MODULAR RADIANCE TRANSFER

by

Bradford J Loos

A dissertation submitted to the faculty of
The University of Utah
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Computing

School of Computing

The University of Utah

May 2015

STATEMENT OF DISSERTATION APPROVAL

The dissertation of **Bradford J Loos**

has been approved by the following supervisory committee members:

Peter-Pike Sloan	, Co-Chair	**11/25/2014**	
		Date Approved	
Charles Hansen	, Co-Chair	**11/25/2014**	
		Date Approved	
Adam Bargteil	, Member	**11/25/2014**	
		Date Approved	
Peter Shirley	, Member	**11/25/2014**	
		Date Approved	
Cem Yuksel	, Member	**11/25/2014**	
		Date Approved	

and by **Ross Whitaker** , Chair /Dean of

the Department/College/School of **Computing**

and by David B. Kieda, Dean of The Graduate School.

ABSTRACT

Real-time global illumination is the next frontier in real-time rendering. In an attempt to generate realistic images, games have followed the film industry into physically based shading and will soon begin integrating global illumination techniques. Traditional methods require too much memory and too much time to compute for real-time use. With Modular and Delta Radiance Transfer we precompute a scene-independent, low-frequency basis that allows us to calculate complex indirect lighting calculations in a much lower dimensional subspace with a reduced memory footprint and real-time execution. The results are then applied as a light map on many different scenes. To improve the low frequency results, we also introduce a novel screen space ambient occlusion technique that allows us to generate a smoother result with fewer samples. These three techniques, low and high frequency used together, provide a viable indirect lighting solution that can be run in milliseconds on today's hardware, providing a useful new technique for indirect lighting in real-time graphics.

CONTENTS

ACKNOWLEDGMENTS

CHAPTER 1

INTRODUCTION

Indirection illumination is a highly desirable feature of realism in computer generated imagery. However, its global nature makes it difficult to compute in real-time. Despite the difficulty, many games are currently attempting to produce global illumination solutions using varying techniques. Most are simple techniques, unlike the fully dynamic systems talked about in academia. Games generally use precomputed global illumination stored in light maps (otherwise known as static baked global lighting) or ambient occlusion as well as placing nonshadowing static lights to fill in areas of shadow.

In live-action films, reflectors and bounce cards are used to reflect additional light back into the scene. These planes and simple shapes only approximate indirect light from geometry in the real-world, but offer a high level of control that allow directors of photography to produce desired results. In digital film production, simple-shaped lights are commonly used to allow artists to quickly iterate and achieve a desired look. The ease-of-use and controllability of these approximations outweighs their physically incorrect nature.

We believe that by following this insight we have created a new, simple, global illumination method that could be applied to current games, where global illumination is to be decomposed and calculated on simple shapes and then reconstructed to generate pleasing visual results. We have developed a technique that creates a set of bases in which we calculate indirect lighting. Our approach is very efficient and uses very little data and a quick, one-time, *scene-independent* precomputation step. It also allows real-time computation of approximate indirect light and is designed with rapid iteration of light design in mind. We precompute light transport operators (LTOs) for a handful of simple canonical "shapes," then interactively warp and combine these shapes, along with their LTOs, to more complex geometry. These shape proxies are used to model direct-to-indirect transport, which is then applied as a light map to the actual scene geometry. The flow of indirect light between shapes is modeled with lightfields, and all computations are performed in very low-dimensional subspaces (Chapter 3: Modular Radiance Transfer). We generate another set of LTOs to capture the

finer details onto, off of, and blocked by objects within our dictionary (Chapter 5: Delta Radiance Transfer). To improve the visual results of these low-frequency calculations we also introduce a novel screen space ambient occlusion method called Volumetric Obscurance (Chapter 6) that, when paired with MRT and DRT, generates visually plausible, real-time (on the order of milliseconds) indirect illumination.

Modular Radiance Transfer (MRT), Delta Radiance Transfer (DRT), and Volumetric Obscurance (VO) result in plausible, dynamic global-illumination effects, rendering at high frame rates with low memory overhead. Our methods have been shown to scale from high-end to mobile platforms and, like precomputed radiance transfer (PRT), provide smooth results that respond to dynamic changes in lighting.

CHAPTER 2

BACKGROUND

One of the driving forces in computer graphics has been an attempt to generate photorealistic images. While there are many different aspects that are required to generate photorealistic images, we are mostly concerned with research in realistic lighting. While the physics – at least at the geometric optics level – are well understood, solving the equations for complicated scenes requires a large amount of processing power. Due to this, many researchers have attempted to generate new methods that approximate the physical solutions.

2.1   Radiometry

To generate an image with realistic lighting we must begin with the physics of light, using radiometry. Radiometry is a set of techniques in optics to measure electromagnetic radiation.

The basic unit of radiometry is the photon. The photon is a small, indivisible unit of energy defined based on the wavelength of the light:

$$e_\lambda = \frac{hc}{\lambda} \tag{2.1}$$

where λ is the particular wavelength, $h \approx 6.63 \cdot 10^{-34} J \cdot s$ is the Planck constant, and c is the speed of light in a vacuum (defined to be 299,792,458 ms).

The next term to define is *spectral radiant energy* for a specific wavelength, which is defined as the number of photons times the energy contained in each one.

$$Q_\lambda = n_\lambda c_\lambda = \frac{nhc}{\lambda} \tag{2.2}$$

In computer graphics, we are concerned with all visible wavelengths of light (roughly 400nm to 700nm), which means we need to integrate over all visible wavelengths:

$$Q = \int_{\lambda=400nm}^{700nm} Q_\lambda d\lambda \tag{2.3}$$

Traditionally, lighting is calculated three times, once each for red, green, and blue. This is mostly due to the fact that traditional output devices use these three colors as primaries. This breakdown is also similar to the long, medium, and short cones in the human eye.

The next term that is defined in radiometry is *Radiant Flux*, the change in radiant energy per unit time:

$$\Phi = \frac{dQ}{dt} \tag{2.4}$$

which is often just called *flux*.

Radiometry also defines *radiant flux area density* as the differential flux on surface per differential area:

$$E(x) = \frac{d\Phi}{dA} \tag{2.5}$$

This equation is used to define two quantities, *radiant exitance* and *irradiance*. Radiant exitance is the amount of light leaving a differential area of a surface in all directions (also known as Radiosity B § 2.4.2), while irradiance is defined as the amount of flux per differential area impinging on the surface from all angles.

The measure of flux coming from a single direction (ω) is called the *radiant intensity*

$$I(\omega) = \frac{d\Phi}{d\omega} \tag{2.6}$$

where unit direction is measured in units of steradians, an SI unit used to measure solid angle defined as the surface of the unit sphere in which we are interested.

If we measure the flux coming from a single direction over a small differential area, this is known as the radiance at a given point on the surface. Mathematically, radiance leaving point x in direction ω ($L(x,\omega)$) is defined as

$$L(x,\omega) = \frac{d^2\Phi}{cos\theta dAd\omega} \tag{2.7}$$

Radiance is arguably the most important quantity in global illumination since it most closely represents the color of an object [1]. It defines the photons coming off a given surface point leaving in a given direction. These photons activate receptors in the eye, allowing us to see images in our environment. Radiance is the value that we compute when generating light transport between surfaces in a vacuum.

When photons hit a surface, they are either scattered or absorbed. The number of photons that are reflected is based on different characteristics of the surface. The relationship between incoming irradiance and outgoing radiance at a surface point is defined to be the Bidirectional Reflectance Distribution Function (BRDF). Introduced by Nicodemus et al. [2], this allows us to define radiance leaving a surface (L_r) in a slightly different fashion:

$$f_r(x, \omega_i, \omega_o) = \frac{dL_r(x, \omega_o)}{dE_i(x, \omega_i)} = \frac{dL_r(x, \omega_o)}{L_i(x, \omega_i)(\omega_i \cdot n)d\omega_i} \tag{2.8}$$

When we rearrange this function to solve for the radiance leaving the surface, we get

$$L_r(x, \omega) = \int_{\Omega_i} f_r(x, \omega_i, \omega_o)L_i(x, \omega_i)(\omega_i \cdot n)d\omega_i \tag{2.9}$$

When we add the light emitted from the point x, this rearrangement gives us what is generally known as the rendering equation

2.2   The Rendering Equation

While there are many different formulations of the rendering equation, in this work we are concerned with the vacuum, time-invariant, gray radiance equation, or VTIGRE [3].

The VTIGRE version of the rendering equation was originally introduced to computer graphics by Kajiya [4] and was defined as

$$I(x, x') = g(x, x')\left[e(x, x') + \int_S \rho(x, x', x'')I(x', x'')dx''\right] \tag{2.10}$$

where

$I(x, x')$	is related to the intensity of light passing from point x' to x
$g(x, x')$	is a "geometry" term
$e(x, x')$	is related to the intensity of emitted light from x' to x
$\rho(x, x', x'')$	ratio of light scattered from x'' to x by a patch of surface at x'

This equation took inspiration from contemporary radiosity research and integrates over all surfaces in the scene (S). In this formulation, $g(x, x')$ is needed to define which surfaces can see each other.

When we rewrite Kajiya's formula using the derivation of BRDF from Section 2.1 we end up with

$$L_r(x, \omega_o) = L_e(x, \omega_o) + \int_{\Omega_i} f_r(x, \omega_i, \omega_o)L_i(x, \omega_i)(\omega_i \cdot n)d\omega_i \tag{2.11}$$

where

$L(x, \omega_o)$	The outgoing radiance at point x in the direction ω_o
$L_e(x, \omega_o)$	The emitted light from point x in the direction ω_o
Ω_i	The full hemisphere of directions above the point x
$f_r(x, \omega_i, \omega_o)$	Percentage of light coming from direction ω_i reflected in direction ω_c
$L_i(x, \omega_i)$	The amount of light impinging on point x from direction ω_i
$(\omega_i \cdot n)$	The cosine term attenuating the light based on the surface normal

While they may look superficially different, the two equations are in fact very similar. In the more recent formulation, the geometry function $g(x, x'')$ is taken care of implicitly due

to the integration over the hemisphere at point x. Directions that would be canceled out due to g in Kajiya's equation are now canceled out due to $L_i(x, \omega) = 0$ for the directions in question.

The main difficulty in solving this equation is due to its recursive nature. When integrating over the hemisphere at any point x, the value $L_i(x, \omega_i)$ for almost any direction is equal to another integration $L(y, \omega_i)$. For diffuse surfaces the number of photons spread in any particular direction decreases with every bounce, meaning only a few bounces need to be calculated to generate a good approximation. Specular surfaces, by definition, reflect most photons in a very similar direction meaning that we need many levels to generate a good approximation.

2.3   The First Global Illumination Method

Until recently, solving equation 2.11 in real-time was not feasible, and therefore many simplifications were used in attempt to approximate the result.

In the late 60s and early 70s, several students at the University of Utah – under the direction of Dr. David C. Evans – investigated simplifying the rendering equation from an integral over all directions to a sum over all lights.

$$L(x) = \sum_{lights} S_p \tag{2.12}$$

where I is the intensity of the light that is then scaled by the shading at S_p that changes at every point on the object. The value S_p was defined in different ways by a few individuals. Romney [5] gave the first definition where

$$S_p = \frac{cos^2\theta}{R^4} * \text{normalization factor} \tag{2.13}$$

where $cos\theta$ is a measure of the orientation of a polygon and R is a measure of the distance to the polygon. Warnock gave a slightly different approximation [6] defining both a diffuse and specular component

$$S_p = \left|\frac{cos\theta}{R}\right| + \frac{cos^m\theta}{R} \qquad 6 \leq m \leq 10 \tag{2.14}$$

for each of the three basic color components (R, G, and B).

Newell et al. [7] in their work on transparent materials were the first to add an ambient term to their shading parameter:

$$S_p = r * cos^n(a) + b \tag{2.15}$$

where

r The desired intensity range

cos^n A function that defines diffuse ($n = 1$) or specular ($n > 1$) shading

a Some measure of the angle between the incident light and the face normal

b The ambient level of lighting

This ambient level b, whether knowingly or not, was an attempt to add more of the directions from (2.11) into the integral. By approximating the indirect lighting with a constant value, Newell et al. were the first to include a global illumination value into their renderings.

2.4   Traditional Global Illumination Methods

While the first attempts at solving light transport did attempt to include an approximation of global illumination with ambient lighting, it was a crude – although computationally inexpensive – attempt.

To improve the visuals of synthetic images, many researchers attempted to generate novel methods to solve the actual rendering equation, including both direct and indirect illumination in their results. The three main branches of research that emerged were ray tracing, radiosity, and photon mapping.

2.4.1   Ray Tracing

Ray tracing was introduced in computer graphics in 1979 by Whitted [8]. The basic idea was to create a "tree" of rays extending from the viewer out into the scene. When these rays encounter their first surface, they reflect back into the scene, allowing us to compute shadows, antialiasing, transparency, and lighting.

Ray tracing enabled the computation of many effects that would be otherwise complicated to reproduce. Distributed ray tracing by Cook et al. [9] allowed for fuzzy reflection; bidirectional path tracing [10] improved lighting convergence by connecting a set of rays both from the viewer and from the lights in the scene, while Metropolis Light transport [11] helps to find the most important samples in the image to ensure that all lighting sources are taken into account.

While hardware speeds have improved over the years, ray tracing is still limited by its acceleration structure. This structure is used to cull portions of the scene that are not required to render the current viewpoint. Building these structures is complex and time consuming. To use ray tracing in real time, the time required to build these acceleration structures will need to be reduced so that they can be built as a fractional part of a normal 33ms frame.

2.4.2 Radiosity

Radiosity is another method to calculate global illumintion by solving a set of linear equations relating visibility between surfaces patches in the scene. Introduced by Goral et al. [12], it works very well for diffuse global illumination and inspires our work in this thesis.

The process to calculate radiosity uses the following equation for each surface in the scene:

$$B_j = E_j + \rho_j \Sigma_{i=1}^{N} B_i F_{ij} \qquad \text{for} j = 1, N \tag{2.16}$$

where

B_j radiosity of surface j (watts/meter^2)
E_j rate of direct energy emission from surface j (watts/meter^2)
ρ_j reflectivity of the surface j (similar to BRDF)
B_i radiosity of surface i (watts/meter^2)
F_{ij} Form factor between surfaces i and j representing visibility between the surfaces

With one of these equations for every surface in the scene we have a linear system of N equations with N unknown B_j values. By defining some of the surfaces as emissive lights $E_j > 0$ the indirect lighting on all surfaces can be determined.

While the method has been extended for specular and translucent surfaces [13], it is much more complex than using ray-tracing techniques for the same purpose. This could be why much radiosity research declined in the mid-1990s.

For real-time dynamic scenes generating new form factors, every frame can be very problematic. Also, solving a linear system every frame for complicated scenes becomes too time consuming.

2.4.3 Photon Mapping

The last of the large branches of global illumination research is photon mapping [1]. Photon mapping uses a map of photons, which are later sampled to generate indirect illumination at different points in the scene. The photons originate at emissive surfaces like lights, and rebound off other surfaces. Diffuse surfaces generate photons in many directions, while specular surfaces have preferential directions, in which to send photons.

In recent literature [14], photon mapping has been shown to be a variant of bidirectional path tracing (see Section 2.4.1). Therefore, while it was a separate branch of research for many years, when attempting to use it in real-time graphics it runs into similar problems as ray tracing.

2.5   Real-Time Global Illumination

While hardware continues to increase in computational power and complexity there has been much research into other methods that improve on the Newell's simple ambient coefficient as well as attempting to simplify the more traditional global illumination techniques we have talked about so far.

In section 2.5.1 we discuss techniques that attempt to improve Newell's ambient term or Kajiya's geometric term by blocking some of the light that reaches each surface, creating greater contrast and a better image.

We then move on to more generic global illumination solutions that attempt to solve the entire rendering equation (2.11) for directions other than those pointing directly to lights, otherwise known as indirect illumination. Most of these techniques are not as physically correct as the traditional methods; however, for what they lack in physical correctness, they attempt to make up in speed. In section 2.5.2 we discuss techniques inspired by ray tracing and in section 2.5.3 techniques that draw inspiration from radiosity.

We conclude in section 2.5.4 with techniques that do not neatly fit into either camp. From new basis functions to attempts to add variation to the global illumination signal and generate more plausible visible images without resorting to traditional global illumination techniques.

2.5.1   Obscurance and Ambient Occlusion

Obscurance [15] and Ambient Occlusion (AO) [16] both model a visibility term for constant illumination. Accessibility [17] is a related technique, where the scene is colored based on the radius of the largest sphere that can touch a given point. While AO only accurately models the shadowing of ambient light, Precomputed Radiance Transfer (PRT) [18, 19] can extend this to more general lighting environments, but requires a precomputation and is only practical for very smooth lighting represented using spherical harmonics. Several recent papers [20, 21] have shown techniques that enable soft shadows from distant low frequency lights for dynamic scenes by approximating objects in the scene using a moderate number of spheres.

For dynamic scenes, several approaches have been taken to generate AO. Bunnell [22] approximated faces as discs and used techniques from hierarchal radiosity to solve for an AO-like quantity. Multiple iterations of Bunnell's technique simulate indirect lighting, but may have problems scaling to complex scenes. Another approach is to define the Ambient Occlusion from an object to any point in space efficiently [23, 24]; this can be problematic for small objects and makes it difficult to reason about combining independent objects.

Concurrent with our work, McGuire [25] analytically computed the AO contribution for polygons in the scene. An approach tailored for characters is to precompute a model of the AO on the character and ground as a function of the joint angles [26, 27], but this does not address how to combine the AO from disjoint characters or objects.

The approaches most closely related to ours work in screen space and use the depth buffer to compute AO [24, 28, 29, 30]. Concurrently to our work a similar technique was proposed [31] that includes a novel method to incorporate the effects of the normal. One benefit of these approaches is that execution is independent of scene complexity, depending solely on display resolution. Two techniques [24, 30] are close to the traditional formulation, scanning the frame buffer in the neighborhood of a point. Other techniques [28, 29] have a more volumetric feel, but no formal mathematical model; these are the closest to our technique. All of these approaches suffer from undersampling the scene, resorting to working at a reduced resolution, randomizing the samples, and then blurring the scene [28, 29, 30] to maintain performance. Our line sampling approach computes the integral more analytically at a set of 2D points but still sometimes requires an edge aware blur. A related technique [32] reprojects previous frames' AO estimates [33] and blends them with the current frame.

Another approach that gives similar results is to use unsharp masking on the depth buffer [34], to enhance some property of the image. Our approach is closer to ambient occlusion because the radius of the effect is fixed in object space. To mimic this behavior using filtering of the depth buffer would require a more complex spatially varying blur function. Our technique can be used to create more stylized imagery as well, for example higher weight lines at boundaries of objects as used in technical illustration [35]. But the primary focus is closer to how ambient occlusion is used in film and games.

Variance Shadow maps [36, 37] use a statistical model of depth, but for the purpose of computing soft shadows. Our area sampling technique for volumetric obscurance uses a similar statistical model, but the query is much more involved. A similar method was described in [38].

A recent paper [39] uses a technique similar to screen-space ambient occlusion (SSAO) to approximate indirect lighting along with a directional model of visibility. While the results are visually pleasing, the technique appears to be far too costly for current game consoles and is still based on point queries of the depth buffer. It would be interesting to try and extend this work using line or area sampling.

2.5.2   Ray-Tracing-Based Global Illumination Techniques

One of the main issues with calculating global illumination in real time is the size of the problem. To generate realistic results many rays and intersections are required, which is the main portion of the calculation. To alleviate this problem, many techniques have attempted to approximate the traditional solution in such a manner that they can be used in real time.

2.5.2.1   Real Time Ray Tracing

One attempt is simply to increase the speed of traditional methods. Known as real time ray tracing, there are many of these approaches that have gained popularity. Some of these methods pay close attention to the allocation of resources [40]. Others attempt to package rays together [41] so as to reduce the number of intersections required.

Others attempt to map traditional algorithms to the GPU [42]. Improvements on these methods [43] attempt to use GPU-based ray tracing and approximate yet further by sorting the ray-traced data using k-means to cluster the data making resampling simpler and faster.

There are also attempts [44] to generate dynamic acceleration structures so as to be able to use real-time ray tracing in a video game context.

The main problem is that while real time ray tracing has become much faster than the original CPU algorithm, the scalability and the high-performance requirements of interactive gaming engines still preclude the widespread adoption of this technology.

2.5.2.2   Instant Radiosity

Another popular method of generating real-time indirect illumination are instant radiosity approaches [45], which trace and deposit virtual light points (VPLs) in a scene and then compute direct lighting from them to approximate diffuse, and sometimes specular, interreflections.

This technique was improved upon with Reflective Shadow Maps [46] (RSM). Instead of storing particle lights around the scene, they use a shadow map technique that stores position, normals, and color. This set of data is then used to generate lighting information.

RSMs have also been used in deferred lighting scenarios [47]. To do this, RSM position data are used as the center of a projected ellipsoid, which is then associated with the other lighting information and used to light data stored in the deferred lighting buffers.

One of the main drawbacks of instant radiosity up to this point was the lack of shadows. Imperfect Shadow Maps [48] were an attempt to ameliorate this problem. By storing a point sampled representation of the scene, the points can be splatted into many tiny shadow maps. Even though these shadow maps have many artifacts, they are used on low powered

lights, which causes the artifacts to be much less visible. This allows artists to use VPLs for indirect lighting as well as shadowing.

While these techniques can and have been used in real time, they can be problematic on low-end graphics platforms such as the iPhone and iPad, where evaluating a single unshadowed point light takes at least 25 ms, precluding the feasibility of VPL techniques.

2.5.2.3 Proxies

Instead of simplifying the lighting, other techniques attempt to simplify the visibility computation by generating simple proxies for objects in the scene, which results in simplified visibility.

In Image Based Proxy Illumination [21] proxy spheres are used for all objects in the scene. These spheres are used to block environment lighting, approximating low frequency indirect lighting from the environment. Spherical proxies can also be used as lights [49]. Instead of blocking light, the proxies are also marked as light emitters allowing for indirect lighting effects such as color bleeding.

2.5.2.4 Spherical Harmonics

Instead of spheres, spherical harmonics (SH) can be used to simplify lighting and visibility equations. Spherical harmonics is a different basis defined over the surface of a sphere. This allows us to encode directions on the sphere using only a set of coefficients. SH has been used in many real-time indirect lighting techniques.

Precomputed Radiance Transfer (PRT) [18] was one of the first techniques used to apply spherical harmonics to real-time lighting. Using a long preprocessing step, the amount of lighting received at every vertex is stored as a set of SH coefficients that are then used at runtime to modulate the changing environment light.

PRT has been modified in many ways [19, 50] to compute shading response to basis illumination, capturing soft shadows, indirect light and caustics for static geometry. However, extensions to local lighting [51] require lengthy preprocessing and large amounts of data, as well as being scene dependent.

Spherical harmonic exponentiation [20] defines blocker geometry as spherical harmonic coefficients. These allow more complicated visibility functions but still can be simply combined using spherical harmonic exponentiation.

A more recent technique, Light Propagation Volumes (LPVs) [52], augment VPL approaches with a discrete, volumetric propagation phase for approximate global illumination. By storing and propagating a linear spherical harmonics (SH) radiance distribution in a

volume grid encompassing the entire scene, LPVs avoid precomputation, capture indirect shadows, and attain high-performance. However, the heavyweight nature of LPVs precludes implementation on low-end platforms. Radiance propagation also causes energy loss, which precludes distant light propagation; e.g., large maze examples would be challenging to render with LPVs. This approach suffers from inherent energy dissipation and is not well-suited for capturing distant indirect lighting and color bleeding in simple scenes can still pose problems for this approach.

2.5.3 Radiosity Based Global Illumination Techniques

Due to the complexity of running ray-tracing techniques at speeds usable for video games, some research has been inspired by radiosity in an attempt to generate indirect illiumination in real time.

2.5.3.1 Direct-to-Indirect Transfer

Direct-to-indirect approaches [53] map direct lighting on the surfaces of a static scene to indirect lighting. This is similar to radiosity where each surface pixel is a small surface patch. During precomputation, a direct-to-indirect transport operator is constructed using another traditional method (for example, ray tracing). This operator defines how the direct light on a surface affects indirect light on other surfaces. This allows us to calculate expensive indirect lighting as a linear function of the direct lighting, which is easy to calculate. This operator is stored and applied to the direct lighting, which is easily calculate at runtime.

2.5.3.2 Point-Based Radiosity

A recent interactive approach [54] attempts to recreate radiosity in real-time. This technique creates a set of points that represent the geometry in the scene. It then solves the radiosity equations using binary visibility between points in the scene. The result at each point is mapped to the actual geometry. Since indirect lighting is low frequency, interpolating the data from a limited point set does not create many artifacts. By solving for different portions of the scene over different frames the technique can generate real-time radiosity for static video game level geometry in real time.

2.5.3.3 Voxel Cone Tracing

Another popular method for generating indirect illumination designed for video games is Voxel Cone Tracing [55]. This technique rasterizes all the direct lighting information into a regular grid. These high frequency data are then averaged and stored in an octree data structure. This structure can then be queried quickly at runtime.

When the scene is being rendered, a set of cones are sent from each screen pixel. These cones use the different levels of the octree to approximate the indirect lighting impinging on the point in question. To generate diffuse indirect illumination, a set of five cones are sent out in many directions over the hemisphere. For specular lighting, a single cone is sent out in the appropriate specular direction for the material in question.

2.5.4 Other Global Illumination Techniques

Sometimes, artists are more concerned with generating an indirect lighting result that is plausible, but not necessarily physically correct. There are many ways outside of the traditional techniques that this can be accomplished.

2.5.4.1 Surface normals

There are many examples of adding high-frequency surface variation to smooth shading in both direct and indirect lighting. This is a popular approach used in many games [56, 57].

2.5.4.2 Modeling surface light variation

There are also ways to modify the indirect surface lighting over a larger area than normal maps typically address. Meyer and Anderson [58] perform principal component analysis (PCA) on noisy indirect light, leveraging the low-frequency nature of color bleeding to quickly filter out noise. Ashdown [59] also performs spectral analysis on transport operators (in a radiosity context). Similarly, lightfields [60, 61] use response bases to propagate and couple light flow to distant geometry.

2.5.4.3 Modeling volume light variation

Many times the desire is to change indirect illumination within the play space instead of simply on the surface of objects. Irradiance volumes [62] store radiance distributions in a volumetric grid so that, at runtime, dynamic objects can be lit by the surrounding environment. While games have used irradiance volumes to give dynamic objects a sense of immersion, our research introduces a method that allows dynamically generated indirect irradiance volumes based on dynamic light transport.

2.5.4.4 Antiradiance

Even with all these techniques, trying to compute accurate visibility between all the objects in the scene still remains difficult. There are interactive techniques that allow indirect lighting to be blocked in real-time. Antiradiance and implicit visibility approaches [63, 64] iteratively compute global illumination without explicitly evaluating visibility. In order to

capture direct and indirect shadows, negative radiance distributions are propagated along with standard radiance.

2.5.4.5 Basis Functions

While spherical harmonics are one set of basis functions that has been used with great success for indirect lighting, that does not exclude others. The idea of using a new basis is one of the main ideas of our research, but has also been applied in other techniques.

Stanton et al. [65] use nonpolynomial galerkin projection to simplify both fluid and radiosity calculations. They show that surface patches can be moved in space and yet still allow us to calculate radiosity between surfaces patches to generate indirect lighting.

Calian et al. [66] use piecewise constant basis functions over the sphere to generate real time lighting for augmented reality applications.

There are also other, nonrendering applications to the simplification of signals to aid in their reproduction. Antani et al. [67] use the idea of rectangular aural proxies to aid in the location and reproduction of sounds, while Gerszewski et al. [68] apply the idea of fluid model reduction to allow improved simulation bases and improved two-way solid fluid coupling.

2.5.4.6 Scene subdivision and coupling

Even with simplified approximations of indirect illumination sometimes the simulation space is too large. To simplify the computation, many techniques attempt to split the play space into many pieces, each of which can be solved separated and then reincorporated. Lewis and Fournier [69] described the concept of lightfields, which could be used to pass lighting information between areas of the scene split by portals. Similar concepts have been used in fluid simulation [70] and off-line rendering [71].

2.5.4.7 Screen-Space Approaches

Instead of trying to simulate all light interactions between objects in a scene, there are techniques that concentrate only on those effects that show up on screen. These can be calculated in less time due to the fact that the number of calculations is constant regardless of the number of objects that appear on screen since they are calculated per pixel.

Multiresolution splatting [72] works like deferred reflective shadow maps. However, instead of calculating the lighting on every pixel, they begin the calculation on a very low resolution buffer and only go to higher resolutions when the lighting is not similar enough over the low resolution buffer sample. This technique was improved in [73] where areas of

discrepancy in depth, and normals were stored as stencil bits. Where the high resolution lighting is only calculated in those areas marked by the stencil bits.

While these techniques are quick enough for real time, by ignoring any information that is not on screen they introduce spatial and temporal artifacts and perform all computation in high-dimensional spaces without exploiting coherence in the underlying light transport operators. For global illumination, screen space approaches do not meet the strict performance constraints of modern gaming engines.

2.6   Issues

Among these approaches, simple methods such as fixed ambient lighting are easy to author and render, but completely unrealistic. Static lighting [57, 74] has fast runtime performance and good quality but does not easily couple with dynamic geometry and can require hours of precomputation, hindering artist iteration. More complex high-quality approaches, such as Instant Radiosity [45] and Light Propagation Volumes (LPVs) [52] do not exploit low-dimensional computation and cannot scale to lower-end platforms, such as consoles or mobile devices. In cases where console platforms are supported (e.g., [54]), mobile platforms are not.

Techniques such as Precomputed Radiance Transfer (PRT) [18] and Screen-Space Ambient Occlusion (SSAO) [30] do not converge to ground truth, but are among the most used techniques in interactive graphics due to the smoothness and plausibility of their approximations, as well as their favorable performance and memory behavior. Modular Radiance Transfer (MRT) targets a similar brand of smooth shading approximation, responds plausibly to dynamic lighting, has extremely high-performance (faster than direct lighting computation), and allows fast author iteration.

CHAPTER 3

MODULAR RADIANCE TRANSFER[1]

3.1 Introduction

Indirect illumination increases the realism of computer generated images. The ambient term is a simple inexpensive approximation that does not respond to dynamic lighting. Accurate real-time techniques [45] have difficulty scaling to complex scenes and often have significant performance requirements, particularly on modern console and mobile platforms. Techniques that approximate different elements of indirect lighting have been extremely successful in interactive graphics applications. Precomputation techniques used in video games [57, 74] tend to assume static scenes and lighting, but suffer from long authoring iteration times and memory requirements. Ambient Occlusion (AO) [15] captures only salient shading effects. Variants of Precomputed Radiance Transfer (PRT) [18] generate soft lighting results. These techniques are favorable compared to more accurate techniques due to their lower storage and computation costs and the pleasing nature of their approximation. Modular Radiance Transfer (see Figure 3.1) targets **coarse-scale, distant** indirect lighting in scene geometry; responds plausibly and smoothly to dynamic lighting; has extremely high-performance; and allows fast author iteration.
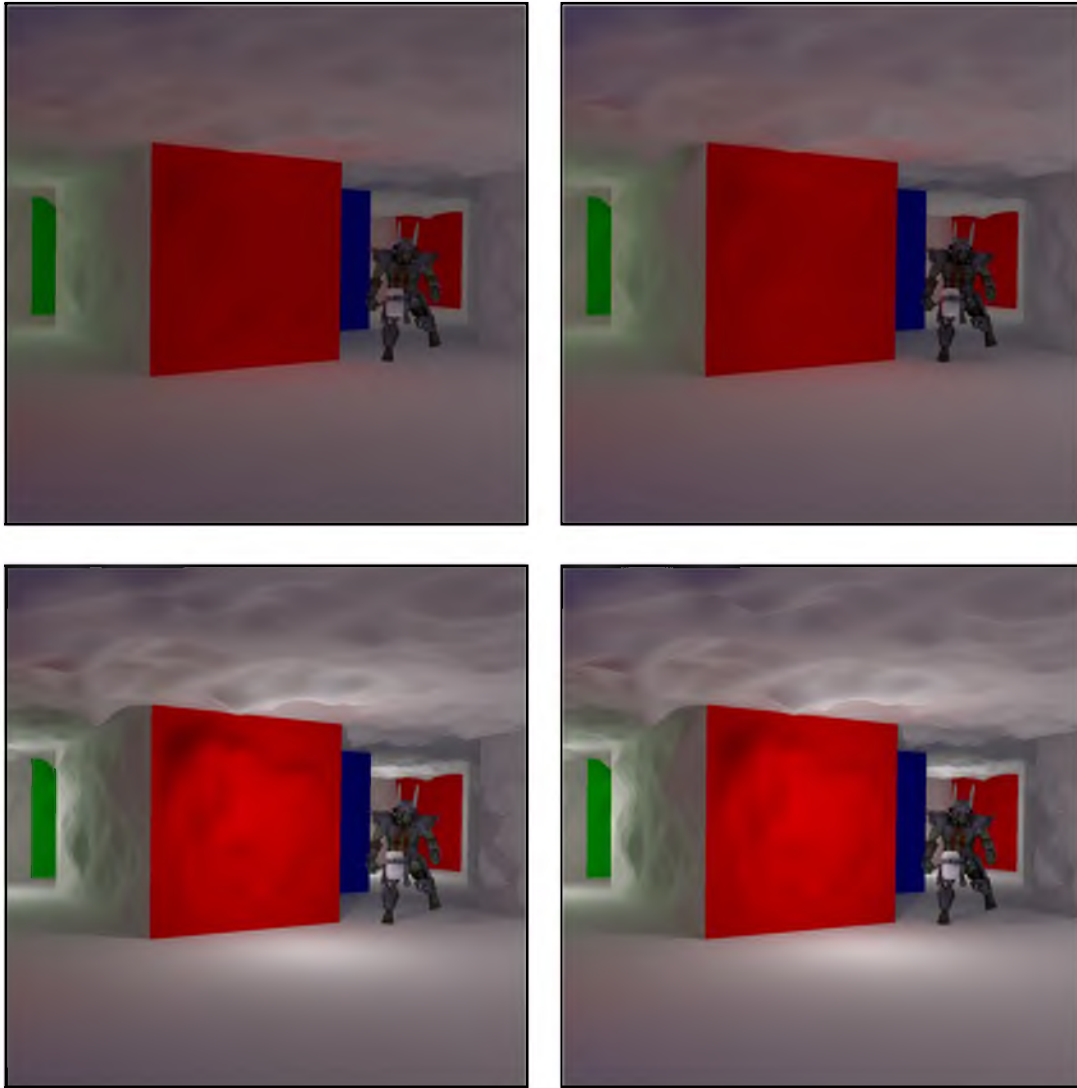
Our shapes are motivated by bounce cards used in live-action films. These planes only approximate indirect light from geometry in the real-world but offer a high level of control to produce the desired lighting. In digital film production, non-shadow-casting lights are commonly used to allow artists to quickly iterate and achieve a desired look. The ease-of-use and controllability of these approximations outweighs their physically incorrect nature.

Our approach is very efficient and uses very little data and a quick, one-time, *scene-independent* precomputation step. It also allows real-time computation of approximate indirect light and is designed with rapid iteration of light design in mind. We precompute light

Figure 3.1: Indirect light computed in reduced subspaces for a cave with 19 blocks and 4 lights. We derive low-dimensional transport operators, on simple proxy shapes, that are warped and combined at runtime, at > 475 FPS on high-end GPUs and > 45 FPS on mobile platforms, and can model indirect light at surfaces (with detailed normal variation) and within volumes of large-scale scene geometry. The left column showing our method with the ground truth is shown in the right column. The top row shows only indirect light while the bottom row shows both direct and indirect lighting (both with multibounce enabled).

transport operators (LTOs) for a handful of simple canonical "shapes", then interactively warp and combine these shapes, along with their LTOs, to more complex geometry. These shape proxies are used to model direct-to-indirect transport, which is then applied as a light map to the actual scene geometry. The flow of indirect light between proxies is modeled with lightfields, and all computations are performed on very low-dimensional subspaces. MRT results in plausible, dynamic global-illumination effects, rendered at high frame rates with low memory overhead. We design special LTOs for secondary transport effects such as light volumes for dynamic characters and higher-order irradiance for normal mapping. We illustrate our solution's ability to scale from high-end to mobile platforms and, like PRT, to provide smooth results which respond to light change.

Keeping the design goals mentioned above in mind, MRT makes the following contributions:

- **Compact transport operators using a novel lighting prior:** we represent direct and indirect light in a specialized *light prior* basis, which enables us to build compact and efficient LTOs to propagate multiple bounces of indirect light.
- **Modular, scene-independent transport computation:** LTOs from different canonical shapes are warped and combined, resulting in on-the-fly mapping of complex scene geometry (and its light transport) to simpler shapes (and their LTOs).
- **Flexible and efficient implementation:** we show that our approach runs on low-end mobile platforms as well as high-end GPUs, all while maintaining high performance.

3.2   Preliminaries

We adopt the following notation: italics for scalars and 3D points/vectors (e.g., ω), boldface lowercase for column vectors (e.g., $\mathbf{l}$), and boldface uppercase for matrices (e.g., $\mathbf{T}$). Notation used in this chapter can be found in Table 3.1.

3.2.1   Standard Direct-to-Indirect Transfer

Suppose we choose n surface locations on the scene to sample direct light; direct-to-indirect transfer maps direct illumination at these points to indirect lighting at these points:

$$\mathbf{l}_{\mathrm{ind}} = \mathbf{F}\,\mathbf{l}_{\mathrm{d}} \tag{3.1}$$

where $\mathbf{l}_{\mathrm{d}}$ and $\mathbf{l}_{\mathrm{ind}}$ are n-dimensional vectors of direct and indirect irradiance, and $\mathbf{F}$ is the one bounce transport operator.

Table 3.1: Notation used in this chapter.

Computed during precomputation then discarded:

$\mathbf{L}_{\mathrm{d}}$	Matrix of all possible direct lighting signals.
$\mathbf{P}$	Light prior basis retaining k_{d} left singular vectors of $\mathbf{L}_{\mathrm{d}}$.
$\mathbf{L}_{\mathrm{imp}}$	Implicit lighting environment.
$\mathbf{H}_{\mathrm{lf}}$	Raw lightfield matrix.
$\mathbf{H}_{\mathrm{rlf}}$	Reduced lightfield basis retaining k_{rlf} modes of $\mathbf{H}_{\mathrm{lf}}$.
$\mathbf{M}$	Light space indirect LTO.

Computed during precomputation, used during level initialization:

$\mathbf{R}_{\mathbf{b}\to\mathrm{rlf}}$	Projects $\mathbf{b}$'s to rlf-space (at each dictionary items' interface).
$\mathbf{R}_{\leftarrow,\uparrow,\nearrow}$	Propagates interface lightfield (input/output are both in rlf-space).
$\mathbf{T}_{\mathrm{rlf}\to\mathbf{r}}$	Maps reduced lightfield values to $k_{\mathbf{r}}$ surface response modes.

Computed during precomputation, used during runtime:

$\mathbf{T}_{\mathbf{d}\to\mathbf{b}}$	Transforms direct light ($\mathbf{l}_{\mathrm{d}}$) to indirect light coefficients ($\mathbf{b}$).
$\mathbf{U}_{\mathbf{b}}$	Indirect light basis after retaining $k_{\mathbf{b}}$ modes. Maps $\mathbf{b}$'s to $\mathbf{l}_{\mathrm{ind}}$.
$\mathbf{U}_{\vec{\mathbf{b}}}$	OHLSH version of $\mathbf{U}_{\mathbf{b}}$ (maps $\mathbf{b}$'s to vector irradiance).
$\mathbf{U}_{\mathbf{b}\mathrm{vol}}$	Maps $\mathbf{b}$'s to indirect volumetric light represented in SH.
$\mathbf{U}_{\mathbf{r}}$	Indirect lightfield response basis with $k_{\mathbf{r}}$ modes. Maps $\mathbf{r}$'s to $\mathbf{l}_{\mathrm{ind}}$.
$\mathbf{T}_{\mathbf{b}\to\mathbf{r}}$	Maps $\mathbf{b}$'s to $\mathbf{r}$'s via lightfields of the level's block connectivity.

Computed during runtime:

$\mathbf{l}_{\mathrm{d}}$	Direct lighting.
$\mathbf{b}$	Spectral lighting coefficients due to self-transfer ($\mathbf{T}_{\mathbf{d}\to\mathbf{b}}\ \mathbf{l}_{\mathrm{d}}$).
$\mathbf{r}$	Lightfield response coefficients on surfaces ($\mathbf{T}_{\mathbf{b}\to\mathbf{r}}\ \mathbf{b}$).
$\mathbf{l}_{\mathrm{ind}}$	Indirect lighting ($\mathbf{U}_{\mathbf{b}}\ \mathbf{b} + \mathbf{U}_{\mathbf{r}}\ \mathbf{r}$).

Evaluating (3.1) quickly limits runtime direct-to-indirect performance since $\mathbf{F}$ grows as $\mathcal{O}(n^2)$. Thus, we approximate $\mathbf{F}$ using singular value decomposition (SVD), $\mathbf{F} = \mathbf{U}_{\mathrm{f}}\ \mathbf{\Sigma}_{\mathrm{f}}\ \mathbf{V}_{\mathrm{f}}^{\mathrm{T}}$, where $\mathbf{U}_{\mathrm{f}}$ and $\mathbf{V}_{\mathrm{f}}^{\mathrm{T}}$ are high-dimensional rotation matrices, and $\mathbf{\Sigma}_{\mathrm{f}}$ is a diagonal matrix of singular values σ_i. Approximate indirect light can be computed by retaining the k largest singular values, $\mathbf{l}_{\mathrm{ind}} \approx \bar{\mathbf{U}}_{\mathrm{f}}\ \bar{\mathbf{\Sigma}}_{\mathrm{f}}\ \bar{\mathbf{V}}_{\mathrm{f}}^{\mathrm{T}}\ \mathbf{l}_{\mathrm{d}}$, where $\mathbf{U}_{\mathrm{f}}/\mathbf{V}_{\mathrm{f}}^{\mathrm{T}}/\mathbf{\Sigma}_{\mathrm{f}}$ are replaced with truncated matrices (retaining only the top k columns/rows) $\bar{\mathbf{U}}_{\mathrm{f}}/\bar{\mathbf{V}}_{\mathrm{f}}^{\mathrm{T}}/\bar{\mathbf{\Sigma}}_{\mathrm{f}}$.

3.2.2 Overview

The truncated SVD of $\mathbf{F}$ requires large k for accurate computation of $\mathbf{l}_{\text{ind}}$ (see Table 3.2), motivating a different approach. We define a novel **lighting prior** to compute reduced-dimensional transfer within a block (Section 3.3) and then couple the transfer between blocks with lightfields at their mutual interface (Section 3.4). Lastly, we show how to warp lighting from simple shapes to complex scene geometry (Section 3.5). These steps involve several intermediate spaces for derivation, but we always perform runtime computation in the low-dimensional spaces. The different spaces and quantities used in our derivations are summarized in Figure 3.2.

## 3.3 Reduced Direct-to-Indirect Transfer

Taking the SVD of $\mathbf{F}$ assumes a uniform distribution of arbitrary n-dimensional direct light patterns, explaining the slow decay of σ_i. In real scenes, direct light at a given point obeys a (simplified) rendering equation and is not drawn from an arbitrary distribution. We define a *light prior* over the distribution of direct light to construct a space spanned by physically-plausible $\mathbf{l}_{\text{d}}$. Our analysis will show that this space has dimensionality **significantly smaller** than n.

We first sample direct illumination patterns, $\{\mathbf{l}_{\text{d}0}, \ldots, \mathbf{l}_{\text{d}m}\}$, in order to construct a low-dimensional basis for plausible direct lighting. These patterns can be computed using any approach; however, it is best to use the same direct illumination at runtime. We generate each sample by placing a sphere light at uniform volumetric locations in our (canonical) block shape and computing the direct lighting.

Next, we place the samples $\{\mathbf{l}_{\text{d}0}, \ldots, \mathbf{l}_{\text{d}m}\}$ into columns of a matrix $\mathbf{L}_{\text{d}}$ and compute its SVD: $\mathbf{L}_{\text{d}} = \mathbf{U}_{\text{d}} \, \mathbf{\Sigma}_{\text{d}} \, \mathbf{V}_{\text{d}}^{\text{T}}$. The left singular vectors yield our *light prior* basis: $\mathbf{P} = \tilde{\mathbf{U}}_{\text{d}}$. Table 3.2 summarizes the singular value fall-off, justifying our earlier observation that physically realizable direct light lies in a low-dimensional linear subspace: $\dim(\mathbf{L}_{\text{d}}) \ll n$. We do not subtract the mean and compute PCA since we wish to represent lighting with arbitrary intensities, which means all scales of input patterns should be well represented.

Table 3.2: Number of singular values to capture percentage of the energy for different matrices (for a cube with $n = 6 \times 16^2$ samples).

Matrix	80%	90%	95%
$\mathbf{F}$ / $\mathbf{L}_{\text{d}}$ / $\mathbf{M}$	164 / 22 / 1	240 / 41 / 5	313 / 62 / 8
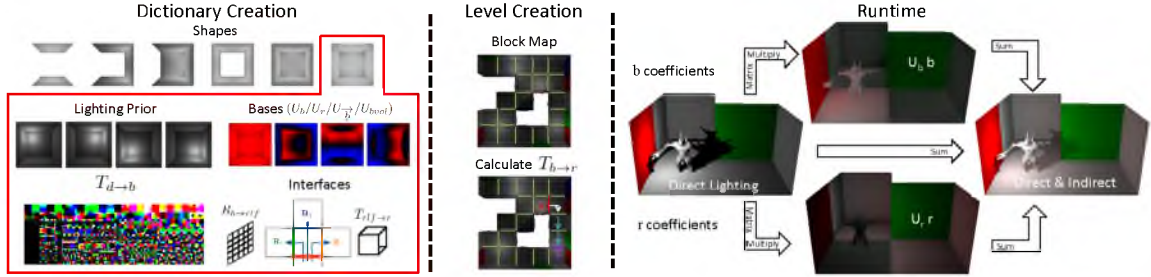
Figure 3.2: Overview: dictionary creation precomputes light priors and bases for scene independent shapes. Level creation maps the shapes and generates the lightfield propagation matrix $\mathbf{T}_{b \to r}$. At runtime, we generate $\mathbf{b}$ and $\mathbf{r}$ vectors to compute indirect light with $\mathbf{U}_b$ and $\mathbf{U}_r$ bases.

3.3.1   Light Transport in Indirect Light Space

Any plausible indirect lighting condition can be approximated as a linear combination of indirect light due to the direct light prior basis vectors: $\mathbf{l}_{\text{ind}} = \mathbf{F} \, \mathbf{P} \, \left[\mathbf{P}^{\text{T}} \, \mathbf{l}_{\text{d}} \right]$, where $\mathbf{P}^{\text{T}}$ projects $\mathbf{l}_{\text{d}}$ onto our light prior, resulting in scaling coefficients for indirect light induced by the light prior basis vectors. In other words, each direct lighting (basis) pattern has a corresponding indirect lighting (basis) pattern.

Unfortunately, this formulation does not exploit correlations in the indirect light (the columns of $\mathbf{F} \, \mathbf{P}$ are not independent). We aim to *directly* obtain an orthogonal basis for indirect light, which accounts for our direct light prior, instead of simply reconstructing direct light and applying the one-bounce operator to it (as described above).

We start by postmultiplying $\mathbf{F} \, \mathbf{P}$ by the scaling matrix $\mathbf{S} = \tilde{\boldsymbol{\Sigma}}_{\text{d}}$, which leads to an equivalent problem:

$$\mathbf{l}_{\text{ind}} = \mathbf{F} \, \mathbf{P} \, \mathbf{S} \, \mathbf{S}^{-1} \, \mathbf{P}^{\text{T}} \, \mathbf{l}_{\text{d}} \tag{3.2}$$

where we define the *light space indirect LTO* $\mathbf{M} = \mathbf{F} \, \mathbf{P} \, \mathbf{S}$, which maps (scaled) direct lighting prior coefficients ($\mathbf{l}_{\text{d}}$) to indirect light, and take its SVD: $\mathbf{M} = \mathbf{U}_{\text{m}} \, \boldsymbol{\Sigma}_{\text{m}} \, \mathbf{V}_{\text{m}}^{\text{T}}$. Table 3.2 shows that the SVD of $\mathbf{M}$ falls off much more rapidly than either $\mathbf{F}$ or $\mathbf{L}_{\text{d}}$. We retain $k_{\mathbf{b}} \ll n$ singular values, yielding the approximation

$$\mathbf{l}_{\text{ind}} \approx \mathbf{U_b} \, \mathbf{T_{d \to b}} \, \mathbf{l_d} = \mathbf{U_b} \, \mathbf{b} \tag{3.3}$$

where $\mathbf{T_{d \to b}} = \tilde{\mathbf{V}}_{\text{m}}^{\text{T}} \, \mathbf{S}^{-1} \, \mathbf{P}^{\text{T}}$ maps $\mathbf{l_d}$ to spectral coefficients $\mathbf{b}$, used to scale columns of $\mathbf{U_b} = \tilde{\mathbf{U}}_{\text{m}} \, \tilde{\boldsymbol{\Sigma}}_{\text{m}}$. These *orthogonal* columns form an *indirect light space* basis, and scaling by $\tilde{\boldsymbol{\Sigma}}_{\text{m}}$ makes the lengths proportional to the statistics from $\mathbf{M}$. Figure 3.2 illustrates several columns of $\mathbf{U_b}$ and rows of $\mathbf{T_{d \to b}}$.

3.3.1.1   Implicit Lighting Environment

An interesting question to consider is whether there exists a direct lighting pattern that, after application of the one bounce transport operator, generates the columns of $\mathbf{U_b}$ as an output indirect lighting pattern. In other words, we wish to find $\mathbf{L_{imp}}$ such that $\mathbf{U_b} = \mathbf{F} \, \mathbf{L_{imp}}$. We derive this direct lighting pattern, which we call the *implicit lighting environment*, as follows:

$\mathbf{M} = \mathbf{U_m} \, \mathbf{\Sigma_m} \, \mathbf{V_m^T} = \mathbf{F} \, \mathbf{P} \, \mathbf{S}$ by definition, and after postmultiplying both sides by $\mathbf{V_m}$, we obtain $\mathbf{U_m} \, \mathbf{\Sigma_m} \, \mathbf{V_m^T} \, \mathbf{V_m} = \mathbf{F} \, \mathbf{P} \, \mathbf{S} \, \mathbf{V_m}$, with the left hand side simplifying to $\mathbf{U_m} \, \mathbf{\Sigma_m} = \mathbf{U_b}$.

And so, $\mathbf{L_{imp}} = \mathbf{P} \, \mathbf{S} \, \mathbf{V_m}$, and it will prove useful in several instances, e.g., when generating higher-order lighting variation on surfaces (Section 3.5.4), volume samples within the scene (Section 3.5.5), and interface lightfields between connected shapes (Section 3.4.1).

3.4   Direct-to-Indirect Transfer Between Shapes

We first couple transport between blocks in order to compute direct-to-indirect transfer on large, interconnected sets of dictionary shapes.

Our high performance relies on computing coupled transport in reduced basis spaces. Given $\mathbf{b}$-coefficients for a simple shape (e.g. a cube with missing faces), our goal is to compute operators that act directly on this vector and scatter light into neighboring shapes.

3.4.1   Interfaces: Far-Field Light Transport Coupling

Our approach is independent of the dictionary's contents, and we illustrate results with cube-based and cylinder-based dictionaries. For cubic shapes, we create five operators: $\mathbf{R}_{b \rightarrow rlf}$ describes how light leaves each shape via its missing faces or interfaces, $\mathbf{R}_{\eta, \uparrow, \Gamma}$ describes transfer between interfaces, and $\mathbf{T}_{rlf \rightarrow r}$ describes transfer from an interface onto the surface of a block. These five operators are concatenated, based on block layout, at level creation time to generate a sparse block matrix $\mathbf{T}_{b \rightarrow r}$, describing how light leaving each block illuminates all other connected blocks.

We compose a raw lightfield matrix $\mathbf{H_{lf}}$ at the N_i dictionary interfaces (see Section 3.5.1). Columns of $\mathbf{H_{lf}}$ are a resampling of implicit lights (columns of $\mathbf{L_{imp}}$), for each dictionary element, at each position and direction of the interface's lightfield. $\mathbf{H_{lf}}$ is an $s \times N_i \, k_{\text{self}}$ matrix, where s is the spatiodirectional lightfield resolution.

Given the SVD of $\mathbf{H_{lf}} \approx \mathbf{\tilde{U}_{rlf}} \, \mathbf{\tilde{\Sigma}_{rlf}} \, \mathbf{\tilde{V}_{rlf}^T}$, the k_{rlf} left singular vectors scaled by the corresponding singular values ($\mathbf{H_{rlf}} = \mathbf{\tilde{U}_{rlf}} \, \mathbf{\tilde{\Sigma}_{rlf}}$) form a low-rank *reduced lightfield basis*. This represents the response at the shape's interfaces to $\mathbf{L_{imp}}$. We use *basis enrichment* to handle more complex propagation operations (see Appendix A).

We define a $\mathbf{R}_{\mathrm{b}\to\mathrm{rlf}}$ operator for each interface of each dictionary element to map the element's $\mathbf{b}$-coefficients to coefficients in the reduced lightfield basis. We construct this operator by lighting each element with its implicit lighting environments, resampling the implicit lighting from the surfaces to the interface(s), and projecting the resampled lightfields into the reduced lightfield basis.

We construct three additional operators to capture near-field interface-to-interface, lightfield propagation, and resampling.

Given an interface (red line in Figure 3.3), its lightfield can be propagated to the interface straight ahead (dark blue line) with $\mathbf{R}_{\uparrow}$, or to the interface on the left adjacent face (green line) with $\mathbf{R}_{\nwarrow}$, or to the interface on the right adjacent face (orange line) with $\mathbf{R}_{\nearrow}$. These square matrices (with dimensions k_{rlf}^2) resample the lightfield at one interface to either the straight-ahead, left-adjacent, or right-adjacent interface and project the resulting lightfield back into the reduced lightfield basis.

Lastly, we define an operator to map reduced lightfield coefficients to lighting response on geometry near a lightfield: $\mathbf{G} = \mathbf{F}_{\mathrm{rlf}}\,\mathbf{H}_{\mathrm{rlf}}$, where $\mathbf{F}_{\mathrm{rlf}}$ is a transport matrix that computes the surface response to the reduced lightfield basis lighting (columns of $\mathbf{H}_{\mathrm{rlf}}$). This only needs to be done for a single canonical interface, due to symmetry.

Using a similar motivation as in Section 3.3.1, we compute the SVD of $\mathbf{G} = \mathbf{U}_{\mathrm{g}}\,\boldsymbol{\Sigma}_{\mathrm{g}}\,\mathbf{V}_{\mathrm{g}}^{\mathrm{T}}$, leveraging coherence to $\mathbf{H}_{\mathrm{rlf}}$'s response and retain the left singular vectors (columns of $\mathbf{U}_{\mathrm{r}}$, where $\mathbf{U}_{\mathrm{r}} = \tilde{\mathbf{U}}_{\mathrm{g}}$) to form an *indirect lightfield basis*. $\mathbf{T}_{\mathrm{rlf}\to\mathrm{r}} = \tilde{\boldsymbol{\Sigma}}_{\mathrm{g}}\,\tilde{\mathbf{V}}_{\mathrm{g}}^{\mathrm{T}}$ maps reduced lightfield coefficients to reduced lightfield responses $\mathbf{r}$.

We only retain operators, $\mathbf{R}_{\mathrm{b}\to\mathrm{rlf}}$ and $\mathbf{T}_{\mathrm{rlf}\to\mathrm{r}}$, that act in reduced spaces. Lightfields never need to be reconstructed, resulting in significant memory and performances savings.
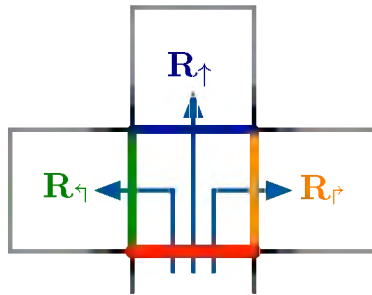
Figure 3.3: Example of all interfaces for one of our connected dictionary shapes.

3.5   Direct-to-Indirect Transfer on Real Scenes

We now combine compact direct-to-indirect transport operators within (Section 3.3) and between (Section 3.4) simple shapes to quickly approximate direct-to-indirect transfer on *arbitrary* scenes.

Smooth, plausible and dynamic light transport is computed every frame using a mapping between dictionary shapes and scene geometry during scene creation. We derive special transport operators for vector-valued irradiance (to model high-frequency surface detail), and volumetric direct-to-indirect light probes (to relight dynamic objects), all of which will be computed entirely in reduced spaces.

3.5.1   2D Shape Dictionary

To simplify our exposition, we first consider 2D mazes with cubes connected to each other along missing faces (we also show cylinder-based results). For these scenes, we populate a dictionary with all possible cubes with 0 to 4 faces missing.[2] Exploiting symmetry, this dictionary has six entries with a total of 12 missing faces.

3.5.2   Authoring the Scene Proxy

We map complex geometry to *only a handful* of connected blocks from our shape dictionary. To do so, we begin by generating a set of connected blocks and associating portions of the complex scene to each block. We are motivated to model scenes with extremely coarse proxy geometry in order to capture large-scale indirect lighting effects as efficiently and compactly as possible, unlike discrete ordinate methods that use a large number of blocks. This is analogous to the use of bounce cards when lighting for film.

The real scene must be mapped to shapes in our dictionary before lighting can be computed. Each shape in this proxy is an instanced transformation of an element in our dictionary and also contains a region in texture space for a light map atlas that will be constructed to light the real scene. We also compute a geometry image that will be used to compute l_d. A key point is that we can reuse the precomputed transfer data from the dictionary. Dictionary shapes can be warped and attached by artists or automatically for simple mappings (see Section 3.6.2). When applying severe warping to shapes, we can reduce artifacts by dynamically warping the prior and U_s (see Section 3.7 and Figure 3.4).

While each shape's texture is self-contained, the final mapping to real scene geometry requires continuous reconstruction between connected shapes. Moreover, clamping is required

---

[2] For 2D mazes, all cubes have a "top" and "bottom" face; 3D mazes do not have this constraint and result in a nine element dictionary.
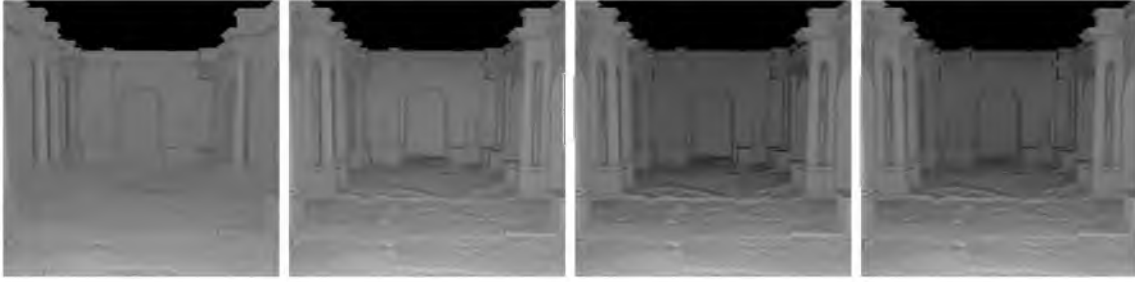
Figure 3.4: The Great Hall mapped to 1 cube is better than using an ambient term, but causes unrealistic lighting on the longest axis. You can see the result in the two left images which use 1 and 32 modes respectively. Ray tracing the operators again creates better results. Raytracing $\mathbf{U}$ again gives us the third image, and ray tracing $\mathbf{U}$ along with stretching the cube prior to the bounding box achieves results similar to recomputing the entire prior (fourth image).

over the shape's internal creases. We rectify these continuity issues by creating a padded light map atlas and a set of records that copy edge/corner values from neighboring unpadded regions to the padded light maps (faces translate to the center of padded faces) [76].

3.5.3   Interface Propagation for Distant Light Transport

Once the set of proxy shapes is generated, we compute a block sparse operator $\mathbf{T}_{\mathbf{b}\to\mathbf{r}}$ to map $\mathbf{b}$-coefficients at all the shapes to $\mathbf{r}$-coefficients at each interface. The interface matrices $\mathbf{R}_{\mathbf{b}\to\mathrm{rlf}}$, $\mathbf{R}_{\nwarrow}, \mathbf{R}_{\uparrow}, \mathbf{R}_{\nearrow}$, and $\mathbf{T}_{\mathrm{rlf}\to\mathbf{r}}$ are combined to propagate transport through a complex scene in our reduced spaces. It is important to note that these steps only depend on the block connectivity, and the input is parameterized by the $\mathbf{b}$-coefficients, which results in an efficient runtime.

To propagate indirect light from block x, breadth-first traversal of the block connectivity (with a maximum traversal depth $d_{\max}$) builds columns of $\mathbf{T}_{\mathbf{b}\to\mathbf{r}}$ to map indirect light coefficients $\mathbf{b}$ from x to reduced lightfield coefficients $\mathbf{r}$ at the traversal's current interface.

The relevant block of $\mathbf{T}_{\mathbf{b}\to\mathbf{r}}$ is initialized to x's $\mathbf{R}_{\mathbf{b}\to\mathrm{rlf}}$ and, as each interface is traversed, is premultiplied with one of $\{\mathbf{R}_{\nwarrow}, \mathbf{R}_{\uparrow}, \mathbf{R}_{\nearrow}\}$, depending on the propagation direction [76]. This traversal concatenates transport operators to "drive" indirect light from source shapes, through interfaces, to the rest of the scene.

At each interface, the accumulated portion of $\mathbf{T}_{\mathbf{b}\to\mathbf{r}}$ is premultiplied by $\mathbf{T}_{\mathrm{rlf}\to\mathbf{r}}$ and stored. If a path from the same source block has already been computed, the matrices are summed; otherwise a new record is computed. These block matrices form a sparse representation of the full matrix that maps the block's $\mathbf{b}$-coefficients to reduced interface response $\mathbf{r}$ at all other lightfields visible from x. When this matrix is multiplied by the $\mathbf{b}$'s, we are aggregating the energy at every scene interface, avoiding more expensive reconstructions into a light

map. Figure 3.5 is a visual representation of a column and row of the matrix. This process is only done when a scene is created and takes less than a second for all our example scenes.

3.5.4 Higher-Order Irradiance in Indirect Light Space

We replace indirect irradiance (columns of $\mathbf{U_b}$ in Equation 3.3) with vector-valued irradiance capable of approximating indirect light due to high-frequency, normal-mapped surface details (see Figure 3.6).

We derive an Optimal Hemispherical Linear Spherical Harmonics (OHLSH) basis for vector-valued irradiance stored at the n sample locations in each canonical block. To compute the vector-valued irradiance response basis, $\mathbf{U_{\vec{b}}}$, we simply light a block with its $\mathbf{L_{imp}}$ and project the irradiance distribution into quadratic SH instead. This 9D vector is analytically mapped to a 4D OHLSH vector (see Appendix B).

3.5.5 Parameterized Irradiance Volumes

We light dynamic geometry, such as animating characters and "ornamental clutter," like pillars and statues that do not map naturally to block faces (see Figure 3.7), by computing parameterized light probes in the volume of the scene. These lightprobes are represented as order-3 SH vectors sampled uniformly in space and allow us to shade animated (or "clutter")
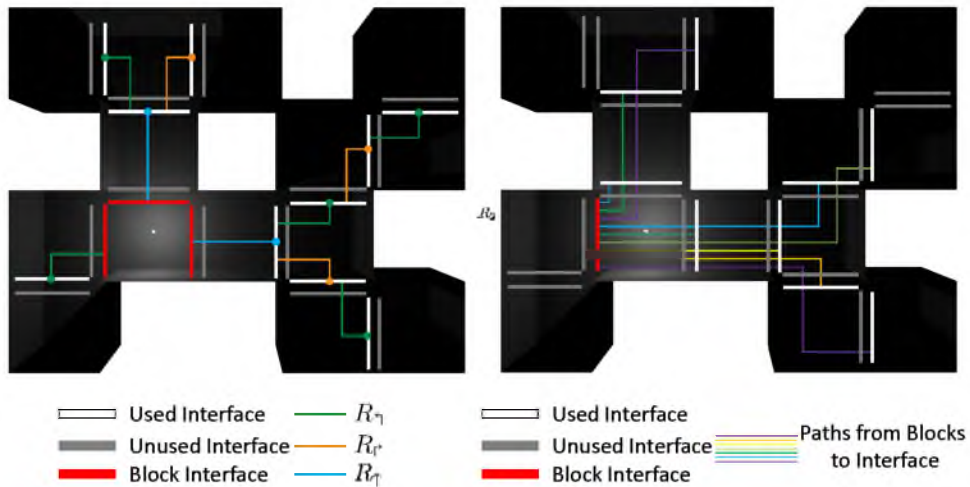
Figure 3.5: An example of how interfaces between blocks work. **Left**: light originating from the block containing the red interfaces propagates to all other visible outgoing interfaces in the maze. **Right**: all the blocks that contribute light to the red interface. These are rows and columns of the matrix $\mathbf{T_{b \to r}}$ that maps $\mathbf{b}$s to $\mathbf{r}$s.
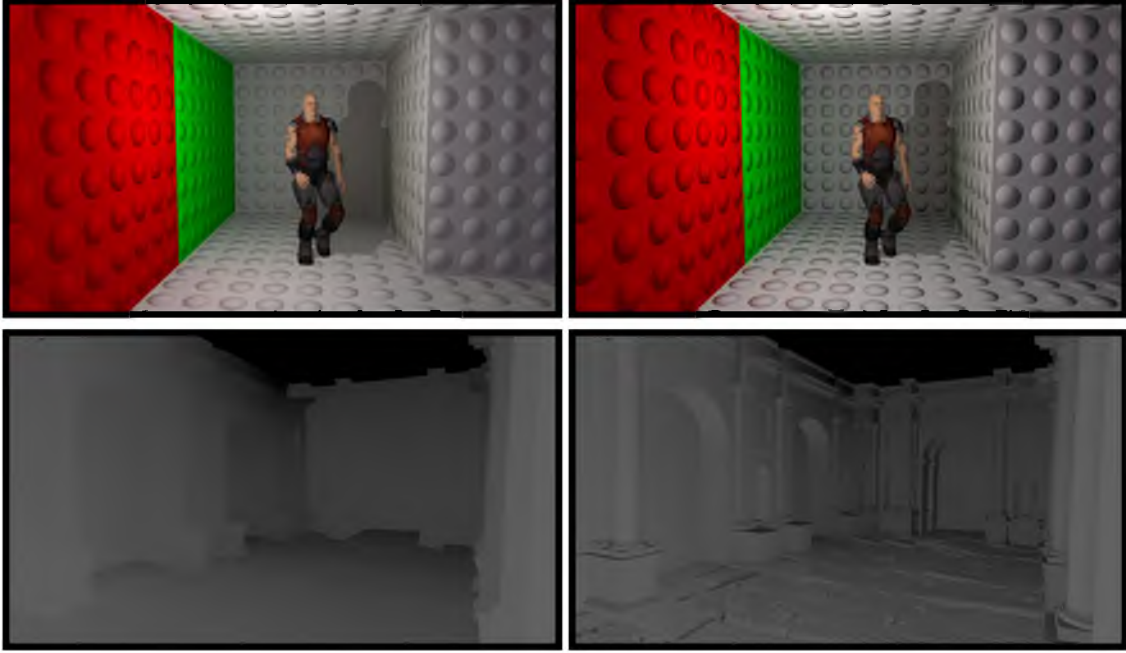
Figure 3.6: Vector irradiance for surface details in a normal-mapped maze (**top**) and the Great Hall (**bottom**). Scalar irradiance is on the left, with vector irradiance on the right. © 2011 B. Loos, L. Antani, K. Mitchell, D Nowrouzezahrai, W. Jarosz, P.P. Sloan

geometry using traditional SH techniques.

We compute an operator $\mathbf{U}_{\mathrm{bvol}}$ (with size $9 \times k_{\mathrm{self}}$) that gathers radiance from implicit lights ($\mathbf{L}_{\mathrm{imp}}$) in a uniform volumetric grid in each block and projects these distributions into SH. $\mathbf{U}_{\mathrm{bvol}}$ maps indirect illumination in the indirect light space to SH coefficients. As with surface and interface response, we use $\mathbf{L}_{\mathrm{imp}}$ to drive the generation of these probes entirely in the indirect light space and need only precompute this operator once for the shapes in our dictionary.

3.6 Implementation and Results

3.6.1 Dictionary Construction and Precomputation

We generate a 2D block dictionary with six basic cube shapes and a cylinder dictionary with three shapes (straight, right turn, left turn).

For these dictionaries, we construct the prior $\mathbf{P}$ by lighting the basic shapes with 6^3 sphere lights placed uniformly in the block's volume and through interfaces. We use $k_{\mathrm{d}} = 64$ and $k_{\mathrm{self}} = 32$, enriching the basis with functions that are constant on only one face at a time, as well as $\mathbf{U}_{\mathrm{b}}$ so that multiple light bounces are well represented.
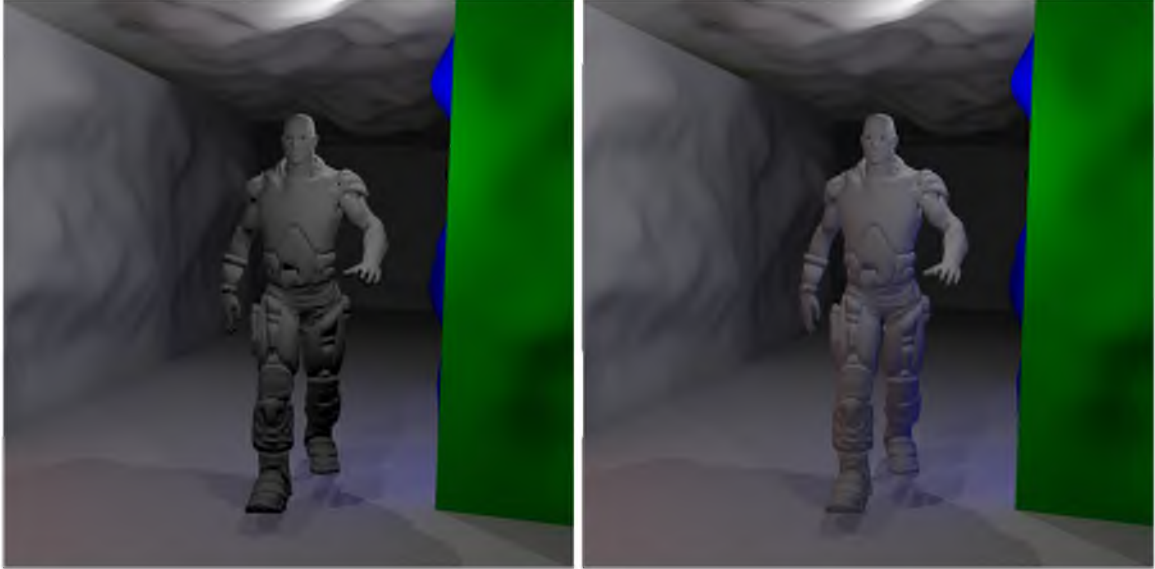
Figure 3.7: Indirect transport on dynamic objects (character mesh) without (**left**) and with (**right**) volume samples. © 2011 B. Loos, L. Antani, K. Mitchell, D Nowrouzezahrai, W. Jarosz, P.P. Sloan

Most of our example blocks have 16^2 samples per face. We evaluate $\mathbf{M}$ by ray tracing the scaled prior vectors (columns of $\mathbf{P}$) using 16K gather rays with importance sampling. We perform this at each of the 16^2 points on the face, from which $\mathbf{U_b}$, $\mathbf{L_{imp}}$, and $\mathbf{T_{d \to b}}$ are computed. The cylinder dictionary uses 64^2 samples per shape. We also compute several lower resolution dictionaries using only scalar surface response for the iPad and x86 CPU implementations. Using fewer than 6^2 samples results in objectionable artifacts.

The $\mathbf{U_{\vec{b}}}$, $\mathbf{U_r}$ and $\mathbf{U_{bvol}}$ are computed by ray tracing implicit lights (columns of $\mathbf{L_{imp}}$) and computing their respective output data. $\mathbf{U_{\vec{b}}}$ and $\mathbf{U_{bvol}}$ are computed once per dictionary shape, and the interface data are computed for only a single canonical interface.

To compute $\mathbf{H_{lf}}$ from $\mathbf{L_{imp}}$ we use 5^2 directional super-sampling of interfaces. Three iterations of enrichment are applied to $\mathbf{H_{rlf}}$, retaining $k_{rlf} = 128$ lightfield modes and $k_r = 32$ response modes.

We use raw interface resolutions of $s = 12^2$ spatial $\times$ 24^2 directional $= 82944$ total samples, mapping the hemisphere to a square [77] for continuous directional interpolation.

3.6.1.1 Precomputation

The dictionary (without interfaces) requires 4.7 MB, takes 59 s to build, and 60 s to enrich. Interfaces increase precomputation by 260 s, adding 1.2 MB of memory. Timings are on a dual 2.93 GHz 6-core Intel CPU. Interface computation does not scale well with the 12

cores, but the other stages scale linearly.

### 3.6.2 Mapping Shapes to Complex Scenes

We use two approaches to compute texture coordinates (and position/normal geometry images) in complex scenes. For the game scene (Figure 3.8), artists map blocks with our level editor: a cube is warped to roughly align with part of the scene, then additional cubes are extruded/warped from existing ones (see video for a modeling session). The artist can preview indirect light directly in the tool. $\mathbf{T_{b \to r}}$ is also computed at this phase and takes a second at most.

As an extreme stress test, we model the Great Hall scene (Figure 3.9) using only a single block. The vertices are mapped to block faces using rasterization: a cube map camera at the center of the block renders a downsampled "geometry texture," which allows us to categorize the vertices belonging to each face. The texture coordinates for a given vertex are computed by orthographically projecting the vertex onto the chosen face, and these coordinates are used to look up indirect light in the padded light-map texture atlas.

We create records, for each block's face, to render indirect light ($\mathbf{l_{ind}} \approx \mathbf{U_b} \, \mathbf{b}$) into an (unpadded) light map. Each record is positioned appropriately in the light map and contains $\mathbf{U_b}$ texture coordinates and a block index for the face to look up $\mathbf{b}$-coefficients.
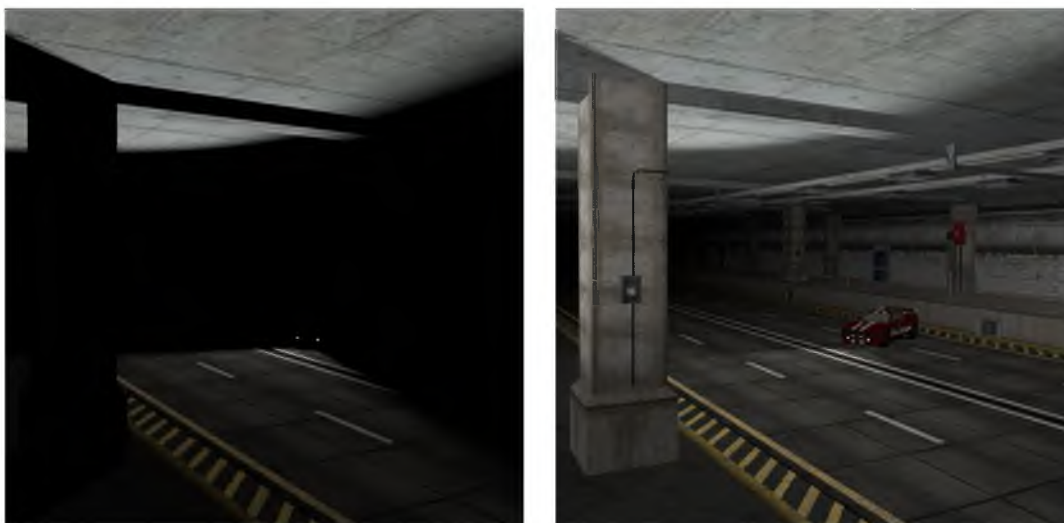
Figure 3.8: An 18 block scene from a video game: direct light (**left**, 1.2ms), direct and indirect (**right**, 2.2ms).
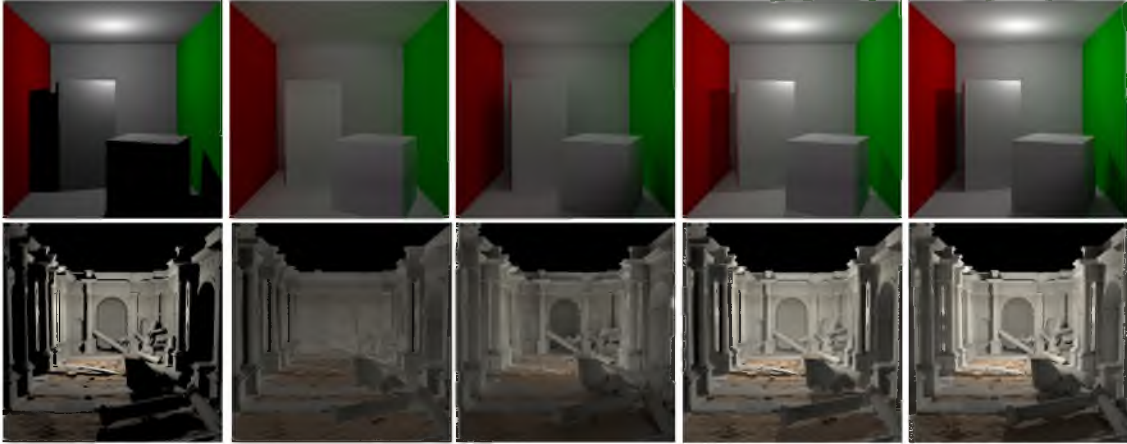
Figure 3.9: Dynamic indirect light at >**500 FPS**. We ignore indirect shadows and focus on adding smooth, approximate dynamic indirect light with low computation cost. In the extreme case of the Great Hall modeled with 1 cube (and volume samples for "clutter" geometry), our results respond to direct light at a cost similar to static ambient terms but with soft shading quality common in e.g. PRT. From left to right the images represent direct lighting only, indirect lighting using MRT (which takes 1.9 ms for the top row and 1.4 ms for the bottom row), indirect lighting using a ray tracer, then direct + indirect lighting using MRT, and direct + indirect lighting using a ray tracer.

3.6.3 Simple Runtime Implementation

Our runtime is quite simple. An indirect light texture, using direct light evaluated at n surface points, is computed as follows:

1. Compute direct light at a reduced resolution in each block ($\mathbf{l_d}$),

2. Compute per-block spectral coefficients ($\mathbf{b} = \mathbf{T_{d \to b}} \, \mathbf{l_d}$),

3. Compute indirect light *within* a block into a lightmap ($\mathbf{U_b} \, \mathbf{b}$),

4. Compute response coefficients at interfaces ($\mathbf{r} = \mathbf{T_{b \to r}} \, \mathbf{b}$),

5. Blend response from "external" blocks into a lightmap ($\mathbf{U_r} \, \mathbf{r}$),

6. Create padded lightmap texture to eliminate texture seams,

7. Render scene using the dynamic lightmaps of indirect lighting,

8. **[optional]** Compute indirect light volume (from $\mathbf{b}$'s and $\mathbf{r}$'s),

9. **[optional]** Render dynamic objects with volume lighting.

As in Enlighten [54] it is easy to feedback indirect light (scaled by albedo) to get multiple bounces with little overhead (Figures 3.10 and 3.11). Figure 3.1 compares our approximate multibounce indirect light (476 FPS) to ground truth (many hours).
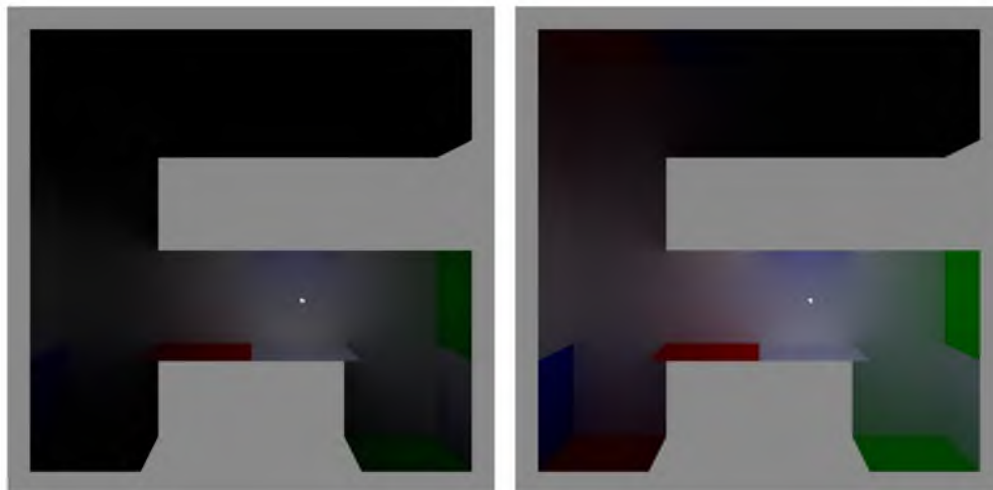
Figure 3.10: Feeding back the previous iteration's indirect light buffer into the direct light buffer generates multiple bounces.
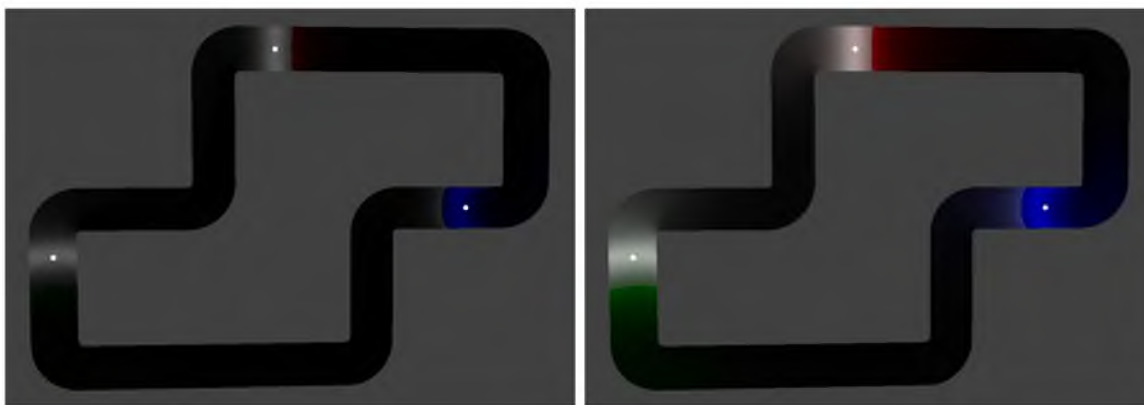

Figure 3.11: Tunnel created with 10 cylinders. Direct lighting (**left**), direct and multibounce indirect lighting (**right**, 5.2 ms = 192 FPS).

3.6.4 Results

Examples of our final results can be found in Figure 3.12. GPU performance was recorded on an NVIDIA 480 GTX with a DX11 runtime. Rendering of omnidirectional shadow maps for direct lighting is by far the largest bottleneck of our runtime renderer for a moderate number of blocks. Our indirect lighting performance scales linearly with the number of blocks (see Figure 3.13) with the most expensive stage being computing **b**'s. This could potentially be optimized by using a single pass reduction or compute shader.

We also have two software implementations, one for x86 CPU's using SSE and one for iPad/iPhone using NEON instructions. The CPU versions support everything but volume
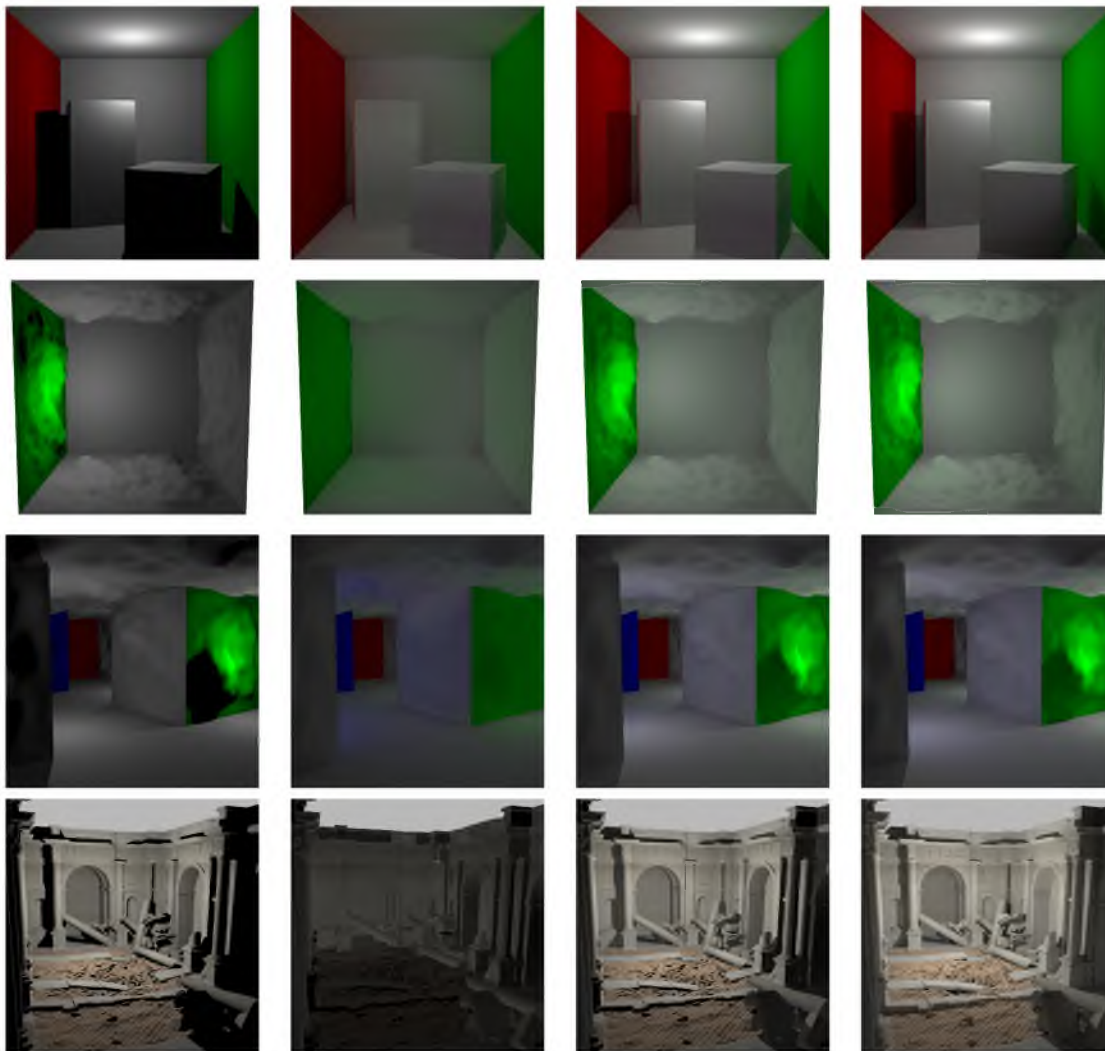
Figure 3.12: Lighting responses computed by Modular Radiance Transfer as compared to a reference path tracing solution for four different scenes. Column 1 shows direct light. Column 2 shows one-bounce indirect light computed using MRT. Column 3 shows combined direct and one-bounce indirect light computed using MRT. Column 4 shows the reference solution computed using path tracing.
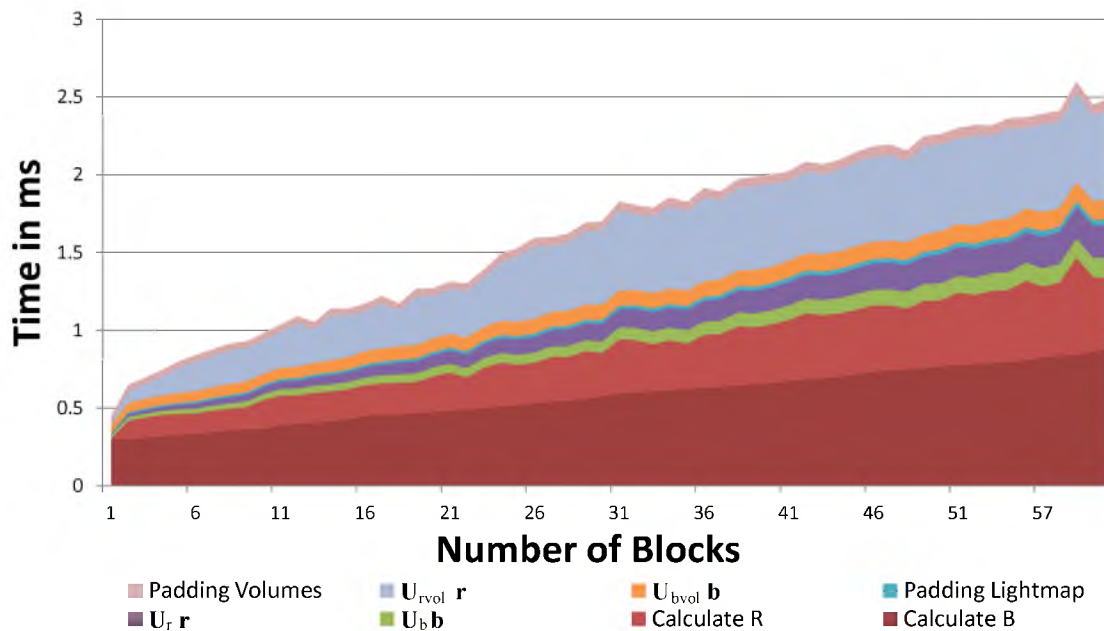
Figure 3.13: Average performance taken from many random mazes.

samples and vector irradiance. We measure performance for two scenes on the CPU code paths in Table 3.3. For comparison one unshadowed point light takes 25 ms on the iPad, making VPL techniques difficult.

3.7   Discussion

Instead of computing ground truth indirect effects, we target large-scale indirect lighting that responds to dynamic direct illumination, can be properly applied to fine scale surface detail, and supports animated character meshes. Our very high-performance allows our

Table 3.3: Performance results. The iPad uses 6^2 textures and 12 modes. The x86 and GPU use 16^2 textures and 32 modes.

|	Small Maze	Large Maze	
# blocks	11	41	
iPad ms	2.0	12.9	
x86 ms	1.4	6.2	
GPU ms	0.4	1.0	

runtime to be easily integrated across many hardware platforms, as well as significantly reducing iteration time (see accompanying video); lighting artists get **instant** indirect lighting feedback that is guaranteed to **match** the in-game rendering.

3.7.1 Limitations

When mapping scenes to a single box, using only the first mode is akin to an "intelligent ambient term" that is computed by averaging the direct light in the scene. Higher order terms add spatial variation. The bottom row of Figure 3.9 illustrates that even in the extreme case of a single cube mapped to complex geometry, our approach "fails gracefully," with smooth and plausible results. Of course, using a more accurate proxy shape would produce better results. With a single cube, light propagates too far along the longest axis of the scene's bounding box; we could alleviate this with a prior that uses the aspect ratio of the scene's bounding box, but almost equivalent results can be created simply by ray tracing the **U** vectors again with an appropriately stretched prior (Figure 3.4), taking only seconds to "re-precompute." Our algorithm is designed for scenarios where a small number of coarse proxies are mapped to complex geometry (like bounce cards in film production); using many proxy shapes to model this scene would defeat our purpose.

Another limitation stems from performing all computation in reduced spaces: lighting conditions outside our prior cannot be captured. Figure 3.14 highlights this issue, where a light is positioned too close to a wall. Shadows, strong albedo changes, and direct lighting used in video game scenes are outside of our prior, but plausible results are generated in these cases using enrichment (Appendix A).

We choose a simple prior, trading off accuracy for flexibility and performance. A more complex prior can be constructed using, e.g., permutable blockers in a scene and different light sources, but this would require a higher-dimensional representation. As an extreme
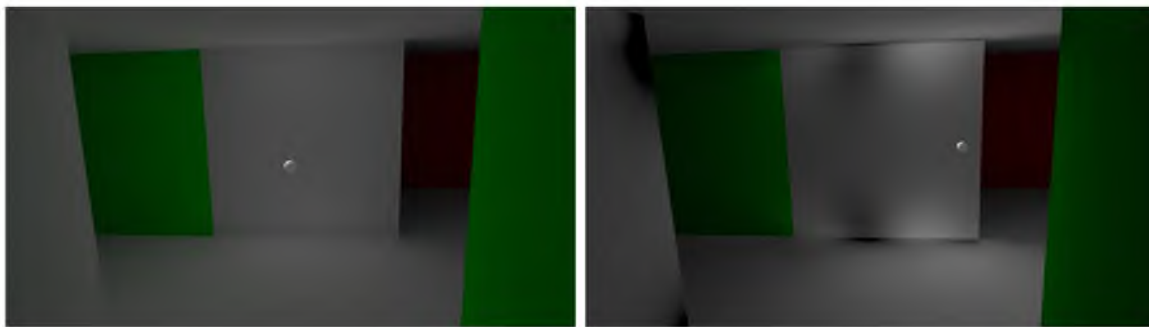
Figure 3.14: Indirect light from a well-represented (**left**) and poorly represented (**right**) direct-illumination pattern in the light prior.

example, we could use the scene from Figure 3.4 as a shape library element, yielding a lighting prior that would capture higher fidelity results for that particular scene (but for no other); however, especially when combined with direct illumination, it is clear that the additional fidelity is not worth the decrease in performance and the elimination of modularity/generality.

Our light prior bases are small, dense matrices derived using the SVD, in contrast to multiresolution bases such as wavelets, which are typically represented as large sparse matrices. Our dense matrices map more effectively to texture mapping hardware, have more coherent access patterns, and require less overall memory than wavelets, which is especially important on low-power handheld devices. This is particularly true with the 12 to 32 coefficients we use in practice.

3.7.2   Nondiffuse Transport

Scalar (diffuse) response can be substituted with a directional radiance representation (e.g., using a basis representation such as SH or wavelets). Our transport operators can also be augmented to support nondiffuse light transport. We note that low-frequency glossy reflectance models can still be applied to, e.g., our SH representation of volumetric light. We have instead targeted flexible and rapid scene development with physically plausible diffuse indirect illumination, as illustrated in our examples. Extending the simplicity of our approach to *high-frequency* glossy transport while keeping the benefits (compactness, high-performance, plausibility) is a challenging problem left to future work. Such directional radiance transport modeling may also prove useful for handling fine-scale indirect shadows, as we discuss below.

3.7.3   Indirect Shadows

Our approach models coarse-scale indirect shadows (e.g., around maze corners) while ignoring large-scale indirect shadowing effects within blocks. While this is sometimes problematic (e.g., the Cornell box in Figure 3.9), when direct lighting is included the lack of exact indirect shadows is often difficult to notice. In fact, in film production, shadow computation for the fill lights used to model indirect light is often disabled.

In the future, we plan on investigating techniques to include fine-scale indirect shadows from objects and clutter geometry *within* blocks by applying light subtraction ideas from antiradiance [63], as well as dynamic blocker accumulation and reflection from [21].

3.7.4   Dictionary Shapes

The restriction to basic shapes allows us to eliminate scene-dependent transport pre-computation. To support modularity, we must restrict the generality of the shape library. This constraint is softened during authoring by combining light transport coupling, warping, volume samples, and support for normal variation. It is possible to use more complicated shapes and extend our lighting prior to support, e.g., internal occluders.

3.7.5   Alternative Real-time Indirect Approaches

We achieve much higher performance than other approximate indirect lighting solutions, and our approach can readily be used in high-end and mobile gaming applications. Moreover, MRT scales favorably with respect to both the number of direct lights and the number of indirect bounces. However, these performance gains are only a by-product of the more substantial, novel contributions of our work: the lighting prior, transport computation in reduced spaces, and modularity (which also eliminates expensive scene-dependent transport precomputation). We feel that these ideas can be applied more generally to other areas in offline and real-time rendering, including other approximate indirect illumination techniques (e.g., instant radiosity).

CHAPTER 4

MRT RUNTIME IMPLEMENTATION[1]

4.1   U Basis Functions

In practice, our **U** basis functions are a set of textures that store data at many points on our dictionary shapes. For most of our experiments we used a dictionary of cubes, each with a different number of faces (see Figure 4.1). These cubes always have a floor and ceiling and are arranged to closely model whatever geometry we wish to light.

The default **U** basis functions are defined on the nonmissing faces of these cubes. Although the textures could be of any resolution, ours are defined with $16x16 = 256$ texels for each face. The **U** basis texture is computed as a preprocess using the methods presented in Section 3.3.1. The process is repeated for each dictionary piece (or cube) one at a time.

4.1.1   Computing the Lighting Prior P

We begin with the calculation of the P and S matrices for each block. First, calculate the direct lighting response at each texel center for multiple lights (L_d). In our tests we used sphere lights to generate the direct lighting. Inside each cube we use $6x6x6 = 216$ light positions, and for every missing face we add another $3x6x6 = 108$ samples to represent

| 2 Faces | 3 Faces | 4 Faces | 4 Faces | 5 Faces | 6 Faces |
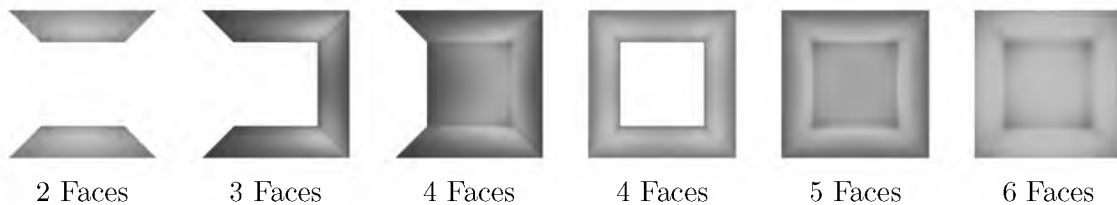
Figure 4.1: Our cube-based dictionary. Here you can see the six components of our dictionary, each one a cube with certain faces removed. These dictionary elements are sparsely pieced together to sparsely represent the actual geometry of a game level.

lighting coming into the cube from the outside. For cubes that are missing two contiguous faces we also sample another $3x3x6 = 54$ samples kitty-corner to the cube.

Once the direct lighting values are stored in a matrix, with the direct lighting results from each sphere position in a separate column, we take the SVD of that matrix, storing the U_d as the lighting prior (P)and the Σ_d as the prior scale values (S), but instead of keeping all the singular triples (a.k.a. modes), we store just the first 64 of them, which results in 64 columns of P and 64 values in Σ_d.

4.1.2   Computing the U Basis Funtions

The next piece we need to calculate is M, which will allow us to compute U_m, Σ_m, and V_m^T. M is defined as $M = FPS$ so we simply apply the operator F to P from section 4.1.1. To do this, we generate the indirect lighting at every texel center when the direct lighting is defined to be the modes of P applied to the walls of our shape. These ray-traced results are stored in a matrix where the indirect lighting from the first prior is in the first column, the second prior in the second column, and so on. This results in an FP matrix that has 64 columns of $16x16x$faceCount samples each. We multiply FP by S (also shown in section 4.1.1), and we have M (which also has $16x16x$faceCount rows). Then we take the SVD of M to generate U_m, Σ_m, and V_m^T.

As we explained in section 3.3.1 $U_b = \tilde{U}_m \tilde{\Sigma}_m$. These tildes represent that we have truncated the number of rows and singular values to simplify our computation. Since the SVD has already sorted the rows, singular values, and columns by energy, we can simply remove the lower rows to generate our truncated values. On the desktop we truncated our matrices to 32 modes (a.k.a. rows/singular values) as we felt that this gave us the results we desired. Other experiments on iOS [76] used only 12 modes to reduce the GPU load when reconstructing the signal.

4.1.3   Storage

Our U basis texture for each dictionary shape represents 32 values per pixel, the ones with the most energy being in the earlier modes. To allow easy access to this information during GPU calculations we store all the information in a single texture (see Figure 4.2).

Each face of each piece in our dictionary has $16x16 = 256$ samples. The texture consists of many of these $16x16$ samples, one for each face of each dictionary piece. If you look at the texture (Figure 4.2), you can see this organization as little squares. We use the color channels of the texture to store the modes. This means that we require 8 rows of $16x16$ samples to store all the data.
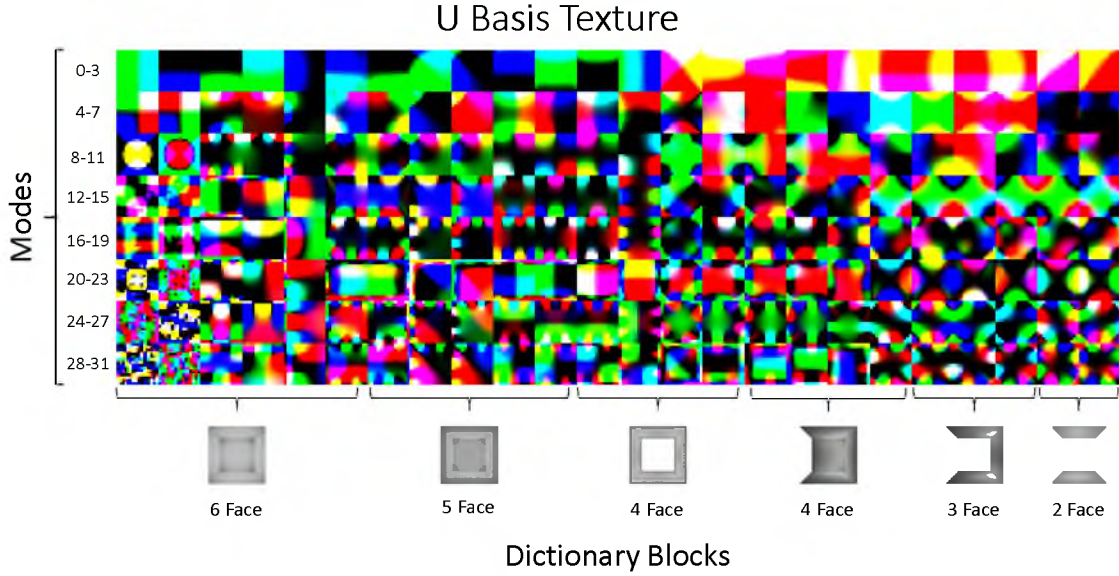
Figure 4.2: The U Basis Texture. The U textures from each dictionary piece are placed into a single large texture. The faces of each dictionary shape are stored horizontally, while the modes of each face are stored vertically.

4.1.4   Other U Basis Textures

With the information we have already computed (§ 4.1.1 and § 4.1.2) we can generate other basis textures by ray tracing the results of $L_{imp} = PSV_m$ (see Section 3.3.1). This allows us to generate our U_{vec} and U_{vol} basis functions defined in section 3.3.1.

4.1.5   Calculating $T_{d \to b}$

The other portion of the preprocessed dictionary we need is the ability to project our direct lighting results into the proper basis so they can be multiplied by U_b to generate the indirect lighting. We already have all the pieces we need to generate this operator. As shown in section 3.3.1, $T_{d \to b} = \tilde{V}_m^T S^{-1} P^T$, the calculations of which were explained in §4.1.2, §4.1.1, and §4.1.1, respectively.

Like the U basis texture, the $T_{d \to b}$ texture (see Figure 4.3) for all dictionary pieces is stored in the same texture, with the data for each face stored from left to right and the modes stored from in color channels going top to bottom.

4.2   Interface Functions

With our technique for calculating and storing the U basis functions explained, we now move onto how to create the interface basis that allows us to pass indirect lighting signals between dictionary pieces.

Figure 4.3: $T_{d \to b}$ texture. This texture, along with U gives us the data to convert b-coefficients into indirect lighting.

4.2.1 Out of a Dictionary Block

To generate $R_{b \to rlf}$ (§3.4.1) we first create a lightfield at each missing cube face for each dictionary element. To do this, we create a set of $12x12$ spatial samples covering each opening. At each spatial sample, we store the result of $24x24$ ray-traced directions, each subsampled at a resolution of $5x5$. This ray tracing is performed against all 32 modes of the implicit lighting (see Section 4.1.4) applied to the walls of the dictionary element.

Since some dictionary elements have more interfaces than others, we need to normalize the values of each interface to represent the same amount of radiance leaving each element. To do this we calculate the sum of the squared singular values Σ_m^2 from the original ray tracing of the lighting prior (section 4.1.4) for each element (which we will call $\text{Sum}_{element}$), as well as the sum total of Σ_m^2 over all elements (which we will call Sum_{total}). For each element, we calculate a scale factor

$$\text{scaleFactor} = \sqrt{\frac{\text{Sum}_{total}}{\text{Sum}_{element}\text{element\_count}}} \tag{4.1}$$

For each mode of each interface we store scaleFactor $* \text{Sum}_{element}$ for later use.

Now we are ready to reduce the dimensions of our signal. We take the multimodal light field we created at each dictionary element opening and put them all into one large matrix H_{lf} (§3.4.1), with the light field values of the first mode of the first interface in the first column, the second mode in the second column, and so on. Once all 32 modes have been inserted, we then insert the 32 modes of the next interface, creating an mxn matrix H_{lf} where $m = (\text{spatial samples})^2 * (\text{directional samples})^2 = 12 * 12 * 24 * 24 = 82944$ and $n = \text{modes} * \text{interfaces} = 32 * 11 = 352$. We then scale each column by (4.1) to normalize each basis.

Now that we have our H_{lf} we take the SVD to generate $U_{lf}\Sigma_{lf}V_{lf}^T$ and keep the first 128 modes (rows of U_{lf} and singular values of Σ_{lf}) to generate our reduced light field basis H_{rlf} (§3.4.1). To generate $R_{b\rightarrow rlf}$, we multiply H_{rlf} and H_{lf}, which results in a $32x128$ operator for each interface.

4.2.2   Between Dictionary Blocks

To move the values between dictionary blocks, we need to create operators to move the values between interfaces on a given block. To do this, we put the full U_{rlf} (§4.2.1) on one interface and ray trace from the other three interfaces. This will give us $R_{\rightarrow}$, $R_{\nwarrow}$, and $R_{\uparrow}$.

4.2.3   Into a Dictionary Block

To generate the response operator U_r (see Figure 4.4) we again ray trace U_{rlf} (§4.2.1) mapped to an interface. Due to the symmetry of our block dictionary elements, we use a single interface to generate results on the other five faces, which can then be used for any other element in our dictionary. We store the results in a matrix G (§3.4.1) where the ray-traced values of mode 0 for each of the five faces are stored in the first column and mode 1 in the second column, all the way to the 127th mode. We then scale each column of our matrix by Σ_{rlf} (which is the same as ray tracing against $H_{rlf} = U_{rlf}\Sigma rlf$ as stated in section 3.4.1).

Now we take an SVD of G resulting in U_g, Σ_g, and V_g^T, which will allow us to calculate $U_r = \tilde{U}_g$ (seen in Figure 4.4) and $T_{rlf\rightarrow r} = \tilde{\Sigma}_g\tilde{V}_g^T$.
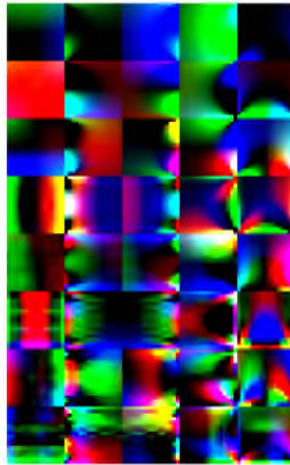
Figure 4.4: The U_r interface response texture. This texture contains the basis functions used to light our dictionary elements, which will be indexed into using r-coefficients.

4.3   Mapping MRT to Levels

Modular radiance transfer is calculated on our basic dictionary shapes, but its intent is to be applied to actual game levels.

4.3.1   Creating Block to Level Mapping

While our dictionary blocks seem rigid and rectilinear, they can be remapped to fit more organic level geometry. Due to the low frequency nature of indirect lighting, and the approximate nature of our approach, the errors introduced in doing so are minimal.

The level in Figure 4.5 is an example of one such remapping. It shows our rectilinear dictionary pieces can be connected and their vertices moved to represent actual geometry. This block to level mapping is done when the level is created. Currently this extra geometry is created manually, not dissimilar from how physics geometry is created for games today.

For the rest of this chapter we will be using a small, simple test level to demonstrate our techniques. A rendering of this level can be seen in Figure 4.6. It will show how we apply modular radiance transfer on, between, and within dictionary blocks.

This level consists of three separate piece of block geometry that map to two dictionary pieces (two four- and two five-piece blocks). In our example the vertices of the geometry and the dictionary blocks are coincident.

4.3.2   Calculate Block to Block Interface Operators

Once we have a mapping of blocks to an actual level, we need to calculate the interface operators that define how light passes through the interfaces between all the blocks (Figure 4.7). For all block pairs that can see each other we create a path between them using $T_{b \to rlf}$, $R_\uparrow$, $R_\nearrow$, $R_\searrow$, and finally $T_{rlf \to r}$. This results in a $32x32$ matrix for each block pair. In our example map (Figure 4.7) you can see we generate 6 such matrices. We stored these matrices in a single large horizontal texture with each matrix to the left of the last

Level Geometry   Level Closeup   Blocks on Level   Blocks Only
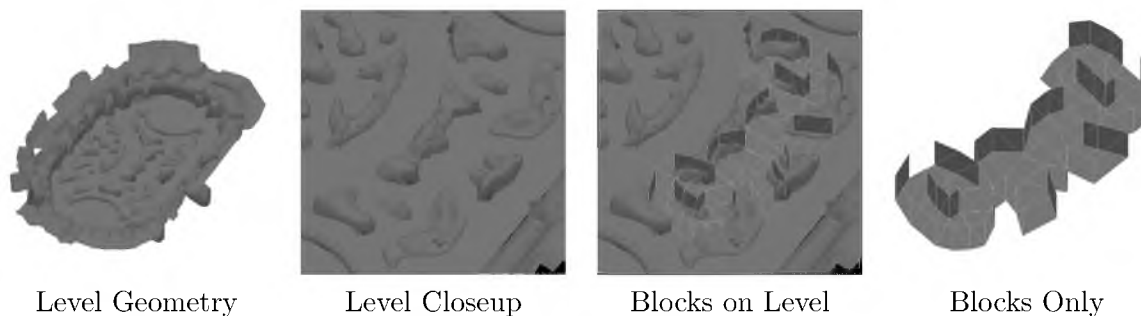
Figure 4.5: An example of block to level mapping.

Figure 4.6: Our example level for explaining the MRT runtime.
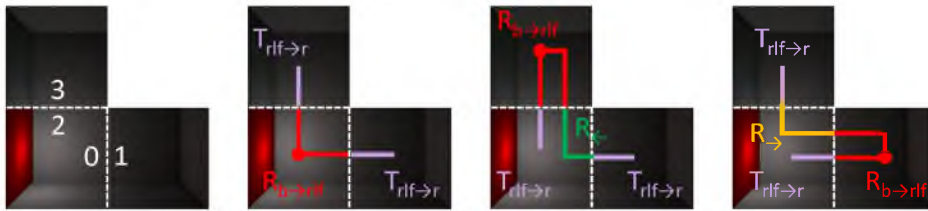
Figure 4.7: All interface operators we need to store for our example map. On the left you can see there are three blocks and four interfaces. The second column shows the interface operators for block one, and the last column the interfaces for block two.

one, although other layouts may work better when more interfaces are used. This texture is *not* scene independent. It will need to be calculated once for each new level created.

4.4   b-coefficients

The first step in modular radiance transfer is creating a set of *b*-coefficients, one set for each dictionary block. These *b*-coefficients will allow us to generate the lighting shown in Figure 4.8.

4.4.1   Generating Per-Block *b*-coefficients

To generate the *b*-coefficients, we first generate a set of records, one for each face of each dictionary block in our level. In our test map, that creates 14 records, four for the front left and five each for the back and left block. Each record has uv coordinates at each vertex that index into the U and $T_{d \to b}$ textures.

Each record covers 16x16 pixels in a render target. We calculate the direct lighting for each pixel using a geometry map that describes the position and normal of the level geometry that face of the block is supposed to represent. The direct lighting output is input to another pass where we use eight render targets for the 32 modes of the direct lighting
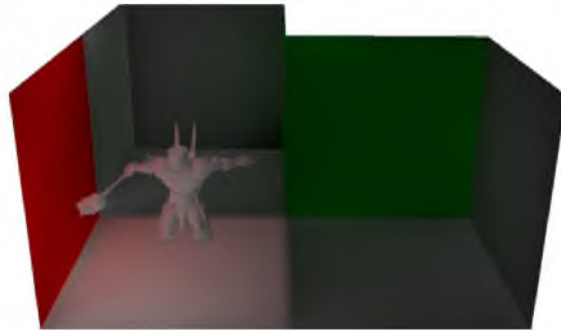
Figure 4.8: Output of $U_b b + U_{vol} b$ on our example level.

multiplied by the associated $T_{d \to b}$ data. The output of that pass is reduced twice to sum the products into a single set of 32 b-coefficient per block.

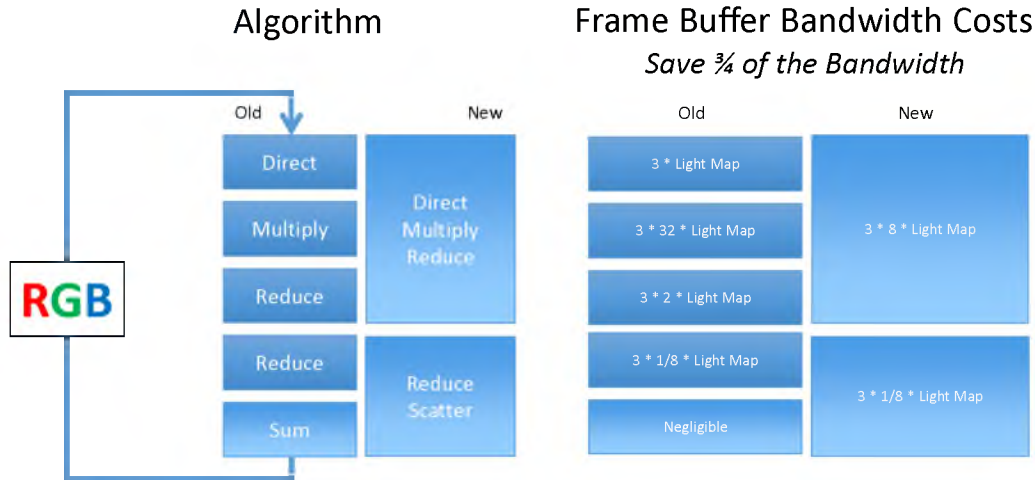### 4.4.2 Optimizing the Calculation

Unfortunately, doing this calculation three times (for r, g, and b) results in too many passes, slowing down the calculation. Also, a separate pass for each color requires us to recalculate the direct lighting 3 times, which is wasted work.

Instead of storing the modes in multiple render targets, we decided to store them vertically like in the U and $T_{d \to b}$ textures. This allows us to calculate the direct lighting once and reduce the number of passes.

The sum at the end is also problematic due to the fact that for any given level there are not many blocks defined, which results in too little parallelism for the GPU. Also, due to each dictionary block having a different number of faces results in divergent flow. Instead of summing all the results ourselves in the shader, we decided to enable blending and allow the hardware to do the final sum for us. We tried summing 4, 16, and 64 values in the shader ourselves, leaving the rest for hardware blending. Too few sums calculated in the shader and we overload the frame buffer bandwidth, too many shader sums and there is not enough parallelism for the GPU. We found that using 64 in-shader sums worked best, as long as the pixels we output were not localized, so as to reduce frame buffer contention. All this work reduces the number of passes and frame buffer bandwidth.

To reduce more frame buffer bandwidth, we also merged the direct lighting calculation, the $T_{d \to b} * l_d$ multiplication, and the first reduction level so we only use one pass. With all these changes, we reduce the frame buffer bandwidth by 75% (see Figure 4.9).

Figure 4.9: Graphical representation of the optimization of the calculation of b-coefficients.

4.4.3 b-coefficients to Lighting

Now that we have a b-coefficient for each block, we can use those data, along with the U basis texture, to create a light map for each dictionary piece (see Figure 4.10). When applied over the direct lighting, we get the results shown in Figure 4.8.

4.5 r-coefficients

Now we need to calculate the indirect lighting that is traveling between the blocks. The result of this calculation can be seen in Figure 4.11. To do this, we convert b-coefficients into r-coefficients and use them to apply the U_r basis functions to the faces of our dictionary elements.

4.5.1 b-coefficients to r-coefficients

Starting with the set of b-coefficients, we convert those values using the $T_{b \to r}$ texture that we precomputed for our level (§4.3.2).

We start with a $3x1x8$ texture representing the 32 modes of block's b-coefficients. We then create an output render target that is $4x1x8$ to store the 32 modes of the r-coefficients – one for each interface, see Figure 4.7. To this render target we render six records – one for each interface operator, see Figure 4.7. Each record will contain uv coordinates that reference our $T_{b \to r}$ texture. This will allow us to perform the calculation $T_{b \to r}b$, which will result in a 32-entry vector that we store in the four color channels of the eight render targets of our output.
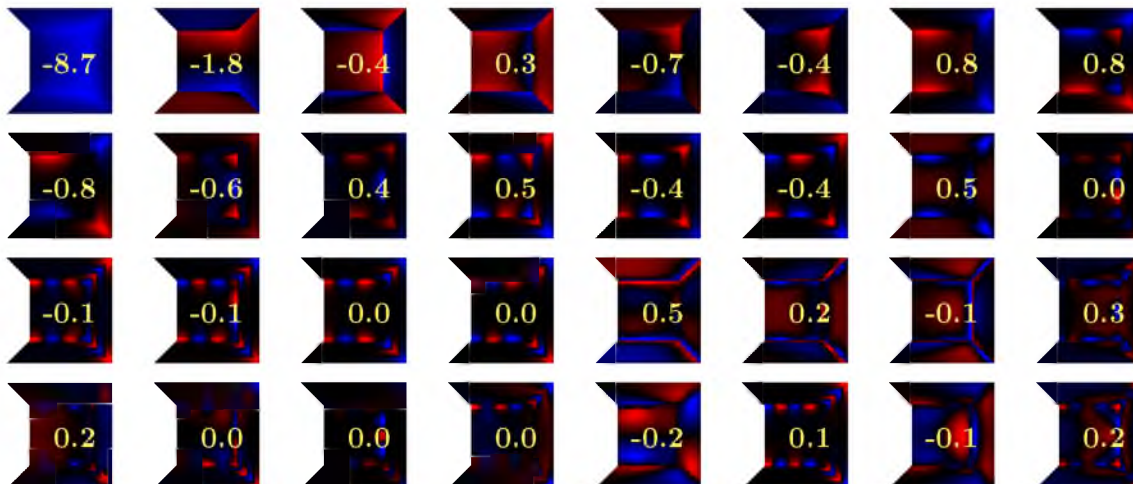
Figure 4.10: Per-pixel U basis shown on the five-face dictionary block. In the center is the weight of each mode in our example map. Red represents positive light values and blue represents negative light values. Mode 0 is in the top left, going left to right, top to bottom.
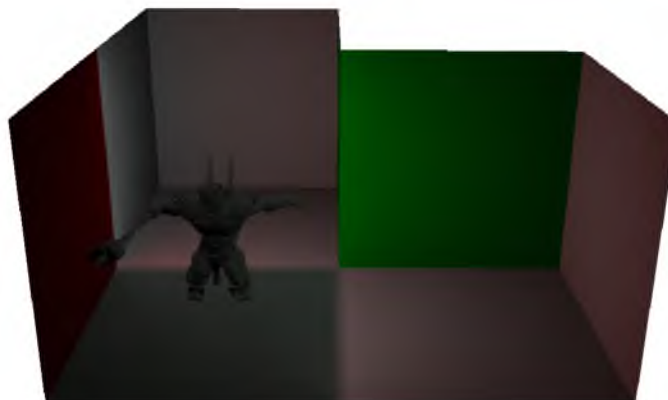
Figure 4.11: Output of $U_r r$ on our example level.

4.5.2 r-coefficients to Indirect Lighting

With our $4x1x8$ r-coefficients, we can now apply the indirect lighting to the walls of our cubes. To do this, we create a $16x16$ pixel render record for each of the 14 faces in our example map. Each record will have uv coordinates that reference both the location in the r-coefficient texture and the proper location in the U_r texture. Then, the 32 modes of each $16x16$ U_r texture are scaled by the 32 modes of the r-coefficient to give us the indirect lighting for each face.

4.6   Lightmap Padding

With all indirect lighting within and between block computed, we have one more small step to ensure a smooth lighting solution, and that is to pad our light map.

4.6.1   Additional Light Map Records

For our example map we have been dealing with $16x16$ pixels on each face of our dictionary elements. To create a padded light map we copy these $16x16$ face records to another texture in a new location. This new texture has a single pixel border around each $16x16$ face record.

When we create our block to level mapping (§4.3.2), we also create a new set of edge ($16x1$) and corner ($1x1$) records. These records copy data from the edges and corners of other face records into the single texture border around the $16x16$ lighting values. Once we offset our texture coordinates by half of a pixel, it ensures that the texel values used on the edges of each face match exactly. This is shown in Figure 4.12.

4.7   Results

We ran this algorithm for block mazes of different sizes and connectivity graphs. A graph of these results can be found in in Figure 4.13.

Figure 4.12: Indirect lighting on our example map without light map padding. When we do not use light map padding, we end up with discontinuities between our blocks.
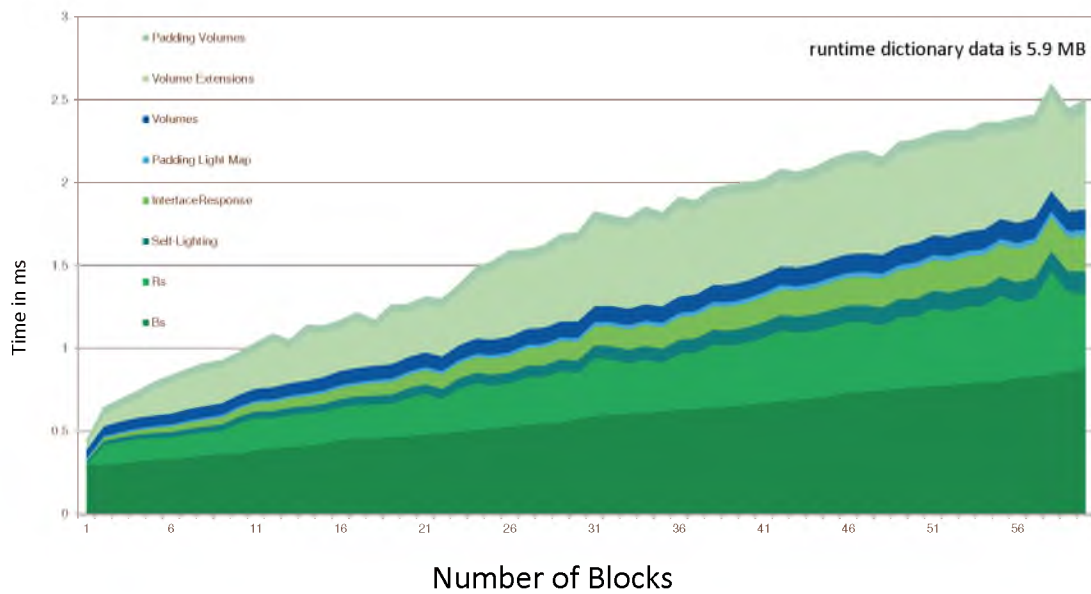
Figure 4.13: Timings for different parts of the MRT runtime. Times are shown in milliseconds for block mazes with different numbers of randomly connected blocks.

CHAPTER 5

DELTA RADIANCE TRANSFER[1]

5.1   Introduction

Interactive graphics applications have started integrating approximate global illumination, often to satisfy art-driven requirements. Precomputation based approaches (e.g., direct-to-indirect transport) are capable of satisfying these requirements; however, long preprocessing times limit their wider-scale adoption.

In Chapter 3 we introduced *Modular Radiance Transfer* (MRT), a coarse-scale direct-to-indirect transport approach leveraging *scene-independent* precomputation. MRT aggregates indirect transport inside and between *simple shapes*, modeling how light is transported when shapes are warped and attached to each other. Level designers can transform and connect shapes to author new scenes or form lighting volumes inside existing scenes. This new style of light transport authoring eliminates time-consuming scene-wide precomputation.

While MRT models coarse-scale light transport within and between shapes, it ignores the effects of finer-scale "clutter geometry" (e.g., a pillar or desk in a room). Specifically, objects inside the simple shapes do not affect light transport at all: they do not cast indirect shadows, nor do they reflect indirect light onto the shape and its neighbors (see Figure 5.1). MRT's generality precludes efficient incorporation of these effects; introducing sharp shadows/interreflections breaks many of MRT's assumptions about the nature of the light transport's dimensionality.

We introduce **Delta Radiance Transfer** (DRT) to carefully remove these constraints while maintaining the important benefits of MRT:

- **High performance:** maintaining the extremely high-performance of MRT is necessary to promote its applicability to content-generation pipelines;
- **Low-dimensional rendering:** in adding finer-scale occlusion and interreflection to MRT, we need to extend its low-dimensional rendering formulation to
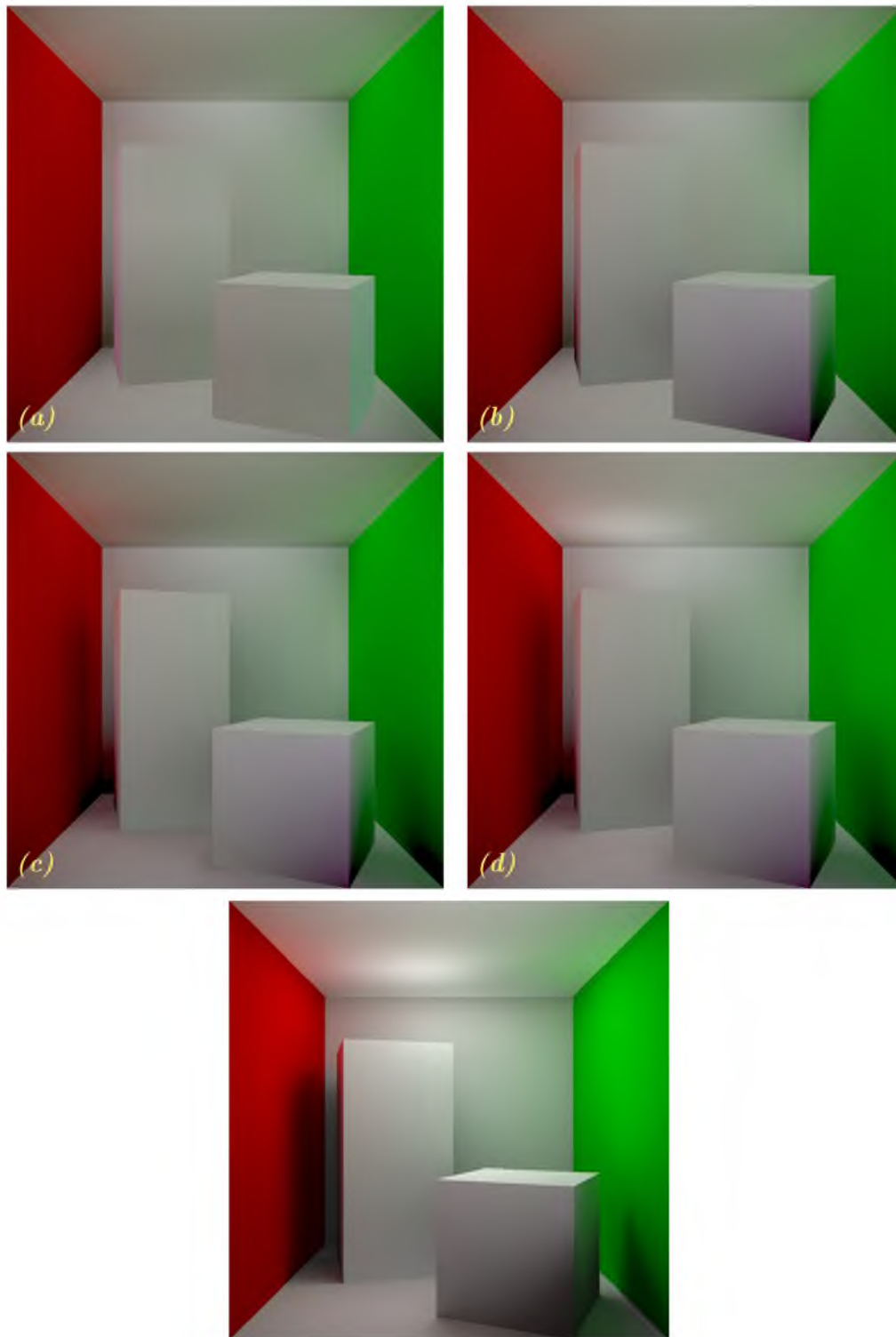
Figure 5.1: Modular Radiance Transfer (a) takes 1.7 ms and does not accurately model light scattered onto "clutter objects" (e.g., the two boxes) (b) or indirect shadows (c) and interreflections (d) from clutter onto the scene. We improve accuracy by adding these effects, at little cost to performance (1.25 ms total). The bottom image shows the sum of all four operators ($a + b + c + d$).

support this added complexity without incurring a substantial performance overhead;

- **Rendering extensions:** supporting dynamic vector irradiance and volume light probes in the presence of occlusion and (potentially near-field) interreflection enables compatibility with other common high-fidelity real-time rendering techniques.

While MRT models indirect light between large scene blocks, we introduce three new compact light transport operators to model the following transport paths missing from MRT (see Figure 5.2):

- indirect shadows from clutter onto the scene (Section 5.3.1),
- interreflections from clutter onto the scene (Section 5.3.2), and
- interreflections from the scene back onto clutter (Section 5.3.3).

In MRT and DRT, indirect light is computed as a weighted sum of dynamically generated lightmap textures. These textures are specially constructed to represent basis-space light transport. Each of our new operators are computed by ray tracing against the lightmaps parameterized over the scene geometry, or the clutter, depending on the transport path being modeled.

5.2 Background – Modular Radiance Transport

Given direct light in a scene, e.g., generated with shadow mapping, MRT computes coarse-scale dynamic indirect light as well as dynamic vector and volumetric radiance to support high-frequency normal variation (e.g., normal maps) and limited shading of clutter.

MRT computes these effects with high-performance on a range of hardware platforms using the key ideas of *modularity* and *low-rank computation*: indirect light transport is
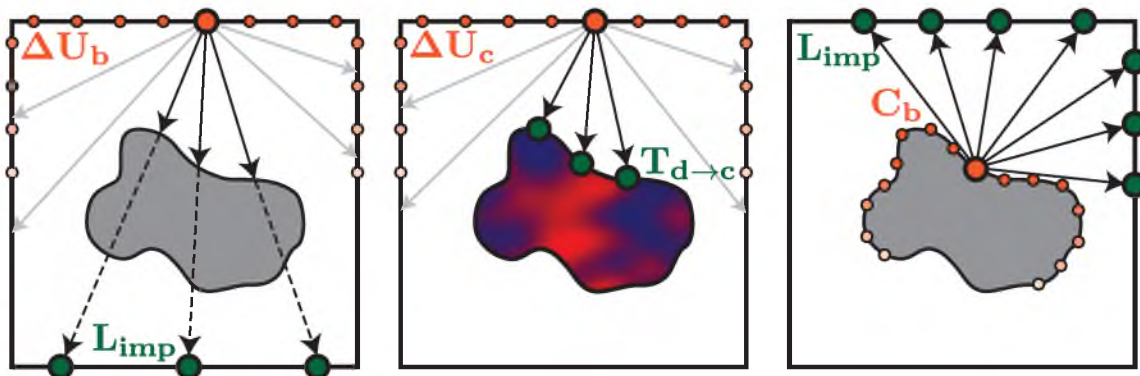
Figure 5.2: The three additional transfer operators we compute for clutter. From left to right: occlusions onto scene (antiradiance), diffuse reflections onto scene, and diffuse reflections onto clutter.

decomposed into the effects *within* and *between* shapes, and these transport paths are computed and coupled entirely in optimized low-dimensional subspaces.

After giving a brief introduction to light transport using matrix operators, we will discuss the modularity and low-rank nature of MRT and then build on top of these concepts in Section 5.3.

5.2.1 Matrix Light Transport and Naïve SVD

Indirect light L_{ind} can be computed by applying a continuous linear operator to the direct light L_{d} in a scene as

$$L_{\text{ind}}(x) = \int_{\Omega_n} L_{\text{d}}(x', -\omega)\, f(x, \omega)\, (n_x \cdot \omega)\, d\omega = \mathcal{F}\{L_{\text{d}}\}(x) \tag{5.1}$$

where x is a point in the scene, n_x is the normal at x, Ω_n is the set of all unit direction vectors in the upper hemisphere about n_x, $x' = \mathbf{ray}(x + t\,\omega)$ is the nearest surface point from x in direction ω given by the ray-tracing operator $\mathbf{ray}$, f is the BRDF at x, and $\mathcal{F}$ is the continuous one-bounce direct-to-indirect transport operator.

We assume diffuse relighting, where $\mathcal{F}$ can be discretized to yield the (discrete) direct-to-indirect transport equation: $\mathbf{l}_{ind} = \mathbf{F}\,\mathbf{l}_{\text{d}}$, where indirect light $\mathbf{l}_{ind}$ is computed by applying the (discrete) one-bounce operator $\mathbf{F}$ to the direct light $\mathbf{l}_{\text{d}}$. Each element of $\mathbf{l}_{ind}$ and $\mathbf{l}_{\text{d}}$ represents outgoing radiance at a different surface location.

Evaluating this discrete equation is expensive and limits the performance of direct-to-indirect transport since $\mathbf{F}$ grows proportionally with $\mathcal{O}(d^2)$, where d is the spatial discretization of the scene.

A common acceleration strategy is to take the singular value decomposition (SVD) of $\mathbf{F}$ and approximate the matrix-vector product using a rank-reduced $\mathbf{F} = \mathbf{U}_f\, \mathbf{\Sigma}_f\, \mathbf{V}_f^{\text{T}}$, where $\mathbf{U}_f$ and $\mathbf{V}_f^{\text{T}}$ contain the left and right singular vectors of $\mathbf{F}$, and $\mathbf{\Sigma}_f$ is a matrix with the singular values σ_i of $\mathbf{F}$ along its diagonal. The discrete transport equation can be approximated by keeping the r largest σ_i.

Unfortunately, as discussed in [75], the singular values of $\mathbf{F}$ fall-off too slowly to yield high-performance using this approximation technique. We will discuss how MRT accelerates evaluation of the (discrete) direct-to-indirect transport equation while inducing a more controlled degradation of accuracy.

5.2.2 Lighting Prior and Implicit Lights

MRT takes a unique approach to accelerating the (approximate) computation of $\mathbf{l}_{ind}$, computing light transport entirely in low-dimensional spaces, by exploiting key observations

1. *plausible* direct lighting in a scene lies in a low-dimensional, highly correlated subspace of all input signals, and

2. applying $\mathbf{F}$ to these highly correlated direct lighting signals yields highly correlated indirect illumination patterns.

Using the SVD of $\mathbf{F}$ to accelerate the matrix-vector product does not account for possible correlations in the input (direct) light patterns, $\mathbf{l_d}$. This SVD is optimal if the $\mathbf{l_d}$s are drawn from *an arbitrary distribution*; however, in reality, they are drawn from the more restrictive set of *possible direct lighting signals*. This set has many correlations, leading to correlations in the resulting indirect light.

5.2.2.1   Lighting Prior

MRT precomputes direct illumination from a set of lights placed uniformly in the volume of a scene shape. By treating each direct light output as a column in a matrix and taking its SVD, the first n left singular vector columns $\mathbf{P}$ yield a low-dimensional basis for direct illumination, called the *lighting prior*.

Correlations in direct lighting cause the effective dimension n to be much lower than the explicit dimension (number of surface locations). To exploit this correlation when computing indirect lighting, MRT first defines $\mathbf{M} = \mathbf{F\,P\,S}$, where $\mathbf{S}$ is the diagonal singular value matrix associated with $\mathbf{P}$. Taking the SVD of $\mathbf{M} = \mathbf{U\,\Sigma V^T}$, MRT's low-dimensional direct-to-indirect transport equation is

$$\mathbf{l}_{ind} \approx \mathbf{U_b}\,\mathbf{T}_{d \to b}\,\mathbf{l_d} = \mathbf{U_b}\,\mathbf{b} \tag{5.2}$$

where $\mathbf{U_b} = \mathbf{U\,\Sigma}$ and $\mathbf{T}_{d \to b} = \mathbf{V^T\,S^{-1}\,P^T}$ projects[2] direct light $\mathbf{l_d}$ to the correlated low-rank indirect light space (with corresponding coefficients $\mathbf{b}$). Despite the derivations outlined above, MRT simply precomputes $\mathbf{U_b}$ and $\mathbf{T}_{d \to b}$ (in a few seconds), and all runtime computations are performed in the low-dimensional spaces.

5.2.2.2   Implicit Lighting

The columns of $\mathbf{U_b}$ are indirect *basis light patterns*. Once $\mathbf{l_d}$ is reduced to $\mathbf{b}$ with $\mathbf{T}_{d \to b}$, (5.2) scales these patterns by the elements of $\mathbf{b}$ to yield dynamic indirect light.

MRT exploits an alternative approach for generating the columns of $\mathbf{U_b}$. Namely, basis lighting patterns result from the application of the one-bounce operator to a set of *implicit lighting patterns*, $\mathbf{U_b} = \mathbf{F\,L}_{imp}$, where $\mathbf{L}_{imp} = \mathbf{P\,S\,V}$.

---

[2]We use rank-reduced matrices where necessary, e.g., $\mathbf{V^T}$.

DRT also uses $\mathbf{L}_{imp}$ to construct operators that model indirect shadows from clutter and interreflections *from* the scene *onto* clutter (Sections 5.3.1 and 5.3.3). DRT computes higher order radiance representations (e.g., for normal mapping) and radiance volumes with $\mathbf{L}_{imp}$, incorporating the occlusion/interreflection effects of clutter.

5.2.3   Inter and Intrashape Transport

MRT allows artists to map simple geometric primitives (e.g., cubes with a number of faces removed) to existing scenes or to build new scenes with them. Given a set of these (potentially warped) connected shapes, resulting from the content creation process, MRT dynamically updates indirect lighting in a modular fashion.

First, direct-to-indirect transport is computed *within* each shape as described in Section 5.2.2. Next, direct-to-indirect transport is propagated from each shape to all others. To do so, at level creation time MRT quickly computes a shape connectivity graph and, for each shape, concatenates light-transport operators, which progressively propagate the lighting to all affected shapes. A handful of low-dimensional operators, precomputed for each *interface* of a shape, map $\mathbf{b}$ coefficients for a shape to a low-dimensional lightfield at the interface. Additional precomputed operators map basis-radiance at the lightfields back onto the surfaces of adjacent shapes.

DRT additionally models the effects of finer-scale occlusions and interreflections (Section 5.3), introducing *delta occlusion operators, delta reflection operators,* and *clutter gather operators* (Sections 5.3.1, 5.3.2, and 5.3.3) to update block and clutter lightmaps. The later stages of our approach either output lightmaps over the clutter objects $\mathbf{l_c}$ or directly update the scene lightmap $\mathbf{l}_{ind}$.

5.3   Indirect Occlusions and Interreflections

Given an empty scene composed of blocks with implicit lighting environments $\mathbf{L}_{imp}$ (computed with MRT), as well as the clutter geometry for the scene, we define new low-dimensional operators to capture the indirect shadows and interreflections from the additional clutter objects. We also need new operators to bounce light from the scene back onto the clutter. MRT uses volume samples for this last scene-to-clutter lighting, whereas we will more accurately and explicitly model this transport with *clutter gather operators* (Section 5.3.3). Our scene-dependent computation also increases performance (see Section 5.4).

A brute-force solution is to recompute direct light in the scene (including clutter) several times, regenerate an updated lighting prior $\mathbf{P}$, and finally regenerate the scene's $\mathbf{U_b}$ operator.

This approach is unnecessarily complex, expensive, and ineffective: direct light on a scene with clutter has high-frequency shadows from the clutter, requiring more bases (a higher n) in the lighting prior $\mathbf{P}$, for accurate results. This would reduce MRT's performance and generality ("scene-independent" shapes would have to include clutter). However, the resulting low-frequency indirect occlusions and interreflections motivate an alternative approach.

The operators we define below are applied to clutter at each block; however, the resulting transport is propagated beyond the current block to all blocks within a predescribed neighborhood.

5.3.1   Scene Occlusions with Delta Occlusion Operators

We first consider indirect shadows *from* the clutter *onto* the scene.

After indirect lighting is computed on the scene surfaces using MRT (ignoring the effects of clutter), we subtract occlusion from this unshadowed shading similarly to antiradiance [63], while operating entirely in the existing low-dimensional subspace. To do so, we rely on the existing implicit lighting to construct the necessary light transport operator, as discussed below.

Recall that applying the direct-to-indirect operator $\mathbf{F}$ to the implicit lighting $\mathbf{L}_{imp}$ yields the $\mathbf{U_b}$ operator. We require an operator, $\mathbf{\Delta U}_b$, to capture (negative) basis light due to clutter occlusions on the scene surfaces.

We compute $\mathbf{\Delta U}_b$ directly using $\mathbf{L}_{imp}$, meaning we do not explicitly recompute direct-to-indirect transport: at each $\mathbf{\Delta U}_b$ texel in the scene, we shoot many uniformly distributed shadow rays. Each ray that intersects the clutter corresponds to a direction that will occlude light. We trace the intersected ray beyond the clutter until it hits a surface in the scene, where it is weighted by the $\mathbf{L}_{imp}$ basis function value associated with the texel at the surface location. Rays that do not hit the clutter clearly do not contribute to indirect occlusion and are ignored. The average of all ray values yields the $\mathbf{\Delta U}_b$ entry for the currently processed texel (Figure 5.2, left).

Conceptually, the columns of $\mathbf{\Delta U}_b$ are indirect *shadowed basis light patterns*, and we compute shading *including* indirect clutter shadows using $\overline{\mathbf{U_b}} = \mathbf{U_b} + \mathbf{\Delta U}_b$ instead of $\mathbf{U_b}$. This amounts to adding the negative light necessary to account for the indirect shadows from the clutter.

In practice, we need to address spatial sampling issues arising from resolution mismatches and misalignments between clutter and scene lightmaps. We discuss this issue in Section 5.4, as well as our proposed in painting solution (Figure 5.3).
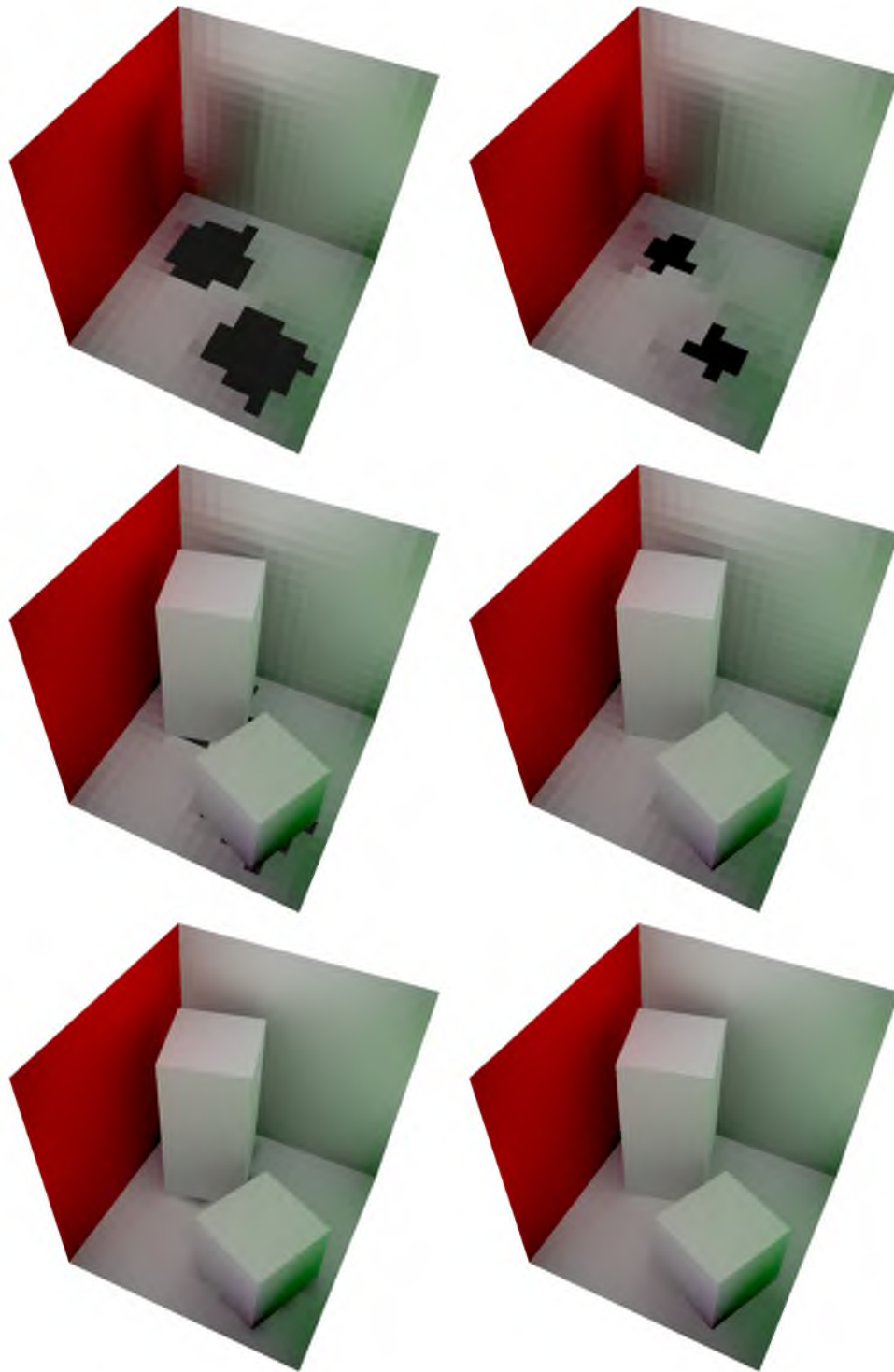
Figure 5.3: We *inpaint* regions inside clutter, ensuring smooth shading around objects that do not align with the scene lightmap texels. This figure shows the Cornell Box without (left) and with (right column) inpainting with two types of sampling: nearest neighbor (first and second row) and bilinear sampling (bottom).

5.3.2   Interreflections with Delta Reflection Operators

Next, we consider interreflections *from* the clutter *onto* the scene.

We note that all the operators we have introduced so far (e.g., $\overline{U_b}$) are not parameterized over the clutter geometry: **b**-coefficients are only computed from $\mathbf{l_d}$ defined over the scene's surfaces. For example, in the case of indirect shadows from clutter (Section 5.3.1), this shadowing only depends on the *presence* of the clutter geometry, not on the value of lighting over its surface. As such, the indirect shadows are dependent (in a parametric sense, as opposed to a geometric one) on the **b**-coefficients.

This is not the case when computing direct-to-indirect (basis-space) transport from the clutter onto the scene since we must derive new operators to map direct illumination *over the surfaces of the clutter* to interreflected light *onto the scene*. Here, the mere presence of clutter is not sufficient to parameterize this lighting effect.

As with direct lighting on the scene, direct light on the clutter is computed in a lightmap and projected into a low-rank basis. We found that using a simple orthonormal basis (constant radiance over each clutter mesh face) was sufficient for our examples, but for more complex clutter we could construct a lighting prior to obtain an optimal basis (as in MRT). In this simplified case, $\mathbf{T}_{d \rightarrow c}$ is our $m \times f$ transformation matrix that projects direct light into this basis (**c**-coefficients), where m is the lightmap resolution and f is the number of clutter mesh faces.

Given this basis over the clutter, we precompute indirect lighting responses *on the scene* to any signal represented in the basis (e.g., direct light) *over the clutter*: at each texel in the scene lightmap, we shoot many gather rays and intersect against the nearby clutter. These intersected rays correspond to directions that will bounce light from the clutter onto that surface location on the scene.

Each gather ray that intersects a clutter object samples the clutter lighting basis and accumulates diffuse (or vector valued) basis-lighting into our $\mathbf{\Delta U_c}$ operator (see Figure 5.2, center). This transport accumulation approach is similar to multibounce PRT transfer computation [18].

At runtime we compute direct light $\mathbf{l_d}$ in the clutter lightmap and project it into the reduced basis. The resulting **c**-coefficients are used to scale interreflection response textures (columns of $\mathbf{\Delta U_c}$) to yield bounced light from the clutter onto the scene.

5.3.3   Interreflections with Clutter Gather Operator

Lastly, we model interreflections *from* the scene *onto* the clutter. Our solution replaces the *volume samples* from standard MRT and also supports clutter shadowing and higher

sampling densities.

Similarly to delta occlusion maps, we define a $\mathbf{C_b}$ operator that maps $\mathbf{b}$ to indirect light (in a lightmap) over the clutter. The bounced light clearly depends on the scene's $\mathbf{b}$-coefficients, but the output space is defined over the surface of the clutter.

For each row (texel) of $\mathbf{C_b}$, we gather light by tracing rays in all directions from the current clutter surface point, averaging the rows of $\mathbf{L}_{imp}$ associated with each ray hit locations on the scene surface (Figure 5.2, right). As in Section 5.3.1, rays that do not intersect the scene do not contribute bounced light.

5.4   Implementation Details

Our simple runtime requires only a handful of small matrix-vector multiplies. Our additions to the standard MRT runtime are

- Add indirect shadows to the scene's $\mathbf{l}_{ind}$ due to clutter, using the *delta occlusion operator* $\mathbf{\Delta U}_b \mathbf{b}$,
- Compute clutter $\mathbf{l_d}$ and project to $\mathbf{c}$ with $\mathbf{T}_{d \to c}$,
- Add interreflections *to the scene's* $\mathbf{l}_{ind}$ from clutter $\mathbf{\Delta U_c} \mathbf{c}$,
- Compute interreflections *from* the scene *onto* the clutter geometry $\mathbf{l_c} = \mathbf{C_b} \mathbf{b}$.

New response textures, computed using our operators, model changes due to occlusions / interreflections to the base shading (computed with standard MRT). We simply blend these intermediate textures into either the scene ($\mathbf{l}_{ind}$) or clutter lightmaps ($\mathbf{l_c}$). We outline the end-to-end algorithm below, as well as details that need to be considered during data generation.

5.4.1   End-to-end Algorithm

Direct illumination is first computed using any standard approach; we use shadow mapping. Notably, direct illumination must be explicitly computed or mapped to the spatially subsampled surface locations that are used to parameterize the $\mathbf{l_d}$ input vector. Subsampled direct illumination is mapped to low-dimensional indirect lighting coefficients $\mathbf{b}$, which will drive the remainder of the MRT components of the algorithm, as well as DRT's indirect shadows from clutter onto the scene and interreflections from the scene onto clutter.

For each block, indirect lighting is computed, ignoring clutter geometry, using (5.2). Light between each block (still ignoring clutter) is propagated using low-dimensional light field propagation operators [75]. MRT also pays careful attention to compute padding regions in the output lightmaps to avoid visible seems when blocks are connected together.

At this point, DRT computes the $\mathbf{\Delta U}_b$, $\mathbf{\Delta U}_c$, and $\mathbf{C_b}$ transport operators, as well as the $\mathbf{T}_{d \to c}$ basis projection matrix.

Indirect shadows from the clutter onto the scene are computed as $\mathbf{\Delta U}_b\,\mathbf{b}$, after which we compute (spatially subsampled) direct illumination on the clutter and project it onto the direct light subspace using $\mathbf{T}_{d \to c}$, yielding $\mathbf{c}$-coefficients.

Indirect bounced light between the scene and clutter are computed in the last stages of DRT. First, interreflections from the clutter onto the scene are computed as $\mathbf{\Delta U}_c\,\mathbf{c}$, and then interreflections from the scene onto the clutter are computed as $\mathbf{C}_b\,\mathbf{b}$.

5.4.2   Transport Inpainting

$\mathbf{\Delta U}_b$ texels corresponding to scene locations *inside* clutter will have all their rays intersecting a clutter object, and the resulting transport will be incorrect.[3] In order to populate these texels with meaningful transport entries, we average transport from all valid (e.g., not inside clutter) neighboring texels. Note that, when inpainting from valid neighbor entries, we average $\overline{\boldsymbol{U_b}}$ entries and then subtract the unshadowed $\mathbf{U_b}$ of the destination texel, instead of averaging $\mathbf{\Delta U}_b$ entries. Other inpainting methods, e.g., gradient interpolation schemes, are also suitable; however, our simpler approach yields pleasing results (Figure 5.3). It is possible that inpainting can fail if there are no samples with valid $\mathbf{\Delta U}_b$ entries nearby (e.g., with high frequency geometry), but we did not encounter this limitation in our test scenes.

5.4.3   Vector and Volume Response

We also compute *vector-valued* $\mathbf{\Delta U}_b$, $\mathbf{\Delta U}_c$, and $\mathbf{C}_b$ operators ($\mathbf{\Delta U}_{\vec{b}}$, $\mathbf{\Delta U}_{\vec{c}}$, and $\mathbf{C}_{\vec{b}}$) to support high-frequency surface details with normal maps. We do so by projecting basis-space radiance into spherical harmonics (SH) and mapping the SH coefficients into MRT's hemispherical basis.

Moreover, we compute a new operator $\mathbf{U}_{dyn}$ to map $\mathbf{b}$ **and** $\mathbf{c}$ vectors to volumetric SH radiance probes. We do so by combining the clutter and scene gathers in Section 5.3 and outputting into a volume texture. These dynamic light probes are applied to animated objects (e.g., characters) in the scene, capturing shadows and interreflection from the clutter and scene at minimal performance cost (Figure 5.4).

MRT uses volume probes to shade clutter, but this only captures a subset of the transport (light bounced from the scene onto clutter) and introduces error (Figure 5.1a). Our *clutter gather operator* computes this transport path more accurately (Figure 5.1b) and with higher performance since radiance is computed directly in the lightmap as opposed to using an SH

---

[3]Consider Figure 5.2 (left) if the orange sampling point were inside the grey region.
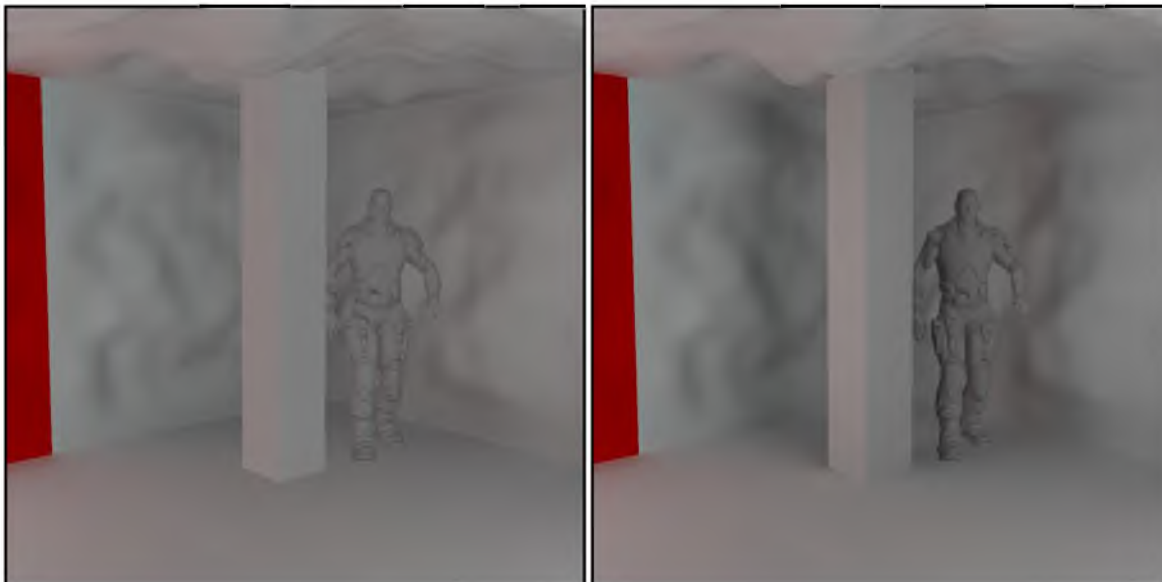
Figure 5.4: MRT (1.5 ms) vs. DRT (2.1 ms): note that the dynamic soldier object is also shadowed/lit by the clutter/scene, using volume probes. Timings exclude direct lighting, which took 8.3 ms.

volume probe. For example, Figure 5.1a uses MRT volume probes and Figures 5.1b–d do not: DRT *outperforms* MRT in this case as volume texture sampling and SH shading are avoided.

5.4.4 Settings

Our results use the following settings: 1024 rays per texel to construct our operators, and 16×16 lightmaps on each shape or clutter face. Our runtime data requirements are similar to MRT: delta occlusion operators (Section 5.3.1), and delta reflection operators (Section 5.3.2) use 32×16^2 values per face (32 modes for 256 texels). Clutter gather operators (Section 5.3.3) use fewer modes, requiring 5×16^2 values per clutter face. We illustrate **DRT** on simple clutter geometry (five-faced pillars), but we can handle arbitrary clutter objects as the position and normal samples required for the ray tracing are generated by rasterizing clutter into UV atlases.

Even in a complex scene with four pillars (partially visualized in Figure 5.4), all of our new operators require only 0.52MB total additional storage and the application of our new operators has a negligible cost on the overall shading, especially when direct lighting computation (using shadow mapping) is included. Updating the operators requires roughly 5 seconds on a 12-core Intel Xeon X5670. We also naïvely ray trace the same scene shapes again to account for distant light transport, leading to redundant ray tracing. Optimized

ray tracing is left to future work as the additional latency is negligible for, e.g., level design use cases.

5.5 Discussion

5.5.1 Advantages

With DRT, we simulate important light transport paths ignored in MRT, allowing clutter geometry to contribute to indirect illumination in the scene. This increases the accuracy of the approach, but maintains the high-performance behavior of the overall algorithm. We avoid brute-force computation of full light transport by exploiting low-dimensional basis-space lighting parameterized over the scene and clutter, which we use to efficiently construct our low-rank operators. Once clutter geometry is placed or moved in the scene, DRT can compute the additional transport operators in just a few seconds (using unoptimized CPU ray tracing), allowing for fast scene design and shading response. At any time, the direct lighting can be changed with the indirect illumination updated immediately. As with MRT, DRT can easily scale to low-end graphics platforms such as the iPad and iPhone.

5.5.2 Disadvantages

DRT introduces scene-dependent computation and data to the original MRT framework, which can be viewed as a disadvantage; however, this added flexibility allows clutter geometry to be added to a scene at runtime with light transport updated in only a few seconds. As discussed in [75], adding more items to the library of precomputed shapes (e.g., to model clutter geometry) would increase the entropy of the various transport operators, precluding accurate low-rank approximations.

Given that we use the implicit lighting environment generated *without* the clutter present in the scene, indirect light reflected onto the clutter from the scene can suffer from artifacts. The closest box in Figure 5.1 exhibits this subtle artifact with incorrect shadow colors. Given the scene-independent precomputed lightmap bases from MRT, DRT's approach still computes the most suitable approximation to these basis-space operators. A fundamentally different approach would be necessary to completely eliminate these artifacts; however, it is unclear if such an approach would still fit into the scene-independent framework of MRT.

CHAPTER 6

VOLUMETRIC OBSCURANCE[1]

6.1 Introduction

Generating realistic images is difficult, doing so in real time even more so. There are methods that attempt to simulate a more complex model for ambient light such as Obscurance [15] and Ambient Occlusion (AO) [16]. These techniques give a softer and more realistic look while providing important contact cues. While it is straightforward to implement these techniques for off-line rendering or static objects and scenes, doing so for dynamic objects has proven to be difficult. Recent games [28, 29] use screen-space techniques, but they suffer from performance and undersampling problems. This chapter builds on those techniques, presenting a method that uses line samples that suffer less from undersampling than point samples along with an area sampling technique that can generate plausible results without undersampling issues (see Figure 6.1).

Volumetric Obscurance (VO) at a point P is defined as the integral of an occupancy

Figure 6.1: Demonstration of the similarities between Screen Space Ambient Occlusion and Volumetric Obscurance. The left image uses 33 point samples and runs at 156 fps. The middle image uses nine line samples and runs at 210 fps. The rightmost image uses a single line sample and runs at 240 fps.

---

[1]This chapter was originally published by B. Loos and P.-P. Sloan as *Volumetric Obscurance* in ACM Symp. on Interactive 3D Graph. and Games, Washington, DC., 2010 [Online] Available: http://doi.acm.org/10.1145/1730804.1730829 [79]

function around P times a compact kernel. The occupancy function is zero for points inside an object and one otherwise. While this does not correspond to any physical process, or special case of global illumination like AO, it generates related imagery. Previous work is effectively computing this integral by point sampling the volume [28, 29]. VO can be computed more efficiently by using line samples or by querying a simple statistical model of the scene's depth buffer. This effectively samples areas of the screen and integrates them against a volumetric piece of the integral instead of using point samples to estimate the integral.

6.2 Obscurance and Ambient Occlusion

Obscurance is defined as

$$A(P) = \frac{1}{\pi} \int_\Omega \rho(d(P,\omega)) cos\theta d\omega$$

where Ω is the hemisphere, ρ is a fall-off function, d is the distance to the first intersection, θ is the angle between the normal at P and the direction ω. The fall-off function should start at zero and become one at some fixed distance, which enables rays to be traced with a limited extent. Ambient Occlusion is a special case of obscurance where the fall-off function is zero for any value besides ∞ (See Figure 6.2). Both these techniques model ambient illumination. AO is the transfer coefficient that maps direct lighting to outgoing radiance for a diffuse surface [18]. Sometimes the surface normal is not included, which makes AO
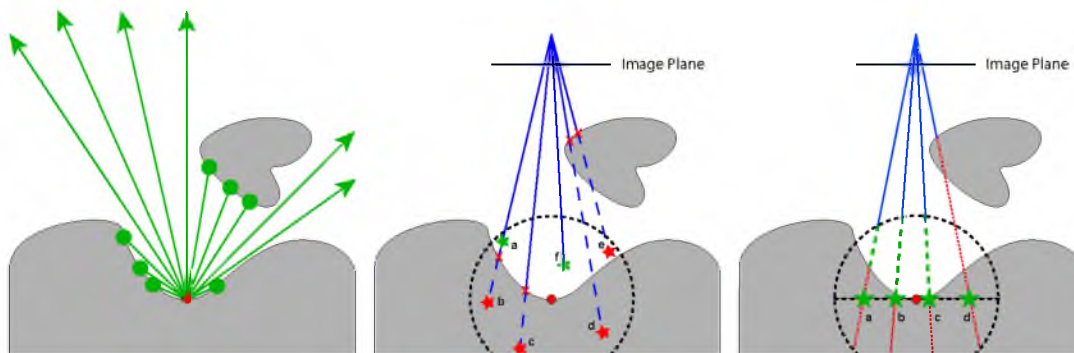
Figure 6.2: An illustration of the three ambient shadowing models. Ambient Occlusion (on the left) is defined as the ratio of rays emanating from a point on the surface that are able to escape the scene. Volumetric Obscurance, originally computed using point samples (in the middle), attempts to replicate this by generating a number of samples around a point on the surface and using a ratio of points that fall behind the depth buffer. We compute this volume integral using line samples (on the right), the visible portions of each line (shown in green) are used to compute the integral.

simply the discrete cosine (DC) projection of the visibility function, which can be used with a triple product to light the surface.[2]

The primary difficulty in mapping these techniques to the GPU is that the queries are over ray directions, which does not interact well with the traditional rasterization framework. These techniques also have the property that they are not separable, which makes combining the AO from multiple objects more difficult.

6.3   Volumetric Obscurance

Volumetric Obscurance (VO) is a related quantity defined as

$$V(P) = \int_X \rho(d(P,x))O(x)dx$$

where X is a 3D neighborhood around P, and O is an occupancy function, zero if there is matter at x and one otherwise. The fall-off function ρ will be defined to be one at P and possibly falling off to zero at a fixed distance. We have experimented with a constant function and a quadratic function that falls off to zero at a finite distance, but the differences are fairly subtle and did not seem to warrant the extra cost. For the remainder of this chapter we assume ρ is a constant function.

VO is a volumetric generalization of obscurance and will have the same results when any ray originating from P intersects a single solid surface. The attenuation function used in obscurance can be thought of as the integral of the function ρ used here from t to the extent of the kernel. There are two primary benefits to this formulation: the contribution of solid noninterpenetrating objects is separable, which makes it easier to combine dynamic parts of the scene with static parts that could precompute VO, and computing this integral is more amenable to GPUs. Both the Crytek [28] and Blizzard [29] techniques effectively use this formulation.

6.3.1   Line Sampling

Instead of using point sampling to numerically compute the integral [28], we use analytic computations in depth and numeric in the other two dimensions. Given a sphere of constant size in object space centered at P we compute the analytic integral of the occupancy function times the depth extent of the sphere at each sample point on a disk.

Given the depth d at a 2D point the occupancy function $f(z)$ is

---

[2]The triple product is extremely simple if one of the terms is constant since that term can be factored out of the integral.

$$f(z) = \begin{cases} 1 & : z \le d \\ 0 & : z > d \end{cases}$$

$f(z)$ is integrated against the function ρ, which is defined to be constant over the unit sphere centered at point P. Because ρ is constant, this integral is simply

$$\int_{-z_s}^{z_s} f(z)dz = max(min(z_s, d_r) + z_s, 0)$$

where $z_s = \sqrt{1 - x^2 - y^2}$, d_r is the depth of pixel (x, y) mapped into the unit sphere's coordinate system, and x and y are carefully sampled points on a unit disk. This is visually depicted in Figure 6.3, where the integral of the step-function against the z interval is shown. The samples on the disk allow an estimation of the full volumetric obscurance integral. Line sampling is more efficient due to the fact that all 2D samples will project to distinct points in the z-buffer, whereas two point samples may project to the same location (see the two points a and b along the same line in middle image of Figure 6.2).

A differential change of the camera will cause a differential change of the VO as long as the depth function is continuous in the neighborhood of the sample. This is not true when using point sampling techniques.[3] This is particularly evident even when blurring the results using a small number of point samples. While line samples perform better than point samples, unless the radius is fairly small it still needs an edge aware blur pass.

6.3.2 Area Sampling

Instead of point or line sampling the depth buffer, a statistical representation over the area associated with each sample can be built. This method generates adequate images using a single sample if darkening creases and corners is all that is desired. Unfortunately, our area sampling method is not competitive performance wise when using multiple samples.
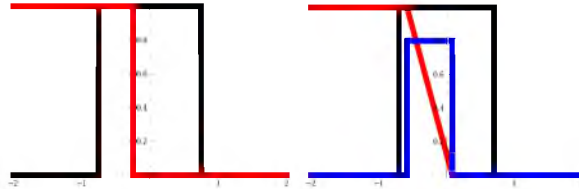
Figure 6.3: Depth from P is along the horizontal axis. Sphere extents (z_s) are in black and occupancy function is in red. For line sampling (left) the occupancy function is a step function (see Section 6.3.1) but falls off smoothly for area sampling (right). The statistical depth model is shown in blue.

---

[3]Filion [29] uses a fall off function that is only used when a sample is occluded; this is closer in spirit to our thickness model.

The simplest statistical model is the mean and variance of the depth values over a given area. As in Variance Shadow Maps [36] this can be computed from the first two moments of the depth buffer. The average depth, $M_1 = E[z]$, and average squared depth, $M_2 = E[z^2]$, can be filtered because the expectation operator E is linear. The mean μ is simply M_1, and the variance σ^2 is $E[z^2] - E[z]^2$, where σ is the standard deviation.

If a large radius or small number of area samples are going to be chosen, reconstruction artifacts from trilinear interpolation can occur. This can be alleviated by computing a Gaussian blur on the top level of the mip-map and carefully placing the taps when downsampling to compensate.

The depth buffer is sampled at each pixel to determine the screen space extent, but all quantities are mapped to a unit sphere centered at the origin before the integral is computed. When computing with multiple samples only a single log_2 has to be computed; the logs for the fractional area covered by a sample are precomputed and added to the log for the entire sphere to determine the level of detail (LOD) each time the depth model needs to be sampled.

This mip-map is sampled based on the area of a given sample to generate a distribution of depths over that area. Given this distribution, we compute a simple integral using the x, y coordinates of the disk at the center of the area sample. In Figure 6.3 the depth distribution is modeled as a box function,[4] so the visibility function now has two regions that can contribute to the integral, the leading constant part and linear part. Below are the equations for computing the integral with a constant attenuation function. z_{min} and z_{max} are the entry and exit points of the sphere at a given point, z_0 and z_1 are the extents of the linear part of the visibility function intersected with the sphere extents, and $a = \frac{-1}{2\sigma}$ and $b = -a(\mu + \sigma)$ are the coefficients of the line from the integrated visibility function

$$V_c(z_{min}, z_{max}) = z_{max} - z_{min}$$

$$V_c(z_0, z_1, a, b) = a(z_1^2 - z_0^2)/2 + b(z_1 - z_0)$$

6.3.3 Thickness Model

One of the limitations of all screen space algorithms is that there is no information about what happens behind the depth buffer. One simple approximation is to assume that all surfaces have a fixed depth. This can be done by replacing the step function in the convolution above with some model of occlusion behind the front most surface. Figure 6.4

---

[4]The convolution of the distribution with the visibility function results in the complimentary error function for a Gaussian, which has no closed form solution. We approximate the Gaussian with a box function with a half-width of one standard deviation

Figure 6.4: On the left thickness is not used (177 fps), on the right it is (163 fps). Notice how the silhouette edges are no longer shadowed when the thickness model is used.

shows a comparison with and without the thickness model. To compute VO using the thickness model all that is required is to compute the integral twice, once with the usual visibility function and again with $1 -$ visibility function shifted back by the fixed thickness. The results are simply summed.

6.3.4 Incorporating the Surface Normal

If a surface normal is available it can be used to restrict the sampling to a hemisphere instead of a sphere. For point samples [29] this has been done by reflecting the points that are underneath the hemisphere defined by the surface normal. Line and area sampling can be modified to compute the depth of the plane defined by the surface normal[5] at the x, y coordinates for the given sample; simply clamp the evaluation of the integral with this depth. Figure 6.5 shows a comparison between using and not using the normal. Using the normal allows the capture of finer scale details, even with a moderately large radius. An intriguing alternative was concurrently proposed in [31], where the sphere sampled is the largest sphere contained by the hemisphere centered at the point. This also incorporates the cosine weighted fall-off commonly used in AO techniques.

---

[5]Since all the computations are done in the space of the unit sphere, the plane equation is just the eye-space surface normal

Figure 6.5: Comparison between using and not using the normal in our volumetric obscurance method. Left: using the normal. Right: no normal is used.

6.3.5 Sample Generation

Considerable time and effort was spent on generating good sample patterns for line, area, and point sampling. Initially, we tried using Lloyd relaxation techniques [80], Quasi Monte Carlo methods, and generating Poisson distributions using dart throwing. The technique outlined below performed significantly better, particularly at low sample counts.

Solving an electrostatics problem generates good results over the surface of the sphere [81], we extend this technique to samples inside a sphere and a disk. Since we are computing integrals, the goal is to have the integral over the Voronoi region of every sample to be equal. While the electrostatics solution works well on a surface, in a disk or sphere the particles need to be coerced to stay inside the boundary. Simply adding a charge to the boundary does not work. Gauss's law implies that the net force from that charge will always be zero. Instead we use a $\frac{1}{d^4}$ repulsion force and optimize the boundary charge to push points toward the weighted centroid of their Voronoi regions periodically. This process is initialized with tens of starting point sets generated using Poisson sampling. The net force is reduced until the error decreases (not applying the force until there is a reduction) and increased otherwise. The Voronoi regions are computed using a discrete Voronoi diagram with 128^3 cells for the volume and 512^2 for the disk. Each disk sample has a weight proportional to the Z extent on the sphere.[6]

---

[6] The disk is an orthographic projection of a sphere, the integral being computed. Each sample will scale its line integral by the integral of all the discrete samples in its Voronoi region.

As in [28] we randomize our sample sets at every pixel using a random texture. Experiments with various sizes of random textures seem to imply that the algorithm is not sensitive to the size of the texture nor does random texture size seem to have an impact on performance. For line samples we compute a reflection about a random direction in 2D instead of 3D; otherwise randomization of point samples and line samples is treated in the same manner.

It is interesting to note that certain generated sample sets produce much less noise than those around them (see Figure 6.6). In 3D 6, 12, and 33 samples have extremely low error compared to sets generated with other numbers of samples. In 2D 5, 7, 9, 13, and 16 samples have low error when the center is one of the samples, whereas 12, 13, and 15 samples seem to work optimally if the center is not included. We imagine that these sample sets work better due to symmetries. For example, 6 and 12 samples in 3D closely match the vertices of platonic solids.

6.4 Results

6.4.1 Performance

In Table 6.1 we compare the performance of Crytek's point sampling method, our line sampling method, and the Horizon Split Ambient Occlusion (HSAO) technique [30]. These tests were run on a desktop with an NVIDIA QuadroFX 4800 GPU at a resolution of 1024x1024 using full resolution AO buffers.

The difference in visual quality between Crytek's point sampling using 12 samples and line sampling using 5 samples is negligible (see Figure 6.7). In animation tests, small numbers of samples caused the Crytek method to exhibit temporal aliasing, whereas even with as few as 5 line samples the temporal aliasing was much less pronounced. Table 6.1 shows similar visual quality much quicker using line samples with dual-radii results shown in Figure 6.8.

Figure 6.6: Some sample selections turn out to be much better than others. On the left, you can see that nine line samples create much less noise than ten line samples. On the right you can see 12 point samples create much less noise than 13 point samples.

Table 6.1: Timing Breakdown per pass in milliseconds. The bilateral blur used for each scene required a median time of 2.23 ms. *Using double radii (see Figure 6.8), not included in the average

| | Geometry | Ambient Occlusion | | | | |
| | | Lines | | Points | | HSAO |
		5	9	12	33	
Dragon	3.39	0.71	1.01	1.18	3.11	51.52
Dragon*	3.39	1.06	1.85	2.44	7.36	N/A
Cornell	0.45	0.71	1.03	1.31	3.55	60.55
Soldiers	2.37	0.54	0.77	1.14	2.32	17.04
Knight	0.27	0.59	0.82	0.97	2.52	29.13
Average		0.64	0.91	1.15	2.87	39.56
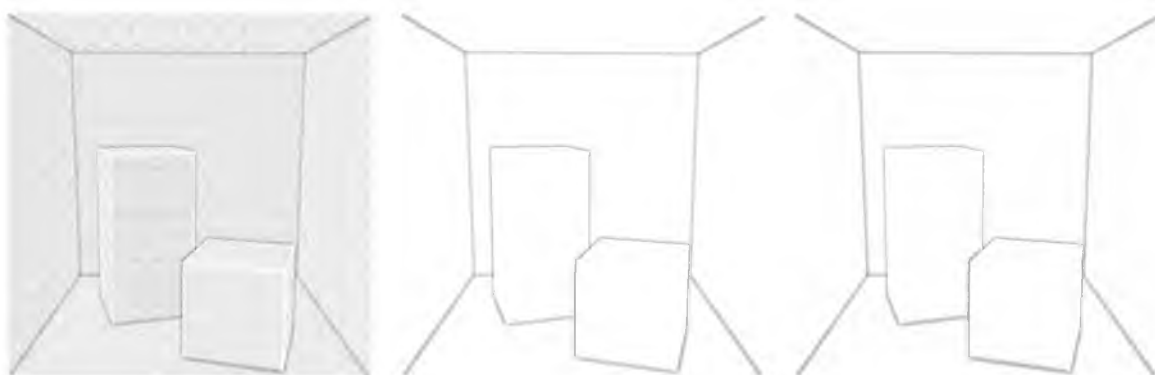
Figure 6.7: Line samples produce better quality results in less time. The left image shows four point samples (0.84 ms), the middle image shows five line samples (0.87 ms), and the right image shows twelve point samples (1.77 ms).

Figure 6.8: When small radii are used, fine features can be captured, but contact shadows and other large scale features are lost. By using dual radii both sets of features can be captured for more pleasing results. The left image uses a small radius (127 fps), the middle image uses a large radius (112 fps), and the right image uses both (95 fps)

CHAPTER 7

FUTURE WORK

There are a few main directions that we could take this research in the future. One, we could create a set of modeling tools to put this technique in the hands of artists. Two, we could study the availability of creating new tools to allow more direction of the bases created for the dictionary pieces. Third, we could study the integration of MRT and other AO methods used in game production (such as [82]). Lastly, we could do additional mathematical work to create a method that would allow us to recreate more high-frequency signals along with the low-frequency lighting.

7.1   Modeling Tools

We implemented a prototype of this technology into the pipeline at Black Rock Studios. We added the ability to add MRT geometry to an existing scene using their Tomcat tool, which allowed a real-time visualization of the indirect lighting using our dictionary pieces. Our experience there and at other video game studios have shown us that artists already generate multiple sets of geometry for things such as physics, AI, and other subsystems. Tools to create the required block maps for MRT should be easy to create.

The portions of this work that would generate new research are automated methods to map block maps to already existing world geometry. Due to the limited options with our simple block dictionary this automation might actually be possible. However, if more complicated shapes were used, it might be possible to integrate the physics and AI geometry as well, unifying the systems and making it more likely to use the MRT method in actual game play.

7.2   Directability

Another avenue for future work is the ability to artistically modify the basis functions to achieve a desired look. Most lighting tools used for entertainment require changes that look good, which may not be physically plausible. Currently, our technique only attempts to

reproduce the lighting as defined by the geometry. However, this is not a requirement. As is shown with our basis enrichment (see Appendix A) other signals can be incorporated into the bases. It would be an interesting series of future experiments to find tools that would allow artists to modify the bases to generate a specific look, or feature at a high level. This would most likely require good artist tools first (see Section 7.1).

7.3   Real-time AO and Global Illumination

Some current methods for generating large scale, low frequency, world space ambient occlusion for video games store data per vertex, sometimes requiring large sets of data. If there were a way to integrate AO with the DRT calculations, we might be able to generate pleasing results and save some per-vertex data as is done in [82] and [83]

7.4   High Frequency Indirect Lighting

Our current MRT and DRT methods are for generating fairly low-frequency data that would not be appropriate for specular lighting (a high-frequency signal not covered by our new SSAO technique). Lighting models in video games are getting much of their inspiration from physically based systems. Beginning with Burley [84] many other games (such as Assassin's Creed Unity, The Order: 1886, and others) are starting to use physically based shading, of which specularity is a very important part. Other methods, such as Crassin et al. [55], allow high frequency indirect lighting. Either modifications to our MRT algorithm or an additional algorithm to calculate high frequency indirect lighting would make this solution more tenable in future video games.

CHAPTER 8

CONCLUSION

In this dissertation we have introduced a novel method of reproducing an approximation to low-frequency global illumination in real time. We believe this work has shown that global illumination can be decomposed, calculated on simple shapes, and then reconstructed. Our method has been split into three parts to transmit indirect illumination (Chapter 3) and block indirect illumination (Chapter 5), as well as improve the signal with higher frequency visibility calculations (Chapter 6). These three methods calculate a new set of basis functions that allow us to map a simplified, approximate, indirect lighting solution onto a dense geometric level in real time.

Modular Radiance Transfer creates a small, level-independent set of preprocessed data that is used to generate indirect lighting, which is then remapped onto actual geometry as a light map. While the result is only an approximation to the real indirect lighting that would occur on the actual geometry, the lighting it generates is a plausible, real time signal, which could be used in video games in the near future.

Delta Radiance Transfer is a method that uses a similar set of bases defined to block indirect lighting in the spirit of Antiradiance [63]. This method integrates well with Modular Radiance Transfer by using the same set of basis coefficients, allowing us to tightly integrate the two methods to generate even more plausible, but still approximate, real-time indirect lighting.

To aid in ameliorating the visual signal, we also present Volumetric Obscurance. This method allows us to add additional higher frequency visibility calculations to calculate screen space ambient occlusion (SSAO). Our method uses fewer samples but gives us smoother results than traditional point sampled volumes. These results, when added to our low-frequency MRT and DRT signals, gives a pleasing, real-time indirect illumination signal that can aid in the realism of video game scenes, where a more robust physics simulation requires too much time and memory.

Our methods of generating a better basis function for complex phenomena has also

been applied in areas other than rendering. They have been used to simplify fluid dynamic calculations [68] as well as attempting to improve sound propagation [67].

Real-time Global Illumination is the next frontier in real-time gaming. We are starting to see commercial games engines (Unreal Engine 4, Fox Engine, Snowdrop Engine, RAD Engine 4.0, AnvilNext) move towards physically based rendering, following in the footsteps of the film industry. Even in current engines with deferred rendering, we see large overdraw counts as artists attempt to simulate indirect illumination bounces using many lights. In film, lighting rigs began to get so complex that when a new artist was assigned to a given shot, all previous lighting work was discarded due to the complexity of the lighting setup. While games have not yet reached that level of lighting complexity, other performance metrics may soon dictate a new way of lighting, a method that integrates a more holistic approach to indirect lighting.

APPENDIX A

BASIS ENRICHMENT

When constructing the direct light ($\mathbf{P}$) or interface ($\mathbf{H}_{\mathrm{rlf}}$) priors, some functions are not well represented in the reduced space: e.g., direct light with shadows cast by objects not present during the computation of $\{\mathbf{l}_{\mathrm{d}0}, \ldots, \mathbf{l}_{\mathrm{d}m}\}$ indirect light $\mathbf{U}_{\mathbf{b}}$ vectors for multiple bounces, and blocks of almost a single color (see Figure A.1). To model targeted effects, we can *enrich* our basis.

Given bases stored in columns of matrices, we first rescale the matrices to have a prescribed Frobenius norm, then concatenate the matrices into a larger matrix with the same number of rows and compute the SVD, retaining the left singular vectors and singular values (optionally). This new prior better spans all subspaces.

To generate a new prior for blocks, we equally weight the scaled prior $\mathbf{P}\,\mathbf{S}$ from Section 3.3, fall back basis functions that are constant on one face and zero on others to approximate changes from shadows/albedo and indirect lighting represented by $\mathbf{U}_{\mathbf{b}}$.

For interfaces, lightfield-to-lightfield operators $\{\mathbf{R}_{\nwarrow}, \mathbf{R}_{\uparrow}, \mathbf{R}_{\nearrow}\}$ can generate output outside of $\mathbf{H}_{\mathrm{rlf}}$'s space. In this case, we compute the reduced basis response through these operators, yielding a temporary basis $\mathbf{L}_{\mathrm{temp}}$ and balance with $\mathbf{H}_{\mathrm{rlf}}$ generating a new basis $\bar{\mathbf{H}}_{\mathrm{rlf}}$. The new basis better captures functions from both spaces; repeated enrichment is used to handle multiple propagation steps (or bounces.)

Figure A.1: A box has five red and one white face. Without enrichment, red faces bounce all colors onto the white face, which is incorrect.

APPENDIX B

QUADRATIC SH TO HEMISPHERICAL
LINEAR SH

We present an alternative to vector irradiance: a closed-form solution for the optimal hemispherical projection of *quadratic* irradiance back into a linear (vector) model. This effectively models the higher frequency energy that "bleeds" into the linear band after clamping reflectance response to the upper hemisphere. We solve for linear-SH coefficients that minimize the difference, over the hemisphere, between quadratic (L_2) and linear (L_1) expansions of radiance,

$$
\begin{aligned}
E &= \int_{\mathcal{H}} \left[L_2(\omega) - L_1(\omega) \right]^2 d\omega \\
&= \int_{\mathcal{H}} \left[\sum_{j=1}^{9} b_j \, y_j(\omega) - \sum_{i=1}^{4} a_i \, y_i(\omega) \right]^2 d\omega
\end{aligned}
\tag{B.1}
$$

where $\mathcal{H}$ is the hemispherical domain, a_i (unknowns) and b_j (knowns) are order-2 and order-3 SH coefficients, and we single index the SH basis functions, $y_k(\omega)$. Setting the derivative of error with respect to the unknowns to zero and solving yields

$$
\begin{aligned}
\frac{dE}{da_k} &= 2 \int_{H^2} \left[L_2(\omega) - L_1(\omega) \right] \frac{dL_1(\omega)}{da_k} \, d\omega = 0 \\
0 &= 2 \sum_{j=1}^{9} b_j \int_{\mathcal{H}} y_j(\omega) y_k(\omega) d\omega - 2 \sum_{i=1}^{4} a_i \int_{\mathcal{H}} y_i(\omega) y_k(\omega) d\omega
\end{aligned}
\tag{B.2}
$$

which we express in matrix form: $\mathbf{H_1} \, \mathbf{a} = \mathbf{H_2} \, \mathbf{b}$, where the 4×4 matrix $[\mathbf{H_1}]_{ik} = \int_{\mathcal{H}} y_i(\omega) y_k(\omega) d\omega$ and the 4×9 matrix $[\mathbf{H_2}]_{jk} = \int_{\mathcal{H}} y_j(\omega) y_k(\omega) d\omega$. The top 4×4 block of $\mathbf{H_2}$ is $\mathbf{H_1}$, and an analytic expression for $\mathbf{a}$ is

$$
\mathbf{a} = \mathbf{H_1}^{-1} \mathbf{H_2} \mathbf{b} = \{ b_1 - c_1 b_7, b_2 + c_2 b_6, b_3 + c_3 b_7, b_4 + c_2 b_8 \}
\tag{B.3}
$$

where $c_1 = 3\sqrt{5}/4, c_2 = 3\sqrt{5}/8, c_3 = \sqrt{15}/2$. The H4 basis [85] and OHLSH span the same space.

REFERENCES

[1] H. W. Jensen, *Realistic Image Synthesis Using Photon Mapping*. Natick, MA: A. K. Peters, Ltd., 2001.

[2] F. E. Nicodemus *et al.*, *Geometrical Considerations and Nomenclature for Reflectance*. Washington, DC: National Bureau of Standards, 1977.

[3] A. S. Glassner, *Principles of Digital Image Synthesis*. San Francisco, CA: Morgan Kaufmann Publishers Inc., 1994.

[4] J. T. Kajiya, "The rendering equation," in *Proc. 13th Ann. Conf. Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '86. New York, NY: ACM, 1986, pp. 143–150. [Online]. Available: http://doi.acm.org/10.1145/15922.15902

[5] G. W. Romney, "Computer assisted assembly and rendering of solids," Ph.D. dissertation, The Uni. Utah, Salt Lake City, UT, 1969, aAI7003776.

[6] J. E. Warnock, "A hidden surface algorithm for computer generated halftone pictures," Ph.D. dissertation, Univ. Utah, Salt Lake City, UT, 1969, aAI6919002.

[7] M. E. Newell, R. G. Newell, and T. L. Sancha, "A solution to the hidden surface problem," in *Proc. of the ACM Ann. Conf. - Volume 1*, ser. ACM '72. New York, NY: ACM, 1972, pp. 443–450. [Online]. Available: http://doi.acm.org/10.1145/800193.569954

[8] T. Whitted, "An improved illumination model for shaded display," in *Proc. 6th ann. conf. Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '79. New York, NY: ACM, 1979. [Online]. Available: http://doi.acm.org/10.1145/800249.807419

[9] R. L. Cook, T. Porter, and L. Carpenter, "Distributed ray tracing," in *Proc. 11th Ann. Conf. Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '84. New York, NY: ACM, 1984, pp. 137–145. [Online]. Available: http://doi.acm.org/10.1145/800031.808590

[10] E. Lafortune, "Mathematical models and Monte Carlo algorithms for physically based rendering," Katholieke Universiteit Leuven, Leuven, Belgium, Tech. Rep., 1996.

[11] E. Veach and L. J. Guibas, "Metropolis light transport," in *Proc. 24th Ann. Conf. Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '97. New York, NY: ACM Press/Addison-Wesley Publishing Co., 1997, pp. 65–76. [Online]. Available: http://dx.doi.org/10.1145/258734.258775

[12] C. M. Goral, K. E. Torrance, D. P. Greenberg, and B. Battaile, "Modeling the interaction of light between diffuse surfaces," in *Proc. 11th Ann. Conf. Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '84. New York, NY: ACM, 1984, pp. 213–222. [Online]. Available: http://doi.acm.org/10.1145/800031.808601

[13] H. E. Rushmeier and K. E. Torrance, "Extending the radiosity method to include specularly reflecting and translucent materials," *ACM Trans. Graphics*, vol. 9, pp. 1–27, 1990.

[14] I. Georgiev, J. Křivánek, T. Davidovič, and P. Slusallek, "Light transport simulation with vertex connection and merging," *ACM Trans. Graph.*, vol. 31, no. 6, pp. 192:1–192:10, Nov. 2012. [Online]. Available: http://doi.acm.org/10.1145/2366145.2366211

[15] S. Zhukov, A. Inoes, and G. Kronin, "An ambient light illumination model," in *Rendering Techniques*. Vienna, Austria: Springer-Verlag, 1998.

[16] H. Landis, "Global illumination in production," ACM SIGGRAPH 2002 Course #16 Notes, July.

[17] G. Miller, "Efficient algorithms for local and global accessibility shading," in *Proc. SIGGRAPH 1994*, ser. Computer Graphics Proceedings, Ann. Conference Series, A. Glassner, Ed., ACM SIGGRAPH. ACM Press, Jul. 1994, pp. 319–326.

[18] P.-P. Sloan, J. Kautz, and J. Snyder, "Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments," *ACM Trans. Graph.*, vol. 21, no. 3, 2002.

[19] J. Lehtinen, "A framework for precomputed and captured light transport," *ACM Trans. Graph.*, vol. 26, no. 4, 2007.

[20] Z. Ren, R. Wang, J. Snyder, K. Zhou, X. Liu, B. Sun, P.-P. Sloan, H. Bao, Q. Peng, and B. Guo, "Real-time soft shadows in dynamic scenes using spherical harmonic exponentiation," *ACM Trans. Graph.*, vol. 25, no. 3, pp. 977–986, Jul. 2006.

[21] P.-P. Sloan, N. K. Govindaraju, D. Nowrouzezahrai, and J. Snyder, "Image-based proxy accumulation for real-time soft global illumination," in *Proc. Pac. Conf. Comput. Graph. Appl.* Washington, DC, USA: IEEE Computer Society, 2007, pp. 97–105. [Online]. Available: http://dx.doi.org/10.1109/PG.2007.35

[22] M. Bunnell, "Dynamic ambient occlusion and indirect lighting," in *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation*, M. Pharr, Ed. Boston, MA: Addison-Wesley Professional, 2005.

[23] J. Kontkanen and S. Laine, "Ambient occlusion fields," in *ACM Symp. Interactive 3D Graphics and Games*. New York, NY: ACM, 2005, pp. 41–48.

[24] P. Shanmugam and O. Arikan, "Hardware accelerated ambient occlusion techniques on gpus," in *ACM Symp. Interactive 3D Graphics and Games*. New York, NY: ACM, 2007, pp. 73–80.

[25] M. McGuire, "Ambient occlusion volumes," Williams College, Williamstown, MA, Tech. Rep., 2009.

[26] J. Kontkanen and T. Aila, "Ambient occlusion for animated characters," in *Rendering Techniques 2006 (Eurographics Symp. Rendering)*, T. A.-M. Wolfgang Heidrich, Ed. Eurographics, Jun. 2006. [Online]. Available: http://www.tml.hut.fi/ janne/aoc

[27] A. G. Kirk and O. Arikan, "Real-time ambient occlusion for dynamic character skins," in *ACM Symp. Interactive 3D Graphics and Games*, Seattle, WA, April 2007.

[28] M. Mittring, "Finding next gen: Cryengine 2," in *SIGGRAPH Courses*, San Diego, CA, 2007, pp. 97–121.

[29] D. Filion and R. McNaughton, "Effects & techniques," in *SIGGRAPH Classes*, Los Angeles, CA, 2008, pp. 133–164.

[30] L. Bavoil *et al.*, "Image-space horizon-based ambient occlusion," in *SIGGRAPH Talks*, Los Angeles, CA, 2008.

[31] L. Szirmay-Kalos *et al.*, "Volumetric ambient occlusion," *IEEE Comput. Graph. Appl.*, vol. 30, no. 1, pp. 70–79, Feb. 2010.

[32] N. Smedberg and D. Wright, "Rendering techniques in Gears of War 2," in *Game Developers Conf.*, San Francisco, CA, 2009.

[33] D. Nehab *et al.*, "Accelerating real-time shading with reverse reprojection caching," in *Graph. Hardware*, San Diego, CA, 2007.

[34] T. Luft *et al.*, "Image enhancement by unsharp masking the depth buffer," in *ACM SIGGRAPH Papers*, Boston, MA, 2006, pp. 1206–1213.

[35] B. Gooch *et al.*, "Interactive technical illustration," in *ACM Symp. Interactive 3D Graph.*, Atlanta, GA, 1999, pp. 31–38.

[36] W. Donnelly and A. Lauritzen, "Variance shadow maps," in *Proc. Symp. Interactive 3D Graph. and Games*, Redwood City, CA, 2006, pp. 161–165.

[37] A. Lauritzen and M. McCool, "Layered variance shadow maps," in *Proc. Graph. Interface*, Toronto, ON, 2008, pp. 139–146.

[38] A. Pesce, *Variance methods for Screen-Space Ambient Occlusion in ShaderX7*, 2009, ch. 6.7.

[39] T. Ritschel *et al.*, "Approximating dynamic global illumination in image space," in *Proc. ACM Symp. Interactive 3D Graph. and Games*, Boston, MA, 2009, pp. 75–82.

[40] S. Parker *et al.*, "Interactive ray tracing," in *ACM Symp. Interactive 3D Graph.*, Atlanta, GA, 1999, pp. 119–126.

[41] I. Wald *et al.*, "Interactive rendering with coherent ray tracing," in *Comput. Graph. Forum Proc. EUROGRAPHICS*, vol. 20, no. 3, 2001, pp. 153–164.

[42] T. J. Purcell *et al.*, "Ray tracing on programmable graphics hardware," *ACM Trans. Graph.*, vol. 21, no. 3, pp. 703–712, July 2002.

[43] R. Wang *et al.*, "An efficient gpu-based approach for interactive global illumination," *ACM Trans. Graph.*, vol. 28, no. 3, 2009.

[44] D. Kopta *et al.*, "Fast, effective bvh updates for animated scenes," in *Proc. ACM SIGGRAPH Symp. Interactive 3D Graph. and Games*, Costa Mesa, CA, 2012, pp. 197–204.

[45] A. Keller, "Instant radiosity," in *Proc. Conf. Comput. Graph. and Interactive Techs.*, Los Angeles, CA, 1997, pp. 49–56.

[46] C. Dachsbacher and M. Stamminger, "Reflective shadow maps," in *Proc. Symp. Interactive Graph. and Games*, 2005, pp. 203–231.

[47] C. Dachsbacher and M. Staminger, "Splatting indirect illumination," in *Proc. Symp. Interactive 3D Graph. and Games*, Redwood City, California, 2006, pp. 93–100.

[48] T. Ritschel *et al.*, "Imperfect shadow maps for efficient computation of indirect illumination," *ACM Trans. Graph.*, vol. 27, no. 5, pp. 129:1–129:8, Dec. 2008.

[49] P. Guerrero *et al.*, "Real-time indirect illumination and soft shadows in dynamic scenes using spherical lights," in *Comput. Graph. Forum*, vol. 27, no. 8, 2008, pp. 2154–2168.

[50] R. Ramamoorthi, "Precomputation-based rendering," *Foundations Trends Comput. Graph. Vision*, vol. 3, no. 4, 2009.

[51] A. W. Kristensen *et al.*, "Precomputed local radiance transfer for real-time lighting design," *ACM Trans. Graph.*, vol. 24, no. 3, 2005.

[52] A. Kaplanyan and C. Dachsbacher, "Cascaded light propagation volumes for real-time indirect illumination," in *Proc. ACM SIGGRAPH Symp. Interactive 3D Graph. and Games*, Washington, DC, 2010, pp. 99–107.

[53] M. Hašan *et al.*, "Direct-to-indirect transfer for cinematic relighting," *ACM Trans. Graph.*, vol. 25, no. 3, 2006.

[54] S. Martin and P. Einarsson, "A real-time radiosity architecture for video games," SIGGRAPH Courses: Advances in Real-Time Rendering in 3D Graphics and Games, Vancouver, BC, 2010.

[55] C. Crassin *et al.*, "Interactive indirect illumination using voxel cone tracing," *Comput. Graph. Forum*, vol. 30, no. 7, sep 2011.

[56] G. McTaggart, "Half-Life 2 source shading," in *Game Developers Conf.*, San Francisco, CA, 2004.

[57] H. Chen, "Lighting and Materials of Halo 3," in *Game Developers Conf.*, San Francisco, CA, 2008.

[58] M. Meyer and J. Anderson, "Statistical acceleration for animated global illumination," *ACM Trans. Graph.*, vol. 25, no. 3, 2006.

[59] I. Ashdown, "Eigenvector radiosity," Master's thesis, Dept. Comp. Sci., Univ. British Columbia, April 2001.

[60] M. Levoy and P. Hanrahan, "Light field rendering," in *Proc. Conf. Computer Graph. and Interactive Techniques*, New Orleans, LA, 1996, pp. 31–42.

[61] S. J. Gortler *et al.*, "The lumigraph," in *Proc. Conf. Comput. Graph. and Interactive Techniques*, New Orleans, LA, 1996, pp. 43–54.

[62] G. Greger *et al.*, "The irradiance volume," *IEEE Comput. Graph. Appl.*, vol. 18, no. 2, pp. 32–43, Mar. 1998.

[63] C. Dachsbacher *et al.*, "Implicit visibility and antiradiance for interactive global illumination," *ACM Trans. Graph.*, vol. 26, no. 3, 2007.

[64] Z. Dong *et al.*, "Interactive global illumination using implicit visibility," in *Pacific Conf. Comput. Graph. and Appl.* Maui, HI: IEEE Computer Society, 2007, pp. 77–86.

[65] M. Stanton *et al.*, "Non-polynomial galerkin projection on deforming meshes," *ACM Trans. Graph.*, vol. 32, no. 4, pp. 86:1–86:14, Jul. 2013.

[66] D. A. Calian *et al.*, "The shading probe: Fast appearance acquisition for mobile ar," *SIGGRAPH Asia Tech. Briefs*, pp. 20:1–20:4, 2013.

[67] L. Antani and D. Manocha, "Aural proxies and directionally-varying reverberation for interactive sound propagation in virtual environments," *IEEE TVCG*, vol. 19, no. 4, Apr. 2013.

[68] D. Gerszewski *et al.*, "Enhancements to model-reduced fluid simulation," *Proc. Motion in Games*, pp. 201:223–201:228, Nov. 2013.

[69] R. R. Lewis and A. Fournier, "Light-driven global illumination with a wavelet representation of light transport," Univ. British Columbia, Vancouver, BC, Canada, Tech. Rep. FTR-95-28, 1995.

[70] M. Wicke, M. Stanton, and A. Treuille, "Modular bases for fluid dynamics," *ACM Trans. Graph.*, vol. 28, no. 3, 2009.

[71] H. Xu, Q.-S. Peng, and Y.-D. Liang, "Accelerated radiosity method for complex environments," in *Eurographics*, 1989, pp. 51–61.

[72] G. Nichols and C. Wyman, "Multiresolution splatting for indirect illumination," in *Proc. Symp. Interactive 3D Graph. and Games*, Boston, MA, 2009, pp. 83–90.

[73] G. Nichols *et al.*, "Hierarchical image-space radiosity for interactive global illumination," *Comput. Graph. Forum*, vol. 28, no. 4, 2009.

[74] D. Larsson and H. Halen, "The unique lighting of Mirror's Edge," in *Game Developers Conf.*, San Francisco, CA, 2009.

[75] B. J. Loos *et al.*, "Modular radiance transfer," in *Proc. SIGGRAPH Asia Conf.*, Hong Kong, China, 2011, pp. 178:1–178:10. [Online]. Available: http://doi.acm.org/10.1145/2024156.2024212

[76] B. Loos *et al.*, "Runtime implementation of modular radiance transfer," in *ACM SIGGRAPH Talks*, Vancouver, BC, 2011, pp. 59:1–59:1. [Online]. Available: http://doi.acm.org/10.1145/2037826.2037905

[77] P. Shirley and K. Chiu, "A low distortion map between disk and square," *J. Graph. Tools*, vol. 2, no. 3, pp. 45–52, Dec. 1997.

[78] B. J. Loos *et al.*, "Delta radiance transfer," in *Proc. ACM SIGGRAPH Symp. Interactive 3D Graph. and Games*, Costa Mesa, CA, 2012, pp. 191–196. [Online]. Available: http://doi.acm.org/10.1145/2159616.2159648

[79] B. J. Loos and P.-P. Sloan, "Volumetric obscurance," in *Proc. ACM SIGGRAPH Symp. Interactive 3D Graph. and Games*, Washington, DC, 2010, pp. 151–156. [Online]. Available: http://doi.acm.org/10.1145/1730804.1730829

[80] S. P. Lloyd, "Least squares quantization in PCM," *IEEE Trans. Inf. Theory*, vol. IT-28, no. 2, pp. 129–137, Mar. 1982.

[81] V. Bulatov, "Point repulsion newsgroup posting," 1996. [Online]. Available: http://www.math.niu.edu/ rusin/known-math/96/repulsion

[82] P.-P. Sloan *et al.*, "Ambient obscurance baking on the GPU," in *SIGGRAPH Asia Tech. Briefs*, Hong Kong, China, 2013, pp. 32:1–32:4.

[83] L. Kavan *et al.*, in *Proc. Eurographics Conf. Rendering*, Prague, Czech Republic, 2011, pp. 1319–1326.

[84] B. Burley, "Physically-based shading at disney," in *ACM SIGGRAPH Courses*, Los Angeles, CA, 2012.

[85] R. Habel and M. Wimmer, "Efficient irradiance normal mapping," in *Proc. ACM SIGGRAPH Symp. Interactive 3D Graph. and Games*, Washington, DC, 2010, pp. 189–195.