Eurographics Symposium on Rendering 2010
Jason Lawrence and Marc Stamminger
(Guest Editors)

Volume 29 (2010), Number 4

Multi-Image Based Photon Tracing for Interactive Global Illumination of Dynamic Scenes

Chunhui Yao[1,2,3,4], Bin Wang[1,3,4], Bin Chan[5], Junhai Yong[1,3,4] and Jean-Claude Paul[6,1]

[1]School of Software, Tsinghua University, Beijing, China
[2]Department of Computer Science and Technology, Tsinghua University, Beijing, China
[3]Key Laboratory for Information System Security, Ministry of Education of China, Beijing, China
[4]Tsinghua National Laboratory for Information Science and Technology, Beijing, China
[5]The University of Hong Kong, Hongkong, China [6]INRIA, France

Abstract

Image space photon mapping has the advantage of simple implementation on GPU without pre-computation of complex acceleration structures. However, existing approaches use only a single image for tracing caustic photons, so they are limited to computing only a part of the global illumination effects for very simple scenes. In this paper we fully extend the image space approach by using multiple environment maps for photon mapping computation to achieve interactive global illumination of dynamic complex scenes. The two key problems due to the introduction of multiple images are 1) selecting the images to ensure adequate scene coverage; and 2) reliably computing ray-geometry intersections with multiple images. We present effective solutions to these problems and show that, with multiple environment maps, the image-space photon mapping approach can achieve interactive global illumination of dynamic complex scenes. The advantages of the method are demonstrated by comparison with other existing interactive global illumination methods.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture

1. Introduction

Global illumination (GI) is essential for synthesizing photo-realistic images. Currently, accurate and efficient computation of GI is still a challenging research topic. Photon mapping [Jen96] is one of the most important advances in tackling the problem in the past decade. It is an efficient method for high quality photo-realistic image synthesis, capable of producing many GI effects, such as color bleeding and caustics. Photon mapping is a consistent rendering technique in the sense that it converges to the correct physical result as the number of photons and bounces increases [Jen01].

The idea of photon mapping is to trace photons throughout the scene, and to cache the hit points in a data structure called the photon map. Radiance is then reconstructed in the rendering pass using photon density estimation. Existing methods achieve interactive rate by speeding up spatial traversal with an acceleration structure, such as KD-trees, either on CPU [WKB*02] [LC04] or on GPU [SZS*08] [WWZ*09]. This requirement on pre-computation of the acceleration structures restrict their use to static scenes or scenes with limited object movement. For truly dynamic scenes, methods for interactive building of the spatial structures have been proposed [HMS06] [SSK07] [ZHWG08]. While achieving impressive performance, these methods are relatively complicated to implement.

An alternative implementation of photon mapping is the image space approach, in which the scene geometry is represented by a depth image, and ray-geometry intersections required by photon tracing are computed directly using the depth information. This approach has two distinct advantages: it is relatively easy to implement because textures can naturally be processed in the GPU; and it is more efficient because it does not need to build any complex acceleration structure for a dynamic scene. However, the existing image space methods [SKALP05] [SKP07] [Wym08] use only one image and only caustic photon rays are traced. Therefore

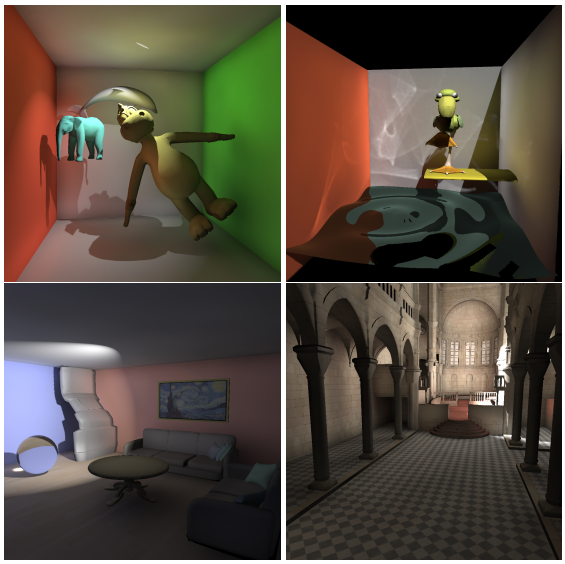


Figure 1: *Without pre-computation, our technique produces global illumination of dynamic scenes in real-time.*

they are limited in both scene complexity and the range of global illumination effects that can be simulated.

To achieve interactive global illumination of dynamic complex scenes, we propose to use multiple environment maps for photon tracing and photon splatting for radiance estimation. That is, photon tracing pass and rendering pass of the photon mapping algorithm are both approximated in image space. Our contributions are an effective heuristic method for selecting multiple environment maps to provide adequate coverage of scene geometry and a robust method for computing ray-geometry intersections with multiple environment maps. Our method is capable of computing all GI effects of photon mapping. Dynamic scenes and dynamic lightings are supported without preprocessing of scene geometry or light transfer. The method is easy to implement on a consumer GPU and achieves interactive frame rates for dynamic scenes with global illumination fully recomputed at every frame.

2. Related Work

Interactive Global Illumination Interactive GI is one of the most challenging problems in graphics. Keller [Kel97] introduced instant radiosity, which traces a small number of photons and treats the hit points as virtual point lights. To avoid expensive visibility testing, Dachsbacher et al. [DSDD07] proposed to produce GI with implicit visibility in radiosity. Ritschel et al. [RGK*08] introduced Imperfect Shadow Maps (ISM) for efficient visibility testing for instant radiosity. Reflective Shadow Maps (RSM) [DS05] computes single-bounce indirect illumination from an extended shadow map, but visibility for indirect lighting is ignored. Lehtinen et al. [LZT*08] proposed a meshless hierarchical representation for light transport with pre-computation and reconstructed GI interactively. Fabianowski and Dingliana [FD09] achieved interactive photon mapping in static scenes by differential photon tracing with a pre-computed KD-tree. Wang et al. [WWZ*09] dynamically built a KD-tree on the GPU for tracing primary and photon rays and the shading points are partitioned with K-means clustering for final gathering. McGuire and Luebke [ML09] proposed an interactive photon mapping technique in image space, which however still uses a pre-computed KD-tree of the scene for photon tracing. A recent technique [KD10] simulated light propagation with lattices for approximating one-bounce indirect lighting in real-time without any pre-computation.

Multi-Image Representation Image representation of geometry has been used extensively in the literature of computer graphics. Depth peeling [Eve01] samples the scene from a single viewpoint and stores multiple layers of depth values. Shanmugam and Arikan [SA07] used depth peeling to generate sample buffers for estimating ambient occlusion. Obviously, sampling the whole scene from a single viewpoint is impossible in most cases. Ritschel et al. [RGS09] used several cameras to capture the scene, but the positions of these cameras are manually set which are quite inaccurate for covering the whole surfaces. Many studies on viewpoint selection [KK88] [BDP00] [VFSH04] go beyond coverage and focus on determining “good viewpoints". Werner et al. [WPH*02] used multiple images to represent a single object. To the best of our knowledge, there has been no research on using multiple environment maps to represent complex scene geometry for photon mapping.

3. Overview

As same as traditional photon mapping, our technique consists of two passes: the photon tracing pass and the rendering pass. Figure 2 shows the process of our photon tracing pass. Photon information is encoded into photon textures. Each texel in the texture stores a photon’s position, direction, power and surface normal. Photons are traced through the scene in image space. The images used for intersection estimation in our method are environment maps representing the scene’s geometry. To construct the environment map for a certain center point, the surrounding objects are rendered to the six faces of a cube map. Unlike traditional environment maps that store colors, the maps used to represent geometry contain the positions, normals and material indices of the surrounding surfaces. Whenever a photon-surface intersection is detected, the geometric and material information is read from the environment map and is stored in the intersection textures which contain all the intersection information between photons and surfaces. New directions and powers are sampled based on the values in the intersection textures and they are stored in the photon textures for the next bounce. This process is repeated until the desired level

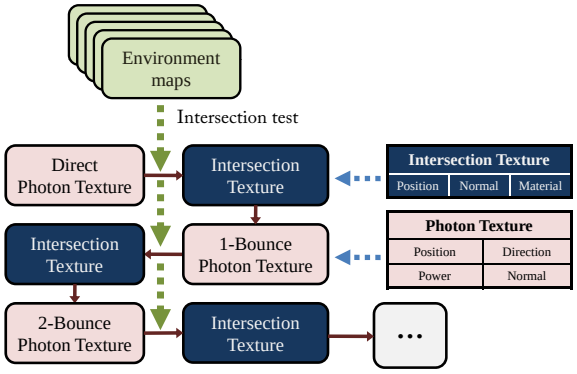


Figure 2: *Photon tracing process. Photon rays are stored in photon textures. With the help of multiple environment maps, intersections of these photon rays can be found and stored in intersection textures. Then new photon rays are generated by the geometry and material property of the intersections. This process is repeated until the required level of light bounces is reached.*

of light bounces is reached. Indirect illumination is produced by photon splatting to take advantage of the features of modern GPU's. The final result is the composite of this indirect illumination and direct lighting computed with shadow mapping and deferred shading.

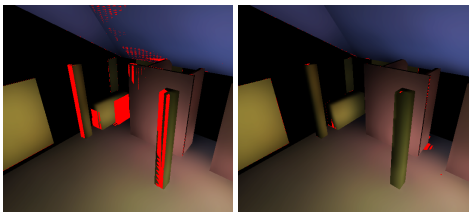
4. Selecting Environment Maps

When using multiple environment maps for photon mapping, it is important to appropriately select the maps to ensure adequate coverage of a given scene. Areas not covered by any environment map cannot receive photons, thus leading to inaccuracy. In the following we introduce an efficient method for quickly choosing environment maps with satisfactory scene coverage.

4.1. Image Selection Algorithm

Ideally one wishes to select a minimal number of environment maps that cover the entire scene. Such a formulation is equivalent to the classical set-covering problem and the art gallery guide problem, which are NP-hard [CLRS01] [OJ87]. We are going to describe a heuristic method that yields an approximate yet satisfactory results and, most importantly, is efficient enough for our purpose of interactive rendering.

Initially, a set U of 30 to 50 random points is generated inside the bounding box of the scene and we use these points as candidate *environment map centers* EMCs. Then we select a subset C of k EMCs in U and hope that these selected EMCs in C provide maximal scene coverage. The value of size k is defined by users; in our experiments, k is typically set to 8 to 15. Our selection algorithm runs as follows. Initially, the



(a) (b)

Figure 3: *Red pixels mask the uncovered regions. (a) Random selection of 8 EMCs; (b) Our selection of 8 EMCs out of 27 random EMCs;*

set C is set to be empty. In each step, for every unselected EMC $v \in U - C$, we compute the area that is visible from v and invisible from all the EMCs already in C, and call this area *new coverage area* of v and denote it by $A(v|C)$. Then the EMC v with largest new coverage area will be selected and added to C. We iterate this selection process k times to find k EMCs of C.

The above selection algorithm is used at the starting for initializing the k selected EMCs in C. To maintain a good coverage for a dynamic scene, the set C is updated regularly as follows. In each update step, we first select an EMC from $U - C$ that has the largest new coverage area and add it to C. Then among the resulting $k+1$ EMCs in C, the EMC v with the smallest new coverage $A(v|C - \{v\})$ is removed from C. Hence at any time a good coverage is maintained while keeping the number of EMCs in C unchanged. Note that the update process is carried out only in two cases. The first case is when finishing the initial selection. In this case, the updating step is repeated for several times until elements in C is invariant, i.e. the EMC added and the EMC removed are the same point. According to our experiments, the initial updating needs only 2 to 5 such update steps. The second case is when the scene is changed. When the geometry is changed continuously, this update process is run progressively, that is, the update is carried out once every 5 frames. Figure 3 shows a comparison between using randomly selected EMCs and using the EMCs selected with our heuristic method; although 8 maps are used in both cases, our selection obviously produces better coverage.

4.2. Computing New Coverage Area

A key problem in image selection is how to efficiently compute the new coverage area $A(v|C)$ by an EMC v. We compute it using shadow mapping [Wil78], based on the the observation that if we put a point light source at every $v' \in C$, the new covered region by v will be lit by none of these light sources. Therefore, the region must lie within the shadows cast by the light sources which are the EMCs of the existing set C; in fact, the required region is precisely the part of the shadow region that is visible from v. Shadows are gen-

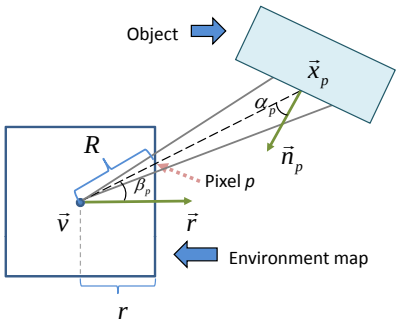


Figure 4: *Illustration of pixel covered area. $\vec{v}$: position of EMC, $\vec{x}_p$: object position, $\vec{n}_p$: object normal, $\vec{r}$: forward direction of camera, r: half side length of projection plane, R: distant between $\vec{v}$ and pixel in projection plane, α_p: angle between $\vec{n}_p$ and projection direction, β_p: angle between $\vec{r}$ and projection direction.*

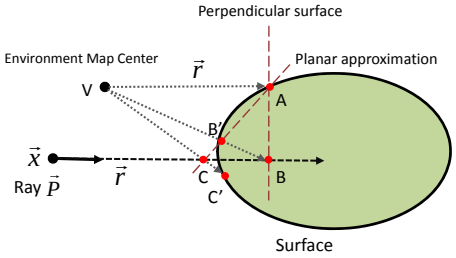


Figure 5: *Computing intersection by iterations in image-space. The first guess is point A which is obtained from EMC along the direction of the ray. After one iteration we get the approximated intersection B' and then C' after second iteration, and so on.*

erated using shadow mapping and the area is computed in image-space. Since the distance information for every pixel is available in the environment maps, the extra depth-map-generation step in a standard shadow map algorithm is unnecessary. The six faces of the cube map are rendered with shadow testing. The resolution used for this rendering is $32 \times 32 \times 6$; at such a relative low resolution the shadow algorithm can run efficiently and the accuracy is sufficient for our purposes.

The new coverage area by v is computed by summing up the areas of all shadow pixels p in the image:

$$A(v|C) = \sum_p A(p)S(p) = \frac{4}{N_{\text{pixel}}} \sum_p \frac{((\vec{x}_p - \vec{v}) \cdot \vec{r})^3}{(\vec{v} - \vec{x}_p) \cdot \vec{n}_p} S(p) \qquad (1)$$

where N_{pixel} is the number of pixels in the image, $\vec{v}$ is the position of the EMC v, $\vec{x}_p$ and $\vec{n}_p$ are the position and normal of the surface projected at p, and $\vec{r}$ is the forward direction of the camera, as shown in Figure 4. $S(p)$ is shadow function, i.e. $S(p) = 1$ for p in shadow, 0 otherwise. The derivation of Equation 1 is presented in Appendix A. In our implementation, the summation is obtained with the aid of an image pyramid, whose top level produces the desired value.

5. Image-Space Intersection Estimation

Another key problem with multiple-image representation of a scene is how to reliably compute the intersection of photon rays and scene geometry. In this section we will first explain the basic idea of the method used in previous works involving a single environment map, point out its pitfalls, and discuss how this computation can be carried out robustly when using multiple environment maps.

5.1. Ray-geometry intersection with a single image

The issue of computing ray-geometry intersections has been encountered in some earlier works, such as tracing specular reflected rays by environment maps [SKALP05], or tracing caustic rays by an image of light view space [SKP07] [Wym08]. Their methods all use a single image for intersection computation because they just need to find hit-points of specular reflected rays in simple scenes.

When a scene is represented by an environment map with the center point V, the position information of the scene geometry is encoded in the texels of the map. Consider a ray originating from $\vec{x}$ and pointing in the direction of a normalized vector $\vec{r}$. The ray is defined as $\vec{P} = \vec{x} + d \cdot \vec{r}$, where d is the distance from the starting point $\vec{x}$. Suppose that the ray intersects the scene surface represented by the environment map under consideration. The environment map can be modeled as an implicit function $f(X) = 0$. Finding the ray-surface intersection is then equivalent to finding a zero of the equation $\vec{f}(\vec{P}) = f(\vec{x} + d \cdot \vec{r}) = 0$. There are several existing algorithms for solving this problem. Below we briefly introduce the one from [SKALP05], which is adopted in our implementation.

Let S denote the surface represented by the environment map. The process for finding the intersection of S with a ray $\vec{P}$ from $\vec{x}$ along a direction $\vec{r}$ is illustrated in Figure 5. We first use the ray from V (i.e. the EMC) along the direction $\vec{r}$ to produce the initial guess A; this is done by looking up the distance from the environment map. Suppose that we now approximate the surface S by the plane passing through A and perpendicular to $\vec{r}$. Then the next approximate intersection points B is obtained as the intersection of this plane and the ray $\vec{P}$. By looking up the distance information again along the direction $\overrightarrow{VB}$, we have a corrected point B' on the actual surface S. If the distance $|\text{VB}|$ is larger than $|\text{VB}'|$, it is called an *overshoot*, otherwise an *undershoot*. Next the surface S is approximated by a plane passing through points A and B$'$ and perpendicular to the plane defined by A, $\vec{x}$ (the

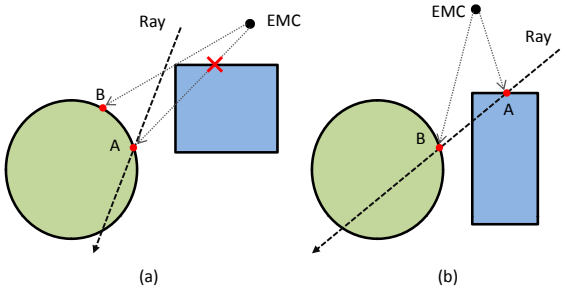


Figure 6: *Problems of intersection estimation with a single environment map. (a) Required point is not stored in the environment map; (b) Obtained result may not be the first intersection along the ray.*

origin of the ray) and V. The intersection between this plane and the ray $\vec{P}$ is point C, which is an undershoot. Distance lookup will correct the estimate C to the point C' on S. We continue this iterative process using points B' and C' for the next iteration. In each iteration an undershooting point and an overshooting point are kept to keep the refinement going.

The above method is in fact the false position method [Wei03] for root finding. In practice, this method converges quickly (if it does converge–see below) and can find reasonably accurate intersections within 3 to 5 iterations in most places of our test scenes.

Even with just one environment map, the above method from [SKALP05] has two problems causing erroneous results. One is that the desired intersection point may not exist in the environment map due to object occlusion, as shown in Figure 6 (a). In this case the ray intersects the scene at the point A, which is invisible from the EMC and thus not encoded in the map. As a result, the nearest visible point B is returned instead. The other problem is that the obtained result may not be the first intersection point along the ray. For example, in Figure 6 (b), either the point A or the point B may be produced by the method as the intersection point of the ray with the scene, depending on the initial guess used. On the other hand, B is a wrong output and only A is acceptable since it is the first intersection on the ray.

These problems with the existing ray-geometry intersection method become prominent when the source point of the ray is far from the EMC. Previous applications of this algorithm are usually on tracing reflected rays of a single reflective object, which is relatively local. However, these problems become more serious when applied to photons tracing since photon rays may be shot from any place over the scene.

5.2. Ray-geometry intersection with multiple images

When using multiple environment maps for photon mapping, we need to address the above problems with the existing ray-geometry intersection method to ensure robust and accurate computation. Suppose that several environment maps with different EMCs are used and photon rays are tested for intersections with each of these environment maps, which produce multiple intersection points. Our task is to detect those improper results and select only those acceptable intersection points.

We start with a classification of the intersections that can be generated. There are four types of possible resulting points, as shown in Figure 7:

- **Type A:** A point at infinity, which indicates no intersection with geometry;
- **Type B:** A point on the surface but not on the ray due to the problem in Figure 6 (a);
- **Type C:** An intersection point but not the first one due to the problem in Figure 6 (b);
- **Type D:** A proper intersection.

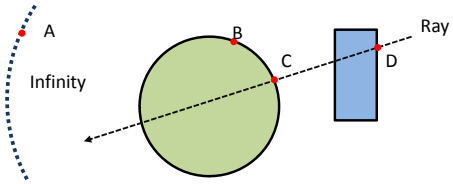


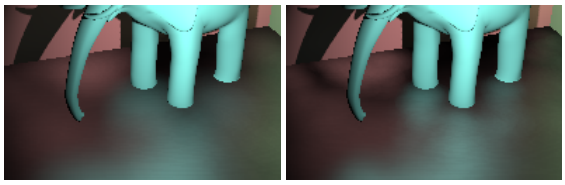
Figure 7: *Four types of possible results of the intersection computation.*

When an intersection point is returned by any environment map for a ray, the proper intersection is identified by the following procedure. First, a type A point can easily be identified and the ray will not be further pursued. To detect a type B intersection $\vec{p}$, an error e is computed by the angle deviation θ between the direction indicated by $\vec{p}$ and the direction $\vec{r}$ of the ray $\vec{P}$, evaluated as

$$e = 1 - \frac{(\vec{p} - \vec{x})}{||\vec{p} - \vec{x}||} \cdot \vec{r} \approx \theta^2/2. \quad (2)$$

Only points with errors smaller than a predefined ε are accepted. We set the threshold to $\varepsilon = 0.001$ in practice and reject the point as a type B point if $e > \varepsilon$; in this case an accepted point, even if it is of type B, lies very close to the ray and so is acceptable. Finally, to distinguish a point of type C from the true first intersection, distances to the origin of the ray are compared and the one with the smallest distance is chosen as the final, correct, result. If no acceptable intersection point is found the photon ray will not be traced further.

Modern graphics cards typically have a limited number of concurrent texture count, i.e. the environment maps are not all accessible in a single pass. Therefore the process needs to be broken down into several passes. In each pass, all the photon rays are tested with a single environment map. Points not on the ray are rejected. The distance of an accepted point to the origin of the ray is stored as the depth value of the



Fixed bandwidth Variable bandwidth

Figure 8: *Glossy reflection of Phong material with exponent=100. Fixed bandwidth splatting produces too blurry reflection of the limbs and shadows, while variable bandwidth splatting greatly alleviates this problem.*

pixel, thus comparison of distances can be performed automatically by depth testing of the graphics hardware.

6. Rendering

For the second pass, rendering, we adopt the photon splatting method [SB97] [LP03] for indirect illumination, with only little variations. In this section we will outline of the procedure briefly to make the exposition self-contained.

Photons are splatted onto screen to provide light for nearby pixels within a radius of influence h. Irradiance is computed and accumulated for each covered pixel whose geometric information is provided by the deferred-shading geometry buffers. The radius h is defined as $h = C\sqrt{(A/N)}$, like that in [LP03], where A is the total area of the scene, N is the number of emitted photons, and C is a parameter that controls the global bandwidth, which is a balance between variance and bias. We use a variable bandwidth approach like that in [HHK*07]. C is selected according to the probability density function (PDF): $C = C_0/\sqrt{p}$, where C_0 is a global parameter and p is the PDF which is clamped to a certain range $[P_{\min}, P_{\max}]$. The PDF is approximated as $p = \cos\theta/d^2$ where d is the traveled distance of the photon ray and θ is the incident angle. See Figure 8 for the difference between fixed bandwidth and variable bandwidth.

In order to reduce fill rate, a multi-level up-sampling strategy is adopted. Since indirect illumination is often quite smooth, photons can be splatted at a low resolution, which is then up-sampled progressively to full resolution. Places with high-frequency illumination such as corners are recomputed at higher resolution. The up-sampling process is performed multiple times, as shown in Figure 9, like the reverse of a mip-map generation process. Note that it is not the final radiance, but the incoming irradiance on the diffuse surfaces, that is up-sampled because the final radiance could be of high-frequency caused by factors like textures. In addition, for glossy surfaces, up-sampling is only performed for its diffuse component, while the glossy component is computed at full resolution.

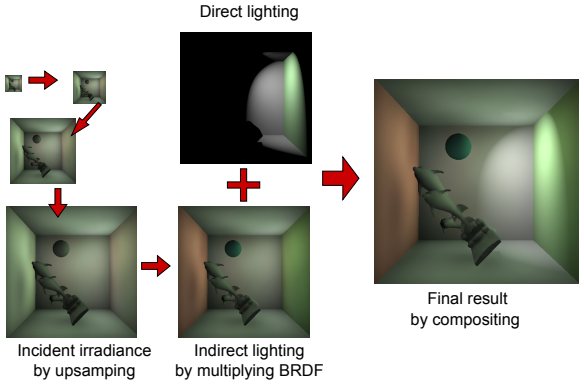


Figure 9: *Rendering with multi-level up-sampling. Incident irradiance is first rendered to a small image. Then it is up-sampled to a higher resolution. This up-sampling is performed multiple times until the full resolution is constructed. The irradiance is multiplied by the BRDF recorded for every pixel in screen-space to produce indirect illumination, which is combined with direct lighting to get the final result.*

7. Implementation

Our implementation is on DirectX 9.0c and HLSL with shader model 3.0. The initial photon textures are generated like reflective shadow maps, i.e. the scene is rendered using the light source as the viewpoint. Four textures that contain the position, direction, flux, and surface normal are generated. Like traditional photon mapping, caustic photons are stored separately in an additional texture. Caustic photons are only shot towards the bounding box of specular objects.

Random numbers needed for photon tracing are pregenerated and stored in a random number texture. To maintain temporal coherence, these random numbers are not changed when changing elements such as light sources or geometry of the scene to reduce the variations between consecutive frames.

Direct lighting is computed by deferred shading with geometry buffers in screen-space. Shadows are generated by shadow mapping with simple PCF [RSC87] for anti-aliasing. We adopt the method in [SKALP05] for reflection and refraction on specular surfaces. The final image is gamma corrected with $\gamma = 2.2$.

8. Results

Our experiments are done on an NVIDIA Geforce 260 GTX with 512MB video memory, and the CPU is an Intel Core 2 Duo running at 2.2GHz with 2GB system memory.

Figure 1 demonstrates global illumination effects computed with our method. There is no pre-computation so all elements can be changed interactively. The first scene demonstrates diffuse indirect illumination inside a box, with

three bounces of indirect light. The second scene shows reflective caustics caused by a dimpled specular surface. The third scene is a model of a living room, containing glossy and glass materials, and some caustic effects under the ball. The last one is the Sibenik model, which is relatively complex, that requires 12 environment maps to sample the geometry while all other scenes are sampled using 8 maps.

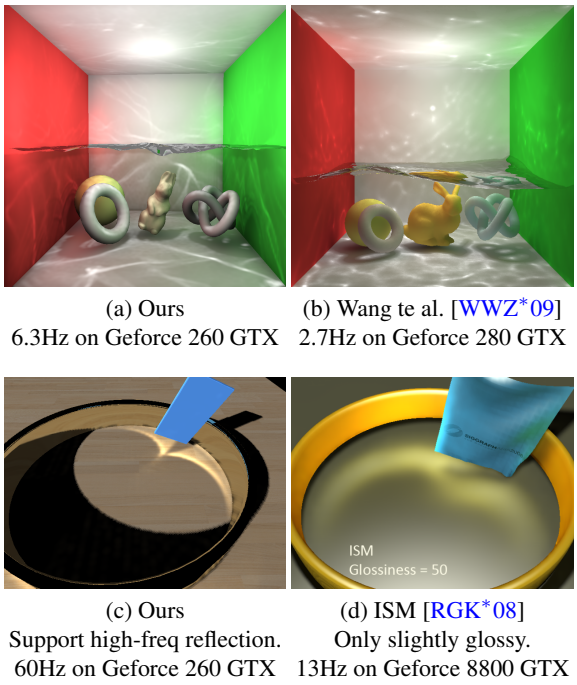
Comparison with other techniques are shown in Figure 10. The first row shows a dynamic water scene with the illumination under the water surface entirely produced by caustics. Before ours, Wang et al.'s technique [WWZ*09] is the only technique which can interactively simulate this type of illumination without pre-computation. Our technique produces similar effects with better performance even on a weaker graphics card. The second row shows an occluder inside a ring. ISM technique [RGK*08] supports low frequency reflection only, so the caustics and shadow are quite blurry. High frequency effects such as specular caustics are supported by our technique, and the occluder's shadow on the caustics is cast correctly with a clear edge.

We also compare our results with some off-line techniques: path tracing for diffuse inter-reflection effects and traditional photon mapping for caustic effects (Figure 11). It is clear that our technique can capture all the GI effects such as color bleeding, shape of indirect shadows and caustics.

Table 1 shows the performance of our examples rendered at both resolutions of 1024×1024 and 512×512. All elements including light, geometry and camera can be changed interactively. The EMC selection step takes between 100 ms to 300 ms, but it is called only once during initialization. A progressive EMC update is carried out whenever the scene is changed. When the scene changes continuously, the update is performed every 5 frames and the amortized time is between 1 ms and 10 ms per frame.

Fig.	T	GP	CP	B	$FPS_{512^2/1024^2}$
1 (a)	15k	376k	0	3	22.3 / 11.4
1 (b)	21k	91k	89k	1	13.6 / 12.8
1 (c)	84k	403k	2.9k	3	7.2 / 5.4
1 (d)	79k	739k	0	2	13.3 / 5.5
11 (a)	58k	464k	0	1	20.8 / 9.2
11 (c)	101k	358k	0	2	15.1 / 7.5
11 (e)	2.1k	0	164k	1	38.9 / 28.2
10 (a)	54k	536k	822k	3	7.4 / 6.3

Table 1: *Performance Statistics. T = number of triangles; GP = number of global photons; CP = number of caustic photons; B = number of light bounces.*



(a) Ours 6.3Hz on Geforce 260 GTX
(b) Wang te al. [WWZ*09] 2.7Hz on Geforce 280 GTX
(c) Ours Support high-freq reflection. 60Hz on Geforce 260 GTX
(d) ISM [RGK*08] Only slightly glossy. 13Hz on Geforce 8800 GTX

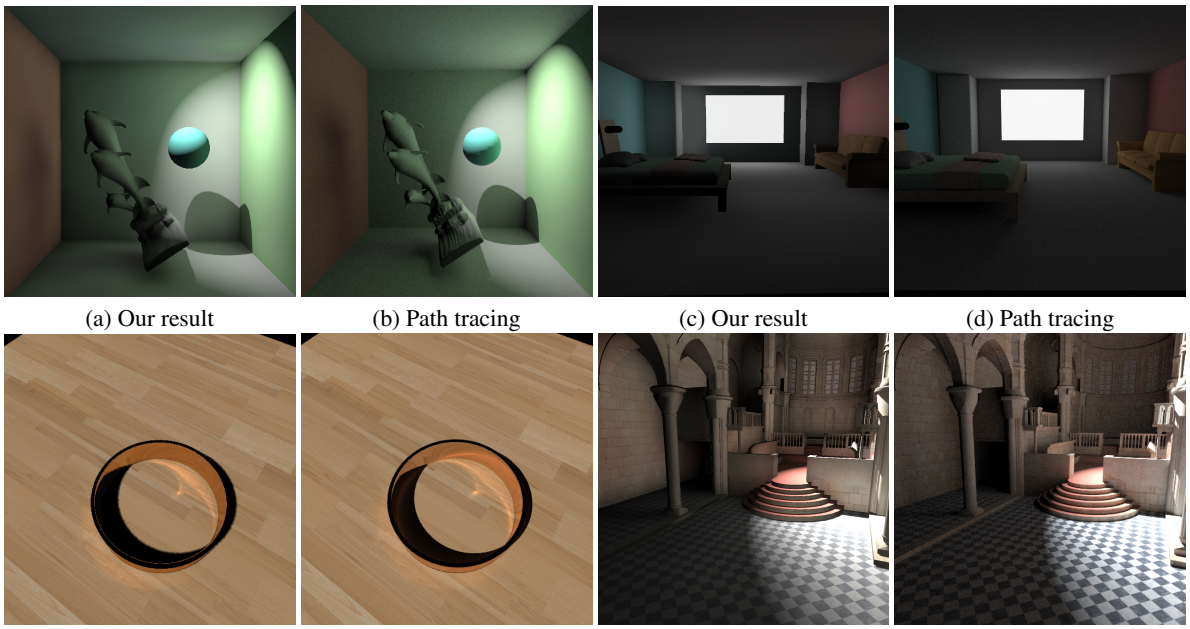
Figure 10: *Comparison with other techniques. Images and performance of other techniques are copied from their papers. (a) and (b) in* 1024×1024*; (c) and (d) in* 640×480*.*

9. Discussions

9.1. Image-Space Intersection

Advantages Compared with geometry-space ray tracing, an image-space method has several advantages. Firstly, because of the GPU's powerful rasterization and texture lookup capabilities, the cost of constructing environment maps on the fly is so cheap that dynamic scenes are supported without pre-computations. Secondly, intersection computation time in image-space is not affected by geometry complexity because it depends more on image resolution which is rather constant. Thirdly, image space algorithms can easily adapt to objects of almost any geometric representation as long as they can be rendered using graphics hardware. Finally, its simplicity makes the implementation easy. In our implementation, we utilize only the most common graphics hardware features such as environment mapping, depth and stencil testing and alpha blending. It does not depend on GPGPU framework such as CUDA so it is easy to integrate it into existing games or other real-time applications.

Accuracy Intersections are obtained in image-space. The discrete nature of pixels could cause errors during intersection testing. However, it does not cause much noticeable artifact in the final result because each photon illuminates an area, when overlapped and accumulated, the errors are usually averaged out, see Figure 12.



(a) Our result (b) Path tracing (c) Our result (d) Path tracing

(e) Our result (f) Photon mapping (g) Our result (h) Path tracing

Figure 11: . *Comparisons of our results (5-20 Hz) and results by traditional techniques (minutes to hours).*

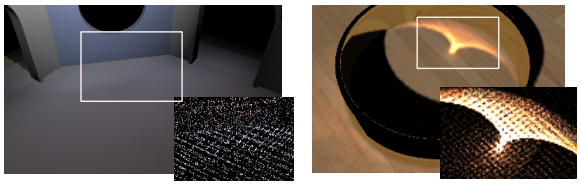


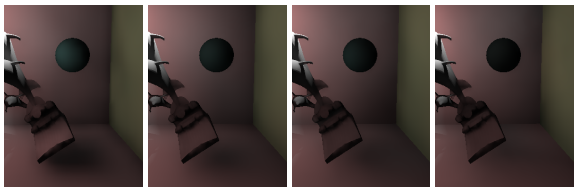
Figure 12: *Rendering results of global and caustic photons and their point distributions (lower right). The photons are distributed rather regularly due to the discrete nature of pixels, but no visible artifact appear in the final results.*

9.2. Limitations

Geometry Complexity The intersection algorithm in image-space is independent of number of triangles, but obviously complex topology of the scene requires more environment maps to sample the whole scene. Our examples show that for a single room, even with complex objects inside, usually 10 maps are sufficient. However, more complex scenes such as a building with many rooms could require dozens of environment maps. For such situations, it is possible that our technique will no longer work due to limited amount of video memory and processing speed. However, this is a common problem of all existing interactive GI techniques. In this paper we demonstrate equal or more complex test scenes than previous no-pre-computation methods [WWZ*09] [DS05] [NW09] or light-pre-computation methods [RGK*08]. Some techniques [LZT*08] [ML09] can handle very complex scenes with the help of a time-consuming pre-computation. In fact, if the scenes were pre-processed for visibility, our method has the potential to handle similar complex scenes in an out-of-core manner by maintaining a large number of environment maps and uses just a few of them inside the current potentially visible region for photon tracing. However, we only focus on the method without pre-computation, so handling complex scenes with pre-computation based methods is outside the scope of this paper.

The factor of geometry quantity (e.g. number of triangles) limits the performance mainly during the stage of EMC selection because in that stage the scene needs to be rendered into low-res environment maps for each EMC. Since exact accuracy is not necessary, we can adopt the imperfect shadow map technique [RGK*08] to speed up rendering, with the expense of an extra pre-processing stage. Other techniques, such as occlusion culling and level of details rendering, are also applicable.

Bias In Figure 11, some bias is visible in our results. The bias is caused by two factors. First, photon mapping is a biased rendering technique therefore our method also inherit its weakness such as blurring and boundary bias (darken on edges), such as the blurred indirect shadow in Figure 11 (g). Nevertheless, since photon mapping is consistent, the bias can be reduced by using more photons. Figure 13 shows the balance between performance and quality shifts as the number of photon changes. The other factor is up-sampling, which lead to incorrect blurring and other errors,



459k / 4.2Hz 75k / 9.8Hz 35k / 14.5Hz 7.7k / 25.5Hz

Figure 13: *Quality and performance with different number of photons. Label under the image indicates Photon Count / Speed.*

such as edges in Figure 11 (c). This bias can be reduced by re-computation at higher resolution, with a tradeoff between performance and quality.

9.3. Comparisons with Similar Techniques

Here we compare our technique with other momentous works on global illumination in the past few years.

RSM [DS05] [DS06] [NW09] produces one-bounce indirect illumination only. Besides, indirect occlusions are totally ignored. In contrast, multiple bounces of indirect light and indirect shadows are fully supported in our technique.

ISM [RGK*08] handles dynamic scenes but requires preprocessing and only deformation based on original geometric representation is supported. Additionally, it is only suitable for diffuse or slightly glossy materials, while ours supports high frequency glossy materials (Figure 8) and specular caustics (Figure 10 (c)).

McGuire and Luebke [ML09] use similar ways as ours to store photons in textures and use splatting for rendering. Nevertheless, since they need to pre-compute a KD-tree of the scene geometry on CPU, dynamic scenes are only limited to rigid movement of some pre-defined objects.

Compared with [WWZ*09], which is able to produce GI of a box scene at 2 to 5 Hz, we can also create most of the visual effects that their method produces (Figure 10 (a)(b)), except for specular reflection and refraction effects which they achieve by ray tracing (we use the technique in [SKALP05], which only produces approximated single-bounce specular reflection). The biggest advantage is that our method is straightforward and is simple to implement. Their implementation is very complex and challenging, not only for the technique itself but also for the techniques it relies on, e.g. a dynamic GPU KD-tree. Moreover, their implementation depends heavily on special features, such as arbitrary gathering and scattering of GPU memory, of advanced graphics hardware (e.g. NVIDIA's CUDA). In contrast, our technique does not require any GPGPU function and can work with very basic GPU features.

10. Conclusion

We have presented a simple GPU-based photon mapping technique that traces photon rays in image-space to produce global illumination interactively. To tackle the difficult problem of handling global photon mapping in complex scenes, we represent scene geometry using multiple environment maps. We described an effective method for selecting environment map centers to provide adequate coverage of scene geometry and discussed how to robustly compute ray-geometry intersections with multiple environment maps. With photon splatting in the rendering stage, the whole photon mapping algorithm is implemented entirely on GPU. Global illumination is fully recomputed at every frame at interactive rate, supporting dynamic objects and light sources without the need for preprocessing.

11. Acknowledgements

We thank the anonymous reviewers for their helpful comments and the great help of Prof. Wenping Wang from the University of Hong Kong. This work was supported by National Basic Research Program of China (Grant No. 2010CB328001), National Science Foundation of China (Grant Nos. 60773143,90818011), and National High-Tech Research & Development Program of China (Grant No. 2007AA040401). Prof. Paul was supported by ANR-NSFC(60911130368), and Prof. Yong was supported by the Fok Ying Tung Education Foundation (111070).

References

[BDP00] BARRAL P., DORME G., PLEMENOS D.: Scene understanding techniques using a virtual camera. In *Proceedings of Eurograph 2000, short presentations* (2000).

[CLRS01] CORMEN T. H., LEISERSON C. E., RIVEST R. L., STEIN C.: *Introduction to Algorithms*. MIT Press, 2001.

[DS05] DACHSBACHER C., STAMMINGER M.: Reflective shadow maps. In *Proceedings of the Symposium on Interactive 3D Graphics and Games* (2005), pp. 203–231.

[DS06] DACHSBACHER C., STAMMINGER M.: Splatting indirect illumination. In *Proceedings of the Symposium on Interactive 3D Graphics and Games* (2006), pp. 93–100.

[DSDD07] DACHSBACHER C., STAMMINGER M., DRETTAKIS G., DURAND F.: Implicit visibility and antiradiance for interactive global illumination. *ACM Trans. on Graphics (Proc. ACM SIGGRAPH 2007) 26*, 3 (2007), 61–es.

[Eve01] EVERITT C.: *Interactive order-independent transparency*. Technical report, NVidia, 2001.

[FD09] FABIANOWSKI B., DINGLIANA J.: Interactive global photon mapping. *Computer Graphics Forum 28*, 4 (July 2009), 1151–1159.

[HHK*07] HERZOG R., HAVRAN V., KINUWAKI S., MYSZKOWSKI K., SEIDEL1 H.-P.: Global illumination using photon ray splatting. *Computer Graphics Forum 26*, 3 (2007), 503–513.

[HMS06] HUNT W., MARK W. R., STOLL G.: Fast kd-tree construction with an adaptive error-bounded heuristic. In *IEEE Symposium on Interactive Ray Tracing* (2006), pp. 81–88.

[Jen96] JENSEN H. W.: Global illumination using photon maps. In *Proceedings of the eurographics workshop on Rendering techniques' 96* (1996), pp. 21–30.

[Jen01] JENSEN H. W.: *Realistic image synthesis using photon mapping*. A. K. Peters, Ltd, 2001.

[KD10] KAPLANYAN A., DASCHBACHER C.: Cascaded light propagation volumes for indirect illumination. In *Proceedings of the Symposium on Interactive 3D Graphics and Games* (2010), pp. 99–107.

[Kel97] KELLER A.: Instant radiosity. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques* (1997), pp. 49–56.

[KK88] KAMADA T., KAWAI S.: A simple method for computing general position in displaying three-dimensional objects. *Computer Vision, Graphics, and Image Processing 41*, 1 (Jan. 1988), 43–56.

[LC04] LARSEN B. D., CHRISTENSEN N. J.: Simulating photon mapping for real-time applications. In *Proceedings of Eurographics Symposium on Rendering* (2004).

[LP03] LAVIGNOTTE F., PAULIN M.: Scalable photon splatting for global illumination. In *GRAPHITE* (2003), pp. 203–ff.

[LZT*08] LEHTINEN J., ZWICKER M., TURQUIN E., KONTKANEN J., DURAND F., SILLION F. X., AILA T.: A meshless hierarchical representation for light transport. *ACM Trans. on Graphics (Proc. SIGGRAPH 2008) 27*, 3 (2008), 1–9.

[ML09] MCGUIRE M., LUEBKE D.: Hardware-accelerated global illumination by image space photon mapping. In *Proceedings of the 1st ACM conference on High Performance Graphics* (2009), pp. 77–89.

[NW09] NICHOLS G., WYMAN C.: Multiresolution splatting for indirect illumination. In *Proceedings of the Symposium on Interactive 3D Graphics and Games* (2009), pp. 83–90.

[OJ87] O'ROURKE, JOSEPH: *Art Gallery Theorems and Algorithms*. Oxford University Press, 1987.

[POC05] POLICARPO F., OLIVEIRA M. M., COMBA J. L. D.: Real-time relief mapping on arbitrary polygonal surfaces. *ACM Trans. on Graphics (Proc. ACM SIGGRAPH 2005) 24*, 3 (2005), 935–935.

[RGK*08] RITSCHEL T., GROSCH T., KIM M. H., SEIDEL H.-P., DACHSBACHER C., KAUTZ J.: Imperfect shadow maps for efficient computation of indirect illumination. *ACM Trans. on Graphics (Proc. ACM SIGGRAPH Asia 2008) 27*, 5 (2008), 1–8.

[RGS09] RITSCHEL T., GROSCH T., SEIDEL H.-P.: Approximating dynamic global illumination in image space. In *Proceedings of the Symposium on Interactive 3D Graphics and Games* (2009), pp. 75–82.

[RSC87] REEVES W. T., SALESIN D., COOK R. L.: Rendering antialiased shadows with depth maps. *Computer Graphics (Proc. ACM SIGGRAPH 1987)* (1987), 283–291.

[SA07] SHANMUGAM P., ARIKAN O.: Hardware accelerated ambient occlusion techniques on gpus. In *Proceedings of the Symposium on Interactive 3D Graphics and Games* (2007), pp. 73–80.

[SB97] STÜRZLINGER W., BASTOS R.: Interactive rendering of globally illuminated glossy scenes. In *Proceedings of the Eighth Eurographics Workshop on Rendering* (1997), pp. 93–102.

[SKALP05] SZIRMAY-KALOS L., ASZÓDI B., LAZÁNYI I., PREMECZ M.: Approximate ray-tracing on the gpu with distance impostors. *Computer Graphics Forum 25*, 3 (2005), 695–704.

[SKP07] SHAH M., KONTTINEN J., PATTANAIK S.: Caustics mapping: An image-space technique for real-time caustics. *IEEE Transactions on Visualization and Computer Graphics 13*, 2 (2007), 272–280.

[SSK07] SHEVTSOV M., SOUPIKOV A., KAPUSTIN A.: Highly parallel fast kd-tree construction for interactive ray tracing of dynamic scenes. *Computer Graphics Forum 26* (2007), 395–404. (Proc. Eurographics 2007).

[SZS*08] SUN X., ZHOU K., STOLLNITZ E., SHI J., GUO B.: Interactive relighting of dynamic refractive objects. *ACM Trans. on Graphics (Proc. ACM SIGGRAPH 2008) 27*, 3 (2008), 1–9.

[VFSH04] VÁZQUEZ P.-P., FEIXAS M., SBERT M., HEIDRICH W.: Automatic view selection using viewpoint entropy and its application to image-based modelling. *Computer Graphics Forum 22*, 4 (Nov. 2004), 689–700.

[Wei03] WEISSTEIN E.: World of mathematics. *http://mathworld.wolfram.com/Curvature.html* (2003).

[Wil78] WILLIAMS L.: Casting curved shadows on curved surfaces. *Computer Graphics (Proc. ACM SIGGRAPH 1978)* (1978), 270–274.

[WKB*02] WALD I., KOLLIG T., BENTHIN C., KELLER A., SLUSALLEK P.: Interactive global illumination using fast ray tracing. In *Proceedings of the 13th Eurographics workshop on Rendering* (2002), pp. 15–24.

[WPH*02] WERNER T., PAJDLA T., HLAVÁČ V., LEONARDIS A., MATOUŠEK M.: Selection of reference images for image-based scene representations. *Computing 68*, 2 (Mar. 2002), 163–180.

[WWZ*09] WANG R., WANGY R., ZHOU K., PAN M., BAO H.: An efficient gpu-based approach for interactive global illumination. *ACM Trans. on Graphics (Proc. ACM SIGGRAPH 2009) 28*, 3 (2009), 1–8.

[Wym08] WYMAN C.: Multiresolution splatting for indirect illumination. In *Proceedings of the Symposium on Interactive 3D Graphics and Games* (2008), pp. 163–171.

[ZHWG08] ZHOU K., HOU Q., WANG R., GUO B.: Real-time kd-tree construction on graphics hardware. *ACM Trans. on Graphics (Proc. ACM SIGGRAPH Asia 2008) 27*, 5 (2008), 1–11.

Appendix A: Derivation of Equation 1

Here we derive the expression of the covered area of pixel p, $A(p)$, with notations shown in Figure 4. We have

$$A(p) = \frac{\Delta\omega_p ||\vec{x}_p - \vec{v}||^2}{\cos\alpha_p} = \frac{A_0 \cos\beta_p ||\vec{x}_p - \vec{v}||^2}{R^2 \cos\alpha_p}$$

where $\Delta\omega_p$ is the solid angle subtended by pixel p and A_0 is the the image area of each pixel, then:

$$\begin{aligned} A(p) &= \frac{A_0 \cos^3\beta_p ||\vec{x}_p - \vec{v}||^2}{r^2 \cos\alpha_p} \\ &= \frac{A_0}{r^2} \frac{\left(\frac{\vec{x}_p - \vec{v}}{||\vec{x}_p - \vec{v}||} \cdot \vec{r}\right)^3 ||\vec{x}_p - \vec{v}||^2}{\frac{\vec{v} - \vec{x}_p}{||\vec{v} - \vec{x}_p||} \cdot \vec{n}_p} \\ &= \frac{A_0}{r^2} \frac{((\vec{x}_p - \vec{v}) \cdot \vec{r})^3}{(\vec{v} - \vec{x}_p) \cdot \vec{n}_p} \end{aligned}$$

Recalling that the area of each pixel is the total image area divided by the number of pixels: $A_0 = 4r^2/N_{\text{pixel}}$, we get the expression in Equation 1.