DOI: 10.1111/cgf.12239
Pacific Graphics 2013
B. Levy, X. Tong, and K. Yin
(Guest Editors)

Volume 32 (2013), Number 7

Adaptive Ray-bundle Tracing with Memory Usage Prediction: Efficient Global Illumination in Large Scenes

Yusuke Tokuyoshi[1], Takashi Sekine[1], Tiago da Silva[1,2], and Takashi Kanai[2]

[1]Square Enix Co., Ltd., Japan
[2]University of Tokyo, Japan

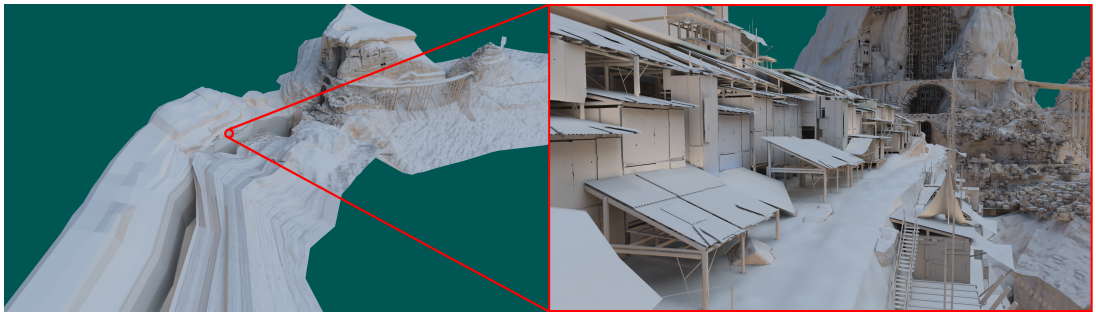


Figure 1: *Indirect illumination represented by 45.3M texel light maps for a large outdoor scene (4.9 km in diameter, 3.7M triangles). The texel distribution of these light maps is strongly inhomogeneous and locally dense. Light map computation time with our method: 1396 secs (GPU: NVIDIA®GeForce®GTX 580).*

Abstract

This paper proposes an adaptive rendering technique for ray-bundle tracing. Ray-bundle tracing can be done by per-pixel linked-list construction on a GPU rasterization pipeline. This rasterization based approach offers significant benefits for the efficient generation of light maps (e.g., hardware acceleration, tessellation, and recycling of shaders used in real-time graphics). However, it is inapplicable to large and complex scenes due to the limited capacity of the GPU memory because it requires a high-resolution frame buffer and high-capacity node buffer for the linked-lists. In addition, memory overflow can potentially occur on the per-pixel linked-list since the memory usage of the lists is usually unknown before the rendering process. We introduce an adaptive tiling technique with memory usage prediction. Our method uses an appropriately tiled frame buffer, thus eliminating almost all of the overflow risks thanks to our adaptive tile subdivision scheme. Using this technique, we are able to render high-quality light maps of large and complex scenes which cannot be computed using previous ray-bundle based methods.

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

1. Introduction

Global illumination effects are perceptually important for interactive applications such as video games and virtual reality systems. While many dynamic global illumination methods have been developed, cheap static solutions such as light maps are often still required, especially for vast background objects as shown in Fig. 1 for which global illumination is difficult to compute in real-time frame rates. However, *baking light maps* (storing precomputed irradiance into texture maps) is also computationally expensive, given the necessity to solve difficult illumination problems for an entire scene.

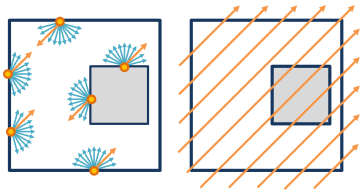


Figure 2: *Focusing on a single global direction, a set of parallel rays can be used instead of shooting random rays from different positions.*

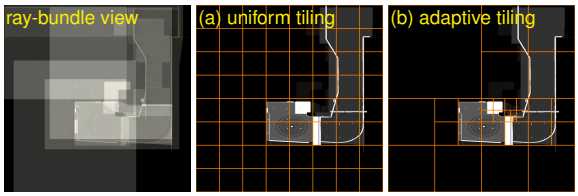


Figure 3: *Our adaptive tiling (b) produces more efficient results than uniform tiling (a) (Brightness represents importance, described in Sect. 3). Tiles are subdivided taking into account not only importance, but also the predicted memory usage of the per-pixel linked-list to reduce the risk of memory overflow.*

Ray-bundle tracing is an acceleration technique leveraging a GPU rasterizer [SKP98, Hac05]. Via ray-bundle tracing, light maps can be efficiently generated [HHGM10, TSO11] by repeatedly rendering the scene from sample directions using a parallel projection, similar to rendering a shadow map from a directional light source (Fig. 2). Multiple depth fragments in a single pixel can be handled by a per-pixel linked-list [Car84, YHGT10, TSO11]. Although fast GPU ray tracers such as NVIDIA OptiX^{TM} [PBD*10] are currently available, this rasterization-based approach allows easy reuse of techniques developed for real-time graphics, such as hardware tessellation with arbitrary displacements. Although this feature is suitable for baking light maps of real-time applications, a huge amount of memory is consumed especially for large and complex scenes.

Light map texel distribution is often strongly inhomogeneous for practical scenes because they are generally given by artists. In order to capture high-density light map texels for a scene, it has to be rendered using a high-resolution ray-bundle buffer. Moreover, for complex scenes, a high-capacity node buffer for the linked-lists is also necessary to store many depth fragments. In addition, the memory usage of such linked-lists is usually unknown before the rendering process. Hence, an excessive amount of memory has to be allocated to avoid overflow. In other words, avoidance of memory overflow given an arbitrary memory capacity is a challenging problem.

Memory reduction for per-pixel linked-list can be addressed by *tiling*. This multi-pass rendering technique splits a render target into smaller tiled regions as shown in Fig. 3. Uniform tiling (Fig. 3 (a)) [Thi11] was introduced for real-time order independent transparency (OIT), but its actual memory usage is still unpredictable. Furthermore, rendering with many tiles causes speed degradation. Since appropriate parameters (e.g., number of tiles, resolution for each tile, and buffer size) are strongly scene-dependent, they have to be chosen through trial and error, which is unsuitable for off-line light map computation. Moreover, uniformly distributed rays are inefficient for inhomogeneous light maps.

This paper proposes an adaptive tiling technique (Fig. 3 (b)) which takes into account the memory usage for off-line rendering. With our method, only the resolution for scene analysis and maximum buffer size for the per-pixel linked-list are specified. Based on the buffer size and light map densities, tiles are subdivided adaptively. Furthermore, the appropriate ray-bundle resolution for each tile is automatically determined. This subdivision scheme does not only adapt the ray-bundle resolution to the light map density, but also almost completely eliminates the memory overflow risks, since tiles are subdivided when overflow is predicted. Our method thus efficiently produces higher-quality images with less parameter tuning than uniform tiling.

The contributions of this paper are as follows:

- A memory usage estimator for linked-list based ray-bundles is introduced to reduce the memory overflow risk.
- Adaptive tile subdivision using the above estimator is demonstrated. As far as we know, this is the first demonstration of the adaptive solution for the resolution issue in ray-bundle based global illumination algorithms.

2. Related Work

Path tracing [Kaj86] is a well-established technique based on Monte Carlo integration to calculate indirect illumination. It is done by evaluating a hemispherical integral via recursive ray tracing. On the other hand, *ray-bundles*, which are a set of parallel rays, are used for the efficient implementation of approximate ray tracing. This method focuses on a single global direction, and computes visibility for all fragments in a scene in parallel as shown in Fig. 2. The use of global ray-bundles was first introduced by Sbert [Sbe96] to compute radiosity. Recently, they are often used for accelerating visibility tests, because they can be implemented using GPU rasterization.

Szirmay-Kalos and Purgathofer [SKP98] proposed hardware-assisted ray-bundle tracing for a finite element based global illumination algorithm, but like OIT, it was difficult to render complex objects such as intersecting triangles. In other wards, this problem can be solved by using techniques developed for OIT. Hachisuka [Hac05] introduced the complete GPU ray-bundle tracing for final gathering. To obtain each depth fragment, a multi-pass method called depth peeling [Eve01] was used. Baking light

maps using this method was also demonstrated [Hac02]. Hermes et al. [HHGM10] used a *k*-buffer [CICS05] as an alternative to depth peeling. High-quality multiple glossy interreflections were achieved due to the use of texture atlases as intermediate data structures. Although the *k*-buffer can be created in a single-pass, the number of fragments per pixel is fixed. Stochastic depth buffering for approximate radiosity [TN11] is more simply processed in a single pass, but it often induces light leaking artifacts.

Our ray-bundle tracing method is based on a per-pixel linked-list [Car84] on a DirectX®11 GPU [YHGT10]. This single-pass linked-list construction is faster than depth peeling, and provides unlimited storage per pixel unlike the *k*-buffer. Tokuyoshi et al. [TSO11] computed light maps using this ray-bundle tracing method. Furthermore, by leveraging GPU tessellation, they inexpensively generated light maps of highly tessellated objects. These linked-list based ray-bundles can also be employed for interactive global illumination. Nießner et al. [NSS10] developed a semi-static method based on pre-computing layered depth images (i.e., per-pixel linked-list) from every direction for a static object. A completely dynamic approach was demonstrated by Tokuyoshi and Ogaki [TO12b]. They introduced bidirectional sampling combining virtual point lights and a few dynamic ray-bundles at interactive frame rates. To generate many ray-bundles in a single-pass, an approximation technique called imperfect ray-bundle tracing [TO12a] was also proposed.

However, ray-bundle tracing via GPU rasterization has critical limitations. Details of large scenes cannot be rendered due to uniformly distributed rays with limited resolution. In this case, details finer than the pixel size are missed and light leaking is often produced. In addition, memory overflow can potentially occur in linked-lists used for ray-bundle tracing.

Since the resolution issue is identical to the aliasing problem of shadow mapping, it can be solved using anti-aliasing methods developed for shadow maps. Perspective shadow maps [SD02, WSP04, MT04, LGQ*08] and cascaded shadow maps (CSMs) [Eng06, LTYM06, ZSXL06] have been developed to adapt the sampling rate to a view frustum for real-time applications. More robust approaches are done by analyzing the scene before creating the shadow map. Lauritzen et al. [LSL11] analyzed the depth distribution in a G-buffer for CSMs. Adaptive shadow mapping (ASM) [FFBG01, LSK*05, LSO07, GW07a, GW07b] is more robust than view frustum based approaches. It builds a quadtree which samples the scene at multiple resolutions based on an analysis of the scene. Tiled shadow mapping [Arv04] is one type of ASM approach using a uniform tile grid. Instead of quadtree-based adaptation, each tile has a different resolution. Such ASMs are generally rendered in a multi-pass fashion. Recently, Rosen [Ros12] proposed rectilinear texture warping shadow maps (RTWSMs). Instead of multi-pass rendering, a simple warping technique is applied to control the sampling rate. Several scene analysis techniques, namely *forward analysis*, *backward analysis*, and *hybrid analysis* were introduced. In this context, the forward and backward methods analyze scenes from the light view or desired view respectively, while the hybrid approach combines both. This paper demonstrates the adaptive ray-bundle tracing based on ASM to solve the resolution issue.

However, memory overflow of the per-pixel linked-list is still an open problem [VF13]. To solve this, Thibieroz [Thi11] reduced memory usage via uniform tiling for real-time OIT, but overflow was still unpredictable. Vasilakis and Fudos [VF12] proposed an S-buffer which linearly organized memory into variable contiguous regions for each pixel with a two-pass rendering. Even though this pointerless structure is memory efficient, the memory had to be reallocated when overflow occurred. Our main contribution is an efficient memory usage prediction for adaptive tiling.

3. Scene Analysis

For shadow mapping, the analysis of importance distribution on a scene is in general used to select the sampling rate. For ray-bundles, our method additionally counts the depth fragments to potentially reduce the risk of memory overflow for the per-pixel linked-lists.

Importance Function for Light Maps In this paper, ray-bundles are adaptively generated according to an importance value which represents the required ray density on a surface. For baking light maps, we define the importance as light map texel density, which is independent of sample direction. In fact, the given texel density viewed from any sample direction is proportional to $\frac{1}{\cos\theta}$, where θ is the angle between the sample direction and surface normal. However, since the incident radiance is weighted by $\cos\theta$ in the rendering equation [Kaj86], we also weight the importance function by $\cos\theta$ for efficiency, similar to the importance sampling of lambertian surfaces. Finally, from $\frac{\cos\theta}{\cos\theta} = 1$, the importance function is given as only the light map texel density viewed from the surface normal direction. In this paper, since importance is constant in a triangle primitive, it is precomputed and stored into each triangle for acceleration. This precomputation is described in the supplemental material.

Analysis for Memory Usage Prediction Unlike shadow maps, linked-list based ray-bundles have the risk of memory overflow. To avoid this overflow, we predict the memory used and set appropriate parameters before creating the ray-bundles. In our analysis, a number of depth fragment counts viewed from a ray-bundle direction is first sampled. The average of these samples is then estimated, being used to calculate the maximum ray-bundle resolution which does not induce memory overflow. Next, our adaptive approach determines the ray-bundle sampling rate based on this estima-

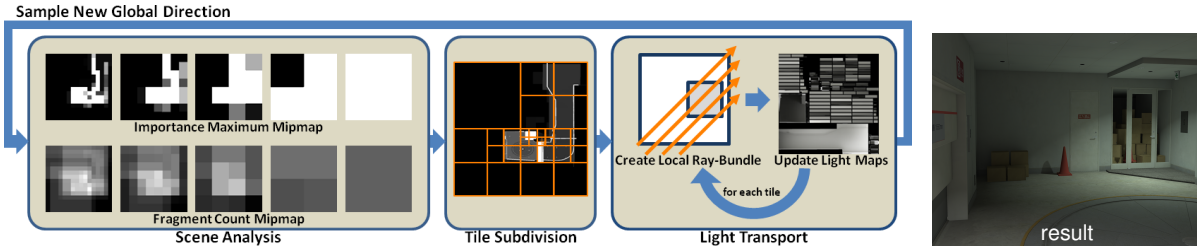


Figure 4: *Our adaptive ray-bundle tracing algorithm for light map baking. The scene analysis phase renders the importance map and fragment count map, and generates their mipmaps. Next, the tile subdivision phase adaptively subdivides the tiles using the mipmaps. Finally, for each tile, a local ray-bundle is created, and light maps are updated.*

tion, unlike deterministic methods such as the S-buffer. Conversely, it means the sampling points of the analysis most often differ from the generated ray-bundle. Our approach therefore still potentially has the risk of memory overflow due to prediction error, but this risk can be more dramatically reduced than without the prediction. To reduce the overflow risk even further, a prediction metric is introduced in Sect. 4.3.

4. Adaptive Tiling for Ray-bundles

In this section, an adaptive tiling method for ray-bundle tracing is introduced based on ASM. As shown in Fig. 4, our algorithm is composed of the following three phases: scene analysis phase, tile subdivision phase, and light transport phase. Our contribution consists of the scene analysis and tile subdivision. The scene analysis builds two 2D mipmaps called *importance map* and *fragment count map*. The importance map represents the required ray density similar to RTWSM. Unlike shadow mapping, the fragment count map is additionally introduced, storing a per-pixel fragment count used to predict memory usage. These maps are built per sample direction by rendering the entire scene viewed from that direction. In the tile subdivision phase, tiles are recursively subdivided by descending the mipmaps and taking memory usage into account. Finally, for each tile, a local ray-bundle is generated, and light maps are updated by computing the light transport as described in [HHGM10, TSO11]. For large scenes, the importance map and fragment count map are often given lower resolution than the generated tiled ray-bundles, but they are sufficient for our tile subdivision scheme.

4.1. Building the Importance Map

An importance map is built by rendering the entire scene using the same parallel projection as the global ray-bundle, similar to the forward analysis in RTWSM. The pixel shader then stores an importance value (i.e., light map density) in the importance map. To take into account multiple depth fragments in a single pixel, the maximum value is stored by using an atomic instruction, unlike the original forward analysis. Once the importance map is rendered, its maximum mipmap [GW07a, TIS08] is generated. This mipmap is built up by computing the maximum value of the four underlying samples. To build a quadtree implicitly, our importance map resolution is given as a power of two. After that, we adaptively subdivide tiles according to this mipmap, and compute the required ray-bundle resolution for each tile from the mipmapped value. We use maximum mipmapping to capture locally dense importance distribution in a tile.

4.2. Building the Fragment Count Map

A fragment count map can be rendered in the same rendering pass as the importance analysis described in Sect. 4.1. The pixel shader then stores a layered fragment count in the fragment count map, which has the same resolution as the importance map. The fragment count is incremented with an atomic instruction. Similar to the importance analysis, once the fragment count map is rendered, its mipmap is generated. Unlike the importance map, this mipmapping is done via a general averaging operation such as the *GenerateMips* function in Direct3D®. Using this mipmap, the averaged fragment count for each tile is estimated for memory usage prediction.

4.3. Tile Subdivision Based on Prediction

This subsection describes our tile subdivision scheme based on the importance map and fragment count map. The ray-bundle resolution required for a tile is determined using the importance map; the fragment count map is used to estimate the upper bound of the resolution limited by the buffer size. If the required ray-bundle resolution is greater than the upper bound, the tile is subdivided in a recursive fashion.

First, we compute the required ray-bundle resolution for a tile. Let i be the mip level and $\mathbf{x}$ be the pixel position corresponding to a tile, then the number of required rays for tile $C_I(i,\mathbf{x})$ is given by computing the product of the tile area $A(i)$ and importance $I(i,\mathbf{x})$ as follows:

$$C_I(i,\mathbf{x}) = A(i)I(i,\mathbf{x}), \tag{1}$$

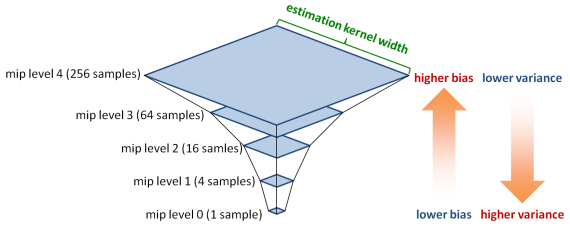


Figure 5: *Kernel based fragment count estimation using mipmaps. A pixel of a low mip level has high variance caused by undersampling, while the upper level has lower variance with higher bias due to wider kernel bandwidth. Since these are different errors, the upper bound of memory usage can be predicted by considering the maximum error for the pixel.*

since the importance $I(i,\mathbf{x})$ obtained from the importance map represents the required ray density. Assuming a square resolution, the required ray-bundle width is $\sqrt{C_I(i,\mathbf{x})}$ pixels.

Next, we compute the upper bound of the ray-bundle resolution. The per-pixel linked-list structure has a head pointer buffer and a node buffer. Let W^2 be the maximum head pointer buffer resolution and N be the node buffer size, then the upper bound of the ray-bundle pixel count can be given by:

$$C_{max}(i,\mathbf{x}) = \min\left(W^2, \frac{\alpha N}{F(i,\mathbf{x})}\right), \tag{2}$$

where $F(i,\mathbf{x})$ is the estimated fragment count per pixel, and $\alpha = [0,1]$ is the user-specified parameter to avoid memory overflow. In this paper, $\alpha = 0.9$ is used in the expectation that the estimation error is less than 10%. The estimated fragment count $F(i,\mathbf{x})$ can be directly obtained from the fragment count map. This estimation is reasonable for large tiles corresponding to the upper mip level, but it induces high prediction error for small tiles because a pixel of the lower mip level has higher variance caused by undersampling. As shown in Fig. 5, our mipmapping based fragment count estimation is a kernel based estimation. There are several estimates due to different kernel widths. The estimation using only pixel $(i,\mathbf{x})$ is unbiased and often produces high variance, whereas the upper level has lower variance with higher bias due to wider kernel bandwidth. Therefore, to reduce overflow risk, Eq. 2 is extended by computing the minimum of the estimated pixel counts above the current level i as:

$$C_{max}(i,\mathbf{x}) = \min\left(W^2, \min_{j=i}^{L}\left(\frac{\alpha N}{F(j,\mathbf{x})}\right)\right), \tag{3}$$

where L is the number of the mipmap levels.

If $C_I(i,\mathbf{x}) > C_{max}(i,\mathbf{x})$, the tile is recursively subdivided because memory overflow is predicted. This subdivision scheme allows less parameter tuning, since only buffer sizes W, N, and L are actually specified depending on the hardware used. Tiles are adaptively subdivided according to the complexity of the input scene. The ray-bundle resolution for each tile is determined by Eq. 1.

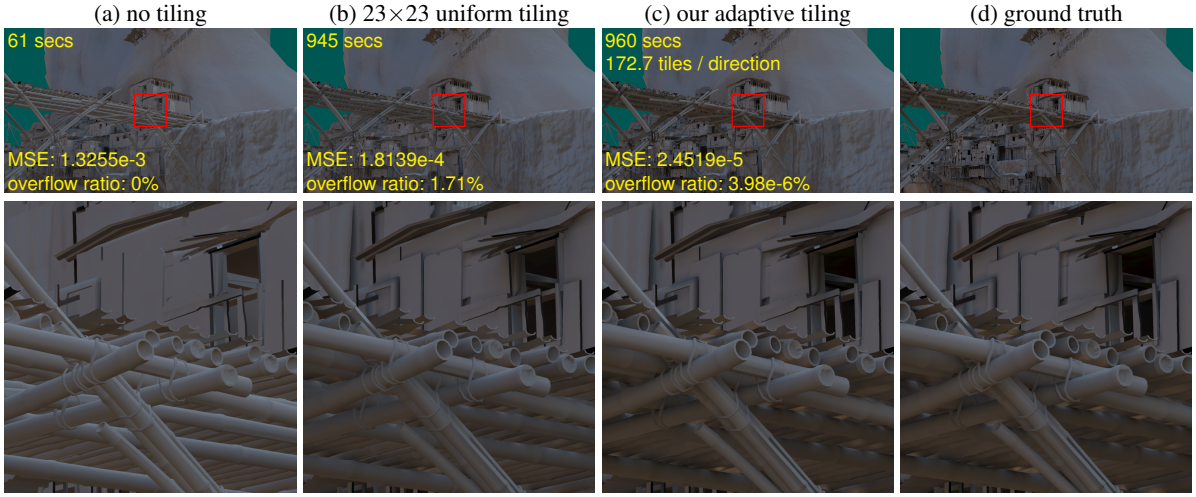
5. Experimental Results

In the following, we show results rendered on an NVIDIA GeForce GTX 580 with 1.5GB memory. The maximum size of the head pointer buffer and node buffer are 1024^2 pixels and 5M nodes respectively. The resolution of the importance map and fragment count map is 1024^2 pixels. The implementation details are described in Appendix A and the supplemental material.

Quality Fig. 6 shows the comparison of light leak errors using the mean squared error (MSE) metric in light map space. The ray-bundle resolution for each tile is the same as the maximum resolution of the head pointer buffer for no tiling (a) and uniform tiling (b), while our method (c) uses only the required resolution given by Eq. 1. The overflow ratio is the ratio of the overflowed node count to the total node count. Computations sufficiently converge with 2000 sample directions for these scenes. Since a ray-bundle without tiling (a) is low-resolution, it cannot render illumination details. Uniform tiling (b) can reduce error by increasing resolution, but there are still noticeable light leaks. Furthermore, uniform tiling often causes memory overflow, which in turn causes visibility of locally-complex objects to be neglected, resulting in additional light leak errors, as shown in the bottom image of Fig. 6 (2)(b). Our method (c) significantly reduces overflow and light leak errors, producing higher-quality images with low-resolution analysis. The light leak error caused by the ray-bundle resolution cannot be removed completely, but it can be reduced without limit by tiling. Although there is still some overflow in the resulting images, it is in fact hardly detectable due to our efficient adaptive tiling. With uniform tiling, overflow is always produced in the same local area such as Fig. 6 (2)(b) bottom. On the other hand, with our adaptive tiling, overflow is stochastically produced, being caused by prediction error. Consequently, unlike uniform tiling, errors can be dispersed.

Performance Table 1 shows the computation time of each procedure in Figs. 6 and 7. The overhead of our method (i.e., the total of Analysis Rendering, Mipmapping, Tile Subdivision and GPU-CPU Data Copy) is about 2% of the total. The computation time of our parallel tile subdivision is about 0.3 ms, while the data transfer time is about 0.7 ms. The total computation time per direction depends on the light map density and scene complexity, since tiles are adaptively subdivided according to them.

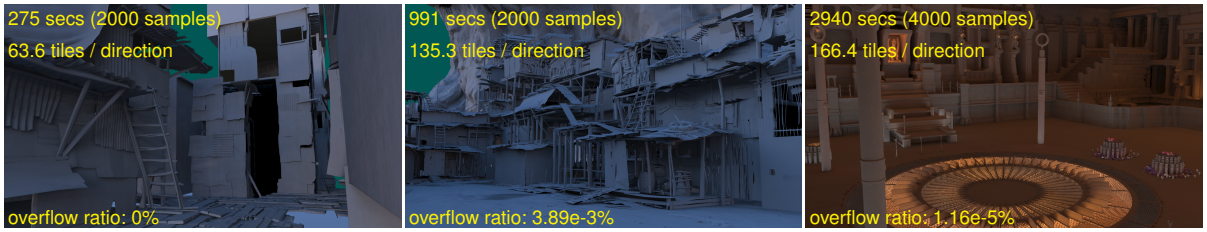
Parameters Fig. 8 shows the computation time per sample direction for the scenes shown in Fig. 7 with different maximum head pointer buffer resolutions and node buffer sizes. Since the computation time strongly depends on the



(1) 27.3M texel light maps, 0.99 km in diameter, and 2.7M triangles scene.
The lower images are close-ups of the upper ones.

(2) The same scene as Fig. 1 (45.3M texel light maps, 4.9km in diameter and 3.7M triangles).
The middle and bottom images are close-ups of the top ones.

Figure 6: *Comparison of light leaks (indirect illumination only). All images are rendered with 2000 sample directions. The tile count of uniform tiling (b) is determined with reference to the same computation time as our adaptive tiling (c). No tiling (a) cannot render the illumination details due to the low-resolution ray-bundles. Uniform tiling (b) often induces memory overflows and light leaks because of locally-complex objects as shown in the bottom image. Our adaptive method (c) produces a closer image to the ground truth and reduces overflow risk.*



(a) 14M texel light maps
0.23 km in diameter, 0.33M triangles

(b) 32 M texel light maps
0.71 km in diameter, 1.1M triangles

(c) 47M texel light maps
0.19 km in diameter, 4.5M triangle

Figure 7: *Experimental scenes (indirect illumination only). The indirect illumination is represented using light maps. The left image (a) is a semi-outdoor scene which has lower resolution light maps and fewer polygons than the other scenes. The middle image (b) is an outdoor scene which has complex objects illuminated by an environment map. The right image (c) is an indoor scene which has a difficult illumination problem since there are many spot lights and small holes. Scenes (a) and (b) were computed with 2000 sample directions, while scene (c) was computed with 4000 sample directions due to illumination difficulty.*

Table 1: *Computation time per sample direction (ms).*

	Fig. 6 (1)	Fig. 6 (2)	Fig. 7 (a)	Fig. 7 (b)	Fig. 7 (c)
Analysis Rendering	7.9	13.3	2.4	7.5	12.5
Mipmapping	0.3	0.5	0.3	0.2	0.3
Tile Subdivision	0.3	0.2	0.3	0.4	0.4
GPU-CPU Data Copy	0.7	0.8	0.6	0.8	0.7
Ray-bundle Creation	291.6	405.7	69.9	269.8	418.9
Light Map Update	180.3	274.7	63.8	217.4	286.9

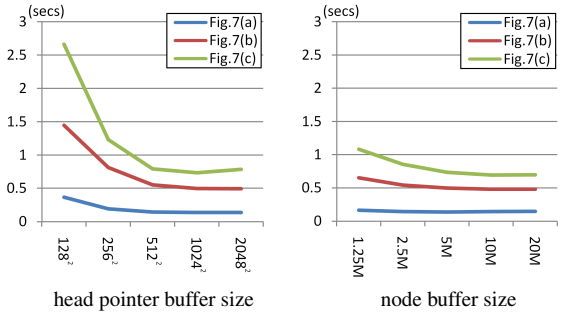


Figure 8: *Computation time per sample direction of the scenes shown in Fig. 7 with different maximum head pointer buffer resolutions (left) and different node buffer sizes (right).*

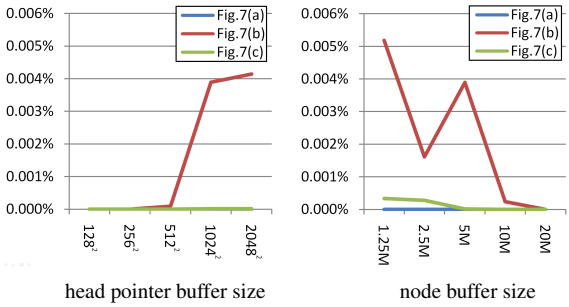


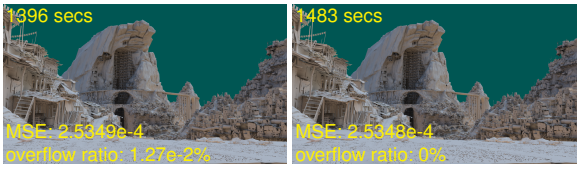
Figure 9: *Overflow ratio of the scenes shown in Fig. 7 with different parameters. Increasing the head pointer buffer size increases overflow, while high-capacity node buffer reduces it.*

tile count, the upper bound resolution given by Eq. 3 should be relaxed. Given that such upper bound is calculated based on both parameters, they both should be increased in order to accelerate the computation. We have also remarked that the computation time slightly increases when the parameters are excessively large. This is because our method employs importance maximum mipmapping to capture locally dense texel distribution, which induces oversampling. The overflow ratio behaves differently regarding these two parameters as shown in Fig. 9. A high-resolution head pointer buffer increases overflow, while a high-capacity node buffer reduces it. Hence, the node buffer size should be increased in proportion to the maximum head pointer buffer resolution within the available memory capacity.

6. Limitations

Memory Overflow Memory overflow is dramatically reduced by using our method, but it cannot be avoided completely. A possible approach to avoid overflow completely should be to subdivide a tile on the CPU and then recompute the ray-bundles when overflow occurs. To check for overflow, the counter value of the node buffer has to be copied from GPU memory to CPU memory for non-unified memory architectures. This recomputation overhead is approximately 6% in our implementation. However, the quality difference is negligible for practical use, as shown in Fig. 10, because the generated light maps are often converted to lower-precision ones for interactive applications.

Analysis With our method, importance maps are rendered in a forward analysis fashion. Using hybrid analysis, conser-



(a) w/o recomputation (b) with recomputation

Figure 10: *Memory overflow can be completely avoided by subdividing the tile and re-creates the ray-bundles when overflow occurs. However, the quality difference is indistinguishable, whereas the recomputation overhead is about 6%.*

Table 2: *False alarm ratio (the unnecessary split count / the total split count) and overflow ratio.*

		$\alpha = 0.9$	$\alpha = 1.0$
Fig. 6 (1)	false alarm ratio:	34.3%	20.2%
	overflow ratio:	3.98e-6%	8.81e-3%
Fig. 6 (2)	false alarm ratio:	35.3%	21.9%
	overflow ratio:	1.27e-2%	3.25e-2%

vative importance distribution can be produced. Since backward analysis is suitable for deferred shading, it can be faster than forward analysis for real-time applications. However, for light map computation of large scenes, the use of backward analysis is inefficient due to high-resolution light maps, and induces more memory access conflicts with atomic operations. For our scene shown in Fig. 1, backward analysis time was 793 ms whereas forward analysis time was 13.3 ms. Hence, we employed only forward analysis, which has the side-effect of possibly missing scene details, excluding them from the importance map. However, this is not a big problem for our mipmap based tiling technique, since all details are collapsed.

False Alarms As shown in Table 2, our overflow prediction has a trade-off between false alarms and memory overflow. These false alarms little affect image quality, but they increase tile count and reduce computation efficiency. False alarm reduction is our future work for acceleration.

7. Discussions

Ray-bundle Warping Our adaptive tiling method can produce many tiles and oversampled areas due to the importance maximum mipmap. These redundant computations can be reduced with warping [Ros12], if the GPU has a fast tessellator. Instead of maximum mipmapping, general average mipmapping is applied to reduce the tile count. The ray distribution of the local ray-bundle can be adapted to the importance distribution by warping with tessellation, as shown in Fig. 11. However, the challenging problem is predicting memory usage for warped ray-bundles. The fragment

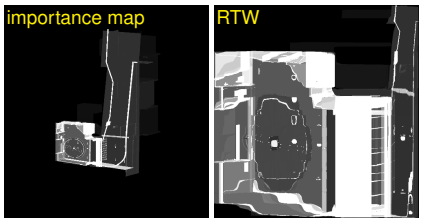


Figure 11: *RTW ray-bundle tracing for a single tile. The ray-bundle can be warped according to the importance map with GPU tessellation.*

count estimation must be modified. Assuming an ideal warping whose pixel size is proportional to its importance, the warped fragment count per pixel $\hat{F}(i,\mathbf{x})$ can be estimated by computing a weighted average as follows:

$$\hat{F}(i,\mathbf{x}) = \frac{\sum_{(0,\mathbf{y})\in(i,\mathbf{x})} F(0,\mathbf{y})I(0,\mathbf{y})}{\sum_{(0,\mathbf{y})\in(i,\mathbf{x})} I(0,\mathbf{y})} = \frac{F_w(i,\mathbf{x})}{I(i,\mathbf{x})}, \quad (4)$$

where $F_w(0,\mathbf{x}) = F(0,\mathbf{x})I(0,\mathbf{x})$ and $F_w(i,\mathbf{x})$ is obtained via mipmapping. Although available warping techniques such as RTW are only approximations and, therefore may not be ideal, they can be applicable if sufficiently accurate for our purposes. However, since the warping quality depends on analysis accuracy, details of the importance map should be rendered unlike in the maximum mipmapping strategy which collapse details. As mentioned in Sect. 6, hybrid analysis can render details, but it is computationally expensive for light map computation. We believe warping for each tile is a better solution, when a fast tessellator and efficient analysis methods are available.

Improving the Analysis Accuracy The analysis accuracy can be improved by increasing the resolution of the importance map and fragment count map, but it is memory inefficient for our mipmap based tiling. To improve the accuracy keeping the resolution, supersampling can be used. Since our analysis uses atomic maximum and increment operations, supersampling without additional storage cost can be performed with more memory access conflicts. However, supersampling can still miss fragments for small triangles. To avoid it, conservative rasterization [HAMO05] can be used for the importance map. This conservative rasterization can be done with a geometry shader for current GPUs. Such supersampling and conservative rasterization should be considered for larger scenes.

Data Compression for Analysis Our current importance and fragment count map size is 32 bits per pixel, as atomic instructions are restricted to 32 bits in DirectX 11.2 or OpenGL®4.4, but 16 bits per pixel is sufficient for a reliable analysis. 16 bits atomic operations can be emulated using a loop and compare-and-swap instruction, similar to the emulations described in [CG12]. In the future, we would like to implement these and evaluate their performance.

Other Applications This paper demonstrated a light map computation method using adaptive ray-bundle tracing. Our adaptive tiling is also applicable for other ray-bundle based applications such as final gathering [Hac05], ambient occlusion, and interactive global illumination [NSS10, TO12b]. If a scene is rendered from a camera view, an efficient backward analysis method and the importance functions described in [Ros12] can be used. As previously mentioned, an importance map should be generated by hybrid analysis for reliable ray-bundle warping. Real-time OIT using per-pixel linked-list is another application. Our adaptive tiling can avoid memory overflow for an arbitrary memory capacity. Although our tiling method requires overhead for data transfer from GPU memory to CPU memory for non-unified memory architectures, this is reducible for unified memory architectures such as PlayStation®4.

8. Conclusions

This paper has presented an adaptive tiling technique for linked-list based ray-bundle tracing. This per-pixel linked-list has the critical limitation of the risk of memory overflow. The proposed technique does not only adapt the ray-bundle resolution to importance distribution similar to ASM approaches, but also reduces the overflow risk by subdividing tiles when overflow is predicted. To estimate memory usage, fragment count analysis and a mipmapping based estimation were introduced. Since our method dramatically reduces overflow risk with less parameter tuning, large and complex scenes can now be efficiently rendered with ray-bundle tracing. This paper demonstrated baking light maps of large scenes. Although our method cannot avoid overflow risks completely, the resulting error is almost undetectable in the resulting images. In addition, overflows can be erased completely by recomputing the ray-bundles with a 6% overhead.

We believe our technique is also effective for general global illumination rendering from a camera view or other multi-fragment rendering such as OIT. In addition, our method can be optimized for those applications. In future work, we would like to investigate such applications and optimizations.

Acknowledgements

The polygon models are courtesy of Square Enix Agni's Philosophy project, Visual Works, and Advanced Technology Division. The authors would like to thank the anonymous reviewers for valuable comments and helpful suggestions.

Appendix A: Single-pass Tile Subdivision on a GPU

The recursive tile subdivision should be performed on a GPU, because the importance map and fragment count map are available in GPU memory. For such subdivision or tree construction, a multi-pass algorithm which parallelizes each level is generally employed [GPM11, ZGHG11, CG12], but it produces synchronization overhead and induces more programming complexity. Therefore, we introduce a simple single-pass algorithm.

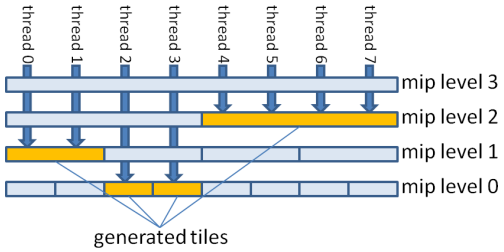


Figure 12: *Our single-pass parallel tile subdivision on a GPU. A thread is launched for each pixel on the finest mip level, and descends the mipmaps with duplicate calculations.*

Program 1: *Pseudo code of parallel tile subdivision.*

```
void TileSubdivision(const uint2 id : SV_DispatchThreadID) {
  float Cmax = min(W^2, α*N/FragmentCountMap[L][(0, 0)]);
  float A = GetGlobalRayBundleArea();
  for(uint i=L; i>0; --i) {
    const uint TileWidth = 2^i;
    const uint2 pos = id / TileWidth;
    const float CI = A * ImportanceMap[i][pos];
    Cmax = min(Cmax, α*N/FragmentCountMap[i][pos]);
    if(CI < Cmax) {
      const uint2 outputId = pos * TileWidth;
      if(CI > 0 && id.x == outputId.x && id.y == outputId.y)
        TileBuffer.Append(id, TileWidth, sqrt(CI));
      return;
    }
    A /= 4;
  }
  const float CI = A * ImportanceMap[0][id];
  Cmax = min(Cmax, α*N/FragmentCountMap[0][id]);
  TileBuffer.Append(id, 1, sqrt(min(CI, Cmax)));
}
```

Program 1 is the pseudo code of our parallel algorithm. The system value *id* is the thread ID. As shown in Fig. 12, a thread is created for each pixel in the finest mip level. Descending the mip level from the top, each thread evaluates the subdivision condition described in Sect. 4.3. If the tile is subdivided, the thread descends to the next level. Otherwise it outputs the final tile information (i.e., tile position, tile width, and ray-bundle resolution) to *TileBuffer*, and then the algorithm terminates. Since an identical tile can be evaluated by several threads, the output operation is restricted to a single thread to avoid tile duplication. For acceleration, the iteration to calculate Eq. 3 is combined with the descending iteration. Although there are many duplicate calculations, especially for the upper mip level, this is not so computationally expensive since the upper level is cache efficient and there is no branch divergence.

After the subdivision process, the resulting tile information is copied from the GPU memory to the CPU memory for non-unified memory architectures with overhead. This is because ray-bundle creation commands such as viewport setting and draw calls have to be processed by a CPU on current CPU-GPU systems.

References

[Arv04] ARVO J.: Tiled shadow maps. In *Proc. of CGI 2004* (2004), pp. 240–247. 3

[Car84] CARPENTER L.: The a-buffer, an antialiased hidden surface method. *SIGGRAPH Comput. Graph. 18* (1984), 103–108. 2, 3

[CG12] CRASSIN C., GREEN S.: Octree-based sparse voxelization using the gpu hardware rasterizer. In *OpenGL Insights*. CRC Press, 2012, ch. 22, pp. 303–319. 8, 9

[CICS05] CALLAHAN S. P., IKITS M., COMBA J. L. D., SILVA C. T.: Hardware-assisted visibility ordering for unstructured volume rendering. *IEEE TVCG 11* (2005), 285–295. 3

[Eng06] ENGEL W.: Cascaded shadow maps. In *ShaderX5*. Charles River Media, 2006, pp. 197–206. 3

[Eve01] EVERITT C.: *Interactive Order-Independent Transparency*. Tech. rep., NVIDIA Corporation, 2001. 2

[FFBG01] FERNANDO R., FERNANDEZ S., BALA K., GREENBERG D. P.: Adaptive shadow maps. In *Proc. of SIGGRAPH 2001* (2001), pp. 387–390. 3

[GPM11] GARANZHA K., PANTALEONI J., MCALLISTER D.: Simpler and faster hlbvh with work queues. In *Proc. of HPG 2011* (2011), pp. 59–64. 9

[GW07a] GIEGL M., WIMMER M.: Fitted virtual shadow maps. In *Proc. of GI 2007* (2007), pp. 159–168. 3, 4

[GW07b] GIEGL M., WIMMER M.: Queried virtual shadow maps. In *Proc. of I3D 2007* (2007), pp. 65–72. 3

[Hac02] HACHISUKA T.: Parthenon renderer, 2002. URL: http://www.bee-www.com/parthenon/. 3

[Hac05] HACHISUKA T.: High-quality global illumination rendering using rasterization. In *GPU Gems 2*. Addison-Wesley Professional, 2005, ch. 38, pp. 615–634. 2, 9

[HAMO05] HASSELGREN J., AKENINE-MÖLLER T., OHLSSON L.: Conservative rasterization. In *GPU Gems 2*. Addison-Wesley Professional, 2005, ch. 42, pp. 677–690. 8

[HHGM10] HERMES J., HENRICH N., GROSCH T., MUELLER S.: Global illumination using parallel global ray-bundles. In *Proc. of VMV 2010* (2010), pp. 65–72. 2, 3, 4

[Kaj86] KAJIYA J. T.: The rendering equation. *SIGGRAPH Comput. Graph. 20* (1986), 143–150. 2, 3

[LGQ*08] LLOYD D. B., GOVINDARAJU N. K., QUAMMEN C., MOLNAR S. E., MANOCHA D.: Logarithmic perspective shadow maps. *ACM Trans. Graph. 27*, 4 (2008), 106:1–106:32. 3

[LSK*05] LEFOHN A., SENGUPTA S., KNISS J., STRZODKA R., OWENS J. D.: Dynamic adaptive shadow maps on graphics hardware. In *ACM SIGGRAPH 2005 Sketches* (2005). 3

[LSL11] LAURITZEN A., SALVI M., LEFOHN A.: Sample distribution shadow maps. In *Proc. of I3D 2011* (2011), pp. 97–102. 3

[LSO07] LEFOHN A., SENGUPTA S., OWENS J. D.: Resolution-matched shadow maps. *ACM Trans. Graph. 26*, 4 (2007), 20:1–20:17. 3

[LTYM06] LLOYD B., TUFT D., YOON S.-E., MANOCHA D.: Warping and partitioning for low error shadow maps. In *Proc. of EGSR 2006* (2006), pp. 215–226. 3

[MT04] MARTIN T., TAN T.-S.: Anti-aliasing and continuity with trapezoidal shadow maps. In *Proc. of EGSR 2004* (2004), pp. 153–160. 3

[NSS10] NIESSNER M., SCHAFER H., STAMMINGER M.: Fast indirect illumination using layered depth images. *Vis. Comput. 26*, 6-8 (2010), 679–686. 3, 9

[PBD*10] PARKER S. G., BIGLER J., DIETRICH A., FRIEDRICH H., HOBEROCK J., LUEBKE D., MCALLISTER D., MCGUIRE M., MORLEY K., ROBISON A., STICH M.: Optix: a general purpose ray tracing engine. *ACM Trans. Graph. 29*, 4 (2010), 66:1–66:13. 2

[Ros12] ROSEN P.: Rectilinear texture warping for fast adaptive shadow mapping. In *Proc. of I3D 2012* (2012), pp. 151–158. 3, 8, 9

[Sbe96] SBERT M.: *The Use of Global Directions to Compute Radiosity - Global Monte Carlo Techniques*. PhD thesis, Catalan Technical University, 1996. 2

[SD02] STAMMINGER M., DRETTAKIS G.: Perspective shadow maps. *ACM Trans. Graph. 21*, 3 (2002), 557–562. 3

[SKP98] SZIRMAY-KALOS L., PURGATHOFER W.: Global ray-bundle tracing with hardware acceleration. In *Rendering Techniques '98* (1998), pp. 247–258. 2

[Thi11] THIBIEROZ N.: Order-independent transparency using per-pixel linked lists. In *GPU Pro 2*. AK Peters, 2011, ch. VII,2, pp. 409–431. 2, 3

[TIS08] TEVS A., IHRKE I., SEIDEL H.-P.: Maximum mipmaps for fast, accurate, and scalable dynamic height field rendering. In *Proc. of I3D 2008* (2008), pp. 183–190. 4

[TN11] THOMSEN A., NIELSEN K. H.: Approximate radiosity using stochastic depth buffering. *Journal of Graphics, GPU, and Game Tools 15* (2011), 225–234. 3

[TO12a] TOKUYOSHI Y., OGAKI S.: Imperfect ray-bundle tracing for interactive multi-bounce global illumination. In *HPG 2012 Posters* (2012). 3

[TO12b] TOKUYOSHI Y., OGAKI S.: Real-time bidirectional path tracing via rasterization. In *Proc. of I3D 2012* (2012), pp. 183–190. 3, 9

[TSO11] TOKUYOSHI Y., SEKINE T., OGAKI S.: Fast global illumination baking via ray-bundles. In *ACM SIGGRAPH Asia 2011 Technical Sketches* (2011), pp. 25:1–25:2. 2, 3, 4

[VF12] VASILAKIS A. A., FUDOS I.: S-buffer: Sparsity-aware multi-fragment rendering. In *EG 2012 Short Papers* (2012), pp. 101–104. 3

[VF13] VASILAKIS A. A., FUDOS I.: Depth-fighting aware methods for multifragment rendering. *IEEE TVCG 19*, 6 (2013), 967–977. 3

[WSP04] WIMMER M., SCHERZER D., PURGATHOFER W.: Light space perspective shadow maps. In *Proc. of EGSR 2004* (2004), pp. 143–151. 3

[YHGT10] YANG J. C., HENSLEY J., GRÜN H., THIBIEROZ N.: Real-time concurrent linked list construction on the gpu. *Comput. Graph. Forum 29* (2010), 1297–1304. 2, 3

[ZGHG11] ZHOU K., GONG M., HUANG X., GUO B.: Data-parallel octrees for surface reconstruction. *IEEE TVCG 17*, 5 (2011), 669–681. 9

[ZSXL06] ZHANG F., SUN H., XU L., LUN L. K.: Parallel-split shadow maps for large-scale virtual environments. In *Proc. of VRCIA 2006* (2006), pp. 311–318. 3