Journal of Computer Graphics Techniques Vol. 5, No. 1, 2016 http://jcgt.org

Stochastic Light Culling

Yusuke Tokuyoshi
Square Enix Co., Ltd.

Takahiro Harada
Advanced Micro Devices, Inc.

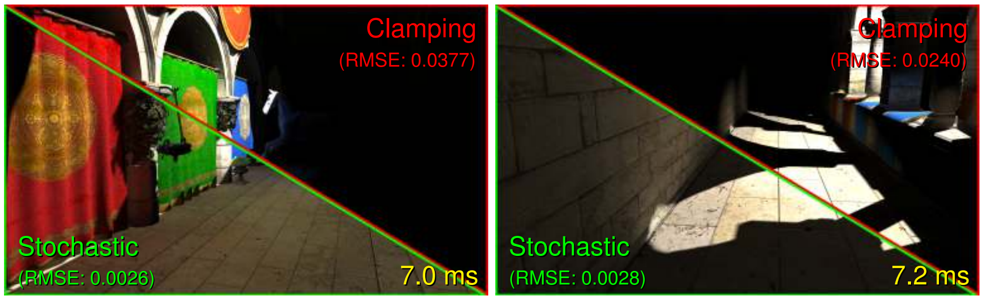


Figure 1. Equal time comparison between tiled lighting with clamping light ranges (red) and our stochastic tiled lighting (green) for 65,536 VPLs without shadow maps (264K triangles scene, 1920×1152 resolution, GPU: AMD Radeon™ R9 290X). Clamping-based approach produces darkening bias, while our method does not. (The scene is courtesy of F. Meinl, R. W. Sumner, and J. Popovic.)

Abstract

Light culling techniques (e.g., *tiled lighting*) with clamping light ranges are often used in real-time applications such as video games. However, they can produce noticeable image darkening for many lights, because a bias is introduced as the part of the light emission farther than the clamping range is always disregarded. To avoid such undesirable bias, the method proposed in this paper uses a stochastic light culling method that randomly determines a light range using *Russian roulette*. When this random range is calculated based on a user-specified error bound, our method produces a sublinear cost for shading after the culling process due to the unbiasedness. Our method changes the fall-off function of light stochastically. Since this approach is independent from culling techniques, any existing light culling frameworks can be used. In addition, our stochastic light culling is not only applicable to real-time rendering, but also offline rendering using path tracing. For randomly distributed shading points produced by a multi-bounce path tracer, we also introduces a bounding sphere tree-based light culling algorithm and its GPU implementation. Using our method, we are able to render tens of thousands of light sources with a smaller error than previous techniques for both real-time and offline rendering.

1. Introduction

For real-time or interactive rendering, light culling techniques such as splatting-based approaches [Dachsbacher and Stamminger 2006; Nichols and Wyman 2010] and *tiled lighting* [Olsson and Assarsson 2011; Harada et al. 2012; Olsson et al. 2012] have been developed to handle many light sources. However, these techniques assume a limited influence range of light, even though physically-based light sources (including virtual point lights (VPLs) [Keller 1997]) have an infinite influence range. If this range is clamped for light culling, noticeable bias and inconsistent estimation are induced for many lights (Figure 1). Furthermore, although the number of light sources to be evaluated per shading point is reduced by culling, it is still linear with respect to the total number of light sources when using constant light ranges.

To avoid these problems, this paper proposes a stochastic light culling method that randomly determines the influence range using *Russian roulette* [Arvo and Kirk 1990] for each light source. When this random range is calculated based on a user-specified error bound, the number of lights to be evaluated per shading point is sublinear with respect to the total number of lights unlike the clamping-based approaches. Although the culling process before shading still has a linear cost similar to existing approaches, it has a negligibly small overhead compared to the shading process. The proposed method is simple and easy to integrate into any existing real-time light culling frameworks. In this paper, we demonstrate a stochastic tiled lighting technique for VPL-based single-bounce global illumination as an example.

Even though our light culling method is unbiased, real-time rendering systems can have other biases due to, for example, shadow mapping and other commonly employed real-time algorithms. Therefore, this paper introduces a light culling method for path-tracing–based [Kajiya 1986] unbiased offline rendering. To apply stochastic light culling to path tracing, this paper proposes a tree-based culling algorithm. This algorithm performs efficiently for progressive rendering. We show that practical scenes with tens of thousands of area lights can be rendered by using our method. To implement this algorithm on the GPU, this paper also describes optimization techniques.

The contributions of our paper are as follows:

- Unbiased light culling using a stochastic fall-off function is proposed (Section 3). Using this method, the number of light sources to be evaluated per shading point is sublinear with respect to the total number of lights for a user-specified error bound.

- Tiled lighting using the proposed method is demonstrated (Section 4). By combining with interleaved sampling, tens of thousands of VPLs can be rendered at real-time frame rates on a commodity GPU.

- For the above mentioned VPL-based global illumination algorithm, this paper also describes a simple acceleration technique for imperfect shadow maps (ISMs) [Ritschel et al. 2008a] using a finite light range.
- A practical area light sampling method based on stochastic light culling is introduced for progressive path tracing (Section 5).

2. Background

2.1. Related Work

Many-lights rendering. VPLs are often used for representing indirect illumination [Keller 1997]. For real-time rendering, single-bounce VPLs lit from a point or directional light can be generated by using reflective shadow maps (RSMs) [Dachsbacher and Stamminger 2005]. To generate shadow maps for many lights, Ritschel et al. [2008a] proposed ISMs. They also proposed bidirectional reflective shadow mapping (BRSM) and adaptive imperfect shadow maps (AISMs) to take view-dependent importance into account [Ritschel et al. 2011]. For radiance evaluation of many lights, sophisticated methods such as lightcuts [Walter et al. 2005] have been developed, but they are focused on offline rendering. In order to roughly estimate the radiance at real-time frame rates, interleaved sampling and geometry-aware filtering have often been used [Segovia et al. 2006; Ritschel et al. 2011]. Although this approach can be a consistent estimator, light sources are sampled uniformly. This uniform sampling often produces a large variance or oversampling problem. To avoid these problems, our stochastic light culling samples light sources according to their fall-off function.

Light culling. Dachsbacher and Stamminger [2006] rendered indirect illumination by splatting bounding geometries of VPLs. Nichols and Wyman [2010] proposed an adaptive multiresolution approach to reduce the fill rate of splatting. Tiled lighting is a well-established light culling technique used in recent video games [Balestra and Engstad 2008; Andersson 2011]. In this technique, lights are binned into 2D screen-space tiles by limiting the influence range of light. This can accelerate shading, unless a tile spans a large depth range. To improve the culling performance in the presence of large depth ranges in a tile, clustered shading [Olsson et al. 2012] and 2.5D culling [Harada 2012] were proposed.

However, the above light culling approaches assume limited light ranges. Therefore, such light culling has not been used for accurate rendering. Our approach avoids this problem by introducing a stochastic fall-off function.

Shadowing for light culling. In the lighting techniques mentioned above, shadow calculation has also been investigated. Harada et al. [2013] demonstrated ray-traced shadows in conjunction with a tiled forward-rendering method called *Forward+*

[Harada et al. 2012]. Olsson et al. [2014] developed virtual shadow maps using clustered shading. The target application of virtual shadow maps is hundreds of light sources, sparsely distributed in a scene, for direct illumination. For these lights, relatively accurate hard shadows are usually employed. In contrast, our target application is over thousands of virtual point lights (i.e., indirect illumination) that produce soft indirect shadows. For this application, ISMs are often used [Ritschel et al. 2008a; Ritschel et al. 2011]. However, existing ISMs do not take light ranges into account, while virtual shadow maps do. Therefore, we introduce a light range-based culling technique for ISMs in this paper.

2.2. Naïve Radiance Evaluation

For a shading point, $\mathbf{x}$, lit by point lights, the radiance, $L(\mathbf{x}, \hat{\omega})$, viewed from the direction $\hat{\omega}$ is given by

$$L(\mathbf{x}, \hat{\omega}) = \sum_{i}^{N} I_i(-\hat{\omega}_i') f(\|\mathbf{x}_i - \mathbf{x}\|) V(\mathbf{x}, \mathbf{x}_i) \rho(\mathbf{x}, \hat{\omega}, \hat{\omega}_i') \max(\hat{\omega}_i' \cdot \hat{n}, 0),$$

where N is the number of light sources, $\mathbf{x}_i$ is the position of the ith light source, $\hat{\omega}_i' = \frac{\mathbf{x}_i - \mathbf{x}}{\|\mathbf{x}_i - \mathbf{x}\|}$ is the light direction, $I_i(-\hat{\omega}_i')$ is the radiant intensity of the light source, $f(\|\mathbf{x}_i - \mathbf{x}\|)$ is the fall-off function, $V(\mathbf{x}, \mathbf{x}_i)$ is the visibility between $\mathbf{x}$ and $\mathbf{x}_i$ to represent shadow, $\rho(\mathbf{x}, \hat{\omega}, \hat{\omega}_i')$ is the bidirectional reflectance distribution function (BRDF), and $\hat{n}$ is the surface normal at $\mathbf{x}$. For physically-based lights, the fall-off function is defined as

$$f(l) = \frac{1}{l^2},$$

where $l = \|\mathbf{x}_i - \mathbf{x}\|$. This function is monotonically decreasing, but always larger than zero (i.e., it has an infinite influence range). Therefore, light culling techniques cannot be directly applied for this fall-off function.

3. Stochastic Light Culling

3.1. Stochastic Fall-off Function

This section introduces a stochastic fall-off function for unbiased light culling. If a light source is unimportant for a shading point, $\mathbf{x}$, this light can be stochastically rejected for shading. This is done by using a Russian roulette technique that randomly samples $f(l)$ by probability $p_i(l) \in [0, 1]$. This probability represents the spatially-varying importance, which is recommended to be proportional to $f(l)$ for efficiency. Therefore, this paper uses the following probability:

$$p_i(l) = \min\left(\frac{f(l)}{\alpha_i}, 1\right), \tag{1}$$

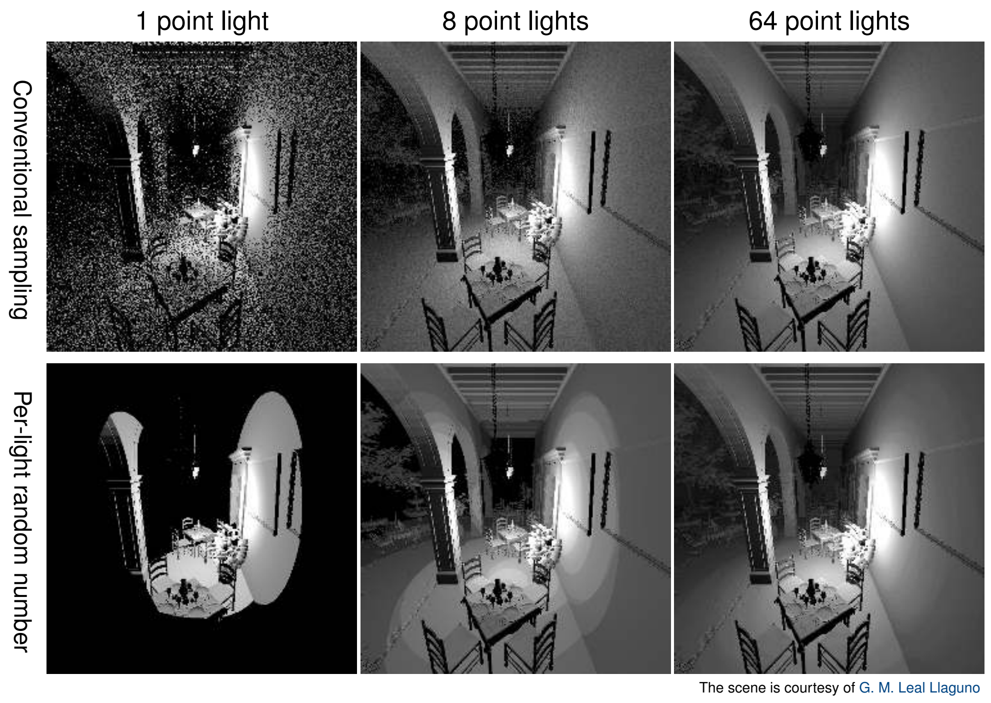


Figure 2. Artifacts of the fall-off function using Russian roulette. The same α_i is used for all images. Generating a random number ξ_i for each light source (bottom row), a finite light range is obtained. For this implementation, variance is visible as banding artifacts.

where α_i is a constant value to control variance. The fall-off function approximated by Russian roulette is given by

$$f(l) \approx \begin{cases} \frac{f(l)}{p_i(l)} = \max\left(\alpha_i, f(l)\right), & (p_i(l) > \xi_i) \\ 0, & (\text{otherwise}) \end{cases}, \tag{2}$$

where $\xi_i \in [0, 1)$ is the uniform random number.

A straightforward implementation draws a random number for each shading point. However, the proposed method uses a single random number ξ_i for a light source, and all the shading points will use the same value for a given light. The advantage of using a single random number is that we can bound the influence range for each light which allows us to cull lights in an unbiased fashion (Figure 2). Since $p_i(l)$ is monotonically decreasing, the range r_i is derived from $p_i(r_i) = \xi_i$ as follows:

$$r_i = f^{-1}\left(\alpha_i \xi_i\right) = \frac{1}{\sqrt{\alpha_i \xi_i}}. \tag{3}$$

Hence, if $\|\mathbf{x}_i - \mathbf{x}\| \geq r_i$, the light source can be culled before shading. This light culling has a trade-off between the variance and computation time, which is controlled by α_i. This occurs because a smaller range, which induces more aggressive culling, can be produced by using larger α_i.

3.2. Sublinear Shading Cost Using an Error Bound

In order to easily control variance, in this section we determine α_i based on a user-specified error bound. In addition, using this error-bound–based α_i, the number of VPLs to be evaluated per shading point is sublinear with respect to the total number of VPLs, while the existing clamping approach has a linear shading cost (Appendix A).

The variance of the stochastic fall-off function is given by

$$(\sigma_i(l))^2 = \left(\frac{1}{p_i(l)} - 1\right)(f(l))^2 . \tag{4}$$

Thus, the imaged error for a single light source is represented as follows:

$$\epsilon_i(\mathbf{x}, \hat{\omega}) = \sigma_i(\|\mathbf{x}_i - \mathbf{x}\|) E I_i(-\hat{\omega}'_i) V(\mathbf{x}, \mathbf{x}_i) \rho(\mathbf{x}, \hat{\omega}, \hat{\omega}'_i) \max(\hat{\omega}'_i \cdot \hat{n}, 0), \tag{5}$$

where E is the exposure scale parameter of the camera. The maximum values are assumed for each term of Equation (5) in order to estimate the view-independent error bound. From Equation (4), $\max_l(\sigma_i(l)) = \frac{\alpha_i}{2}$ is obtained. The maximum visibility is trivially equal to one. The maximum value of reflection lobes should also be assumed, but it is often difficult in practice. Instead, for simplicity, BRDFs are assumed to be the Lambert model as: $\rho(\mathbf{x}, \hat{\omega}, \hat{\omega}'_i) = \frac{1}{\pi}$. Therefore, the reflection lobes can be represented by the cut cosine whose maximum value is one. Thus, the error bound is estimated by

$$\epsilon_{\max} = \frac{\alpha_i E \max_{\hat{\omega}'}(I_i(\hat{\omega}'))}{2\pi}.$$

In other words, α_i is defined by using a user-specified error bound as follows:

$$\alpha_i = \frac{2\pi\epsilon_{\max}}{E \max_{\hat{\omega}'}(I_i(\hat{\omega}'))}. \tag{6}$$

If the exposure setting is adjusted in post-processing (i.e., auto-exposure), E is unavailable before shading. For such auto-exposure, the E of the previous frame can be used.

For VPLs, the radiant intensity is inversely proportional to the number of lights, N. Therefore, larger α_i is used for larger N, and thus a smaller light range can be produced. Hence, using this error-bound–based α_i, the number of VPLs to be evaluated per shading point is sublinear with respect to the total number of VPLs.

One limitation of this error-bound calculation is that the directionality of radiant intensity, $I_i(\hat{\omega}')$, is ignored to be able to utilize existing light culling frameworks for point lights. Therefore, in this paper, we assumes low-frequency radiant intensity (e.g., VPLs on diffuse surfaces).

4. Stochastic Tiled Lighting for Real-time Indirect Illumination

4.1. Stochastic Tiled Lighting

Stochastic light culling is easily integrated into any real-time lighting framework that supports limited range lights. This section presents a sample implementation that is applicable to existing tiled lighting methods, such as tiled deferred rendering, Forward+, and clustered shading. The details of these techniques can be found in publicly available source codes [Persson and Olsson 2013; AMD 2014].

Algorithm 1 is a pseudocode of our stochastic tiled lighting. This algorithm consists of three steps: (1) light range computation, (2) light culling, and (3) shading. Our contributions are given in red. The light range computation (1) is a new step for tiled lighting. This step randomly determines the influence range for each light source on the GPU. Using these random ranges, light sources are culled by the light culling step (2). This culling step is exactly the same as existing tiled lighting techniques. In addition, the only modification of the shading step (3) is the fall-off function.

For stochastic light culling, the image quality for each frame is limited to the user-specified error bound. The image quality can be further improved by using temporal reprojection [Nehab et al. 2007] and a different random number sequence for every frame. However, in this section we demonstrate that our light culling method can produce visually acceptable results even without such temporal techniques.

Algorithm 1 Stochastic tiled lighting.

```
// 1. Light range computation
for all $i$ in light sources in parallel do
    Calculate $\alpha_i$ using Equation (6)
    Generate random number $\xi_i$
    Calculate $r_i$ using Equation (3)
end for
// 2. Light culling
for all $j$ in tiles in parallel do
    Store overlapping light sources into $j$th light list
end for
// 3. Shading
for all $\mathbf{x}$ in shading points in parallel do
    Get the light list of the shading point $\mathbf{x}$
    for all $i$ in the light list do
        Calculate the fall-off function using Equation (2)
        Evaluate the radiance of $i$th light
    end for
end for
```

4.2. Integration for VPL-based Indirect Illumination

We demonstrate VPL-based indirect illumination using BRSM and AISMs [Ritschel et al. 2011] in conjunction with our stochastic tiled lighting. VPLs are sampled by using BRSM, and then an AISM is generated for each VPL to represent indirect shadow. The original BRSM paper used interleaved sampling and geometry-aware filtering to reduce the shading cost. Using our stochastic tiled lighting, shading can be accelerated although it introduces some variance.

In Ritschel et al. [2008a]'s implementation, non-interleaved deferred shading of interleaved sample patterns [Segovia et al. 2006] is used. Their technique first deinterleaves pixels in a regular sampling pattern (e.g., 8×8 sampling pattern) into subregions of the screen. For each region, shading is performed using a different subset of VPLs. We propose to use our stochastic tiled lighting algorithm for each region. Furthermore, using interleaved sampling helps reducing the memory requirements of the lists of lights we store per tile, which is often allocated in local memory (e.g., local data share of AMD GPUs [AMD 2012]). Therefore, tens of thousands of light sources can be handled without spilling the light lists to global memory. For example, 65,536 light sources can be rendered by using 8×8 interleaved sampling and a list of 1024 light indices.

4.3. Point Cloud Culling for Imperfect Shadow Maps

By limiting the influence range of lights, ISM rendering can also be accelerated similar to the VPL rendering described above. Although it also requires an overhead with a linear cost similar to tiled lighting, it is negligibly small compared to naïve ISM rendering. ISM is done by approximating the scene geometry using a point cloud. These points are rendered using splatting instead of polygons. This splatting pass can be the main bottleneck for real-time or interactive applications. However, since there are many points invisible to a light source (Figure 3), they can be culled before rendering

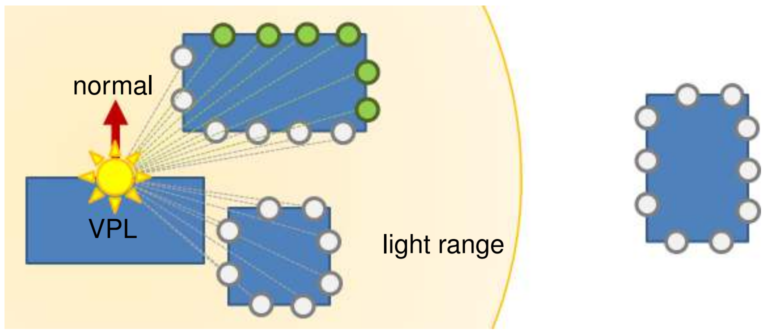


Figure 3. Visible (green) and invisible (gray) shadow caster points to a VPL (front-face culling). Invisible points can be culled for ISMs.

ISMs. For point splatting, front- or back-face culling can be used, similar to general shadow mapping. Therefore, approximately half of the points can be invisible to light sources. In addition, points under the surface of a VPL are also invisible. Tokuyoshi [2014] demonstrated a simple implementation of point cloud culling based on such visibility. Although this is efficient, the rendering time of ISMs is still linear. Our method, additionally, takes into account the distance between a point and VPL for the visibility, since the stochastic fall-off function has a finite influence-range. This improvement enables a sublinear cost for the splatting pass after the culling process, because points are more aggressively culled by increasing the number of VPLs, similar to light culling.

Implementation using DirectX® 11. Program 1 is the compute shader of point cloud culling. The system value *id* is the point index. This shader simply outputs only visible points using *AppendStructuredBuffer*. In order to splat these visible points onto shadow maps, triangle-based vertex shader splatting [Wu 2012] using the *DrawInstancedIndirect* API is used. Although this point cloud culling has a linear complexity, the computation time per point is small compared to drawing a splat. Thus, we can reduce the computational cost for a frame.

```
AppendStructuredBuffer< uint > pointIdBuffer : register( u0 );

[ numthreads( WORKGROUP_SIZE, 1, 1 ) ]
void CullIsmPointsCS( const uint id : SV_DispatchThreadID )
{
  const float3 pos = GetPointPosition( id );
  const float3 normal = GetPointNormal( id );
  const uint vplIndex = GetVplIndex( id );
  const float3 vplPos = GetVplPosition( vplIndex );
  const float3 vplNormal = GetVplNormal( vplIndex );
  const float vplSquaredRange = GetVplSquaredRange( vplIndex );
  const float3 relativePos = pos - vplPos;

  if( dot( relativePos, normal ) > 0.0 && // front-face culling
      dot( relativePos, vplNormal ) > 0.0 &&
      dot( relativePos, relativePos ) < vplSquaredRange ) {
    pointIdBuffer.Append( id );
  }
}
```

Program 1. Culling invisible points for ISMs.

4.4. Experimental Results

We show the rendering results of the BRSM-based diffuse indirect illumination using stochastic tiled lighting with $\epsilon_{\max} = 0.0005$ performed on an AMD Radeon R9 290X.

To evaluate the scalability, lighting without AISMs is first shown, and then the rendering cost of AISMs is discussed in the final paragraph. The frame buffer and RSM resolution are 1920×1152 and 256×256, respectively. 8×8 interleaved sampling and 17×17 geometry-aware filtering are used to reduce the shading cost. The tile size is 16×16 pixels for tiled lighting. In this experiment, the frame buffer resolution is the multiple of the product of the interleaved sampling size and tile size (i.e., 128×128), for simplicity. The random number ξ_i is generated by using the tiny encryption algorithm [Zafar et al. 2010]. A deferred-rendering–based implementation [Andersson 2011] is used for the tiled light culling and shading steps in Algorithm 1, since BRSM and geometry-aware filtering use a G-buffer. In addition, 2.5D culling is employed for the tiled light culling step to simply improve the performance.

Quality. Figure 4 shows the quality comparison using the root mean squared error (RMSE) metric for a dynamic scene. The variance caused by interleaved sampling is visible as high-frequency noise (left column). In addition, the clamping-based approach, Figure 4(a), produces noticeable image darkening (i.e., bias). On the other hand, our stochastic tiled lighting (b) avoids such bias by introducing small variance. Although this variance is increased by interleaved sampling, it is reduced by geometry-aware filtering for final images (right column). Hence, our approach, Figure 4(b), significantly reduces the RMSE for almost the same computation time as (a). Even if a clamping error is quite small for each light source, it is not negligible for many lights. This is because such darkening biases are accumulated unlike variance. Our unbiased light culling avoids this accumulated error.

Figure 5 shows the comparison using different numbers of VPLs. For this scene, since thousands of VPLs are insufficient to represent indirect illumination, undesirable flickering is produced (please refer to the supplemental video). To reduce this flickering, a larger number of VPLs must be used. However, when tiled lighting with clamping light ranges is used for acceleration, an error, where the image becomes darker, increases in effect while increasing the number of VPLs. On the other hand, by using our stochastic tiled lighting, flickering is efficiently reduced without the darkening bias.

Performance scalability. Figure 6 shows the performance comparison between clamping-based tiled lighting, (a), and our stochastic tiled lighting, (b), using the same error bound. Although both (a) and (b) use the same culling process, which has a linear cost, our method, (b), has a significantly smaller computation time. This is because the number of lights to be evaluated after the culling process is sublinear for our method unlike the clamping-based approach, and this shading process is the main bottleneck. Our stochastic light ranges can be decreased using the user-specified error bound as described in Section 3.2. The computation times of other passes are shown in Table 6. For this scene, 65,536 VPLs can be handled in about 5 ms with a slight error.

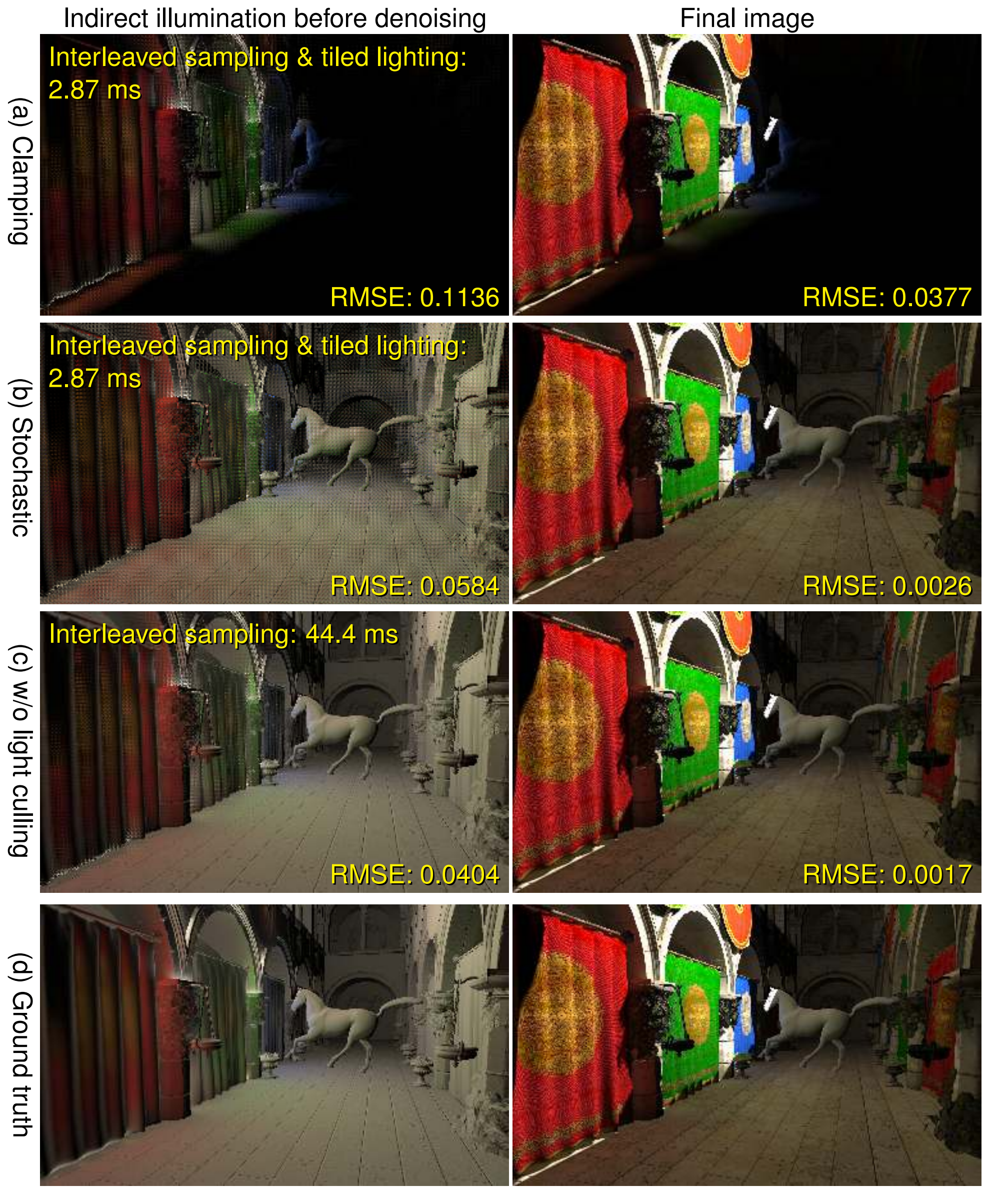

Indirect illumination before denoising
Final image
(a) Clamping
Interleaved sampling & tiled lighting: 2.87 ms
RMSE: 0.1136
RMSE: 0.0377
(b) Stochastic
Interleaved sampling & tiled lighting: 2.87 ms
RMSE: 0.0584
RMSE: 0.0026
(c) w/o light culling
Interleaved sampling: 44.4 ms
RMSE: 0.0404
RMSE: 0.0017
(d) Ground truth

Figure 4. Quality comparison of tiled lighting with clamping light ranges (a) and our stochastic tiled lighting (b) for a dynamic scene (264K triangles, 65,536 VPLs without AISMs, total rendering time: 7.0 ms). The image (c) is rendered without light culling. Left: indirect illumination computed using interleaved sampling. Right: final rendering results with denoising, texturing, and adding direct illumination. The clamping ranges are determined with reference to the same computation time as stochastic tiled lighting.

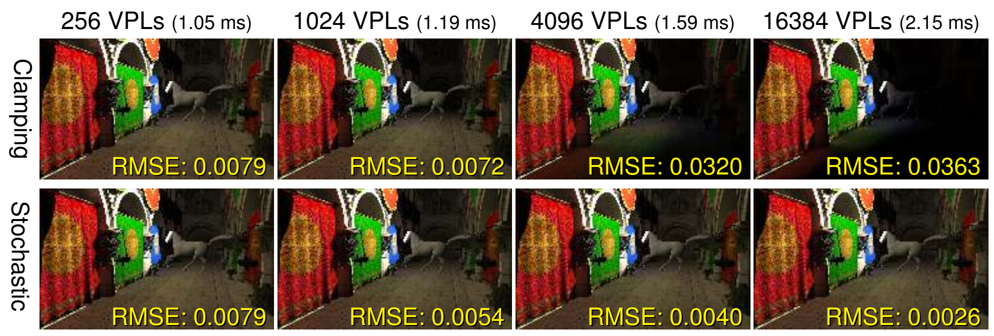


Figure 5. Same frame as Figure 4 using different numbers of VPLs. Each computation time is the combination of interleaved sampling and tiled lighting. The clamping ranges are determined with reference to the same computation time as stochastic tiled lighting. For a smaller number of VPLs, errors are visible as undesirable flickering (please refer to the supplemental video). This flickering can be reduced by increasing VPLs, but the clamping-based approach induces darkening to alleviate the computational burden. In contrast, our stochastic approach does not produce such darkening.

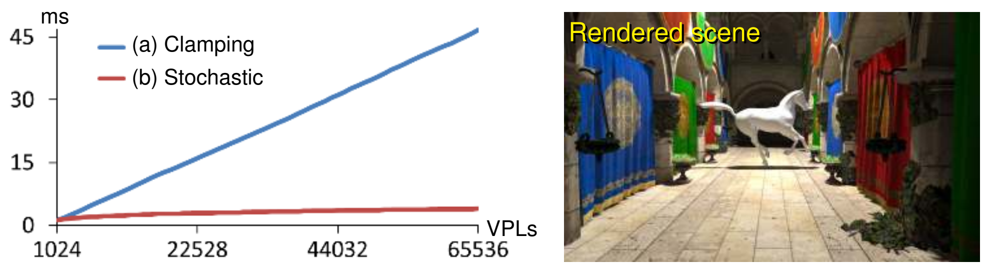
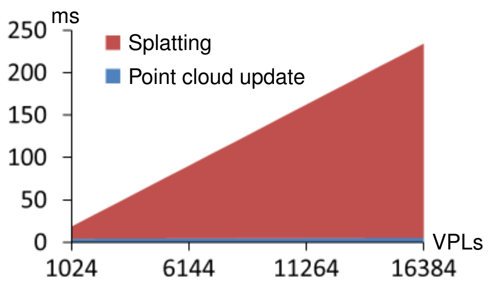


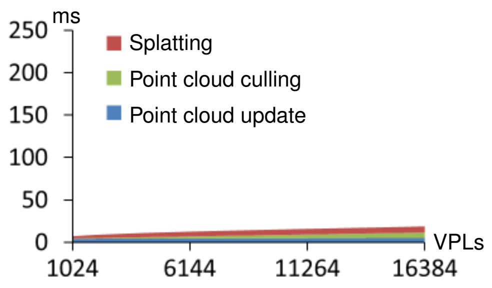
Figure 6. Plots of the computation time of the combination of interleaved sampling and tiled lighting. The blue line (a) is the tiled lighting with clamping light ranges, and the red line (b) is our stochastic tiled lighting. The experimental scene (right image) is a different frame of the scene of Figure 4. The clamping ranges are determined using the same error bound as ours (Appendix A). Our approach has a smaller computation time than the clamping-based approach by increasing the number of VPLs.

G-buffer	0.52
RSM	0.28
PDF of BRSM	2.63
VPL sampling & **light range computation**	**0.05**
Interleaved sampling & **tiled lighting**	**3.98**
Geometry-aware filtering	1.00

Table 1. Computation time for 65,536 VPLs of Figure 6 (ms).



(a) *w/o point cloud culling*



(b) *with point cloud culling*

Figure 7. AISM rendering time for the same scene as Figure 6.

AISM rendering. The AISM generation times with and without point cloud culling are plotted in Figure 7. For each AISM, 8192 points and 32×32 resolution are used. One-sixteenth of the point cloud is updated for each frame in a round-robin fashion. Point cloud culling produces a sublinear cost for the splatting pass, Figure 7(b), similar to light culling. Although this culling time is linear, it is negligibly small compared to the splatting pass without culling, Figure 7(a). Thus, the total AISM generation time is significantly reduced. Figure 8 shows the rendering result using AISMs. Using stochastic tiled lighting and point cloud culling for ISMs, AISMs are generated in about 22 ms for this scene. Thus, 16,384 VPLs with shadows can be rendered at real-time frame rates.

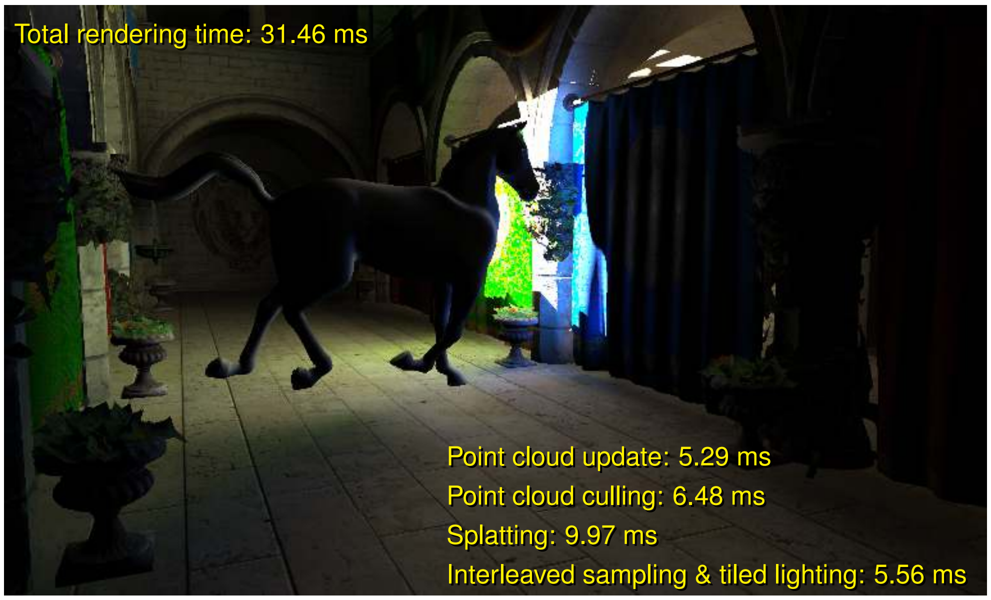

Total rendering time: 31.46 ms
Point cloud update: 5.29 ms
Point cloud culling: 6.48 ms
Splatting: 9.97 ms
Interleaved sampling & tiled lighting: 5.56 ms

Figure 8. Rendered image using AISMs with point cloud culling (16,384 VPLs).

5. Area Light Sampling for Progressive Path Tracing

Rendering a scene with many lights is also challenging for path tracing. In this section, we first describe additional techniques which are necessary to apply stochastic light culling to path tracing and then present optimization techniques for a GPU implementation.

5.1. Application to Area Lights

While real-time applications mostly use infinitesimal lights, such as point lights and spot lights, area lights are used more often than infinitesimal lights in a global illumination renderer. The proposed stochastic light culling is applicable to an area light for which the influence range r_i is defined for each surface location, and the bounding volume of it is approximated using the average emissive radiance of the area light. Although it computes a loose bound of a light, this simplification reduces the cost of the bound computation. Equation (6) to compute parameter α_i is modified for an area light as follows:

$$\alpha_i = \frac{2\pi\epsilon_{\max}}{E\bar{L}_i A_i}, \tag{7}$$

where $\bar{L}_i$ is the average emissive radiance, and A_i is the area of the light.

5.2. Culling via Bounding Sphere Tree

Since shading points of multi-bounce path tracing are distributed randomly, we cannot use view frustum-based clustering techniques unlike VPL-based global illumination. Therefore, we build a bounding sphere tree of lights that is used to find overlapping lights for a shading point. The influence range r_i is computed for all the lights at once before light culling and sampling are executed. Then, for each shading point generated by path tracing, overlapping lights are found by traversing the tree. Unlike lightcuts, multiple importance sampling (MIS) [Veach and Guibas 1995] is suitable for our method as we know the probability function $p_i(l)$ (Equation (1)).

Although this tree-based light culling works with any path tracing implementation, it is more suited for progressive path tracing than non-progressive path tracing. This is because an aggressive culling can be used for progressive path tracing, which allows greater error for each rendering pass, while we need to use a larger light range to render an image with small error in non-progressive path tracing. In this section, we assume area light sources are static, but their light ranges are determined using different random numbers for every frame. In this case, the tree is built during a preprocessing step, and then only the bounding volume of each node is updated in each frame. Thus, the overhead of using a bounding sphere tree for culling in a progressive path tracing is small.

Algorithm 2 Light sampling using stochastic light culling for a GPU path tracer.

```
// Light range computation
for all $i$ in light sources in parallel do
    Generate random number $\xi_i$ and calculate $r_i$ using Equation (3)
end for
Update the bounding volume of each node of the tree
// Light culling and sampling
for all $\mathbf{x}$ in shading points in parallel do
    lightList[] // List of overlapping light indices
    cdf[] // CDF of PDF $q(i)$ given by Equation (8)
    // Light culling via bounding sphere tree
    $m \leftarrow 0$
    node $\leftarrow$ root node of bounding sphere tree
    while node is valid do
        if node overlaps $\mathbf{x}$ and node is a leaf then
            lightList[$m$++] $\leftarrow$ light index
        end if
        node $\leftarrow$ Traverse to the next node
    end while
    // CDF building
    for all $j$ in lightList do
        if $j == 0$ then
            cdf[$j$] $\leftarrow$ importance $g_j(\mathbf{x})$
        else
            cdf[$j$] $\leftarrow$ cdf[$j - 1$] + importance $g_j(\mathbf{x})$
        end if
    end for
    Normalize cdf
    // Light sampling
    $i \leftarrow$ Sample a light source using binary search for cdf
    // Light vertex sampling
    $\mathbf{x}_i \leftarrow$ Sample a vertex on the light $i$
    if $\|\mathbf{x}_i - \mathbf{x}\| < r_i$ then
        Compute the sample density using Equation (10)
        Evaluate the sample radiance
    end if
end for
```

5.3. GPU Implementation

The proposed method is easily implemented in a CPU path tracer. However, a straightforward implementation of it on the GPU, executing a shading and visibility test when a leaf node is found in the tree traversal, leads to an inefficient usage of the GPU mainly because of branch divergence. The divergence gets worse as the computational cost of the visibility test and shading increases. In this section, we present optimization techniques to implement stochastic light culling efficiently on the GPU. A pseudocode using these optimization techniques is shown in Algorithm 2.

5.3.1. Decoupling the Tree Traversal and Shading

To reduce the branch divergence on the GPU, our approach decouples the tree traversal and shading. This decoupling is effective especially for ray-tracing–based algorithms compared to shadow-map–based approximation approaches [Ritschel et al. 2008b] because of the higher computational cost of the visibility test. This decoupling is done by using a light list similar to existing real-time light culling techniques. Overlapping lights are first added into a light list in the tree traversal, and then these lights are evaluated after the tree traversal. Unlike tiled lighting, which allocates a light list per tile (i.e., cluster of shading points), a light list is necessary for each shading point in our approach. Therefore, this decoupling consumes more memory, but it reduces the branch divergence.

To overcome this memory issue, we use reservoir sampling [Vitter 1985] to select overlapping lights to be stored. The method makes it possible to select k overlapping lights with a storage for k lights without enumerating all the overlapping lights. To apply reservoir sampling to Algorithm 2, the red line needs to be changed to Algorithm 3. When there are m overlapping lights for a shading point, where m is greater than k, reservoir sampling chooses k lights at a probability of k/m. Therefore, the use of this method increases the variance in this case.

Algorithm 3 Reservoir sampling.

```
if $m < k$ then
    lightList[$m$++] $\leftarrow$ light index
else
    nRandomIndex $\leftarrow$ random() $\times$ (++$m$)
    if nRandomIndex $< k$ then
        lightList[nRandomIndex] $\leftarrow$ light index
    end if
end if
```

5.3.2. Restricting the Number of Shadow Rays

The technique described above restricts the number of overlapping lights per shading point to k at most. If we compute direct illumination for all the lights found, the number of shadow rays that need to be cast varies from 0 to k. It is preferable for the GPU to process uniform computation; thus we restrict the number of shadow rays to one at most by applying resampled importance sampling [Talbot et al. 2005]. This method is general, so it is applicable to any path tracing implementation even without stochastic light culling.

In the tree traversal, we collect the light list $\mathbb{S}$ which consists of k sampled lights from all the overlapping lights for a shading point. We define the probability density function (PDF) for resampling a light from the light list $\mathbb{S}$ as

$$q(i) = \frac{g_i(\mathbf{x})}{\sum_{j \in \mathbb{S}} g_j(\mathbf{x})}, \tag{8}$$

where $g_i(\mathbf{x})$ is an importance for resampling light i. The product of $g_i(\mathbf{x})$ and the overlapping probability of the leaf is recommended to be approximately proportional to the illumination integral for the light. For simplicity, we approximate the importance as

$$g_i(\mathbf{x}) = f(\|\mathbf{x} - \mathbf{x}_{i,c}\|)\bar{L}_i A_i \max(-\hat{\omega}_{i,c} \cdot \hat{n}_i, 0) \max(\hat{\omega}_{i,0} \cdot \hat{n}, \hat{\omega}_{i,1} \cdot \hat{n}, \hat{\omega}_{i,2} \cdot \hat{n}, 0), \tag{9}$$

where $\hat{\omega}_{i,0}$, $\hat{\omega}_{i,1}$, $\hat{\omega}_{i,2}$, and $\omega_{i,c}$ are directions from $\mathbf{x}$ to three vertices and the center $\mathbf{x}_{i,c}$ of triangle light i, and $\hat{n}_i$ is the geometric normal of light i. Although this importance ignores BRDFs, the variance caused by high-frequency BRDFs can be reduced by MIS for path tracing. To sample according to this PDF, we build the cumulative distribution function (CDF) for the light list, and then a light is sampled using a binary search for the CDF. Finally, a vertex on the sampled light source is sampled, and then the radiance is evaluated using the following sample density:

$$p_i(\|\mathbf{x}_i - \mathbf{x}\|) \min\left(\frac{k}{m}, 1\right) q(i) s_i(\mathbf{x}_i), \tag{10}$$

where $\mathbf{x}_i$ is the sampled vertex, and $s_i(\mathbf{x}_i)$ is the vertex sampling PDF on the light i (e.g., $s_i(\mathbf{x}_i) = \frac{1}{A_i}$ for area-based sampling).

Although importance, $g_i(\mathbf{x})$, could be used for the probability of reservoir sampling in the tree traversal, we compute this importance after selecting k lights to minimize the divergence of code paths and bound the computational cost of the importance.

5.4. Experimental Results

First, we show the effectiveness of using stochastic light culling for area lights by comparing rendered images using a CPU progressive path tracer. Note that none of

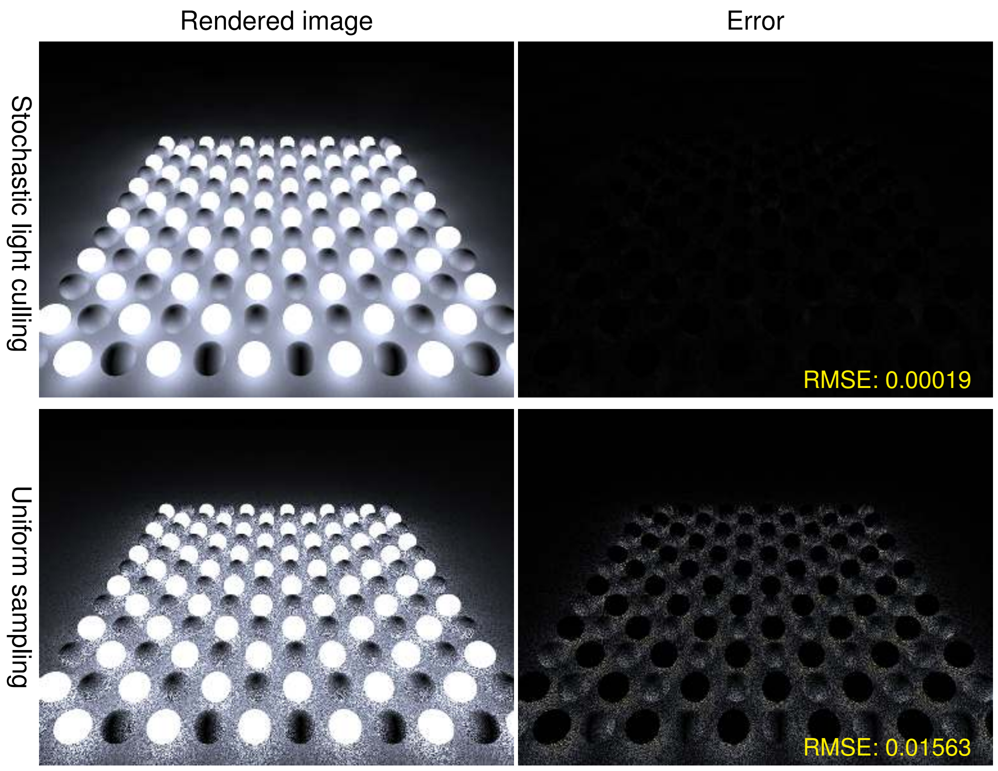


Figure 9. Equal time comparison of stochastic light culling and uniform sampling in a CPU path tracer. (54,720 triangle-lights scene, 640×480 resolution, 30 s).

the techniques presented in Section 5.3 are used in this test. We used $\epsilon_{\max} = 0.125$ to calculate α_i for each sample frame for all the experiments shown in this section. Images in the left column of Figure 9 are each rendered in 30 seconds on dual Intel® Xeon® E5-2670 v3 CPUs. There are 54,720 triangle lights in the scene. We can see that the result rendered with stochastic light culling has far less noise and significant reduction in RSME compared to the result rendered using uniform sampling.

Next, a GPU-based path tracer with stochastic light culling (Algorithm 2) is developed using OpenCL™. All the following tests are performed by rendering images at 1280×720 screen resolution on an AMD Radeon R9 290X GPU.

We rendered four scenes in which light sources are placed in a box with two light bounces using light culling with clamping, stochastic light culling, and the brute-force method to build the light sampling CDF (Figure 10). We used $k = 32$ for the tests. Resampled importance sampling is used for all the images to restrict the number of shadow rays. The light sampling algorithm using light culling with clamping is almost the same as Algorithm 2, but it uses a constant range for r_i, and $p_i(\|\mathbf{x}_i - \mathbf{x}\|)$ is removed from Equation (10). The brute-force method does not cull any lights, thus,

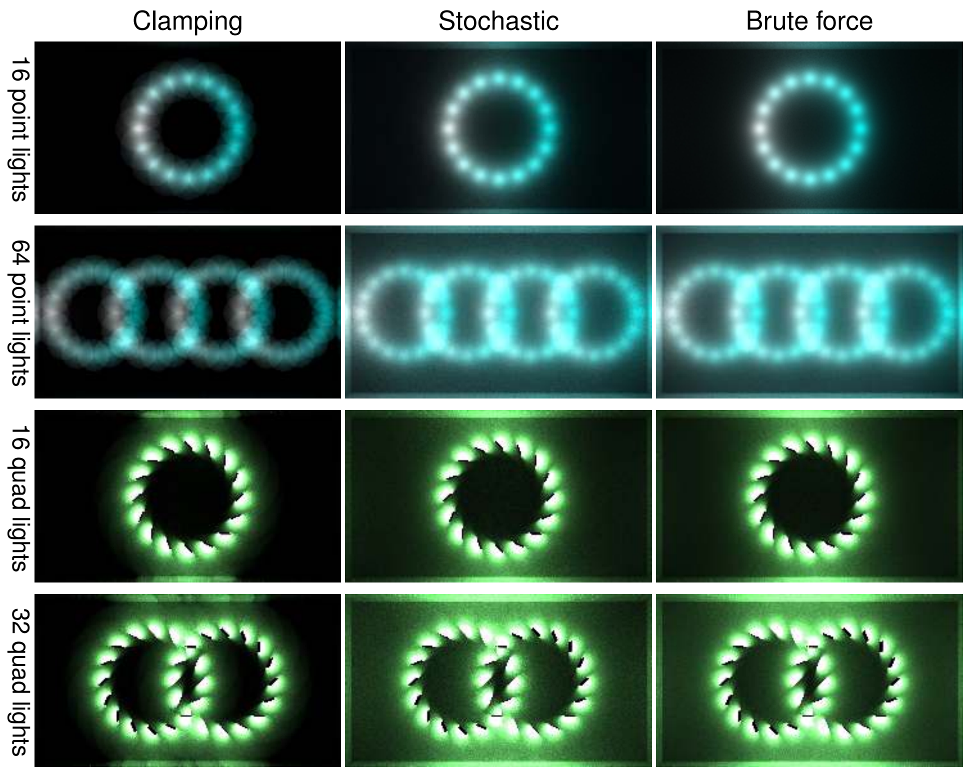
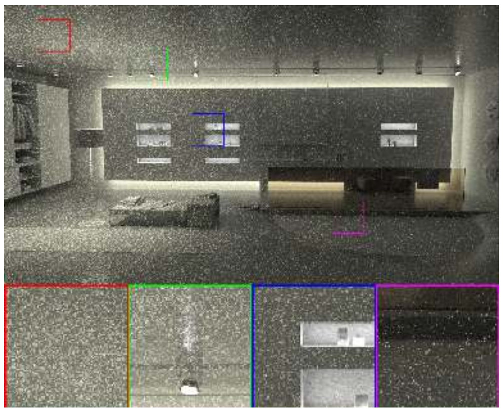
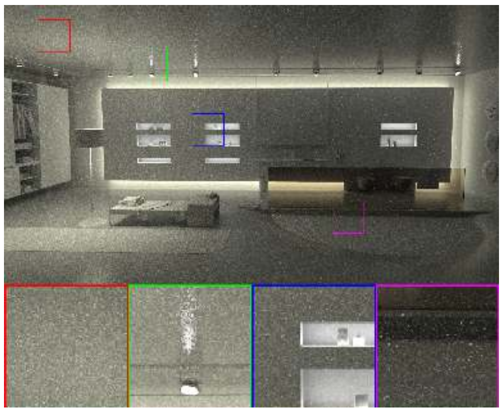


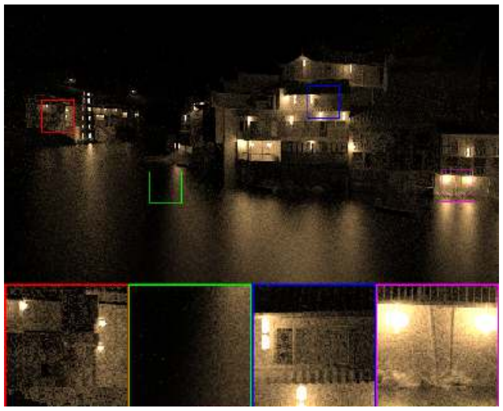
Figure 10. Comparison of rendered images using light culling with clamping, stochastic light culling, and the brute-force method to build the light sampling CDF (100 samples/pixel).

it computes the importance (Equation (9)) at each shading point for all lights in the scene before sampling a light. The rendered image using light culling with clamping is dark as it cannot capture the light bounce effect. On the other hand, stochastic light culling can reproduce the light bounce effect as seen in the reference images rendered using the brute-force method.

More complex scenes are rendered using stochastic light culling with reservoir sampling (Figures 11(b) and (d)). There are 5,186 and 59,270 triangle lights in scene A, respectively, scene B. Although stochastic light culling reduces the number of lights to be processed at each shading point, we need to use reservoir sampling for such scenes with thousands of light sources to keep the memory overhead of storing light lists at a practical level. To show the advantage of our method, we compared these to images rendered using the uniform light sampling shown in Figures 11(a) and (c). The reason we could not use the brute-force method to build the CDF described above is because it is computationally too expensive to calculate importance from all lights and requires too much memory to store the CDF for all lights per shading point. Note that we need to allocate storage for the worst case per shading point.


(a) *Scene A. Uniform sampling. RMSE: 0.0749.*


(b) *Scene A. Ours. RMSE: 0.0464.*


(c) *Scene B. Uniform sampling. RMSE: 0.0355.*

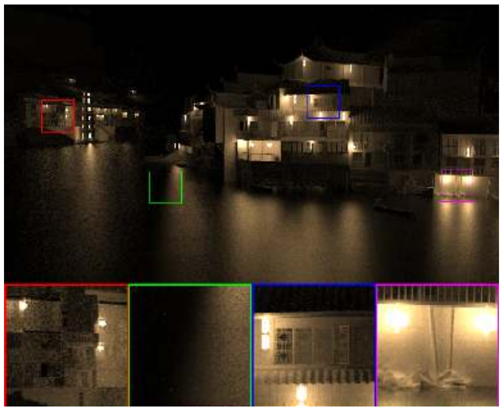

(d) *Scene B. Ours. RMSE: 0.0203.*

Figure 11. Equal time comparison of the proposed light sampling method and uniform light sampling (2 min). (a) and (c) are images rendered using the uniform light sampling. (b) and (d) are images rendered using stochastic light culling and reservoir sampling. There are 1.3M triangles and 5,186 triangle lights in Scene A, and 1.9M triangles and 59,270 triangle lights in Scene B. The reference images are rendered using uniform light sampling in 120 min.

These images are rendered in two minutes on the GPU. We used $k = 8$ for a light list allocated in local data share for each work item processing a shading point. We allocated the CDF in global memory which is 8 floating point values per shading point. Therefore, the global memory overhead to store all the CDFs is 28MB to process 1280×720 shading points concurrently.

6. Discussion

6.1. Limitations

Banding artifacts. Our method exchanges high-frequency noise of Russian roulette for banding artifacts. Compared to high-frequency noise, reduction of banding artifacts is more difficult for geometry-aware filters. However, this limitation is not a big problem for area lights and many point lights with small $\epsilon_{\max}$.

Directional importance. In this paper, high-frequency radiant intensity and BRDFs are not taken into account for the light range computation. For path tracing, BRDF-dependent variance is reduced by MIS. However, we do not address the inefficiency caused by high-frequency radiant intensity (e.g., VPLs on specular surfaces that represent caustics). This problem could possibly be solved by using a non-spherical bounding volume for a light range representation. We would like to investigate efficient bounding volumes for light culling in the future. Culling using tighter bounding volumes cannot only be effective for VPLs, but also other virtual light representations for glossy indirect illumination (e.g., virtual spherical Gaussian lights [Tokuyoshi 2015]).

6.2. Future Work

Imperfect shadow maps. Although the method in this paper accelerated ISMs using point cloud culling to achieve real-time frame rates, it is still the main bottleneck in the VPL-based global illumination algorithm [Ritschel et al. 2011]. For more time-sensitive applications, we have to further improve the performance of ISMs.

Bidirectional reflective shadow mapping. To efficiently sample VPLs from RSMs, view-dependent importance on RSMs is estimated by BRSM, which computes indirect illumination exchanging RSMs and a G-buffer. This importance estimation can also be accelerated by using our stochastic tiled lighting.

Clustering of shading points. Current area-light sampling for path tracing, unlike the tiled lighting algorithm, does not cluster shading points into groups. For path tracing, efficient clustering is an interesting problem, since various optimization techniques can be used once shading points are clustered. For example, overlapping lights can be found using clusters with smaller computational burden than using each shading point. Clustering also makes it easy to combine the proposed method with a method like importance caching [Georgiev et al. 2012] to accelerate importance computation.

Bidirectional path tracing. Veach [1998] used Russian roulette to sample light vertices for bidirectional path tracing. Stochastic light culling can be used for this light vertex sampling. However, it might be better to take BRDFs into consideration to make sampling effective. We would like to investigate its effectiveness in the future.

7. Conclusions

This paper proposed an unbiased light culling method for physically-based lighting. Unlike existing clamping-based light culling, our method has a sublinear cost for shading after the culling process when using a user-specified error bound. Integrating our method into a tiled lighting framework, 65,536 VPLs without shadow maps can now be handled in about 5 ms with a smaller error than previous techniques. 16,384 AISMs can be generated in about 22 ms by using range-based point cloud culling. This paper also introduced a light culling algorithm using a bounding sphere tree for progressive path tracing on the GPU. Hence, light sources can be culled not only for real-time rendering, but also for multi-bounce global illumination using path tracing in an unbiased fashion. Using our method for a path tracer, a scene with over 50,000 area lights could be rendered efficiently. The limitation of our method is that culling using a spherical bounding volume can be inefficient for high-frequency radiant intensity. For future work, we would like to address this issue to achieve a robust algorithm for various lighting conditions.

A. Error Bound of Clamping Light Ranges

For a clamped light range, the bias of the fall-off function is given by

$$\beta_i(l) = \begin{cases} -f(l), & (l \geq r_i) \\ 0, & (\text{otherwise}) \end{cases}.$$

Thus, the imaged absolute error for a single light source is represented as follows:

$$\epsilon_i(\mathbf{x}, \hat{\omega}) = |\beta_i(\|\mathbf{x}_i - \mathbf{x}\|)|\, E I_i(-\hat{\omega}_i') V(\mathbf{x}, \mathbf{x}_i) \rho(\mathbf{x}, \hat{\omega}, \hat{\omega}_i') \max(\hat{\omega}_i' \cdot \hat{n}, 0). \tag{11}$$

Unlike variance, since $\beta(l)$ is always negative, the error is accumulated for several light sources. Therefore, the error bound of clamped ranges is given as

$$\epsilon_{\max} \geq \sum_i^N \epsilon_i(\mathbf{x}, \hat{\omega}).$$

Similar to Section 3.2, Lambertian surfaces and the maximum value for each term of Equation (11) are assumed to estimate the error bound. Using $\max_l(|\beta_i(l)|) = f(r_i)$, the estimated error bound is represented as

$$\epsilon_{\max} = \frac{E \sum_i^N f(r_i) \max_{\hat{\omega}'}(I_i(\hat{\omega}'))}{\pi}.$$

Conversely, when $\epsilon_{\max}$ is given by a user-specified parameter, the light range can be obtained as the following equation:

$$r_i = f^{-1}\left(\frac{\pi \epsilon_{\max}}{EN \max_{\hat{\omega}'}(I_i(\hat{\omega}'))}\right) = \sqrt{\frac{EN \max_{\hat{\omega}'}(I_i(\hat{\omega}'))}{\pi \epsilon_{\max}}}.$$

For VPLs, since the radiant intensity is inversely proportional to N, the range r_i can be constant regardless of the number of VPLs. Hence, when using a user-specified error bound for light culling with clamping ranges, the number of light sources to be evaluated per shading point is linear with respect to the total number of lights.

References

AMD, 2012. AMD graphics cores next (GCN) architecture. URL: https://www.amd.com/Documents/GCN_Architecture_whitepaper.pdf. 42

AMD, 2014. Radeon SDK - enhance your 3D graphics development. URL: http://developer.amd.com/tools-and-sdks/graphics-development/amd-radeon-sdk/. 41

ANDERSSON, J. 2011. Directx 11 rendering in battlefield 3. In *GDC '11*. URL: http://www.dice.se/news/directx-11-rendering-battlefield-3/. 37, 44

ARVO, J., AND KIRK, D. 1990. Particle transport and image synthesis. *SIGGRAPH Comput. Graph. 24*, 4, 63–66. URL: http://doi.acm.org/10.1145/97880.97886. 36

BALESTRA, C., AND ENGSTAD, P.-K. 2008. The technology of uncharted: Drake's fortune. In *GDC '08*. URL: http://www.naughtydog.com/docs/Naughty-Dog-GDC08-UNCHARTED-Tech.pdf. 37

DACHSBACHER, C., AND STAMMINGER, M. 2005. Reflective shadow maps. In *Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games*, ACM, New York, I3D '05, 203–231. URL: http://doi.acm.org/10.1145/1053427.1053460. 37

DACHSBACHER, C., AND STAMMINGER, M. 2006. Splatting indirect illumination. In *Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games*, ACM, New York, I3D '06, 93–100. URL: http://doi.acm.org/10.1145/1111411.1111428. 36, 37

GEORGIEV, I., KŘIVÁNEK, J., POPOV, S., AND SLUSALLEK, P. 2012. Importance caching for complex illumination. *Comput. Graph. Forum 31*, 2pt3, 701–710. URL: http://dx.doi.org/10.1111/j.1467-8659.2012.03049.x. 55

HARADA, T., MCKEE, J., AND YANG, J. C. 2012. Forward+: Bringing deferred lighting to the next level. In *Eurographics '12 Short Papers*, The Eurographics Association, Aire-la-Ville, Switzerland, 5–8. URL: http://dx.doi.org/10.2312/conf/EG2012/short/005-008. 36, 38

HARADA, T., MCKEE, J., AND YANG, J. C. 2013. Forward+: A step toward film-style shading in real time. In *GPU Pro 4: Advanced Rendering Techniques*. A K Peters/CRC Press, Natick, MA, 115–134. 37

HARADA, T. 2012. A 2.5D culling for forward+. In *SIGGRAPH Asia 2012 Technical Briefs*, ACM, New York, NY, USA, SA '12, 18:1–18:4. 37

KAJIYA, J. T. 1986. The rendering equation. *SIGGRAPH Comput. Graph. 20*, 4, 143–150. URL: http://doi.acm.org/10.1145/15902. 36

KELLER, A. 1997. Instant radiosity. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, ACM Press/Addison-Wesley Publishing Co., New York, SIGGRAPH '97, 49–56. URL: `http://doi.acm.org/10.1145/258769`. 36, 37

NEHAB, D., SANDER, P. V., LAWRENCE, J., TATARCHUK, N., AND ISIDORO, J. R. 2007. Accelerating real-time shading with reverse reprojection caching. In *Proceedings of the 22nd ACM SIGGRAPH/EUROGRAPHICS Symposium on Graphics Hardware*, Eurographics Association, Aire-la-Ville, Switzerland, 25–35. URL: `http://dl.acm.org/citation.cfm?id=1280094.1280098`. 41

NICHOLS, G., AND WYMAN, C. 2010. Interactive indirect illumination using adaptive multiresolution splatting. *IEEE Trans. Vis. Comput. Graph. 16*, 5, 729–741. URL: `http://doi.ieeecomputersociety.org/10.1109/TVCG.2009.97`. 36, 37

OLSSON, O., AND ASSARSSON, U. 2011. Tiled shading. *J. Graph. GPU, and Game Tools*, 235–251. URL: `http://www.tandfonline.com/doi/10.1080/2151237X.2011.621761`. 36

OLSSON, O., BILLETER, M., AND ASSARSSON, U. 2012. Clustered deferred and forward shading. In *Proceedings of the Fourth ACM SIGGRAPH / Eurographics Conference on High-Performance Graphics*, Eurographics Association, Aire-la-Ville, Switzerland, EGGH-HPG'12, 87–96. URL: `http://doi.acm.org/10.1145/2383809`. 36, 37

OLSSON, O., SINTORN, E., KÄMPE, V., BILLETER, M., AND ASSARSSON, U. 2014. Efficient virtual shadow maps for many lights. In *Proceedings of the 18th Meeting of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, ACM, New York, I3D '14, 87–96. URL: `http://doi.acm.org/10.1145/2556701`. 38

PERSSON, E., AND OLSSON, O. 2013. Practical clustered deferred and forward shading. In *SIGGRAPH '13 Course: Advances in Real-Time Rendering in Games*, ACM, New York. URL: `http://www.cse.chalmers.se/~olaolss/main_frame.php?contents=publication&id=practical_clustered_2013`. 41

RITSCHEL, T., GROSCH, T., KIM, M. H., SEIDEL, H.-P., DACHSBACHER, C., AND KAUTZ, J. 2008. Imperfect shadow maps for efficient computation of indirect illumination. *ACM Trans. Graph. 27*, 5, 129:1–129:8. URL: `http://doi.acm.org/10.1145/1457515.1409082`. 37, 38, 42

RITSCHEL, T., GROSCH, T., KAUTZ, J., AND SEIDEL, H.-P. 2008. Interactive global illumination based on coherent surface shadow maps. In *Proceedings of Graphics Interface 2008*, Canadian Information Processing Society, Toronto, Ont., Canada, GI '08, 185–192. URL: `http://dl.acm.org/citation.cfm?id=1375714.1375747`. 50

RITSCHEL, T., EISEMANN, E., HA, I., KIM, J. D., AND SEIDEL, H.-P. 2011. Making imperfect shadow maps view-adaptive: High-quality global illumination in large dynamic scenes. *Comput. Graph. Forum 30*, 8, 2258–2269. URL: `http://onlinelibrary.wiley.com/doi/10.1111/j.1467-8659.2011.01998.x/abstract`. 37, 38, 42, 55

SEGOVIA, B., IEHL, J. C., MITANCHEY, R., AND PÉROCHE, B. 2006. Non-interleaved deferred shading of interleaved sample patterns. In *Proceedings of the 21st ACM SIGGRAPH/EUROGRAPHICS Symposium on Graphics Hardware*, ACM, New York, GH '06, 53–60. URL: http://doi.acm.org/10.1145/1283900.1283909. 37, 42

TALBOT, J. F., CLINE, D., AND EGBERT, P. K. 2005. Importance resampling for global illumination. In *Proceedings of the Sixteenth Eurographics Conference on Rendering Techniques*, Eurographics Association, Aire-la-Ville, Switzerland, EGSR '05, 139–146. URL: http://dx.doi.org/10.2312/EGWR/EGSR05/139-146. 51

TOKUYOSHI, Y. 2014. Point cloud culling for imperfect shadow maps. In *SIGGRAPH ASIA '14 Course: GPU Compute for Graphics*, ACM, New York, 8:138–8:140. URL: http://www.jp.square-enix.com/info/library/. 43

TOKUYOSHI, Y. 2015. Virtual spherical gaussian lights for real-time glossy indirect illumination. *Comput. Graph. Forum 34*, 7, 89–98. URL: http://www.jp.square-enix.com/info/library/. 55

VEACH, E., AND GUIBAS, L. J. 1995. Optimally combining sampling techniques for monte carlo rendering. In *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*, ACM, New York, SIGGRAPH '95, 419–428. URL: http://doi.acm.org/10.1145/218380.218498. 48

VEACH, E. 1998. *Robust Monte Carlo Methods for Light Transport Simulation*. PhD thesis, Stanford University. URL: https://graphics.stanford.edu/papers/veach_thesis/. 55

VITTER, J. S. 1985. Random sampling with a reservoir. *ACM Trans. Math. Softw. 11*, 1, 37–57. URL: http://doi.acm.org/10.1145/3165. 50

WALTER, B., FERNANDEZ, S., ARBREE, A., BALA, K., DONIKIAN, M., AND GREENBERG, D. P. 2005. Lightcuts: A scalable approach to illumination. *ACM Trans. Graph. 24*, 3, 1098–1107. URL: http://doi.acm.org/10.1145/1186822.1073318. 37

WU, O. 2012. Enhancing graphics in unreal engine 3 titles using new code submissions. In *GDC '12*. URL: http://amd-dev.wpengine.netdna-cdn.com/wordpress/media/2012/10/Enhancing%20Graphics%20in%20Unreal%20Engine%203%20Titles%20using%20New%20Code%20Submissions.ppsx. 43

ZAFAR, F., OLANO, M., AND CURTIS, A. 2010. GPU random numbers via the tiny encryption algorithm. In *Proceedings of the Conference on High Performance Graphics*, Eurographics Association, Aire-la-Ville, Switzerland, HPG '10, 133–141. URL: http://doi.acm.org/10.1145/1921500. 44

Index of Supplemental Materials

The videos for our stochastic tiled lighting can be found at
http://www.jcgt.org/published/0005/01/02/JCGT-SLC.mp4 and
http://www.jcgt.org/published/0005/01/02/JCGT-SLC2.mp4.

Author Contact Information

Yusuke Tokuyoshi
Square Enix Co., Ltd.
Shinjuku Eastside Square
6-27-30 Shinjuku, Shinjuku-ku
Tokyo 160-8430, Japan
tokuyosh@square-enix.com

Takahiro Harada
Advanced Micro Devices, Inc.
One AMD Place
P.O. Box 3453
Sunnyvale CA 94088-3453
Takahiro.Harada@amd.com

Y. Tokuyoshi and T. Harada, Stochastic Light Culling, *Journal of Computer Graphics Techniques (JCGT)*, vol. 5, no. 1, 35–60, 2016
http://jcgt.org/published/0005/01/02/

Received:	2015-03-23		
Recommended:	2015-10-04	Corresponding Editor:	Angelo Pesce
Published:	2016-03-28	Editor-in-Chief:	Marc Olano