Splatting Indirect Illumination

Carsten Dachsbacher*
University of Erlangen-Nuremberg

Marc Stamminger†
University of Erlangen-Nuremberg

Abstract

In this paper we present a novel method for plausible real-time rendering of indirect illumination effects for diffuse and non-diffuse surfaces. The scene geometry causing indirect illumination is captured by an extended shadow map, as proposed in previous work, and secondary light sources are distributed on directly lit surfaces. One novelty is the rendering of these secondary lights' contribution by splatting in a deferred shading process, which decouples rendering time from scene complexity. An importance sampling strategy, implemented entirely on the GPU, allows efficient selection of secondary light sources. Adapting the light's splat shape to surface glossiness also allows efficient rendering of caustics. Unlike previous approaches the approximated indirect lighting does barely exhibit coarse artifacts – even under unfavorable viewing and lighting conditions. We describe an implementation on contemporary graphics hardware, show a comparison to previous approaches, and present adaptation to and results in game-typical applications.

CR Categories: I.3.3 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture; I.3.3 [Computer Graphics]: Hardware Architecture—Graphics processors;

Keywords: global illumination, real-time rendering, shadow maps, hardware-assisted rendering

1 Introduction

Indirect illumination is a subtle, yet important component for realistic rendering. Due to its global nature the computation of indirect illumination is notoriously slow. Thus in interactive graphics indirect illumination is mainly present in the form of precomputed radiosity light maps, or, as a simple yet impressive approximation, by using ambient occlusion.

On the other hand, coarse approximations for indirect light are usually satisfactory. In most cases one-bounce indirect light is visually sufficient [Tabellion and Lamorlette 2004], and even occlusion for the indirect light can be ignored in many cases. However, indirect light must be temporally consistent and may not exhibit flickering.

In this paper we present a novel algorithm for the approximate computation of indirect light, which also relies on the aforementioned simplificating assumptions. Our algorithm runs in realtime for typical game scenarios. It computes approximate, time-consistent one-bounce indirect illumination from a point light. Both camera and light source can be dynamic. Our algorithm is based on the idea of Reflective Shadow Maps [Dachsbacher and Stamminger 2005]: for a point light source, all surfaces causing one-bounce indirect illumination are captured by a shadow map. In order to compute indirect illumination from these surfaces, a Reflective Shadow Map stores additional information, i.e. surface location, normal and reflected radiant flux, for each pixel. The indirect illumination is approximated by gathering the contribution from a well-chosen subset of these surface samples, also called *pixel lights*. We reorganized the computation of the indirect light in such a way that we can achieve good results with many fewer indirect light sources on average, which increases performance significantly. Essentially, instead of gathering the indirect light as it is done with Reflective Shadow Maps, we use a shooting approach. For every indirect light, we splat its contribution into its screen space neighborhood. By this, we can better limit the area of influence of the indirect lights. The set of indirect light sources is the same for all pixels of an image, so noise is reduced at the cost of much less visible bias. By keeping the indirect light source positions consistent in between frames, flickering can be largely avoided. Altogether, we can reduce the number of indirect lights and achieve significantly better performance.

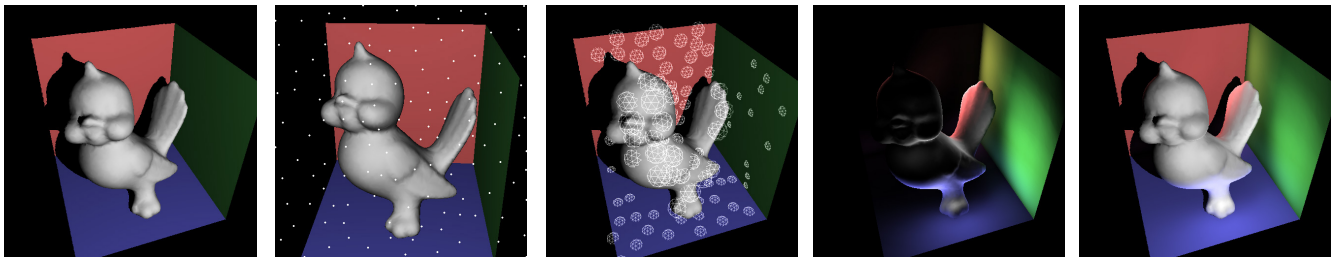
Furthermore, we can compute indirect lighting from glossy reflectors, which enables us to obtain both indirect diffuse light and caustics in good quality with a unified approach. This is shown in the teaser, where we can see some indirect lights for a diffuse and a glossy scene (first and third image), and the resulting images with indirect diffuse and glossy light (second and fourth image). The fifth image shows a screenshot of a Quake walkthrough with indirect light, which is rendered in real-time.

2 Previous Work

Indirect light is usually generated in offline rendering using ray tracing or radiosity. In [Tabellion and Lamorlette 2004], Tabellion et al. presented a simplified indirect illumination model that has been used for film production rendering. Even for their high-quality demands one-bounce indirect lighting turned out to be sufficient.

The first paper that introduced indirect radiosity-like lighting into

*e-mail: dachsbacher@cs.fau.de
†e-mail: stamminger@cs.fau.de



local illumination only | place samples in light view | create light sources | causing indirect lighting | add to local illumination

Figure 1: Diffuse indirect illumination caused by secondary light sources. Their positions are determined by a uniform sampling of the RSM.

interactive graphics was Instant Radiosity [Keller 1997]. A small set of photons (e.g. 200) is traced from the light source stochastically. Along the path of the photons indirect light sources are placed that approximate the indirect lighting. The contribution of these indirect light sources is computed using shadow maps and the accumulation buffer.

Ambient occlusion and bent normals are a simple approximation to modify local lighting in a way that fakes indirect light [Landis 2002]. The approach requires some precomputation but can then achieve indirect lighting effects with almost no performance penalty. Dynamic scenes require special treatment, however. Bunnell [Bunnell 2005] computes ambient occlusion for dynamic scenes on a per-vertex level. He also mentions indirect lighting caused by constantly emitting surfaces.

Reflective Shadow Maps [Dachsbacher and Stamminger 2005] are a more general approach for the inclusion of indirect light into interactive applications. For every scene point to be lit, a set of pixels of a shadow map are considered as indirect light sources. Their indirect contribution is gathered by a fragment program, where occlusion of these indirect lights is neglected. A rather big number of samples is necessary to obtain alias-free images, which requires high computational cost. Together with a screen-space interpolation step for indirect light, interactive to real-time frame rates are possible for moderate scenes.

Our approach computes indirect light in a deferred shading step. Hargreaves [Hargreaves 2004] discusses deferred shading in detail and gives performance hints and implementation strategies.

Wyman [Wyman 2005a; Wyman 2005b] presents techniques for rendering refractions in dynamic scenes for distant and nearby geometry. Shah et al. [Shah and Pattanaik 2005] describe an approach for real-time rendering of caustics from refractive and reflective surfaces solely based on shadow-maps.

Our approach requires a fast way to determine a set of light sources that represents complex illumination. In [Clarberg et al. 2005], an elegant method for this is presented for probability distributions given in a haar wavelet basis. First, a set of light sources is uniformly distributed. In a top-down approach the wavelet representation of the desired probability distribution is traversed and the light source samples are shifted according to the wavelet coefficients, so that the sample density adapts to the probability distribution. This process requires no iteration and is fast enough to be used within an interactive application.

3 Algorithm

Our method is based on the idea of Reflective Shadow Maps (RSMs) [Dachsbacher and Stamminger 2005]. It also uses an extended shadow map to create first-bounce indirect light sources. However, the way these indirect lights are determined and how the contribution of these lights is computed are different. As a result, we achieve a much more robust and faster algorithm. An overview of the algorithm is depicted in Fig. 1.

In the next sections we describe the idea of our new algorithm, show how the algorithm can be adapted to glossy indirect light and how performance can be further improved using fast importance sampling and efficient splatting. We start with the description for diffuse scenes, the extension to glossy indirect light is possible and is described later.

3.1 Reflective Shadow Maps

Our algorithm uses a shadow map to determine the surfaces in the scene that cause first-bounce indirect illumination. Because we need more information than only depth, we additionally store with every pixel the world space position, the surface normal and the reflected flux. Since this extended shadow map is the same as for the Reflective Shadow Map, we keep on referring to it as Reflective Shadow Map (RSM).

Each pixel of an RSM can be considered as a secondary light source. Indirect light is computed by considering all secondary lights as local point lights. These point lights represent a small surface light, so they are hemispherical with surface normal as main direction. We call such lights *pixel lights*. For a diffuse pixel light p with world space location x_p, direction n_p and flux Φ_p, the radiant intensity emitted into direction ω is:

$$I_p(\omega) = \Phi_p \max\{0, n_p \circ \omega\}, \tag{1}$$

where $\circ$ denotes the dot product. The irradiance due to p at a surface point x with normal n is then:

$$E_p(x,n) = I_p\left(\frac{x-x_p}{||x-x_p||}\right)\frac{\max\{0, n\circ(x_p-x)\}}{||x_p-x||^3}. \tag{2}$$

In the original Reflective Shadow Map approach [Dachsbacher and Stamminger 2005], indirect light for a particular point p is *gathered* from the RSM. Since gathering from all pixels is far too expensive, only a subset is used that depends on the point to be illuminated. The subset is focused to pixels close to p by sampling the shadow map neighborhood of p. Roughly 300 such samples are necessary to obtain satisfactory results in the examples. Deferred shading techniques and screen-space interpolation are necessary to achieve a fast enough approximation. Since the sampling pattern varies over the image, noise and striping artifacts become quickly apparent if the number of samples is too low. In order to avoid futile expensive fragment shader passes, deferred shading is used. With a screen-space interpolation technique the number of fragment shader passes is further reduced.

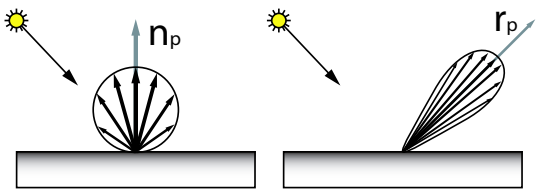


Figure 2: Emission of diffuse and glossy pixel lights (left and right, respectively).

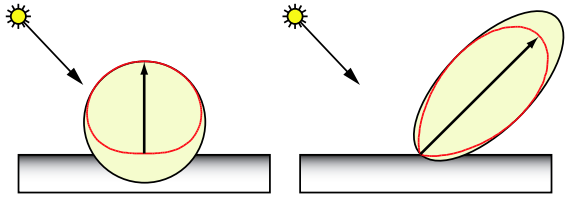


Figure 3: We adapt the bounding geometry for secondary light sources to reflection properties of the surface. The left image shows a diffuse, the right image a glossy reflection.

3.2 Splatting Indirect Illumination

In our novel approach the evaluation process is reversed: instead of iterating over all image pixels and gathering the indirect light from the RSM, we select a subset of RSM pixels, the pixel lights, and distribute their light to the image pixels, using a splat in screen space.

By nature, this splatting algorithm is a deferred shading pass. So we need buffers containing the world space position and normal and the material parameters for each screen pixel as seen from the viewer's camera. They are stored as textures that can be used as render targets, and are updated whenever the camera moves. When rendering the final image, the deferred shading buffers are used to compute the local illumination and shadowing from the light source, before the indirect illumination is computed.

Then the contribution of the indirect lights is computed directly on the deferred shading buffers. In general, such indirect lights only generate significant indirect light in their direct neighborhood. To account for this, we splat a quadrilateral in screen space at the position of the point light. The splat must be big enough to cover all fragments that can receive significant light. This size can be easily computed from the intensity of the pixel light and its distance to the camera. Details on this computation and tighter bounds are described in the next section.

For each covered pixel we retrieve the information of the visible geometry from the deferred shading buffers. Together with the light source parameters (associated with the quadrilateral) a fragment shader evaluates the pixel light's contribution to the underlying pixel according to Eq. 2 and accumulates it in the frame buffer. Thus we splat the contribution of each secondary light source onto the final image.

The computation of the quadrilateral can be efficiently implemented in a vertex shader, where the RSM, which stores the information about surfaces causing the indirect illumination, is sampled and the quadrilateral positioned. As a vertex shader only processes geometry and does not create geometry, the vertex data for the quadrilaterals is sent through the pipeline and modified accordingly. The sampling pattern for the RSM is provided as a texture and each quadrilateral has a texture coordinate to select a sampling position from this texture.

3.3 Non-diffuse Surfaces

We can easily adapt our method to render caustics and indirect illumination from glossy surfaces. To this end, we have to use pixel lights with a non-diffuse emission characteristic. Fig. 2 compares the emission of diffuse and glossy pixel lights.

For Phong-like surfaces, the emission of a glossy pixel light p with Phong exponent P is

$$I_p(\omega) = \Phi_p \max\{0, (r_p \circ \omega)\}^P, \tag{3}$$

where r_p is main light direction of p, which in turn is the reflection direction of light incident at p. r_p can be precomputed per glossy pixel light and is then a parameter of the light source such as P. Note that the emission should also be clamped at the surface of p, so that no light is emitted backwards.

Rendering with such light sources causes, as often seen in global illumination algorithms, problems along common boundaries of walls. This is because the illumination integral has a singularity, which is difficult to integrate numerically. For RSMs [Dachsbacher and Stamminger 2005] these problems are largely reduced by slightly moving the pixel lights in negative normal direction. This is a possible work-around for diffuse illumination, but with glossy reflection the problem becomes more apparent. For this, we replace the constant exponent P by $P'_p(\mathbf{x}) = \min\{P, P||\mathbf{x} - \mathbf{x_p}||a\}$, where $a > 0$ is a user-defined parameter. As a consequence, the narrow high intensity singularity is widened near the light source and thus less noticeable.

Due to their narrow shape, a screen space quad is a very wasteful approximation for a glossy pixel light. Instead, a tighter bounding geometry is mandatory in this case, as it is described in the next session. With such tight bounds, indirect glossy illumination can be achieved at similar speed as for the diffuse case. Usually, a higher number of glossy pixel lights is necessary, but on the other hand the region of influence is smaller and covers less pixels. Fig. 4 shows examples for caustics generated with our approach.

3.4 Tighter Bounding Geometry

With our approach, the performance bottlenecks are the fragment shader execution and memory bandwidth. Obviously an axis-aligned box in screen space is a wasteful representation and thus, as long as vertex processing does not become a critical factor, we

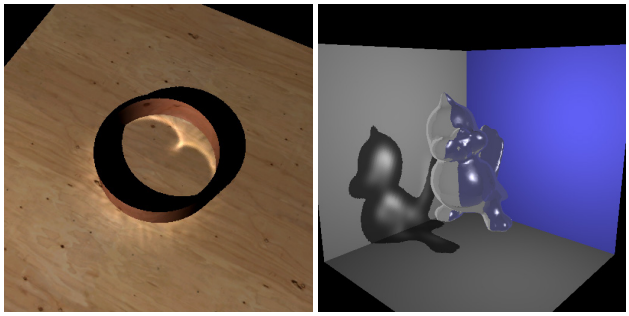


Figure 4: A real-time rendering of a metal ring and its caustics on a wood plate at more than 95 frames per second and a refractive glass tweety (only refraction at the back-facing surface) at 150 frames per second.

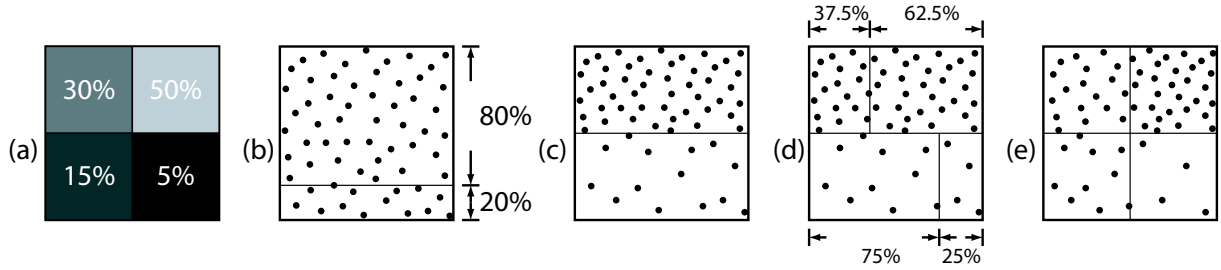


Figure 5: Importance warping for a single level of the hierarchical importance sampling as described in Sec. 3.5.

can afford to put more stress on the vertex engine and create tighter bounding volumes. Ideally, these bounds are computed and rendered in world space, which also allows us to make use of early-z tests to reduce fragment shader executions.

The goal is to compute a simple bounded region, outside of which the illumination falls below a threshold. As one can see in Fig. 3, for a diffuse pixel light, this region is egg-shaped, and for glossy surfaces the shape is similar to a Phong lobe. Note that these surfaces are iso-surfaces of illumination that include spatial attenuation and are not polar plots of exitant radiance!

For practical reasons, we use ellipsoids as bounds in both cases. For each pixel light, we have to compute the ellipsoid parameters and we transform a spherical triangle mesh (with low triangle count) accordingly. In the Appendix, we describe how the parameters of the ellipsoids can be computed quickly for diffuse and for Phong-like surfaces depending on the Phong exponent.

3.5 Importance Sampling

In order to adapt the sampling pattern for the RSM to its actual flux distribution, we perform an importance sampling as proposed by Clarberg et al. [Clarberg et al. 2005] by hierarchical warping. As a start, we assume that the scene consists solely of diffuse surfaces and more samples are to be placed at parts of the scene with higher flux, i.e. as probability distribution we use the normalized flux.

Each level of the hierarchical warping consists of a vertical and horizontal warping step according to the flux distribution. We begin with a set of uniformly distributed sample points $s_i = (x_i, y_i)^T$, $x_i, y_i \in [0;1)$ with sample weight $w_i = 1$. Figure 5 depicts the warping according to the coarsest level: the initial sample set (Figure 5b) is partitioned into two rows according to the upper and lower average flux (Φ_{top} and Φ_{bottom}, Figure 5a). The sample points are scaled such that the separation border halves the sample area $[0;1)^2$ (Figure 5c). A new sample point's y_i' is obtained with:

$$\Phi_r = \frac{\Phi_{top}}{\Phi_{top} + \Phi_{bottom}} \tag{4}$$

$$y_i' = \begin{cases} y_i/(2\Phi_r) & \text{if } y_i < \Phi_r \\ (1 + y_i - 2\Phi_r)/(2(1-\Phi_r)) & \text{otherwise} \end{cases} \tag{5}$$

To compensate for the varying sampling densities, the sample weights are computed such that the total weight of all samples remains equal:

$$w_i' = \begin{cases} w_i/(2\Phi_r) & \text{if } y_i < \Phi_r \\ w_i/(2(1-\Phi_r)) & \text{otherwise} \end{cases} \tag{6}$$

After the vertical warping, the upper and lower halves are warped analogous to obtain x_i' for each sample (Figure 5d). The warped sampling pattern (Figure 5e) is used to sample the RSM and determine the secondary light sources. The flux taken from the RSM is multiplied with a sample's weight before its contribution to the scene is computed. We refer to this warping strategy as method A. We also experimented with a simpler and thus faster variant (method B) which performs the horizontal warping for all samples, not independently for the upper and lower half. The sample warping is done hierarchical and after a vertical and horizontal step, we proceed on the four quadrants recursively.

The importance sampling is particularly important for scenes with non-diffuse surfaces, as these require more samples to approximate their global illumination effects more accurately. We propose to perform the importance sampling not just according to the flux, but to a probability distribution obtained from the product of the flux and the maximum value of the BRDF. When using a energy-preserving Phong illumination model, the maximum value of its normalized BRDF is the Phong exponent P.

Another observation leads to a further criterion for importance sampling: the ambient occlusion term O, used in many computer games and film productions, is the ratio of environment light a surface point would be likely to receive. This is done by shooting rays into the hemisphere above a surface point and computing the ratio between rays not intersecting other surfaces and the total number of rays. Usually only occluders in near proximity are considered for computing this term. In other words: the ambient occlusion term provides information, whether other surfaces are close to certain surface point. If no other surfaces are close, secondary light sources positioned there have very little of no contribution to the scene.

To account for the aforementioned heuristics during importance warping, we replace the flux in Equation 4 by the probability: $p_\Phi = \Phi \cdot (1 - O) \cdot P$. The corresponding texture storing this term for the light view is called importance sampling buffer and replaces the RSM flux buffer for importance sampling.

3.6 Fast Rendering with Ambient Occlusion

The importance warping method described above works perfectly for two-dimensional shadow maps as used for spot-lights. For omni-directional lights most often cube shadow maps or dual-paraboloid shadow maps are used. Although an importance sampling for each individual cube side or paraboloid map can be thought of or an initial pre-warping step can be performed to distribute samples across sub-textures, this is always critical in terms of performance. Keeping in mind that not the geometry processing, that is the number of light sources, but the fragment processing is the bottleneck, we propose another procedure that proved to work well for diffuse surfaces (for all types of primary light sources, such as spot and point lights): assuming a regular sampling of the scene

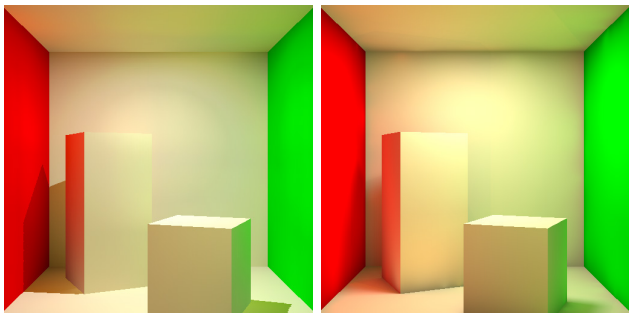


Figure 6: A comparison of our method (left) and Instant Radiosity (integrated into a ray tracer). The latter produces soft-shadows and handles occlusion for indirect lighting, but nevertheless the results are comparable. For both images we used 1000 indirect light sources generated with a quasi-random walk.

in the shadow map textures, we use a uniform distribution of light sources in the shadow map and scale the flux by the ambient occlusion term. By this, secondary light sources that are likely negligible become smaller and thus do not consume valuable processing time and memory band-width. Results from this method are presented in Figure 13 and Table 3.

3.7 Indirect Illumination without RSMs

Our method does not necessarily rely on Reflective Shadow Maps when other methods are used to determine the secondary light sources. In many game typical situations, e.g. indoor games, a space partitioning tree is stored along with the geometry. Actually these data structures are used for collision detection, but they can also be used to intersect rays from the light source with the scene geometry. Either primary rays or a quasi-random walk as for Instant Radiosity can be computed and indirect illumination can be added to the scene without computing RSMs. Figure 6 shows a rendering of the Cornell Box with Instant Radiosity.

4 Implementation

We implemented our approach using Direct3D9 and the High-Level Shader Language and programmable graphics hardware supporting the Vertex Shader 3.0 profile. It is used to read the position, direction and flux for the secondary light sources from the RSM. By using an additional render pass and so-called vertex-textures it is possible to circumvent this profile and to use our method together with older graphics hardware. If an Instant Radiosity approach is used, we need of course no RSM and the light source data can be passed as a static vertex array to the hardware.

Different passes are required to obtain the final image. When using RSMs, the task of the first pass is to render the scene as seen from the light source and store the world space position, normal and reflected flux of the visible surfaces. As in the original RSM implementation, we use a filtered texture look-up for rendering the flux in order to avoid flickering of the indirect lighting. When using non-diffuse surfaces, we also store the exponent of the Phong model for importance sampling. In order to read from RSM textures during vertex processing, contemporary hardware demands that the internal format is a 32-bit float quadruple per texel.

The next step is to create the deferred shading buffers. For this we render the scene (for the camera view) into textures with preferable low memory-consumption. The world-space position is stored together with the distance to the light source (for the shadow test) with $3+1$ 16-bit floats. The surface normals and parameters are represented by 4×8-bit textures. These dynamic textures are view dependent and need to be updated whenever the light source or the camera moves.

For the rendering of the indirect illumination, we prepare a texture which stores the pre-computed uniform sampling pattern as color-coded RGB triples. The importance sampling step – if used – takes these sampling position and a mip-mappable version of the RSM flux buffer (or the importance sampling buffer) as input and computes the new sampling positions and outputs them to another texture. This can be done by rendering a single quadrilateral and applying a fragment shader that computes the warping for each sampling individually. The mip-mappable flux or importance sampling buffer respectively has to be created from the 32-bit floating point textures, as current hardware does not support automatic mip-map generation for these.

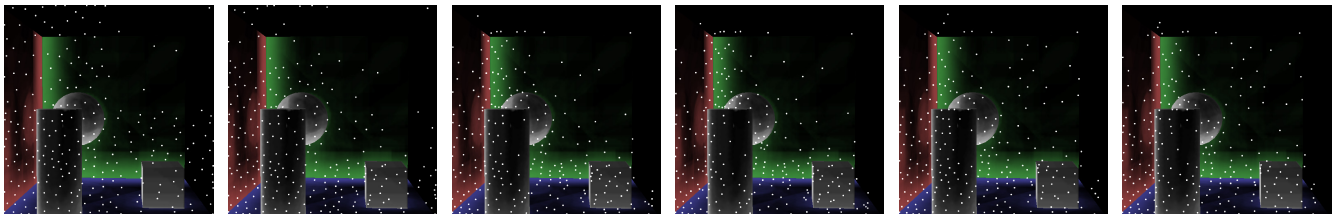
For rendering the contribution of a secondary light source, we send a sphere mesh (together with the texture coordinate, where the sampling position is found in the above-mentioned texture) to the graphics hardware. For each mesh vertex the vertex shader reads the sampling position and herewith the light source data from the RSM. Then it computes the tangent space and computes the elliptical approximation of the reflection properties. Afterwards the vertex is transformed accordingly into world space. The fragment shader then computes the contribution for each covered pixel. Please note that we compute and accumulate the contribution from all light sources to a surface point. The light-receiving surfaces themselves are assumed to be diffuse and then, in a final step, we apply the surface parameters to the accumulated contribution, e.g. multiply the accumulated light with the surface's diffuse color. The submission of the hemisphere mesh and the texture coordinates for the sample position texture can be efficiently done using instancing techniques. Nevertheless we also measured the performance using duplicated meshes (differing only in the texture coordinate) and got nearly identical timings.

5 Results and Comparison

We have tested our novel method in various, also game typical, situations. For the timings given in this section we used a GeForce 6800 GT graphics board with 350 MHz clock rate and 16 shader pipelines. Since our approach performs all computations on the GPU, the CPU has no impact on rendering performance. Unless mentioned otherwise, all images were rendered at a resolution of 512×512. Experiments showed, that the performance bottleneck is – depending on the used graphics hardware – either fragment processing or (more frequently) memory band-width. For the latter reason, we accumulate the indirect light contributions and apply the

scene update	render time (ms)
dynamic light and camera	10.4
dynamic light, static camera	10.8
static light, dynamic camera	9.3
static scene, just render final image	6.9

Table 1: Performance analysis for the caustic ring rendering (see Fig. 4, left). Importance sampling (3 levels, method B) is used for the 4096 rendered secondary light sources, the image size is 512^2. Please note that 'static camera' is slower then the first option, because no fragments are discarded during splatting of the indirect illumination by depth buffer tests. The depth buffer is only valid if the deferred shading buffers, i.e. dynamic cameras, are updated in this example setting.



A, 1 level | A, 2 levels | A, 3 levels | A, 4 levels | B, 3 levels | B, 4 levels

Figure 7: Importance sampling as described in Sec. 3.5 according to $p_\Phi = \Phi \cdot (1 - O) \cdot n$. Three or more levels provide good samplings and both methods provide comparable results.

surface colors in a final render pass. Figure 11 shows the influence of different cut-off values, that is, different bounds for determining a significant contribution by a light source, on image quality and performance. Larger values cause smaller bounding volumes and thus less fragment shader executions and memory traffic. The same effects can be concluded from Table 3 which presents results from the impact of considering ambient occlusion for the distribution of secondary light sources (results are shown in Figure 13). The time required for the computation of the RSM and the deferred shading buffers and the therewith connected render target switches can be taken from Table 1. The importance sampling duration is shown in Table 2. A comparison of RSMs and our method with different quality settings and rendering speeds is shown in Figure 12.

method	samples	importance sampling levels			
		1	2	3	4
A	32	0.123	0.238	0.541	1.739
	64	0.148	0.358	1.019	3.484
B	32	0.111	0.183	0.422	1.274
	64	0.125	0.258	0.754	2.584

Table 2: Duration of the importance warping (in milliseconds) including the necessary render target changes on a GeForce 6800 GT.

resolution of		without	with
final image	indirect lighting	ambient occlusion term	
512^2	512^2	27	37
512^2	256^2	63	85

Table 3: Rendering times (in frames per second) for the scene shown in the teaser and in Fig. 13 with and without the modulation of the 256 secondary light sources by the ambient occlusion term.

6 Discussion

Although it is hard to compare the original Reflective Shadow Map approach and our method directly, besides from subjective comparison of the result images, we can give a qualitative analysis of necessary computations. The indirect light computation and the thereby caused memory transfer has the greatest impact on performance. In this discussion, we only consider the data associated with each light source (position, direction, and flux) and the position and normal of the lit surface. The surface color can be applied after computing the light's contribution (as described in Section 5).

At first we consider the RSM approach with a full per-pixel evaluation, that is, no screen-space interpolation. Let the image resolution be P pixels, and the number of samples be S_{RSM}. Then the number of evaluations of Eq. 2 is $P \cdot S_{RSM}$ and the memory transfer is $P \cdot S_{RSM} \cdot 112$ bits. The memory footprint consists of 16 bits for reading the sample position from a texture, 48 bits for the light source position (3×16 bits float), 24 bits for the light direction and 24 bits for the flux. Note that all look-ups for this data have to be performed as slow dependent texture reads. The receiver data has only to be read once and is ignored here as it has negligible impact.

Regarding our splatting method, we quantify the cost as follows: for each image pixel, we can estimate an average overdraw. For the example shown in Figure 8 this is $O_{SII} \approx 125$. As the light's data is associated with the light's bounding geometry and is thus directly available for the fragment shader, we need to read the receiver data for each pixel, which can be done by non-dependent look-ups (48 bits position plus 24 bits normal data). The memory transfer is $P \cdot O_{SII} \cdot 72$ bits.

Of course, S_{RSM} and O_{SII} cannot be compared directly, but nonetheless we can draw some conclusions. The computational effort and above all the memory transfer for our method is significantly lower than a brute-force RSM implementation and we replace slow dependent texture look-ups by faster non-dependent ones. For RSMs the light sources are independently selected for each rendered pixel, which can cause blocky artifacts if the sample number is too low. Using our approach, the average number of indirect lights for each pixel is O_{SII}, being much lower than S_{RSM} in typical scenarios. Although we reconstruct indirect illumination from fewer samples, we achieve a visually more pleasing and coherent result, because the light source samples are constant for a whole rendered frame.

In order to achieve interactive frame rates, the RSM approach reduces the number of lighting computations by a screen-space interpolation of indirect light from a coarser image. The drawbacks are higher implementation work, more render passes and high dependency on scene complexity. Our method is sufficiently fast to do without an interpolation pass, but it would also be non-trivial: the refinements of the initial, coarse image requires the computation of indirect light for individual pixels.

Figure 8: This image shows the overdraw due to the splatting of 768 indirect lights. The average overdraw in this scenario is 125, pure white corresponds to 256 times. The average number of fragments per pixel light is $512^2 \cdot 125/768 \approx 42600$.

7 Conclusion

We presented a method for the rendering of approximate indirect lighting effects in real-time. When using our approach together with Reflective Shadow Maps all computations are completely performed by the GPU. We showed how importance sampling can be used to increase performance in the case of 2D shadow maps and presented possibilities for speed-ups when using other types of shadow maps. We presented results for both diffuse and non-diffuse surfaces and for game-typical situations. As in previous interactive methods, we cannot account for self-shadowing for indirect lighting and thus propose to use ambient occlusion methods for the rendered models.

In the future we would like to adapt the splat shape for reflective and refractive surfaces to the local surface curvature. By this, indirect illumination effects from such surfaces can be rendered with less artifacts and better approximation of sharp features.

Appendix – Bounds on Indirect Light Sources

In this appendix we describe the computation of bounds for the region that receives significant light from indirect diffuse and glossy light sources. We consider illumination significant if it is above a threshold I_{low}.

It is easiest to describe the significant region for a pixel light p in spherical coordinates centered at p, i.e. each world-space point is described by a direction ω and a distance r. The exitant light of p depends on ω, as given in Eqs. 1 and 3 and as shown in Fig. 2. The irradiance E at a point with polar coordinates (ω, r) thus depends on ω, but also on the distance to the light source r: $E(\omega,r) \propto I_p(\omega)/r^2$. E also depends on the cosine of the incident angle, but since we cannot know this in advance, we have to use the bound of one for this term. So we can bound the irradiance at (ω, r) by $I_p(\omega)/r^2$.

At this point, we can easily unify diffuse and glossy lights. We denote as α the angle to the surface normal n_p for diffuse lights (Eq. 1) and the reflection direction r_p for specular lights (Eq. 3). If we use an exponent $n = 1$ for diffuse lights, the irradiance can be bound by the 2D polar function $B(\alpha,r) = I_0 \cos(\alpha)^n/r^2$. In the following we can restrict to the 2D case, the 3D-shape is rotationally symmetric.

The region of influence of p is then bound by the isosurface $B(\alpha,r) = I_{low}$. Because B is continuously decreasing with r, we can describe this surface as polar function $r(\omega) = \sqrt{\frac{I_0 \cos\alpha^n}{I_{low}}}$. Fig. 9 shows the isosurfaces for $I_0 = 1$, $I_{low} = 1$ and Phong exponents 1, 10, and 100.

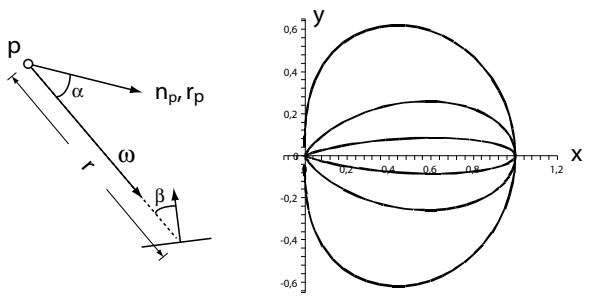


Figure 9: Left: Illumination computation due to a pixel light. Right: Significant regions for pixel lights with exponents 1, 10, and 100.

The upper half of these isosurfaces can be described by the explicit function

$$F(x) = x^{\frac{n}{n+2}} \sqrt{1 - x^{\frac{4}{n+2}}}. \tag{7}$$

We fit an ellipse around these shapes with the following heuristic. First, we compute the maximum of F. If the maximum is at x_{max}, we put the center of our ellipse at $(x_{max}, 0)^T$. We use the x- and y-axes as main axes of the ellipse. As height we select $F(x_{max})$. Finally, we select the width such that the ellipse covers the x-interval $[0,1]$. Fig. 10 shows the isosurfaces in black and the bounding ellipses in gray for $n = 1$ and $n = 10$.

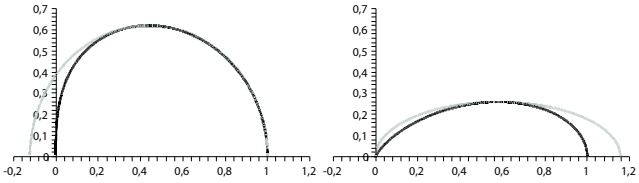


Figure 10: Isosurfaces and bounds for $n = 1$ and $n = 10$.

The maximum can be found by finding the root of F's derivative. Its x-position is at

$$c(n) = \frac{n}{n+2}^{\frac{n+2}{4}}, \tag{8}$$

thus we set the center of the ellipse to $(c(n), 0)^T$ and as its height we use $h(n) = F(c(n))$, and as width $w(n) = \max\{c(n), 1 - c(n)\}$.

References

BUNNELL, M. 2005. Dynamic ambient occlusion and indirect lighting. *GPU Gems 2 – Programming Techniques for High-Performance Graphics and General-Purpose Computation, Edited by Matt Pharr*.

CLARBERG, P., JAROSZ, W., AKENINE-MÖLLER, T., AND JENSEN, H. W. 2005. Wavelet importance sampling: Efficiently evaluating products of complex functions. In *ACM Trans. Graph.*, ACM Press, 1166–1175.

CROW, F. C. 1977. Shadow algorithms for computer graphics. vol. 11, 242–248.

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

HARGREAVES, S. 2004. Deferred shading. *Game Developers Conference*.

KELLER, A. 1997. Instant radiosity. In *Proceedings of SIGGRAPH 97*, Computer Graphics Proceedings, Annual Conference Series, 49–56.

LANDIS, H. 2002. Production-ready global illumination. *Siggraph Course Notes #16, 2002*.

REEVES, W. T., SALESIN, D. H., AND COOK, R. L. 1987. Rendering antialiased shadows with depth maps. In *Computer Graphics (Proceedings of SIGGRAPH 87)*, vol. 21, 283–291.

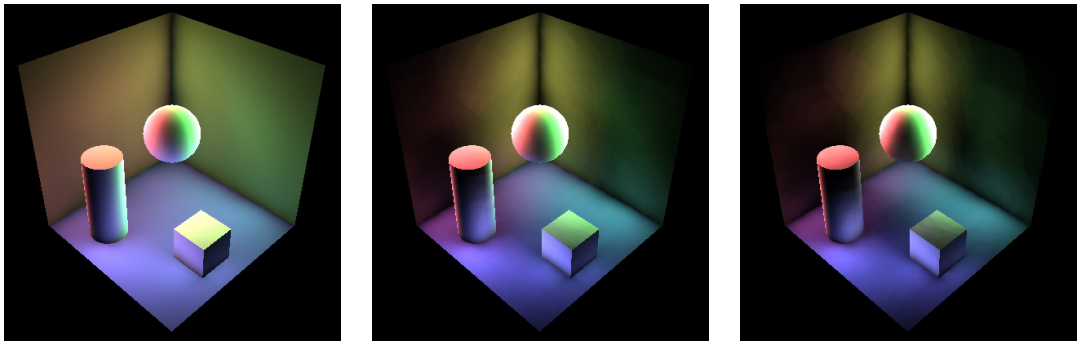
SHAH, M., AND PATTANAIK, S. 2005. Caustics mapping: An image-space technique for real-time caustics. *Technical Report, School of Engineering and Computer Science, University of Central Florida, CS TR 50-07, 07/29/2005 (Submitted for Publication)*.

TABELLION, E., AND LAMORLETTE, A. 2004. An approximate global illumination system for computer generated films. *ACM Trans. Graph. 23*, 3, 469–476.

WILLIAMS, L. 1978. Casting curved shadows on curved surfaces. In *Computer Graphics (Proceedings of SIGGRAPH 78)*, vol. 12, 270–274.

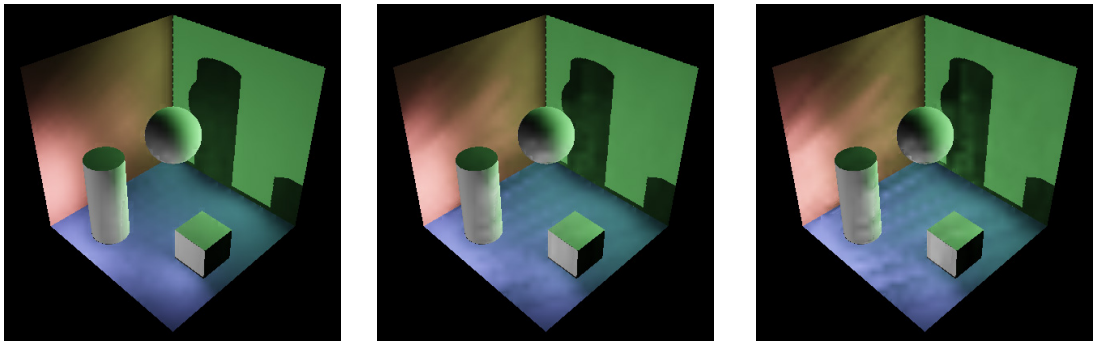
WYMAN, C. 2005. An approximate image-space approach for interactive refraction. *ACM Trans. Graph. 24*, 3, 1050–1053.

WYMAN, C. 2005. Interactive image-space refraction of nearby geometry. *University of Iowa, Department of Computer Science, Technical Report UICS-05-04*.

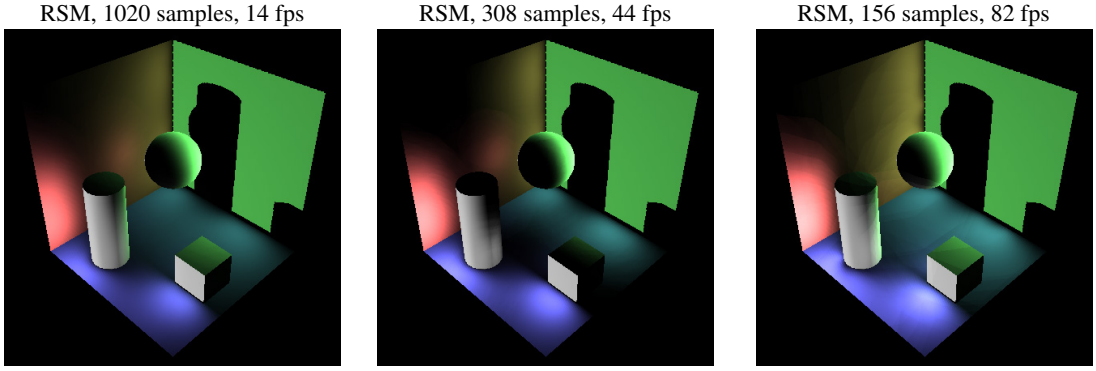


$I_{low} = 1\%I_0$, 14 fps $I_{low} = 5\%I_0$, 54 fps $I_{low} = 7\%I_0$, 74 fps

Figure 11: Different intensity cut-off values, and thus different sizes of the light's bounding volume, affect rendering speed and quality.

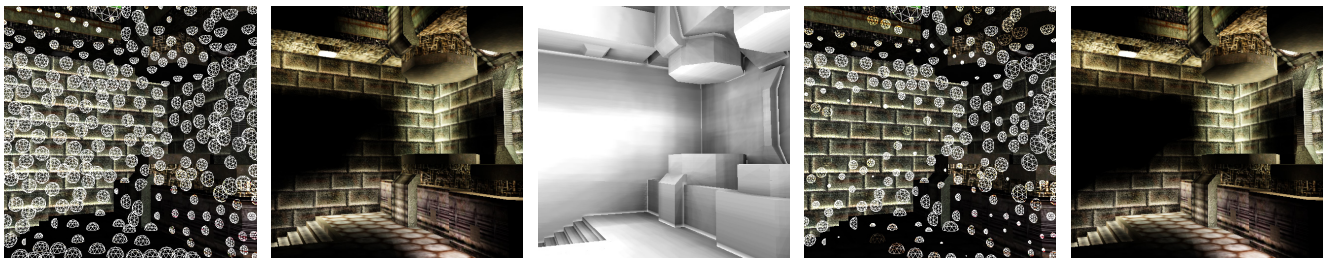


RSM, 1020 samples, 14 fps RSM, 308 samples, 44 fps RSM, 156 samples, 82 fps



SII, 768 lights, $I_{low} = 5\%I_0$, 48 fps SII, 768 lights, $I_{low} = 10\%I_0$, 83 fps SII, 108 lights, $I_{low} = 5\%I_0$, 254 fps

Figure 12: A comparison of Reflective Shadow Maps (RSM) and our method (SII): although the RSMs performance is not dependent on the distance to the secondary light sources, unfavorable lighting situations cause artifacts. Timings for the RSM images were taken with a 64^2 screen space sub-sampling. Our method provides a smooth indirect illumination and higher frame rates. Moreover, the trade-off between speed and quality does not cause severe artifacts. Timings for both methods were made without RSM or deferred shading buffer updates.



secondary light sources uniform distribution, equally weighted indirect illumination only ambient occlusion term secondary light sources modulated by ambient occlusion term indirect illumination only

Figure 13: As fragment processing and memory bandwidth are the bottlenecks, the importance sampling step can be skipped and ambient occlusion modulated light sources can be used to increase performance.