50.

Ramamoorthi, R. and Hanrahan, P. (2001). An efficient representation for irradiance environment maps. In *Proceedings of ACM SIGGRAPH 2001*, ACM Press / ACM SIGGRAPH, pages 497–500. Computer Graphics Proceedings, Annual Conference Series, ACM.

Robinson, J. T. (1981). The k-d-b-tree: a search structure for large multidimensional dynamic indexes. In *SIGMOD '81: Proceedings of the 1981 ACM SIGMOD International Conference on Management of Data*, pages 10–18.

Rogers, D. F. (1985). *Procedural Elements for Computer Graphics*. McGraw-Hill.

Rubinstein, R. Y. (1981). *Simulation of the Monte Carlo Method*. John Wiley & Sons.

Samet, H. (1990). *The Design and Analysis of Spatial Data Structures*. Addison-Wesley.

Schmittler, J., Wald, I., and Slusallek, P. (2002). SaarCOR: a hardware architecture for ray tracing. In *HWWS '02: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*, pages 27–36.

Schmittler, J., Woop, S., Wagner, D., Slusallek, P., and Paul, W. J. (2004). Realtime ray tracing of dynamic scenes on an FPGA chip. In *HWWS '04: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*, pages 95–106, New York, NY, USA. ACM Press.

Shirley, P. (1992). *Graphics Gems III*, chapter Nonuniform random point sets via warping, pages 80–82. Academic Press.

Shirley, P. (2000). *Realistic Ray Tracing*. A. K. Peters.

Steinhurst, J., Coombe, G., and Lastra, A. (2005). Reordering for cache conscious photon mapping. In *Graphics Interface*, pages 97–104.

Steinhurst, J. and Lastra, A. (2006). Global importance sampling of glossy surfaces using the photon map. In *IEEE Symposium on Interactive Ray Tracing 2006*, pages 133–138.

Szirmay-Kalos, L. (1999). Monte-Carlo global illumination methods - state of the art and new developments. In *Proceedings of the Fifteenth Spring Conference on Computer Graphics*, pages 3–21.

Veach, E. (1997). *Robust Monte Carlo Methods for Light Transport Simulation*. PhD thesis, Stanford University.

Veach, E. and Guibas, L. J. (1994). Bidirectional estimators for light transport. In *Rendering Techniques '94 (Proceedings of the Fifth Eurographics Workshop on Rendering)*, pages 147–162.

Veach, E. and Guibas, L. J. (1995). Optimally combining sampling techniques for Monte Carlo rendering. In *Proceedings of ACM SIGGRAPH 1995*, ACM Press / ACM SIGGRAPH, pages 419–428. Computer Graphics Proceedings, Annual Conference Series, ACM.

Venkatasubramanian, S. (2003). The graphics card as a stream computer. In *SIGMOD-DIMACS Workshop on Management and Processing of Data Streams*.

Wald, I., Guenther, J., and Slusallek, P. (2004). Balancing considered harmful – faster photon mapping using the voxel volume heuristic. *Computer Graphics Forum (Proceedings of Eurographics 2004)*, 23(3):595–603.

Wald, I., Slusallek, P., Benthin, C., and Wagner, M. (2001). Interactive rendering with coherent ray tracing. *Computer Graphics Forum (Proceedings of Eurographics 2001)*, 20(3):153–164.

Ward, G. J. and Heckbert, P. S. (1992). Irradiance gradients. In *Rendering Techniques '92 (Proceedings of the Third Eurographics Workshop on Rendering)*, pages 85–98.

Ward, G. J., Rubinstein, F. M., and Clear, R. D. (1988). A ray tracing solution for diffuse interreflection. *Computer Graphics (Proceedings of ACM SIGGRAPH 88)*, 22(4):85–92.

Whitted, T. (1980). An improved illumination model for shaded display. *Communications of the ACM*, 23(6):343–349.

Woop, S., Schmittler, J., and Slusallek, P. (2005). RPU: a programmable ray processing unit for realtime ray tracing. *ACM Transactions on Graphics*, 24(3):434–444.