An Approximate Global Illumination System for Computer Generated Films

Eric Tabellion[*] Arnauld Lamorlette[†]
PDI/DreamWorks

Abstract

Lighting models used in the production of computer generated feature animation have to be flexible, easy to control, and efficient to compute. Global illumination techniques do not lend themselves easily to flexibility, ease of use, or speed, and have remained out of reach thus far for the vast majority of images generated in this context. This paper describes the implementation and integration of indirect illumination within a feature animation production renderer. For efficiency reasons, we choose to partially solve the rendering equation. We explain how this compromise allows us to speed-up final gathering calculations and reduce noise. We describe an efficient ray tracing strategy and its integration with a micro-polygon based scan line renderer supporting displacement mapping and programmable shaders. We combine a modified irradiance gradient caching technique with an approximate lighting model that enhances caching coherence and provides good scalability to render complex scenes into high-resolution images suitable for film. We describe the tools that are made available to the artists to control indirect lighting in final renders. We show that our approach provides an efficient solution, easy to art direct, that allows animators to enhance considerably the quality of images generated for a large category of production work.

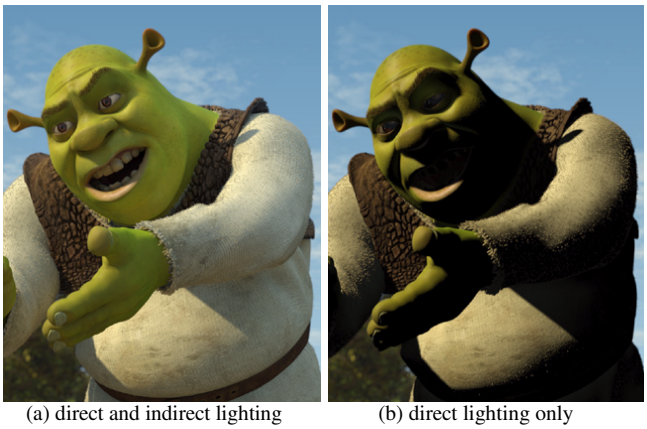
CR Categories: I.3.3 [Computer Graphics]: Picture/Image Generation; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism

Keywords: global illumination, distributed ray tracing, irradiance caching, rendering, micro-polygon

1 Introduction

Typical production renderers use many principles of the Reyes architecture [Cook et al. 1987]. They are designed primarily to handle the tremendous amount of geometry and textures used in production. These renderers tend to use techniques such as streaming and caching because most scenes will not fit into the computer's main memory. Such renderers usually offer a flexible interface through programmable shaders [Upstill 1990; Driemeyer and Herken 2003], that allows users to implement customized lighting and shading models. They also extend their feature set and shading language to offer global illumination techniques, providing strategies to ray trace complex geometry.

The special effects industry has been taking advantage of ray tracing, to physically simulate and match the look of real materials and lighting conditions [Apodaca et al. 2002]. Those techniques greatly facilitate the integration of computer-generated elements within live action plates. Comparatively, very little animation work utilizes global illumination effects in its visuals. Generally, simpler direct lighting models are used to render very complex and highly detailed scenes. As a result, important light interactions between the elements of a scene are missing. Computer animation requires repeated rendering of a very large set of images, while simultaneously pursuing a highly stylized goal. To use global illumination in this image making process, the render times have to be minimized, and tools must be provided to enable flexible art direction with a very short feedback loop.



(a) direct and indirect lighting (b) direct lighting only

Figure 1: Example of a character in outside lighting conditions. (a) and (b) were rendered respectively with and without indirect lighting.

To address this need, we have developed an indirect illumination tool, addressing critical constraints: workflow efficiency and controllability. We orient our efforts towards rendering the global illumination effects that offer the most significant visual contribution without degrading performance, using a number of techniques that speed-up calculations significantly. We only consider a coarse geometric tessellation of the geometry during ray tracing, using an appropriate biasing technique. Radiant exitance texture maps are used to accelerate final gathering and to help reduce the noise of final renders. We use irradiance caching and enhance its efficiency and applicability to non-diffuse surfaces, by using it in conjunction with an approximate lighting model. Falloff and color correction controls allow the user to make final shading adjustments. As a result, we make visually acceptable approximations, and give the artist an efficient tool that greatly enhances creative possibilities.

In this paper, we describe our rendering system and its underlying techniques. First, we outline the previous research that inspired our work. A brief overview of our system is provided, followed by a description of the approximations we make and the optimizations we apply. We describe the tool that we have developed, its usability and its workflow. Finally, images and render times resulting from using this tool during the production of an animated feature film are discussed.

2 Previous Work

Global illumination has been the subject of much research in the past two decades. Solving a broader category of light paths than radiosity algorithms, Monte Carlo path tracing was first used by

[*]e-mail: et@pdi.com
[†]e-mail: arnauld@pdi.com

Kajiya [1986] to solve the rendering equation. Although very general and simple to implement, this technique is very slow. For the solution to converge, the number of paths that need to be sampled in the direction of the light is excessive. Alternative methods such as light tracing use sample paths originating from the light sources instead. Since the number of possible paths here is also very large, efficient light tracing requires the use of probability density functions to concentrate the effectiveness of the random walk towards the camera [Dutré and Willems 1994]. A natural extension called bidirectional path tracing combines paths from the camera and from the light, statistically weighting each paired path contribution to the final image [Lafortune and Willems 1993; Veach and Guibas 1994].

Variance reduction techniques have also been highly studied, as they are beneficial for most Monte Carlo sampling algorithms. They can greatly improve rendering performance by increasing the convergence of the solution [Lafortune 1996; Veach and Guibas 1995; Jensen 95].

Veach and Guibas [1997] propose an approach called metropolis light transport, which applies mutations to previously explored paths. The goal of this technique is to exploit the path space coherence, to concentrate the computational effort on paths with great importance to the final image. Carefully weighting each path and using intelligent mutation strategies lead to an algorithm that efficiently treats lighting situations difficult to solve otherwise.

To apply those algorithms to complex scenes, Pharr and Hanrahan [1996] propose a geometry-caching framework that allows a ray tracing engine to handle scenes that do not fit entirely in the computer's memory. Pharr et al. [1997] further enhance this technique by reordering the operations of the rendering algorithm, so as to maximize caching coherence.

Orthogonal research directions have explored caching strategies to avoid recomputing similar quantities that can be extrapolated without significantly affecting the final image. Jensen [1996] proposed using photon mapping in a two pass rendering algorithm. In the first pass, photons are scattered through the scene following similar paths as in the light tracing approach, and recorded in a photon map. The second pass samples paths from the camera and reuses this information instead of recursively tracing rays towards light sources. The photon map acts as a cache of all the light paths throughout the scene, and using it appropriately significantly improves performance over previous algorithms. A clear and practical overview of photon mapping techniques can be found in [Jensen et al. 2002].

Ward and Heckbert [1992] take advantage of spatial coherence, as well as the characteristic of the irradiance function over ideally diffuse surfaces, to apply sparse sampling of the indirect illumination throughout the scene. When light gathering occurs, the resulting incoming irradiance values are cached in a spatial subdivision data structure along with irradiance gradient vectors. This information is reused to extrapolate irradiance values in between the cached samples. An error function is described, that determines if a cached sample can be reused at a given location, or if it should be disregarded. This technique greatly improves rendering efficiency since it avoids sampling indirect illumination at every pixel in the rendered image. Unfortunately the technique does not apply well and loses its benefit when considering surfaces with complex bidirectional reflectance distribution functions (BRDFs).

The optimizations described in section 4 build upon this research, following a set of approximations to a full global illumination solution.

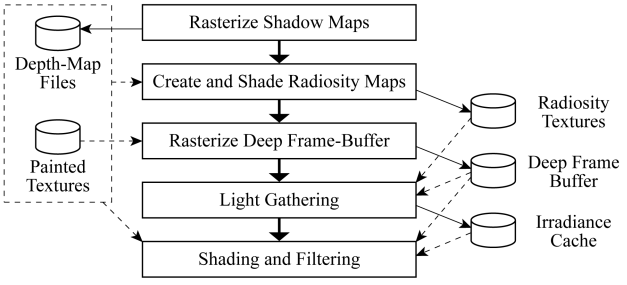


Figure 2: System overview

3 System Overview

Our system is designed to render images in five steps, as illustrated in figure 2. Each step generates information that may be reused, and stored in the appropriate files. Each step can be run in a separate process, multiple times if needed. Texture and depth-map information are loaded on demand and handled by a texture cache memory-manager.

The first three steps generate shadow maps, radiosity maps and a deep frame buffer respectively. Shadow maps are necessary to compute shadowing for direct illumination. Radiosity texture maps are used optionally to accelerate the light gathering computation, as described in section 4.4. The deep frame buffer contains all the micro-polygons visible from the camera, which are rasterized and stored for further processing.

The fourth step iterates over visible micro-polygons and computes indirect illumination using a light gathering ray tracing algorithm. It uses irradiance caching to avoid sampling indirect illumination. The final step outputs a rendered image, after shading each micro-polygon, using the approximate lighting model described in section 4.6. No ray tracing happens during final shading: only the irradiance cache is used to derive indirect illumination.

4 Approximations and Optimizations

Much of the research in the field of global illumination has been focused on efficiently solving the rendering equation [Immel et al. 1986; Kajiya 1986]:

$$L_o(x,\vec{\omega}) = L_e(x,\vec{\omega}) + \int_{\Omega_x} f_r(x,\vec{\omega}',\vec{\omega})\; L_i(x,\vec{\omega}')\; |\vec{n}\cdot\vec{\omega}'|\; d\omega_i' \qquad (1)$$

where L_e is the self emitted radiance and L_i the incoming radiance. f_r denotes the BRDF of the surface at position x, with surface normal $\vec{n}$. When mostly complex light paths contribute to the final image or when simulating indirect light cast by surfaces with complex BRDFs, the efficiency of simple algorithms degrades rapidly. Even sophisticated algorithms can become intolerably slow when computing the solution for complex lighting conditions.

Therefore, we have carefully selected the terms of the rendering equation which yield the most significant visual payoff. These approximations are exploited to speed-up rendering calculations and to improve the user workflow.

4.1 Path Length

Only indirect light paths containing a single bounce are simulated, by using a non-recursive light gathering algorithm. Indirect illumination contributions are computed by evaluating the irradiance at any given point in the scene, tracing rays over the hemisphere, and evaluating the programmable surface shader attached to each intersected surface. When invoked in that context, the shader only sums up direct lighting contributions,

preventing the recursion to happen. If radiosity maps have been pre-computed, then no surface shader is invoked and the corresponding texture map is queried instead.

This limitation requires the user to fake highly indirect lighting situations where light bounces several times before contributing to the image. In practice users can manually place artificial light sources and bouncing geometry, which simulate secondary indirect illumination and offer a more direct and separate control.

The present choice protects the user from excessive render times by restricting the framework to discard highly indirect contributions. It also provides a better workflow, by avoiding the necessity of pre-computing radiosity maps.

4.2 Surface Properties

We decide to narrow the scope of our solution further treating only diffuse surface interreflections. It is desirable to prevent surfaces from casting specular indirect lighting onto each other. Simulating these phenomena tends to require the use of more sophisticated sampling techniques without giving much visual contribution. Caustics and glossy reflections are exceptions to this rule, which we leave for more specific algorithms to solve efficiently.

When programmable surface shaders are evaluated during the light gathering algorithm, they are specifically instructed to only consider the diffuse component of their BRDF. It prevents specular interreflections to happen but captures however local color texturing which will contribute to the richness of color bleeding effects.

During final shading, this restriction would be extreme for surfaces seen directly by the camera or through ideally specular reflections. In those cases, our goal is to let indirect illumination interact with arbitrary BRDFs, while keeping the benefits and efficiency of the irradiance caching scheme. We describe an approximate lighting model in section 4.6, which achieves this goal capturing important characteristics of such interactions.

4.3 Ray Tracing Simplified Geometry

Even though valuable research has been done to efficiently ray trace complex displacement-mapped geometry, we refrain the user from doing so. Our goal is to minimize the ray tracing effort, which is the main bottleneck of the light gathering algorithm. Reordering shading calculations does not apply in our case, since irradiance caching introduces a dependency between the rays cast during the evaluation of each irradiance sample. Inspired by previous work [Rushmeier et al. 1993; Christensen et al. 2003], we make the decision to ray trace coarsely tessellated geometry, even near the ray origin.

Since rays initiate from positions that lie on displaced micro-polygons, we are faced with the problem of detecting self-intersections. Traditional biasing techniques that ignore

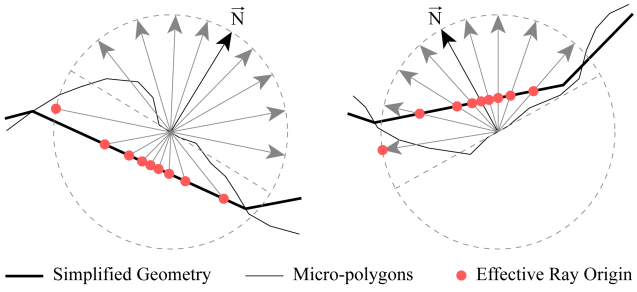
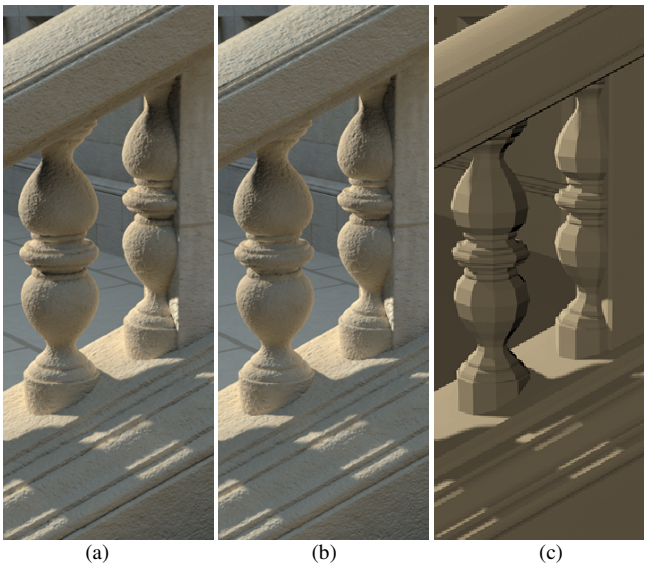


Figure 3: To ray trace simplified geometry, we adjust the ray origin.



(a) (b) (c)

Figure 4: (a) was rendered ray tracing the 2 million displaced micro-polygons seen in that figure, without using the ray offsetting algorithm. (b) was rendered using the ray offsetting algorithm, ray tracing only 4 thousand polygons, shown in image (c).

intersections near the ray origin cannot be applied here, since they would create significant light or shadow leak problems. When tracing a ray, we use the following ray offsetting algorithm:

- Record in a hit-list the ray intersections within a user-defined offset distance along the ray, after and before the ray origin.
- Stop the ray traversal once a hit is found beyond the offset distance along the ray.
- Find the closest hit to the ray origin in the hit-list, within the offset distance. If found, let this intersection become the new effective ray origin. Otherwise, leave the ray origin unchanged.
- Return the next hit in the hit-list as the resulting intersection.

Figure 3 shows two examples and illustrates how the effective ray origin is adjusted. To prevent self-intersection artifacts, the offset distance used in this algorithm needs to be bigger than the maximum offset between the coarse and micro-polygon geometric tessellations. Since we are ray tracing approximate geometry, diffuse self-interreflections cast by geometric micro-displacements might not be captured accurately. This is often visually of small importance, as illustrated in figure 4, since the highly detailed surface normal is considered when sampling the hemisphere.

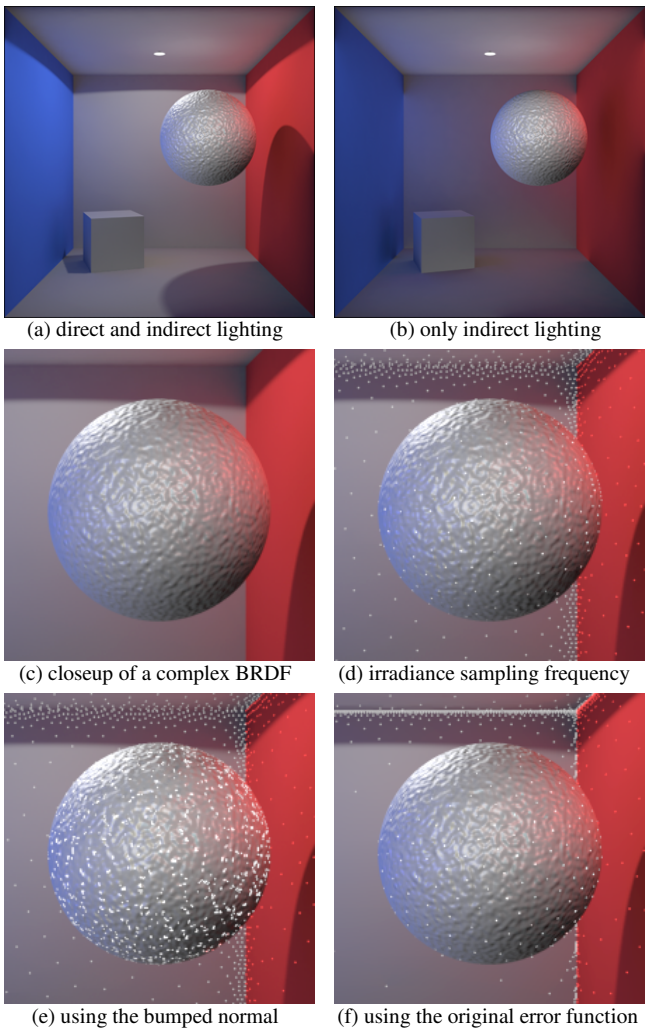
In our system, users can adjust tessellation rates suitable for ray tracing. This tune-up is done per character, prop or environment once and for all. Every shot receives geometry with good default tessellation, which rate can be modified in specific shots if needed (e.g. extreme closeups). The trade-off between object detail and polygon count can therefore be controlled manually. Using solid angle based tessellation was not implemented but would be a valuable extension to our system.

4.4 Radiosity maps

Another bottleneck of the light gathering algorithm is computing a radiance estimate for each ray. Arvo [1986], Heckbert [1990] and many others [Jensen 1996; Christensen 2000] exploit this idea. We opt for a similar strategy using texture maps, which offer a constant time query and take advantage of the texture management engine of the rendering infrastructure.

A texture image is mapped onto each surface in the scene. During a pre-processing stage, we compute and store radiance values for each texel. Since we simulate a single bounce of indirect light, our radiosity maps contain only direct illumination contributions. The maps can be computed by simply running the surface shader for every pixel of each texture. The texture mapping function needs to be inverted to evaluate the surface position and normal for each texel. In our case, we invert the texture coordinates of the four corners of the texel and obtain a corresponding quadrilateral on the surface. We then evaluate the surface shader, capturing the direct lighting, shading and texturing, applying anti-aliasing over the quadrilateral area.

The textures do not require very high-resolution to capture sufficient detail, and using lower resolution textures will work to our advantage. This choice will average high frequency indirect lighting information and greatly reduce the noise in the final solution. It is possible to use ray differentials [Igehy 1999] and mip-mapping techniques to benefit from appropriate filtering regardless of texture sizes. In our implementation, the texture maps resolutions are computed to be proportional to each surface size and aspect ratio. That metric can be computed in world space or in screen space based on what works best in each case. This choice is left to the user, who can also adjust a global resolution multiplier parameter. Screen space was used most often as it would automatically scale the textures proportionally to their screen coverage and provide a good automatic default. This method also assigns very low resolution textures to objects far in the distance from the camera's point of view. However the technique fails to provide consistent texture sizes in cases of extreme camera motion or when objects are outside the camera frustum.



(a) direct and indirect lighting (b) only indirect lighting

(c) closeup of a complex BRDF (d) irradiance sampling frequency

(e) using the bumped normal (f) using the original error function

Figure 5: A bouncing ball in a cornell box rendered with equation (8), using 1000 rays per irradiance sample. (a) and (b) show that most of the lighting on the sphere is indirect. (c) shows a closeup of a bump-mapped complex BRDF. (d) shows the irradiance samples as white pixels when using equation (5) with $\kappa = 0.5$. (e) is similar to (d) but uses a bumped normal in equation (4). (f) uses the error function described in [Ward and Heckbert 1992]. The accuracy was adjusted so that image (d) and (f) contain the same number of irradiance samples. Image (f) shows a higher sample density in corner areas.

Notice that in our context, the radiosity maps can be computed efficiently with no perceptible noise. When simulating multiple bounces of light, the maps can be derived using a light tracing algorithm, or can be replaced by an alternate caching structure such as a photon map. Doing so requires spending substantially more compute time shooting enough light samples to eliminate noise in the maps, which translates otherwise into temporal noise in the final render. To reduce visual artifacts in corner areas, it has also been suggested to disregard the caching structure and use recursive distributed ray tracing when ray intersections are arbitrarily close to the ray origin. Our restricted framework doesn't suffer from these problems, which, in turn, greatly contributes to reduced final render times.

4.5 Irradiance Caching Coherence

The benefits of the irradiance gradient caching technique are very valuable to us [Ward and Heckbert 1992]. During the light gathering step, we compute irradiance samples located on micro-polygons that may be displaced. Every irradiance sample is saved in a cache along with corresponding geometric information and irradiance gradient vectors. This information is used to smoothly extrapolate irradiance values for neighboring positions.

We adjust the technique to enhance caching coherence for scenes containing high geometric detail and complex surface properties. To compute the error of using a sample i at any position x, we use the following function instead:

$$\varepsilon_i(x,\vec{n}) = \kappa \cdot \max\left(\varepsilon_{pi}(x), \varepsilon_{ni}(\vec{n})\right) \tag{2}$$

where

$$\varepsilon_{pi}(x) = \frac{\|x - x_i\|}{\max\left(\min\left(\frac{R_i}{2}, R_+\right), R_-\right)} \tag{3}$$

and

$$\varepsilon_{ni}(\vec{n}) = \frac{\sqrt{1 - \vec{n} \cdot \vec{n}_i}}{\sqrt{1 - \cos(\alpha_+)}} \tag{4}$$

κ controls the general accuracy of the algorithm, and is the only parameter that the user can modify. κ was set to 1 in most cases, and α_+ was hard-coded to 10 degrees. R_i is the closest distance to the intersected surfaces during the evaluation of the irradiance sample i. At each pixel we compute the area of that pixel projected onto the plane passing through x with normal $\vec{n}$ using the camera projection. We then let R_+ and R_- be respectively 10 and 1.5 times the square root of the projected pixel area. These values can easily be derived using the solid angle of the pixel through which x is seen.

To avoid sampling the irradiance, we loop over candidate cached samples with positive weight and compute the weighted average of their extrapolated contribution. The weight of each sample is computed using the equation below:

$$w_i(x,\vec{n}) = 1 - \varepsilon_i(x,\vec{n}) \tag{5}$$

This model nicely correlates the maximum distance to candidate cached samples with the scale of the scene and the amount of detail desired in the final render. The sampling density is adjusted automatically, avoiding sampling patterns that are too sparse or too dense. Figure 5 (d) and (f) illustrate the sampling frequency that is achieved when using equation (5) and the original error function described in [Ward and Heckbert 1992] respectively. They have both been rendered using the same number of irradiance samples. The error function presented above avoids over-sampling the irradiance in corner areas, when compared to the original formulation. It will therefore scale better with highly detailed geometry that contain more corners or creases and less flat surfaces.

In addition, we avoid over-sampling the irradiance due to surface normal perturbations by using a surface normal that has not been altered by bump mapping when computing equation (4). Comparing figure 5 (d) and (e) illustrates the corresponding gain in caching coherence. To speed-up the workflow, users can also disable displacement mapping during interactive lighting sessions, which increases the coherence similarly.

In specific cases that offer great payoff, we allow procedural geometry shaders to enhance the irradiance caching coherence. The high geometric frequency of fur, for instance, completely degrades the efficiency of this interpolation scheme. However, if the fur is short, it can be interesting to apply the same indirect illumination to the fur as to the surface growing it. In this case we allow the geometry shader to specify the location where light gathering should be performed. For each fur fragment, the geometry shader has the option to force indirect illumination queries to occur at the root of the corresponding hair. This greatly enhances caching coherence along and across each hair while producing consistent and acceptable lighting as seen in figure 9.

4.6 Approximate Lighting Model

We propose a simplified lighting model that enables the irradiance caching coherence described in the previous section, while preserving some of the properties of complex BRDFs.

During the final shading step, we apply the extrapolated irradiance to the local surface shader, by converting the irradiance to corresponding approximate incoming field radiance. We choose a dominant incoming light direction $\vec{\omega}'$, and we assume the result of the irradiance to be coming fully from this direction. Since we assumed an ideally diffuse surface property when we sampled the irradiance, we compute the radiance as follows:

$$L_i(x,\vec{\omega}') = \frac{E(x,\vec{n})}{|\vec{n}\cdot\vec{\omega}'|} \tag{6}$$

where $E(x,\vec{n})$ is the extrapolated irradiance at position *x* and is computed as described in section 4.5. This information is then passed to the surface shader to let it sum this contribution applying its full BRDF. Each time the irradiance is sampled during the light gathering step, we derive a dominant incoming light direction $\vec{\omega}_i'$ using the weighted average of the ray directions used in sampling the hemisphere. The luminance of each ray's radiance estimate is used in the weighting process. This provides a direction of dominant incoming indirect light, which varies smoothly across a surface. The result is stored in the cache along with its corresponding irradiance sample, and averaged at any position *x* using weights described in equation (5):

$$\vec{\omega}'(x,\vec{n}) \approx \frac{\sum_{i,w_i>0} w_i(x,\vec{n})\,\vec{\omega}_i'(x_i,\vec{n}_i)}{\sum_{i,w_i>0} w_i(x,\vec{n})} \tag{7}$$

As it is possible to capture enhanced chromaticity effects by applying this technique separately for each wavelength, we choose three dominant incoming light directions instead. Each of the three directions is the result of weighting sample ray directions by the radiance estimate's color components red, green and blue respectively. The radiance for each specific wavelength is then applied coming from its corresponding dominant direction.

This yields the following approximate rendering equation:

$$L_o(x,\vec{\omega}) = L_e(x,\vec{\omega}) + \sum_k f_r(x,\vec{\omega}_k',\vec{\omega})\; L_d(x,\vec{\omega}_k')\;|\vec{n}\cdot\vec{\omega}_k'| + \sum_{\lambda=R,G,B} f_r(x,\vec{\omega}_\lambda',\vec{\omega})\; L_{i\lambda}(x,\vec{\omega}_\lambda')\;|\vec{n}\cdot\vec{\omega}_\lambda'| \tag{8}$$

$L_{i\lambda}$ is computed using equation (6) and $\vec{\omega}_\lambda'$ using equation (7). L_d denotes the direct lighting contribution from a traditional point light source k and is evaluated using the corresponding programmable light shader.

While physically incorrect, this indirect lighting model is an efficient way to apply the irradiance gradient caching technique to surfaces with arbitrary BRDFs. This technique requires storing a little extra information in the irradiance cache and reduces considerably the number of rays required to capture dominant directional effects on indirectly lit surfaces. Figure 5 (c) was rendered using equation (8), which enabled using a sparse distribution of irradiance samples, as illustrated in figure 5 (d). Rendering the sphere in that figure would otherwise require sampling the irradiance at each pixel.

Although this method has limitations when the average is not a good heuristic to compute the dominant light direction, it has been applied to a wide range of practical cases without causing artifacts.

5 Tool Description

We discuss the integration of the algorithms and concepts presented in previous chapters within a proprietary production renderer and a framework of existing shaders. The rendering workflow is presented as well as a description of the many controls available to the user.

5.1 Shader Integration

The indirect illumination is available to the user as a new type of light shader that can be used on any surface. In our framework, surface shaders can be written independently of the desired lighting effects. The light shaders are responsible for providing incoming light directions and colors. Since our lighting model fits this framework, its application does not require pre-existing surface shaders to be modified. The indirect lighting contributes automatically to complex surface effects such as subsurface scattering.

In comparison, some rendering systems require the surface shader to explicitly invoke indirect lighting samples. This choice permits a surface shader to easily apply importance sampling using properties of its BRDF. Since this is not a requirement in our approximate model, it is more flexible for us to alleviate the surface shaders from that task. In cases where it is needed, a more generic interface allows surface shaders to provide a description of their BRDF [Slusallek and Seidel 1995].

The indirect light shader provides only indirect illumination. The separation from the direct illumination is important as it allows independent control and adjustment of each contribution. It also lets each type of light shader be responsible for solving a different problem, using an appropriate sampling strategy [Ward

(a) using a single bounce of indirect light

(b) using multiple bounces of indirect light

Figure 6: Example of a character in interior lighting conditions. Highly indirect lighting contributions can efficiently be rendered and easily controlled, by letting the artist place lights and bouncing geometry that approximate these effects. Image (a) was rendered simulating a single bounce of indirect light, while image (b) was rendered for comparison with multiple bounces.

1995]. An area light, for instance, is modeled using many virtual point light sources scattered over its surface. Ideally specular reflections and refractions are treated similarly, after tracing additional rays.

5.2 Workflow

In our rendering system, the deep frame buffer can be shaded multiple times using an interactive lighting tool. This allows reshading without spending any resources processing and scan-converting the geometry. This works well in a typical workflow, where the animation is established and the lighter need only focus on lighting a few fixed frames.

The lighting tool can also trigger the light gathering pass prior to shading. Since we make use of irradiance caching, two passes are required so that each pixel extrapolates irradiance from cached samples located above and below the current scan line in the image plane. When gathering and shading are performed in a single pass, pixels only extrapolate irradiance from cached samples located on one side of the scan line, which produces unacceptable visual artifacts. Yet, we let the gathering pass provide visual feedback to the user by also shading the image using low quality settings. At the end of the gathering pass, the irradiance cache is saved into a file for future reuse. When the final shading pass happens, only the irradiance cache is used to derive indirect illumination.

The image can be reshaded many times, without having to trace a single ray as long as the indirect illumination remains unchanged. If that is not the case, the user can specifically request use of both gathering and shading passes. This is an important aspect of the workflow, as it provides a way to keep adjusting most of the shader settings, while getting quick visual feedback. In many cases, changes to the direct lighting will have an impact on the indirect lighting. Still, the user can adjust the former many times without triggering a gathering pass. After several iterations and adjustments, the lighter can then fully update the solution.

Also of importance, is the ability to perform light gathering without needing to pre-compute radiosity maps or any alternate caching structure. Our choice to only simulate one bounce of indirect light allows us to do so without excessively degrading interactive rendering performance. In our case, lighters can indeed go through initial adjustments without using the radiosity maps caching technique. This allows immediate visual feedback which speeds up interactive lighting turnaround. The maps are generated and used mainly to optimize the rendering of film-resolution images over the full length of a shot.

Since we have developed an infrastructure to cache shading information into texture maps, we can take advantage of it to precompute and cache static indirect lighting over an entire shot or sequence. That indirect lighting is then made available to the shot as yet another texture map in the scene. Alternatively, we can also reuse the irradiance cache over several frames. Each new frame will compute light gathering only in regions of the image that have not been covered in previous frames. The new irradiance samples are added to the existing cache, which is then used to render the next frame. This approach has not been used in practice as it imposes rendering each image in a shot sequentially, which is incompatible with our strategy of rendering images in parallel using a render-farm.

5.3 Art Direction

Even though a physically based algorithm is used to compute indirect illumination, its result might differ from the artist's aesthetic goal. A hard constraint is imposed, as no object in the scene can be moved to alter the indirect lighting. It would otherwise require changing the layout or the animation, which in most cases has already been determined and finaled. We give here an overview of the controls that are made available to the user.

The user can control which objects in the scene will receive indirect illumination and define the list of all the geometries that need to be ray-traced when gathering light. The user can also redefine which surface shader is invoked to compute the radiance estimate when intersecting a given geometry. This provides great flexibility, and in extreme cases allows defining the direct lighting and the indirect lighting completely independently. The user is able to replace complex geometries by simple bounce planes, to control indirect lighting more easily. It also alleviates the ray tracing engine and optimizes rendering times.

Most shader types inherit from a set of standard parameters that affect the indirect lighting calculation. Surface shaders, for instance, have controls that enable altering color bleeding that occurs between surfaces. Adjustment of the incoming indirect lighting per-surface is also possible. Direct light shaders also inherit parameters that let the user specify which direct lights need to be considered in the indirect lighting calculations. A direct light can potentially be considered only for indirect illumination, but not cast any direct lighting in the final image.

To let the user apply geometric falloffs, we provide a filter light shader, which references a specific indirect light shader in the scene. The filter light provides a filtered version of the indirect illumination, and is manipulated similarly to a spot light, defining

(a) reference (b) saturated (c) warm (d) directional

Figure 7: Example of adjusmtents of the indirect illumination, using a filter light shader. Image (a) is the reference image. Image (b) was rendered increasing the saturation of the indirect lighting. In image (c) the hue of the indirect light was adjusted to provide a warmer feel. The overall brightness on the character's face was increased using geometric falloffs. In image (d) the directionality was increased to emphasize the lighting coming from the right. Each image took 40 seconds to reshade at video resolution.

a position, orientation, and cone angle. It offers a set of artificial cone and distance falloffs, as well as a directionality falloff, that allows the indirect lighting to vary as a function of the angle θ between the surface normal and the filter light direction. We compute it as follows:

$$f(\theta) = [\cos(\theta / 2)]^d \quad (9)$$

In this model, surfaces facing the filter light will receive more indirect lighting than surfaces facing away. d is a user parameter that controls how fast the falloff applies.

The filter light offers many color correction controls allowing adjustments of brightness, contrast, and saturation, as illustrated in figure 7. Using them in conjunction with the proper falloffs has revealed to be a powerful indirect lighting art direction tool. Since adjusting those settings only requires re-evaluation of the shading pass, this provides an efficient workflow in which final color and intensity adjustments can be made quickly.

6 Results

This tool was used extensively by artists during the production of the computer animated feature film "Shrek 2". It was used to light primary and secondary characters in 78% of the shots, and to light props and environments in 30% of the shots. Following is a rendering example, and a discussion of the quantitative aspects of using this framework.

Figure 1 shows an image in which the foreground character has been lit using direct and indirect lighting. The character reflects light onto itself. Image (a) shows the final render, while image (b) shows the same render without indirect lighting. The images were rendered on a Pentium IV running at 2 GHz, using 2 GB of physical memory. The images were rendered at film resolution for final renders (1828 x 1102 pixels), and at a third of that resolution

	Film Resolution High Quality	Video Resolution High Quality	Video Resolution Low Quality
Shadow-maps	0:01:33	0:00:31	0:00:22
Radiosity maps	0:03:20	0:01:37	-
Deep Frame-Buffer	0:03:03	0:01:00	0:00:10
Gathering	0:42:59	0:12:51	0:03:44
Shading	0:17:00	0:03:37	0:00:28
Total	**1:07:55**	**0:19:36**	**0:04:44**

Figure 8: Render times for a foreground character at film and video resolution. The right column shows timing using low quality settings during interactive work. Times are given in hours:minutes:seconds.

for preview work. When working interactively in the lighting tool, low quality settings were used to sample the irradiance, the texture maps, and the shadow maps. In addition, displacement mapping and fur were disabled, and a lower micro-polygon tessellation rate was used. Figure 8 shows the render times for the layer containing the foreground character in figure 1 (a).

The radiosity texture maps were not used during interactive work to allow for a faster adjustment and visual feedback cycle. The textures were only used during batch rendering and the resolutions were computed using a screen space metric. For example, the texture map covering the character's belly was 145 by 118 pixels.

The geometry generated over 7.8 million micro-polygons, while the simplified ray-traced geometry totaled only 49 thousand polygons. The ray-traced geometry and acceleration structure only used 23 MB of memory. Ray tracing required neither swapping nor geometry caching on disk.

The gathering step used 600 rays per irradiance sample at high quality settings, and 50 rays at low quality settings. In cases where the indirect illumination was very uniform, settings as low as 100 were enough to eliminate the noise. In more complex scenes or when the indirect illumination came primarily from smaller areas, settings up to 2000 were required. The average across all shots of the movie was 420 rays per sample with a standard deviation of 250.

The irradiance cache contained 68615 irradiance samples at film resolution, compared to 22301 at video resolution and 10085 using low quality settings. The caching error function provides on average 3x scalability from video to film resolution, and an additional 50% speed-up in this case, when displacement mapping is disabled. The render times given in figure 8 were measured while rendering the full image viewport. However, when users interactively update specific regions of the image, the gathering and shading times are proportionally lower.

7 Conclusion

We have presented a set of techniques that allow animators to render and control approximate global illumination effects that were once considered too expensive and hard to control. Our algorithms improve final gathering render times by ray tracing only simplified geometry and using texture maps to cache parts of the rendering equation. They also increase the spatial coherence and range of application of the irradiance gradient caching technique using an approximate lighting model. We have exposed the integration of our algorithms within a production renderer and its programmable shaders. We have seen how artists are able to

control many aspects of the indirect lighting calculation within a workflow that provides a fast feedback loop.

Without such a tool, artists would otherwise need to spend significant time, placing and animating direct lights, to try to achieve similar results. In some cases, indirect light interactions are very difficult or nearly impossible to reproduce using only direct lighting techniques. As characters or elements are animated through a lit environment, the indirect lighting contributions are automatically updated. We found that this system enhances the image making process because a quick setup phase can provide a solid basis early on. The artist is able to spend more time designing and adjusting a lighting strategy as opposed to attempting to simulate complex interaction effects.

We plan to extend our work in several challenging directions, by providing additional light bounces to our solution, without introducing obstacles in the user's workflow. We would also like to enhance further the spatial coherence by using undisplaced surface positions and normals in the irradiance caching error function during final renders. Finally, we would like to explore new ways of storing additional information in the irradiance cache, to broaden its applicability.

8 Acknowledgements

We wish to specially acknowledge Ken Bielenberg, Michael Day, Philippe Denis, Vanitha Rangaraju, Taylor Shaw and David Hart for extensively participating in designing and refining the tool we have described. We would like to thank Deepak Tolani for writing the code to invert the texture mapping functions. Many thanks to Karl Johann Schmidt, Jonathan Gibbs, Doug Cooper, Dave Walvoord and Mike McNeill for their valuable input during many of the experimentation phases of this project. We would like to acknowledge the anonymous reviewers, as well as Olivier Maury, Eduardo Bustillo, Hector Yee, Reid Gershbein, Rachel Falk and Jonathan Simonoff for their suggestions. We wish to thank PDI/Dreamworks for supporting this work.

References

APODACA, T., QUARONI, G., BREDOW, R., GOLDMAN, D., LANDIS, H., GRITZ, L., AND PHARR, M. RenderMan in production. In *SIGGRAPH'2002, Course #16*, San Antonio, July.

ARVO, J. 1986. Backward ray tracing. In Course Notes of the 1986 Conference on Computer Graphics and Interactive Techniques, no. 12, (Dallas, Texas, Aug. 18--22). ACM.

CHRISTENSEN, P. H. 2000. Faster Photon Map Global Illumination. Journal of Graphics Tools, volume 4, number 3, pages 1-10. ACM.

CHRISTENSEN, P. H., LAUR, D. M., FONG, J., WOOTEN, W. L., AND BATALI, D. 2003. Ray Differentials and Multiresolution Geometry Caching for Distribution Ray Tracing in Complex Scenes. *Computer Graphics Forum (Eurographics 2003 Conference Proceedings)*, pages 543-552. Blackwell Publishers.

COOK, R. L., CARPENTER, L., AND CATMULL, E. 1987. The Reyes image rendering architecture. In Maureen C. Stone, editor, *Computer Graphics (SIGGRAPH '87 Proceedings)*, pages 95–102, July.

DUTRÉ, P., AND WILLEMS, Y. D. 1994. Importance-driven Monte Carlo Light Tracing. In *Proceedings of 5. Eurographics Workshop on Rendering*, pages 185–194, Darmstadt.

DRIEMEYER, T., HERKEN, R. 2003. *Programming mental ray*. Second, revised edition. Springer Verlag Wien New York.

HECKBERT, P. S. 1990. Adaptive Radiosity Textures for Bidirectional Ray Tracing. *Computer Graphics* **24** (4), pages 145–154.

IGEHY, H. 1999. Tracing ray differentials. *Computer Graphics*, 33(Annual Conference Series):179–186.

IMMEL, D. S., COHEN, M. F., GREENBERG, D. P. 1986. A radiosity method for non-diffuse environments. *Computer Graphics* **20** (4), pages 133–142.

Figure 9: Example of a furry character. The fur receives the same indirect lighting as the skin. Only a rough tessellation of the character skin is ray traced.

JENSEN, H. W. 1995. Importance driven path tracing using the photon map. *Rendering Techniques '95 (Proceedings of the Sixth Eurographics Workshop on Rendering)*, pages 326–335. Springer Verlag.

JENSEN, H. W. 1996. Global illumination using photon maps. *Rendering Techniques '96 (Proceedings of the Seventh EurographicsWorkshop on Rendering)*, pages 21–30. Springer Verlag.

JENSEN, H. W., CHRISTENSEN, P. H., KATO, T., AND SUYKENS, F. 2002. A Practical Guide to Global Illumination using Photon Mapping. In *SIGGRAPH'2002, Course #43*, San Antonio, July.

KAJIYA, J. T. 1986. The Rendering Equation. *Computer Graphics* **20** (4), pages 143–149.

LAFORTUNE, E. P. AND WILLEMS, Y. D. 1993. Bidirectional Path Tracing. In *Proceedings of CompuGraphics*, pages 95–104.

LAFORTUNE, E. P. 1996. *Mathematical Models and Monte Carlo Algorithms for Physcially Based Rendering*. Ph.d. thesis, Katholieke University, Leuven, Belgium 1996.

PHARR, M., AND HANRAHAN, P. 1996. Geometry caching for ray tracing displacement maps. In Xavier Pueyo and Peter Schröder, editors, *Eurographics Workshop on Rendering*, pages 31–40.

PHARR, M., KOLB, C., GERSHBEIN, R., AND HANRAHAN, P. 1997. Rendering complex scenes with memory-coherent ray tracing. In *SIGGRAPH 97 Conference Proceedings*, pages 101–108, August.

RUSHMEIER, H., PATTERSON, C., AND VEERASAMY, A. 1993. Geometric simplification for indirect illumination calculations, in Proc. Graphics Interface '93, (Toronto, Ontario) , pp. 227--236.

SLUSALLEK, P., SEIDEL, H. P. 1995. Vision: An Architecture for Global Illumination Calculations, in *IEEE Transactions on Visualization and Computer Graphics*, 1(1), pp. 77-96, March.

UPSTILL, S. 1990. *The Renderman Companion*. AddisonWesley, 1990.

VEACH, E., AND GUIBAS, L. 1994. Bidirectional Estimators for Light Transport. In *Proceedings of the 5th Eurographics Workshop on Rendering*, pages 147–162.

VEACH, E., AND GUIBAS, L. 1995. Optimally Combinig Sampling Techniques for Monte Carlo Rendering. *Computer Graphics* **29** (4), pages 419–428.

VEACH, E., AND GUIBAS, L. 1997. Metropolis Light Transport. *Computer Graphics* **31** (3), pages 65–76.

WARD, G., AND HECKBERT, P. 1992. Irradiance gradients. *Third Eurographics Workshop on Rendering*, pages 85–98.

WARD, G. 1994. The RADIANCE Lighting simulation and Rendering System. In *Computer Graphics*, pages 459--472, July.