

Generalized Light Portals

Shinji Ogaki
shinji.ogaki@gmail.com

HPB 2020

Hi, my name is Shinji Ogaki. I will be presenting my paper titled "generalized light portals".

Light Portals

Light Portal



Roll over image to view scene without light portal (skydome light only)

Skydome lights can use light portals to reduce noise for interior scenes, where the light comes in through relatively small openings. Instead of emitting light, they can be used to guide skydome light sampling. Light portals must be placed to cover all windows, doors, and other openings through which skydome light comes into the scene.

"Why do I use a portal light?"

Since lighting interiors with little or no direct lighting is difficult for a renderer to resolve, portal lights help direct the renderers attention to the exterior source of light. RenderMan can make excellent use of the PxrPortalLight as shown below on the classic Classroom scene. Both images are rendered with the same 32 samples per pixel. But you can plainly see how much more effective the result is with portal lights on the right than without on the left.



Light portals are useful for accelerating the convergence of Monte Carlo path tracing when rendering interiors. Industry standard renderers such as Arnold and RenderMan support light portals.

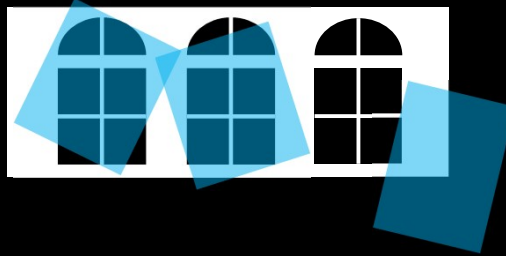
Related Work

- *Light Portals : Light Transport Variance Reduction*
by Phi Hung Le Nguyen. 2014.
- *Portal-Masked Environment Map Sampling*
by Benedikt Bitterli, Jan Novák, and Wojciech Jarosz. 2015.
- *Analytic Spherical Harmonic Coefficients for Polygonal Area Lights*
by Jingwen Wang and Ravi Ramamoorthi. 2018.

However, there are a limited number of literature related to light portals, and each one has its own limitations.

Limitations

- Shapes are normally limited to a flat rectangle
- Many-portals
- Placing a portal object around every single window is a tedious task



Normally shapes are limited to a planar rectangle and none of the previous work addresses many portals problem. Placing a portal object around every single window is a tedious task.

Objective

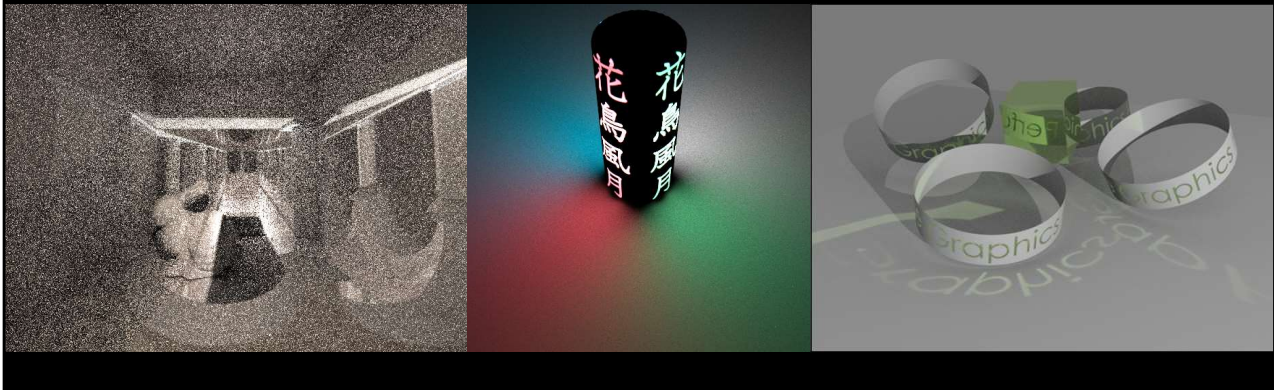
- Make portals user friendly



The goal of this paper is to make light portals user friendly by supporting arbitrary polygon meshes and arbitrary shapes created by 2d texture maps such as clip maps.

Examples

- Pinhole camera, Japanese lantern, etc.
- Caustics created by carved letters



Portals created by textures can be used for a variety of things. For example, we can support a pinhole camera in the path tracing framework. We can also compute illumination from a lantern efficiently. They enable guiding photons as well.

Main Ingredients

- *Fast Random Sampling of Triangular Meshes*
by Martin Sik and Jaroslav Krivanek. 2013.
- *Importance Sampling of Many Lights with Adaptive Tree Splitting*
by Alejandro Conty Estevez and Christopher Kulla. 2018.

The proposed technique is built on these 2 excellent papers.

Main Ingredients

- *Fast Random Sampling of Triangular Meshes*
by Martin Sik and Jaroslav Krivanek. 2013.
- *Importance Sampling of Many Lights with Adaptive Tree Splitting*
by Alejandro Conty Estevez and Christopher Kulla. 2018.
- Min max MIP maps
as tree data structures

Min max mipmaps are also used to accelerate some computations. If you already have these features in your renderer, the proposed technique can be implemented without a significant effort.

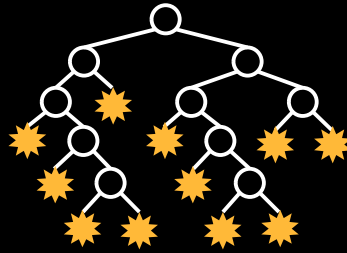
Our Approach

1. Extend *Adaptive Tree Splitting* to support textured mesh lights
2. Apply it to portals (e.g. trimmed / highly transparent regions)

This work mainly consists of two parts. In the first part, adaptive tree splitting is extended to support textured mesh lights. In the second part, it is applied to portal sampling.

Recap: Adaptive Tree Splitting

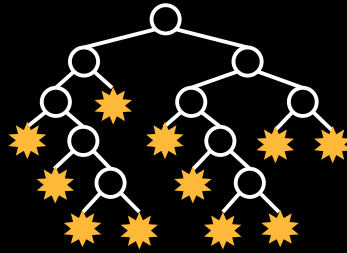
HPB
2020



Let's quickly recap adaptive tree splitting. The algorithm starts by building a bvh over lights. Lights that are spatially close and have similar orientations are clustered together using the surface area orientation heuristic.

To generate a sample, we traverse the BVH by stochastically selecting a node based on its illumination contribution, which is a product of the total energy of the node and so-called conservative geometric term.

Recap: Adaptive Tree Splitting



$$E \frac{\max\{0, \cos \theta'_i\}}{d^2} \times \begin{cases} \cos \theta' & \text{if } \theta' < \theta_e \\ 0 & \text{otherwise} \end{cases}$$

Let's quickly recap adaptive tree splitting. The algorithm starts by building a bvh over lights. Lights that are spatially close and have similar orientations are clustered together using the surface area orientation heuristic.

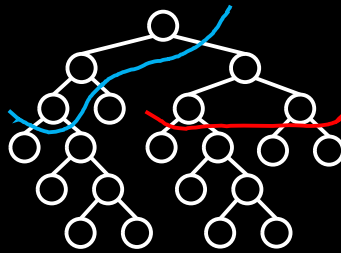
To generate a sample, we traverse the BVH by stochastically selecting a node based on its illumination contribution, which is a product of the total energy of the node and so-called conservative geometric term.

-
- A diagram of a binary tree with 16 leaf nodes, representing a hierarchical structure. The tree has a root node at the top, which branches into two child nodes. Each of these child nodes branches into two more child nodes, and so on, resulting in a total of 16 leaf nodes at the bottom level. The nodes are represented by small circles, and the edges are lines connecting them.

12

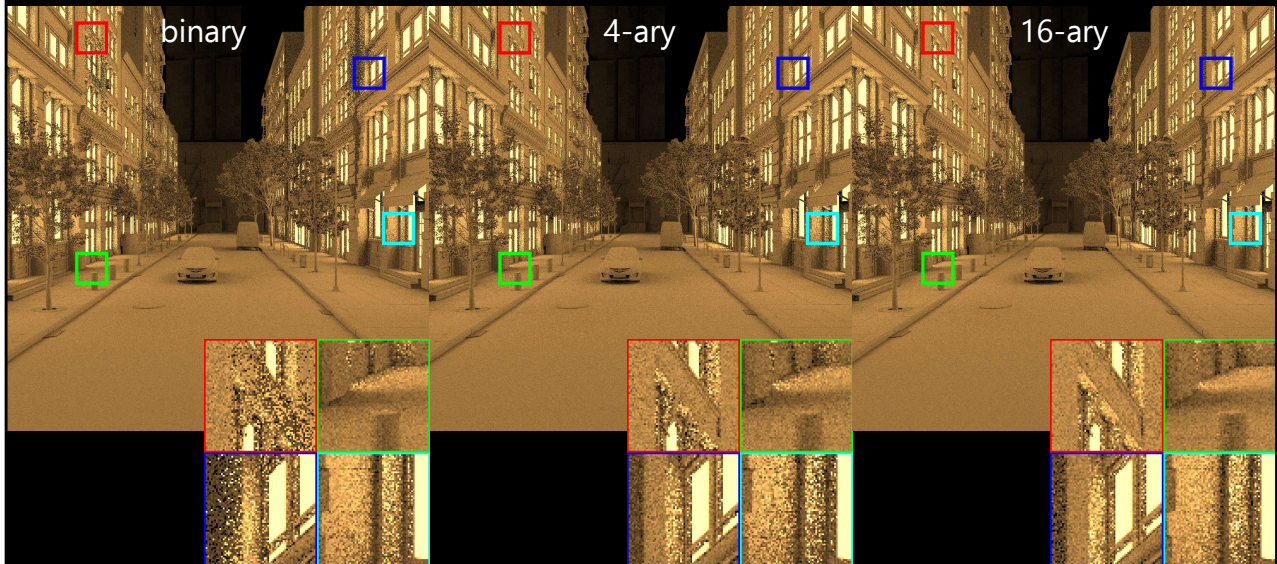
Wide Light BVH

- Collapsing a binary BVH is equivalent to creating multiple cuts
- Nodes that introduce a large amount of error will be removed



Collapsing a binary BVH into wide BVH is equivalent to creating multiple cuts in the original binary BVH. Nodes that introduce a large amount of error will be removed.

Wide Light BVH



As shown here, increasing the arity improves image quality.

1. Textured Mesh Lights

HPB 2020

So how can we support textured mesh lights?

Textured Mesh Light

- Naïve approach - *shade before hit* (like Manuka)
 - duplicates vertex data (position, uvs, etc.)
 - light BVH node is large

```
template<int32_t N> // arity
struct TLASNode    // node of TLAS
{
    int32_t indices [N]; // references to child nodes
    vec3<N> aabb_min;    // bounding box min values
    vec3<N> aabb_max;    // bounding box max values
    vec3<N> axes;        // cluster orientations
    float<N> cos_theta_o; // cos(theta_o), __m256, __m512
    float<N> sin_theta_o; // sin(theta_o)
    float<N> values;     // total cluster energies
};
```

A naïve approach would be to tessellate the input mesh and build a light BVH over the tessellated triangles.

This should work but memory consumption is of concern because vertex data will be duplicated and the node size of a light BVH isn't small.

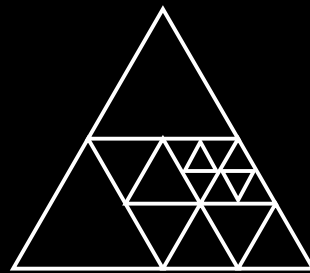
Two-Level Acceleration Structure

- **TLAS**

top level acceleration structure i.e. SAOH optimized BVH
is built over triangles

- **BLAS**

bottom level acceleration structure
is built in the barycentric coordinates
for each triangle



We can avoid this problem by using a two-level data structure. In this paper, the TLAS is built over triangles and BLAS is built in the barycentric coordinates for each triangle.

Why Barycentric Coordinates?

- BLAS node is compact

```
template<int32_t N> // arity
struct BLASNode    // node of BLAS
{
    uint32_t indices[N]; // references to child nodes
    float<N> values;     // average intensities of texels
};
```

- Node size can be further reduced by
 - quantizing
 - reducing the size of indices (when the number of nodes is small)

The node of the BLAS is compact. Of course, we can save more memory by reducing the size of indices, or by quantizing float values.

BLAS Traversal

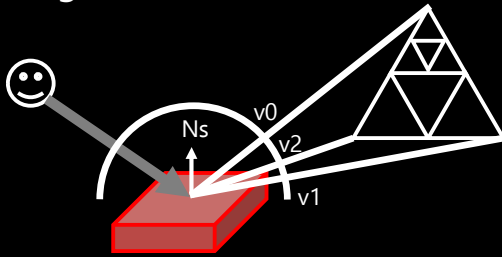
- Child node is probabilistically selected according to its illumination contribution (irradiance)

$$P_i = \frac{I_i}{\sum_{j=0}^{N-1} I_j}$$

When generating a sample, we probabilistically select a child node based on illumination contribution at every traversal step.

Illumination Contribution (BLAS)

- Irradiance from each node can be approximated with boundary integration



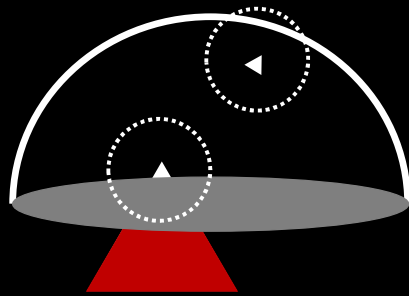
$$I = \frac{L}{2\pi} \sum_{i=0}^{n-1} \gamma_i \cos \delta_i$$

$$\gamma_i = \cos^{-1}(v_i \cdot v_j), \delta_i = N_s \cdot \frac{v_i \times v_j}{\|v_i \times v_j\|}$$

The irradiance from each node can be approximated with boundary integration. Here L is the average color of the texels covered by a node.

Illumination Contribution (BLAS)

- Exact evaluation is slow and suffers from numerical errors
- Tiny triangles are created not only by tessellation but also by clipping



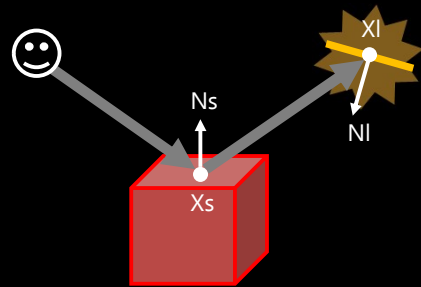
Unfortunately, this computation is slow because of Inverse trigonometric functions and suffers from numerical errors because tiny triangles are created not only by tessellation but also by clipping.

Illumination Contribution (BLAS)

- Point to point form factor?

$$I \simeq LA \frac{\cos \theta_s \cos \theta_l}{\pi \|x_s - x_l\|^2}$$

- Cosine terms can be negative



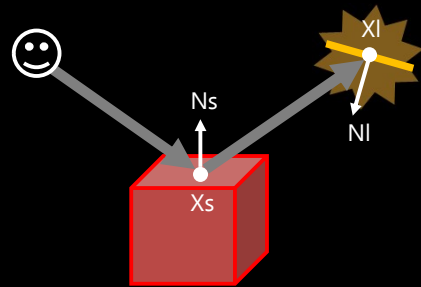
Instead, we can use a point to point form factor. This works but we have to deal with negative cosine terms.

Illumination Contribution (BLAS)

- Extremely conservative term

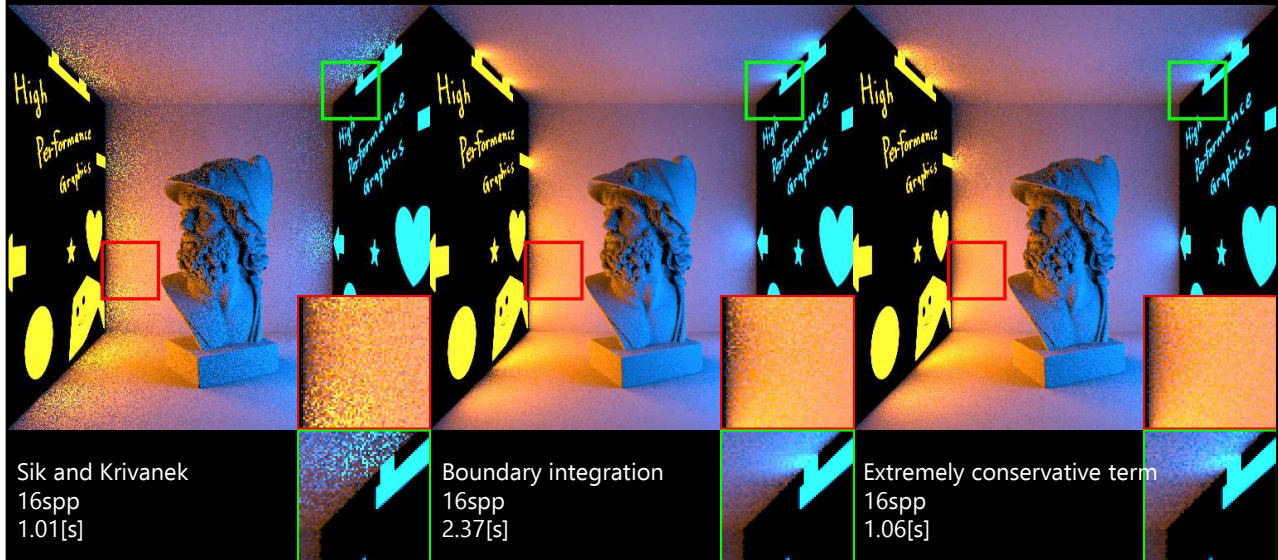
$$I \approx LA \frac{(1 + \cos \theta_s)(1 + \cos \theta_l)}{4\pi \|x_s - x_l\|^2}$$

- Always non negative
- Low evaluation cost



Negative values can be entirely avoided by using this extremely conservative term. At first, the use of this term seemed too bold but turned out to work quite well. This is because if an input mesh is moderately tessellated, the TLAS helps picking a right triangle.

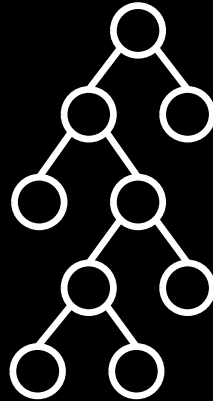
Illumination Contribution (BLAS)



Here is a comparison of 3 approaches. Each textured polygonal light is made of two triangles. The previous work by Sik produces noise because it doesn't take into account the geometric term. Boundary integration produces much less noise for most regions but suffers from fireflies. The extremely conservative term gives much better results compared to the previous work, and is more than two times faster than boundary integration.

PDF Evaluation

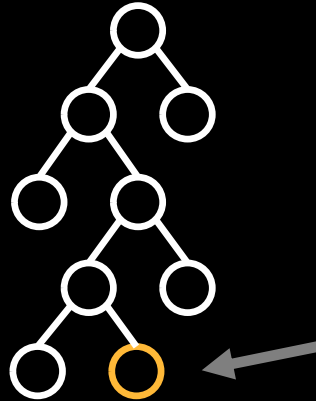
- Requires bit-trails or pointer to the parent node



To combine this technique with other sampling methods using multiple importance sampling, we need to be able to calculate a probability of a ray hitting a certain leaf.

PDF Evaluation

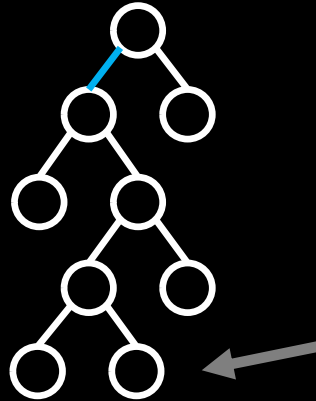
- Requires bit-trails or pointer to the parent node



Now let's assume that a ray hits this leaf node. To compute the probability, we deterministically traverse the tree from the root to the leaf. This is normally done using bit-trails attached to each leaf node. Alternatively, we can add a parent pointer to each node.

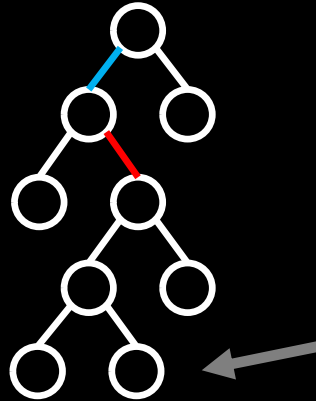
PDF Evaluation

- Requires bit-trails or pointer to the parent node



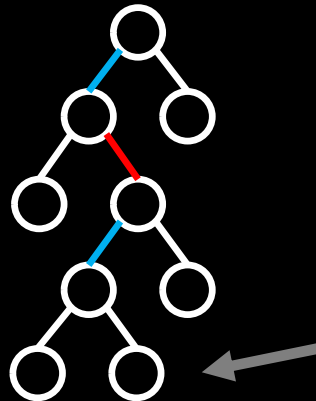
PDF Evaluation

- Requires bit-trails or pointer to the parent node



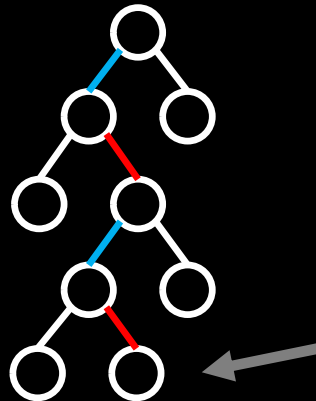
PDF Evaluation

- Requires bit-trails or pointer to the parent node



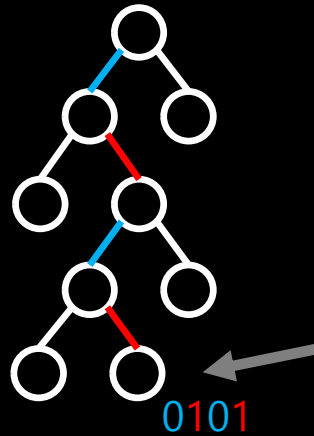
PDF Evaluation

- Requires bit-trails or pointer to the parent node



PDF Evaluation

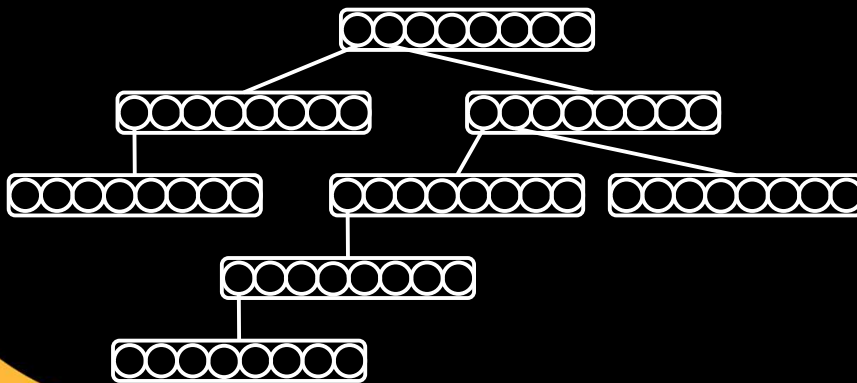
- Requires bit-trails or pointer to the parent node



The path from the root to the leaf can be expressed using these 4 bits.
This works perfectly for binary BVHs.

PDF Evaluation

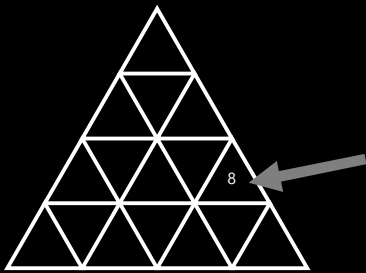
- Wide BVHs can be deep
- Each level requires A bits for 2^A -ary BVHs



However, this can be a problem for wide BVHs. A wide BVH can be deep when it is not well balanced, which is often the case. So we may need many bits to represent a path because, for example, if we use 16-ary BVHs, 4 bits are necessary for each traversal step.

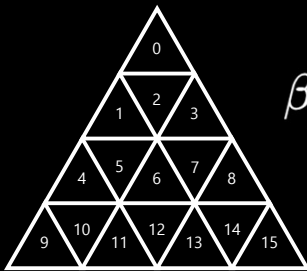
PDF Evaluation

- Neither bit-trails nor parent pointers are necessary for BLAS



PDF Evaluation

- Neither bit-trails nor parent pointers are necessary for BLAS
- Shading point's barycentric coordinates (b_0, b_1, b_2) suffice



$$\beta_j = \min \left\{ \sqrt{N} - 1, \left\lfloor (1 - b_j) \sqrt{N} \right\rfloor \right\}$$

$$i = \beta_0^2 + (\beta_1 - \beta_2) + \beta_0$$

A nice thing about building a tree in the barycentric coordinates is that neither bit-trails nor parent pointers are required to evaluate a probability. Using this indexing scheme.

2. Generalizing Light Portals

HPB
2020

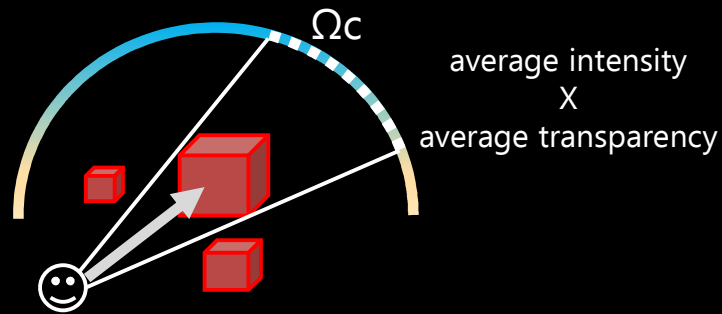
Now we are ready to apply the textured mesh light sampling to light portals.

Generalizing Light Portals

- Replace emission by ray guiding weight such as transparency
- Constant background – That's it!

For a constant environment map, we just need to replace emission values by ray guiding weights such as transparency.


Illumination Contribution (TLAS)



$$E \frac{\max\{0, \cos \theta'_i\}}{d^2} \times \begin{cases} \cos \theta' & \text{if } \theta' < \theta_e \\ 0 & \text{otherwise} \end{cases}$$

To handle general environment maps, we have to replace the E term in the conservative geometric term by the product of the average transparency of this cluster and the average intensity of pixels covered by the solid angle subtended by the cluster bounding box.

Illumination Contribution (BLAS)



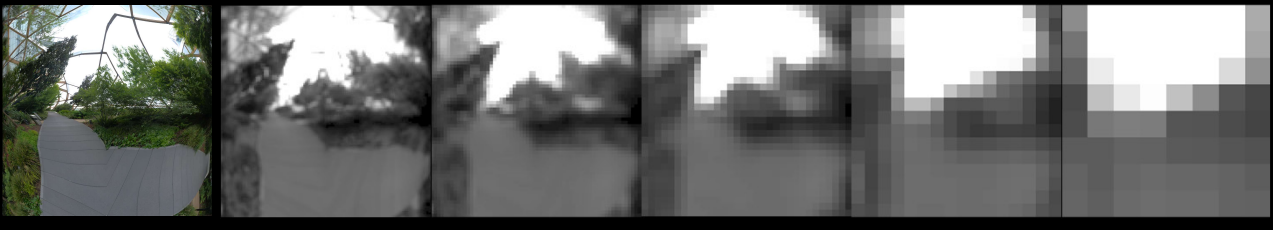
average intensity
×
average transparency

$$I \approx LA \frac{(1 + \cos \theta_s)(1 + \cos \theta_l)}{4\pi \|x_s - x_l\|^2}$$

Likewise, we have to replace the term L of the extremely conservative term.

MIP Maps for Environment Map

- Average intensity of the pixels covered by Ω_c is approximated by a MIP map
- Environment map is stored using Clarberg's equal-area parameterization
- 3x3 box filter is applied to each level image to avoid an abrupt transition between pixels



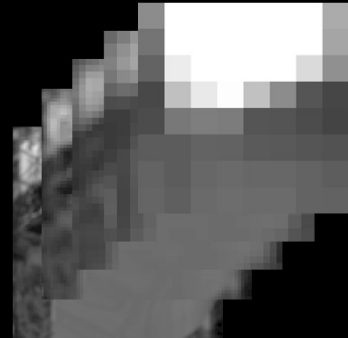
The average intensity of the pixels covered by Ω_c is approximated by a mipmap of the environment map that is stored using Clarberg's equal-area parameterization. A 3x3 box filter is applied to each level image to avoid an abrupt change between pixels.

MIP Map Level

- Solid angle covered by each pixel

$$\Omega = \min \left\{ 4\pi, \frac{(3 \times 3) \times 4\pi}{(2^{\text{level}})^2} \right\}$$

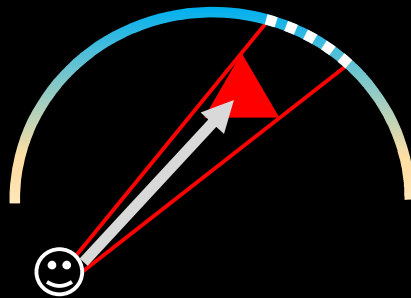
- MIP map level can be found by reverting this



Then the solid angle covered by each pixel is computed using this formula.
We can find the mipmap level by reverting this.

Traversal(Subdivision) Termination Criteria

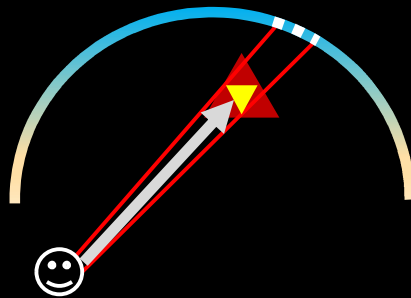
- Stopping traversal at a BLAS leaf may fail to capture bright pixels of the environment map



If we stop traversal when reaching a leaf node of the BLAS, we may fail to capture bright pixels of the environment map. Therefore, we have to keep subdividing a subtriangle until one or both of the following conditions are met.

Traversal(Subdivision) Termination Criteria

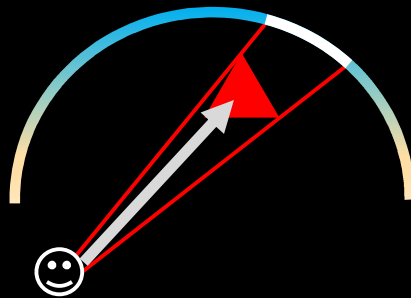
- 1) triangle size matches a single pixel of the environment map



the first condition: we can stop subdividing when a subtriangle covers a single pixel of the environment map.

Traversal(Subdivision) Termination Criteria

- 2) covered area is constant in intensity



The second condition: We can terminate subdivision if the covered area is constant. In the initialization process, min-max mipmaps are created as well as regular mipmaps. We can quickly check whether the area is constant by comparing a pixel's minimum and maximum values. If the values are equal, the covered region can be considered to be constant.

Results: Room (Equal Time Comparison)

- Star-shaped windows are used as portals



Lambertian + environment map sampling

Lambertian + environment map + portal sampling

Next, I'd like to show you some results. This room is illuminated by light passing through star-shaped windows. The left image is rendered using Lambertian and environment map sampling combined via multiple importance sampling. The right image is rendered with the three sampling techniques. As you can see, the proposed method helps reducing noise.

Results: Car Interior (Equal Time Comparison)

- Car windows are used as portals



When portals are large, the effectiveness of portal sampling is reduced. In this example, the thick glass windows are used as light portals. Using portal sampling still yields a better result for this scene because the roof of the car blocks the bright part of the sky.

Results: Lantern

- Diffuse transmittance material



A diffuse transmittance material can be assigned to a portal object. A point light source is placed in the lantern. You can see notable noise reduction.

Results: Simple Photon Guiding (Carved Letters)

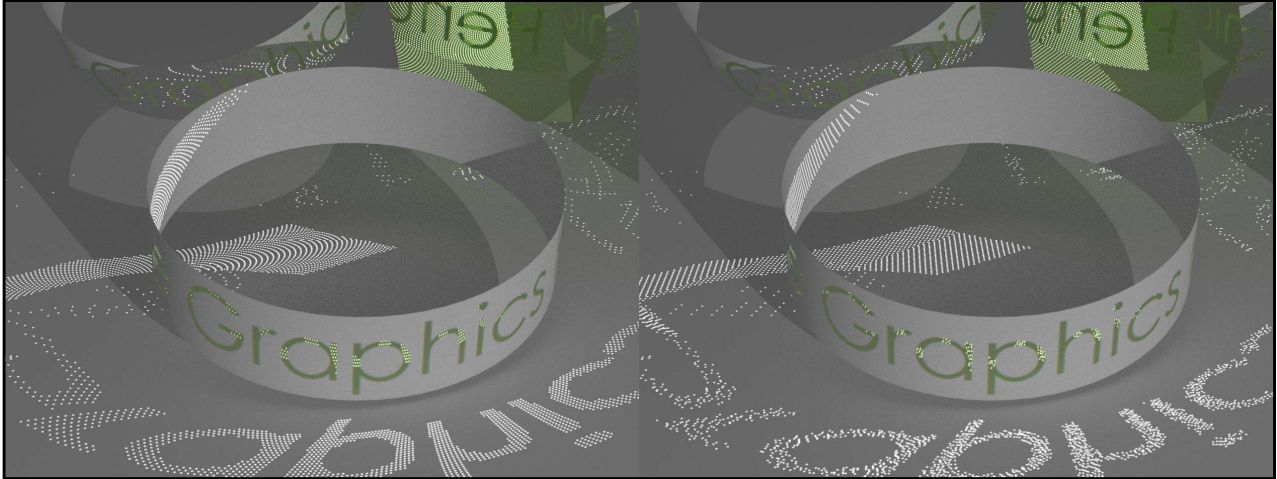
- Our sampling can be used to trace rays from light sources



Photon guiding is another interesting application. These images are rendered using 100k photons. With the proposed technique, photons are generated an order of magnitude faster.

Results: Simple Photon Guiding (Carved Letters)

- Proposed technique generates less stratified samples



Since we repeatedly scale random numbers during traversal, the samples generated with the proposed technique are less stratified.

Limitations / Future Work

- Performance improvement
 - Building CDF
- Better measure for nested light portals
- Product importance sampling (BSDF, Phase function, etc.)
- Tiling/procedural patterns
- Path guiding (the method could be useful in the learning phase)
- BVH optimization (ray specialized contraction by Gu et al. 2015)

The main bottleneck is the construction of a CDF during sampling, especially when the environment map is taken into account. In this paper, Clarberg's equal-area parameterization is used, but we could use similar ones that are cheaper to evaluate.

Our method works when the portals are nested. However, since each sample is drawn stochastically based on its solid angle, transparency, and the intensity of the environment map, its performance can degenerate. For instance, if a wall with a small portal is entirely covered by another wall that has a large portal, many samples would be generated on the large portal object, and be wasted.

Although our method can be easily combined with other sampling strategies via multiple importance sampling, it will be more efficient to take into account other factors such as BRDFs.

We did not consider tiling or procedural textures. Tiling would be relatively easy to support; however, it may increase memory consumption. Supporting patterns defined by implicit functions or resolution

Our method could be useful for accelerating learning by generating good initial paths.

There is also a variety of BVH optimization techniques for static and dynamic scenes, including rotation and restructuring. Applying them could improve SAOH-optimized BVHs. I'm particularly interested in the ray specialized contraction method by Gu et al.

Conclusion

- Simple sampling technique for arbitrary-shaped portals
 - based on the computation of illumination from textured mesh lights
- Easy to use
 - as simple as ticking a binary bit flag attached to a mesh or shader
 - setup can be automated if the names of shaders have a prefix such as 'window'

In summary, a simple sampling technique that works for arbitrary shaped portals has been proposed. This is based on the computation of illumination from textured mesh lights. Its use is as simple as ticking a binary flag attached to a mesh or shader. The setup can be even automated if the names of shaders have a prefix such as 'window'.

Thank you!

HPB 2020

Thank you for your attention.