Vision, Modeling, and Visualization (2010)
Reinhard Koch, Andreas Kolb, Christof Rezk-Salama (Eds.)

# Global Illumination using Parallel Global Ray-Bundles

Jan Hermes[1] Niklas Henrich[2] Thorsten Grosch[3] Stefan Mueller[2]

[1] RTT (Realtime Technology), Germany
[2] University of Koblenz, Germany
[3] University of Magdeburg, Germany

**Abstract**

*A fast computation of unbiased global illumination is still an unsolved problem, especially if multiple bounces of light and non-diffuse materials are included. The standard Monte Carlo methods are time-consuming, because many incoherent rays are shot into the scene, which is hard to parallelize. On the other hand, GPUs can make the most of their computing power if the problem can be broken down into many parallel, small tasks. Casting global, parallel ray-bundles into the scene is a way of achieving this parallelism. We exploit modern GPU features to extract all intersection points along each ray within a single rendering pass. Radiance can then be transferred between pairs of all points which allows an arbitrary number of interreflections, especially for compelling multiple glossy reflections. Beside arbitrary BRDFs, our method is independent of the number of light sources and can handle arbitrary shaped light sources in a unified framework for unbiased global illumination. Since many methods exist for fast computation of direct light using soft shadows, we demonstrate how our method can be built on top of any direct light simulation.*

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation—

## 1. Introduction

The main goal in photorealistic rendering is to solve the rendering equation [Kaj86]. Solving it, or finding a good approximation is crucial to simulate global illumination. While the rendering of the direct illumination and even simple shadows is well understood and can be done in real-time, evaluating the indirect illumination remains a challenge for current real-time rendering systems.

The major approaches for generating realistic images to solve this problem are algorithms based on finite elements [GTGB84], photon mapping [Jen01] or Monte Carlo methods [CPC84, Kaj86, LW93]. One of the main problems of computing a global illumination solution with a Monte Carlo approach is that the rays cast into the scene become very incoherent after the first hit. This incoherence of rays can be avoided by using global ray-bundles [Sbe96]. It was shown that casting local rays from every point in the scene is equivalent to casting bundles of parallel global rays into random directions (see Fig. 2). Instead of using ray-tracing, casting global ray-bundles and thus exploiting the coherence of these rays can be done very efficiently using the z-buffer hardware of current GPUs. Depth peeling [Eve01] is used to extract the hit-points along a global ray direction using rasterization. Recently, Bavoil and Myers showed that the linear complexity of depth peeling can largely be avoided with the help of a *k-buffer* [BM08a]. A *k-buffer* is capable of storing up to $k$ fragments per pixel (where $k$ is bound by the multisampling resolution of the graphics hardware) with one geometry pass. Each stored fragment corresponds to a depth-layer of the scene. This is contrary to previous depth-peeling techniques which needed $n$ geometry passes to extract $n$ depth layers.

We propose to combine global ray-bundles and the k-buffer with an atlas-based representation of the scene to progressively compute an accurate global illumination solution. Our method is capable of exchanging radiance along multiple global ray-directions in a fraction of a second. The algorithm supports diffuse as well as glossy scenes. We propose to reuse the stored indirect illumination information in the atlas to construct light paths of arbitrary length with nearly

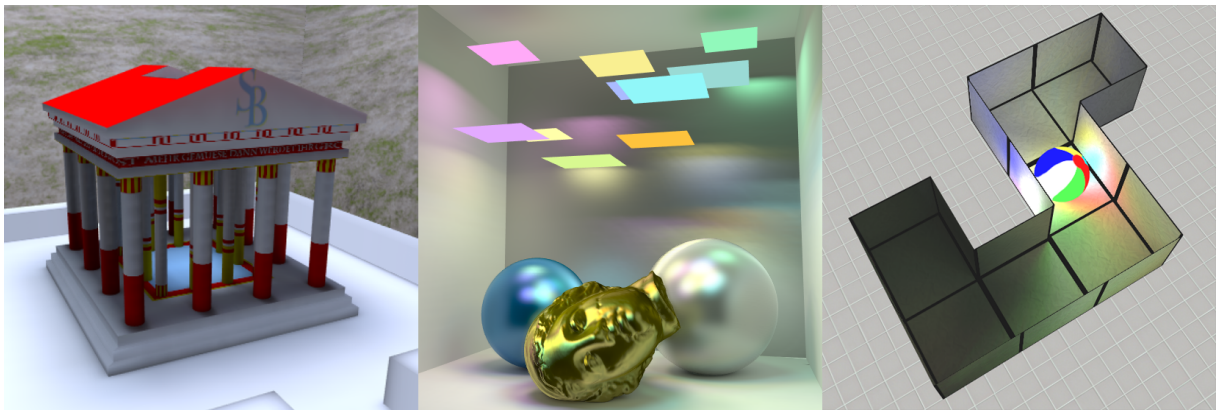

**Figure 1:** *Global illumination computed with global ray-bundles: Multiple diffuse bounces originating from natural illumination (left), glossy reflections and arbitrary number of light sources without additional cost (center) and multiple bounces of a non-planar, textured light source (right). The computation time is 2-3 minutes for all images. Standard Path Tracing remains noisy for the given scenes at the same computation time.*

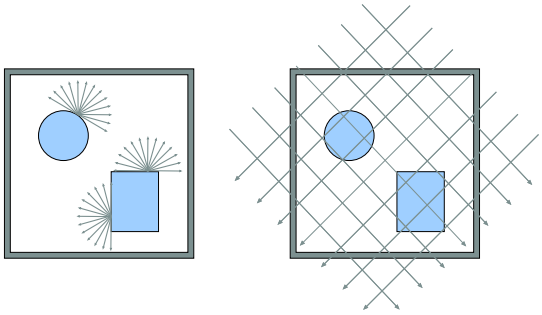

**Figure 2:** *Instead of casting random rays from different positions into the scene (left), bundles of parallel rays can be used (right). Both methods yield the same result [Hac05].*

no additional costs. In contrast to other methods, the light transport for all samples of the scene, and not just the visible ones, are computed at once. This allows the user to navigate through the converged diffuse scene in real-time. Furthermore, we propose to use global ray-bundles to sample arbitrary shaped light sources. This allows for sampling any number of complex light sources in constant time and results in very accurate near-field illumination and shadows. No special data structures are needed, but a texture atlas which is readily available for interactive applications, like games. As our method does not rely on any pre-calculations, and due to the progressive nature of our approach, dynamic scenes are possible as well.

This paper is structured as follows: After describing the related work in Section 2 we give an overview of our method in Section 3. Afterwards, we describe the GPU algorithms and data structures in Section 4. In Section 5, the computation of global illumination is explained. Results and discussion are given in Section 6 before we conclude in Section 7.

## 2. Related Work

### 2.1. Depth Peeling

Everitt introduced depth peeling [Eve01] as a technique to extract all depth values of a pixel as depth layers in multiple rendering passes. To reduce the number of required rendering passes, several extensions to the basic algorithm have been developed (i.e. [BM08b], [BCL*07]). Some of these extensions use multiple render targets which allow them to store several fragments (which in turn equal depth layers) per pixel in one geometry pass. If these approaches are used, pipeline hazards can occur, as they are reading from and writing to the same render target. The stencil routed *k-buffer* [BM08a] avoids these hazards by using the stencil buffer to direct fragments to unused entries in the multisampled texture. Each time a pixel is accessed, the numbers stored in the corresponding fragments in the stencil buffer are changed to select the next target position. After one rendering pass, all depth values of a pixel are stored in the fragments, see [BM08a] for details.

### 2.2. Global Ray-Bundles

The use of global ray-bundles has been introduced by Sbert [Sbe96]. Szirmay-Kalos and Purgathofer used global ray-bundles for their global illumination algorithm [SKP98]. They presented a combined finite element and Monte Carlo method based on a global random walk, which can be used in diffuse and moderately glossy scenes. To compute the radiance exchange between visible surfaces, the scene is rasterized into two images perpendicular to the direction of

radiance transfer. One image contains the emitter patches, the other image the receiver patches. The two images are scanned and pairs of emitter and receiver patches are identified. The patches have to be sorted and rendered multiple times as the method relies on the fact that the receiver patches are not occluded.

Méndez et al. [MSC*06] use global ray-bundles to compute obscurances [IKSZ03], a view-independent lighting model taking only the interactions between nearby diffuse surfaces into account. The scene is stored in a texture atlas and depth-peeling [Eve01] is used to extract hit-points along a certain global ray direction. To update an entry in the texture atlas containing the indirect illumination, each pair of consecutive layers is turned into pixel-sized streams of points and directed to their appropriate atlas position. During this step the information of both layers are used to compute the obscurances for one direction. To compute the obscurances for the opposite direction, the point streams have to be created again.

A method using global ray-bundles for final gathering was proposed by Hachisuka [Hac05]. This method relies on a preprocessing step in which a coarse global illumination solution is computed. The result of this global illumination solution is attached to the vertices as vertex colors. Afterwards, the geometry is rasterized multiple times to extract all hit-points along a certain random direction. These hit-points are used in a final gathering step to compute the final solution for all visible pixel. This method works for ideally diffuse scenes.

In contrast to the presented techniques, our method does not rely on a preprocessing step in which a coarse global illumination solution is computed, as for example in [Hac05]. Furthermore, our method supports any number of arbitrary shaped area light sources without additional computation time. The cost of extracting the hit-points along a global ray-direction are dramatically reduced compared to the previous techniques through the help of the k-buffer. Furthermore, none of the existing methods reuse the already computed indirect illumination allowing us to compute several thousand bounces of light with arbitrary BRDFs, requiring just one additional texture lookup. Additionally, our method supports real-time walk-throughs in diffuse environments.

## 3. Overview

Our method for solving the rendering equation works entirely on the graphics processing unit (GPU) and it is summarized in Fig. 3.

Fig. 4 visualizes our method for one direction: First, an orthographic camera is placed on the bounding sphere of the scene. Using the k-buffer, all intersection points of all parallel rays are stored in the multi-sampling pixels in a single rendering pass. The information about the extracted points is retrieved from multiple texture atlases (Sec. 4). Afterwards, light transfer is computed between all pairs of successive points (Sec. 5). To provide interactive modifications of the geometry of the scene, the result is displayed in a progressive manner so that the solution can be refined as long as the geometry remains unmodified.

```
Select a random direction
Set orthographic camera with this direction
Draw scene using a k-buffer
Sort intersection points in ascending order
For each pair of consecutive intersec-
tion points (x,y)
  If x and y face each other
    Exchange radiance from x to y
Repeat process for next random direction
```

**Figure 3:** *Overview of the Global Ray-Bundles algorithm.*

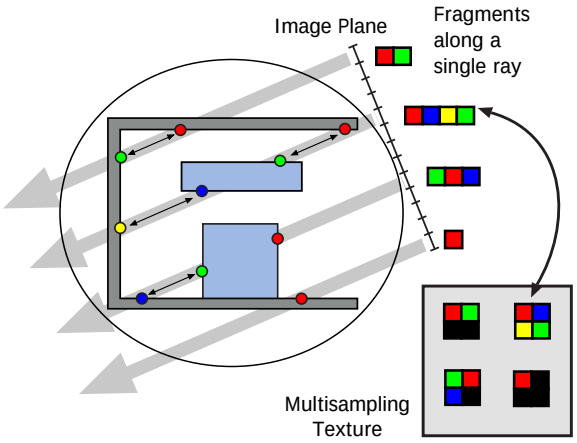


**Figure 4:** *Global ray-bundle cast into the scene from an orthographic camera. Each pixel stores k fragments with a stencil-routed k-buffer: All fragments (colored circles) are rendered to a multi-sampling texture. Each fragment is stored with its uv-coordinate, to identify the corresponding atlas texel, and its depth since the fragments need to be sorted after this step. Radiance can then be transferred between successive points.*

## 4. GPU Implementation

### 4.1. Discrete Sampling

To be able to transfer light between two arbitrary points of the scene, we use a series of *texture atlases* to acquire the information about the points. The following properties are stored for each texel: *Position*, *Normal*, *BRDF* and *Emission*. Furthermore, we use an atlas to accumulate radiance for diffuse surfaces and one additional atlas for the glossy transfer (explained in Sec. 5.4) The accuracy of the solution can easily be controlled by the resolution of the texture atlas.

### 4.2. Global Ray-Bundle Casting using the k-buffer

Before we detail the computation of the direct and indirect illumination, we give an overview on how global ray-bundle casting is done with the help of the k-buffer.

The stencil routed k-buffer uses a multi-sampling texture storing a certain number of fragments for each pixel. Each sample represents a different depth layer or a hit-point along a global ray, respectively. If the depth complexity of a given scene is less than or equal to the multi-sampling resolution, our method is able to compute the radiance exchange for one global direction in only two geometry passes. Current GPUs like NVIDIA's GTX 285 can store up to 32 samples. A higher depth complexity can be addressed by more than two render passes and the usage of a texture array instead of a single multi-sampling texture, whereas the stencil routed k-buffer is limited by the precision of the stencil buffer to 254 depth layers.

In the first pass the k-buffer is generated and written to a multi-sampling texture. For a given random direction, an orthographic camera is placed on the bounding sphere and the scene is rendered with z-buffering disabled. While the surfaces are processed by the GPU pipeline, the uv-coordinates and depth of each fragment of a pixel are stored in the multisampling texture (see Fig. 4).

Now we have a complete list of depth values for each pixel and a second pass is used to perform the radiance exchange between opposite sampling points. The k-buffer texture is re-projected onto the scene using exactly the same camera configuration as before. For each fragment processed by the GPU, the k-buffer is sorted by depth and the position of the fragment inside the k-buffer is located (see Fig. 5).

To determine the opposite sampling point (which can either be the next or the previous element of the k-buffer) the normal of the current fragment is compared to the global ray direction. Having identified the opposite sampling point, its uv-coordinates are used to access its radiance values and reflectance properties in the texture atlas.

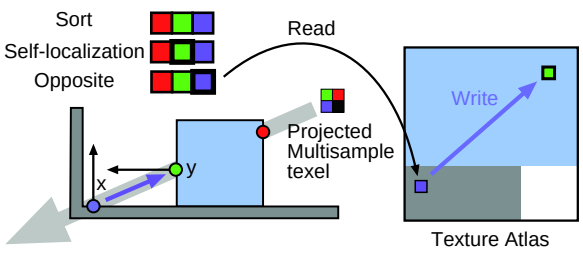


**Figure 5:** *For radiance exchange at the receiver point* $\mathbf{y}$*, the sender point* $\mathbf{x}$ *in a given direction* $\omega_i$ *must be determined. This requires a sorting of the fragments.*

### 4.3. Global Ray Directions

To generate the uniformly distributed, global ray-directions, low-discrepancy sampling is used. We have experimented with the Halton and the Hammersley sequence [PH04]. We found that the Halton sequence is better suited, since two successive sample directions vary in both spherical angles whereas the Hammersley sequences generates similar values for one of the spherical angles. Therefore, the Halton sequence generates a better coverage of the whole bounding sphere with fewer samples.

## 5. Global Illumination

To compute the radiance $L_o$ at a point $\mathbf{x}$, viewed from direction $\omega_o$, the rendering equation must be solved

$$L_o(\mathbf{x},\omega_o) = \int_{\Omega^+} f_r(\mathbf{x},\omega_i,\omega_o) L_i(\mathbf{x},\omega_i) cos\theta d\omega_i \qquad (1)$$

where $f_r$ is the BRDF, $L_i$ is the incoming radiance in direction $\omega_i$ and $\theta$ is the angle between the incoming direction and the surface normal at $\mathbf{x}$. Monte-Carlo Integration leads to an unbiased estimate of $L_o$:

$$L_o(\mathbf{x},\omega_o) \approx \frac{1}{N}\sum_{i=1}^{N} \frac{f_r(\mathbf{x},\omega_i,\omega_o) L_i(\mathbf{x},\omega_i) cos\theta}{p(\omega_i)} \qquad (2)$$

where $p(\omega_i)$ is an arbitrary probability density function and $\omega_i$ are $N$ sample directions, generated from $p$. In our case, we use uniformly distributed directions, resulting in a constant density of $p = 1/2\pi$. As described in Sec. 4, for a selected random direction $\omega_i$, two points $\mathbf{x}$ and $\mathbf{y}$ are extracted along the ray for radiance exchange. Now we can compute the radiance at the receiver $\mathbf{y}$ by transferring the radiance from the sender $\mathbf{x}$:

$$L_y(-\omega_i) = \frac{2\pi}{N} f_r(\mathbf{y},\omega_i,\omega_o) L_x(\omega_i) cos\theta \qquad (3)$$

where $L_x$ and $L_y$ are the radiance values *stored* at the atlas texels corresponding to $\mathbf{x}$ and $\mathbf{y}$ (see Fig. 5). Repeated evaluation of Eq. 3 with $N$ uniform directions $\omega_i$ and accumulation of the receiver values results in Eq. 2. Depending on the types of materials and number of indirect bounces, the sender radiance $L_x$ and receiver radiance $L_y$ are determined differently, as described in the following subsections.

### 5.1. Direct Illumination

Our algorithm allows the computation of direct light of arbitrary light sources, ranging from textured area lights with arbitrary shape over environment maps to surface light fields. The only requirement is to render all light sources from an

arbitrary view direction $\omega_i$. This results in the emissive radiance which is stored in the pixels and can then be used as incoming radiance $L_x(\omega_i)$ for illumination. Computing Eq. 3 for random directions then evaluates the direct light transfer of all lights in parallel. Although this allows an efficient computation of many complex light sources, it can be slow for scenes with only a few, simple light sources. This is discussed later in Sec. 6.

### 5.2. Diffuse Interreflections

After computation of direct light, one indirect diffuse bounce can be computed in a similar fashion. In case of diffuse materials ($f_r = \frac{\rho}{\pi}$), the radiance values $L_x$ and $L_y$ are view-independent and can be stored in a *sender atlas* and a *receiver atlas*. The sender radiance $L_x$ is now set to the sum of the emissive radiance and the direct radiance at **x**. Each time Eq. 3 is computed, the receiver radiance $L_y$ is simply accumulated to the value already stored in the texel corresponding to **y**. After $N$ directions, further bounces of indirect light can be computed in a similar fashion by exchanging sender and receiver atlas (Ping-Pong rendering). For practical reasons, we compute multiple bounces in a different way, as explained later in Sec. 5.5.

### 5.3. Final Glossy Bounce

After multiple bounces of diffuse light, a final specular bounce towards the eye can be computed (Final Gathering). Here, Eq. 3 is used with an arbitrary glossy BRDF towards the viewing direction ($\omega_{i+1} = \omega_o$) for each pixel in the image.

### 5.4. Glossy Interreflections

The most difficult case are multiple glossy interreflections, because the radiance at a point is *view-dependent* and we can not simply store one accumulated radiance value at the texel corresponding to a point. To obtain the correct sender radiance for a point on a glossy surface, we include the *next* random direction $\omega_{i+1}$ when computing the radiance exchange for direction $\omega_i$. To prepare the glossy transfer for the next iteration, we compute the BRDF $f_r(\mathbf{y}, \omega_i, \omega_{i+1})$. The reason for this is shown in Fig. 6. In this way, we compute the amount of radiance which is reflected at point **y** from direction $\omega_i$ to direction $\omega_{i+1}$. Since we need this value for the next iteration, we store

$$L_x(\omega_i) f_r(\mathbf{y}, \omega_i, \omega_{i+1}) cos\theta \tag{4}$$

in a *transfer atlas* at the texel corresponding to **y**. When computing the radiance exchange for the *next* direction $\omega_{i+1}$, one of the pairs of points to exchange radiance will be **y** and **z**. Now, **y** is the sender and we can use the value stored in the transfer atlas as the glossy sender radiance at **y**. In this way, we connect a path from **x** over the glossy surface at **y** to the receiver **z**. To extend the length of the glossy path to an arbitrary length, we can repeat this process by computing the glossy radiance at **z** for the next direction $\omega_{i+2}$, and so on. Since we use uniformly distributed random directions, this results in the computation of all possible combinations of paths of arbitrary length.

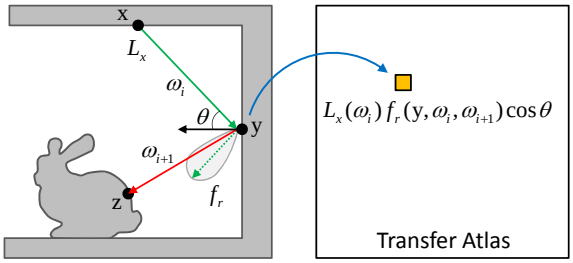


**Figure 6:** *For a sender point* **y** *with a glossy BRDF, the sender radiance at* **y** *in direction of the receiver* **z** *is precomputed and stored in the transfer atlas.*

### 5.5. Interleaved Light Transfer

The simple solution to compute multiple bounces of light is to use the described algorithm to compute one bounce of light until convergence and then use the result as input for the next bounce by exchanging sender and reciever atlas. However, we found that the best results are achieved if the radiance transfer of one direction is *directly* used for the next direction as input. To accomplish this, we read the values $L_x$ and $L_y$ from the sender atlas and add the transferred radiance to the value stored in the receiver atlas at the texel corresponding to **y**. Sender and receiver atlas are then exchanged after *each* direction $\omega_i$. Consequently, when computing the transferred radiance along a direction $\omega_i$, the sender radiance $L_x$ contains light that was reflected *up to i* times. Similar to a Gauss-Seidel Iteration, we include indermediate results which leads to a faster convergence. Since we transport light of different path lengths, we call this method an *interleaved* light transfer. For a better understanding, Fig. 7 shows a simple example. If not stated otherwise, all images in this paper are computed with the interleaved multi-bounce method.

## 6. Results

In the following we present our results. All images were rendered with an NVIDIA GeForce GTX 285 and a 3 Ghz CPU.

### 6.1. Quality

Our algorithm is capable of handling very complex lighting situations as shown in Fig. 1. The left image shows multiple diffuse bounces from an environment map and an additional area light from the top. The image in the center contains multiple light sources and a final glossy bounce. Since

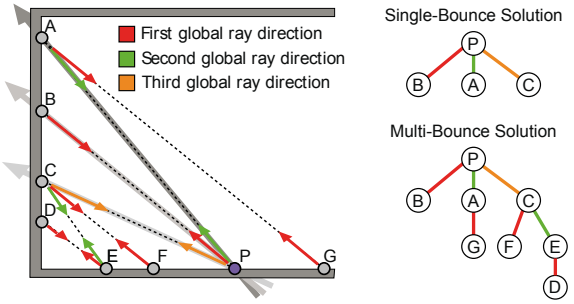


**Figure 7:** *Propagation of indirect radiance for three random directions using the single-bounce and the interleaved multi-bounce method for point P. Using the single bounce method, each direction propagates the radiance of the opposite hit point only. With just one additional texture lookup, the interleaved multi-bounce method exchanges the radiance of paths of different length. The first direction includes the direct light of the opposite hit-point, the second direction the light for a path of length two. The third direction includes the light of a path of length two and a path of length three.*

the light transfer is computed by rendering the scene from different directions, the computation time is *independent* of the number of light sources. The image on the right shows a textured, spherical light source inside a scene with many corners. Note the soft, multiple colored bounces of light around the corners. The backfaces of the wallpapers are displayed for better visualization of the light distribution.

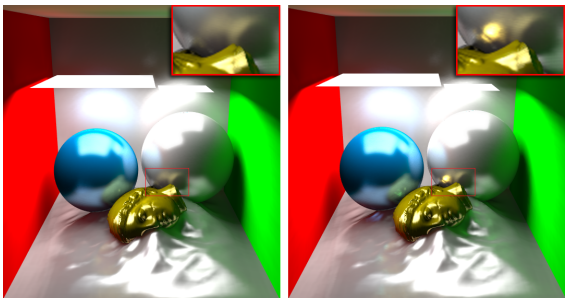

**Figure 8:** *Glossy reflections using a Phong BRDF. The left image shows a final glossy bounce after multiple diffuse interreflections. The right image shows multiple glossy interreflections. Note the indirect highlights on both spheres. Computation time is 2 minutes for 600 directions.*

Another strength of the global ray-bundles is the ability to compute multiple glossy bounces. Fig. 8 shows multiple objects with a Phong BRDF, illuminated by area light sources. Note how our method correctly displays the glossy interreflections between the head and the spheres which are omitted by a standard final gathering. In all these conditions, computing the image with a path tracer is typically very noisy.

### 6.2. Physical Correctness

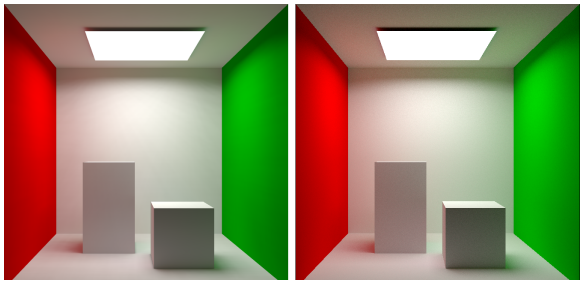

**Figure 9:** *Our algorithm (left) compared to a Path Tracer (right). Both images are generated with identical physical setups, equal gamma correction and without tone-mapping. Rendering times were 178 seconds for our solution and 27 minutes for the Path Tracer.*

To demonstrate the accuracy of the presented algorithm, a comparison to a Path Tracer is shown in Fig. 9. In this example, the k-buffer resolution was set to $2048 \times 2048$ pixels and the illumination was computed for 4.000 random directions using our proposed multi-bounce, interleaved light transfer method. The Path Tracer used 768 rays per pixel and the maximum depth of a ray was set to 10. Note that we obtain an identical radiance distribution and correct indirect shadows. Although the Path Tracer uses a real-time ray-tracing engine, the computation took 27 minutes and the result is still noisy. In contrast, using global ray-bundles took only 178 seconds to display a noise-free, converged solution.

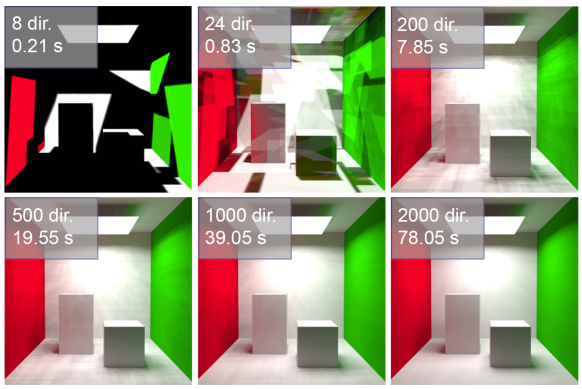


**Figure 10:** *Convergence of the interleaved multi bounce method for 2.000 global ray directions. The first images show how (direct) light is projected along the global ray directions. After a few seconds, a good visual preview is obtained.*

### 6.3. Performance

The performance of the presented method largely depends on the resolution of the texture atlases and the k-buffer as these textures have to be filled in each pass to compute the radiance exchange for one global random direction. Tab. 1 lists benchmarks for varying k-buffer resolutions to compute the direct (diffuse and specular) and global (direct and indirect) radiance transfer along one ray-direction (*ms/global dir.*). As the resolution of the texture atlas directly influences the accuracy of the solution, it is set to $2048 \times 2048$ for all timings.

The computation of the global illumination is slower than the computation of the direct illumination as more render targets have to be filled. Additionally, a case distinction has to be carried out whether the global ray-bundles have hit the light source and therefore direct illumination is exchanged or if they have hit another scene element from which indirect radiance has to be exchanged.

| Illumination | M-S | K-buffer | ms / global dir |
|---|---|---|---|
| Direct | 16 | $1024 \times 1024$ | 121 |
| Global | 16 | $1024 \times 1024$ | 208 |
| Direct | 8 | $2048 \times 2048$ | 36 |
| Global | 8 | $2048 \times 2048$ | 51 |
| Direct | 8 | $1024 \times 1024$ | 26 |
| Global | 8 | $1024 \times 1024$ | 39 |
| Direct | 8 | $512 \times 512$ | 22 |
| Global | 8 | $512 \times 512$ | 36 |

**Table 1:** *Benchmarks for NVIDIA's GTX 285. M-S indicates the multi-sampling resolution of the k-buffer (and therefore the number of hit-points along a global ray-direction which can be extracted in one geometry pass). The time it takes to compute the radiance exchange in both directions for all entries in the texture-atlas along one global ray-direction is given in the last column. The resolution of the texture atlas is set to* $2048 \times 2048$ *for all timings.*

Our proposed method can cast up to 932.067.555 $rays/sec$. for the computation of the direct light and up to 657.930.039 $rays/sec$. for the computation of the global illumination (with a 2k k-buffer resolution). Using a setup with a 1k k-buffer, it is possible to compute 38.46 $directions/sec$. for the direct light and 25.64 $directions/sec$. for the global illumination. This allows us to display the intermediate results with interactive frame-rates, hence real-time walk-throughs are possible while the computation takes place.

### 6.4. Rate Of Convergence

Fig. 10 shows how our proposed method converges towards the correct solution. The resolution of the k-buffer is set to $1024 \times 1024$ pixel and eight-times multi-sampling and the texture atlas has a resolution of $2048 \times 2048$ pixel. For the first 8 directions only the direct light is computed, from the ninth direction on the global illumination is computed using our interleaved multi-bounce method.

After the first few directions, large differences are visible since the illumination is estimated for a very small number of samples. This results in some visual artifacts like visible projections along a certain direction. Up to approximately 200 random directions (7.8s), such artifacts may appear but the image provides a good first impression of the final result. Between 1000 and 2000 random directions (39s-78s), only slight differences are visible.

If the intermediate results of the indirect illumination should not be display in a progressive fashion, one possible optimization to achieve a faster rate of convergence is to enable the global illumination shaders at a later time, as the indirect illumination converges faster than the direct one (300 to 500 random directions compared to 1000 to 2000 directions). With the most optimized setup (1700 directions for the direct illumination only and 300 for the global illumination) we can achieve the same result as seen in Fig. 10, 2000 directions, within 55.9s compared to 78.05s if we use the progressive scheme and enable the global illumination from the beginning.

### 6.5. Combination with Fast Direct Light

If the scene contains only a few, simple light sources, the use of global ray-bundles can be inefficient for direct light computation. In these cases, a different method can be used for direct light computation and the global ray-bundles can be used for indirect light only. Fig. 11 shows an example where the Percentage Closer Soft Shadow-Mapping (PCSS) algorithm [Lau07] is used for real-time direct light computation with approximate soft shadows of a small area light. Given the atlas with the direct light, the resulting indirect bounces of light can be computed with global ray-bundles. If the approximations in penumbra regions are acceptable, this allows an efficient combination of direct and indirect light.

### 6.6. Discussion

Though the presented scheme has a large number of advantages, there are several limitations due to the maximum resolution of the k-buffer, the orthographic camera and the atlas. As already mentioned, the number of depth layers is limited by the multi-sampling resolution of the graphics hardware. This prevents scenes with a higher depth complexity, but could be addressed by using a texture array as a render target. The second limitation is the accuracy of the presented scheme, that is mainly limited to the size of the texture atlas. Common GPUs support a texture size up to $8192 \times 8192$ texel, but a massive number of texels would have to be processed in the fragment shader, reducing the performance.

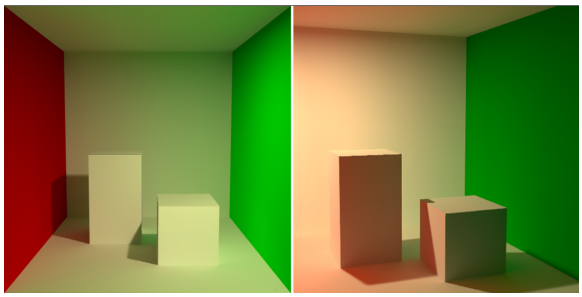

**Figure 11:** *Instead of using global ray-bundles for the computation of the direct illumination, a Percentage Closer Soft Shadow-Mapping algorithm is used for the area light. The indirect illumination is then computed with our proposed interleaved multi-bounce solution. The total rendering time is 3.65 seconds (300 global directions).*

The accuracy in extended scenes is therefore lower since one atlas texel covers a large area. Furthermore, extremely spiky BRDFs might fall between two global directions and cannot be reproduced correctly due to the atlas discretization. We also point out that the low noise is also a consequence of the discretization. However, we found that the combination of global ray-bundles and atlas discretization is a good compromise between noise and blurring.

## 7. Conclusion

In this paper, we demonstrated that global ray-bundles can be used for physically correct global illumination with an arbitrary number of interreflections. Especially, difficult multiple bounces of glossy BRDFs can be simulated. Global ray-bundles exploit the parallelism of current GPUs by extracting all hitpoints on bundles of parallel rays in a single rendering pass. This allows an unbiased computation of the rendering equation with only the screen and atlas discretization as a limitation. Global ray-bundles are independent of the type and number of light sources and thus a fast alternative to standard path tracing that often shows noise in such cases.

As future work, we will investigate larger scenes. Although there are limitations due the maximum resolution of the atlas, recent work [YCK*09] has shown that approximations in indirect light can be accepted in many cases. Furthermore, an importance-based sampling of directions that takes the intensity distribution of the light sources as well as the scene BRDFs into account is an interesting avenue of future research.

## References


[BCL*07] BAVOIL L., CALLAHAN S. P., LEFOHN A., COMBA L. D., COMBA C. T.: Multi-Fragment Effects on the GPU Using the K-Buffer. In *I3D 07, Symposium on Interactive 3D Graphics and Games* (2007), ACM, pp. 97 – 104. 2

[BM08a] BAVOIL L., MYERS K.: Deferred Rendering Using a Stencil Routed K-Buffer. In *ShaderX 6. Advanced Rendering Techniques* (2008), Charles River Media, pp. 189 – 198. 1, 2

[BM08b] BAVOIL L., MYERS K.: Order Independent Transparency with Dual Depth Peeling. In *Technical Report, NVIDIA Corp.* (2008). 2

[CPC84] COOK R., PORTER T., CARPENTER L.: Distributed Ray Tracing. In *Computer Graphics* (1984), vol. 18 (3), pp. 137 – 145. 1

[Eve01] EVERITT C.: Interactive Order-Independent Transparency. In *Technical Report, NVIDIA Corp.* (2001). 1, 2, 3

[GTGB84] GORAL C. M., TORRANCE K. E., GREENBERG D. P., BATTAILE B.: Modeling the Interaction of Light between Diffuse Surfaces. In *SIGGRAPH 84, Computer Graphics Proceedings* (1984), pp. 213 – 222. 1

[Hac05] HACHISUKA T.: High-Quality Global Illumination Rendering Using Rasterization. In *GPU Gems 2*. Addison-Wesley Professional, 2005, ch. 38, pp. 615–633. 2, 3

[IKSZ03] IONES A., KRUPKIN A., SBERT M., ZHUKOV S.: Fast, Realistic Lighting for Video Games. *IEEE Comput. Graph. Appl. 23*, 3 (2003), 54–64. 3

[Jen01] JENSEN H. W.: *Realistic image synthesis using photon mapping*. A. K. Peters, Ltd., Natick, MA, USA, 2001. 1

[Kaj86] KAJIYA J. T.: The Rendering Equation. In *SIGGRAPH 86, Computer Graphics Proceedings* (1986), vol. 20 (4), pp. 143 – 150. 1

[Lau07] LAURITZEN A.: Summed-Area Variance Shadow Maps. In *GPU Gems 3* (2007), Addision-Wesley. 7

[LW93] LAFORTUNE E. P., WILLIAMS Y. D.: Bi-Directional Path Tracing. In *Proceedings of Third International Conference on Computational Graphics and Visualization Techniques (Compugraphics '93)* (1993), pp. 145 – 153. 1

[MSC*06] MENDEZ A., SBERT M., CATA J., SUNYER N., FUNTANE S.: Realtime Obscurances with Color Bleeding. In *ShaderX4: Advanced Rendering Techniques* (2006), Charles River Media, pp. 121–133. 3

[PH04] PHARR M., HUMPHREYS G.: *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufmann, 2004. 4

[Sbe96] SBERT M.: The Use of Global Random Directions to Compute Radiosity - Global Monte Carlo Techniques. In *Ph. D. diss.* (1996), Universitat Politecnica de Catalunya. 1, 2

[SKP98] SZIRMAY-KALOS L., PURGATHOFER W.: Global Ray-Bundle Tracing with Hardware Acceleration. In *Ninth Eurographics Workshop on Rendering* (1998). 2

[YCK*09] YU I., COX A., KIM M. H., RITSCHEL T., GROSCH T., DACHSBACHER C., KAUTZ J.: Perceptual influence of approximate visibility in indirect illumination. *ACM Trans. Appl. Percept. 6*, 4 (2009). 8