ITERATIVE ADAPTIVE SAMPLING FOR ACCURATE DIRECT ILLUMINATION

A Thesis

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Master of Science

by

Michael Donikian

August 2004

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
10 AUG 2004	**N/A**	**-**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Iterative Adaptive Sampling For Accurate Direct Illumination	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Cornell University**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release, distribution unlimited

13. SUPPLEMENTARY NOTES
The original document contains color images.

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**UU**	**96**	

ABSTRACT

This thesis introduces a new multipass algorithm, Iterative Adaptive Sampling, for efficiently computing the direct illumination in scenes with many lights, including area lights that cause realistic soft shadows. Real world architectural scenes frequently contain large numbers of lights; however many current algorithms do not scale well in performance when rendering these types of scenes.

Our algorithm is based upon an observation that although many hundreds of lights may contribute to the illumination of a single image, much lower lighting complexity typically exists on a localized basis within subsections of the image. Since the predominant cost of computing the direct illumination at a point is the testing of light source visibility, our algorithm works to exploit this observation of low localized lighting complexity to reduce the number of visibility tests (shadow rays) needed to accurately render each pixel.

This reduction of shadow rays is made possible by sampling light sources in proportion to their actual contribution to a pixel's luminance value. We do this by iteratively modifying a probability density function (PDF) until it adaptively captures the local lighting configuration. We use sample data collected during rendering as feedback to drive the optimization of the PDF. Our algorithm takes advantage of coherence in image space by aggregating sample data on both a per-pixel and per-block level as well as coherence in world space by aggregating sample data on light clusters. We have tested this algorithm on several complex lighting environments and demonstrated roughly an order of magnitude improvement over standard procedures.

Biographical Sketch

Not much is known about Michael Donikian since he tends to be pretty bad when it comes to writing these biographical sketches. They say he grew up on the mean streets of New York City and decided to come to Ithaca to pursue his undergraduate and graduate education. It is rumored that he will be moving to Florida as soon as this thesis gets published... something about the government wanting him.

For great justice...

Acknowledgments

First of all I would like to thank my advisor Professor Donald P. Greenberg for giving me the opportunity to study here at the PCG. It was a wonderful opportunity to learn new things and to get a good perspective on the research world of Computer Graphics.

I would also like to thank the "Many Lights" team of Bruce Walter, Kavita Bala and Sebastian Fernandez. I will miss the times we spent together over a cup (or two, or three, or four...) of espresso. I probably won't miss the crazy late hours we all put in for those two papers, but I'll have fond memories of how hard we all worked together.

I'd also like to thank the following people, groups, organizations, places, sports, and miscellaneous items.

- The Air Force Institute of Technology (AFIT) for offering me a chance to pursue a master's degree for my first assignment as an active duty Air Force officer.

- All my program managers at AFIT (Major Melissa Flattery and Lieutenants Angela Bjorge, Sharmine Lynch, and Timothy Adams) for helping me get through this and working with all my leave and extension requests.

- Captain Anthony Gamboa at the Air Force Personnel Center, for being pa-

tient with me and working with me on my getting next assignment.

- The Air Force Research Lab in Rome NY, specifically people in IFSB I worked with on a daily basis. If it wasn't for my very positive experience there, I probably would not have chosen Computer Graphics as something I'd like to pursue in greater depth. I'd especially like to thank Jason Moore and Steve Farr for writing recommendations and Chad Salisbury for getting me into motorcycles and helping me pick one out.

- Professor Julia D'Souza for being my minor advisor and for opening me up to finance and management.

- Adam Kravetz, for introducing me to road and mountain biking. It's something I'll probably take with me for the rest of my life. And of course, how can I forget all the crazy conversations about "high-quality problems" we had while spending our final weeks in the lab finishing up our theses.

- Jacky Bibliowicz, for pursuing an "unofficial minor" with me on a most interesting topic. We have some great stories to tell for years to come.

- Henry Letteron, for bringing in all the food and candy every day and for playing a couple of innings of office baseball with me.

- Vikash Goel, for all the time you spent with me on my car and for all the interesting conversations on automobile technology.

- Will Stokes, for his random commentary.

- Hurf Sheldon, for tipping the scales in favor of getting a motorcycle and of course for all the work you do maintaining our network and computer systems.

- Linda Stephenson, for keeping me on Don's radar at all times and for helping me fax all those things to the Air Force. I don't know if I would have been able to graduate if you had not been around.

- Peggy Andersen, for keeping a well stocked cabinet of office supplies, for helping me buy whatever software I needed, and for offering to help me finish this thesis in every way possible.

- Moreno Piccolotto, for getting me (and half the lab) hooked on espresso.

- Martin Berggren, for helping me figure out how to use all the lab's audio/video equipment.

- Mary, for keeping our offices clean and for feeding us when we're overworked and starving.

- John Mollis, for being a cool laid-back officemate and for having someone with whom to go on motorcycle rides.

- Ryan Ismert, for helping me pick out cool classes for my minor.

- Jeremiah Fairbank, for answering all my questions on using 3ds max.

- Alex Liberman, for really coming through and helping me move.

- Taekwondo, Boxing, the Teagle Hall weight room, Golf, and the roads of Tompkins County for being an outlet for my excess physical energy.

- Everyone at AFROTC Det 520 – Colonel Donald Hoover, Majors Byron Breese and Tracy Higgins, Captain Michael Mowry, Sergeants Robert Roy and Sara Heit, and of course what would I do without Helen Jessop and all the verbal smack-talk battles we had going over the course of two years.

- Kevin Seaman, for introducing me to boxing (and a whole lot of other martial arts), for giving me a new way to look at the world and for making me a much better athlete.

- David D, for all the humor and for showing me the lesson to take away from a negative experience.

- Tool, Disturbed, System of a Down, A Perfect Circle, Linkin Park, The Beatles, Metallica, Rage Against The Machine, The Offspring, Staind, Evanescence, Papa Roach, Stone Temple Pilots, Earshot, Econoline Crush, Year of the Rabbit, Korn, Limp Bizkit, Godsmack, Taproot, Chevelle, Adema, Cold, and Megadeth, for giving me something to listen to while I wrote this thesis.

- Master Han Cho and the members of Sport TKD, who have been like a second family to me during my six-year stay at Cornell. We've shared both the good times and the bad.

- and last but not least, my parents, for always being there for me.

The views expressed in this article are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the U.S. Government.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Real world architectural scenes contain many lights. In computer graphics models, a light can be represented by a point light approximation or by a light source with a finite area. Area lights are critical for physically-correct rendering because they cast realistic soft shadows unlike the synthetic-looking hard shadows of point lights. In addition, soft shadows are important because they provide visual cues on the relative positioning of objects through the hardness of the shadow boundary. We can also approximate light coming from distant sources such as skylight by using an environment map. Environment maps are an effective means of adding natural outdoor lighting effects to a scene and can further enhance the realism of our rendering.

This realism comes at great cost however, as rendering shadows from large numbers of lights in general, and area lights and environment maps in particular, can be very computationally expensive. This cost is dominated by the casting of shadow rays to determine the visibility of the lights. For area lights and environment maps, visibility can be especially difficult to predict since these types of lights can have varying degrees of partial visibility.

Many current direct illumination algorithms are able to efficiently render soft shadows from area lights, but their execution time scales linearly with the number of lights in the scene. Since real world scenes often contain more than just a few lights, there exists a need for algorithms that scale sub-linearly in the number of lights. Some of the algorithms that are designed for scenes with many lights are dependent upon high occlusion where a majority of the lights present in the scene are not visible in a single image, but this is not always the case. We have made a set of observations regarding light source visibility and have found that in many architectural scenes there could be hundreds of lights visible in a single image. In contrast, we have noticed that on average at a single point only about 4-35% of the lights in the scene are visible. This work introduces an iterative multipass rendering algorithm that takes advantage of our finding that scenes with high global lighting complexity often contain low local lighting complexity.

Our algorithm divides the image into 8×8 pixel blocks for processing. Creating the image block-by-block allows our algorithm to have a compact memory footprint. It also exploits the fact that the illumination within a small region of the image will be relatively simple and coherent. Within each block we also subdivide into a characteristic set of surface points that can accurately represent the underlying geometric and shadow boundaries. For each surface point within the block, we construct a probability density function (PDF) over the light sources and then sample it to estimate the illumination using a small number of shadow rays. In the simple case, we use only one surface point per pixel. If an anti-aliased image is desired, we generate multiple surface points per pixel.

Rendering time will depend upon how similar the PDF is to the function representing the illumination coming from the light sources. The mean value of the

light samples will quickly converge to be visually indistinguishable from the actual illumination if the PDF is a good estimate; however, finding a good PDF is a difficult task because it requires knowing the solution to the problem we are trying to solve. Our algorithm starts with a simple PDF and uses the resulting sample data to modify the PDF in between rendering passes over the image block. Each subsequent rendering pass will use an adaptively improved PDF constructed using feedback from samples computed thus far.

We construct our PDFs using a range of sample granularities over the image plane: a coarse granularity of 8×8 pixel blocks and a finer granularity of per-pixel sample data. We create block- and pixel-based PDFs between each pass, using all sample information from previous passes to refine our sampling PDF while ensuring that the resultant sampling PDF is unbiased. A weighted combination of these PDFs initially uses block information to guide sampling. As more samples are collected the pixel PDFs get increasingly more accurate and therefore are used to guide adaptive sampling. By using locally-adaptive PDFs, our algorithm evaluates significantly fewer shadow rays than a standard ray tracer to achieve the same image quality. Since tracing shadow rays is the dominant cost in a ray tracer, our performance improvement is a direct result of our adaptive PDFs. Our algorithm evaluates significantly fewer shadow rays than previous approaches, yet provides the same level of quality.

For efficiency, we also cluster light sources to aggregate visibility information on a spatial basis over several lights. This clustering of light sources also reduces the effective number of lights in the scene, which allows us to more efficiently build our PDFs. In addition we use the clusters to stratify our sampling of the light sources for the radiance calculation at each surface intersection point.

Since our system takes advantage of coherence in both image space and world space, we have been able to reduce the time it takes to render scenes with many point lights and area lights. We have achieved speedups of nearly one order of magnitude for complex lighting environments including scenes containing direct illumination from scenes with many point lights, area lights, and environment maps.

In the following chapter (Chapter 2), we will provide some background on direct illumination concepts and also explore some of the previous work that has been done to accelerate direct illumination rendering. In Chapter 3 we will discuss how our algorithm is specifically designed to render scenes with complex lighting. In Chapter 4, we will show the effectiveness of our algorithm by comparing our results to that of an industry standard reference solution. Chapter 5 will be the conclusion and summary of the benefits and contribution our approach to rendering direct illumination for complex scenes.

Chapter 2

Related Work

The beginning of this chapter will discuss some of the background behind direct illumination, ray tracing, and Monte Carlo integration. The rest of this chapter will provide a summary of previous work relating to direct lighting starting with techniques designed for accurately and efficiently rendering area lights and transitioning to techniques designed to handle scenes with many lights. The end of this chapter will also cover some of the research that has been conducted regarding direct illumination from environment maps.

2.1 Background

2.1.1 Types of Light Sources

There are many different types of light sources in computer graphics. The simplest is the point light source in which light is emitted from an infinitesimally small point. Point light sources can be omni-directional and emit light evenly in all directions, or they can have an arbitrary directional distribution or orientation. Directional

light sources are point lights that are infinitely far away and emit light in the same direction regardless of the point in the scene they illuminate. The sun can be approximated by a directional light source over small surfaces on the earth such as a building or a small town. Area light sources emit light from a surface of finite area. We can best represent real world light sources by area light sources, but unfortunately they are very difficult to accurately render in computer graphics. Like point light sources, area lights can also be omni-directional (e.g. a sphere), or oriented (e.g. a planar surface). Another source of lighting is the environment map. We discuss this in more detail in section 2.4.

2.1.2 Rendering Equation

Computing the direct illumination at a point involves integrating the contributions from all the lights in the scene as:

$$L(\vec{x}) = \int_S V(\vec{x},\vec{y})\, f_r(\vec{x},\vec{y})\, L_e(\vec{x},\vec{y})\, G(\vec{x},\vec{y})\, d\vec{y} \tag{2.1}$$

where

- $L(\vec{x})$ is the exitant radiance reflected from a point $\vec{x}$ toward the eye due to direct illumination.
- S is the set of light sources
- $\vec{y}$ is a point on a light source
- $V(\vec{x},\, \vec{y})$ is the visibility of $\vec{y}$ from $\vec{x}$
- $f_r(\vec{x},\, \vec{y})$ is the BRDF (Bidirectional Reflectance Distribution Function) at point $\vec{x}$ evaluated for the viewing and light direction. It defines how lights

bounces when it strikes a surface and represents the material properties of a point $\vec{x}$.

- $L_e(\vec{x}, \vec{y})$ is the emitted radiance from $\vec{y}$ in the direction of $\vec{x}$
- $G(\vec{x}, \vec{y})$ is purely geometric and depends on the type of light we are dealing with and its orientation and distance from $\vec{x}$.

In simple cases, such as scenes with only a few point light sources, this equation can be solved exactly using ray tracing. Ray tracing is also an effective solution for computing direct illumination at interactive rates for simple scenes. As the complexity of the scene, lights, and materials increases, the cost of an exact direct illumination solution rapidly becomes prohibitive. This cost is usually dominated by determining visibility which is expensive to compute and difficult to predict. As a result, most direct illumination algorithms have focused on reducing the cost of evaluating visibility or finding ways to avoid the evaluation entirely. The remainder of this chapter will examine some of the techniques that have been developed to accomplish this goal.

2.2 Single Light Techniques

The techniques that follow are optimized for computing the illumination and shadows due to a single area light source. They can be used on scenes with more lights but run time complexity will scale linearly with light count. Rendering times will be unacceptably high for scenes with hundreds of lights, thus they are best used for scenes with low lighting complexity.

Amanatides [Ama84] introduce a technique called Cone Tracing. Assuming circular or spherical light sources, they construct a cone from the point to be ren-

dered to the light source. The proportion of the cone obstructed by scene geometry indicates the intensity level of the penumbra or soft shadow region. This approach has two problems. First, it assumes spherical or circular light sources. Second, analytical intersections with cones can be expensive to compute for arbitrary geometry.

Stewart [SG94] analytically computes the umbral and penumbral discontinuity events for every light against every object in the scene. This information is then used to generate a mesh that will not cross lighting discontinuities. This approach, while effective, is limited to purely polygonal environments. Furthermore, the algorithm identifies discontinuities which may not be perceptually apparent in an environment with many lights and thus is overly conservative.

Soler and Sillion [SS98] approximate soft shadows for interactive viewing using an image-processing approach. They find objects that are at similar distances from the light plane and project them onto a single plane. They then take advantage of the fact that when the light, occluder, and receiver lie on parallel planes, the shape of the shadow is a convolution of the shape of the light with the shape of the occluder. Using hardware convolution, they obtain soft shadows at interactive rates but unfortunately this approximation breaks down when the occluders are mostly perpendicular to the light source.

Hart et al. [HDG99] use an image-plane based flood-fill to propagate occluder information from pixel to pixel. With a list of the blockers affecting each light at each pixel, and under the assumption that the environment is polygonal, they were able to analytically compute soft shadows for environments with reasonable complexity, though it was never tested with real world architectural scenes.

Parker et al. [PSS98] render soft shadows by casting an individual ray per-pixel,

per-light and modulating the visibility based on how close the ray came to intersecting an object. Although this approach can quickly generate approximations of soft shadows, the algorithm has problems dealing with multiple occluders. It is also limited in that it can only work with spherical light sources and it is unclear how well it would perform under greater lighting complexity.

Agrawala et al. [ARHM00] present two approaches to dealing with area lights. The first combines multiple shadow maps into a single layered shadow map. This allows for fast soft shadows but introduces significant errors since interpolation of shadow maps can be an inaccurate approximation. Their second approach computes soft shadows by densely sampling the light source, but relying on coherence in the shaded points to reduce the number of actual cast shadow rays. One drawback is that this approach must reproject a potentially large number of occlusion points per light. Such an approach does not scale well when the environment consists of a large number of lights.

Akenine-Moeller [AMA02] and Assarsson [AAM03] approximate the soft shadows cast by area lights through the use of "penumbra wedges". The algorithm operates in two stages, first determining a hard shadow through shadow volumes as if only the center of the light was illuminating the scene. The second pass attempts to correct the visibility by computing the amount of light coverage along the silhouette of the occluding object. This approach generates realistic-looking soft shadows at interactive rates for individual lights. However, because they generate the shadows cast by each object independently, they cannot correctly render shadows cast by multiple objects whose shadows overlap.

2.3 Many Light Techniques

When dealing with real world scenes that contain hundreds of lights, it is critical to use an algorithm that is specifically designed for the task. There are several publications that deal explicitly with the problem of rendering scenes with large numbers of lights. The scalability of these algorithms with respect to the number of lights has not yet been clearly studied, but they certainly reduce the cost of rendering scenes with many lights. Some algorithms are designed for still images while other are designed for interactive use.

Ward [War94] accelerates the rendering of many lights using a user-specified threshold to eliminate lights of low importance. For each pixel in an image, the system sorts the lights according to their maximum possible contribution (assuming no occlusion). Occlusion for each of the largest possible contributors at the pixel is tested, measuring their actual contribution to the pixel, and stopping when the total energy of the remaining lights reaches a predetermined threshold. This approach can reduce the number of occlusion tests; however, it does not reduce the cost of the occlusion tests that do have to be performed and does not do very well when illumination is uniform.

Shirley et al. [SWZ96] propose an approach that subdivides the scene into voxels and, for each voxel, partitions the set of lights into an important set and an unimportant set. Each light in the important set is sampled explicitly. One light is picked at random from the unimportant set as a representative of the set and sampled. The assumption is that the unimportant lights all contribute the same amount of energy.

To determine the set of important lights, the authors construct an "influence box" around each light. An influence box contains all points on which the light

could contribute more than the threshold amount of energy. This threshold is assigned somewhat arbitrarily and not according to any perceptual metrics. This box is intersected with voxels in the scene to determine when the light is important. This is an effective way to deal with many lights when rendering static scenes and viewpoints. The chief drawback of this system is that it does not include the visibility term in the calculation of the influence box. Consequently, this algorithm is ineffective if there are occluders near the bright lights.

Paquette et al. [PPD98] present a light hierarchy for quickly rendering scenes with many lights. This system builds an octree around the set of lights, subdividing until there are less than a predetermined number of lights in each cell. Each octree cell then has a "virtual light" constructed for it that represents the illumination caused by all the lights within it. They derive error bounds which can determine when it is appropriate to shade a point with a particular virtual light representation and when traversal of the hierarchy to finer levels is necessary. Their algorithm can deal with thousands of point lights. The major limitation of this approach is that it does not take visibility (i.e., occlusion) into consideration.

The techniques previously mentioned can substantially reduce the amount of time to render scenes with high lighting complexity; however, even though they are aimed at producing high quality individual images, they still do not offer any form of efficient anti-aliasing. Anti-aliasing must be done through supersampling which can greatly increase the cost of rendering. In addition they all oversimplify the problem by ignoring the effect of the visibility term in their calculations. Some of them produce incorrect results, while others are just inefficient. In contrast, the techniques described below are designed for interactive use and typically produce artifacts and other rendering errors.

Fernandez et al. [FBG02] deals with scenes with many lights in an interactive setting. They voxelize the scene and maintain a list of visible lights and potential blockers for each voxel. This list is found through stochastic sampling which occurs asynchronously from rendering. Their system provides an order of magnitude speedup for small to medium sized scenes. They achieve good performance because they avoid evaluating shadow rays for lights that are either fully visible or fully occluded; however, it is unclear that their algorithm can scale to scenes with hundreds of lights or millions of polygons.

Wald et al. [WBS03] render complex environments of millions of polygons and thousands of lights at interactive rates. They do so by constructing a probability density function (PDF) of the light sources for the current image using a few paths in a path tracer. This PDF is then used to determine which lights to render for the current image. However, in order for this approach to work efficiently, they require environments with very high occlusion, where only a small number of light sources affect the illumination in any particular viewpoint.

2.4 Environment Maps

An environment map is an image that represents an infinitely far away background. Each pixel in an environment map translates to light coming from a specific direction from the background. In 1976, Blinn and Newell [BN76] introduced environment map rendering to computer graphics. They rendered realistic specular reflections off of surfaces by looking up the pixel values in the environment map. In 1986, Greene [Gre86] followed up on this work and also demonstrated how one could generate an environment map from a real world scene by using a fish eye

lens. Environment maps as used in these two papers are known more specifically as irradiance environment maps because each pixel has an irradiance value associated with it.

Miller and Hoffman [MH84] extended Blinn and Newell's original work by also using the environment map to render diffuse reflections. Prior to rendering they prefiltered the environment map by convolving it with a diffuse Bidirectional Reflectance Distribution Function (BRDF). The resulting image was termed a radiance environment map since the pixel values represented exitant radiances leaving the surface for a particular normal.

Cabral et al. [CMS87] showed how to efficiently convolve environment maps with glossy BRDF's by expanding the BRDF into spherical harmonics (SH). Because gloss is a view dependent reflection their rendering approach is limited to a single camera view. Since they only use a low number of SH coefficients, they are also limited to broad gloss lobes. In 1999, Cabral et al. [CON99] extend this work to allow a dynamic camera position. They create several radiance environment maps–one per camera–and use Image Based Rendering (IBR) to interpolate between the available reflection maps for new camera locations and orientations.

The problem with many of these techniques is that they require an expensive prefiltering step where the environment map must be convolved with each BRDF or material in the scene. This preprocessing can get even more expensive when dealing with real world scenes that can contain many materials. The prefiltering step also assumes full visibility, thus objects do not cast any shadows in the scene.

Kautz and McCool [KM00] demonstrated that it was possible to render glossy reflections with nearly arbitrary isotropic BRDFs at interactive rates. They represented their BRDF's as single or multiple lobes and used greedy local fitting

techniques to efficiently compute the lighting equation. Their system is limited in that it only supports BRDFs with radially symmetric lobes.

Kautz et al. [KVHS00] used a faster hierarchical method for generating radiance environment maps. They also showed how to use hardware acceleration to speed up the process to interactive rates. Furthermore, they extended their method to anisotropic BRDFs.

Ramamoorthi and Hanrahan [RH01] [RH02] introduce a new representation called the Spherical Harmonic Reflection Map (SHRM) which results in significantly faster preprocessing and rendering.

All these techniques, however, suffer from the problem of ignoring visibility completely. Shadows and reflections between objects are not handled, leading to limited applications for these algorithms. For example, they are not appropriate for architectural scenes or scenes with complex visibility and materials.

Sloan et al. [SKS02] introduce a prefiltering method that takes visibility into account. Their method captures effects like shadows, reflections, and caustics. They also allow for soft shadows and caustics from rigidly moving objects to be cast onto arbitrary, dynamic receivers; however, their examples are limited to low frequency lighting effects. Sharp shadows and other high frequency features are not supported due to the use of only low order spherical harmonics. The main drawback to this approach is the extremely high precomputation cost requiring a detailed ray traced sampling of visibility.

Agarwal et al. [ARBJ03] take a different approach to environment map rendering. Their method called Structured Importance Sampling stratifies the environment map into a number of regions which are preintegrated into a set of directional lights. Their metric for determining the sizes of the strata takes both the intensity

of the pixels and visibility coherence into account, though not the actual visibility of the strata in scenes. The primary benefit of their metric is that it prevents small bright regions from getting oversampled. Another major benefit of this approach is that it does not require an expensive preprocessing step. The drawback is the point lights do tend to cause banding in the shadows and it is not quite suitable for scenes with many small shadow features. It is possible however to randomly sample pixels in strata to achieve soft shadows, but this unfortunately introduces significant Monte Carlo noise in all but the simplest of scenes.

2.5 Conclusion

In contrast to many of the above works, the research we present in this thesis does take actual scene visibility into account. Furthermore, our algorithm is designed to quickly render high quality anti-aliased still images of scenes containing arbitrary geometric and lighting complexity. We also integrate [ARBJ03] for computing direct illumination from environment maps but without introducing any bias or perceptible noise in the shadow regions.

Chapter 3

Iterative Adaptive Sampling

The goal of this research is to create a high-quality software renderer that can effectively render real-world architectural scenes with arbitrary lighting complexity. We want to consider scenes that contain hundreds or even thousands of area lights, scenes that contain direct illumination from environment maps, and scenes with millions of geometric primitives.

In this chapter we will first discuss how our rendering algorithm is designed to meet these challenges and then we will describe our implementation. The first two sections (Sections 3.1 and 3.2) in this chapter provide necessary background information relevant to understanding some of the design choices in our algorithm. Section 3.3 is a high level overview of the algorithm. Section 3.4 discusses how our algorithm scales to handle scenes with many lights. The final two sections (Sections 3.5 and 3.6) provide the details about our implementation.

3.1 Characteristics of Architectural Scenes

One of our primary considerations in accelerating the rendering process is to consider real world lighting conditions. We want our renderer to be able to handle direct lighting from point lights, area lights, and environment maps. We do not want to place any limitations on the number of lights in the scene or the type of geometry in the scene.

When addressing the problem of direct illumination from many light sources, it is useful to consider some previous approaches for different types of real world scenes. One example for which solutions already exist is scenes with very high occlusion, where only a very small subset of lights contribute for each viewpoint. For example, in a large office building, the only lights that will contribute within each room might be the lights in the room itself and nearby lights in the hallway. These few lights can be found using either sparse sampling of the current viewpoint [WBS03] or through creating a set of cells (e.g. rooms) and portals (e.g. doors) [ARJ90], [TS91], and [LG95]. This will not be the case if the building has a large open lobby with a thousand lights. In many cases such as the lobby, occlusion is not so extreme; thus, we want to design an algorithm that can deal with a wider range of lighting characteristics.

In scenes with more open spaces or large architectural rooms, we find a different pattern of visibility of light sources. We have tested several scenes and found that typically only about 4-35% of the lights are visible from any particular point in the environment. This implies that when calculating the luminance for a particular surface point, it would be inefficient to consider all of the light sources. However, over an entire image nearly all the lights contribute significantly on some surface visible in the current view. Approaches based on high occlusion will perform poorly

on such scenes. The scene in Figure 3.1 an example of this kind of behavior. We present more detailed visibility statistics on several scenes in the results section of this thesis.

The rendering algorithm we present in this thesis provides good speedups for almost any architectural scene and can handle environments with very high global lighting complexity while still taking advantage of lower local lighting complexity whenever possible.

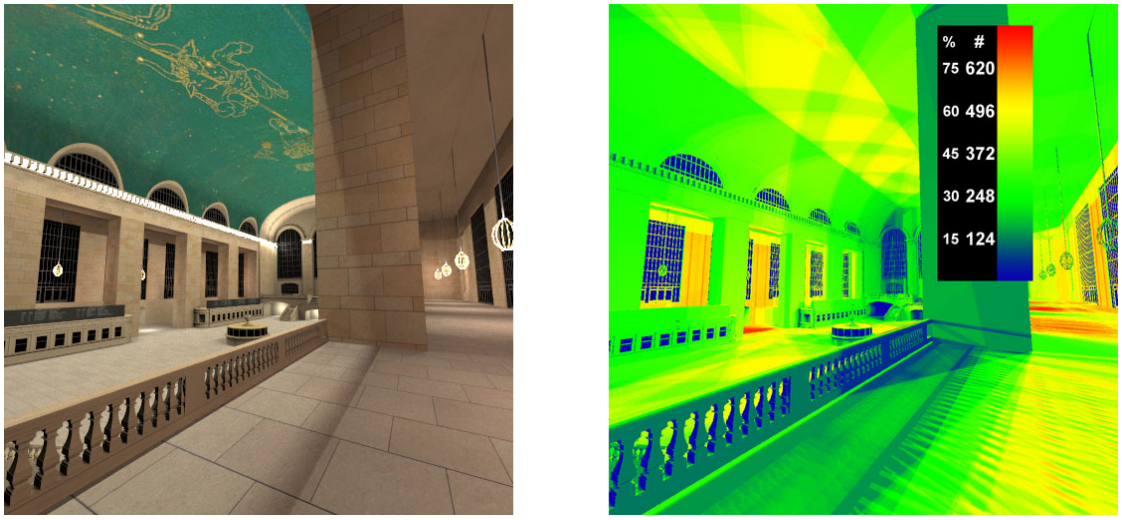

Figure 3.1: *On the left is a rendering of Grand Central Terminal. This model contains over 800 light sources with nearly all of them contributing to the illumination of some surface visible in this viewpoint. On average, however, less than 35% are visible at any particular point. On the right is a false-color visualization showing the number of lights visible from the surface points intersected by the view rays through each pixel.*

3.2 Monte-Carlo Integration

For many complex lighting situations, including area lights, one solution is to evaluate the direct lighting equation (Equation 2.1) using Monte-Carlo integration. Monte-Carlo integration is a powerful technique that allows one to evaluate an integral of arbitrary complexity using statistical sampling. The weakness of Monte-Carlo integration is that it may take an unreasonably large number of samples and thus a large amount of computation time, to arrive at a correct final result.

To evaluate direct illumination at a point $\vec{x}$ using Monte-Carlo integration we generate N sample points $\{\vec{y}_1, \vec{y}_2, ...\vec{y}_N\}$[1] on the lights according to a probability density function (PDF) $p()$ to get:

$$L(\vec{x}) \approx \frac{1}{N} \sum_{i=1}^{N} \frac{V(\vec{x}, \vec{y}_i)\, f_r(\vec{x}, \vec{y}_i)\, L_e(\vec{x}, \vec{y}_i)\, G(\vec{x}, \vec{y}_i)}{p(\vec{y}_i)} \tag{3.1}$$

The amount of noise or error in this estimator and hence the number of samples needed to produce a sufficiently good estimate is strongly dependent on the probability density function $p()$. To determine the amount of error we use a variance metric. Variance is a measure of the average difference between a set of data points and their expected value. For direct illumination, the expected value is $L(\vec{x})$, and each data point is the result of evaluating the direct illumination equation for a sample $\vec{y}_i$.

A poor choice for the PDF will require much more sampling than a PDF that closely represents the function we are trying to estimate. Unfortunately, finding a

[1]The nomenclature for points and rays may cause some confusion. Point $\vec{x}$ is found by tracing a ray from the camera through a pixel and into the scene. This ray is also referred to as a primary visibility ray or an eye ray. The first surface point this ray intersects is our point $\vec{x}$. Point $\vec{y}_i$ is a sample point on a light source randomly selected according to our probability density function $p()$. The ray from $\vec{x}$ to $\vec{y}$ is called a shadow ray and is used to evaluate the visibility of $\vec{y}$ from $\vec{x}$.

good PDF is extremely difficult in part because it depends on the visibility function $V()$, which is both hard to predict and expensive to evaluate.

We choose Monte-Carlo integration as the base of our rendering algorithm because of the flexibility it provides in terms of scene and lighting complexity. Our goal is to design a renderer that has the advantages of Monte-Carlo, but that converges to the right solution rapidly through the use of intelligently selected probability functions.

3.2.1 Choosing probability functions

The ideal probability distribution function would be zero on non-visible light source points $\vec{y}_i$ and otherwise exactly proportional to the other terms in Equation 3.1. In this case the terms inside the sum become simply a constant. If the probability is zero, then we do not need to evaluate Equation 3.1, thus saving time. Unfortunately, computing the ideal probability function is only achievable and cost effective in the simplest cases. Even when excluding visibility, exact bounds can be difficult to compute *a priori* if the BRDF, geometry, or light's directional distribution is complex. In practice, various approximations to the ideal probability functions are used.

The simplest estimator samples all sources uniformly regardless of their actual contribution. A more advanced estimator would assign probabilities in proportion to at least some of the terms in Equation 3.1. Sampling according to the unoccluded irradiance of each light source is one frequently used option.

The irradiance due to a light source at a point $\vec{x}$ is defined as the emitted radiance times the projected solid angle of the visible portion of a light source when viewed from $\vec{x}$. The mathematical difference between irradiance and exitant

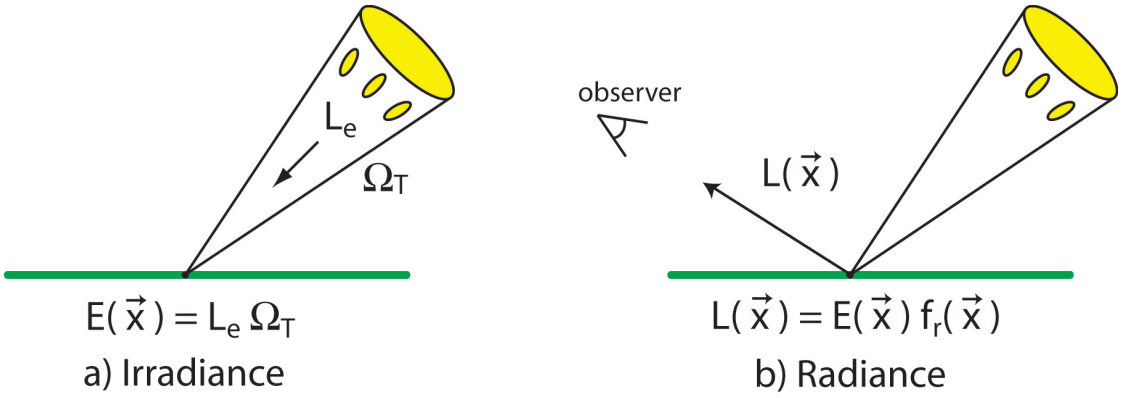

Figure 3.2: *L_e is the emitted radiance of the light source and Ω_T is its projected solid angle when viewed from $\vec{x}$.* ***a)*** *The irradiance is the radiant energy leaving the light source and arriving at point $\vec{x}$. Assuming the emitted radiance is constant over the light source, the irradiance is simply the emitted radiance times the projected solid angle. More generally, it is an integral expression over the domain of Ω_T.* ***b)*** *The exitant radiance is the luminance of point $\vec{x}$ when viewed by an observer from a specific direction. Assuming the BRDF, f_r is constant over Ω_T, exitant radiance is the irradiance arriving at point $\vec{x}$ times the BRDF. Otherwise, exitant radiance is also an integral expression over Ω_T as in Equation 2.1.*

radiance (Equation 2.1) is that irradiance does not include the BRDF $f_r()$.

Unoccluded irradiance also factors out the visibility term $V()$. In practice this difference is very minor in most situations and thus setting $p()$ proportional to unoccluded irradiance serves as a good approximation to the ideal PDF, unless the brightest lights contributing at point $\vec{x}$ are occluded. In this case, it is actually a very poor estimate as is shown in Figure 3.3.

The farther the actual probability is from the ideal probability, the more variance there will be in the estimator, and the longer it will take for the results to converge. On the contrary, if the PDF closely resembles the function, it will have

a low variance associated with it, which is also demonstrated in Figure 3.3. In this case, the average value of all the samples (i.e. sample mean) will quickly converge to the expected value. The algorithm we present in this thesis is designed to iteratively find an increasingly better PDF for each point we render. The next section provides an overview of our algorithm and how it proceeds to find such PDFs in a fast and efficient manner.

3.3 Rendering with Multiple Passes

3.3.1 Design Goals

Our goal is to start with a simple approximation to the ideal probability function and then adaptively optimize the function while rendering. We achieve this through performing multiple rendering passes. We use the result of evaluating Equation 3.1 from previous passes as feedback to generate the improved PDFs for the subsequent pass. This process will adapt the probability function based on the local lighting configuration without requiring precomputation or detailed knowledge about the scene. Conditions such as occluded lights are automatically detected statistically and progressively exploited as their reliability increases. During early phases, feedback data is aggregated over larger image regions to generate statistically meaningful information. As more data becomes available, the adaptation shifts toward smaller image regions. Our stopping criteria is based on the variance of the samples computed. Once the sample variance for a pixel falls below a threshold, we can stop rendering that pixel.

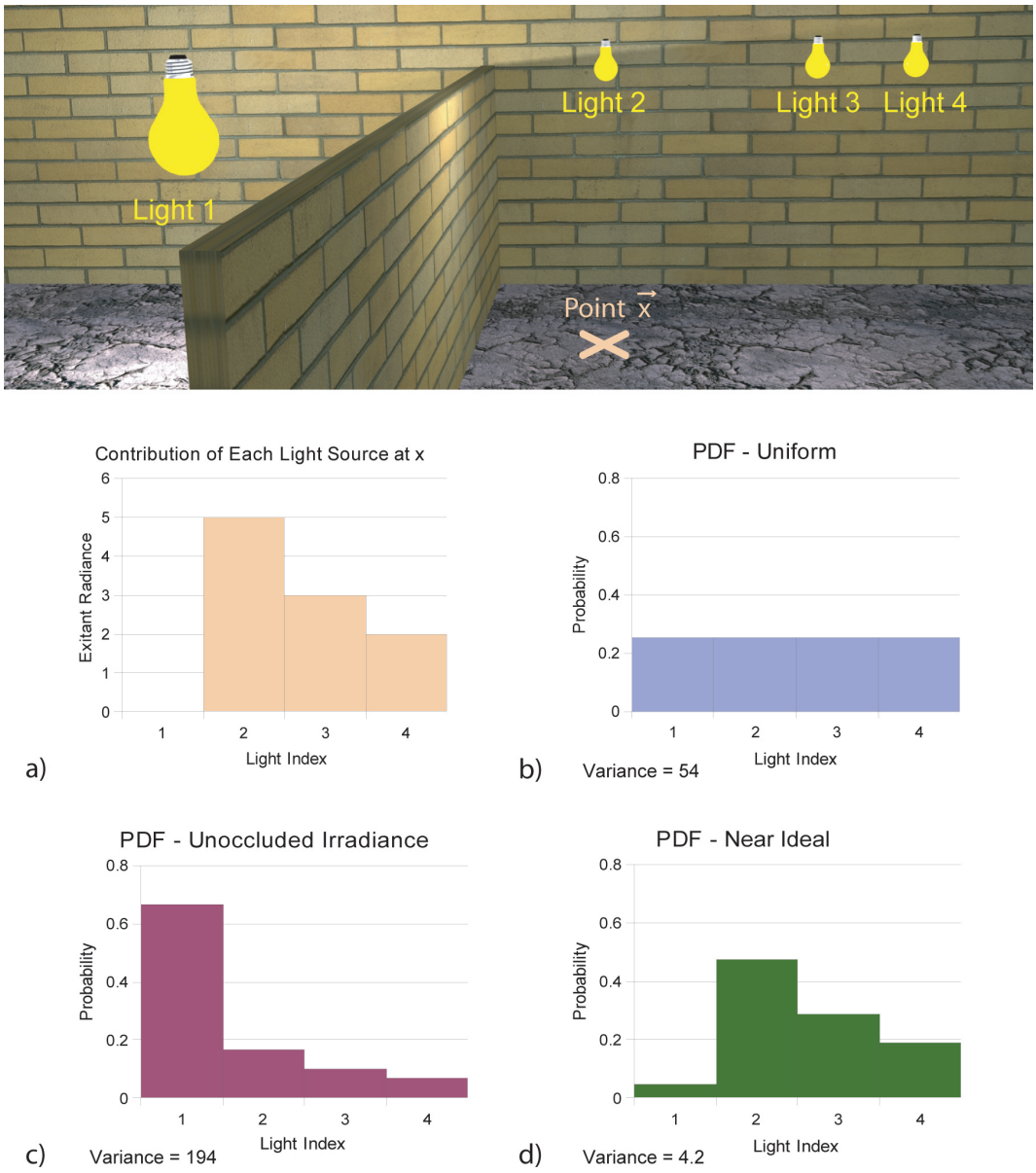

Figure 3.3: *A scene demonstrating the efficiency of three different PDFs. The brightest light is blocked at the point where we construct the PDF.* ***a)*** *The contribution to the exitant radiance is shown for each of the four light sources.* ***b)*** *A uniform PDF yields a high variance.* ***c)*** *The PDF based on the unoccluded irradiance does a poor job representing the actual contribution from the light sources.* ***d)*** *The near ideal PDF yields a low variance.*

3.3.2 Algorithm Overview

Our algorithm starts by dividing the image into 8×8 pixel blocks for processing where each block is computed independently through multiple rendering passes. The block structure allows us to aggregate sampling information across several pixels while keeping our data structures small and permitting easy parallel processing. For every point we render, we begin with a very simple PDF. We modify each PDF between rendering passes as we collect light visibility and contribution information through statistical sampling. We continue to perform rendering passes on the image block until each pixel in the block has reached our target quality setting. Figure 3.4 provides a simplified overview of our algorithm as a flow diagram. We numbered the steps in the flow diagram and refer to them when explaining parts of the algorithm.

Each block is first converted to a set of points in world space where we need to compute the direct illumination [**Step 1**]. In the simplest case, this just means shooting one viewing ray through each pixel to see what surface it hits. If an anti-aliased image is desired then multiple points are generated per pixel as discussed in Section 3.6. A block contains multiple pixels and each pixel contains one or more points. We will use this hierarchy of scales when aggregating statistical lighting information. A block is then computed using a variable number of passes as follows.

For each point in every pass we construct a probability function over the set of lights [**Step 2a**] and sample it some predetermined number of times [**Step 2b**]. This produces a set of samples on the lights which can be evaluated according to Equation 3.1. This includes shooting a shadow ray to the light sample to check visibility [**Step 2c**]. We also store the result of the sample evaluations [**Step 2d**] so that we may use them during our feedback stage [**Step FB**] at the end of

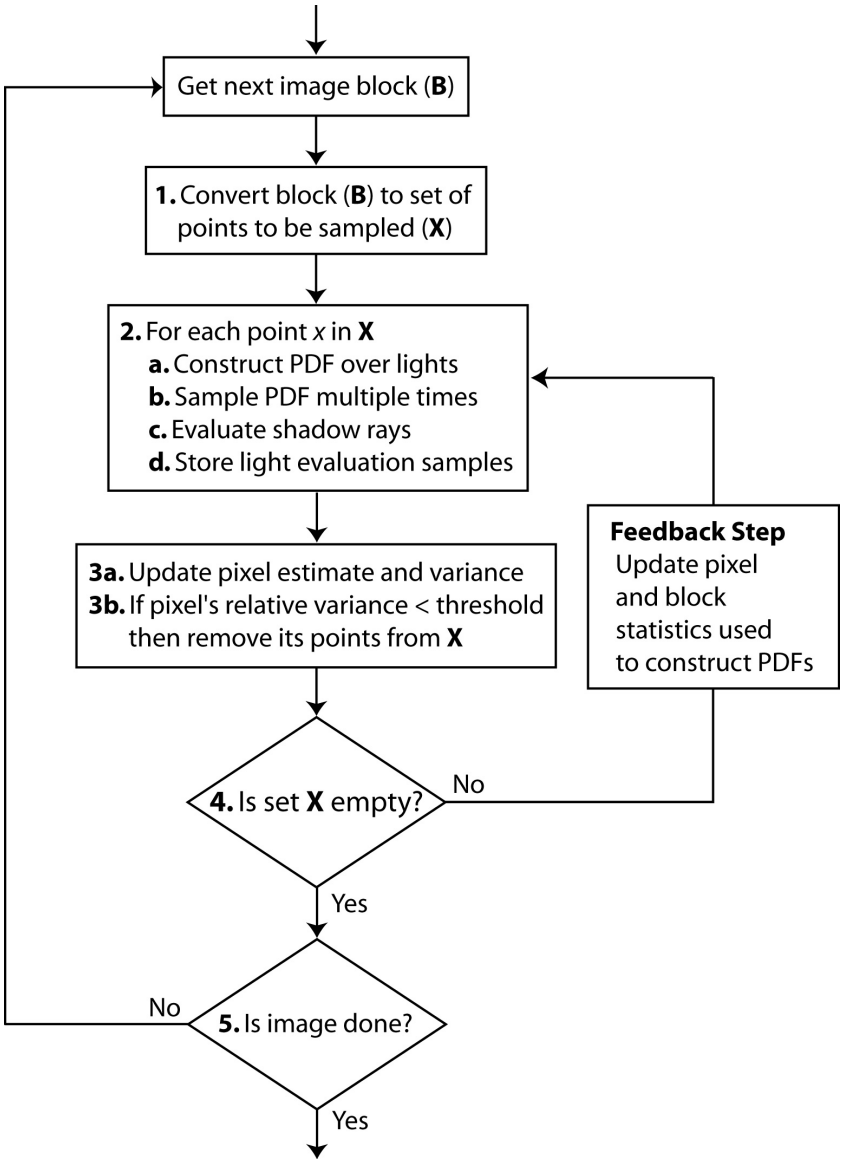

Figure 3.4: *Overview of multipass algorithm. We refer to the numbered steps in this diagram throughout the thesis.*

the current pass. The first pass uses simple probability functions that do not use any feedback information. Subsequent passes use probability functions that blend this simple probability function with probability functions constructed based on the results of prior samples averaged over the block and pixel. Regardless of the sampling data, we never assign a zero probability to any light source. Assigning a zero probability for a light that actually does contribute would introduce bias into our PDF. To ensure that our PDF remains unbiased all lights maintain at least a small base probability.

Next we compute an estimate for the shading value and variance of each pixel by combining the results from all of the sample points associated with the pixel [**Step 3a**]. These results are combined with the results of any prior passes as described in Section 3.5.3. If the combined variance for the pixel is less than our target variance threshold, then we assume that the sample mean for the pixel has converged to be indistinguishable from the expected value [**Step 3b**]. In this case, we stop further processing of this pixel, otherwise we will compute more samples for the points associated with this pixel in the next pass. At the end of a pass we check to see if there are any remaining points/pixels in the block that have not satisfied the threshold criteria [**Step 4**]. If any pixels need further processing, we use the sampling results from this pass to update the pixel and block statistics that we use to compute our probability functions. This allows us to improve our sampling probabilities in subsequent passes. Once all pixels have converged, we clear all the pixel and block data structures and start processing the next image block until the image is finished [**Step 5**].

3.3.3 PDF Construction and Refinement

We render image blocks with multiple passes so that we can adapt our PDFs in between passes. The sample data, which is stored at the pixel and image block level acts as feedback [**Step FB**] for refining our PDF. To construct the PDF over the lights for a point during a particular pass j, we combine three different component PDFs: a uniform PDF $p^U(\ell)$ and two feedback PDFs: $p^P(\ell)$, which is based on pixel sample data and $p^B(\ell)$ which is based on block sample data. These functions are defined in detail in Section 3.5.4 and by Equation 3.8. These are combined together using the weights shown in Figure 3.5 to get an overall PDF:

$$p_j(\ell) = c_j^U p^U(\ell) + c_j^B p_j^B(\ell) + c_j^P p_j^P(\ell) \tag{3.2}$$

The exact values of these weights is less important than maintaining a few important properties. The weights must sum to one. The initial pass can only use the uniform PDF because no feedback is yet available. Afterward, early passes should weight block PDF, p^B most heavily because it is averaged over the most data and converges faster. As more data becomes available, the more localized pixel PDF, p^P becomes more reliable and should be given larger weights, since it is able to locally adapt more precisely.

We also considered adding a point-based PDF that contained sample data only for its own intersection point. After some testing, we determined that this additional component PDF did not improve image quality nor improve rendering speed. In many cases the inclusion of a point PDF increased noise in the rendering due to the low availability of sample data at such a small scale. The overhead of working with an additional component PDF also increased rendering times.

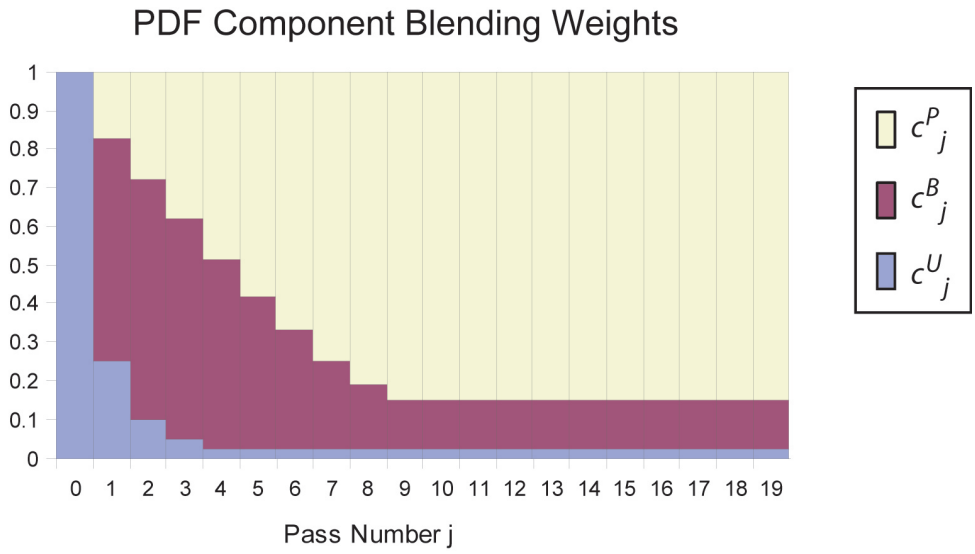

Figure 3.5: *The weights used to combine PDFs. c_j^P, c_j^B, c_j^U refer to the weight assigned to the pixel, block, and uniform PDF for pass j, respectively. Initially, we use just the uniform PDF. In later passes, we weight the block PDF and then the pixel PDF more heavily. Toward the end, we use the pixel PDF almost exclusively.*

3.4 Clustering

One of the advantages of rendering images on a block basis is that it is an excellent way to aggregate sample data on light sources. If a scene contains many lights, it can still be very expensive to generate sufficient sample data for all the light sources. If we have N_L light sources in a scene, we need $O(N_L)$ samples in order to be able to build an accurate PDF. Although trying to generate a PDF from very sparse sampling data is possible, it is problematic because our algorithm would interpret the lack of data from unsampled lights as evidence they were occluded.

Ideally we want to subsample the lights as well as assign appropriate probabilities to unsampled light sources. We do this by aggregating sample data on light sources in the form of light clusters. Again, we wish to take advantage of coher-

ence. In an image block, neighboring pixels may have similar luminance because the intersected surfaces most likely have the same BRDF, they see the same light sources, or they are the same distance away from light sources. Clusters are a form of spatial coherence. Lights near each other can be clustered due to their similar visibility, directionality, or distance from the point they illuminate. Figure 3.6 shows an example of how we might want to cluster light sources in a simple scene.

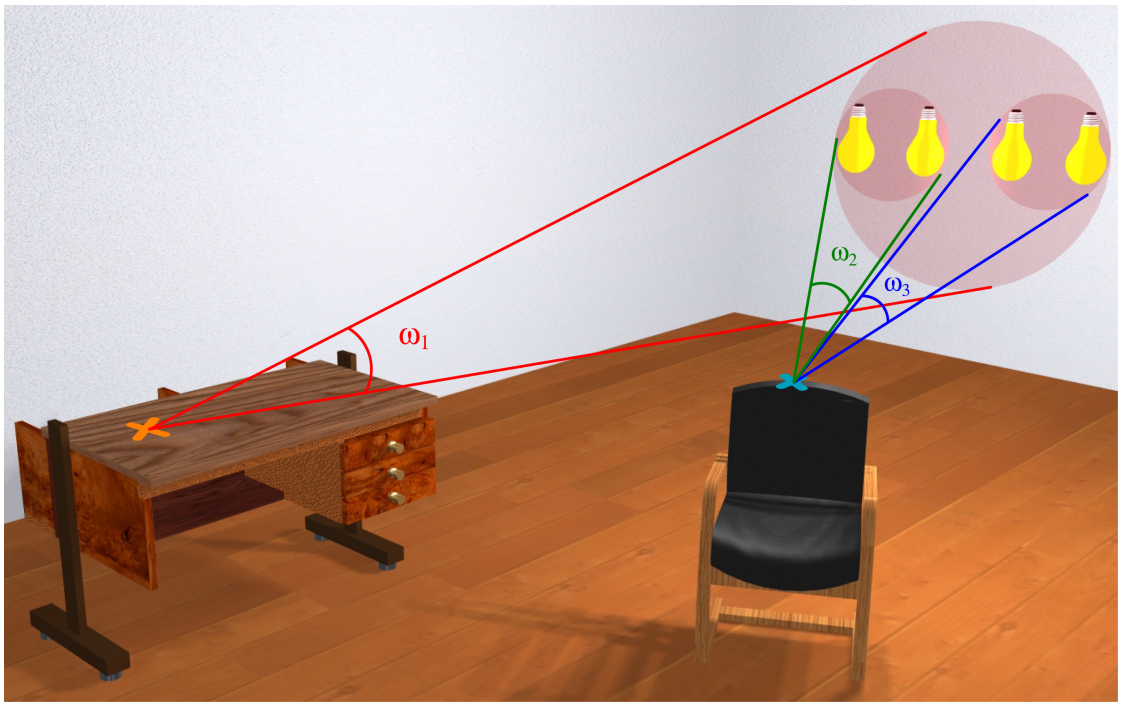

Figure 3.6: *We dynamically compute an appropriate clustering based on position. For the point on the table, all four lights are likely to make a good cluster because their bounding sphere subtends a small solid angle. For the point on the chair we may want to subdivide our cluster into two smaller clusters.*

If we sparsely sample only a few lights in the cluster, we can estimate the contribution of the cluster as a whole. We can then make a prediction about the contribution of individual light sources based on the contribution of their parent

cluster. In his thesis, Sebastian Fernandez [Fer04] also uses sparsely-sampled hierarchical light clusters to estimate the contribution of many lights. For each cluster he defines a representative light source to serve as the estimate for the contribution of the entire cluster. In our system, we sample all the lights within a cluster according to their emitted radiance, L_e, as referenced in Equation 2.1. The next section explains why this is a good estimate.

3.4.1 Ideal Cluster Properties

Recall from Equation 2.1 that the exitant radiance due to a light source is a result of a product of the BRDF $f_r()$, emission $L_e()$, visibility $V()$, and geometric term. Consider the point on the table in Figure 3.6 illuminated by the four light sources in the scene. When evaluating each light in the four-light cluster, the BRDF and geometric term are likely to have minimal variation. This is related to the fact that the cluster's bounding sphere subtends a small solid angle.

The emission can vary greatly (e.g. one light can be much brighter than the others), but because we know this in advance we can include this into our PDF. If we sample light sources within a cluster according to their intensity, it will not be a source of variation or error.

As we mentioned previously, visibility is very difficult to predict and any variation in visibility across a cluster can invalidate our assumption that the contribution from all sample points $\vec{y}_i$ within a cluster is roughly constant. This can reduce the effectiveness of our PDF; however, [ARBJ03] shows that visibility is likely to be very coherent across a small solid angle. Because of this simple fact, we can usually sample just a few light sources within a cluster and get a very good estimate of visibility for the entire cluster. Furthermore, in the following chapter we

show extensions that show how to optimally sample clusters with varying degrees of partial visibility.

3.4.2 Hierarchical Light Clusters

Since the suitability of a cluster depends on its subtended solid angle from the point being rendered, our clustering scheme needs to be locally adaptive. No single partitioning of the lights into clusters is likely to work well over the entire image, but dynamically finding a new cluster partitioning for each surface point could easily prove prohibitively expensive. To solve this problem we use a global cluster hierarchy[2] to rapidly compute locally adaptive cluster partitions.

A cluster hierarchy is a tree where the leaves are the individual lights and the interior nodes are light clusters that contain exactly the lights below them in the tree. We define a *cut* through the tree as a set of nodes such that every path from the root of the tree to a leaf will contain exactly one node from the cut. Each tree cut thus corresponds to a valid partitioning of the lights into clusters.

We use a greedy algorithm to build our cluster hierarchy. We start by converting each light source into a cluster that contains just the light itself. We then use a bottom up binary tree building approach where we progressively pair clusters together starting with the pair that has the smallest bounding sphere. For efficiency, we can also define a maximum cluster size based on the dimension of the scene. In addition we want to prevent dissimilar lights from being clustered together. The emitted radiance of a cluster that contains both omni-directional lights and oriented lights will vary greatly with different viewing angles. For this

[2]This is part of work done in collaboration with Dr. Bruce Walter, Prof Kavita Bala, Dr. Sebastian Fernandez, and Prof. Donald P. Greenberg

reason, omni-directional lights have their own hierarchy and oriented lights are only clustered with other oriented lights if they have a similar orientation. Environment maps are also handled explicitly in a separate data structure. In the end we have a "forest" of binary light cluster trees: one for environment maps, one tree for omni-directional lights, and six trees for oriented lights–one corresponding for each of the six cardinal directions in world space (i.e. X+, X-, Y+, Y-, Z+, Z-). For static environments, we compute the cluster hierarchy only once per scene. Then when shading each point, we can quickly and dynamically compute a cut, or appropriate clustering of lights that is specific to that point by traversing the hierarchies.

3.5 Implementation

In this section we discuss the implementation details of all the steps outlined in the flow diagram (Figure 3.4).

3.5.1 Preprocessing a block

Before we begin rendering our pixels, we need to compute the appropriate set of points and to compute a cut through the cluster hierarchies for each point [**Step 1**]. This is done for every pixel in the block. We only compute this information once and then reuse it for every rendering pass on that block.

This initial stage begins with computing the set of points that we want to sample during the rendering passes. For each pixel p in the current block B, we calculate one or more intersection points of the eye rays with the scene geometry. We cache this set of points called X_p for each pixel since we will reuse the same

set of intersection points for each pass. In Section 3.6 we discuss a novel approach toward pixel anti-aliasing that is particularly well suited for this algorithm.

For each element of X_p we also compute our light cluster cut. We start at the root node of each cluster hierarchy tree and progressively subdivide clusters that are larger than a pre-specified solid angle. We add the largest clusters that are below the solid angle threshold to the cluster cut. As explained in Section 3.4.1, using a solid angle metric takes advantage of coherence in visibility across a small region in space. The clustering also has some benefits when it comes to sampling a point, which we discuss in Section 3.5.2. Refer to Figure 3.7 for a graphical representation of this step.

For each cluster we encounter while traversing down the tree, we also perform some simple geometric tests to determine if it can be immediately disregarded because it will not contribute any irradiance at our intersection point. These tests include pruning out clusters on the opposite side of the surface normal or clusters with oriented lights that all face away from our intersection point. Because the size of the cut typically varies logarithmically with the number of lights in the scene, the expense of these tests is minimal even in scenes with hundreds or thousands of lights.

For environment maps, we do not have a hierarchical structure; each region in the map will always subtend the same solid angle regardless of the point in the scene. As a result, we just add each environment region to the cut unless the entire region is on the opposite side of point's surface normal.

Performing these geometric tests on the clusters is effective and worthwhile because it allows us to quickly prune out any clusters that will provably not contribute. As we mentioned in Section 3.3.2, it is important that our PDF be un-

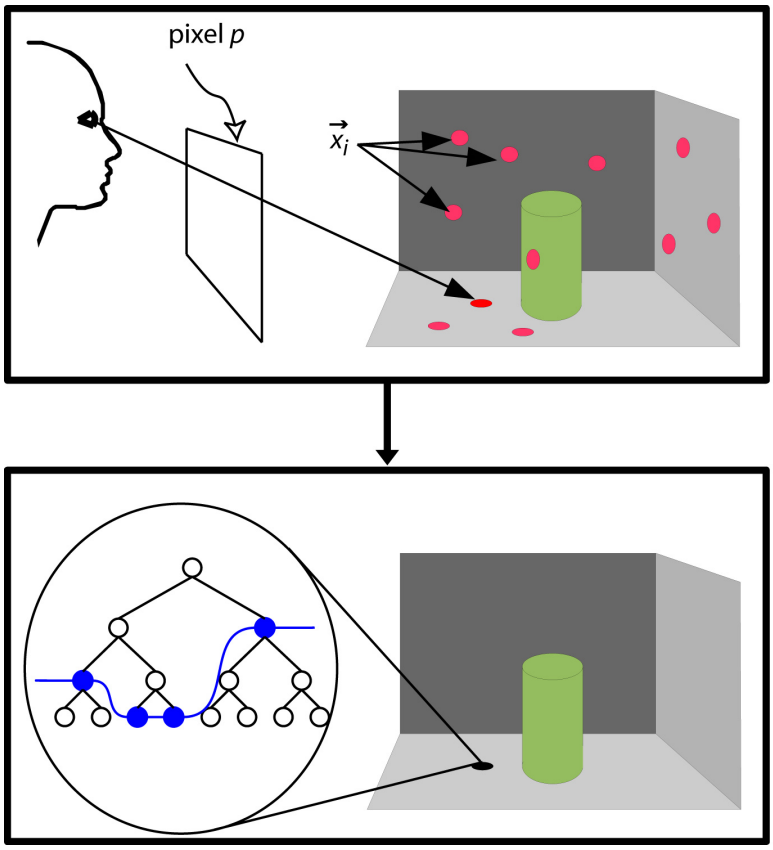

Figure 3.7: *A graphical representation of **[Step 1]** of the multipass algorithm showing how we **a)** compute the surface intersection points for each pixel and **b)** the cuts through the cluster hierarchies for each point.*

biased. Setting a zero probability to a non-contributing light does not introduce any bias, but it does help our PDF converge to the ideal more rapidly. It will also prevent any wasted effort going toward setting up and tracing shadow rays to lights that we know cannot contribute.

3.5.2 Sampling at a point

For each point $x \in X_p$ we want to define a PDF **[Step 2a]** over the light sources specific to that point and then proceed to sample that PDF **[Step 2b]** and trace

shadow rays [**Step 2c**]. The initial pass is different from all subsequent passes because we start without any sample data. The primary purpose of the first pass is therefore not to estimate the value of the pixel, but rather to "seed" our sample data [**Step 2d**]. We do this by constructing an initial PDF that equally weights all clusters in the cut. Sampling uniformly over light clusters is superior to sampling uniformly over all light sources because it allows us to radially stratify our samples in a hemispherical space centered at $\vec{x}$. This sampling is roughly equivalent to sampling inversely-proportional to the distance-squared, except that it allows us to stratify our samples to ensure that each cluster receives a sample. In effect, this places a greater density of samples close to the intersection point and allows us to capture small visibility features. Given an equal budget of samples between the two sampling techniques, uniform sampling over the light sources would oversample distant groups of light sources and undersample nearby light sources. Uniformly sampling the clusters is also consistent with how we construct the PDFs for subsequent passes. Because our algorithm constructs PDFs by assigning probabilities to clusters, it is important that each cluster have a good distribution of samples. When constructing our first sample-driven PDF, it is better for all clusters to have some samples than for some clusters to have many samples and some clusters to have few or no samples.

For all other passes we compute PDFs that depend on sample data collected in previous passes. The feedback step [**Step FB**] accumulates and averages sample data collected during the last pass at [**Step 2d**] and combines it with sample data collected during all the previous passes. The PDF for a point is a blending of three different functions, two of which are based on sample data. Section 3.5.4 describes in detail how we store and use our sample data to generate these functions.

3.5.3 Computing the pixel estimate

In each pass j, and for each pixel p, we compute a pixel estimate [**Step 3a**], $I_{p,j}$, of the exitant radiance using Equation 3.1 and an associated error estimate [**Step 3b**] that is used as a termination criterion. We want to use all previous pixel estimates to increase the accuracy of our combined estimate. We now describe how to compute these estimates across multiple passes.

The error estimate for pixel p in pass j is computed as the sample variance, $s^2_{p,j}$, of all the light samples for that particular pixel. We must be careful to use an unbiased estimator of sample variance. The exact estimator used is described in the appendix. The sample variance is computed numerically using the stored sample data. We can then compute an overall pixel estimate and sample variance that is weighted over all passes in a way that minimizes total variance [DBB03]. Given the current pass's sample variance $s^2_{p,j}$ as well as the sample variance computed from the previous passes $s^2_{p,0} \cdots s^2_{p,j-1}$ the overall sample variance for the pixel, s^2_p, is then computed using equation 3.3:

$$s_p^2 = \left(\sum_{i=0}^{j} \frac{1}{s_{p,i}^2} \right)^{-1} \tag{3.3}$$

Similarly, let I_p be the estimated value of pixel p that combines pixel intensities from passes $0..j$ as given by equation 3.4 below:

$$I_p = \left(\sum_{i=0}^{j} \frac{I_{p,i}}{s_{p,i}^2} \right) * s_p^2 \tag{3.4}$$

Termination criterion: For termination, we test $s_p^2 / I_p^2 > t$, where t is the threshold we use as a termination criterion [**Step 4**]. If this inequality is satisfied, then we mark that pixel as completed and compute its final value I_p. Note that this inequality normalizes for pixel intensity. In subsequent passes, we only continue

working on pixels in the block that have not yet converged. If after any pass, we find that all the pixels in a block have a sample variance s_p^2 that is below our threshold variance, we can stop rendering that block and move on to the next one [**Step 5**].

3.5.4 Storing statistics and constructing PDFs

We keep statistics about the results of prior lighting samples in order to evolve and improve our sampling PDFs. This allows our PDFs to automatically adapt to handle conditions such as occluded clusters. Since we sample clusters with a high contribution with greater probability, our algorithm is also able to efficiently capture glossy highlights.

To accomplish this task we keep track of the average contribution and visibility of each light averaged over the block and each pixel in the block. Figure 3.8 explains the need for multiple levels of granularity in the storage of sample data and the resulting need for a blending of multiple PDFs. Rather than store the result of every single sample evaluation, we only keep track of running sums of exitant radiances for each of the light sources as well as a counter on the total number of samples and the number of visible (unoccluded) samples. We also propagate the sample information from the individual light sources up the cluster hierarchies. This way, if two points have different cluster cuts they can still share sample information to some extent.

As mentioned in Section 3.5.2, we assign probabilities to clusters instead of individual light sources. This reduces the overhead involved in building a PDF, which can be a significant performance gain in scenes with many lights since the size of the cut is significantly less than the number of lights in the scene (Table 4.5).

We sample light sources within a cluster proportionally to the emitted radiance of each light in the cluster. Standard sampling techniques can be used to pick the point within a light once it is chosen, such as uniform area sampling or the techniques of [SWZ96]. We use uniform area sampling for all planar light sources and uniformly directional space sampling for spherical lights.

In order to construct PDFs from sample data, we need an effective method of organizing and accessing the sample data. Because an optimal PDF is proportional to the actual function it is trying to estimate, our sample data must keep track of the average contribution of light clusters. In addition we want to measure the average behavior of clusters at different image space granularities–namely at the block and pixel level.

We can express the process of storing statistics as follows. For any block, let R be a set consisting of pairs of points $\{\vec{x}_i, \vec{y}_i\}$ that define light evaluations (i.e. $\vec{y}_i$ is a point on the light source and $\vec{x}_i$ is a point being illuminated) which were evaluated using Equation 3.1. Let R_ℓ be the sample mean for all evaluations that sampled light ℓ. Remember that in our hierarchical light clustering, ℓ can be either a light or a cluster of lights. Let R_ℓ^A be the set of all light evaluations for light ℓ from points $\vec{x}_i \in A$. Figure 3.9 uses color coding on a simple scene to visualize the set A for a block or pixel. Finally, let $R_{\ell,j}^A$ be the set of all light evaluations using Equation 3.1 from points in the set A to points on light ℓ up through pass j.

Let $L(\vec{x}_i, \vec{y}_i)$ be the result of the light evaluation which is the same as evaluating Equation 3.1 using just one sample.

$$L(\vec{x}, \vec{y}_i) = \frac{V(\vec{x}, \vec{y}_i)\, f_r(\vec{x}, \vec{y}_i)\, L_e(\vec{x}, \vec{y}_i)\, G(\vec{x}, \vec{y}_i)}{p(\vec{y}_i)} \tag{3.5}$$

The estimated contribution of a light ℓ over a set A for samples through pass j

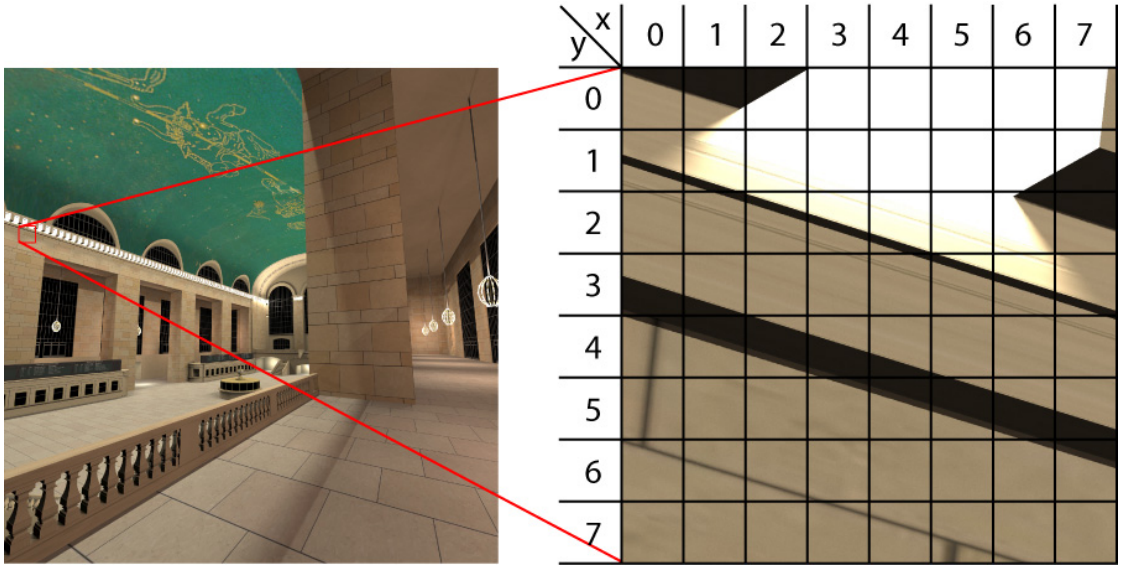

Figure 3.8: *The figure on the left is a rendering of Grand Central Terminal. On the right we show a closeup of an* 8×8 *pixel block. The white area at the top is a light that is visible through the* 8×8 *pixel block. This light has a very high contribution for the pixels at the top of the block and little to no contribution for the pixels at the bottom. Averaged over the entire block the exitant radiance of the light is about an order of magnitude greater than all the other contributing lights combined. Sampling according to just the block PDF would produce very good results for the top pixels and very noisy results for the bottom pixels. We need a finer measure of granularity in our PDF construction to be able to capture the sharp differences in lighting that can occur across the pixels in a block. For this reason our algorithm creates a final PDF at a point from a linear combination of a uniform PDF as well as other PDFs generated from block- and pixel-based data.*

can be written as:

$$C^A_{\ell,j} = \frac{1}{\left|R^A_{\ell,j}\right|} \sum_{\{\vec{x}_i,\vec{y}_i\}\in R^A_{\ell,j}} L(\vec{x}_i,\vec{y}_i) \tag{3.6}$$

Naively, we may want to assign probabilities to clusters proportionally to the contribution C, but it turns out that this is ideal only when considering point lights. Area lights and clusters are not point entities and may have a continuous distribution and partial visibility, but algorithm assigns probabilities in a very discretized way.

Our intuition is that we need to consider partial visibility as well as radiance contribution when assigning probabilities. To find the optimum probability, we minimize for variance which we define in terms of contribution and occlusion percentage. Let u be the visible fraction of a light when viewed from the point we are trying to render. The variance minimizing probability is thus proportional to $\frac{C}{\sqrt{u}}$. We provide a proof of this result in the appendix.

Another benefit of the $\sqrt{u}$ term is that it changes the relative weighting of light sources in a helpful way. Lights with partial visibility are given more samples than they would get without including this term. Given the positive feedback nature of this algorithm it is very possible that the PDF may undersample some of the highly occluded but still important lights. The $\sqrt{u}$ term reduces this tendency.

To be consistent with our algorithm let $u^A_{\ell,j}$ be a fraction where the numerator is the number visible light evaluations and the denominator is the total number light evaluations sent from all surface points in the set A to all sample points on light ℓ up through pass j. Thus the probabilities should be proportional to:

$$F^A_{\ell,j} = \frac{C^A_{\ell,j}}{\sqrt{u^A_{\ell,j}}} \tag{3.7}$$

A PDF for pass j can be constructed from the samples from all prior passes as:

$$p^A_j(\ell) = \frac{F^A_{\ell,j-1}}{\sum_{l \in S} F^A_{l,j-1}} \tag{3.8}$$

where S is the set of all clusters in the current cut.

We use this equation to compute the pixel and block PDFs for a pass by evaluating this equation with the set A replaced by the points associated with a pixel or block respectively. Remember that we do not actually need to keep all the individual light sample results; instead, we can just keep track of the running sums in Equation 3.6 for the block and each pixel in it.

We are able to further improve our performance by breaking the pixel and block statistics into different sets based on the surface normal of the point being shaded. We divide the normals into six sets using a cube decomposition of direction space aligned with the world space axes (i.e. X+,X-,Y+,Y-,Z+,Z-). We do this to prevent dissimilar samples from being averaged together. We then split the pixel and block statistics correspondingly and only use data from points with the same normal classification when computing the pixel and block PDFs for a point. Figure 3.9 shows how points on different surfaces can have their sampling statistics combined if their surface normals are similar.

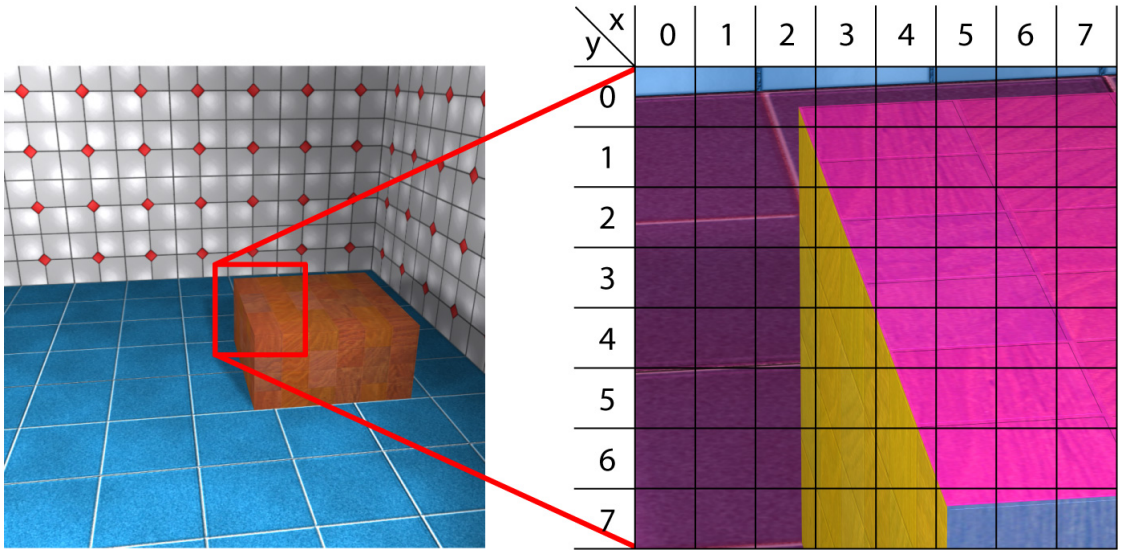

Figure 3.9: *The image on the left is a rendering of a simple scene while the color coded image on the right represents what we might see through an* 8×8 *pixel block. Since we only combine sample data for surfaces with similar normals, the color coding (Cyan, Magenta, and Yellow) shows how we would separate our sampling statistics for this particular image block. When generating a block PDF for a point on a magenta region, we use all sample data from all magenta regions. When generating a pixel PDF for a point on a magenta region, we use all sample data from all magenta regions within just that pixel. We combine sample data even if they lie on separate planes such as the top of the box and the floor the box is resting on.*

3.6 Adaptive Anti-Aliasing

To anti-alias our image we use the standard technique of supersampling the pixels by generating multiple eye ray intersections per pixel. The degree of aliasing varies inversely with the number of samples (intersections) used to estimate the pixel, therefore we want to have as many intersections as possible.

3.6.1 Requirements

Many adaptive anti-aliasing algorithms for Monte-Carlo ray tracing use a technique known as adaptive progressive refinement [PS89]. Typically, adaptive progressive refinement traces and shades eye rays to the pixel until all pixel samples reach some variance threshold. Since the multipass rendering approach in this thesis assumes that we know all intersection points within a pixel before we start rendering it, we cannot just simply create more intersections per pixel as we need them. Our other requirement is keeping the number of intersections per pixel low because a large number would add significant overhead to PDF construction for each pass. We would like to find a small set of representative points per pixel that still accurately represents the discontinuities present in the pixel. Our first priority should be to handle the geometric boundaries rather than shadow boundaries since the former tend to be more visually apparent. Also, since area lights create soft shadows, shadow anti-aliasing is less of an issue.

3.6.2 Approach

The simplest way to meet our first requirement is to trace a predetermined number of eye rays and record their intersections. A more intelligent method would use an

adaptive stopping criterion that is based on some estimate of either the complexity or variation of the intersection points within a pixel. Both of these approaches however would prevent us from meeting our second requirement of keeping the number of intersection points low.

Our solution is to group similar intersection points together and create non-uniform sized sub-pixel regions. Each region will have a representative intersection point near the center and an estimate of its sub-pixel area. The best way to visualize these regions is to imagine them as being small planar surfaces in space of constant color. This allows us to get the anti-aliasing benefits of tracing multiple eye rays while only having to shade a small subset of those points. Since we expect geometric discontinuities to be the major source of aliasing, our metric groups intersection points based on similar geometric features.

Finally we need to place a limit on the size of these regions. We set our region size limit to one-quarter of the pixel radius. In pixels with no geometric discontinuities this will generate a minimum of four regions per pixel and should be sufficient for almost all shadow anti-aliasing needs. The number of regions per pixel increases with the geometric complexity within the pixel. We use the ray differential [Ige99] as a convenient means to determine how close two intersection points (which are in world space) are in image space.

3.6.3 Implementation

Now that we have an approach that meets our requirements, we will explain how we construct these regions, how we choose the representative point, and how we determine the sub-pixel area of these regions. The representative point will contain a record of material at the point and its surface normal. The selected point

determines the cut in the global light cluster hierarchy and the construction of our PDF.

To find these regions, we start by firing an initial set of twenty rays from the eye through each pixel. We want to bundle similar intersections with similar characteristics into regions. Each intersection from a traced ray can either form it's own region or it can join another existing region. If a new region is formed, the first intersection point becomes the representative point for that region. The criteria for a ray joining a region are that its intersection point must:

(a) lie in the same plane as an existing region and have the same material (BRDF)

(b) be within the region's radial extent.

After shooting this initial set of rays we need to decide if we should send more rays or if our initial set is sufficient to characterize the entire pixel. If all the rays hit the same surface then the initial set is probably sufficient. We can keep track of the number of unique surfaces by checking when the new intersection point fails the grouping test. If it fails at the first condition then we know we hit a unique surface. Our metric for determining how many eye rays to send is the number of unique surfaces in a pixel. We maintain a ratio of twenty pixel samples per unique surface up to a maximum of sixty samples for three unique surfaces.

After we have evaluated all our eye rays and created the regions for a pixel, we only need to determine what ratio of eye rays hit that region with respect to all the pixels for the region. One can think of this ratio as the area of the sub-pixel region with respect to the area of the pixel. To minimize the variance of the pixel, we want to minimize the $variance \times area$ of sub pixel regions. We know that

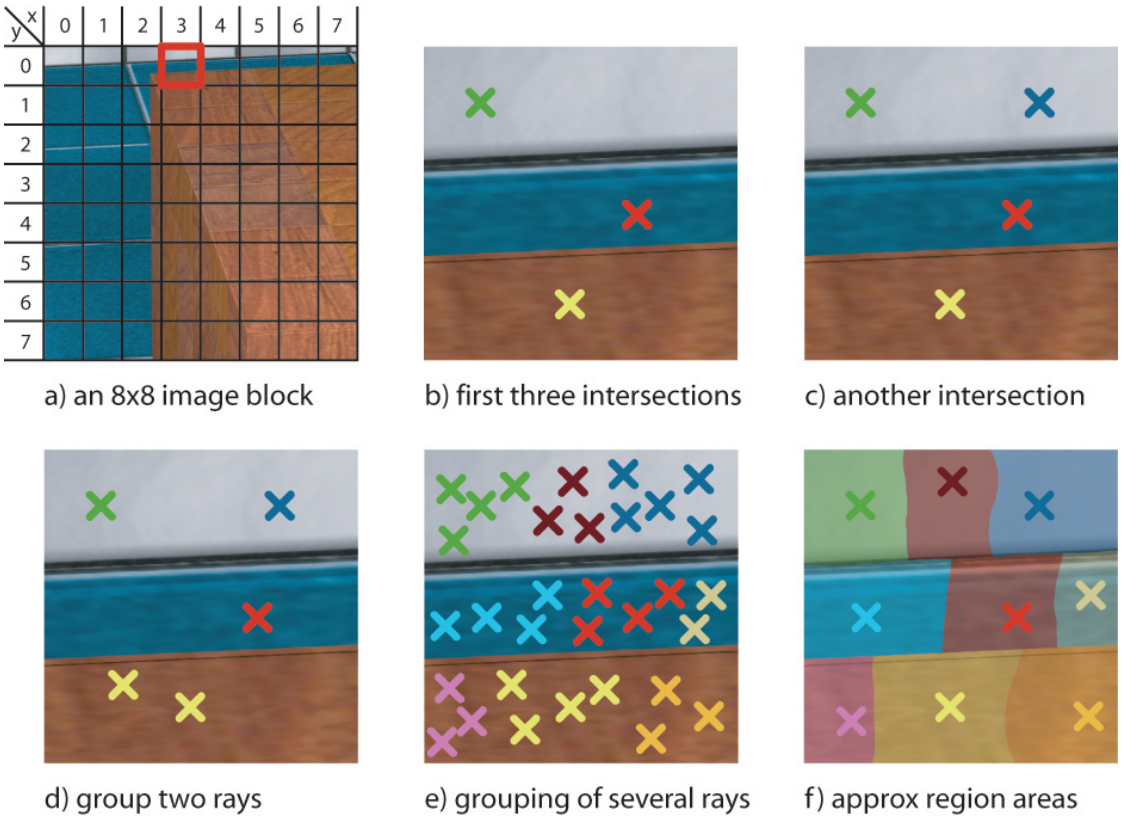

Figure 3.10: *In this figure we will show the process we use to subdivide a pixel into regions and the representative points that we use for pixel anti-aliasing.* ***a)*** *An* 8×8 *pixel block from figure 3.9.* ***b)*** *Closeup of pixel (3,0) after firing the first three eye ray intersections. All rays hit a unique surface.* ***c)*** *The fourth ray (blue) hits a non-unique surface but it is too far away to be grouped with the previous ray (green) that hit the same surface so it forms a new region.* ***d)*** *The fifth ray (yellow) hits a non-unique surface and is sufficiently close to one of the pre-existing regions.* ***e)*** *Pixel after firing a total of 32 rays and grouping them according to our criteria.* ***f)*** *A false-color visualization show what the regions for this pixel look like. The X's in each region symbolize the first eye-ray intersection that created the region. Any additional rays fired would be grouped in one of these existing regions. The sub-pixel area of these regions is estimated by the number of rays that hit the region.*

variance varies inversely with the number of samples and thus to minimize overall pixel variance during rendering, we split our budget of shadow rays for a pixel proportionally to the area of the region.

Chapter 4

Results

In this chapter we will provide results for our direct illumination algorithm (Iterative Adaptive Sampling) and compare it to a reference solution in terms of both speed and quality. Recall our primary observation. While a scene with complex lighting may have many lights visible in a single image, at any particular surface intersection point only a few lights may actually be visible and contributing to the total irradiance at the surface. Our algorithm achieves nearly an order of magnitude speed increase because we exploit this finding in a variety of ways.

Our algorithm is based upon the concept of sampling light sources in proportion to their actual contribution of a pixel's exitant radiance. We do this by iteratively modifying a probability density function (PDF) until it captures the local lighting configuration. We use sample data collected during rendering as feedback to drive the modification of the PDF. Our algorithm takes advantage of coherence in image space by aggregating sample data on both a per-pixel and per-block level as well as coherence in world space by aggregating sample data on light clusters. Through feedback and aggregation we are able to reduce the number of shadow rays we have to evaluate in order to accurately determine the exitant radiance for a pixel. Since

shadow rays are typically the most expensive computation of a ray tracer, our primary performance speed-up comes from this reduction. We are able to achieve further performance enhancement though the efficient usage of hierarchical light clusters and "cuts" through the tree hierarchy.

4.1 Test Setup

4.1.1 Algorithm Comparison

We compare our algorithm to a reference algorithm that samples according to the unoccluded irradiance of a light source. In scenes that contain direct lighting from an environment map, samples are split between the lights in the scene and the environment map proportionally to total unoccluded irradiance. Sampling within the environment map is performed using Structured Importance Sampling [ARBJ03]. For both Iterative Adaptive Sampling and the reference solution we stratify the environment map with 300 regions and use jittering and pre-integration.

We rendered our images at 1024×1024 resolution with anti-aliasing on a dual-processor 1.7 GHz Pentium 4 Xeon computer with 1024MB of memory.

4.1.2 Models

We tested our algorithm on three different models. One of which, the *Kitchen*, has two different lighting scenarios, thus providing us with a total of four testing environments. The *Kitchen* model is our simplest scene with 72 area lights and 338 thousand triangles. *Kitchen 2* is identical to the *Kitchen* except that it also contains lighting from an environment map. The windows are not visible in the viewpoint of the kitchen which we rendered but they are directly behind the camera. The third

model, *Ponderosa* is geometrically simple with only 131 thousand triangles, but it contains 138 point light sources as well as direct lighting from an environment map. Our final model, *Grand Central Terminal* is a model of the Grand Concourse Lobby of the famous train station in New York City. It is our most complex model with over 1.5 million triangles and over 800 light sources, of which 219 are spherical area light sources. We have summarized the basic statistics for the four models in Table 4.1. Renderings of the four environments are shown in Figure 4.1 through Figure 4.4.

Table 4.1: *Model Statistics*

Model	Triangles	Lights		Environment Map
		Point	Area	
Kitchen	338K	0	72	No
Kitchen 2	338K	0	72	Yes (300 Regions)
Ponderosa	131K	138	0	Yes (300 Regions)
Grand Central Terminal (GCT)	1527K	613	219	No

Figure 4.1: *Kitchen Model*

Figure 4.2: *Kitchen with Environment Lighting*

Figure 4.3: *Ponderosa*

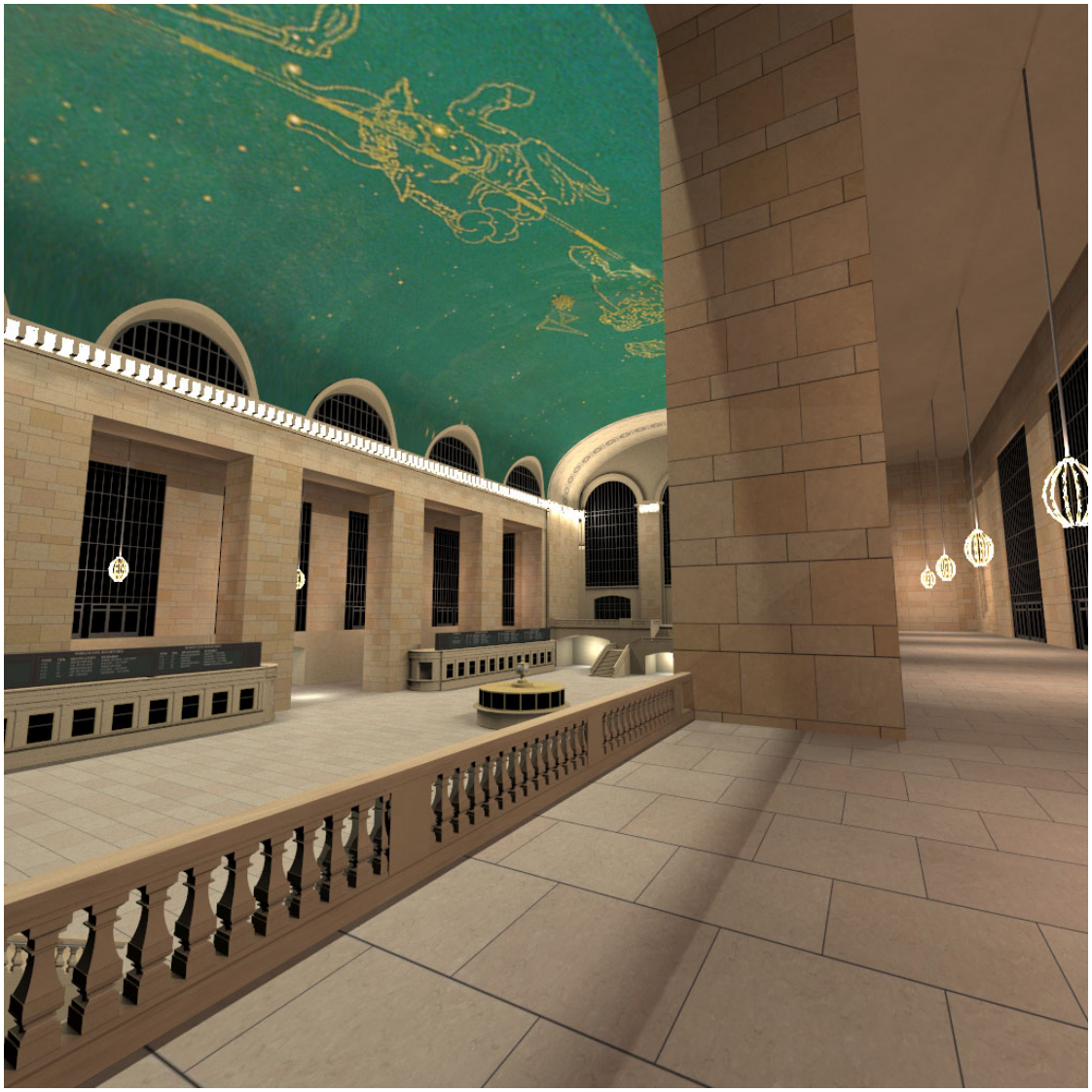

Figure 4.4: *Grand Central Terminal*

4.2 Reference Solution Implementation Details

We want to compare our algorithm to an industry standard ray tracer in order to have a fair and meaningful comparison. The reference algorithm we use for the purposes of comparison is also a Monte-Carlo ray tracer except that it uses a fixed PDF on the light sources that is proportional to their unoccluded irradiance. We also tested a reference solution with a PDF that weights the lights sources uniformly, but found that unoccluded irradiance outperforms it by nearly a factor of two for all of our scenes.

To anti-alias the image for our reference algorithm we intersect each pixel a minimum of twenty times. For each eye ray intersection we compute an appropriate PDF which we then sample multiple times. The reference solution also uses an adaptive stopping criterion that continues to intersect the pixel and sample a new PDF until the relative sample variance of all of the light samples is below a threshold. The identical stopping criteria allow us to perform an equal quality comparison of our algorithm with the reference solution. Since the relative variance stopping criterion for the reference solution also considers variance due to pixel aliasing, we also achieve a form of adaptive anti-aliasing for each pixel. We also perform equal time comparisons to show how well the reference solution would perform if given an equal time budget as our algorithm.

One of the drawbacks of the reference algorithm is trying to find the right distribution of shadow rays versus eye rays per pixel. Since the cost to compute a PDF based on unoccluded irradiance can be expensive for scenes with multiple lights, it is important to sample a PDF multiple times per eye ray intersection for best performance. For optimum performance we need to minimize the number of primary intersections (and hence PDF calculations) while still maintaining suffi-

cient pixel anti-aliasing properties. If we do not meet the right balance, we may oversample the lights for a pixel while undersampling the pixel's geometry and spending a disproportionate amount of time trying to define PDFs. Our multipass algorithm does not suffer from this problem since we compute the degree of pixel anti-aliasing before we begin rendering.

4.3 Resulting Images and Execution Timings

In this section we provide both qualitative and quantitative comparisons of our algorithm with respect to the reference ray tracer described in Section 4.2.

4.3.1 Quantitative Comparison

We list the times needed to render a reference image, an image of equal quality using our algorithm, and the speed-up of our algorithm over the reference solution in Table 4.2.

Table 4.2: *Same Quality Rendering Time Performance Results*

	Rendering Time		
Model	Reference (s)	Iterative Adaptive (s)	Speed-up
Kitchen	15,364	1,726	**8.9x**
Kitchen 2	68,392	8,368	**8.2x**
Ponderosa	53,328	5,604	**9.5x**
GCT	57,432	7,136	**8.0x**

By examining the average number of shadow rays per pixel in Table 4.3, we can see immediately that the reduction in shadow rays is the dominant factor in the computation acceleration. This reduction is primarily attributable to the adaptive PDF based on unoccluded light sources. Further performance speed-ups beyond

Table 4.3: *Same Quality Light Sample Count Performance Results*

	Average Shadow Rays Per Pixel		
Model	Reference (s)	Iterative Adaptive (s)	Reduction
Kitchen	1,870	278	**6.7x**
Kitchen 2	9,916	1,604	**6.2x**
Ponderosa	10,772	1,290	**8.3x**
GCT	6,048	1,012	**6.0x**

shadow ray reduction are achieved through the use of hierarchical clusters and tree-cuts, as explained in Section 3.4.

4.3.2 Qualitative Comparison

For the Grand Central model we show a side-by-side equal time and equal quality comparison in Figure 4.5. Due to the limited printing resolution, it is almost impossible to notice any differences in the renderings. For this reason we provide closeup shots of the images to bring attention to quality differences and similarities. Notice that there is no perceptible difference in the equal quality comparison even in the closeup of the soft shadow region. For the remaining scenes (Figures 4.6, 4.7, and 4.8) we only show equal time image comparisons.

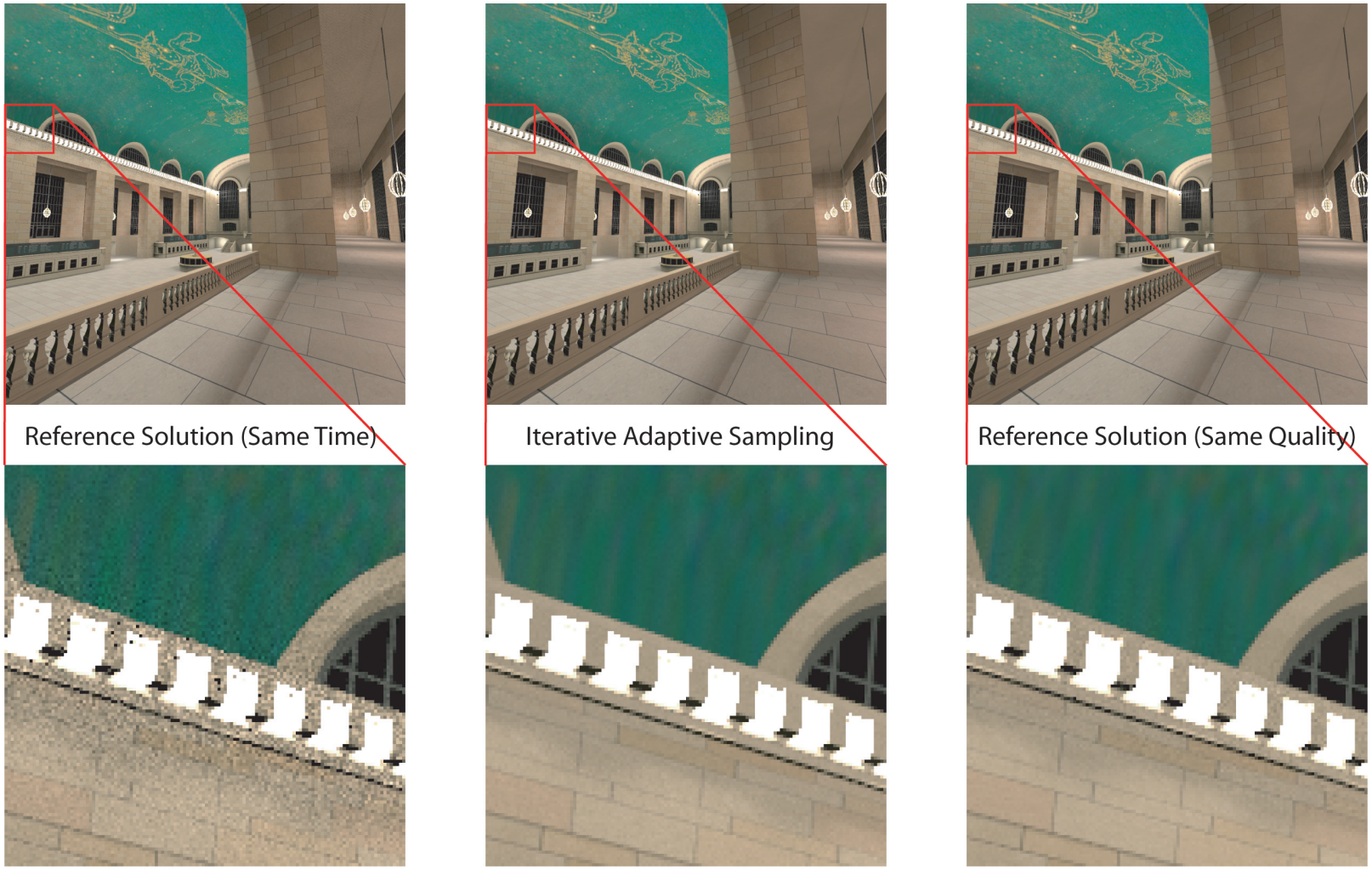

Figure 4.5: *Grand Central Terminal Qualitative Comparison. The bottom row contains closeups showing how the two algorithms compare when rendering soft shadows caused by many lights and many occluders*

Figure 4.6: *Kitchen Qualitative Comparison. The bottom row contains closeups showing how algorithm is better able to capture glossy highlights.*

Figure 4.7: *Kitchen with Environment Lighting Qualitative Comparison. The bottom row contains closeups showing the effectiveness of our algorithm when rendering scenes with direct illumination from environment maps.*

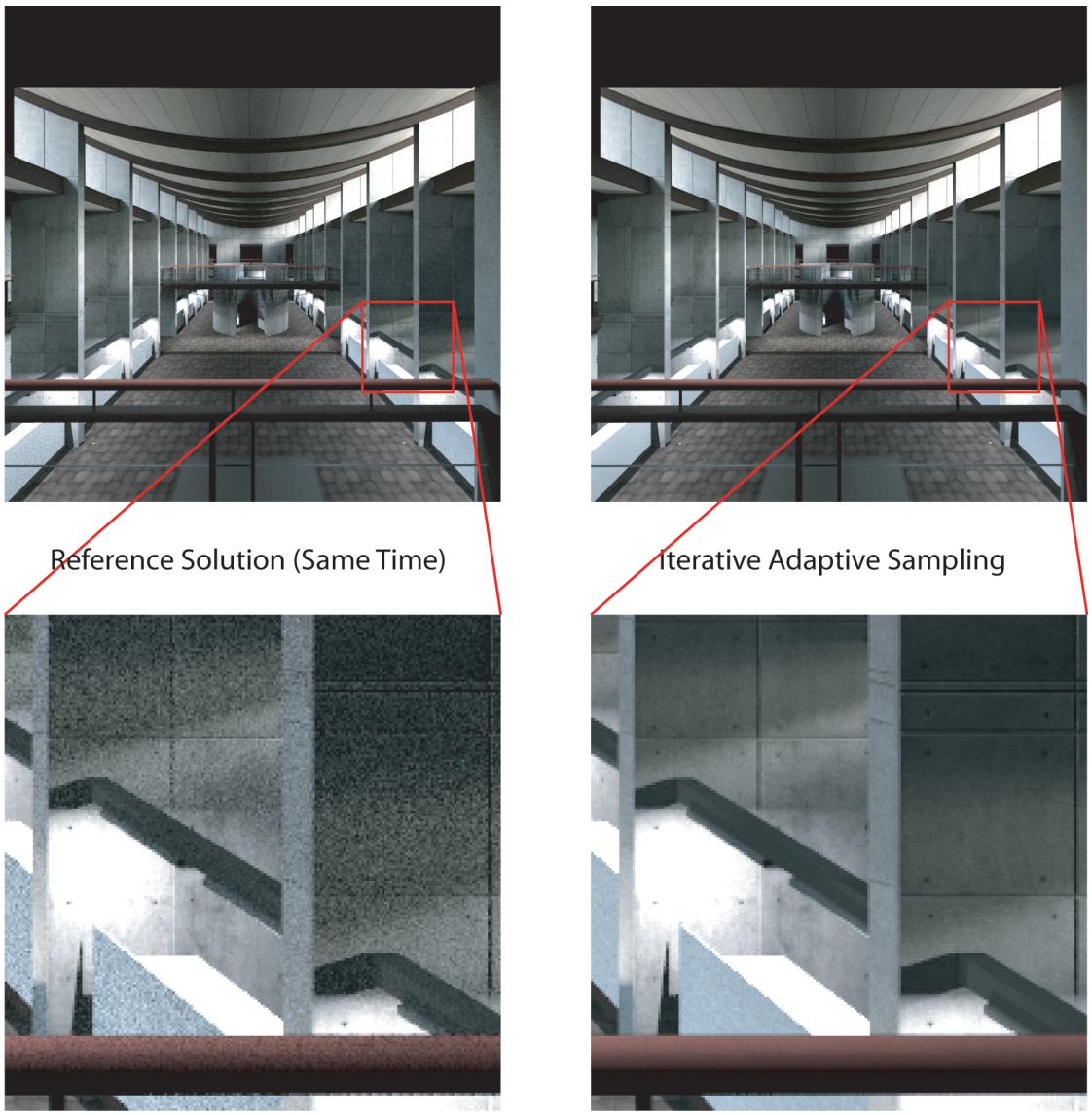

Figure 4.8: *Ponderosa Qualitative Comparison. In this scene, on average 96% of the lights are occluded for any point in this scene. The bottom row contains closeups that show how our algorithm can effectively render scenes with environment maps even when occlusion is very high.*

4.4 Visualizations and Detailed Results

In this section, we provide additional images and statistics to better explain the behavior of our algorithm. The explanatory figures reveal the effects of visibility and occlusion, the benefits of using an adaptive PDF, and the efficiency of the clustering and tree-cut routines.

4.4.1 Visibility and Occlusion

In Table 4.4 we provide details on the visibility of the light sources and environment map regions in each of the scenes. Note that for all scenes, a large majority of the lights and environment map regions contribute somewhere in the viewpoint. On a per-pixel level, the visibility statistics are quite different. The *Ponderosa* model has the largest disparity where 100% of the sources are visible at some intersection point in the image and on average only 4% are visible from individual surface points. Figure 4.9 is a false-color visualization of the combined per-pixel visibility of light sources and environment map regions.

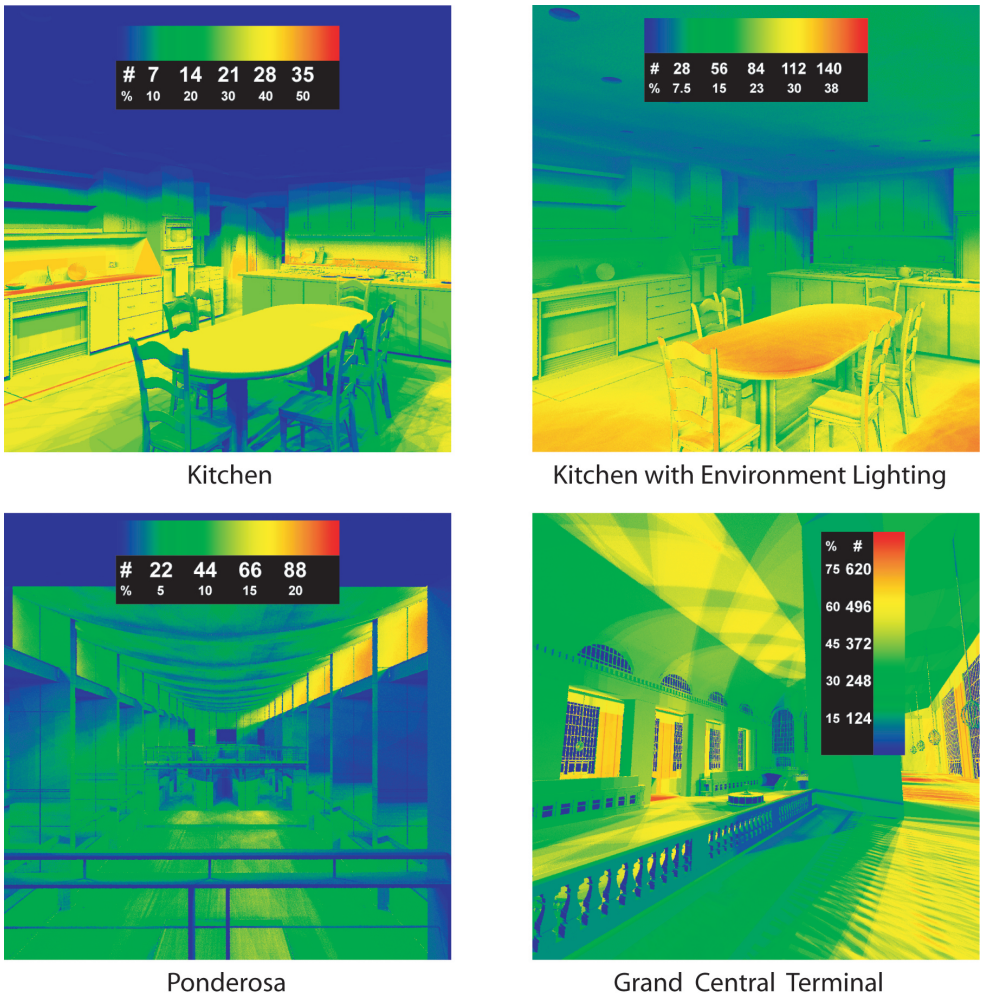

Figure 4.9: *Average Light Source Visibility False Color Visualizations. Blue represents surfaces where a great majority of the lights are occluded while red represents surfaces that have a greater number of contributing lights.*

Table 4.4: *Visibility Statistics*

Model	Total Lights	Environment Map Regions	Light Emitter Visibility		
			Per Viewpoint		Per Pixel
			Lights	Regions	Combined
Kitchen	72	0	62 (86.1%)	N/A	11.5 (16.0%)
Kitchen 2	72	300	62 (86.1%)	197 (65.7%)	64.1 (17.2%)
Ponderosa	138	300	138 (100%)	300 (100%)	17.5 (4.0%)
GCT	832	0	822 (98.8%)	N/A	279 (33.5%)

4.4.2 PDF Adaptation

Figure 4.10 shows how our PDF adapts to local lighting conditions at two different surface intersection points–one point (x_1) has more open visibility while the other point (x_2) is mostly in shadow. This environment has 832 lights, but it has been reduced to 103 clusters for point x_1 and 81 clusters for point x_2. From this figure we can see that the PDF for the first feedback-driven pass, where the block component PDF is weighted heavily, varies greatly from the initial uniform PDF each point starts with. With additional passes we are able to achieve more accurate PDFs due to a combination of two reasons: 1) the greater availability of samples and 2) the heavier weighting of the pixel component PDF that is able to better capture local lighting configurations. Consequentially, we have a reduction in the number of shadow rays cast.

After ten passes, the relative weights we assign for the uniform, block, and pixel component PDFs no longer change (see Figure 3.5). We do however continue evolving the probabilities within our component PDFs past that stage because we still collect sample data. After fifteen passes, we determined that additional sampling did not have much of an effect on final image quality for the scenes we tested.

4.4.3 Clustering

As described in Section 3.4, clustering reduces the effective number of lights we have to consider. The real issue is how large can we make the clusters to maximize efficiency without sacrificing the benefits they provide us. In [ARBJ03], the authors determine that 0.01 steradians is a conservative metric for the average visibility feature size of most scenes. This means that a single sample on a light source

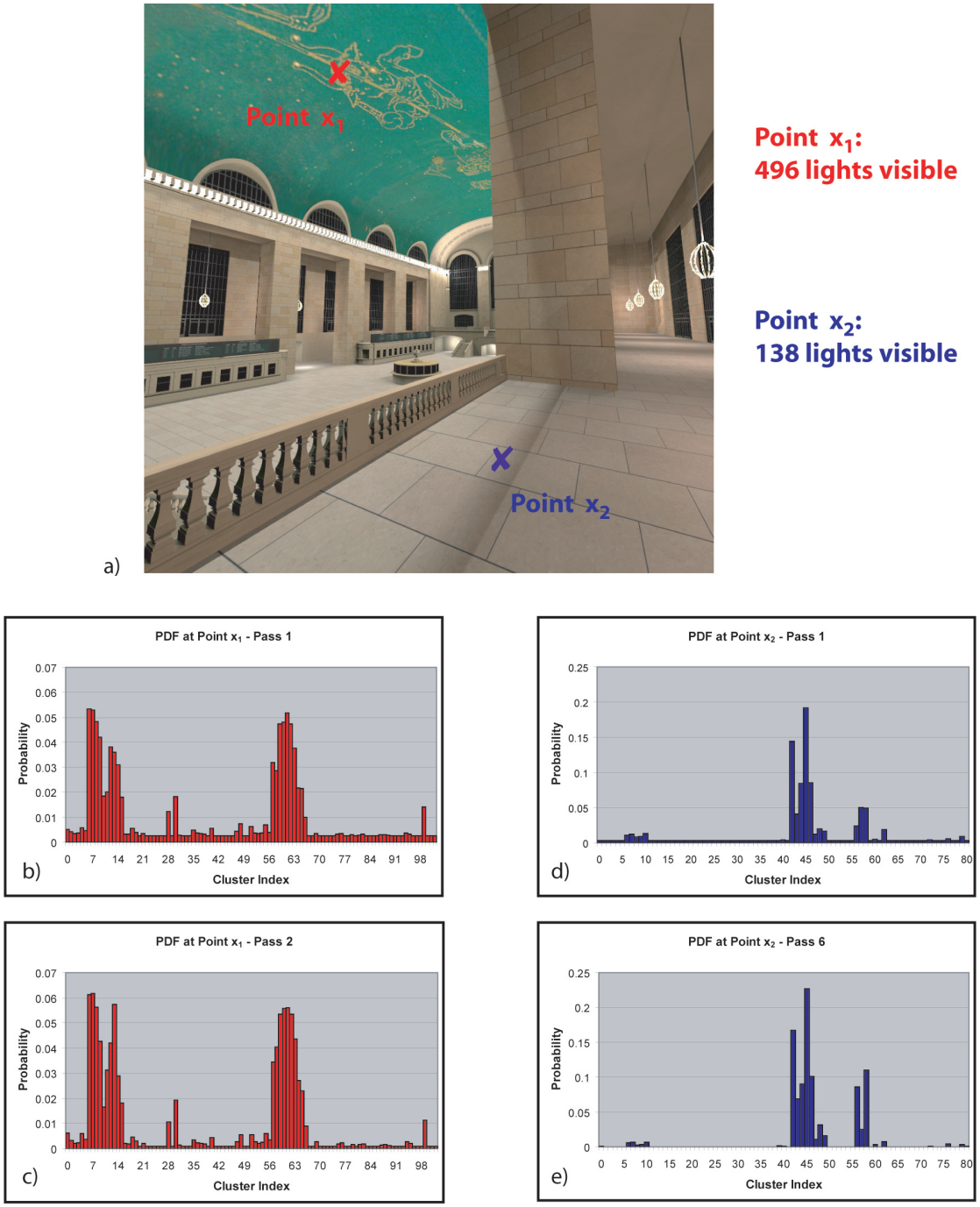

Figure 4.10: *a) A rendering of Grand Central Terminal with two points highlighted. b-e) The adaptation of PDFs for the two points. We have omitted the initial PDF (pass 0) for both passes since it is just uniform for all lights. Point x_1 requires two additional passes, while point x_2 requires six additional passes.*

(or environment map region or cluster) that subtends less than 0.01 steradians from a surface intersection point should reliably predict visibility. We use a more aggressive value of 0.02 steradians because we will sample each cluster multiple times. A higher cluster size maximum reduces the number of clusters in the cut which effectively makes PDF construction more efficient. Setting the threshold much larger than 0.02 steradians would significantly reduce spatial coherence across a cluster.

Table 4.5 provides statistics on the average number of clusters per cut in each of the four test environments. Remember that in our system we define a cluster as either a group of lights in the scene or an environment map region.

Table 4.5: *Clustering Statistics. The fourth column represents the average number of clusters per cut that are composed of grouped light sources, while the fifth column represents the the number of clusters per cut that are environment map regions. The two numbers combined represent the number of clusters per cut. The percentage in column four refers to the number of clusters in the cut when compared to the number of lights in the scene. In effect this shows the scalability of our clustering with the number of lights.*

Model	Total Lights	Environment Map Regions	Average Number of Clusters Per Cut	
			Lights	Map Regions
Kitchen	72	0	24.4 (33.9%)	N/A
Kitchen 2	72	300	24.4 (33.9%)	175
Ponderosa	138	300	47.6 (34.5%)	164
GCT	832	0	81.0 (9.70%)	N/A

4.5 Performance and Efficiency Analysis

Our algorithm performs better than the reference solution for two main reasons. The primary performance benefit is a result of evaluating fewer shadow rays. This is possible because of our adaptive PDFs, which yield low sample variance results.

The secondary benefit comes from lower overhead for PDF construction, which is affected by several factors. The use of clustering can have a very noticeable impact on PDF construction speed, especially in scenes with many lights such as *Grand Central Terminal* where the number of clusters in each cut is on average less than one-tenth the number of lights in the scene. Because we do not need to consider every single light source individually, our algorithm is far more efficient when defining a PDF over multiple light sources. Furthermore, due to our adaptive anti-aliasing, we can significantly reduce the number of viewing rays we have to trace. This in turn also reduces the number of PDFs we have to construct.

4.6 Implementation Details

In this section we list and discuss the various constants and settings we use in our algorithm. These constants are not critical to the understanding the Iterative Adaptive Sampling algorithm or its effectiveness, but they are necessary to someone implementing this algorithm. We used these parameters for all of our renderings and therefore believe them to be conservative and scene independent.

4.6.1 Shadow Rays Density per Pass

We sample the PDF a variable number of times per pass depending on the number of clusters in the cut, N_C. For all passes we sample the PDF $1.5 \times N_C$ times

per pixel appropriately distributed according to the sub pixel area associated with each pixel's intersection points. We choose $1.5 \times N_C$ as our metric because when combined with stratified sampling, it places at least one sample per cluster in the initial pass, assuming all surface intersection points share the same cut. While $1.0 \times N_C$ may provide a similar guarantee, we choose a more conservative value of $1.5 \times N_C$ because we know that all surface points within the pixel will not always have the same cut. We also set a minimum of 50 samples per pixel per pass, and a maximum of 150 samples per pixel per pass for all passes except the initial pass. We do not set a maximum for the initial pass since it is important to have at least one sample per cluster per pixel

As an optimization, at the end of a pass, we further sample a pixel with the same PDF if its relative variance is very close to the threshold. Given the combined sample variance of all previous passes and our threshold variance, we solve for the sample variance we need for a pass to reach the threshold. If the sample variance for the current pass is within a factor of two of the needed sample variance, we continue sampling the current pixel with the current PDFs associated with the points in increments of 25 shadow rays until we achieve the needed sample variance. This eliminates some overhead in building PDFs and it also allows us to sample the pixel in smaller increments when we are close to reaching convergence.

4.6.2 Avoiding Undersampling

While the main purpose of the uniform PDF is to ensure that we have a good distribution of samples to start with, it also ensures that we have an unbiased PDF. The problem arises when both the point and block PDFs assign a zero probability to a light that has a noticeable contribution. The additional weight

added by the uniform PDF is typically not enough to accurately represent the actual lighting and as a result we have noise in our PDF. If the block PDF fails to capture a contributing light, then there is no way the pixel PDF can account for this light since it only contains a subset of the block's sampling data.

To ensure that clusters are not prematurely given a zero probability in the block PDF, we make a special sample density requirement. If a cluster in a cut has not received any samples on the block level, or if it has received less than four samples, all of them being occluded, we set its probability to be equal to the average probability of all the other clusters in the cut rather than zero.

4.6.3 Anti-Aliasing

Recall that in our anti-aliasing algorithm described in Section 3.6 we trace a large number of primary visibility rays and group their surface intersection points to form a smaller set of surface points. Our metric to determine how many rays to trace is not only dependent on the number of unique surfaces we hit within a pixel, but also image resolution. At a higher resolution, we will have less aliasing than at a lower resolution, thus reducing our need for many primary rays. We use a ratio of twenty-five rays per unique surface for resolutions of 512×512, twenty rays per unique surface for image resolutions of 1024×1024, and a ratio of of fifteen rays per unique surface for resolutions above 1200×1200.

For the scenes and image resolutions that we tested, on average our adaptive anti-aliasing uses less than ten representative points per-pixel. This is a good result considering we group a minimum of twenty intersections and as many as sixty. See Table 4.6 for exact per-scene averages.

Table 4.6: *Additional Per Pixel Statistics. Recall that our adaptive anti-aliasing algorithm creates a set of "regions," which are a one-to-one mapping of intersection points that represent sub-pixel areas or regions.*

	Per Pixel Averages	
Model	Anti-Aliasing Regions	Rendering Passes
Kitchen	10.1	2.7
Kitchen 2	10.1	4.5
Ponderosa	9.8	3.3
GCT	8.8	5.5

4.6.4 Termination Condition

Through Figures 4.9 and 4.11 we can see that the most difficult pixels to render are those that have the greatest amount of occlusion. Though it may seem counter-intuitive, rendering time increases as the number of contributing light sources for each pixel decreases. The required number of light samples and shadow ray evaluations necessary to reach convergence for a dark pixel can be unreasonably large, especially if the pixel is near the black point[1] of the image. This is because our metric of relative sample variance is based on Weber's Law, which states that amount of error the human visual system can perceive in a pixel is proportional to the base luminance of the pixel. The error tolerance is far too conservative for very dark pixels because the limiting factor is actually our display device at those low luminances.

To prevent unnecessary oversampling, we apply a maximum cutoff on the number of shadow ray evaluations for a pixel. For our Iterative Adaptive Sampling algorithm, we set a maximum of fifteen passes. We chose this maximum because we

[1]The black point is the pixel intensity below which all values are mapped to black by our tone-mapper.

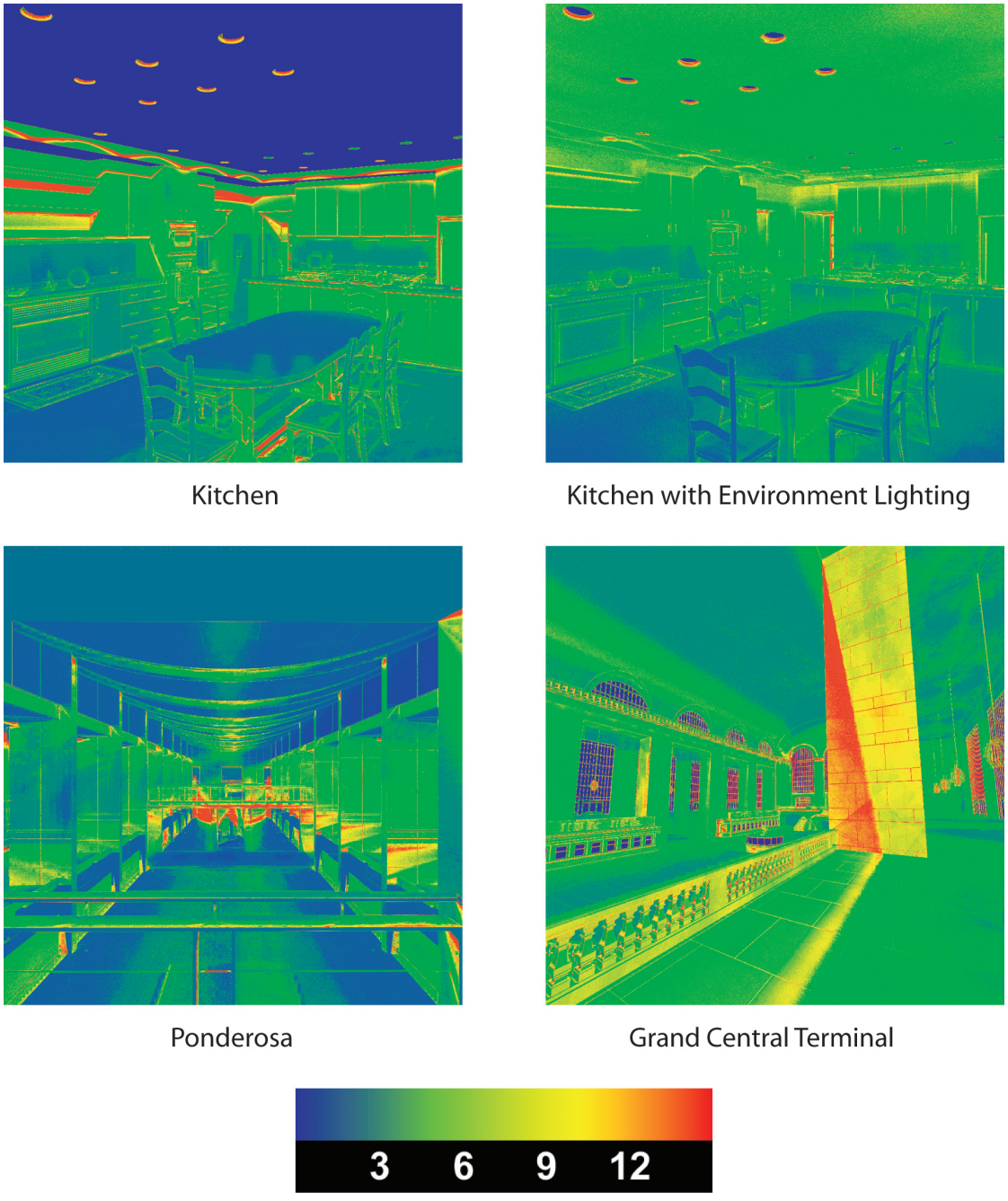

Figure 4.11: *Number of Rendering Passes Performed per Pixel*

found that the image does not show any perceptible improvement after a certain number of passes. For the reference solution we set a maximum of 20,000 shadow rays. The ideal solution would be to use a more sophisticated stopping criterion based on perceptual metrics.

On the other hand, it is also important not to prematurely label a pixel as black simply because all shadow rays evaluated for the pixel were occluded. For both our algorithm and the reference solution, we require a minimum number of shadow ray evaluations before determining that a pixel is completely in shadow and therefore black. These minimums are necessary to prevent speckling in our images from false-positives (i.e. prematurely labeling pixels as black).

Since our adaptive algorithm aggregates sample data across multiple pixels and multiple light sources, we can more reliably determine if a pixel is indeed black. In our algorithm, if we have not sampled a visible light after at least 3 passes and 250 shadow rays, we stop sampling the pixel and set it to black. This works very well in almost all situations and produces no false-positives. In both the *Grand Central* and *Ponderosa* scenes, the reference solution has difficulty in reliably detecting a fully occluded surface within a pixel unless given the much larger minimum threshold of 2,500 shadow rays. The optimum number for the reference solution is highly scene dependent, but in order to provide a valid comparison, we performed several renderings and set the value as low as possible for each scene.

Chapter 5

Conclusion

Through the use of intelligent sampling over the light sources, Iterative Adaptive Sampling is able to achieve nearly an order of magnitude speed-up over a standard Monte-Carlo ray tracer in architectural scenes containing complex direct illumination. We designed our algorithm based upon observations we made on average light source visibility patterns. While a scene with many lights may have nearly all the lights contributing in a single image, on average only a small subset of the lights actually affect the illumination of any surface intersection point. We are able to achieve our performance improvement as a result of several novel components working together to profit from our finding on the average visibility of light sources.

Our primary performance gain comes from reducing the amount time spent on the most expensive part of Monte-Carlo ray tracing, namely computing the visibility of light samples. We evolve adaptive probability density functions (PDFs) that automatically find and exploit conditions such as occlusion, glossy highlights, and partial visibility, which causes soft shadows. We rely on statistical feedback from previously computed light samples to guide the optimization of the PDF.

To increase the effectiveness and reliability of our sampling statistics we take advantage of the coherence in image space by considering our image in 8×8 pixel blocks. This allows us to aggregate sampling information for feedback on a block level as well as the pixel level. We use block and pixel sampling statistics to construct component block and pixel PDFs, respectively. These component PDFs are blended together along with a uniform PDF to generate a final PDF over the light sources. The initial pass uses only the uniform PDF since no feedback is yet available. Early feedback-driven passes weight the block PDF most heavily because it is averaged over the most data and converges faster. As more data becomes available, the more localized pixel PDF becomes more reliable and is given larger weights, since it is able to locally adapt more precisely.

We are also able to capitalize on the coherence of light source visibility in world space through the use of light clusters. Furthermore, we use these clusters to stratify our sampling more effectively. We also use a form of surface intersection point clustering on a per-pixel level to achieve low-cost pixel anti-aliasing. The use of clustering for light sources and intersection points also reduces the overhead needed for PDF construction. Because our algorithm renders images in small 8×8 pixel blocks, it is easily parallelized for distributed computing.

Our algorithm can handle many different types of light sources including point lights, area lights and environment maps for natural lighting. One of the greatest strengths of Iterative Adaptive Sampling as compared to sampling according to a fixed PDF is that the use of feedback data helps to greatly reduce the number of shadow ray tests to occluded light sources. One possible extension would be to also use feedback data to reduce the number of shadow rays evaluated for fully visible light sources. Often times a single sample can accurately determine the

contribution of an area light in the case that it is fully visible.

In summary, the use of an adaptive PDF along with clustering and sample aggregation has provided us with nearly a 10× performance improvement on several scenes with complex lighting. It is hoped that these types of strategies can be used in future algorithms to accelerate the computations for rendering complex real world scenes.

Appendix A

Proofs and Equations

A.1 Statistics

The average value of a function f is known as the mean μ. Sometimes it is convenient to use expected value notation $\langle\,\rangle$ to represent the average value of a function.

For direct illumination with area lights, we are trying to solve for this mean value. We do not know the actual distribution of the function, but we do have a set of N sample points[1] $(x_1 \cdots x_N)$ that estimate the mean. We refer to the average value of these samples as the sample mean, m, which is an unbiased estimator for μ.

$$m = \frac{1}{N} \sum_{i=1}^{N} x_i$$

[1] In our system $x_i = L(\vec{x}, \vec{y}_i)$, where $L(\vec{x}, \vec{y}_i)$ is defined by Equation 3.5

Variance, σ^2 is a measure of the deviation a set of data points has from its expected value.

$$\sigma^2 = \langle (x - \mu)^2 \rangle \tag{A.1}$$

Sometimes it is convenient to rearrange the terms in Equation A.1 as:

$$\sigma^2 = \langle x^2 \rangle - \mu^2 \tag{A.2}$$

Again, in the case where we do not know the underlying distribution, we may compute the sample variance of N samples as:

$$s_N^2 = \frac{1}{N} \sum_{i=1}^{N} (x_i - m)^2$$

s_N^2, however, is not an unbiased estimator of σ^2. Biased-corrected sample variance is defined as:

$$s_{N-1}^2 = \frac{1}{N-1} \sum_{i=1}^{N} (x_i - m)^2$$

The main problem with using s_N^2 over s_{N-1}^2 is that s_N^2 is likely to underestimate the variance, especially for a low sample count. Underestimating the variance will prematurely terminate rendering and result in noisy images. It is better to be conservative and oversample than to undersample and get a wrong result.

Since we use sample variance as a stopping condition, we not only want to be conservative, but what we really we want to know is the variance in our estimator of the mean. In effect we want to know how good of an estimate the sample mean m is of the actual mean μ. The estimator we use, s^2 does not converge to the final value of σ^2; rather it decreases linearly with the number of samples, N we use for

our estimator. This is the behavior we want because we expect a decrease in error with additional sampling.

$$s^2 = \begin{cases} V & \text{if } E < 0 \\ V - N \cdot E^2 & \text{otherwise} \end{cases} \tag{A.3}$$

where N is the number of samples and V and E are as defined below:

$$\begin{aligned} V &= \frac{\sum_{i=1}^{N} x_i^2}{N^2} \\ E &= \frac{m}{N} - \frac{3 \cdot \sqrt{V}}{N^2} \end{aligned}$$

A.2 Minimizing Cluster Variance

Here we provide the derivation of our weighting parameters for each light cluster based on sample data. Our derivation is based on area light sources, but we can extend it to apply to clusters as well due to their properties described in Section 3.4.1. We work with a few simple assumptions that we use in finding the variance-minimizing PDF. For any point receiving direct illumination we assume that:

(1) The exitant radiance due to a light source is uniform across the source

(2) Visibility is the only source of variance when sampling a cluster

These assumptions are not always true in practice, but visibility is definitely the largest source of variance and thus we will work to minimize variance due to partial visibility. It is important to note that we use these assumptions only to

approximate the ideal probabilities to assign to lights. Our algorithm does not depend on these assumptions being true.

Consider a single point $\vec{x}$ illuminated by two fully visible light sources contributing exitant radiance L_1 and L_2 at $\vec{x}$. If each light is partially visible by some amount u_i, then then exitant radiance due to each cluster will be $u_1 L_1$ and $u_2 L_2$ assuming the exitant radiance is uniform across the source. We want to find the distribution of samples between the sources that minimizes variance. Let p be the probability of sampling the first light source and $1 - p$ be the probability of sampling the second. For practicality, we use the definition of variance in Equation A.2. Thus, we can express the variance of total exitant radiance in this case as $\sigma^2(f)$.

$$\sigma^2(f) = \langle f^2 \rangle - \langle f \rangle^2 \tag{A.4}$$

$$\begin{aligned} \langle f^2 \rangle &= u_1 p \left(\frac{L_1}{p} \right)^2 + u_2 (1-p) \left(\frac{L_2}{1-p} \right)^2 \\ \langle f \rangle &= u_1 p \frac{L_1}{p} + u_2 (1-p) \frac{L_2}{1-p} \end{aligned}$$

If we differentiate $\sigma^2(f)$ (Equation A.4) with respect to p and set the result equal to zero we find that:

$$p = \frac{\sqrt{u_1} L_1}{\sqrt{u_1} L_1 + \sqrt{u_2} L_2}$$

$$1 - p = \frac{\sqrt{u_2} L_2}{\sqrt{u_1} L_1 + \sqrt{u_2} L_2}$$

In general for any number of light sources the probability p_i for light i that minimizes the variance is: $p_i \propto \sqrt{u_i}\, L_i$. If C is the contribution of a light source according to our sample data (which already contains the visibility term) and u is the unoccluded fraction of the light (also according to sample data) then we can assume $C = u\, L$ if the exitant radiance is uniform across the source. Therefore, we chose our sampling to be proportional to $\sqrt{u}\, L = \frac{\sqrt{u}\, C}{u} = \frac{C}{\sqrt{u}}$.

Bibliography

[AAM03] Ulf Assarsson and Tomas Akenine-Möller. A geometry-based soft shadow volume algorithm using graphics hardware. *ACM Transactions on Graphics*, 22(3):511–520, July 2003.

[Ama84] John Amanatides. Ray tracing with cones. In *Computer Graphics (Proceedings of SIGGRAPH 84)*, volume 18, pages 129–135, July 1984.

[AMA02] Tomas Akenine-Möller and Ulf Assarsson. Approximate soft shadows on arbitrary surfaces using penumbra wedges. In *Rendering Techniques 2002: 13th Eurographics Workshop on Rendering*, pages 297–306, June 2002.

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

[ARHM00] Maneesh Agrawala, Ravi Ramamoorthi, Alan Heirich, and Laurent Moll. Efficient image-based methods for rendering soft shadows. In *Proceedings of ACM SIGGRAPH 2000*, Computer Graphics Proceedings, Annual Conference Series, pages 375–384, July 2000.

[ARJ90] John M. Airey, John H. Rohlf, and Frederick P. Brooks Jr. Towards image realism with interactive update rates in complex virtual building environments. volume 24, pages 41–50, March 1990.

[BN76] James F. Blinn and Martin E. Newell. Texture and reflection in computer generated images. *Commun. ACM*, 19(10):542–547, 1976.

[CMS87] Brian Cabral, Nelson Max, and Rebecca Springmeyer. Bidirectional reflection functions from surface bump maps. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 273–281. ACM Press, 1987.

[CON99] Brian Cabral, Marc Olano, and Philip Nemec. Reflection space image based rendering. In *Proceedings of the 26th annual conference on*

Computer graphics and interactive techniques, pages 165–170. ACM Press/Addison-Wesley Publishing Co., 1999.

[DBB03] Phil Dutre, Philippe Bekaert, and Kavita Bala. *Advanced Global Illumination*. A K Peters, Natick, MA, 2003.

[FBG02] Sebastian Fernandez, Kavita Bala, and Donald P. Greenberg. Local illumination environments for direct lighting acceleration. In *Rendering Techniques 2002: 13th Eurographics Workshop on Rendering*, pages 7–14, June 2002.

[Fer04] Sebastian Fernandez. *Interactive Direct Illumination in Complex Environments*. PhD thesis, Cornell University, August 2004.

[Gre86] Ned Greene. Environment mapping and other applications of world projections. *IEEE Comput. Graph. Appl.*, 6(11):21–29, 1986.

[HDG99] David Hart, Philip Dutré, and Donald P. Greenberg. Direct illumination with lazy visibility evaluation. In *Proceedings of SIGGRAPH 99*, Computer Graphics Proceedings, Annual Conference Series, pages 147–154, August 1999.

[Ige99] Homan Igehy. Tracing ray differentials. In *Proceedings of SIGGRAPH 99*, Computer Graphics Proceedings, Annual Conference Series, pages 179–186, August 1999.

[KM00] Jan Kautz and Michael D. McCool. Approximation of glossy reflection with prefiltered environment maps. In *Graphics Interface*, pages 119–126, 2000.

[KVHS00] Jan Kautz, Pere-Pau Vázquez, Wolfgang Heidrich, and Hans-Peter Seidel. A unified approach to prefiltered environment maps. In *Rendering Techniques 2000: 11th Eurographics Workshop on Rendering*, pages 185–196, June 2000.

[LG95] David Luebke and Chris Georges. Portals and mirrors: Simple, fast evaluation of potentially visible sets. In *1995 Symposium on Interactive 3D Graphics*, pages 105–106, April 1995.

[MH84] Gene S. Miller and C. Robert Hoffman. Illumination and reflection maps: Simulated objects in simulated and real environments. *SIGGRAPH 84 Advanced Computer Graphics Animation course notes*, 1984.

[PPD98] Eric Paquette, Pierre Poulin, and George Drettakis. A light hierarchy for fast rendering of scenes with many lights. *Computer Graphics Forum*, 17(3):63–74, 1998.

[PS89] J. Painter and K. Sloan. Antialiased ray tracing by adaptive progressive refinement. In *Proceedings of the 16th annual conference on Computer graphics and interactive techniques*, pages 281–288. ACM Press, 1989.

[PSS98] Steven Parker, Peter Shirley, and Brian Smits. Single sample soft shadows. Technical Report UUCS-98-019, October 1998.

[RH01] Ravi Ramamoorthi and Pat Hanrahan. An efficient representation for irradiance environment maps. In *Proceedings of ACM SIGGRAPH 2001*, Computer Graphics Proceedings, Annual Conference Series, pages 497–500, August 2001.

[RH02] Ravi Ramamoorthi and Pat Hanrahan. Frequency space environment map rendering. *ACM Transactions on Graphics*, 21(3):517–526, July 2002.

[SG94] A. James Stewart and Sherif Ghali. Fast computation of shadow boundaries using spatial coherence and backprojections. In *Proceedings of SIGGRAPH 94*, Computer Graphics Proceedings, Annual Conference Series, pages 231–238, July 1994.

[SKS02] Peter-Pike Sloan, Jan Kautz, and John Snyder. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. *ACM Transactions on Graphics*, 21(3):527–536, July 2002.

[SS98] Cyril Soler and François X. Sillion. Fast calculation of soft shadow textures using convolution. In *Proceedings of SIGGRAPH 98*, Computer Graphics Proceedings, Annual Conference Series, pages 321–332, July 1998.

[SWZ96] Peter Shirley, Changyaw Wang, and Kurt Zimmerman. Monte carlo techniques for direct lighting calculations. *ACM Transactions on Graphics*, 15(1):1–36, January 1996.

[TS91] Seth J. Teller and Carlo H. Séquin. Visibility preprocessing for interactive walkthroughs. In *Computer Graphics (Proceedings of SIGGRAPH 91)*, volume 25, pages 61–69, July 1991.

[War94] Gregory J. Ward. The radiance lighting simulation and rendering system. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 459–472. ACM Press, 1994.

[WBS03] Ingo Wald, Carsten Benthin, and Philipp Slusallek. Interactive global illumination in complex and highly occluded environments. In *Eurographics Symposium on Rendering: 14th Eurographics Workshop on Rendering*, pages 74–81, June 2003.