國立臺灣大學電機資訊學院資訊工程學系

碩士論文

Department of Computer Science and Information Engineering
College of Electrical Engineering and Computer Science
National Taiwan University

Master Thesis

利用能見度近似之反射性陰影圖改良

Improved Reflective Shadow Maps with Visibility Approximation

馮禹翰

Mifan Bang

指導教授：莊永裕 博士

Advisor: Yung-Yu Chuang, Ph.D.

中華民國 100 年 7 月

July, 2011


國立臺灣大學
人格教育・學生愛國

中文摘要

反射性陰影圖是個被廣泛應用於即時影像生成的全域照明演算法，它延伸了傳統陰影圖之概念與應用，並且能繪製經過第一次反射的間接照明效果。雖然反射性陰影圖被視為一個效率良好的演算法，然而出自於運算複雜度方面的考量，它並不針對間接照明過程中物體對光線之遮蔽效應進行處理。在實際應用當中，若是忽略間接光源在照明過程中被場景物體所遮蔽之效應，經常會造成諸如照明效果不正確地溢出等的負面結果。為了解決此問題，我們以反射性陰影圖做為基礎，進而提出一個有效率的能見度估計演算法以及一個全新的採樣模式。所有原本演算法的優點都被我們刻意保留，如：完全運作於螢幕空間內、不須對資料進行任何預先處理等，因此我們的演算法同樣適合於高度動態之場景與照明環境。另外，我們尚詳述了針對當代圖形處理單元（GPU）所設計的演算法實作系統，其中引入了許多常用的即時繪圖加速技巧以達到高度效能表現。最後的實驗結果顯示，我們所提出之改良演算法相較於傳統反射性陰影圖，能夠產生更為真實的影像。


國立臺灣大學
博愛・學品教育

Abstract

In this work we propose an improved Reflective Shadow Map algorithm by approximating visibility to account for indirect shadows. Reflective Shadow Map (RSM) is a popular approach for real-time global illumination. It approximates the first-bounce indirect lighting by using an extension of standard shadow maps. RSM, however, does not take visibility into consideration. Ignoring occlusion between secondary light source and shading point would result in light leakage and produce undesired results. To address this problem, we present an efficient algorithm for visibility approximation and a novel sampling pattern that cooperates well with it. Our approach works entirely in screen space and requires no precomputation of any sort. Therefore, it is suitable for highly dynamic scenes and lighting changing environments. We also describe a detailed implementation on contemporary GPU with many widely used real-time rendering techniques to achieve high performance. Results show the image quality produced by our approach is more realistic compared to the original RSM.


國立臺灣大學
博愛 · 學術 · 教育

Contents

List of Figures

List of Tables


國立臺灣大學
博雅教育・學術研究

Chapter 1

Introduction

Global illumination has been demonstrated to be perceptually important [15] and is frequently applied to synthesize photorealistic image. Despite an abundance of recent algorithms have been proposed recently, computing global illumination for fully dynamic scenes at real-time frame rate still remains a challenging problem. Evaluating global illumination requires incident radiance from the entire hemisphere to be gathered at every shading point. Because the complexity of light transport requires a large amount of calculation, the computation budget can easily run out. Therefore, indirect illumination often has to be coarsely approximated in real-time applications.

The Reflective Shadow Maps (RSM) [2] algorithm has been proposed recently and applied widely to approximate one-bounce indirect lighting. Pixels on these extended shadow maps are interpreted as small secondary light sources (also referred to as pixel lights or virtual point lights, the VPLs) and used to illuminate the scene. The major advantage of RSM is that neither expensive particle tracing nor acceleration structures are required. However, the original RSM approach does not include the visibility computation for indirect lights. The ignoring of visibility leads to light leakage, which may produce inconvincible results.

Previous work [4, 11] has shown that visibility queries are the major bottleneck of computing global illumination. The goal of our work is to propose an efficient algorithm to approximate visibility for pixel lights as various other previous work did [11, 5, 16]. Our main idea is to reuse information from RSMs and determine whether a pixel light can

illuminate the shading point. We also propose a new sampling pattern for selecting pixel lights which cooperates well with our novel visibility approximation. Since indirect illumination is usually smooth, the approximation produces visually plausible indirect shadows. Our approach runs entirely in screen space and thus it scales well with the scene complexity. In addition, we implemented our system on modern GPU hardware with a number of widely used real-time rendering techniques to achieve high frame rate even with large and complex scene environments.

Chapter 2

Related Work

In this chapter we will review the previous work of real-time and interactive global illumination. Offline rendering algorithms such as ray tracing and photon mapping are not taken into consideration. In addition, we focus on approaches that support fully dynamic scenes without precomputation.

2.1 Image-Space Global Illumination

Image-space, or interchangeably screen-space, approaches have advantages of good scalability with scene complexity and excellent support for dynamic scenes. Dachsbacher and Stamminger used reflective shadow maps (RSMs) [2] to compute one-bounce indirect illumination. They further adapted splatting and importance sampling to produce more complicated lighting effects [3]. Based on RSMs, recently Nichols et al. proposed image-space radiosity [9] and multi-resolution splatting [10] to reduce the gathering cost from virtual point lights. However, none of these methods consider visibility of indirect lighting. Similar to the work just mentioned, our work is also based on RSMs but focuses on approximating visibility for indirect lights which previous work lacks.

2.2 Screen-Space Ambient Occlusion

Ambient occlusion (AO) is another popular approach to roughly simulate global illumination effect and able to render soft self-shadows. Screen-space ambient occlusion (SSAO) approximates AO in screen space and is widely adopted by contemporary high-

end gaming applications. Shanmugam and Arikan gathered ambient occlusion in screen space using a GPU-friendly approach [14]. Ritschel et al. then extended this idea to support directional occlusion and gathering one-bounce indirect illumination effects from near fields [12]. Our work is largely inspired by their concept and image sampling strategy for occlusion determination.

2.3 Visibility Approximation

Ritchel et al. proposed using low-resolution shadow maps (imperfect shadow maps) to estimate visibility function for virtual point lights [11]. To accelerate rendering shadow maps, surfaces in the scene were represented as disks to be splattered into the depth buffer. Their method produced visually pleasant results and can support various types of light sources. However, as the geometry became more complex, a larger number of point samples was required, further increasing the render cost. Dong et al. proposed an approach to cluster VPLs into virtual area lights (VALs) and used convolution soft shadow maps to generate visibility function [5]. Their solution only achieved interactive frame rates for moderate scenes. Kaplanyan and Dachsbacher used fuzzy occlusion for indirect lighting in their cascaded propagation volume approach [6]. Since low-order spherical harmonics were used to represent the geometry and lights, their method worked well for low-frequency illumination. Nevertheless, their work still suffered from light leakage when the low-order spherical harmonic coefficients were insufficient to capture complex geometric details. Thiedemann et al. used texture atlases and created a rough voxelization of the scene to capture the geometry not presenting in the final image [16]. They proposed a fast GPU ray-casting algorithm to trace rays in the voxelized grid for visibility determination. However, their approach required precomputation of the mapping from geometry to texture atlas and did not suit well with warping-based animation.

Chapter 3

Methodology

Our method is based on reflective shadow maps (RSMs) [2] and is capable to approximate occlusion for indirect illumination. The basic idea is that during the process of RSM sampling, some information about depth values and world-space coordinates of previously sampled pixel lights can be maintained and used to determine the visibility of succeeding pixel lights.

In the following sections, we describe in detail how pixel lights are sampled and visibility approximated for indirect illumination.

3.1 Reflective Shadow Maps

An RSM is an extended shadow map. It follows the observation that all pixels on an RSM can be viewed as the exact surface patches directly lit by a light source. Furthermore, we can treat these patches as secondary light sources, named pixel lights, and use them to calculate the first-bounce indirect illumination. In addition to the depth value d_p that a standard shadow map maintains for each pixel p, an RSM also stores the world-space position $\mathbf{x}_p$, the normal $\mathbf{n}_p$, and the reflected radiant flux Φ_p. By assuming that pixel lights are infinitely small in area size and perfectly diffuse, the emitted radiant intensity along direction ω is:

$$I_p(\omega) = \Phi_p \max\{0, \langle \mathbf{n}_p | \omega \rangle\}, \tag{3.1}$$

where $\langle \ | \ \rangle$ denotes the dot product.

Because taking all pixel lights into account for indirect illumination is impractical for

real-time application, only a subset of them is gathered. The original RSM algorithm re-projects a shading point to RSM and uses an importance-driven sampling pattern to pick up the contributing pixel lights close to the projection in screen space. A precomputed Poisson distribution is used to obtain an even sampling distribution. The original work had mentioned that using the same sampling pattern for all indirect illumination calculation gave good coherence but might suffer from aliasing when the number of samples were insufficient. It was suggested to sample about 400 pixel lights for each shading point. However in our experiment, 400 samples can still result in apparent aliasing in complex scenes.

In order to simplify the calculation, visibility of pixel lights are completely ignored in the original RSM approach. Its sampling pattern is also difficult to estimate visibility with the only information stored in RSM itself. Consequently, indirect illumination approximated by RSM will always be free of indirect shadow and may suffer from light leakage. To address this drawback, we propose an improved algorithm.

3.2 RSM Sampling

To render the indirect illumination result on a given shading point of the final image, we follow the original RSM approach to re-project the shading point onto the RSM and sample neighboring pixel lights in screen space. The reason of selecting neighboring samples is that if two points are close in world space, they likely are close to each other in screen space as well. Moreover, neighboring pixel lights usually contribute more and are desirable to be sampled densely. The original RSM uses importance sampling to realize such idea. Sampling density decreases with longer distance between the projected shading point and a pixel light while the weight increases proportionally to the square of distance in order to compensate the varying density.

The original RSM approach is efficient in gathering irradiance from pixel lights; however, it is incapable of determining visibility of samples to the shading point. We try to address this problem by keeping some additional knowledge of closer selected pixel lights in the sampling process and use them to estimate the visibility of farther ones. Our algo-

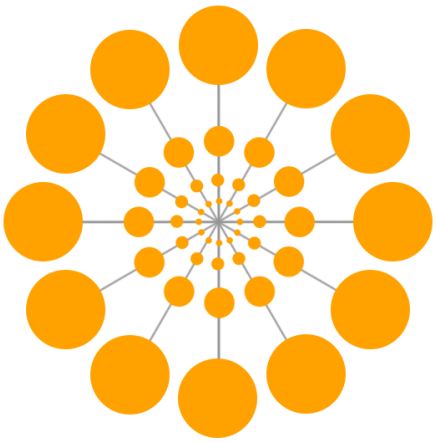


Figure 3.1: Example of our base sampling pattern. Similar to the original RSM, sampling density decreases and weight (represented as the radius of disk) increases with the distance to the shading point. However, we restrict the sampling path to be on the same lines originating from the projection.

rithm exploits the following observation: if there exists an occluder between the shading point and a pixel light in world space, the occluder must also lie on the connecting line of the two in screen space, given that the projection is either perspective or orthogonal. Therefore, after re-projecting a shading point to RSM, we sample along multiple straight lines originating from the projection. Along each line, pixel lights are sampled in a near-to-far fashion. This allows us to check if any selected pixel light occludes the newly sampled one to the shading point, based on the knowledge we gather from those earlier samples. Description of our algorithm for visibility estimation and the specific *knowledge* required to be kept will be presented in the next section. Figure 3.1 shows our radial- and concentric-looking base sampling pattern. Similar to the original RSM, we use the strategy of varying density and weighting along each sampling path. Algorithm 1 describes our sampling method more detailedly. For a pixel light of world-space position $\mathbf{x}_p$, normal $\mathbf{n}_p$, and radiant flux Φ_p, its contribution to a shading point's irradiance is:

$$E_p(\mathbf{x}, \mathbf{n}) = \Phi_p \frac{\max\{0, \langle \mathbf{n}_p \mid \mathbf{x} - \mathbf{x}_p \rangle\} \max\{0, \langle \mathbf{n} \mid \mathbf{x}_p - \mathbf{x} \rangle\}}{\|\mathbf{x} - \mathbf{x}_p\|^4}, \tag{3.2}$$

where $\mathbf{x}$ and $\mathbf{n}$ are world-space position and normal of the shading point.

The number of samples per shading point is a trade-off between performance and quality of indirect illumination. To obtain high performance while avoiding severe aliasing,

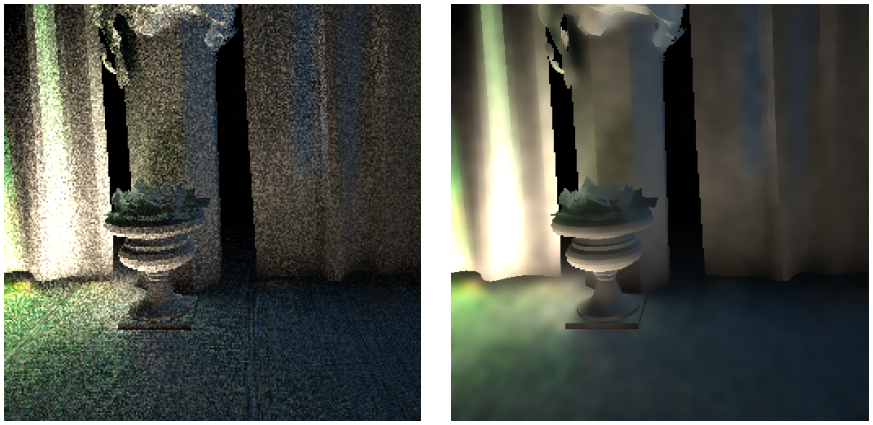
Algorithm 1 Sample RSM for a shading point

```
1:  irradiance ⇐ 0
2:  for each sampling direction θ do
3:     for each sampling radius r along a path do
4:        Slightly jitter θ.
5:        Slightly scale and jitter r.
6:        Transform (θ, r) to Cartesian coordinate (u, v).
7:        proj_p ⇐ p's projection on RSM
8:        samplePosition ⇐ proj_p + (u, v)
9:        if samplePosition is inside the RSM then
10:          Retreive pixel light p on RSM at position samplePosition.
11:          Calculate p's visibility ν to the shading point.
12:          if ν = "visible" then
13:             Calculate weight w of sample p.
14:             Calculate irradiance contribution E_p due to p with Eq. 3.2.
15:             irradiance ⇐ irradiance + w · E_p
16:          end if
17:       end if
18:    end for
19: end for
20: return irradiance
```

we use the same base pattern for all shading points but with slightly difference of rotation, scaling, and jitters. In our experiments, a rather low number of samples (about 100-150 samples per shading point) is enough to generate aliasing-free results. In spite of the absence of aliasing, using less samples with jittered sampling pattern suffers from high frequency noises (Figure 3.2). Therefore another render pass is added as post processing which applies an edge-aware blur to the noised irradiance of indirect illumination, and its result turns out to be visually plausible.

3.3 Visibility Determination

As the previous section described, pixel lights are sampled along a couple of straight lines originating from the projection of shading point on RSM. For each pair of a shading point and a pixel light, we transform them to eye space of the light source and evaluate the elevation angle, where the xy-plane is considered the horizontal plane. To correctly calculate the angle, z-coordinate in eye space, which is a linearized depth, must be stored as the depth value in RSM. A small array of buffers, named micro linear buffer, is introduced



(a) (b)

Figure 3.2: Noised irradiance of indirect illumination. (a) High-frequency noise appears due to a low number of samples and varying sampling patterns for each shading point. (b) We apply an edge-aware blur to approximate the low-frequency irradiance of indirect illumination.

to maintain the status of in which ranges of elevation angle exist some previously sampled pixel lights. Each buffer represents a range of angles, and the whole buffer array equally divides all possible angles in $[-\pi/2, \pi/2]$ according to its size (Figure 3.3).

Given a newly sampled pixel light, we calculate its elevation angle with respect to the shading point and find the corresponding value in micro linear buffer. If the buffer slot is empty, it is interpreted as that there resides no previous samples in the same range of angle, and therefore that pixel light should illuminate the shading point. In this case, we evaluate the lighting contribution and fill the buffer to represent its occlusion on any farther samples in the same range of angle. Otherwise, the pixel light is viewed as occluded and should not be taken into account for indirect illumination.

Algorithm 2 Approximate visibility of a pixel light to the shading point

Require: Micro linear buffer is initialized as "visible" in all its buffer slots.

1: Calculate elevation angle $\phi \in [-\pi/2, \pi/2]$.
2: Map ϕ to the index idx of micro linear buffer $buffers$.
3: **if** $buffers[idx] =$ "visible" **then**
4: $\quad buffers[idx] \Leftarrow$ "occluded"
5: $\quad$ **return** "visible"
6: **else**
7: $\quad$ **return** "occluded"
8: **end if**

The size of micro linear buffer should be some fraction of the number of samples on the

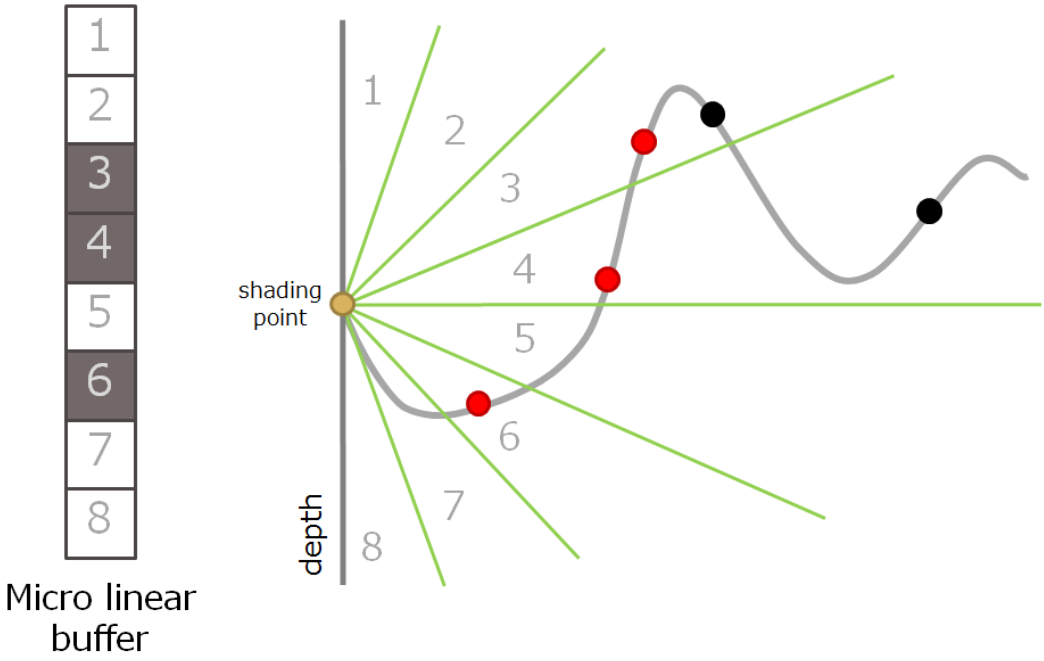


Figure 3.3: Example of micro linear buffer. In this case we eight buffer slots (stack of squares on the left). Numbers appearing on both sides illustrate the mapping from certain range of elevation angles (right) to buffer index. The terrain-like curve on the right represents the depths in RSM. Red dots on the curve are samples considered occluders by our algorithm, and black dots are samples occluded.

same sampling path. The main reason is that we look forward to some occluded pixel light on each path, or else indirect shadows will be inapparent and results similar to the original RSM can be produced. Fewer buffer slots lead to higher probability of buffer collision. In our experiments, 5 slots of micro linear buffer together with 10 samples along each path yield good balance between quality and performance. Artifacts may appear when the size of micro linear buffer is smaller than 5. We also observed that neighboring samples, especially those on the same smooth geometry, have high probability to be mapped into the same buffer (Figure 3.4), making earlier samples always considered as occluders to their subsequent neighbors. An additional check can solve part of this problem: visibility will be propagated if a sample has the same buffer index as its predecessor. Algorithm 3 shows an improved version of visibility approxmation that includes the additional check, where idx_{-1} and ν_{-1} are the index to micro linear buffer and the visibility of the last sampled pixel light.

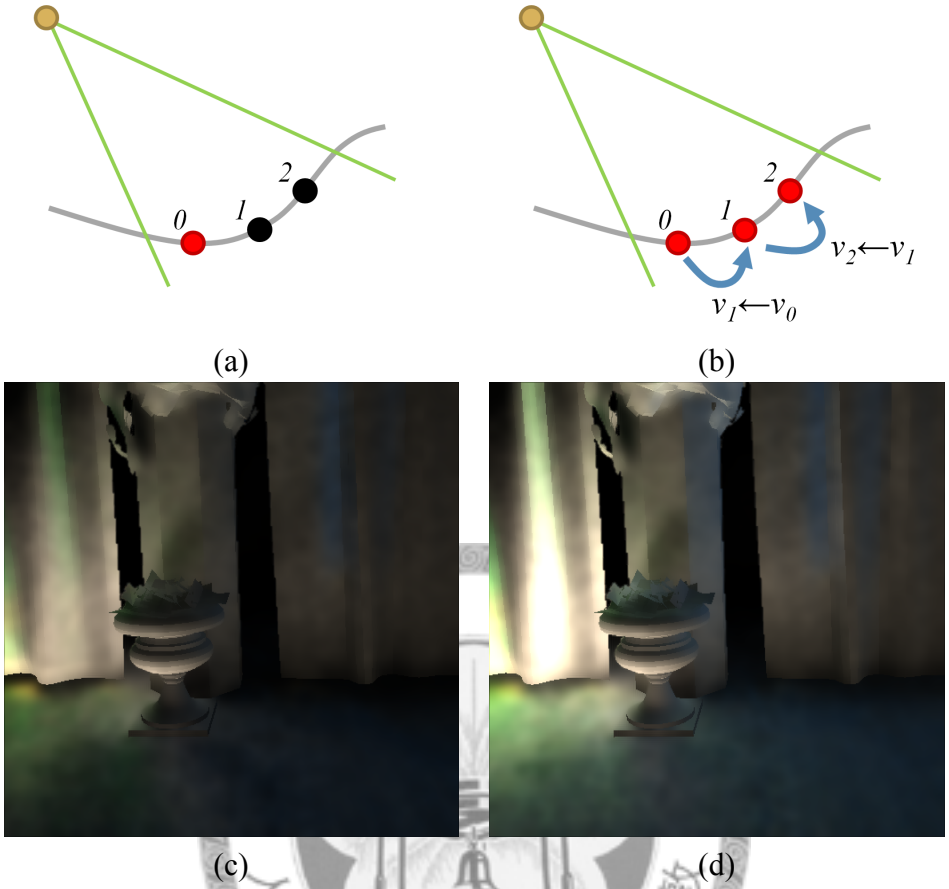


Figure 3.4: Problem with small size of micro linear buffer. (a) Neighboring samples on the same smooth surface can be mapped to the same range of elevation angle, making the two samples on the right being incorrectly determined as occluded. (b) When we encounter neighboring samples with the same buffer index, visibility will be propagated to avoid problems in (a). (c) Result of indirect irradiance that suffers serverely from problems in (a). (d) Result of indirect irradiance after we applied Algorithm 3 instead.

Algorithm 3 Improved visibility approximation of a pixel light to the shading point

Require: Micro linear buffer is initialized as "visible" in all its buffer slots.

1: Calculate elevation angle $\phi \in [-\pi/2, \pi/2]$.
2: Map ϕ to the index idx of micro linear buffer $buffers$.
3: **if** $idx = idx_{-1}$ **then**
4: $\quad \nu \Leftarrow \nu_{-1}$
5: **else**
6: $\quad \nu \Leftarrow buffers[idx]$
7: $\quad buffers[idx] \Leftarrow$ "occluded"
8: **end if**
9: **return** ν


國立臺灣大學
博愛 · 學品 · 教育

Chapter 4

Implementation

4.1 System Overview

Our algorithm is completely platform-independent, and thus it can be implemented without concerning compatibility issues with certain hardware or software. We would also like to preserve this advantage when it comes to code portability. Therefore, the system was implemented with OpenGL and OpenGL Shading Language (GLSL), which were open standards and widely supported. More specifically, we chose OpenGL 3.1 and GLSL 1.40 on GPUs since they were well designed to provide features that could fulfill today's high demands on performance and graphics quality while having relatively minimal hardware requirements. OpenGL 3.1 has removed fixed rendering pipeline features and enforces programs to store most of the data in graphics memory. Therefore it opens up the possibility of advanced shading techniques and rapid rendering for massive geometric primitives.

In addition to the choice of a modern graphics application programming interface (API), we also adopted a number of GPU performance-tuning techniques in the programmable rendering pipeline. These techniques helped further save graphics memory bandwidth and shading computation, greatly reducing the render time. Our system consists of the following four render passes:

1. RSM generation,

2. G-buffer generation,

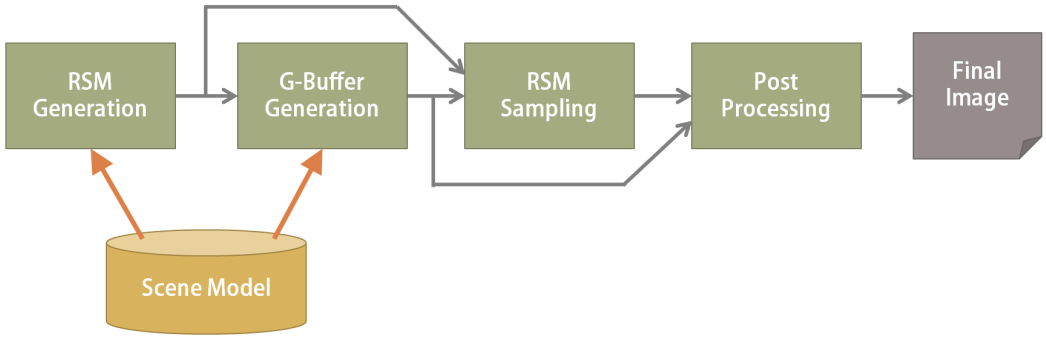


Figure 4.1: Workflow of our render system.

3. RSM sampling, and

4. post processing.

Each of them was implemented as a shader program composed of a vertex shader and a fragment shader. Figure 4.1 shows a workflow of our system. When the system starts up, both the geometry data and material textures are loaded from disk and copied to graphics memory for subsequent accesses. Each pass outputs to a render buffer which is directly allocated on graphics hardware and therefore will not consume the bandwith between graphics hardware and host memory.

4.2 RSM Generation

In the first pass, an RSM that contains every pixel's world-space position, world-space normal, linearized depth value, and radiant flux is written to several render buffer-bound textures. As the previous chapter mentioned, the linearization of depth value is required to correctly evaluate the elevation angle. We obtain such depth value by simply applying view matrix to a world-space position vector and taking the transformed z-component, preventing projection matrix from breaking its linearity. Except for the depth value, other calculation for RSM generation remains the same as the original work.

A render buffer-bound texture is a writable texture bound to a render buffer as the target memory region for fragment shader's outputs. Storing data in textures rather than elsewhere has the major advantage: a shader program in the subsequent render passes

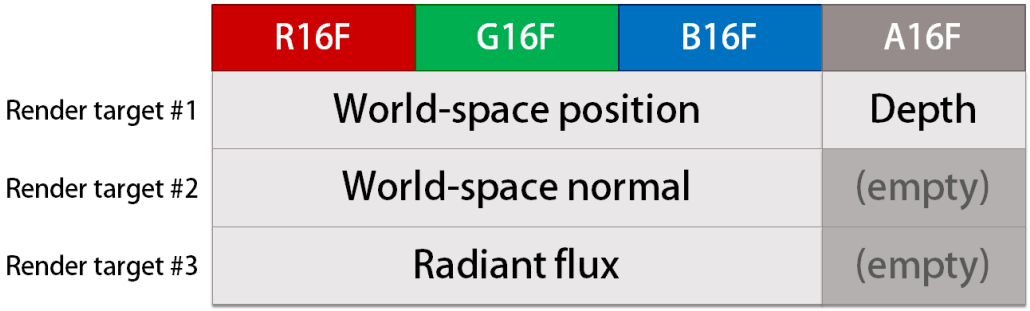


Figure 4.2: Layout of our reflective shadow map. Note that the depth is linearized.

can easily access these data as it fetches ordinary textures. Moreover, modern GPUs are specially optimized to cache and fetch textures in graphics memory, eliminating the need to transfer data back and forth between graphics hardware and host memory. To even further save the memory bandwidth, we choose the internal format for each texel to be 4 components of 16-bit half precision floating point number, which is defined in the OpenGL specification [13] and natively supported by most GPUs. In our experiment, this compact format had enough precision to represent the necessary data and saved a significant amount of space compared to its 32-bit counterpart. The layout of RSM textures is shown in Figure 4.2.

4.3 G-buffer Generation

The second pass, G-buffer generation, is designed to incorporate deferred shading. Because we take scalability with large and complex scenes seriously, deferred shading is adopted in order to decouple scene complexity from heavy shading calculation. Its implementation may vary from case to case, but the concept remains static. It renders the whole scene without complicated shading, and the result, namely the G-buffer, is the set of exact fragments with their attributes that the camera finally sees. Subsequent render passes that take G-buffer as input can save the redundant shading calculation for fragments that will eventually be discarded due to hidden surface removal. However, deferred shading is difficult to incorporate with translucent objects and multi-sample anti-aliasing (MSAA). Since neither rendering translucency nor MSAA closely relates to our interests, we adopt

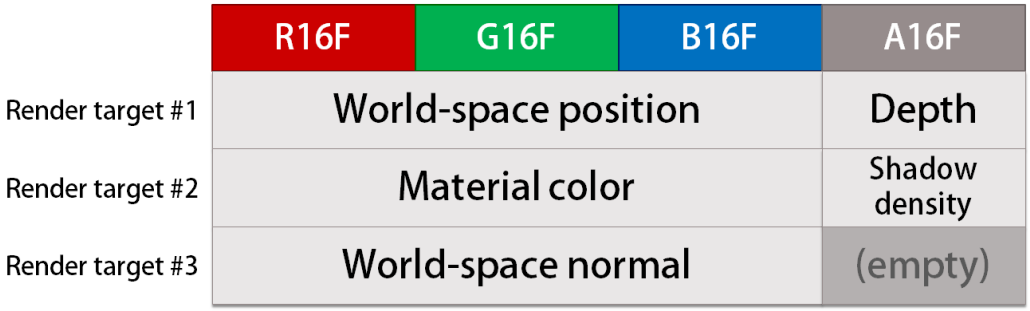


Figure 4.3: Layout of our G-buffer. Note that the depth is linearized.

deferred shading without these concerns. Our G-buffer consists of per-pixel world-space position, world-space normal, material color, and shadow density from direct illumination. A G-buffer is also stored in render buffer-bound textures to be accessed efficiently in subsequent passes. The layout of G-buffer textures is shown in Figure 4.3.

Shadow density in our G-buffer is obtained by multi-sampling the depth values in RSM to soften the direct shadow edge (Algorithm 4). As a standard shadow map is part of an RSM, we simply use the information already available in RSM for shadow mapping and average the results over a tiny kernel around the projection. A single result of shadow determination is either 0 or 1, indicating the presence of shadow or not (present for the value 0, absent otherwise). Averaging several such results yields some real number in interval $[0, 1]$ and can be considered the alpha value for material color to be blended with shadow color. In order to avoid artifacts due to truncation errors of floating point number, here we introduce a constant z-bias, which shifts all depth values on RSM a little bit farther along z-axis with respect to the origin. The post processing pass will later take meterial color and shadow density fields in G-buffer to reconstruct the result of direct illumination.

4.4 RSM Sampling

The third pass is RSM sampling. This is virtually the bottleneck of the system because it samples hundreds of pixel lights from RSM to render a single shading point. We would like to reduce the time spent here as much as possible or it will greatly affect the performance of the whole system. In this render pass, performance can dramatically drop

Algorithm 4 Calculate shadow density for pixel p

```
1: shadowCount ⇐ 0
2: for i = −kernelSize/2 to kernelSize/2 − 1 do
3:    for j = −kernelSize/2 to kernelSize/2 − 1 do
4:       samplePosition ⇐ p.position + (i, j)
5:       if samplePosition is inside RSM then
6:          if RSM.depth(samplePosition) − zBias ≥ p.depth then
7:             shadowCount ⇐ shadowCount + 1
8:          end if
9:       end if
10:   end for
11: end for
12: return shadowCount/kernelSize^2
```

due to either too many pixel lights being sampled or output resolution being too large. On the other hand, image quality must still be ensured by enough number of samples. Balance between performance and quality requires experiments and tuning those parameters accordingly. The sampling complexity can be further divided into two factors: number of paths and number of samples along the same path. Our experiment showed that the number of path had more effects on the image quality, suggesting that it would be better to traverse more paths than to sample more pixel lights along each path. In our implementation, 150 pixel lights (15 paths and 10 samples along each) are sampled from RSM for every shading point.

As the previous chapter illustrated, we applied a novel algorithm of visibility approximation for pixel lights within this pass. During the traversal of a given sampling path, a micro linear buffer of size 5 is used to record the elevation angles of all sampled pixel lights. We choose this size so that farther samples (totally 10 along each path, as mentioned earlier) have higher probability of buffer collision and therfore are prone to be determined as occluded. In other words, some occlusion is desirable. Nevertheless, introducing micro linear buffer increases graphics memory footprint. Data transfer between graphics memory and GPU is one of the most significant bottlenecks for GPU applications because of its long latency [7]. This consideration also leads to our decision of mapping all angles in $[-\pi/2, \pi/2]$ to relatively fewer buffers.

Visibility determination may also have the impact of increased branch divergence if not

handled carefully. Branch divergence diminishes the order of parallelism due to the design of modern GPU hardware [7]. GPUs are only able to execute the same instruction for multiple data streams parallelly. Branches create different execution flows of the shader program and often leave some processing units in the GPU unoccupied because there are not enough threads in the same execution flow. We avoid increasing branch divergence by assigning real numbers instead of Boolean flags (e.g. "visible" and "occluded") that represents visibility to micro linear buffer. A micro linear buffer is initialized by assigning 1 to all of its buffer slots before any path has been sampled. For every sampled pixel light, the visibility value in its corresponding buffer slot will be looked up and later multiplied to its illumination contribution on a shading point. A decrement of visibility value takes place after the lookup as an update to the micro linear buffer. Before the visibility is actually multiplied to illumination contribution, we use GLSL's built-in function `clamp` to have negative values become 0 without a branch. As the result, whether a pixel light is occluded or not, its contribution on the shading point is always evaluated. The visibility value stored in micro linear buffer then decides either the contribution or nothing should be accumulated to the shading point. Algorithms 5 and 6 are modified versions of Algorithms 1 and 3 to incorporate with this technique.

Besides attemps to avoid bringing significant slowdown to render time due to the introduction of micro linear buffer, we applied two more approaches to actively push the performance.

First, as mentioned before, deferred shading is adopted to benefit the render time in this pass. Using deferred shading ensures that each pixel in the result image is shaded exactly once and therefore independent of geometric complexity. The second approach is to render indirect illumination in lower resolution (quartered size of final result) and then upsample it in the next render pass as previous work does [11]. Since in general, indirect illumination has a smooth appearance, rendering it in low resolution is usually acceptable when the goal is to produce visually pleasant images in real time. Our implementation shows that these two approaches altogether can improve the performance for nearly three times while preserving the same level of image quality.

Algorithm 5 Sample RSM with minimal branches

$irradiance \Leftarrow 0$
for each sampling direction θ **do**
  for each sampling radius r along a path **do**
    Slightly jitter θ.
    Slightly scale and jitter r.
    Transform (θ, r) to Cartesian coordinate (u, v).
    $proj_p \Leftarrow p$'s projection on RSM
    $samplePosition \Leftarrow proj_p + (u, v)$
    if $samplePosition$ is inside the RSM **then**
      Retreive pixel light p on RSM at position $samplePosition$.
      Calculate p's visibility ν to the shading point.
      Clamp ν to $[0, 1]$.
      Calculate weight w of sample p.
      Calculate irradiance contribution E_p due to p with Eq. 3.2.
      $irradiance \Leftarrow irradiance + \nu \cdot w \cdot E_p$
    end if
  end for
end for
return $irradiance$

Algorithm 6 Approximate visibility with minimal branches

Require: Micro linear buffer is initialized as 1 in all its buffer slots.
Calculate elevation angle $\phi \in [-\pi/2, \pi/2]$.
Map ϕ to the index idx of micro linear buffer $buffers$.
if $idx = idx_{-1}$ **then**
  $\nu \Leftarrow \nu_{-1}$
else
  $\nu \Leftarrow buffers[idx]$
  $buffers[idx] \Leftarrow buffers[idx] - 1$
end if
return ν

Figure 4.4: Layout of our indirect illumination buffer. Note that the depth is linearized.

We output the gathered irradiance of all shading points to the indirect illumination buffer. Linearized depth is also written to the last component of the render buffer-bound texture (Figure 4.4). This is because that the next render pass will use depth values as input to an edge-aware blur. Although the depth values are already stored in one of the textures composing G-buffer, we make a copy of depth values from the G-buffer and pack it with irradiance into the indirect illumination buffer, which consists of only one texture, therefore elimating the need to fetch another texture in G-buffer for the same information.

4.5 Post Processing

Finally in this pass, data are aggregated to synthesize a final image. The indirect illumination buffer from the previous pass is first upsampled. A subsequent edge-aware blur is then applied in order to remove the high frequency noises. The blur is a variation of bilateral filtering with a kernel sized 17 that depends on both depth difference and normal vector continuity in addition to the distance within kernel. In the end, direct illumination is reconstructed from G-buffer and combined with the indirect illumination to form a final result.

Chapter 5

Results and Discussions

5.1 Environments

We tested our system on an NVIDIA GTX260-based graphics card with 768MB graphics memory and an Intel Core i7-920 CPU. Our test scene was a trianglized version [8] of Crytek's Sponza model [1] which consisted of some 262 thousand triangles. We rendered both RSM and final image in 512×512 resolution, and indirect illumination in 256×256. Average render time of each frame will be used to compare the performance between different parameter settings.

5.2 Results

As previously illustrated, in comparison with the approach of original RSM, relatively fewer samples are sufficient to render aliasing-free approximate indirect illumination with randomly jittered sampling pattern. The unwanted high frequency noises produced by our sampling process can be effectively eliminated by an edge-aware blur in post processing.

Figure 5.1 presents the comparison of our visibility approximation with a non-occlusion version. Gathered irradiance is visualized because it represents the contribution from sampled pixel lights. If, for instance, pixel lights with green color should be occluded to some geometric areas, one can obverse the irradiance lacking some specific channel there. In the figure, directional shadows' presence in certain areas can be easily noticed and appears more realistic than the non-occlusion counterpart. We also provide a reference image gen-

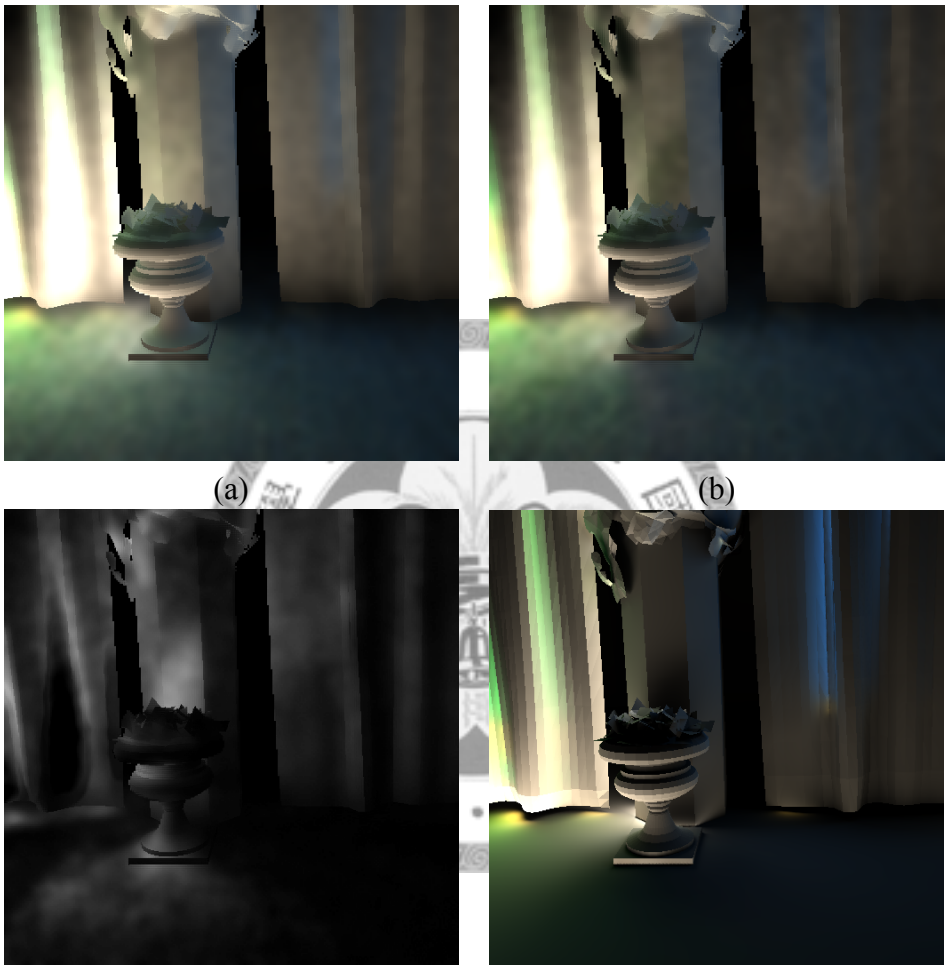


Figure 5.1: Comparison of indirect irradiance with and without visibility approximation. (a) Result without visibility approximation. (b) Result with our visibility approximation algorithm. (c) Amplified difference of luminance of (a) and (b). (d) Reference image generated by gathering all pixel lights on RSM with visibility determination using ray casting.

erated on CPU in which all pixel lights on RSM are gathered for every shading point, with visibility determination by ray casting. The shapes and locations of soft indirect shadows rendered by our approach generally match those in the reference image.

Three more sets of results are shown in Figures 5.2, 5.3, and 5.4. They are all parts of the Sponza scene model with different lighting and camera views. Difference images in Figure 5.5 present areas where soft indirct shadows appear.

5.3 Performance

Table 5.1 lists the performance of our system with different parameter settings. Although introducing the micro linear buffer consumes more graphics memory and adds an *if-else* branch, the overhead of visibility approximation takes merely 13% of the render time without occlusion in the three test cases.

The performance gain from lower resolution of indirect illumination is shown in Table 5.2. It is clear that the render time has dropped dramatically. In addition, the overhead of visibility approximation increases with the resolution from 13% of render time in 256×256 to 17% in 512×512. Higher resolution of indirect illumination are still possible for real-time application with future generations of graphics hardware to provide even better image quality.

Table 5.3 further shows the fraction of render time spent on each render pass. The first two passes, RSM and G-buffer generation, take the whole scene as input. In spite of our test scene consists of 252 thousand triangles which can be considered large and complex, the results have illustrated the capability of modern GPUs to rapidly render complex geometry. Due to our deferred shading pipeline, the other passes do not depend on the scene complexity. We also experimented the case in which deferred shading was not incorporated with, and the frame rate would drop from around 30 fps to approximately 20 fps or sometimes even worse. In addition, to better eliminate the high frequency noises, a relatively large kernel size of 17 is used in post processing, which results in a noticeable, yet acceptable, render time roughly a quarter of the total amount.

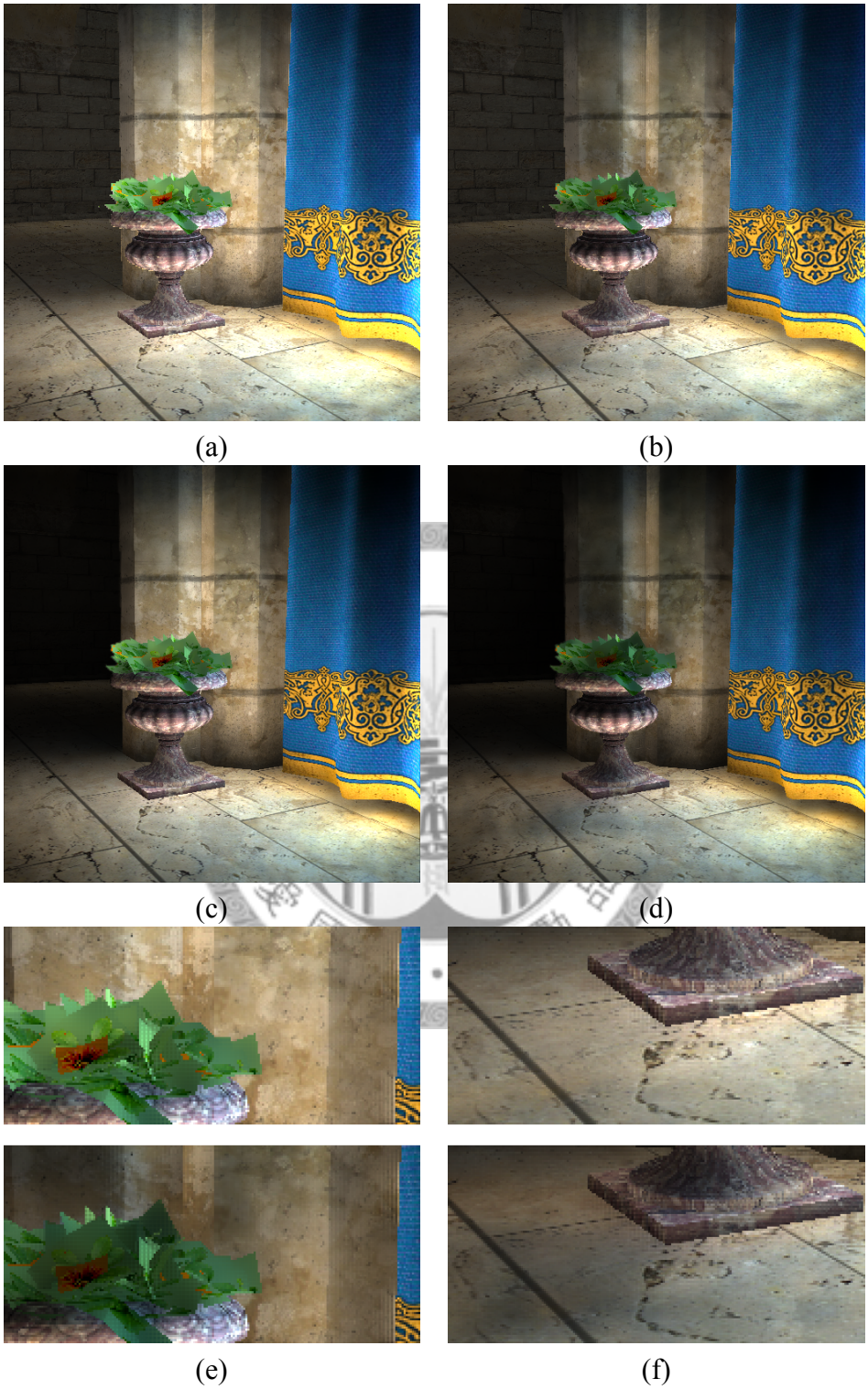


Figure 5.2: Test case A. (a) Global illumination without indirect shadows. (b) Global illumination with our visibility approximation for indirect lights. (c) Indirect illumination only, without visibility test. (d) Indirect illumination only, with our visibility approximation. (e) and (f) are enlarged regions of (c) and (d), where on the bottom are results of our visibility approximation.

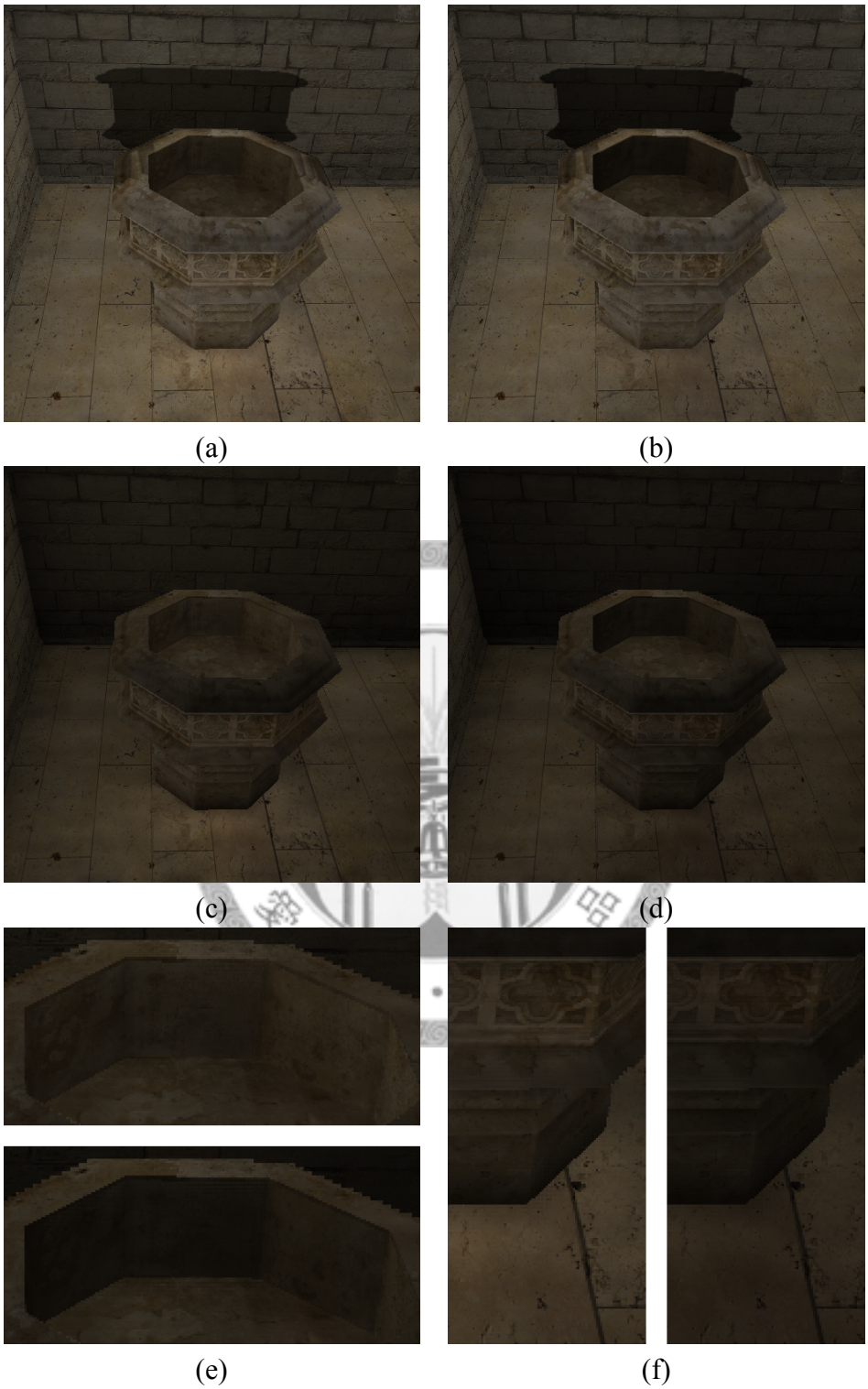


Figure 5.3: Test case B. (a) Global illumination without indirect shadows. (b) Global illumination with our visibility approximation for indirect lights. (c) Indirect illumination only, without visibility test. (d) Indirect illumination only, with our visibility approximation. (e) Enlarged regions of (c) and (d), where on the bottom is the result of our visibility approximation. (f) Enlarged regions of (c) and (d), where on the right is the result of our visibility approximation.

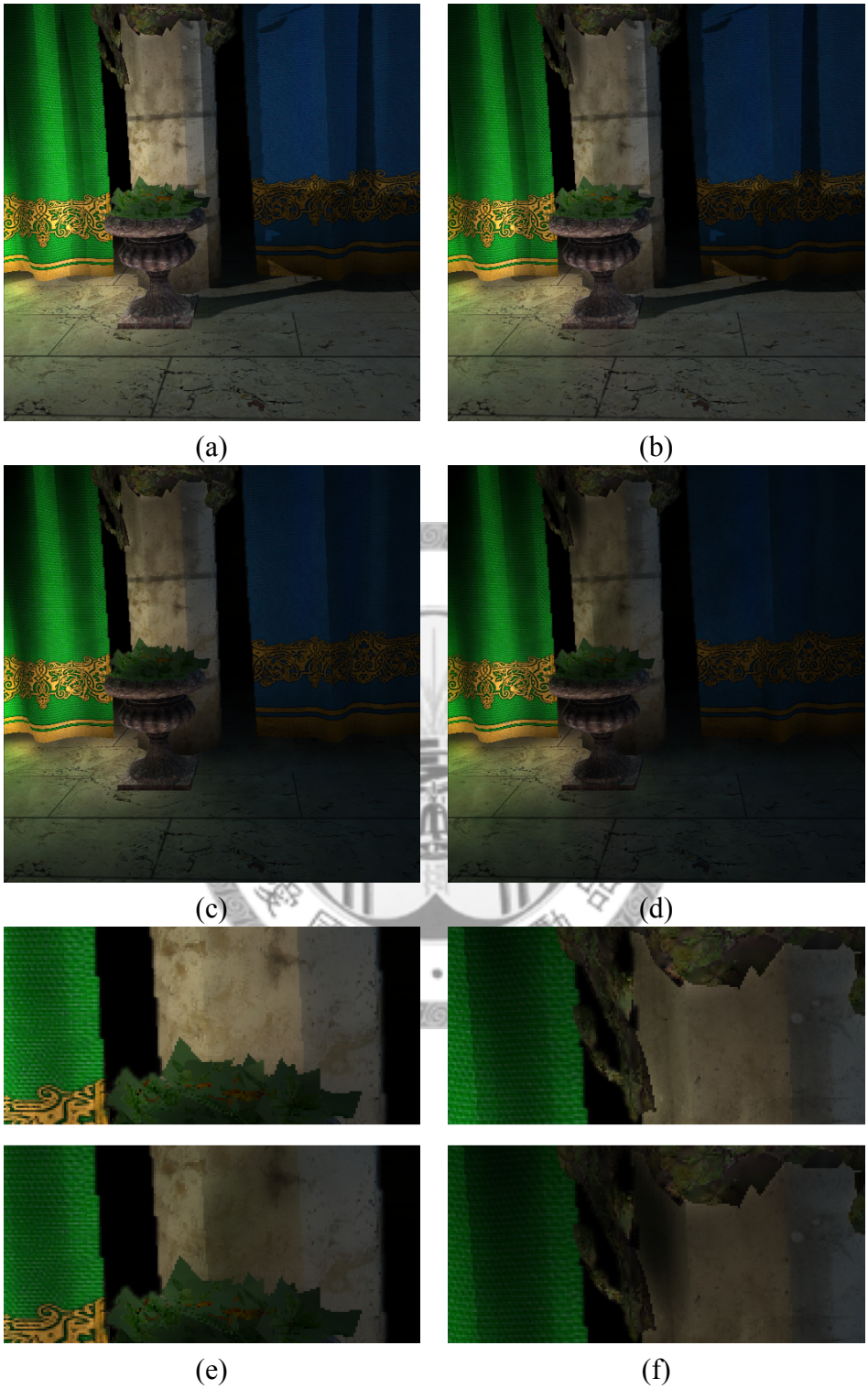


Figure 5.4: Test case C. (a) Global illumination without indirect shadows. (b) Global illumination with our visibility approximation for indirect lights. (c) Indirect illumination only, without visibility test. (d) Indirect illumination only, with our visibility approximation. (e) and (f) are enlarged regions of (c) and (d), where on the bottom are results of our visibility approximation.

Figure 5.5: Amplified difference between results with and without visibility approximation. From left to right: difference images of results from test cases A, B, and C.

Table 5.1: Average render time with and without visibility approximation.

Test case	Render time without visibility approximation	Render time with visibility approximation	Time increasement
A	31.6 ms (32.1 fps)	35.9 ms (27.9 fps)	14%
B	32.4 ms (30.9 fps)	36.5 ms (27.4 fps)	13%
C	33.0 ms (30.3 fps)	37.3 ms (26.8 fps)	13%

Table 5.2: Timing for different indirect illumination resolution in test case A.

Visibility approximation	Indirect illumination resolution	Render time
No	256×256	31.6 ms (32.1 fps)
Yes	256×256	35.9 ms (27.9 fps)
No	512×512	79.8 ms (12.5 fps)
Yes	512×512	93.4 ms (10.8 fps)

Table 5.3: Timing for different render passes in test case A.

Render pass	Render time	Percentage
RSM generation	2.03 ms	5.7%
G-buffer generation	2.47 ms	6.9%
RSM sampling	21.8 ms	60.7%
Post processing	9.60 ms	26.7%
Total	35.9 ms	100.0%


國立臺灣大學
學術倫理・愛國愛人

Chapter 6

Conclusions

We improved RSM by integrating visibility approximation for indirect illumination to the process of sampling. The main contributions of our work include a new sampling pattern and a visibility approximation approach using micro linear buffer. Our results have shown that light leakage problem produced by many previous RSM-based works has been relieved. Besides, our algorithm produces soft indirect shadows that make the global illumination result more realistic without sacrificing advantages of the original RSM method. In other words, we can obtain better quality without compromise of high performance. Moreover, a GPU-based implementation and performance analysis are also presented in this thesis. We believe our performance-tuning techniques can be widely adopted to other GPU applications where render budget for each frame is of major concern. Finally, our approach is fully image-based and dynamic. It requires no precomputation of neither the scene nor the lights, and thus it is appropriate for real-time application like games or interactive designing tools.


國立臺灣大學
人格教育・學生愛國

Bibliography

[1] M. Dabrovic and F. Meinl. Crytek sponza model. http://www.crytek.com/cryengine/cryengine3/downloads.

[2] C. Dachsbacher and M. 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.

[3] C. Dachsbacher and M. Stamminger. Splatting indirect illumination. In *Proceedings of the 2006 symposium on Interactive 3D graphics and games*, I3D '06, pages 93--100, New York, NY, USA, 2006. ACM.

[4] C. Dachsbacher, M. Stamminger, G. Drettakis, and F. Durand. Implicit visibility and antiradiance for interactive global illumination. *ACM Trans. Graph.*, 26, July 2007.

[5] Z. Dong, T. Grosch, T. Ritschel, J. Kautz, and H.-P. Seidel. Real-time indirect illumination with clustered visibility. In *Proceedings of Vision, Modeling, and Visualization Workshop*, 2009.

[6] A. Kaplanyan and C. 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.

[7] D. Kirk and W. mei Hwu. *Programming Massively Parallel Processors: A Hands-on Approach*. Morgan Kaufmann, Februrary 2010.

[8] A. Kirsch. A trianglized version of the crytek sponza model. http://blog.blackhc.net/2010/07/light-propagation-volumes/.

[9] G. Nichols, J. Shopf, and C. Wyman. Hierarchical image-space radiosity for interactive global illumination. *Computer Graphics Forum (Proceedings of EGSR)*, 28, June 2009.

[10] G. Nichols and C. Wyman. Interactive indirect illumination using adaptive multiresolution splatting. *IEEE Transactions on Visualization and Computer Graphics*, 16, September 2010.

[11] 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.*, 27:129:1--129:8, December 2008.

[12] T. Ritschel, T. Grosch, and H.-P. Seidel. Approximating Dynamic Global Illumination in Screen Space. In *Proceedings ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, 2009.

[13] M. Segal and K. Akeley. *The OpenGL Graphics System: A Specification (Version 3.1)*. Khronos Group Inc., March 2009.

[14] P. Shanmugam and O. Arikan. Hardware accelerated ambient occlusion techniques on gpus. In *Proceedings of the 2007 symposium on Interactive 3D graphics and games*, I3D '07, pages 73--80, New York, NY, USA, 2007. ACM.

[15] W. A. Stokes, J. A. Ferwerda, B. Walter, and D. P. Greenberg. Perceptual illumination components: a new approach to efficient, high quality global illumination rendering. *ACM Trans. Graph.*, 23:742--749, August 2004.

[16] S. Thiedemann, N. Henrich, T. Grosch, and S. Müller. Voxel-based global illumination. In *Symposium on Interactive 3D Graphics and Games*, I3D '11, pages 103--110, New York, NY, USA, 2011. ACM.