Summer 2010

Multiresolution image-space rendering for interactive global illumination

Gregory Boyd Nichols
University of Iowa

MULTIRESOLUTION IMAGE-SPACE RENDERING FOR INTERACTIVE GLOBAL ILLUMINATION

by

Gregory Boyd Nichols

An Abstract

Of a thesis submitted in partial fulfillment of the
requirements for the Doctor of Philosophy
degree in Computer Science
in the Graduate College of
The University of Iowa

July 2010

Thesis Supervisor: Assistant Professor Chris Wyman

ABSTRACT

Global illumination adds tremendous visual richness to rendered images. Unfortunately, such illumination proves quite costly to compute, and is therefore often coarsely approximated by interactive applications, or simply omitted altogether. Global illumination is often quite low-frequency, aside from sharp changes at discontinuities. This thesis describes three novel multiresolution image-space methods that exploit this characteristic to accelerate rendering speeds. These techniques run completely on the GPU at interactive rates and require no precomputation, allowing fully dynamic lighting, geometry, and camera.

The first approach, *multiresolution splatting*, is a novel multiresolution method for rendering indirect illumination. This work extends reflective shadow maps, an image space method that splats contributions from secondary light sources into eye-space. Splats are refined into multiresolution patches, rendering indirect contributions at low resolution where lighting changes slowly and at high resolution near discontinuities; this greatly reduces GPU fill rate and enhances performance.

The second method, *image space radiosity*, significantly improves the performance of multiresolution splatting, introducing an efficient stencil-based parallel refinement technique. This method also adapts ideas from object-space hierarchical radiosity methods to image space, introducing two adaptive sampling methods that allow much finer sampling of the reflective shadow map where needed. These modifications significantly improve temporal coherence while maintaining performance.

The third approach adapts these techniques to accelerate the rendering of *direct* illumination from large area light sources. Visibility is computed using a coarse screen-space voxelization technique, allowing binary visibility queries using ray marching. This work also proposes a new incremental refinement method that considers both illumination and visibility variations. Both diffuse and non-diffuse surfaces are supported, and illumination can vary over the surface of the light, enabling dynamic content such as video screens.

MULTIRESOLUTION IMAGE-SPACE RENDERING FOR INTERACTIVE GLOBAL ILLUMINATION

by

Gregory Boyd Nichols

A thesis submitted in partial fulfillment of the
requirements for the Doctor of Philosophy
degree in Computer Science
in the Graduate College of
The University of Iowa

July 2010

Thesis Supervisor: Assistant Professor Chris Wyman

Graduate College
The University of Iowa
Iowa City, Iowa

CERTIFICATE OF APPROVAL

PH.D. THESIS

This is to certify that the Ph.D. thesis of

Gregory Boyd Nichols

has been approved by the Examining Committee for the thesis requirement for the Doctor of Philosophy degree in Computer Science at the July 2010 graduation.

Thesis Committee: ______________________________
Chris Wyman, Thesis Supervisor

____
James Cremer

____
Joseph Kearney

____
Juan Pablo Hourcade

____
Steve Cunningham

____
Jun Ni

For my grandfather,
Dr. Arland Pauli
who did it first,
and did it with style

ACKNOWLEDGEMENTS

My name is on the cover, but this thesis was by no means a solo effort. I'd first like to thank the most direct contributors, my coauthors – Chris Wyman, Jeremy Shopf, and Rajeev Penmatsa – for the huge amount of work they put into the papers that comprise this document. The faculty and staff at the University of Iowa were tremendously helpful, as were my peers Scott Davis, Qi Mo, Rajeev Penmatsa, and Erik Krohn. Thanks also to my computer science professors at Central College, who first convinced me that graduate school would be a good idea.

Outside of school, I am grateful for the steady encouragement I received from those around me: my friends, my parents, my sisters, my extended family, and of course Melanie. My faith in myself sometimes wavered; theirs never did.

Finally, I owe an enormous debt to my advisor Chris Wyman not only for funding, but for his patience, high standards, and incredibly hard work on my behalf. Without his guidance, I never would have come this far.

This work was partially supported by DARPA grant HR0011-09-1-0027 and a University of Iowa Graduate College Summer Fellowship. 3D models are from Google's 3D Warehouse, the Stanford repository, the AIM@SHAPE repository, the Utah 3D Animation Repository, Keenan Crane, and the 3D Meshes Research Database from the INRIA Gamma Group. Video stills in Chapter 5 are from "Big Buck Bunny" (copyright © Blender Foundation), and "Sita Sings the Blues" by Nina Paley. I would also like to acknowledge hardware donations from NVIDIA and AMD.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

Much of our perception of reality comes from the human visual system, which allows us to recognize the world around us from the light in our environment. Before reaching our eyes, this light may be reflected or refracted; it may be partially absorbed; it may be focused or it may diverge. And although only the tiniest fraction will makes it to our retinas, that incoming illumination allows us to perceive the world: to recognize objects, to discern color, to comprehend the spatial relationship of one thing to another, to guess at the feel and texture of materials, and much more.

Humans can also extract meaningful visual information from *synthetic* images. The creation of these images is far from new; the attempt to synthesize images that have the appearance of reality is as old as art itself. More recent is the attempt to synthesize, or *render*, photorealistic images using computers, by modeling the effects of light in an artificial environment. Creating a truly convincing synthetic image requires plausible reproduction of reflection, refraction, and the many other effects of light that enable our accurate perception of the world.

This is an ambitious goal, but an achievable one: given enough time and computational power, most of what we see can be plausibly recreated in a synthetic image. However, "enough" can add up to a great deal. A still more demanding goal is to generate photorealistic images *interactively*: to render each image, start to finish, in under 30 milliseconds. At these speeds, a user can smoothly move around a rendered scene and interact with its contents. Much current research in computer graphics attempts to move towards the creation of photorealistic graphics at interactive speeds. The work described in this thesis takes several small steps in that direction.

1.1 Local and Global Effects

Each pixel in a rendered image depicts some 3D point in a synthetic scene; to determine what color that pixel should be, the light reaching the corresponding point must be computed. The simplest and least computationally expensive methods of computing illumination at a point consider only information local to that point: the point's location in 3D space, the surface normal at that point, the location of any light sources, and the direction in which the viewer is looking. The light sources in these models are simple as well; although light sources in the real world have area, local models allow only infinitely small point lights. Furthermore, every surface facing towards the light source receives illumination, regardless of any occluders. Due to these restrictions, local illumination models are incapable of reproducing effects such as shadows, color "bleeding" between surfaces, reflection and refraction effects, and other lighting effects that add a crucial level of realism.

In the earliest interactive graphics applications, given the scarcity of computation resources available for shading objects, these limitations were acceptable. Constant or "flat" shading, Gourad shading [40], Phong shading [108], and even wireframe rendering [5] were widely used. The addition of techniques such as shadow mapping [21] enabled the interactive rendering of hard shadows, which proved to be an important visual cue for judging spatial layout of a scene [57].

These local illumination models are quite efficient, but the results are rarely convincing. In reality, light reaching a point includes not only *direct illumination* - light that comes straight to the point from a light source - but also *indirect illumination* - light that first bounces off one or more other surfaces. Additionally, other surfaces may block some or all of that light, producing shadows. Creating these effects requires knowing all surfaces that may affect incident illumination. Because global information is necessary, this lighting is often referred to as *global illumination*. Global illumination adds an essential level richness to rendered images, and as such, a huge amount of research has focused on its efficient rendering.

1.2 GPU Rendering

Much work of the past decade has focused not just on algorithms to create compelling images, but on the computing hardware that runs them. Recent widespread availability of powerful consumer-level graphics cards has dramatically changed how graphics algorithms are created.

For much of the history of computer graphics, the only available computational resource was a computer's Central Processing Unit (CPU), which is responsible for most of the general purpose computation that takes place. In addition to handling the basic functions of a computer, such as running the operating system and application software, a system's CPU also had to handle any processing required by graphics algorithms. Higher level machines sometimes offered multiple CPUs, capable of processing multiple instruction streams in parallel, but the number of CPUs – and thus the available parallelism – was often limited.

In the 1990s, specialized graphics coprocessors found widespread success in consumer market. These processors, known as *Graphical Processing Units* (or GPUs), were initially designed to accelerate 2D graphical operations required by graphical user interfaces such as Microsoft Windows. As computers became more powerful, 3D applications – primarily high end CAD applications and consumer video games – grew in popularity.

Driven primarily by these markets, GPUs exploded in processing power during the late 1990s and 2000s. The massively parallel nature of computer graphics spurred the development of GPUs with dozens or hundreds of computational cores that could process many elements in parallel. These cores, originally limited to a fixed set of functions, grew in sophistication and eventually became programmable, enabling algorithms to run directly on the GPU with minimal CPU intervention. Modern GPUs are highly programmable, massively parallel, and offer a great deal of processing power – a combination ideal for computer graphics. Consequently, much modern graphics research focuses on algorithms tailored to execute partially or wholly on the GPU.

1.3 Overview of this Work

Our work focuses on plausible, GPU-based approximations of indirect and direct illumination. Specifically, the techniques described in this thesis are image-space effects, which avoid wasted processing on off-screen geometry as well as enabling a number of important performance advantages on the GPU.

This thesis is organized as follows: Chapter 2 discusses previous and alternate approaches to global illumination, with a focus on interactive techniques. Chapters 3, 4, and 5 present the core contributions of this thesis: our research towards interactive global illumination. Specifically, Chapter 3 presents an image-space multiresolution approach [102,103] to interactively rendering indirect illumination on the GPU. Chapter 4 discusses an extension of that work [101] that improves overall efficiency and increases accuracy and temporal coherency. Chapter 5 presents additional background, and then proposes a a novel technique towards the rendering of *direct* illumination from large dynamic area lights [100]. Finally, conclusions and future directions are discussed in Chapter 6.

CHAPTER 2
BACKGROUND AND PREVIOUS WORK

Even considering the relative youth of the computer graphics field, photorealistic rendering has seen an enormous amount of work, much of it focused on global illumination. This chapter describes the broad categories of that work. Although this thesis as a whole focuses on interactive algorithms, many of the methods presented in this chapter were not published as real-time methods. However, many either became real-time thanks to computational or algorithmic advances, or inspired other real-time algorithms.

In this chapter, Section 2.1 briefly presents a background to the global illumination problem. The remaining sections discuss particular classes of algorithms that have been proposed to solve that problem.

2.1 Global Illumination

To compute physically-based global illumination, a renderer must approximate the behavior of light in the real world. This section introduces radiometric units, describes how surface reflectances affects light, and describes the Rendering Equation [68], which mathematically models interactions between surfaces and scene lighting.

2.1.1 Radiometric Units

Technically, global illumination algorithms compute or approximate the steady-state distribution of radiant energy in an environment. The study and measurement of this radiant energy is known as *radiometry*. The radiant energy in a synthetic scene may be measured from an actual environment, or specified using real-world units. In either case, it is helpful to understand the measurement of real-world electromagnetic radiation.

The simplest radiometric quantity is *radiant energy*, measured in joules, which describes the energy of electromagnetic waves. *Radiant flux* (often denoted as Φ) is measured in watts, and describes how much energy per unit time flows to, from, or through a surface. *Irradiance* (E) is the radiant flux incident on a surface, per unit surface area. Radiant exitance, on the other hand, is known as *radiosity*; both quantities are measured in watts/m^2.

Radiance is radiant flux per unit area per unit time per unit solid angle (watts / (steradian $*$ m^2)). Put simply, radiance describes how much energy leaves a given area of a surface in a given direction. This quantity can be used to describe the "look" of a surface as it is seen from a particular angle of view.

2.1.2 Surface Reflectance

The appearance of a surface comes from the way it reflects light. A surface that reflects each incident light ray perfectly will appear mirror-like; a surface that reflects light equally in all directions, regardless of incident direction, will yield a paper-like diffuse appearance. Early graphics techniques often employed very simple reflectance models such as these, allowing purely specular and purely diffuse surfaces but very little else.

Real world materials, however, are rarely purely diffuse or specular. To better approximate the reflectance characteristics of real surfaces, Nicodemus et al. describes the *Bidirectional Reflectance Distribution Function*, or BRDF [104]. A material's BRDF defines how much light is reflected in an outgoing direction $\vec{w_o}$ for an incoming direction $\vec{w_i}$ at a given position x.

The BRDF has become a ubiquitous method of specifying reflectance for realistic materials, and many different BRDFs have been proposed. For example, the Cook-Torrance [20] and He [50] models are physically based BRDFs that attempt to model physical reality. The Ward [159] and Lafortune [87] models, which are based on empirical data captured from real-world reflectance measurements, have also been shown to be effective.

An underlying assumption of the BRDF is that an incident ray of light will reflect away from the surface at the same point. This assumption breaks down in surfaces that exhibit effects such as subsurface scattering and translucency, where the outgoing ray exits the surface at a different point than the incident ray. The *Bidirectional Scattering Surface Reflectance Distribution Function*, or BSSRDF, allows for these effects.

2.1.3 The Rendering Equation

Given the radiometric quantities defined in Section 2.1.1 and the BRDF function as presented in Section 2.1.2, the way that light moves around within a scene can be described by the *rendering equation* [68]. Hence, rendering systems that aim for photorealism attempt to solve or approximate this equation. Specifically, the rendering equation defines an outgoing radiance at a given point x and in a given direction $\vec{w_o}$:

$$L_o(x, \vec{w_o}) = L_e(x, \vec{w_o}) + \int_{\Omega} L_i(x, \vec{w_i})\ f_r(x, \vec{w_i} \rightarrow \vec{w_o})\ (\vec{w_i} \cdot \vec{n})\ d\vec{w_i} \tag{2.1}$$

where:

- $L_o(x, \vec{w_o})$ is the 'result' of the equation: the light leaving x in direction $\vec{w_o}$
- $L_e(x, \vec{w_o})$ is the light emitted from the surface itself at x in direction $\vec{w_o}$
- $L_i(x, \vec{w_i})$ is incoming light at x from direction $\vec{w_i}$
- $f_r(x, \vec{w_i} \rightarrow \vec{w_o})$ is the *BRDF* (Bidirectional Reflectance Distribution Function), which defines the amount of light reflected from incoming direction $\vec{w_i}$ to outgoing direction $\vec{w_o}$ at point x
- $(\vec{w_i} \cdot \vec{n})$ is the attenuation of the incoming light based on its angle with the surface normal

To simplify, the rendering equation says that the radiance leaving a point in a given direction is equal to the radiance emitted from that point, plus any illumination in the visible hemisphere reflected toward that point. Of course, a complete evaluation of the L_i term requires further invocation of the rendering equation; the resulting recursive nature of this integral is one reason global illumination effects are so costly.

2.2 Coarse Ambient Approximations

Early interactive applications lacked the computational resources to incorporate global illumination. In these early applications (and even in simple 3D applications today), indirect lighting was often approximated using an ambient term - a constant color that added to all the colors in the scene to ensure that each point has at least some color, even if that point receives no direct illumination. Because the ambient term is constant and independent of any geometric occlusion in the scene, it adds almost no cost to the rendering process. However, the ambient term is an extremely crude approximation of global illumination and is inadequate for any system aiming towards photorealism.

Further work such as that of Parker et al. [106] improves this somewhat, allowing for the use of a directionally variant ambient term. This reduces the somewhat flat appearance that can come from the use of a static ambient term, and allows shading variation on surfaces that are not directly lit.

Similarly, Castro et al. [14] compute an extended ambient term by grouping each surface into one of six classes by normal vector, and solving a small system of linear equations. This method yields a better approximation than a constant ambient term, but inherits many of the disadvantages of the radiosity approach it resembles (see Section 2.3) – notably, dynamic scenes become difficult.

All of these techniques assume an extremely simple illumination setup, and as such are generally considered inadequate approximations to global illumination.

2.3 Radiosity

Radiosity [16, 39] is an application of the finite element method to solving the rendering equation for scenes with purely diffuse surfaces. In the radiosity algorithm, each surface of the scene to be rendered is divided into small patches. In an extensive preprocessing step, a links are created between each pair of mutually visible patches. By computing the energy transfer and visibility between each set of patches, a linear system of equations can be constructed; the result approximates the global illumination equilibrium throughout the environment.

With n patches, the initial link computations are $O(n^2)$ in both time and space – quite expensive, especially with complex environments. Later work seeks to reduce this expense. Cohen et al. [17] imposes a two-level hierarchy of patches on the environment to reduce the number of interactions. Hanrahan et al. [47] further extend this idea with *hierarchical radiosity*, which subdivides each patch into a hierarchy; patch to patch interactions use the appropriate levels on both ends. Other researchers [78, 139] propose augmenting hierarchical radiosity with a clustering approach, allowing small, similar surfaces to be "grouped" into larger ones in addition to subdividing large surfaces into smaller elements. The computational cost of computing links can be further reduced by considering importance [140], which avoids computation in areas that have little effect on the surfaces of interest, and by preprocessing global visibility to avoid link computation when patches cannot "see" each other [145].

Because computed radiosity solutions are viewpoint independent, they have long been used in interactive applications. While the radiosity computations themselves are not generally possible in real time, the results can easily be “baked” into a scene, allowing easy and cheap runtime global illumination in static, diffuse scenes. Further work attempts to allow limited dynamism by recomputing only the changed areas of a scene [15]. Another approach tries to compensate for lighting and geometry changes by shooting “negative light” [109].

2.4 Ray Tracing and Path Tracing

Whitted [162] first describes *recursive ray tracing*, which shades points by tracking individual “rays” of light as they move through a scene, computing precise interactions with other objects in order to determine color and shadowing information. Recursive ray tracing (often just called ray tracing) offers simplicity and elegance, and can be easily extended to handle a variety of complex illumination effects like reflection and refraction. However, ray-based techniques must rely on efficient ray-geometry acceleration structures such as bounding volume hierarchies [72,122,161,162], kd-trees [49], or uniform grids [7,36] to achieve reasonable efficiency; the overhead required to build and maintain these structures may be substantial.

Early ray tracing approaches select rays deterministically, disallowing complex BRDFs or “fuzzy” effects such as motion blur and depth of field, which require the evaluation of complex integrals. Cook et al. [19] addresses this with a solution known as *distribution ray tracing* or *stochastic ray tracing*. This approach uses multiple rays at each surface point to sample the analytic function in question, essentially applying

Monte Carlo integration [97] to image synthesis. Further extensions of ray tracing allow its use in rendering densities within a volume grid [67] by allowing light to scatter within such a volume. This extension allows ray traced rendering of effects such as clouds, fog, flames, dust, and particle systems.

Kajiya [68] describes *path tracing*, extending distribution ray tracing to fully solve the rendering equation. Path tracing methods calculate the complete path of a ray from the light source to the eye, including its reflection from each intervening surface. By computing many such paths (probablistically selecting a ray at each surface according to that surface's BRDF) path tracing allows computation of a complete global illumination solution. A key problem, however, is the amount of computation required to produce images with acceptable levels of noise. Using Monte Carlo integration, path tracing converges at a rate of $1/\sqrt{N}$: quadrupling the number of sample rays only halves the amount of error. In complex environments, convergence may require an unacceptable amount of rendering time.

Conventional path tracing often requires many indirect ray bounces before encountering a light source. *Bidirectional path tracing* [86] improves speed by simultaneously originating ray paths from both the eye and any light sources, reusing each intersection point to create multiple eye-light paths. Other approaches, such as Metropolis light transport [149], attempt to reuse known-good paths by probabilistically mutating them. Furthermore, other approaches [29, 30] reduce the number of required paths by considering the relative importance of each path to the scene, leading to fewer paths in areas that do not contribute much useful illumination.

2.4.1 Interactive Ray Tracing

Thanks to the explosion of computing power (both on the CPU, and more recently on the GPU), real-time ray tracing has received considerable attention over the past decade. Early work in this area focuses on interactive ray tracing of static scenes: once acceleration structures have been built, ray tracing benefits from massive parallelism, scaling linearly with the number of processors used. Early interactive ray tracers often required large supercomputers [73, 106] or PC clusters [151] and used only simplistic material models.

Ray tracing performance largely depends on ray traversal costs – the raw speed at which rays can be intersected with geometry. Several approaches reduce these costs by exploiting spatial coherence (i.e., rays that are close together and moving in similar directions). One technique, known as *beam tracing* [52], traces wide "beams" instead of infinitely thin rays, capitalizing on the likelihood of adjacent rays to hit the same surfaces. Wald et al. [150] suggest a similar approach, aggregating groups of rays into bundles or "packets". Each packet simultaneously traverses through acceleration structures, amortizing the costs of memory access, function calls, and traversal computations over the rays in the packet. Where applicable, SIMD instructions are employed to process the packet in parallel. Further work [26, 115] uses interval arithmetic to avoid intersection or traversal steps based on conservative bounds of the rays. These techniques work well for coherent eye rays, but are not as effective with less coherent secondary rays. Instant Caching [24] mitigates this effect by exploiting temporal coherence, reusing diffuse ray paths between frames when possible.

While past work on interactive ray tracing focused on the efficient *traversal* of acceleration structures, much recent work examines their efficient *creation* – a necessity when rendering fully dynamic scenes. For an overview of work in this area, see Wald et al.'s [152] relatively recent survey of the state of the art in ray tracing animated scenes.

Much recent research focuses on GPU ray tracing, capitalizing on the massive parallel processing power modern GPUs offer. Methods exist that execute entirely on the GPU [111]; others emphasize specific strategies such as optimal traversal of kd-trees [56] and bounding volume hierarchies [45], as well as efficient construction of acceleration structures on the GPU [128, 168]. Aila and Laine [2] describe strategies to boost performance by enhancing parallelism of ray tracing on GPUs. GPU ray tracing is an active area of research, and given the advent of comprehensive toolkits such as NVIDIA's OptiX [121], it will likely remain so.

2.5 Photon Mapping

Some ray paths, especially those containing specular interactions, often prove difficult to compute when originating from the eye. The number of such rays that are successfully traced back to the light source can be vanishingly small, requiring an enormous number of rays for a converged image.

One alternative described by Arvo [6] is to originate rays (or "shoot photons") from the light source instead of the eye. This alteration guarantees that all ray paths intersect the light source. In this technique, photon hits are stored in *illumination* *maps*, a 2D bitmap similar to a texture map, created for each diffuse surface that may receive specular illumination. Thus, incident illumination at each shade point can be simply determined by accessing the illumination map at the correct location. An extension of this technique [18] allows adaptive creation of these illumination maps, densely sampling illumination only where necessary (i.e., in high frequency regions such as shadow edges).

Jensen [65] proposes a similar, now ubiquitous, approach known as the *photon map*. Like previous approaches, photons are emitted from the light source(s) and traced through a scene. Instead of storing irradiance with an illumination map within a per-surface texture, however, the photons themselves are stored (including direction, color, and energy) in 3D space, often within a kd-tree. When computing incident illumination at a point, nearby photons are gathered and combined to produce a result.

One way to reduce variation in the computed illumination is to shoot photons as evenly as possible. This can be accomplished using Monte Carlo methods [123] or low-discrepancy quasi-random sampling [74]. Other methods [62, 107] suggest importance sampling to concentrate photons in areas of interest. Photon mapping can be extended to create soft shadows efficiently [64], and to work with non-diffuse surfaces [63] as well as participating media such as smoke and fog [66].

Like interactive ray tracing, GPU photon mapping is an active area of ongoing research; particular attention is being paid to GPU photon mapping strategies. Purcell et al. [110] were the first to describe a GPU-based photon mapping implementation. In this work, a simple GPU ray tracer [111] traces photons into the scene; these are then scattered into a regular grid using a vertex shader and the stencil buffer. Other methods take a hybrid approach, dividing the workload between the CPU and GPU. Hachisuka [46] suggests performing conventional photon mapping on the CPU, and describes a rasterization-based final gather approach on the GPU. Wang et al. [157] partitions sparse sets of raytraced sample points into spatially coherent clusters, computing a final gather for each cluster using GPU-based photon mapping. Image Space Photon Mapping, a recent hybrid technique from McGuire and Luebke [93] achieves interactive rates at high resolution. Their method renders the first and final photon bounces as image-space buffers on the GPU, performing world space ray tracing on the CPU to compute the intermediate bounces. The GPU then performs final scattering as an image space operation, yielding high quality results in real-time.

2.6 Illumination Caching

Aside from sharp changes at discontinuities, illumination often changes slowly over surfaces. This property enables a class of techniques that compute illumination (usually using ray or path tracing) only at sparse points, storing them in a cache (often using an acceleration structure to optimize retrieval speed). These points are then used to interpolate the rest of the illumination, avoiding the necessity and expense of computing it at every point.

Ward et al. [160] first describe this idea as *irradiance caching*. The authors apply the technique to indirect illumination with path tracing, obtaining significant performance improvements by storing illumination samples within an octree. Later work [83–85] extends this approach to handle glossier BRDFs, by caching radiance instead of irradiance. Lehtinen et al. [90] use a similar approach, although rather than using the cached values directly, their method uses them to define basis functions that represent output lighting. Gautron et al. [38] describe a GPU-friendly variant of radiance and irradiance caching that employs a scattering approach rather than gathering, although this reformulation precludes indirect shadows. A related approach known as *lightcuts* [154, 156] computes a similar set of point samples and then builds a hierarchy over them, adaptively selecting the correct level of point samples for each shadepoint. Herzog et al. [55] combines lightcuts with a cache splatting method to increase efficiency and allow anisotropic BRDFs.

Ward and Heckbert [158] augment irradiance caching by computing *irradiance gradients* while sampling the visible hemisphere to evaluate irradiance. Irradiance gradients provide a reasonable approximation of how irradiance is changing on a particular surface with respect to position and orientation. This information can then be used to guide interpolation of the cached samples, yielding higher quality results.

Another influential approach known as the *irradiance volume* [43] precomputes a coarse volumetric approximation of the irradiance function. With an irradiance volume, an approximation of the irradiance of an arbitrary point within the scene can quickly be approximated by sampling nearby values within the volume. Several similar approaches have been proposed for interactive applications. Nijasure and Goel [105] create an extremely coarse volumetric grid similar to an irradiance volume, rendering a cube map at each grid point to evaluate incoming radiance. Irradiance volume methods also form the basis of cascaded light propagation volumes [69], a real-time technique discussed in Section 2.11.

Other approaches involving illumination caching attempt to take advantage of temporal as well as spatial coherence [37], and decouple sample generation from rendering [147, 155], achieving essentially arbitrary framerates. With this approach, however, scene modifications may fade in over time as the cached samples “catch up” to the current application state.

2.7 Precomputed Radiance Transfer

A broad class of real-time global illumination techniques partially or entirely precompute light transport. In these methods, an expensive precomputation pass determines how incident lighting is reflected at each point on selected geometry, yielding a *transfer function.* This function is compressed to allow a low-frequency approximation of the lighting to be efficiently reconstructed at runtime.

Although a number of basis functions have been employed to encode the transfer function (such as wavelet bases [80, 99] and radial basis functions on the sphere [148]), the most common representation uses *spherical harmonics*, the solutions to Laplace’s equation when restricted to the sphere. With spherical harmonics, a function on the sphere is approximated using a set of orthonormal basis functions known as the Associated Legendre Polynomials. These functions are broken into bands; higher order approximations of spherical functions are more accurate but require more coefficients (an n-band approximation uses $(n+1)^2$ coefficients), and are thus more computationally expensive to evaluate.

While spherical harmonics have long been employed to solve potentials problems in physics – notably those related to heat transfer [12] – they have proved quite useful in computer graphics as well. Ramamoorthi [112] uses spherical harmonics to encode irradiance environment maps, allowing interactive rendering of diffuse objects with distant illumination.

Sloan [71, 136] uses precomputed radiance transfer, represented with spherical harmonic functions, to represent the response of an object or scene to a lighting environment. This method convincingly reproduces such effects as soft shadows and diffuse interreflections. It also works quite well for indirect illumination, which is often low frequency and thus can be approximated convincingly with low-order spherical harmonics. However, early PRT approaches impose severe limitations: surfaces are restricted to diffuse (or slightly glossy) materials, models are required to be static, and lights must be distant and low-frequency for convincing results. These methods also preclude the use of local high-frequency effects such as normal maps. Finally, the large amount of required prepocessing limit the applicability these methods to dynamic scenes and models.

Further work by others has mitigated these limitations to some degree. Additional work by Sloan [135] and Lehtinen et al. [89] extends PRT methods to work with to a broader class of BRDFs, and enables visibility due to self-occlusion [41]. Two later publications [137, 138] focus on the use of precomputed radiance at a local scale, allowing for limited amounts of deformation. Sun and Ramamoorthi [142] relax limits on viewpoint and light position, allowing for interactive relighting. Sloan [133] explores the use of PRT with normal maps, allowing the rendering of local high frequency detail. Although PRT still works best with environmental lighting, other work [82] enables the limited use of PRT with local lights. Transport fields [61, 169] extend this approach to allow scenes with a few rigid, dynamic objects. Related techniques [70, 114] allow simple deformable models, albeit without indirect illumination. However, all of these approaches still require substantial precomputation, largely precluding the use of completely dynamic geometry.

2.8 Approximate Methods

Expensive accurate global illumination solutions may not be necessary when speed trumps realism. For many applications, a coarse approximation often suffices.

One common example is *ambient occlusion* [170], a widespread effect in the visual effects and animation industries. Ambient occlusion approximates indirect illumination by darkening direct lighting based upon the amount of occlusion from neighboring geometry. Even though ambient occlusion is computed independently of any light source, when modulated with simple local lighting models, the results behave similar to those using indirect lighting.

Precomputed ambient occlusion provides a cheap alternative to complex radiosity solutions, but assumes static geometry. Bunnell [11] describes an iterative ambient occlusion approximation for simple dynamic models. Other techniques [79, 92] precompute occlusion "fields" for rigid objects, allowing these objects to move while occluding nearby geometry. In recent years, an approximation of ambient occlusion known as *screen space ambient occlusion* (SSAO) [10,96] has become ubiquitous. This method only computes occlusion using visible geometry, which can lead to visible artifacts. However, SSAO is easily achievable in real-time, and yields results of sufficient quality for interactive applications such as video games.

Ambient occlusion computes visibility, not lighting; however, similar approaches can be applied to create coarse lighting effects. Iones et al. [60] employ ideas from ambient occlusion to create an ambient light model based on obscurances. This model yields crude viewpoint-independent indirect lighting for diffuse surfaces that can be baked into lightmaps and easily accessed in real-time. Mendez et al. [94] describe an extension to obscurance-based ambient lighting that adds color bleeding, and allows for a limited amount of scene dynamism. Another approach, *Screen Space Direct Occlusion* (SSDO), [120] generalizes SSAO to directional occlusion, adding a single bounce of indirect light.

2.9 Instant Radiosity

An influential publication by Keller [75] introduces *instant radiosity*, a method for computing indirect illumination in which the lighting in the scene is approximated by point light sources generated by a quasi-random walk. In this technique, a set of *virtual point lights* (VPLs) are traced into a scene, potentially reflecting from surfaces according to their BRDFs. For each VPL, the scene is rendered with hard shadows (using rasterization hardware for fast rendering), with a point light source at the origin of the photon. The cumulative illumination from the VPLs approximates the indirect illumination in the scene.

Instant radiosity – and more specifically, the virtual point light concept – has proven hugely influential in interactive rendering techniques. However, instant radiosity requires a separate shadow map to be rendered for each VPL. However, as each object must be rasterized once for each virtual point light, this proves a severe performance bottleneck, directly demonstrating the cost of visibility queries in global illumination.

One drastic method of avoiding these costs is to ignore indirect occlusion altogether, obviating the need to render shadow maps at all. The indirect illumination techniques presented in Chapter 3 and Chapter 4 adopt this approach. Short of this extreme, many other approaches attempt to lessen the cost of computing visibility. These techniques are broadly reviewed in Section 5.2.

2.10 Image Space Algorithms

Many graphics algorithms require global information about the scene to compute results, but do not take into account how the results will be used. Radiosity (see Section 2.3), for example, requires information about all the diffuse surfaces in a scene, but does not consider the eventual viewpoint from which the results will be

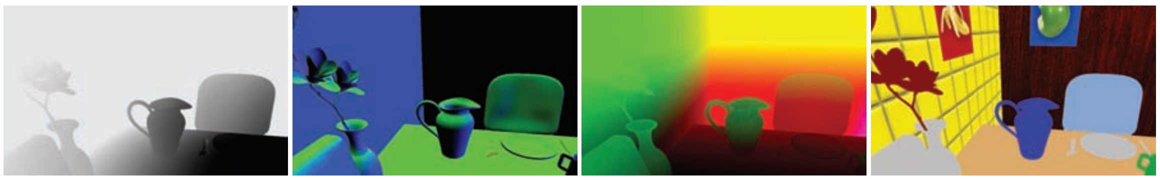


Figure 2.1: Components of a reflective shadow map: a linear distance from the eye (top left), a surface normal (top right), a world space fragment position (bottom left), and a reflected flux (bottom right).

seen. Such algorithms are said to operate in *object space* or *world space*.

A growing set of algorithms instead operate in *image space*. Image space algorithms operate using one or more images as input, avoiding more complex structures when possible. For example, one oft-used strategy known as *deferred shading* [25] renders data about visible surfaces (such as color, surface normal, and view-space position) into a G-buffer [125]. Shading for these surfaces is computed in a later pass, using solely the information from the G-buffer as input; this strategy ensures that illumination is only computed for surfaces that will actually be visible.

Image space algorithms map very well to GPUs, which are designed to allow fast coherent texture fetches from input images. As such, GPU-accelerated image space algorithms have become an active area of research in the past several years.

2.10.1 Reflective Shadow Maps

Our work relies heavily upon *reflective shadow maps*, an image-space algorithm designed to compute indirect illumination. Reflective shadow maps in turn build on the idea of instant radiosity, using shadow mapping hardware to generate virtual lights directly instead of using quasi-random path tracing. The map itself consists of a standard shadow map augmented by additional buffers to store surface normals, positions, and reflected flux (see Figure 2.1)—essentially a light-space G-buffer [125]. Rather than computing or approximating visibility, reflective shadow maps [22] entirely ignore visibility for indirect rays, assuming that viewers will not notice incorrect visibility for secondary illumination.

Dachsbacher et al.'s original method [22] uses a gathering approach to sample nearby locations in the reflective shadow map for each pixel in the final image; eye-space interpolation reduces illumination artifacts due to low sampling rates in both eye and light space. Later work [23] reformulates reflective shadow mapping using a shooting algorithm (see Figure 2.2), splatting each VPL's contribution onto a nearby region in eye-space. This technique extends to glossy materials and simple caustics by elongating the splat size based upon the material's BRDF, and importance sampling the shadow map allows selection of a good set of VPLs based upon flux distribution.

One of the problems with splatting illumination from VPLs is excessive overdraw. In theory, each VPL can affect final illumination everywhere in the scene; using 1000 point lights requires computing contributions for 1000 splats at each eye-space pixel. The authors propose reducing overdraw by restricting splat sizes. Beyond a certain distance from each VPL, indirect contributions are ignored. This gives an ideal parameter for tuning performance, but darkens illumination significantly as splat sizes shrink.

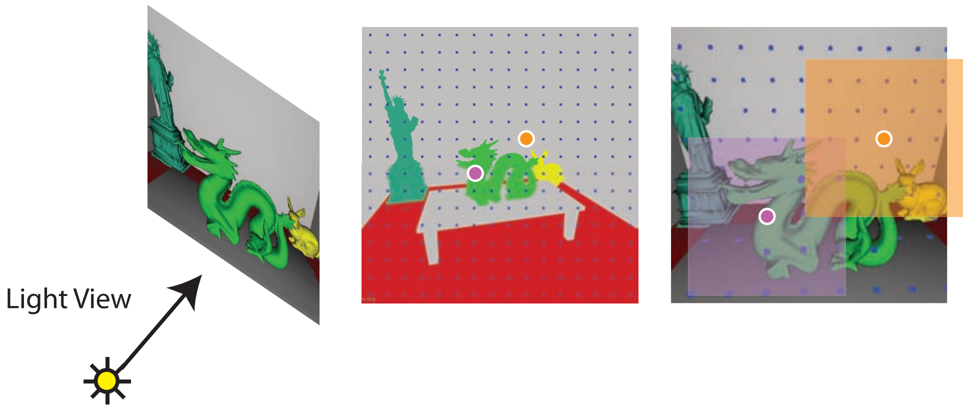


Figure 2.2: A light space rendering of the scene (left) yields a reflective shadow map, which is then sampled to create virtual point lights (VPLs). Here, 256 uniformly sampled VPLs are created (middle). In the splatting formulation of reflective shadow maps, a large is splatted for each VPL, centered around its eyespace location (right), to compute its illumination contribution to the scene. These splats frequently have limited area of effect. This process is illustrated using two VPLs marked with orange and purple.

2.10.2 Splatting Approaches

Interactive techniques often rely on splatting, as gathering sometimes proves less amenable to GPU acceleration. Gautron et al. [38] approximate global illumination using a splat-based renderer with a radiance cache. Shanmugam and Arikan [132] use billboards as splats to compute ambient occlusion on surfaces within the splat's influence. Sloan et al. [134] use splats to accumulate indirect illumination from spherical proxy geometry. Caustic mapping [131] frequently uses splats to represent photon energy, varying splat size to account for divergent photons and reduce sampling noise [166].

Even offline illumination rendering has investigated splatting as an alternative to gathering techniques, allowing more accurate illumination on high-frequency geometry and the removal of low-frequency noise [54].

However, many interactive splat-based illumination algorithms simply assume splats must be rendered at full resolution, or clamp splats to a "reasonable" size to maintain performance. Point-based rendering [124] and volume rendering [88] use multi-resolution splatting to achieve interactive speeds. Lehtinen et al. [90] use an adaptive hierarchical point sampling method to induce a basis function for PRT, which is rendered using GPU-based splatting. Offline rendering techniques frequently use multi-resolution and hierarchical techniques to reduce computational costs (e.g., hierarchical radiosity [47]).

In our work, we draw inspiration from recent caustic work [165] that renders illumination from splats into a multi-resolution image. This allows capturing illumination from very large splats into coarse buffers and fine illumination details in high resolution buffers while maintaining small splat sizes, dramatically reducing the overhead introduced by overdraw.

2.11 Recent Work

Since the publication of the work in Chapter 3 and Chapter 4, work towards GPU-based interactive global illumination has continued. Several recent techniques drew inspiration from the methods presented in this thesis.

Notably, Kaplanyan proposes *cascaded light propagation volumes* [69], an adaptation of irradiance volumes for real-time indirect illumination. Like our work, this method computes an eye-space G-buffer and a reflective shadow map (see Section

2.10.1); rather than splatting or gathering directly from the RSM, it is used to populate a coarse volumetric grid with spherical harmonic radiance coefficients. Illumination is then propagated through the grid. The illumination for a rendered point can be obtained by interpolating oefficients and evaluating an intensity function. The authors also describe methods for selectively increasing grid resolution using cascaded grids, and introducing coarse indirect occlusion. This technique proves quite efficient, but is limited to diffuse (or slightly glossy) materials, and provides no clear method to increase quality using additional VPLs.

Soler et al. [141] describe an image-space indirect illumination technique. Like the method we present in Chapter 4, theirs is a hierarchical technique. However, instead of dividing the screen into multiresolution patches, their technique computes indirect illumination for the entirety of each mipmap level. To yield acceptable performance, only local sampling is performed at each level, taking advantage of texture cache coherence to achieve acceptable speeds. A final indirect image is created by combining each level using joint bilateral upsampling [81]. This technique works efficiently, but lacks indirect occlusion and can suffer from temporal artifacts.

CHAPTER 3
MULTIRESOLUTION SPLATTING FOR INDIRECT ILLUMINATION

3.1 Introduction

Indirect illumination represents light reaching a surface after previously interacting with other surfaces. While this lighting adds tremendously to visual richness and scene realism, the costs to track multi-bounce light reflections often prove prohibitive. Chapter 2 describes many different methods to address this problem, but these techniques impose restrictions, often ignoring color bleeding or restricting the motion of geometry, lights, or viewer. Rather than deal with these restrictions, many interactive applications forgo complex global illumination entirely.

However, physically accurate global illumination may be unnecessary in most contexts. Tabellion and Lamorlette [143] found that even in visually demanding applications, such as feature films, single bounce indirect illumination provides plausible lighting. Accepting this limitation avoids tracing complex light paths that add little to a scene and dramatically simplifies the rendering equation.

Section 2.10.1 describes a single bounce approach that uses an augmented shadow map to either gather during a final deferred render pass [22] or scatter indirect illumination via splatting [23]. Both techniques build on the idea of instant radiosity [75], where pixels in the shadow map represent virtual point lights (VPLs) used as secondary light sources for computing indirect lighting. This approach effectively reformulates the rendering equation from a complex integral over surfaces to a

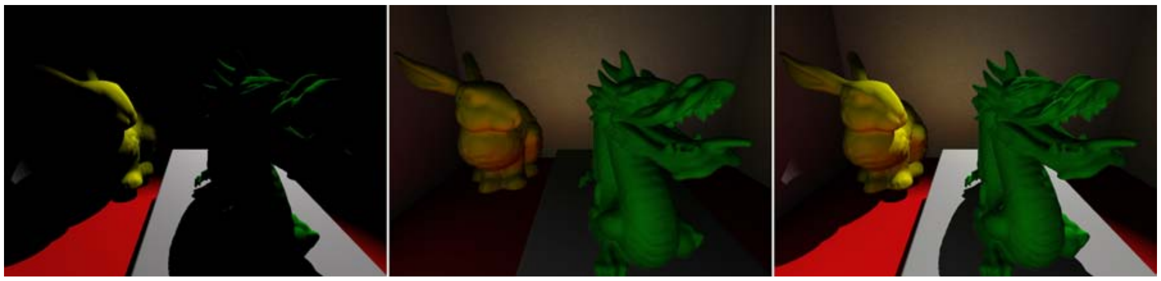


Figure 3.1: Direct light only (left); indirect light generated with our method (center); the combined image (right). This scene is generated at 29 fps with fully dynamic lighting, geometry, and camera.

sum over all texels in the shadow map. The key to achieving performance then lies in reducing the costs of this summation.

This chapter proposes a novel multiresolution splatting technique that reduces costs for RSM-based indirect illumination. Previous techniques either gathered light from a subset of the virtual point lights or splatted light into limited regions. Our approach instead recognizes that each virtual light potentially affects the whole scene, but due to the low-frequency nature of indirect illumination many pixels receive radiance quite similar to their neighbors and can be processed as a group. This idea is similar to hierarchical radiosity approaches [47], but instead works in image-space. We divide the image into regions with similar indirect illumination, which we call subsplats. When indirect light from VPLs is splatted into our multiresolution buffer, each subsplat is rendered as a single fragment into an appropriate layer of the the buffer. Effectively, we use a piecewise-constant illumination approximation inside subsplats and reduce fillrate by rendering subsplats at varying resolutions (rather than always splatting into an image-resolution buffer). The buffer layers are upsampled and additively blended, and a new interpolation technique removes discretization artifacts from the final indirect illumination.

Our most general formulation defines different subsplats (i.e., regions of near-constant indirect illumination) for each virtual point light, allowing the rendering of arbitrary BRDFs. Recomputing subsplats for each VPL quickly becomes the bottleneck, so we propose an alternative approach using the same set of subsplats for an entire frame. This significantly improves performance, though only diffuse materials are properly handled.

Like other RSM approaches, our technique does not consider visibility, and therefore does not converge to correct single bounce indirect illumination. In spite of this limitation, our method yields plausible results at reasonable framerates. In many applications, the lack of strict physical accuracy is an acceptable compromise.

3.2 Algorithm

Like Dachsbacher and Stamminger's algorithm [23] and other splatting approaches, we splat illumination from each virtual point light onto the scene. Naive splatting approaches render one splat for each VPL into a single resolution buffer. Choosing the optimal resolution for this buffer is nontrivial. Indirect lighting from a point light generally changes quite smoothly as the $1/r^2$ falloff gradually goes to zero; even a coarse buffer often suffices to capture this slowly-changing illumination. Because splats are rendered in eye-space, however, high-frequency normal variations and

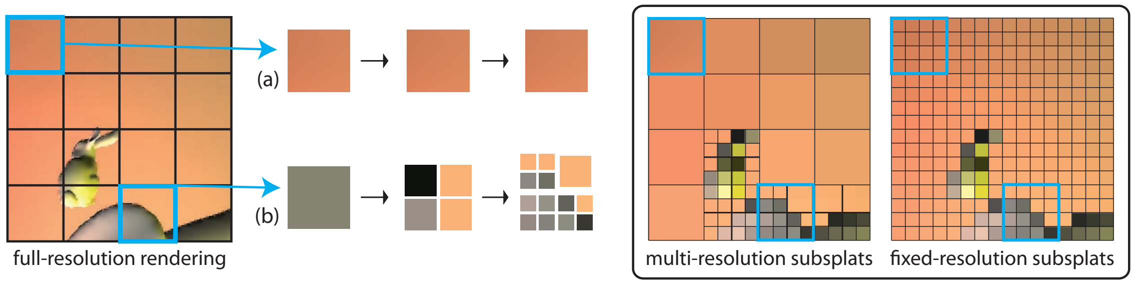

(a)
(b)
full-resolution rendering
multi-resolution subsplats
fixed-resolution subsplats

Figure 3.2: A full-screen quad divided into single-texel subsplats, each of which are then adaptively refined twice. Here, the first subsplat (a) remains at its current resolution after each refinement pass. The second subsplat (b) is refined to a higher resolution. Three of the resulting subsplats are further refined, while the fourth remains at its current resolution. Summation of the multiresolution subsplats (lower right) and for comparison, a fixed-resolution summation where *all* subsplats are refined to the highest resolution. Multiresolution refinement allows comparable quality with many fewer subsplats.

depth discontinuities can introduce rapid changes into the illumination that cannot be adequately captured with a coarse buffer. While this problem can be eliminated by simply rendering the splats at a higher resolution, doing so imposes much higher fillrate requirements and thus negatively impacts performance.

Instead of splatting into a single-resolution buffer, we propose a multiresolution approach. We begin with a full-screen quad for each VPL. This quad is divided into a set of primitives called *subsplats*: each subsplat covers just a single texel at some resolution, though its image space area may contain up to several thousand pixels. The initial set of subsplats is produced at very low resolution: 16^2 or even 8^2. We then adaptively refine the subsplats, subdividing those that are too coarse. After repeated refinements, the final set of subsplats is rendered, with some subsplats output to a 16^2 buffer, some refined and output to a 32^2 or 64^2 buffer, and some refined all the way to the final output image. The"splat" is the summation of this set of multiresolution subsplats. Figure 3.2 illustrates this concept, depicting the results as two different subsplats are refined twice.

This idea has roots in various previous techniques. It could be seen as a variant of hierarchical radiosity [47], where patches are chosen based upon image-space rather than object-space constraints. Radiance caching [38] typically focuses illumination samples near edges, and Tole et al. [147] rely on image-space criteria to better select cache samples for an interactive render. Mitchell [95] describes a method of reconstructing an image from nonuniform samples.

Ultimately, our algorithm simply allows splatting-based techniques to reduce fillrate costs by avoiding redundant computations, grouping them, and rendering to the coarsest buffer allowable for each group. In our algorithm each subsplat covers a single texel, though that texel might lie in a low resolution buffer and thus affect hundreds of pixels in the final image.

The rest of this section describes, in greater detail, the steps of our algorithm. A quick breakdown follows:

1. Compute reflective shadow map and direct light,
2. Select VPLs used to splat indirect illumination,
3. Generate mipmap to detect discontinuities,
4. Create and iteratively refine the list of subsplats,
5. Render to multi-resolution illumination buffer,

6. Upscale and combine buffer for total indirect light,

7. Add direct and indirect light for final result.

Steps 1 and 2 are described in Section 3.2.1, steps 3 through 5 are described in Sections 3.2.2 through 3.2.4, and steps 6 and 7 are described in Section 3.2.5.

3.2.1 RSM and VPL Creation

We begin by generating a reflective shadow map [23] by rendering from the light, storing world-space position, distance from the light, surface normal, and reflected flux for each texel (for an example, see Figure 2.1). Next, we render from the eye, computing only direct light with shadow mapping. During this step, we also generate a G-buffer [125] containing data needed for deferred shading (i.e., world-space position, normal, and distance from the eye).

We then sample the reflective shadow map to create VPLs. In our implementation, we always use the same fixed grid of RSM sampling locations to create VPLs, without regard to the content of the RSM. A flux-based importance sampling or quasi-random sampling may ultimately give better quality.

3.2.2 Min-Max Mipmap Creation

To correctly refine subsplats, we need to identify image-space discontinuities. Toward this end, we make use of the min-max mipmap, which is similar to a subdivided quad-tree [126]. Each layer in such a mipmap stores both the maximum and the minimum (instead of the average) of all corresponding texels from finer resolutions in the map. Sampling any texel of a min-max mipmap gives the largest and smallest values in that texel's image space region. Recent uses of min-max mipmaps include the rendering of soft shadows [44], geometry intersection [13], and dynamic height field rendering [146].

To best detect depth and normal discontinuities, we investigated several different types of min-max mipmaps, described in the following sections.

3.2.2.1 Min-Max Mipmaps for Depth Discontinuities

We begin by describing the construction of a conventional min-max mipmap using linear depth values. We start with the full-resolution linear depth buffer (computed in Section 3.2.1) and run the mipmap generation process. We halve the resolution at each step, computing for each output element the maximum and minimum values of the four input elements. When completed, sampling a texel within the min-max mipmap gives us the minimum and maximum depth values in that texel's image-space area. This allows efficient detection of depth discontinuities: if the difference between these values is greater than a threshold, then we consider a depth discontinuity to exist within that texel's region.

This method conservatively detects discontinuities, causing excess subsplat subdivision when a surface is viewed at a steep angle. In this case, depth differences along smooth surfaces viewed obliquely will be incorrectly treated as discontinuities. To address this, we investigated the use of a depth *derivative* to detect discontinuities. We compute a screen-space depth derivative, calculating $\sqrt{(dz/dx)^2 + (dz/dy)^2}$ for each texel at the highest resolution. When computing each lower-resolution mipmap texel, we select only the largest of the four input texels, generating a max-mipmap rather than a min-max mipmap. When refining subsplats, an area contains a discontinuity if the depth derivative for that region is greater than a threshold.

3.2.2.2 Min-Max Mipmaps for Normal Discontinuities

Detecting surface normal discontinuities using a min-max mipmap is less straightforward. As the goal is to detect significant differences in surface normal orientation, we note that surface normals with significantly different orientations vary significantly in at least one component. Therefore, we generate three sets of min-max mipmaps, one for each component of the unit surface normal. If the difference between the max and min values of *any* of the normal components exceeds a threshold, we consider it a normal discontinuity. In practice this method works well for detecting sharp surface features.

Because we generate three separate min-max mipmaps, this approach consumes excess memory. As an alternative, we experimented with detecting surface normal discontinuities based on variations in surface curvature. Given the normals $\vec{N_1}$ and $\vec{N_2}$ of two neighboring texels, the curvature κ of the surface between them can be estimated by: [35]

$$\kappa = 2 * \sin\left[\frac{\arccos(\vec{N_1} \cdot \vec{N_2})}{2}\right] \tag{3.1}$$

Similar to the depth derivative, we compute surface curvature for each pair

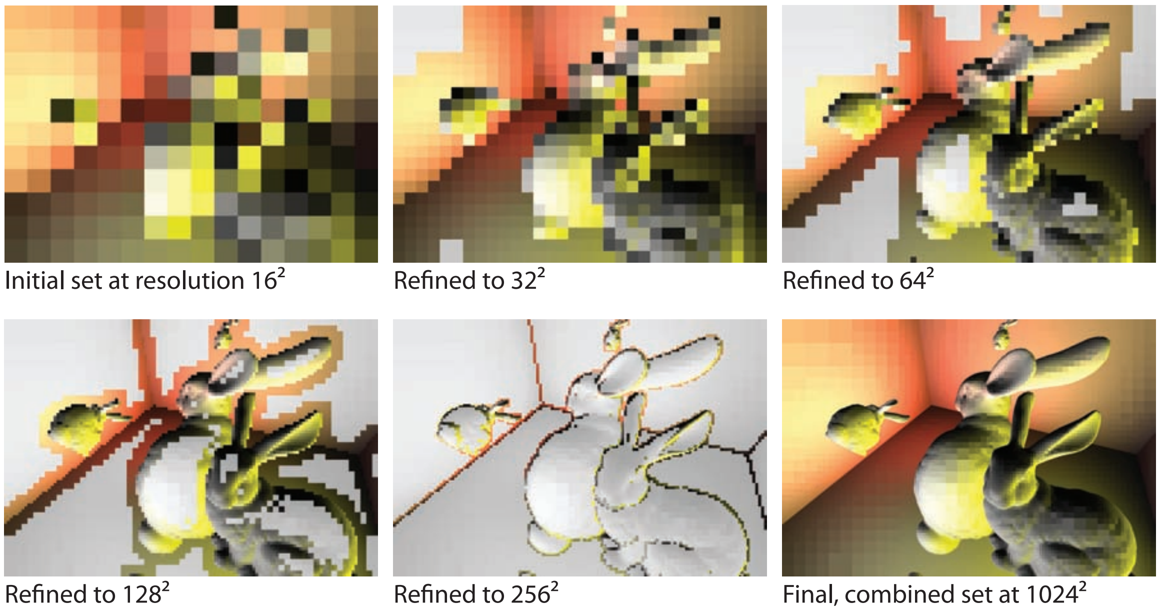


Figure 3.3: Subsplat refinement occurs in areas with depth or normal discontinuities. In this case, we demonstrate how an initial set of subsplats at 16^2 might be refined four times. For clarity, the set of unrefined subsplats at each step is displayed in grayscale. The combined result at 1024^2, after 6 refinement passes (lower right).

of neighboring texels at the highest-resolution mipmap level, and choose the largest input at each additional step. When detecting discontinuities, we sample the min-max mipmap and compare the largest curvature value to a threshold to determine whether a significant change exists in the surface within that texel.

3.2.3 Refining the List of Subsplats

We begin with a very coarse set of subsplats at the coarsest resolution in our *illumination buffer*, the multi-resolution image used to accumulate our indirect illumination. This coarse set is rarely sufficient to represent indirect illumination for the entire image at a reasonable level of quality. To generate an adequate set of subsplats, many of them must be refined to higher resolutions. In our implementation, we use a 16^2 buffer for the coarsest resolution; we therefore start with 256 subsplats.

During each refinement step, each subsplat can either be rendered as a point at a coarse resolution or refined into four new subsplats corresponding to the next layer in the illumination buffer. Since these subdivided subsplats each represent a fragment in a higher resolution layer, each refinement quadruples the required fillrate. Thus, the key to performance is determining when to refine a subsplat and when a coarser one suffices.

In general indirect illumination changes slowly, based upon a distance-squared falloff from the light and cosine falloffs dependent on surface patch orientation. In simple scenes, coarse sampling and bilinear interpolation give plausible lighting. However, complex models create depth discontinuities along silhouettes and normal discontinuities along creases that introduce rapid changes to the indirect illumination seen by a viewer.

We detect these discontinuities by sampling the min-max mipmaps. Sampling these mipmaps at the subsplat's resolution allows us to detect significant depth or normal variations within the subsplat. If the difference between the max and min depth values exceeds a threshold, or the difference in any of the normal components exceed a similar threshold, then we consider a discontinuity to exist within that subsplat.

If a discontinuity exists, we subdivide the subsplat into four higher-resolution subsplats (see Figure 3.3). During each pass, we iteratively refine the subsplats, subdividing some and not others:

```
SubsplatList inputList;
SubsplatList outputList;

for all (subsplats s ∈ inputList) do
  if ( ContainsDepthDiscontinuity( s ) or
      ContainsNormalDiscontinuity( s ) ) then
    outputList.Add( s.TopLeftChild() );
    outputList.Add( s.TopRightChild() );
    outputList.Add( s.BottomRightChild() );
    outputList.Add( s.BottomLeftChild() );
  else
    outputList.Add( s );
  end if
end for
```

Each refinement pass doubles the resolution of the indirect illumination. If the resulting set of subsplats has reached the desired resolution, they can be rendered. Otherwise, the set of subsplats is used as the input to a subsequent pass.

3.2.3.1 Refinement for Arbitrary BRDFs

Complex lighting models often give rise to regions of focused light, which must be rendered at high resolution to be properly reproduced. Refinement based solely on depth and surface normal discontinuities is insufficient in these cases: if a highlight occurs on a surface without discontinuities, the refinement of that surface will likely be too coarse to adequately reproduce them.

To address this, we explored a variation of our method that uses discontinuities in *illumination*, rather than depth or surface normal, to guide the refinement process. When evaluating whether to subdivide a subsplat, we compute the indirect illumination in each of the subsplat's quadrants and compare the results. We refine the subsplat if any of the samples differ significantly:

```
SubsplatList inputList;
SubsplatList outputList;

for all (subsplats s ∈ inputList) do
   l_1 = IlluminationAtPoint( s.TopLeft() )
   l_2 = IlluminationAtPoint( s.TopRight() )
   l_3 = IlluminationAtPoint( s.BottomRight() )
   l_4 = IlluminationAtPoint( s.BottomLeft() )

   if ( SamplesDiffer( l_1, l_2, l_3, l_4 ) ) ) then
      outputList.Add( s.TopLeftChildPoint() );
      outputList.Add( s.TopRightChildPoint() );
      outputList.Add( s.BottomRightChildPoint() );
      outputList.Add( s.BottomLeftChildPoint() );
   else
      outputList.Add( s );
   end if
end for
```

Many ways exist to detect a "significant difference" in illumination samples. In our implementation, we achieve reasonable results by computing the maximum and minimum values of each component of the inputs, and comparing their differences to a threshold **T**:

```
boolean SamplesDiffer( l_1, l_2, l_3, l_4 )

// component-wise max/min of the input samples
maxValues = max( max( max( l_1, l_2 ), l_3, l_4 );
minValues = min( min( min( l_1, l_2 ), l_3, l_4 );

if ( (maxValues.r - minValues.r) > T or
     (maxValues.g - minValues.g) > T or
     (maxValues.b - minValues.b) > T ) then
   return true;
else
   return false;
end if
```

Because this style of refinement depends on point samples to detect illumination discontinuities, very sharp highlights may go undetected, and areas that contain them may not be sufficiently refined. Thus, this method may not yield sufficient quality using BRDFs that often exhibit high-frequency illumination detail (i.e., mirror-like BRDFs). For many BRDFs, though, refining subsplats in this manner effectively captures illumination behavior, allowing the plausible reproduction of lighting features and enabling our method to be used with complex materials (such as the Phong material shown in Figure 3.4). The performance implications of this variation of our method are explored in Section 3.3.2.

3.2.4 Rendering Indirect Illumination

After iterative refinement, we have a large list of subsplats. Our implementation requires a three-tuple to store each subsplat, with one value specifying the subsplat's output resolution, and two values specifying its screen-space location.

During rendering, a vertex shader positions each subsplat in the correct layer of the illumination buffer, a fragment shader computes indirect illumination for each subsplat, and additive blending accumulates all contributions. For each subsplat, our indirect illumination is:

$$I(x_s, x_l) = \rho_l(\vec{L}, \vec{V_{ls}})\rho_s(\vec{V_{sl}}, \vec{E})\Phi_l \frac{\langle \vec{V_{ls}}, \vec{N_l}\rangle\langle \vec{V_{sl}}, \vec{N_s}\rangle}{\pi|\vec{V_{ls}}|^2}, \tag{3.2}$$

where x_s and x_l are the shaded point and the virtual light point, $\vec{N_s}$ and $\vec{N_l}$ are the surface normals at x_s and x_l, Φ_l is the flux to the VPL at x_l, ρ_s and ρ_l are the BRDFs at x_s and x_l, $\vec{L}$ is the vector from x_s to the light, $\vec{E}$ is the vector from x_s to the eye, and $\vec{V_{ls}}$ and $\vec{V_{sl}}$ are respectively the vectors from x_l to x_s and from x_s to x_l. These

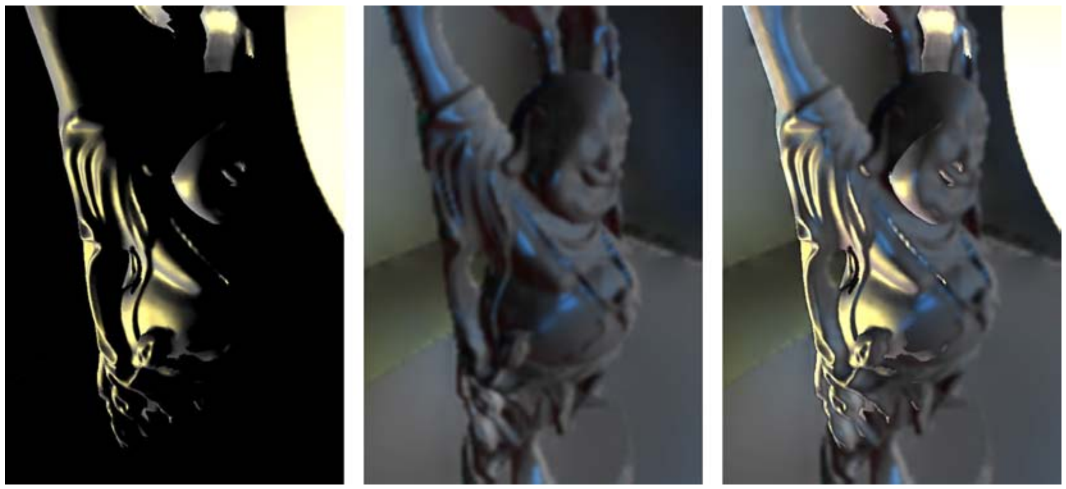


Figure 3.4: Indirect lighting from a Buddha with a Phong BRDF; here, subsplats are refined separately for each VPL. To emphasize the contribution, a blue tint was added to indirect light reflected by the Buddha. Direct lighting (left), indirect lighting (middle), and combined result (right). Indirect illumination rendered at 256^2 at 23 fps.

values are retrieved from the appropriate location either in the RSM or the G-buffer. Here, $\langle \vec{A}, \vec{B} \rangle = \max(\vec{A} \cdot \vec{B}, 0)$.

After rendering all subsplats, the illumination buffer contains all indirect illumination split into disjoint components at various resolutions. Naively summing the layer contributions gives a blocky representation of total indirect light (see Figure 3.5) that requires interpolation.

3.2.5 Upsampling and Combination

Figure 3.5 demonstrates the artifacts from naive methods of combining multiresolution illumination: blocky illumination when using nearest neighbor upsampling, and strange multiresolution haloing and ringing when using bilinear interpolation. Clearly, neither is acceptable.

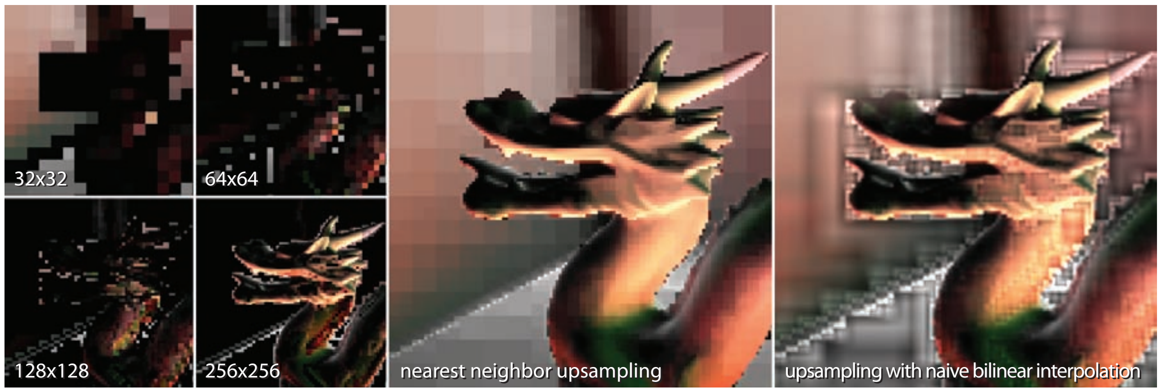

32x32
64x64
128x128
256x256
nearest neighbor upsampling
upsampling with naive bilinear interpolation

Figure 3.5: Each illumination buffer level contains pieces of the indirect illumination at a different resolution. Artifacts from combining the illumination buffers using nearest neighbor upsampling (middle) and naive bilinear interpolation (right).

The problem lies in the complex structure of the illumination buffer. Each texel contains either all of the illumination for that location, or none at all. Interpolation between a texel containing all of its relevant light and one containing no illumination makes little sense, as it spreads energy from texels containing energy to those deemed too coarse or too fine. Roles may be reversed at other levels in the illumination buffer; multiresolution linear interpolation leads to ringing and haloing arising from the varying regions of support for the interpolation filter at multiple scales.

We address this with a unique upsampling scheme. Upscaling progresses from coarsest to finest layers in the illumination buffer. At each resolution, texels containing illumination information are linearly interpolated with neighboring texels of the same resolution, whether they were originally rendered at that resolution or upsampled from a lower resolution during a previous pass:

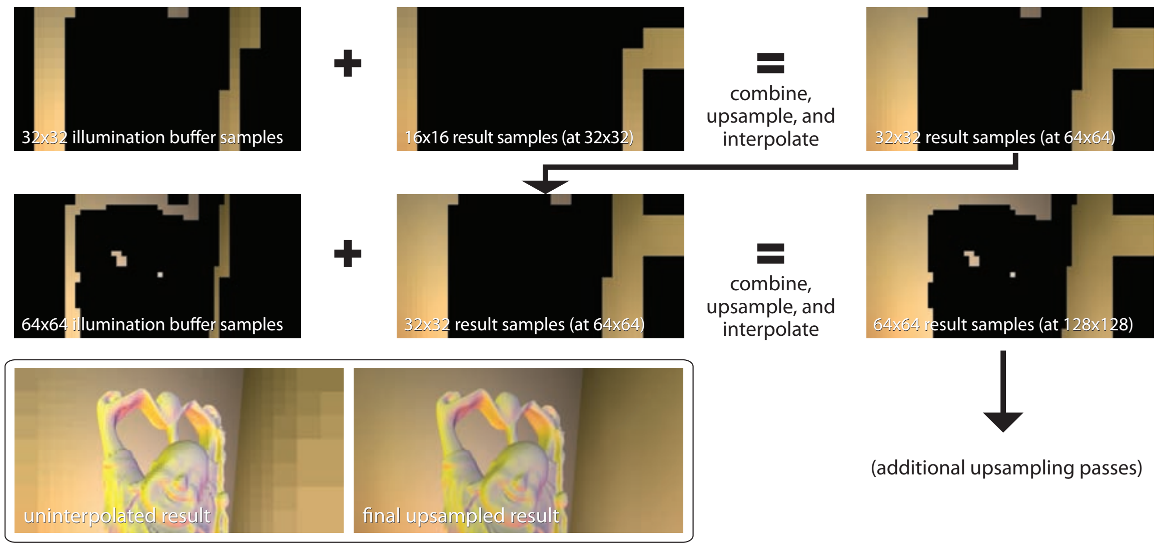

32x32 illumination buffer samples
+
16x16 result samples (at 32x32)
=
combine, upsample, and interpolate
32x32 result samples (at 64x64)
64x64 illumination buffer samples
+
32x32 result samples (at 64x64)
=
combine, upsample, and interpolate
64x64 result samples (at 128x128)
uninterpolated result
final upsampled result
(additional upsampling passes)

Figure 3.6: Two upsampling passes, from 32^2 to 64^2 to 128^2. At each level, missing samples are combined with interpolated data from previous levels. The result is interpolated to higher resolution, and used as the input for the next upsampling pass. The non-interpolated summation (lower left), and the final interpolated and upsampled result (lower middle).

```
for all layers l ∈ illumination buffer, coarsest to finest do
   for all (texels t ∈ l) do

      // 3x3 input texels both from the illumination
      // buffer and from previous passes
      inputTexels[9];
      renderedTexels[9];
      interpolationWeights[9];

      outputTexel = EmptyTexel();
      totalWeight = 0;

      for (i=0 to 8) do
         if HasData(inputTexels[i]) or
            HasData(renderedTexels[i]) then
            totalWeight += interpolationWeights[i];
            outputTexel += interpolationWeights[i] *
                 (renderedTexels[i] + inputTexels[i]);
         end if
      end for

      outputTexel /= totalWeight;
   end for
end for
```

After each layer has been upsampled, the result then becomes the input for the next resolution. To avoid the haloing and ringing shown in Figure 3.5, each upsampling pass only outputs interpolated texels in locations where the illumination buffer already contained data for that resolution: energy is never pulled from empty areas nor spread into them, and all texels that contained no energy at the start of each pass remain empty. Figure 3.6 demonstrates this process graphically through two upsampling steps.

After processing all the layers, we have a single combined and upsampled image that varies smoothly, without ringing artifacts. Furthermore, this method does not spread energy across major discontinuities. Our refinement process ensures that areas with these discontinuities are refined into high resolution subsplats. Because each texel is interpolated only with texels of equal resolution and those upsampled from coarser resolutions, energy stays on the correct side of a discontinuity.

3.3 Implementation

We implemented our method using OpenGL and GLSL on a machine with a dual-core 3GHz Pentium 4 and a GeForce GTX 280. Our implementation uses OpenGL's geometry shader and transform feedback extensions. We use a geometry shader to subdivide subsplats, selectively either dividing each input into four new outputs, or passing it through unchanged; the resulting subsplats are output to a vertex buffer object. We repeat the refinement process until the highest-resolution

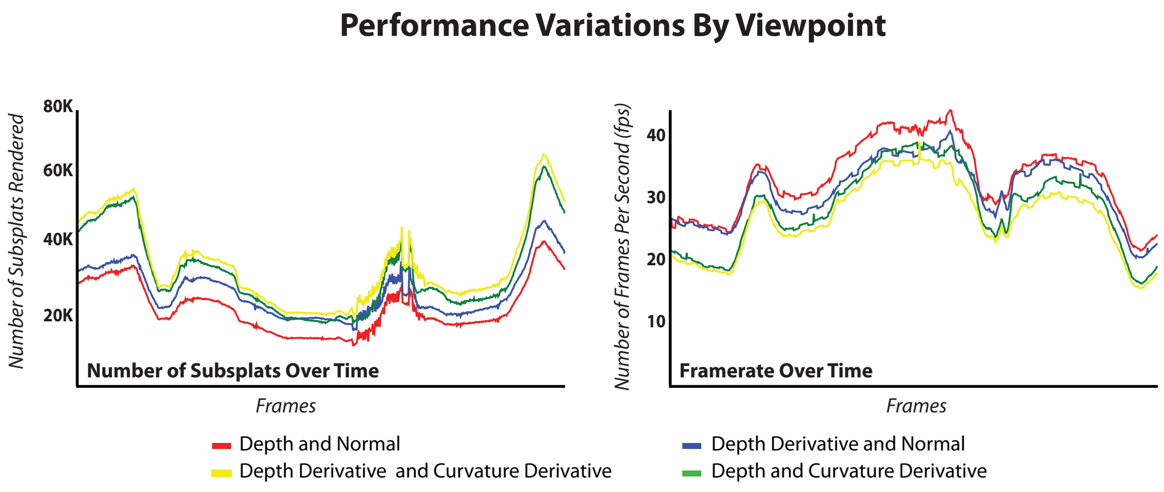


Figure 3.7: Framerates and subsplat counts during a flythrough of the dragon scene. Four different combinations of depth and normal discontinuity detection are illustrated, rendered at 1024^2 with similar visual quality.

subsplats have reached an appropriate resolution.

All images in this paper were generated using a final output resolution of 2048^2, which was downsampled to a 1024^2 window for an antialiased rendering.

3.3.1 Discontinuity Detection

We found that neither depth derivatives nor surface curvature offer a substantive advantage in detecting discontinuities. Min-max mipmaps constructed with depth derivatives sometimes produce a smaller set of subsplats than those built with direct linear depth values. However, they are slightly more costly to construct due to the additional samples required to calculate the derivative. In practice, using a depth derivative min-max mipmap is slightly slower than using one constructed with linear depth values for a similar level of quality.

Min-max mipmaps built using surface curvature values do not fare as well.

In particular, detection of gradually curving surfaces is problematic using curvature mipmaps: even if a great deal of change occurs within the complete image-space area of a low-resolution texel, curvature between neighboring high-resolution texels may be minimal. To detect these gradual changes, a very low threshold is required, leading to unnecessary subdivisions and a corresponding performance hit.

Figure 3.7 demonstrates the relative performance of these discontinuity detection methods as the scene viewpoint changes. Although each method was configured to produce results of similar quality, an absolute qualitative comparison is difficult. In particular, the different normal discontinuity detection methods each accentuate certain features. For example, when reproducing a gently curved surface with the surface curvature method, a very low threshold must be used to match the quality of the surface normal method. However, this also results in sharp, detailed reproductions of small surface features that would be smoothed over by the surface normal method at anything but a very low threshold. Additionally, different viewpoints can yield results of varying quality depending on the detection method used.

3.3.2 VPLs and Subsplat Refinement

Our initial implementation refined a separate list of subsplats for each VPL as described in Section 3.2.3.1, computing illumination during refinement to guide the splitting process. With this approach, generating and refining a list of subsplats for each VPL is the most expensive part of our method, often by a large margin. Refining a single subsplat into four new subsplats using the method outlined in Section 3.2.3.1

Table 3.1: Refinement costs for a Phong Buddha

Indirect Resolution	16^2 VPLs	32^2 VPLs	64^2 VPLs	128^2 VPLs
16^2	65.5K 0.0ms 95 fps	262.1K 0.0ms 60 fps	1.05M 0.0ms 25 fps	4.19M 0.0ms 7.4 fps
32^2	120.5K 3.9ms 65 fps	484.9K 28.0ms 21 fps	1.94M 49.4ms 13 fps	7.20M 201.6ms 2.83 fps
64^2	220.4K 11.7ms 52 fps	893.9K 37.1ms 14 fps	3.57M 86.3ms 6.4 fps	N/A
128^2	421.8K 17.7ms 39 fps	1.73M 37.1ms 14 fps	6.66M 149.0ms 4.0 fps	N/A
256^2	852.0K 35.3ms 27 fps	3.22M 60.1ms 8.9 fps	N/A	N/A

Note: Subsplats are refined separately for each VPL. Each cell contains the number of refined subsplats, refinement time, and the resulting framerate. "N/A" entries required excessive memory or execution time.

takes roughly 30 nanoseconds, and rendering it takes a comparable amount of time. At these speeds, hundreds of thousands of subsplats can be refined and rendered while easily maintaining interactive rates. However, with each VPL generating tens of thousands of subsplats, the refinement and rendering costs quickly become unmanageable. As Table 3.1 illustrates, subsplat counts can easily climb into the millions, even at low VPL counts and relatively low resolutions. Additionally, because this method requires the calculation of illumination during refinement, more expensive BRDFs increase these costs yet further.

For diffuse scenes we found that splats were being split almost identically for each VPL. Noting this, we implemented an alternate approach that performs splat

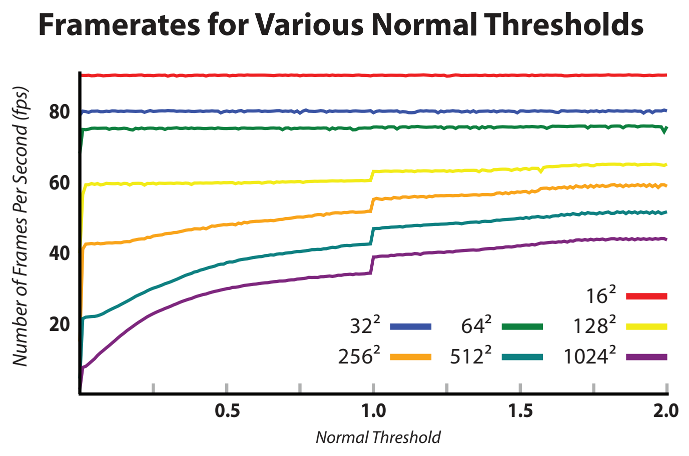


Figure 3.8: Framerates of our method in the dragon scene with an increasing normal refinement threshold, at various resolutions of indirect illumination.

refinement just once per frame, avoiding illumination calculations and refining solely based on normal and depth discontinuities. All VPLs then reuse the same subsplats for rendering. While there is no guarantee that normal and depth discontinuities are completely predictive of areas that will need refinement, we found that in diffuse scenes, this approach yields a significant increase in performance while maintaining similar image quality. Unless otherwise stated, all images and results in this paper are generated using this approach.

3.3.3 Adding Surface Detail

The number of subsplats generated has a significant impact on the performance of our technique. Since this number is heavily influenced by thresholds within the refinement process, these thresholds can serve as a parameter for tuning performance. This is particularly true of the threshold used to detect normal discontinuities, which ranges from [0..2]; a higher threshold allows faster framerates, at the expense of fine surface details. Figure 3.8 illustrates this effect for one viewpoint in the dragon scene. While having no effect at low resolutions, at high resolutions the normal threshold substantially affects the overall framerate. This parameter's effect on image quality can be observed in Figure 3.9: with high normal thresholds, insignificant surface features of the Buddha do not trigger refinement. These areas are then rendered at low resolution, effectively blurring them out.

One solution is to approximate the missing detail, modulating the indirect illumination using the surface normal's alignment towards the camera. For each image-space pixel of illumination c, with view vector $\vec{V}$ and surface normal $\vec{N}$ at the same location, and a parameter α (ranging from [0..1]) controlling the effect's intensity:

$$c_{out} = c * (\alpha * \langle \vec{V}, \vec{N} \rangle + (1 - \alpha)) \tag{3.3}$$

Ideally, instead of modulating the indirect illumination with the viewing vector, we would introduce cosine falloff by modulating with the vector to each VPL. However, once the indirect illumination has been splatted into the illumination buffer, VPL locations are no longer easily accessible. While our method is somewhat ad-hoc, it gives visually plausible results and incurs almost no runtime cost. This allows the use of higher normal thresholds, which in turn leads to faster framerates. This approximation may cause some areas to appear overly dark, which may be objectionable in some circumstances. The third panel of Figure 3.9 illustrates the effect of replacing

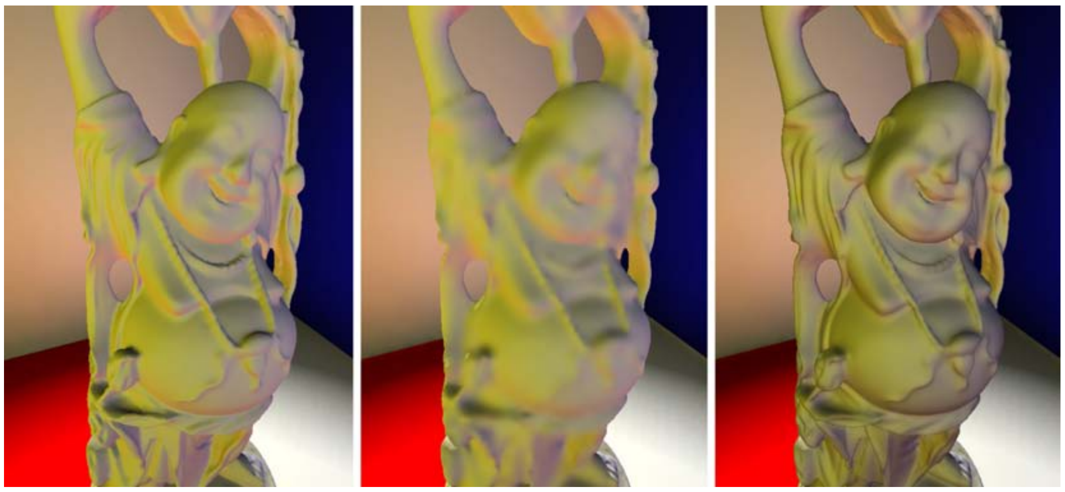


Figure 3.9: The Buddha model rendered at 1024^2. The left image, rendered with a normal splitting threshold of 0.25, runs at 8 fps; the two images on the right use a threshold of 1.4 and run at 32 fps. The rightmost image approximates surface detail with the technique described in Section 3.3.3, using $\alpha = 0.5$.

surface detail using this method.

3.4 Results and Discussion

Like other splatting approaches for indirect illumination, the performance bottleneck in our technique is the cost of rendering the indirect subsplats. The resolution of each subsplat does not matter: in our approach, subsplats each cover a single texel, so the cost of rendering them is independent of resolution. This also allows us to render subsplats as points, rather than triangle meshes, quads, or point sprites. We achieved a 5% speed increase simply by switching from quads to points.

As demonstrated by the data in Table 3.2, our performance has little dependency on geometric scene complexity: when rendering indirect illumination at 1024^2, performance is similar whether the scene contains a single teapot or many complex

Table 3.2: Framerates at various resolutions of indirect illumination

Indirect Resolution	Simple Teapot (6.3K triangles)	Spinning Buddha (250K triangles)	Dragon & Bunny (405K triangles)	Flying Bunnies (417K triangles)
128^2	68 fps	60 fps	59 fps	52 fps
256^2	50 fps	42 fps	42 fps	41 fps
512^2	33 fps	30 fps	28 fps	31 fps
1024^2	22 fps	24 fps	21 fps	25 fps

models. Geometry is rasterized twice, one rendering from the light and one from the eye. Only at fairly low resolutions does the geometric complexity of a scene have a significant effect on performance. However, our method is sensitive to the *visual complexity* of a scene. When looking at a flat wall, few subsplat refinements are required; complex geometry requires refinement around edges, creases, and crevices. High frequency random geometry may require a uniformly dense refinement, where naive splatting would outperform our multiresolution approach.

As the viewpoint moves within the scene, changing the visual complexity of the current view, total subsplat count can vary up to 300%. Since splatting remains the bottleneck in our method, overall performance is strongly tied to the number of subsplats rendered, an effect that can be easily observed in Figure 3.7.

As with other splatting techniques, we ignore visibility considerations when accumulating indirect illumination, which can result in an overly bright image. VPLs on a scene's ceiling illuminate not only the surface of a table, but also the floor below the table. One method of addressing this might be to approximate visibility independently, e.g., using ambient occlusion, and modulate the results. In the future, we also hope to explore multiresolution techniques that account for visibility.

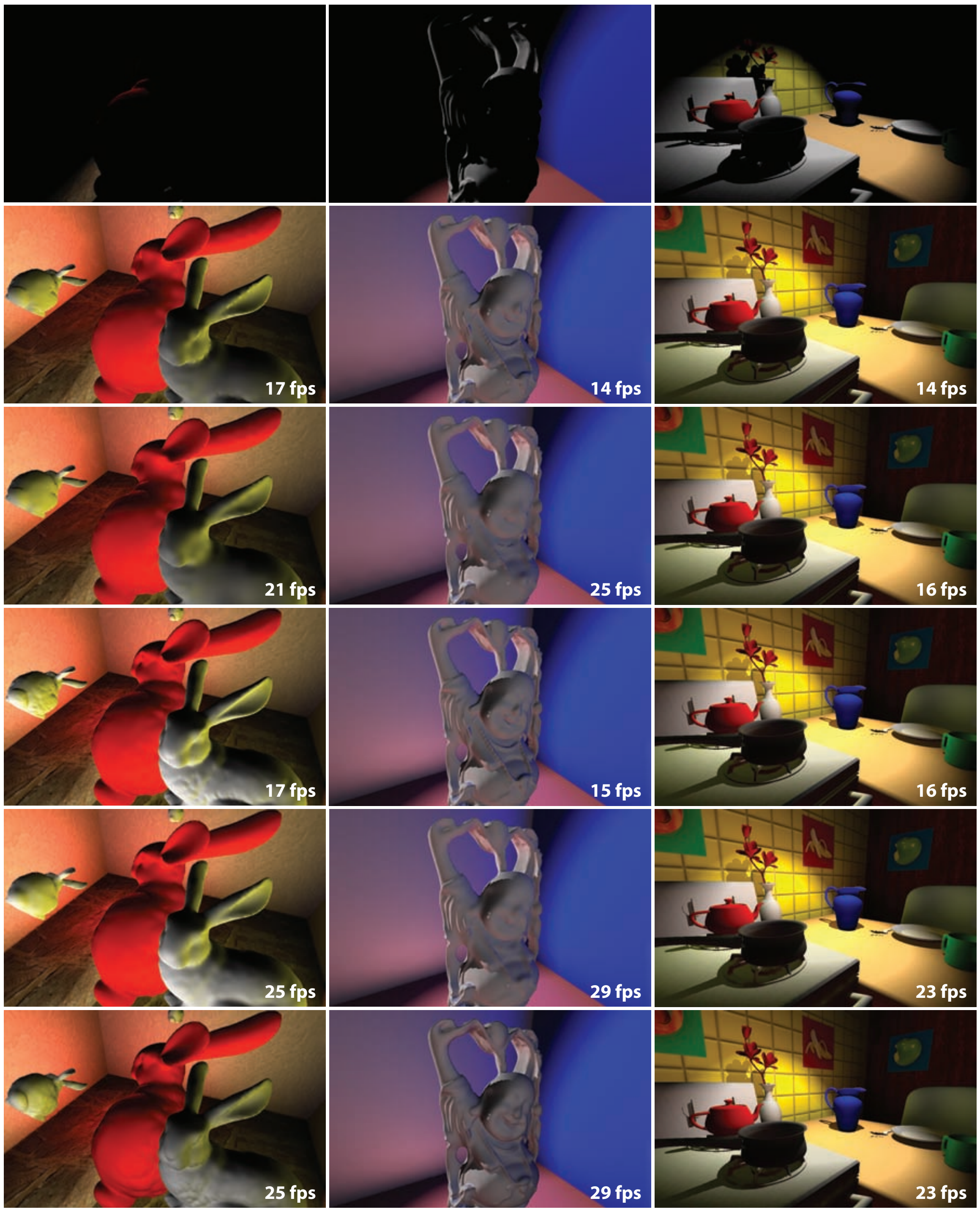


Figure 3.10: Example scenes using our technique, with indirect illumination rendered at 1024^2. These scenes were rendered with direct illumination only (top row); indirect illumination generated with a curvature min-max mipmap, using threshold 0.1 (second row) and 0.3 (third row); indirect illumination generated with a surface normal min-max mipmap as described in Section 3.2.2, using normal threshold 0.2 (fourth row) and 1.5 (fifth row); using normal threshold 1.5, and replacing surface detail using the method described in Section 3.3.3 (with $\alpha = 0.5$).

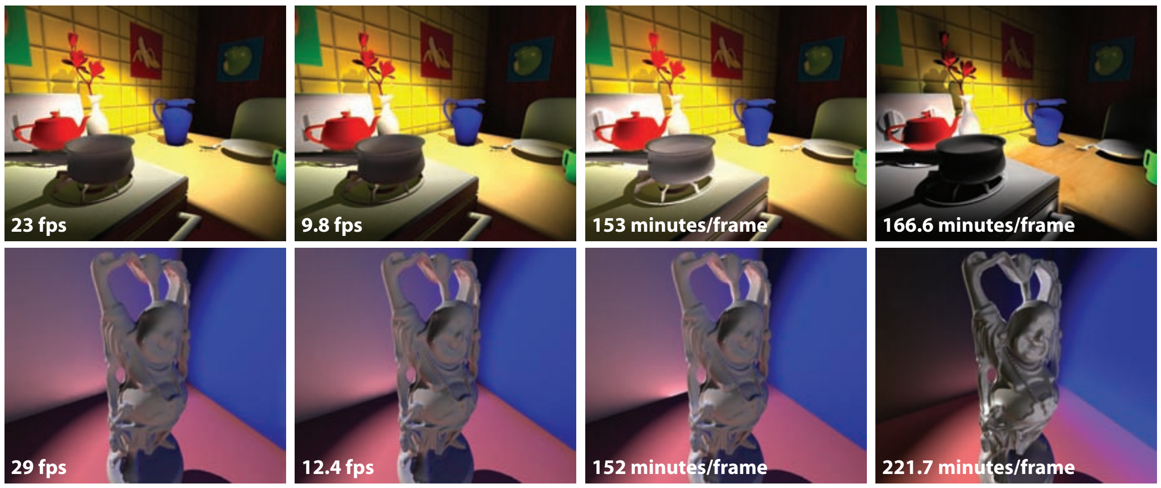


Figure 3.11: Compare our work (top) to reflective shadow mapping (upper middle), with near indistinguishable results. One problem our work shares with RSMs is the need to clamp the minimum r^2 falloff to avoid point singularities. When using every texel of the RSM as a VPL, no clamping is required and generates a brighter image (lower middle). A path traced comparison (bottom), shows the main differences are lack of indirect visibility, clamping the r^2 falloff to avoid singularities, and an approximation that all RSM texels subtend the same solid angle. The solid angle approximation causes our work and RSM implementation to appear dimmer towards the inside of the spotlight and brighter towards the outside.

Figure 3.1 demonstrates the dragon and bunny scene rendered with direct illumination only, and with indirect illumination subdivided to a maximum refinement of 512^2. Figure 3.10 depicts several different scenes under varying rendering parameters, rendered with a maximum refinement level of 1024^2. In the second and third rows, discontinuities were detected using min-max mipmaps built using the surface curvature method described in section 3.3.1, using a low threshold for the images in the second row and a higher threshold for those in the third row. The remaining three rows were rendered with min-max mipmaps built from surface normals, using a low surface normal threshold for the fourth row, and a high threshold for the fifth and

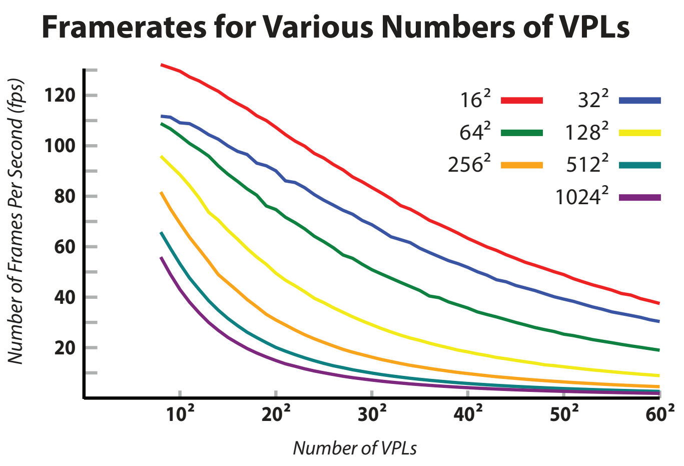


Figure 3.12: Framerates of our method in the dragon scene using increasing numbers of VPLs, at various resolutions of indirect illumination.

sixth rows. The sixth row illustrates the effects of the reintroduction of detail using the ad-hoc method described in Section 3.3.3. The visual appearance of complex objects, especially those close to the camera, depends heavily on refinement parameters. On the other hand, scenes with high geometric complexity, such as the kitchen scene in the right-most column, often suffer little degradation in quality even when rendered with higher normal thresholds.

In Figure 3.11, we compare the results of our technique to naive reflective shadow maps and a ground truth image produced by a Monte Carlo path tracer.

Figure 3.12 explores performance on the dragon scene, showing variations due to the number of VPLs and refinement passes. With no refinement, indirect illumination is rendered at 16^2 resolution. Each refinement doubles the resolution (up to 1024^2 after 6 passes).

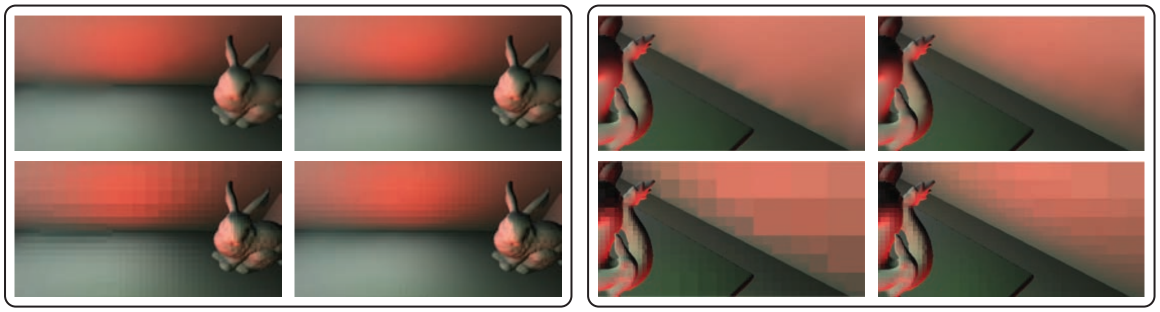


Figure 3.13: Examples of upsampling artifacts, including interpolated (top) and uninterpolated versions (bottom). Left group: artifacts arising from insufficient refinement along coarse mipmap boundaries (left column), and correct refinement (right column). Right group: artifacts from insufficient refinement along diagonals (left column), and correct refinement (right column).

3.4.1 VPL Sampling Artifacts

We regularly sample the reflective shadow maps to choose our virtual lights. While this sampling scheme is simple, it can result in flickering under animation as VPL locations jump on and off surfaces in successive frames. In simple scenes we found that 16^2 to 32^2 VPLs gave smooth results under animation, with minimal flickering. More complicated scenes (in which VPLs jump between surfaces more often) require larger number of VPLs to achieve smooth results. This represents a tradeoff; increasing the number of VPLs reduces performance (as demonstrated by Figure 3.12). A more sophisticated VPL sampling scheme may reduce the number required and minimize flickering.

3.4.2 Upsampling Artifacts

Our upsampling method assumes that adjacent texels in the illumination buffer can be safely interpolated together without visual artifacts. This relies heavily on sufficient subsplat refinement, which in turn depends on the successful detection of discontinuities. Suboptimal behavior of either of these processes results in a set of subsplats that does not adequately represent the scene. Additionally, when discontinuities are present in low-resolution subsplats, energy is blurred across the discontinuity, causing distracting artifacts that flicker and pop under animation as subsplats are re-refined each frame.

Figure 3.13 presents two examples of artifacts arising from inadequate refinement. In the top example, a horizontal edge lies on a mipmap boundary, and is not correctly detected as a discontinuity. The min-max mipmap can be constructed to detect discontinuities such as these by sampling outside the usual neighborhood, allowing refinement when a discontinuity is present not just in the cell being refined, but in any adjacent cell. However, this modification causes an excessive amount of refinement, incurring as much as a 40% speed penalty.

Artifacts also occur when a mipmap cell is situated diagonally to a discontinuity, as shown in the lower half of Figure 3.13. To address these, when evaluating whether to refine a given subsplat, we detect discontinuities not just in the current subsplat, but also in all subsplats situated diagonally. This does result in overrefinement (and a roughly 2–3% speed penalty), but it is necessary to minimize these particularly distracting artifacts.

3.5 Conclusions

This chapter introduced a novel multiresolution splatting technique, and described its application to the rendering of indirect illumination with a reflective shadow map. Our method reduces the cost of rendering indirect illumination by rendering each piece of it at the lowest possible resolution. This allows indirect illumination to be rendered at high resolutions at interactive rates, without artificially restricting each VPL's ability to contribute illumination to the entire scene.

We discussed a variant of our method that works efficiently with diffuse surfaces, and a more general form that allows its use with arbitrary BRDFs. We discussed artifacts that can arise and potential solutions, presented a simple and efficient method of adding plausible detail to indirect illumination, and evaluated the effects of various parameters and configurations on performance.

CHAPTER 4
HIERARCHICAL IMAGE SPACE RADIOSITY FOR INTERACTIVE GLOBAL ILLUMINATION

4.1 Introduction

This chapter introduces *image-space radiosity*, based on our observation that image-space techniques often have advantages over object and world-space techniques for interactive applications. In particular, image-space techniques divorce algorithmic and scene complexities and avoid wasting computations on off-screen portions of the scene. Because we aim for completely dynamic content, complex acceleration structures would be discarded each frame. Image-space techniques lend themselves to trivial quad-tree based hierarchical techniques that leverage hardware-accelerated mipmap creation and contain only relevant geometry (see Figure 4.1).

We use an instant radiosity [75] approach that builds on the work presented in Chapter 3, as well as other recent multiresolution techniques [165], reformulated for higher performance. Combined with a technique similar to Lightcuts [156] and Light Pyramids [77], we propose a one-bounce radiosity algorithm where each fragment gathers illumination from a unique set of image-space light clusters selected to reduce fragment error below a user specified threshold. While we currently ignore visibility for indirect illumination, future work should enable incorporation of visibility approximations, such as imperfect shadow maps [118].

Note that in this chapter, the "subsplats" from Chapter 3 are referred to as "patches", reinforcing our method's similarities to hierarchical radiosity.

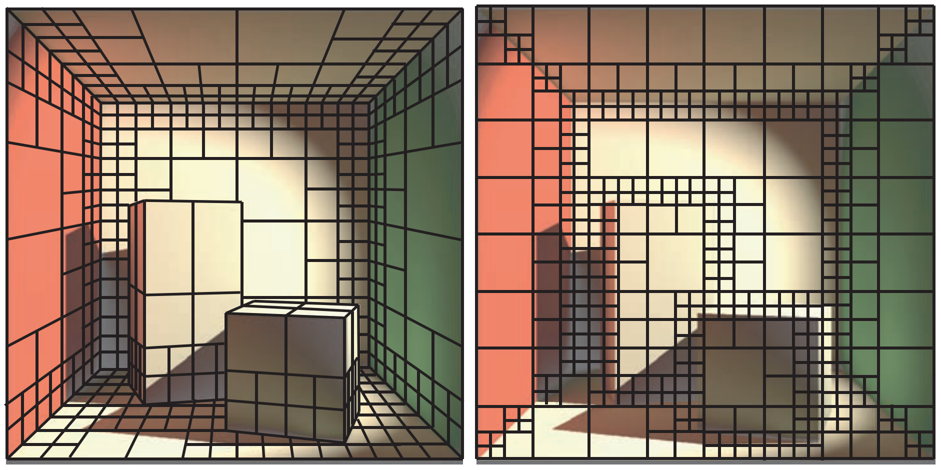


Figure 4.1: Object-space patches (left) compared to image-space patches (right). Object-space patches remain constant through a simulation; image-space patches vary each frame and between eye and light space.

4.2 Image-Space Radiosity

In the Chapter 3, we introduced a novel multiresolution splatting algorithm. This section describes a stencil-based rendering scheme that improves performance of multiresolution splatting by nearly an order of magnitude. We then describe two light-clustering techniques that extend this to image-space radiosity, clustering lights either on a per-frame basis or selecting VPL clusters individually for each fragment.

4.2.1 Stenciled Multiresolution Illumination

Prior multiresolution rendering techniques [77, 102, 165] use a multipass geometry shader to refine a list of primitives and corresponding fragment locations. Unfortunately, outputting variable length computation results into temporary memory and GPU pass overhead leads to relatively poor performance, even though speeds exceed comparable single-resolution techniques.

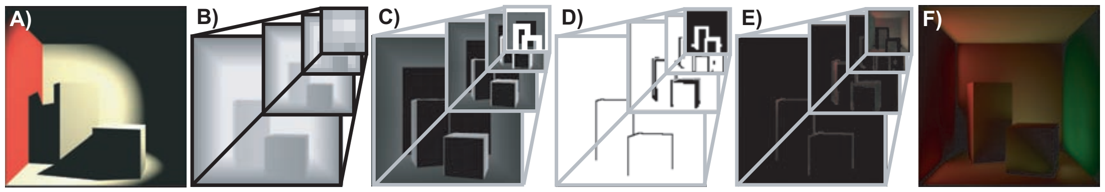


Figure 4.2: Multiresolution illumination splatting begins by drawing (a) just the direct illumination, as seen from the eye. Creation of a (b) multiresolution depth map occurs simultaneous with creation of a (c) depth derivative max mipmap. Locations of high depth or normal discontinuities (found using a similar normal mipmap) require higher resolution detail. Our improved approach first computes a (d) multiresolution stencil buffer that stores relevant fragments at each level of detail, then gathers illumination from VPLs at all valid fragments. The resulting (e) multiresolution illumination buffer is upsampled, interpolated, and combined into the (f) final indirect illumination (brightened for display).

We propose a stencil-based approach that vastly improves performance. Figure 4.3 provides a high level comparison between our approach and prior work [102]. We first replace the costly iterative refinement with a single pass that selects appropriate fragments in a stencil buffer. We then reformulate the final pass to gather indirect illumination rather than splatting illumination. This provides an immediate performance improvement, in addition to enabling the hierarchical VPL selection described in Section 4.2.4.

4.2.1.1 One Pass Stencil Refinement

By observing that GPUs internally use hierarchical z-buffers [42] that enable hardware-controlled early stencil culling, we can replace the software refinement loop depicted in Figure 4.3 with an equivalent, optimized technique that leverages this GPU hardware. Multiresolution splatting [102] identifies the appropriate resolution image-patches, seen in Figure 4.1, as follows:

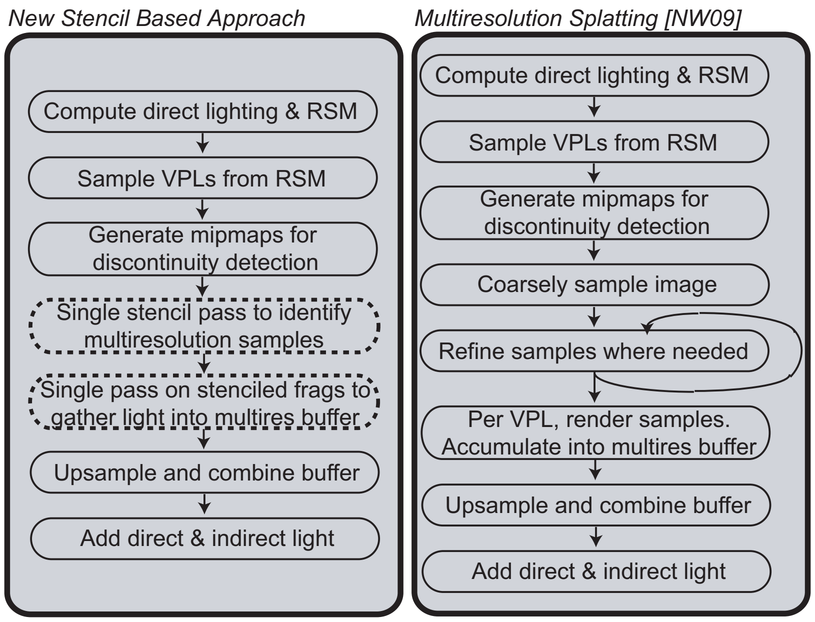
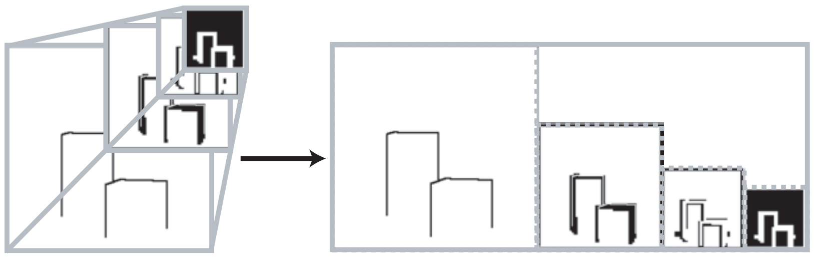

New Stencil Based Approach
Compute direct lighting & RSM
Sample VPLs from RSM
Generate mipmaps for discontinuity detection
Single stencil pass to identify multiresolution samples
Single pass on stenciled frags to gather light into multires buffer
Upsample and combine buffer
Add direct & indirect light
Multiresolution Splatting [NW09]
Compute direct lighting & RSM
Sample VPLs from RSM
Generate mipmaps for discontinuity detection
Coarsely sample image
Refine samples where needed
Per VPL, render samples. Accumulate into multires buffer
Upsample and combine buffer
Add direct & indirect light

Figure 4.3: Our stencil-based multiresolution illumination differs from prior work in two important ways. First, we avoid an iterative multi-pass refinement, costly on current GPUs. Second, we reformulate the final pass as a gather instead of a scatter, which proves more cache-friendly.

```
$patches \leftarrow$ CoarseImageSampling();
for (i=1 to numRefinementPasses) do
    for all ($\mathbf{p} \in patches$) do
        if ( NoDiscontinuity( $\mathbf{p}$ ) ) then
            continue;
        end if
        $patches \leftarrow (patches - \{\mathbf{p}\})$;
        $patches \leftarrow (patches \cup$ SubdivideIntoFour( $\mathbf{p}$ ) );
    end for
end for
```

The resulting patch list is used as a vertex array identifying which fragments in the multiresolution buffer will accumulate light. When splatting, this causes each patch to generate one vertex and one fragment for each VPL.

We simplify this by noting that each refinement pass does not truly depend on prior passes and can be parallelized. First, we flatten our multiresolution buffer into a 2D image:



Then we render a single full-screen quad over this flattened image to set a stencil bit for valid multiresolution fragments:

```
for all (fragments $\mathbf{f} \in$ image) do
  if ( $\forall i$, $\mathbf{f} \notin$ MipmapLevel( $i$ ) ) then
    continue; // Fragment not actually in multires buffer
  end if
  $i \leftarrow$ CurrentMipmapLevel( $\mathbf{f}$ );
  if ( IsDiscontinuity( $\mathbf{f}$, $i$ ) ) then
    continue; // Patch not valid (needs subdivision)
  end if
  if ( NoDiscontinuity( $\mathbf{f}$, $i+1$ ) ) then
    continue; // Coarser patch did not need subdivision
  end if
  SetStencil( $\mathbf{f}$ );
end for
```

This approach observes multiresolution fragments are required only when they do not need subdivision, but corresponding next-coarser fragments do.

Once the appropriate stencil bits have been set, every multiresolution “splat” can be accomplished by drawing a single full-screen quad with stenciling enabled. Early stencil culling avoids generating fragments in sparsely populated regions of the illumination buffer, and we avoid explicitly creating and repeatedly processing the list of patches.

4.2.1.2 Gathering Illumination

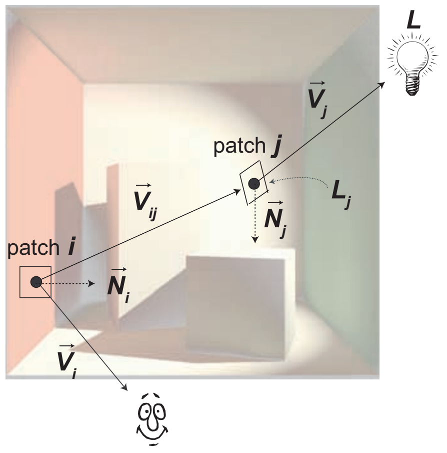
In an instant radiosity-based algorithm, indirect illumination may either be scattered to pixels via splatting or gathered to pixels during a single pass over all pixels. If all VPLs affect all patches, as in our work, both scattering and gathering give equivalent results. Prior work where VPLs illuminate only a subset of the image [23] achieve better performance via splatting. More recent techniques achieved better performance via gathering [118]. We also found gathering to perform better.

To improve cache coherency during this gather, we first create a *VPL cache*, which stores a list of VPL positions, normals, and colors. This avoids fragments fetching VPLs from incoherent locations in the RSM and condenses all relevant data into a single texture.

When a fragment passes the stencil test, we gather contributions of all VPLs to the corresponding eye-space patch. To calculate the contribution from each light-space patch, we approximate the patch-to-patch form factor using Wallace et al.'s [153] disk-to-point approximation:

$$F_{j \to i} = \frac{A_j(\vec{N_i} \cdot \vec{V_{ij}})(\vec{N_j} \cdot \vec{V_{ji}})}{\pi \|\vec{V_{ij}}\|^2 + A_j}, \tag{4.1}$$

where patches i and j are, respectively, the multiresolution fragment seen from the eye and the current VPL, L_j. See Figure 4.4 for a visual depiction of the area A_j, the normals $\vec{N_i}$ and $\vec{N_j}$ and the vectors $\vec{V_i}$, $\vec{V_j}$, $\vec{V_{ij}}$, and $\vec{V_{ji}}$. We do not consider visibility,



L	Scene light source
E	Viewpoint
N_i	Surface normal at patch i
$\vec{V}_i$	Vector from i to the eye E
N_j	Surface normal at patch j
$\vec{V}_j$	Vector from j to the light L
L_j	VPL corresponding to patch j
$\vec{V}_{ij}$	Vector from patch i to patch j
$\vec{V}_{ji}$	Vector from patch j to patch i

Figure 4.4: Notation used for image-space radiosity.

so we omit the visibility from Equation 4.1.

Every VPL, L_j, corresponds to some block of pixels in the reflective shadow map. Each block of pixels, and thus each L_j, represents some solid angle ω_j. The total intensity I_j of L_j relates to the total intensity I of the light L as follows:

$$I_j = I\frac{\omega_j}{4\pi}. \tag{4.2}$$

Additionally, we can approximate area A_j based upon the solid angle ω_j and distance from L:

$$A_j \approx \omega_j \|V_j\|^2. \tag{4.3}$$

Finally, for ease of computation we make the approximation that all RSM texels have equal solid angle. For slightly reduced performance, this approximation can be eliminated. Each texel t thus represents solid angle ω_t:

$$\omega_t \approx \frac{\omega_{frustum}}{4\pi}\frac{1}{\mathrm{RSM}^2_{res}}. \tag{4.4}$$

As we use a 256^2 RSM with a $90°$ field-of-view, $\omega_t \approx 1/(6 \times 256^2)$. The solid angle ω_j represented by each VPL depends linearly on the number of RSM texels it represents.

We obtain the indirect color for each fragment, corresponding to an eye-space patch i, by summing over all VPLs:

$$C_i^{indirect} = \rho_i \sum_j I_j \rho_j F_{j \to i}, \tag{4.5}$$

where ρ_i and ρ_j are the diffuse colors of patches i and j.

After gathering illumination into the multiresolution illumination buffer, we produce a combined, interpolated indirect illumination image using the approach described by previous work [102]. We combine this with the direct illumination C_i^{direct} to produce the final result.

4.2.2 Multiresolution VPL Clusters

This stencil-based technique renders indirect illumination significantly faster than prior work, but it still exhibits temporal incoherence. As geometry and lights move around the scene, the sampled VPLs can jump suddenly from one surface to another, as illustrated in Figure 4.5.

While this is clearly a sampling issue, it can also be viewed in another light (see Figure 4.6). Typically, VPLs are selected by quasi-Monte Carlo ray casting or sampling on a regular grid, without regard to image discontinuities that introduce high frequencies. Effectively, this means light-space patches may contain multiple surfaces! Thus, point sampling patches to find a single representative VPL can give significant variations from similar viewpoints.

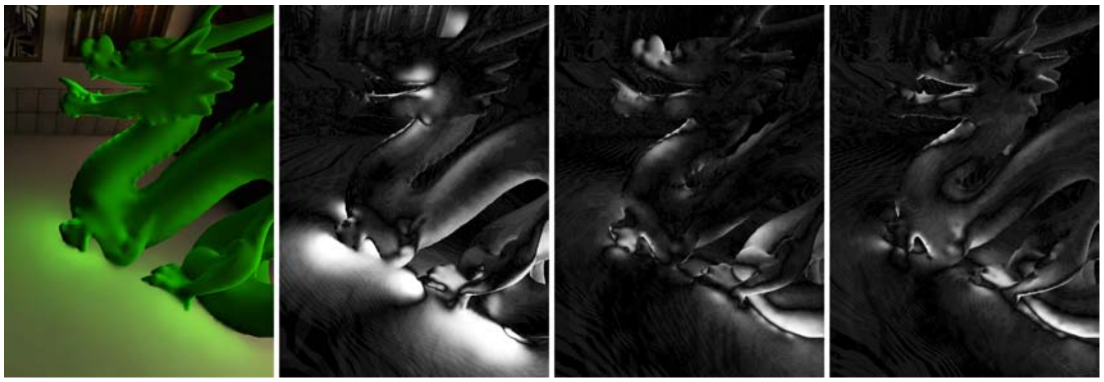


Figure 4.5: Temporal coherence between two successive animation frames. From left to right: the brightened indirect illumination, frame-to-frame difference using RSMs with 256 VPLs, difference using image-space radiosity, and difference using hierarchical image-space radiosity.

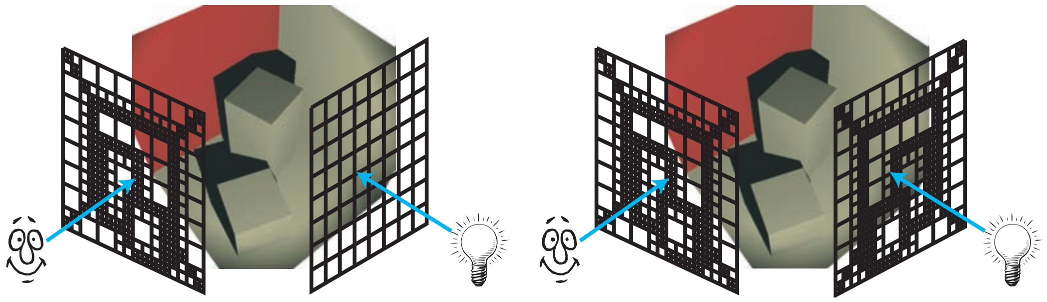


Figure 4.6: (Left) Multiresolution splatting has temporal coherence issues when light samples jump suddenly from one surface to another. This is due to poor light-space patch selection. (Right) Using multiresolution VPLs lessens this problem, as VPLs near boundaries split into multiple, less important patches.

We propose subdividing VPL patches, just as we divide patches in eye-space. Each light-space patch will cover a single surface, with energy varying by patch size. As a VPL approaches a discontinuity its patch splits into smaller pieces, each contributing less. On the other side of the boundary, these pieces recombine to form larger patches.

We describe two light-clustering techniques. The first creates a set of light-space patches and corresponding VPLs each frame. All contribute energy to all eye-space patches, just as basic radiosity [39] evaluates form factors between all patch pairs. The second approach realizes that VPLs may contribute little energy to distant patches, and thus clusters VPLs independently for each eye-space patch, reducing per-fragment computation.

4.2.3 Basic Image-Space Radiosity

Our first clustering technique extends the stencil-based gather from Section 4.2.1. Instead of naively sampling the RSM to identify VPLs, we form VPLs by clustering RSM texels at various resolutions. A single geometry shader pass depicted in Figure 4.7 processes initial, finely sampled VPLs by discarding ones without nearby discontinuities. Discontinuity detection relies on light-space normal and depth max-min mipmaps, similar to those used during eye-space rendering. This produces a set of light-space patches of varying size: large patches in low-frequency areas, and smaller patches densely clustered around discontinuities.

After identifying our patches, corresponding VPLs are stored in the VPL cache.

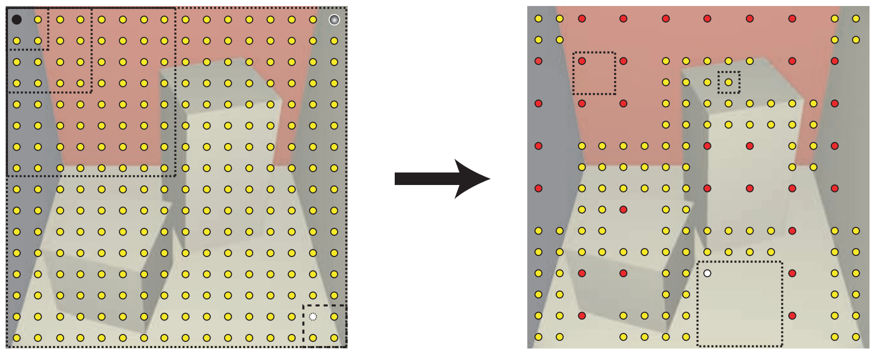


Figure 4.7: (Left) Per-frame light patch construction starts from a dense RSM sampling, here shown at 16^2. Imagine an implicit mipmap on this sampling; all samples represent a texel at the finest resolution. Some, like the upper right point, *only* correspond to texels at the finest level. Others, like the white point, are valid at multiple levels. After implicitly clustering once, the white point represents the patch outlined by the dashed box. A few points, like the black sample, represent patches after numerous clustering stages, depicted by the dotted regions. A geometry shader performs our light clustering, testing each patch to identify any nearby discontinuities. If no nearby discontinuities are detected at any valid mipmap level, it is discarded. (Right) Here, the resulting VPLs are shown in yellow, red, or white, depicting regions after 0, 1, or 2 clustering stages. Three representative patches are displayed at varied resolutions.

Each multiresolution eye-space fragment then gathers illumination as before, taking care to assign VPL solid angles ω_j based upon their hierarchy level, ensuring all contribute an appropriate amount of light.

4.2.4 Hierarchical Image-Space Radiosity

While partially addressing the undersampling depicted in Figure 4.6, this approach is not optimal. Although light patches may contain discontinuities that require a finer subdivision, it is not necessary to always *use* these finer patches. For distant eye-space fragments, coarser VPL sampling may introduce negligible error. Ideally, we would select the best VPL clusters for each fragment, using the finest light-space patches only where noticeable error otherwise occurs.

Our second clustering technique approaches this goal. When rendering fragments, we adaptively gather from light-space patches as coarse as 16^2 or as fine as the RSM resolution. We gather illumination for each fragment independently, with no assumed knowledge of which patches are required for any fragment.

Instead of creating a VPL cache each frame, we reorganize the full resolution RSM data into a *VPL tree*, a structure akin to the single texture, implicit mipmap depicted in Figure 4.7. However, we reorganize this texture to facilitate cache coherence by clustering VPLs representing coarser mipmap levels (i.e., similar to the clustering of Harr wavelet coefficients after applying a 2D wavelet transform). When gathering illumination, we start with the 16^2 coarsest light patches and adaptively subdivide whenever the error introduced by clustering exceeds a user-specified threshold.

4.2.4.1 VPL Tree Traversal

To determine when to subdivide patches, we must quantify image-space error. To do this, we find a computationally cheap, conservative bound to patch j's contribution to the fragment i, only refining light patches when this contribution exceeds a user threshold. First, from Equations 4.1 and 4.5, we know the contribution $C_{j\to i}$ from j to i:

$$C_{j\to i} = \rho_i \rho_j I_j \left(\frac{A_j(\vec{N_i} \cdot \vec{V_{ij}})(\vec{N_j} \cdot \vec{V_{ji}})}{\pi \|\vec{V_{ij}}\|^2 + A_j} \right). \tag{4.6}$$

Observing that $(\vec{N_i} \cdot \vec{V_{ij}})$, $(\vec{N_j} \cdot \vec{V_{ji}})$, ρ_i, and ρ_j always remain less than 1, we bound $C_{j\to i}$ by replacing them. We then apply Equations 4.2, 4.3, and 4.4 to rephrase this bound only in terms of the constant RSM resolution and the distances $\|\vec{V_j}\|$ from light to VPL and $\|\vec{V_{ij}}\|$ from i to j:

$$C_{j\to i} \leq \frac{2I\|V_j\|^2}{3\cdot \mathrm{RSM}_{res}^2\|\vec{V_{ij}}\|^2 + 2\|V_j\|^2} \equiv \mathbb{B}(C_{j\to i}). \tag{4.7}$$

We refine patch j whenever the bound $\mathbb{B}(C_{j\to i})$ exceeds a user threshold τ. By reorganizing further, we identify a simpler per-frame constant T such that refinement occurs when:

$$\|\vec{V_{ij}}\|^2 \leq \frac{2(I-\tau)}{3\tau\cdot \mathrm{RSM}_{res}^2}\|V_j\|^2 \equiv T\|V_j\|^2, \tag{4.8}$$

allowing evaluation of each patch's refinement criteria with a single texture lookup, multiply, and comparison.

Point sampling the RSM to find $\|V_j\|^2$ may mistakenly end refinement too soon, especially on surfaces seen obliquely from the light. To avoid the resulting artifacts, we sample $\|V_j\|^2$ from a min-max mipmap and use the maximum depth over the entire patch as $\|V_j\|^2$.

4.3 Results and Discussion

Our implementation uses OpenGL and GLSL, with performance measured on a dual-core 3GHz Pentium 4 with a GeForce GTX 280. Because we use a geometry shader only to discard geometry, which can be done via clipping, older Shader Model 3 hardware can also run our techniques.

Figure 4.8 compares static results from our hierarchical technique, our stencil-based multiresolution gather, a full-resolution RSM gather, and a splatting-based RSM approach [23] that limits VPL influence to nearby fragments to reduce overdraw.

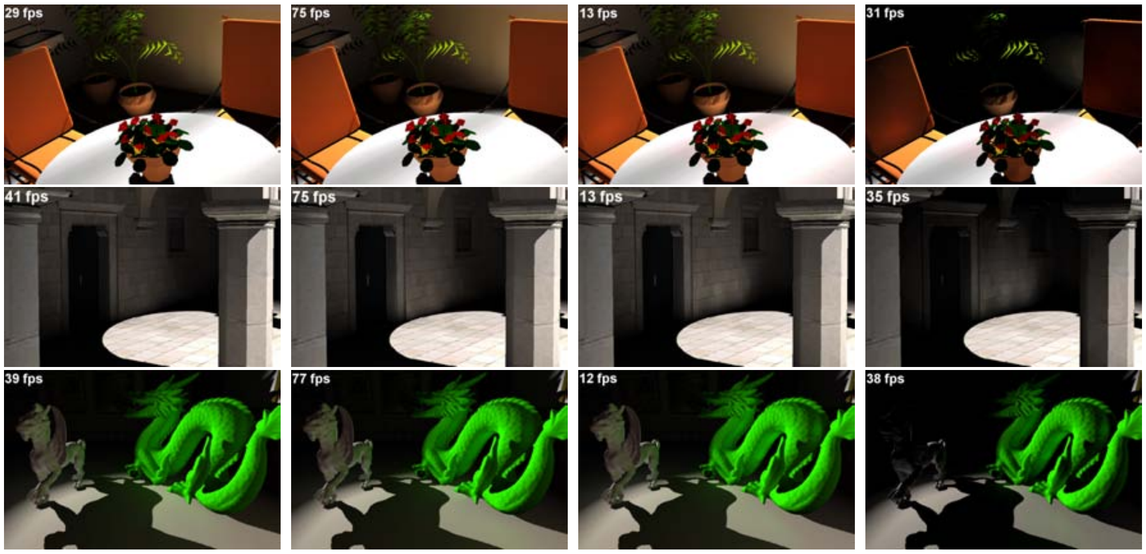


Figure 4.8: Scenes depicted with four rendering techniques (left to right): hierarchical image space radiosity, our stencil-based multiresolution gather, RSMs using full-screen splats, and RSM splatting with reduced overdraw (i.e., VPL influence limited to nearby fragments). Note that static images do not convey the vastly improved temporal coherence enabled by our approach.

Here, the threshold is chosen to generate the same number of fragments per splat as our multiresolution approach.

4.3.1 Stencil Refinement

Table 4.1 demonstrates the large performance gains achieved with our stencil refinement, as compared to splatting using a geometry shader for iterative patch refinement. Note the constant speed of the “stencil setup” phase, which identifies the multiresolution fragments. Comparatively, the iterative refinement of prior work varies in performance based upon the specified subdivision thresholds and the number of refined fragments.

The big performance win comes from the stenciled gathering process. Mul-

Table 4.1: Comparison of per-frame costs

Multiresolution Stenciled Gathering	Cornell Box	Feline Scene	Sponza Atrium	Indoor Garden
Direct Light	1.8 ms	5.8 ms	2.8 ms	2.8 ms
Generate Reflective Shadow Map	0.9 ms	2.3 ms	1.4 ms	1.6 ms
Create Min-Max Mipmap	0.7 ms	0.7 ms	0.7 ms	0.7 ms
Multires Stencil Setup	0.7 ms	0.7 ms	0.7 ms	0.7 ms
Multires Light Gather	2.5 ms	3.5 ms	3.0 ms	6.6 ms
Upsample Indirect Illumination	1.6 ms	1.7 ms	1.3 ms	1.3 ms
Total	8.2 ms	14.7 ms	9.9 ms	13.7 ms

Multiresolution Splatting [102]	Cornell Box	Feline Scene	Sponza Atrium	Indoor Garden
Direct Light	1.8 ms	5.8 ms	2.8 ms	2.8 ms
Generate Reflective Shadow Map	0.9 ms	2.3 ms	1.4 ms	1.6 ms
Create Min-Max Mipmap	0.7 ms	0.7 ms	0.7 ms	0.7 ms
Geometry Shader Refinement	1.3 ms	1.7 ms	1.3 ms	1.6 ms
Multires Light Splatting	45.8 ms	67.0 ms	58.1 ms	130.0 ms
Upsample Indirect Illumination	1.6 ms	1.7 ms	1.3 ms	1.3 ms
Total	52.1 ms	79.2 ms	65.6 ms	138.0 ms

Costs of each step for a single frame for (top) our new stencil-based multiresolution gathering and (bottom) multiresolution splatting [102], using 256 VPLs. “Direct light” includes all other rendering overhead.

tiresolution splatting typically generates around 50,000 multiresolution fragments for a 1024^2 image. For each VPL, all these fragments are processed by both a vertex and fragment shader and scattered incoherently into a multiresolution buffer. This is clearly the most expensive stage. Reformulating as a gather roughly halved this stage’s cost, and leveraging the hierarchical z-buffer for early stencil culling avoided the incoherent scatter of fragments into the buffer. Combined, these techniques reduce the cost of the indirect illumination by over an order of magnitude.

As with multiresolution splatting, performance depends more on eye-space *visual complexity* than on scene complexity. For instance, the feline scene contains more geometry than the garden, but high frequencies introduced by the plants require

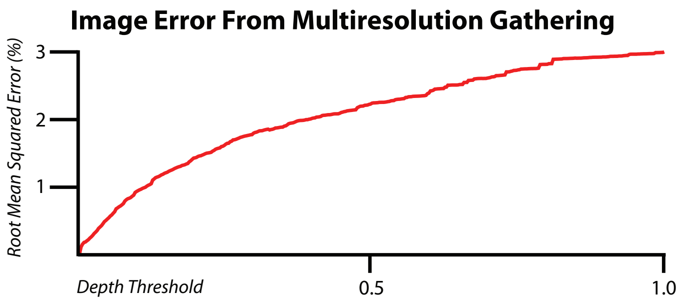


Figure 4.9: Root mean squared error between multiresolution gathering and gathering illumination at every eye-space pixel. Both use one million VPLs (i.e., a 1024^2 RSM) to help differentiate between error caused by clustered eye-space patches and clustered light-space patches (light patch error is depicted in Figure 4.13).

finer eye-space patches, increasing costs.

Figure 4.9 illustrates the error introduced solely by multiresolution eye-space patches as a function of the depth discontinuity required to refine fragments. At a depth threshold of zero, indirect light is gathered at full resolution, yielding zero RMS error. Increasing this threshold increases error as fragments are clustered together. At thresholds that yield reasonable performance, error ranges from 2-3%.

4.3.2 Light Hierarchies

Just as Table 4.1 demonstrates a performance dependence on eye-space visual complexity, clustering VPLs adds a performance sensitivity to light-space visual complexity. Figure 4.10 shows framerate variations in the feline scene as the light moves. As visual complexity changes, so does the required number of VPLs, leading to fluctuations in performance.

Performance characteristics for image space radiosity and our hierarchical variant are displayed in Figures 4.11 and 4.12. These figures start with 1024 coarse VPLs

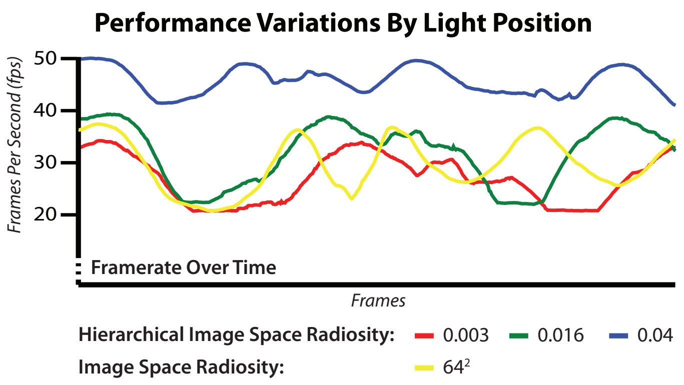


Figure 4.10: Framerate varies as a light moves through the feline scene. Compared are image space radiosity with a maximum VPL subdivision to 64^2 and hierarchical image-space radiosity with three different error thresholds.

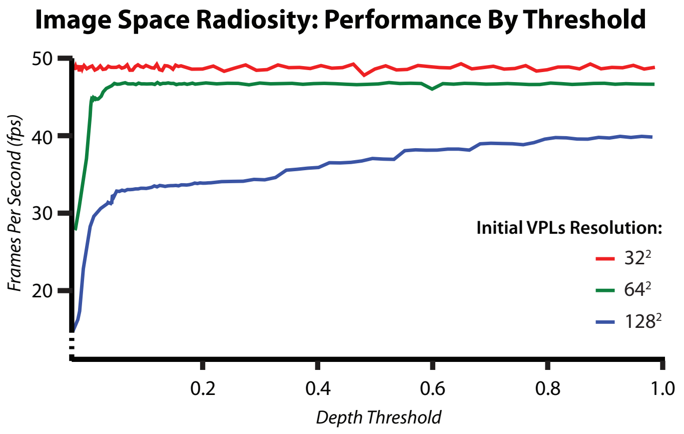


Figure 4.11: Performance of image space radiosity in the feline scene, with different initial VPL resolutions. Lower depth thresholds introduce more light patches.

and subdivide in important areas. Performance tops out around 50 fps, when gathering into the illumination buffer is no longer the bottleneck; fixed cost passes including VPL tree creation or per-frame patch construction then limit performance.

Figure 4.13 explores the error introduced by hierarchical image space radiosity as a function of the user-specified error threshold. To distinguish error from our

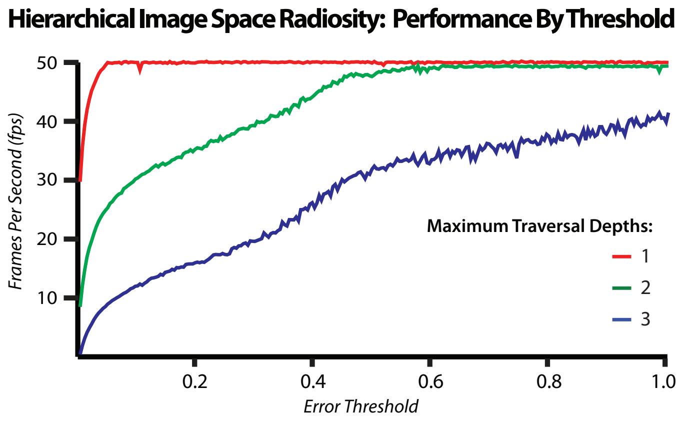


Figure 4.12: Performance of hierarchical image space radiosity in the feline scene, using different maximal traversal depths. As we allow additional subdivisions, lower error thresholds introduce more light patches.

light- and eye-space hierarchies, the baseline image uses our multiresolution gathering with one million VPLs; even with multiresolution gathering, this requires more than 10 seconds. A threshold of zero forces a full traversal of the VPL tree, though due to reduced VPL sampling (32^2–128^2) error is not completely eliminated. At higher thresholds, fewer VPLs contribute illumination, and error increases to 3–4% for thresholds with acceptable performance.

Combined with the error demonstrated in Figure 4.9, we get interactive performance with a 5–6% RMS error compared to a baseline gather of a million VPLs at every pixel in eye-space. Considering the half hour computation for the baseline and the temporal coherence maintained by our method, we believe this is an acceptable trade-off.

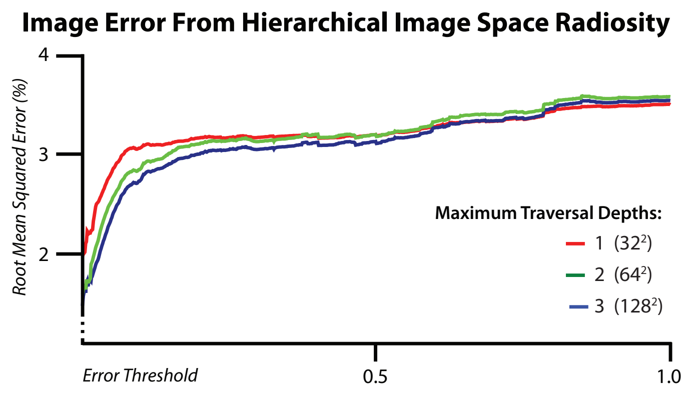


Figure 4.13: Root mean squared error between hierarchical image space radiosity (with different maximal traversal levels) and our multiresolution gathering using one million VPLs (i.e., from a 1024^2 RSM).

4.4 Conclusions

This chapter presented image-space radiosity and a hierarchical variant, techniques that build upon our improved one-pass stenciled multiresolution gathering for indirect illumination. These techniques dramatically reduce the cost of approximating one-bounce indirect illumination using reflective-shadow map based instant radiosity. We achieve this by gathering indirect light at varying resolutions in image space, depending on proximity to discontinuities. When gathering illumination for each image space patch, we proposed two light-space clustering techniques to reduce the number of VPLs used per patch; one creates VPL clusters once per frame, the other selects VPL clusters based upon a per-fragment error metric and a user specified error threshold.

CHAPTER 5
INTERACTIVE, MULTIRESOLUION IMAGE-SPACE RENDERING FOR DYNAMIC AREA LIGHTING

5.1 Introduction

Interactive rendering has long relied on illumination from infinitesimal points, where efficient lighting and visibility are well understood. Illumination from more complex area sources requires integrating both visibility and radiance over the light's surface. Given framerate constraints in interactive applications, per-pixel computation of such integrals is infeasible. Current solutions include precomputing light transfer [136], discretely sampling the light source [75], and caching samples in object-space data structures [83] that require updates for dynamic geometry.

Researchers have developed numerous algorithms for soft shadows that integrate light over an area light [48], but most break with larger lights and few allow radiance fluctuations over the emitter. A few sophisticated algorithms account for radiance variations in penumbral regions using 4D lookup tables [9] or summed area tables [44], but they generally treat unoccluded radiance as coming from a single-colored emitter, using the average source radiance.

In this chapter, we propose a novel image-space technique for interactive lighting from area lights that draws inspiration from recent global illumination research. In particular, recent techniques demonstrate that rendering of slowly varying lighting can be accelerated via multiresolution image-space computations like those described Chapters 3 and 4, and can rely on crude visibility approximations [119].

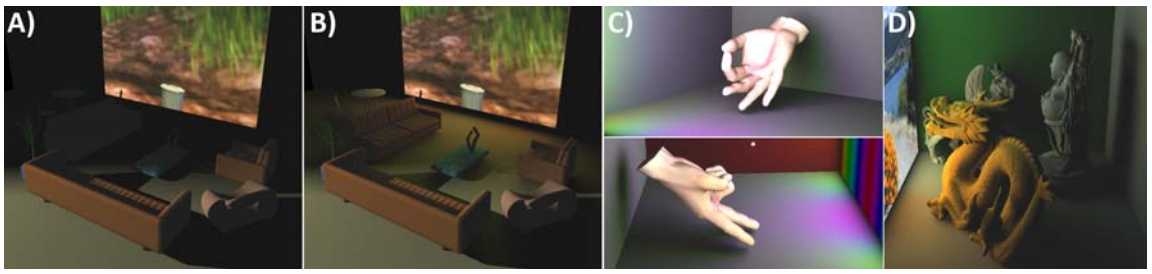


Figure 5.1: Traditional real time renderers (a) allow only point lighting. We introduce a multiresolution image-space approach (b) that enables illumination from dynamic area lights. No precomputation is needed, so we handle dynamic geometry (c). The image-space computations decouple algorithmic and geometric complexity, so scenes with hundreds of thousands of polygons (d) still render interactively.

This chapter presents **three main contributions**:

1. A multiresolution approach that gathers illumination from dynamic area lights without visibility. This runs in real time for diffuse and non-diffuse BRDFs.
2. A multiresolution algorithm using screen-space voxelization to quickly compute per-pixel area light visibility.
3. An incremental refinement that considers both illumination and visibility variations when choosing an appropriate resolution for rendering illumination.

Since we avoid precomputation, our work handles dynamic lights, viewpoint, and geometry (see Figure 5.1). Image-space computations allow our algorithm to scale with visual complexity rather than geometric complexity. Additionally, iterative refinement provides a clear quality/performance tradeoff: simply stop sooner for improved performance. However, our discrete geometric representation can introduce aliasing.

5.2 Previous Visibility Approximations

Researchers have long developed techniques to interactively compute light visibility. Until recently, these focused on visibility from point lights and approximate methods for uniform area lights. Here, we present additional background, discussing the most relevant of these techniques. Discussion of global illumination techniques with characteristics similar to our work may be found in Chapter 2.

5.2.1 Single Shadow Map

Standard shadow maps [163] quickly compute visibility from point lights. The image-space computations scale well with increased scene complexity and enable efficient post processing [34, 113] for approximating more complex effects, such as soft shadows. Most soft shadow map algorithms [48] rely on crude, but plausible, visibility approximations and ignore radiance variations over the light.

Recent work aims to improve accuracy. Backprojection soft shadows [44] replace each shadow map texel by a micropolygon, allowing computation of analytical per-patch visibility. Disjoint micropolygons introduce light leakage and overshadowing, but more sophisticated meshing reduces the problem [129]. Backprojection creates a per-fragment list of potential occluders; to avoid searching all micropolygons, hierarchical shadow maps [44, 129] accelerate identification. While these hierarchies break for large lights, subdividing the lights [167] improves quality.

While backprojection soft shadow maps accurately compute visibility from area lights, they largely ignore variations due to multicolored lights. A few techniques use 4D lookup tables [9] or summed area tables [44] to represent varying radiance. Bitmask soft shadows [129] use a bitmask approach that inspires our work. However, none of these algorithms consider how radiance variations affect unoccluded fragments.

One common problem for all single map algorithms stems from use of a single silhouette edge to approximate occlusions. While sufficient for small lights, it presents challenges for large lights, especially as occluders approach the light.

5.2.2 Multiple Shadow Map

Using many shadow maps avoids this problem, at the cost of additional render passes. Heckbert and Herf [51] suggest rendering shadow maps from many samples. While this converges for increasing sampling density, rendering hundreds of shadow maps proves a serious bottleneck.

Annen et al. [3] split lights into multiple samples and use fast per-sample soft shadowing [4], though they handle an order of magnitude fewer samples than our work due to memory limits and shadow map construction costs. Coherent shadow maps [116] precompute and compress shadow map per-object visibility, enabling moving geometry and complex lighting if object convex hulls do not overlap [118]. However, this requires substantial precomputation and memory overhead.

An ingenious way to accelerate multi-shadow map illumination uses crude, imperfect maps [119] and relies on averaging over thousands of shadow lookups to remove low-resolution shadow map artifacts. To efficiently create imperfect shadow maps, a preprocess densely point-samples the scene. While relatively inexpensive, this preprocess limits dynamic scenes to deformations that need no resampling. Recent work [117] shows point primitives also work with more complex illumination and material properties, albeit at significantly increased cost.

5.2.3 Multi-Layer

Shadow maps help accelerate many complex illumination problems, but extending the depth buffer to contain 3D data, e.g., using layered depth images [130], provides an interesting alternative. Agrawala et al. [1] precompute visibility for interactive rendering, and Im et al. [59] achieve near-interactive soft shadowing with layered depth.

Multi-layer techniques often use depth peeling [33] to achieve interactivity, and recent work simultaneously peels many layers [91]. One visibility approximation using layer depth is screen-space ambient occlusion [10, 96]. While it interactively captures visibility from large area lights, it poorly handles highly directional lighting.

We use another layered visibility approximation: image-space scene voxelization [27, 31]. Voxelization discretizes the volume enclosed by the view frustum, with each framebuffer pixel representing multiple single-bit voxels (see Figure 5.2). Voxelization occurs in a single render pass, requiring under a millisecond even for relatively complex scenes. This representation proves useful for numerous applications [32], such as refraction, translucency, and collision detection. We observe that screen-space voxelization efficiently creates a perspective grid [58], suitable as a simple

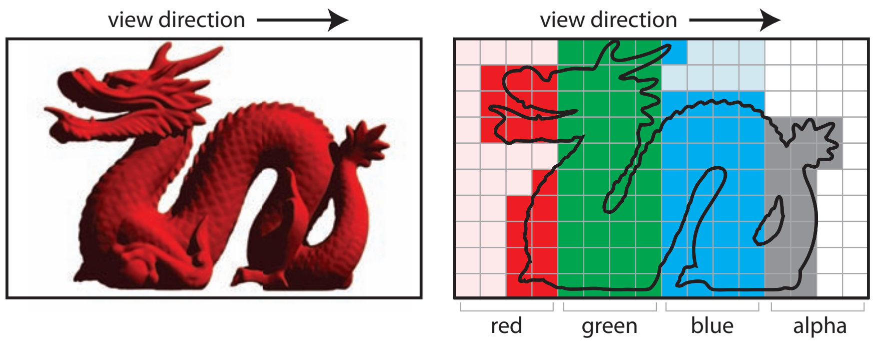


Figure 5.2: An example of voxelization, viewed from the side. Each quanta of depth, represented by a single bit in the color buffer, identifies whether geometry is present in the voxel.

ray acceleration structure (similar to Woo [164]).

5.3 Multiresolution Illumination

Direct illumination from an area light requires integrating over the light for each pixel, combining light contributions, material reflectance, and any occlusions. Using an area formulation of the rendering equation [68], we write this:

$$L(\mathbf{x}, \vec{\omega}) = \int_{\mathbf{y} \in S} f_r(\vec{\omega}, \mathbf{x} {\rightarrow} \mathbf{y}) I(\mathbf{y}) V(\mathbf{x} {\rightarrow} \mathbf{y}) G(\mathbf{x} {\rightarrow} \mathbf{y}) dA, \tag{5.1}$$

where $\mathbf{x}$ is the point to shade, $\vec{\omega}$ the viewing direction, S the light surface, I the light intensity, $\mathbf{x} {\rightarrow} \mathbf{y}$ the vector from $\mathbf{x}$ to the light sample $\mathbf{y}$, V the binary visibility between $\mathbf{x}$ and $\mathbf{y}$, and $G(\mathbf{x} {\rightarrow} \mathbf{y})$ the geometry term:

$$G(\mathbf{x} {\rightarrow} \mathbf{y}) = \frac{\cos(\mathbf{x} {\rightarrow} \mathbf{y}, \vec{N}_{\mathbf{x}}) \cos(\mathbf{y} {\rightarrow} \mathbf{x}, \vec{N}_{\mathbf{y}})}{||\mathbf{x} {\rightarrow} \mathbf{y}||^2}.$$

$\vec{N}_{\mathbf{x}}$ and $\vec{N}_{\mathbf{y}}$ are the surface normals at $\mathbf{x}$ and $\mathbf{y}$. In an interactive context, sampling provides the only feasible solution, so we rewrite Eq. 5.1 as a sum over discrete light samples:

$$L(\mathbf{x},\vec{\omega}) \approx \sum_{i=0}^{N} f_r(\vec{\omega},\mathbf{x}{\rightarrow}\mathbf{y}_i) I(\mathbf{y}_i) V(\mathbf{x}{\rightarrow}\mathbf{y}_i) G(\mathbf{x}{\rightarrow}\mathbf{y}_i) A(\mathbf{y}_i). \tag{5.2}$$

Here, light samples $\mathbf{y}_i$ can be thought of as virtual point lights, allowing algorithms akin to recent global illumination research (e.g., [22, 75]).

5.3.1 Overview

We have three main contributions. Section 5.3.2 ignores the visibility in Eq. 5.2 and adapts the methods from Chapters 3 and 4 to interactively gather from virtual point lights (VPLs) on the area light. While this retargeting is relatively straightforward, area illumination enables opportunities for improved performance and, unlike our earlier work, allows rendering of non-diffuse materials. Section 5.3.3 considers *only* the visibility from Eq. 5.2 and applies screen-space voxelization [32] to coarsely represent the scene. We introduce an incremental, multiresolution ray marching approach that interactively approximates light visibility. Section 5.3.4 combines these into a multiresolution approach to interactively render area illumination, allowing both visibility and radiance variations over the light.

5.3.2 Gathering Without Visibility

Illumination from area sources generally changes smoothly, giving low frequency lighting where coarser than per-pixel illumination sampling should suffice. One could apply object-space caching [83], but this requires data structures that incur expensive updates in dynamic environments. We instead build on the multiresolution image-space work presented in Chapter 4, akin to an image-space cache, with a cheap per-frame build cost that allows dynamic geometry.

First we simplify area illumination in Eq. 5.2 by ignoring visibility (setting $V = 1$) and assuming a diffuse material with albedo ρ, giving the following equation:

$$L(\mathbf{x}, \vec{\omega}) \approx \frac{\rho}{\pi} \sum_{i=0}^{N} \frac{I(\mathbf{y}_i) A(\mathbf{y}_i) \cos(\mathbf{x}{\rightarrow}\mathbf{y}, \vec{N}_{\mathbf{x}}) \cos(\mathbf{y}{\rightarrow}\mathbf{x}, \vec{N}_{\mathbf{y}})}{||\mathbf{x}{\rightarrow}\mathbf{y}||^2}.$$

This per-sample contribution resembles Equation 3.2, the per-VPL contribution from Chapter 3, suggesting that a similar multiresolution approach may also work for direct area lighting. The earlier approach works as follows:

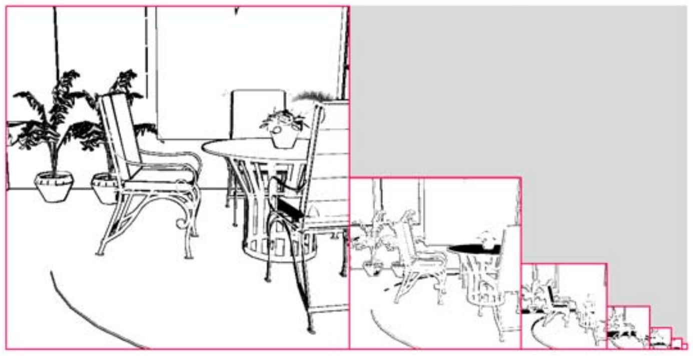
1. Render an eye-space G-buffer [125];
2. Compute multi-scale image-space depth and normal variations by creating a mipmap structure over the G-buffer;
3. In parallel, set stencil bits in a multiresolution buffer at the texels where illumination will be gathered (where depth & normal variations exceed a threshold);
4. In parallel, gather illumination only at specified texels;
5. Combine and upsample to full-screen resolution.

We observe that illumination discontinuities for area sources occur at locations similar to indirect light: at image-space depth and normal boundaries. We retain the depth and normal metrics from Chapter 3, computing per-pixel depth derivatives (in Step 2) as $\sqrt{(dz/dx)^2 + (dz/dy)^2}$, and creating a max-mipmap. We use a similar computation for normals, performed on a per-component basis; normal discontinuities occur when at least one component of the normal changes significantly.

Step 3 stencils all resolutions in parallel, via a full-screen pass over a flattened multiresolution buffer:

```
for all (fragments $\mathbf{f} \in$ quad) do
  if ( $\forall i, \mathbf{f} \notin$ MipmapLevel( $i$ ) ) then
    continue; // Fragment not actually in multires buffer
  end if
  $i \leftarrow$ CurrentMipmapLevel( $\mathbf{f}$ );
  if ( HasDepthOrNormalDiscontinuity( $\mathbf{f}, i$ ) ) then
    continue; // Patch not valid (needs subdivision)
  end if
  if ( NoDiscontinuity( $\mathbf{f}, i+1$ ) ) then
    continue; // Coarser patch did not need subdivision
  end if
  SetStencil( $\mathbf{f}$ );
end for
```

The resulting stencil has bits set for fragments to be sampled:



We gather illumination from VPLs on the area light, using the stencil to cull unneeded computations. Figure 5.3 shows the resulting area lighting, with coarse samples in slowly varying regions and denser samples near discontinuities.

For area lighting, a priori knowledge of the source location enables multiresolution improvements infeasible for global illumination. First, gathering light on emitters is unnecessary; we stencil them out, resulting in dramatic performance improvements

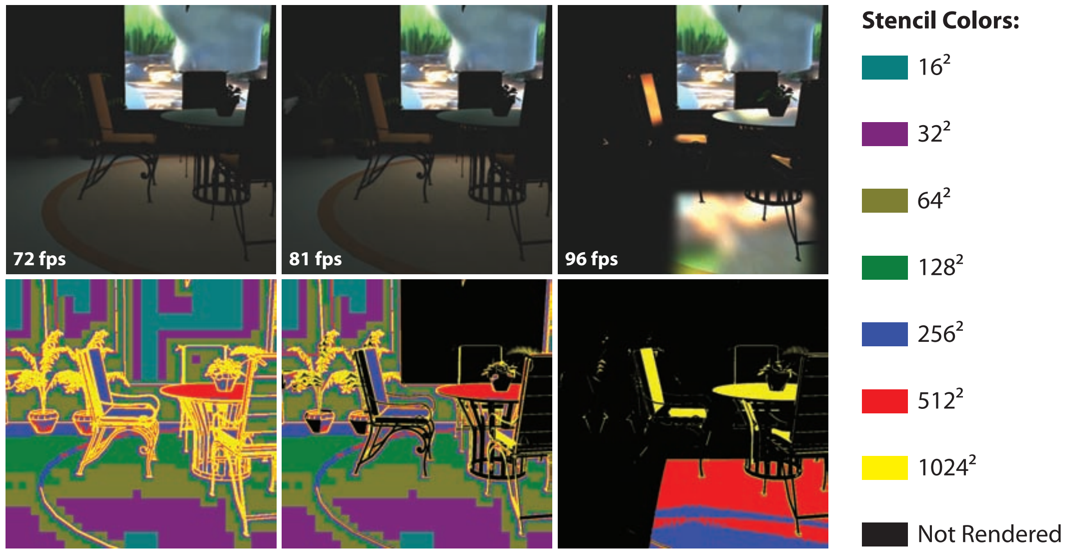


Figure 5.3: The indoor garden with diffuse (left, center) and Phong (right) materials. We show the final rendering above a pseudocolor image depicting the resolution we rendered each region. Our naive approach (left) gathers illumination everywhere. Discarding fragments on or facing away from the light (center) improves performance. Phong surfaces (right) are refined extensively or trivially discarded.

in scenes with large light sources.

Second, while indirect light often arrives incoherently, direct light often has a preferred direction. For point sources, a cheap $\vec{N} \cdot \vec{L} < 0$ identifies unlit fragments. For area lights, we detect geometry facing away from the light by testing $\vec{N} \cdot \vec{L}_j$ for vectors $\vec{L}_j$ towards the corners (or bounding box) of the light. If $\vec{N} \cdot \vec{L}_j < 0$ for all j, we trivially discard the fragment. Additionally, for one-sided lights, such as television screens, we also discard fragments behind the source.

5.3.2.1 Non-Diffuse Materials

Non-diffuse materials require keeping the BRDF inside the sum in Eq. 5.2. This complicates multiresolution rendering, as shiny materials introduce high frequencies not considered by image-space depth and normal metrics. Specular surfaces need additional refinement to capture these frequencies.

Seemingly, further refinement degrades performance by adding additional samples. But while shiny materials reflect high frequencies near the reflection direction, they often reflect little light in others. This suggests traditional importance sampling techniques directly translate into material-specific refinement metrics for multiresolution rendering.

Consider a Phong BRDF without a diffuse component, reflecting light according to the term $(\vec{R}\cdot\vec{L})^n$. For even low shininess n, this reduces off-specular illumination significantly. We discard fragments where the Phong lobe misses the light. We identify these fragments by locating where the reflection vector $\vec{R}$ misses the light, and checking that the Phong lobe does not overlap the source (by guaranteeing $(\vec{R}\cdot\vec{L}_j)^n$ falls below a threshold for all j).

Frequencies in a Phong reflection depend on lobe size. Large lobes give blurry reflections representable via coarser sampling. When we create our depth and normal mipmaps, we also compute the cross-sectional Phong lobe size at the light. Image-space sample spacing must be denser than the lobe width, which we add as a Phong-specific refinement metric. We separate diffuse and Phong components into separate gather passes (see Figure 5.3), which should work for many other BRDFs (e.g., Ashikhmin-Shirley [8]).

As we use 256 VPL samples, highly specular materials can show reflections of individual VPLs. We avoid this using a simplistic adaptive strategy: when a specular lobe includes too few VPLs (we used a threshold of 9) we directly sample the light texture 25 times inside the lobe using a mipmap level dependent on the lobe size when it hits the light. We sample on a regular 5×5 grid centered on the reflection direction, with sample spacing dependent on lobe width.

5.3.3 Voxel-Space Visibility

Interactive rendering algorithms frequently rely on depth maps to approximate the visibility in Eq. 5.2. Creation of a z-buffer as a rasterization byproduct encourages this ubiquity, and algorithmic improvements naturally start from prior shadow mapping work. However, creation of multi-layer buffers [91] or thousands of shadow maps [119] is costly, and accessing the results strains bandwidth and texture cache without providing true 3D data.

Fortunately, screen space voxelization [27, 31] provides a compelling alternative. Voxelization is efficient, taking under a millisecond even in complex scenes, and produces a 3D structure where each bit represents the presence or absence of geometry in a single voxel (see Figure 5.2). Yet, there is a clear tradeoff. 32-bit z-buffers uniquely store 2^{32} depth values, whereas 32-bit voxelizations uniquely represent just 32. But the voxelization can simultaneously store 32 depths while the z-buffer stores just one. We found a 128-bit voxelization worked well for our scenes, though additional render targets could extend this to 2048 bits.

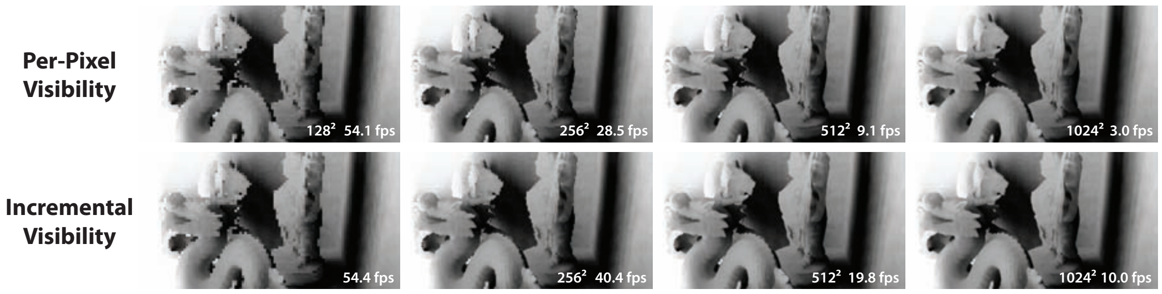


Figure 5.4: Compare (top) brute force visibility and (bottom) our incremental refinement. Brute force computations ray march from each pixel to every VPL, whereas incremental computations reuse results from coarser visibility passes.

5.3.3.1 Naive Voxel-Space Visibility

Inspired by the visibility quality attained using imperfect shadow maps [119], we suggest that image-space voxelization, a similarly crude approximation, can achieve similar quality. Instead of creating thousands of shadow maps and querying each, we propose marching rays through a coarse scene voxelization.

Our naive query spawns a ray traversal between every light sample at each pixel (see Figure 5.4). Ideally, we would perform a 3D DDA for each traversal, checking all intermediate voxels for intersections. This performed poorly on our GPU, due to varying numbers of loop iterations, so we instead sample the ray uniformly between each fragment and VPL. While a sparse sampling may miss occlusions from thin geometry, we partially compensate for missed occlusions by checking multiple bits in texels fetched from the voxel buffer. This essentially thickens the ray.

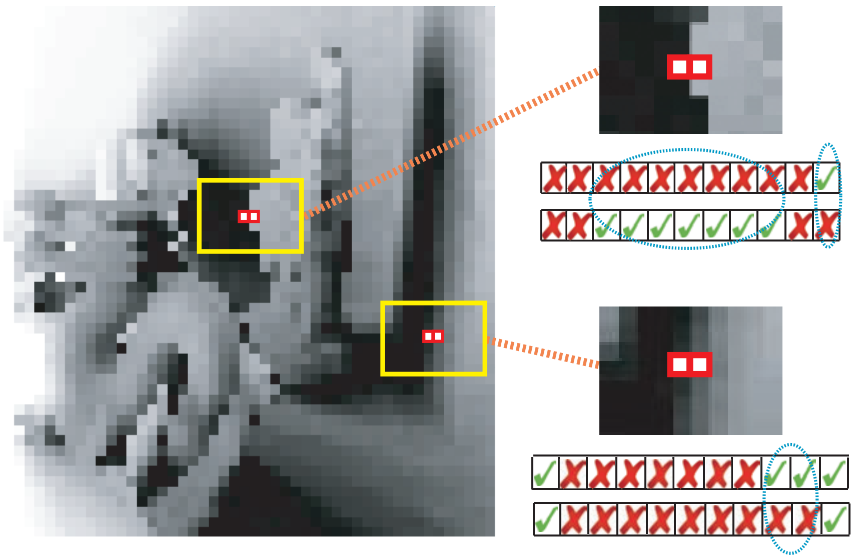


Figure 5.5: Each pixel stores a bitstream of visible and occluded VPLs. When refining, we only resample lights whose visibility changes between neighboring pixels. Even at significant discontinuities (top), only a subset of VPL visibilities change. Often visibility changes more slowly (bottom); here only a few VPLs need resampling during refinement.

5.3.3.2 Incremental Voxel-Space Visibility

While we need per-pixel visibility to each VPL, this visibility, like illumination, changes slowly throughout our image. We leverage this observation, introducing two algorithms that incrementally refine visibility from sparser samples.

Consider shadows from a point light. By definition, these are coherent except at shadow boundaries. A coarse shadow sampling is correct at low resolution, except near these boundaries. Clearly, this also holds for any single VPL. Instead of computing visibility to each VPL at every pixel, we first sample on a coarser pixel grid (e.g., 64^2) and incrementally refine only near visibility discontinuities (see Figure 5.5).

When refining, we reuse a VPL's visibility when coarse samples in a 3×3 region agree, only spawning a new visibility ray when inconsistencies arise. When using hundreds of VPLs, usually fewer than 5% vary in any region, significantly reducing the rays needed. To this end, our visibility passes output a binary bitstream (as in bitmask soft shadows [129]) with one bit per VPL, allowing identification of regions needing additional queries.

While our incremental approach can converge to a correct solution, this is rarely necessary. For instance, imperfect shadow maps have inaccurate per-VPL visibility that gets averaged out, and ambient occlusion algorithms often sample coarsely and then apply a blur. Given our multiresolution approach from Section 5.3.2 essentially blurs illumination, we seek to avoid computing per-pixel visibility prior to blurring.

Instead of refining visibility everywhere, we only generate denser samples near discontinuities. It turns out that most visibility discontinuities occur at image-space depth and normal boundaries, so we can use the same refinement criteria from Section 5.3.2. However, these metrics miss discontinuities caused by contact shadows. To detect these discontinuities, we add a bit-counting metric. When enough VPLs change visibility in an image-space neighborhood, refinement is needed. Surprisingly, this threshold can be quite high: in our scenes, we captured light discontinuities by refining only when at least half the VPLs (i.e., 128) were partially visible in a neighborhood. We compare naive voxel-space visibility with a combination of both our incremental improvements in Figure 5.4.

Given the gradual visibility variations in most scenes, we found interleaved sampling [76] on 2×2 pixel blocks worked at all our sampling resolutions. This allows a 4-channel 16-bit render target to store visibility to all 256 VPLs used in our implementation. Interleaved sampling is essentially orthogonal to the incremental refinements proposed above, so our results include all three.

5.3.3.3 Visibility with Sharp BRDFs

While visibility from a fixed set of 256 VPLs gives good results for diffuse surfaces, for sharper BRDFs individual shadows are clearly visible for the few VPLs reflected in the specular highlight. We propose an alternative for such materials based on the simple adaptive strategy proposed in Section 5.3.2.1. We send visibility rays to the 25 samples chosen in the specular lobe.

This improves shadow quality, but with only 25 visibility samples some shadow banding remains. We could add additional visibility queries in the lobe, but this increases cost, potentially highlights voxel aliasing, and behaves poorly using our incremental visibility. Instead we use a variance shadow query, inspired by variance shadow maps [28].

Instead of storing a single bit per visibility query, we store the distance (and distance squared) to the nearest occluder along the ray. We then estimate the per-query distance variance using a screen space blur, applying the equations described by Donnelly [28] to compute the per-VPL light intensity. This eliminates the shadow banding from individual shadows (see Figure 5.6), as it essentially performs per-query percentage closer filtering.

Unfortunately, this greatly increases storage for visibility queries—from 1 bit

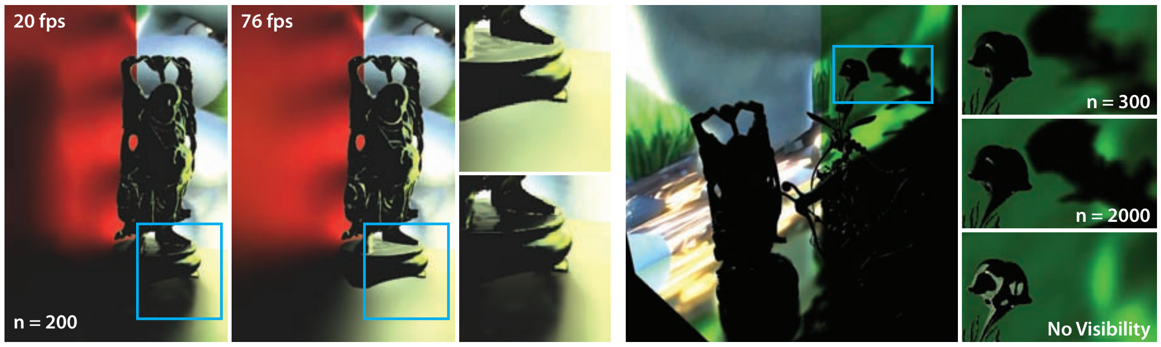


Figure 5.6: Two examples of our method with a Phong BRDF, with and without visibility, and varying levels of shininess.

to 2 floats per VPL. We manage by sending only 24 visibility queries (ignoring one corner on the 5×5 grid), for a total of 48 floats. Using 2×2 interleaving, this requires only 12 floats per pixel, which fits in three 4-channel render targets.

5.3.4 Incremental Stencil Rendering

While our algorithms from Sections 5.3.2 and 5.3.3 give either area illumination or visibility, we need both simultaneously. Additionally, the algorithms refine image-space regions differently, so separate evaluation of lighting and visibility followed by a per-fragment combination is infeasible. Shadows introduce high frequencies where illumination varies slowly, and for specular materials the reverse can hold.

To address this, we propose an incremental approach to multiresolution rendering. We still rely on a stencil mask to identify which mulitresolution fragments to render. But instead of creating the stencil masks simultaneously before rendering, we create the masks and render one resolution at a time, from coarsest to finest. This allows each stencil mask to depend on illumination variations at coarser levels, in

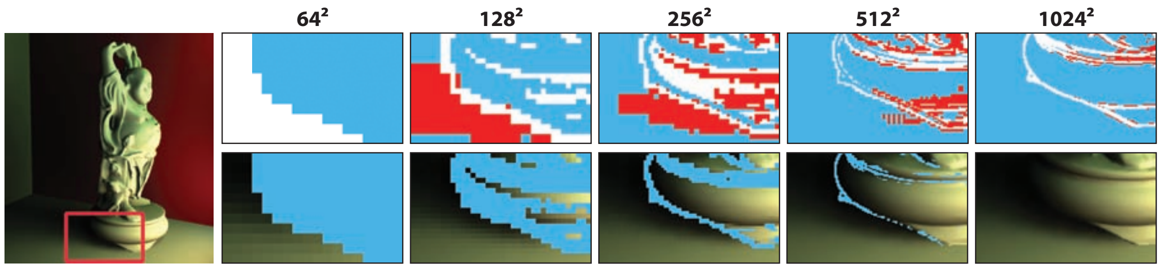


Figure 5.7: An example of incremental stencil refinement. (Top) The refinement stencil for each resolution. Areas in blue are not rendered; areas in white are refined due to depth or normal discontinuities, and areas in red are refined due to illumination discontinuities. (Bottom) The resulting image after each step, with unrendered areas in blue.

addition to prior metrics (e.g., depth and normal). This requires only minor changes from sequential stenciling:

```
$i \leftarrow$ CurrentMipmapLevel( **f** );
**for all** (fragments $\mathbf{f} \in$ quad( mip-level( $i$ ) ) ) **do**
  **if** ( HasIlluminationDiscontinuity( **f**, $i+1$ ) ) **then**
    SetStencil( **f** ); // Coarse patch had light discontinuity;
    **continue**;    //  re-render at higher resolution
  **end if**
  **if** ( HasDepthOrNormalDiscontinuity( **f**, $i$ ) ) **then**
    **continue**; // Patch not valid (needs subdivision)
  **end if**
  **if** ( NoDiscontinuity( **f**, $i+1$ ) ) **then**
    **continue**; // Coarser patch did not need subdivision
  **end if**
  SetStencil( **f** );
**end for**
```

Rapid color changes between coarse samples indicates that higher sampling may capture higher frequencies. When iterating, we compare the light variations between computed samples to a user threshold to determine if additional sampling is needed (see Figure 5.7).

Our incremental rendering iterates the following steps once per rendering resolution, beginning at the coarsest level and progressing through the finest:

1. Create a stencil based on image-space depth, normal, and BRDF frequency metrics *and* any illumination discontinuities observed in coarser levels,

2. At set stencil bits, incrementally compute visibility and illumination to locally approximate Eq. 5.2.

3. When the stencil is *not* set, upsample and interpolate any coarser results.

Figure 5.7 depicts incremental stencil rendering. At the lowest resolution, we create a stencil based solely on depth and normal discontinuities. At higher resolutions, observed illumination discontinuities (shown in red) supplement regions known to require refinement. At each level, we upsample and interpolate unrendered areas using the method developed for our earlier work, described in Section 3.2.5 - essentially an edge-aware bilinear interpolation that avoids using uncomputed fragments (those shown in blue).

This enables adaptive refinement at shadow boundaries. Furthermore, the illumination metric can capture high frequency BRDFs. Discontinuities introduced by specularities naturally trigger refinement, though we find that material specific metrics (e.g., Section 5.3.2.1) typically provide better quality, performance, and avoid aliasing.

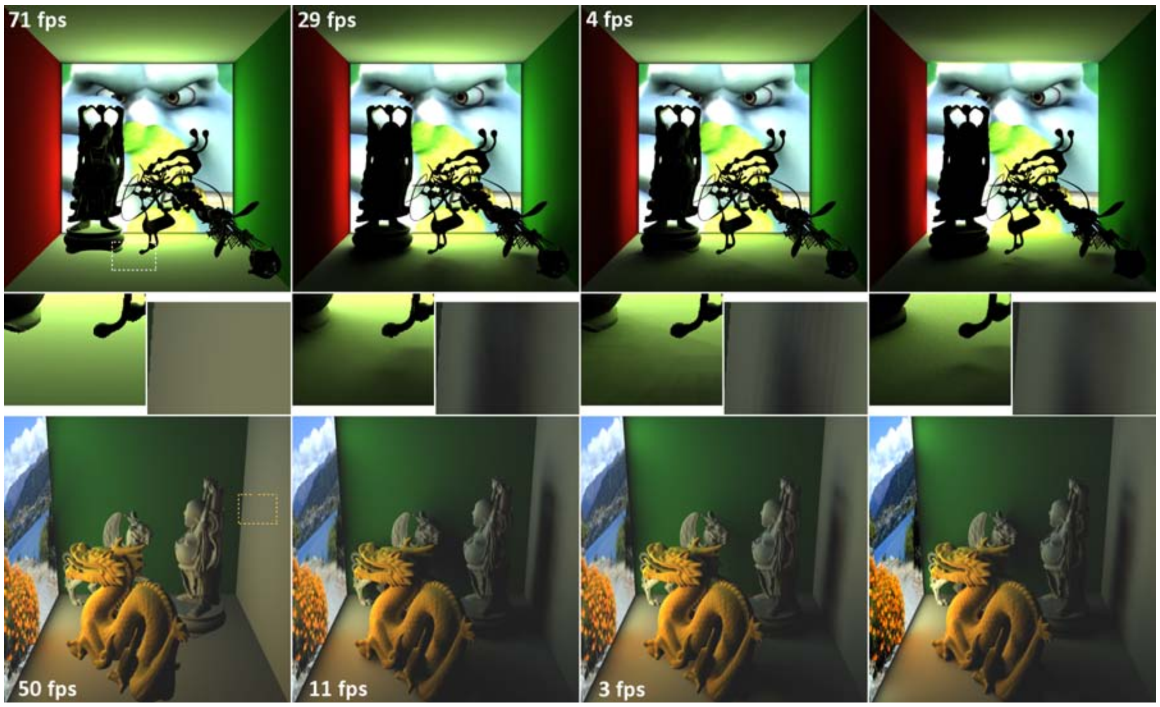


Figure 5.8: Two complex scenes (left) without visibility, using our multiresolution approach; (center left) our incremental visibility; (center right) coarse, 64^2 shadow maps, with quality similar to imperfect shadow maps; and (right) path tracing.

5.4 Results and Discussion

Our timings come from a dual-core 3GHz Pentium 4 and a GeForce GTX 280, using OpenGL. Unless otherwise stated, all images and reported results were for an output resolution of 1024^2. Key GPU functionality required includes integer textures (for voxelization) and early stencil culling (to cull unnecessary fragments).

Figure 5.1 demonstrates the realism area sources add to a scene. The top compares a living room with and without our area lighting. Figure 5.6 shows surfaces with a Phong BRDF, comparing our results with and without visibility. The selective stenciling described in Section 5.3.2 allows us to discard many fragments unprocessed, improving performance.

Figure 5.8 compares our work, with and without visibility, to a path traced reference and a rendering using 1024 coarse (64^2) shadow maps, similar to rendering with imperfect shadow maps. We created the coarse shadow maps via traditional rasterization, leading to higher quality but poorer performance than ISMs. In general, our work compares favorably to path tracing, and captures higher shadow frequencies than those possible with coarse shadow maps.

5.4.1 Performance

Ultimately, multiresolution rendering performance depends on refinement quality. Illuminating each fragment requires gathering light from hundreds of VPLs and many visibility queries; thus, performance depends directly on the number of fragments rendered per frame. Because we require only a few passes over the geometry, geometric complexity plays a relatively minor performance role. But like prior multiresolution image-space algorithms, our performance varies with *visual* complexity. Scenes with high frequency details require more rendered fragments with a corresponding performance hit, regardless of polygon count.

Figure 5.9 explores this effect for a moving camera in the Yeah Right scene atop Figure 5.8. While polygon count stays constant, the motion provides varying visual complexity and causes the fragment count to fluctuate each frame. The top graph shows the percentage of fragments rendered from each resolution. At 64^2, we render roughly half, or 2000, of the fragments. Finer resolutions cheaply interpolate and

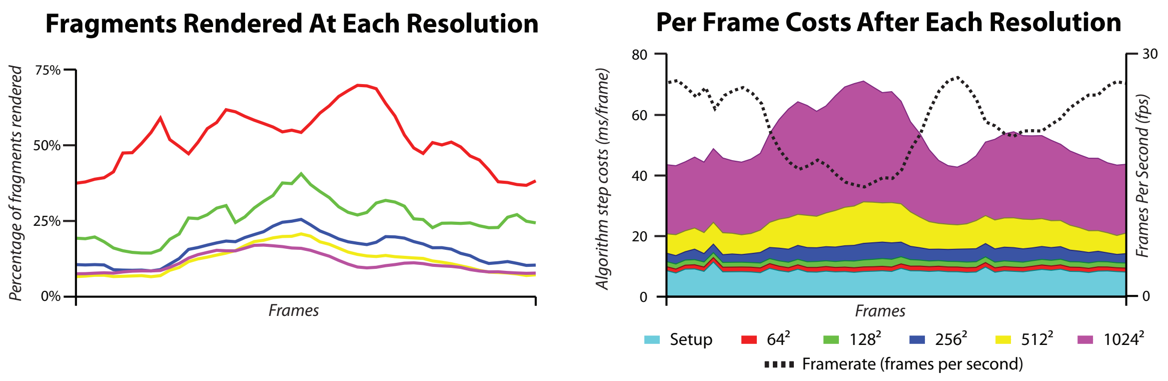


Figure 5.9: Percentage of fragments rendered from each resolution (left) and the per-frame costs for each refinement (right) as the camera moves through the “Yeah Right” scene. The rightmost graph also displays the framerate at 1024^2.

reuse many of these, and only 10-15% of the final image needs per-pixel computation.

While we process a small *percentage* at high resolution, 10% of 1024^2 is significantly larger than 50% of 64^2. The bottom graph of Figure 5.9 reveals the cost for each refinement level. The quantity of fragments at higher resolutions still contribute much of our overall cost. To improve speed, some applications might render at only 256^2 or 512^2 and apply edge-preserving bilateral filters to upsample for final display.

As refinement thresholds directly affect fragment count, they greatly impact performance, as shown in Figure 5.10. Both strongly affect performance, but the lowest relative threshold has the strongest impact; with a color threshold of 0.005 the normal threshold has a minor impact, whereas the normal threshold dominates for higher color thresholds.

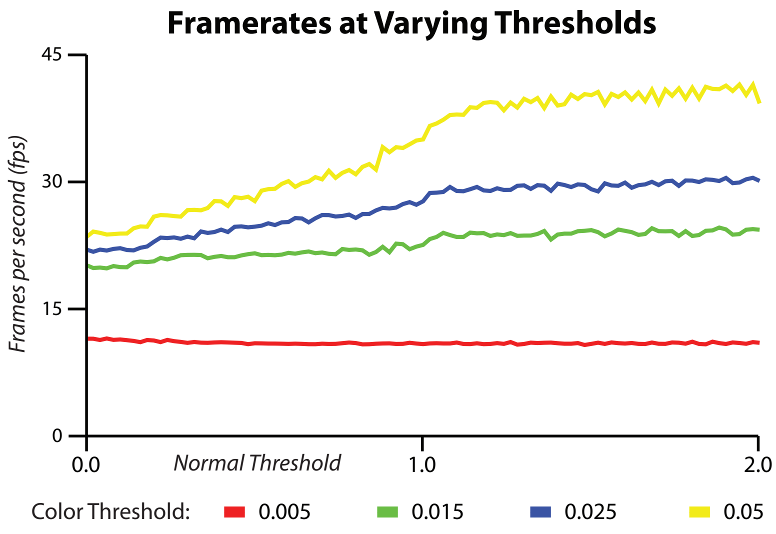


Figure 5.10: Speeds with varied normal and color thresholds.

5.4.2 Voxel Buffer Resolution

We achieve compelling visibility using surprisingly coarse voxelizations. Figure 5.11 shows a closeup of the 755k triangle, genus 131 YeahRight model with six different voxel resolutions. All our results use 128 bits of depth, via a 32-bit per channel buffer. Visibility from a $128^2 \times 128$ buffer is almost indistinguishable to the $1024^2 \times 128$ visibility. While coarser resolutions give noticeably different results, this may be acceptable for large geometry with low-frequency visibility. Additionally, small voxel buffers increase cache coherence during ray traversal and significantly improve performance.

5.4.3 Limitations

Coarse voxelization, especially in the z dimension, leads to self occlusions. This occurs when initial visibility ray steps remain inside voxels representing the

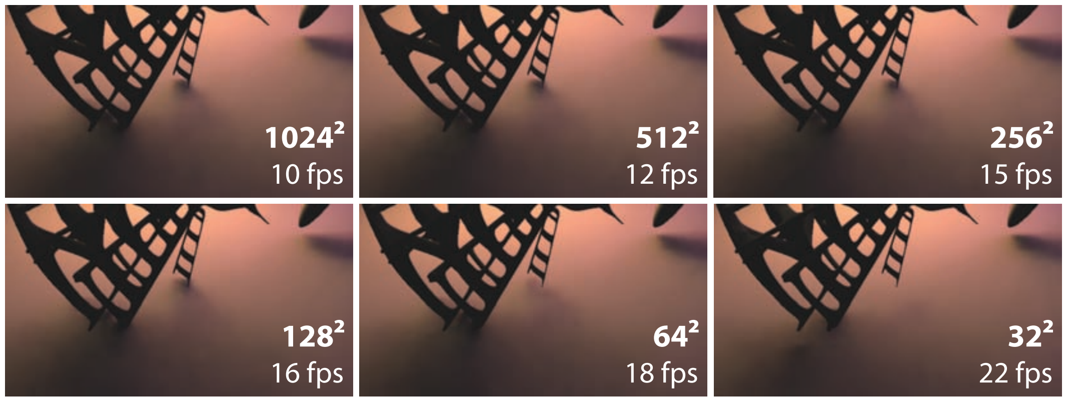


Figure 5.11: The basket of the YeahRight model (also in Figure 5.8) rendered with varying voxel buffer dimensions.

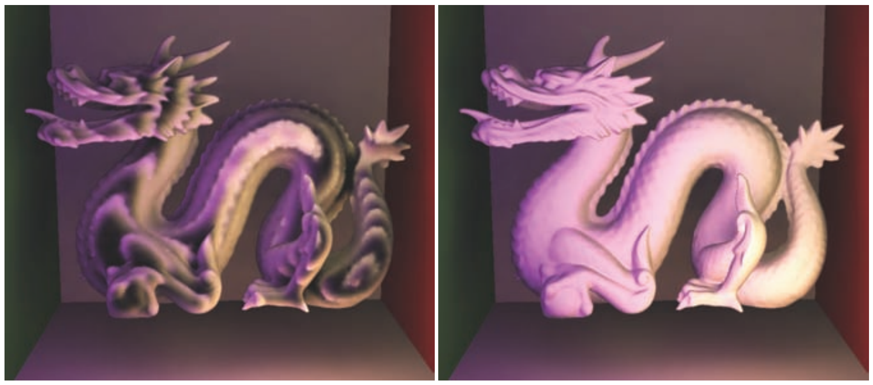


Figure 5.12: A dragon with self-occlusion artifacts due to limited voxel resolution. A bias helps alleviate the problem.

originating surface (see Figure 5.12). This is a variant of the “shadow acne” problem that plagues shadow mapping. We address it similarly, adding a bias to push the ray origin away from the surface.

While reducing self-occlusion errors, a bias causes missed contact shadows for thin geometry. In general, our method yields best results for geometry occupying multiple voxels; such geometry exhibits less aliasing during ray marching. Increasing

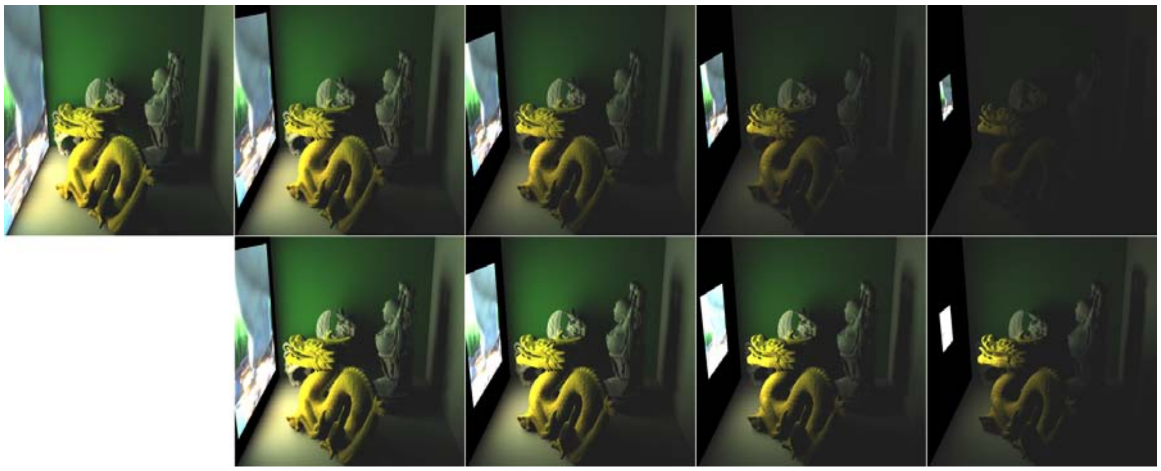


Figure 5.13: Results of our technique as the light size varies. In the top row, the $watts/m^2$ is kept constant, yielding less illumination as the light size decreases; in the bottom row, the overall wattage of the light source remains constant.

steps per visibility query reduces aliasing as does increasing voxel resolution, though both impact performance. Using additional render targets would capture higher fidelity voxels (for up to 2048-bit z-resolution). Though, surprisingly, we found a single 128-bit buffer sufficient for all but the thinnest geometry (e.g., the furniture in Figure 5.3).

As visual complexity dramatically impacts performance, naively adding normal or bump mapping may refine a lot of pixels, degrading performance. We see numerous ways to mitigate this, which we plan to address in future work.

5.4.4 Varying Light Sizes

While we designed our algorithm with large, dynamic area lights in mind, it can be applied to smaller sources, as demonstrated in Figure 5.13. As light size decreases, at some point the rationale for using our work disappears. Our main advantage over more traditional “soft shadow” work (e.g., [129]) is twofold: our avoidance of artifacts from the single silhouette assumption and our capture of lighting variations in unshadowed regions that arise from radiance variations across the light surface. With small area lights, the single silhouette assumption is usually acceptable and a single color light source closely approximates the results for most pixels. In such a scenario, traditional soft shadow algorithms may be faster.

Furthermore, as the light size decreases one would expect the voxelized visibility to become more apparent. This is, in fact, visible for the smaller lights in Figure 5.13, though the 128-bit voxel depth exacerbates the problem. For smaller light sizes, a 256, 512, or 1024-bit voxel depth could easily be added (as in [32]).

5.4.5 Comparison to LPVs

Recently published work introduced the idea of Cascaded Light Propagation Volumes [69]. This work was published after the initial submission of our research, and its still-recent publication date makes it difficult to provide comparison images. Below, we instead provide a brief qualitative comparison.

Basically, Kaplanyan introduces a multiresolution implementation of irradiance volumes, using spherical harmonics to represent illumination and visibility samples at the grid lattice points. These SH coefficients are populated by sampling VPLs from a reflective shadow map [22] and projecting them to a SH basis. Because they compute illumination and visibility only at these lattice points, instead of the hundreds of thousands of fragments we use, their work runs significantly faster than ours.

They also handle participating media, which we do not address.

However, their coarse sampling makes rendering of high frequency illumination and visibility quite difficult. They claim to sample on 32^3 grids, which suggests their aliasing from low volumetric resolution (e.g., poor contact shadows, difficulty with sharp illumination boundaries) will be significantly worse than ours. Our use of a 2D image-space structure suggests we might scale to higher sampling rates than their 3D structure, due to the curse of dimensionality, though we must reconstruct our structure each frame.

Finally, while reliance on a SH-basis enables propagation volumes to easily shade normal mapped surface, they are unable to render high frequency BRDFs or sharp shadows, such as our Phong images from Figure 8.

It may be possible to use a hybrid between our work and theirs, sampling light and visibility to a SH basis in a multiresolution screen space fashion, though this is future work.

5.4.6 Comparison to ISMs

Ritschel et al. [118] proposed imperfect shadow maps (ISMs), the other work closely comparable to ours. Figure 7 compares our work with a rendering using many perfect 64^2 shadow maps. These shadow maps were generated using the traditional rasterization pipeline instead of using the precomputed, uniform sampled points and the subsequent hole filling algorithm proposed for ISMs. This leads to a comparison in Figure 7 with significantly slower performance than ISMs, but higher quality, due to the elimination of aliasing and shadow map holes arising from coarsely sampled points.

Our rendering times are faster than those reported by Ritschel et al. even though we output higher resolution images. However, their speeds may be comparable to ours after accounting for GPU improvements. We likely run slower on high polygon models but faster on lower polygon models, due to ISMs' use of user-specified numbers of point samples.

As far as quality, the coarse resolution of ISMs makes hard contact shadows impossible. While our voxel buffer also leads to errors for contact shadows, we can capture higher frequencies in these regions and a finer voxelization reduces our errors for a modest cost. The banding visible in the shadow map comparison in Figure 7 comes from the low resolution shadow maps, which our per-pixel visibility sampling avoids. As the BRDF becomes more specular, shadow map texel aliasing becomes more objectionable for ISMs as fewer shadow maps are used to average visibility. Conversely, ISMs sample the light surface more densely, giving better quality when high VPL sampling is needed (see Section 5.4.7 below).

One key difference: ISMs require a preprocess to point sample the scene geometry. This somewhat limits geometric changes. Additionally, due to fixed sampling, the samples used to create individual imperfect shadow maps may be outside the light frustum or too sparse on nearby geometry, potentially leaving visible parts of the scene undersampled.

In general, we view ISMs as somewhat orthogonal to our work. As stated in

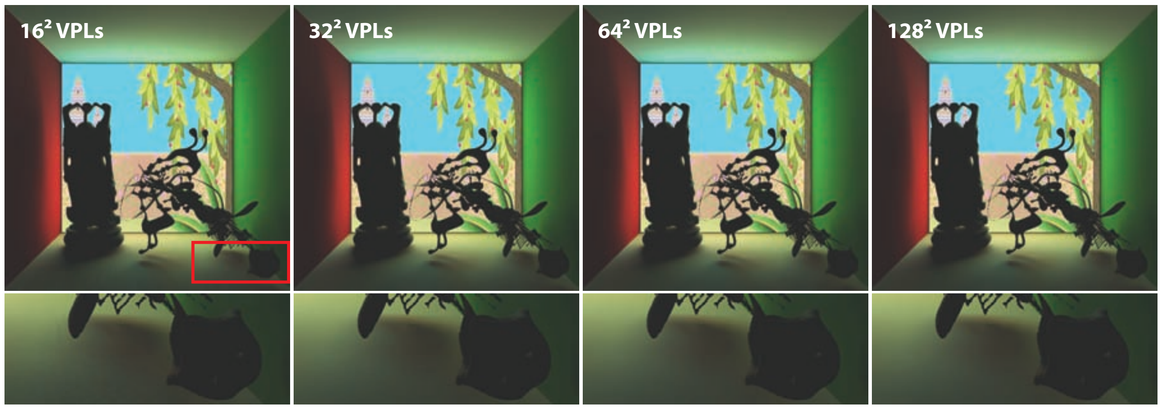


Figure 5.14: A scene using 16^2, 32^2, 64^2, and 128^2 VPLs. Increasing VPL sampling generally yields little visible change for diffuse surfaces.

our conclusion, we envision using ISMs in our incremental rendering process instead of ray-marched voxel visibility, combining the strengths of these different techniques.

5.4.7 VPL Sampling

For diffuse and slightly glossy surfaces, our implementation uses a fixed set of 256 VPLs. We found this coarse sampling sufficient; Figure 5.14 compares a scene with varying VPLs counts. There is no perceptible difference between the images (even in a difference image), despite the factor of 64 change in VPL count. While the larger VPL count does not often yield increased quality, it certainly increases cost. A naive implementation scales linearly with VPLs, though the incremental visibility detailed in Section 5.3.3.2 may allow sub-linear scaling. We did not explore scaling issues, as we found 256 VPLs a good quality-performance tradeoff; lower sampling does start introducing banding.

While 256 VPLs suffice for many diffuse scenes, pathological scenarios will

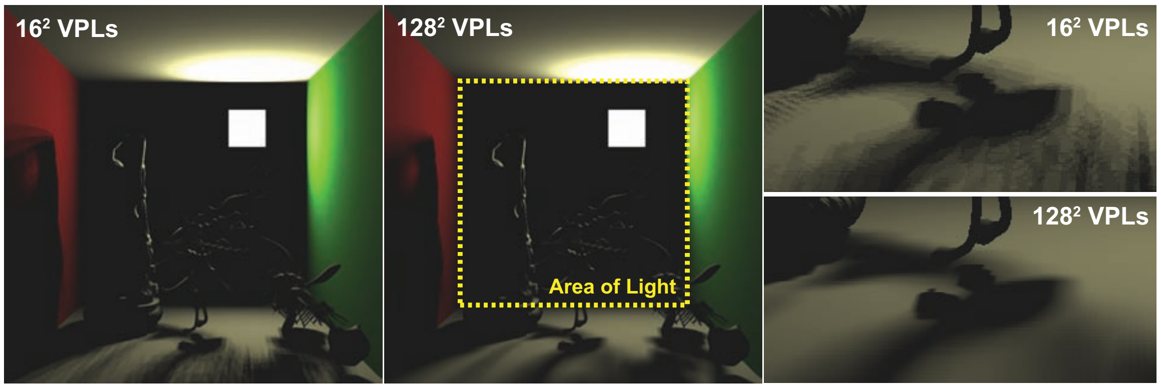


Figure 5.15: A pathological scene demonstrating artifacts from a fixed 16×16 VPL sampling. Here, a video on the back wall has a white square bouncing around a black screen, so most VPLs uselessly represent large black regions on the light.

show artifacts. Figure 5.15 compares 256 and 16,384 VPLs in such a scene; the geometry is identical to Figure 5.14, but the video on the back wall contains a white square bouncing around a mostly black screen. In this case our uniform VPL sampling wastes most samples on black regions of the video and undersamples the square. An adaptive sampling would reduce artifacts, though a better option would simply treat the video as a dynamic square light.

Non-diffuse surfaces complicate matters. Light samples should be focused inside the material's reflection lobe. One approach would be to adaptively sample the light for each fragment, as proposed by Nichols et al. [102]. Instead, as outlined in Section 5.3.2.1, we use the regular sampling where sufficient and adapt to a per-lobe texture sampling scheme when fixed VPL samples become visible.

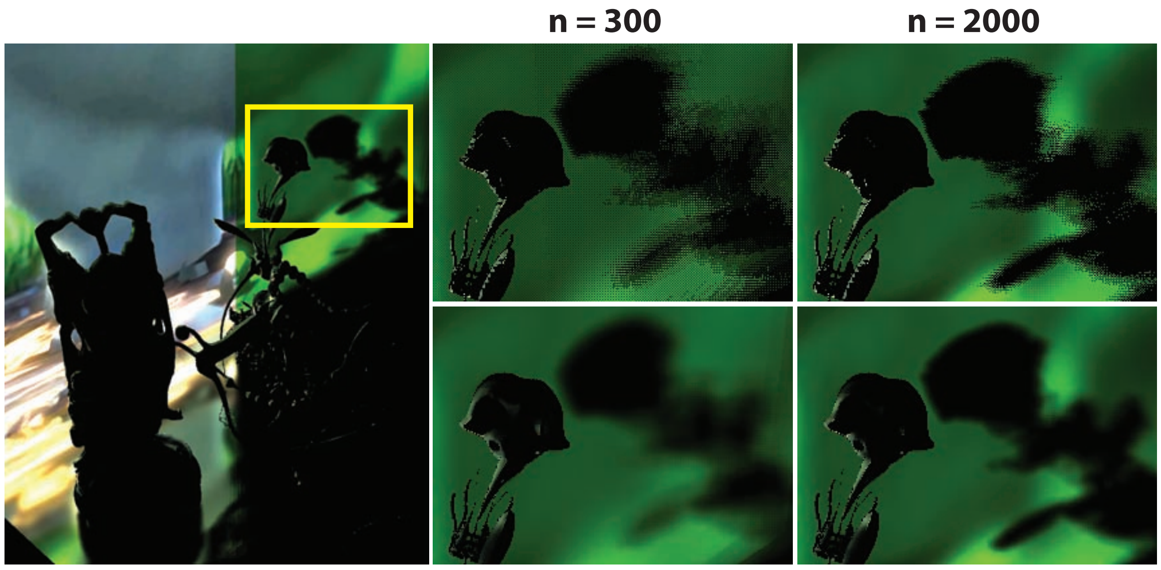


Figure 5.16: With sharp BRDFs, simple binary visibility queries may require many VPLs to produce artifact-free results. Here, the limited set of VPLs produces banding and interleaving artifacts with binary visibility queries (top). Querying using filtered variance values yields much better results (bottom).

5.4.8 Variance Visibility Queries

Figure 5.16 illustrates banding and interleaving artifacts that may arise using binary visibility queries to a small number of VPLs. Applying filtering and using variance queries as described in Section 5.3.3.3 greatly reduces these artifacts.

5.5 Conclusions

We introduced a multiresolution image-space rendering algorithm able to compute direct illumination from dynamic area lights. We described refinement methods to accelerate the rendering of diffuse and non-diffuse surfaces, proposed a method of coarsely approximating visibility using screen space voxelization, and combined them using incremental refinement. We achieve interactive speeds for a variety of scenes, require no precomputation, and impose no restrictions on the light, camera, or geometry.

CHAPTER 6
CONCLUSIONS AND FUTURE WORK

This thesis presented our work towards interactive global effects on the GPU. Chapter 3 described our initial work in this area, a multiresolution image space algorithm that refined indirect illumination into patches, or “subsplats”, and rendered each at the lowest possible resolution. Chapter 4 proposed a more efficient method for achieving identical results by reformulating patch refinement as a parallel process and gathering illumination instead of splatting it. Additionally, this chapter presented two improved methods for improving temporal coherence by applying ideas used in hierarchical radiosity to image space. Finally, Chapter 5 applied similar methods to the computation of direct illumination, also discussing a novel method for computing visibility and an enhanced variant of multiresolution rendering.

Our work contributes a number of interesting ideas towards the pursuit of interactive global illumination. Our techniques operate in image-space, avoiding wasted computation on areas outside the view frustum and limiting the effect of geometric complexity on performance. Unlike comparable techniques such as imperfect shadow maps [119], our work requires no preprocessing and places no limits upon scene dynamism. Stencil-based multiresolution rendering, as introduced in Chapter 4, can greatly accelerate computation of largely low-frequency global illumination effects. Finally, in contrast to techniques like cascaded light propagation volumes [69] that require diffuse or nearly-diffuse surfaces, we show in Chapter 5 that our methods can function with more complex BRDFS – potentially even at higher speeds than diffuse surfaces.

Naturally, other methods carry some advantages over our work. When certain limitations are accepted (i.e. diffuse-only scenes with solely low-frequency lighting), other approximations such as light propagation volumes often prove more efficient. Furthermore, our methods create approximations – sometimes coarse ones – and thus may not be suitable where strict accuracy is required. However, they offer versatility and yields plausible results at interactive speeds. We believe that our work is therefore a useful and worthwhile contribution to the field.

Our results suggest a number of future directions:

1. **Improved BRDF-specific refinement metrics:** Both the indirect and direct illumination methods presented in this thesis would benefit from improved refinement metrics, especially for non-diffuse surfaces. Section 5.3.2.1 presented a method of refining patches for a Phong BRDF, showing that our multiresolution methods can apply to nondiffuse materials in addition to the diffuse surfaces depicted throughout this thesis. Additional research towards refinement for specific BRDFs would increase rendering efficiency, and widen the usefulness of our methods.

2. **Improved visibility computation:** Chapter 5 presented a method of computing visibility by marching rays through coarse screen space voxelization. We demonstrated that this method can achieve quality results at interactive speeds, but ray marching remains quite an expensive operation. The ray tracing community has spent decades researching efficient coherent ray traversal strategies

(some of which are mentioned in Section 2.4.1); we believe that dramatic performance increases could be achieved by applying them to this new context. Additionally, our illumination refinement is orthogonal from our proposed visibility queries; incremental lookups (from Section 5.3.3.2 into other visibility approaches such as imperfect shadow maps [119] may work well.

3. **Improved visibility for direct illumination with complex BRDFs:** The more complex reflection of light from non-diffuse BRDFs simultaneously simplify and complicate their usage with our techniques. With a complex BRDF, more of the illumination at a given point comes from a smaller area of the visible hemisphere; as suggested above, this can simplify and accelerate rendering. With these BRDFs, however, our results suggest that a finer sampling of visibility is required to yield plausible shadows. Rather than uniformly increase visibility sampling, a hierarchical approach (similar to that described in Chapter 4), applied to visibility computation for non-diffuse BRDFs, may produce quality results.

4. **Visibility for indirect illumination:** Our methods for computing indirect illumination from Chapters 3 and 4 do not consider occlusion. While indirect occlusion is arguably less important than occlusion for direct lighting, an efficient approximation for indirect visibility would be interesting future work. Our method of computing occlusion for direct lighting, presented in Section 5.3.3, may work well for this purpose. Investigation into alternate methods of

computing coarse occlusion approximation may also prove worthwhile.

5. **Multi-bounce indirect illumination:** Techniques derived from reflective shadow maps, including those described in this thesis, approximate only a single bounce of indirect illumination. Although this has been shown to be sufficient for many applications [144], others may benefit from multiple bounces of indirect illumination. Even with just an eye-space G-buffer and a light-space reflective shadow map, there may be enough information to generate a coarse approximation of the second and higher bounces; additional research could determine if this is the case, or what additional scene information would be required to efficiently approximate additional bounces.

6. **Temporal caching:** Each frame, our techniques completely recompute all illumination. Given that illumination often changes slowly temporally as well as spatially, there may be additional performance or quality to be gained from caching and reusing results from previous frames. While temporal caching approaches tend to complicate scene dynamism, much recent work [53, 98, 127] has focused on alleviating these problems. Combining these ideas with our multiresolution approach may prove worthwhile.

7. **Additional applications:** Our multiresolution methods provide a flexible base for rendering other global effects, such as caustics, participating media, volumetric caustics and shadows, etc. Some parts of our work have further applicability: for example, we think that screen-space voxelization may yield superior results

when applied to screen space ambient occlusion.

In summary, this dissertation presented three multiresolution image-space techniques for interactively computing global effects on the GPU. Each of these techniques yields plausible results at interactive speeds, taking a small step closer to the goal of completely interactive photorealistic rendering.

REFERENCES

[1] Maneesh Agrawala, Ravi Ramamoorthi, Alan Heirich, and Laurent Moll. Efficient image-based methods for rendering soft shadows. In *Proceedings of SIGGRAPH*, pages 375–384, 2000.

[2] Timo Aila and Samuli Laine. Understanding the efficiency of ray traversal on gpus. In *Proc. High-Performance Graphics 2009*, 2009.

[3] Thomas Annen, Zhao Dong, Tom Mertens, Philippe Bekaert, Hans-Peter Seidel, and Jan Kautz. Real-time, all-frequency shadows in dynamic scenes. *ACM Trans. Graph.*, 27(3), 2008.

[4] Thomas Annen, Tom Mertens, Philippe Bekaert, Hans-Peter Seidel, and Jan Kautz. Convolution shadow maps. In *Proc. Eurographics Symposium on Rendering*, pages 51–60, 2007.

[5] Arthur Appel. The notion of quantitative invisibility and machine rendering of solids. In *Proceedings of the ACM National Conference*, pages 387–393, 1967.

[6] James Arvo. Backward ray tracing. *Developments in Ray Tracing*, pages 259–263, 1986. ACM SIGGRAPH Course Notes.

[7] James Arvo and David Kirk. A survey of ray tracing acceleration techniques. pages 201–262, 1989.

[8] Michael Ashikhmin and Peter Shirley. An anisotropic phong brdf model. *Journal of Graphics Tools*, 5:25–32, 2000.

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

[10] Louis Bavoil and Miguel Sainz. Multi-layer dual-resolution screen-space ambient occlusion. In *SIGGRAPH Talks*, 2009.

[11] Michael Bunnell. Dynamic ambient occlusion and indirect lighting. In *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation*, pages 223–233. Addison-Wesley, 2005.

[12] William Elwood Byerly. *An Elementary Treatise on Fourier's Series and Spherical, Cylindrical, and Ellipsoidal Harmonics, with Applications to Problems in Mathematical Physics.* Ginn and Company, 1893.

[13] Nathan A. Carr, Jared Hoberock, Keenan Crane, and John C. Hart. Fast gpu ray tracing of dynamic meshes using geometry images. In *Proceedings of Graphics Interface*, pages 203–209, 2006.

[14] Francesc Castro, László Neumann, and Mateu Sbert. Extended ambient term. *J. Graph. Tools*, 5(4):1–7, 2000.

[15] Shenchang Eric Chen. Incremental radiosity: An extension of progressive radiosity to an interactive image synthesis system. In *Proceedings of SIGGRAPH*, pages 135–144, 1990.

[16] Michael Cohen, Shenchang Eric Chen, John Wallace, and Donald Greenburg. A progressive refinement approach to fast radiosity image generation. In *Proceedings of SIGGRAPH*, pages 75–84, 1988.

[17] Michael F. Cohen and John R. Wallace. *Radiosity and Realistic Image Synthesis.* Academic Press Professional, Boston, MA, 1993.

[18] Steven Collins. Adaptive splatting for specular to diffuse light transport. In *In Fifth Eurographics Workshop on Rendering*, pages 119–135, 1994.

[19] Robert Cook, Thomas Porter, and Loren Carpenter. Distributed ray tracing. In *Proceedings of SIGGRAPH*, pages 137–145, 1984.

[20] Robert L. Cook and Kenneth E. Torrance. A reflectance model for computer graphics. In *Proceedings of SIGGRAPH*, pages 307–316, 1981.

[21] Franklin Crow. Shadow algorithms for computer graphics. In *Proceedings of SIGGRAPH*, pages 242–248, 1977.

[22] Carsten Dachsbacher and Marc Stamminger. Reflective shadow maps. In *Proceedings of the Symposium on Interactive 3D Graphics and Games*, pages 203–231, 2005.

[23] Carsten Dachsbacher and Marc Stamminger. Splatting indirect illumination. In *Proceedings of the Symposium on Interactive 3D Graphics and Games*, pages 93–100, 2006.

[24] Kurt Debattista, Piotr Dubla, Francesco Banterle, Luís Paulo Santos, and Alan Chalmers. Instant caching for interactive global illumination. *Computer Graphics Forum*, pages 2216–2228, 2009.

[25] Michael Deering, Stephanie Winner, Bic Schediwy, Chris Duffy, and Neil Hunt. The triangle processor and normal vector shader: a vlsi system for high performance graphics. In *SIGGRAPH '88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, pages 21–30, New York, NY, USA, 1988. ACM.

[26] Kirill Dmitriev, Vlastimil Havran, and Hans-Peter Seidel. Faster ray tracing with simd shaft culling. Research Report MPI-I-2004-4-006, Max-Planck-Institut für Informatik, Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany, December 2004.

[27] Zhao Dong, Wei Chen, Hujun Bao, Hongxin Zhang, and Qunsheng Peng. Real-time voxelization for complex polygonal models. In *Proceedings of Pacific Graphics*, pages 43–50, 2004.

[28] William Donnelly and Andrew Lauritzen. Variance shadow maps. In *Symposium on Interactive 3D Graphics and Games*, pages 161–165, 2006.

[29] Philip Dutre and Yves Willems. Importance-driven monte carlo light tracing. In *Proceedings of the Eurographics Rendering Workshop*, pages 188–197, 1994.

[30] Philip Dutre and Yves Willems. Potential-driven monte carlo particle tracing for diffuse environments with adaptive probability functions. In *Proceedings of the Eurographics Rendering Workshop*, pages 306–315, 1995.

[31] Elmar Eisemann and Xavier Décoret. Fast scene voxelization and applications. In *Proceedings of the Symposium on Interactive 3D Graphics and Games*, pages 71–78, 2006.

[32] Elmar Eisemann and Xavier Décoret. Single-pass gpu solid voxelization for real-time applications. In *Proceedings of graphics Interface*, pages 73–80, 2008.

[33] Cass Everitt. Interactive order-independent transparency. Technical report, nVidia, http://developer.nvidia.com/object/interactive_order_transparency.html, 2001.

[34] Randima Fernando. Percentage-closer soft shadows. In *SIGGRAPH Sketches*, 2005.

[35] Robert B. Fisher. *From surfaces to objects: computer vision and three dimensional scene analysis.* John Wiley & Sons, Inc., 1989.

[36] A. Fujimoto, Takayuki Tanaka, and K. Iwata. ARTS: accelerated ray-tracing system. pages 148–159, 1988.

[37] Pascal Gautron. Temporal radiance caching. In *ACM SIGGRAPH classes*, pages 1–49, 2008.

[38] Pascal Gautron, Jaroslav Křivánek, Kadi Bouatouch, and Sumanta N. Pattanaik. Radiance cache splatting: A GPU-friendly global illumination algorithm. In *Proceedings of the Eurographics Symposium on Rendering*, pages 55–64, 2005.

[39] Cindy M. Goral, Kenneth E. Torrance, Donald P. Greenberg, and Bennett Battaile. Modeling the interaction of light between diffuse surfaces. *SIGGRAPH Comput. Graph.*, 18(3):213–222, 1984.

[40] Henri Gouraud. Continuous shading of curved surfaces. *IEEE Transactions on Computers*, 20(6):623–629, June 1971.

[41] Paul Green, Jan Kautz, and Frédo Durand. Efficient reflectance and visibility approximations for environment map rendering. *Computer Graphics Forum (Proc. EUROGRAPHICS)*, 26(3):495–502, 2007.

[42] Ned Greene, Michael Kass, and Gavin Miller. Hierarchical z-buffer visibility. In *Proceedings of SIGGRAPH*, pages 231–238, 1993.

[43] Gene Greger, Peter Shirley, Philip Hubbard, and Donald Greenberg. The irradiance volume. *IEEE Computer Graphics & Applications*, 18(2):32–43, March-April 1998.

[44] Gaël Guennebaud, Loïc Barthe, and Mathias Paulin. Real-time soft shadow mapping by backprojection. In *Proceedings of the Eurographics Symposium on Rendering*, pages 227–234, 2006.

[45] Johannes Günther, Stefan Popov, Hans-Peter Seidel, and Philipp Slusallek. Realtime ray tracing on gpu with bvh-based packet traversal. In Alexander Keller and Per Christensen, editors, *IEEE/Eurographics Symposium on Interactive Ray Tracing 2007*, pages 113–118. IEEE, Eurographics, 2007.

[46] Toshiya Hachisuka. High-quality global illumination using rasterization. *GPU Gems 2*, pages 615–634, 2005.

[47] Pat Hanrahan, David Salzman, and Larry Aupperle. A rapid hierarchical radiosity algorithm. In *Proceedings of SIGGRAPH*, 1991.

[48] Jean-Marc HasenFratz, Marc Lapierre, Nicolas Holzschuch, and Francois Sillion. A survey of real-time soft shadow algorithms. *Computer Graphics Forum*, 22(4):753–774, 2003.

[49] Vlastimil Havran. *Heuristic Ray Shooting Algorithms*. Ph.d. thesis, Department of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague, 2000.

[50] Xiao He, Kenneth Torrance, Francois Sillion, and Donald Greenberg. A comprehensive physical model for light reflection. In *Proceedings of SIGGRAPH*, pages 175–186, 1991.

[51] Paul Heckbert and Michael Herf. Simulating soft shadows with graphics hardware. Technical Report CMU-CS-97-104, Carnegie Mellon University, January 1997.

[52] Paul S. Heckbert and Pat Hanrahan. Beam tracing polygonal objects. In *Proceedings of SIGGRAPH*, pages 119–127, 1984.

[53] Robert Herzog, Elmar Eisemann, Karol Myszkowski, and Hans-Peter Seidel. Spatio-temporal upsampling on the gpu. In *Proceedings of ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D) 2010*. ACM, 2010.

[54] Robert Herzog, Vlastimil Havran, Shinichi Kinuwaki, Karol Myszkowski, and Hans-Peter Seidel. Global illumination using photon ray splatting. *Computer Graphics Forum*, 26(3):503–513, 2007.

[55] Robert Herzog, Karol Myszkowski, and Hans-Peter Seidel. Anisotropic radiance-cache splatting for efficiently computing high-quality global illumination with lightcuts. In Marc Stamminger and Philip Dutré, editors, *Computer Graphics Forum (Proc. Eurographics)*, volume 28(2), pages 259–268, München, Germany, 2009. Wiley-Blackwell.

[56] Daniel Reiter Horn, Jeremy Sugerman, Mike Houston, and Pat Hanrahan. Interactive k-d tree gpu raytracing. In *I3D '07: Proceedings of the 2007 symposium on Interactive 3D graphics and games*, pages 167–174, 2007.

[57] Helen Hu, Amy Gooch, William Thompson, Brian Smits, J. Rieser, and Peter Shirley. Visual cues for imminent object contact in realistic virtual environments. In *Proceedings of Visualization*, pages 127–136, 2000.

[58] Warren Hunt and William R. Mark. Adaptive acceleration structures in perspective space. In *Proceedings of the Symposium on Interactive Ray Tracing*, pages 11–17, 2008.

[59] Yeon-Ho Im, Chang-Young Han, and Lee-Sup Kim. A method to generate soft shadows using a layered depth image and warping. *IEEE Trans. Vis. Comput. Graph.*, 11(3):265–272, 2005.

[60] Andrey Iones, Anton Krupkin, Mateu Sbert, and Sergey Zhukov. Fast, realistic lighting for video games. *IEEE Comput. Graph. Appl.*, pages 54–64, 2003.

[61] K Iwasaki, Y Dobashi, F Yoshimoto, and T Nishita. Precomputed radiance transfer for dynamic scenes taking into account light interreflection. In *Proceedings of the Eurographics Symposium on Rendering*, pages 35–44, 2007.

[62] Henrik Wann Jensen. Importance driven path tracing using the photon map. In *Proceedings of the Eurographics Rendering Workshop*, pages 326–335, 1995.

[63] Henrik Wann Jensen. Rendering caustics on non-lambertian surfaces. *Computer Graphics Forum*, 16(1):57–64, 1997.

[64] Henrik Wann Jensen and Niels Jørgen Christensen. Efficiently rendering shadows using the photon map. In *Proceedings of Compugraphics*, pages 285–291, December 1995.

[65] Henrik Wann Jensen and Niels Jørgen Christensen. Photon maps in bidirectional monte carlo ray tracing of complex objects. *Computers & Graphics*, 19(2):215–224, March 1995.

[66] Henrik Wann Jensen and Per H. Christensen. Efficient simulation of light transport in scenes with participating media using photon maps. In *Proceedings of SIGGRAPH*, pages 311–320, 1998.

[67] James Kajiya and Brian Von Herzen. Ray tracing volume densities. In *Proceedings of SIGGRAPH*, pages 165–174, 1984.

[68] James T. Kajiya. The rendering equation. In *SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pages 143–150, New York, NY, USA, 1986. ACM.

[69] Anton Kaplanyan and Carsten Dachsbacher. Cascaded light propagation volumes for real-time indirect illumination. In *Proceedings of the 2010 symposium on Interactive 3D graphics and games*, 2010.

[70] J. Kautz, J. Lehtinen, and T. Aila. Hemispherical rasterization for self-shadowing of dynamic objects. *Rendering Techniques 2004 (Proceedings of the Eurographics Symposium on Rendering)*, pages 179–184, 2004.

[71] Jan Kautz, Jaakko Lehtinen, and Peter-Pike Sloan. Pre-computed radiance transfer: theory and practice. 2005.

[72] Timothy L. Kay and James T. Kajiya. Ray tracing complex scenes. In *SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pages 269–278, New York, NY, USA, 1986. ACM.

[73] M. J. Keates and R.J. Hubbold. Interactive ray tracing on a virtual shared-memory parallel. *Computer Graphics Forum*, 14:189–202, 1995.

[74] Alexander Keller. Quasi-monte carlo radiosity, 1996.

[75] Alexander Keller. Instant radiosity. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 49–56, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[76] Alexander Keller and Wolfgang Heidrich. Interleaved sampling. In *Proceedings of the Eurographics Workshop on Rendering*, pages 269–276, 2001.

[77] Hyunwoo Ki and Kyoungsu Oh. A gpu-based light hierarchy for real-time approximate illumination. *Visual Computer*, 24(7–9):649–658, 2008.

[78] A. J. KOK. Grouping of patches in progressive radiosity. In *Proceedings of the Eurographics Workshop on Rendering*, 1993.

[79] Janne Kontkanen and Samuli Laine. Ambient occlusion fields. In *I3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games*, pages 41–48, New York, NY, USA, 2005. ACM.

[80] Janne Kontkanen, Emmanuel Turquin, Nicolas Holzschuch, and François Sillion. Wavelet radiance transport for interactive indirect lighting. In *Proceedings of the Eurographics Symposium on Rendering*, pages 161–171, 2006.

[81] Johannes Kopf, Michael Cohen, Dani Lischinski, and Matt Uyttendaele. Joint bilateral upsampling. *ACM Transactions on Graphics*, 2007.

[82] Anders Wang Kristensen, Tomas Akenine-Möller, and Henrik Wann Jensen. Precomputed local radiance transfer for real-time lighting design. *ACM Trans. Graph.*, 24(3):1208–1215, 2005.

[83] Jaroslav Krivanek and Pascal Gautron. *Practical Global Illumination with Irradiance Caching*. Morgan and Claypool, 2009.

[84] Jaroslav Krivanek, Pascal Gautron, Sumanta Pattanaik, and Kadi Bouatouch. Radiance caching for efficient global illumination computation. *IEEE Transactions on Visualization and Computer Graphics*, 11(5):550–561, 2005.

[85] Jaroslav Křivánek, Kadi Bouatouch, Sumanta N. Pattanaik, and Jiří Žára. Making radiance and irradiance caching practical: Adaptive caching and neighbor clamping. In *Rendering Techniques 2006, Eurographics Symposium on Rendering*, June 2006.

[86] Eric Lafortune and Yves Willems. Bi-directional path tracing. In *Proceedings of Compugraphics*, pages 145–153, 1993.

[87] Eric P. F. Lafortune, Sing-Choong Foo, Kenneth E. Torrance, and Donald P. Greenberg. Non-linear approximation of reflectance functions. In *Proceedings of SIGGRAPH*, pages 117–126, 1997.

[88] David Laur and Pat Hanrahan. Hierarchical splatting: a progressive refinement algorithm for volume rendering. In *Proceedings of SIGGRAPH*, pages 285–288, 1991.

[89] Jaakko Lehtinen and Jan Kautz. Matrix radiance transfer. In *Proceedings of the 2003 symposium on Interactive 3D graphics*, pages 59–64, 2003.

[90] Jaakko Lehtinen, Matthias Zwicker, Emmanuel Turquin, Janne Kontkanen, Frédo Durand, François Sillion, and Timo Aila. A meshless hierarchical representation for light transport. *ACM Transactions on Graphics*, 27(3):1–9, 2008.

[91] Fang Liu, Meng-Cheng Huang, Xue-Hui Liu, and En-Hua Wu. Efficient depth peeling via bucket sort. In *Proceedings of High Performance Graphics*, pages 51–57, 2009.

[92] Mattias Malmer, Fredrik Malmer, Ulf Assarsson, and Nicolas Holzschuch. Fast precomputed ambient occlusion for proximity shadows. *Journal of Graphics Tools*, 12(2):59–71, 2007.

[93] Morgan McGuire and David Luebke. Hardware-accelerated global illumination by image space photon mapping. In *Proceedings of the Conference on High Performance Graphics*, pages 77–89, New York, NY, USA, 2009. ACM.

[94] Alex Mendez, Mateu Sbert, Jordi Cata, Nico Sunyer, and Sergi Funtane. Real-time obscurances with color bleeding. In Wolfgang Engel, editor, *ShaderX4: Advanced Rendering Techniques*. Charles River Media, 2005.

[95] Don P. Mitchell. Generating antialiased images at low sampling densities. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 65–72, New York, NY, USA, 1987. ACM.

[96] Martin Mittring. Finding next gen: Cryengine 2. In *SIGGRAPH '07: ACM SIGGRAPH 2007 courses*, pages 97–121, New York, NY, USA, 2007. ACM.

[97] National Bureau of Standards. Monte carlo method. In A. S. Housholder, editor, *Applied Mathematics Series 12*, 1951.

[98] Diego Nehab, Pedro V. Sander, Jason Lawrence, Natalya Tatarchuk, and John R. Isidoro. Accelerating real-time shading with reverse reprojection caching. In *Graphics Hardware*, 2007.

[99] Ren Ng, Ravi Ramamoorthi, and Pat Hanrahan. All-frequency shadows using non-linear wavelet lighting approximation. *ACM Trans. Graph.*, 22(3):376–381, 2003.

[100] Greg Nichols, Rajeev Penmatsa, and Chris Wyman. Interactive, multiresolution image-space rendering for dynamic area lighting. Technical Report UICS-10-01, University of Iowa, April 2010.

[101] Greg Nichols, Jeremy Shopf, and Chris Wyman. Hierarchical image-space radiosity for interactive global illumination. *Computer Graphics Forum*, 28(4):1141–1149, 2009.

[102] Greg Nichols and Chris Wyman. Multiresolution splatting for indirect illumination. In *Proceedings of the Symposium on Interactive 3D Graphics and Games*, pages 83–90, 2009.

[103] Greg Nichols and Chris Wyman. Interactive indirect illumination using adaptive multiresolution splatting. *IEEE Transactions on Visualization and Computer Graphics (to appear)*, 2010.

[104] Fred E. Nicodemus, Joseph C. Richmond, Jack J. Hsia, Irving W. Ginsberg, and Thomas Limperis. *Geometrical considerations and nomenclature for reflectance.* Monograph 160. National Bureau of Standards, October 1977.

[105] Mangesh Nijasure, Sumanta N. Pattanaik, and Vineet Goel. Real-time global illumination on gpus. *Journal of Graphics Tools*, 10(2):55–71, 2005.

[106] Steven Parker, William Martin, Peter-Pike Sloan, Peter Shirley, Brian Smits, and Charles Hansen. Interactive ray tracing. In *Proceedings of the ACM Symposium on Interactive 3D Graphics*, pages 119–126, 1999.

[107] Ingmar Peter, Georg Pietrek, and Fachbereich Informatik. Importance driven construction of photon maps. In *In Rendering Techniques '98 (Proceedings of the 9th Eurographics Workshop on Rendering*, pages 269–280, 1998.

[108] Bui-Thong Phong. Illumination for computer generated images. *Communications of the ACM*, 18:311–317, 1975.

[109] Claude Puech, Francois Sillion, and Christophe Vedel. Improving interaction with radiosity-based lighting simulation programs. In *Proceedings of the Symposium on Interactive 3D graphics*, 1990.

[110] Timothy Purcell, Craig Donner, Mike Cammarano, Henrik Wann Jensen, and Pat Hanrahan. Photon mapping on programmable graphics hardware. In *Proc. Graphics Hardware*, pages 41–50, 2003.

[111] Timothy J. Purcell, Ian Buck, William R. Mark, and Pat Hanrahan. Ray tracing on programmable graphics hardware. *ACM Trans. Graph.*, 21(4):703–712, 2002.

[112] Ravi Ramamoorthi and Pat Hanrahan. An efficient representation for irradiance environment maps. In *Proceedings of SIGGRAPH*, pages 497–500, 2001.

[113] William T. Reeves, David H. Salesin, and Robert L. Cook. Rendering antialiased shadows with depth maps. In *Proceedings of SIGGRAPH*, pages 283–291, 1987.

[114] Zhong Ren, Rui Wang, John Snyder, Kun Zhou, Xinguo Liu, Bo Sun, Peter-Pike Sloan, Hujun Bao, Qunsheng Peng, and Baining Guo. Real-time soft shadows in dynamic scenes using spherical harmonic exponentiation. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*, pages 977–986, New York, NY, USA, 2006. ACM.

[115] Alexander Reshetov, Alexei Soupikov, and Jim Hurley. Multi-level ray tracing algorithm. *ACM Transactions on Graphics*, 24(3):1176–1185, 2005.

[116] T. Ritschel, T. Grosch, J. Kautz, and S. Mueller. Interactive illumination with coherent shadow maps. In *Proceedings of the Eurographics Symposium on Rendering*, 2007.

[117] Tobias Ritschel, Thomas Engelhardt, Thorsten Grosch, Hans-Peter Seidel, Jan Kautz, and Carsten Dachsbacher. Micro-rendering for scalable, parallel final gathering. *ACM Trans. Graph.*, 28(5), 2009.

[118] Tobias Ritschel, Thorsten Grosch, Jan Kautz, and Hans-Peter Seidel. Interactive global illumination based on coherent surface shadow maps. In *Proceedings of Graphics Interface*, pages 185–192, 2008.

[119] Tobias Ritschel, Thorsten Grosch, Min H. Kim, Hans-Peter Seidel, Carsten Dachsbacher, and Jan Kautz. Imperfect shadow maps for efficient computation of indirect illumination. In *Proceedings of SIGGRAPH Asia*, 2008.

[120] Tobias Ritschel, Thorsten Grosch, and Hans-Peter Seidel. Approximating Dynamic Global Illumination in Image Space. In *Proceedings Symposium on Interactive 3D Graphics and Games*, 2009.

[121] Austin Robison. Interactive ray tracing on the gpu and nvirt. Technical report, nVidia, http://www.nvidia.com/research, 2008.

[122] Steven M. Rubin and Turner Whitted. A 3-dimensional representation for fast rendering of complex scenes. In *SIGGRAPH '80: Proceedings of the 7th annual conference on Computer graphics and interactive techniques*, pages 110–116, New York, NY, USA, 1980. ACM.

[123] Reuven Y. Rubinstein and Dirk P. Kroese. *Simulation and the Monte Carlo Method (Wiley Series in Probability and Statistics)*. 1981.

[124] Szymon Rusinkiewicz and Marc Levoy. Qsplat: A multiresolution point rendering system of large meshes. In *Proceedings of SIGGRAPH*, pages 343–352, 2000.

[125] Takafumi Saito and Tokiichiro Takahashi. Comprehensible rendering of 3-d shapes. In *Proceedings of SIGGRAPH*, pages 197–206, 1990.

[126] Hanan Samet. The design and analysis of spatial data structures. Addison-Wesley, 1990.

[127] Daniel Scherzer, Stefan Jeschke, and Michael Wimmer. Pixel-correct shadow maps with temporal reprojection and shadow test confidence. In *Rendering Techniques 2007 (Proceedings Eurographics Symposium on Rendering)*, pages 45–50, 2007.

[128] Arne Schmitz, Markus Tavenrath, and Leif Kobbelt. Interactive global illumination for deformable geometry in cuda. *Computer Graphics Forum*, 27(7):1979–1986, 2008.

[129] Michael Schwarz and Matc Stamminger. Bitmask soft shadows. *Computer Graphics Forum*, 26(3):515–524, 2007.

[130] Jonathan Shade, Steven Gortler, Li wei He, and Richard Szeliski. Layered depth images. In *Proceedings of SIGGRAPH*, pages 231–242, 1998.

[131] Musawir Shah, Jaakko Konttinen, and Sumantra Pattanaik. Caustics mapping: An image-space technique for real-time caustics. *IEEE Trans. Vis. Comput. Graph.*, 2007.

[132] Perumaal Shanmugam and Okan Arikan. Hardware accelerated ambient occlusion techniques on gpus. In *I3D '07: Proceedings of the 2007 symposium on Interactive 3D graphics and games*, pages 73–80, New York, NY, USA, 2007. ACM.

[133] Peter-Pike Sloan. Normal mapping for precomputed radiance transfer. In *I3D '06: Proceedings of the 2006 symposium on Interactive 3D graphics and games*, pages 23–26, New York, NY, USA, 2006. ACM.

[134] Peter-Pike Sloan, Naga Govindaraju, Derek Nowrouzezahrai, and John Snyder. Image-based proxy accumulation for real-time soft global illumination. In *Proceedings of Pacific Graphics*, pages 97–105, 2007.

[135] Peter-Pike Sloan, Jesse Hall, John Hart, and John Snyder. Clustered principal components for precomputed radiance transfer. *ACM Transactions on Graphics*, 22(3):382–391, 2003.

[136] Peter-Pike Sloan, Jan Kautz, and John Snyder. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 527–536, 2002.

[137] Peter-Pike Sloan, Xingou Liu, Heung-Yeung Shum, and John Snyder. Bi-scale radiance transfer. *ACM Transactions on Graphics*, 22(3):370–375, 2003.

[138] Peter-Pike Sloan, Ben Luna, and John Snyder. Local, deformable precomputed radiance transfer. *ACM Trans. Graph.*, 24(3):1216–1224, 2005.

[139] Brian Smits, James Arvo, and Donald Greenberg. A clustering algorithm for radiosity in complex environments. In *Proc. ACM SIGGRAPH*, pages 435–442, 1994.

[140] Brian Smits, James Arvo, and David Salesin. An importance-driven radiosity algorithm. In *Proceedings of SIGGRAPH*, pages 273–282, 1992.

[141] Cyril Soler, Olivier Hoel, Franck Rochet, Frederic Jay, and Nicolas Holzschuch. Hierarchical Screen Space Indirect Illumination For Video Games. Research Report RR-7162, INRIA, 2009.

[142] Bo Sun and Ravi Ramamoorthi. Affine double- and triple-product wavelet integrals for rendering. *ACM Trans. Graph.*, 28(2):1–17, 2009.

[143] Eric Tabellion and Arnauld Lamorlette. An approximate global illumination system for computer generated films. *ACM Trans. Graph.*, 23(3):469–476, 2004.

[144] Eric Tabellion and Arnauld Lamorlette. An approximate global illumination system for computer generated films. *ACM Transactions on Graphics*, 23(3):469–476, 2004.

[145] Seth Teller and Pat Hanrahan. Global visibility algorithms for illumination computations. In *SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 239–246, 1993.

[146] Art Tevs, Ivo Ihrke, and Hans-Peter Seidel. Maximum mipmaps for fast, accurate, and scalable dynamic height field rendering. In *Proceedings of the Symposium on Interactive 3D graphics and games*, 2008.

[147] Parag Tole, Fabio Pellacini, Bruce Walter, and Donald Greenberg. Interactive global illumination in dynamic scenes. In *Proceedings of SIGGRAPH*, pages 537–546, 2002.

[148] Yu-Ting Tsai and Zen-Chung Shih. All-frequency precomputed radiance transfer using spherical radial basis functions and clustered tensor approximation. In *ACM SIGGRAPH 2006 Papers*, pages 967–976, 2006.

[149] Eric Veach and Leonidas J Guibas. Metropolis light transport. In *Proceedings of SIGGRAPH*, pages 65–76, 1997.

[150] Ingo Wald, Carsten Benthin, Markus Wagner, and Philipp Slusallek. Interactive rendering with coherent ray tracing. In *Computer Graphics Forum (Proceedings of EuroGraphics 2001)*, volume 20, pages 153–164, 2001.

[151] Ingo Wald, Thomas Kollig, Carsten Benthin, Alexander Keller, and Philipp Slusallek. Interactive global illumination using fast ray tracing. In *EGRW '02: Proceedings of the 13th Eurographics workshop on Rendering*, pages 15–24, Aire-la-Ville, Switzerland, Switzerland, 2002. Eurographics Association.

[152] Ingo Wald, William R Mark, Johannes Günther, Solomon Boulos, Thiago Ize, Warren Hunt, Steven G Parker, and Peter Shirley. State of the Art in Ray Tracing Animated Scenes. In *Eurographics 2007 State of the Art Reports*.

[153] John Wallace, Kells Elmquist, and Eric Haines. A ray tracing algorithm for progressive radiosity. In *Proceedings of SIGGRAPH*, pages 335–344, 1989.

[154] Bruce Walter, Adam Arbree, Kavita Bala, and Donald P. Greenberg. Multidimensional lightcuts. *ACM Transactions on Graphics*, 25(3):1081–1088, 2006.

[155] Bruce Walter, George Drettakis, and Steven Parker. Interactive rendering using the render cache. In *Proceedings of the Eurographics Rendering Workshop*, pages 19–30, June 1999.

[156] Bruce Walter, Sebastian Fernandez, Adam Arbree, Kavita Bala, Michael Donikian, and Donald Greenberg. Lightcuts: a scalable approach to illumination. *ACM Trans. Graph.*, 24(3):1098–1107, 2005.

[157] Rui Wang, Rui Wang, Kun Zhou, Minghao Pan, and Hujun Bao. An efficient gpu-based approach for interactive global illumination. In *Proceedings of ACM SIGGRAPH*, pages 1–8, 2009.

[158] Greg Ward and Paul Heckbert. Irradiance gradients. In *Eurographics Rendering Workshop*, pages 85–98, May 1992.

[159] Gregory J. Ward. Measuring and modeling anisotropic reflection. pages 265–272, 1992.

[160] Gregory J. Ward, Francis M. Rubinstein, and Robert D. Clear. A ray tracing solution for diffuse interreflection. In *Proceedings of SIGGRAPH*, pages 85–92, 1988.

[161] Hank Weghorst, Gary Hooper, and Donald P. Greenberg. Improved computational methods for ray tracing. *ACM Trans. Graph.*, 3(1):52–69, 1984.

[162] Turner Whitted. An improved illumination model for shaded display. *Communications of ACM*, 23(6):343–349, 1980.

[163] Lance Williams. Casting curved shadows on curved surfaces. In *Proceedings of SIGGRAPH*, pages 270–274, 1978.

[164] Andrew Woo, Pierre Poulin, and Alain Fournier. A survey of shadow algorithms. *IEEE Computer Graphics & Applications*, 10(6):13–32, November 1990.

[165] Chris Wyman. Hierarchical caustic maps. In *Proc. ACM Symp. Interactive 3D Graphics*, pages 163–171, 2008.

[166] Chris Wyman and Carsten Dachsbacher. Reducing noise in image-space caustics with variable-sized splatting. *Journal of Graphics Tools*, 13(1):1–17, 2008.

[167] Baoguang Yang, Jieqing Feng, Gael Guennebaud, and Xinguo Liu. Packet-based hierarchal soft shadow mapping. *Computer Graphics Forum*, 28(4):1121–1130, 2009.

[168] Kun Zhou, Qiming Hou, Rui Wang, and Baining Guo. Real-time kd-tree construction on graphics hardware. Technical Report MSR-TR-2008-52, Microsoft, April 2008.

[169] Kun Zhou, Yaohua Hu, Stephen Lin, Baining Guo, and Heung-Yeung Shum. Precomputed shadow fields for dynamic scenes. *ACM Trans. Graph.*, 24(3):1196–1201, 2005.

[170] S. Zhukov, A. Iones, and G. Kronin. An ambient light illumination model. *Rendering Techniques 1998 (Proceedings of the Eurographics Symposium on Rendering)*, pages 44–45, 1998.