SOFT SHADOW VISUALIZATION IN SYNTHETIC SCENES

by

ELVIS KO-YUNG JENG

A dissertation submitted to the Graduate Faculty in Computer Science
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy, The City University of New York

2005

This manuscript has been read and accepted for the Graduate Faculty in Computer Science in satisfaction of the dissertation Requirement for the degree of Doctor of Philosophy

11/19/2002
Date

Chair of Examining Committee

2/2/05
Date

Executive Officer

Dr. Ari Gross

Dr. George Wolberg

Dr. Xin Wang

Supervisory Committee

THE CITY UNIVERSITY OF NEW YORK

Abstract

SOFT SHADOW VISUALIZATION IN SYNTHETIC SCENES

by

Elvis Ko-Yung Jeng

Adviser: Professor Zhigang Xiang

This dissertation presents two two-stage soft shadow visualization methods for area light sources. The first method, visualizing soft shadows for deformable moving area light sources, uses adaptively sampled shadow maps to allow viewers to rapidly modify the shape and the location of area light sources and visualize the changes of soft shadows in a synthetic scene. During the pre-processing stage, this method creates a shadow slab that represents intermediate shadow information for all light samples and a scene map that represents surface samples captured from a pre-defined viewpoint. To reduce storage cost, our importance-based adaptive sampling technique stores the shadow maps that form the shadow slab and the scene map in quad-trees, and only stores the fine details for essential areas. In the visualization stage, we update the attenuation map and screen buffer whenever light sources are modified in the light template by accessing the shadow slab and the scene map to provide rapid visual feedback.

The second method, forward area light map projection, is suitable for arbitrary shape of area light sources. It projects sampled surface points (stored in a layered area light map)

onto the viewing screen to rapidly generate high-quality soft shadow images. The pre-processing stage creates the layered area light map. The map is multi-layered since each map cell stores the visibility ratio (light attenuation value) of the area light source with respect to multiple surface points at various depths. In order to rapidly generate the final image in the soft shadow rendering stage, we forward project light attenuation values of surface points that are stored in the layered area light map onto the screen buffer to attenuate the shadowless reference image. This forward area light map projection renders images much faster than ray tracing and still results in soft shadows with comparable quality.

In addition, we also present the "priority point lists for point-based object rendering" method. This method uses an adaptive sampling technique similar to the one introduced in the first method and the dynamic splatting technique introduced in the second method.

ACKNOWLEDGEMENTS

Firstly, I want to thank Dr. Zhigang Xiang, my advisor, for his enthusiasm, support, and insights into how research should be performed. Dr. Xiang is not only a teacher but also a good friend to me. He taught me the way to think, he gave me hints when I needed them, and he always supported me whenever I needed help.

Secondly, I wish to express sincere appreciation to Dr. Gross and Dr. Wolberg, the members of the committee, for their assistance and valuable input in the preparation of this manuscript. In addition, special thanks to Dr. Wang, whose familiarity with the needs and ideas of the work was helpful during the early programming phase of this undertaking.

Finally, I want to thank my lovely wife, Mishire, who cheerfully encouraged me with her endless support for the past few years. Without her, I would never have finished this study.

TABLE OF CONTENTS

LISTS OF TABLES

LISTS OF FIGURES

Chapter 1

Introduction

Shadows are important in enhancing both the realism and the intelligibility of synthetic 2D images of 3D scenes. The presence of shadows in an image helps viewers to better understand the spatial relationships between objects. It provides strong clues about the shape, relative position, and surface characteristics of the objects. It also indicates the approximate location, intensity, shape, and size of the light sources. Therefore, shadow generation has long been a concern in 3D computer graphics.

This dissertation focuses on the study of different algorithms, techniques, and implementations for soft shadow rendering in synthetic scenes and presents two methods. The first method is "visualizing soft shadows for deformable moving area light sources" and the second method is "forward area light map projection".

Moreover, we also present a two-stage point-rendering method (priority point lists for point-based object rendering) using adaptive sampling and dynamic splatting techniques. Although this method is not a soft shadow rendering method, it actually uses the adaptive sampling technique similar to that introduced in the first method and the dynamic splatting technique introduced in the second method.

1.1 Basic Terminology

Various terminologies have been used in computer graphics literature regarding the generation of shadow images. We introduce these terminologies before we go into the introduction of the techniques.

1.1.1 Synthetic scene

A synthetic scene in computer graphics consists of light sources and 3D objects. Each light source and each 3D object is given a set of properties. The properties of a light source could be the location, the shape, the intensity, and the color of the light source. The properties of a 3D object could be the geometry and surface attributes (e.g. material and reflectivity). A synthetic scene can then be rendered using different rendering algorithms.

1.1.2 Realistic image synthesis

In the field of computer graphics, the objective of realistic image synthesis is to compute the distribution of light given a description of the light sources and the 3D objects.

A difficulty in achieving a realistic image is the complexity of the real world. Our environment is rich with many surface materials, shadows, reflections, and slight irregularities in the surrounding objects. This makes the computational costs of simulating these effects high. Nevertheless, not all the computation efforts contribute to

the final image. Therefore, researchers have been trying to develop cost-effective methods for realistic image synthesis.

1.1.3 Viewing geometry

We may consider synthetic image generation as taking pictures of a synthetic scene using a virtual camera. In order to generate an image of a synthetic scene, the virtual camera needs to be placed at the desired location in the scene. This location is called "viewpoint". Then, we decide which direction to set the camera. This direction specifies which area of the scene is to be captured. We also rotate the camera around the line of sight to set the "up" direction for the picture. The direction to set the camera is called "viewing direction". The camera window indicates a specific region inside the scene area that defines the sub-area to be taken in the final image. Finally, the view volume (frustum), a pyramid with apex at the viewpoint, defined by the viewpoint, the viewing direction, and the window, bounds the region of the scene. Changing the viewpoint, the viewing direction, and the position and size of the window, which changes the view volume, can capture different portions of the scene seen from the viewpoint.

Viewing geometry determines the display of the 3D objects in the image generated from the viewpoint. That is, viewing geometry determines the portion of the scene seen from the viewpoint and the 3D objects that the eye sees. In short, the viewing geometry specifies where in 3D space the viewpoint is located (where the scene is viewed from), the direction the eye is looking in, and the window the eye sees.

1.1.4 Object model

There are many representations possible for three-dimensional object modeling. A graphics system typically uses a set of primitives (such as points and lines) or complex geometric forms (such as curves and quadric surfaces) to model a variety of objects. For example, polygon mesh surfaces, parametric surfaces, and quadric surfaces are three of the common geometric representations for object surface.

Many graphics systems store object descriptions as surface polygons, which simplifies and quickens the surface rendering. For this reason, surface polygons are often referred to as "standard graphics objects" [Hearn97]. If the original representations are not composed of polygons, users may transform objects into polygon meshes before rendering. Usually, this is done with triangle meshes to ensure that all vertices of any polygon are in one plane.

To produce realistic displays of synthetic scenes, we also need to describe the interaction of light and the transmittance and reflectance properties of various materials. Realism is further enhanced if the material properties of each object are taken into account when the object's shading is determined.

Natural phenomena such as cloud and fog that cast soft shadows do not have conventional surfaces and are not efficiently represented by large-scale geometric models. We focus in this section on geometric representation only. The interaction of the objects and light will be discussed in the illumination model (see 1.1.9).

1.1.5 Visible-surface determination

A major consideration in the generation of shadows is identifying those parts of a scene that are visible from a viewpoint at a light source. The shadow algorithms can be carried out in the object space or the image space. The terms, "object space" and "image space", are used to indicate the precision with which computations are performed. The most important fact is that the object space is continuous and the image space is discrete.

The shadow algorithms are broadly classified according to whether they deal with object definitions directly or with their projected images. The two approaches are called the object-space method and the image-space method, respectively. We can think of the object-space method as operating on the original continuous object data and the image-space method as operating on sampled data. Thus, the image-space method falls prey to aliasing in computing visibility, whereas the object-space method does not.

1.1.5.1 Object-space method

An object-space method compares objects to each other within the scene definition to determine which surfaces or which partial surfaces should be in shadow. Object space calculations are done without regard for display resolution, so they must be followed by a step to display the objects at the desired resolution. Only this final display step needs to be repeated whenever the viewing geometry or the size of the final image is changed. This is because the result of each visible object's computation is kept in the object space and needs to be projected onto the final image as the final step.

1.1.5.2 Image-space method

In an image-space method, information such as visible surfaces can be projected from the object space and kept in the fields of one or more projection images as the intermediate shadow information (see 1.1.7). The shadows in every final image are then calculated pixel by pixel according to the intermediate shadow information kept in these projection images. The sizes of the images vary, depending on the techniques used.

1.1.6 Image-based technique

Two major concepts related to image-based techniques are image-based modeling (IBM) and image-based rendering (IBR). Image-based modeling has been used to denote any sampled representation of a 3D scene. Here, we discuss more about rendering rather than modeling.

Image-based rendering refers to a class of rendering methods that generate new images from a series of images of a scene rather than from a model containing geometry and material specifications. These methods combine the provided images together, which are generated from different camera locations, to form new images. IBR is a shortcut for the traditional modeling/rendering pipeline and hides the latency between rendered frames, and often allows for shorter rendering time.

Image-based rendering techniques are independent from the complexity of the scene to be rendered, which makes real-time interactions with complex scenes possible. In addition, these techniques can make use of scanned images, or images of actual physical

environments. As a result, complex, real-life scenes can be viewed without difficulty or impossible geometric modeling.

However, current image-based rendering techniques rely on many types of sampling and interpolation of source images. When we use a small number of input images, these techniques cannot accurately capture high frequency components and changes in the scene such as specular highlights. Therefore, when the input images are too sparse or the samples of the environment are insufficient, errors may occur.

1.1.7 Intermediate shadow information

When calculating shadows, shadow algorithms process the scene to extract useful information and keep this information in the object space or the image space for shadow image generation. Information such as shadow volume, shadow boundary, shadow polygon, and shadow mesh is determined and kept in the object space; information such as the shadow map is determined from the object space and kept in the image space before the final shadow can be generated. This intermediate shadow information is essential for the computation of shadows.

1.1.8 Light source

In the real world, there are many kinds of self-luminous light sources. Several kinds of light sources have been discussed, such as light bulbs, direction light sources, spot light sources, linear light sources, curved light sources, and rectangle light sources.

In addition to self-luminosity, there is a diffuse, non-directional source of light, which is the product of multiple reflections of light from the many surfaces present in the environment. This is known as ambient light. In a simple lighting model, ambient light is assumed to produce constant illumination on all surfaces regardless of their position or orientation. Therefore, there is no shadow generated from ambient light.

In this dissertation, we classify light sources used in shadow algorithms into three kinds: point light source, area light source, and other light source.

1.1.8.1 Point light source

A point light source uses a point to represent the light source. A light source such as a light bulb radiates light energy in all directions, and, the center of the light bulb is referred to as the point light source. While rendering a scene that contains a point light source, we use visibility functions to determine if something lies between the light point and the observed surface point.

1.1.8.2 Area light source

In a realistic scene, most of the light sources are not merely a single point but also come in various shapes. There are different shapes of area light sources such as polygon light sources, linear light sources, curved light sources, and parallel linear light sources (i.e. two or more linear light segments parallel to each other). An area light source can be approximated by a number of point light sources.

1.1.8.3 Other light source

In additional to point light sources and area light sources, there are other light sources. For example, direction light sources and spot light sources can be treated as light sources located far away from the scene and each ray of the light sources is parallel to all the other rays. The sun can be a direction light source to the objects on earth; its light rays can be viewed as being parallel.

1.1.9 Illumination model

Light sources emit light energy. When reflected/transmitted by objects, light energy goes into an observer's eyes and allows the objects to be seen. Illumination models that are also frequently called lighting models or shading models, are used to calculate the intensity and the hue of light that we should see at a given point on the surface of an object. In general, illumination models can be divided into two classes: local illumination model and global illumination model.

1.1.9.1 Local illumination model

Local illumination models assume that the shading of each surface is independent from the shading of every other surface. All the local illumination models are represented as the summation of diffuse reflections and specular reflections inspired by direct illumination of light sources. These models typically assume that light comes from a finite set of point light sources only. Dozens of point light sources need a large amount of computation. However, when the number of point light sources is insufficient, highlight

and shadow roughness may occur. The shadow algorithms introduced in this study, which generate shadows by computing the illumination directly from the light sources only, are based on local illumination.

1.1.9.2 Global illumination model

Global illumination models recognize that the shading of a surface in the environment is interrelated with the shading of other surfaces. That is, the shade of a surface point is determined by the shades of all of the surfaces visible from that point. In complex environments, shadow generation is usually complex and time consuming because of the consideration of all the light sources, objects, diffuse reflections, specular reflections, and inter-object reflections in the environment. Algorithms such as ray tracing (see 2.1.1) and radiosity (see 2.2.8) are based on global illumination.

1.1.10 Shadows

There are three types of surface areas in a scene: lit area, umbra area, and penumbra area (see Figure 1). The entire light source is seen from a lit area. No light source is seen from an umbra area. Parts of the area light source are seen from the penumbra area (the area light is partially-occluded). Realistic shadows have penumbra areas surrounding umbra areas.

Different shadows are generated depending on the types of light sources. In the local illumination model, the intensity of the shadow for a single point light source is fixed, but the intensity varies for an area light source. That is, there is only the umbra area for a single point light source and there are both umbra and penumbra areas for an area light source (or multiple point light sources). Thus, there are hard shadows and soft shadows.

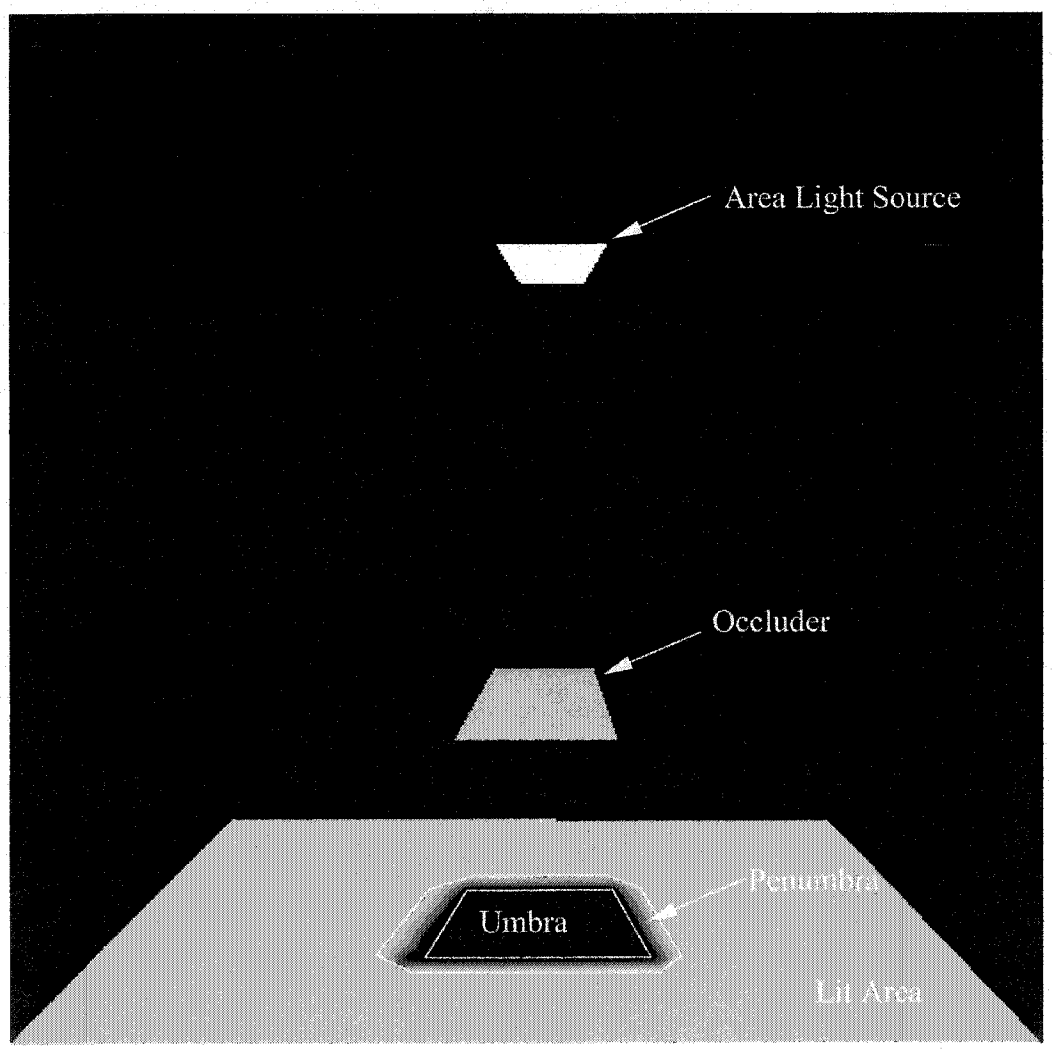


Figure 1: Soft shadow generated by an area light source.

Hard shadows are generated by a single point light source (see Figure 2). From the viewpoint of a single point light source, an object either occludes a surface point in the scene or not. A surface point occluded by an opaque object (or not lit by the light) is in umbra area; otherwise it is in lit area. The transitions from lit area to umbra area are abrupt. The illumination intensity of incoming energy in an umbra area is fixed, meaning

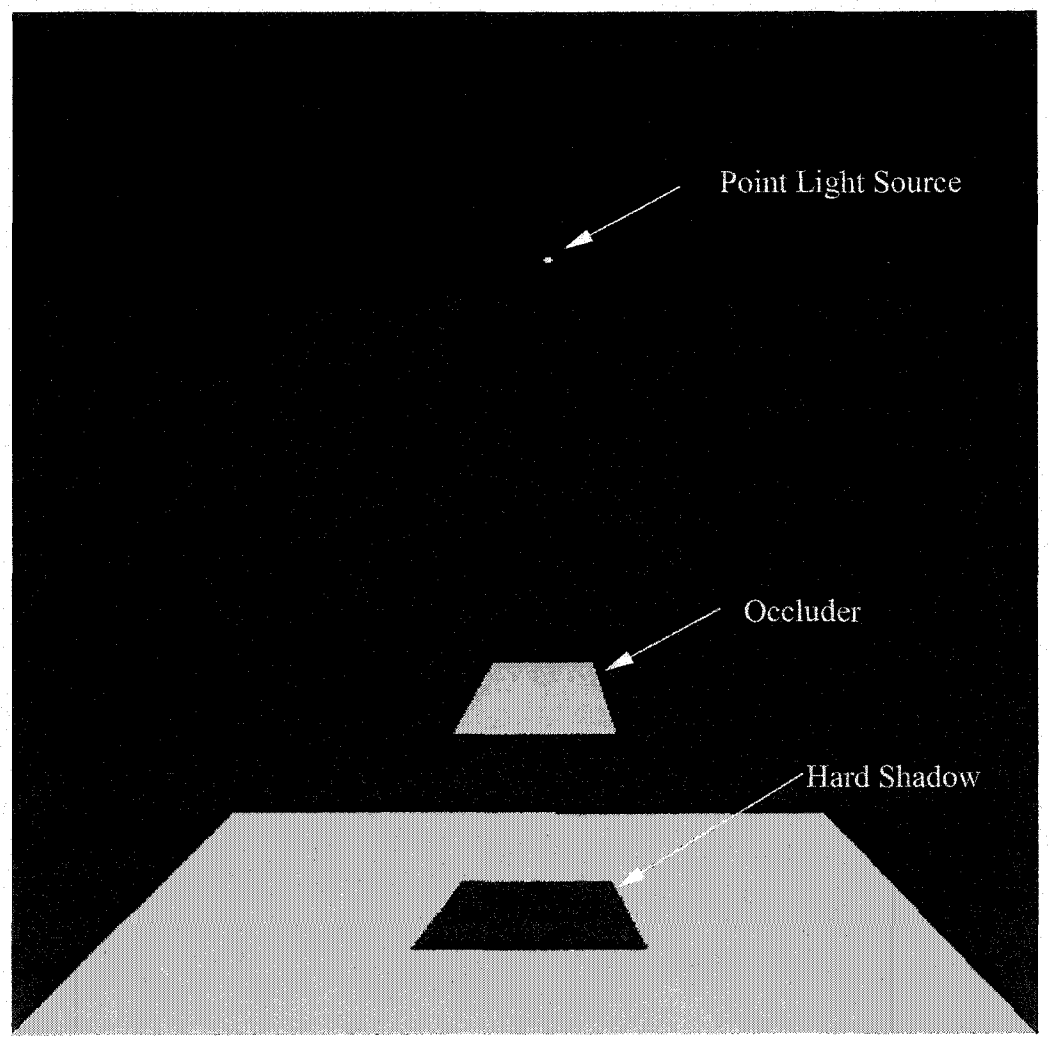


Figure 2: Hard shadow generated by a point light source.

the surface points in the umbra area are invisible from the light source in the local illumination model without considering inter-reflection and transparency of objects. Only the ambient light illuminates the umbra area. Nevertheless, from the global point of view, an umbra area is only illuminated by the inter-reflection of light between surfaces and by attenuated light from the source if a transparent object occludes this area.

Soft shadows are generated by an area light source and are a combination of penumbra area and umbra area (see Figure 1). However, the umbra area does not necessarily exist in a soft shadow. The partially occluded area light source results in graduated shadow transitions in the penumbra area. The intensity of illumination in a penumbra area varies depending on the shape of the light source and the object that occludes the light source.

1.2 Preliminaries for the New Methods

In order to achieve fast production of soft shadow images, a two-stage strategy is employed in our two soft shadow generation methods. This strategy pre-processes the scene to extract and save useful information so that the rendering operations can later be carried out efficiently in the second stage. For example, to shade a surface point properly, especially a surface point that belongs to a penumbra, the amount of light energy coming to that surface point from the area light source is a crucial piece of information. This information can be used either to determine the shading of the surface point directly or to attenuate its shading in a shadowless image of the scene later in the

second stage. This two-stage strategy is used in different ways and the challenge here is to produce high-quality soft shadows while limiting the amount of pre-processing efforts with carefully tailored approximation (e.g. discrete sampling).

1.3 Contributions

The main contribution of this study is the two soft shadow rendering methods. In addition, we also present "priority point lists for point-based object rendering" which is the application of adaptive sampling in importance-based sampling and point rendering.

1.3.1 Visualizing soft shadows for deformable moving area light sources

This method dynamically generates soft shadows for static objects of a scene in which the light sources are deformable and movable. In the pre-processing stage, this method creates a shadow slab that represents intermediate shadow information for all light samples and a scene map that represents surface samples captured from a pre-defined viewpoint. To reduce storage cost, our importance-based adaptive sampling technique stores the shadow maps that form the shadow slab and the scene map in quad-trees and only stores the fine details for those areas that need them. In the visualization stage, by accessing the shadow slab and the scene map to provide rapid visual feedback, we then update the attenuation map and the screen buffer whenever light sources are changed in the light template.

1.3.2 Forward area light map projection

"Forward area light map projection" generates soft shadow images by projecting sampled surface points kept in a layered area light map onto the viewing screen. In the pre-processing stage we generate a layered area light map for each area light source. The structure of the layered area light map is similar to the layered attenuation map introduced in [Agrawala00]; however, the generation of the layered area light map is different. We render soft shadows by forward projecting light attenuation values of surface points that are kept in various layers of the area light map onto the screen buffer to attenuate the pre-created shadowless reference image. The approach is especially efficient for creating a series of shadowed images from different view locations for a static scene.

1.3.3 Priority point lists for point-based object rendering

To produce points in our point rendering approach, we adaptively sample surface points from a triangle-mesh object. This adaptive sampling technique samples surface points from each triangle adaptively. For image rendering, we use dynamic splatting technique, which divides the square that contains a triangle of the triangle-mesh object into smaller triangles when the square is close to the edges of the triangle. Thus, more points are sampled from edge areas than the centers.

Unlike [Rusinkiewicz01] and [Wand01] who used fixed-sized splats (only one sized splat used in an image) and assumed the surface normal always faced the front of the

rendering screen, our method uses dynamic splatting that renders multi-sized splats in an image according to their surface normal and the distances to the viewpoint. As a result, while point rendering methods using dynamic splatting generally need a huge number of points in order to reconstruct the surfaces, our method stores fewer sampled points. Meanwhile, fewer sampled points also result in less memory usage and shorter rendering time.

1.4 Organization

We start with a review of the previous work and then introduce our three methods in 1.3 in the following order: "visualizing soft shadows for deformable moving area light sources," "forward area light map projection," and "priority point lists for point-based object rendering." Finally, a conclusion is made. Therefore, the rest of this dissertation will proceed as:

Chapter 2: Previous Work

We review the previous work on soft shadow rendering and categorize them into eight types.

Chapter 3: Visualizing Soft Shadows for Deformable Moving Area Light Sources

We describe the techniques to create adaptively sampled shadow maps and the scene map in order to construct/update the shadowed image in the visualization stage.

Chapter 4: Forward Area Light Map Projection

We introduce our technique to create the layered area light map in the pre-processing stage and forward project light attenuation values of surface point onto the screen buffer to generate the final image in the soft shadow rendering stage.

Chapter 5: Priority Point Lists for Point-based Object Rendering

We describe the adaptive sampling technique and the point-based object rendering technique used in this point rendering method.

Chapter 6: Conclusion

The importance of the intermediate shadow information is emphasized. Meanwhile, future work in the field is also proposed.

Chapter 2

Previous Work

Many algorithms have been proposed to calculate soft shadows. Soft shadow algorithms, first calculate intermediate shadow information, then use this information in conjunction with the original object definition to render the final shadowed images. An extensive review of early shadow algorithms can be found in Woo et al. [Woo90]. Since then, researchers have focused on high quality and/or interactive soft shadow rendering that often permits the movement of viewpoints, scene objects, and/or area light sources without a complete re-calculation to speed up shadow calculation.

During processing, most algorithms keep this intermediate shadow information in a media such as memory or hard disk for the subsequent shadow calculation step. Nevertheless, ray tracing calculates and uses this information to generate shadows without keeping it. Moreover, depending on the algorithms used, intermediate shadow information may need to be recalculated when light sources, objects, and viewpoints change. For example, in shadow map algorithms, the intermediate shadow information stored in the shadow maps needs to be recalculated whenever objects or light sources change. However, the intermediate shadow information does not need to be recalculated for a new viewpoint if light sources and objects are unchanged. The shadow image can still be generated for a new viewpoint from the existing shadow maps. In ray tracing, on

Table 1: Eight types of shadow computation algorithms. (S: Static; D: Dynamic) Only types 1, 2, 4, and 5 are found in the existing algorithms.

Types	**1**	**2**	3	**4**	**5**	6	7	8
Light Source	**S**	**S**	S	**D**	**S**	D	D	D
Scene Object	**S**	**S**	D	**S**	**D**	S	D	D
Viewpoint	**S**	**D**	S	**S**	**D**	D	S	D

the other hand, the intermediate shadow information needs to be recalculated to generate shadow images whenever light sources, objects, or viewpoints are changed.

In this chapter, we review soft shadow algorithms and categorize them into eight types according to our criteria. The light source, the scene object, and the viewpoint of a shadow algorithm are defined as either "static" or "dynamic" according to the requirements of intermediate shadow information recalculation (see Table 1). Each type is labeled as "static" if the shadow algorithm must recalculate the intermediate shadow information for generating shadows if the light sources, the scene objects, or the viewpoints changed. Otherwise, the type is labeled as "dynamic".

At this time, there is no existing algorithm that is of type 3, 4, 6, 7, and 8. However, one of the new shadow algorithms ("visualizing soft shadows for deformable moving area light sources" introduced in chapter 3 of this dissertation) falls into the type-four category. Therefore, we only discuss the four types of algorithms in the following descriptions.

2.1 Type 1 Algorithm: Static Viewpoint under Static Light and Static Scene

Ray tracing is the only one algorithm that falls into this category. When ray tracing is used to render an image, everything must be static because the intermediate shadow information is not stored. In other words, whenever the light sources, the scene objects, or the viewpoints are changed, all the intermediate shadow information is recalculated.

As a fundamental rendering technique, ray tracing [Appel68] [Whitted80] is very powerful and is quite suitable for shadow generation. Ray tracing performs visibility calculation in the object space. Guo [Guo98] accelerates ray tracing by using image space coherence to reduce the number of primary rays traced. Hart et al. [Hart99] also develop a view dependent method to speed up soft shadow computation for a ray tracer.

For the rendering of soft shadows, ray tracing can be extended to distributed ray tracing [Cook84] [Shirley96] [Formella98]. Amanatides et al. [Amanatides84] propose cone tracing that is similar to distributed ray tracing. Meanwhile, tracing multiple light samples to approximate an area light source will also create soft shadows [Campbell90].

2.2 Type 2 Algorithm: Dynamic Viewpoint under Static Light and Static Scene

Most of the algorithms fall in this category. Once the intermediate shadow information is calculated, final shadowed images are created for the new viewpoints without recalculation.

2.2.1 Shadow map

Shadow map algorithms keep intermediate shadow information in the shadow maps that are created from a static scene for predefined light sources. A surface point is determined to be in the shadow, if the distance from the point to the light source is greater than the corresponding distance stored in the depth map. For example, Williams' shadow map algorithm [Williams78] is a sampling based method that creates a discrete depth map from the viewpoint of a predefined point light source.

A lot of efforts have been devoted to improving shadow map approaches. Reeves et al. [Reeves87] use percentage-closer filtering to alleviate the aliasing problem along shadow boundaries. Self-shadowing of surfaces and missing shadows, caused by numerical problems in depth comparison, are resolved by Woo [Woo92]. He modifies the algorithm to use a shadow map where the reference value is actually a weighted sum of the visible surface and the first surface point behind it. Segal et al. [Segal92] introduce "projective textures" to record the occlusion of the light source and support soft shadows using high-end graphics workstations. Chen and Williams [Chen93] render a few key

shadow maps at the vertices of the light source and then use view interpolation to compute shadow maps for each sample location inside the region.

In addition, forward shadow mapping [Zhang98] is a solution to solve the bottleneck problem for normal rendering. Parker et al. [Parker98] assume a point light source and use ray tracing to approximate the soft shadow by manipulating the geometry of the occluders. Brabec et al. [Brabec00] implement their shadow map on OpenGL pipeline without support from high-end hardware. Heidrich et al. [Heidrich00] also use Brabec et al.'s approach to implement their soft shadow extension to shadow maps. Agrawala et al. [Agrawala00] use a view independent method to generate soft shadows. From a set of pre-rendered shadow maps, this image-based method can generate shadows for any view of the scene.

2.2.2 Shadow volume

Shadow volume algorithms perform visibility calculation in the object space, using a complete representation of the scene geometry as a backdrop [Crow77]. Shadow volumes, which are constructed from a point light source and polygonal occluders [Crow84] or curved boundary occluders [Heflin93], contain intermediate shadow information. Thus, surface points can be quickly compared against the volumes for inclusion in shadows.

Brotman et al. [Brotman84] and Bergeron et al. [Bergeron86] approximate soft shadows by replacing linear or area light sources with multiple light samples, each of

which corresponds to a shadow volume. Chin et al. [Chin89] use binary space partitioning (BSP) trees for complex volumes. Chin and Feiner [Chin92] use BSP trees to refine the boundary mesh for point and area light sources for umbra and penumbra shadow boundaries. Diefenbach and Badler [Diefenbach94] develop a shadow volume algorithm for graphics hardware using the stencil buffer. Udeshi and Hansen [Udeshi99] modify Diefenbach's shadow volume approach for special situations, in which the near plane of the view frustum straddles one of the shadow volumes.

2.2.3 Back-projection

Back-projections are data structures that determine the visible parts of the sources anywhere in the penumbra. Drettakis and Fiume [Drettakis94] and Stewart and Ghali [Stewart93] [Stewart94] back-project distinct regions of the scene onto the area light source. In their approach, a discontinuity mesh contains intermediate shadow information built on a plane that is parallel to the area light source. Ouellette and Fiume [Ouellette99] detect singular points and lines to represent intermediate shadow information on the light sources by using Random Seed Bisection (RSB) and Two Discontinuity Finding (TDF) algorithms to approximate the locations of discontinuities for area light sources and for the classes of locally convex occluders.

2.2.4 Discontinuity meshing

The discontinuity meshing approach [Campbell90] [Heckbert92] [Lischinski92] [Durand97] improves the quality of rendering using radiosity for scenes containing soft

shadows and avoids the inaccurate solution and shadow leaks of surface meshing. When area light sources illuminate a polygonal scene, "discontinuity meshing" partitions the scene surfaces into regions. As a result, all the discontinuity lines are inserted into the structure. Within each region, all surface points receive the same illumination value. This intermediate shadow information kept in the discontinuity mesh structure is then used for the final image rendering. Even though this approach produces high quality images, it suffers from numerical instability and is very expensive to compute, especially for very complex scenes.

2.2.5 Hybrid method (shadow volume + shadow map)

Shadow volume is an object-space method, while shadow map is a purely sampling based approach that works with depth images of the scene. This hybrid approach uses the image-space method to obtain shadow volume information from shadow maps.

McCool generates shadow volumes directly from a depth image (shadow map) of the scene to reduce the geometric complexity of shadow volumes [McCool98]. He uses an edge detection algorithm to obtain the discontinuities in the map, which then act as the boundary polygons for a shadow volume. Heidrich et al. [Heidrich00] propose an algorithm that produces a soft shadow for a linear light source with a small number of light samples in an interactive rate. This method uses the intermediate shadow information in the shadow maps of the two endpoints of a linear light to find the shadow value of a point on the surface. Heidrich et al. use edge-detection to obtain discontinuities

that are used for rendering soft shadows, rather than extracting shadow volume information like McCool.

2.2.6 Accumulation buffer

The accumulation buffer is an approximation approach that samples the area light source. This method calculates a hard shadow from each light sample. These hard shadows are then averaged to produce a soft shadow.

The illumination from area light sources can be simulated by multiple point light sources. Heckbert et al. [Heckbert97] combine shadow maps to create the soft shadows for the linear or area light sources. Each shadow map corresponds to a sample point on the light source. The shadow maps are created, registered and averaged on each polygon of the scene. A soft shadow texture map can then be created as the intermediate shadow information using the accumulation buffer for each polygon. Finally, soft shadows in a static scene are displayed in real-time with simple texture mapping of the created textures.

2.2.7 Convolution

Soft shadows can also be approximated using the convolution technique. This technique uses fast Fourier transform to efficiently convolute an image of the light source with an image of blockers to form a blurred blocker image. This blurred blocker image, which keeps the intermediate shadow information, is then projected onto receiver objects to generate the soft shadows. Max applies the convolution technique to render complex

shadows [Max91]. Soler and Sillion [Soler98] extend the convolution technique to create approximate soft shadows quickly. Their method is based on the calculation of shadow maps that are created using convolution and rendered in off-screen buffers.

2.2.8 Radiosity

The radiosity method is first developed in the field of thermal engineering. It is used to calculate radiant interchange between surfaces, as in boilers and radiator design [Siegel81] [Sparrow78] [Wiebelt66]. It is then generalized and optimized in a number of ways for use within the field of computer graphics.

The basic assumption of the radiosity method is that most of the materials in a scene are diffusive. In this approach, each light source in the 3D virtual environment is treated as a source of energy that casts energy (flux) towards the scene [Cohen93]. Patches are used to compute, store and reconstruct illumination functions. Each patch computes the amount of energy that casts over the rest of the patches in the scene. The exchange of energy between patches (mesh elements) is evaluated using form factors to represent the effect of orientation, geometric attenuation and visibility. Frequently, an implementation assumes a constant radiosity distribution across each surface patch, and then the energy is interpolated across each mesh element and saved as the intermediate shadow information. Each surface patch that contains the intermediate shadow information is credited with a unique ability to render soft shadows. During the rendering stage, it renders scenes

without recalculating the illumination unless either the light sources or scene objects are changed.

It is recognized quite early that solutions can be improved if the form factors are computed analytically. Analytical solutions have been determined for specific geometries. Generally, such solutions assume that each surface is flat and fully visible from the other. Baum et al. [Baum89] propose computing the unoccluded surface point to polygon form factor analytically. To avoid violation of visibility assumptions, they propose that each source patch be subdivided until all components are either fully visible or fully hidden from all elements in the environment. Zatz [Zatz93] pushes this idea further by proposing the use of sampled shadow masks. In his approach, visibility is calculated separately and at a much higher resolution than unoccluded radiosity. This results in a shadow mask, which represents occlusion across the receiving patch, and can be post-multiplied into the radiosity function to produce good shadows. Meanwhile, Haines [Haines94] introduces shaft culling to speed up visibility computation in radiosity system.

In addition, several authors observe that in the case of ideal diffuse scenes, the entire illumination can be recorded into radiosity textures. Such textures can be pre-computed off-line, allowing high-quality rendering with soft shadows at interactive rates [Heckbert90][Myszkowski94]. Keller's "instant radiosity" technique [Keller97]

computes radiosity textures in a manner similar to Heckbert's, by averaging shadow images from chosen point samples on the surfaces.

2.2.9 Interactive texture-mapping

Herf et al. [Herf96] use texture-mapping hardware for displaying illumination maps. These maps contain intermediate shadow information, and generate soft shadows during real-time walkthroughs of static environments. However, this method is impractical for large scenes, since the precomputation time grows quadratically with the number of objects being shadowed. Beside the approach of Herf et al., some interactive techniques pre-compute and display soft shadow textures for each object in the scene [Heckbert97] [Soler98].

2.2.10 Image-based method

Lischinski and Rappoport [Lischinski98] use hierarchical ray tracing of depth images (which contain intermediate shadow information) as one of several image-based techniques for computing secondary rays in synthetic scenes. Keating and Max [Keating99] point out that light leaking is a problem in this approach, since each depth sample is treated independently as a 2D surface without connections to adjacent samples. They then extend Lischinski and Rappoport's idea and generate shadows from image-based scene representations (which contain intermediate shadow information). Although this approach reduces light leakage, it forms relatively large flat surfaces that can significantly alter the scene geometry.

Max's hierarchical rendering method [Max99] uses the Precomputed Multi Layer Z Buffers (which contain intermediate shadow information), which are similar to the layered depth image (LDI) [Shade98]. Agrawala et al. [Agrawala00] use a layered attenuation map approach to combine a similar number of depth images rendered from different light samples on the light source. They improve Heckbert and Herf's method by pre-computing image-based textures (which contain intermediate shadow information) simultaneously for the entire scene. This approach achieves interactive rendering rates but limits sampling flexibility.

2.3 Type 4 Algorithm: Static Viewpoint under Dynamic Light and Static Scene

A new shadow algorithm that is introduced in this dissertation falls into this category. The "visualizing soft shadows for deformable moving area light sources" (see chapter 3) is a method for soft shadow visualization for a static scene where the light sources are deformable and movable. In a way akin to Levoy and Hanrahan's creation of a light slab with an array of scene images [Levoy96], we create a shadow slab with an array of shadow maps to keep the intermediate shadow information. The detailed descriptions are in chapter 3.

2.4 Type 5 Algorithm: Dynamic Viewpoint under Static Light and Dynamic Scene

Recently, researchers have focused on interactive scene manipulating and editing. They develop methods to create images for scenes in which moving scene objects project shadows onto the other moving or non-moving scene objects. Baum et al. [Baum89] pre-calculate each object's movement, which accelerates the form factor calculation for each frame. Chen et al. [Chen90] use the progressive refinement radiosity algorithm to move shadows. Shadows are removed by shooting positive energy, then re-instated by shooting negative energy. Forsyth et al. [Forsyth94] use hierarchical radiosity to calculate the soft shadow of a dynamic scene. Similar to [Forsyth94], Shaw [Shaw97] uses "motion volume" to identify the links affected by the displacement of an object. Worrall et al. [Worrall95] treat each shadow area as a collection of individual shadow lines and process each line in isolation. They determine the parts of the scene needing modification to enable all changes in shadow lines and then to handle the soft shadows of dynamic scene objects. Chrysanthou and Slater [Chrysanthou97] propose a method using dynamic BSP trees to calculate soft shadows. From the ideas of Worrall et al. [Worrall95], Loscos et al. [Losco97] propose an approach that identifies visibility changes by using information contained in the discontinuity mesh of each scene polygon. This method fails for scenes with more than several thousand polygons. Worrall et al. [Worrall98] use discontinuity meshing to enable the interactive animation of soft shadows.

In addition, Fernandez et al. [Fernandez02] propose an interactive renderer that computes direct illumination in dynamic scenes with soft shadows and the complex Bidirectional Reflectance Distribution Functions (BRDFs) [He91]. Users can both navigate and modify the scene interactively. Shadow information is not recalculated when the viewpoint changes, but some blocker lists may need regeneration when scene objects are moved.

2.5 Summary

We have categorized soft shadow algorithms in this chapter according to the requirements of intermediate shadow information recalculation resulting from the changes of light sources, scene objects, and viewpoints. The calculation of intermediate shadow information is a key component for this categorization. Even though there are eight possible types of soft shadow generation algorithms, we find that the existing algorithms only fall into four types, including the algorithms introduced in this dissertation. From the soft shadow algorithms we have reviewed here, we conclude that designing a good way to extract and keep useful intermediate shadow information for a given scene is an important step in shadow algorithms.

In this dissertation, we introduce two soft shadow algorithms. The first one (visualizing soft shadows for deformable moving area light sources) falls into our type-four category (static viewpoint under dynamic light and static scene). This algorithm partially updates intermediate shadow information whenever the light sources are moved

and/or deformed. The second one (forward area light map projection) falls into our type-two category (dynamic viewpoint under static light and static scene). This algorithm creates intermediate shadow information, which does not need to be recalculated when we move the viewpoint, for static scene objects and light locations.

Chapter 3

Visualizing Soft Shadows for Deformable Moving Area Light Sources

In this chapter, we present a two-stage soft shadow visualization method for a static scene where the light sources are deformable and movable. This process includes using a two-stage approach where lighting components are manipulated easily and viewing components are updated dynamically. Therefore, by moving/deforming the area light sources, viewers visualize the changes of soft shadows.

This method is composed of two stages: the pre-processing stage and visualization stage. The system outline is introduced in section 3.1. An introduction of the importance-based adaptive sampling technique used in the pre-processing stage is presented in section 3.2. The descriptions of the pre-processing stage and the visualization stage are in section 3.3 and section 3.4 respectively. Finally, the implementation of the method is described in section 3.5 and the summary is discussed in section 3.6.

3.1 System Outline

The system is composed of lighting components and viewing components. The lighting components include the light template, the differential map, and the shadow slab, whereas the viewing components include the scene map, the attenuation map, and the screen buffer (see Figure 3.)

The light template, which keeps track of active light samples and illuminates the synthetic scene, is used to define the shape, the location, and the intensity of area light sources in the visualization stage. The differential map identifies which light samples have been activated or deactivated when lights are moved/deformed and then triggers the system to update the attenuation map. Finally, the shadow slab is a two-dimensional array and is used to accelerate shadow generation in the visualization stage.

On the other hand, the scene map is created from a predefined viewpoint to store the surface normal, location, and surface attributes (such as RGB values, which are generated by the illumination equation I = Ki where Ki is the surface's intrinsic intensity) of each sampled surface point. The attenuation map is used in the visualization stage to store the attenuation value in each map cell. Finally, the screen buffer is used to display the result

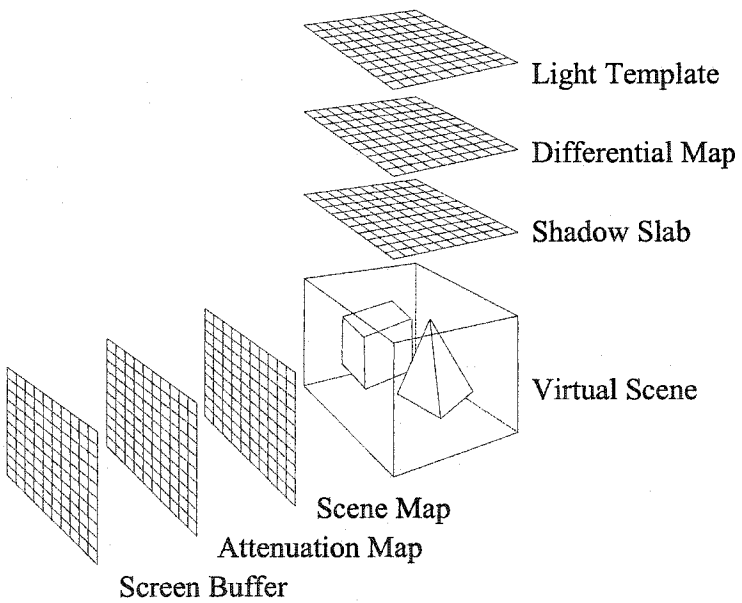


Figure 3: The viewing and lighting components.

on the screen.

In the pre-processing stage, we create a shadow slab and a scene map. To visualize the synthetic scene in the visualization stage, we rapidly update the attenuation map (which is associated with the screen buffer) through the differential map to access shadow maps that correspond to the modified light samples. While the light template is the representation of working light sources (including activated and deactivated light samples,) the shadow slab represents the intermediate shadow information of all the light samples. The screen buffer is updated using the surface attributes in the scene map and the illumination values in the attenuation map. As a result, the surface shading and shadowing, which are subject to the illumination in the synthetic scene, are updated dynamically.

3.2 Importance-Based Adaptive Sampling

In order to reduce memory usage, we use the importance-based adaptive sampling technique to capture the important samples from the scene. The modification of the sampling frequency is the core of the importance-based sampling, which is described in section 3.2.1. The adaptive sampling that generates the quad-tree, which is used to store the sampling result, is introduced in section 3.2.2. Finally, the retrieval of the quad-tree is introduced in section 3.2.3.

3.2.1 Uniform sampling vs. importance-based adaptive sampling

In a digital picture of the real world and a synthetic image, each pixel represents a sampled surface point of the scene objects. To generate synthetic images, ray tracing, a rendering algorithm, can sample surface points at full resolution of the image pixel-by-pixel. Besides this uniform sampling approach, Phong shading [Phong75], an interpolative shading method, only samples important points (vertices) to find normal vectors of the points and then approximates the resulting image by interpolating the normal vectors and filling the projected pixels for the surface points. Although Phong shading only samples important points, the quality of the resulting image generated and approximated from the points is close to the fully sampled one.

In a digital image, the edge that separates one triangle from another triangle is important in distinguishing the transition between triangle mesh surfaces. It is hard to approximate edges correctly from a set of surface points sampled insufficiently. Information that is closer to the edge is more important, so more sampling points are needed for the more important areas.

We propose importance-based adaptive sampling technique to sample more points from more important areas and fewer points from less important areas. The size of the node square near an edge decreases, resulting in relatively high sampling density that alleviates the jagged edge artifact. The technique reduces memory consumption and, meanwhile, provides sufficient information for image quality.

In short, we reduce the number of sample points by going from sampling at full resolution to sampling important surface points only. The images generated from our importance-based adaptive sampling yield similar quality to those from uniform sampling.

3.2.2 Importance-based adaptive sampling and quad-tree generation

When we capture important samples from the scene, a quad-tree data structure is used to store the sampling result. Starting from the root of each quad-tree, which corresponds to the square region (focal plane) for each depth map, we shoot four rays through the corners of the square. The IDs and the depth values of the intersected triangles of the sampled points at the four corners of each square are stored in the node.

Theoretically, when a flat surface consists of two or more triangles with the same surface normal, the edges between triangles are not important. Therefore, the introduction of surface ID would easily avoid wasting resources to work on edges between triangles on the same plane. However, since the triangle mesh model we use in the implementation lacks such information, the triangle ID is a good indicator of a shared edge of two triangles within the square range. Besides, when the IDs at the four corners belong to the same triangle, the depth values of the four corners provide surface slope information. The steeper the slope is, the larger the difference of the depth values between two neighboring samples is and a greater distortion results from the bilinear interpolation. Thus, more

samples are needed to get a closer approximation to reduce the distortion between the two sample points. As a consequence, we use the triangle IDs and the depth values to determine if the node needs to be further divided into four smaller squares. If it does, four more quad-tree nodes are created to represent the four sub-regions and then their four

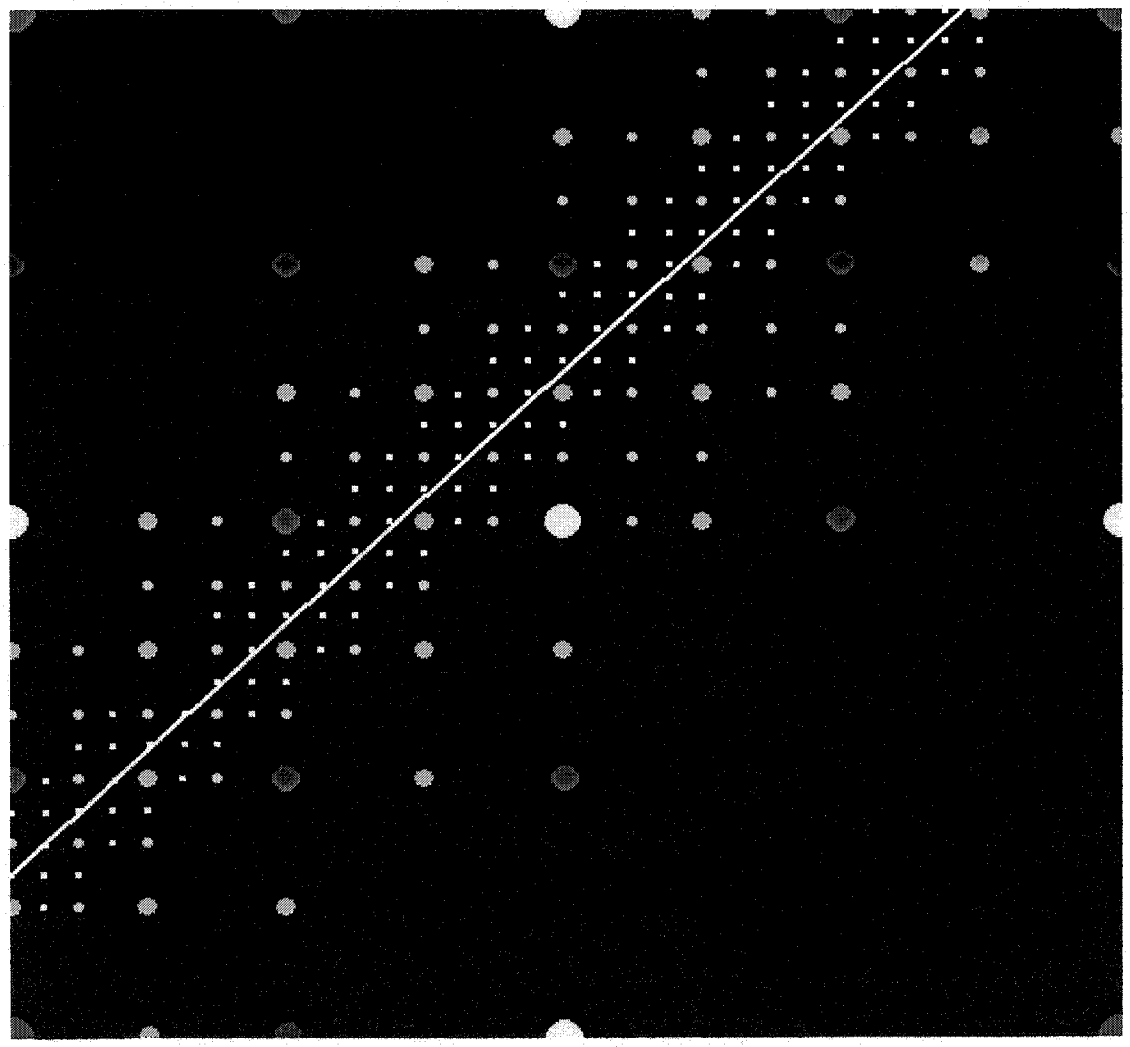


Figure 4: Samples captured from the scene using the importance-based adaptive sampling technique.

corner points of each sub-region are recorded.

We preset a division level value for uniform division. After the preset division level has been reached, a square region is further divided into four sub-squares until they reach the pixel level if it satisfies any of the criteria listed below:

- Triangle IDs stored in four children do not belong to the same triangle. (If the triangle IDs are not the same, there is at least one potential edge in the region.)
- The depth values for the four corners differ by more than a preset threshold. (Denser sampling is needed to prevent distortion for a tilted surface.)

In Figure 4, a black surface (triangle) and a blue surface (triangle) share a white edge. The pink points on the four corners of the image represent the recorded sample points of the root. The square then is divided into four smaller squares with the yellow sample points as their corners. Thus, the red points, the orange points, the green points, and the white points represent the new corners of ever-smaller squares. As a result, the denser the sample points are, the closer the points are to the edge.

3.2.3 Quad-tree retrieval

To retrieve information from the quad-tree of each depth map, we use bi-linear interpolation to approximate the desired value from the map's sample points. While retrieving the value of a surface point, which is traced from the viewpoint through an image pixel, we check the four sub-square regions of a node to see which region the point belongs to. After we move one level deeper into the found sub-square node, we check the node recursively until we reach a leaf node. We then bi-linearly interpolate the desired point value from the leaf node's four corner points.

3.3 Pre-processing Stage

In this section, we explain the processes in creating the shadow slab (an array of shadow maps) and the scene map in the pre-processing stage.

The shadow slab represents the intermediate shadow information of all the light samples. Unlike Levoy and Hanrahan [Levoy96] who utilize the light field buffer to store scene information (color texture), we create a shadow slab with an array of shadow maps in our method. Within each cell of the shadow slab, an adaptively sampled shadow map is created from the "viewpoint" of the corresponding light sample (see the illustrations in Figure 5 and Figure 6). To reduce memory usage of the shadow maps, we employ the importance-based adaptive sampling technique to enable the shadow maps to store only the depth values of the "important" samples rather than all the samples. The creation of the shadow slab increases the pre-calculation time and memory consumption but it

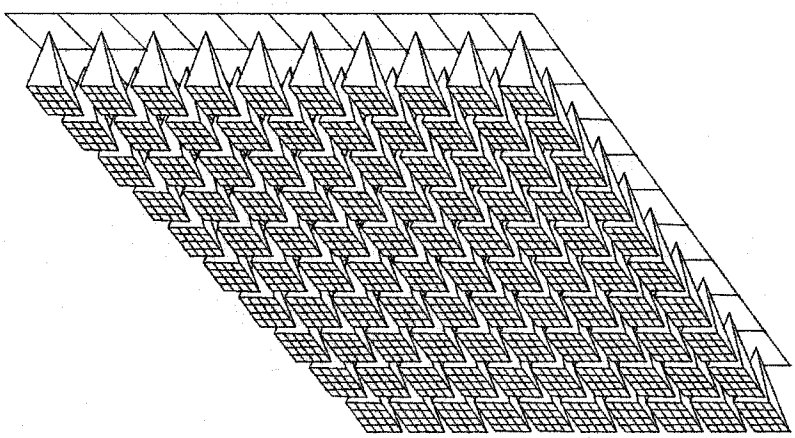


Figure 5: Shadow slab formed by shadow maps for all light samples.

provides the flexibility for multi-shaped light sources for viewer's visualization.

The scene map represents the sampling result of the scene from the viewpoint. We trace rays from the predefined viewpoint into the scene. Unlike the regular ray-tracing technique that computes surface shading, we sample adaptively and store the surface normal, coordinates, and the RGB values of the sampled surface points in the scene map. In the scene map, each pixel, which corresponds to a pixel in the screen buffer and determines the attenuation value from the shadow slab, is used to retrieve a surface point in the visualization stage.

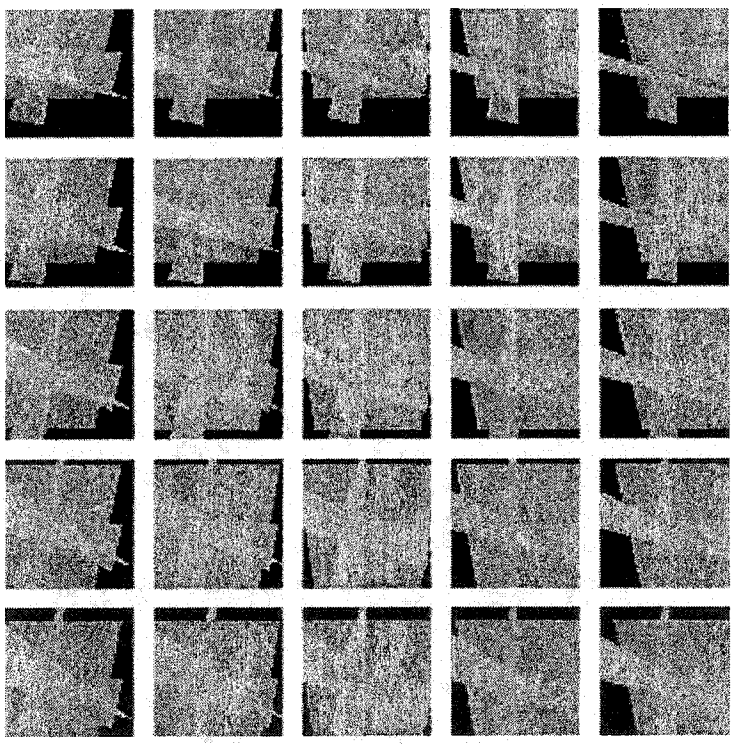


Figure 6: Shadow maps rendered from the shadow slab for testing scene in figure 10.

3.3.1 Adaptively sampled shadow map

The importance-based adaptive sampling technique introduced in 3.2 is used in the creation of the shadow maps to form the shadow slab as one of the lighting components. In contrast to Fernando et al. [Fernando01] who create the Adaptive Shadow Map dynamically with the need to maintain scene geometry, we create a shadow slab in the pre-processing stage and store critical scene information in the scene map, thus we free the system from storing scene geometry for the visualization stage. In Figure 7, we show several shadow maps and shadowed images for the same scene at different sampling resolutions. Rows from top to bottom are labeled A, B, C, and D, respectively. Columns from left to right are labeled 1, 2, 3, and 4. Images A1, A2, A3, and A4 are adaptively sampled shadow maps with resolutions at 65x65, 129x129, 257x257, and 513x513, respectively. They are all sampled from a 513x513 square region at the focal plane, with the light sample set a little higher than the highest box as the "viewpoint". The numbers of sample points in the maps are 1453, 3837, 8187, and 17655. When we compare our shadow map A4 with the shadow map of the conventional method, the total sample points used are 17655 to 263169. That is, our method uses only 6.7% of the sample points of the conventional method. With such a small fraction of samples, we are able to generate shadowed images at the same quality level as the conventional method. Row B displays shadow depth images retrieved from our shadow maps in row A. Row C represents our test scenes with hard shadow derived from the respective shadow maps above them. Row D displays our test scenes with soft shadows rendered from 49 (7x7)

light samples and their associated shadow maps at the respective resolution. Note that while the quality of the hard shadows in image C3 is somewhat lower than that in image C4, the quality of the soft shadows in image D3 is as good as that in image D4.

Consider images in row C and row D. In row C, the more samples used, the sharper the hard shadows. The differences of the images in row C are obvious. However, the differences for the soft shadows of the images in row D are not clear. Moreover, when compared with the conventional shadow map, our method uses the shadow map with only 8187 samples in C3, which is 3.1% of the samples in a conventional shadow map. Image D2 uses shadow maps with 1.5% of the samples in a conventional shadow map from a 513x513 square region. In conclusion, our method uses less memory than that of the conventional shadow map for a similar quality.

Nevertheless, since self-shadowing (a surface casting the shadow onto itself) and light leaking (the light ray emitting and passing through a surface without being blocked by the surface) can happen due to under-sampling near the edge, getting a higher sampling density is a remedy to the problem and sharpens the shadow edge retrieved from a single shadow map as well.

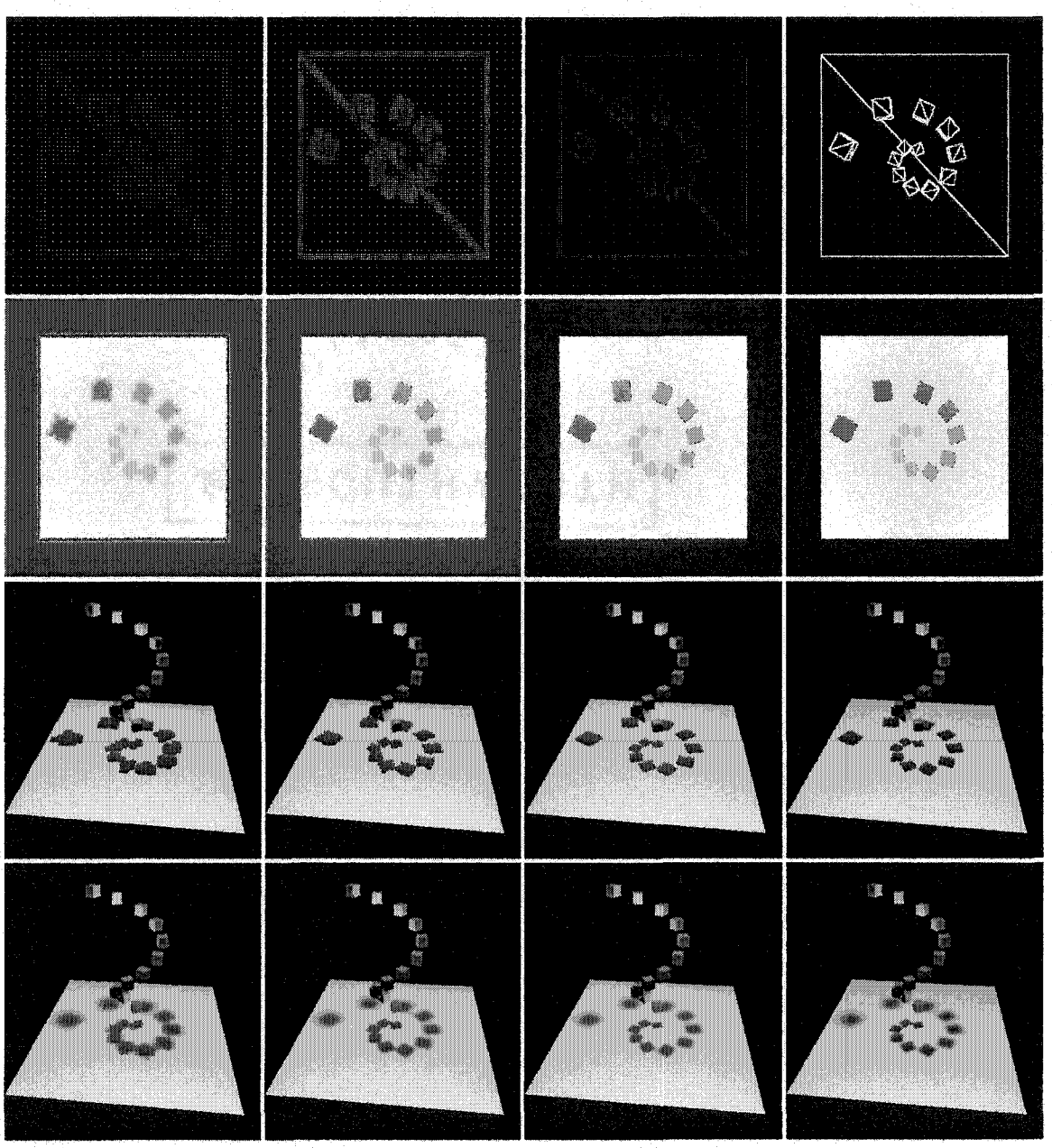


Figure 7: Adaptively sampled shadow maps at different resolutions, the corresponding shadow depth images, hard shadows from a single point light source, and soft shadows from an area light source (from top to bottom).

3.3.2 Adaptively sampled scene map

The scene map, one of the viewing components, is used to store the adaptively sampled scene information for the viewpoint. In the scene map, we store the surface normal, coordinates, and the RGB values of each sampled surface point. Without using the original scene information, we store only the sampled surface points for the predefined viewpoint to create the final image in the visualization stage.

In Figure 8, we display images derived from four adaptively sampled scene maps. We create these scene maps by sampling the 513x513 scene image square in different resolutions with the importance-based adaptive sampling technique introduced in 3.2. From left to right, images are generated from scene maps with 2774, 8561, 25967, and 73259 samples respectively. Creating an image from an under-sampled scene map results in an image that is blurred and has color bleeding on the edges. We find that to generate an image that matches the quality of the original one, the threshold of subdivision must be set to match the resolution of the original image. For example, for a 513x513 plane,

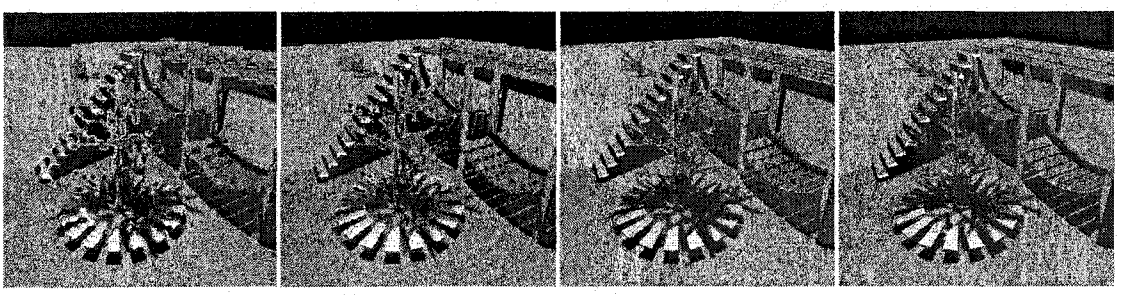


Figure 8: Images rendered from different levels of adaptively sampled scene maps.

setting the threshold to 10 would allow subdivision to reach the pixel level. In the right-most image, we only use 27.8% (73259 out of 263169) of the sample points, in comparison with that of the regular image size at the 513x513 resolution.

3.4 Visualization Stage

In Figure 9, we illustrate the visualization of the synthetic scene, which is derived from the scene map, and describe the frame buffers of the system below. The upper right frame buffer shows an example of the light template with two rectangular light sources. The lower right frame buffer is a differential map representing the movement of a rectangular light source. The attenuation map is updated when a light source is moved/deformed. However, instead of recalculating the effect of all light samples and accumulating illumination values for each surface point, we focus on only those light

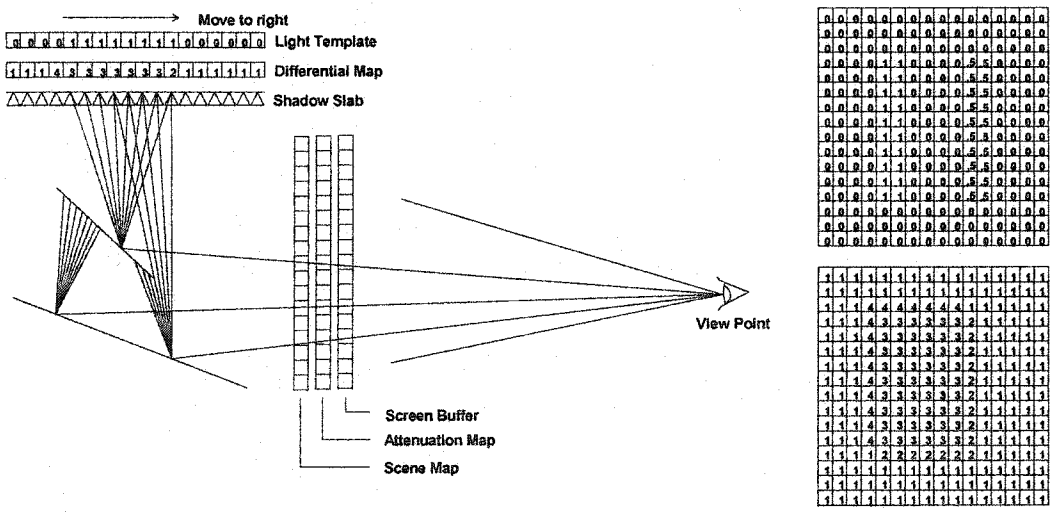


Figure 9: Visualization and frame buffers of the synthetic scene.

samples where changes occur (states 2 and 4 in the differential map) and update the values in the attenuation map accordingly in order to reduce computation time.

3.4.1 Lighting

The light template is a representation of the light source plane. Active light samples and their intensities are recorded in the light template to represent multiple shapes of the working light sources. In the visualization stage, arbitrary light shapes are generated and recorded in the light template. One may move/deform an area light source by modifying the light source in the light template to activate/deactivate the corresponding light samples.

The state of each light sample is recorded in the differential map. The differential map is dynamically updated whenever modifications are made to the light template. There are four possible states in the differential map for each light sample:

1. Not active currently or previously.
2. Active only currently.
3. Active both currently and previously.
4. Active only previously.

3.4.2 Shadow update

The initial state for all light samples in the differential map is state 1. When adding or modifying light sources in the light template, we update the states in the corresponding differential map cells and then trigger the rendering procedures to update the final image. The pseudo code for updating the state of a cell in the differential map is as follows:

```
if active:      state = (state==1 || state==4) ? 2 : 3;
if not active:  state = (state==1 || state==4) ? 1 : 4;
```

The attenuation map is created and updated in the visualization stage (see Figure 9). The initial value of each pixel in the attenuation map is zero. Whenever the differential map is updated, the system updates the values in the attenuation map accordingly. For example, in the differential map, cells with state 2 indicate that their respective light samples are new to the working light (activated) and all screen buffer pixels illuminated by these light samples should increase the illumination values in their appropriate attenuation map cells. Cells in state 4 show that their respective light samples are removed from the working light (deactivated) and all screen buffer pixels no longer illuminated by these light samples should decrease their illumination values in the attenuation map.

To update the values in the attenuation map, we emit rays from the viewpoint through each pixel of the scene map to get the information of the intersected surface point, which is stored in the cell of the scene map, and then trace from the coordinates of the surface point to the light samples with state either 2 or 4 in the respective differential map cells. When each ray traces these light samples from the intersected surface point, some light samples may be blocked by other surfaces (see Figure 9). We first increase the attenuation value based on the calculation result from those unblocked light samples with state 2 in their respective cells in the differential map and then decrease the attenuation value based on that from those unblocked light samples with state 4 in their respective

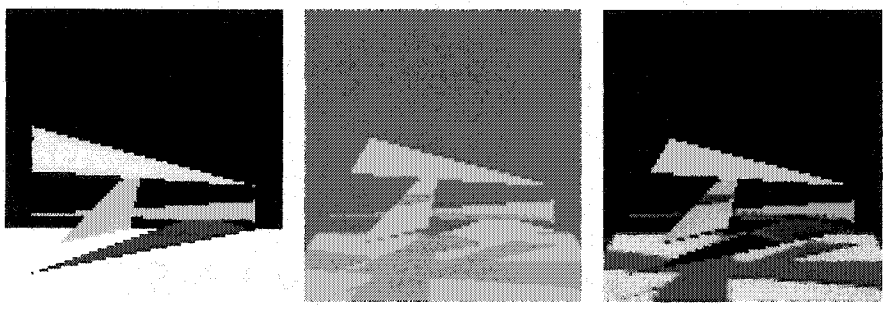


Figure 10: A test image with soft shadow. The right image is created from an original colored image (left) using reduced illumination values in the attenuation map (middle).

cells in the differential map. After the attenuation map is updated, the shadowed image is created dynamically by using the corresponding attenuation values in the attenuation map and by attenuating the color (R, G, B) in each pixel of the scene map. As a result, the soft shadow in our screen buffer is changed as well. In Figure 10, we display a shadowless image, a rendering image of the attenuation map, and a shadowed image of one of the test scene. In Figure 11, we display various soft shadows to show the ability of our method to visualize soft shadows projected by a square area light from different locations.

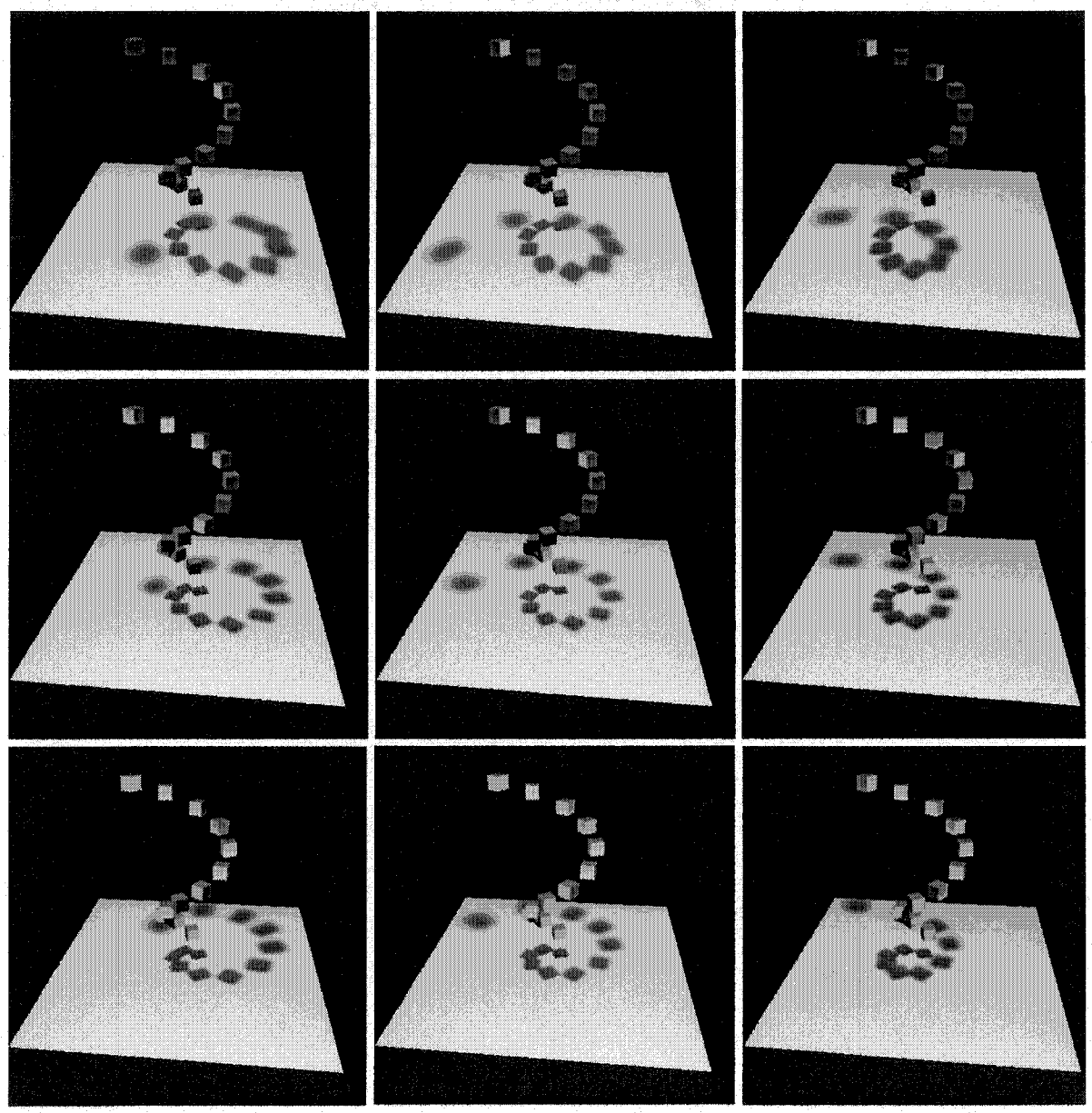


Figure 11: Soft shadows of cubes projected by a square area light source from different locations.

3.5 Implementation

We have implemented our method to generate dynamic soft shadows described in this chapter using C++ on a Pentium III 866Mhz personal computer with 256M RAM.

3.5.1 Pre-processor

We import 3D models into our pre-processor to create the scene map for a given viewpoint and create the shadow maps for the light samples to form the shadow slab. The pre-processor stores the sampling results of the desired depth images for the shadow maps and the scene map in the quad-tree data structure. Each node of the quad-tree represents a square region of the depth image and each square is further divided into four smaller squares if it satisfies the criteria of division (listed in 3.1.2). Each node of our quad-tree contains four pointers that point to its four child nodes. It also contains four integer values representing IDs of intersected triangles from the sampling result at each corner of the square. The root node represents the square region of the whole depth image. We then output the data in these maps to binary files.

3.5.2 Soft shadow visualization

We use two tools to visualize soft shadows of deformable moving area light sources – an interactive shadow viewer and a script shadow viewer. The first one is set up for shadow generating with the interactive graphical user interface. The second one is used for shadow generating without the interactive graphical user interface.

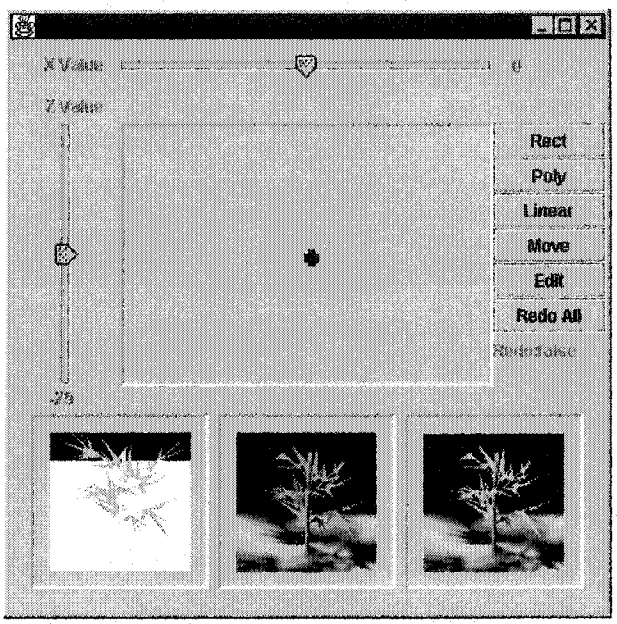


Figure 12: Interactive Shadow Viewer. The upper view port is for the light template editor. The lower left view port is for the shadowless image. The lower right view port is for the attenuation map. The view port at lower center is for the resulting image with soft shadows.

3.5.2.1 Interactive shadow viewer

We create an Interactive Shadow Viewer that is associated with the graphical user interface (see Figure 12) to allow users to visualize soft shadows interactively. In Figure 12, there is a light template editor at the upper portion of the viewer. This editor allows users to create and modify the light sources in order to render shadows. There are three view ports at the lower portion of the viewer. The lower left view port displays the shadowless image created from the scene map. The lower right view port displays the attenuation map after users change the light sources using the light template editor. The lower center view port displays the final soft shadow image.

Binary files from the pre-processor are loaded to access the interactive shadow viewer. Whenever users edit the area light sources in the editor, this editor triggers the system to update the light template and process the soft shadow rendering.

While rendering shadows, the system updates the states of cells in the differential map according to the light template and then updates the attenuation map. Finally, the shadowed image is created and displayed in the view port of the interactive shadow viewer.

3.5.2.2 Script shadow viewer

To view the soft shadows without the interactive graphical user interface, we setup a script file to manipulate the changes of the light sources for image rendering. We create shadow slab for each light source and create a scene map for each viewpoint in the pre-processing stage. We then turn on/off the lights, change the light intensities, and move the light locations-according to the script file. When we render the image by following the scripts in the script file, users can really see the change of the soft shadows from different light sources.

For a complex scene, there may be multiple light sources and viewers may also want to see the scene from different viewpoints. Although, theoretically, there is only one shadow slab and one scene map used in the algorithm, the script shadow viewer, as an application of the algorithm, extends the use of the scene map and the shadow slab to multiple scene maps and shadow slabs. This does not change the classification of the

algorithm as the type 4 algorithm: static viewpoint under dynamic light and static scene (see 2.3).

3.5.3 Results

We create several scenes to demonstrate the results of our method. In Figure 13, the same rectangle area light source (the top row) projects light onto the same scene object (pipe) from different locations, resulting in soft shadows at different positions. In Figure 14, rectangular light sources of various sizes project different sharpness levels of soft shadows. From left to right, top to bottom we can see that when the size of the light sources decreases (the top row), the soft shadow becomes sharper. Images in the second row indicate depth maps of the center point in the light source. In Figure 15, area light sources of multiple shapes generate different soft shadows. Figure 16 displays images with their soft shadows projected from light sources of different shapes from different locations.

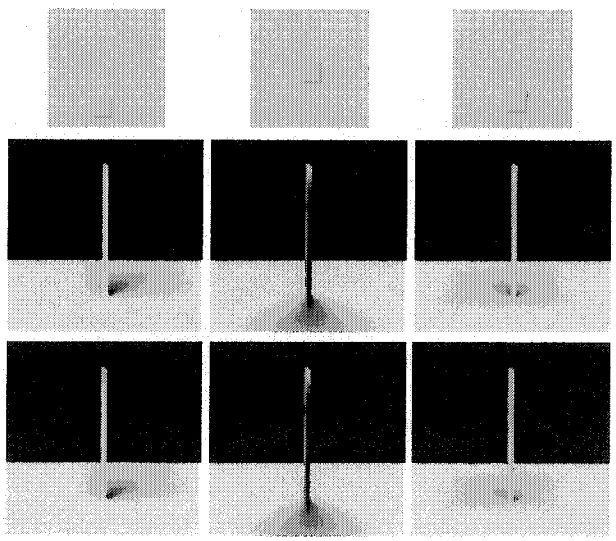


Figure 13: Soft shadows of a pipe generated by the same light source at different locations.

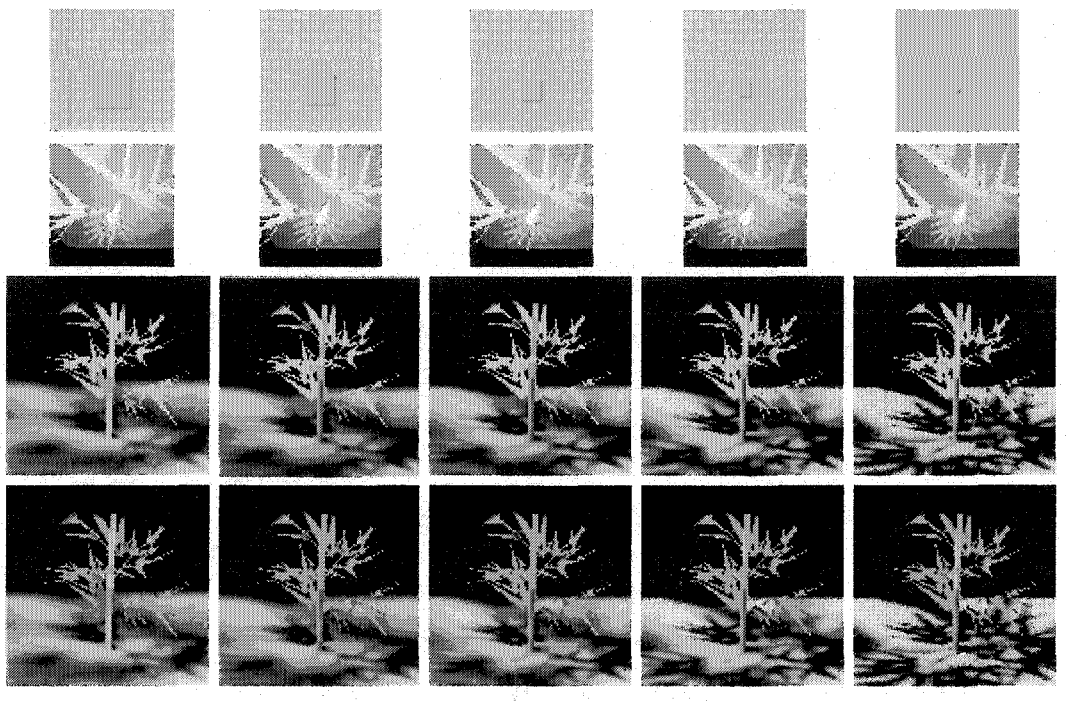


Figure 14: Soft shadows generated by rectangular light sources of different sizes.

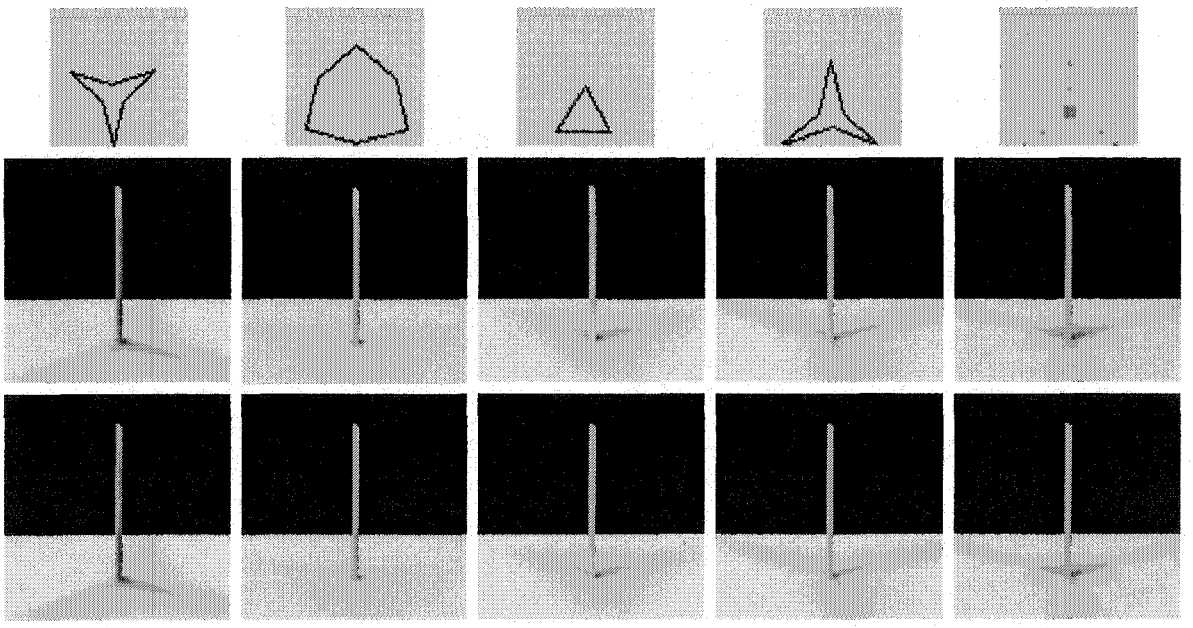


Figure 15: Shadows generated from light sources of different shapes. The upper row shows light templates. The middle row shows attenuation maps. The lower row shows shadowed images.

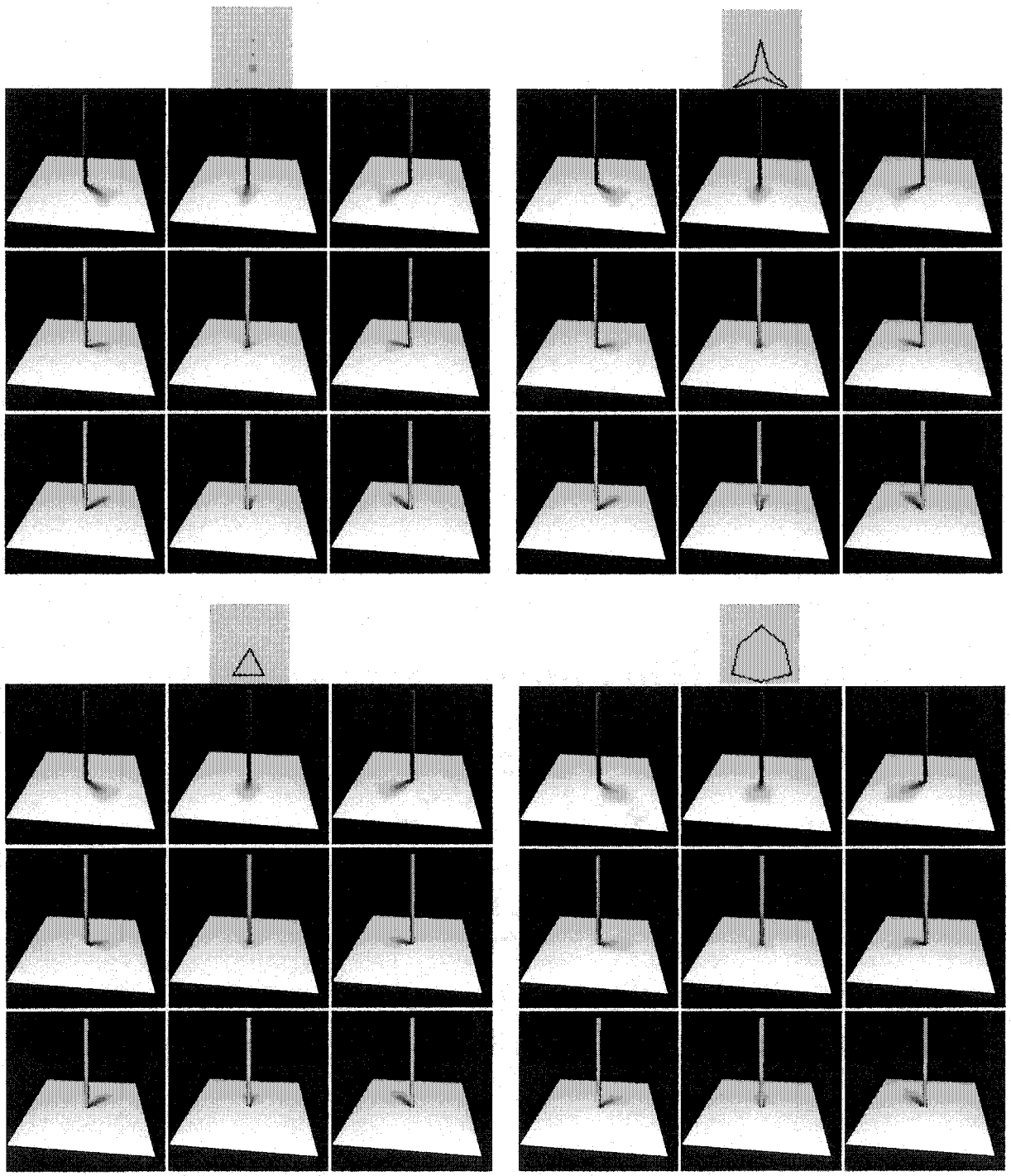


Figure 16: Four sets of shadows generated from area light sources of different shapes and different directions.

In Figure 17, we also create an office model and generate several shadow slabs for different light sources such as the ceiling light, the wall lamp, and the desk lamp. From the top to the bottom are rows A, B, C, and D. From left to right are columns 1, 2, 3, and 4. Rows A and C are rendered from one view point and rows B and D are rendered from another view point. Images A1 and B1 have a single light source projected from outside to simulate moonlight. Images A2 and B2 have a lamp on the wall turned on. Images A3 and B3 have a desk lamp on the left of the image turned on. Images A4 and B4 have a rectangle light that projects downward from the ceiling. Images C1 and D1 have both the wall lamp and a desk lamp on the right turned on. Images C2 and D2 have both lamps on the desk turned on. Images C3 and D3 have the ceiling light and the desk lamp on the right turned on. Images C4 and D4 have both desk lamps and the ceiling light turned on. Notice that in rows C and D, multiple layers of shadows are projected on the floor from different locations and directions of the light sources.

The sizes of the shadow slabs vary depending on the size of the light sources. Shadow map resolutions are different for various light sources and depend on the locations and the directions of the light source, and the locations of the viewpoint. For example, since a ceiling light, (unlike the desk lamp), illuminates the whole room, the shadow map for the ceiling light should be larger than the one for the desk lamp because it contains more details of the scene objects. Moreover, when we put a viewpoint closer to the desk in

order to take a better look of the details, the shadow map of the desk lamp should be larger. Otherwise, artifacts in the left column of Figure 7 will occur because of insufficient sampling.

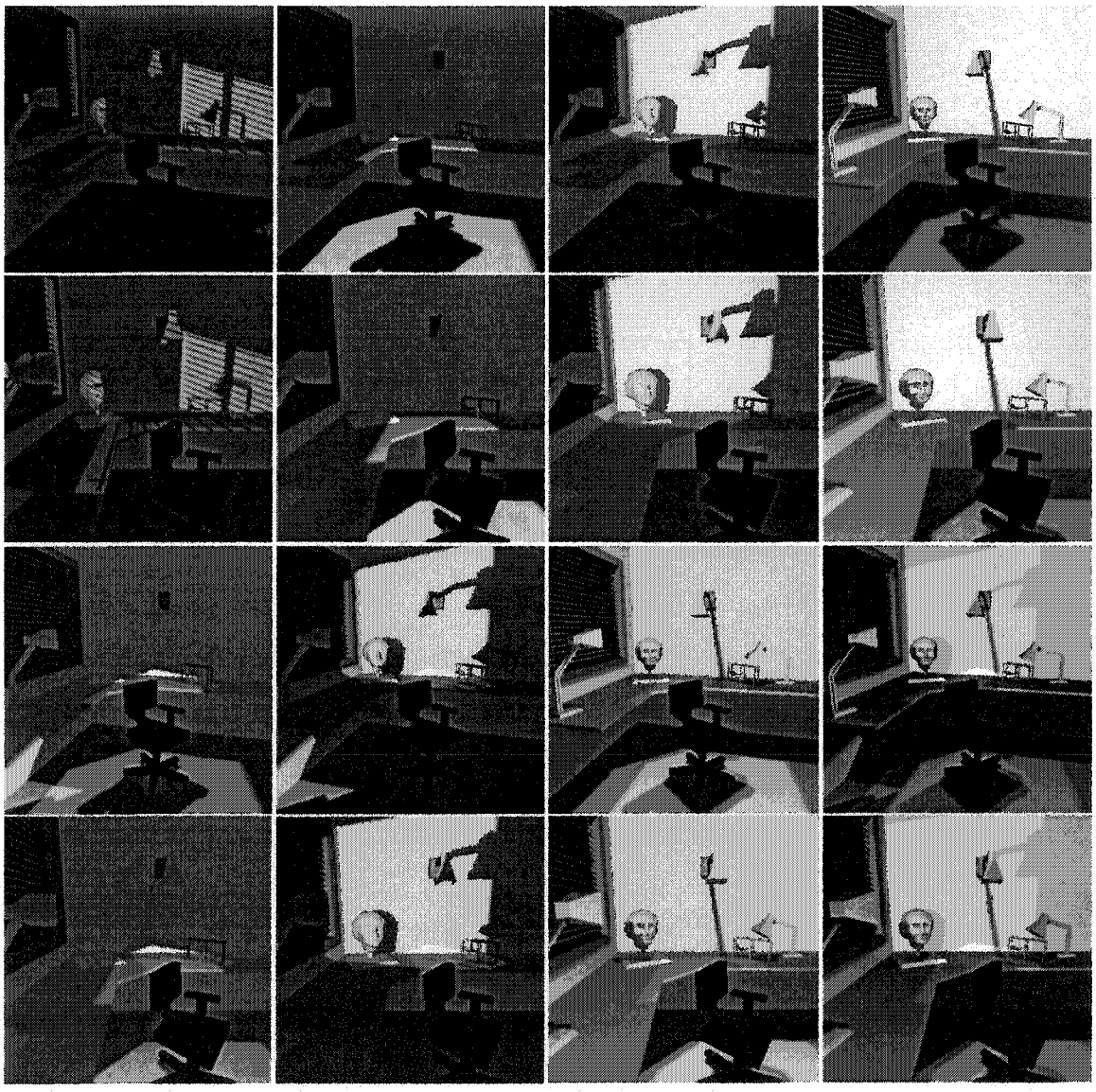


Figure 17: Office scene with multiple light sources.

While visualizing the scene, viewers can turn on and off each light source separately. In Figure 17, we see each image with different lights turned on and off. Lights can be turned on and off, intensity increased, and the color of each light source modified. Since the light and the viewpoint are independent from each other, the same shadow slab is used to render images for different viewpoints. By switching lights on and off, changing the shapes and the sizes of the lights, and adjusting the light intensities and colors, viewers can really visualize the changes of shadows in a synthetic image.

3.6 Summary

We have presented a method for users to visualize soft shadows for area light sources that are movable and deformable through manipulating the interactive shadow viewer or executing the script shadow viewer. The ability to change light shapes and light positions helps the viewer see the differences in soft shadows. The pre-calculated scene map and shadow slab along with the differential map and the attenuation map make it possible to animate the effects of deformable moving area light sources without involving the original scene geometry. The use of the differential map and the attenuation map allows for greater flexibility and efficiency in updating the intermediate shadow information when the light source moves.

In this method, the computation time and the memory requirement for the resulting image depends on the image size and the number of modified light samples. Through several steps, we reduce memory consumption in comparison to the conventional shadow map approach. First, the importance-based adaptive sampling technique introduced in 3.2 is used in this method to reduce the number of samples in the shadow maps and the scene map. Second, we implement the shadow slab using a flat plane in our method. However, other surface such as semi-sphere (a dome) can be used as a shadow slab surface as long as we calculate shadow maps for sample points on the surface. Since each shadow map in the shadow slab are handled independently and each cell of the scene map is independent from other cells, parallel computation may be utilized to increase the speed of processing.

Bilinear interpolation in the adaptively sampled shadow map and scene map lookups is used in the method. Third, instead of retrieving all the active light samples to get the attenuation value for each cell in the attenuation map whenever the light source moves, the system will update the attenuation values by retrieving the shadow maps of those light samples with states 2 and 4 in the corresponding differential map cells. Thus, we avoid retrieving shadow maps of those light samples with unchanged states (states 1 and 3) after the light sources are changed.

However, the degree of saving memory and calculation time by using importance-based adaptive sampling to capture important samples is subject to the complexity of the scene and the image size. More complex scenes require more important samples that may result in no savings of memory or time at all. Moreover, the performance in our visualization stage depends on the complexity of the scene geometry, the size of the shadow slab, and the size of the screen buffer. Therefore, since we simulate the soft shadow by accumulating illumination of each light sample in the area light sources, we are not able to render the image in real time especially when modifying multiple light samples at a time. When we move the light source quickly, the number of the modified light samples in the differential map becomes larger. Our method can hardly update the image quickly enough under this rapid changing condition. Furthermore, our method is not suitable for scenes with dynamic objects and dynamic viewpoints. Resolutions to these situations are future study topics.

Chapter 4

Forward Area Light Map Projection

To accurately compute the penumbra produced by an area light source requires the measurement of the visibility of the area light source from each sampled surface point in the scene. This is very expensive. Therefore, a lot of effort has been devoted to speed up this shadow calculation. Amongst the various efforts to achieve good approximation with significantly reduced computational burden, Agrawala et al. [Agrawala00] use image-based techniques to create soft shadows and their method is an efficient approach. In this chapter, we present the "forward area light map projection" method that renders soft shadows efficiently for synthetic scenes. This method can be seen as a reverse approach to [Agrawala00].

Unlike Agrawala et al. [Agrawala00], who retrieves the attenuation values in the layered attenuation map in a backward fashion from the viewpoint, we render soft shadows by projecting light attenuation values in each layer of the area light map forward onto the screen buffer. The backward retrieving approach (see right drawing of Figure 18) projects eye rays into the scene and then retrieves the attenuation values from the layered map by comparing the depth value in a map cell with that of the intersected surface point. This method retrieves a map cell only when the eye ray intersects the scene at a corresponding surface point. In contrast, the forward projection approach (see left

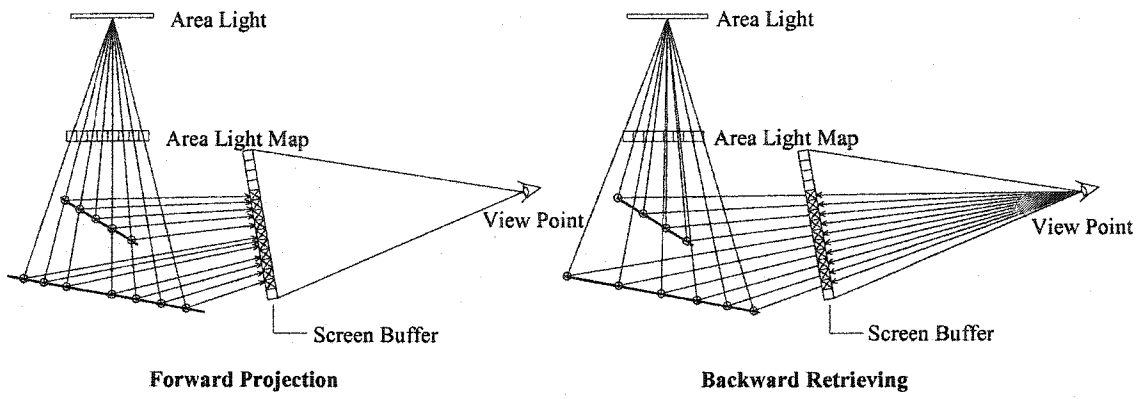


Figure 18: Forward projection and backward retrieving of the area light map.

drawing of Figure 18) projects all points stored in the area light map from their coordinates on the surfaces onto the screen buffer.

In addition, although the structure of the layered area light map in our method is similar to the layered attenuation map introduced in [Agrawala00], the creation of the map and the rendering of the soft shadows differ. In short, our two-stage approach method (pre-processing and soft shadow rendering stages) is especially efficient in creating a series of high-quality soft-shadowed images from different viewpoints for a static scene/ light environment.

To present our method, we begin with the description of the system outline in section 4.1. Then, we state the creation of the layered area light map in the pre-processing stage in section 4.2. Section 4.3 describes the techniques used in the soft shadow rendering stage. Section 4.4 describes the implementation of the method. Section 4.5 demonstrates the results, and finally the summary is in section 4.6.

4.1 System Outline

There are two stages in this method – pre-processing stage and soft shadow rendering stage (see Figure 19). In the pre-processing stage, we create a two-dimensional layered area light map, which is similar to the LDI used in [Agrawala00]. For each map cell, we store information of surface points that are intersected by a ray that starts from the center of the area light and goes through the center of the map cell. The stored information of each point is the surface normal, the coordinates, the polygon-mesh ID where the point resides, and the visibility ratio of the area light source seen from the point. Here, we propose an alternative method to calculate visibility ratios (attenuation values) for sampled surface points. This method is slower but more flexible in sampling for arbitrary shaped light sources when compared with [Agrawala00].

Once the layered area light map is created, a series of high quality soft shadowed images are created in the soft shadow rendering stage. The creation of a shadowless image for a given scene is efficient due to the use of the existing graphic pipeline and hardware accelerator. Therefore, the computation time of the final image with soft shadows depends mainly on the size of the layered area light map and its projection onto the screen buffer. Although it takes time to create the layered area light map, this method quickly creates a series of high-quality soft shadowed images for a fixed scene.

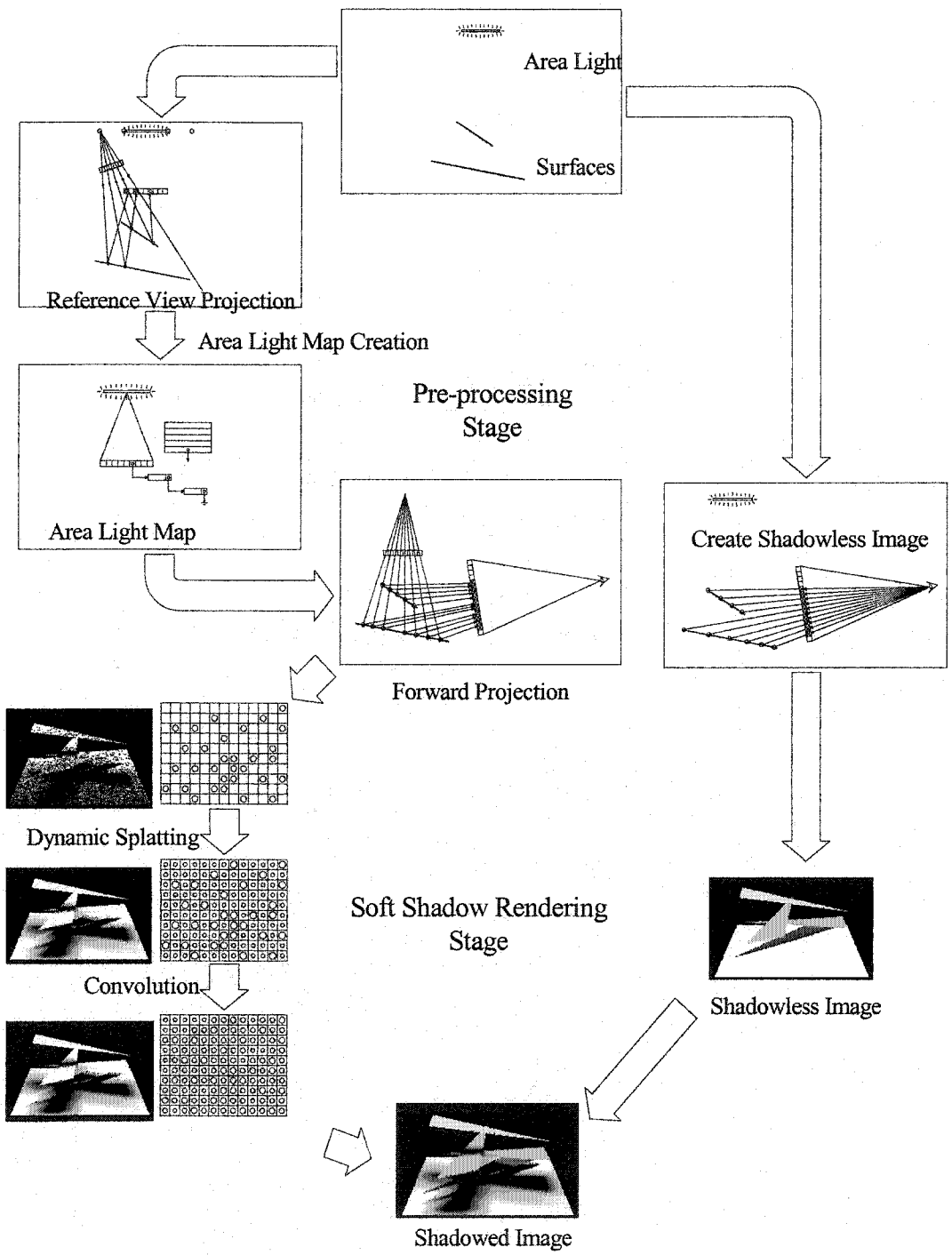


Figure 19: Outline of forward area light map projection.

4.2 Pre-processing Stage

The major task in this stage is to create the layered area light map. In doing so, we use a visibility function, which back-projects from each sampled surface point to the light source, to calculate the ratio or percentage of the area light seen from the point. The measured ratio is stored in the layered area light map.

Here, we first describe the visibility function in section 4.2.1 and then introduce the structure and the creation of the layered area light map in sections 4.2.2 and 4.2.3.

4.2.1 Visibility function

Shadow algorithms are visibility functions. For hard shadows, the visibility functions return either "TRUE" or "FALSE" by detecting whether the surface point is lit or not. For soft shadows, the visibility functions return the visible ratio of the area light source from the sampled surface point in the scene.

The visibility function in this method is an image-space back-projection approach, with the sampled surface point set as the viewpoint and the center of the area light source set as the viewing destination of the viewpoint. Meanwhile, the whole area light should be inside the view frustum. The visibility function makes use of OpenGL graphics function to render the light and scene objects into a view port. We first reset the color of the view port to black, render the area light object onto the view port as a white object, and then count the total number of white pixels inside the view port. We count the white pixels again after the rendering of all other objects in black color in the scene. The

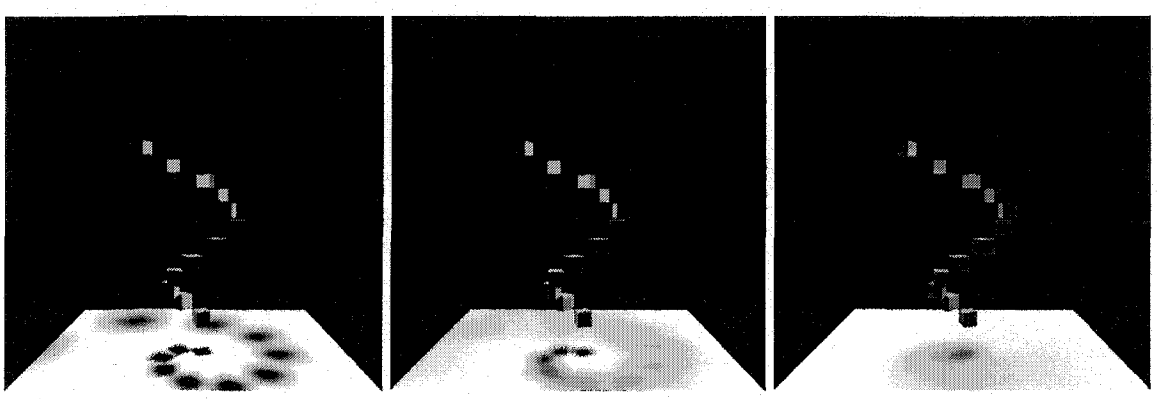


Figure 20: Soft shadow images rendered using image-space back-projection approach from different sized area light sources. The rendering time is independent from the size of the area light sources.

visibility ratio of the area light source from the sampled surface points is determined by dividing the second count with the first count. If the size of the view port is too small, the pixel count is limited to a small value, and banded shadow artifacts may occur in the resulting image due to insufficient illumination scales.

Figure 20 contains three soft shadow images rendered using this back-projection method from three-sized area light sources. The rendering time for the three images is identical since the operations involved are virtually independent from the size of the light sources. Figure 21 presents two images using different view port sizes for the visibility function to measure the ratio of the light sources. The image on the left results from a sufficiently big view port. Although it takes a longer time to generate the image by using the big view port, the image has a smoother shading transition in the shadow area. The image on the right results from a relatively smaller view port that generates the image

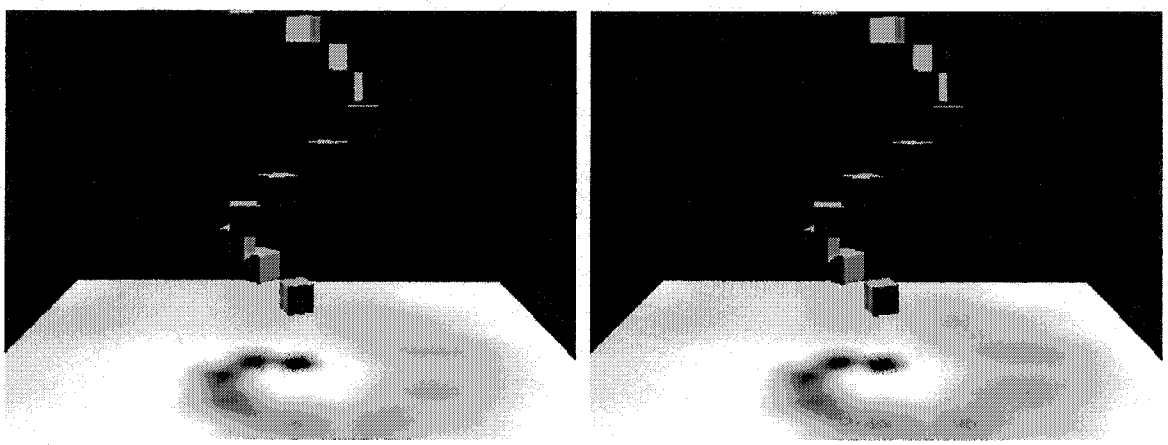


Figure 21: Soft shadowed images using different sizes of view port in the frustum of the visibility function. The soft shadow on the left displays a smooth transaction whereas the one on the right exhibits banding artifacts due to under-sampling from the smaller sized view port.

faster but produces banding artifacts in the soft shadow area. Moreover, with this visibility function of image-space back-projection, we can use arbitrarily shaped 3D object as the light source because we use OpenGL graphics function to render the light onto the view port.

4.2.2 Structure of the layered area light map

The traditional shadow map stores a single depth value in each cell of the depth buffer. Nevertheless, the layered depth image (LDI) [Shade98] stores multi-layered depth values in each buffer cell. The depth values are used to compare with the depth value of the sampled surface point to decide if the point is in shadow. Moreover, Fernando et al. [Fernando01] use a combination of polygon-mesh IDs and depth comparisons to perform visibility determination in their adaptive shadow map.

Similar to the layered depth image, our layered area light map stores visibility information in a multi-layered buffer. In addition, our layered area light map stores surface normals and polygon-mesh IDs. In Figure 22, we show the structure of the layered area light map. There is a base point for the map that is normally at the center of the light source. For each cell of the layered area light map, there is a linked list of layer elements. In each layer element that represents a sampled surface point, we store the point's polygon-mesh ID, attenuation value (visibility ratio), surface normal, and coordinates. Instead of using the depth value as the layer index like [Agrawala00], we use a polygon-mesh ID for each layer element's index.

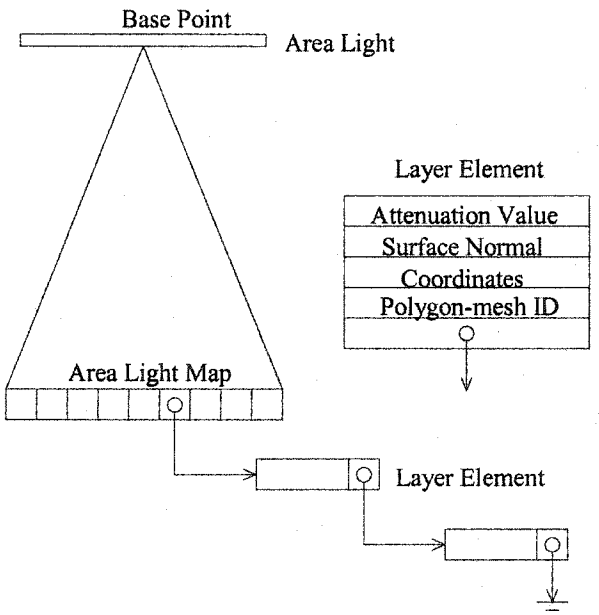


Figure 22: Structure of the layered area light map.

4.2.3 Layered area light map creation

One way to select surface points for inclusion in the layered area light map is to construct "reference views" from a number of reference points (see Figure 23). Each reference view represents a discrete sampling of the object surfaces with each pixel corresponding to a surface point. Our method places no restriction on the position and the number of the reference points when creating the layered area light map, though we normally set the locations of the reference points on or near the area light source. The more reference views we use, the higher quality soft shadow we are able to achieve in the rendering stage. Increasing the amount of reference views increases the quality of the soft shadows, but adversely increases memory usage and computation time.

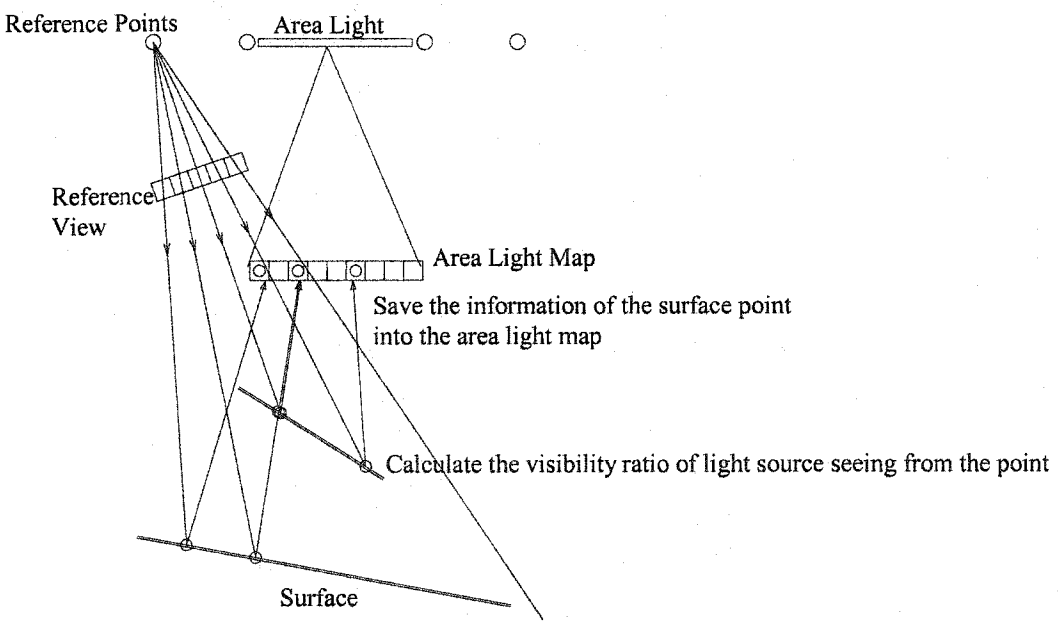


Figure 23: Creation of the layered area light map.

To create the layered area light map (see Figure 23), we first place the view at each reference point and then shoot rays through each pixel of the reference view to obtain the intersected points on the object surface. We then perform the visibility function from each sampled surface point to the center of the area light to calculate the visibility ratio seen from the point. Before carrying out back projection to calculate the visibility ratio with respect to the point, we first check the corresponding polygon-mesh ID against the elements of the linked list of the found area light map cell. The area light map cell represents the intersected point on the map. This cell is determined by tracing a ray from the surface point through the area light map to the center point of the area light. If the polygon-mesh ID is not on the cell's link list, we calculate the attenuation value of the sampled surface point and then insert a new element into the list and index the element by the polygon-mesh ID. Otherwise, we skip the tracing ray to prevent duplicate calculation.

Note that even though calculation time for back projection is relatively long for the first few reference views, only a fraction of the surface points generated from the remaining reference views need further calculation. This is because a large number of points have already been processed and stored in the layered area light map.

4.3 Soft Shadow Rendering Stage

After the creation of the layered area light map in the pre-processing stage, a series of soft shadow images are rendered from any desired viewpoints in the soft shadow rendering stage. The light attenuation values stored in the layered area light map represent the distribution of area light energy to the sampled surface points. Since these surface points are captured from the object surfaces, they are similar to those points described in the point rendering algorithms that captures 3D surface points from 3D scenes [Grossman98] [Pfister00] [Rusinkiewicz01] [Wand01] [Stamminger01]. In other words, the layered area light map is a data structure for a collection of the points sampled from a 3D scene. However, there is one difference, the layered area light map does not sample points from surfaces facing away from all the reference points. (Usually these surfaces are in the umbra area since the reference points are set near or on the light source.)

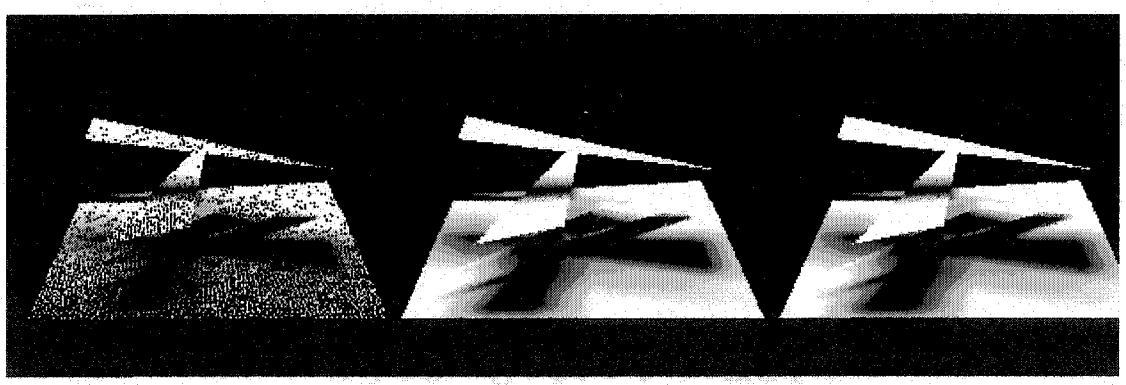


Figure 24: Three steps taken in Forward Area Light Map Projection. Images from left to right are results at the steps of point projection, dynamic splatting, and convolution.

In this stage, the screen buffer stores the temporary information when we generate soft shadow images. The screen buffer consists of three-layers which contain; RGB values, attenuation values, and polygon-mesh IDs. We first render a shadowless reference image by graphics pipeline to produce RGB values in the RGB layer and polygon-mesh IDs in the polygon-mesh ID layer of the screen buffer. We then project the sampled surface points of the layered area light map onto the screen buffer.

The "hole" problem in point rendering occurs here too, since our forward area light map projection for soft shadow visualization is similar to other point rendering methods that render object images from sampled surface points. Splatting and convolution are used to handle this problem and improve shadow quality. Figure 24 illustrates the results of forward area light map projection with a test scene. The images from left to right are results at the steps of point projection, dynamic splatting, and convolution. 4.3.1, 4.3.2, and 4.3.3 describe the next three steps used in this stage.

4.3.1 Point projection

The first step of the forward area light map projection is point projection. All elements of the surface point list at an area light map cell contain, light attenuation values, as well as other values. In the left drawing of Figure 18, from the layered area light map, we retrieve each cell of the layered area light map and project all the surface points (exclude those facing away from the viewpoint) stored in each cell's linked list from their coordinates on the surfaces onto the screen buffer. Those pixels without a

point projected are invalid pixels and become "holes". Because two or more surface points may be projected onto the same pixel of the screen buffer, we accumulate the attenuation values of projected points at the respective pixel of the attenuation layer of the screen buffer if their polygon-mesh IDs are the same as the one kept in the polygon-mesh ID layer at that pixel's location. We also increase the value of a pixel's counter by one each time a surface point has been accumulated. Upon completion of surface point projection, the attenuation value at each pixel's location (if associated counter is not zero) is updated to store the averaged value (accumulated attenuation value divided by the corresponding counter).

4.3.2 Dynamic splatting

Dynamic splatting occurs after the point projection step. Because of the insufficiently projected points on the attenuation layer of the screen buffer, holes occur in the point projection step, which also happens in point rendering. Dynamic splatting is used to fill these holes. We first check the pixel counter to see if there is any surface point deposited in a pixel. If the counter value is zero, which indicates no point is deposited; we derive an attenuation value for this invalid pixel from its neighbors. That is, we accumulate the attenuation values from valid pixels (exclude those that belong to different surfaces of the invalid pixel) surrounding this invalid pixel within a small kernel of 3x3 array. The averaged value of the attenuation values is deposited into the invalid pixel. If there is no valid pixel found, we then increase the kernel size and check again until it reaches the

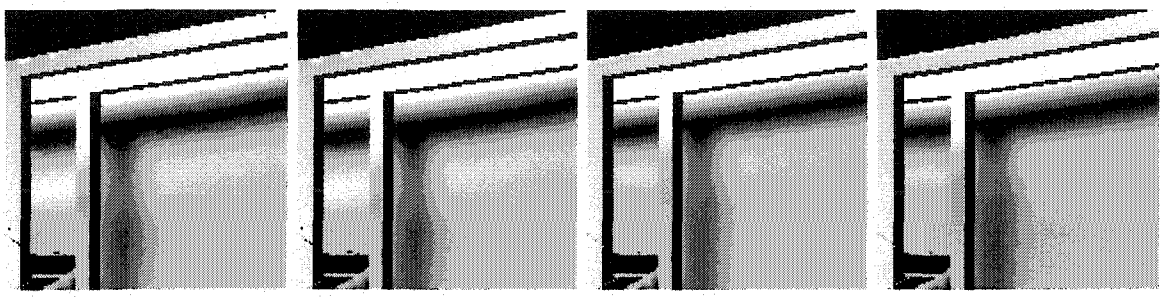


Figure 25: Splatting with insufficient sampling results in blocky artifacts (left) while splatting with denser sampling results in better quality (middle right). In the above two cases, convolution increases the smoothness of the soft shadows (middle left and right).

predefined maximal size. Note that the polygon-mesh ID is always checked to exclude pixels that belong to different surfaces and larger kernels may produce blocky artifacts (see Figure 25).

4.3.3 Convolution

Finally, the convolution step occurs in the soft shadow rendering stage. After the holes are filled, blocky artifacts often appear in under-sampled shadow area. Then convolution/filtering of the attenuation values occurs to produce high quality penumbras. Convolution methods convolute the attenuation layer of the screen buffer to improve the quality of the soft shadow in the buffer. The final attenuation value stored in each pixel is applied to modulate RGB values in the screen buffer, which generates the final image.

Agrawala et al. [Agrawala00] state that insufficient sampling causes blockiness and use simple bilinear filtering of four neighboring attenuation map values to diminish these

artifacts. However, this method blurs the edges of objects. Here, we use the polygon-mesh IDs in the screen buffer to identify the boundaries between surfaces. Our convolution operation is applied only to pixels that have the same polygon-mesh ID. This assures that pixels of a soft shadow, which is close to an edge and separates two object surfaces, is not be mixed with the pixels on the nearby surface, since this would blend the shadow with the nearby surface's shading.

In Figure 25, from left to right, the first image displays the blocky soft shadows due to insufficient sampling; the second image displays improvement from object surface-based convolution; the third image is from a more densely sampled area light map; and the fourth image displays the result of applying object surface-based convolution.

4.4 Implementation

We have implemented our soft shadow visualization method on a Pentium III 866MHz personal computer with 256M RAM, using OpenGL as the base renderer. The implementation is purely a software approach. We first develop an ALM-creator to generate the layered area light map. Then a soft shadow renderer is applied to quickly project sampled surface points from the area light map onto the screen buffer.

To implement this method, we import the 3D models into our program to create the layered area light map and output the results to binary files. We use the image-space back-projection approach described in section 4.2.1 to calculate the visibility ratio of the area light source with respect to each sampled surface point and store the values in the layered area light map. The map is also used to create the right image in Figure 26 using backward retrieving.

The time needed to create the area light map depends largely on the number of the reference points and the resolution of the reference view. In our method, for a rectangular area light source, we use nine reference points: one is located at the center of the area light, four at the corners of the rectangle, and four along the two extended diagonal lines outside the area light source. The sizes of the reference views, the layered area light map, and the screen buffer are all 512x512.

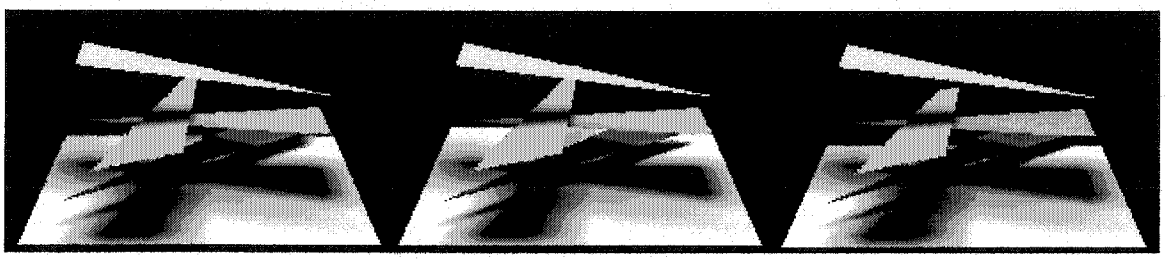


Figure 26: Soft shadows generated by forward area light map projection (left), ray-traced back-projection (middle), and backward area light map retrieving (right).

Figure 26 illustrates three soft shadow images generated by our forward area light map projection, the ray-traced back-projection, and the backward area light map retrieving, respectively. Retrieving the layered area light map, and implementing a backward retriever create soft shadow images on the right hand side. This works in a way that is consistent with the layered attenuation map approach in [Agrawala00]. One can see that some banding artifacts appear in the soft shadow area in the right image. Because splatting and convolution are used in the left image, the image displays a very smooth soft shadow. There is an indiscernible difference between the soft shadows of the left and middle images.

Table 2: Performance of three test scenes on PC. The pre-processing stage takes most of the running time. Our forward projection approach uses double to quadruple the running time of backward retrieving, which mainly depends on the ratio of the shadowed areas to the entire screen.

Scene	Triangles	Area Light Map Size	Pre-processing Time	Forward Projection	Backward Retrieving	Ray-traced
Spiraling Boxes	148	512x512	256 sec	1.892 sec	0.425 sec	284 sec
Garden	1,273	512x512	1,675 sec	2.183 sec	1.405 sec	1,446 sec
Bunny	16,301	512x512	6,386 sec	2.312 sec	0.611 sec	3,364 sec

4.5 Results

Table 2 shows a sample of performance data for three test scenes (see Figures 27, 28, and 29). When comparing our forward area light map projection method with the backward area light map retrieving method, we find that although our approach doubles (or more, depending on the size of shadow area in the resulting image) the average per-

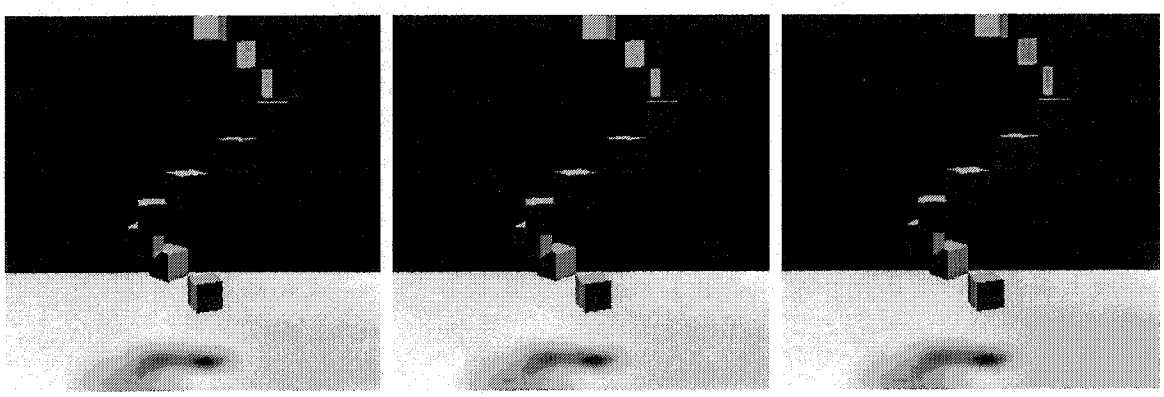


Figure 27: Shadow images of spiraling boxes. The images are created by forward area light map projection (left), ray-tracing (middle), and backward area light map retrieving (right).

frame rendering time, it is able to produce better soft shadows. In particular, when there are only a small number of sample points in the layered area light map, our approach can generate higher quality soft shadows than the backward area light map retrieving method. Generally, higher quality soft shadows occur when the more surface points are sampled, although the projection time is increased.

Although forward projection takes more time to render a frame than backward retrieving, it is still much faster than ray tracing in terms of per-frame computation. In addition, it is able to produce high-quality soft shadows that are comparable to those created by the time-consuming ray-tracing approach. This timesaving feature becomes more evident when a series of images are created by moving the viewpoint.

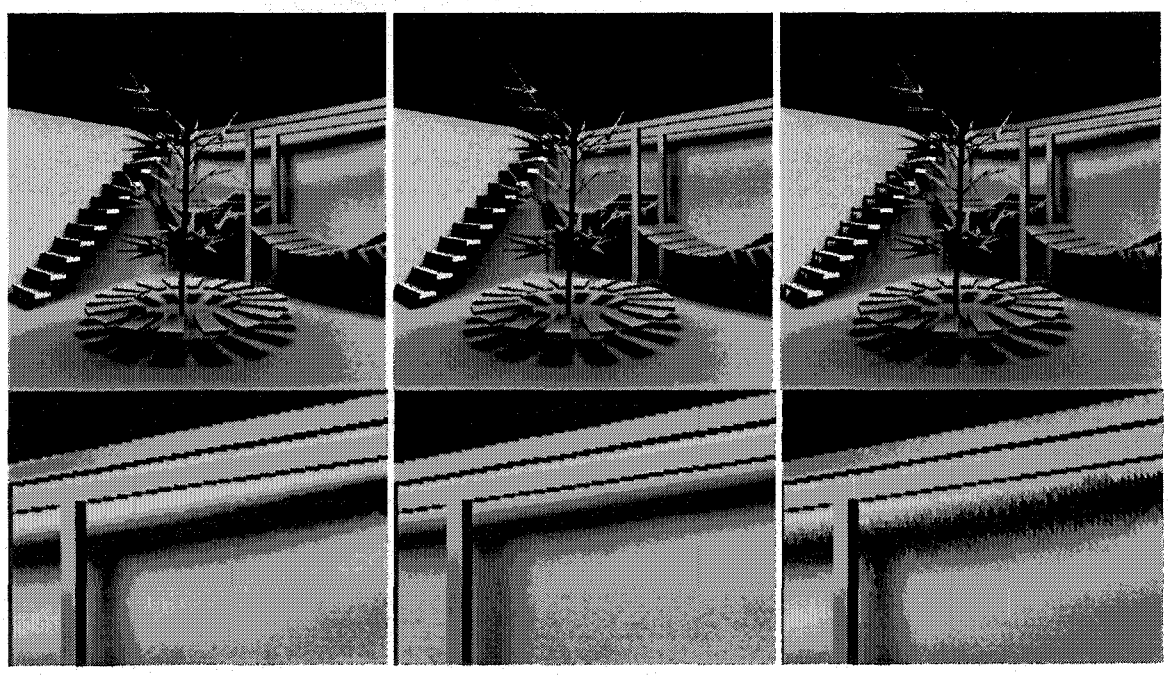


Figure 28: Shadow images of a garden scene. The images are created by forward area light map projection (left), ray tracing (middle), and backward area light map retrieving (right).

In Figure 27, the quality of soft shadows created by the three approaches is similar since we sample sufficient surface points and store them in the layered area light map. In Figure 28, the shadows of the garden model are created from an under-sampled scene, with focus on the wall section. The forward projection approach (the left image) is able to cope with the inadequate surface points to generate better soft shadows than the backward retrieving method (the right image). Figure 29 displays a soft shadow image of Stanford Bunny generated by our method. The model contains 16,301 triangles, which is more complex than the models of the Spiraling Boxes and the Garden in Figures 27 and 28.

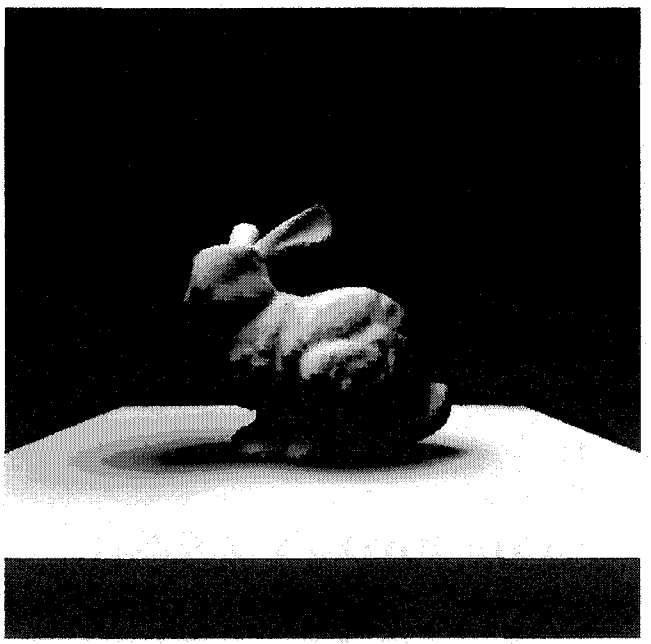


Figure 29: Soft shadow of Stanford Bunny.

4.6 Summary

The layered backward retrieving used in the layered attenuation map approach [Agrawala00] is a fast soft shadow algorithm. The pre-computation stage occurs for a few seconds and then the soft shadows are displayed at several frames per second. Although this approach achieves interactive rendering rates, it causes banding artifacts because of under-sampling. Radiosity, on the other hand, generates high quality soft shadow images but is time consuming. We do not compare our approach with radiosity because radiosity is a global illumination algorithm and ours is a local illumination algorithm. (Although ray tracing, which we use to generate images in order to compare with forward area light map projection approach and backward area light map retrieving approach, is a global illumination algorithm, we calculate information without including the inter-reflection of light between surfaces during image generation.)

Although our forward area light map projection approach seems to take longer than the layered attenuation map approach [Agrawala00] in the pre-processing stage and it appears to have a slower frame rate, it produces higher-quality soft shadows that are comparable to the benchmark soft shadows generated by ray tracing.

Note that splatting and convolution are the two steps applied in our approach for improving the quality of soft shadow rendering. Without these two steps, the results from the forward area light map projection is worse than those from backward retrieving because of the uneven distribution from the point projection (see Figure 24).

In conclusion, soft shadow generation is a time consuming task. If the two algorithms of Agrawala et al. [Agrawala00] represent the two extremes of a spectrum, then our algorithm falls somewhere within these extremes, with our rendering speed being closer to that of their layered attenuation map approach and our shadow quality being closer to that of their ray-tracing based method.

Chapter 5

Priority Point Lists for Point-based Object Rendering

Point rendering is also an important research topic in computer graphics. Although the point rendering method, "priority point lists for point-based object rendering", introduced here is not a soft shadow rendering method, it is an application of adaptive sampling (The shadow map generation that was introduced in our first method in chapter 3 is another application of adaptive sampling.) Meanwhile, this point rendering method also uses the dynamic splatting technique that was introduced in our second method in chapter 4.

This point rendering method consists of two stages: the point sampling stage (the pre-processing stage) and the point-based object rendering stage. This point-based method is used for dynamic rendering of three-dimensional objects with complex shapes. The points are sampled from the mesh objects. This method stores only a small number of sample points and still renders a high quality image by adaptive sampling of points from each triangle of the mesh object (which is different from the dense sampling used in the conventional point rendering algorithm).

To present this method, we start with reviewing the previous work on point rendering in section 5.1, followed by the descriptions of the point sampling and the point-based

object rendering stages in sections 5.2 and 5.3. Finally, the implementation of the method is displayed in section 5.4 and the method is summarized in section 5.5.

5.1 Previous Work on Point Rendering

Levoy and Whitted [Levoy85] use points as rendering primitives. Cook et al. [Cook87] propose decomposing details to points in the image space during rendering. Grossman and Dally [Grossman98] generate a dense set of surface points that contain the information of color, depth, and normal, then render these points to reconstruct the surface. Pfister et al. [Pfister00] propose another method using surfel (surface element) technique to reconstruct a fragment of a surface, then generate points in a fixed resolution from three orthogonal directions and then merge these points into a smaller set of points. Rusinkiewicz et al. [Rusinkiewicz01] present QSplat that uses points as its rendering primitives. QSplat is designed with the intent of visualizing scanned models that contain a significant amount of fine details at scales near the scanning resolution. However, they expect QSplat to be less effective for scenes with large and smooth regions.

Wand et al. [Wand01] reconstruct surfaces from a dynamically chosen set of randomly sampled surface points, which are chosen with the probability proportional to the projected area of the objects. Stamminger and Drettakis [Stamminger01] present an integrated approach that sample densities locally according to the projected size in the image. Other approaches that differ from sampling points in the pre-processing stage are; local sample re-generation that generates samples dynamically for procedural and

complex objects that are dynamically changed. Zwicker et al. [Zwicker01] also convert geometric models into point-based objects in their pre-processing stage. This surface splatting technique focuses on high quality texture filtering. Chen and Nguyen [Chen01] and Cohen et al. [Cohen01] combine points and polygons for rendering. Kalaiah and Varshney [Kalaiah01] use "Differential Point" to capture the local differential geometry near a sampled point, and efficiently renders the surface as a collection of local neighborhoods.

5.2 Point Sampling Stage

Points are the primitive element for point rendering approaches. First, point samples are captured in the point sampling stage (pre-processing stage) to obtain points for object rendering. Appropriate point selection is the key to the dynamic splatting technique used in the point-based object rendering stage. While uniform resolution and mass dense point sampling are used in most of the point rendering approaches, the point sampling in our method, samples important points from the mesh surface adaptively to reduce the number of points and increase rendering efficiency.

In this first stage, we sample surface points from each triangle by dividing the basic square plane, which contains the edges of the triangle, into four sub-squares. Each sub-square is then divided into smaller sub-squares recursively until it reaches a pre-defined threshold. This information is stored in a quad-tree. After the division, the information of the center points of these squares is saved into different priority point lists according to

point levels. The data structure used in the stage and the sampling procedures are stated in section 5.2.1 and section 5.2.2.

5.2.1 Quad-tree data structure

The quad-tree (see Figure 30) is used as the temporary data structure to store the divided sub-squares for the basic square plane. The basic square plane is the plane where the triangle is located and contains the edges of the triangle of the mesh objects. Each node of the quad-tree represents a square region on the basic square plane. The attributes stored in the node are; the level of the node in the tree, the "isObject" flag, the intersection point of the sample ray at each corner, and pointers to the node's four children.

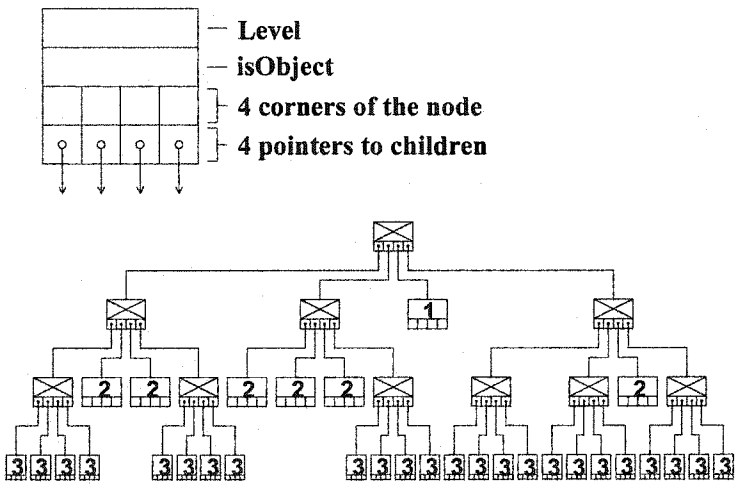


Figure 30: Quad-tree data structure in the adaptive sampling approach. Nodes with level numbers are valid nodes that are leaf nodes with "isObject" flag on. The resulting points are generated from valid nodes.

There are valid nodes and invalid nodes in the tree. A leaf node with the "isObject" flag on is a valid node and the center point of its square is stored as the resulting point. The quad-tree is not needed in the object rendering stage since all the resulting points are stored in the priority point lists. In Figure 30, the number in each node indicates the level of the node. The smaller the number is, the higher its priority is. The resulting points are generated from those leaf nodes with the "isObject" flag on.

5.2.2 Sampling

Our sampling method is a per-triangle based approach. Every triangle in the scene is sampled. The sampling procedures for each triangle stay within its local coordinate system. For each local coordinate system of the triangle, the plane where the triangle lies is the x-y plane of the coordinate system and serves as the basic square plane. The four steps for sampling are listed below:

1) Set up the location of viewpoint in the triangle's coordinate system.
2) Choose the basic square for sampling.
3) Adaptively sample the triangle and store the results in the quad-tree.
4) Store the center points of the valid nodes in the priority point lists.

Firstly, in each triangle's local coordinate system, we set the viewpoint in front of the triangle. Secondly, we choose the basic square on the x-y plane and the triangle should be inside this square.

Thirdly, similar to the technique introduced in 3.2.2, we sample the triangle adaptively. The difference is that we sample one triangle at a time. The whole basic

square that contains a triangle is the default root of the quad-tree. We first divide the root node's basic square into four sub-squares. These sub-squares are the node's children. Then, we divide each sub-square into four even smaller squares and each of the four squares will be divided recursively until the division reaches our pre-defined threshold. That is, the division stops when the size of the leaf node square reaches a predefined minimum size. After the division reaches this threshold, we update the value of the "isObject" flag in each leaf node depending on the result of the intersection between the four corner points of the node and the triangle. The initial value of the "isObject" flag is "off". The "isObject" flag is set to "on" whenever there is at least one corner point of the node intersecting the triangle. If the four corner points of a node do not intersect with the triangle, this indicates that there is either no edge in the square region or there are two edges of a sharp angle in the square region. This conditions forces continued sub-division.

Now we start edge detecting for the following division decision. For each leaf node, when the value of the "isObject" flag is "on" and at least one of the four corner points does not intersect the triangle, this indicates that there are edges in the square region that need to be divided further. After the division, we set the value of the "isObject" flag as described previously.

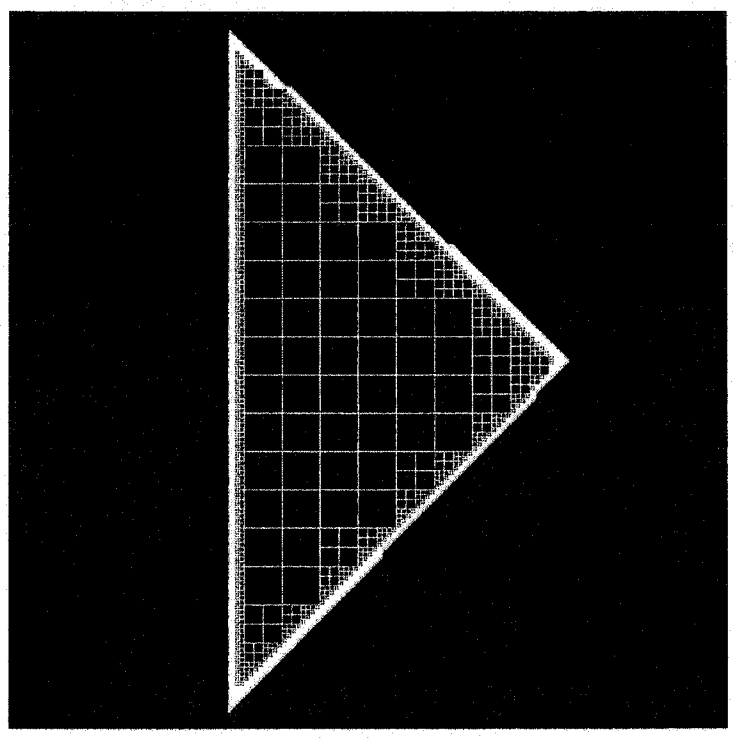


Figure 31: Adaptive sampled triangle. The resulting squares of the valid nodes are drawn from the quad tree of a triangle that is sampled adaptively. The point located at the center of each square is the point that is saved in the node's priority point list. Note that squares closer to the edges are smaller and denser.

Lastly, we generate points for the priority point lists. We retrieve all the valid nodes (any leaf node whose "isObject" flag is "off" is not a valid node) and add the squares' center points (with coordinates in the world coordinate system, level of subdivision, surface normal of the triangle, and material information of the triangle) to the appropriate priority point lists according to their node levels.

In Figure 31, we display a triangle divided into different levels (sizes) of squares. Points at the top level of the quad-tree are the highest priority and points at the bottom

level of a quad-tree are the lowest priority. This level attribute is used in the dynamic splatting stage as one of the decoding factors of base size of splats and level-of-detail control. The center point of each square is stored in its point lists. We have determined that 3 to 4 levels of lists are suitable for most of the object rendering cases.

In conclusion, the points may be perceived as being sampled at different frequencies, since the points are generated from different sized squares (see Figure 31). In other words, we use lower sampling frequencies in the center area of the triangle and higher sampling frequencies in the area close to the triangle edge.

5.3 Point-based Object Rendering Stage

The point-based object rendering stage follows the generation of the priority point lists. In this object rendering stage, the priority point lists are used to render the splats in an orderly way. We render the splat of each point in the priority point lists into an alternative z-buffer with points of higher priority rendered first. The dynamic splatting technique only uses points from the priority point lists in the rendering process, regardless of the scene objects. Furthermore, the priority point list provides the flexibility of rendering optimization by excluding the rendering of tiny surfaces that are smaller than a pixel on the screen.

To avoid producing holes in surface reconstruction, splatting is widely used in point rendering [Grossman98], [Rusinkiewicz01], [Wand01], [Stamminger01]. A dynamic splatting technique (see 5.3.3) is used in this point rendering method. Though splat shape

can be rectangle, round, or oval [Rusinkiewicz01], we use a rectangle splat in our implementation. Although points at each level share the same base size of splats, points with the same base-sized splats project splats onto the screen buffer at different sizes due to their distances from the viewpoint. Therefore, the distance between the visualized point and the viewpoint is one of the factors for splat size determination. The width and the height of the splat are also modified according to the surface normal of the visualized point.

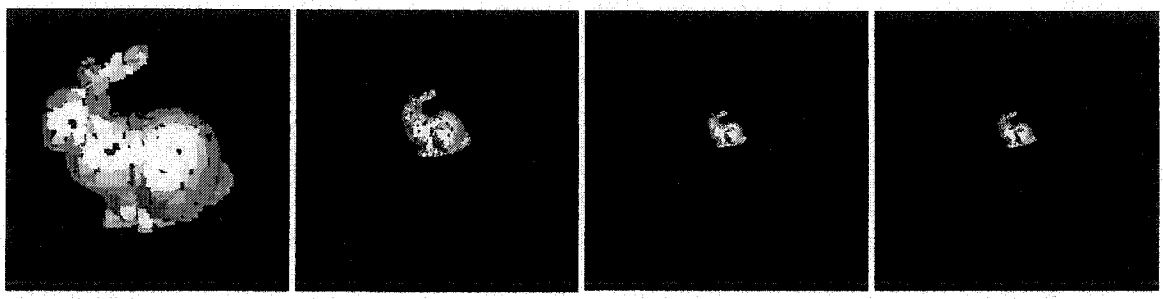
Not all the points are used for rendering. There are several steps for point selection and surface rendering. We select points that pass through the back-face culling step and the detail-level control step, and render the selected points onto the screen in the dynamic splatting step. The descriptions of the three steps are stated in section 5.3.1, section 5.3.2, and section 5.3.3.

5.3.1 Back-face culling

Each point contains its surface normal when we create it. This surface normal is used to check if the point faces away from the viewpoint. All points facing away from the viewpoint are invisible and not used in rendering. Thus, the step of back-face culling is employed to filter out these unused points from the point lists.

5.3.2 Detail level control

After the points pass through the back-face culling step, the remaining points face the viewpoint and are ready to be rendered. For large data sets, primitives are often projected



d=100 d=700 d=1300 d=1300 (all levels)

Figure 32: Rendering of the splats of level-1 list from different distances. The right image is a rendering result using points in all levels of the priority point lists at distance 1300.

to less than one pixel size. To accelerate rendering, most of the detailed points that are far away from the viewpoint (normally in the lower level priority point lists) are not rendered even though they pass the back-face culling step, especially when the sizes of splats they project onto the screen are smaller than a single pixel. We avoid the rendering of these detailed points according to a detail-leveled threshold and their distances from the viewpoint.

That is, if the point is far away from the viewpoint and the projected splat is smaller than a single pixel, it will not be rendered because the higher priority splats from the priority point lists have rendered the region already and the rendering of the point is too small to be seen on the screen. Therefore, we use detail level control to filter out those points before rendering.

However, there is an exception when the point is the only point generated from a very small triangle. This exception is because when a group of small triangles is located so close to each other they consume a visible area that cannot be ignored. In Figure 32, we

display three rendering results (left, middle left, and middle right) of the splats of level-1 list, in which the viewpoints are located away from the center of the object in different distances. The rendering result for points in all levels of priority point lists with the viewpoint at the distance of 1300 is on the right. There is an indiscernible difference between the right image rendered from all levels of priority point lists and the middle-right image rendered from level-one list only.

5.3.3 Dynamic splatting

All points that have passed the previous steps are now rendered to the screen buffer. The rendering size and the shape of each splat is decided by the point's surface normal and location, the level of the list that the point belongs to, and the distance between the point and the viewpoint. The surface normal sets the shape of the splat, the point level decides the base size of the splat, and the distance from the viewpoint decides the scaling factor for the splat.

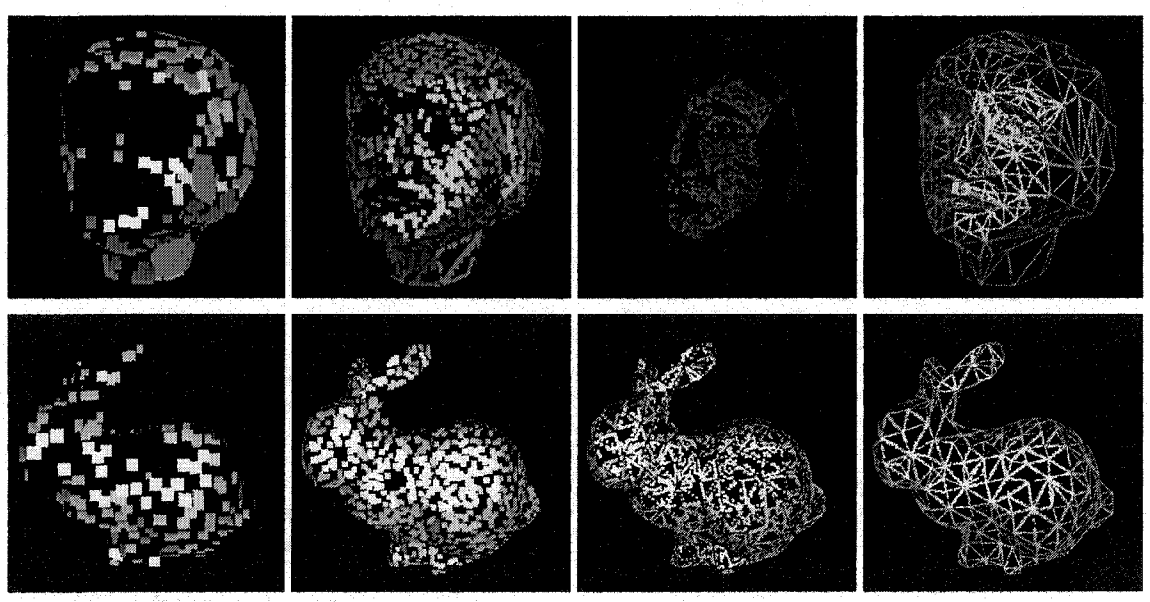
Our dynamic splatting uses a multi-sized splatting technique (section 5.3.3.1) according to each point's base size of splat and the priority of each point list (section 5.3.3.2) together with an alternative z-buffer to render points (section 5.3.3.3).

5.3.3.1 Multi-sized splatting

[Rusinkiewicz01] and [Wand01] set "refinement" (splat size) for each level. They only render the splats of one level whose identical size is fixed according to the predefined refinement for that level in a single rendering stage. Our method uses a

dynamic splatting technique that allows multi-sized splats for multi-leveled rendering at the same stage.

In each splat level, we derive various width and height of splats from the projection of the splats onto the screen buffer according to the surface normal of the sample points, base size of splats, and the distances between the sample points and the viewpoint. During rendering, we render all splat levels from the top to the bottom into the z-buffer orderly. In Figure 33, from left to right, we display four levels of splats rendered separately. The bigger the splat size is, the smaller the number of sample points is.



Base splat size: A: 8x8 B: 4x4 C: 2x2 D: 1x1

Figure 33: Splatting of each priority point list of sampled points for the color-coded human head and bunny. Each layer's base splat size and the actual size of each splat is adjusted according to the view and the surface normal of the corresponding sampled points.

5.3.3.2 Priority of the point lists

We create priority point lists from sampled points according to their levels in the point sampling stage. No quad-tree traversal is necessary in our rendering stage since all the important points are already in the priority point lists. While in the rendering stage, points in the same list reside at the same level and the higher-level point list has rendering priority over the lower-level lists. Note that the base size of splats in each priority point list varies from one list to the next. Larger base-sized splats are rendered to the z-buffer first.

We demonstrate the splat rendering of each priority point list of sampled points for both the human head and the bunny in Figure 33. Viewers can see that the images on the right demonstrate the splats rendered from the lowest priority point list, which are the smallest and located the closest to the edges of the triangles. While the images on the left demonstrate the splats rendered from the highest priority point list, which are the largest and located the closest to the center of the triangles. Even though points at the same level have splats of the same base size, the shapes and sizes of splats vary in the leftmost images.

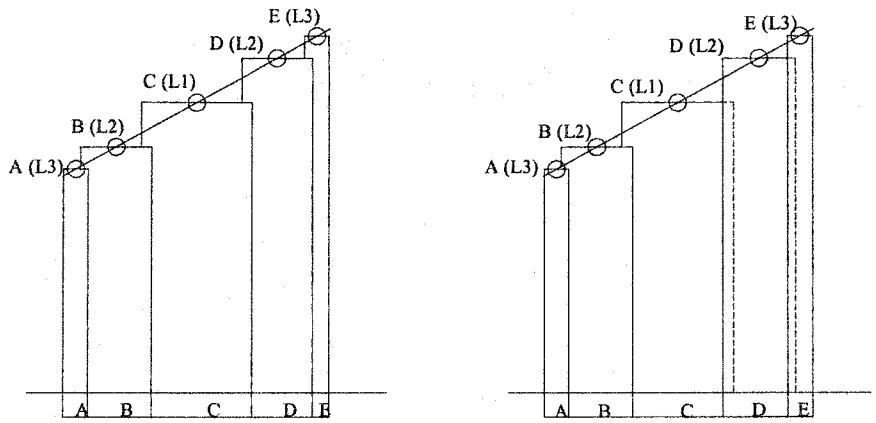
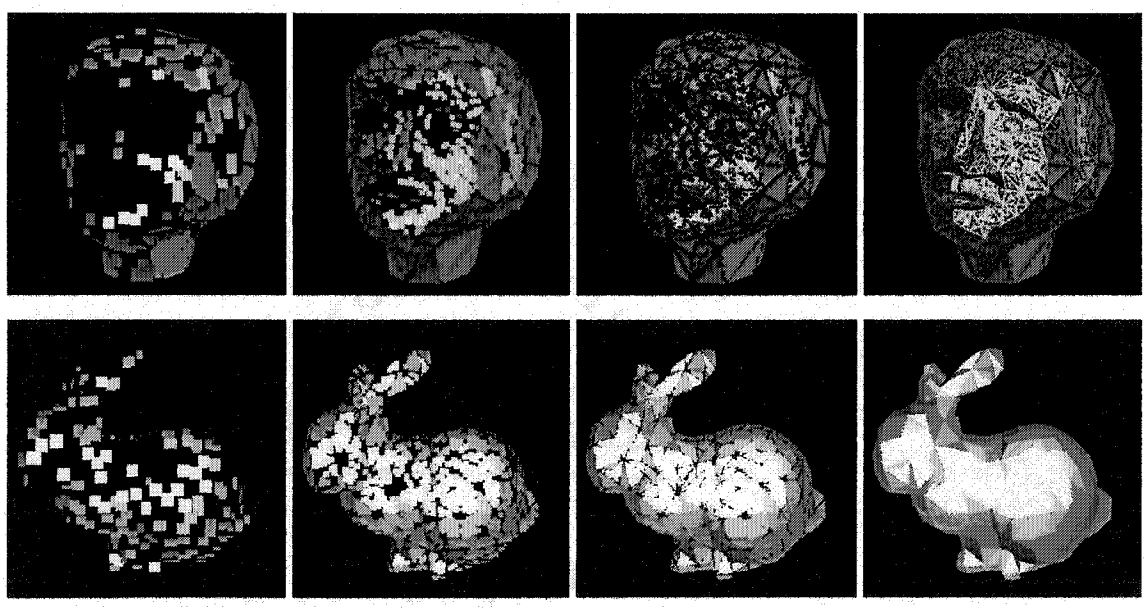


Figure 34: Update of the z-buffers. Unlike the conventional z-buffer, the lower-level splats should cover the upper-lever splats when they are close to each other in the alternative z-buffer.

5.3.3.3 Alternative z-buffer rendering

The conventional z-buffer rendering technique renders points in space by comparing their depths with the values kept in the appropriate locations of the z-buffer. This technique updates the depth values in the z-buffer only when the depth values kept in the buffer are larger than those of the visualized points are. In the rendering stage, the upper-level (larger size) point lists are rendered first. A dynamic threshold for z-buffer rendering is added here for the rendering of splats of different levels. The z-buffer algorithm is used in the rendering of the highest level of the point list. However, in the following lower-level point lists, there is an adjustable threshold added for the depth value comparison. In Figure 34, we display the rendering results of different level points in the conventional z-buffer and the alternative z-buffer. In the conventional z-buffer

rendering technique, if there are two splats close to each other, with the higher-level splat (A(L3)) in front of the lower-level one (B(L2)), the higher-level splat will obscure the lower-level splat in the overlapped region. In this situation, the lower-level splats that are used to render the detail region will be covered by the higher-level splats. However, in our approach, when the higher-level splat is closer to the viewpoint, the lower-level splat is rendered on top of the higher-level splat provided the two splats are close to each other. In order to do so, we set an adjustable threshold to update the depth value of the following lower level (and smaller) splat in the z-buffer if the depth difference between the two splats is within the threshold.



Base splat size: A: 8x8 B: 4x4 C: 2x2 D: 1x1

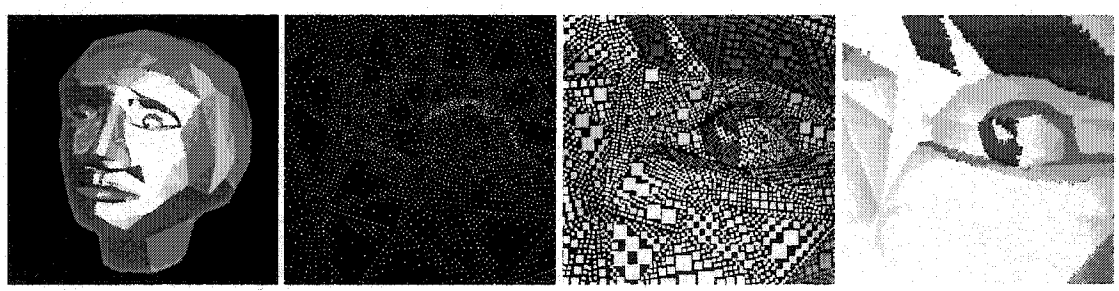
Figure 35: Rendering of the human head and the bunny using dynamic splatting. The human head is a non-uniformly sized triangle mesh object and the splats are color-coded for easy identification. There are holes in the two left images of the top row because the detailed points are stored in the lower-level lists. The uniformly sized triangle mesh object, Stanford bunny in the bottom row contains fewer holes.

In Figure 35, from left to right (both human head and bunny), we render splats into the z-buffer with one list at a time and from higher to lower levels. This displays the differences after the rendering of each list. First, column A (base splat size 8x8) is rendered into the z-buffer. Higher-level splats render the major area in the center of the triangles while lower level splats fill in the detailed portion of the object surface. In columns B, C, and D, the z-buffer values are set according to the adjustable threshold.

5.4 Implementation

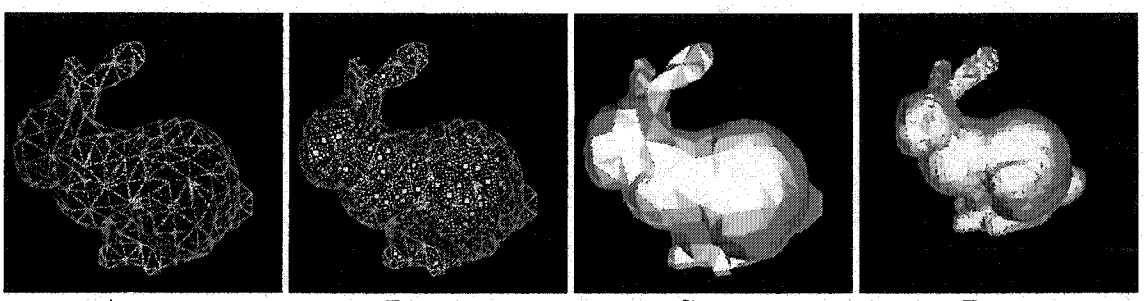
We have implemented a method to create images using dynamic splatting. First, we create an adaptive sampling program using C++ to generate sample points from the mesh models. The resulting points from this program are stored in a binary file. Second, we create a dynamic splatting renderer to read the binary file for image rendering. The program renders images dynamically according to the location of the viewpoint and the view direction. Both programs run on a Pentium III 866MHz personal computer with 256M RAM.

Images in Figure 36 are the rendering results of our method using a human head model and Figure 37 are the rendering results of a bunny model downloaded from Stanford University. In Figure 36, image A displays the rendering result of the human head from the priority point lists using dynamic splatting, image B displays sampled points around the right eye region of image A, image C reveals the location and the shape of each splat from points in image B by reducing the size of each splat, and image D



A B C D

Figure 36: Rendering of the human head. The rendering result of the human head (A) using adaptively sampled points (B) derived from the mesh object. We render image C using reduced splat sizes to show the pattern of the splats. Splats in image D are in actual sizes.

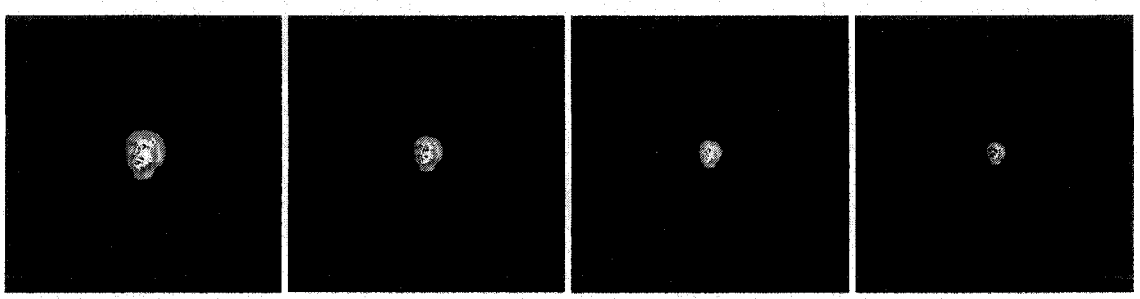


A B C D

Figure 37: Rendering of the bunny. The rendering of the bunny using adaptively sampled points (A) derived from the mesh object. Image B is rendered using reduced splat sizes to reflect splat pattern. Splats of image C are in actual sizes. Image D is the result of rendering with a higher resolution model.

displays the actual size of splats. In Figure 37, image A displays sampled points facing the viewpoint, image B displays splats at reduced sizes, image C displays the actual sized rendering of the bunny, and image D is the result of rendering the bunny model at a higher resolution.

We first filter out the points that face away from the viewpoint by back-face culling. When the viewpoint is very far away from these points, the size of some points' splats projected on the screen is smaller than a single pixel. We can then filter out these points (not including level-1 points) by using detail level control. Figure 38 illustrates the rendering results of the human head located at different locations far away from the viewpoint. The total number of sampled points generated from the human head mesh object is 91,496. The distances of the object to the viewpoint from left to right are 2000, 3000, 4000, and 5000, respectively. The actual numbers of points used for splatting after

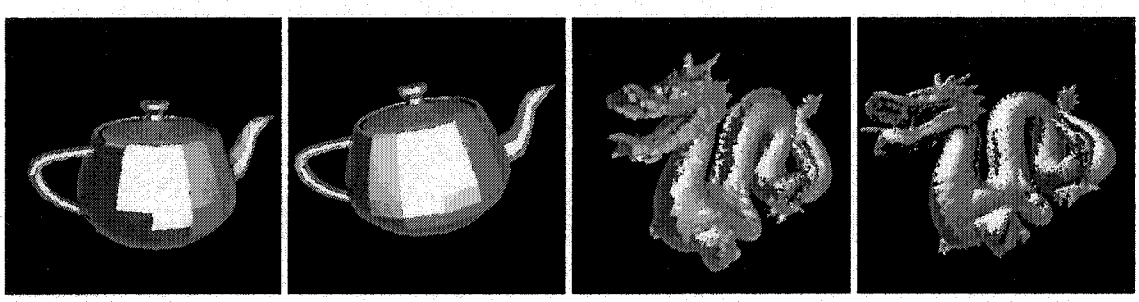


Column:	A	B	C	D
Distance:	2,000	3,000	4,000	5,000
# Points:	52,151	10,519	10,505	2,919

Figure 38: Rendering results of the human head located far away from the viewpoint. The human head is a collection of 91,496 points with four levels of priority lists. Distance refers to the distance from object to view location and #Points refer to the number of points used to render each image. Points not rendered are excluded by back-face culling and detail level control.

back-face culling and detail level control are 52151, 10519, 10505, and 2919, respectively.

Point sampling resolution determines the quality of the resulting image. In Figure 39, we sample points from Utah Teapot model with 1,024 triangles at two resolutions. Image

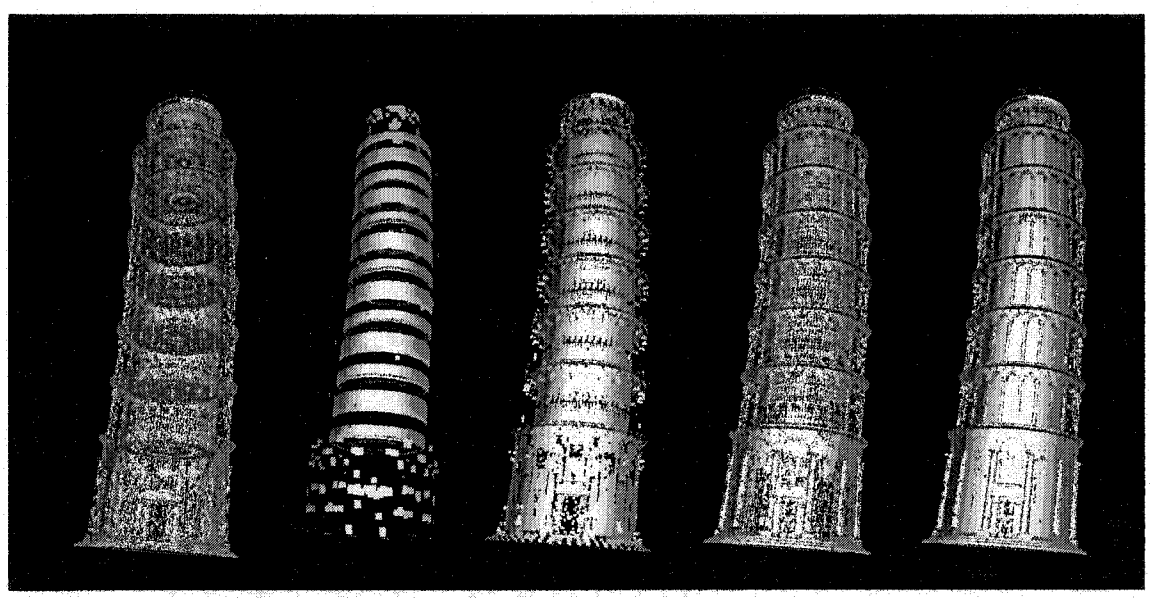


Column:	A	B	C	D
Resolution:	33,552	384,828	11,102	47,794

Figure 39: Rendering results of Utah Teapot and the dragon. The rendering results of Utah Teapot are sampled at a lower resolution (A 33,552 points) and a higher resolution (B 384,828 points). The dragon images are rendering results from the lower resolution model with 11,102 triangles (C) and higher resolution model with 47,794 triangles (D).

A is the rendering result from 33,552 points and B is the rendering result from 384,828 points. In addition to point sampling resolution, the resolution of the model itself is another factor for the quality of rendering. In Figure 39, the dragon images are the rendering results of points generated from an 11,102-triangle model (C) and a 47,794-triangle model (D), respectively. Clearly, image D is of higher quality than image C.

Our approach succeeded with a very complex object. We generated 2,239,674 points in total from sampling the Tower of Pisa Model, which are derived from three priority



Column:	A	B	C	D	E
Splatting:	0	4x4	2x2	1x1	Dynamic
Level:	All	1	2	3	All

Figure 40: Point rendering results of Pisa Tower in various sized splatting. From left to right, column A is the point rendering result of no splatting, column B is the rendering result of 4x4-based splatting from level 1 list, column C is the rendering result of 2x2-based splatting from level 2 list, and column D is the rendering result of 1x1-based splatting from level 3 list. Column E image is the rendering result of all level lists using our dynamic splatting method.

point lists with 37,127 points in level one, 467,183 points in level two, and 1,735,364 points in level three and their base sizes are as small as 4x4, 2x2, and 1x1 respectively. Due to the fine details of the model, we use these small splats for rendering. In Figure 40, from left to right, Image A is the rendering result of all points facing the viewpoint without splatting. Images B, C, and D are the rendering results of splatting from level 1, level 2, and level 3 lists, respectively. Image E is the rendering result using our dynamic splatting method, with a total of 1,215,345 points out of 2,239,674 points.

5.5 Summary

We present here a point rendering method that samples fewer points using the adaptive sampling technique to reduce memory consumption and uses the dynamic splatting to generate images from those sample points efficiently. This is a point-based method for dynamic rendering of three-dimensional objects with complex shapes.

[Rusinkiewicz01] and [Wand01] render images according to the points of a single selected level. That is, no more than points of one level will be rendered in the same image. The higher (or bigger) the splat level is (that is, the lesser amount the points are used), the rougher the object in the image will be. In other words, when smaller splats are used (close to one pixel), the image will be clearer. Of course, small splats require more points than larger splats. Instead of retrieving points in the same level and rendering splats for those points like [Rusinkiewicz01] and [Wand01] do, we use a multi-leveled and dynamically sized splat method at the visualization stage, so the small base-sized splats are used only when the points are close to the view location according to the detail level control. Therefore, while the methods of [Rusinkiewicz01] and [Wand01] only render the splats for points in a single higher-level list, our method also renders detail points in lower-level lists (with smaller base-sized splats) to accomplish the rendering of the fine detail region.

In our point cloud, most of the points are for detail rendering. For any object in a closer location, detail points are required. However, if objects are located far away from

the viewpoint, rendering must consider the following two situations. First, when there are groups of small triangles that are close to each other, even though each of them is very small and can hardly be seen, they may still consume a visible area. Therefore, holes occur when their splats are skipped from rendering. Second, if the lower-level points are generated from a rather large triangle, these points are used to fill up the detail area after the rendering of the higher-level splats. Thus, to increase rendering efficiency, we can ignore these lower-level points via our detail level control because they are not visible when the objects are far away from the viewpoint.

Unlike the other point rendering approaches, the triangle mesh used in our method does not require a uniformly sized mesh or mesh model generated from scanned models containing millions of small triangles. Our method is suitable for both arbitrarily and uniformly sized triangle mesh objects. Moreover, unlike Wand et al. [Wand01], who use a hybrid method that combine the point rendering method and the conventional triangular z-buffer method for scenes with both details and large triangles, our method is a uniform representation for not only small triangles but also large triangles in the scene objects. In our method, there is no need to resample the surface after the pre-processing stage like Stamminger and Drettakis [Stamminger01]. Therefore, geometric mesh object information is not necessary during rendering.

However, unlike the other dense sampling approaches, it is difficult for our method to reconstruct surface texture since less sample points are stored. One disadvantage of our

method is that shadow generation is awkward to implement especially for those splats in higher-level lists. However, rendering performance is improved due to fewer sample points.

Chapter 6

Conclusion

Researchers have been studying the generation of shadows in a synthetic image for over two decades. Shadows play an important role in photo-realistic rendering. Soft shadow algorithms are especially helpful in generating photo-realistic images but their execution is a time and memory-consuming task. As a result, soft shadow generation has been a research topic in the field of computer graphics for a long time.

We have reviewed soft shadow algorithms and categorized them into eight types according to the intermediate shadow information recalculation that results from changes in light sources, scene objects, and viewpoints. We also propose two soft shadow algorithms that render soft shadows in synthetic scenes. Both algorithms generate intermediate shadow information in the pre-processing stage and render soft shadows in the second stage.

The first algorithm, "visualizing soft shadows for deformable moving area light sources", partially updates intermediate shadow information whenever the light sources are moved and/or deformed. The second algorithm, "forward area light map projection", creates intermediate shadow information, which is not recalculated when we move the viewpoint, for static scene objects and light location. In conclusion, the intermediate shadow information plays a crucial role in shadow algorithms.

In addition, a point rendering algorithm (priority point lists for point-based object rendering) is also introduced. Although this algorithm is not a shadow algorithm, it provides information for viewers to understand our two shadow algorithms.

6.1 Importance of the Intermediate Shadow Information

Importantly, the success of a shadow algorithm depends on how the data structures that contain the intermediate shadow information, are defined, how the information is derived from the scene, how it retrieves information from the structure, and how it processes retrieved information to generate final shadow information.

Object-space algorithms generate this information through a complex computation that renders shadowed images of good quality. Image-space algorithms generate intermediate shadow information by quickly projecting the objects onto a discrete image. Hybrid approaches use image-space method to access depth images and then transform the resulting information to object space for intermediate shadow information for soft shadow generation. In general, image-space algorithms perform soft shadow generation faster than that of object-space algorithms. However, object-space algorithms produce images with better quality than those of image-space algorithms.

6.2 Contributions

Two soft shadow algorithms are proposed and adaptive sampling and point rendering techniques are also studied. Although these are well-known techniques and have been adopted in many fields, we provide an alternative way to apply them.

To visualize soft shadows for deformable moving area light sources, we create the shadow slab to store pre-created shadow maps and use the shadow maps to efficiently create/update soft shadows. An adaptively sampled shadow map approach, which is non-traditional, is proposed in this algorithm. Even though the adaptively sampled shadow map approach reduces memory consumption, it costs time and memory to generate the shadow slab. The drawback of this method is the limitation of the viewpoint's location in the visualization stage.

Forward area light map projection is the other soft shadow algorithm proposed in this dissertation. It provides a different mechanism to generate soft shadows. By combining the ideas of multi-layered shadow maps and point rendering, we generate a high quality soft shadow image in a few seconds from the layered area light map that consumes less memory than the shadow slab does.

Both the adaptive sampling technique in visualizing soft shadows for deformable moving area light sources (chapter 3) and the point rendering technique in forward area light map projection (chapter 4) are studied. These two techniques are used in the priority lists for point-based object rendering (chapter 5) that renders point-sampled objects and is

suitable for scenes containing both small and large triangles. Note that the adaptive sampling techniques introduced in chapter 3 and chapter 5 are not exactly the same. We store sample points for the corners of each valid node in chapter 3, but record the center points in chapter 5.

In general, soft shadow generation is a time-consuming and memory-consuming task. Although several researchers propose their timesaving approaches to create soft shadow, they all use large machines with special graphics features. Our methods are dedicated to working on a PC without using special graphics hardware features while still displaying visual results in a few seconds.

6.3 Future Work

To improve shadow algorithms, there are two subjects for developers to work on: the generation of intermediate shadow information and the rendering of shadowed images. In the future, our goal is to explore sampling strategies to capture surface points with speed and without point duplication. We suggest two directions for future researches: the layered texture mapping and the texture map slicing.

Texture mapping has been adapted in soft shadow algorithms. It is possible to use layered texture maps together with the layered area light map since we can use the hardware accessory feature for texture mapping. On the other hand, in order to put multiple texture maps into the layered shadow map and project them onto the screen

buffer quickly, texture map slicing may be a solution. Each texture map must be sliced in order to deposit them into layers of the map.

BIBLIOGRAPHY

[Agrawala00] Agrawala, M., Ramamoorthi, R, Heirich, A. and Moll, L. "Efficient Image-Based Methods for Rendering Soft Shadows". In Computer Graphics (Proc. SIGGRAPH 00), August 2000.

[Amanatides84] Amanatides J. "Ray tracing with cones". In Computer Graphics (Proc. SIGGRAPH 84), volume 18, pages 129-135, July 1984.

[Appel68] Appel, A. "Some techniques for shading machine renderings of solids". In AFIPS 1968 Spring Joint Computer Conference, pages 37-45, 1968.

[Baum89] Baum, D.R., Rushmeier, H.E. and Winget, J.M. "Improving radiosity solutions through the use of analytically determined form-factors". In Computer Graphics (Proc. SIGGRAPH 89), volume 23, pages 325-334, July 1989.

[Bergeron86] Bergeron, P. "A general version of crow's shadow volumes". In IEEE Computer Graphics and Applications, 6(9), pages 17-28, 1986.

[Brabec00] Brabec, S., Heidrich, W. and Seidel, H.-P. "OpenGL shadow maps". In Technical Report TR 2000-4-002, Max-Planck-Institute for Computer Science, March 2000.

[Brotman84] Brotman, L. S. and Badler, N. I. "Generating soft shadows with a depth buffer algorithm". In IEEE Computer Graphics and Applications, 4(10), pages 71-81, October 1984.

[Campbell90] Campbell, A.T. III and Fussell, D.S. "Adaptive mesh generation for global diffuse illumination". In Computer Graphics (Proc. SIGGRAPH 90), volume 24, pages 155-164, August 1990.

[Chen90] Chen, S.E., "Incremental radiosity: An extension of progressive radiosity to an interactive image synthesis system". In Computer Graphics (Proc. SIGGRAPH 90), volume 24, pages 35-144, August 1990.

[Chen93] Chen, S.E. and Williams, L. "View interpolation for image synthesis". In Computer Graphics (Proc. SIGGRAPH 93), pages 279-288, July 1993.

[Chen01] Chen, B. and Nguyen, M. "Pop: A hybrid point and polygon rendering system for large data". In IEEE Visualization 2001, 2001.

[Chin89] Chin, N. and Feiner, S. "Near real-time shadow generation using BSP trees". In Computer Graphics (Proc. SIGGRAPH 89), volume 23, pages 99-106, July 1989.

[Chin92] Chin, N. and Feiner, S. "Fast object-precision shadow generation for area light sources using BSP trees", In Computer Graphics (1992 Symposium on Interactive 3D Graphics), volume 26, pages 21-30, March 1992.

[Chrysanthou97] Chrysanthou, Y. and Slater, M. "Incremental updates to scenes illuminated by area light sources". In Rendering Techniques '97 (Proc. of the 8th Eurographics Rendering Workshop), pages 103-114, 1997.

[Cohen93] Cohen, M.F. and Wallace, J.R. "Radiosity and realistic image synthesis". Held in San Diego, CA. 1993.

[Cohen01] Cohen, J., Aliaga, D. and Zhang, W. "Hybrid simplification: Combining multi-resolution polygon and point rendering". In IEEE Visualization 2001, 2001.

[Cook84] Cook, R.L., Porter, T. and Carpenter, L. "Distributed ray tracing". In Computer Graphics (Proc. SIGGRAPH 84), volume 18, pages 137-145, July 1984.

[Cook87] Cook, R.L., Carpenter, L. and Catmull, E. "The Reyes image rendering architecture". In Computer Graphics (Proc. SIGGRAPH 87), pages 95-102, July 1987.

[Crow77] Crow, F.C. "Shadow algorithms for computer graphics". In Computer Graphics (Proc. SIGGRAPH 77), pages 242-248, July 1977.

[Crow84] Crow, F. "Summed-area tables for texture mapping". In Computer Graphics (Proc. SIGGRAPH 84), volume 18, pages 207-212, July, 1984.

[Diefenbach94] Diefenbach, P.J. and Badler, N. "Pipeline rendering: Interactive refractions, reflections and shadows". In Special Issue on Interactive Computer Graphics, 15(3), pages 173-180, 1994

[Drettakis94] Drettakis, G.S., "Sampling and reconstruction of illumination for image synthesis". PhD thesis, University of Toronto, 1994.

[Drettakis97] Drettakis, G. and Sillion, F.X. "Interactive update of global illumination using a line-space hierarchy". In Computer Graphics (Proc. SIGGRAPH 97), pages 57-64, August 1997.

[Durand97] Durand, F., Drettakis, G. and Puech, C. "The visibility skeleton: A powerful and efficient multi-purpose global visibility tool". In Computer Graphics (Proc. SIGGRAPH 97), August 1997.

[Fernandez02] Fernandez, S., Bala, K., Piccolotto, M.A. and Greenberg, D.P. "Interactive direct lighting in dynamic scenes". Program of Computer Graphics Technical Report: PCG-00-02, Cornell University

[Fernando01] Fernando, R., Fernandez, S., Bala, K. and Greenberg, D.P. "Adaptive shadow maps". In Computer Graphics (Proc. SIGGRAPH 01), pages 387-390, August 2001.

[Foley90] Foley, J., van Dam, A., Feiner, S. and Hughes, J. "Computer graphics: principles and practice" 2nd ed. ISBN 0-201-12110-7 Addison-Wesley 1990.

[Formella98] Formella, A. and Lukaszewski, A. "Fast penumbra calculation in ray tracing". In Proceedings of WSCG'98, the 6th International Conference in Central Europe on Computer Graphics Visualization'98, pages 238-245, February 1998.

[Forsyth94] Forsyth, D., Yang, C. and Teo, K. "Efficient radiosity in dynamic environments". In Rendering Techniques '94 (Proc. of the Fifth Eurographics Workshop on Rendering), pages 313-324, June 1994.

[Guo98] Guo, B. "Progressive radiance evaluation using directional coherence maps". In Computer Graphics (Proc. SIGGRAPH 98), pages 255-266, July 1998.

[Grossman98] Grossman, J.P. and Dally, W.J. "Point sample rendering". In Rendering Techniques'98, pages 181-192, 1998.

[Hagen86] Hagen, M. "Varieties of Realism". In Cambridge University Press, Cambridge, England, 1986.

[Haines94] Haines, A.E. and Wallace, J.R. "Shaft Culling for Efficient Ray Cast Radiosity". In Photorealistic Rendering in Computer Graphics (Proc. of the 2nd Eurographics Workshop on Rendering), pages 122-138, 1994.

[Hart99] Hart, D., Dutre, P. and Greenberg, D.P. "Direct illumination with lazy visibility evaluation". In Computer Graphics (Proc. SIGGRAPH 99), pages 147-154, July 1999.

[He91] He, X., Torrance, K., Sillion, F., and Greenberg, D. "A comprehensive physical model for light reflection", In Computer Graphics (Proc. SIGGRAPH 91), pages 175-186, July 1991.

[Hearn97] Hearn, D. and Baker, P. "Computer graphics" 2nd ed. ISBN 0-13-530924-7 Prentice Hall, Inc. 1997.

[Heckbert90] Heckbert, P. "Adaptive Radiosity Textures for Bidirectional Ray Tracing". In Computer Graphics (Proc. SIGGRAPH 90), pages 145-154, July 1990.

[Heckbert92] Heckbert, P. "Discontinuity meshing for radiosity". In Rendering Techniques '92 (Proc. of the 3rd Eurographics Rendering Workshop), pages 197-212, June 1992.

[Heckbert97] Heckbert, P. and Herf, M. "Simulating soft shadows with graphics hardware". Technical Report CMU-CS-97-104, Carnegie Mellon University, January 1997.

[Heflin93] Heflin, G. and Elber, G. "Shadow volume generation from free form surfaces". In Communicating with virtual worlds, Proceedings of CGI'93, Springer-Verlag, pages 115-126, June 1993.

[Heidrich00] Heidrich, W., Brabec, S. and Seidel, H. "Soft shadow maps for linear light". In Rendering Techniques '00, (Proc. of the 11th Eurographics Rendering Workshop), August 2000.

[Herf96] Herf, M. and Heckbert, P.S. "Fast Soft Shadows". Technical Sketches (Visual Proc. SIGGRAPH 96), page 145, July 1996.

[Kalaiah01] Kalaiah, A. and Varshney, A. "Differential point rendering". In Rendering Techniques '01, (Proc. of the 12th Eurographics Rendering Workshop), Springer-Verlag, pages 139-150, August 2001.

[Keating99] Keating, B. and Max, N. "Shadow penumbras for complex objects by depth-dependent filtering of multi-layer depth images". In Rendering Techniques '99 (Proc. of the 10th Eurographics Rendering Workshop), pages 197-212, June 1999.

[Keller97] Keller, A. "Instant Radiosity". In Computer Graphics (Proc. SIGGRAPH 97), pages 49-55, 1997.

[Levoy85] Levoy, M. and Whitted, T. "The use of points as display primitives". Technical Report TR85-022, Univ. of North Carolina at Chapel Hill, 1985.

[Levoy96] Levoy, M. and Hanrahan, P. "Light field rendering". In Computer Graphics (Proc. SIGGRAPH 96), pages 31-42, July 1996.

[Lischinski92] Lischinski, D., Tampieri, F. and Greenberg, D.P. "Discontinuity meshing for accurate radiosity". In IEEE Computer Graphics and Applications, 12(6), pages 25-39, November 1992.

[Lischinski98] Lischinski, D. and Rappoport, A. "Image-based rendering for non-diffuse synthetic scenes". In Rendering Techniques '98 (Proc. of the 9th Eurographics Rendering Workshop), pages 301-314, June 1998.

[Losco97] Losco, C. and Drettakis, G. "Interactive high-quality soft shadows in scenes with moving objects". In Eurographics, 16(3), 1997.

[Max91] Max, N. "Unified sun and sky illumination for shadows under trees". CVGIP: Graphical Models and Image Processing. 53(3), pages 223-230, May 1991.

[Max99] Max, N., Deussen, O. and Keating, B. "Hierarchical image-based rendering using texture mapping hardware". In Rendering Techniques '99 (Proc. of the 10th Eurographics Rendering Workshop), pages 57-62, June 1999.

[McCool98] McCool, M. "Shadow volume reconstruction". Technical Report CS-98-06, University of Waterloo, 1998.

[Myszkowski94] Myszkowski, K. and Kunii, T.L. "Texture Mapping as an Alternative for Meshing During Walkthrough Animation". In Rendering Techniques '94 (Proc. of the 5th Eurographics Rendering Workshop), pages 375-388, 1994.

[Ouellette99] Ouellette, M. and Fiume, E. "Approximating the location of integrand discontinuities for penumbral illumination computation with area light sources". In Rendering Techniques '99 (Proc. of the 10th Eurographics Rendering Workshop), pages 213-224, June 1999.

[Parker98] Parker, S., Shirley, P. and Smits, B. "Single sample soft shadows". Technical Report UUCS-98-019, Computer Science Department, University of Utah, 1998.

[Pfister00] Pfister, H., Zwicker, M., J. van Baar, and Gross, M. "Surfels: Surface elements as rendering primitives". In Computer Graphics (Proc. SIGGRAPH 00), pages 335-342, 2000.

[Phong75] Phong, B.T. "Illumination for computer-generated images". CACM, volume 18(6), pages 311-317, 1975.

[Reeves87] Reeves, W. T., Salesin, D.H. and Cook, R.L. "Rendering antialiased shadows with depth maps". In Computer Graphics (Proc. SIGGRAPH 87), pages 283-291, July 1987.

[Rusinkiewicz01] Rusinkiewicz, S. and Levoy, M. "QSplat: A multiresolution point rendering system for large meshes". In Computer Graphics (Proc. SIGGRAPH 01), pages 343-352, August 2001.

[Segal92] Segal, M., Korobkin, C., R. van Widenfelt, Foran, J. and Haeberli, P. "Fast shadow and lighting effects using texture mapping". In Computer Graphics (Proc. SIGGRAPH 92), volume 26(2), pages 249-252, July 1992.

[Shade98] Shade, J. W., Gortler, S. J., He, L., and Szeliski, R. "Layered depth images". In Computer Graphics (Proc. SIGGRAPH 98), pages 231-242, July 1998..

[Shaw97] Shaw, E. "Hierarchical radiosity for dynamic environments". In Computer Graphics Forum, 16(2), pages 107-118, 1997.

[Shirley96] Shirley, P., Wang, C.Y. and Zimmerman, K. "Monte Carlo Techniques for Direct Lighting Calculations". ACM Transactions on Graphics, 15(1), pages 1-36, January 1996.

[Siegel81] Siegel, R. and Howell, J.R. "Thermal radiation heat transfer". Hemisphere Publishing Corporation, Washington, 1981.

[Soler98] Soler, C. and Sillion, F.X. "Fast calculation of soft shadow textures using convolution". In Computer Graphics (Proc. SIGGRAPH 98), pages 321-332, July 1998.

[Sparrow78] Sparrow, E.M. and Cess, R.D. "Radiation heat transfer". Hemisphere

Publishing Corporation, Washington, 1978.

[Stamminger01] Stamminger, M. and Drettakis, G. "Interactive Sampling and Rendering for Complex and Procedural Geometry". In Rendering Techniques '01 (Proc. of the 12th Eurographics Rendering Workshop), 2001.

[Stewart93] Stewart, A.J. and Ghali, S. "An output sensitive algorithm for the computation of shadow boundaries", Fifth Canadian Conference on Computational Geometry, pages 291-296, August 1993.

[Stewart94] Stewart, A.J. and Ghali, S. "Fast computation of shadow boundaries using spatial coherence and backprojections". In Computer Graphics (Proc. SIGGRAPH 94), pages 231-138, July 1994.

[Udeshi99] Udeshi, T. and Hansen, C. "Towards interactive, photorealistic rendering of indoor scenes: A hybrid approach". In Rendering Techniques '99 (Proc. of the 10th Eurographics Rendering Workshop), pages 63-76, June 1999.

[Wand01] Wand, M., Fischer, M., Peter, I., Meyer auf der heide, F. and StraBer, W. "The randomized z-buffer algorithm: interactive rendering of highly complex scenes". In Computer Graphics (Proc. SIGGRAPH 01), pages 361-370, August 2001.

[Whitted80] Whitted, T. "An Improved Illumination Model for Shaded Display". Communications of the ACM, 23:343-349, 1980.

[Wiebelt66] Wiebelt, J.A. "Engineering radiation heat transfer". Holt, Rinehart and Winston, Inc., New York, 1966.

[Williams78] Williams, L. "Casting curved shadows on curved surfaces". In Computer Graphics (Proc. SIGGRAPH 78), pages 270-274, August 1978.

[Worrall95] Worrall, A., Willis, C. and Paddon, D. "Dynamic discontinuities for radiosity". In Edugraphics + Compugraphics Proceedings, pages 367-375, Dec. 1995.

[Worrall98] Worrall, A., Hedley, D. and Paddon, D. "Interactive animation of soft shadows". In Proceeding of Computer Animation 1998, pages 88-94, June 1998

[Woo90] Woo, A., Poulin, P. and Fournier, A. "A survey of shadow algorithms". IEEE Computer Graphics and Applications, 10(6), pages 13-32, November 1990.

[Woo92] Woo, A., Graphics Gems III, chapter The Shadow Depth Map Revisited, pages 338-342, Academic Press, 1992.

[Zatz93] Zatz, Harold R. "Galerkin Radiosity: A Higher-order Solution Method for Global Illumination". In Computer Graphics (Proc. SIGGRAPH 93), pages 327-334, August 1993.

[Zhang98] Zhang, H. "Forward Shadow Mapping". In Rendering Techniques '98 (Proc. of the 9th Eurographics Rendering Workshop), June 1998.

[Zwicker01] Zwicker, M., Pfister, D., J. van Baar, and Gross, M. "Surface splatting". In Computer Graphics (Proc. SIGGRAPH 01), pages 371-378, August 2001.