COMBINING IMAGE-SPACE INSTANT RADIOSITY AND PHOTON MAPPING FOR REAL-TIME DIFFUSE INDIRECT ILLUMINATION AND CAUSTICS

LE HOANG QUYEN
BEng

A DISSERTATION SUBMITTED FOR THE DEGREE OF MASTER OF COMPUTING

DEPARTMENT OF COMPUTER SCIENCE
NATIONAL UNIVERSITY OF SINGAPORE

2014

Declaration

I hereby declare that this dissertation is my original work and it has been written by me in its entirety. I have duly acknowledged all the sources of information which have been used in the dissertation.

This dissertation has also not been submitted for any degree in any university previously.

Le Hoang Quyen
August 2014

Acknowledgments

I would like to acknowledge Professor Kok-Lim LOW, Ph.D., for his advice for the completion of this dissertation. I am also very grateful for the cooperation with Mr. Hua Binh Son. The discussions with Son helped improve my understanding of concepts, current research trends, as well as ideas and directions for my topic.

Abstract

Global illumination greatly improves the realism of visual applications. Although frequently being used in off-line rendering, it is difficult to achieve in real-time applications. Several techniques has been invented to produce such effect in high performance frame rate, but most only consider diffuse inter-reflection and ignore visibility, as well as sacrificing accuracy for speed. This dissertation introduces a hybrid image-space technique being able to illustrate diffuse indirect illumination and caustics in real-time. This technique combines an Instant Radiosity based algorithm that supports only diffuse inter-reflection with a Photon Mapping based approach to render the caustic effects. For fast approximations, the Photon Mapping approach uses image-space informations to trace photons and spread their energy to nearby pixels. Ray-geometry intersections are solved on a per-fragment basis by using per-pixel linked lists. The advantages of the technique are supporting dynamic scenes without pre-computing any accelerated structures such as kd-tree, and it runs entirely on the GPU.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Introduction

As photo-realistic is always an ultimate goal for computer graphics, only limited range of real life phenomena can be simulated in real-time and interactive frame rate. With Graphics Processing Units (GPUs) become more powerful, more advanced effects has been made possible while retaining acceptable frame rate. Among those, global illumination (GI) has been increasingly popular in games and real-time applications. This effect does not only account the direct light sources when displaying the illumination of an object, but also considers its surrounding environment.

There are two main components of GI, direct illumination and indirect illumination. Direct illumination appears on an object based on its relation with light sources. It can be implemented efficiently and runs very fast on today hardwares. Indirect illumination, on the other hand, is the result of multiple light bounces between objects. One example is that when the light is reflected on a wall having red color and reach surrounding objects, these objects will have similar red color on them (figure 1.1). There are three common indirect illumination effects: diffuse inter-reflection which is a result of light bounces between diffuse surfaces, caustics happening when light rays reflect or refract from a surface and focus in only certain areas of receiving objects, and glossy resulting from indirect light arriving on shiny surfaces after being reflected between other diffuse or shiny surfaces.

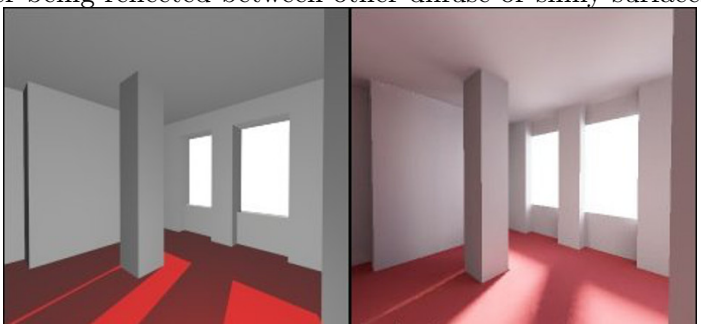


Figure 1.1: Direct illumination only (left) and indirect illumination (right).

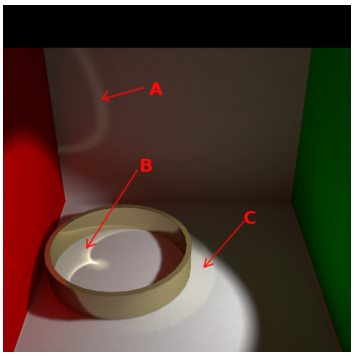


Figure 1.2: Caustics (A & B & C regions) caused by light reflected on a metal ring.

Although GI has been featured in off-line rendering for a long time, there is still no perfect solution even today for real-time counterpart. Since these applications typically have to produce an image in a fraction of second, full global illumination is still not feasible. The involvement of sampling a large number of light directions and recursive tracing their bounces between objects make this GI approach too complicated to be computed in short amount of time.

Several algorithms have been pioneered in attempts to produce real-time indirection illumination. They usually rely on some approximations or assumptions such as ignoring indirect visibility in order to achieve plausible GI. Due to low frequency nature of diffuse inter-reflection, it can be generated quickly in reasonable quality. While the other two remain being challenges due to requiring high number of light path samplings.

This dissertation investigates and evaluates a new hybrid image-space method to render two of the indirect illumination effects in real-time: diffuse inter-reflection and "middle" to slightly high frequency caustics. This method is image-space based and thus have very little dependence on scene complexity. It supports fully dynamic scene without pre-computing data structures, which means the scene can be rendered completely from scratch every frame.

- In first step, a multi-resolution Instant Radiosity shading approach is utilised to render diffuse effects. This approach is currently one of the fastest indirect diffuse algorithms, and it is screen-space based.

- In second step, a screen-space Photon Mapping method is applied to accumulate caustics to the diffuse illuminated scene from first step. This method has two sub-passes as traditionally:

    1. The Photon tracing pass employs the following strategy: the photon rays are rasterized as lines and collisions are test against per-pixel linked lists representing the scene's geometry.

2. The second pass is finding energy contribution of photons to their nearby pixels. To do this, a splatting method is adopted, which renders oriented disks around photons' hit points and spreads energy to every screen-space pixel residing within.

Even though Photon Mapping methods excel at rendering caustics and are capable of simulating a wide range of effects, they are not efficient to simulate diffuse indirect illumination. The reflectivity nature of diffuse surfaces causes the photons to be scattered all over the scene. If only a sparse set of photons are traced, visible noises can occur (figure 1.3), while a large number increases the performance cost. In contrast, caustics have high concentration of photons on certain areas, hence illumination discontinuities are hard to notice even if there are low number of traced photons. Therefore, in our method, we replace diffuse indirect lighting using photons part by an Instant Radiosity technique while keeping the caustics illumination of Photon Mapping method. Instant Radiosity uses Virtual Point Lights (VPLs) to illuminate the diffuse surfaces, however, the number of VPLs is at least one order of magnitude lower than photons'. Finally, in practice, the errors occur during photons tracing are usually averaged out when accumulating contributions of photons close to each other, thus we can approximate this step instead of performing the accurate ray tracing procedure. Image-space Photon tracing methods have an advantages of simple implementation, easy integration into existing OpenGL/Direct3D rendering frameworks, and no pre-computation since they can use the scene's geometry information in images such as depth buffer to calculate photons' intersections. Nevertheless, one image is not sufficient to store information of the entire scene, thus existing techniques uses multiple images in hope that they can cover the majority parts of the scene. These images are obtained by rendering the scene repeatedly from different perspectives. With the general purpose power of modern GPUs, our method employs GPU per-pixel linked lists to represent the geometry and it is possible to capture the entire scene in one rendering pass.

Figure 1.3: A sparse set of photons gives noises to the rendered image. Courtesy of Lavignotte et al. [LP03].

1.2 Dissertation's structure

In Chapter 2, we will review some existing GI algorithms, most of them are real-time/interactive focused methods. Chapter 3 will explain our technique in detail. Some results for evaluating our technique are presented in Chapter 4. Finally, conclusions and future improvements are discussed in Chapter 5.

Chapter 2

Related works

2.1 The Rendering Equation

Before going through a list of GI algorithms, it is important to understand GI problem first. This section will explain some relevant GI related radiometry concepts.

Solid angle (Ω) subtended by a surface S is a surface area of a unit sphere covered by projection of S on the sphere (figure 2.1). We often use the term $\mathrm{d}\omega$ to denote a differential solid angle at direction ω from the sphere's center. Radiant flux (Φ) is an amount of energy/power flowing through a surface per unit time. Irradiance (E) is area density of flux arriving at a surface (A):

$$E = \frac{\mathrm{d}\Phi}{\mathrm{d}A} \tag{2.1}$$

Radiance is the most important GI concept, it is defined as flux density per unit solid angle, per unit projected area.

$$L = \frac{\mathrm{d}\Phi}{\mathrm{d}\omega\ \mathrm{d}A\ \cos\theta} \tag{2.2}$$

Where θ is an angle between ω and and surface normal.

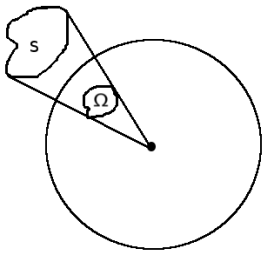


Figure 2.1: Solid angle Ω subtended by surface S.

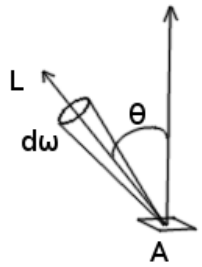


Figure 2.2: Radiance exits surface A through direction ω.

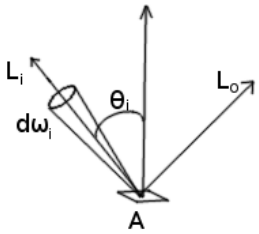


Figure 2.3: BRDF describes ratio of reflected radiance in direction ω_0 to irradiance from direction $-\omega_i$.

The last term is Bi-directional Reflectance Distribution Function (BRDF) which describes how much radiance reflected from a surface point x through direction ω_0 due to incident radiance at that surface point from direction -ω_i. Let L_r be the reflected radiance and L_i be the incident radiance, BRDF is given by the following formula:

$$f(x, \omega_0, \omega_i) = \frac{\mathrm{d}L_r(x, \omega_0)}{L_i(x, \omega_i)\ \cos\theta_i\ \mathrm{d}\omega_i} = \frac{\mathrm{d}L_r(x, \omega_0)}{\mathrm{d}E(x, \omega_i)} \tag{2.3}$$

Given these definitions, to simulate the Global Illumination's reflection model, we need to measure the radiance coming to our eye from every surface point in the scene by the following Rendering Equation:

$$L_r(x, \omega_0) = \int_{\Omega_x} f(x, \omega_0, \omega_i)\ L_i(x, \omega_i)\ \cos\theta_i\ \mathrm{d}\omega_i \tag{2.4}$$

Where ω_0 is the direction from the surface point x to the eye. The equation integrates over a hemisphere at x oriented toward its normal vector, gathering all incident radiance $L_i(x, \omega_i)$ which are reflected from light sources or other surrounding surface points. If only diffuse indirect lighting toward the eye is considered, equation 2.4 can be rewritten as:

$$
\begin{aligned}
L_r(x,\omega_0) = &\sum_{i=1}^{N} f(x,\omega_0,\omega_i)\ L_{i,l}(x,\omega_i)\ \cos\theta_i\ \mathrm{d}\omega_i \quad + \\
&\frac{p(x)}{\pi}\int_{\Omega_x} L_{i,d}(x,\omega_i)\ \cos\theta_i\ \mathrm{d}\omega_i \quad + \\
&\frac{p(x)}{\pi}\int_{\Omega_x} L_{i,c}(x,\omega_i)\ \cos\theta_i\ \mathrm{d}\omega_i
\end{aligned}
\tag{2.5}
$$

$$
\begin{aligned}
L_r(x,\omega_0) = &\sum_{i=1}^{N} f(x,\omega_0,\omega_i)\ L_{i,l}(x,\omega_i)\ \cos\theta_i\ \mathrm{d}\omega_i \quad + \\
&\frac{p(x)}{\pi}\int_{\Omega_x} L_{i,d}(x,\omega_i)\ \cos\theta_i\ \mathrm{d}\omega_i \quad + \\
&\frac{p(x)}{\pi}\int_{\Omega_x} \frac{\mathrm{d}^2\Phi_{i,c}}{\mathrm{d}A_i}
\end{aligned}
\tag{2.6}
$$

Where $p(x)$ is diffuse reflection coefficients of the surface. $L_{i,l}(x,\omega_i)$ is radiance via direct lighting from a light source in direction ω_i. $L_{i,d}(x,\omega_i)$ is indirect radiance via lighting reflected diffusely at least one on other surfaces. $L_{i,c}(x,\omega_i)$ is caustic radiance caused by specular reflection on another surface point.

In many cases, the integrations can be estimated by Quasi-Monte Carlo method which accumulates irradiance over a finite number of directions obtained by sampling randomly with probability density function $pdf(\omega)$:

$$
\int_{\Omega_x} f(x,\omega_0,\omega_i)\ L_i(x,\omega_i)\ \cos\theta_i\ \mathrm{d}\omega_i \approx \frac{1}{N}\sum_{i=1}^{N}\frac{f(x,\omega_0,\omega_i)\ L_{i,l}(x,\omega_i)\ \cos\theta_i}{pdf(\omega_i)} \tag{2.7}
$$

2.2 Instant Radiosity and Virtual Point Lights

2.2.1 Instant Radiosity

Keller presented Instant Radiosity in 1997 [Kel97]. This algorithm first shoots N particles from original light source at quasi-random directions. VPLs are created at every collision location between the particles and the scene's surfaces. These VPLs will be used as secondary light sources to illuminate the scene. After first collision with the scene's surface, each particle is determined to be terminated or bounced off the surface based on a probability calculated as:

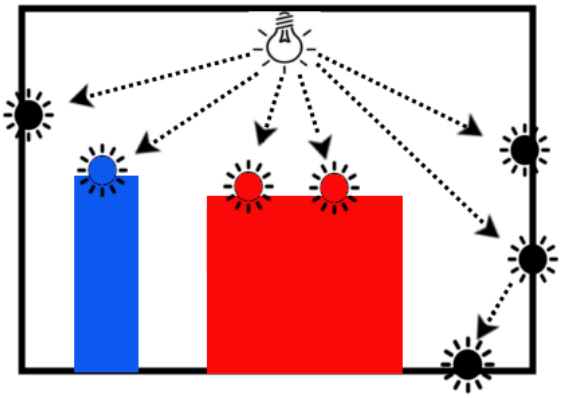


Figure 2.4: Virtual Point Lights ("shiny" dots) are created when particles shot from original light source collide with surfaces.

$$\overline{p} = \frac{\sum_{k=1}^{K} p_{d,k} \ |A_k|}{\sum_{k=1}^{K} |A_k|}$$

Where K is number of surface elements in the scene, each has area A_k and average diffuse reflectivity $p_{d,k}$.

If bouncing is chosen, the particle continues travelling and bouncing off several surfaces until it is terminated. There should be $\overline{p}$N particles remain after first-bounce, $\overline{p}^i$N after ith bounce.

The original implementation uses multiple rendering passes to accumulate lighting from multiple VPLs. Shadow Mapping can be used to check occlusions between VPLs and illuminated surfaces. This algorithm is more suitable for diffuse scenes. Glossy reflections can be generated with millions of VPLs, while caustics are nearly impossible. Caustics are results of lighting coming from specular reflections then arriving at diffuse surfaces, placing a VPL at a specular surface then using it to illuminate the scene will result in zero evaluation.

2.2.2 Reflective Shadow Maps

Dachsbacher et al. [DS05] proposed a real-time variant of Instant Radiosity in 2005, which supports once-bounce VPLs only. By using multi render targets, they extends traditional shadow map to store more information in light view. Besides depth value, there are world space position, normal and reflected flux of every pixel rendered from light view. This extension called Reflective Shadow Map (RSM), see figure 2.5. Every pixel in RSM is then considered a VPL (figure 2.6).

To illuminate the scene, direct lighting is rendered to geometry buffer (G-buffer) first. It is

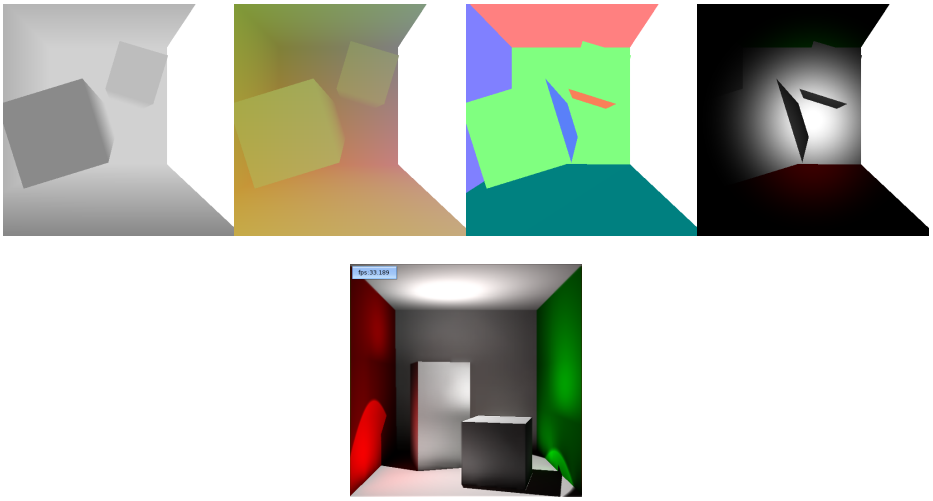


Figure 2.5: From left to right: depth, world space position, normal, flux in RSM obtained from light view, and rendered scene from eye view.

then combined with the indirect illumination computed by a gathering step accumulating contribution of VPLs in RSM to every G-buffer pixels. This process can be illustrated by the pseudo-code 2.1.

Algorithm 2.1: Gathering step to shade every screen pixel with indirect lighting.

```
1  for every pixel in G-buffer p do
2      c ← color of p
3      L ← subset of pixels in RSM
4      for every l in L do
5          c += contribute(l, p)
6      end
7  end
```

Suppose number of VPLs chosen to lighten the scene is L, number of visible pixels stored in G-buffer is P, then the complexity of the gathering step is O(L * P). Current hardwares still cannot run efficiently if L is too high, thus it is unwise to use every VPL in RSM (which usually has a size of 512x512 or larger). At each G-buffer pixel p whose projected position on RSM is x, the original algorithm chooses a random subset of VPLs that are near x in RSM (figure 2.7). This is based on an assumption that if two projected points are close to each other on RSM, then their world space positions are likely to be close too. In the paper, L = 400 VPLs are randomly sampled around the pixel's projected RSM position.

One important thing to note is that this algorithm ignores visibility between VPLs and

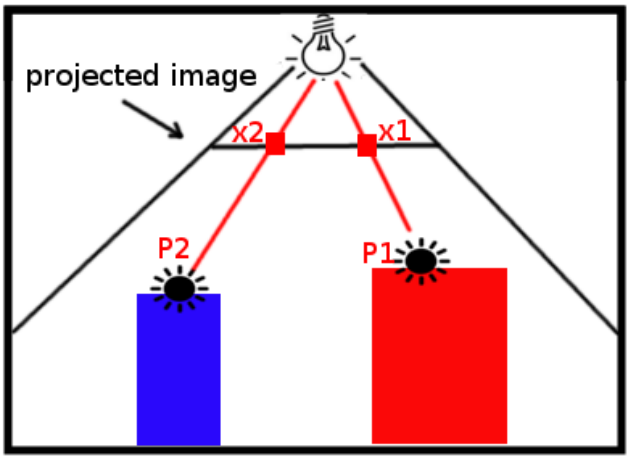


Figure 2.6: Two VPLs P1 and P2 have projected counterparts x1 and x2 on RSM.

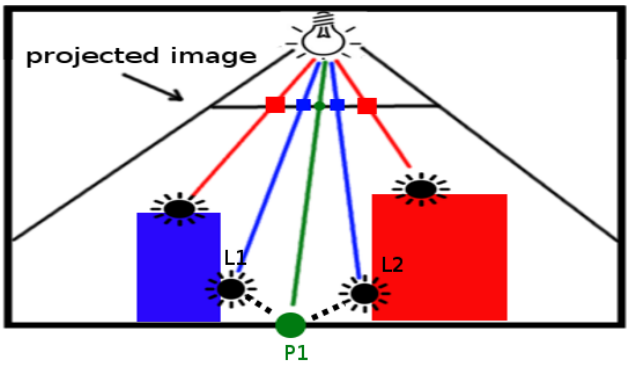


Figure 2.7: Two VPLs L1 and L2 are chosen to illuminate pixel P1 since their projected pixels (blue ones) are closest to P1's (green one) on RSM.

shaded pixels when calculating contributions. This is because visibility checking would require a complex ray tracing process which is still not feasible in real-time. Despite all of this, the indirect lighting computation is still too expensive to be done at every pixel in interactive frame rate. One additional optimization of the technique is rendering the indirect lighting in low-resolution buffer (typically 64x64), then upsampling it to full-resolution in final step.

This technique shares the same shortcomings as original Instant Radiosity's, caustics and glossy are not possible since number of usable VPLs is quite low. Furthermore, the sampling method for VPLs is too random, leading to inefficient memory accesses on GPUs.

2.2.3 Imperfect Shadow Maps

Ritschel et al. [RGK+08] improved Reflective Shadow Maps by Imperfect Shadow Maps. They determine visibility between a VPL and a eye view's pixel by using shadow map rendered from the VPL's view. However, rendering accurate shadow maps for hundreds of VPLs is impractical for interactive purpose. Thus, the technique only uses a point based approximate representation of the scene to create shadow maps (it's the reason that each map is called Imperfect Shadow Map or in short, ISM).

Initially, in preprocessing step, to create the point based representation, a subset of the scene's mesh triangles are randomly selected with probabilities proportional to the their areas. Then, points are placed on random positions in these chosen triangles. These points can be transformed during runtime to support dynamic scenes.

During ISMs creation, each map does not need to contain the whole scene. Instead, the representing points are split and randomly distributed to different ISMs. These points are rendered to each map by projecting parabolically the point primitives. The ISMs are low resolution and can be contained in one big texture.

Since points are randomly sent to ISMs, holes will appear in each map (figure 2.9). The paper uses pull-push approach presented in [MKC07] to fill the holes. The first step is pull phase in which the ISM is scaled down to several levels (similar to mipmapping). The second step is push phase which fills the holes at each level by interpolating values from coarser level.

The rest of the algorithm is similar to RSM, VPLs' contributions are computed at each pixel, visibility information from respective ISMs can be used to reject obscured VPLs. To support multiple bounces, the algorithm stores VPLs in each ISM, which can now be called Imperfect Reflective Shadow Map (IRSM). Important sampling can be used to sample second-bounce VPLs from these IRSMs. Then, the same steps can be applied to render second-bounce lighting.

There are several disadvantages of this technique. First, it requires preprocessing which may be a hindrance during design time, as any modification to the scene can lead to re-preprocessing of the point representation. Second, the visibility checking does not use precise geometry information of the scene, hence complex scenes would have very inaccurate indirect shadows. Lastly, even though an important sampling is used to provide better VPLs selection compare to original RSM algorithm, they are only sufficient to produce slight caustics and glossy effects.

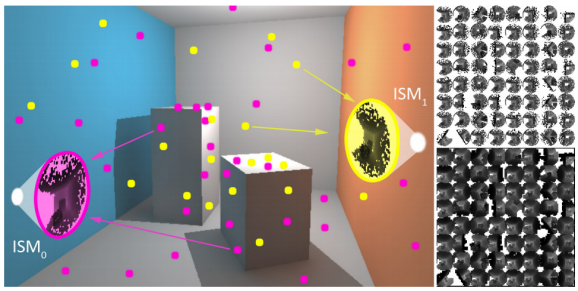


Figure 2.8: Left: two ISMs rendered from two white VPLs's respective view. Right top: many low-resolution ISMs with holes. Right bottom: corrected ISMs using pull-push. Courtesy of Ritschel et al. [RGK+08].

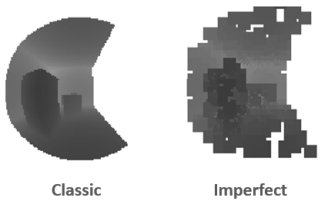


Figure 2.9: Classic shadow map vs ISM with holes. Courtesy of Ritschel et al. (SIGGRAPH 08 Talk).

2.2.4 Multi-resolution splatting for indirect illumination

Even with relative small number of VPLs (hundreds to thousands), gathering step at every screen pixel still has a high computational cost. The original RSM technique proposes an upsampling strategy which renders the indirect illumination to low resolution frame buffer first, then upscales it by interpolation. Places with high-frequency illumination such as corners are rendered at full resolution and combined with the upscaled image. This can leads to a very blurry image if the low resolution is too small. Nichols et al. [NW09] described a more sophisticate method in their paper, in which they used a multi-resolution approach. Instead of rendering the indirect illumination image at only two resolutions: low and high, they split it into several regions at various resolutions, then each region is treated as one pixel to be shaded, dramatically reduce number of pixels in the gathering step.

Starting by dividing the full-resolution frame buffer into a coarsest grid of regions (let say 16x16 in size). If there are regions where certain depth or normal discontinuities occur, each will be subdivided into 4x4 finer regions. Similarly, the finer regions may also be subdivided further, even to the finest level where each region covers exactly one pixel in the frame buffer (figure 2.10).

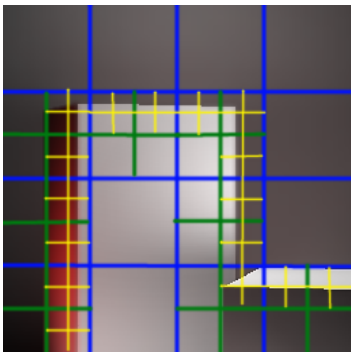


Figure 2.10: Multi-resolution regions: Coarsest level is separated by blue lines. Finer one uses green lines. Finest level is divided by yellow lines.

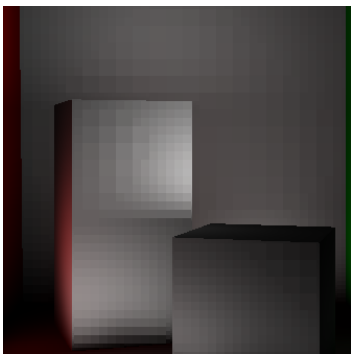


Figure 2.11: Final image without interpolation between disjoint regions.

One major problem is that the rendered frame buffer will be "blocky" as seen in figure 2.11, because of disjoint shaded regions. To solve that, the algorithm then performs interpolation progressively between regions within a same level and between coarser level and its next finer level. The interpolation starts from coarsest level, until the final image at finest level is produced.

The first implement of the technique uses Geometry Shader to subdivide the frame buffer and store the list of multi-resolution regions in a vertex buffer. Then for each VPL, the process calculates the contribution for each region and renders into respective layer in a multi-resolution buffer. For example, a region in 16x16 grid will be rendered to a pixel in 16x16 layer. The result is additive blended with previous VPLs' contributions. This is called splatting approach. The final step is interpolating between layers.

Later implement [NSW09] uses stencil-based approach. They pack all layers into single big texture (figure 2.12). The stencil buffer is utilised to mark those pixels in each layer that will cover valid regions. Finally, those marked pixels will be shaded by gathering VPLs' influence (as opposed to splatting method in previous implementation).

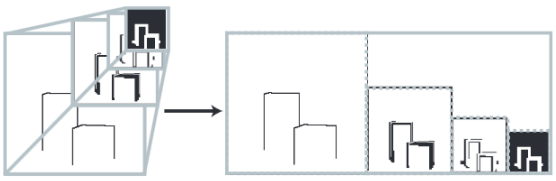


Figure 2.12: Left: multi-resolution images. Right: One image contains all layers. Courtesy of Nichols et al. [NSW09].

This multi-resolution technique is only suitable for diffuse scenes, since high-frequency lighting such as glossy and caustics requires more accurate frame buffer's splitting. Even worse, different VPLs will need different frame buffer's division because of high-frequency change of pixel's illumination. Thus, the splitting must be performed multiple times during rendering, once for each VPL. This would severely reduce performance of the algorithm. Visibility is also omitted in indirect lighting calculation.

2.3 Photon Mapping

2.3.1 Original Photon Mapping algorithm

Jensen [Jen96] invented classic Photon Mapping which shares some similarities with Instant Radiosity method. In first pass, particles (called photons) are emitted from the lights. Russian roulette is used to determine the survival of a photon at bouncing point, if reflection is determined, the photon will be reflected and continue to be traced. At every bouncing point, photon is stored in a Photon Map which records its incident direction and power. However, they aren't VPLs like those in Instance Radiosity techniques.

In second pass, equation 2.4 is approximated as:

$$L_r(x,\omega_0) = \int_{\Omega_x} f(x,\omega_0,\omega_i)\frac{\mathrm{d}^2\Phi_i}{\mathrm{d}A_i} \approx \sum_{i=1}^{N} f(x,\omega_0,\omega_i)\frac{\Delta\Phi(\omega_i)}{\Delta A} \tag{2.8}$$

Where A is a gathering area, $\Delta\Phi(\omega_i)$ is power of a photon within that area and its incident direction is $-\omega_i$. Intuitively, the algorithm uses power of N nearest photons to estimate the exitant radiance at x (figure 2.13). The more photons are used, the more accurate the formula 2.8 becomes. Based on an assumption that surface is flat around x, A could be calculated as an projected area of the gathering sphere with radius r:

$$A = \pi r^2$$

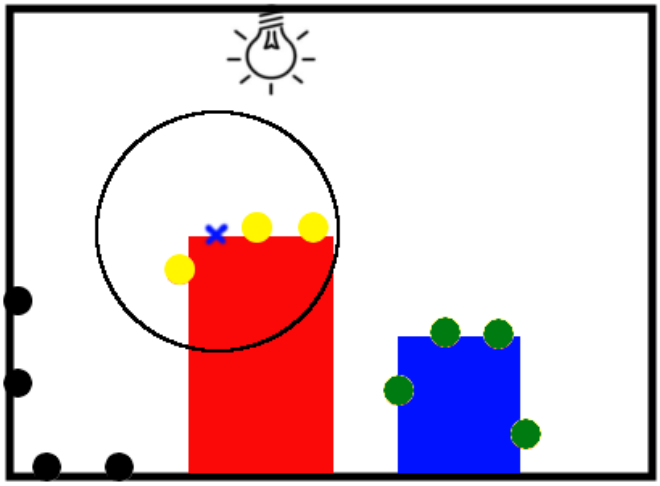


Figure 2.13: There are several photons in the scene, which are denoted by dot shapes. Only yellow ones are used to calculate radiance at x (blue cross). The gathering sphere is illustrated by a circle.

If number of photons are low, the estimation can be blurry at the edges. To reduce the artifacts, a filtering method such as Gaussian falloff function can be applied to weight the contribution of photons based on their distance to the surface point x.

Traditional implementation used kd-tree to locate nearest photons. To render caustics, photons bounced from specular surfaces are stored in a separate Photon Map. This map typically contains much higher photons than the global map storing regular photons.

2.3.2 Image-space Photon Mapping

2.3.2.1 Interactive Screen-Space Accurate Photon Tracing on GPUs

As Photon-Geometry intersection test is still computational expensive for interactive applications, especially in a scene where ther are a lot of triangles. Krüger et al. [KBW06] proposed a screen-space approximate intersection test by using depth buffer. First, the photon ray is projected onto screen-space by utilising hardware rasterized line. Afterwards, at every fragment inside the rasterized ray, the intersection is detected if the fragment's depth is nearly the same as the respective value in screen-space depth buffer (figure 2.14).

Since there is no guaranteed order of Fragment Shader executions on GPUs, it is not possible to terminate the process upon detecting the nearest intersection. Thus, the test has to be divided into two passes. First pass records all intersections between the ray and the depth buffer. Second pass will find the intersection closest to the ray's origin.

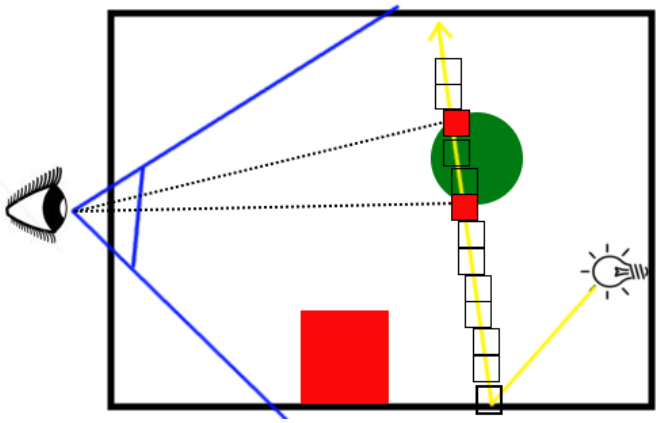


Figure 2.14: The reflected yellow ray is rasterized as line using GPU. There are two intersection points detected at red fragments. This is because the ray's red fragments have identical depth values compare with the sphere's ones stored in depth buffer.

Compare to linear search which can exit the test as soon as an intersection is found, this algorithm exploits the massively parallelism of GPUs to process several fragments together. Furthermore, a ray could consist of hundreds of fragments, hence it is impractical to perform linear search where the first intersection point could lie somewhere between the last fragments.

Certainly, occluded geometry will be ignored because depth buffer only stores surfaces visible to the eye. This could easily introduce false results and artifacts. The technique employs depth peeling to capture multiple depth layers. The paper suggests that 8 layers of depth are sufficient.

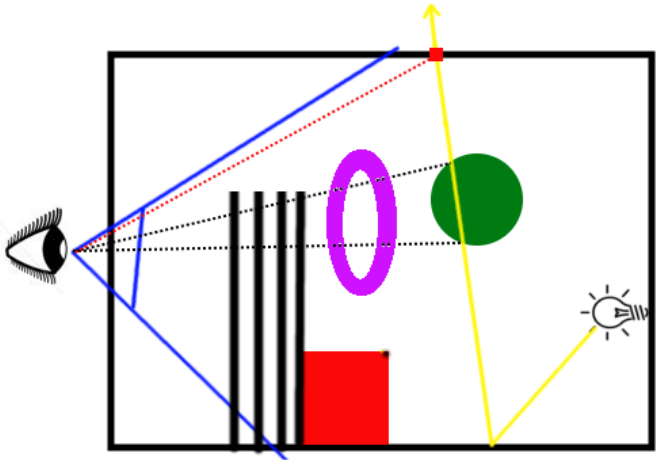


Figure 2.15: A scenario where even 8 depth layers are not able to store the green sphere. The testing then incorrectly detects red square at the top to be the first collision between the yellow ray and the geometry.

One of this technique major drawbacks is that in the scenes with high number of depth

layers, the suggested depth peeling method may fail to capture many objects even those they are close to the eye, see figure 2.15. Moreover, depth peeling requires rendering the scene multiple times, which could be a bottleneck in complex scenes having too many triangles.

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

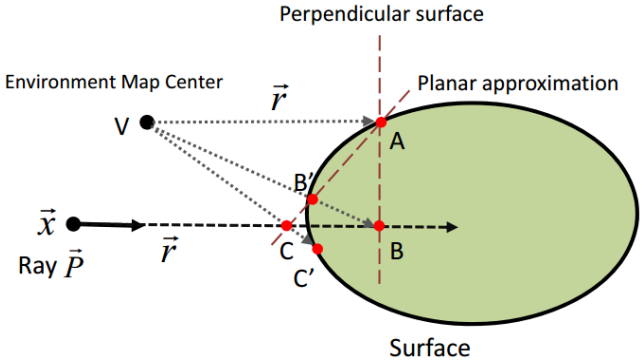


Figure 2.16: Computing intersection by iterations in image-space. The first guess is point A obtained from the direction of the ray. After one iteration, approximated intersection B' is obtained, then C' after next iteration, and so on. Courtesy of Yao et al. [YWC+10].

Yao et al. [YWC+10] presented a multi-image based Photon Mapping which applies Distance Impostors approach to approximate photon ray-geometry collisions [SKALP05]. As seen in figure 2.16, suppose we have a Cube Environment Map (EMC) centered at V storing information of the geometry visible to its center, and a ray $\vec{P} = \vec{x} + d\vec{r}$ which needs to be traced. First, point A on the ellipse's surface is obtained by shooting a ray from V along direction $\vec{r}$. Now, suppose a plane passing through A and perpendicular to $\vec{r}$ intersects $\vec{P}$ at B. By sampling the EMC along direction $\overrightarrow{VB}$, we have point B'. Next step is finding the intersection C between $\vec{P}$ and the plane passing through A, B', perpendicular to the plane containing A, $\vec{x}$ and V. Again, C' is obtained by sampling along the direction $\overrightarrow{VC}$. The process continues iteratively using point B' and C' to converge to the accurate intersection solution. In their experiments, 3 to 5 iterations are sufficient to retrieve accurate result.

If there was only one EMC, the technique would produce false positive solution similar to [KBW06], as an result of occluded geometry (figure 2.17). Thus, they employ multiple EMCs, each one covers a part of the scene. As finding a minimal set of EMCs covering an entire scene is an NP-hard set-covering problem, a heuristic method is applied to find a possible set of k EMCs providing maximal scene coverage. Denote this set by C, which is initialized to be empty. First, a group of 30-50 uniform randomized points are distributed around the scene, denoted by U. These points are candidate centers of EMCs. For every EMC v $\in$ U - C, the area of visible surfaces to v, but invisible to all EMCs in C is computed. v having maximum new covered area is chosen and added to C. The process continues iteratively until k EMCs

are found. To support dynamic scenes, the technique updates the set C progressively every 5 frames to choose a new v having largest new coverage and removes an existing EMC in C covering smallest area.

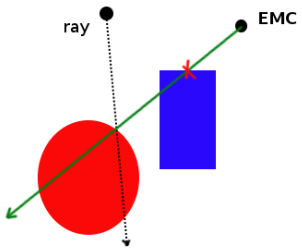


Figure 2.17: Wrong intersection point due to occluded geometry.

During the final step, in contrast with locating nearest photons, an opposite way is applied: each photon is splatted to the screen by drawing a 2D disk centered at its location. The energy is scattered to screen-space pixels falling within this disk, and additively accumulated with power from other photons.

In the implementation, to record the visible geometry at seen from an EMC, the scene is rendered into 6 faces of the cube map, similar to point light shadow mapping method. For reducing processing time, these EMCs are rendered at very low resolution (usually 32x32x6). The following process is implemented in order to calculate the area of surfaces visible to that EMC but invisible to C: Every rendered pixel will be test against all EMCs of the set C using standard shadow mapping technique. Only those in shadow region of C will be considered. These pixels' areas will be projected back to world space and summed up together, resulting in the total new covered area of that EMC with respect to C.

While the ray testing method using multiple EMCs of this algorithm is fast and independent of scene complexity, highly complex scenes would require dozen of EMCs to achieve good scene coverage. In addition, high polygon count has a big impact on the performance of EMC recording step which renders the scene repeatedly, 6 times for each EMC. As a result, this step is only performed progressively instead of updating every frame, thus restricting the applicable scenes to have only smooth movements. Furthermore, since EMCs only contain low resolution geometry, the ray testing may produce inaccurate results, these errors can be reduced by increasing number of photons with the expense of decreased performance. Finally, EMCs selection is a heuristic method, hence some parts of the scene could still be missed.

2.3.2.3 Hardware-accelerated global illumination by image-space photon mapping

Another image-space method proposed by McGuire in 2009 et al. [ML09]. This technique utilises both GPU and CPU to implement Photon Mapping. First, RSM is used to store first bouncing location of photons. The survived ones will be transferred back to CPU to perform accurate tracing as the classic algorithm. In the final step, similar to splatting method in [YWC+10], each photon is splatted to the screen by drawing a 3D ellipsoid volume bounding its location, screen-space pixels within this volume will receive its power.

While this algorithm can achieve highly accurate photon collisions comparable to CPU methods, its CPU tracing pass can be a bottleneck in complex scenes. In addition, space partitioning data structures such as kd-tree is precomputed to accelerate the tracing pass, which limits the dynamic scenes.

2.4 Other works

Cascaded Light Propagation Volumes (LPV) was introduced by Kaplanyan et al. [KD10] providing another approach to real-time GI. They inject VPLs to a coarse 3D grid, which contains the whole scene, to form a volume light at each grid's cell. The volume light's radiance distribution is approximated by low-order Spherical Harmonic (SH), so only low frequency lighting is supported. Radiance transfers are implemented by propagating iteratively within the grid. At each step, light is transferred from a cell to its 6 adjacent cells taking into account the occlusions by probability method. The result of all iterations represents the lighting distribution in the scene. The volume lights then contribute lighting to surface points within their cells. While this method can achieve plausible GI in high frame rate without pre-computation, it is only suitable for diffuse scenes. Moreover, indirect visibility between elements inside a cell is ignored, and artifacts occur due to coarse discrete representation of indirect lighting.

Recently, more and more attentions has been attracted to voxel-based GI. Crassin et al. [CNS+11] introduced their Voxel Cone Tracing method which is capable of glossy indirect lighting in real-time. First, the scene geometry is encoded in a texture representing a Sparse Voxel Octree. Irradiance from light sources is then injected into leaves nodes of the octree, then downsampled for the lower levels nodes. Finally, to shade a surface point, the method approximates ray tracing by using a cone representing a group of rays to be traced together, then steps along this cone, performing texture lookup on the Octree. Despite not being a completely accurate solution, the method achieves visual pleasing diffuse and glossy effects in real-time. However, Octree takes a large amount of time to construct, thus the majority

parts of the scene are static and their octree components are pre-computed. Furthermore, the tree consumes a lot of memory, even up to 1GB of texture in their implementation.

Prutkin et al. [PKD12] proposed a RSM clustering algorithm to reduce number of VPLs. The technique is based on k-mean clustering and able to reduce pixels in RSM to few hundreds/thousands of area lights, while keeping good temporal coherence, behaving well under sudden changes of lighting. Moreover, the technique has little overhead, thus can be included in the existing RSM-based rendering frameworks without having noticeable degraded performance. Nevertheless, their method only works on diffuse surfaces.

Recently, Lensing et al. [LB13] presented an efficient shading method which greatly reduces the number of surface points to be shaded during VPLs gathering step. Their motivation is similar to Multi-resolution method [NW09], however the work in [NW09] reduces number of surface points on screen-space while this technique does the same thing on object space. During pre-processing step, a sparse set of points called light probes are chosen randomly on a mesh's triangles, each point also has a list of vertices that it influences, similar to skeleton bones used in animation systems. These light probes approximately represent the mesh's geometry and indirect lighting is calculated on them instead of the G-buffer's pixels. During final step, a mesh's vertex is illuminated by weighted averaging its influencing light probes' illuminations. This technique appears to have better performance than the Multi-resolution method, especially when number of VPLs increases. However, lighting quality heavily depends on the preprocessing step and geometry complexity. Moreover, only diffuse surfaces are supported and indirect visibilities are not considered.

Chapter 3

The hybrid technique

3.1 Overview

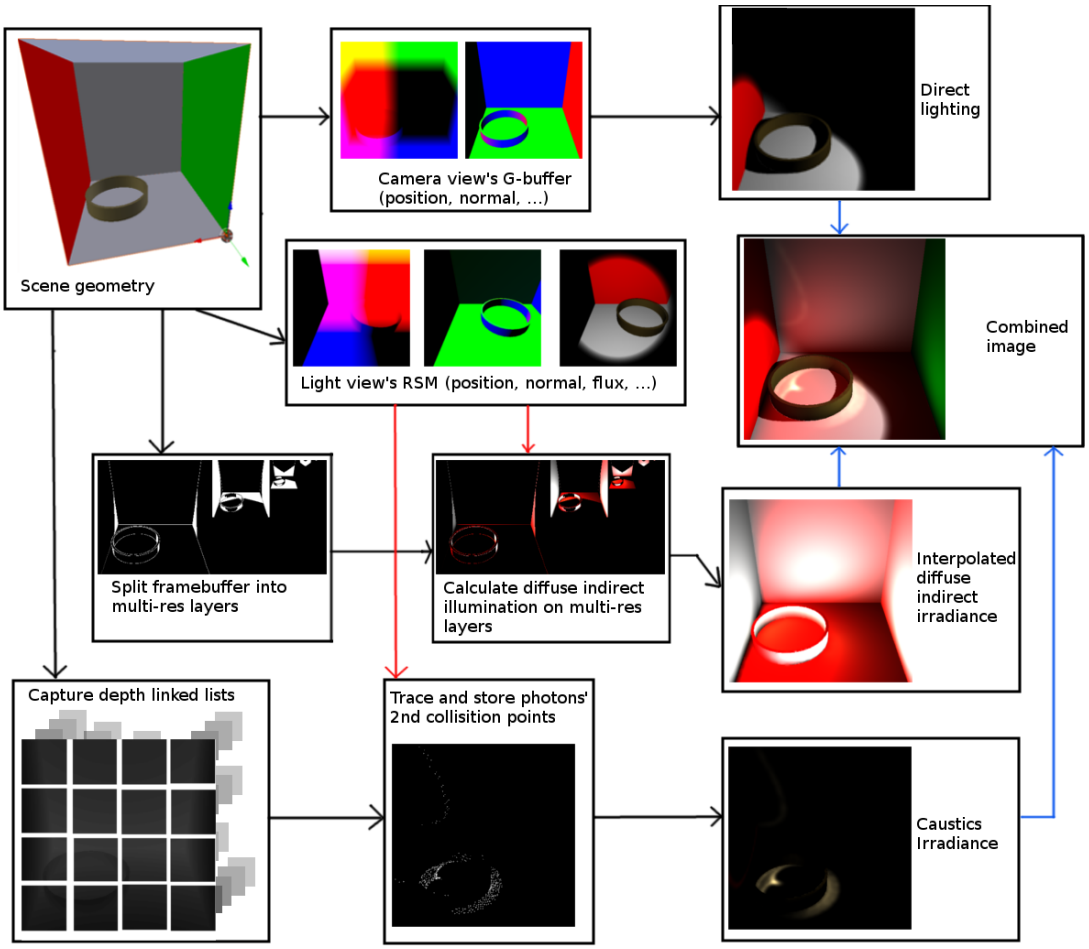


Figure 3.1: Overview of the method.

Figure 3.1 provides an overview of our method. We combine image-space Instant Radiosity with image-space Photon Mapping to render one-bounce diffuse indirect lighting and caustics. Image-space visibility check is used to trace photons to build the Caustics Photon Map. As discussed in the first chapter, it is possible to use photons to render diffuse inter-reflections, however, this requires a large amount of photons to avoid splotches. In this aspect, VPLs

are more superior. We use RSM to obtain both first-bounce VPLs and photons since they are similarly created by shooting particles from the light source. Observe from equation 2.6 that direct lighting, diffuse indirect reflections and caustics terms are independent, thus they can be computed separately. Our method can be divided into three main steps. First, direct illumination (including shadow) is calculated as normal non-GI rendering processes. Second, using an Instant Radiosity method, VPLs' contributions are gathered to the multi-resolution layers. These layer's results are then interpolated to produce diffuse indirect illumination. Third, a Photon Mapping technique traces those photons emitted from the light source, reflected on specular objects, arriving at diffuse surfaces and spreading their energy to nearby points. This results in caustic irradiance. Finally, the outcomes of the three steps are combined to produce the final image. The remainder of this chapter will give detailed explanations for the two latter steps. The technique mentions only one light source, but the concepts can easily be extended to multiple light sources.

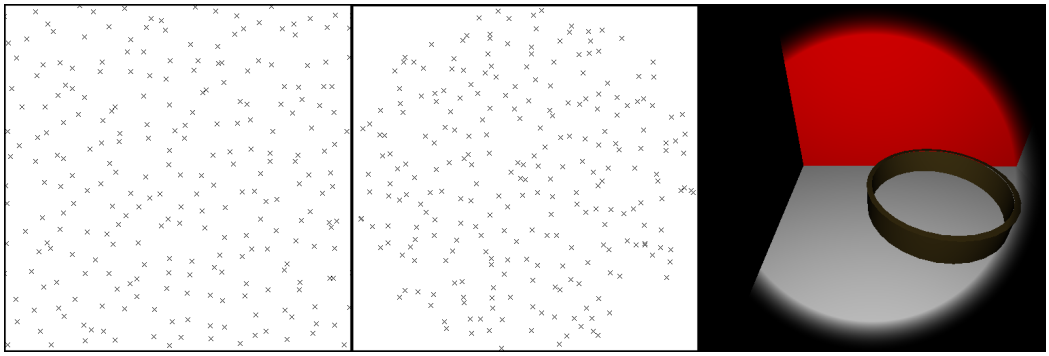
3.2 Diffuse inter-reflections using Instant Radiosity

3.2.1 VPLs sampling

VPLs are stored in RSM, which is rendered from light source's perspective similar to shadow maps. In spot light and directional light cases, RSM is a set of single sided textures. One for positions, one for normals of the surface points that the VPLs are created, and one for reflected flux. For point light sources, RSM can be 6 sides of a Cube Map textures or 2 sides of Paraboloid Mapped textures. To save memory, the positions and normals can be stored in half precision floating point textures.

Sampling a pixel in RSM results in position, normal, and reflected flux of a VPL. Certainly, using every VPL pixels for gathering step is impractical for real-time rendering, hence, only a subset of them are extracted by a random sampling method. For point light source, a grid sampling method can be used. Suppose we need to sample N x N number of VPLs on one side of Cube Map RSM. First, divide the texture into a grid of N x N cells, then for each cell, generate a pair of uniform variables (x,y) which define a point inside that cell. These points are used as sampling positions of VPLs (figure 3.2(a)). Directional light can use the same sampling method. A spot light source, however, has a little different pattern, since its light forms a cone shape, every VPL will reside in a circle within RSM (figure 3.2(c)). Therefore, we employs a circle sampling method for this type of light, which increases the density proportional to distance to the circle's center (figure 3.2(b)). Let ξ_1 & ξ_2 be uniform random variables between [0,1]. A sampling position (u,v) on RSM is obtained by the following formula:

$$u = 0.5 + 0.5\ \sqrt{\xi_1}\ \cos(2\pi\ \xi_2); \quad v = 0.5 + 0.5\ \sqrt{\xi_1}\ \sin(2\pi\ \xi_2)$$



(a) (b) (c)

Figure 3.2: From left to right: VPL's grid sampling, circle sampling and RSM from a spot light.

The sampling pattern is initialized at loading time and used for the rest of the application's life time. In order to optimize GPU's cache coherent during gathering step, VPLs are pre-sampled each frame and stored in an N x N texture before gathering. This avoids texture reading from incoherent locations in RSM.

3.2.2 Multi-resolution shading

For diffuse indirect lighting, we utilises the multi-resolution shading approach introduced in [NW09]. As our method computes diffuse and caustics in separate passes, this step has to be as fast as possible to leave a sufficient amount of time for the latter computation.

3.2.2.1 Min-max mipmap

Recall that this approach requires discontinuity information to correctly split the screen-space into regions. To do this, a min-max mipmap is used, which store min & max depth values at each level. The highest resolution mipmap level is obtained by reading linear depth value from G-buffer (the linear depth is distance to camera). The next level is generated by halving the resolution of previous step, calculating for each output the minimum and maximum values of 4 input pixels from higher level. Reading a pixel in the min-max mipmap gives the min & max depth of corresponding screen-space region. Depth discontinuity is detected if the region's min and max values differ by a value greater than a pre-defined threshold.

Surface normal variance within a region can also be defined in similar fashion. However, the min-max mipmap instead stores 3 sets of min and max values, one for each coordinate of the surface normal vector. If the difference between the min and max of any component exceeds a similar threshold, then normal discontinuity is determined.

3.2.2.2 Multi-resolution shading regions

Shading every pixels in screen-space with hundreds to thousands of VPLs is a very expensive process. Given the low-frequency nature of diffuse reflections, lighting varies quite slowly between adjacent pixels on a smooth surface. Therefore, it is more efficient to shade these smooth surfaces at low resolution, then interpolate the results for the higher resolution pixels.

The multi-resolution approach divides screen-space into several regions at various resolutions. For instance, the resolution of screen-space frame buffer is 512x512, then every finest level region contains exactly one pixel, while the 4th coarser level ones enclose 32x32 pixels each. One possible splitting strategy is a bottom-up process which initializes at the coarsest level, then at each refinement step, every region in which there exist depth or normal discontinuities is subdivided into 4 smaller regions. One example of the completed screen-space clustering can be seen in figure 2.10 on page 13. The better strategy [NSW09] stems from the fact that each refinement step does not actually depend on prior step, thus can be done in parallel. In fact, a region is valid if it has no discontinuity but the coarser one containing it does. This improved strategy is describes by the pseudo-code 3.1.

Algorithm 3.1: Multi-resolution regions splitting.

```
1  R ← list of possible regions
2  for every region r in R do
3      i ← level of r
4      if discontinuity(r, i) is true then
5          continue // region is invalid
6      end
7      if discontinuity(r, i + 1) is false then
8          continue // coarser region is valid
9      end
10     setValid(r) // mark as valid region
11 end
```

Stencil feature of graphics hardware provides an efficient way to implement this shading technique. First, several layers at different resolutions are provided, where a pixel represents a region at respective resolution. The layers are flattened into a big 2D image (figure 3.3(a)). To do the splitting process, a full-screen quad is rendered onto this image, in fragment shader, the pixel corresponds to the valid region sets a flag in the stencil buffer.

In the gathering step, we draws a full-screen quad onto the flattened image again, only pixels that pass the stencil test will be shaded by the VPLs (figure 3.3(b)). The irradiance at a pixel/region due to contribution from a VPL is calculated as follows:

$$E_x = \Phi_l \frac{max(0, |n_x \cdot \hat{v_{l,x}}|)\ max(0, |n_l \cdot \hat{v_{l,x}}|)}{|v_{l,x}|^2 + \epsilon}$$

Where Φ_l, n_l are reflected flux and surface normal of the VPL, respectively. n_x is the normal of the region, which is obtained from G-buffer. $v_{l,x}$ is vector from the region's position to the VPL's. ϵ is small value added to avoid division by zero.

(a) (b)

Figure 3.3: From left to right: Stencil marked multi-resolution regions and their shaded result. The white pixels on the left image correspond to the valid regions.

Finally, the shaded layers are interpolated by a bottom-up procedure beginning from coarsest level. At each step, a pixel containing illumination is bi-linear interpolated with its 8 nearest neighbours of the same resolution, which are either originally rendered in the gathering step or upsampled from the lower resolution (figure 3.4). The pseudo-code 3.2 further explains this procedure.

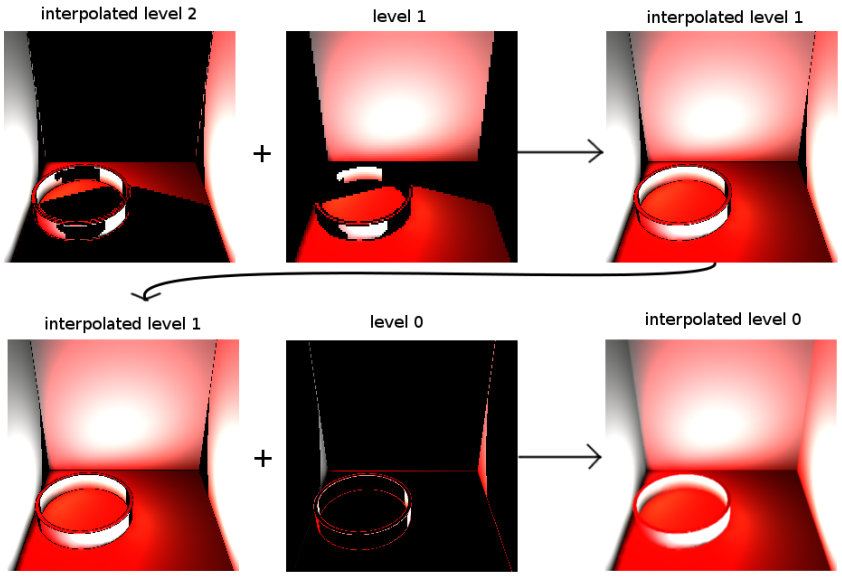


Figure 3.4: Multi-resolution interpolation process.

Algorithm 3.2: Multi-resolution interpolation.

```
1  for every level l do
2      for every pixel p in l do
           // 3x3 input pixels
3          interpolated[9] ← interpolated from previous step
4          rendered[9] ← rendered in gathering step
5          weight[9] ← interpolating weight
6          p ← 0
7          totalWeight ← 0
8          for i = 0 to 8 do
9              if hasData(interpolated[i]) or hasData(rendered[i]) then
10                 totalWeight += weight[i]
11                 p += weight[i] * (interpolated[i] + rendered[i])
12             end
13         end
14         p /= totalWeight
15     end
16 end
```

3.3 Caustics using Photon Mapping

3.3.1 Photons' initial bounces

3.3.1.1 Photons sampling

The photons' initial intersections are also obtained from RSM. In this step, instead of viewing pixels as VPLs, they are considered photons' first hit points. Since only caustic photons are concerned, we need to identify which RSM pixels are on specular surfaces. To do this, during RSM rendering step, a special mask texture is generated, in which a pixel is set to one if its corresponding RSM's pixel has specular material, otherwise it is set to zero. This gives a way to filter out caustic photons from RSM. The number of photons can easily be retrieved by inspecting the percentage of marked pixels, which is stored in the lowest level of the mask texture's mipmap(see figure 3.5 for more details). This mipmap is generated by the graphics hardware automatically. Since the number of photons in RSM could become intractable, one additional step is finding a subset of them to sample. Using an uniform sampling method

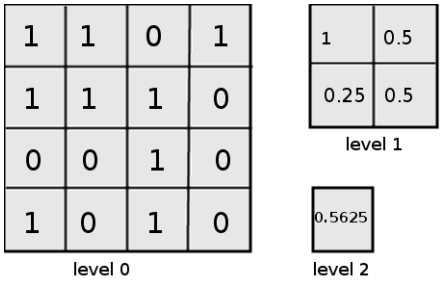
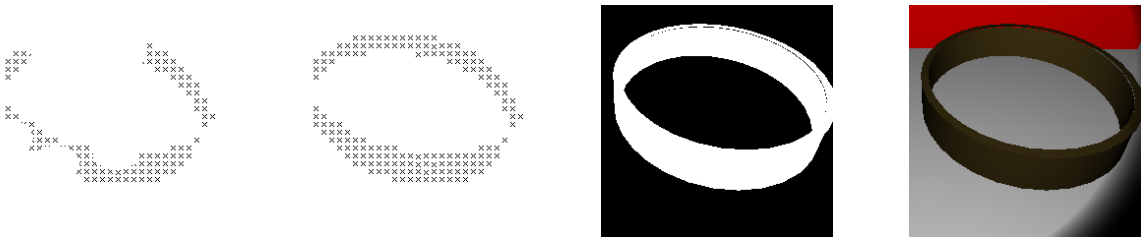


Figure 3.5: The mipmap of a 4x4 mask texture corresponding to 4x4 RSM. Interestingly, hardware generated mipmap process computes a lower level pixel by averaging its 4 higher level pixels. Thus, lowest mipmap level will actually contain the percentage of specular pixels in highest resolution RSM, which is 0.5625, so the number of specular pixels is 0.5625 * 16 = 9.



(a) **(b)** **(c)** **(d)**

Figure 3.6: From left to right: Uniform sampling positions, our sampling method's positions, mask texture, and RSM. The mask texture is represented in color mode, where black and white correspond to zero and one respectively.

could result in sampling points covering the objects' shapes unevenly (the ring object in figure 3.6(a)). To address this, we propose a simple sampling method using down-sampled mipmap level of the mask texture. Because GPU generates mipmap using box-filtering, a generated pixel's color is one only if its 4 higher level pixels' are also one (figure 3.5). Thus, the number of one-marked pixels in a level (denoted by $s(l)$) can be guessed by the following formula:

$$s(l) \leq s_{max}(l) = \frac{s(l-1)}{4}$$

Suppose the number of marked pixels in highest resolution of RSM is M, then for a mipmap level l, $s_{max}(l) = \frac{M}{4^l}$. Assume that we would like to sample no more than N photons (N $<$ M), the task becomes as simple as finding l such that $s_{max}(l) \leq N$, then sampling those photons from RSM using one-marked positions in the l mipmap level(figure 3.6(b)).

3.3.1.2 Photons' reflections

Upon retrieving the initial hit points, as traditional Photon Mapping, the algorithm will decide which type of reflection or absorption a photon will have. Let the diffuse and specular reflection coefficients of a surface point be (d_r, d_g, d_b) and (s_r, s_g, s_b) respectively. A probability for reflection can be computed as:

$$P_r = max(d_r + s_r, d_g + s_g, d_b + s_b)$$

The probability of diffuse reflection is:

$$P_d = \frac{d_r + d_g + d_b}{d_r + d_g + d_b + s_r + s_g + s_b} P_r$$

The probability of specular reflection is:

$$P_s = P_r - P_d$$

With these probabilities and an uniform random variable $\xi \in [0, 1]$, the type of reflection or absorption will be chosen as follows:

$\xi \in [0, P_d] \Longrightarrow$ diffuse reflection
$\xi \in (P_d, P_d + P_s] \Longrightarrow$ specular reflection
$\xi \in (P_d + P_s, 1] \Longrightarrow$ absorption

Only specular reflected photons will be traced in our technique. Once a photon is specular reflected, its power needs to be adjusted to account for the probability of survival:

$$\Phi_{ref,r} = \Phi_{inc,r} s_r / P_s$$
$$\Phi_{ref,g} = \Phi_{inc,g} s_g / P_s$$
$$\Phi_{ref,b} = \Phi_{inc,b} s_b / P_s$$

Where Φ_{ref} is the reflected power, and Φ_{inc} is the incident power.

Next, the photon will be reflected in random direction according to its collided surface's BRDF. For Phong material, the reflection's directions are distributed close to the mirror reflection vector. Let (θ, ϕ) be the spherical coordinates of the random direction (figure 3.7), α be the Phong exponent of the material, and ξ_1 & ξ_2 be uniform randomized values within [0,1]. (θ, ϕ) can be calculated as:

$$\theta = \arccos(\xi_1^{\frac{1}{\alpha + 1}})$$
$$\phi = 2\pi\xi_2$$

This gives Cartesian reflection vector $\vec{r}$=$(\sin\theta\ \cos\phi, \sin\theta\ \sin\phi, \cos\theta)$ in mirror reflection space. To transform it back to world space, we need to define the mirror reflection space. This space has mirror reflection vector $\vec{r}_{mirror}$ as its z-axis, the remaining two axes can be

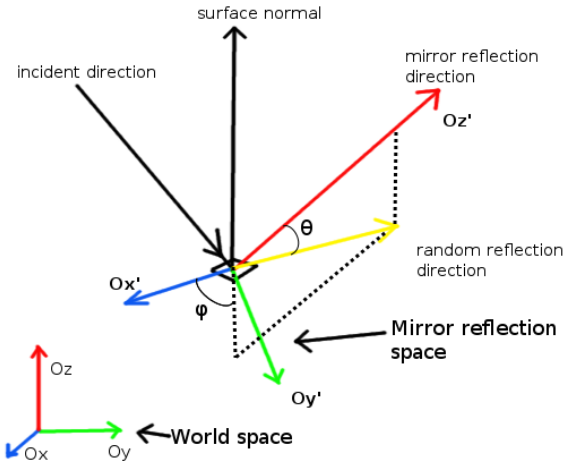


Figure 3.7: Random reflection direction (yellow) can be expressed in (θ, ϕ) which is a spherical coordinates in mirror reflection space.

obtained by the pseudo-code 3.3.

Algorithm 3.3: Mirror reflection space's axes calculation.

$\overrightarrow{Oz'} \leftarrow \vec{r}_{mirror}$ `// reflection space's z-axis`
$\overrightarrow{Ox'}, \overrightarrow{Oy'}$ `// reflection space's x and y-axis`
$\overrightarrow{Ox}, \overrightarrow{Oy}, \overrightarrow{Oz}$ `// world space's x, y and z-axis`
if $\overrightarrow{Oz'}$ *is parallel to* $\overrightarrow{Oy}$ **then**
    $\overrightarrow{Ox'} = \overrightarrow{Ox}$
    $\overrightarrow{Oy'} = -\overrightarrow{Oz}$
end
else
    $\overrightarrow{Ox'} = \text{normalize}(\text{cross}(\overrightarrow{Oy}, \overrightarrow{Oz'}))$
    $\overrightarrow{Oy'} = \text{cross}(\overrightarrow{Oz'}, \overrightarrow{Ox'})$
end

With $\overrightarrow{Ox'}, \overrightarrow{Oy'}, \overrightarrow{Oz'}$, reflection vector can be transformed back to world space by the following formula:

$$\vec{r}_{world} = \begin{pmatrix} \overrightarrow{Ox'}_x & \overrightarrow{Oy'}_x & \overrightarrow{Oz'}_x \\ \overrightarrow{Ox'}_y & \overrightarrow{Oy'}_y & \overrightarrow{Oz'}_y \\ \overrightarrow{Ox'}_z & \overrightarrow{Oy'}_z & \overrightarrow{Oz'}_z \end{pmatrix} \begin{pmatrix} \vec{r}_x \\ \vec{r}_y \\ \vec{r}_z \end{pmatrix}$$

3.3.2 Photon tracing on screen-space

3.3.2.1 Per-pixel linked list

To approximately represent the scene's geometry, the technique utilises a per-pixel linked list construction method on Direct3D 11/OpenGL 4-capable GPUs [YHGT10], where there are multiple fragments in a single pixel, as show in figure 3.8. Each fragment stores depth and surface normal data of the respective surface point. As a way to save storage space, normal vector's data will be stored in spherical coordinates format (2 half precision floats instead of 3 floats).

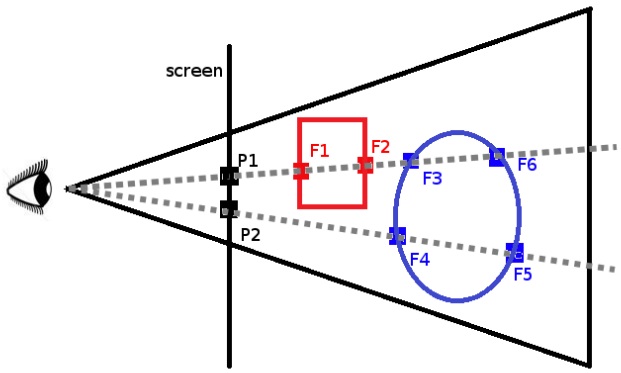


Figure 3.8: Pixel P1 has a linked list of 4 fragments: F1 & F2 from red object, F3 & F6 from blue object. Similarly, Pixel P2 has a list of 2 fragments: F4 & F5 both from blue object.

To construct these linked lists, we first pre-allocate a global buffer on the GPU. This buffer will be used as the storage for the lists' nodes' data. The node data is defined as follows:

```
struct FragmentNode {
        float depth;
        int normal;// 16 bit floats * 2
        int next; //pointer to the next node.
        //a value of -1 means there is no next node
};
```

The scene is then drawn to an off-screen frame buffer as usual with depth testing disabled. However, instead of color, this buffer stores at each pixel a pointer pointing to the head node of the respective linked list, which is initialized to -1. In fragment shader, the generated fragment's data will be inserted to the linked list of the corresponding pixel location (figure 3.9), this procedure is described by the pseudo-code 3.4. The min and max depth values of every fragment in each linked list will also be stored. Note that we use perspective projection to capture the scene, so only geometry inside the view frustum is available in the linked list. Alternative way could be using orthogonal projection with a frustum covering the entire

Algorithm 3.4: Per pixel linked list insertion in fragment shader.

Input: p // fragment's position in screen-space
Input: d // fragment's depth
Input: n // fragment's surface normal
Data: $buffer$ // global buffer
Data: $head$ // head pointer buffer
Data: $totalFrags$ // global counter couting total number of fragments

```
1 newFragNode // node data
2 newFragNode.depth = d
3 newFragNode.normal = n
4 newFragNode.next = head[p] // points to current head node
5 buffer[totalFrags] = newFragNode // store data in global buffer
6 head[p] = totalFrags // new fragment node becomes head node
7 totalFrags = totalFrags + 1 // increase global counter
```

scene.

3.3.2.2 Line rasterization

Similar to [KBW06], to trace the reflected photon rays and find their second collision points, we rendered them as line primitives. The difference is that [KBW06] uses layered depth images to calculate the intersections, while our technique uses per-pixel linked lists. At each fragment generated by line rasterization, the fragment shader will search linearly in the respective linked list, comparing the depth of generated fragment with the depth of every fragment in the list. If both values differ less than a specific threshold, an intersection point between the photon ray and scene's geometry will be stored (see figure 2.14 on page 16). This procedure is illustrated by the pseudo-code 3.5. The min & max depth values are used in fragment shader for skipping empty space. Clearly, per-pixel fragment lists is faster to be generated because they need only one rendering pass compare to multiple passes of multiple depth layers. Furthermore, the layered depth images will miss some parts of the scene that are further than the maximum depth layer (figure 2.15 & 4.2(b) on pages 16 & 40), while fragment lists will not. That is because the depth layer only contains the fragments closest to the eye and discards the rest, on the other hand, our method's linked lists keep all the rasterized fragments.

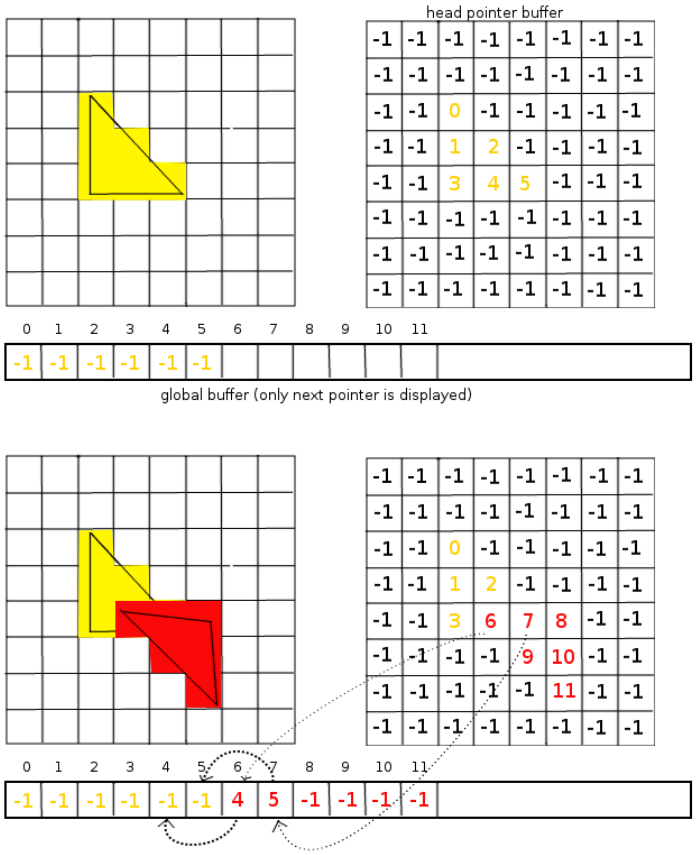


Figure 3.9: Linked list construction. Yellow object is rasterized first, the red object is second. The process can capture more than one depth value at overlapped locations between 2 objects. Note: only "next" pointers are shown in the global buffer, other data's members such as depth, normal are omitted.

We store first-bounce photons' positions and their reflected directions in a vertex buffer. Geometry shader reads this information and generates line primitives, the interpolated fragments are passed to fragment shader to perform the intersection test. Similar to [KBW06], each line has a dedicated row in an off-screen buffer (a texture) to store the list of its intersections with the scene's geometry, see figure 3.10. Since textures have limited rows, not all photons can be traced in one single pass. Hence, the photons tracing will be repeated until there are no remaining photons left. To eliminate the need for synchronization between CPU & GPU to let CPU know how many untraced photons left, indirect rendering feature of modern GPUs, which stores rendering parameters in a GPU buffer, can be utilised. The first intersection on the row will be the second hit point of the photon with the scene's geometry.

Because a long ray would generate dozen of fragments, which could become a fill-limited bottleneck in the system, we instead render the rays at low resolution (commonly 128x128), the accuracy of the low resolution rays is sufficient for our needs.

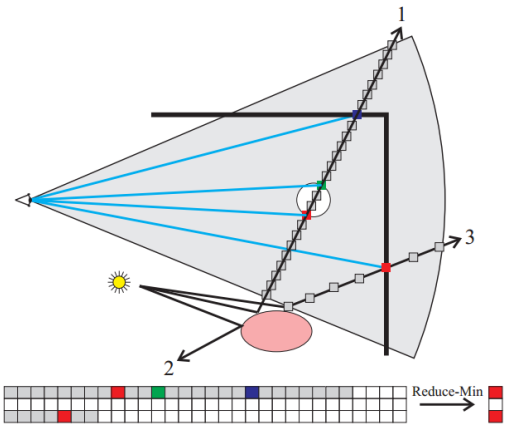


Figure 3.10: The pink object at the bottom reflects the light from the light source and causes three caustics rays. These are processed by the geometry shader, which computes the reflected rays and generates line primitives. The rendering of these lines generates the image shown at the bottom. Red, green and blue pixels indicate hits with an object, and grey cells indicate fragments that have been skipped in the fragment shader. White cells are fragments having never been rendered. Courtesy of Krüger et al. [KBW06].

As seen in figure 3.11, self-intersection error may occur because of discrete nature of pixels. To overcome this, the intersection will be discarded if the angle between the normal of the surface point and the ray's direction is less than 90 degrees, which means the ray is pointing away from the surface point.

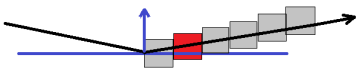


Figure 3.11: Red fragment is wrongly detected as intersection between the reflected ray and the blue object due to pixelating error.

Figure 3.12 shows that the self-intersection problem can still occur if the object is very thin. Fortunately, this error is acceptable since the angle between the surface normal and photon's ray is very close to 90 degrees, thus having close to zero cosine, therefore the photon's power transferred to the surface point is negligible.

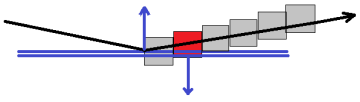


Figure 3.12: Inaccurate intersection can still occur on a very thin object which has multiple surfaces close to each other.

Algorithm 3.5: Intersection test using depth linked list in fragment shader.

```
Input  : p  // fragment's position in screen-space
Input  : d  // fragment's depth
Output : hit  // intersection occurs at this fragment?
Data: buffer  // global buffer
Data: head  // head pointer buffer
1  idx = head[p]
2  hit = false
3  while idx not equals -1 and hit equals false do
4  |   fragDepth = buffer[idx].depth
5  |   if |fragDepth − d| < ε then
6  |   |   hit = true
7  |   end
8  |   else
9  |   |   idx = buffer[idx].next;
10 |   end
11 end
```

3.3.2.3 Photon splatting

For the radiance estimation step, we adopt a splatting method similar to [LP03] and [YWC+10]. Each photon is splatted onto a disk centered at the surface's hit point and perpendicular to its normal, see figure 3.13(b). The contribution is added to every pixel covered by the projection of the disk on the screen and that resides on the same surface. The disk has a radius h which is defined as $h = C\sqrt{\frac{A}{N}}$, where A is total area of the scene, N is total number of emitted photons, and C is a variable that controls the bandwidth. C is defined as $C = C_0/\sqrt{p}$, where C_0 is a modifiable parameter and $p = cos\theta/d^2$ where d is traveled distance of photon, and θ is incident angle. With these density estimation formulas, the further the photon travels, the larger the disk becomes.

The photon's splatting disk is rendered by generating a quad in geometry shader. For each rasterized fragment of the quad, fragment shader reads world space position from G-buffer and computes the contribution of the photon to the irradiance of that fragment as follows:

$$\Delta E = \frac{\Phi\ \kappa(d)}{\pi h^2}$$

Where Φ is incident power of the photon , d is distance between the fragment and the disk's

center. And κ is a weighting function, which could be defined as:

$$\kappa(d) = max(0, 1 - \left|\frac{d}{h}\right|)$$

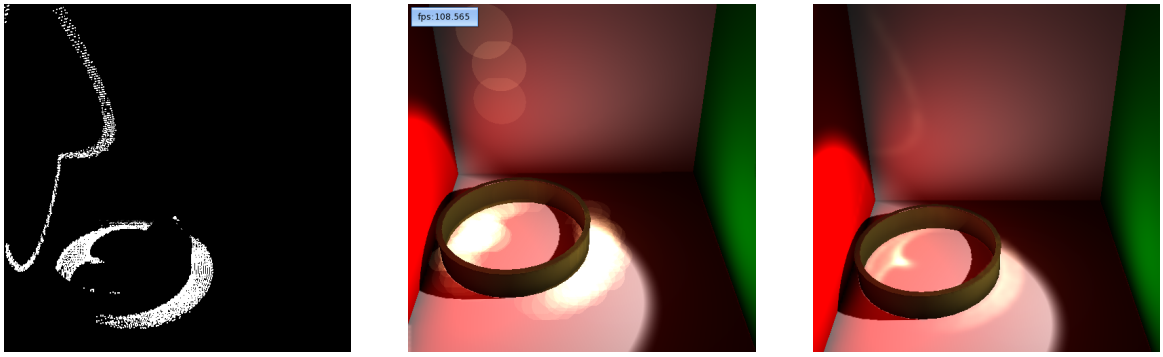
or using Gaussian falloff function which gives a better result:

$$\kappa(d) = \mathrm{e}^{-\frac{(\frac{d}{h})^2}{2\delta}}$$

Where δ is a user-defined value controlling the bell shape. For faster computation, the weighting values can be pre-computed and stored in a 1D texture, then in fragment shader, $\kappa(d)$ is obtained by looking up this texture:

$$\kappa(d) = tetureFetch(\frac{d}{h})$$

This contribution is additively blended with the frame buffer which also contains contributions from other photons.



(a) **(b)** **(c)**

Figure 3.13: From left to right: photon incident positions, splatting result without weighting and with Gaussian weighting.

3.3.2.4 Irradiance sub-sampling

Unfortunately, similar to the tracing pass, the splatting method could become fill-limited, because of excessive overdraws of the photon disks. Therefore, we perform a final optimization similar to bilateral upsampling method [SGNS07]. The disks are rendered first onto a low resolution frame buffer, with its dimension is a quarter of the full resolution. The low resolution image is then upsampled using position, normal, and depth weights. The upsampling method is done as follows: First, two versions of G-buffer are generated, one in full resolution and one in low resolution. Next, a 3x3 box of the low resolution irradiance buffer, which containing the high resolution pixel, is located. Let c_i, x_i, n_i, d_i be the irradiance, screen-space position, world space normal, and depth of the full resolution pixel, c'_j, x'_j, n'_j, d'_j be the respective values of a low resolution pixel within that 3x3 box. Irradiance at full resolution is calculated by this formula:

$$c_i = \frac{1}{w} \sum_j c'_j \, w_j$$

(a) (b)

Figure 3.14: From left to right: Low resolution caustics irradiance and its upsampled result.

Where $w = \sum_j w_j$ and $w_j = w_{j,x}\ w_{j,n}\ w_{j,d}$.
$w_{j,x}$ is bi-linear weight.
$w_{j,n}$ is normal weight, which is $(max(n_i \cdot n'_j, 0))^k$, k is a user-defined value.
$w_{j,d}$ is depth weight defined as $\dfrac{1}{1 + \left| d_i - d'_j \right|}$.

While achieving good performance gain, this optimization produces blurred results, especially in the areas of highly detailed surfaces.

Chapter 4

Results

Our experiments are done on a PC with Intel Core i5 650@3.20GHz, NVIDIA Geforce GTX 460, and 4GB of RAM. The implementation uses DirectX 11 with Shader Model 5. RSM is rendered at 512x512 resolution, while per-pixel linked lists are captured at the same resolution as the photon ray tracing does (which is either 128x128 or 256x256). Total VPLs used for diffuse indirect illumination is 256. Random numbers for Photon Mapping are pre-generated and store in a texture, so every frame will use the same random numbers to maintain temporal coherent. 20MB memory on the GPU is also pre-allocated for storing per-pixel linked lists. All test scenes are rendered using one spot light and evaluated with various settings, such as photon rays' resolution, number of photons and final image resolution. Reference images, produced by the off-line method called Stochastic Progressive Photon Mapping (SPPM) [HJ09], are also provided for comparison.

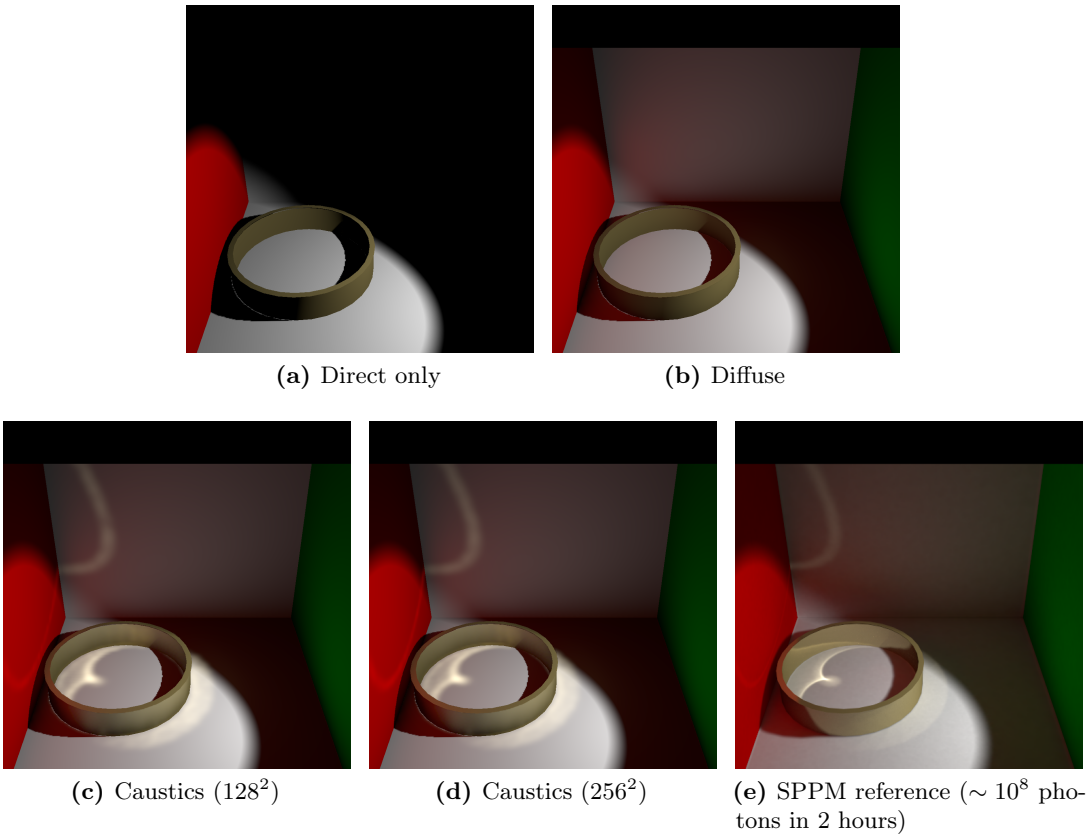
4.1 Caustic Ring scenes

4.1.1 Simple scene

The scene consist of a ring having specular material within a room. The total triangles count is 202. Figure 4.1 shows the scene rendered with direct light only (a), with diffuse indirect lighting (b), with caustics (c) & (d). Figure 4.1(c) renders the photon rays at 128x128 resolution, while Figure 4.1(d) does it at 256x256. We can clearly see that there is not much difference between the two images. The performance measures are presented in table 4.1.

4.1.2 Occluded ring

This is a variant of the first scene, in which there are 4 walls hiding the ring from the camera (figure 4.2). There are two platforms, the higher one is where the ring resides. The total triangles is 214. Figure 4.2(b) shows the occurrence of light leaking if using 4 depth layers to



(a) Direct only **(b)** Diffuse

(c) Caustics (128^2) **(d)** Caustics (256^2) **(e)** SPPM reference ($\sim 10^8$ photons in 2 hours)

Figure 4.1: Simple ring scene.

Table 4.1: Statistics of the simple ring scene.

Resolution	Caustic photons	Caustic ray resolution	FPS	Diffuse illumination time(ms)		Caustics illumination time(ms)		Frame time(ms)
				Multi-resolution splitting	Multi-resolution gathering	Photon tracing	Photon splatting	
512^2	17021	128^2	90	0.087	1	7	0.477	11
512^2	17021	256^2	54	0.088	1	14	0.270	18
1024^2	17021	128^2	65	0.318	2	7	0.721	15
1024^2	17021	256^2	43	0.318	2	14	1	22

Table 4.2: Statistics of the occluded ring scene.

Resolution	Caustic photons	Caustic ray resolution	FPS	Diffuse illumination time(ms)		Caustics illumination time(ms)		Frame time(ms)
				Multi-resolution splitting	Multi-resolution gathering	Photon tracing	Photon splatting	
512^2	17021	128^2	46	0.089	1	17	0.325	21
512^2	17021	256^2	26	0.089	1	33	0.346	37
1024^2	17021	128^2	39	0.320	2	17	0.325	25
1024^2	17021	256^2	25	0.320	2	33	0.346	38

Table 4.3: Statistics of the bunnies scene.

Resolution	Caustic photons	Caustic ray resolution	FPS	Diffuse illumination time(ms)		Caustics illumination time(ms)		Frame time(ms)
				Multi-resolution splitting	Multi-resolution gathering	Photon tracing	Photon splatting	
512^2	17021	128^2	57	0.090	1	11	0.500	17
512^2	17021	256^2	33	0.090	1	23	0.872	30
1024^2	17021	128^2	43	0.322	3	11	1	25
1024^2	17021	256^2	28	0.322	3	23	1	35

trace the photons since the ring's platform cannot be captured by the depth layers. Figure 4.2(c) & (d) demonstrates the correct result when using per-pixel linked lists. See table 4.2 for evaluated results.

4.1.3 Three bunnies and a ring

The scene has 3 bunnies and a caustic ring (figure 4.3). Total triangles count is 208619. The purpose of this scene is testing the accuracy and performance of the Photon Mapping step when number of triangles is high. See table 4.3 for evaluated results.

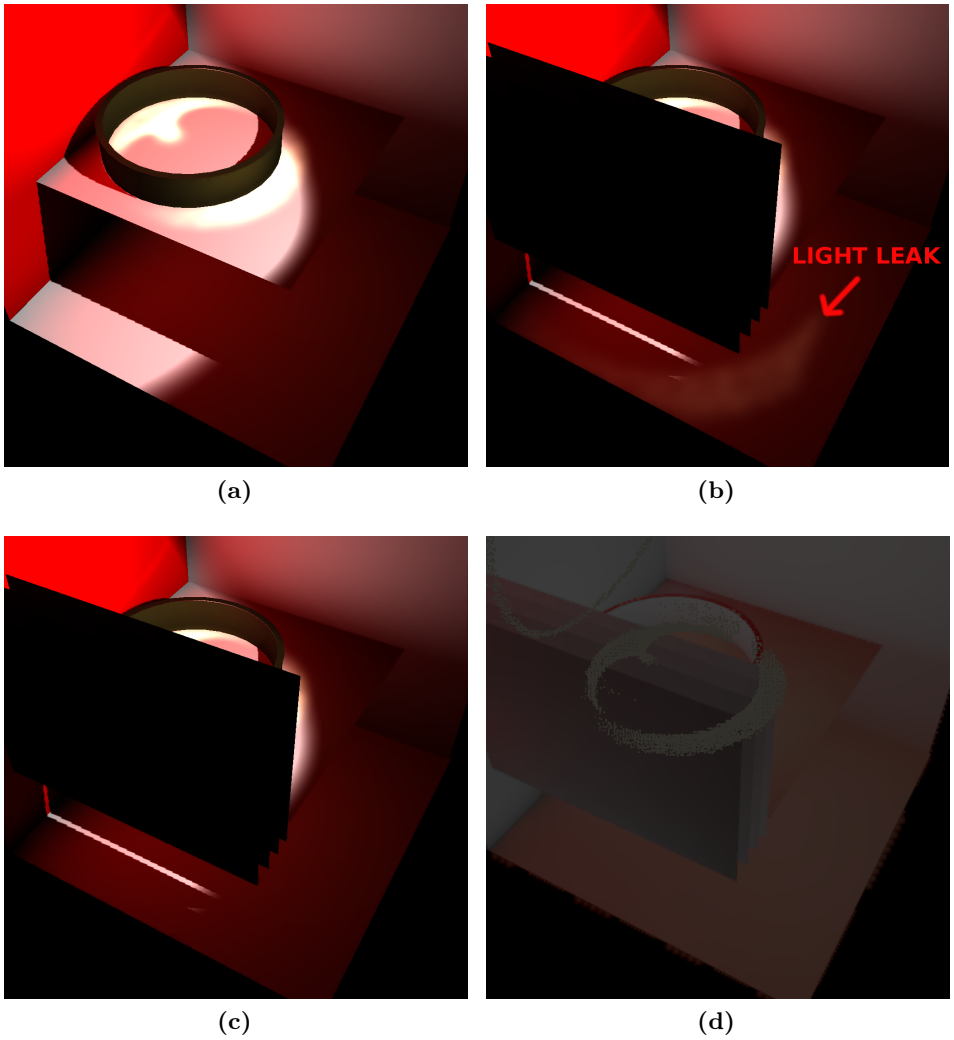
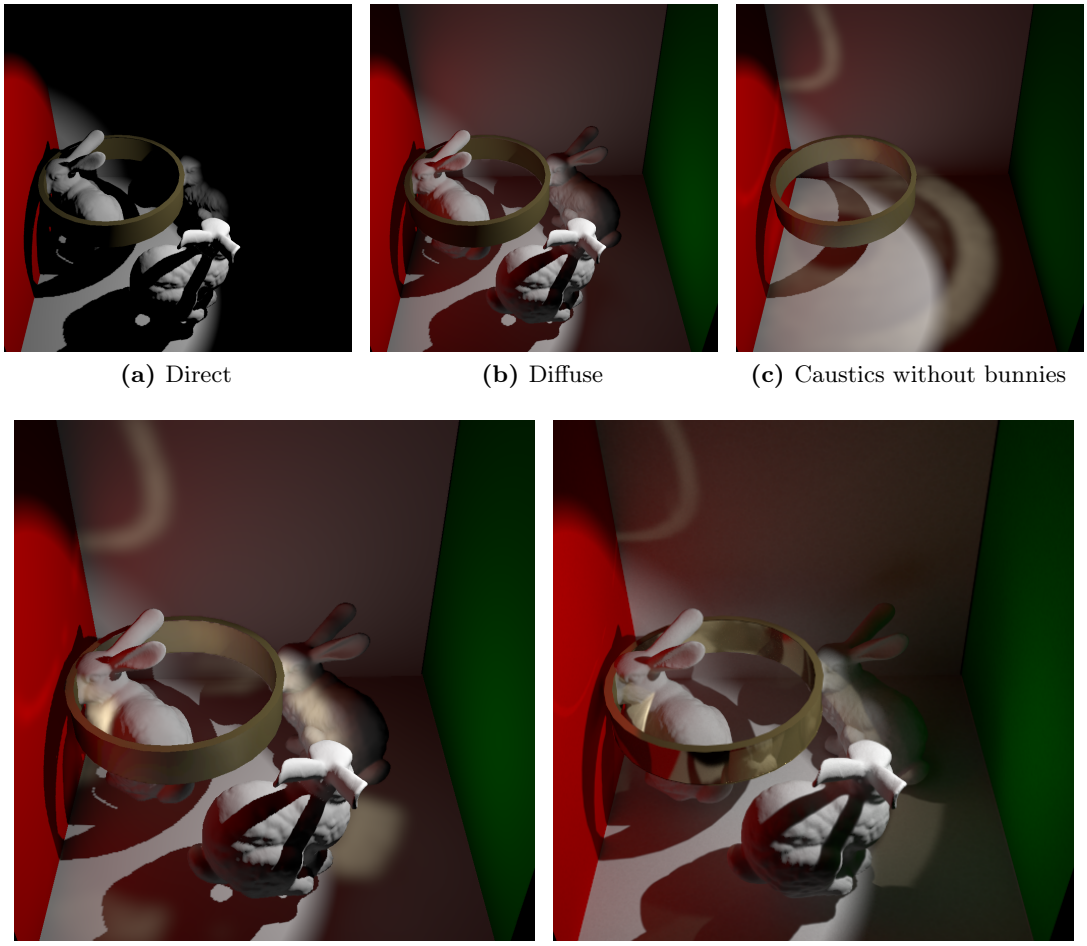


Figure 4.2: Occluded ring scene. The image (b) shows light leaking due to missing of the ring's platform in the depth layers. The image (d) shows the correct photons' locations using per-pixel linked list tracing method.

4.2 Water room

This scene comprises dynamic water at the bottom of a room. There are 1932 triangles. Figure 4.4(e) shows the accuracy of Photon tracing at 256^2 resolution, especially on surfaces viewed at a steep angle. 128^2 tracing resolution produces some aliased caustics shapes. Fortunately, these errors can be reduced by increasing photon splatting size, albeit having blurred result. Similarly, decreasing number of photons greatly increases the performance, but also leading to blurred image (figure 4.4(a) & (b)).



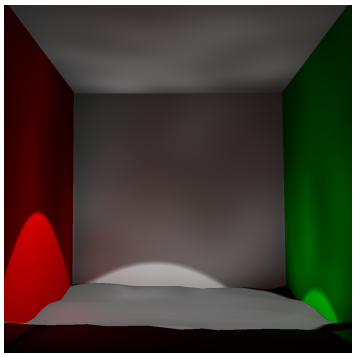
(a) Direct **(b)** Diffuse **(c)** Caustics without bunnies

(d) Caustics **(e)** SPPM reference ($\sim 10^8$ photons in 8 hours)

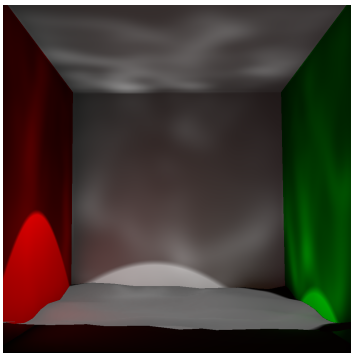
Figure 4.3: Bunnies scene.

4.3 Crytek's Sponza

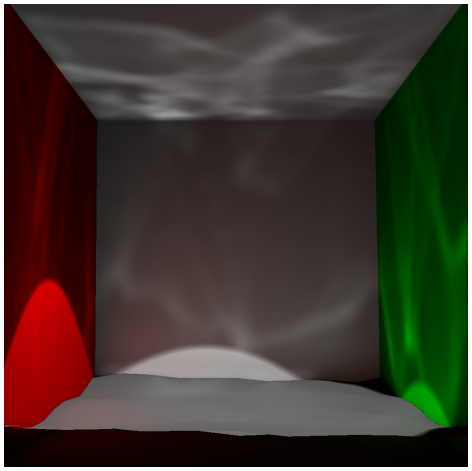
This is the most complex test scene, having 261989 triangles and high depth complexity. In order to test the caustics effect, we have modified the original model from Crytek to add several water pools at various locations. Figure 4.6 and 4.8 both are tested with low, medium and high number of photons, where higher number is about four times as many as lower number. Table 4.5 shows that Photon tracing performance almost scales linearly with the number of photons and with the tracing resolution.



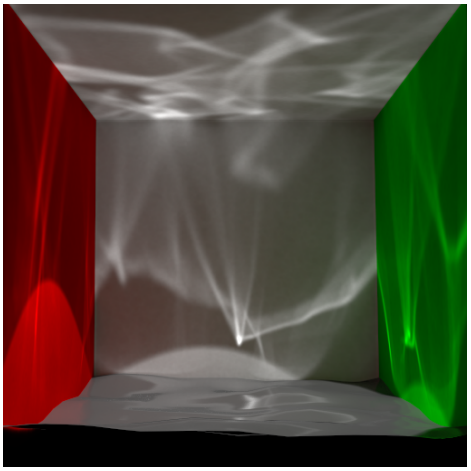
(a) 2313 photons



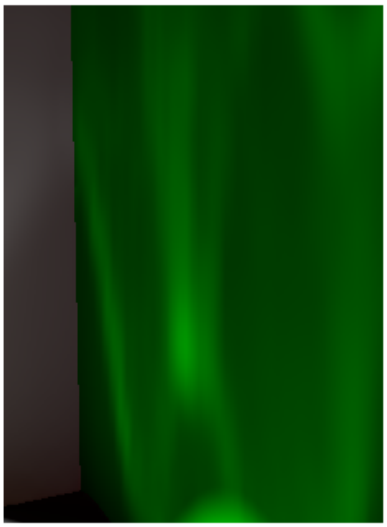
(b) 9417 photons

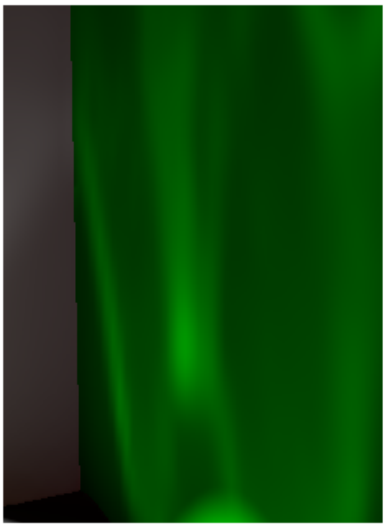


(c) 38146 photons



(d) SPPM reference ($\sim 10^8$ photons in 2 hours)



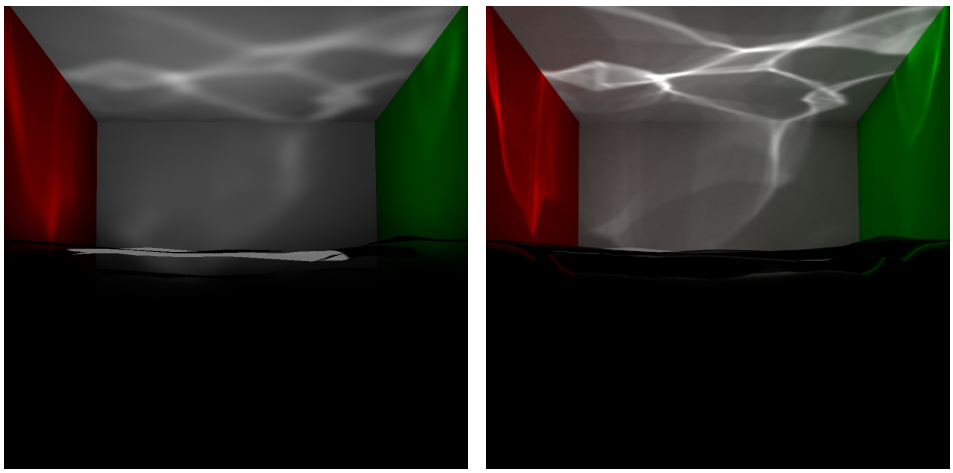


(e) Results of 128^2 and 256^2 Photon tracing resolution

Figure 4.4: Water scene.

Table 4.4: Statistics of the water room scene.

Figure	Resolution	Caustic photons	Caustic ray resolution	FPS	Diffuse illumination time(ms)		Caustics illumination time(ms)		Frame time(ms)
					Multi-resolution splitting	Multi-resolution gathering	Photon tracing	Photon splatting	
4.4(a)	1024^2	2313	128^2	68	0.320	4	2	2	14
	1024^2	2313	256^2	56	0.320	4	5	2	18
4.4(b)	1024^2	9417	128^2	45	0.320	4	10	2	22
	1024^2	9417	256^2	31	0.320	4	19	2	31
4.4(c)	512^2	38146	128^2	20	0.089	2	42	0.748	49
	512^2	38146	256^2	12	0.089	2	79	0.719	85
	1024^2	38146	128^2	18	0.320	4	42	3	55
	1024^2	38146	256^2	11	0.320	4	79	2	92
4.5(a)	1024^2	16384	128^2	40	0.320	4	13	2	25
	1024^2	16384	256^2	26	0.320	4	26	2	38



(a) 16384 photons

(b) SPPM reference ($\sim 10^8$ photons in 2 hours)

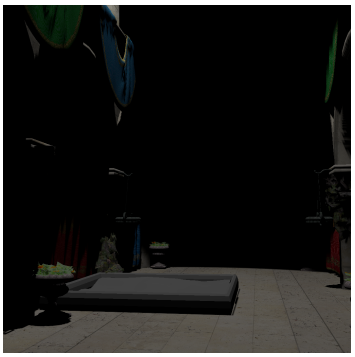
Figure 4.5: Water scene 2.

(a) Direct **(b)** Diffuse **(c)** 1707 photons **(d)** 6954 photons

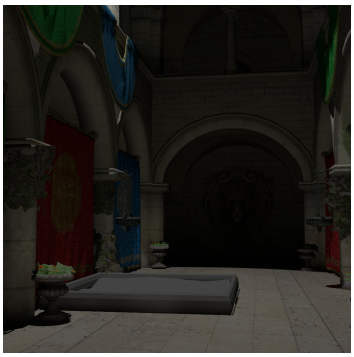
(e) 28111 photons **(f)** SPPM reference ($\sim 10^8$ photons in 8 hours)

(g) Diffuse **(h)** 28111 photons

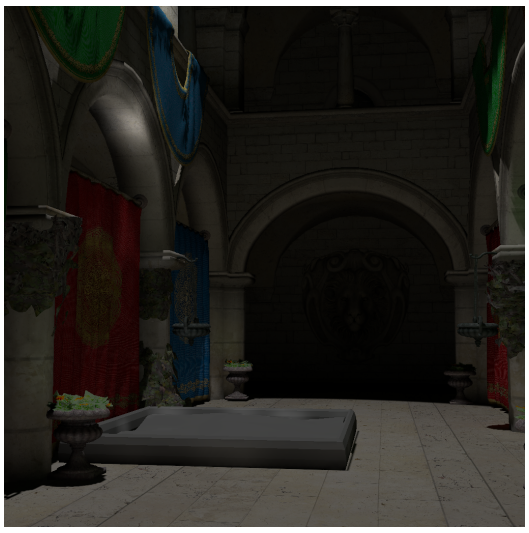
Figure 4.6: Sponze scene 1.



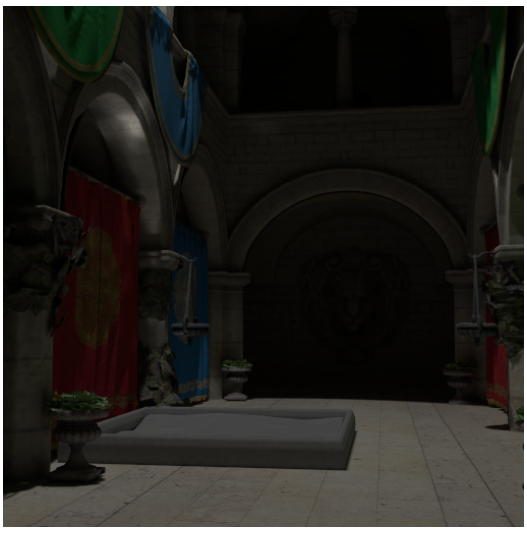
(a) Direct



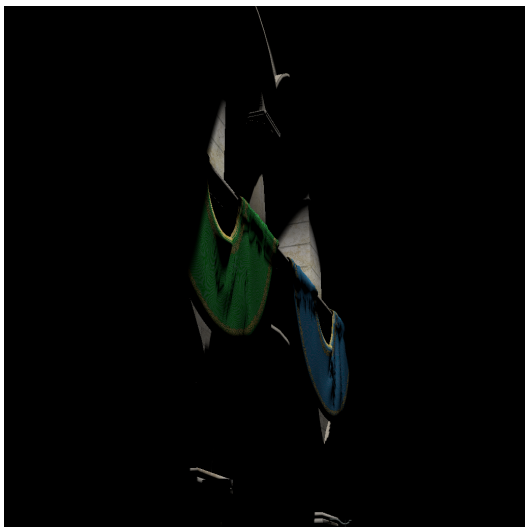
(b) Diffuse



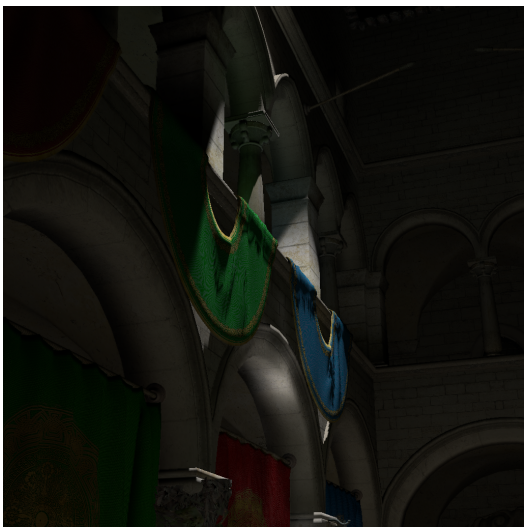
(c) Caustics



(d) SPPM reference ($\sim 10^8$ photons in 8 hours)



(e) Direct



(f) Caustics

Figure 4.7: Sponze scene 2.

Table 4.5: Statistics of the Sponza scenes.

Figure	Resolution	Caustic photons	Caustic ray resolution	FPS	Diffuse illumination time(ms)		Caustics illumination time(ms)		Frame time(ms)
					Multi-resolution splitting	Multi-resolution gathering	Photon tracing	Photon splatting	
4.6(c)	1024^2	1707	128^2	66	0.320	3	2	2	15
	1024^2	1707	256^2	52	0.320	3	5	2	19
4.6(d)	1024^2	6954	128^2	43	0.320	3	10	2	22
	1024^2	6954	256^2	28	0.320	3	21	2	35
4.6(e)	512^2	28111	128^2	19	0.089	2	43	0.760	50
	512^2	28111	256^2	11	0.089	2	85	0.760	92
	1024^2	28111	128^2	17	0.319	3	43	2	56
	1024^2	28111	256^2	10	0.319	3	85	3	99
4.7(c)	512^2	15049	128^2	31	0.091	4	22	0.145	31
	512^2	15049	256^2	18	0.091	4	44	0.593	54
	1024^2	15049	128^2	23	0.328	10	22	0.859	41
	1024^2	15049	256^2	15	0.328	10	44	0.924	65
4.8(c)	1024^2	1500	128^2	52	0.323	8	2	1	19
	1024^2	1500	256^2	46	0.323	8	4	1	22
4.8(d)	1024^2	6106	128^2	38	0.323	8	8	2	26
	1024^2	6106	256^2	28	0.323	8	16	2	35
4.8(e)	512^2	24726	128^2	23	0.091	3	34	0.929	42
	512^2	24726	256^2	13	0.091	3	67	0.929	76
	1024^2	24726	128^2	19	0.323	8	34	2	52
	1024^2	24726	256^2	11	0.323	8	67	2	86

(a) Direct

(b) Diffuse

(c) 1500 photons

(d) 6106 photons

(e) 24726 photons

(f) SPPM reference ($\sim 10^8$ photons in 8 hours)

Figure 4.8: Sponze scene 3.

Chapter 5

Conclusion

In summary, our method can achieve plausible diffuse indirect illumination and caustics in interactive to real-time frame rate, depends on the scene and number of photons. The method makes full use of the rasterization power of modern GPUs and supports fully dynamic scenes. Thus, it can be integrated easily into existing rasterization frameworks. Without a doubt, the most expensive step is Photon tracing. That is because the intersection test has to be performed on many fragments of the rasterized rays. Nevertheless, the tracing performance depends more on rasterization's resolution rather than on the scene's number of polygons. Hardware's early z-rejection mechanism can help skip a lot of fragments in the empty spaces, or spaces with min depth smaller than the fragment's depth. Furthermore, we can rasterize the rays at low resolution without significant loss of the accuracy.

5.1 Limitations and future improvements

In our implementation, both techniques for diffuse and caustics are screen-space based, hence inconsistencies may occur between successive frames because they are dependent on viewing direction. Alternative way to capture the linked lists could be using a fixed projection's direction, instead of the camera adaptive one. Another limitation is that linked lists are only generated from the geometry within the projection's frustum. Thus, certain parts of the scene such as those behind the camera, if we use camera adaptive projection, could be omitted. One possible solution is using a big frustum containing the whole scene, or only nearby objects around the camera, the latter option is better because of limited memory on the GPU. Limiting the linked list to only contain nearby objects also helps reduce the depth complexity, which in turn benefits the intersection test.

Our method does not consider diffuse indirect occlusion. However, it is usually hard to notice because in real-life, diffuse light scatters and bounces all over the scene. If diffuse indirect shadow is desired, we can integrate the Imperfect Shadow Map method [RGK+08], albeit having reduced performance, or using a more efficient, but less physically correct method -

Light Propagation Volumes [KD10].

Currently, medium to high frequency glossy indirect lighting is not supported, since our method only traces rays from light source while glossy reflections are view dependent and thus require tracing rays from the camera. Not only that, number the rays required to trace is much larger than the number of Caustics rays because in order to estimate the Rendering Equation integral at each shaded pixel, the reversed tracing procedure needs to sample more than one ray. Consider that the forward tracing step in our method already takes a toll on the performance, glossy effects are hard to achieve in real-time frame rate.

Finally, in the future we would like to investigate a hierarchical tracing method, in which low resolution photon rays are rendered first to remove those portions that are sure to not intersect any fragments in the linked lists. Then the remaining portions are rasterized at higher resolution to do fine-grained intersection tests.

References

[CNS+11] Cyril Crassin, Fabrice Neyret, Miguel Sainz, Simon Green, and Elmar Eisemann. Interactive indirect illumination using voxel-based cone tracing: An insight. In *ACM SIGGRAPH 2011 Talks*, SIGGRAPH '11, pages 20:1–20:1, New York, NY, USA, 2011. ACM.

[DS05] Carsten Dachsbacher and Marc Stamminger. Reflective shadow maps. In *Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games*, I3D '05, pages 203–231, New York, NY, USA, 2005. ACM.

[HJ09] Toshiya Hachisuka and Henrik Wann Jensen. Stochastic progressive photon mapping. In *ACM SIGGRAPH Asia 2009 Papers*, SIGGRAPH Asia '09, pages 141:1–141:8, New York, NY, USA, 2009. ACM.

[Jen96] Henrik Wann Jensen. Global illumination using photon maps. In *Proceedings of the Eurographics Workshop on Rendering Techniques '96*, pages 21–30, London, UK, UK, 1996. Springer-Verlag.

[KBW06] Jens Krüger, Kai Bürger, and Rüdiger Westermann. Interactive screen-space accurate photon tracing on gpus. In *Proceedings of the 17th Eurographics Conference on Rendering Techniques*, EGSR'06, pages 319–329, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association.

[KD10] Anton Kaplanyan and Carsten Dachsbacher. Cascaded light propagation volumes for real-time indirect illumination. In *Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, I3D '10, pages 99–107, New York, NY, USA, 2010. ACM.

[Kel97] Alexander Keller. Instant radiosity. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '97, pages 49–56, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[LB13] Philipp Lensing and Wolfgang Broll. Efficient shading of indirect illumination applying reflective shadow maps. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, I3D '13, pages 95–102, New York, NY, USA, 2013. ACM.

[LP03] Fabien Lavignotte and Mathias Paulin. Scalable photon splatting for global illumination. In *Proceedings of the 1st International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia*, GRAPHITE '03, pages 203–ff, New York, NY, USA, 2003. ACM.

[MKC07] Ricardo Marroquim, Martin Kraus, and Paulo Roma Cavalcanti. Efficient point-based rendering using image reconstruction. In Mario Botsch, Renato Pajarola, Baoquan Chen, and Matthias Zwicker, editors, *SPBG*, pages 101–108. Eurographics Association, 2007.

[ML09] Morgan McGuire and David Luebke. Hardware-accelerated global illumination by image space photon mapping. In *Proceedings of the Conference on High Performance Graphics 2009*, HPG '09, pages 77–89, New York, NY, USA, 2009. ACM.

[NSW09] Greg Nichols, Jeremy Shopf, and Chris Wyman. Hierarchical image-space radiosity for interactive global illumination. In *Proceedings of the Twentieth Eurographics Conference on Rendering*, EGSR'09, pages 1141–1149, Aire-la-Ville, Switzerland, Switzerland, 2009. Eurographics Association.

[NW09] Greg Nichols and Chris Wyman. Multiresolution splatting for indirect illumination. In *Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games*, I3D '09, pages 83–90, New York, NY, USA, 2009. ACM.

[PKD12] Roman Prutkin, Anton Kaplanyan, and Carsten Dachsbacher. Reflective shadow map clustering for real-time global illumination. In Carlos AndÃžjar and Enrico Puppo, editors, *Eurographics (Short Papers)*, pages 9–12. Eurographics Association, 2012.

[RGK+08] T. Ritschel, T. Grosch, M. H. Kim, H.-P. Seidel, C. Dachsbacher, and J. Kautz. Imperfect shadow maps for efficient computation of indirect illumination. *ACM Trans. Graph.*, 27(5):129:1–129:8, December 2008.

[SGNS07] Peter-Pike Sloan, Naga K. Govindaraju, Derek Nowrouzezahrai, and John Snyder. Image-based proxy accumulation for real-time soft global illumination. In *Proceedings of the 15th Pacific Conference on Computer Graphics and Applications*, PG '07, pages 97–105, Washington, DC, USA, 2007. IEEE Computer Society.

[SKALP05] LÃąszlÃş Szirmay-Kalos, BarnabÃąs AszÃşdi, IstvÃąn LazÃąnyi, and MÃątyÃąs Premecz. Approximate ray-tracing on the gpu with distance impostors. *Comput. Graph. Forum*, 24(3):695–704, 2005.

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

[TO12] Yusuke Tokuyoshi and Shinji Ogaki. Real-time bidirectional path tracing via rasterization. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, I3D '12, pages 183–190, New York, NY, USA, 2012. ACM.

[YHGT10] Jason C. Yang, Justin Hensley, Holger Grün, and Nicolas Thibieroz. Real-time concurrent linked list construction on the gpu. In *Proceedings of the 21st Eurographics Conference on Rendering*, EGSR'10, pages 1297–1304, Aire-la-Ville, Switzerland, Switzerland, 2010. Eurographics Association.

[YWC+10] Chunhui Yao, Bin Wang, Bin Chan, Junhai Yong, and Jean-Claude Paul. Multi-image based photon tracing for interactive global illumination of dynamic scenes. In *Proceedings of the 21st Eurographics Conference on Rendering*, EGSR'10, pages 1315–1324, Aire-la-Ville, Switzerland, Switzerland, 2010. Eurographics Association.