Vis Comput (2010) 26: 679–686
DOI 10.1007/s00371-010-0486-2

ORIGINAL ARTICLE

Fast indirect illumination using Layered Depth Images

Matthias Nießner · Henry Schäfer · Marc Stamminger

Published online: 8 April 2010
© Springer-Verlag 2010

Abstract We present a novel hybrid rendering method for diffuse and glossy indirect illumination. A scene is rendered using standard rasterization on a GPU. In a shader, secondary ray queries are used to sample incident light and to compute indirect lighting. We observe that it is more important to cast many rays than to have precise results for each ray. Thus, we approximate secondary rays by intersecting them with precomputed layered depth images of the scene. We achieve interactive to real-time frame rates including indirect diffuse and glossy effects.

Keywords Real-time global illumination · Hybrid rendering

1 Introduction

We present a method to compute indirect diffuse and glossy lighting on the GPU within the rasterization pipeline. Indirect lighting is computed in a pixel shader by casting secondary rays with *approximate* intersections. This is motivated by the assumption that it is more important for these effects to have many rays than to have precise intersection points. As a result, we achieve interactive rendering times for single-bounce indirect diffuse and glossy lighting. An example is shown in Fig. 1, which contains indirect glossy and diffuse lighting and is rendered at 27 frames per second.

M. Nießner (✉) · H. Schäfer · M. Stamminger
Computer Graphics Group, University of Erlangen-Nuremberg, Erlangen, Germany
e-mail: Matthias.Niessner@cs.fau.de

H. Schäfer
e-mail: Henry.Schaefer@cs.fau.de

M. Stamminger
e-mail: Marc.Stamminger@cs.fau.de

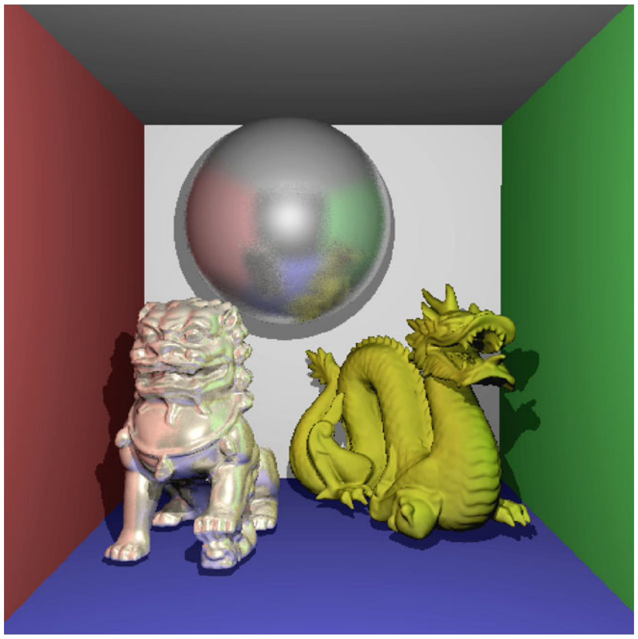
Fig. 1 Indirect lighting on the three objects in the Cornell Box rendered at 27.0 fps (512 × 512 resolution) using our method: glossy BRDF on sphere and lion, diffuse BRDF on dragon

Approximate ray intersections are computed using a set of orthographic layered depth images (LDIs) taken for different directions. For each secondary ray, the LDI with the most similar direction is determined and the intersection within this LDI is computed. Due to the similar projection and ray direction, only a few pixels have to be traversed in this step, making the computation very fast. On the downside, the LDIs may require significant memory. However, in our experiments a rather limited number of LDIs (about 100) and rather low LDI resolution (128 × 128 up to 512 × 512) was sufficient for indirect lighting. Furthermore, LDIs can be compressed well: in our example scenes, memory consumption was always less than 100 MB, even for scenes like Sponza or Sibenik.

Our method is hybrid in the sense that eye rays are computed using rasterization, whereas indirect lighting is computed by sampling incident light using secondary rays. Shadows can be computed using rasterization techniques, typically by shadow maps or volumes, or using our ray intersection approximation. In order to further accelerate indirect lighting computation, we reduce the number of indirect rays and filter indirect light in screen space. Our approach allows us to utilize the advanced anti-aliasing capabilities of current GPUs at almost no additional cost.

2 Previous work

With increasing computational power of GPUs, the inclusion of global illumination effects into real-time rendering has attracted more and more attention. Two main directions of research can be observed: one is to include global illumination to the standard rasterization pipeline, the other one is to make ray tracing interactive. Hybrid algorithms try to combine both rendering paradigms. In the following, we try to give an overview of papers that are related to our work.

Rasterization based global illumination Soft shadows are an important subproblem of global illumination. There have been many approaches to extend shadow volumes, e.g. [3, 7], or shadow maps, e.g. [8], to also generate soft shadows. Several of the shadow map based approaches use a single layered depth image taken from the light source's center (termed multilayer shadow map) to approximate visibility [1, 4, 20, 24, 27]. Ritschel et al. [16] use a series of shadow maps from different parallel directions for global lighting computations and presented ways to compress the shadow maps.

A very early method that generates indirect diffuse light is instant radiosity [12]. Indirect light is represented using virtual point lights, which are rendered using shadow maps. Dachsbacher et al. [6] add the indirect light due to virtual point lights using a deferred shading approach. Ritschel et al. [17] present an efficient method to interactively compute all necessary shadow maps. Since indirect light is generally smooth, image space interpolation as proposed by Segovia et al. [21] is often used to reduce the sample density and blur noise or discretization artifacts. Recently, Ritschel et al. [18] presented a method to compute indirect sample rays within the standard rendering pipeline, using low resolution per-pixel buffers.

Interactive ray tracing An overview of ray tracing acceleration structures for CPUs is given in [10]. Wald et al. were among the first to propose ray tracing for interactive rendering [25]. When GPUs became freely programmable, their application for ray tracing has been examined by Purcell et al. [15].

The architecture of today's GPU requires specific optimization in order to achieve optimum performance. Aila et al. [2] give an overview of current GPU ray tracing methods and present their own implementation. A general observation for ray tracing based global illumination is that computation time strongly depends on ray coherence. Aila et al. report up to 150 million rays per second for coherent primary rays, but only 20–40 million rays per second for incoherent secondary diffuse rays. Ray packets in [26] are used to accelerate the traversal of coherent rays using SIMD parallelism.

A major problem with interactive ray tracing is the generation and update of the acceleration structure for dynamic scenes, but efficient solutions are available meanwhile [13].

Hybrid methods Hybrid approaches try to combine the high efficiency of rasterization for the computation of eye ray hits with the flexibility of indirect lighting computation by ray tracing. Roger et al. [19] build on the fly a 5D acceleration structure to compute (coherent) secondary rays. Hertel et al. [11] compute shadows using a shadow map and cast additional rays to decide about uncertain shadow map results. Bürger et al. [5] use a cube of three orthogonal LDIs to trace arbitrary rays for secondary lighting effects.

Layered Depth Images Layered Depth Images [23] are rasterized depth values of a scene where for each pixel not only the frontmost depth value is stored, but all depth values are kept in a per-pixel list. Depth Peeling [14] is a simple way to generate LDIs with fixed length list on a GPU. Such lists can be well compactified using a *scan operation* [22] computing a *prefix sum*. Hatchisuka et al. also use LDIs in a non-interactive final gather pass.

3 Ray approximation using LDIs

The core idea of our method is to use layered depth images (LDIs) to approximate secondary ray intersection. We will first describe the generation of the LDIs and then their usage for approximate intersection computations.

3.1 LDI generation

In a preprocessing step, we generate n orthographic LDIs of the scene, using uniformly distributed projection directions. We generate the n directions by uniformly sampling a unit cube with a resolution of k^2 per face, resulting in $n = 6k^2$ directions. This sampling is not perfectly uniform, but allows

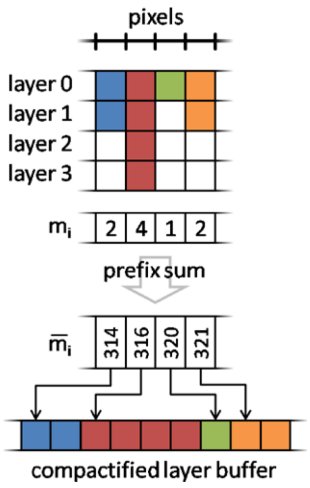

Fig. 2 To save memory for areas with low depth complexity, we count the depth values for each pixel of a given LDI (m_i) and apply stream compactification

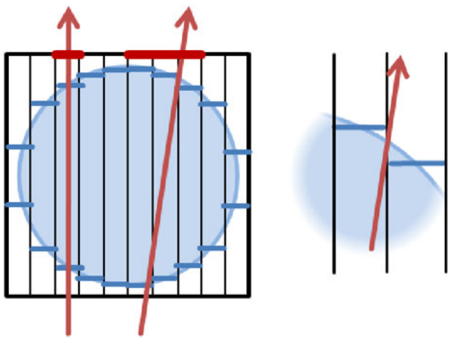

Fig. 3 (*Left*) We intersect rays with a discretized approximation (LDI, dark blue). (*Right*) For steep geometry, leaks can occur

us to quickly determine the closest sample direction for a given ray later. The LDIs for opposite direction are redundant, so memory consumption could be halved by using a single LDI for both directions, but we currently do not exploit this.

Each LDI is generated using depth peeling [14], which results in a series of m depth layers. Next, we compactify the m layers (cf. Fig. 2). Be $p_i, i = 1, \ldots, N$, the ith pixel, e.g., in scanline order, and $m_i \leq m$ the number of depth layers found for pixel p_i. Then, we compute the *prefix sum* [22] of the m_i, resulting in a sequence $\bar{m}_i = \sum_{j=1}^{i-1} m_i$.

The $\bar{m}_i$ are used as indices to a compactified array, which is a linear float texture. The depth values and corresponding triangle ids of each pixel i are stored in positions $\bar{m}_i, \ldots, \bar{m}_{i+1} - 1$. The $\bar{m}_i$ are stored in an integer texture, so we can quickly access the data associated with each pixel.

3.2 LDI ray tracing

Given a ray $\mathbf{k} + \alpha \mathbf{l}$, we use the previously computed LDIs to approximate an intersection point. First, we determine the LDI with the most similar projection direction compared to the ray direction $\mathbf{l}$. Next, the ray starting point $\mathbf{k}$ is projected into the orthogonal screen space of the LDI, giving us the starting pixel of the ray. We fetch the depth layers for that pixel and determine whether the ray intersects one of these before it leaves the pixel (cf. Fig. 3). If yes, we directly return the found sample as intersection. Otherwise, we iterate through the neighboring pixels as touched by the ray. These *pixel crossings* are continued until an intersection is found or the ray leaves the LDI.

Ideally, the projection and ray direction are identical (left ray in Fig. 3). In this case, only the intersection with the depth layers from the start pixel needs to be determined—no pixel crossings are necessary. The more projection and ray direction differ, the more pixel crossings are needed (e.g., right ray in Fig. 3). Furthermore, there is a danger that a ray leaks through the discretization of a closed surface (Fig. 3, right). We solve this issue using a simple ϵ-tolerance for the intersection computation.

Having obtained an intersection for a ray with the scene geometry, the triangle id which is also stored is used as a reference back to the original scene geometry. Thus, necessary data for shading such as normals, texture coordinates and material information can be obtained.

The time to compute such an approximate intersection mainly depends on the number of pixel crossings. Thus, a lower LDI resolution is faster, but also results in less accurate results. A larger number of LDIs on average results in more similar ray and LDI directions, and thus in fewer pixel crossings and better performance. On the other hand, more LDIs require more memory and thus reduce cache coherence during traversal. We will examine these issues in Sect. 7.

4 Hybrid rendering

We embedded the LDI tracing into a hybrid renderer. In a preprocessing step, LDIs are rendered and packed into textures. Eye rays are computed by conventional rasterization, pixel shaders can cast secondary rays using our approximate method. We implemented pixel shaders with global illumination effects, namely mirroring, glossy, and diffuse reflections by distribution ray tracing, hard shadows, and soft shadows by area light source sampling. In the following, we will concentrate on indirect illumination, which is ideal for our approach. Deferred shading is used to avoid costly computations for occluded pixels. We support different materials by storing a shader identifier per pixel. Since we shoot the rays from within a shader, arbitrary BRDFs and textures are directly supported.

Hybrid rendering also allows us to add anti-aliasing without significant extra costs, see Fig. 4. While common ray tracers need to shoot a multiple of additional primary rays for multi-sampling, we employ graphics hardware for this task. Note that with anti-aliasing, as supported by state-of-the-art GPUs, multiple eye ray samples are computed per pixel. However, for every visible primitive within a pixel, only a single shader call is executed, i.e., per pixel only one set of secondary rays are cast. In order to have anti-aliasing support in connection with deferred shading, all render targets have to be multi-sample render targets.

5 Approximating indirect illumination

In order to realize indirect illumination, we evaluate glossy and diffuse BRDFs using distribution ray tracing. While a single reflecting ray is needed for perfectly mirroring surfaces, glossy BRDFs need several secondary rays scattered around the reflection direction. Diffuse BRDFs require most samples because rays are distributed over the complete hemisphere around a pixel's normal. Fortunately, approximation errors for diffuse secondary rays are well blurred out.

We only compute a single bounce of indirect light. For the hit points of the secondary rays, we use a standard local lighting model including a shadow map for visibility queries. If the hit surface is textured, we access a coarse mipmap level in order to filter the texture and reduce aliasing.

Efficient rendering is achieved by using a coarse LDI resolution (128×128 is a good choice). For indirect diffuse and glossy light, such a low resolution was sufficient in our scenes, for highly specular surfaces, however, higher resolution is required or discretization artifacts become visible.

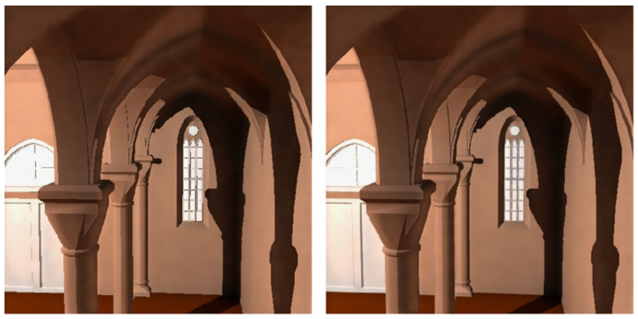
Fig. 4 The Sibenik scene without (*left*, 17.5 fps) and with anti-aliasing (*right*, 16.5 fps)

Our time budget allows us to compute between 8 and 64 secondary rays per pixel, which still results in visible noise. To counteract this, we use filtering and combine information about secondary rays of adjacent pixels. Basically, an $n \times n$ filter kernel causes a pixel's color to be influenced by $rays/pixel \cdot n^2$ indirect rays. For fast filtering, we employ a separable Gaussian filter implemented on the GPU. Preventing filtering over hard edges as suggested in [21] did not seem to be necessary in our test scenes, however, this is an option to assess in the future.

In summary, we realize indirect illumination with the following pipeline:

– Rasterize the scene into a G-Buffer.
– Shoot diffuse secondary rays using LDIs.
– Filter indirect diffuse light contribution with a two pass Gaussian.
– Shoot secondary rays for glossy materials.
– Combine direct, indirect diffuse and glossy light contributions.

The interim results of each step are visualized in Fig. 5.

6 Handling dynamic objects

Once having built the LDI structure, all scene geometry is fixed. The LDIs are rasterized on the GPU, but depth peeling requires several render passes for a single LDI. Hence, the precomputation consists of rendering a few hundred up to several thousand frames plus a subsequent compactification (which we currently do on the CPU). This can be done in a few seconds considering moderately complex scene environments (see Sect. 7). However, we do not expect to be able to update the LDIs interactively on current graphics hardware. Consequently, only light and viewing positions are fully dynamic.

A practical solution for handling dynamic objects is to simply skip them during LDI creation. As a result, only the

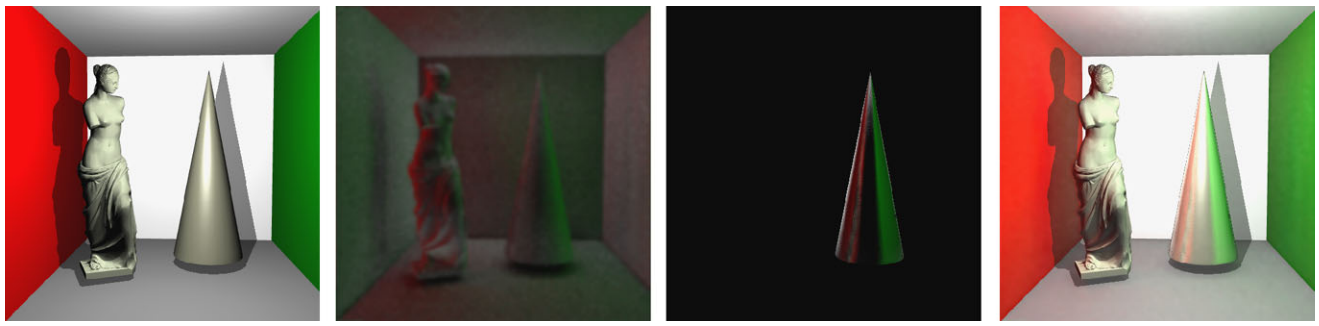
Fig. 5 The different steps of our indirect illumination pipeline from *left* to *right*: direct light including shadows, filtered diffuse indirect light, glossy indirect light, and all combined

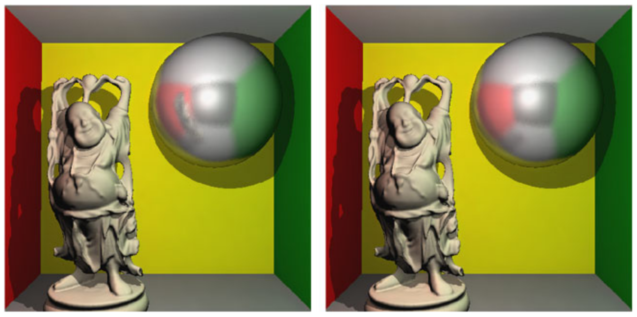
Fig. 6 Both images show indirect illumination and a glossy reflection on the sphere. On the *right*, the Buddha is not part of the LDIs. Consequently he cannot be seen on the sphere. However, although he cannot emit light, the Buddha still receives indirect light (note the color bleeding) and casts shadows that are seen in the reflection. This allows him to be a fully dynamic receiver without recomputing LDIs

static geometry is present in the LDIs and can emit indirect light. Even so, dynamic objects that are not part of the LDIs can receive indirect light from them. So it makes sense to use LDIs for static environments, but to gather indirect light also on the dynamic objects. An apparent difference exists mainly for glossy objects, since the dynamic objects are missed in reflections (cf. Fig. 6).

7 Results

Our implementation is based on DirectX 10 and was tested on an NVIDIA GTX 285 with 1024 MB of memory. All images shown have been computed at a screen resolution of 512 × 512.

The LDI generation including compactification is done offline and takes from 1 second (96 LDIs with 128 × 128 pixels) up to 23 seconds (384 LDIs with 512 × 512 pixels) for the Sponza scene. We did not spent efforts to optimize this, and compactification is currently done on the CPU. Memory consumption is depending on the scene's average depth complexity, LDI resolution and LDI number. We need around 15 MB up to 60 MB memory for our test scenes using 150 LDIs with 128 × 128 pixels (this setup with 150 and 96 LDIs is used for indirect illumination).

Examining LDI ray tracing For a start, we examine the quality and performance using LDIs for ray tracing, regarding their resolution and number. Figure 7 shows the loss of quality by decreasing LDI resolution from 512 × 512 down to 128 × 128, both with 150 LDIs. As expected, the low resolution results in pixelized reflections. Performance also varies significantly: 336 fps (512 × 512 pixel LDIs) versus 600 fps (128 × 128 pixel LDIs). A smaller resolution decreases the number of required pixel crossings, and consequently ray tracing performs faster.

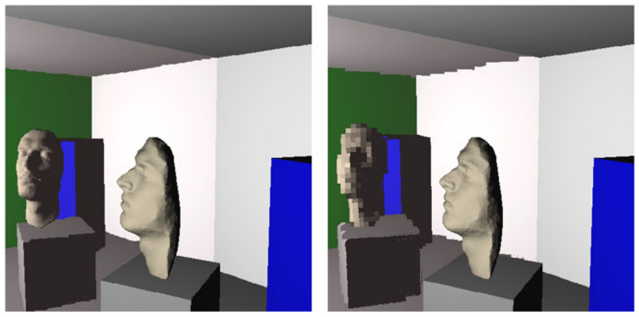
Fig. 7 The quality of a reflection with LDI resolution of 512 × 512 (*left*) and 128 × 128 pixels (*right*)

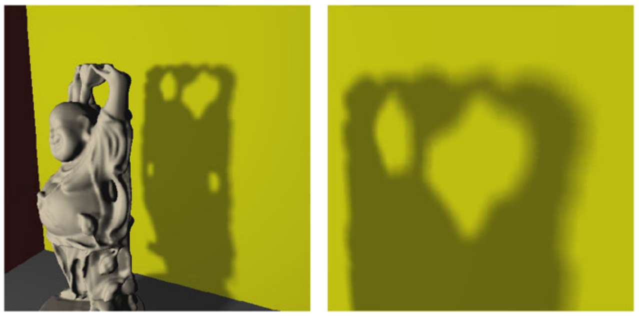
Fig. 8 Soft shadows: rendered at 19.5 fps (*left*) and 22.5 fps (*right*) using 384 LDIs (512 × 512 resolution)

Table 1 Rays/s for different BRDFs using LDIs with 512 × 512 and 128 × 128 pixels in the Sibenik scene including shading

BRDF	512 × 512	128 × 128	LDIs
diffuse	11–16 M	37–50 M	96
glossy	63–71 M	151–209 M	150
soft shadows	79–103 M	283–377 M	384

The comparison shows that a tradeoff between quality and performance is possible. For glossy and particularly diffuse secondary rays, artifacts from lower LDI resolutions are blurred out. In this case a lower LDI resolution is acceptable with the benefit of much faster ray intersections and lower memory consumption. In contrast to this, soft shadows are less error tolerant and tend to reveal inaccuracies at shadow borders. Nevertheless, an LDI resolution of 512 × 512 is sufficient to render smooth soft shadows as shown in Fig. 8. The images are generated by sampling an area light with 16 shadow rays per pixel plus additional filtering (19.5 and 22.5 fps).

Table 1 shows how many rays we are able to cast for two different LDI resolutions. All rays/s refer to secondary rays only, however, hardware rasterized additional primary rays in parallel (the relation of primary to secondary rays was 1

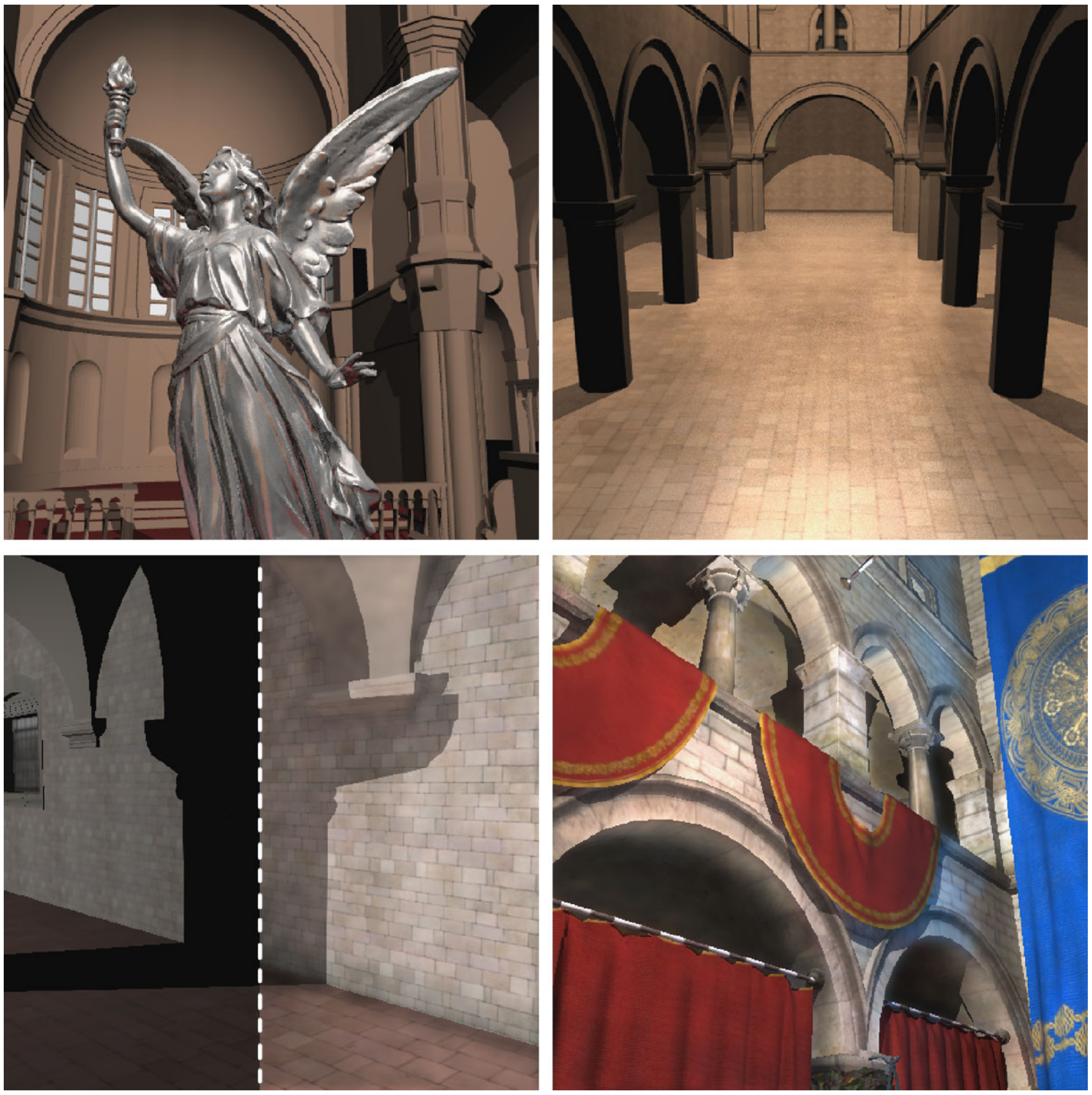

Fig. 9 Indirect illumination results. (*Top left*) Glossy BRDF on Lucy included in the Sibenik scene (45.0 fps). (*Top right*) Glossy BRDF on floor of Sponza (52.5 fps). (*Bottom left*) Diffuse indirect lighting in the Sponza scene (19.0 fps). For comparison, the left half is without indirect lighting. (*Bottom right*) Diffuse indirect lighting for an enhanced Sponza (15.5 fps)

to 16). We see that coherent soft shadow rays behave best. Since glossy rays are at least more coherent within a single pixel shader call, we are able to shoot a significant higher number of glossy rays/s than diffuse rays/s. We also determined that using 150, 96 and 384 LDIs for the respective BRDF optimized the performance in our test scenes. In theory more LDIs inflict fewer pixel crossings and theoretically boost performance. However, in practice using (particularly for incoherent rays) more LDIs results in incoherent memory access and penalizes caching effects. Consequently, only soft shadow performance benefits from an increased LDI count. The more important thing is revealed by the comparison of LDI resolutions. While for 512×512 pixels the performance is comparable to common GPU ray tracers, for lower resolutions we are much faster (see Aila et al. [2], Sibenik scene results).

Moreover, for each ray we already applied shading including a visibility query using a shadow map lookup. The hybrid renderer also provides cheap extra primary rays for anti-aliasing. Figure 4 shows the Sibenik scene without (left) and with (right) anti-aliasing including 8 secondary light rays/pixel. The left image is rendered at 17.5 fps and the right one at 16.5 fps, while right includes anti-aliasing (4 samples/pixel). The aliasing artifacts on the window and arcs are obviously reduced. A comparable ray tracer would need to trace 4 instead of 1 primary rays per pixel for the same effect.

Performance depended less on the polygon count of a scene than on the current viewing position. Thus, for all test scenes the performance was approximately constant. The framerate in tight corridors of the Sponza scene was higher than in an empty Cornell Box, which is reasonable since rays which sooner hit objects trigger fewer pixel crossings.

Applying indirect illumination In order to cast many rays, we use LDIs of 128 × 128 pixels for both glossy and diffuse indirect illumination. Glossy reflections are rendered with 16 secondary rays/pixel (no filtering) and diffuse BRDFs are applied with 8 secondary rays/pixel, plus subsequent filtering with a 11 × 11 Gaussian filter kernel.

Our results for rendering indirect illumination can be seen in Fig. 9. Top left shows Lucy embedded in the Sibenik scene rendered at 45.0 fps using 150 LDIs. Note the reflection of the environment on Lucy's surface. Top right is also a glossy effect on the floor of the Sponza scene rendered at 52.5 fps using 96 LDIs. Bottom left shows the Sponza scene with diffuse indirect illumination rendered at 19.0 fps using 96 LDIs. For comparison, the left half shows the result without indirect light. Bottom right shows an enhanced Sponza scene rendered at 15.5 fps. All illumination of shadowed pixels is exclusively obtained through secondary rays (shadow map absorbs all direct light). We see a performance gap between glossy and diffuse BRDFs, which is caused by the following reasons: first, for the upper images we had to trace fewer secondary rays. Second, diffuse rays are less coherent and thus more costly. Filtering also generates small extra costs.

Figure 1 demonstrates the combination of diffuse and glossy BRDFs on different objects in a single scene rendered at 27.0 fps. We used 150 LDIs, 16 secondary rays for the glossy and 8 secondary rays for the diffuse BRDF. While the dragon has a glossy shading (notice the color bleeding from the green wall), the sphere and the lion are glossy.

In summary, we are able to add diffuse global illumination at around 15–25 fps. Glossy BRDFs behave far better and allow higher frame rates or more secondary rays/pixel. It is also possible to apply our method on objects selectively in order to use distinct BRDFs for different models, or just to reduce computation costs.

For larger scenes, an increased LDI resolution is required in order to maintain the same quality. However, this mainly has an impact on precomputation time and on memory consumption, performance is not affected significantly since the number of pixel crossings remains roughly constant.

8 Conclusion

We presented a novel and very efficient method for approximate ray tracing on the GPU. Using low resolution LDIs allows us to cast over 200 million glossy and 50 million diffuse rays/s. We demonstrated that the approximation errors of single rays average out, and consequently LDI ray tracing is ideal for indirect illumination. In order to harness advantages of both rasterization (i.e., anti-aliasing) and ray tracing (indirect illumination), we combined these methods resulting in a hybrid renderer that achieves real-time frame rates.

Acknowledgements We would like the thank Marko Dabrovic for the Sponza and Sibenik model, Frank Meinl for the enhancements on Sponza, the Stanford Computer Graphics Laboratory for the Buddha and Lucy model, and Daniel Vlasic for the animation models.

References

1. Agrawala, M., Ramamoorthi, R., Heirich, A., Moll, L.: Efficient image-based methods for rendering soft shadows. In: Proceedings Siggraph 2000, pp. 375–384. ACM, New York (2000)
2. Aila, T., Laine, S.: Understanding the efficiency of ray traversal on gpus. In: Proceedings of the 1st ACM Conference on High Performance Graphics, pp. 145–149. ACM, New York (2009)
3. Assarsson, U., Akenine-Möller, T.: A geometry-based soft shadow volume algorithm using graphics hardware. ACM Trans. Graph. **22**(3), 511–520 (2003). http://doi.acm.org/10.1145/882262.882300
4. Bavoil, L., Callahan, S., Silva, C.: Robust soft shadow mapping with backprojection and depth peeling. J. Graph. GPU Game Tools **13**(1), 19–30 (2008)
5. Bürger, K., Hertel, S., Krüger, J., Westermann, R.: Gpu rendering of secondary effects. In: Vision, Modeling and Visualization (2007)
6. Dachsbacher, C., Stamminger, M.: Splatting indirect illumination. In: Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games, p. 100. ACM, New York (2006)
7. Forest, V., Barthe, L., Paulin, M.: Accurate shadows by depth complexity sampling. Comput. Graph. Forum **27**, 663–674 (2008)
8. Guennebaud, G., Barthe, L., Paulin, M.: High-quality adaptive soft shadow mapping. Comput. Graph. Forum **26**, 525–533 (2007)
9. Hachisuka, T.: High-quality global illumination rendering using rasterization. In: GPU Gems 2. Addison-Wesley, Reading (2005)
10. Havran, V., Purgathofer, W.: On comparing ray shooting algorithms. Comput. Graph **27**(4), 593–604 (2003)
11. Hertel, S., Hormann, K., Westermann, R.: A Hybrid GPU Rendering Pipeline for Alias-Free Hard Shadows, pp. 59–66. Eurographics Association, Geneva (2009)
12. Keller, A.: Instant radiosity. In: SIGGRAPH '97: Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques, pp. 49–56. ACM Press/Addison-Wesley, New York (1997). doi:http://doi.acm.org/10.1145/258734.258769
13. Lauterbach, C., Garland, M., Sengupta, S., Luebke, D., Manocha, D.: Fast BVH construction on GPUs. Comput. Graph. Forum **28**, 375–384 (2009)
14. NVIDIA: Interactive order-independent transparency (2001). http://developer.nvidia.com
15. Purcell, T.J., Buck, I., Mark, W.R., Hanrahan, P.: Ray tracing on programmable graphics hardware. In: SIGGRAPH '05: ACM SIGGRAPH 2005 Courses, p. 268. ACM, New York (2005). http://doi.acm.org/10.1145/1198555.1198798

16. Ritschel, T., Grosch, T., Kautz, J., Müller, S.: Interactive illumination with coherent shadow maps. In: Proceedings of the Eurographics Symposium on Rendering, Citeseer (2007)
17. Ritschel, T., Grosch, T., Kim, M., Seidel, H., Dachsbacher, C., Kautz, J.: Imperfect shadow maps for efficient computation of indirect illumination. ACM Trans. Graph. **27**(5), 129 (2008)
18. Ritschel, T., Engelhardt, T., Grosch, T., Seidel, H., Kautz, J., Dachsbacher, C.: Micro-rendering for scalable, parallel final gathering. In: ACM SIGGRAPH Asia 2009 papers, pp. 1–8. ACM, New York (2009)
19. Roger, D., Assarsson, U., Holzschuch, N.: Whitted ray-tracing for dynamic scenes using a ray-space hierarchy on the GPU. In: Kautz, J., Pattanaik, S. (eds.) Symposium on Rendering, Rendering Techniques 2007, pp. 99–110. Eurographics Association, Geneva (2007). doi:10.2312/EGWR/EGSR07/099-110
20. Schwarz, M., Stamminger, M.: Bitmask soft shadows. Comput. Graph. Forum **26**, 515–524 (2007)
21. Segovia, B., Iehl, J., Mitanchey, R., Péroche, B.: Non-interleaved deferred shading of interleaved sample patterns. In: Proceedings of the 21st ACM SIGGRAPH/EUROGRAPHICS Symposium on Graphics Hardware, p. 60. ACM, New York (2006)
22. Sengupta, S., Harris, M., Zhang, Y., Owens, J.: Scan primitives for gpu computing. In: Proceedings of the 22nd ACM SIGGRAPH/EUROGRAPHICS Symposium on Graphics Hardware, p. 106. Eurographics Association, Geneva (2007)
23. Shade, J., Gortler, S., He, L., Szeliski, R.: Layered depth images. In: Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques, pp. 231–242. ACM, New York (1998)
24. Sintorn, E., Eisemann, E., Assarsson, U.: Sample based visibility for soft shadows using alias-free shadow maps. Comput. Graph. Forum **27**(4), 1285–1292 (2008)
25. Wald, I., Kollig, T., Benthin, C., Keller, A., Slusallek, P.: Interactive global illumination using fast ray tracing. In: Proceedings of the 13th Eurographics Workshop on Rendering, pp. 15–24. Eurographics Association, Geneva (2002)
26. Wald, I., Ize, T., Kensler, A., Knoll, A., Parker, S.G.: Ray tracing animated scenes using coherent grid traversal. ACM Trans. Graph. **25**(3), 485–493 (2006). http://doi.acm.org/10.1145/1141911.1141913
27. Xie, F., Tabellion, E., Pearce, A.: Soft shadows by ray tracing multilayer transparent shadow maps. Comput. Graph. Forum (Proc. Symposium on Rendering) (2007)

Matthias Nießner is a Ph.D. student at the Computer Graphics Group of the University Erlangen-Nuremberg, Germany. He received his Diploma degree in Computer Science from the same university in 2009. His current research focus is on interactive GPU ray tracing. Furthermore, he is working on perceptual visualization of human vision on modern graphics hardware.

Henry Schäfer is a Ph.D. student at the Computer Graphics Group of the University Erlangen-Nuremberg, Germany. He received his Diploma degree in Computer Science from the same university in 2009. Currently he is researching in the field of interactive global illumination. Moreover, he is working on texture compression techniques using graphics hardware.

Marc Stamminger is a professor of computer graphics at the University of Erlangen-Nuremberg, Germany. He received his Doctor degree in Computer Science from the same university in 1999 for a thesis on finite element methods in global illumination. His current research focus is on real-time rendering and global illumination for complex scenes. Marc Stamminger was program co-chair of Eurographics 2009 in Munich and he is on the editorial board of Computer Graphics Forum.