Fast Global Illumination Baking via Ray-Bundles (sap_0046)

Yusuke Tokuyoshi* Square Enix Co., Ltd.
Takashi Sekine† Square Enix Co., Ltd.
Shinji Ogaki‡ Square Enix Co., Ltd.

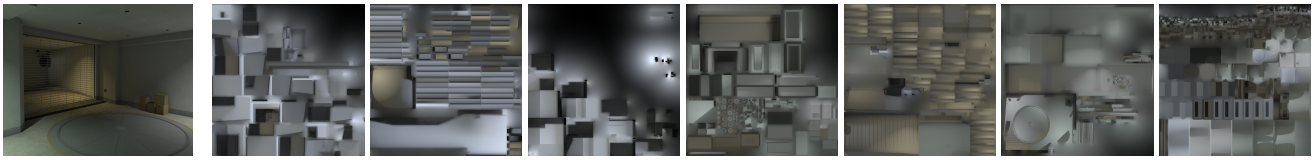


Figure 1: *Left: real-time rendering using light maps (122640 triangles scene). Right: the light maps generated by our renderer (1024×1024 pixels per image). The total rendering time for the light maps is 181 seconds (ray-bundle resolution: 2048×2048 pixels, 10000 directional samples, GPU: NVIDIA GeForce GTX 580).*

1 Introduction

In interactive applications such as video games, light maps are often used to generate realistic images. However, baking light maps is time consuming because it is necessary to compute global illumination. This sketch presents a simple and fast rendering system for light maps. Our system exploits ray-bundles on DirectX®11 capable GPUs and outperforms ray tracing based methods. Furthermore, it supports tessellation for DirectX 11 games.

1.1 Related Work

Ray-bundles The use of global ray-bundles was introduced by Sbert [1996]. Szirmay-Kalos and Purgathofer [1998] used global ray-bundles for global illumination algorithm based on finite elements. They compute radiance exchange between visible surfaces using rasterization. Hachisuka [2005] proposed ray-bundles using rasterization for final gathering. To obtain each depth fragment, depth peeling [Everitt 2001] was used. Hermes et al. [2010] used *k*-buffer [Callahan et al. 2005] and they demonstrated high-quality global illumination with multiple glossy reflections using their radiance exchange.

Per Pixel Linked-list Everitt [2001] introduced a multi-pass rendering method called depth peeling for order independent transparency. Ideally, we would need to store a list of fragments per pixel as A-buffer [Carpenter 1984] in a single-pass. Callahan et al. [2005] proposed *k*-buffer. Although it can be created in a single-pass, the number of fragments per pixel is fixed. Yang et al. [2010] introduced a method to dynamically construct highly concurrent linked-list on DirectX 11 GPUs. This method is faster than depth peeling for calculating order independent transparency, and provides unlimited storage per pixel unlike *k*-buffer. However, the fragments in the list need to be sorted for order independent transparency.

Tessellation DirectX 11 GPUs support hardware tessellation. It enables real-time graphics to use arbitrary tessellation methods. However, it is not easy for offline rendering. In order to obtain exactly the same appearance both in real-time graphics and offline rendering, offline renderers and real-time rendering engines have to use the same tessellation method. The simplest solution is to first tessellate the polygons. However, this is memory consuming. A complex and costly out-of-core rendering system may be needed. Direct ray tracing [Smits et al. 2000; Ogaki and Tokuyoshi 2011] is one of the solutions, but arbitrary tessellation methods are still difficult. Our renderer is able to support the same tessellation methods for real-time graphics without a complex implementation.

*e-mail:tokuyosh@square-enix.com
†e-mail:sekine@square-enix.com
‡e-mail:ogaki@square-enix.com

2 Method

2.1 Ray-Bundle Tracing on the GPU

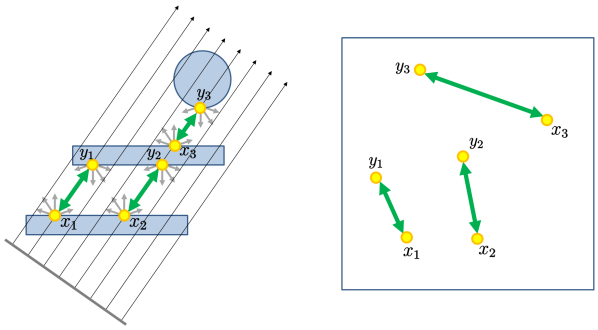
Our algorithm is based on ray-bundle tracing. It focuses on a single global direction, and computes the visibility for all fragments in a scene in parallel as shown Figure 2 (left). This can be done by rendering the scene from the sample direction using parallel projection, similar to rendering a shadow map from a directional light source. In this pass, the fragment data is stored to a buffer. Then, in the next pass, the radiance of a fragment is obtained from the buffer for each shading point. We create ray-bundles using per pixel linked-list construction on DirectX 11 GPU [Yang et al. 2010]. For opaque objects, this can be done much faster because there is no need to sort the fragments unlike order independent transparency.

2.2 Radiance Exchange

We solve the light transport problem by radiance exchange described in [Hermes et al. 2010]. They used texture atlas for intermediate data structures, whereas we use light maps directly. The light maps are updated in an iterative fashion. The light transfer is computed between all pairs of successive points. These pairs can be found using ray-bundles (see Figure 2). Let $\boldsymbol{x}_1$ and $\boldsymbol{y}_1$ be a pair of successive points. The pixel corresponding to $\boldsymbol{x}_1$ in the light maps is updated using the radiance at $\boldsymbol{y}_1$. The approximated radiance at $\boldsymbol{y}_1$ can be obtained by light maps computed in the previous iteration. Similarly, the pixel corresponding to $\boldsymbol{y}_1$ in the light maps is updated using the previous radiance at $\boldsymbol{x}_1$. Although this updating scheme is a biased method, it allows an arbitrary number of interreflections and converges to the truth. The weight of a bounce is reduced according to number of interreflections. Therefore, the bias at the ith bounce is given by:

$$B_i \propto \min\left(\frac{i}{m}, 1\right)\rho^i, \tag{1}$$

where m is the number of samples and ρ the albedo of surfaces. Since we can assume $\rho < 1$ in most scenes, the bias converges to zero.



Ray-bundle Light map

Figure 2: *Ray-bundle tracing and radiance exchange. The sample direction is randomly generated and the light maps are updated in an iterative fashion. A ray-bundle is created as a per pixel linked-list in each iteration. And light transport is computed by looking up the radiance of visible fragments from the light maps which were computed in the previous iteration.*

2.3 Sampling of Ray Directions

Ray-bundles transfer radiance not only into the direction $\boldsymbol{\omega}$, but also into $-\boldsymbol{\omega}$. If the orientation is sampled uniformly, then its probability density function is $p(\boldsymbol{\omega}) = 1/2\pi$. When a scene is lit by an environment map, importance sampling is done according to the emitted radiance of the environment map to reduce variance. We calculate the probability density function as follows:

$$p(\boldsymbol{\omega}) = \frac{L_e(\boldsymbol{\omega}) + L_e(-\boldsymbol{\omega}) + 2L_o}{\int_\Omega L_e(\boldsymbol{\omega}') + L_o d\boldsymbol{\omega}'}, \tag{2}$$

where $L_e(\boldsymbol{\omega})$ is the emitted radiance from the direction $\boldsymbol{\omega}$ of the environment map and L_o is the user-specified offset to account for indirect illumination.

2.4 Tessellation

Since our visibility test is done by rasterization on DirectX 11 GPU, our baking system can easily utilize hardware tessellation. In addition, the domain shader can be shared with the real-time rendering engine. Therefore, we are able to render highly tessellated scene without complex implementation such as an out-of-core algorithm.

3 Results

Figure 1 shows generated light maps. The scene contains 122640 triangles. 7 high-quality light maps are rendered in 181 seconds with NVIDIA GeForce GTX 580. The resolution of ray-bundle is 2048×2048 pixels, and 10000 directions are sampled. The performance of our renderer is over 200 M rays per second on a commodity GPU.

4 Discussion and Future Work

The computational complexity of our method is $O(mn)$, where m is the number of samples and n is the number of primitives. On the other hand, the complexity of general ray tracing based algorithms is $O((n+m)\log n)$, because the acceleration structure construction needs $O(n \log n)$, and ray tracing needs $O(m \log n)$. Therefore, in the case of enormously high-polygon scenes with a huge number of samples, our method may be slower than general ray tracing based methods. However, in real-time applications such as games, the number of primitives is limited by memory size and run-time performance. If a scene is highly tessellated, ray tracing based system may need to use slow direct ray tracing methods or out-of-core algorithms. On the other hand, the use of hardware tessellation is simple and fast. For most scenes, we can achieve sufficient quality with less than 10 thousand samples and our system outperforms ray tracing based baking systems.

Currently, our renderer only supports lambertian materials. However, we can use arbitrary BRDFs with a slight modification as shown in [Hermes et al. 2010]. Ray-bundle based methods are weak in computing highly glossy reflections, especially perfect specular surfaces. If a scene has perfect specular surfaces, we must use ray tracing based methods for caustics. Since our renderer entirely works on the GPU, ray (or photon) tracing can be executed on the CPU in parallel.

If a scene is vast, artifacts may occur since the resolution of ray-bundles is limited by memory size. This problem is addressed by tiling [Thibieroz 2011]. We would like to investigate its effectiveness.

Acknowledgements

The authors would like to thank Toshiya Hachisuka, Arun Mehta and Fabien Gravot for valuable comments, and Takashi Sugata for providing the models.

References

CALLAHAN, S. P., IKITS, M., COMBA, J. L. D., AND SILVA, C. T. 2005. Hardware-assisted visibility ordering for unstructured volume rendering. *IEEE Transactions on Visualization and Computer Graphics 11*, 3, 285–295.

CARPENTER, L. 1984. The a-buffer, an antialiased hidden surface method. *SIGGRAPH Comput. Graph. 18*, 3, 103–108.

EVERITT, C. 2001. Interactive order-independent transparency. Tech. rep., NVIDIA Corporation.

HACHISUKA, T. 2005. High-quality global illumination rendering using rasterization. In *GPU Gems 2*. Addison-Wesley Professional, ch. 38, 615–634.

HERMES, J., HENRICH, N., GROSCH, T., AND MUELLER, S. 2010. Global illumination using parallel global ray-bundles. In *Vision, Modeling and Visualization*.

OGAKI, S., AND TOKUYOSHI, Y. 2011. Direct ray tracing of phong tessellation. *Comput. Graph. Forum 30*, 4, 1337–1344.

SBERT, M. 1996. *The Use of Global Directions to Compute Radiosity - Global Monte Carlo Techniques*. PhD thesis, Catalan Technical University.

SMITS, B., SHIRLEY, P., AND STARK, M. M. 2000. Direct ray tracing of smoothed and displacement mapped triangles. Tech. rep.

SZIRMAY-KALOS, L., AND PURGATHOFER, W. 1998. Global ray-bundle tracing with hardware acceleration. In *in Rendering Techniques '98*, Springer, 247–258.

THIBIEROZ, N. 2011. Order-independent transparency using per-pixel linked lists. In *GPU Pro 2*. AK Peters, ch. VII,2, 409–431.

YANG, J. C., HENSLEY, J., GRÜN, H., AND THIBIEROZ, N. 2010. Real-time concurrent linked list construction on the gpu. *Comput. Graph. Forum 29*, 4, 1297–1304.