Second-Order Illumination in Real-Time (Student Paper)

Robert Cochran
Department of Computer Science
Clemson University
Clemson, SC 29634
rcochra@cs.clemson.edu

Jay Steele
Department of Computer Science
Clemson University
Clemson, SC 29634
jesteel@cs.clemson.edu

ABSTRACT

This paper presents a simple and efficient algorithm for achieving real-time performance on current consumer graphics hardware when rendering complex, dynamic scenes with direct and second-order diffuse (indirect) illumination. An image space, low-discrepancy sampling technique for positioning point lights is presented. These point lights simulate second-order diffuse illumination throughout the scene. A novel use of negative point lights allows fast approximation of occlusion of second-order diffuse illumination in image space. Finally, an optimization technique is provided that improves frame rates while maintaining image quality by approximating the illumination of all point lights at lower resolutions for less detailed areas of the scene.

Categories and Subject Descriptors

I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism—*Color, shading, shadowing, and texture*

General Terms

Algorithms, Performance

Keywords

global illumination, GPU processing, hardware acceleration, real-time rendering

1. INTRODUCTION

Approximating global illumination is key in adding realism to computer generated images. The two main components of global illumination are direct illumination, which arrives directly from light sources, and indirect illumination, which results from light reflecting off surfaces in the scene. Indirect illumination contributes greatly to the realism of images (figure 1); however, computing indirect illumination is very expensive. Non-real-time global illumination algorithms generate very realistic results, but the rendering time for each frame often takes minutes or hours. Real-time applications, which require frames to be rendered in a few milliseconds, can easily compute direct illumination, but these applications have historically relied on precomputed light maps or other static techniques for approximating indirect illumination.

In this paper we present an algorithm for rendering complex, dynamic scenes in real-time with direct and indirect illumination. Specifically, we approximate second-order diffuse reflections (indirect illumination) as well as occlusion of these reflections. Our work is an extension of Keller's Instant Radiosity [8]. Our algorithm does not rely on precomputations and supports fully dynamic scenes, that is, both viewer and scene objects may be in motion. All light properties (position, shape, color) can change per frame. Our algorithm works in image space; therefore, the processing time per frame is largely independent of scene complexity. To approximate indirect illumination, our algorithm adds point lights to the scene at each frame, which we refer to as indirect lights. We describe a method for determining the indirect light properties, which is key to generating quality images, using a low-discrepancy sampling technique. Calculating shadows (occlusion) due to each indirect light is too expensive; therefore, we describe a technique for approximating occlusion of second-order diffuse reflections using additional point lights, called negative indirect lights, that subtract light from the scene. In order to achieve real-time performance with many point lights while maintaining image quality, we optimize our algorithm by using a low resolution approximation for the less detailed areas of the scene. We maintain a real-time (at least 24 fps) frame rate when rendering complex scenes.

In the next section, we provide an overview of global illumination and previous techniques, both real-time and non-real-time. In Section 3 we discuss the details of our algorithm. Section 4 provides implementation details, and section 5 contains results. Finally, we discuss current issues and future work.

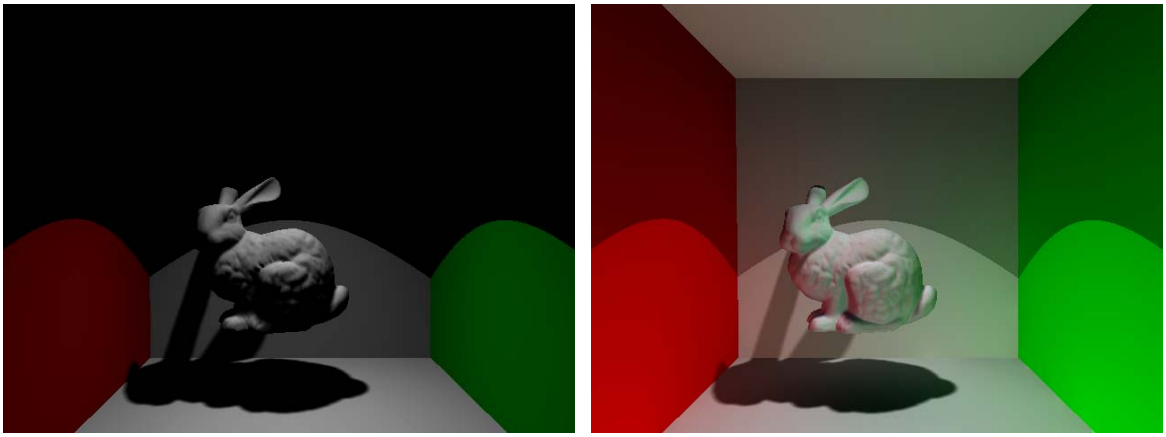
2. PREVIOUS WORK

Radiance is the fundamental quantity of interest in global illumination algorithms. Human eyes and other sensors are sensitive to the amount of radiance incident on them [5]. Radiance is a measure of light intensity per unit surface area perpendicular to the flux ($Watts/(sr \cdot m^2)$). Flux, which is expressed in Watts ($Joule/sec$), is a measure of energy flowing per unit time. The rendering equation

$$L(x,\vec{w}) = \int_S \beta(x,\vec{w_i},\vec{w}) L(x_o,\vec{w_o}) \frac{\cos\theta_i \cos\theta_o}{|x-x_o|^2} V(x,x_o) dA$$

describes how light flows in a vacuum. Specifically, it provides the exiting radiance, $L(x,\vec{w})$, at point x in direction $\vec{w}$. (See figure 2.) Let $\vec{w_i} = -\vec{w_o}$ and let x and x_o denote base points of $\vec{w}$ and $\vec{w_o}$ respectively. Visibility is defined by $V(x,x_o)$, which is 1 if x_o is visible from x and 0 otherwise. The bi-directional reflectance

ACMSE 2007, March 23-24, 2007, Winston-Salem, N. Carolina, USA



(a) (b)

Figure 1: Our test scene, the Stanford Bunny floating in a box lit by a spotlight, demonstrates direct and indirect illumination due to second-order diffuse reflections. (a) The scene lit with only direct illumination. (b) The scene lit with direct and indirect illumination.

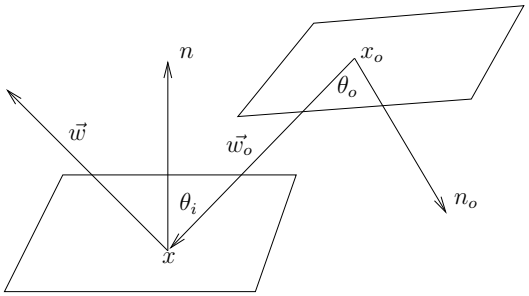


Figure 2: Components of the rendering equation.

distribution function (BRDF), $\beta(x, \vec{w_i}, \vec{w})$, is the ratio of reflected radiance at x in direction $\vec{w}$ to the incident irradiance at x from direction $\vec{w_o}$, where irradiance is the incident energy per unit surface area ($Watts/m^2$). Put simply, the BRDF describes at x how much light from direction $\vec{w_o}$ is reflected in direction $\vec{w}$. Summing over all surfaces from which irradiance arises gives us $L(x, \vec{w})$.

Many non-real-time rendering techniques exist for approximating the rendering equation and, therefore, global illumination. These techniques range from radiosity [6], which is concerned with higher-order diffuse reflections, to ray tracing, which is largely concerned with specular reflections, to photon mapping [7], which handles both higher order specular and diffuse reflections. Non-real-time techniques can produce very realistic results; but, because of the complexity of global illumination and the high quality results, each rendered frame often takes minutes or even hours to compute.

Real-time rendering techniques make use of graphics cards. Fortunately, the processing units on graphics cards, known as GPUs, have consistently defied Moore's Law by doubling performance every six months. As GPU power increases, more and more algorithms become available for approximating global illumination in real-time. Real-time methods attempt to find a balance between frame rate and image quality, and many of these methods introduce restrictions, such as static lights or static objects. The classical approach involves precomputing light maps for static lights and then texturing static objects with these light maps. Obviously, this greatly restricts the interactivity of the application. Newer techniques, such as Coombe, et al. [2], perform calculations at runtime that mimic non-real-time techniques. These techniques usually compute part of the final solution per frame and slowly converge to the final solution. The image quality improves as the application runs, but it will be incorrect if the scene changes.

Keller [8] introduced the idea of simulating indirect illumination with consumer graphics hardware by placing new point lights throughout the scene, which we call indirect lights. Each indirect light approximates light reflecting off a surface. Higher-order diffuse and specular reflections can be approximated using this technique. The locations of these indirect lights are determined by shooting particles from the light into the scene. New indirect lights are positioned at the intersection of each particle and scene. The scene is rendered (with shadows) with each of the indirect lights; the resulting images are accumulated to produce a final image that contains both direct illumination and indirect illumination.

Recently, new derivatives of Keller's work have introduced algorithms that achieve real-time frame rates for dynamic scenes. Sergovia, et al. [9], introduced a novel technique that reduces the workload of each indirect light while maintaining reasonable image quality. This is accomplished by restricting the effect of each indirect light to a subset of visible surface points; a Gaussian blur is then applied to distribute the indirect illumination of a surface point to neighboring surface points. While not the basis of their algorithm, they rely on the precomputation of a kd-tree for determining indirect light properties. Dachsbacher and Stamminger [3] introduced the idea of sampling from the light's image space; thus, they avoid any precomputation necessary for intersection calculation. This was extended by the same authors in [4] to allow fast indirect illumination, including caustics, through splatting indirect lights. Splatting is a technique in which lighting calculations for an indirect light are restricted to the pixels that are directly affected by the light. Splatting works best when each indirect light affects a small portion of image space; as the size of each indirect light's influence increases in image space, the size of the splat increases and performance degrades. None of the previous techniques dynamically account for occlusion of indirect lighting. This can lead to unrealistic results since a surface may be illuminated by indirect lights that are not visible from the surface. Lastly, Bunnell [1] intro-

duced a technique that allows for computing diffuse reflections and occlusions of diffuse reflections in real-time. This method requires converting polygonal data to surface elements, which are oriented disks, and works best with highly detailed models.

The algorithm we present differs from previous approaches in three main ways: we provide a new image space, low-discrepancy sampling technique for positioning indirect lights; we allow an approximation of occlusion of second-order diffuse reflections through a novel use of negative point lights in image space; we provide a new technique that reduces the average work per pixel while maintaining image quality.

3. ALGORITHM

We approximate indirect illumination by positioning point lights, which we call indirect lights and negative indirect lights, throughout the scene. We are only concerned with second-order diffuse reflections and occlusion of these reflections since these tend to contribute the most to the final image quality [10]. The quality and the frame rate of an interactive scene is heavily dependent on the number of point lights created for each frame. Our algorithm achieves real-time frame rates for complex scenes with many point lights per frame. In discussing our algorithm, we will assume one spotlight exists in the scene, but extending our algorithm to multiple spotlights is straightforward.

3.1 Positioning Indirect Lights

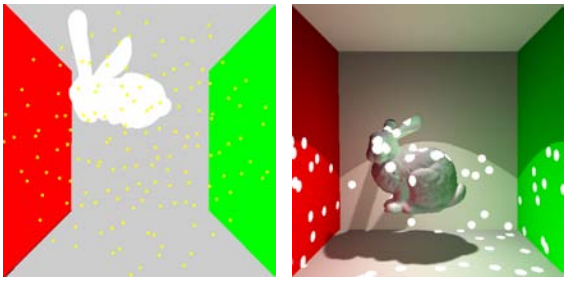
Our algorithm automatically positions indirect lights, which are point lights that approximate second-order diffuse reflections, into the scene for each frame. Determining our indirect light positions is key in generating a realistic image. Indirect lights should only be positioned on surfaces that are directly lit by the main spotlight. Flickering, which occurs when indirect lights are drastically altered from frame to frame, in a truly dynamic scene is hard to avoid; however, our algorithm ameliorates the affects of flickering due to temporal changes and camera changes. Previous approaches have positioned indirect lights by casting rays from the main light into the scene. The intersection of each ray and scene designates a new indirect light position. For real-time applications, preprocessing steps, such as kd-trees, have been used [9] to achieve fast ray/scene intersection calculations. But, these preprocessing steps can severely limit scene interactivity, and ray casting requires specific ray/geometry intersection routines.

Instead of explicitly checking for ray/scene intersections, we simplify this necessary step by rendering the unlit scene from the spotlight's point of view. Each pixel in the 2D image corresponds to a surface point in the scene that is an ideal candidate for a indirect light position. Not only do we need the color (indirect light color) of each surface point, we also need each point's world space position (indirect light position) and normal (indirect light direction). Thus, we do not render directly into the framebuffer. Similar to [3, 4], we render each value into a separate texture using a GPU fragment program. Combined, these three textures describe a set of surface points, S_{light}. Each point in S_{light} is visible from the view of the spotlight and represents a potential indirect light; also, all possible indirect light positions are contained in S_{light}. This technique integrates well with shadow maps, which require storing depth information for the scene while rendered from the spotlight's point of view. Figure 3 shows the sample points from the spotlight's point of view.

We now perform a low discrepancy sampling from S_{light} to determine the properties (position, direction, and diffuse color) of our N indirect lights. Similar to [8], our samples are based on the Halton sequence. The Halton sequence provides a uniformly distributed, quasi-random, low discrepancy sampling pattern. The quasi-random aspect allows our algorithm to minimize flickering due to temporal changes or camera changes. Indirect light positions only vary frame to frame due to transformations of scene objects or changes of the spotlight. First, we generate sample points, s_i, in $\Re^3$ uniformly on the unit sphere surrounding the scene's spotlight using

$$\begin{aligned} \varphi &= 2\pi\Phi_2(i) \\ \delta &= \arcsin(2\Phi_3(i) - 1) \\ s_i &= (\sin(\varphi)\cos(\delta), \cos(\varphi)\cos(\delta), \sin(\delta)) \end{aligned}$$

where $\Phi_j(i)$ is the i-th number in the Halton sequence based on the j-th prime number. (Write i in base j, $i = \sum_{k=0}^{\infty} a_k j^k$, then $\Phi_j(i) = \sum_{k=0}^{\infty} a_k j^{-k-1}$.) We select the first N sample points that fall into intersection of the cone of the spotlight and the unit sphere surrounding the spotlight, rejecting all others. Accepted sample points are projected into the image space of the spotlight. The N projected sample points, which are in $\Re^2$, allow us to select N surface points from S_{light}. Each surface point defines the position, direction, and color for one of our N indirect lights. Figure 3 shows the projected sample points overlapped on an image rendered from the spotlight's point of view and the resulting indirect light positions from the camera's point of view.



(a) (b)

Figure 3: (a) The view from the spotlight, with the indirect light sample points displayed in yellow. (b) The view from the camera, with the indirect light positions highlighted with white circles.

3.2 Computing Indirect Illumination

Computing illumination due to all N indirect lights requires treating each indirect light as a point light. We make the simplifying assumption that indirect lights are visible from each surface point; that is, we do not calculate shadows directly for indirect lights. Accurate shadows from each indirect light are far too expensive to compute. In Section 3.3, we introduce a technique for approximating shadows for indirect lights.

We exploit a deferred shading pass to avoid repetitive calculations due to multiple passes and to avoid expensive calculations for hidden fragments. Similar to rendering from the view of the light, we render the scene from the camera's point of view without lighting into multiple textures; for each pixel, we store its world space position, normal, and material information. These textures define a set of surface points, S_{cam}. Each surface point corresponds to a pixel in the final rendered frame, and S_{cam} only contains surface points that are visible from the camera (i.e. those that contribute to the final image). We can now perform our expensive lighting calculations only on points in S_{cam} and avoid performing any lighting calculations on hidden surfaces. Combining deferred shading and

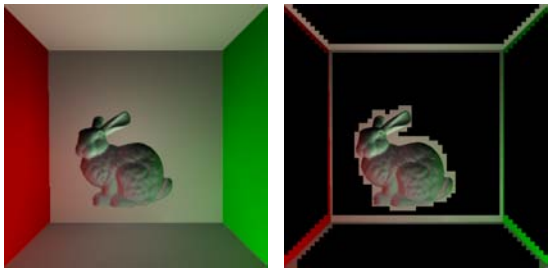
our image space light positioning technique allows our algorithm to be largely independent of scene complexity. The actual scene is only rendered twice: once from the point of view of the spotlight and once from the camera's point of view. The indirect illumination calculations, of which there may be N per pixel, now dominate our per frame processing time.

3.3 Adding a Visibility Approximation

Recall that the rendering equation (Section 2) makes use of a visibility function $V(l, p)$, which, given the position of a light source l and a point p determines whether l is visible from p. The use of visibility function improves image quality by accurately portraying the shadows cast by all lights in a scene. However, the construction of a visibility function V, using a method such as shadow mapping, for N indirect lights is prohibitively expensive; therefore, to compute indirect illumination in real-time we avoid the explicit determination of whether a light l is visible from a point p.

We categorize the calculation of indirect illumination by whether or not a visibility function is used. We define *simple indirect illumination* as indirect illumination calculated without a visibility function, i.e., $V(l, p)$ is always defined as 1. We denote the simple indirect illumination at a point p as $I_s(p)$. Put simply, the indirect lights do not cast shadows. So far, our algorithm calculates simple indirect illumination. A more accurate model is *complex indirect illumination*, which is calculated with a visibility function $V(l, p) = \{0, 1\}$; in other words, the indirect lights do cast shadows. We denote the complex indirect illumination at a point p as $I_c(p)$. Note that $I_s(p)$ is always at least $I_c(p)$, since the assumption that the indirect lights do not cast shadows introduces additional illumination. We define this additional illumination as *negative indirect illumination* which we denote at a point p as $I_n(p)$. Given $I_n(p)$, we have $I_s(p) - I_n(p) = I_c(p)$. As we have described how to approximate $I_s(p)$, the key insight of our algorithm is an efficient method of approximating $I_n(p)$. This in turn provides an efficient approximation of $I_c(p)$.

We approximate negative indirect illumination using additional point lights which we call negative indirect lights. The properties of the negative indirect lights are determined using the same general scheme described in Section 3.1, where we render the properties of the surface points visible from the spotlight's point of view into multiple textures, and then sample from these textures to determine the indirect lights' properties. To determine the properties of the negative indirect lights, we sample from the surface points visible from the camera's point of view. For each of these points, we compute the "negative" illumination from the (already selected, "positive") indirect lights. At a point p, given the surface normal, $\vec{n_p}$ and the vector, $\vec{t_{l,p}}$, from p to an indirect light l, the "negative" illumination is simply the illumination at p from l computed with $\vec{n_p} \cdot \vec{t_{p,l}}$ negated. Note that this dot product implies the surface at p is directed *away* from l, and would not be illuminated by l in the standard lighting model. Also at each point p, we weight the direction vector of each light l with a weight function $w_p(l)$ and compute a weighted average over the direction vectors of all indirect lights. $w_p(l)$ is equal to the intensity of the indirect illumination arriving at p from l. This average then corresponds to the direction of a negative indirect light if such a light were to be placed at p. The "negative" illumination and direction at each point p can be stored in multiple textures by a fragment shader. These textures form a set of surface points S_{neg}. We then determine the properties (position, direction, and diffuse color) of the negative indirect lights via a Halton sampling of the points in S_{neg}. After the negative indirect light properties have been determined, we compute a negative indirect illumination texture using the techniques in section 3.2.



(a) (b)

Figure 4: (a) A low resolution second-order diffuse approximation is computed for all pixels. (b) A high resolution second-order diffuse approximation is computed for all pixels that the low-resolution approximation is not suitable.

3.4 Final Gather

Our final step requires accumulating the contributions of the spotlight, indirect lights, and negative indirect lights into one image. We render the scene from the camera's point of view, and a fragment program calculates the direct lighting and shadows (using shadow mapping) for each pixel. Also, for each pixel, we add the value of the corresponding texel in our indirect illumination and subtract the value of the corresponding texel in our negative indirect illumination texture. Figure 5 shows each step.

3.5 Optimization

Our algorithm's performance and image quality is dependent on the number of point lights (indirect lights and negative indirect lights) inserted into the scene: increasing the number of point lights improves the image quality but degrades performance. By optimizing our algorithm, we can minimize the degradation associated with this increase while maintaining image quality.

As noted in [3], for many of the surface points in S_{cam}, a cheaper evaluation may be sufficient for approximating indirect illumination. First, we downsample the deferred shading textures that form S_{cam} to form a smaller (lower resolution) set of surface points called S_{locam}. This has the effect of storing the surface points from a low resolution render from the camera's point of view into S_{locam}. Using a fragment program, we compute the second-order diffuse illumination for each surface point in S_{locam} due to all N indirect lights and store these values in a texture. This low resolution, indirect illumination texture is upsampled to full resolution to form the basis for our final indirect illumination texture. For each surface point $p \in S_{cam}$, we determine if the corresponding indirect illumination value computed from S_{locam} is a good approximation. This is accomplished by comparing p's normal and world space position with the normal and world space position of p's corresponding surface point in S_{locam}. If the difference is greater than a threshold (which we call the discontinuity threshold), we calculate p's indirect illumination due to all N indirect lights and overwrite the value in our indirect illumination texture. (See figure 4.) This same technique is used to computer our negative indirect illumination texture.

The performance increase achieved through use of this optimization technique is scene dependent. It is most effective when the indirect illumination for a majority of the surface points in image space can be approximated by the low resolution pass. If needed, the discontinuity threshold can be adjusted to find a suitable balance between image quality and frame rate.

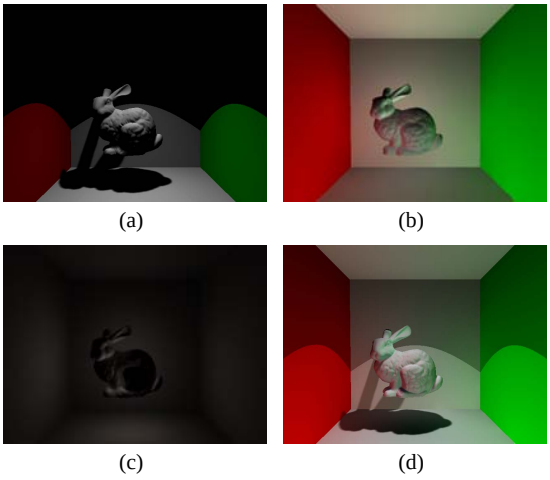


Figure 5: Illumination from (a) the spotlight, (b) the indirect lights, and (c) the negative indirect lights. (d) The final result is a combination of (a), (b), and (c).

Indirect Lights	Negative Indirect Lights	Frame rate (FPS)
100	50	26.5
150	0	32.6
200	100	15.6
300	0	19.2
300	150	11.0
450	0	13.6

Table 1: Frame rates when rendering our test scene at 640 x 480 with varying numbers of indirect lights and negative indirect lights.

Low Resolution Percentage	RMS	Frame rate (FPS)
100.0%	1.5082%	42.5
87.5%	0.1688%	26.5
50%	0.0885%	15.4
0%	0%	12.6

Table 2: "Low Resolution Percentage" is the percentage of pixels for which a low resolution approximation was sufficient based on a discontinuity threshold. "RMS" is the root mean square error in the resulting image when compared to an image rendered without optimization.

4. IMPLEMENTATION

We implemented our algorithm using OpenGL and GLSL. Our algorithm requires multiple passes, with only two passes requiring the scene to be rendered. The first pass renders the scene from the spotlight's point of view. In this pass, we use a fragment program to store each pixel's world space position, normal, and material information into three 16-bit float textures, each with four components. These textures define our set S_{light} of surface points that are visible from the spotlight's point of view. Although 32-bit float textures are available, the additional precision does not provide sufficient benefit to overcome the extra memory bandwidth requirements. A depth texture is also created in this pass and is used to generate shadows from the main spotlight.

Next, we perform a deferred shading pass from the camera's point of view. Another fragment program stores world space coordinates, normal, and material properties for each pixel in three 16-bit float textures, each with four components. These textures define our set S_{cam} of surface points that are visible from the camera's point of view. Our lower resolution set S_{locam} is created by downsampling the full resolution deferred shading textures. Each surface point in S_{locam} corresponds to four surface points in S_{cam}.

We use a discontinuity buffer to determine the subset of S_{cam} for which a low resolution, indirect illumination approximation is suitable. For surface points in S_{cam} that differ greatly from the corresponding surface point in S_{locam}, a red pixel is stored in the discontinuity buffer. Otherwise, a black pixel is written. The discontinuity buffer is then downsampled to a lower resolution. This lower resolution discontinuity buffer is transferred to the CPU for analysis. Transferring and analyzing a full resolution discontinuity buffer adversely affects performance.

Our indirect illumination texture is computed in multiple steps. First, for all surface points in S_{locam}, we store the indirect illumination due to all N indirect lights in a 16-bit float texture. This texture is upsampled to full resolution; each texel defines the indirect illumination for a surface point in S_{cam}. But, this approximation may not be accurate enough. For every red pixel in the discontinuity buffer, we recompute the indirect illumination for the corresponding surface point in S_{cam} and replace the value stored in our indirect illumination texture. This is accomplished by drawing quadrilaterals over pixels that require recalculation of indirect illumination. Our fragment program will only run over the pixels covered by each quadrilateral.

In the same passes that we compute our indirect illumination texture, we also compute the textures that are sampled to create our negative indirect illumination texture. We compute our final negative indirect illumination texture using the hierarchical method described above for computing our indirect illumination texture. Our final pass simply combines direct illumination and shadow mapping with our indirect illumination and negative indirect illumination values.

5. RESULTS

Our test scene was rendered on a NVIDIA Quadro FX 5500. All tests were run at a resolution of 640 x 480. Our sample scene consists of a box (each wall is composed of two triangles) and the Stanford Bunny (composed of 69,451 triangles). Table 1 illustrates that the performance of our algorithm is highly dependent on the number of point lights (indirect lights and negative indirect lights) added to the scene. More point lights results in higher image quality, but lower frame rates. Note that negative indirect lights are more expensive than indirect lights. This is due to extra overhead (processing and memory bandwidth) required in positioning the negative indirect lights. This cost is amortized away as the number of negative indirect lights increases.

Figure 6 demonstrates the effectiveness of our visibility approximation that makes use of negative indirect lights. Note the enhanced detail around the bunny's eye and ear. Our occlusion approximation improves the realism of the scene without the cost associated with more accurate shadowing methods.

Recall that our optimization technique applies a low resolution approximation to certain pixels based on a discontinuity threshold. As shown in table 2, the percentage of pixels for which a low resolution approximation is used has a direct affect on the image quality and frame rate. For all scenes, a balance between frame rate and

(a) (b)

Figure 6: Comparison of the scene with (a) no negative indirect lights and (b) with negative indirect lights.

(a) (b)

Figure 7: Comparison of a (a) low resolution approximation for all pixels and a (b) low resolution approximation for 87.5% of pixels.

image quality can be found by adjusting the discontinuity threshold. Using a low resolution approximation for all pixels, while maximizing frame rate, results in visible artifacts, as seen in figure 7. But, rendering the scene without any optimization leads to less than ideal frame rates. Notice that rendering with a low resolution percentage of 87.5% resulted in less than 0.2% error and significantly faster frame rates when compared to rendering without optimization.

6. CONCLUSION

Based on the work of Keller, we have presented a simple and efficient technique for approximating indirect illumination in real-time. Our algorithm exploits the performance capabilities of current GPUs to achieve real-time performance when rendering complex, dynamic scenes. We have introduced an image space, low-discrepancy sampling technique for positioning indirect lights in the scene. These indirect lights allow simulation of second-order diffuse illumination. We described a technique, using negative point lights, that provides a fast approximation of occlusion of the second-order diffuse reflections. Finally, we provided an optimization technique that approximates second-order diffuse illumination at lower resolutions for less detailed areas of the scene. This provides improved frame rate while maintaining image quality.

Future work will focus on improving the performance and the image quality of our results. Currently, we rely on low resolution approximations to attain real-time frame rates. While not adversely affecting image quality, the efficiency of this method is highly scene dependent. The computation time per frame is based highly on the number of point lights (indirect lights and negative indirect lights); therefore, techniques for reducing the computation cost per point light are desirable. We feel that we can improve the accuracy of our occlusion approximation while retaining its benefits. Finally, we believe that distributing the workload to multiple GPUs will allow us to simulate higher-order reflections, including caustics, while maintaining our real-time performance.

7. ACKNOWLEDGMENTS

We would like to thank NVIDIA for partial support of this research through their Fellowship Program. We would also like to thank our advisor, Dr. Robert Geist, for his suggestions and insight throughout this process.

8. REFERENCES

[1] M. Bunnell. Dynamic ambient occlusion and indirect lighting. In M. Pharr, editor, *GPU Gems 2*, chapter 14, pages 223–233. Addison Wesley, Mar. 2005.

[2] G. Coombe, M. J. Harris, and A. Lastra. Radiosity on graphics hardware. In *GI '04: Proceedings of the 2004 conference on Graphics interface*, pages 161–168, School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 2004. Canadian Human-Computer Communications Society.

[3] C. Dachsbacher and M. Stamminger. Reflective shadow maps. In *SI3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games*, pages 203–231, New York, NY, USA, 2005. ACM Press.

[4] C. Dachsbacher and M. Stamminger. Splatting indirect illumination. In *SI3D '06: Proceedings of the 2006 symposium on Interactive 3D graphics and games*, pages 93–100, New York, NY, USA, 2006. ACM Press.

[5] P. Dutr, P. Bekaert, and K. Bala. *Advanced Global Illumination*. A K Peters, Natick, USA, 2003.

[6] C. Goral, K. Torrance, D. Greenberg, and B. Battalie. Modeling the interaction of light between diffuse surfaces. In *Proc. SIGGRAPH '84*, pages 213–222, July 1984.

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

[8] A. Keller. Instant radiosity. In *Proc. SIGGRAPH '97*, pages 49–56, August 1997.

[9] B. Segovia, J.-C. Iehl, R. Mitanchey, and B. Péroche. Non-interleaved deferred shading of interleaved sample patterns. In *Proceedings of SIGGRAPH/Eurographics Workshop on Graphics Hardware 2006*, 2006.

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