2012

Algorithms For Rendering Optimization

Jared Johnson

University of Central Florida
ALGORITHMS FOR RENDERING OPTIMIZATION

by

JARED M. JOHNSON
B.S. Mathematics, University of Utah, 2002
B.S. Computer Science, University of Utah, 2005
M.S. Computer Science, University of Central Florida, 2007

A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in the Department of Electrical Engineering and Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Spring Term
2012

Major Professor: Charles E. Hughes
ABSTRACT

This dissertation explores algorithms for rendering optimization realizable within a modern, complex rendering engine. The first part contains optimized rendering algorithms for ray tracing. Ray tracing algorithms typically provide properties of simplicity and robustness that are highly desirable in computer graphics. We offer several novel contributions to the problem of interactive ray tracing of complex lighting environments. We focus on the problem of maintaining interactivity as both geometric and lighting complexity grows without effecting the simplicity or robustness of ray tracing. First, we present a new algorithm called occlusion caching for accelerating the calculation of direct lighting from many light sources. We cache light visibility information sparsely across a scene. When rendering direct lighting for all pixels in a frame, we combine cached lighting information to determine whether or not shadow rays are needed. Since light visibility and scene location are highly correlated, our approach precludes the need for most shadow rays. Second, we present improvements to the irradiance caching algorithm. Here we demonstrate a new elliptical cache point spacing heuristic that reduces the number of cache points required by taking into account the direction of irradiance gradients. We also accelerate irradiance caching by efficiently and intuitively coupling it with occlusion caching.

In the second part of this dissertation, we present optimizations to rendering algorithms for participating media. Specifically, we explore the implementation and use of photon beams as an efficient, intuitive artistic primitive. We detail our implementation of the photon
beams algorithm into PhotoRealistic RenderMan (PRMan). We show how our implementation maintains the benefits of the industry standard Reyes rendering pipeline, with proper motion blur and depth of field. We detail an automatic photon beam generation algorithm, utilizing PRMan shadow maps. We accelerate the rendering of camera-facing photon beams by utilizing Gaussian quadrature for path integrals in place of ray marching. Our optimized implementation allows for incredible versatility and intuitiveness in artistic control of volumetric lighting effects. Finally, we demonstrate the usefulness of photon beams as artistic primitives by detailing their use in a feature-length animated film.
To Lucy, Mom, and Dad
ACKNOWLEDGMENTS

I would like to thank my adviser, Charles Hughes, who has given me all of the support I needed semester after semester, and has always been available as both an advisor and a friend. I would like to thank Charlie for introducing me to my girlfriend, Lucy. She has been a tireless partner through everything, even from afar. I know I have a bright future as long as she is in it.

None of my successes would have been possible without my mom and dad. Beyond the literal truth of this fact, I feel incredibly privileged to have such amazing parents who have always given me tremendous support.

I would like to thank Peter Shirley for inspiring me to enter the field of computer graphics, showing me how cool it can be to combine calculus, statistics, and physics to make pretty pictures. He is an incredible source of knowledge and insight. I would also like to thank my committee members, Hassan Foroosh and Marshall Tappen, for their time, feedback, and support. Also, I would like to thank Erik Reinhard for advising me during the initial years of graduate school.

I would like to thank the Walt Disney Animation Studio for two amazing internships. Specifically, I would like to thank my mentors and collaborators: Wojciech Jarosz, Dylan Lacewell, Andrew Selle, Michael Kaschalk, Brent Burley, and Derek Nowrouzezahrai. Also, I would like to thank Adobe Systems for providing me with two rewarding internships. In particular,
I would like to thank Nathan Carr and Aravind Krishnaswamy for their mentorship and collaborations.

I would like to thank SREAL and UCF, specifically those who have helped me learn computer graphics throughout the years: Mark Colbert, Steven Braeger, Musawir Shah, Jaakko Konttinen, Kévin Boulanger, Juraj Obert, Remo Pilat, and Elena Erbiceanu. Special thanks to Alex Zelenin for the restaurant model.

Finally, I would like to acknowledge the UCF Trustees Doctoral Fellowship, NSF grants SES0527675 and DRL0638977, and the UCF Computer Science Department for funding my studies over the years.
TABLE OF CONTENTS

LIST OF FIGURES ................................................................. xii

LIST OF TABLES ................................................................. xiv

CHAPTER 1 INTRODUCTION .................................................. 1

1.1 Motivation ................................................................. 1

1.1.1 Shadows ............................................................... 4

1.1.2 Global Illumination .................................................. 6

1.1.3 Participating Media .................................................. 7

1.2 Contributions ............................................................... 9

CHAPTER 2 BACKGROUND .................................................. 11

2.1 Radiometry ................................................................. 11

2.1.1 Power ................................................................. 12

2.1.2 Radiance ............................................................... 13

2.1.3 Irradiance and Radiosity ............................................. 14

2.1.4 The Bidirectional Reflectance Distribution Function .............. 16

2.1.5 The Rendering Equation ............................................ 17

2.2 Direct Lighting ............................................................. 19

2.2.1 Sampling Methods .................................................. 21

2.2.2 Caching Methods .................................................... 22
2.3 Indirect Lighting .................................................. 23
  2.3.1 Monte Carlo Ray Tracing .................................. 23
  2.3.2 Irradiance Caching ........................................ 25
  2.3.3 Other Approaches .......................................... 31
  2.3.4 Our Direction of Exploration ............................ 32
2.4 Participating Media ............................................. 33
  2.4.1 Rendering Participating Media ........................... 36
  2.4.2 Designing Volumetric Lighting ............................ 39
  2.4.3 Reyes and the Production Pipeline ...................... 42

CHAPTER 3 OCCLUSION CACHING 44
  3.1 Cache Structure ............................................... 46
  3.2 Filling the Cache ............................................. 48
  3.3 Shadow Quality ............................................... 48
  3.4 Results ........................................................ 51
    3.4.1 Performance ............................................. 51
    3.4.2 Application ............................................. 55

CHAPTER 4 IRRADIANCE CACHE IMPROVEMENTS 57
  4.1 Anisotropic Cache Spacing .................................. 57
  4.2 A New Error Metric .......................................... 60
  4.3 Cache Tunneling ............................................. 61
  4.4 Results ........................................................ 64
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Frame rendered by a modern rendering engine</td>
<td>2</td>
</tr>
<tr>
<td>1.2</td>
<td>Ray traced shadows from many light sources</td>
<td>5</td>
</tr>
<tr>
<td>2.1</td>
<td>Surface projection</td>
<td>15</td>
</tr>
<tr>
<td>2.2</td>
<td>Domain of surface reflectance</td>
<td>18</td>
</tr>
<tr>
<td>2.3</td>
<td>Limited spatial variability of indirect lighting</td>
<td>26</td>
</tr>
<tr>
<td>2.4</td>
<td>Effects of irradiance gradients</td>
<td>29</td>
</tr>
<tr>
<td>2.5</td>
<td>Physical model of participating media</td>
<td>34</td>
</tr>
<tr>
<td>2.6</td>
<td>Domain of reflectance in participating media</td>
<td>35</td>
</tr>
<tr>
<td>2.7</td>
<td>Participating media model complexity</td>
<td>37</td>
</tr>
<tr>
<td>2.8</td>
<td>Volumetric photon mapping comparison</td>
<td>39</td>
</tr>
<tr>
<td>2.9</td>
<td>Mixed realism in animated movies</td>
<td>41</td>
</tr>
<tr>
<td>3.1</td>
<td>Shadow rays per-pixel</td>
<td>45</td>
</tr>
<tr>
<td>3.2</td>
<td>Scenes rendered for the benchmarks in table 3.1</td>
<td>50</td>
</tr>
<tr>
<td>3.3</td>
<td>Occlusion cache artifacts</td>
<td>52</td>
</tr>
<tr>
<td>3.4</td>
<td>Occlusion cache performance scaling</td>
<td>53</td>
</tr>
<tr>
<td>3.5</td>
<td>Occlusion caching on an immersive display</td>
<td>56</td>
</tr>
<tr>
<td>4.1</td>
<td>Anisotropic cache spacing</td>
<td>58</td>
</tr>
<tr>
<td>4.2</td>
<td>Irradiance cache sample locations</td>
<td>62</td>
</tr>
</tbody>
</table>
Figure 5.1  Photon beams from shadow maps ............................... 69
Figure 5.2  Impostor geometry comparison ................................. 72
Figure 5.3  Photon beam motion blur artifacts with PRMan .............. 75
Figure 5.4  Integrals of first-degree polynomials ........................... 80
Figure 5.5  Integrals of third-degree polynomials ........................... 80
Figure 5.6  Photon beams performance scaling .............................. 82
Figure 5.7  Keyhole shot from the movie Tangled ........................... 83
Figure 6.1  Photon beam shading components .............................. 88
Figure 6.2  Non-physical shading exploration ............................... 90
Figure 6.3  Comparison of keyhole scene production shots ............... 91
Figure 6.4  An early experiment with non-linear photon beams .......... 93
Figure 6.5  Non-linear photon beams in Tangled ........................... 95
Figure 7.1  Underwater volumetric lighting ................................. 103
**LIST OF TABLES**

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Radiometry quantities and symbols</td>
<td>12</td>
</tr>
<tr>
<td>3.1</td>
<td>Occlusion cache performance, varying cache radii</td>
<td>50</td>
</tr>
<tr>
<td>3.2</td>
<td>Occlusion cache one frame render times</td>
<td>54</td>
</tr>
<tr>
<td>4.1</td>
<td>Combining occlusion caching with irradiance caching</td>
<td>65</td>
</tr>
<tr>
<td>5.1</td>
<td>n-point Gaussian quadrature sample locations and weights</td>
<td>78</td>
</tr>
</tbody>
</table>
CHAPTER 1
INTRODUCTION

Modern rendering engines (figure 1.1) offer a breadth of complex tools which can benefit from optimized rendering algorithms that improve both quality and speed of existing approaches. This thesis showcases elegant and robust caching approaches utilizing a ray tracing framework in chapters 3 and 4. In chapters 5 and 6 we address rendering problems within the industry standard Reyes rendering architecture, presenting the implementation and development of an efficient, art-directable volume rendering tool. This chapter contains our motivation for these efforts and summarizes our contributions.

1.1 Motivation

Interactive ray tracing of high quality and complexity is a difficult problem. Ray tracing a scene with global illumination, many dynamic light sources, and dynamic geometry at high resolution with anti-aliasing is probably intractable without further advances in desktop computing power.

Until recently, ray tracing even visibility—that is, whether or not scene geometry is visible to a camera’s view frustum—was not possible at a high frame rate. Today, visibility testing in interactive graphics applications is still almost exclusively done using triangle rasterization
Modern rendering engines combine many rendering algorithms, handle high geometric complexity, and allow for detailed art-direction.

Figure 1.1: Frame rendered by a modern rendering engine

done on Graphics Processing Units (GPUs). The GPU pipeline has evolved to be increasingly flexible and programmable. Ray tracing methods have been utilized within the framework of the GPU pipelines, but the overriding framework of triangle rasterization remains the standard.

For visibility, rasterization has enjoyed an advantage over ray tracing, not only because it is more amenable to hardware acceleration, but also algorithmically. The invention of the GPU came at a time when triangle counts were low, and scene geometry was comparatively simplistic. Triangle count has increased with memory size and processing throughput. These triangles have also necessarily become smaller, covering fewer screen pixels. More compli-
cated scenes and models have also led to more scene triangles that are not observable from the camera perspective; i.e., the probability that a given triangle is occluded by other triangles from the camera’s perspective has increased as triangles become more numerous. These issues have led to solutions which necessarily complicate the previously simple rasterization pipeline.

The increase in triangle count and decrease in triangle size has now made ray tracing a much more attractive option, relative to rasterization, without complicating the simplicity of the algorithm. Again, rasterization is extremely efficient for large triangles, since each triangle is only accessed once and the pixels it covers are rapidly computed. With small triangles that barely cover a single pixel, the advantage of rasterization diminishes since the rays traced through each pixel, or sub-pixel, are much more likely to intersect unique triangles. This means that the number of times any given triangle is accessed in memory becomes similar whether using rasterization or ray tracing. Also, since ray tracing uses an acceleration structure to preclude shading occluded triangles, greater geometric complexity means a ray tracer never accesses many triangles in a scene that a naive rasterizer would. For these reasons, even using ray tracing for visibility testing scenes with high geometric complexity has become competitive with rasterization.

The point of drawing this comparison is not to argue that visibility testing in modern graphics applications should be done with ray tracing. Only that it is a viable option as scene geometry and shading complexity increases. Ray tracing has maintained much of its simplicity and
has shown superior robustness as computing power has facilitated much greater demands on 3D rendering pipelines. Therefore, we assume that increasingly many interactive rendering engines will utilize ray tracing not only for complicated shading, but exclusively throughout the pipeline.

Given the tools of a modern ray tracing engine, we aim to accelerate difficult rendering problems within this context while maintaining algorithmic simplicity and interactivity.

1.1.1 Shadows

Shadows are areas of a scene which lack visibility to some light source. Shadows are an area for which ray tracing has long-offered a straight-forward solution, while rasterization has been paired with many different algorithms for drawing shadows [EAS09]. The most widely-used algorithm in a rasterization rendering engine is shadow mapping. Although efficient, shadow mapping can become cumbersome when many different shadow-casting light sources are used.

Ray tracing offers an algorithmically-simple solution to many rendering problems, including shadows (figure 1.2). We aim to accelerate ray-traced shadows while maintaining this simplicity advantage over more popular rasterization methods. We observe that, for any given light, the visibility function is constant over large, connected areas in most scenes. Even
Ray tracing allows for algorithmically simple solutions to rendering problems like reflection, refraction, global illumination, and shadows.

Figure 1.2: Ray traced shadows from many light sources

though there are areas where visibility is changing in almost every pixel, it is rare for the visibility function to be high frequency over an area for every single light in the scene (Figure 3.1). We design a method called occlusion caching to take advantage of these constant visibility areas in order to accelerate the calculation of direct lighting.

Conceptually, occlusion caching can be thought of as the inverse of shadow mapping. Shadow maps store visibility information for the scene per-light. The occlusion cache stores visibility information for the lights spatially across the scene. Shadow mapping is a view of discrete
points in the scene from individual light sources. Occlusion caching is a view of the light sources from discrete points across the scene.

1.1.2 Global Illumination

Global illumination involves modeling both how light sources illuminate a scene (direct lighting), and how lit surfaces in a scene illuminate the scene. This recursion of surfaces lighting each other could continue infinitely. However, with each surface reflection, light power is diminished, making the first few light bounces the most visually important.

A frequently used algorithm for rendering global illumination is irradiance caching [KFC10]. Irradiance caching is an algorithm for accelerating the computation of inter-reflection of light on diffuse surfaces, surfaces which reflect light equally in all directions. The algorithm works by storing irradiance sparsely and then reconstructing irradiance per pixel in the final rendering through interpolation and extrapolation.

Irradiance cache spacing is determined by both surface orientation and how close objects in a scene are to each other. When stored in the cache, the information about scene geometry proximity is usually reduced to a single, directionless value. We note that in the area near corners where two flat surfaces meet, scene geometry is very close in less than half of the hemisphere. The resulting density of cache points is mismatched with the indirect
lighting function. We use the direction and magnitude of irradiance gradients to control cache spacing, allowing fewer cache points to be used in these areas.

1.1.3 Participating Media

Up to this point we’ve been concerned only with the lighting of opaque surfaces. The physical world is full of complicated phenomena that cannot benefit from the techniques introduced in the previous sections for lighting of opaque surfaces. One of the most interesting and widely modeled of these is participating media. Participating media includes fog, smoke, fire, and airborne particles, all of which can scatter light in any direction, absorb light, or even emit light. When shading opaque surfaces, we are only concerned with the hemisphere of directions above a surface point. With participating media, we must consider the whole sphere of directions. Also, opaque surfaces are flat and two-dimensional. Participating media is defined over a volume of space, where changing densities of this media can be incredibly complex.

The beauty and complexity of participating media requires rendering techniques that are more complicated in comparison to rendering opaque surfaces. Instead of visibility rays from the camera ”seeing” a single surface point, pixels in an image might ”see” an entire path through a volume of participating media. Consequently, instead of a single value of reflected or emitted light along this visibility ray towards the camera, we have a path of reflected or
emitted light. In order to calculate this total, we perform a path integral. A path integral is simply a one-dimensional definite integral. In the case of participating media, the path integral is evaluated over the interval a ray intersects the participating media volume.

When a participating medium has a constant density, there exists an analytic solution to this path integral. However, when the density of the medium varies spatially, numerical integration must be used. Traditionally, this integral is reduced to a simple Riemann Sum. We observe that the density or lighting of a media often varies smoothly. For such path integrals, the numerical method of Gaussian quadrature can estimate the path integral result with greater accuracy at much less computational cost. Consequently, we introduce the use of this quadrature rule in the domain of the rendering of participating media.

Most renderers are built from the paradigm of rendering geometry. Geometry is a good mathematical representation for physical surfaces, but not for participating media. Participating media is intrinsically mismatched to the capabilities of most rendering engines. Recently an algorithm called photon beams has been developed to allow for rendering participating media more efficiently, utilizing geometric primitives to represent physically-based lighting within a medium. We have implemented photon beams into an animated film production renderer.

The complicated nature of volumetric lighting has traditionally precluded artistic design or at the least made it labor-intensive. It is much simpler for artists to reason about geometry and utilize familiar tools than to think about manipulating the density or physical scattering
properties of a medium. In implementing photon beams, we have explored direct creation and manipulation of photon beams without physical simulation. We introduce a novel editing and design paradigm that can allow for not only fast physically-based results, but also provide effective tools for designing lighting within volumes.

1.2 Contributions

This dissertation will detail methods for accelerating the rendering of shadows, global illumination, and participating media. Our contributions to shadow and global illumination rendering come within the context of an interactive ray tracer. Our contributions to the design and rendering of participating media are presented within the context of a renderer used for computer generated visual effects and animated film production.

In chapter 2 we provide detailed background information relating to our contributions. In chapter 3 we present the occlusion cache, a method for accelerating the calculation of shadows in scenes with many light sources and high frequency occlusion functions. In chapter 4 we detail our integration of the occlusion cache with the irradiance cache, as well as our contribution to improving the placement of irradiance cache sample points and our optimization to the acceleration structures used in irradiance caching. In chapter 5 we present the novel use of Gaussian quadrature for estimating path integrals and rendering participating media efficiently. We also walk through our implementation of photon beams into a production ren-
derer. In chapter 6 we present a system for artistic design of volumetric lighting, using our implementation of fast participating media rendering and detail its use in a feature-length animated film. Finally, in chapter 7 we discuss our results and describe possible future work.
Interactive ray tracing on desktop PCs has been around since the late 1990s. Ray tracing quality and speed has increased dramatically over this time period. This is due in part to processing power improvements, and in part to algorithmic improvements.

Since most interactive graphics research over this time period has been done within the domain of the GPU pipeline, much of the background research presented in this chapter was originally purposed for offline rendering systems. However, hardware advancements have made much of it relevant to interactive ray tracing.

This chapter presents background information for our rendering contributions.

2.1 Radiometry

In computer graphics, physically-based rendering relies on the abstractions of both geometric optics and radiometry to render realistic images. Geometric optics is the modeling of light propagation as discretized rays: an idealized, one-dimensional beam of light. Ray tracing is the simulation of light propagation with these discretized rays. Radiometry details the
Table 2.1: Radiometry quantities and symbols

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Φ</th>
<th>E</th>
<th>B</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quantity</td>
<td>Power</td>
<td>Irradiance</td>
<td>Radiosity</td>
<td>Radiance</td>
</tr>
</tbody>
</table>

measurement of electromagnetic radiation, including visible light. Radiometry, in terms of geometric optics, is foundational to computer graphics.

2.1.1 Power

Most radiometric quantities are measured in terms of power. In radiometry, power is calculated over all wavelengths of electromagnetic radiation. In computer graphics, we are only interested in visible light: electromagnetic radiation with wavelengths between 400 and 800 nanometers. In general, power in computer graphics is further simplified to calculations done at only three discrete wavelengths. We calculate power at only three discrete wavelengths because our color displays only require this data to display a color image.  

As with the discretization of radiometry into three color channels, the main goal of most computer graphics is rendering of images to color displays. Our rendering is calculated with the output in mind. For this reason, it is unusual to formalize rendering results to absolute

---

1Power over a narrow band of electromagnetic radiation is known specifically as Spectral power. Computing radiometric quantities in terms of Spectral power is known as Spectroradiometry. However, being this precise can be cumbersome and it is rarely done in computer graphics literature. Likewise, we will trade precision for simplicity and not distinguish between Radiometry and Spectroradiometry.
units. Instead, power is measured as a relative quantity, relative either to other values within a rendering, or to the output capabilities of the color display being used.

Power is energy per unit time. In addition to being discretized into measurement over narrow wavelength bands of the electromagnetic spectrum and rarely being measured in absolute terms, the time component of power calculations is also usually only measured implicitly, in relative terms.

### 2.1.2 Radiance

Radiance represents the power per unit solid angle per unit projected area. In a color rendering, we can think of each final pixel color as representing a radiance measurement. The surfaces in the image are projected from the 3D scene world into a 2D image. We see the power reflected off of these surfaces through the solid angle of each pixel area, relative to some virtual eye location. We can see how this represents a measurement of power per unit solid angle per unit projected area since we average over the pixel to arrive at a final pixel color.

With ray tracing, we shoot virtual rays from the eye through the pixels to calculate the reflected power that passes through each. Each ray is a radiance query where we calculate the radiance $L$ reflected in the ray direction $\hat{v}$ from the surface intersection point $\vec{p}$, specifically
$L(\vec{p},\hat{v})$. In order to calculate the reflected radiance, we need to know how much power is incident on the surface (irradiance) and also model in some way how incident light is reflected.

\subsection{2.1.3 Irradiance and Radiosity}

Irradiance is how much power is incident on a surface per unit area. As a first step to calculating how much power a surface reflects in any given direction, we need to know how much power is incident on that surface. Given an incoming radiance value for some small solid angle $d\omega$, we can convert to irradiance by projecting that radiance value to the surface: multiplying by the cosine of the angle between $\omega_i$ and the surface normal $\hat{n}$: $\theta_i$ (Equation 2.1). This allows us to know the contribution from some small part of the hemisphere on the power per unit area of a surface (figure 2.1).

\begin{equation}
  dE(\omega_i) = L_i(\omega_i)\cos(\theta_i)d\omega_i, \quad \cos(\theta_i) = \omega_i \cdot \hat{n}
\end{equation}

The simplest way surfaces can reflect light is equally in all directions. These surfaces are known as perfectly diffuse or Lambertian. Radiosity is a useful concept in such instances.
From direction $\omega_i$ for some small solid angle $d\omega_i$, light power is spread over a larger area when $\omega_i$ is more perpendicular with the surface normal $\hat{n}$. This means there is less power per unit area, or irradiance. Irradiance is proportional to the cosine of the angle between $\omega_i$ and $\hat{n}$.

Figure 2.1: Surface projection

Radiosity is the total power reflected and emitted from a surface per unit area (also known as Radiant exitance). Both irradiance and radiosity have the same units, since they are describing similar phenomena just in opposite directions. If we know the total amount of light incident on a Lambertian surface, we also know how much light is reflected. In this case, radiosity $B$ is equal to $\rho E$, where $E$ is the irradiance, and $\rho$ is the albedo (percent of radiation reflected) of the surface. Once we’ve calculated the radiosity of a surface, we know the reflected radiance in any direction. We just divide the radiosity value by the total projected solid angle of the hemisphere, $\pi$, giving us the radiosity per unit projected solid angle which is also the reflected radiance.
2.1.4 The Bidirectional Reflectance Distribution Function

The bidirectional reflectance distribution function (BRDF) is a convention for describing power transfer at an opaque surface [Nic65]. The function describes the ratio of outgoing radiance to incoming irradiance as a function of outgoing and incoming angles (Equation 2.2, see figure 2.2). A BRDF can also be dependent on spatial location, time, and wavelength. Since we usually only calculate reflectance for three color channels, it’s common for the wavelength-variance of a BRDF to be implicit. That is, we write it as a function without parameterizing by wavelength but the calculation is done three times, once for each color channel.

\[
f_r(\omega_i, \omega_o) = \frac{dL_o(\omega_o)}{dE_i(\omega_i)}, \quad (2.2a)
\]

\[
f_r(\omega_i, \omega_o) = \frac{dL_o(\omega_o)}{L_i(\omega_i)\cos(\theta_i)d\omega_i} \quad (2.2b)
\]

A BRDF allows us to simplify complicated surface reflectance phenomenon into a single function. To compute the differential contribution of reflected radiance in the direction \(\omega_o\) reflected from direction \(\omega_i\), we calculate incoming radiance from \(\omega_i\), project it to the surface by multiplying by the cosine of \(\theta_i\), and multiply by the BRDF (Equation 2.3).
Conceptually, the BRDF specifies the proportion of light that is reflected in the outgoing direction, for some small solid angle of the hemisphere. In practice, this function can be completely arbitrary. If geometric optics are to be modeled accurately, a BRDF must be reciprocal, \( f_r(\omega_i, \omega_o) = f_r(\omega_o, \omega_i) \), and not reflect more light than is incident, \( \forall \omega_i \int_{\Omega} f_r(\omega_i, \omega_o) \cos(\theta_o) d\omega_o \leq 1 \) (see appendix A).

For example, the BRDF of a Lambertian surface is simply \( \rho/\pi \). Again, \( \rho \) represents the albedo of the surface. The integral of this function over the hemisphere with respect to projected solid angle, \( \int_{\Omega} \frac{\rho}{\pi} \cos(\theta_o) d\omega_o \), is just \( \rho \). Consequently, for a Lambertian surface to reflect less light than is incident, \( \rho \) must be less than one.

### 2.1.5 The Rendering Equation

We can now detail mathematically the equation we attempt to solve when rendering an image using the physically based concepts of Radiometry. Equation 2.4 details physically-based rendering and is known as the rendering equation [Kaj86].

\[
dL_o(\omega_o) = f_r(\omega_i, \omega_o)L_i(\omega_i)\cos(\theta_i)d\omega_i
\]
We have taken the equation for differential reflected radiance (Equation 2.3) and simply integrated over the entire hemisphere of directions. We then add a dependence on position \( \vec{p} \), since we are performing this calculation at many different positions in a scene and the BRDF can vary spatially over these positions. We also model light emission \( L_e(\vec{p}, \omega_o) \) at any point in a scene, allowing the specification of light sources. As stated earlier, this calculation can also be dependent on time and wavelength. Put simply, the total radiance reflected from point \( \vec{p} \) in direction \( \omega_o \) is the sum of emitted radiance and reflected radiance.
When rendering an image, for every surface point $\vec{p}_i$ that is visible along the direction $\omega_o$ we perform the rendering equation calculation. This simple equation is central to every computer graphics rendering algorithm. Although it is a simple formulation, the difficulty comes with the fact that the incoming radiance over the hemisphere of directions is not known. Each unknown incoming radiance value requires another rendering equation calculation. Doing this calculation recursively until a solution converges is incredibly computational intensive. For this reason, most graphics algorithms detail ways to simplify this calculation.

2.2 Direct Lighting

In computer graphics, direct lighting is modeled as incoming radiance to the camera emitted from light sources reflected from precisely one surface. This means the rendering equation is simplified to calculating incoming and reflected radiance only in the directions of light sources. Also, reflected radiance is not computed from these directions, only emitted radiance, so there is no recursion. Put in mathematical terms, over all light sources $l_i$, the rendering equation becomes:

$$L_o(\vec{p}, \hat{v}) = L_e(\vec{p}, \hat{v}) + \sum_{i=1}^{k} f_r(\hat{l}_i, \hat{v})L_i(\vec{p}, \hat{l}_i)(\hat{l}_i \cdot \hat{n})$$

(2.5)
When light sources are abstracted to points or directions, this calculation is very efficient. Early on, researchers formulated several efficient formulae to model the reflected radiance [Gou71, Pho75] from point lights or directional lights. However if lighting is to more closely resemble our physical world, this calculation becomes more complicated. Physically-based lighting requires a more complex model including both area light sources and light source visibility (whether or not a shading point can ‘see’ a light source). These more physically-correct abstractions bring us to the main difficulty of the direct lighting calculation: rendering realistic shadows.

The most widely-used shadow algorithm in rasterization pipelines is shadow mapping [Wil78]. Shadow mapping is a two-pass algorithm. First, depth images are rendered from the point of view of each light source, then the final image is rendered. At each pixel in the virtual camera’s perspective, the corresponding shadow map pixel is calculated by projecting the shading point into each shadow map. Visibility between light source and shading point is calculated by comparing the depth value of the corresponding shadow map with the distance between the light source and the point being shaded. If the distance between light source and shading point is greater than the depth value of the shadow map, the point is occluded.

Shadow volumes [Cro77] is another popular algorithm for rendering shadows in rasterization rendering pipelines. A shadow volume is the three-dimensional space an object occludes from a light source. Shadow volumes are created in a pre-pass by rendering from the point of view of the light source, finding object silhouettes, and extruding the silhouettes into
This geometry is not directly drawn in the final render, but instead used to classify shading points as inside or outside the shadow volumes. This allows for very clean shadow boundaries.

Although ray tracers could employ shadow mapping or shadow volumes, a ray tracing framework gives us the tools needed to calculate visibility between shading points and light sources in a way that is incredibly simple and robust, in contrast to these rasterization-based methods. However, calculating visibility for many light sources or area lights using naive methods can quickly bottleneck a ray tracer. For this reason, algorithms for accelerating ray-traced direct lighting are numerous [PH04]. We have separated an overview of some of these algorithms into the next two sections, starting with Sampling Methods and followed by Caching Methods.

### 2.2.1 Sampling Methods

Wald et al. [WBS03] construct a Probability Density Function (PDF) for currently visible light sources by tracing a few random paths through the scene. This PDF is then utilized to render the full frame, increasing the probability that visible lights are sampled. This approach accelerates direct lighting only when some lights that would be sampled naively do not contribute to the current frame being rendered.
Donikian et al. [DWF06] iteratively refine light sampling PDFs for scenes with many area and point lights. Though we have not attempted to integrate this algorithm into a more interactive context, we believe that the overhead of multiple-pass refinement relegates it to accelerating only extremely complex lighting scenarios of dozens of area lights or hundreds of point lights. Also, the approach of Donikian et al. uses a light's contribution to pixel values to calculate PDFs, meaning that these PDFs are view-dependent.

2.2.2 Caching Methods

Fernandez et al. [FBG02] accelerate direct lighting by sub-dividing space and storing which geometry and which lights influence the lighting calculation in each spatial sub-division. This method effectively speeds up the lighting calculation, but requires a fair amount of pre-computation.

Clarberg and Akenine-Mller [CA08] cache visibility per-strata at sparse cache points in world space and use the cached visibility functions as control variates in Monte Carlo integration. Their method is directed at environment map lighting using Monte Carlo integration.
2.3 Indirect Lighting

As detailed in the previous section, direct lighting describes radiance from camera-visible surfaces reflected directly from light sources. Consequently, indirect lighting describes incoming radiance to the camera emitted from light sources that has been reflected off more than one surface.

2.3.1 Monte Carlo Ray Tracing

The simplest way of combining direct and indirect lighting is to indiscriminately solve the rendering equation, treating direct and indirect lighting as identical contributions to reflected radiance at shading points. Unfortunately, there exists no general, analytic solution to the rendering equation, only numerical solutions which converge to an unbiased, accurate result in the limit. The most popular numerical approach in computer graphics is that of Monte Carlo integration.

Monte Carlo integration is a broad category of numerical integration techniques involving random numbers. It is an incredibly powerful tool for solving difficult, multi-dimensional integrals. An estimate for the rendering equation (equation 2.4) using basic Monte Carlo integration is shown in equation 2.6, using uniform samples over the domain of the hemisphere of surface-incident directions $\Omega_i$. 
\[ L_o(\vec{p}, \omega_o) \approx L_e(\vec{p}, \omega_o) + \frac{1}{N} \sum_{i=1}^{N} f_r(\vec{p}, \omega_i, \omega_o)L_i(\vec{p}, \omega_i)\cos(\theta_i) \] (2.6)

We select \( N \) random, uniform samples in the domain of the hemisphere above our shading point. This domain can be as many dimensions as needed, as long as the space is sampled uniformly. A Monte Carlo estimate turns a previously complex definite integral into random number generation and a simple summation of function evaluations. Even with rendering models which include integration over time or wavelength, the general form of the integral does not change. For higher-dimensional integrals, more samples are probably required, but the only additional complexity is sampling higher-dimensional space.

Monte Carlo integration is incredibly simple, powerful, and robust. The cost of this powerful simplicity is computation time. In fact, to decrease error by a factor of ten, \( N \) must increase by one-hundred times, since error is proportional to \( 1/\sqrt{N} \). Non-uniform distribution of sample points can be chosen to increase the speed of Monte Carlo integration, like importance sampling and stratified sampling. However, the general nature of the technique usually precludes performance on par with more specialized rendering methods. Although, with careful importance sampling Monte Carlo performance can be improved dramatically.

Importance sampling requires some knowledge of the integrand so that samples can be distributed commensurately to areas most likely to contribute to the final integral value. This requirement can preclude the use of importance sampling, because in many cases knowledge
of the integrand is simply not possible. A common example of this is the case of indirect lighting on diffuse surfaces. With diffuse surfaces the BRDF is constant in all directions, and it is difficult to know the source of indirect lighting a priori. One way to increase efficiency is to separate unknown lighting from known lighting. Specifically, importance sampling can be used for the direct lighting component, since light source locations are usually known, and general Monte Carlo integration can be used for indirect lighting. In the next section we develop this idea further and introduce an algorithm for amortizing the cost of Monte Carlo integration of indirect lighting over many screen pixels.

2.3.2 Irradiance Caching

Direct and indirect lighting are both reflected radiance from camera-visible surfaces. Therefore, differentiating the two is not entirely necessary. However, doing so usually leads to increased efficiency, since direct lighting is probably more important visually than indirect lighting. Separating the two allows us to more explicitly decide how much computation time we want to dedicate to each part of a scene’s lighting. Also, the general character of these two lighting components is very different. Direct lighting often has sharp discontinuities and indirect lighting is usually smooth, with little spatial variance (see figure 2.3). We can refer to this separation of lighting contributions as reflected radiance decomposition.
Figure 2.3: Limited spatial variability of indirect lighting
The term ‘caching’ in rendering usually refers to the reuse of data from one part of a render or a previous time step to render other parts faster. Caching exchanges memory storage for computation time. We save the results of past work, and check whether or not we can use these results for subsequent calculations. The overall effect on the workflow of algorithms employing caching is minor. We are not necessarily adding to an algorithm’s complexity, but just trying to find instances where work does not need to be repeated.

These two ideas–reflected radiance decomposition and caching–form the foundation for irradiance caching [WRC88]. Decomposing reflected radiance into direct and indirect lighting led to the observation that indirect lighting and spatial location are highly correlated. A simple way to exploit this correlation and amortize work over many nearby pixels was by utilizing caching.

Irradiance caching works by separating out the lighting caused by inter-reflection among diffuse surfaces from direct lighting in the rendering equation [Kaj86]. Due to its low frequency nature, indirect lighting can be sparsely evaluated over the scene at a set of carefully chosen sample points. Interpolation can be performed to reconstruct the signal over the entire scene. At each sample point, Monte Carlo integration is used over the hemisphere of directions to calculate an accurate estimate of irradiance. In addition to this estimate, information from hemisphere sampling is used to estimate how likely spatial changes in lighting. In the original formulation, the harmonic mean of the distances to intersection points in hemisphere sampling is used. This number is linearly scaled by a user-provided
parameter to designate how close new irradiance calculations must be to stored records in order to have a cache hit.

Originally, hemisphere integration was only used to calculate a single irradiance value and average distance to intersection points. However, the hundreds or even thousands of ray traces required for this integral contains a lot more information about the lighting at a given point. Ward introduced irradiance gradients [WH92] for irradiance caching to better utilize this information. Irradiance gradients are computed during hemisphere sampling and are used to improve interpolation, extrapolation, and cache spacing [SM02] of the irradiance cache. Irradiance gradients are an estimate of the magnitude of irradiance change for each spatial direction and rotation angle. Using irradiance gradients results in a marked improvement in interpolation quality with little additional computation and no additional ray tracing. Figure 2.4 shows how well irradiance gradients work to smooth out the appearance of indirect lighting rendered with irradiance caching.

Because irradiance caching is a world-space caching approach, distant surfaces which cover few pixels in a rendered image, but a large area in world-space can result in too many cache misses. Tabellion and Lamorlette introduced an error metric and weighting function to space cache points evenly in screen space [TL04], replacing the harmonic mean calculation with a minimum, clamped to ensure adjacent image pixels on the same surface do not create new cache records. This optimization serves to place irradiance cache points in a way
Figure 2.4: Effects of irradiance gradients
more sensitive to the final output image. To our knowledge, this technique marks the first published documentation of irradiance caching being used in a feature animated film.

In 2006, Krivánek et al. introduced neighbor clamping and adaptive caching \cite{KBP06}, reducing the area in which poor estimates are visible and increasing the density of cache points where a visual discontinuity of irradiance is detected, respectively. Neighbor clamping works by looking at nearby cache points when a point is added to the irradiance cache and checking whether or not the radius of the cache point being added is geometrically plausible. This process helps prevent cases where hemisphere sampling misses small, nearby scene objects, or rays leak through geometry because of numerical precision problems. Also when adding a cache point to the irradiance cache, adaptive caching can detect whether or not interpolation between nearby cache points will result in a visible discontinuity. Once detected, this discontinuity can be prevented by reducing the radius of the cache point to be added and nearby cache points, consequently increasing the local density of cache points.

At SIGGRAPH 2008 work was presented for enhancing the smooth appearance of indirect illumination by increasing the area that irradiance cache points overlap \cite{KGW08}. This change causes a simple blurring of the indirect lighting function an irradiance cache represents. More cache points are used in any given interpolation, reducing the influence of outliers, trading local accuracy for a generally smoother appearance.
2.3.3 Other Approaches

Radiosity [GTG84] is one of the earliest algorithms for computing reflection between diffuse surfaces. It discretizes the scene into surface patches and computes diffuse light transport between all possible pairs of patches in the scene. Although this can be an expensive pre-process, subsequently the scene can be rendered with full global illumination from any camera position.

Photon Mapping [JC95] uses a pre-process to trace rays from light sources and store scene intersection points as 'photons' in a spatial database. In the final render, the density of these deposited photons is used to estimate reflected radiance for the indirect lighting calculation. Photon mapping is sometimes used in concert with irradiance caching, since viewing the photon map directly can be artifact-prone.

Caching has long been employed in many ways to accelerate rendering. The Render Cache [WDP99, WDG02, VLB06] stores computed pixels in screen space and reprojects them as the view changes. The approach maintains visible samples and relies on inter-frame coherence. More general schemes for storing view independent shading data in caches are possible [DS07].

Forgoing caching, irradiance filtering [KRK04] takes many noisy estimates over an area and combines them in a filtering operation, removing high-frequency noise. This work shows some efficiency improvements over irradiance caching but does not allow acceleration of
walkthroughs of a static scene. Also, the technique is not nearly as well-tested as irradiance caching, and may prove to be less-robust.

Radiance caching [KGB05, KGP05] is a technique very similar to irradiance caching, but extends caching and interpolation to glossy surfaces by storing the full incoming radiance function at each cache point with spherical harmonics. Arikan et al. decompose irradiance into near and far components, using radiance caching for far components, and a local geometric query to calculate irradiance contribution from near geometry [AFO05].

It is worth mentioning some other parallel improvements to irradiance caching which are not directly related to the work of this thesis. However, there is nothing precluding the integration of our work with these other advancements: Irradiance caching has been implemented in graphics hardware [GKB05], implemented on large computing clusters [DSC06], made more robust for animation [GBP07, SKD05], and improved in the presence of participating media [JZJ08b].

### 2.3.4 Our Direction of Exploration

The original irradiance caching algorithm performed well, but—as can be inferred from the literature—there has been significant motivation and success in improving the algorithm. Hemisphere sampling and integration is incredibly expensive. Even though using an irra-
dance cache can be orders of magnitude faster than rendering without a cache, it still can represent a bottleneck in the rendering pipeline. For this reason, there is continuing research into reducing the number of cache points used or optimizing the irradiance cache as a whole.

We chose to pursue research in irradiance caching because it represents a simple acceleration to a slow but robust rendering technique. Also, the technique fits in well with our other contributions, especially the occlusion cache. Alternative non-caching techniques necessitate algorithmic complications to the rendering pipeline. Alternative caching techniques have not proven as useful or as widely-used.

2.4 Participating Media

Moving from surfaces to volumes breaks many of the assumptions we have had up to this point. We model participating media as a volume of microscopic particles which can either scatter, absorb, or emit light (figure 2.5). With the rendering equation, we modeled light traveling in a vacuum, projecting and reflecting on locally-flat surfaces. Light traveling through a participating media can be attenuated or scattered throughout the media’s volume. With these changes, the rendering equation (equation 2.4) becomes a differential equation [LW96] within participating media:
Here we see the four kinds of physical interactions we seek to model in a participating medium at virtual, microscopic particles. These physical interactions take place over a volume of space. The increased complexity of participating media volumes contrasts strongly with modeling only the reflection and emission of light at surface points.

Figure 2.5: Physical model of participating media
Although this equation looks remarkably similar to the rendering equation for surfaces, there are clearly some important changes. The function $\sigma(\vec{p})$ represents scattering within the medium. The function $\kappa(\vec{p})$ represents extinction in the medium, the combined effects of scattering and absorption. With the rendering equation, we had a BRDF $f_r(\vec{p}, \omega_i, \omega_o)$ and projection of radiance onto a surface, $\cos(\theta_i)$. Here we have a new function $\varrho_r(\vec{p}, \omega_i, \omega_o)$, the phase function of the medium. A phase function gives us the fraction of radiance from
direction $\omega_i$ scattered in direction $\omega_o$ at point $\vec{p}$. Also, the integral in equation 2.7 is over the sphere of directions $\omega_i$ around point $\vec{p}$, not just the hemisphere of directions.

Since equation 2.7 is a differential equation, giving us differential radiance $dL_o$ per some differential length $ds$. Integrating over the path of a viewing ray towards the virtual camera will give us the total reflected radiance that can be 'seen' from the camera’s location. There have been a number of approaches to solving this integral which we will detail in the following section.

In addition to discussing the background work on rendering of participating media, we will give a background to efforts in artistically designing the look of volumetric lighting and other components of rendering. We will also give some details about the tools and infrastructure with which we built and integrated the contributions of chapters 5 and 6.

### 2.4.1 Rendering Participating Media

Many of the solutions for rendering participating media involve an underlying simplification of the model. Simplifying the volume rendering model also simplifies the rendering algorithms. These simplifications often come at the cost of visual complexity, the inability to render some physical phenomena. In order to differentiate these simplifications in the subse-
sequent discussion of background work, figure 2.7 gives an overview of the different types and complexities of physical phenomena which can be modeled.

A simple model for participating media is that of single-scattering in a medium of uniform density first introduced to computer graphics by Blinn [Bli82]. Single-scattering means only one scattering event between light source and virtual camera is modeled (i.e. only the lighting directly illuminating a participating medium.) Max [Max86] first used this model with shadow volumes to specify integration domains for adaptive quadrature using a simplified lighting model. Nishita et al. [NMN87] increase the complexity of the lighting model and use ray marching to integrate illumination. Given intervals of constant light visibility and medium density, Pegoraro and Parker [PP09] introduced an analytic solution to the lighting integral.

For rendering heterogeneous density participating media, Perlin and Hoffert [PH89] introduced an oft-used, robust ray marching algorithm. Improving dramatically on this approach
very recently, Kulla [Kul11] presented work on "decoupled" ray marching, capable of efficiently rendering heterogeneous media.

To generalize to multiple scattering events, Pauly et al. [PKK00] applied metropolis light transport to participating media, a bidirectional path tracing method. Adjoint photon tracing [MBJ06] is a robust, Monte Carlo simulation approach which can model multiple scattering, as well as wavelength-dependent scattering, and polarization.

Photon mapping has been generalized to participating media [JC98] and utilized to render complicated lighting within volumes efficiently. Since photon mapping accelerates bidirectional path tracing, volumetric photon mapping is similarly robust in its ability to handle heterogeneous media, anisotropic scattering, and multiple scattering. Improving on the efficiency of path tracing and the quality of photon mapping, irradiance caching has also been generalized to be used in participating media [JDZ08]. The approach efficiently calculates lighting gradients and caches radiance in spherical harmonics for anisotropic scattering.

The photon beams algorithm [JNS11] is a more efficient version of volumetric photon mapping, created specifically for participating media. The method uses a much more compact representation of lighting within a medium as well as a more efficient method of using this pre-computed representation for final renders. As with volumetric photon mapping, the method can model multiple scattering, anisotropic scattering, and heterogeneous media.
Photon points: scattering locations are stored.

Photon beams: the path of the photon trace is stored in "photon beam" ray segments.

Figure 2.8: Volumetric photon mapping comparison

In contrast to volumetric photon mapping, photon beams are a representation of the entire path a photon takes in a medium instead of just scattering locations (figure 2.8). We can think of photon beams as small little spotlights within a medium. When enough photon beams are used, and the volume over which their lighting energy is spread is small enough, the aggregate result of these small little spotlights represents physically-based volumetric lighting in a medium. In addition to being efficient and flexible, modeling lighting power in this way offers a unique, intuitive opportunity for artistic intervention and control (chapter 6).

2.4.2 Designing Volumetric Lighting

Much of the initial research in computer graphics was directed at developing rendering algorithms to produce results indistinguishable from photographs. Photorealistic goals still
drive much of the research today. However, from the days of the first computer-generated animations, using computer graphics for artistic expression and design has been a growing area of investigation. Photorealistic renderings have been blended successfully with live-action to create believable special effects, but we have still only barely come close to computer generated movies that are indistinguishable from reality. On the other hand, there have been dozens of successful, stylized, and non-photorealistic computer generated movies.

With the first computer animated films, artistic needs had little bearing on the direction of the technology. In fact the technology probably had more influence on the design and art direction. For example, the choice of subject matter in the first computer animated feature film, *Toy Story*, was probably heavily influenced by the state of technology at the time. The stars of *Toy Story* were plastic toys, something computers were already very good at rendering. Lighting was also simple in the first animated movies. Most shots are sun-lit, with a single strong shadow-casting light source. Before these films became successful, the main question focusing technological development had been whether or not an animated feature-length film could even be made. This questions was decidedly answered and development moved to the tools and technology for greater artistic freedom and control.

Throughout the early successes of animated films, both computing power and computer graphics algorithms developed rapidly. The visual effects industry demanded increasingly detailed and realistic computer generated imagery. Today, computers create visual effects which are seamlessly blended with live-action. The animated film industry does not require
Clothing, hair, and other materials appear realistic. Characters and sets are more stylized.

Figure 2.9: Mixed realism in animated movies

photorealism, but does seem to value it in certain parts of an animated world. For instance, non-character materials are often incredibly life-like and lack stylization, but characters are more stylized and cartoonish (figure 2.9). There always has to be some grounding in reality, or it is difficult to relate with and visually understand the created world. Since much of computer graphics research is directed towards recreating reality, animated films have increasingly used this as a starting point for art-direction and stylization.

An increased reliance on a physical basis for animated films brings believable richness with little artistic intervention. However, utilizing physically-based animation and lighting paradigms comes with some drawbacks. Actual simulation of lighting or the physics of an animated world can be difficult to control for a desired, art directed result. There
is current, ongoing research into solutions for this problem. Selle et al. [SMC04] use physical simulation to render highly-stylized smoke. Angelidis et al. [ANS06] describe how to control the look of physical smoke simulations. Obert et al. [OKP08] present a method for artistic control of indirect lighting. Song et al. [STP09] allow users to edit physically-based, measured sub-surface scattering properties in a controllable way. Sadeghi et al. [SPJ10] build artist tools that allow for intuitive control over physically-based hair shading. Kerr et al. [KPD10] bend the physics of lights and shadows to facilitate the design of smooth and consistent lighting cinematography.

Our own contributions in chapter 6 follow this same pattern: taking something designed in the paradigm of physical-correctness, and moving it into a paradigm where physical plausibility is secondary to stylization. We implement the physically-based photon beams algorithm into a production renderer and then use photon beams as an intuitive primitive for controlling the appearance of lighting within participating media.

2.4.3 Reyes and the Production Pipeline

The first renderer to create a feature-length, computer-generated film was Photorealistic RenderMan™ (PRMan), based on the Reyes image rendering architecture [CCC87]. PRMan was built to render animations of complex geometry and complex shading. It uses a z-buffer algorithm for calculating visibility of shaded micropolygons and limits the use of ray tracing,
even for non-local lighting effects. PRMan has been further developed to include many different rendering algorithms and techniques, including ray tracing and global illumination, but the underlying architecture remains the same. The modern version of PRMan is an even more powerful, robust renderer for animated scenes of incredible complexity, including motion blur, depth of field, and superb image quality. We use PRMan’s efficient geometry and shading pipeline to implement the photon beams algorithm (chapter 5).

Besides the final rendering, there are many other content creation utilities which go into authoring an animated movie. Maya™ and Houdini™ are software applications which can be used for the entire production pipeline, but are often combined with other applications or augmented to fit the specific needs of production. Maya™ has a very strong modeling and animation core. Houdini™ uses an entirely procedural graph environment, a powerful and flexible paradigm for creating procedural geometry, animation, and particle effects. SeExpr is an expression language that can be used by non-programmers to write complicated rendering procedures. The needs of any single animated feature can be incredibly diverse and it’s common to create completely novel tools and techniques to meet specific needs.
CHAPTER 3
OCCLUSION CACHING

For computational efficiency and simplicity, the rendering equation [Kaj86] is often simplified to a summation over a discrete set of \( k \) scene lights:

\[
L_o(\vec{p}, \hat{v}) = L_e(\vec{p}, \hat{v}) + \sum_{i=1}^{k} f_r(\hat{l}_i, \hat{v}) L_i(\vec{p}, \hat{l}_i)(\hat{l}_i \cdot \hat{n})
\] (3.1)

\( L_o \) and \( L_e \) represent the outgoing and emitted radiance, respectively, as a function of surface point position \( \vec{p} \) and view direction \( \hat{v} \). \( f_r \) represents the BRDF as a function of the view direction and light direction. \( L_i \) represents the incoming radiance, and \( \hat{l}_i \cdot \hat{n} \) represents the surface projection attenuation.

For this simplified approximation, the incoming radiance \( (L_i) \) is usually the most computationally expensive part, because visibility between the light source and surface point must be computed. We cache this visibility function and share it with nearby shading points, sometimes reusing it and precluding the need for shadow rays.
Figure 3.1: Shadow rays per-pixel

(a) Sponza scene with 10 point lights

(b) False color image, colors represent the number of shadow rays cast at each pixel: No false color = 0, Green = 1, Yellow = 2, Light Orange = 3, Orange = 4, Red > 5. Shadow rays are cast when nearby occlusion cache points do not have matching occlusion information.
3.1 Cache Structure

For fast storage and searching, we store cache points in a uniform grid. An octree is fast enough for searching, but dynamically creating the tree as we create cache points becomes a bottleneck.

The grid dictates a constant density of cache information. This suits the variability of shadows from point lights, because point lights can cause high frequency shadows independent of occluder distance. Each grid cell stores four sets of distinct occlusion data to account for different occlusion information depending on surface orientation. To which set calculated visibility data is added depends on the surface normal of the point \( \mathbf{p} \) where the visibility data was calculated. This involves taking the dot product of the surface normal \( \mathbf{n}_p \) with four vectors that divide 3D space evenly \( (\mathbf{s}_0, \mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3) \). We cache data at the set \( c \) corresponding to the maximum of these four dot products (Equation 3.2), where \( i \) is a value from 0 to 3.

\[
\begin{align*}
\mathbf{s}_0 &= (1, 1, 1) & \mathbf{s}_1 &= (-1, -1, 1) \\
\mathbf{s}_2 &= (-1, 1, -1) & \mathbf{s}_3 &= (1, -1, -1)
\end{align*}
\]

\[
i = \text{IndexOf}(\max(\mathbf{n}_p \cdot \mathbf{s}_0, \mathbf{n}_p \cdot \mathbf{s}_1, \mathbf{n}_p \cdot \mathbf{s}_2, \mathbf{n}_p \cdot \mathbf{s}_3)) \tag{3.2}
\]
Visibility is calculated by casting shadow rays to each light and is represented as a string of single bits $v_p$, 1 for each visible light, 0 for each occluded light. Occlusion information is stored at grid cell $c$ as two strings of single bits: a string for visible lights $l_c$, and a string for occluded lights $o_c$. Initially $l_c$ and $o_c$ are set to 1 for every light. Occlusion data $v_p$ is added to grid cell $c$ with Equation 3.3 ($\&$ represents a bitwise AND, $\sim$ represents a bitwise NOT).

$$l_c = l_c \& v_p \quad o_c = o_c \& \sim (v_p) \quad (3.3)$$

Effectively, $l_c$ and $o_c$ represent the lights the grid cell agrees are visible or occluded based on calculated values within and nearby the grid cell. Looping through the summation of equation 3.1, lights for which $o_c$ is 1 can be skipped. Lights for which $l_c$ is 1 require no visibility test. Only lights for which $l_c$ is 0 require shadow rays. This approach allows us to cast many fewer shadow rays in umbra and lit areas of a rendering, while still rendering sharp shadow boundaries.
3.2 Filling the Cache

The occlusion cache is first seeded by rendering the scene without shading at a small fraction of the resolution of the final image. After the initial seeding step, data is cached in a “lazy” manner, calculating and caching occlusion information only when a cache miss occurs.

A cache miss occurs when not enough shading points have added visibility data to the grid cell in which the query point is located. A cache miss also occurs when visibility data from within the grid cell has not been added to the cache.

After a cache miss and visibility data $v_p$ has been calculated, it is added to every grid cell with center point within radius $R$ of $p$. We give an overview of an occlusion cache query in Algorithm 1.

3.3 Shadow Quality

Shadow quality depends mainly on the size of occlusion cache cells and the radius $R$ by which visibility tests are added to the cache. Intuitively, a very large radius $R$ will preclude any light-visibility agreement in cached values. A very small radius will result in too few cache hits, since cache cells will not have enough contributors.
Algorithm 1 Occlusion Cache Query ($\vec{p}, \vec{n}_p$)

Calculate grid cell $c$ from $\vec{p}$ and $\vec{n}_p$

- $l_c = \text{per-light bits for visibility}$
- $o_c = \text{per-light bits for occlusion}$
- $n_c = \text{number of points added to } c$
- $t_c = \text{number of points added to } c \text{ located within } c$

**if** $n_c > 3$ AND $t_c > 0$ **then**

- Calculate $v_p$, using $l_c$ and $o_c$
- Cast shadow ray to light $\lambda$ only when $l_{c\lambda} = 0$

**else**

- Calculate $v_p$, casting shadow rays for every light

**for all** Grid cells $\gamma$ within radius $R$ of $\vec{p}$ **do**

- $l_\gamma = l_\gamma \& v_p$
- $o_\gamma = o_\gamma \& \sim (v_p)$
- $n_\gamma = n_\gamma + 1$

**if** $\vec{p}$ is within $\gamma$ **then**

- $t_\gamma = t_\gamma + 1$

**end if**

**end for**

**end if**

Use $v_p$ with Equation 3.1 to calculate $L_o$
Table 3.1: Occlusion cache performance, varying cache radii

<table>
<thead>
<tr>
<th>Scene</th>
<th>Tris/Lights/O.C. Grid</th>
<th>Render Time (s)</th>
<th>Error (%)</th>
<th>Memory (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cornell</td>
<td>60k/1/64x63x64</td>
<td>0.02/0.02/0.02</td>
<td>0.1/0.01/0.001</td>
<td>2/1.4/1.3</td>
</tr>
<tr>
<td>Sponza</td>
<td>273k/66/256x108x158</td>
<td>0.10/0.13/0.16</td>
<td>0.001/0.0005/0.0003</td>
<td>18/18/18</td>
</tr>
<tr>
<td>Restaurant</td>
<td>1.5m/36/208x33x256</td>
<td>0.16/0.19/0.22</td>
<td>0.0050/0.0026/0.0015</td>
<td>7.1/7.2/7.2</td>
</tr>
<tr>
<td>Medusa</td>
<td>396k/16/256x197x109</td>
<td>0.17/0.19/0.21</td>
<td>0.03/0.01/0.009</td>
<td>23/23/23</td>
</tr>
<tr>
<td>Trees</td>
<td>346k/20/384x76x378</td>
<td>0.34/0.44/0.49</td>
<td>0.05/0.02/0.01</td>
<td>46/45/45</td>
</tr>
</tbody>
</table>

Change in render time, error, and memory as a function of cache radii. The renderings are made at 512x512, one sample/pixel (see figure 3.2). The radii are sized relative-to and expressed in terms of the grid cells. For example, a radius of 2 will extend across the width, height, and depth of 2 grid cells.

Figure 3.2: Scenes rendered for the benchmarks in table 3.1

The same intuition holds for the size of cache cells. Very large cache cells preclude storing accurate occlusion information. Very small cells result in fewer cache hits. The memory usage is also a concern with very small cells. Table 3.1 explores results of varying $R$ and the cache cell size. We note that the performance and memory usage of the cache is robust across a wide range of both variables.

Even with suitable cell sizes and $R$ value, our method does not guarantee perfect shadows. Possible shadow artifacts can be seen in Figure 3.3. Only tiny, thin shadows can be missed,
with the vast majority of shadows being identical to the ground truth; the artifacts of our method are inconspicuous and do not adversely affect the overall quality of the rendered shadows.

3.4 Results

Our rendering system is a recursive ray-tracer. Rays are traced in large packets (64-128 rays) for both primary and secondary rays. Benchmarks were run on a 3.2 GHz Intel Xeon quad core processor.

3.4.1 Performance

Figure 3.4 shows the performance change with respect to the number of lights in a complex scene, as compared to naively ray traced shadows for each point light at each pixel. With a single point light performance is comparable, but with more than one light the occlusion cache improves performance. Using occlusion caching, the performance curve drops off much more slowly as more point lights are added to the scene, as the cost of the cache look up is amortized over more lights. Also, the difference in rendering speed from one point light
The difference image shows little error, even with many small and thin shadows. The direct lighting in this scene rendered 3x faster with occlusion caching.

Figure 3.3: Occlusion cache artifacts
We compare occlusion caching to ray traced visibility tests by plotting frame rate as a function of the number of point lights in a scene. The occlusion cache shows a performance advantage with more than one point light, and a less-steep performance curve as the number of point lights increases. Benchmarks were performed with full texturing and bump mapping.

Figure 3.4: Occlusion cache performance scaling

to many point lights is not as stark. This makes the cost of shadows in a scene much more predictable for a wider range of lighting complexities.

To our knowledge, the state of the art in real time Whitted Ray Tracing with shadows from point lights involves tracing rays in large packets [ORM08]. Table 3.2 shows a comparison between tracing shadow rays in large packets and using the occlusion cache. We have included naive un-packeted shadow-ray benchmarks as a reference point for comparison. Table 3.2 also shows a reduction in the number of shadow rays beyond the screen-space method of Artzi et al. [BRA06].
Table 3.2: Occlusion cache one frame render times

<table>
<thead>
<tr>
<th>Scene</th>
<th>Naive</th>
<th>Packet</th>
<th>O.C.</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Lights)</td>
<td>Speedup (% S. Rays)</td>
<td>(OC Points)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cornell</td>
<td>0.51s</td>
<td>0.19s</td>
<td>0.18s</td>
<td>0.005%</td>
</tr>
<tr>
<td>(1)</td>
<td>1.06x (6.1%)</td>
<td>(4715)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sponza</td>
<td>28.85</td>
<td>3.24s</td>
<td>1.29</td>
<td>0.0004%</td>
</tr>
<tr>
<td>(66)</td>
<td>2.51x (0.09%)</td>
<td>(11321)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Restaurant</td>
<td>30.30</td>
<td>3.10s</td>
<td>1.91s</td>
<td>0.002%</td>
</tr>
<tr>
<td>(36)</td>
<td>1.62x (0.4%)</td>
<td>(11274)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Medusa</td>
<td>10.73s</td>
<td>2.47s</td>
<td>1.97s</td>
<td>0.01%</td>
</tr>
<tr>
<td>(16)</td>
<td>1.25x (0.75%)</td>
<td>(29363)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Trees</td>
<td>14.10s</td>
<td>5.63</td>
<td>4.42</td>
<td>0.02%</td>
</tr>
<tr>
<td>(20)</td>
<td>1.27x (1.9%)</td>
<td>(25580)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Test scene benchmarks at 1024x1024, four samples/pixel. We show the time to render shadows with and without occlusion caching, as well as the percentage of shadow rays (% S Rays) needed when rendering with the occlusion cache. The speedup is between tracing shadow rays in packets (Packet) and using the occlusion cache (O.C.). The naive, single-ray shadow test (Naive) is shown for reference. Error is in terms of occlusion cache false positives and false negatives as a percentage of total shadow rays.
3.4.2 Application

In previous chapters, we have repeatedly mentioned ray tracing’s simplicity. As proof of the robustness of our approach and the efficiency of occlusion caching, we adapted our rendering engine to a large, immersive display with special rendering constraints. This adaption was incredibly simple in the context of a ray tracing engine.

Abramyan et al. [APN12] developed an immersive display which requires a cylindrical projection with a 270 degree field-of-view. Rasterization cannot properly render cylindrical projections, because cylindrical projections turn straight lines into curved lines. Also, rendering 180 degrees or more with a rasterization pipeline’s perspective projection is not possible, and rendering much more than 90 degrees results in extreme distortion. One advantage of ray tracing is the ability to render using arbitrary camera models and projections.

In order to adapt our renderer to this immersive display we added a cylindrical projection that changes the way rays are generated for each pixel in the image. With this simple change, our efficient rendering engine created the correct rendering for this unique display. Occlusion caching facilitated interaction with the restaurant scene, even with many light sources and over four million pixels (figure 3.5).
To interact with the complex restaurant scene on this immersive display, we’ve combined occlusion caching with a simple cylindrical projection. Rendering such a wide field-of-view with the standard rasterization pipeline would necessitate a more complex solution. Ray tracing allows for arbitrary camera models and projections.

Figure 3.5: Occlusion caching on an immersive display
Irradiance caching is a well-proven algorithm invented over twenty years ago that has become ubiquitous in rendering engines [KG09]. It allows for rapid calculation of inter-diffuse reflections by amortizing the cost of these calculations over many pixels. A cache is used to store irradiance values sparsely across the scene. Cache values are used to compute irradiance in areas between cache points. The spacing of cache points is determined by both the distance to surfaces in the hemisphere above cache points and the local irradiance gradients.

We will present improvements to the irradiance caching algorithm, and finally show some results of our natural integration of irradiance caching with occlusion caching.

4.1 Anisotropic Cache Spacing

Irradiance cache points are often created too densely along the straight line where two planes meet. The gradient and indirect illumination along these lines are often constant as you move parallel to them (Figure 4.1). However, the gradient often changes rapidly as you move away from the edge, requiring close cache points to properly represent the rapidly changing function. Using cache points with a constant radius of influence precludes accounting for this
This crop of the Cornell Box shows Irradiance caching often produces too many cache records where two planes meet. Our anisotropic spacing shows similar density perpendicular to this edge, but many fewer records parallel to it. Cache records are represented in white. The checkerboard floor shows our method is still sensitive to illumination changes.

Figure 4.1: Anisotropic cache spacing
anisotropy in function change. Instead, we propose using cache points with appropriately anisotropic areas of influence.

Finding areas to employ this optimization is relatively straightforward. During neighbor clamping, we check the gradients of nearby points that will be interpolated with the point we are adding to the cache. If all of the nearby points have a translational gradient in the same direction as the point we are adding, we know that our function does not change very much in the direction perpendicular to the gradient angle. To find if nearby gradients have approximately equal direction, we find the minimum cosine $\zeta$ between our new cache point gradient and all nearby gradients. We modulate $\zeta$ by a user provided scale factor $\lambda > 1$, that controls the spacing of cache points perpendicular to the gradient.

$$
\frac{(\vec{V} \cdot \hat{T})^2}{R_i^2} + \frac{(\vec{V} \cdot \hat{N})^2}{R_i^2} + \frac{(\vec{V} \cdot \hat{B})^2}{(\zeta \lambda R_i)^2} < \alpha^2
$$

(4.1)

$$
\hat{T} = \frac{\nabla_i(E)}{\|\nabla_i(E)\|} \quad \hat{B} = \hat{T} \times \hat{N} \quad \vec{V} = P - P_i
$$

Calculating the error for interpolation amounts to projecting the vector $\vec{V}$ between the query point $P$ and the cache point $P_i$ into the coordinate frame created by the gradient vector $\hat{T}$, normal $\hat{N}$, and $\hat{B}$. The error calculation for interpolation from projected coordinates is demonstrated in equation (4.1). An error of less than $\alpha^2$ implies a cache hit. To speed this computation, we store the gradient vector scaled by the inverse square of $R_i$, and the vector
perpendicular to the gradient and the surface normal ($\hat{B}$) scaled by the inverse square of $\zeta\lambda R_i$.

4.2 A New Error Metric

The original error metric in irradiance caching was based on the lighting gradient of a bad-case, split-sphere model. It is by no means an upper bound for the gradient, nor does it result in perfect cache sample placement. Křivánek et al. [KBP06] noted that it did not detect small features well, such as a short staircase.

Form factor is a value defined between two points which characterizes the light transport between them. We calculate the form factor (4.2) for each sample when calculating the irradiance value of a new cache point. With the irradiance value and irradiance gradients, we also store the maximum form factor calculated during hemisphere sampling. This maximum form factor gives us a good approximation of how much a given point’s lighting can be influenced by nearby geometry. Like the harmonic mean in the original implementation of irradiance caching [WRC88], this value is used to control the spacing of sample points. We can see in equation (4.3) our calculation consists of a simple product of the maximum form factor with the squared distance between our stored point and query point summed with one minus the cosine between the normals of these points. If the error specified by this calculation ($\epsilon_p$) is below some pre-defined constant $\alpha$, we have a cache hit.
\[ F_{i\rightarrow j} = \frac{\cos(\theta_i)\cos(\theta_j)}{\pi r^2} \]  \hspace{1cm} (4.2)

\[ \epsilon_{p_i} = ((\vec{p} - \vec{p_i}) \cdot (\vec{p} - \vec{p_i}))F_{p_{\text{max}}} + (1 - \hat{n} \cdot \hat{n}_p) \]  \hspace{1cm} (4.3)

Similar to using the minimum distance in hemisphere integration, we have found that this value is more sensitive to small, nearby surfaces which have a predominant influence on the lighting at a given point. However, unlike the minimum distance, this value does not oversample on concave surfaces, and gives a more desirable sample spacing at corners. In addition to these benefits, this metric also speeds up the irradiance cache query. The form factor scales directly with the distance squared, which means that when we are searching for points in our irradiance cache, we can use the squared distance in our error calculation and save many expensive square root calculations for each query.

\subsection*{4.3 Cache Tunneling}

The dynamic memory allocation required by an irradiance cache usually precludes integration with the scene acceleration structure. Ray tracing scenes with hundreds of thousands to millions of polygons requires a compact acceleration structure, for both cache coherency and
Figure 4.2: Irradiance cache sample locations
to actually fit these complex scenes into memory. In most irradiance cache implementations cache points are stored within an octree, with nodes dynamically created when needed as the cache points are lazily computed. Consequently, we must traverse both acceleration structures to shade any pixel.

The cache octree is usually shallower than the scene acceleration structure, and finding a point in an octree is not nearly as complex as tracing a ray through an acceleration structure. However, both acceleration structures serve a similar purpose and occupy the same space. Such redundancy is inefficient. Instead, our solution allows work done in one acceleration structure to carry over to the other.

When adding a new cache point to our octree, we find the smallest node in our tree which completely encloses the leaf node of our acceleration structure. We then store a pointer to this node back in the leaf of our acceleration structure. When finding cached irradiance values, this pointer allows us to start our traversal much deeper in the octree, accelerating our query.

Our implementation makes use of a Bounding Interval Hierarchy (BIH) [WK06], a type of BSP-tree. A BIH is similar to a kd-tree, but stores 2 separate split planes. This allows the building of the structure to be very simple and allows a bound on memory usage as a function of the number of scene primitives. However, since a BIH stores one more split plane than a kd-tree, the internal nodes themselves, are at the least 4 bytes larger than a kd-tree’s inner-nodes. The leaf nodes of a BIH do not store split planes, and only half of this space
is used by a reference to scene geometry. Conveniently, we utilize the remaining 4 bytes to store a pointer to our octree node.

4.4 Results

We utilize the occlusion cache to accelerate the single bounce global illumination calculation of our irradiance cache. Because single-bound global illumination has to calculate direct lighting at each scene intersection point during hemisphere integration, our occlusion cache is used much more, and the relative reduction in shadow rays is increased.

Table 4.1 shows the utility of a world-space caching approach for static lighting and geometry walk-throughs. The occlusion cache shows a much more pronounced speed increase over large-packet ray tracing when cache values can be re-used for subsequent frames. The improvement of using occlusion caching in concert with irradiance caching is even more pronounced. Also, the occlusion cache provides more interactive performance both with and without irradiance caching.
Table 4.1: Combining occlusion caching with irradiance caching

<table>
<thead>
<tr>
<th></th>
<th>Sponza</th>
<th>Restaurant</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Direct Lighting</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Packet</td>
<td>66.5s</td>
<td>35.7s</td>
</tr>
<tr>
<td></td>
<td>1.5fps</td>
<td>2.8fps</td>
</tr>
<tr>
<td>O.C.</td>
<td>18.1s</td>
<td>13.7s</td>
</tr>
<tr>
<td></td>
<td>5.5fps</td>
<td>7.3fps</td>
</tr>
<tr>
<td>Speedup</td>
<td>3.7x</td>
<td>2.6x</td>
</tr>
<tr>
<td><strong>Global Illumination</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Packet</td>
<td>185.2s</td>
<td>71.4s</td>
</tr>
<tr>
<td></td>
<td>0.54fps</td>
<td>1.4fps</td>
</tr>
<tr>
<td>O.C.</td>
<td>21.7s</td>
<td>16.4s</td>
</tr>
<tr>
<td></td>
<td>4.6fps</td>
<td>6.1fps</td>
</tr>
<tr>
<td>Speedup</td>
<td>8.5x</td>
<td>4.3x</td>
</tr>
</tbody>
</table>

We show the speedup of a 100 frame walk-through at 512x512, one sample/pixel. The comparison is a packeted ray tracer (Packet) versus using the occlusion cache (O.C.). The global illumination algorithm used in both cases is a single-bounce irradiance cache (100 hemisphere samples/record).
CHAPTER 5
FAST VOLUME PRIMITIVES

We implemented the photon beams algorithm in PhotoRealistic RenderMan (PRMan) to efficiently render artistically-directed volumetric lighting effects for the feature-length animated movie Tangled [JLS11]. Photon beams generalize volumetric photon mapping by storing the entire path of a photon instead of just the scattering location. Conceptually, each photon beam represents a truncated conical beam of light through the medium. Jarosz et. al formulated several ways to compute a radiance estimate from photon beams, promoting the so-called Beam × Beam 1D estimate which interprets each beam as a flat axial billboard.

The original implementation of the photon beams method was in the context of a ray tracer [JNS11]. Similar to the original photon mapping algorithm, photon beams are generated from light sources, intersected with scene geometry and then stored in an acceleration structure. Camera viewing rays are then ray traced through the scene and intersected with photon beams. These intersected photon beams are then used to calculate a lighting integral for each camera viewing ray.

More recently photon beams have been rendered on the GPU using an approach similar to our implementation in PRMan [JNT11]. Pre-simulated beams are converted into impostor geometry and visibility tested. Then the lighting integral is calculated in the fragment shader and additively blended per-pixel for each visible beam fragment. The algorithm is very efficient on the GPU, with the ability to render tens of thousands of beams in realtime.
The similarity of the GPU pipeline to the Reyes algorithm led us to our PRMan photon beam implementation: a similar splatting approach using RenderMan curve primitives (RiCurves) as impostor geometry to compute the beam radiance estimate [JZJ08a].

In the original formulation, the radiance along a beam has a physically-based exponential falloff based on the scattering properties of the medium. Fortunately, the billboard representation easily handles arbitrary, artist-controlled fall-off functions along beams, making it an ideal candidate for artistic volumetric effects. Though the billboard representation gives the correct approach in the limit, it can sometimes produce artifacts. When the beams are aimed at the camera a flat billboard is a poor approximation for the conical photon beam. To avoid this problem, we use the Beam × Beam 2D estimate which treats each beams as a conical frustum with a finite cross-section. In this case, to compute the beam’s contribution, we need to consider the integral along each camera ray through a beam.

### 5.1 Photon Beam Splatting with RiCurves

PRMan uses rasterization to compute the visibility of shaded geometry in an image. Specifically, shaded micropolygons are projected into the space of the screen raster and sampled for visibility. To render photon beams properly in PRMan we must control both the way our beam geometry is shaded and how visible beams are blended to compute a final pixel color.
Beam data is either precomputed outside of PRMan or created from shadow maps, as in figure 5.1. We convert each beam into a linear RiCurve with two control points: a start and an end. The width of the beam in world space is also specified at both points. RenderMan’s RiCurves are flat curves that always face the camera. A linear RiCurve with two control points is the same geometry as a billboard.

These photon beams are diced into micropolygons along with the rest of the scene geometry and shaded. For each shading point, PRMan gives us two-dimensional texture coordinates which give us the percentage distance from the beam source and percentage distance from one edge of the RiCurve along its length. With this much information we can directly calculate a one-dimensional blur.

Shaded geometry is rasterized, sorted, and blended from front to back. The photon beams are assigned an opacity of zero when shaded, which PRMan interprets as additive blending. This means that every shaded beam—with energy spread over one dimension–overlapping a given pixel sample will be added to the final color of that sample. This is identical to a beam query on beam data with a one-dimensional blur [JNS11], equation 5.1.

\[
L_m(x_c \leftarrow \omega_c) \approx \sigma_s \sum_{b \in R_b} \frac{\Phi_b f(\theta_b) e^{-\sigma_t v} e^{-\sigma_t z}}{\mu_R(u) \sin(\theta_b)}
\]  

(5.1)

The summation in equation 5.1 is represented by splatting photon beams to the pixel samples they overlap and additively blending them. Therefore the inner term of this summation must
(a) Different resolution shadow maps

(b) We choose a shadow map in (a) and for each pixel we create a photon beam, with length equal to the depth value.

Figure 5.1: Photon beams from shadow maps
be calculated at shading points along each beam. We facilitate this by passing in the photon beam length as a parameter to each \texttt{RiCurve}. \( \sigma_t \) and \( \sigma_s \) are designated by shader arguments. The PRMan surface shading environment gives us all of the other values we need: distance from the camera to the shading point, percentage distance from the beam origin, percentage distance along the width of the beam, the width of the beam, and direction of the beam \( (\frac{\delta P}{\delta v}) \).

Combining shading environment variables with attributes of each \texttt{RiCurve}, we can calculate the distance from the beam origin \( (v) \), the distance from the center of the beam \( (u) \), the distance from the camera \( (z) \), and the angle the beam makes with the camera ray \( (\theta_b) \).

### 5.1.1 3D Beams

\texttt{RiCurves} are an efficient and convenient part of PRMan and represent photon beams very well in the vast majority of cases. However, when the camera is positioned in a way to peer directly down an \texttt{RiCurve} their two-dimensional geometry becomes conspicuously apparent.

If you rotated a linear \texttt{RiCurve} with two control points 180 degrees about its center, it would sweep out the volume of a cone (or a cone with the apex cut off in the case of a non-zero radius at both control points). We use this model of \texttt{RiCurves} to identify curves which will probably appear flat when rendered. That is, we model the curves as a cone section and find whether or not the camera is contained within this volume. In such a case, we replace the \texttt{RiCurve} geometry with a screen-space quad which covers the entire screen.
For each shading point on the quad we do a ray/cone intersection given the viewing direction and photon beam shader arguments of the quad, finding the ray interval enclosed in the beam volume. Since the camera is effectively inside the photon beam, the light from the beam will be added to every screen pixel. This means that no shading time is wasted by the quad covering the whole screen, as every ray will travel through the cone geometry.

After calculating the interval over which the viewing ray is enclosed in the photon beam, we numerically integrate the lighting contribution from the photon beam (described in more detail in section 5.1.3). We can perform this integral without visibility testing because we know that we at least have visibility along the center of the beam between the beam source and the projected distance of the camera. We assume the beams are thin enough that the likelihood of occluders within the beam volume significantly attenuating the lighting contribution is small. This assumption is true in most instances. If artifacts are present, beams can be sub-divided into smaller beams until artifacts are no longer visible.

We make the replacement of *RiCurves* seamless by accounting for the difference in integrating through a volume and the lighting calculation of equation 5.1. We model the ray intersection with the cone section as a ray/cylinder intersection. If the *RiCurves* were all cylinders, the rays going through them would be attenuated approximately according to the function $\sqrt{(1 - r)^2}$. Where r is the percentage distance from the center of the curve to the edge. This represents the distance a ray travels through a cylinder if it is perpendicular to both the cylinder direction and parallel to a vector tangent to the cylinder.
Figure 5.2: Impostor geometry comparison

We take this approximation of the ray/cone intersection and divide by $\sqrt{(1 - r)^2}$ to match the lighting to the RiCurves in the scene. The approximation works extremely well. In animated scenes photon beams blend seamlessly back and forth between RiCurves and path-integrated volume rendering.

Figure 5.2 compares RiCurves with three-dimensional photon beams. The renderings appear identical, save the one beam running directly down the center of the camera view frustum. In the RiCurve rendering, the photon beam appears flat. Rendering the beam with full-screen quad impostor geometry gives the beam volume and accurately renders the lighting of the camera enclosed within a photon beam.
5.1.2 Cone Impostors and Motion Blur

It is worth noting that we initially used cones as impostor geometry to do ray intersections at shading points. The renderings were identical to those using full-screen quads, except in presence of occluders. If a photon beam is big enough to contain an occluder, some of the pixels on the screen will never be drawn, and there will be a visible discontinuity where the cone geometry intersects the occluder.

The full-screen quad does the ray-trace at every pixel, so in the presence of an occluder, it will instead overestimate the lighting integral at the same pixels the cone would not be visible. We decided the over-brightening was preferable to the discontinuity. However, both artifacts can be eliminated with small enough beams.

With this in mind, there are still a few advantages to using cones. The shading interpolation seems to perform better with fewer shading samples. Although we did not investigate this enough to know the extent of the performance advantage. Over a whole frame, there are so few photon beams with the camera in them, there isn’t much performance impact to be had. For our purposes, increasing the shading rate for full-screen quads above that of the RiCurves worked well enough without a noticeable performance impact.

Also, cones have the benefit of not being defined in screen space, so they can be properly motion blurred with camera motion. This advantage might actually be a good reason for
using cones instead of a screen-space quad. However, we didn’t run into any cases where this was necessary. In fact, we ran into a few cases where the motion blur caused artifacts.

For both cones and RiCurves, beams that the camera would move through within a frame can be improperly motion blurred, resulting in visual popping artifacts shown in Figure 5.3. PRMan only shades at a single time per frame, but does visibility testing at a range of time in a given frame. A photon beam is very bright when pointed at the camera, but then the brightness quickly diminishes as it points away. In short, the artifact occurs because shading changes rapidly over the frame but PRMan only shades once per frame. To solve this with RiCurves we can just average the inverse sine term at the start and end frame time and this will remove the artifacts.

This RiCurve fix is unnecessary when using screen-space quads. As they replace the RiCurves which are improperly motion-blurred, and the lack of motion blur for what we are rendering does not seem to be a problem. If using cones to give photon beams volume, we would need to develop a way to solve these motion blur artifacts. Just because we did not run into a case where motion blur was necessary for the few photon beams containing the camera does not mean the case for it does not exist.
Figure 5.3: Photon beam motion blur artifacts with PRMan
5.1.3 Non-Physical Light Fall-off

One important change to photon beams is allowing more control over the fall-off behavior of photon beams. We allow artists to designate any fall-off or color change of the beam along its length. This is a simple enough proposition when the beams are RiCurves, but when the beams are rendered with volume and integration is required, arbitrary fall-off functions can be difficult to accommodate.

We use Gaussian quadrature to integrate the light contribution through photon beams. This quadrature rule converges much quicker than quadrature rules with uniformly spaced samples. In practice, we found just a few function evaluations are necessary for good results. With complicated color changes across a beam, many more samples might be necessary.

5.2 Gaussian Quadrature

Although there are analytic solutions to this path integral for physically-based single-scattering [PP09], we need to allow for arbitrary, non-physical fall-off functions for artistic control. With the knowledge that most fall-off functions defined by our artists would be polynomial-smooth, we use Gaussian quadrature [SS66] to accurately and efficiently estimate the lighting contribution of these camera-containing beams. Our numerical approach allows for robust artistic
control over beam appearance, while reducing the number of lighting samples compared to other numerical approaches with no loss of accuracy.

Like the more commonly-used Riemann sum, Gaussian quadrature is a method for approximating a definite integral. It has greater accuracy for the same number of sample points as a Riemann sum because sample locations are weighted and placed in specific locations to solve for an exact result with \( n \)-samples for polynomials of degree \( 2n - 1 \). We can use Gaussian quadrature successfully whenever we are integrating smooth functions and the domain of integration is known.

### 5.2.1 Definition

For a prescribed number of sample points, Gaussian quadrature computes an integral in the canonical domain \([-1, 1]\) using a precomputed set of positions \( x_i \) and corresponding weights \( w_i \):

\[
\int_{-1}^{1} f(x) \, dx \approx \sum_{i=1}^{n} w_i f(x_i).
\] (5.2)

It is common practice to use a precomputed table of values for this domain \([-1, 1]\) and then transform sample points and scale weights accordingly. A table for the 1-point through 5-point Gaussian quadrature rules is shown in figure 5.1.
Table 5.1: n-point Gaussian quadrature sample locations and weights

<table>
<thead>
<tr>
<th>n</th>
<th>((x_i, w_i))</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>((0, 2))</td>
</tr>
<tr>
<td>2</td>
<td>(\left(-\frac{1}{\sqrt{3}}, 1\right), \left(\frac{1}{\sqrt{3}}, 1\right))</td>
</tr>
<tr>
<td>3</td>
<td>(\left(-\sqrt{\frac{3}{5}}, \frac{5}{9}\right), \left(0, \frac{8}{9}\right), \left(\sqrt{\frac{3}{5}}, \frac{5}{9}\right))</td>
</tr>
<tr>
<td>4</td>
<td>(\left(-\sqrt{\frac{3+2\sqrt{3}/7}{36}}, \frac{18-\sqrt{30}}{36}\right), \left(-\sqrt{\frac{3-2\sqrt{3}/7}{36}}, \frac{18+\sqrt{30}}{36}\right))</td>
</tr>
<tr>
<td>5</td>
<td>(\left(-\frac{1}{3}\sqrt{\frac{5+2\sqrt{10/7}}{900}}, \frac{322+13\sqrt{70}}{900}\right), \left(-\frac{1}{3}\sqrt{\frac{5-2\sqrt{10/7}}{900}}, \frac{322-13\sqrt{70}}{900}\right), \left(0, \frac{128}{225}\right))</td>
</tr>
</tbody>
</table>

Sample positions \(x_i\) and weights \(w_i\) are specified on the domain of integration: \([-1,1]\).
An integral over an arbitrary interval \([a, b]\) is computed using a simple transformation of the samples points and weights:

\[
\int_a^b f(x) \, dx \approx \frac{b - a}{2} \sum_{i=1}^{n} w_i f \left( \frac{b - a}{2} x_i + \frac{a + b}{2} \right). \tag{5.3}
\]

### 5.2.2 Intuition

It might not be immediately clear why simply moving the sample points and using specific, variable weights yields such an increase in accuracy. We find it illustrative to visualize the Gaussian quadrature integral evaluation.

Figure 5.4 shows different linear functions which all have the same integral. Without coincidence, all functions share the same value at their domain midpoint, precisely where the 1-point Gaussian quadrature rule specifies a function evaluation. Consequently, Gaussian quadrature allows us to integrate any linear function on a closed interval with a single function evaluation.

Continuing to higher order polynomials, figure 5.5 shows different third-degree polynomials. As with the case in Figure 5.4, each integral is equal and each function shares the same value at Gaussian quadrature rule sample points. Here the sample points are designated by
the two-point Quadrature rule. We can integrate any third-degree polynomial with only 2 function evaluations.

Putting all of this together, we can visualize Gaussian quadrature as making a best fit degree $2n - 1$ polynomial for an $n$-point quadrature rule. If the function is a polynomial of degree $2n - 1$ or fewer, a perfect “fit” exists and Gaussian quadrature allows for an exact solution. If the function is non-polynomial, but still smooth and could be approximated well with a polynomial, then Gaussian quadrature should give a visibly accurate result.
5.3 Discussion

As opposed to GPUs, the PRMan pipeline is very robust in how it renders visible surfaces. It supports depth of field, motion blur, and properly renders any number of transparent surfaces. However, additive blending of photon beams does not require any of these features, but PRMan does not currently allow us to disable them. Before a final sub-pixel color is calculated, every pixel sample will still store a sorted list of all visible photon beams at that sample. With many pixel samples, and many overlapping photon beams within a sub-pixel, memory usage can quickly get out of hand. The cost of sorting is also a concern.

The sorting time and inefficiency in memory usage seems to change the algorithmic complexity of the photon beams algorithm from what we would ideally expect. In figure 5.6 we show a comparison between a PRMan implementation of photon beams and a GPU implementation of photon beams. Both implementations render the same data sets. The rendering performance is plotted as a function of the number of photon beams rendered. The algorithmic complexity of the GPU implementation appears to be sub-linear. In contrast, the PRMan implementation is linear.

Still, even with this less-than-ideal algorithmic performance, our implementation of photon beams in PRMan is relatively efficient. In a more physically-based, simulation workflow we might expect to need tens of thousands of beams to render complicated volumetric lighting. With more artistically-driven scenes, we have instead gotten away with only thousands of
beams to good result. This number of beams has been easily handled by PRMan, with memory usage and performance not being markedly impacted.

Ultimately, our comparison with GPU performance shows a lot of room for performance improvement. A demanding enough photon beam dataset might necessitate creating an alternative solution. However, for our purposes thus far, our PRMan implementation has performed well, and has integrated seamlessly and conveniently with our rendering pipeline. The advantage of quick implementation and full-integration have allowed us to add photon beam effects to a feature animated film with a very short development cycle.
5.4 Results

The keyhole shot from the film Tangled (Figure 5.7) used photon beams in PRMan to represent light shining through a keyhole in participating media. Our artists used a one-dimensional color texture to design a color change as the beams extinguish in the media. This color change was multiplied by physically-based attenuation. As the camera passes through photon beams, our Gaussian quadrature estimate on full-screen quads blends seamlessly with the RiCurve estimate, accurately rendering the camera inside photon beams.

All of this is done within the PRMan pipeline, maintaining proper motion-blur, depth-of-field effects, and rendering correctly in stereoscopic 3D.
In addition to being a compact and efficient storage paradigm for light power in a participating media, photon beams also represent an intuitive way to visualize light propagation in a media. This is especially true when single-scattering accounts for most observable light within a medium. It is this natural parallel between photon beams and how people think about light within a volume that makes them particularly useful for artistic tweaking.

After we integrated photon beams into PRMan and other software tools of the Walt Disney Animation Studios, artists pushed for further development in increasing their ability to bend or even break the physical nature of photon beams. We had already given artists control over the fall-off behavior of photon beams, as well as how photon beam energy is spread across their width (as detailed in chapter 5). In addition to these non-physical changes, artists wanted to break the linearity of photon beams, and have more control over the shape of the lighting within volumes.

We will now detail the tools we created to allow photon beams to be used in such appealing, inventive ways.
6.1 Our Contribution

In cooperation with Disney Research Zurich and the Walt Disney Animation Studio, we designed tools to adapt the physically-based photon beams algorithm for non-physical art-direction. A Programmable System for Artistic Volumetric Lighting [NJS11] covers much of this work. Our contributions include: the core implementation of photon beams as described in chapter 5, a detailed exploration of non-physical photon beam shading, development of tools for converting artist’s designs into photon beams, and the invention and implementation of non-linear photon beam primitives. Nowrouzezahrai et al. refine our work by allowing photon beams to act as light sources for surfaces, develop a tool for determining physical scattering and absorption parameters from an artist’s coloring of a medium, and report informal feedback from artists on the usefulness of our tools.

6.2 Non-Physical Beam Shading

Equation 5.1 specifies the physically-based shading we use in our implementation of photon beams. This equation is a function of several variables. Some of these variables are properties of the medium in which photon beams are created: the scattering coefficient $\sigma_s$, the absorption coefficient $\sigma_a$, and the extinction coefficient $\sigma_t$ (the sum of the absorption and scattering coefficients, $\sigma_t = \sigma_s + \sigma_a$). The remaining variables deal with the geometry of
photon beams: the distance from the photon beam and the camera \((z)\), the angle between the photon beam and the camera ray \((\theta_b)\), and the location on the photon beam from it’s origin \((v)\) and from it’s centerline \((u)\).

Designing the look of volumetric lighting through physical parameters like the absorption and scattering coefficients can be unintuitive. For example, our atmosphere scatters blue and violet light much more than red or orange light. When the sun is overhead, the sky’s appearance is dominated by in-scattered blue light. As the sun sets, light must travel much farther through the atmosphere to be observed, in-scattering more red and orange light and out-scattering more blue light. This simple physical parameter has a dramatic effect on atmospheric appearance, dependent on other physical parameters. It seems difficult to develop an intuition for predicting visual changes resulting from control of physical scattering and absorption parameters.

Our implementation of photon beams as geometric primitives gives us an intuitive paradigm for designing volumetric lighting. If the lighting in a medium is represented by a set of discrete photon beams, it is much easier to think about the lighting as the aggregate effect of modifying the shading of individual photon beams. For this reason, we propose refactoring our physical shading of photon beams (equation 5.1) into a set of functions that can each be modified independently for intuitive control of non-physical shading. As there are four geometric parameters for beam shading, we reformulate physical beam shading as the product of four functions, one for each geometric parameter:
• attenuation along the beam $f_b(v)$,
• shading dependent on a beam’s width $f_t(u)$,
• attenuation towards the camera $f_e(z)$, and
• shading dependent on the camera viewing angle $f_f(\theta_b)$.

Equation 5.1 is reformulated to equation 6.1:

$$L_m(v, u, z, \theta_b) = \sum_b \left( e^{-\sigma_t v} \left( \frac{\Phi_b}{\mu_R(u)} \right) (e^{-\sigma_t z}) \left( \sigma_s f(\theta_b) \right) \right)$$

$$f_b(v) = e^{-\sigma_t v}, \quad f_t(u) = \frac{\Phi_b}{\mu_R(u)}, \quad f_e(z) = e^{-\sigma_t z}, \quad f_f(\theta_b) = \sigma_s \frac{f(\theta_b)}{\sin \theta_b}$$

$$L_m(v, u, z, \theta_b) = \sum_b f_b(v) f_t(u) f_e(z) f_f(\theta_b). \quad (6.1)$$

This reformulation leaves equation 5.1 unchanged if the physical definitions for the four component functions are used. However, this new model allows us to replace any or all of these four functions with something non-physical that can be controlled in an intuitive way. Figure 6.1 shows the effects of modifying each of the four functions individually. The resulting changes are predictable while allowing for diverse control over the appearance of photon beams. The physically-accurate shading functions can be replaced with basically any function imaginable. However, the space of functions which make sense in this context seems relatively constrained. We expect things like texture maps, spline function manipulations, or even rather simplistic linear or quadratic functions.
(a) Physically-based shading of three photon beams

(b) $f_b(v)$ is a noise function

(c) $f_t(u)$ is held constant

(d) $f_e(z)$ is held constant

(e) $f_f(\theta_b) = \frac{1}{\cos(\theta_b)}$ for the red channel

We demonstrate the effects of isolated changes to strictly physical shading for individual photon beams using our four-function parameterization. Each image shows three beams, with each successive beam aimed more directly at the virtual camera.

Figure 6.1: Photon beam shading components
We explore the aggregate effect of photon beam shading tweaks on a much denser set of photon beams in figure 6.2, reproducing the bumpy sphere scene from Walter et al. [WZH09]. The bumpy sphere refracts and scatters illumination. Observable light within the sphere is refracted at least once more, resulting in an amber-like appearance. We explored non-physical shading of this physically-simulated photon beam data using our four-function model, as well as modifying the physical scattering and absorption parameters. The exploration yielded some very interesting and diverse results, from color shifted scattering to a more cloudy scattering effect.

### 6.3 Procedural Photon Beam Generation

Photon beams detail how light power is distributed in a volume. These simulated light paths in a participating medium define the lit shape of a volume. In considering and rendering photon beams as geometric primitives, we have taken them beyond a volume-rendering acceleration algorithm to something innately understandable. We can look at photon beams directly and reduce a complicated medium of varying particle densities, sizes, and physical scattering properties to the light that can actually be seen within a volume. This led us to a powerful design paradigm for volumetric lighting: Instead of manipulating the physical description of a participating medium and then simulating how light interacts with this medium, we allow the visible light to be directly manipulated.
We replicate the bumpy sphere scene from Walter et al. with photon beams. We then explore various non-physical shading tweaks within the constraints of our parameterization, allowing us to design interesting, non-physical volumetric lighting.

Figure 6.2: Non-physical shading exploration
Directly manipulating and procedurally generating photon beams has a number of advantages for designing scattered light in a volume. There are many pre-existing geometric design tools for computer animation. Authoring volumes with geometry utilizing these tools can be much more intuitive than designing a volume for lighting simulation without geometric primitives. Beyond these well-developed tools for geometric design, directly designing exactly what is seen within a volume can be much more intuitive than trying to tease a desired appearance out of a physical simulation of some complex volume description.

In figure 6.3, we illustrate one of the advantages of being able to directly design the visible light within a volume. The problem solved by this example is very common in computer generated films. The three-dimensional world of these films is usually not modeled in a complete way, nor is it detailed enough to facilitate precise physical simulation. In the keyhole shot, even if the geometry of the keyhole and cracks around the doors were modeled
completely and correctly, actual simulation of light shining within the room and spilling out would be computationally demanding. Instead, we allow artists to directly "sculpt" the desired distribution of volumetric lighting by procedurally generating particles within the cracks and keyhole. The particles have attributes such as beam length, direction, and power. These attributes and particle locations can easily be perturbed, jittered, or even animated. Procedural photon beam generation allows for very precise art-direction and rapid authoring of intricate volumetric lighting.

6.4 Non-Linear Photon Beams

Geometric optics simulation leads to conical light frusta which we represent with linear photon beams in the form of RiCurves. Creating beams from other physical simulations like fluid-based simulations, or even more general procedural generations can lead to non-linear structures. We give artists the freedom to create photon beams from these structures and control the shading in the same way that they do with linear photon beams. This breaking of photon beam linearity gives artists the freedom to create artistic lighting effects that are interesting and easy to imagine but not possible in a physically-constrained environment.

Implementation-wise, instead of restricting beams to linear RiCurves with only two control points, we simply allow for the addition of more control points. An example is shown in figure 6.4. However, there are some issues when converting procedurally generated data to
non-linear RiCurves. Unfortunately, RiCurves do not render well when there is a combination of high curvature, thickness, and the curves are directed towards the camera. These things together will produce conspicuous artifacts where the curves cannot remain continuous in texture space while still facing the camera. We minimize these artifacts by limiting the curvature and width of RiCurves and by not pointing the camera directly down them.

We did not integrate non-linear RiCurves with ”3D Beams.” It’s unclear how these could be integrated together seamlessly, since non-linear beams break many of the assumptions we make for selecting photon beams which will appear flat and need replacing. Also, ray tracing non-linear photon beams would be non-trivial.
6.5 Results

Our tools were used to render a sequence of shots for the animated film *Tangled* (figure 6.3 and figure 6.5). The look of the resulting volumes matched closely with the artistic style of the film and was successful in realizing what the directors had envisioned.

The keyhole shot from the movie was done twice (figure 6.3). The first time an ad hoc method was used for producing the shot for a theatrical teaser trailer. For the final shot, we used procedurally generated photon beams. Our tools allowed artists to create volumetric effects more rapidly and with better result than previous methods.

The revive sequence of shots (figure 6.5) shows a mix of both linear and non-linear photon beams. We leveraged tools artists were already familiar with for authoring particle effects to designate the shape and position of photon beams. We then procedurally generated photon beams from this particle data. Shading of the beams was controlled through our four-function model. Because of the physically-based origins of both the shading and geometry of non-linear beams, they were plausible and blended well with linear photon beams in the sequence. Our intuitive tools for control of volumetric lighting allowed for art direction of a unique, dramatic effect.
Figure 6.5: Non-linear photon beams in *Tangled*
CHAPTER 7
SUMMARY AND FUTURE WORK

Throughout the previous chapters we have described both the strengths and weaknesses of our methods. What follows is some possible future work to augment our contributions and address some of the shortcomings.

7.1 Caching Algorithms

We want to emphasize that improving the efficiency of both our occlusion and irradiance cache point computations through code optimization would probably have the most marked effect on our frame rate. Although there are some possible algorithmic improvements which may also lead to substantial performance improvement.

7.1.1 SIMD Optimizations

In spite of very little low-level code optimization, our caching approaches compare favorably with more mature, highly-optimized ray tracing implementations. Some interesting future work could start with parallelizing our cached shadow and shading computation with
SIMD (single instruction, multiple data) optimizations. This implementation would be quite straightforward, since there is little chance for nearby points to branch off into differing instructions when there is a cache hit. With these changes and other incremental improvements to cache look-up efficiency, we would expect the performance advantage of occlusion caching to increase relative to more mature approaches.

7.1.2 Dynamic Lighting

Our approach has the benefit of allowing for static walkthroughs of complex lighting with global illumination at interactive rates. Future work might focus on increasing performance in presence of dynamic light sources. Since we are only caching light visibility information, changes in light intensity or color are already handled with our system. For example, a flickering light source like a candle could be modeled as a point light source in most cases. Changing the intensity of this light source over time as the flame fluctuates would be handled well with occlusion caching.

For changing visibility, currently we just cycle through our cache points and update visibility information for any lights that have moved between time steps. One improvement to this approach might involve tracing rays along a moving light source’s path, sampling visibility along this path and caching visibility information in time, adding another dimension to the
spatial cache. This might allow us to more efficiently render shadows from moving lights or even the dancing shadows caused by occluders nearby a campfire or television.

For the irradiance cache side of dynamic lighting, implementing temporal gradients [KBP06] into our system should allow us to have interactive global illumination in dynamic scenes. Our direct lighting cache can already accommodate dynamic scenes with a modest performance hit. For single-bounce global illumination, this kind of system could certainly be more dynamic than one using radiosity or photon mapping to accelerate global illumination computation, as we do not require pre-computation.

\section*{7.1.3 Large Area Lights}

In future work, we would also like to explore actual lighting interpolation for large area lights in addition to the visibility interpolation we already do. We experimented with putting direct lighting in a standard irradiance cache and it failed quite spectacularly in most instances. As we have discussed in previous chapters, the direct lighting function is high frequency. We can’t calculate lighting gradients accurately as is done with irradiance caching because the visibility function of light sources is discontinuous. That is, there is no way to know whether visibility of light sources at nearby points is in any way correlated to the lighting at a cache point. This is the underlying problem of spatial interpolation of direct lighting and the problem we have attempted to solve by creating occlusion caching in the first place.
However, large area light sources do not have discontinuous lighting gradients. Often, the shadows from large area lights are no more high-frequency than indirect lighting. As is the case with irradiance caching, we should be able to calculate a lighting gradient at cache points and use it to interpolate the combined lighting and visibility function from large area lights. We could use the occlusion cache to decide whether or not area lights are large enough to take advantage of this optimization. We benefit from the fact that irradiance caching already performs well when there are nearby occluders at a cache point. Even large area lights create sharp shadow boundaries when occluders are nearby shading points. Irradiance caching correctly increases cache point density under these conditions.

7.1.4 Higher Order Lighting Derivatives

Cache lookups can bottleneck modern, interactive renderers. With an algorithm like irradiance caching, shading points often use many cached values to interpolate a final shading calculation. As an area of future investigation, we propose storing the second-order partial derivatives of irradiance at cache points (a Hessian matrix) in addition to the gradient vectors. This matrix could be calculated during neighbor clamping, and then used to interpolate irradiance utilizing fewer cache points. It may be possible to return from a cache lookup after a single cache point is found, and extrapolate an irradiance value visually identical to one calculated with all nearby cache hits. This would substantially speed-up cache lookups.
7.2 Photon Beams

Before our implementation, photon beams had never been integrated into a production renderer so there were many questions we needed to answer. Initially we brainstormed several different approaches, even rendering photon beams outside of PRMan then deep compositing them into the final render. Ultimately we went with the approach described in chapter 5 which facilitated the rapid development of the artist tools described in chapter 6.

7.2.1 PRMan Functionality

Our photon beams implementation is efficient and fits in well with the PRMan framework. However, the PRMan rendering pipeline could be augmented with some additional functionality which might allow photon beams performance to scale in a way similar to GPU implementations. As we hypothesized in chapter 5, the unnecessary sorting of visible photon beams is probably why performance scales linearly. We suggest adding an additive blending hider which would bypass this sorting procedure for transparent surfaces with zero opacity.

For non-linear photon beams, investigating a more fitting representation within PRMan would certainly be worthwhile, since thick non-linear RiCurves are quite artifact-prone.
7.2.2 Artist Tools

Beyond implementation details, there is certainly future work in refining the authoring process of artist-designed volumetric lighting using photon beams. Allowing photon beams to be manipulated in a completely general way to design volumetric lighting brings up a number of usability concerns. As originally formulated, photon beams are only a pre-computed representation of physically-simulated light transport for rendering volumes in a physically-based way. As soon as artists manipulate this representation directly without physical simulation, there is no longer any explicit attachment to the physics of light transport.

The lack of physical integrity is certainly not a problem within the use-case of our implementation: animated films. Animated films are purposefully full of non-physical lighting, shading, and effects. The design and artistic nature of these films necessitates stylization and exaggeration, often in a way that is an intrinsic part of the storytelling. It is rare to rely on physical simulation in these films, and usually only done in instances when the desired effect is complex enough that artists lack the tools or ability to have a hand in them.

The tools we have created to design volumetric lighting have proven successful in the hands of skilled—and even unskilled—artists. Our tools allow manipulation of photon beams in the most general of ways, with very few constraints. Consequently not only is it possible to create volumetric lighting that is not plausible physically, but also not recognizable as lighting within a participating media.
One example of a difficulty with our system came in attempting to design the lighting of an underwater scene for the movie *Tangled*. Our lack of skill and experience at designing effects artistically did not preclude us from creating the keyhole shot detailed in chapter 5. Following the success of this shot, we attempted to design—utilizing photon beams—underwater lighting from glowing hair for a sequence of shots (figure 7.1). Given the short time we had and lack of experience, we were unsuccessful in completing something usable. This experience exposed some of the weaknesses in our tools.

Looking back at designing the lighting in this scene, the frustration mostly stems from not having a completely physical solution from which to start. With the other shots utilizing photon beams, the origin and direction of lighting was clear. This limited the degrees of freedom of lighting design, making physical simulation unnecessary and undesirable. However in contrast to other shots, the light source in this scene was relatively large and could create volumes of light in any direction; The problem was too unconstrained. If we had created a tool for creating photon beams from physical simulation, we believe it would have given us a good starting point for artistic experimentation, similar to the exploration done on physically simulated data in figure 6.2. Altering a physically accurate simulation in this way may have led to a more interesting result than the—albeit satisfying—final theatrical version, which did not utilize photon beams.

Overall, we believe future work in artist-driven volumetric lighting design should focus on integrating photon beam generation more tightly with rendering and lighting tools. Photon
Figure 7.1: Underwater volumetric lighting

beams could then be generated from light sources in a physically-based way, while still being able to be freely placed within the scene like any other light source or set of light sources. Once placed within the scene, the shading and shape of photon beams could be designed similarly to how we have described in chapters 5 and 6. This change would allow instances where photon beam orientation and placement can still be handled in a more artist-driven way, while giving a physically plausible starting point for artistic design in more unconstrained use-cases.

7.3 Conclusions

We have presented many algorithms for optimization of modern, complex rendering engines. We have presented an interactive caching solution for ray traced scenes even in the presence of
many lights which maintains the quality of ray-traced shadows. Integrating with irradiance caching, we have shown an increase in the speed of first-frame render time with complex direct lighting and single bounce global illumination. We have also demonstrated improvements to the irradiance caching algorithm. Our algorithm allows us to view a ray-traced scene with global illumination interactively under static lighting once the irradiance cache has been filled.

Our efforts have then segued to participating media, addressing efficient volumetric rendering and volume design within the Reyes rendering architecture. We have described our novel implementation of photon beams into PRMan. We have then presented our tools for generating art-directable volumetric effects. We observe that aspects of the physically-accurate photon beams approach relate closely to the methods artists normally use to reason about, and hand draw volumetric effects. By generalizing this approach and tying it to a geometric interpretation for volumes, and the lighting within them, we are able to expose an intuitive programmable model for designing volumetric effects.
APPENDIX A
ALBEDO AND ENERGY CONSERVATION
The term albedo has been used to refer to several different surface reflectance properties, even strictly limiting usage to the field of computer graphics. In general, albedo is a ratio of reflected light to incident light. Here the term means the percentage of reflected light from an opaque surface as a function of a single incoming direction $\omega_i$ of incident light. In this appendix, we develop the formulation of albedo from this simple definition into terms of the Bidirectional Reflectance Distribution Function (BRDF).

A.1 Directional Albedo Formulation

We begin our formulation in the most general terms, defining albedo as a ratio of incident power $\Phi_i$ and reflected power $\Phi_r$: $\Phi_r/\Phi_i$. We then refine our formulation by considering the case of a single surface point with incident radiance $L_i$ from direction $d\omega_i$. If we know the incident radiance $L_i$, we can calculate the incident power $\Phi_i$ by multiplying incident radiance by $dA$ and the projected solid angle $d\omega'_i$.

$$\frac{\Phi_r}{L_i dA d\omega'_i}$$

Similarly, to express $\Phi_r$ in terms of reflected radiance $L_r$, we multiply by $dA$ and $d\omega'_i$. In our definition of albedo, incident power comes from only a single direction, but can be reflected from a surface point throughout the entire hemisphere of directions. In contrast to
incident radiance over $d\omega_i$, reflected radiance is not necessarily constant over the hemisphere. Therefore, we must integrate differential reflected radiance $dL_r$ at $dA$ over the hemisphere of directions, represented by $\Omega_r$.

\[
\int_{\Omega_r} \frac{dL_r dA d\omega'_r}{L_i dA d\omega'_i}
\]  

(A.2)

Simplifying and using the definition of a BRDF $f_r$ (equation 2.2), equation A.2 becomes:

\[
\int_{\Omega_r} \frac{dL_r}{L_i d\omega'_i} = \int_{\Omega_r} f_r(\omega_i, \omega_r) d\omega'_r
\]  

(A.3)

### A.2 Energy Conservation

As referenced in chapter 2, in order for this surface reflectance model to be plausible, energy must be conserved. That is, the ratio of total power reflected $\Phi_r$ to total power incident $\Phi_i$ must be less than or equal to 1.

\[
\frac{\Phi_r}{\Phi_i} = \int_{\Omega_r} f_r(\omega_i, \omega_r) d\omega'_r \leq 1
\]  

(A.4)
We have restricted our formulation to a single direction of incoming light. To ensure energy conservation of a surface reflection function $f_r$, equation A.4 must be true for all directions:

$$\forall \omega_i \in \Omega_i \quad \int_{\Omega_r} f_r(\omega_i, \omega_r) d\omega_r' \leq 1$$  \hspace{1cm} (A.5)
APPENDIX B
COPYRIGHT NOTICES
In accordance with the copyright laws of the United States, this dissertation, in good faith, uses various copyrighted images under the Fair Use educational provisions. All copyrighted images are labeled as such with a watermark. Images not labeled with a watermark are either owned by Jared M Johnson or are public domain images.

This dissertation discusses various aspects of animated movies from the perspective of academic research. The purpose of this dissertation is purely educational, and non-commercial, being freely available to the public.

The copyrighted images reproduced within this dissertation have all been previously published. Also, we have reproduced as few as possible copyrighted images: a handful of frames from the feature-length animated film *Tangled*, just enough to aid in the expression of the research presented within this dissertation.

The copyrighted images have been used to create a new work, augmenting the original work with a new way of analyzing and viewing these images. This new work in no way competes with nor supersedes the original work. If anything, we have added value to the original work and possibly influenced readers of our work to purchase the original copyrighted works.

All Disney characters, including but not limited to, Rapunzel, Flynn Rider, and Pascal, are registered trademarks, trademarks and/or copyrights of Disney Enterprises Incorporated. In no case does this dissertation make any explicit or implicit claim to any intellectual property rights associated with Disney Enterprises Incorporated.
LIST OF REFERENCES


[DSC06] Kurt Debattista, Luis Paulo Santos, and Alan Chalmers. “Accelerating the Irradiance Cache through Parallel Component-Based Rendering.” In EGPGV2006 -


