Interactive Screenspace Fragment Rendering for Direct Illumination from Area Lights Using Gradient Aware Subdivision and Radial Basis Function Interpolation

Ming Di Koa

Nanyang Technological University, Singapore

Henry Johan

Fraunhofer IDM@NTU, Singapore

Alexei Sourin

Nanyang Technological University, Singapore

Abstract

Interactive rendering of direct illumination from area lights in virtual worlds has always proven to be challenging. In this paper, we propose a deferred multi resolution approach for rendering direct illumination from area lights. Our approach subdivides the screenspace into multi resolution 2D-fragments in which higher resolution fragments are generated and placed in regions with geometric, depth and visibility-to-light discontinuities. Compared to former techniques that use inter-fragment binary visibility test, our intra-fragment technique is able to detect shadow more efficiently while using fewer fragments. We also make use of gradient information across our binary visibility tests to further allocate higher resolution fragments to regions with larger visibility discontinuities. Our technique utilizes the stream-compaction feature of the transform feedback shader (TFS) in the graphics shading pipeline to filter out fragments in multiple streams for soft shadow refinement. The bindless texture extension in graphics pipeline allows us to easily process all these generated fragments in an unsorted manner. A single pass screenspace irradiance upsampling scheme which uses radial basis functions (RBF) with an adaptive variance scaling factor is proposed for interpolating the generated fragments. This reduces artifacts caused by large fragments and it also requires fewer fragments to produce reasonable results. Our technique does not require precomputations and is able to render diffuse materials at interactive rates.

Keywords: Area Lights, Interactive Rendering, Soft Shadows

1. Introduction

Interactive rendering of direct illumination from area lights has often been constraint by the integration of the visibility function and radiance over the light surfaces. Direct illumination from area lights produces varying illuminated regions. These effects are usually visible as soft shadows. A complex scene with multiple objects of complex geometry usually requires a large amount of visibility samples to produce a noise-free image if an area light is present. These illumination effects are essential for realism in virtual worlds.

Several methods have been developed to render direct illumination from area lights. Monte Carlo approaches with distributed ray tracing can be used by taking numerous shadow rays per pixel, restricting the rays to the solid angle extended by the lights. There exist real-time methods such as variance shadow maps (VSM) [1] and convolution shadow maps (CSM) [2], that avoid the computation overheads of Monte Carlo methods. Nevertheless, these methods approximate visibility by blurring edges in shadow maps, which only produces a rough approximation of visibility. This rough approxi-

*Ming Di Koa
Email address: mdkoa1@e.ntu.edu.sg (Ming Di Koa)

mation would not be sufficient for realism as they tend to produce overly smooth shadows edges. In our work, we rely on point sampling on the light source for more accurate results.

Multi resolution rendering [3] is an effective adaptive sampling method for reducing samples in screenspace. While standard Monte Carlo approaches [4] and distributed ray tracing techniques [5] emphasize on concentrating more samples and rays on difficult regions to allow an estimated value to converge per pixel, multi resolution rendering instead focuses on finding ways to share and re-use information between large areas of pixels. Its concept is similar to irradiance caching [6], in which samples in low-varying illuminated regions can be re-used for computing illumination information for areas without samples. However, existing multi resolution screenspace techniques are overly conservative which cause them to generate excessive fragments and visibility tests.

In this paper, we aim to handle dynamic lights and viewpoints while interactively rendering direct illumination from area lights for diffuse materials. Our technique draws inspiration from the multi resolution approach [3]. We present three main contributions in this work.

- A screenspace sub-fragment visibility test (SFVT) for detecting shadow boundaries. We also propose a gradient-aware soft shadow refinement (GASS) framework, which enables us to accelerate fragment refinement compared to former techniques. This greatly reduces the amount of visibility queries required between each mipmap level as well as reduces the total number of fragments generated compared to previous work.
- A single pass upsampling method that approximates shadow boundaries with scattered samples by radial basis functions (RBF). It is able to produce high quality soft shadow boundaries with a reduced number of fragments.
- A shadow refinement stage that fully utilizes the multiple stream-compaction feature of the graphics pipeline's transform feedback shader (TFS). An efficient bindless image rendering approach has been used to render fragments of different sizes.

This paper is an extended version of a recently published work [7]. Additional comparisons between our work and former multi-resolution technique by Nichols et al.'s [8] are provided in Section 5. We explain the similarities and differences of our work to former multi-resolution techniques in Section 2.2. A much refined single pass upsampling method, which reduces fixed pattern noise from our previous work, is provided in Section 3.6. This single pass upsampling method has a lesser error compared to previous multi resolution approaches. Additional figures on the stream-compaction method are added. Newer improvement in the graphics pipeline, which utilizes bindless texture rendering, have been used to accelerate the fragment generation process which will be described.

2. Related Work

2.1. Shadow Map Based Methods

Standard shadow map techniques [9] can approximate visibility fast and are commonly used in real-time applications. They operate by back-projecting a visible point onto the light viewing plane. Comparisons are made between the projected visible points' depths and the depths stored on the shadow map. However, standard shadow maps are neither able to estimate penumbra regions nor capable of generating soft edges. Fernando [10] proposed percentage-closer soft shadows (PCSS). PCSS gives an approximation for the penumbra size and a filter corresponding to the penumbra size is used to take samples from a specific region on the shadow map. Schwärzler et al. [11] extended the PCSS by re-using visibility values across frames. Annen et al.'s [12] exponential shadow maps (ESM) replaced the binary output in visibility test to one with an exponential function. The visibility function is smoothed with an exponential function, where the exponential function used appears to overly smooth regions even with sharp visibility discontinuities. (VSM) [1] method creates an upperbound for visibility probability, which usually is an exact result when the receiver surface is parallel to the light plane. However, in scenes with high depth complexity, such as having multiple overlapping receivers, high frequency light leaking artifacts can be observed. This is known as the 'non-planar' condition, where the Chebyshev's inequality gives a poor upper bound approximation due to high variance from samples in the filter. Variance soft shadow maps (VSSM) [13] on the other hand, made use of a kernel subdivision scheme, that identifies particular regions in the shadow map that are 'normal' (regions with low variance) and 'non-planar'. VSM can be used for regions that are 'normal', while PCSS works well for regions that are 'non-planar'. Annen et al.'s [2] convolution shadow maps represent the visibility function in Fourier basis functions which allows for filtering to be applied. These shadow maps are able to handle shadows in high depth complexity environments but they only approximate visibilities by blurring areas near the penumbra. They only produce a rough approximation to the pixel visibility and do not take into consideration the nature of the light (e.g, shape of the light, normal of the light). He et al.'s [14] multi-rate shading algorithm detects shadow edges using depth derivatives on shadow map and forces their pipeline to perform shadow calculations for these identified locations at higher resolutions. Their method is similar to our work as it uses a multi resolution approach in finding regions that require sampling at higher resolutions. However, using a single shadow map only limits the sampling on the area light source to a single point due to the perspective projection used. Although we can create multiple shadow maps to represent multi point sampling on area lights, performance issues such as a rise in textured memory and drop in rendering speed are expected.

2.2. Multi Resolution Algorithms

Direct illumination from area lights are known to vary smoothly across flat regions. Coarse sampling techniques, such as multi resolution splatting by Nichols et al. [8, 3, 15, 16], were devised previously to take advantage of this property. Multi resolution splatting proposes to dissect an image into patches known as fragments, where the fragment size depends on the depth, normal and illumination variations within the patch. As illumination variation decreases, the illumination on a fragment can be represented using information from lower resolution fragments which reduces computation time. We improve on the work of multi resolution rendering. In Nichols et al's work in [8], visibility discontinuity is detected by measuring bit differences within a neighbourhood of fragments. We instead choose to focus on discontinuity within the interior of a fragment and use a refinement scheme based on bit gradients which generates fewer fragments at faster rates. The standard multi resolution technique uses multiple passes of upsampling and interpolation which tends to blur out illumination from different layers while our single pass upsampling does not require that. Our sampling method gives greater weights to nearby higher resolution fragments such that texels, which require interpolation, can acquire more accurate values from higher resolution fragments near them. This also reduces artifacts related to the lack of refinement in visibility discontinuities. Though these artifacts are not present in Nichols et al.'s work [8] due to their overly conservative diagonal refinement method, we show that those artifacts are reproducible (Section 3.6) when the number of fragments at these visibility discontinuities are reduced.

2.3. Image Space Sparse Samples Reconstruction

Image space methods perform per-pixel error estimates and allocate more samples to difficult regions using various sampling techniques. A fixed set of samples per pixel is initially used to obtain an error estimate and variance. Rousselle et al. [17] and Li et al. [18] aimed to focus on using bilateral filters to reduce the variance in filtered pixels. Every pixel has a variance associated to it, and it is blurred respectively with a kernel of varying size depending on its variance. Similarly, Mehta et al. [19, 20] and Yan et al. [21] described how to analyze light field based on its frequency domain. The image is later rendered with sparse samples for each pixel and filtered with a shear filter. All the works mentioned above focus on reducing samples per pixels while our approach focuses on reducing samples per fragment. However, their filtering methods are still complementary to ours in smoothing images. Skala [22] reconstructed images with sparsely distributed samples by radial basis functions, however these samples are reconstructed from a uniformly distributed set of samples in a stratified grid pattern and are not targeted at reconstructing illumination transitions.

3. Our Direct Illumination Rendering Pipeline

Figure 1 shows an overview of our deferred shading method for diffuse materials. Our pipeline receives input textures (depth, normal and albedo) from the screenspace deferred shading. A center stage converts these input textures into fragments based on discontinuities in the normals, depth and visibility. The final rendered image is an overlayed result of the deferred shading using direct illumination multiplied by the albedo of visible objects. The red boxes indicate new methods added to the multi resolution pipeline [8].

3.1. Overview

The direct illumination stage starts by generating a multi resolution depth-curvature discontinuity mipmap. This depth-curvature discontinuity mipmap undergoes a thresholding process using a *TFS*, in which fragments that have geometric discontinuities are identified and generated in its relevant mipmap resolution. In our work, we refer to a fragment as a texel unit belonging to a mipmap level. In Figure 1, these fragments are represented as square patches, where the cyan texels represent fragments at the finest resolution. These fragments are transferred into our light culling and shadow refinement processes where they are processed again to detect visibility changes. After the process is completed, we

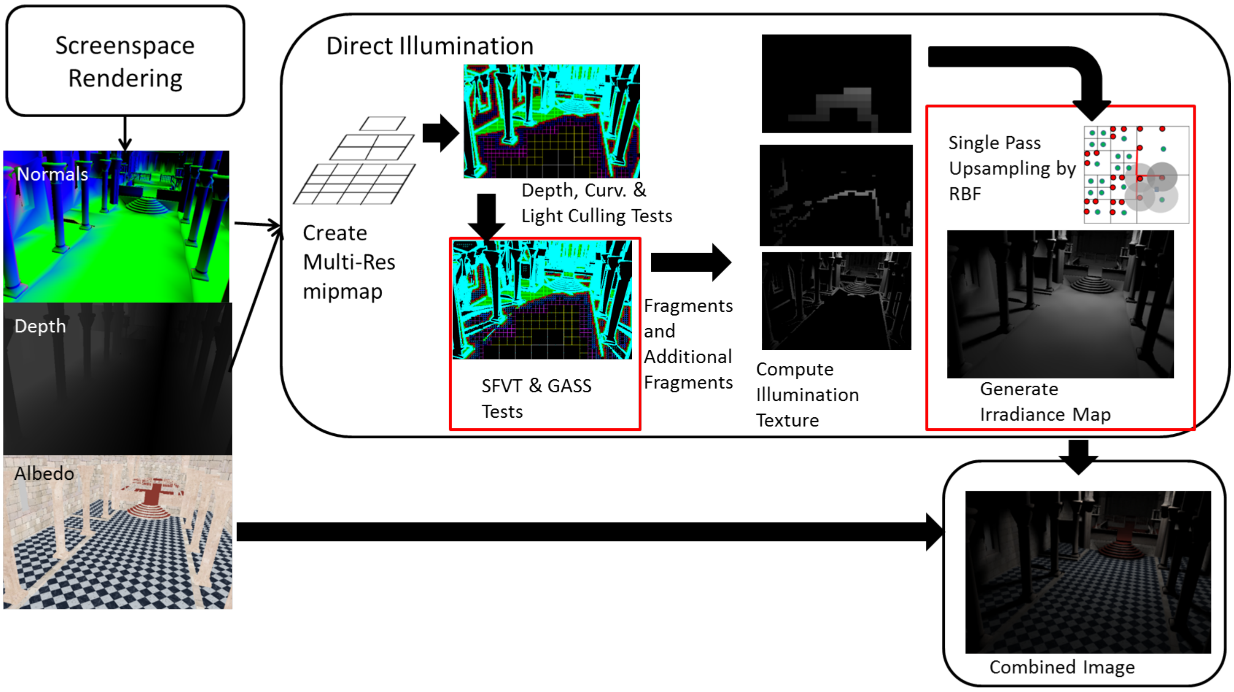


Figure 1: Complete pipeline of our direct illumination with area lights. Boxes in red indicate new proposed stages in the multi resolution framework.

have a set of 'stable' fragments. The irradiance of all 'stable' fragments are computed for different mipmap levels and stored in a multi resolution texture known as the *illumination* texture. This texture is used in a single pass *RBF* interpolation process to approximate texels with no samples in the finest resolution irradiance texture. The irradiance texture is multiplied by the albedo to obtain direct illumination.

3.2. *Geometric Discontinuity and Light Culling*

Here, we give a brief description of the initial fragment refinement stages, Geometric Discontinuity and Light Culling stages which are also described in Nichols et al.'s paper [8].

3.2.1. *Geometric Discontinuity*

Our first stage of multi resolution refinement receives the depth and normal curvature discontinuity mipmaps similar to Nichols et al. [3, 15, 16, 8]. This depth and curvature discontinuity maps are obtained by first rendering a scene as seen from the camera, as well as storing depth, normal and albedo into textures. Next, a depth derivative and normal curvature max-mipmap can be generated by downsampling the depth and normal maps. This is generated from the maximum depth derivative from each of the four finer resolution texels. The normal curvature is computed by $\kappa_x = 2 * sin(arccos(\vec{N} \cdot \vec{N_x})/2)$, where $\vec{N}$ is the normal of the current texel and $\vec{N_x}$ is the normal of the neighbouring texel in the x-direction. The same is done for κ_y in the y-direction. The magnitude of both curvature derivatives are computed by $\sqrt{\kappa_x^2 + \kappa_y^2}$. Fortunately, computing the derivative using neighbouring fragment information is a highly parallel process in the graphics shader pipeline. We currently restrict our depth discontinuity to only refine fragments up to 2x2 pixel size. This will avoid over-refinement caused by glancing camera angles on points far away from the camera.

3.2.2. *Light Culling*

Fragments in screenspace can be culled off easily using the information of the location and orientation of the light. First, we can ignore any fragments on the light surface, since we do not render surface illumination on the light source. Secondly, we can detect geometry that are facing away from the light by testing $\vec{N} \cdot \vec{L_j}$, where $\vec{L_j}$ is the vector from the fragment center to a corner on the light and $\vec{N}$ is the normal of the fragment center. We can discard the fragments if $\vec{N} \cdot \vec{L_j} < 0$ for all j on the light. The light culling step can be performed by ensuring that all fragments produced in the Geometric Discontinuity stage fulfill the front facing light condition.

3.3. *Soft Shadow Refinement*

We illustrate our shadow refinement technique named sub-fragment visibility test (SFVT) in Figure 2. The shadow refinement pipeline retrieves fragments that have passed the depth, curvature and light culling tests. It further performs ray tracing tests to check whether these fragments receive consistent illumination from the area light. We describe the stages of our refinement method in this section.

3.3.1. *Sub-Fragment Visibility Test (SFVT)*

It is important to locate fragments where shadow boundaries are likely to appear. These fragments need further refinement to represent soft shadows. Shadow refinement is performed in Nichols et al. [8] by employing ray tracing to 256 samples (Virtual Point Lights (VPLs)) on the light surface. The visibility to VPLs are stored in a 256 binary bit array, where each bit represents the visibility to a light sample. The ray traced results are compared against their 8 neighbouring fragments in a 3x3 neighbourhood to check for discontinued visibility to the light samples. This requires 9 binary bit arrays to be computed for comparisons. If the visibility bit arrays differ in the neighbourhood, the fragment is further subdivided. For this fragment refinement metric, comparisons are made against neighbouring information outside the fragment instead of information purely within the fragment. This misalignment potentially causes unnecessary subdivisions as well as having potential misses for discontinuities within the fragment.

In our work, instead of comparing the visibility bit arrays with neighbouring fragments, we compare the visibility bit arrays computed from the 4 sub-fragments. This is because our refinement metric should be based on information located on the fragment of interest rather than information located outside of fragment. We refer to sub-fragments as evenly divided points within a fragment that are used for visibility testing. We check if the 4 sub-fragments' visibility arrays differ from each other by a certain threshold. We use a threshold of 2 for small fragments of pixel size 1, 2^2 and 4^2. For larger fragments of 8^2 pixels and above, we use a threshold of 1. This threshold indicates that we flag a discontinuity for approximately 8.25% difference in visibility bits. We use a low threshold, compared to Nichols et al.'s work [8], for a few reasons. Firstly, their work was measuring visibility bit differences across larger distances, while we are measuring across smaller distances. We are expected to have smaller changes in visibility differences compared to theirs. Secondly, we do not have a conservative diagonal refinement criteria like theirs which helps to generate extra fragments on the diagonals. We have to rely on a low threshold to generate these fragments instead. Lastly, we made observations that the chance of a refinement being flagged is low if we only have a small number of VPLs and are using a high threshold. This applies to Nichols et al. [8] work as well. Our proposed method resolves the potential issues caused by misalignments in Nichols et al.'s [8] work.

The number of bit arrays that we need to compute per fragment in Nichols et al.'s work [8] varies from 1 to 9, while it is 4 in our case. Checking for discontinuity within itself also generates lesser fragments as discontinuities tend to be smaller when comparisons are done across smaller distances compared to those of larger distances in neighbouring fragments. We note that it is redundant to further subdivide any fragments at the finest resolution, and hence these fragments should be ignored from the shadow refinement.

This implementation is still too generic if applied to all fragments as larger fragments might require more sample points rather than four. The largest fragment size in our case is at 128x128 pixel resolution. Subdividing the fragment to four sub-fragments of size 64x64 would still be too coarse to detect any visibility change. We instead decide that fragments at mipmap level, m, which are larger or equal to a certain mipmap level N, have to be subdivided into 16 sub-fragments of 2^{m-2} pixel width instead of 4 sub-fragments of 2^{m-1} pixel width. We use N=5, hence only splitting fragments that are 32^2 and above to 16 sub-fragments for visibility testing.

3.3.2. *Ray Generation and Ray Tracing*

We generate K rays from each sub-fragment to random stratified positions on the light source. In our implementation, we use K=16 due to CUDA's efficiency in dealing with threads of warp sizes. Hence, each fragment generates 64 rays from its sub-fragments. In cases where there are 16 sub-fragments, we trace 4 rays from each sub-fragment. This keeps the total number of rays fired to 64 rays per fragment.

3.3.3. *Bit Array Computation*

Our ray tracing produces visibility results between each of the 4 sub-fragments and points on the light. We use the visibility bit array similar to Nichols et al.'s work [8] for each sub-fragment. We label each of our sub-fragments in this section as A, B, C, D (refer to Figure 2). Our main fragment thread counts the bit difference, ray_diff, in visible rays between each of its subdivided fragment. ray_diff can be computed by several OR (|) operations of all XOR ($\oplus$) operation of all

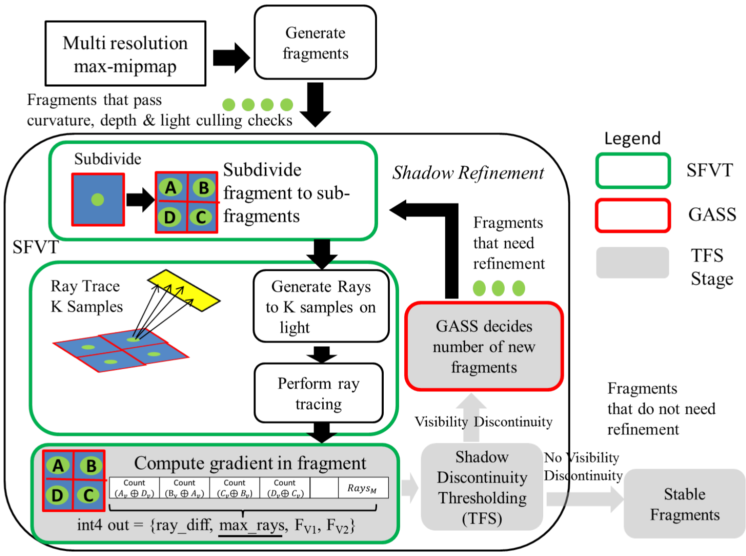


Figure 2: Our shadow refinement pipeline. The pipeline receives fragments (except those of finest resolution) that have passed the depth-curvature tests and light culling check. It performs a sequence of process: Subdivision, Ray Generation, Ray Tracing, Bit Comparison, and finally Shadow Thresholding. The SFVT (green outlined boxes) performs the visibility testing. The number of new fragments generated is decided by its gradient in GASS (red outlined rounded box). Newly generated fragments undergo visibility tests again, while fragments that are 'stable' are transferred out. The filled rounded boxes in gray indicate components of the transform feedback shader (TFS) stage that we use to receive and process fragments as well as stream out fragments.

pair combinations of the sub-fragments's binary visibility array, A_v to D_v, as seen in Equation 1.

$$ray\_diff = (A_v \oplus B_v)|(B_v \oplus C_v)|(C_v \oplus D_v)| \\ (D_v \oplus A_v)|(A_v \oplus D_v)|(B_v \oplus D_v) \quad (1)$$

$$Rays_M = max(Count(A_v), Count(B_v), \\ Count(C_v), Count(D_v)) \quad (2)$$

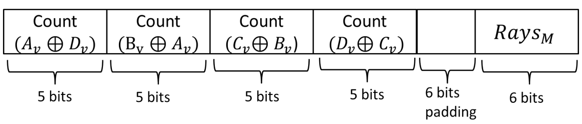


Figure 3: Four sub-fragment gradients are stored into the first 20 bits of the integer *max_rays* and the maximum number of light rays, $Rays_M$, visible among the sub-fragments is stored in the remaining bits.

In addition, we store three additional integers. One integer variable (32 bits), *max_rays*, stores the maximum number of rays ($Rays_M$) (Equation 2) that reach a fragment and gradient information of sub-fragments $(A_v \oplus D_v, B_v \oplus A_v, C_v \oplus B_v, D_v \oplus C_v)$. The gradient information is used in Section 3.3.5. $Rays_M$ can be computed by counting the bits of the sub-fragment with the most binary '1' bits (refer to Equation 2). The function, *Count*, returns the number of '1' bits in a bit array. The 4 gradients are stored inside the first 20 bits of *max_rays*. Figure 3 shows how the gradient information are stored together with $Rays_M$ into the integer, *max_rays*. Another 2 integer variables, F_{v1}, F_{v2}, store the visibility array (64 rays into 64 bits) of the 4 sub-fragments. Figure 2 (gray box with green outline) shows the output from the SFVT. In cases where 16 sub-fragments are used, the ray information (4 rays per sub-fragment) in each of the 16 sub-fragments are accumulated to 4 lower resolution sub-fragments (16 rays each). We use nearest neighbour downsampling for this work. SFVT is performed on the 4 sub-fragments after downsampling. Similarly, the ray information from the 4 lower resolution fragments are stored in integers F_{v1}, F_{v2}.

3.3.4. Discontinuity Thresholding for Soft Shadows

In our work, we use a GPU ray tracer which is meant for processing a point array rather than fragments from a 2D image. We use the standard graphics pipeline's *TFS*, which can generate a compact array of fragments' data which needs ray tracing. We supply a fragment list containing visibility information of its sub-fragments into the *TFS*. The total bit difference, *ray_diff*, is threshold against a user-defined value (we use 1-2 bit difference). If the bit difference is higher than the threshold value, it indicates that the fragment has varying visibility and should be subdivided into four or more fragments and output into a transform feedback stream (see Section 4.3). We note that our shadow refinement process could identify regions near shadow boundaries as visibility discontinuities tend to appear within fragments corresponding to these boundaries.

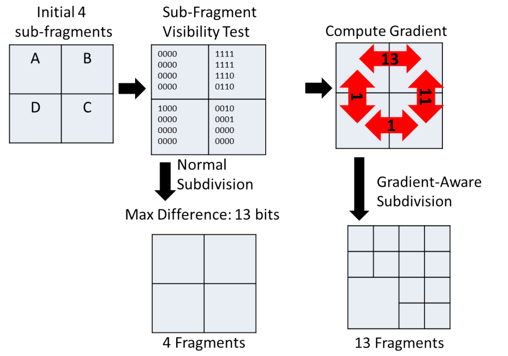
3.3.5. Gradient Aware Soft Shadow Refinement (GASS)

Once the above discontinuity thresholding is done, we can perform an additional refinement process that is able to refine a fragment to a maximum of 16 fragments of higher resolution instead of 4. We note that in some obvious scenarios, such as Figure 4a, a refinement to 4 higher resolution fragments is not sufficient to simulate soft shadow transitions and additional refinement passes are required in the next Transform Feedback pass. These additional refinement passes would generate additional visibility ray queries during each pass. The key idea to reducing unnecessary refinement passes is to identify regions where a single refinement is not sufficient. These can be identified from the areas with high gradients. We first compute the gradients based on the absolute number of visibility bit differences in each sub-fragment along the directions indicated by the red arrows in Figure 4b. The numbers inside the red arrows refer to the absolute gradient between the neighbouring sub-fragments.

If the absolute gradient is higher than the threshold value (we use 7 bit differences), the 2 sub-fragments used in the gradient calculation are refined into 4 higher resolution fragments each. This is equivalent to being refined 2 levels finer than the original fragment. For small gradients (below 7 bit differences), the sub-fragment only produces a single fragment. While it is arguable that we should use information within the fragment to decide for the second level of refinement, the GASS basically skips the need for this extra information by predicting fragments' configurations ahead by one level. This enables us to refine down two mipmap levels instead of one, which in turn reduces unnecessary visibility tests that are usually required in-between. These four gradient information of the four sub-fragments can be easily stored into the first 20 bits of the integer (*max_rays*), where each gradient data uses 5 bits for its magnitude.

3.3.6. Dark Region Culling

Fragments that are completely occluded from the light, based on $Rays_M$, can be removed from further refinement because they do not contain illumination. We can directly write the zero color value with an alpha bit set into the illumination texture at the same mipmap level as the fragment. This indicates that the fragment in the illumination texture still has valid information for upsampling and interpolation.



(a) Normal Subdivision (b) Gradient Aware Soft Shadow Refinement

Figure 4: In the normal subdivision scheme, subdivision is performed whenever the total number of bit differences between the 4 sub-fragments exceeds a threshold. This high discontinuity is determined by the total bit difference within a fragment. (a) The original subdivision scheme only refines a fragment to its next level of mipmap, which is insufficient and requires additional refinement in the next pass. (b) In our work, we subdivide each sub-fragment with high gradients into 2x2 sub fragments. The refinement criteria is described in the *GASS* refinement scheme.

3.4. Additional Fragments Generation

The single sample location in a large 'stable' fragment center may still miss regions with thin shadows. Hence, additional fragments are added to the refinement. Placing additional fragments along the edges of each fragment also resolves interpolation/extrapolation issues. We generate three additional fragments with a specific pattern to maximize coverage using a minimum set of samples. These three fragments of mipmap level 0 are positioned at the top-center, top-left and left-center of the fragment. Additional fragments are only generated on fragments of mipmap level greater than 2. Figure 5 shows the placement of these new fragments (highlighted in cyan). The green region describes the 'stable' region that is defined by the soft shadow discontinuity thresholding process. For a scene shown in Figure 6, only additional 2100 fragments were added on top of the previous 50k fragments, in which their additional computation time in the final render are negligible.

3.5. Screenspace Irradiance Computation

The ray intersection visibility information obtained from ray tracing can be re-used directly to compute irra-

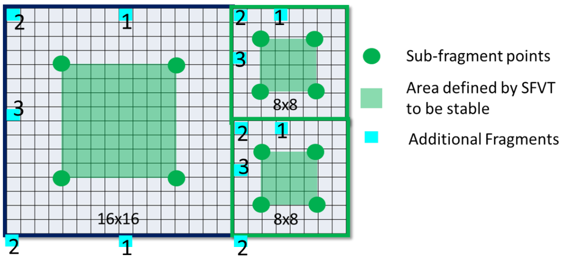


Figure 5: Three fragments (in red) are positioned in the 16x16 and 8x8 sized fragments. Their location enables us to use radial basis functions to compute fragment values in between. Texels in green indicate the regions with little visibility discontinuity during SFVT.

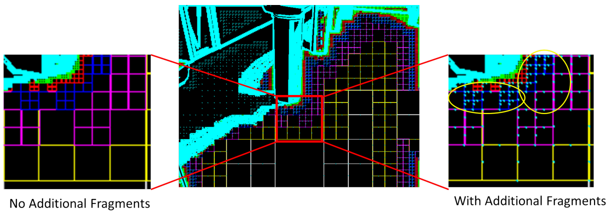


Figure 6: Image in the center shows a fragment map of the Sponza Scene after visibility discontinuities have been done. Image on the left shows a zoomed in version of the fragment map of the center image. The image on the right shows additional fragments, particularly more obvious in the yellow circle (seen as small cyan dots), that are generated to improve the accuracy of interpolating larger fragments.

diance for fragments between mipmap level 1 to N_{max}, where N_{max} is the lowest resolution fragment mipmap level. For fragments in mipmap level 0, rays need to be generated from the point to a stratified sampled position on the light to compute visibility. The irradiance at point x_i can be computed with the ray information from K rays as follows:

$$L(x_i, \vec{\omega}) = \frac{1}{\pi} \sum_{1}^{K} \frac{(\vec{N} \cdot \vec{L_k}) * (\vec{N}_{light} \cdot -\vec{L_k}) * A * I_{intensity}}{|x_i - x_{k\_light}|^2}, \tag{3}$$

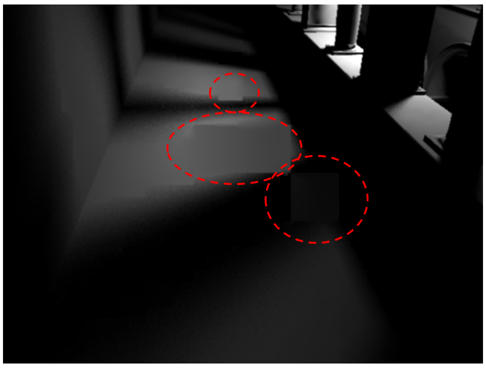
where $\vec{N}$ refers to the normal of point x_i. $\vec{L_k}$ refers to the vector from x_i to a point x_{k_light} on the light. $\vec{N}_{light}$ refers to the normal direction of the light. $I_{intensity}$ refers to the intensity at x_{k_light} with area A. This irradiance computation is similar to that in distributed ray tracing. We do not include the bi-directional reflectance distribution function (BRDF) as intensity might not change smoothly when there are BRDF differences between neighbouring fragments. We instead multiply the final irradiance texture with an albedo term when we render the final image.

3.6. Screenspace Single Pass Upsampling

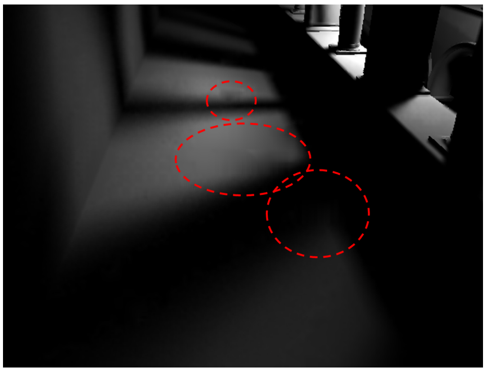
After generating fragments and computing their irradiance information in an illumination texture, the irradiance values of various fragment sizes are combined into a full image at its finest resolution, known as the irradiance texture. In the previous work by Nichols et al. [8], a multi pass upsampling algorithm is used, which performs bilinear interpolation upsampling and addition of illumination information for each mipmap level starting from the coarsest resolution. The multi pass algorithm gives too much influence to fragments from lower resolution. This is mainly because in each pyramid upsampling step, only information from lower resolutions can be obtained. This usually leads to artifacts seen in Figure 7a, as fragments from lower resolution may have errors propagating to the higher resolution fragments. These errors can be reduced by using their conservative diagonal refinement criteria [16] which generates excessive fragments near such visibility discontinuities. However, if these refinements were to be missed in their refinement stage, as seen in the fragment map in Figure 7c, these artifacts are expected to be observed. In our proposed single pass upsampling, we were able to properly reduce the impact of those errors (Figure 7b) while using the same set of fragments as Nichols et al. [8]. This is done by using radial basis functions (RBF) to interpolate fragments' value.

Our single pass algorithm works by processing a full resolution texture, generating a fragment thread for each texel. In this section, we refer to the texel of the final irradiance texture as a *target texel*. If the target texel has illumination from mipmap level 0 to 2, we perform a direct copying of texel value from the illumination texture into the target texel. Subsequently, if the target fragment is from mipmap level 3 and above, we perform a boundary search as defined by the nearest two edges based on the quadrant that the target fragment falls in (Figure 9b). We only record down information of the nearest neighbouring fragment for each colored edge. We always use the additional fragments that were generated previously if they are closer to the target fragment than the neighbour texel's center. The chosen neighbouring sample must be also within a distance of less than two times the texel size of the target fragment's mipmap level.

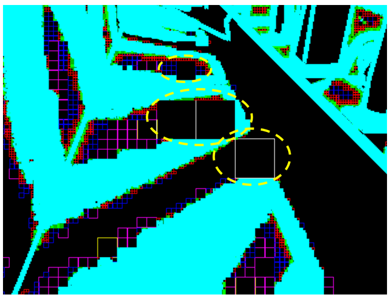
Since our sample data are scattered, we use scattered data interpolation techniques [23]. We employ Gaussian RBF as they provide a naturally smoothing function for interpolating scattered samples. Any two neighbouring texels or internal additional samples with the 2D center position of the target texel form a group of scattered samples, texel $\mathbf{x_i}$, needed for our basis functions Φ_i in Equation 4. To obtain an estimated irradiance value $\hat{I}(\mathbf{x})$



(a) Nichols et al. [8] multi pass upsampling



(b) Our single pass upsampling



(c) Fragment map

Figure 7: (a) Artifacts (dotted red ellipse) in the multi pass upsampling algorithm. Larger fragments are able to dominate pixel values and ignore smaller neighbouring fragments despite being less accurate. This artifacts appear as small holes or spikes near shadow boundaries. (b) Our single pass algorithm reduces these artifacts (not completely) by giving more weights to smaller fragments. (c) Both images in (a)(b) are rendered with the same fragment map based on Nichols et al. refinement [8]. As observed by the yellow dotted ellipse, sometimes a large fragment may fail to be refined.

(a) Adaptive Variance Scaling Factor

(b) Fixed Variance Scaling Factor [7]

Figure 8: Comparisons in per pixel square error with a Monte Carlo reference image of the Sponza scene (1280x960 pixels, 64 samples per fragment). Red color channel image on the right signifies the per pixel difference. The same fragment refinement and number of fragments are used for both renders. The errors have been scaled up to 9x for easier visualization. (a) Our adaptive variance scaling factor approach results in siginificantly smaller sum of square errors (1.112) than (7.482) our previous fixed variance scaling approach [7] in (b).

at target texel $\mathbf{x}$, we first need to evaluate weight w_i for each basis function Φ_i. This can be done by solving the linear Equation 5a, where $\mathbf{w}$ is a vector of weights w_i and $\mathbf{\Phi}$ is a correlation/distance matrix consisting of i rows and j columns of Φ. $\mathbf{I}$ is a vector consisting of irradiance values from the chosen samples.

$$\hat{I}(\mathbf{x}) = \sum_{i}^{3} w_i \Phi_i(||\mathbf{x} - \mathbf{x_i}||), \tag{4}$$

where

$$\mathbf{w} = \mathbf{\Phi}^{-1} * \mathbf{I}, \tag{5a}$$

$$\Phi_{ij} = \exp\left(-d^2/C\right) \tag{5b}$$

In Equation 5b, d is the L2 distance in texels between the chosen sample location, $\mathbf{x_i}$, and the target fragment $\mathbf{x}$. We note that d was divided by the mipmap width of the target texel in our previous work [7]. This scaling factor enabled samples from coarser resolution to have higher variance, and those of finer resolution to have lower variance. Although, this approach has solved some artifacts related to the previous multi pass upsampling method, it led to fixed pattern artifacts in some parts of our results as seen in the error map in Figure 8b (right). For large fragments, this inversely large scaling factor would cause nearby high resolution samples to have no significant difference in weights. In the worst case, the correlation matrix ,$\mathbf{\Phi}$, will be singular. We introduce a varying scaling factor, C, for fine tuning the variance in Equation 5b based on the samples chosen. We describe the computation for C in Equation 6 and also in the next paragraph.

$$C = 2 * (\hat{M} + 1) \tag{6}$$

$$\alpha_i = \exp\left(-\hat{d}^2/(2 * t_{min})\right) \tag{7}$$

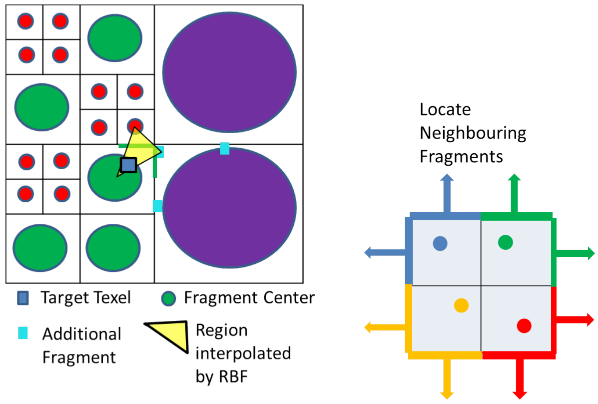
$$\hat{M} = \frac{\sum \alpha_i * M_i}{\sum \alpha_i} \tag{8}$$

Firstly, for a chosen set of samples for a target texel, we find the smallest fragment width, t_{min} (in texels), among the samples and use them as a variance scaling factor for our Gaussian weights in Equation 7. The value $\hat{d}$ is the normalized L2 distance between a sample i and the target texel. Next, we compute a Gaussian weight, α_i, and retrieve the mipmap level, M_i, for each sample and compute a normalized weighted sum to get a distance weighted mipmap level $\hat{M}$. This term is used to compute our variance scaling factor, C. An additional of one added to $\hat{M}$ prevents the division by zero error. Intuitively, Equation 8 conveys that a high resolution sample among the chosen set would reduce the variance. Hence, if a target texel is located in a large fragment but has several higher resolution samples chosen, it will have a low variance. These higher resolution samples would reduce the impact of the lower resolution fragment.

We use only 3 to 4 RBFs such that the inverse could be easily calculated using the inverse function in the standard shader pipeline. In Figure 9a, we show the region interpolated by our RBFs in the yellow triangle for computing the value of a target fragment (in blue). In target texels with less than 3 RBFs, we only need to do weighted interpolation using the function in Equation 5b between 2 samples, $\mathbf{x_1}$ and $\mathbf{x_2}$, which produces the following interpolated value:

$$\hat{I}(\mathbf{x}) = \frac{(\Phi(||\mathbf{x} - \mathbf{x_1}||) * I(\mathbf{x_1}) + \Phi(||\mathbf{x} - \mathbf{x_2}||) * I(\mathbf{x_2}))}{\Phi(||\mathbf{x} - \mathbf{x_1}||) + \Phi(||\mathbf{x} - \mathbf{x_2}||)} \tag{9}$$

We show comparisons with our previous work [7] which uses a fixed variance scaling factor based on fragment size in Figure 8b and our current work which uses an adaptive variance scaling factor based on weighted fragment size in Figure 8a. We exhibit 5x lower sum of square pixel error for this particular image. Further



(a) Our upsampling scheme

(b) Searching for neighbouring fragments

Figure 9: (a) Our upsampling scheme on a target blue fragment. The blue fragment is computed by using radial basis functions selected by samples closest to the two boundaries edge (in green) and its nearest fragment center. The cyan squares refer to the additional fragments that were generated for large fragments. (b) If the target texel falls within the top left of its parent fragment, the two edges in blue are traversed to look for neighbouring samples. Similarly if the fragment falls in the bottom right, the red edges are traversed. The similar can be said for the green and yellow fragments which falls in the top right and bottom left. The arrow indicates the direction to search for neighbouring samples.

comparisons with Nichols et al. and Monte Carlo reference images are presented in Section 5.

4. Implementation

4.1. Depth and Curvature Discontinuity Check

We utilize the stream-compaction feature for the transform feedback shader (*TFS*) in OpenGL 4.0 (also available in DirectX 11). This pipeline allows us to produce four separate arrays for our results. We are able to generate a separate list of 2D fragments that recursively require to be checked for discontinuity in its finer mipmap levels. The input fragments for the shadow refinement stage are also accompanied by their normals and positions. We use the transform feedback shader since it is the fastest parallel processing pipeline to generate a filtered compact stream from an unordered list of inputs.

4.2. Ray Intersection Test

As our input to the shadow refinement stage is in a tightly packed array, we can easily make use of CUDA's GPGPU advantage to subdivide these input sample points and create ray information which are suitable for OptiX Prime ray tracer [24]. CUDA's *shuffle* operations or *SHFL* also makes it easier for us to perform reduction operations such as ray counting or ray summation when generating the visibility array.

4.3. Discontinuity Thresholding - Transform Feedback Shader

Similar to the depth-curvature discontinuity *TFS*, we make use of OpenGL's stream-compaction feature to branch our results in the shadow refinement process. The first stream stores 4 to 16 sub-fragments coordinates depending on how GASS decides. This stream is for transferring fragments that need further visibility testing. The second stream stores the 2D normalized screenspace positions of fragments that are greater than level 0 for those 'stable' fragments. Tagged together with the second stream stores the visibility information of 64 rays using two 32-bit integers, this data can be re-used for computing irradiance. The last stream stores the 2D normalized screenspace positions of fragments that belong to mipmap level 0, the finest resolution. Once the refinement process is completed, we can read the 3D positions and normals from the fragments' normalized 2D screenspace coordinates since screenspace positions and normals are provided in the initial screenspace render. Fragments of level 0 are separated into a different stream because they do not have any visibility information that can be re-used for irradiance computation. They should be stored separately and appended to the remaining fragments. We refer the reader to Figure 10 for the streaming process.

The irradiance of each fragment in stream 2 is computed in CUDA using the visibility information that is also present from the stream. They are then rendered into an illumination texture. We note that there is an implementation difference compared to our previous work [7]. In our previous work [7], we have to sort the fragments into their respective mipmap level such that each fragments can be rendered into their appropriate mipmap level in the framebuffer via multiple passes of rendering. In this work, we use bindless image textures (introduced in OpenGL 4.2 GL_ARB_bindless_texture together with ARB_shader_image_load_store) to write into every mipmap level of the irradiance texture concurrently without attaching any textures to the framebuffer. This approach removes any computation overheads involved in sorting fragments. In Nichols et al. [8]'s method, a flattened texture, which consists of all mipmap levels being appended to a single layer, is used instead. We avoid using their method as we have to recopy the texture to its un-flattened version for more efficient texture reading. We describe our bindless rendering process in Figure 11.

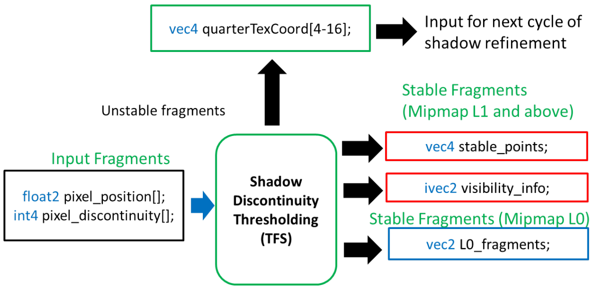


Figure 10: In our transform feedback shader for shadow refinement, three streams of output are produced. The *TFS* receives fragments locations together with its ray tracing information. The first output stream (box in green outline) returns 'unstable' fragments of mipmap level $>$ 1, which should be further refined in the next cycle. Stream 2 (box in red outline) stores corresponding screenspace 2D coordinates and visibility information as 2 output arrays. Stream 3 (box in blue outline) stores screenspace 2D coordinates of level 0 fragments.

5. Results and Discussion

We show our rendering results (Sponza, Sibenik and hairball scenes) in Figures 12a, 13a and 14a. The images are rendered with 64 samples per fragment in 1280x960 resolution with a large majority of samples being re-used from the shadow refinement stage. Table 1 shows the performance and data of the 3D models that we rendered. The rendering was performed on an Intel i5 3.40GHz CPU with a NVIDIA GeForce GTX 980 GPU. The time needed in milliseconds (ms) for the visibility tests and total rendering time are provided in the table. Upsampling takes a fairly little amount of time in all scenarios (1ms). This is mainly due to the fact that it is a screenspace algorithm. We show the Monte Carlo references with 64 samples per pixel in Figures 12c, 13c, 14c while Figures 12d, 13d, 14d show visual representation of the fragments we used, with cyan being the color of the highest resolution fragment and white being the lowest resolution fragment. Our results, particularly shadow boundary regions such as the shadows caused by a large area light in Figure 12a behind the pillars, are similar to our Monte Carlo references.

As seen in Table 1, our timings are 24% to 45% faster and generates 9% to 37% fewer fragments than Nichols et al. [8]. Our single pass upsampling stage also significantly reduces errors in magnitudes lower than Nichols et al.'s multi pass technique (33x to 81x). The improvement in per pixel sum of square error is more evident in scenes with complex geometry. In Figure 12a, our technique directly reduces the number of visibility samples compared to Nichols et al. [8] by 2.8 times. Figures 12d (right), 13d(right), 14d (right) show that the difference in our results (for direct illumination) compared

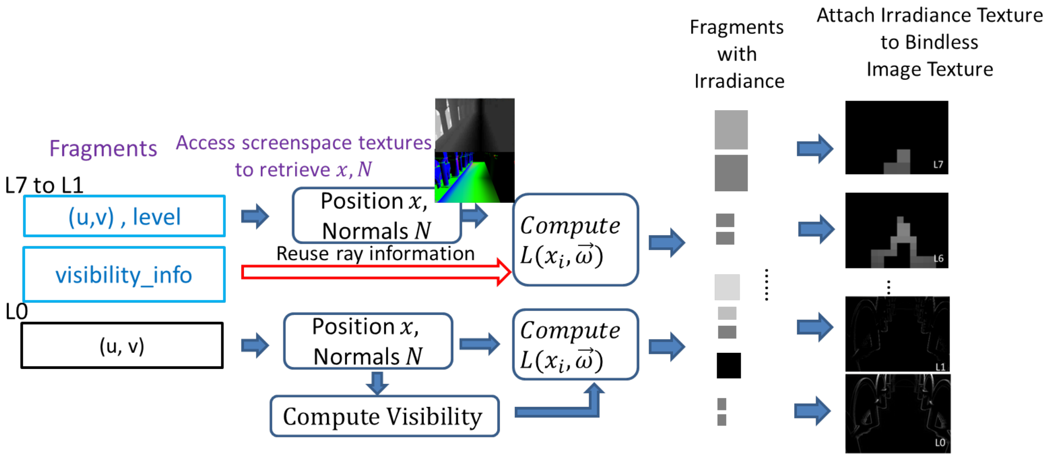


Figure 11: Once our set of 'stable' fragments is retrieved, fragments are converted to their 3D positions, x and normals N. Fragments larger than mipmap level 0 can reuse their visibility information from our SFVT to compute their irradiance based on the formulation for distributed ray tracing. Those at mipmap level 0 will need to perform a visibility test before their irradiance values can be computed. The fragments, with their irradiance values, will then be written to the appropriate mipmap level of textures via bindless image textures.

to the Monte Carlo references is barely visible unless it is scaled. The absolute difference in the pixel values to the reference images is very small unless magnified. Nevertheless, it should be understood that error metrics such as sum of square error scales up with the overall brightness in the scene. Hence we have also provided the normalized sum of square error (NMSE), which is equivalent to the sum of square error normalized by the sum of pixel intensity in the reference image. Similarly, the results from Nichols et al.'s work [8] are shown in Figures 12b, 13b, 14b with their error maps visualized in Figures 12e(right), 13e(right), 14e(right).

The hairball object in Figure 14 is a much more complicated object and we were able to well approximate its fine details and the shadows it generates. For this particular scene, we were able to achieve errors of 80x lesser than that in Nichols et al.'s work [8], mainly because we avoid blurring the geometric details on the surface of the hairball. However in terms of fragments generated, our scheme only reduces it by 9%. This is because the soft shadow regions only take up a small proportion of pixels compared to the entire image. The rendering time for this model is higher compared to that in the Sponza and Sibenik scene despite fewer fragments. This is mainly due to the computational overheads of ray tracing through a complicated mesh of 2.8 million triangles.

Our technique can handle dynamic lighting, dynamic viewpoints and deformable or moving geometry. However, rendering time for moving geometry can be constrained by the number of triangles in the geometry. This is due to the time needed for recontructing the acceleration structure in a ray tracer. A rasterizer ray tracer, which uses a voxel acceleration structure, would perform better in this aspect, but we would need to consider the amount of ray marching in rasterizer ray tracers which would have make ray tracing slower.

The single pass upsampling algorithm, although improved since our previous work [7], still produces certain fixed pattern artifacts. This is mainly due to the artifacts caused by extrapolation in RBF interpolation, as well as the lack of correlation between samples used between neighbouring fragments. Nevertheless, this issue can be solved by adding more samples to the RBF but with performance in consideration, we only used up to 4 samples. Skala [22] proposed using an incremental block matrix method to compute the inverse of the correlation matrix. His method could be used when dealing with more than 4 samples, which we leave for future work. In our single pass upsampling method, we do gain some trade-offs in using lesser texture memory due to the removal of intermediate textures that were formerly needed in the multi pass upsampling approach. Further optimizations, such as approximating depth discontinuity based on the size of the light, can be considered. Artifacts may also be present due to undersampling in the presence of large area lights. For these cases, it is advisable to use more visibility rays. We show further results with planar lights of varying sizes. Although casting 16 rays per sub-fragment is sufficient for these light sizes based on our experiments in Figure 16, noise from undersampling would be expected for larger lights. This

Table 1: Rendering statistics for 1280x960 images. Figures 12a, 13a, 14a show our results, while Figures 12c, 13c, 14c are Monte Carlo references. Nichols et al.'s [8] result are shown in Figures 12b, 13b and 14b. L2 Error refers to the sum of square error when compared to the reference image. The normalized mean square error (NMSE) are also provided in the brackets after the L2 error. NMSE is represented using scientific notation. Each pixel is stored in normalized float format. A lower error indicates a better quality.

Figure	Triangles	Fragments	Visibility Rays	Visibility Test (ms)	Upsampling (ms)	Time (ms)	L2 Error (NMSE)
12a - Ours	66450	327,775	7,817k	44	1	105	12.14 (5.42e-4)
12b - Nichols et. al. [8]	66450	517,778	23,071k	91	4	194	204.1 (7.28e-3)
12c - Monte Carlo	66450	1,228,800	-	-	-	219	-
13a - Ours	75284	380,889	4,143k	28	1	89	7.108 (2.82e-4)
13b - Nichols et. al. [8]	75284	431,637	12,032k	58	4	131	134.3 (5.32e-3)
13c - Monte Carlo	75284	1,228,800	-	-	-	173	-
14a - Ours	2,850,000	223,231	1,820k	57	1	258	18.38 (1.07e-3)
14b - Nichols et. al. [8]	2,850,000	245,864	5,450k	118	4	340	1473 (7.10e-2)
14c - Monte Carlo	2,850,000	1,228,800	-	-	-	484	-

noise can be observed in Figure 15a where the light source is large and 64 samples per fragment is insufficient in removing the noise. However, this noise is unlikely to contribute towards higher error rates. This is because for larger light sources, higher number of fragments are expected to be produced, which in turn, reduces overall error as observed in Figure 16e. We show further results in our 'NMSE vs Light Size' chart in Figure 17 for the Sibenik scene configuration in Figure 13a. The NMSE error rate does not fluctuate much and remains significantly lower than Nichols et al.'s [8] approach. Although we have not demonstrated using textured lighting in our work, it can be done by generating a set of VPLs on the textured light for visibility testing in a similar way to Nichols et al.'s [25]. We can set a limit on the maximum number of VPLs allowed to ensure interactive performance. Similar to Nichols et al's paper [8], we do not refine a fragment based on differences in light energy entering it but on visibility differences, hence the textured appearance of the light has no effect on the rendering performance.

6. Conclusion and Future Work

We have presented a multi resolution approach that is able to render direct illumination efficiently by culling off large portion of unnecessary fragments using our sub-fragment visibility test (SFVT) and gradient aware soft shadow refinement (GASS) techniques. The SFVT scheme performs visibility discontinuity check across a smaller distance and area, and tends to generate less fragments compared to previous conservative methods. Our GASS technique then decides on the number of fragments for refinement. A single pass Gaussian RBF interpolation upsampling approach was proposed to reduce the impacts of shadow artifacts that were visible in the previous multi pass upsampling approach. In addition, our shadow refinement approach was able to fully utilize the streaming architecture of the transform feedback shader as well as the bindless texture extension.

As the next step in our research, we are going to consider incorporating various filtering techniques, as discussed in the related work, which could further reduce the amount of ray samples needed on each fragment. This multi resolution approach can be run orthogonally with many sampling techniques. For example, a screenspace analysis of the variance of each pixel can let us determine the minimum number of samples required for rendering illumination from area lights. This will identify fragments that can be rendered with less than 64 samples. Currently, this work is able to render diffuse materials in a deferred manner. We intend to extend our work to specular or other complex materials in the future. This would mean performing visibility sampling based on the specular cone of the material rather than the solid angle extended by the surface of the light.

Acknowledgments

The research done by Henry Johan is supported by the National Research Foundation, Prime Ministers Office, Singapore under its International Research Centres in Singapore Funding Initiative. The 3D models used in our examples are obtained from: http://graphics.cs.williams.edu/data.

References

[1] W. Donnelly, A. Lauritzen, Variance shadow maps, in: Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games, I3D '06, ACM, 2006, pp. 161–165.

[2] T. Annen, T. Mertens, P. Bekaert, H.-P. Seidel, J. Kautz, Convolution shadow maps, in: Proceedings of the 18th Eurographics Conference on Rendering Techniques, EGSR'07, Eurographics Association, 2007, pp. 51–60.

[3] G. Nichols, J. Shopf, C. Wyman, Hierarchical image-space radiosity for interactive global illumination, in: Proceedings of the Twentieth Eurographics Conference on Rendering, EGSR'09, Eurographics Association, 2009, pp. 1141–1149.

[4] P. Shirley, C. Wang, K. Zimmerman, Monte carlo techniques for direct lighting calculations, ACM Trans. Graph. 15 (1) (1996) 1–36.

[5] R. L. Cook, T. Porter, L. Carpenter, Seminal graphics, ACM, New York, NY, USA, 1998, Ch. Distributed Ray Tracing, pp. 77–85.

[6] G. J. Ward, F. M. Rubinstein, R. D. Clear, A ray tracing solution for diffuse interreflection, in: Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '88, ACM, New York, NY, USA, 1988, pp. 85–92.

[7] M. Koa, H. Johan, A. Sourin, Interactive screenspace stream-compaction fragment rendering of direct illumination from area lights, in: Proceedings of Cyberworlds 2016, International Conference on CyberWorlds '2016, IEEE Computer Society, 2016, pp. 57–64.

[8] G. Nichols, R. Penmatsa, C. Wyman, Interactive, multiresolution image-space rendering for dynamic area lighting, in: Proceedings of the 21st Eurographics Conference on Rendering, EGSR'10, Eurographics Association, 2010, pp. 1279–1288.

[9] L. Williams, Casting curved shadows on curved surfaces, SIGGRAPH Comput. Graph. 12 (3) (1978) 270–274.

[10] R. Fernando, Percentage-closer soft shadows, in: ACM SIGGRAPH 2005 Sketches, SIGGRAPH '05, ACM, 2005.

[11] M. Schwärzler, C. Luksch, D. Scherzer, M. Wimmer, Fast percentage closer soft shadows using temporal coherence, in: Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, I3D '13, ACM, 2013, pp. 79–86.

[12] T. Annen, T. Mertens, H.-P. Seidel, E. Flerackers, J. Kautz, Exponential shadow maps, in: Proceedings of Graphics Interface 2008, GI '08, Canadian Information Processing Society, Toronto, Ont., Canada, Canada, 2008, pp. 155–161.

[13] B. Yang, Z. Dong, J. Feng, H.-P. Seidel, J. Kautz, Variance soft shadow mapping, Computer Graphics Forum 29 (7) (2010) 2127–2134. doi:10.1111/j.1467-8659.2010.01800.x.

[14] Y. He, Y. Gu, K. Fatahalian, Extending the graphics pipeline with adaptive, multi-rate shading, ACM Trans. Graph. 33 (4) (2014) 142:1–142:12.

[15] G. Nichols, C. Wyman, Multiresolution splatting for indirect illumination, in: Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games, I3D '09, ACM, 2009, pp. 83–90.

[16] G. Nichols, C. Wyman, Interactive indirect illumination using adaptive multiresolution splatting, IEEE Transactions on Visualization and Computer Graphics 16 (5) (2010) 729–741.

[17] F. Rousselle, C. Knaus, M. Zwicker, Adaptive rendering with non-local means filtering, ACM Trans. Graph. 31 (6) (2012) 195:1–195:11.

[18] T.-M. Li, Y.-T. Wu, Y.-Y. Chuang, Sure-based optimization for adaptive sampling and reconstruction, ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH Asia 2012) 31 (6) (2012) 186:1–186:9.

[19] S. U. Mehta, B. Wang, R. Ramamoorthi, Axis-aligned filtering for interactive sampled soft shadows, ACM Trans. Graph. 31 (6) (2012) 163:1–163:10.

[20] S. U. Mehta, B. Wang, R. Ramamoorthi, F. Durand, Axis-aligned filtering for interactive physically-based diffuse indirect lighting, ACM Trans. Graph. 32 (4).

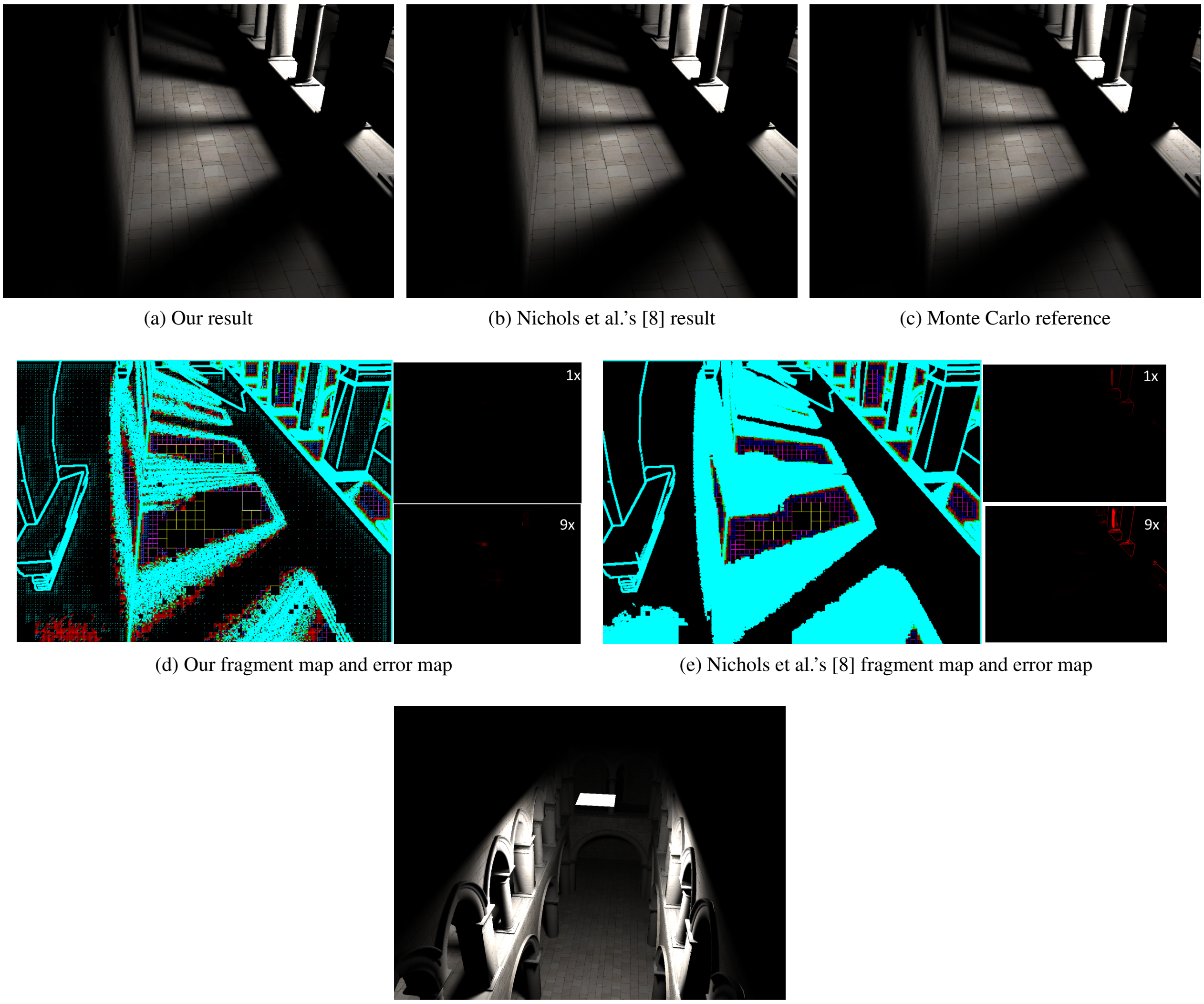
[21] L.-Q. Yan, S. U. Mehta, R. Ramamoorthi, F. Durand, Fast 4d sheared filtering for interactive rendering of distribution effects, ACM Trans. Graph. 35 (1) (2015) 7:1–7:13.

[22] V. Skala, A practical use of radial basis functions interpolation and approximation, in: Proceedings of Revista Investigacion Operacional, Vol. 37 of IWOR '15, 2016, pp. 137–145.

[23] K. Anjyo, J. P. Lewis, F. Pighin, Scattered data interpolation for computer graphics, in: ACM SIGGRAPH 2014 Courses, SIGGRAPH '14, ACM, New York, NY, USA, 2014, pp. 27:1–27:69.

[24] S. G. Parker, J. Bigler, A. Dietrich, H. Friedrich, J. Hoberock, D. Luebke, D. McAllister, M. McGuire, K. Morley, A. Robison, M. Stich, Optix: a general purpose ray tracing engine, ACM Trans. Graph. 29 (4) (2010) 66:1–66:13.

[25] G. Nichols, C. Wyman, Direct illumination from dynamic area lights, in: SIGGRAPH '09: Posters, SIGGRAPH '09, ACM, New York, NY, USA, 2009, pp. 82:1–82:1. doi:10.1145/1599301.1599383.



(a) Our result

(b) Nichols et al.'s [8] result

(c) Monte Carlo reference

(d) Our fragment map and error map

(e) Nichols et al.'s [8] fragment map and error map

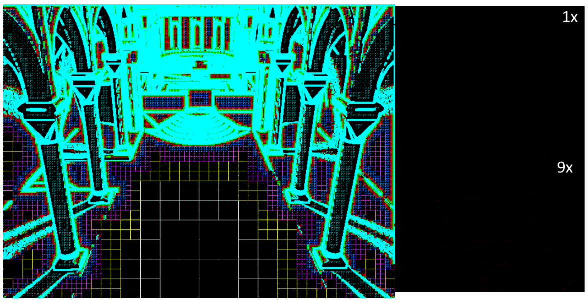
(f) Light size 1 unit

Figure 12: Rendering of the Sponza (McGuire Graphics Data) in 1280x960 pixels with direct illumination. In Figure 12d (right), we show the L2 error map in irradiance values (with scaling factor of 1x and 9x) between Figures 12a and 12c. The error map is mapped to the red color channel. Similarly Figure 12e (right) shows the L2 error map between Figures 12b and the reference image 12c. (f) shows the location and size of our planar light source.

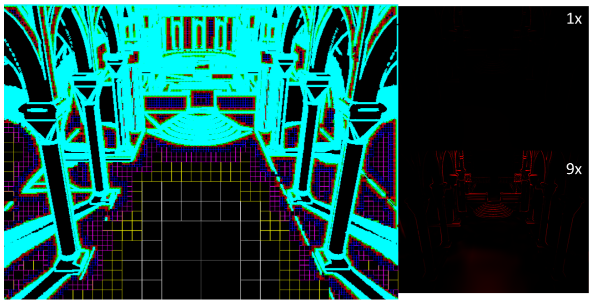
(a) Our result

(b) Nichols et al.'s [8] result

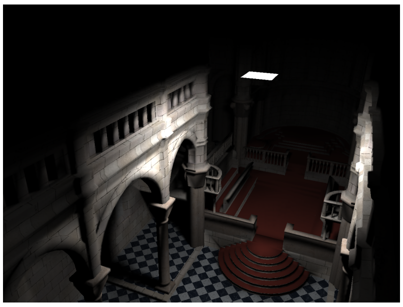
(c) Monte Carlo reference



(d) Our fragment map and error map

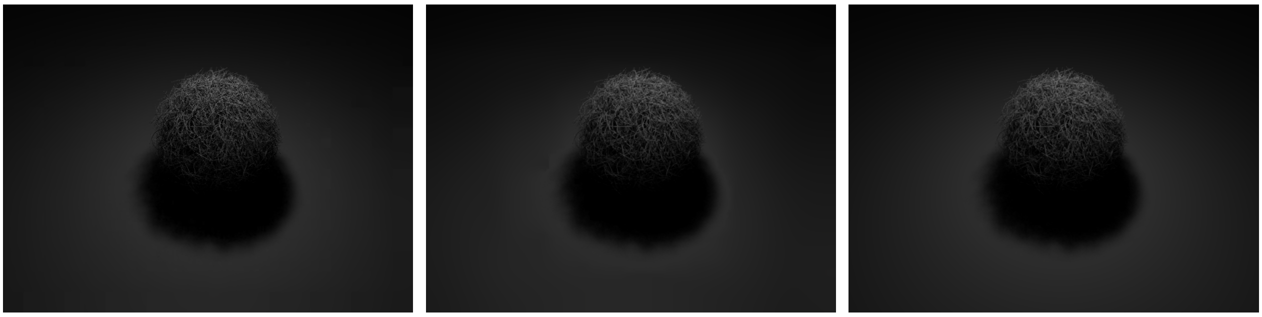


(e) Nichols et al.'s [8] fragment map and error map

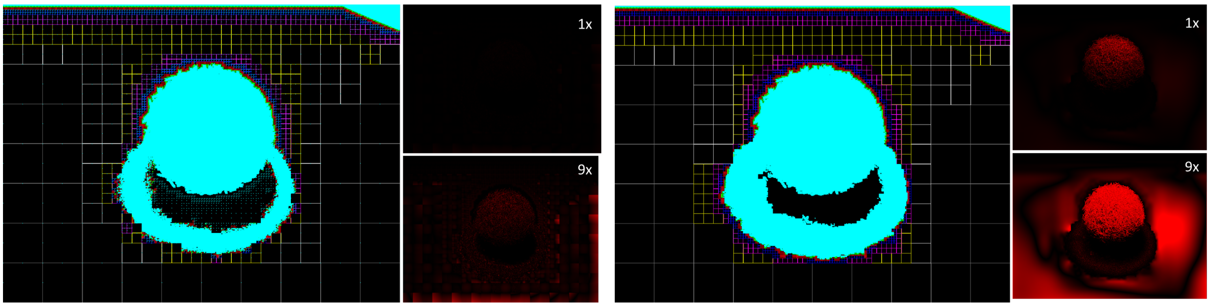


(f) Our light source

Figure 13: Rendering of the Sibenik (McGuire Graphics Data) in 1280x960 pixels with direct illumination. In Figure 13d (right), we show the L2 error map in irradiance values (with scaling factor of 1x and 9x) between Figures 13a and 13c. The error map is mapped to the red color channel. Similarly Figure 13e (right) shows the L2 error map between Figures 13b and the reference image 13c. (f) shows the location and size of our planar light source.

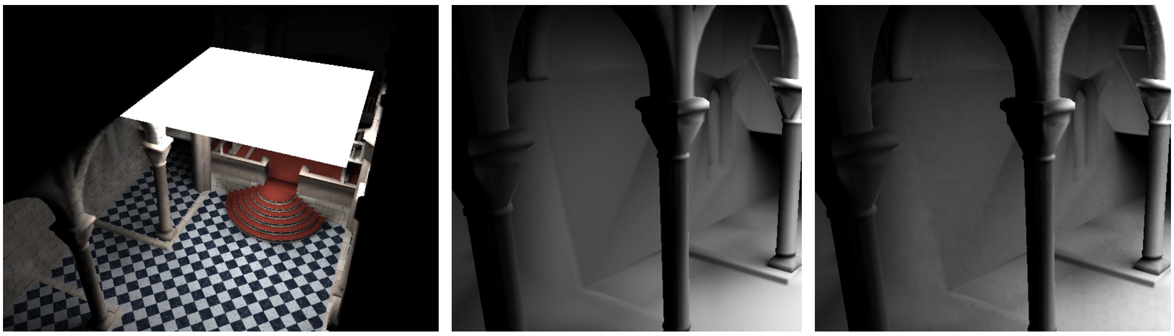


(a) Our result　(b) Nichols et al.'s [8] result　(c) Monte Carlo reference



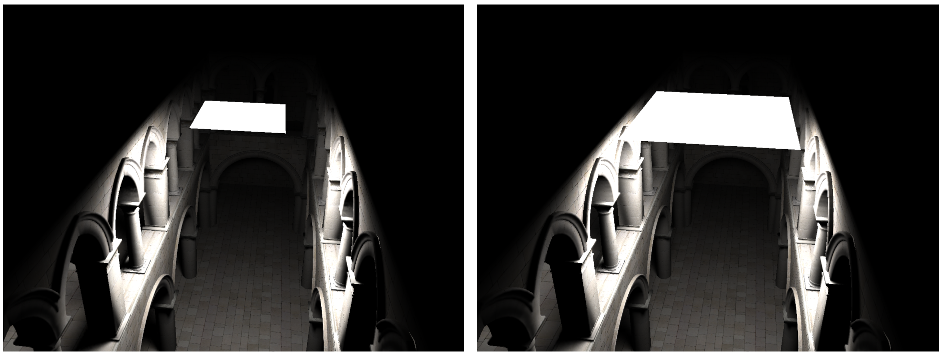
(d) Our fragment map and error map　(e) Nichols et al.'s [8] fragment map and error map

Figure 14: Rendering of the hairball object (McGuire Graphics Data) in 1280x960 pixels with direct illumination. In Figure 14d (right), we show the L2 error map in irradiance values (with scaling factor of 1x and 9x) between Figures 14a and 14c. The error map is mapped to the red color channel. Similarly Figure 14e (right) shows the L2 error map between Figures 14b and the reference image 14c.

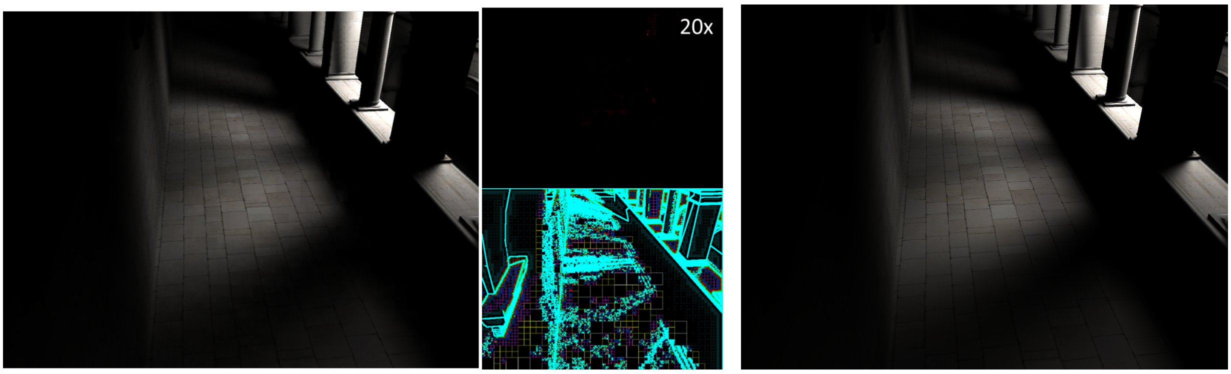


(a) Light configuration in Sibenik scene　(b) Monte Carlo reference　(c) Our result

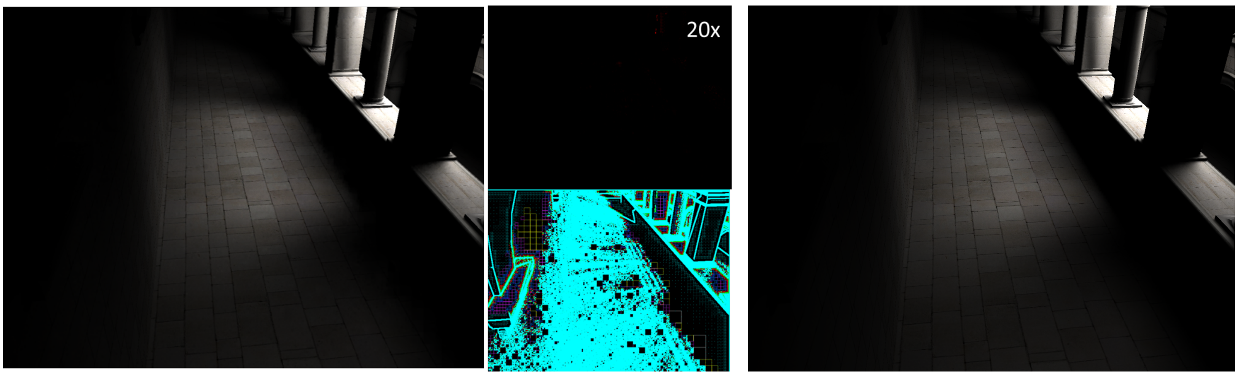
Figure 15: (a) The light configuration in the Sibenik scene. (b) A Monte Carlo reference is rendered for direct illumination from the area light source at 64 samples per fragment. (c) Rendering of our result at 64 samples per fragment. Visible random noise from under-sampling can be observed when the light is too large.



(a) Light size 2 unit (b) Light size 3.5 unit



(c) Our result for size 2 (d) Monte Carlo reference for size 2



(e) Our result for size 3.5 (f) Monte Carlo reference for size 3.5

Figure 16: Experiments with varying area light sizes and their positions. Renderings are done in 1280x960 pixels. First row indicates 2 various sizes of lights that we used. Second row indicates the results of rendering for a light of size 2 for our result (c) and it's error map, at the top right, (scaled 20x) is computed by comparing against a Monte Carlo reference(d). Third row indicates the results of rendering for a light of size 3.5. Sum of square errors (NMSE) for the rendered images in (c) and (e) are 6.225 (4.87e-4) and 4.297 (3.59e-4). The bottom right image of (c),(e) shows the fragment map.

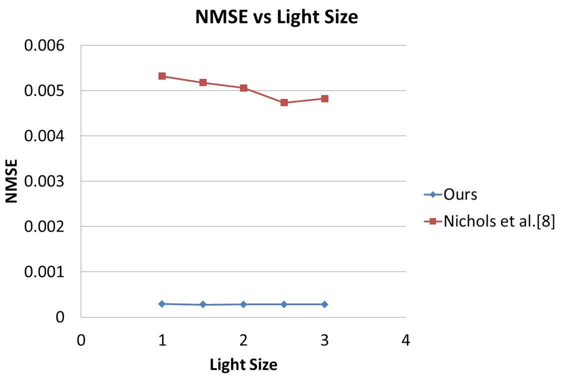


Figure 17: Plot of NMSE vs Light Size. The horizontal axis represent the one dimension length of the light, where the actual area of the lights used in the experiments are from 1^2, 1.5^2, to 3^2. NMSE of our work represented by the blue lines. Our error rate remains significantly lower than Nichols et al. [8]. Overall error in the image is unaffected by the light size.