THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par *l'Université Toulouse III - Paul Sabatier*

Discipline ou spécialité : *Informatique*

Présentée et soutenue par **Vincent Forest**

Le 16 décembre 2008

Robust object-based algorithms for direct shadow simulation

JURY

Rapporteurs :	Nicolas Holzschuch - CR HDR - INRIA Rhône Alpes
	Michael Wimmer - Ass. Prof. - TU Wien
Examinateurs :	George Drettakis - DR - INRIA Sophia Antipolis
	Christophe Schlick - Pr - Université de Bordeaux
	Loïc Barthe - MCF - Université de Toulouse
	Mathias Paulin - Pr - Université de Toulouse

Ecole doctorale : Mathématiques Informatique Télécommunications de Toulouse
Unité de recherche : Institut de Recherche en Informatique de Toulouse - UMR 5505
Directeur(s) de Thèse : Mathias Paulin

Acknowledgments

I would like to thank all the people who have helped me in one way or another during my PhD. First, I would like to express my gratitude to my advisor, Mathias Paulin. He pushed me towards new challenges and brought me unnumbered scientific skills as well as support and encouragement. I sincerely thank him for his ongoing confidence in me.

Second, I would like to thank Loïc Barthe who has been an invaluable help during the three years of my PhD. He was always available for discussions and gave me precious advices. I thank him especially for his patience and his help for the completion of the papers.

Next, I want to thank the other members of my reading committee, Michael Wimmer, Nicolas Holzschuch, George Drettakis and Christophe Schlick for taking interest in my work. Their insights and perspective comments were also much appreciated.

I would like to thank the members of the VORTEX team, for the work and entertainment we had. I also want to thank all my colleagues and friends from the University of Toulouse for the amazingly good time I had with them. I thank especially, the "Voxariotes" Mathieu Muratet, Olivier Gourmel, Robin Bouriane and Anthony Pajot that have provided help, insight into their research and ideas, and tough discussions. Special thanks go to my old friends Guillaume Cabanac and Sylvain Rougemaille for their comments and help during my research and teaching activities and for all the pleasant time I had discussing with them.

I thank also Gaël Guennebaud, Tamy boubekeur and Benjamin Segovia for all passionating and stimulating discussions we had.

I would like to thank my parents Yves and Marie-France Forest, for their love and support over the years. Special thanks go to my uncle and godfather Michel André who taught me the passion for computer science. Finally, my deepest gratitude goes to my fiancée, Binh. She supported me at all steps of this thesis, and without her this thesis would never have been possible. I dedicated this thesis to her.

To Binh

Contents

1

Introduction

The goal of this dissertation is to develop robust algorithms solving the problem of interactive direct shadow generation. The direct shadows represent the region of the scene that is not *directly* enlighten by an emitter. By robustness we mean the unconditional accuracy of the solution according to the underlying scene representation. Currently, the computation of robust direct shadows in interactive applications is particularly challenging. In this dissertation we present solutions to reach this goal, by developing new rendering algorithms. We also point out the limitations of current interactive direct shadow solutions.

Most offline and real time renderers generate approximative direct shadows rather than accurate ones. Despite their efficiency, these approximations have to deal with strong limitations and robustness issues. On the one hand, many efforts are made to increase the general quality and the performances of these biased solutions. On the other hand, surprisingly few attention is focused on interactive and robust direct shadow algorithms. In fact, the accurate direct shadow computation is often considered as an highly time consuming task where the interest is reserved to offline applications that require high image fidelity. As a consequence, very few algorithms were designed to generate robust shadows in interactive renderers while there is a considerable interest in solving this problem.

To democratize robust direct shadows, it is important to find algorithmic tools that are well suited for common renderers. Direct shadow algorithms have to avoid hand fixed parameters. In addition, they must be independent of the world organization in order to efficiently handle arbitrary animated scenes. Finally, they have to evaluate physically plausible soft shadows whatever the geometric complexity, materials and illuminations of the scene.

Our researches target the robust generation of physically plausible direct shadows for rasterizers. Despite impressive improvements in real time ray tracing, the rasterization

algorithm is widely used in real time rendering. The rasterizers are very efficient for local computations but do not provide an elegant and general purpose way to access the world organization. Our goal is to keep the efficiency of the rasterization for local calculations and to find robust physically plausible direct shadow algorithms despite the lack of a global access to the scene.

In the following section we start with a brief overview of the direct lighting problem to explicitly define what are direct shadows. We also discuss the common assumptions of the direct shadow problematic. Then, we give a high-level view of the main direct shadow techniques. Finally, we summarize the original contributions of this dissertation and outline the thesis organization.

1.1 The direct lighting problem

The direct lighting corresponds to the amount of light that *directly* reaches the scene surface. Considering a scene description, including lights, viewpoint of the rendering image, geometries and their associated scattering properties, the main goal of the direct lighting computation is to define how the visible receivers are directly enlighten by a light. In fact the direct shadow problematic is the complementary of the direct lighting problem: we have to evaluate how the light is directly occluded for each visible surface.

1.1.1 Why efficient robust direct shadows are important

Today, the interactive approximations of direct shadows produce visual pleasant results improving the overall realism. However, they do not give sufficient visual informations to define, for instance, the distance between the fallen character and the ground or the mountain peak height. Robust realistic direct shadows do not exhibit such drawbacks and naturally give clues to correctly perceive geometric shape, light positions or object relationship.

Another issue of most interactive approximations of direct shadows is their inefficiency to uniformly treat geometries with different scale. On the one hand, the direct shadows cast by the trunk of an oak tree are often convincing. On the other hand, the direct shadows of its leaves exhibit strong artifacts. Thanks to their robustness property, robust direct shadow algorithms do not exhibit this drawback.

Despite incontestable advantages, robust realistic direct shadows are still associated to prohibitive computation time. Ideally, robust interactive direct shadow algorithm would be an attractive tool for direct lighting design of fully animated scenes. The robustness of these algorithms would avoid scene specific bias parameter. In addition, thanks to their

efficiency, they would propose a very intuitive feeling of the actions of the user on the direct lighting parameters.

Starting from these observations, efficient robust direct shadow algorithms would be very attractive in many ways. The combination of simplicity, realism and interactivity would give a very powerful tool solving the direct lighting problem. As a result, despite the high performances of current interactive approximations, we think that the benefits of efficient robust direct shadow algorithms will outweighs their computational overhead.

1.1.2 Assumptions about the direct shadow models

Our works target the rasterization rendering algorithm and consequently, we are submitted to its assumptions. Firstly, we assume a geometric optics model where the light travels along a straight line between surfaces. The geometric optics ignores not very significant effects for common environments (*e.g.* diffraction) but is adequate to accurately simulate a high range of visual effects (including direct shadows).

Secondly, all the incoming geometric primitives have to naturally fulfill the rasterizer requirements (*e.g.* triangles). For parametric or subdivision surfaces we simply consider their tessellated discretization. However, we ignore primitives as voxels or points since they are not particularly better suited for the efficient rendering of dynamic environments.

Finally, our researches target the lack of robust shadows for realistic direct lighting with a rasterizer. We do not investigate other difficult rasterization tasks as the refraction, the reflections or the global illumination. Thus, any existing rendering techniques targeting such global effects are orthogonal to our work.

1.2 Computing shadows

One can define shadows as a visual effect in the general context of the light simulation. Indeed, the robust light transport algorithms do not explicitly address the shadow problematic while they produce realistic images with shadows. In fact, the shadows are the visual result of the visibility queries between the emitters and the receivers; according to an emitter, receivers are in shadow if they do not see the emitter. Unlike ray tracers, rasterizers do not explicitly test the reciprocal receiver/emitter visibility. As a result, they require additional treatments in order to generate shadows.

In ground truth, the shadows are due to both direct emitters (direct shadows) as well as indirect ones (indirect shadows). However, the shadow term is commonly used to describe only direct shadows. Thus, in the rest of the dissertation we use this terminology simplification in order to propose a synthetic discussion.

1.2.1 Hard and soft shadows

In computer graphics, it is convenient to define light sources by infinitesimal thin emitters. With such point lights, the receivers are either illuminated or not. Despite the simplicity of the solution, the *hard* transition of this binary repartition cannot pretend to realism.

In real environments, light sources are extended and the receivers can lie in the light, in the shadow or in the penumbra (*i.e.* it is partially illuminated). The penumbra defines the realistic soft transition between light and shadow and it is useful to define the size of the lights as well as the distance between objects. Indeed, the more the distance from occluder to receiver increases, the more the penumbra region is large.

In order to generate accurate soft shadows, one have to evaluate the amount of light that directly reaches each visible receiver. The obvious solution consists in numerically solve the direct lighting problem by discretizing the surfacic light in a set of infinitesimal thin emitters. Hard shadow tests finally define which light sample are visible for the receivers. Unfortunately, in rasterization, accumulating several hard shadow computations is both inadequate and very inefficient. In order to achieve interactive rendering of realistic rasterized soft shadows, specific algorithms must be designed.

1.2.2 Image-based and object-based shadow algorithms

Paradoxically, the strength and the weakness of the rasterization rendering algorithm relies on its Z-buffer visibility algorithm. This algorithm works as follows. First, the rasterizable primitives are projected onto the image plane and discretized according to the view sample distribution. Then the Z-buffer stores for each view sample which projecting surface points are the closest to the viewpoint. The out-of-order style of the Z-buffer algorithm is particularly well suited for animated scenes. Nevertheless, it does not provide a general-purpose visibility algorithm, *i.e.* it cannot define the reciprocal visibility between arbitrary surface points.

On the one hand, the Z-buffer algorithm can be generalized to the shadow generation. Indeed, by rasterizing the scene from the light viewpoint, one can retrieve which surfaces are "visible" by the light. The points store into the light Z-buffer are consequently directly illuminated while others are in the shadow. This image based approach is a very intuitive and efficient tool for generating shadows with a rasterizer. However, due to the light Z-buffer discretization, the resulting shadows exhibit strong accuracy issues.

On the other hand, one can build for each occluder a volume that includes the region that it occludes. Then, the Z-buffer visibility algorithm defines which receivers lie into this volume, *i.e.* which are in the shadow of the associated occluder. Unlike image based shadows, the accuracy of these object-based shadows is not biased by the discretization

of the visibility information. However, the robust extrusion of the shadow bounding geometry imposes geometric constraints and limits the overall rendering performances.

1.3 Summary of contributions

Our contributions fall in three areas: a new penumbra blending heuristic that addresses the shadow overestimation artifact of the penumbra wedge algorithm [AAM03], a new robust soft shadow algorithm, and a new algorithm that addresses the lack of robust soft shadows cast by triangles having a spatially varying transmittance property.

1.3.1 Penumbra wedge blending

Today, the efficient rasterization of physically plausible soft shadows is based on restrictive assumptions. The accuracy of the proposed solutions are thus limited to specific cases.

We investigate the penumbra wedge algorithm [AAM03]. Despite the apparent robustness of this object-based framework, it exhibits a lack of accuracy when penumbrae are overlapped. We present a penumbra blending heuristic that drastically reduces this penumbra overestimation artifact. We then discuss the limitations of the proposed solution from both theoretical and implementation point of views.

1.3.2 Depth complexity sampling

We present a new robust object-based soft shadow algorithm that merges the efficiency of the penumbra wedge algorithm [AAM03] with the accuracy of the offline soft shadow volume approach [LAA+05]. To compute shadows, we distribute a set of samples onto light sources. Then, we define which light samples are visible for each visible receiver by evaluating locally the number of occluders (*i.e.* the depth complexity) lying between the receivers and the light samples.

This new robust soft shadow algorithm is particularly well suited for rasterizers and is comparable in quality with ray traced shadows even in very difficult direct lighting situations. It is interactive, physically based and its low memory cost is independent of the geometric complexity of the scene. Furthermore, it performs especially well with fully animated scenes and non uniform light sources, and does not exhibit common soft shadow issues (*e.g.* light leaking, shadow popping, magnification aliasing, *etc.*).

The key advantage of our algorithm is that it uses the penumbra wedge framework to sample the depth complexity. This has several consequences. First, the penumbra wedge algorithm brings the global visibility problem to a local depth complexity evaluation. The computations are thus independent for each view sample and efficiently evaluated

by a high end rasterizer. Second, due to the Monte Carlo sample distribution, averaging several runs would give a depth complexity evaluation that would be very close to the exact solution. Third, the visibility information is naturally derived from the depth complexity. By evaluating the depth complexity between receivers and light samples we can thus numerically solve the direct lighting.

1.3.3 Soft textured shadow volumes

In order to improve the performances, it is convenient to define very detailed and thin objects with few triangles and textures encoding their binary transmittance (*e.g.* fence). Despite their robustness in computing shadows cast by fully opaque triangles, object-based shadow frameworks are not able to deal with this object representation.

We address this limitation. We present a general algorithm computing robust and accurate soft shadows for triangles with a spatially varying transmittance denoted as S-triangles. First, we extrude a primitive from each S-triangle that includes its shadow influence. For the receivers lying into this volume, we then evaluate its direct lighting by numerically computing the light/receiver visibility.

This soft shadow algorithm computes physically based soft shadows that are cast by S-triangles. It is interactive, robust and it is efficiently evaluated by rasterizers. In addition, we show how this technique can be efficiently included into object-based soft shadow algorithms. This results in unified object-based frameworks computing robust direct shadows for both standard and perforated triangles in fully animated scenes.

1.4 Thesis organization

The following chapter presents shadow algorithms for rasterizers. We first introduce the main hard shadow approaches and describe how their quality and performances can be improved. We then investigate both visually and physically plausible soft shadows. Visually plausible algorithms favor visual pleasant results rather than accurate solutions. Due to their strong assumptions, we outline that they exhibit issues that drastically limit their realism. Physically plausible approaches compute more convincing soft shadows. We give an overview of some algorithms and we point out that despite their physically plausible background, they must deal with restrictive assumptions as well as accuracy issues. Starting from these observations, we finally detail our technical choices for our investigation of the robust soft shadow problematic.

In chapter *three* we detail the penumbra wedge algorithm [AMA02]. We then describe how we improve its realism using a new penumbra blending heuristic [FBP06]. From a

more practical point of view, we develop an algorithm that implements the penumbra wedges and our penumbra blending heuristic. After a detailed performance analysis we discuss the intrinsic limitations of this algorithm from both theoretical and implementation point of view.

The following *two* chapters describe a new robust soft shadow framework for rasterizers. Chapter $four$ presents how the depth complexity function can be used in order to compute soft shadows. We detail how we combine our new efficient local depth complexity evaluation with the rasterization rendering algorithm in order to compute *accurate shadows by depth complexity sampling* [FBP08]. In chapter $five$ we investigate one of the main limitation of object-based shadows. We design a new soft shadow algorithm that computes soft shadows cast by triangles with a spatially varying transmittance. We then describe how this approach can be efficiently integrated in common object-based framework in order to propose a new unified and robust object-based algorithm for the accurate simulation of direct shadows [FBP09].

2

Rasterizing shadows

Shadows computation is a widely studied topic. Many algorithms have been designed for both offline and real time applications in order to reduce the computational cost of the shadow generation. In this chapter we first formalize the problematic of direct shadows. We then discuss about some shadow algorithms designed for the rasterization rendering algorithm. These approaches fall into two categories: hard shadow algorithms and soft shadow algorithms. We briefly summarize their principle and discuss their results in order to exhibit their advantages and drawbacks. From these observations, we finally detail our technical choices for our investigation toward the robust simulation of direct shadows.

2.1 The shadow problematic

In this section we briefly formalize the concept of direct shadows. We then outline what may be evaluated when targeting robust solutions.

2.1.1 The light transport equation

By assuming a geometric optics, the light transport equation applied to each wavelength is given by [Kaj86]:

$$L\left(x' \rightarrow x''\right) = L_e\left(x' \rightarrow x''\right) + \int_{\mathcal{M}} L\left(x \rightarrow x'\right) f_s\left(x \rightarrow x' \rightarrow x''\right) G\left(x \leftrightarrow x'\right) dA\left(x\right) \tag{2.1}$$

$\mathcal{M}$ is the union of all scene surfaces, A is the area measure on $\mathcal{M}$, $L_e\left(x' \rightarrow x''\right)$ is the emitted radiance from x' to x'', L is the radiance function and f_s is the Bidirectional Scattering Distribution Function (BSDF). The function G is the geometric term given by:

$$G\left(x \leftrightarrow x'\right) = V\left(x \leftrightarrow x'\right) \frac{cos\theta_o cos\theta_i}{||x - x'||^2} \tag{2.2}$$

where θ_o and θ_i are the angles between the segment $x \leftrightarrow x'$ and the surface normals at x and x', respectively. Finally, $V(x \leftrightarrow x')$ represents the binary visibility between x and x'.

2.1.2 The direct illumination formulation

The function $L(x' \rightarrow x'')$ defines the radiance leaving x' in the direction of x''. The incoming radiance of x' includes the radiance directly emitted from the light sources to x' as well as the radiance scattered from scene surfaces in the direction of x'. However, the direct shadow simulation only depends on direct illumination, *i.e.* the radiance that directly reaches the surface from the light.

The incoming direct and indirect radiance reaching x' can be separated from the transport equation as follows:

$$L\left(x' \rightarrow x''\right) = L_e\left(x' \rightarrow x''\right) + L'\left(x' \rightarrow x''\right) \tag{2.3}$$

where L' is given by:

$$\begin{aligned} L'\left(x' \rightarrow x''\right) &= \int_{\mathcal{M}} L_e\left(x \rightarrow x'\right) f_s\left(x \rightarrow x' \rightarrow x''\right) G\left(x \leftrightarrow x'\right) dA\left(x\right) \\ &+ \int_{\mathcal{M}} L'\left(x \rightarrow x'\right) f_s\left(x \rightarrow x' \rightarrow x''\right) G\left(x \leftrightarrow x'\right) dA\left(x\right) \end{aligned} \tag{2.4}$$

The first integral of the equation 2.4 represents the radiance directly emitted by the union of the scene surfaces $\mathcal{M}$ in the direction of x'. The function $L_e(x \rightarrow x')$ is equal to zero everywhere excepted for emissive surface, *i.e.* the light sources. Thus, this integral simply defines the radiance directly emitted from x to x'. The second integral of the equation represents the indirect illumination, *i.e.* the radiance scattered from the scene surfaces $\mathcal{M}$ to x'.

Our research targets the robust shadow generation in order to simulate the direct lighting. Consequently, we assume that the indirect illumination is evaluated with respect to any algorithms. The equation 2.4 becomes:

$$\begin{aligned} L'\left(x' \rightarrow x''\right) &= \int_{\mathcal{M}'} L_e\left(x \rightarrow x'\right) f_s\left(x \rightarrow x' \rightarrow x''\right) G\left(x \leftrightarrow x'\right) dA\left(x\right) \\ &+ I\left(x' \rightarrow x''\right) \end{aligned} \tag{2.5}$$

where I is the indirect incoming radiance. This results in the following light transport

equation:

$$\begin{aligned} L\left(x' \rightarrow x''\right) &= L_e\left(x' \rightarrow x''\right) + I\left(x' \rightarrow x''\right) \\ &+ \int_{\mathcal{M}} L_e\left(x \rightarrow x'\right) f_s\left(x \rightarrow x' \rightarrow x''\right) V\left(x \leftrightarrow x'\right) \underbrace{\frac{cos\theta_o cos\theta_i}{||x - x'||^2}}_{G'(x \leftrightarrow x')} dA\left(x\right) \end{aligned} \tag{2.6}$$

2.1.3 The problematic of rasterizing shadows

In order to simulate realistic direct illumination, we have to evaluate the equation 2.6. However, the visibility query $V\left(x \leftrightarrow x'\right)$ is globally dependent of the environment. The rasterization rendering algorithm does not provide any global information on the scene organization. As a result, the equation 2.6 cannot be directly evaluated by a rasterizer. In fact, the fixed direct lighting evaluation of the common rasterization Application Programming Interfaces (API) [SA06, Mic08] does not perform any visibility query. Thus, each light illuminates all the scene; in other words, there is no shadow.

In order to compute realistic direct lighting, one have to develop specific shadow algorithms, *i.e.* algorithms that solve the visibility queries. In the following sections we investigate the existing solutions that rasterize shadows.

2.2 Hard shadows

Hard shadows are a simplification of the shadow problematic. They are based on the assumption that light sources are infinitesimal small: receivers are either in the shadow or not. The resulting radiance evaluation is thus given by:

$$\begin{aligned} L\left(x' \rightarrow x''\right) &= L_e\left(x' \rightarrow x''\right) + I\left(x' \rightarrow x''\right) \\ &+ \sum_{x \in \mathcal{L}} L_e\left(x \rightarrow x'\right) f_s\left(x \rightarrow x' \rightarrow x''\right) V\left(x \leftrightarrow x'\right) \frac{cos\theta_i}{||x - x'||^2} \end{aligned} \tag{2.7}$$

with $\mathcal{L}$ the set of positions of the unsurfacic light sources in the scene. This section presents the two main algorithms that allow the interactive rendering of hard shadows. We also describe some methods develop to improve the quality as well as the efficiency of these algorithms.

2.2.1 The shadow volumes

The shadow volume algorithm [Cro77] generates object-based hard shadows. For each light source, the silhouette edges of a 2D-manifold closed mesh are extruded through the direction from the light to the edge vertexes. (A silhouette edge is defined as an edge shared by *two* faces which the light lies on the positive side of one face and the negative side of the other). The resulting extruded geometry includes the scene region directly occluded by the mesh, *i.e.* the scene volume lying in its shadow. The face orientation with respect to the camera is then used to list the number of enters into and exits out the shadow volumes. Surface samples with less outputs than inputs are included into a shadow volume and consequently are in the shadow.

Despite specific software [BB84] or hardware [FGH$^+$85] implementations and a generalization to common meshes (open models, non-planar polygons) [Ber86] current shadow volume rendering is based on the frame buffer reformulation [FF88] of the initial algorithm. Using this formulation, the shadow volume test is performed as follows. First, the Z-buffer is initialized with the scene depth as seen from the viewpoint. Then a stencil defines which view samples are in the shadow or not [Hei91]. This stencil buffer is first initialized to *zero* (*i.e.* no shadow). The update of the stencil values is then performed by the rasterization of the shadow volumes with respect to a specific stencil update strategy (Z-pass or Z-fail). View samples are in the shadow if their resulting stencil value are not equal to *zero*.

Z-pass stencil update

To perform the stencil update, the shadow volume geometry is decomposed into *two* batches according to the primitive orientation from the camera point of view. On the one hand, the stencil value of visible surfaces is incremented when they are occluded by the rasterization of the front facing faces of the shadow volumes. On the other hand, the stencil value is decremented for the visible surfaces behind the back facing shadow volume polygons. In fact, the stencil buffer is updated for the shadow volume geometry that *passes* the Z-buffer visibility test (figure 2.1).

This initial Z-pass strategy has to deal with a strong robustness issue. Indeed, the stencil update rules must be inverted for the view samples lying in a shadow volume, *i.e.* when the image plane clips the shadow volume. This case is particularly difficult to handle since it is not obvious to define if a view sample is included into a shadow volume or not. A solution consists in capping the shadow volumes by the image plane of the camera [BJ, McC00, HHLH05]. However, despite its apparent robustness, this solution is in practice quite fragile (*e.g.* precision issues, singularities, *etc.*).

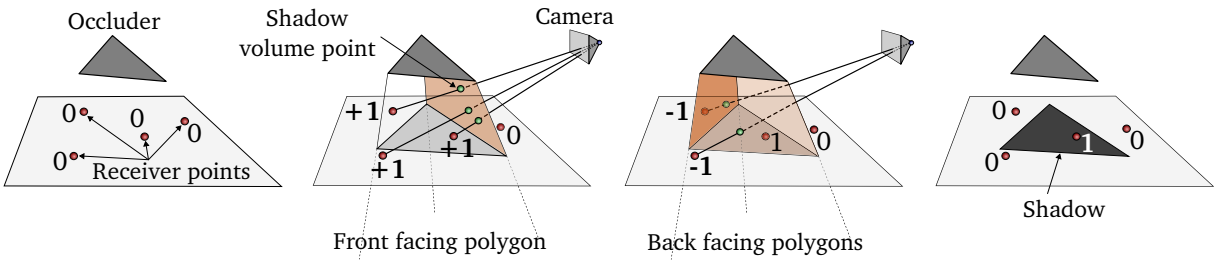


Figure 2.1: Illustration of the Z-pass stencil update strategy.

Z-fail stencil update

An alternative to the Z-pass stencil test consists in inverting the stencil update strategy [BS99, Car00] (figure 2.2): the stencil values are incremented or decremented for the surfaces that are *not occluded* by respectively the back or front faces of the shadow volumes. This shadow test is based on the *failure* to render the back faces of the shadow volume with respect to the viewpoint. Consequently, shadow volumes must be closed; *i.e.* they have to be capped at both front and back side. The front cap geometry is simply defined by the polygons of the mesh that front faces the light. For the back cap, triangles of the mesh which the light lies on their negative side are projected in the same way of the extruded silhouette vertexes.

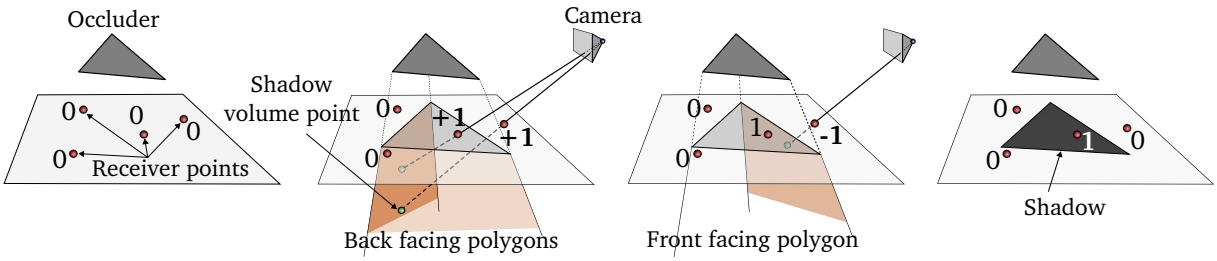


Figure 2.2: Illustration of the Z-fail stencil update strategy.

Unlike the Z-pass approach, the Z-fail strategy do not have to deal with the drawback of the image plane clipping since it lists the shadow volume enters/exits from infinity rather than from the viewpoint. Nevertheless, it must pay attention to the clipping of the shadow volume by the far plane of the frustum. In practice, this clipping plane is either disabled or set to infinity during the Z-fail shadow volume rasterization [EK02]. This results in a robust stencil update strategy providing pixel accurate hard shadows.

Z-pass VS Z-fail algorithm

Despite its robustness, the Z-fail shadow test is more time consuming than the Z-pass algorithm. The additional capping geometry requires additional geometric processing

and its rendering increases the overall fill rate cost. In order to improve performances, the *two* strategies can be combined [Len02, EK02]. Indeed, the Z-fail strategy is only required for the rendering of the shadow volumes that are clipped by the image plane. In all other situations, using the Z-pass algorithm does not compromise the robustness of the solution.

The selection of the stencil update strategy can be performed per pixel rather than per shadow volume [Lai05]. In this situation, the selection is made by comparing a low resolution Z-buffer with a per shadow volume split plane. This per pixel selection reduces the stencil buffer updates but does not address the Z-pass robustness issue nor the fill rate cost. Indeed, hardware modifications are required in order to cull multiple pixels before they would be processed. In addition, the selection of the Z-pass strategy still requires the perfectible capping of the shadow volumes by the image plane [HHLH05].

Semi infinite VS infinite projection

The silhouette edge extrusion and the back capping geometry construction are performed using either a semi infinite or an infinite projection with respect to the light position. In the first case, the projection does not strictly guarantee that the shadows are correctly computed since the shadow volume includes only a part of the occluded region. In practice, extruding the shadow volumes up to the bounding limit of the scene is sufficient to obtain accurate shadow tests.

On the other hand, the infinite projection is a general, easy and elegant solution constructing robust shadow volumes. This infinite transformation is very simply performed by setting to *zero* the homogeneous coordinate of the projection direction. The resulting shadow volume includes the whole occluded region and thus it has not to pay attention to the scene specific bounding volume.

The fill rate bottleneck

The rendering of the shadow geometric primitives drastically increases the overall fill rate cost of the shadow volume algorithm. Several approaches were designed in order to limit this bottleneck. These algorithms are generally complementary and can be used concurrently in order to achieve high fill rate reduction.

A common optimization consists in limiting the shadow volume rendering to the light influence in the image space [MHE+03]. Better fill rate reduction can be achieved by taking also into account the geometry of the receivers [Len05].

Hierarchical shadow volumes [AAM04] are also designed to reduce the shadow volume rasterization cost. This algorithm limits the full resolution shadow volume rendering

to the shadow borders. In addition, thanks to its multi-resolution property, it provides an efficient way to decrease the memory bandwidth requirements of the stencil buffer updates.

Removing shadow volumes that are themselves in shadow does not influence the accuracy of the solution. In addition, the shadow volume rendering affects only the appearance of the receivers. Starting from these observations, the culling and clamping algorithm [LWGM04] culls the shadow volumes that do not influence the accuracy of the shadows and finally clamped the resulting shadow volume polygons to its receiver regions. This drastically reduces the fill rate cost of the shadow volume rendering.

2.2.2 The shadow maps

The shadow map algorithm is the most popular real time hard shadow algorithm. Unlike the shadow volumes, its first formulation [Wil78] was directly consistent and practical for common environments. This image-based approach first rasterizes the scene from the viewpoint of an infinitesimal thin light. This rendering pass initialized a light Z-buffer (*i.e.* the shadow map) that stores the closest surfaces for a set of samples distributed onto the light image plane (figure 2.3). In common rasterizers, this light Z-buffer is defined by a regular grid of pixels where a sample is set at the center of each pixel. In a second step, the scene is rendered from the camera point of view. During this pass, each visible surface sample is projected onto the light image plane in order to define its associated shadow map texel. The real distance from the light source to this surface sample is then compared to the one stored into the shadow map (figure 2.3). If its distance is equal to the one stored into the shadow map, then this surface point is directly visible by the light and so it is illuminated. Otherwise, it is in shadow.

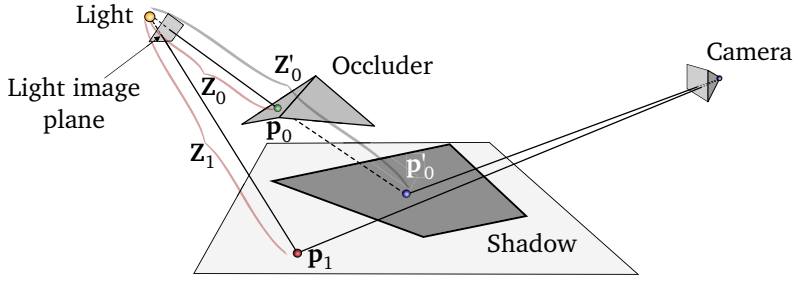


Figure 2.3: The shadow map algorithm. First, the light Z-buffer stores the closest distance from the light to the scene surface (*e.g.* Z_0 and Z_1). For each visible receiver, its distance to the light source is compared to the distance stored into the light Z-buffer. According to the result of the comparison, the receiver is either in the shadow (point P'_0) or illuminated (point P_1).

The simplicity of the shadow map formulation is particularly well suited for their hardware support [SKvW+92]. In addition, thanks to its image-based nature, it is independent of the geometric representation; in other words, it works on all rasterizable primitives without specific treatment. However, this shadow algorithm exhibit a strong accuracy issue: the discretization of the shadow map leads to undersampling artifacts. Indeed, every pixel in the shadow map represents a pyramid of straight lines starting from the light and passing through the pixel on the light image plane. Undersampling artifacts appear when this ray bundle hits a visible surface region greater than the pixel size onto the camera image plane [SD02] (figure 2.4).

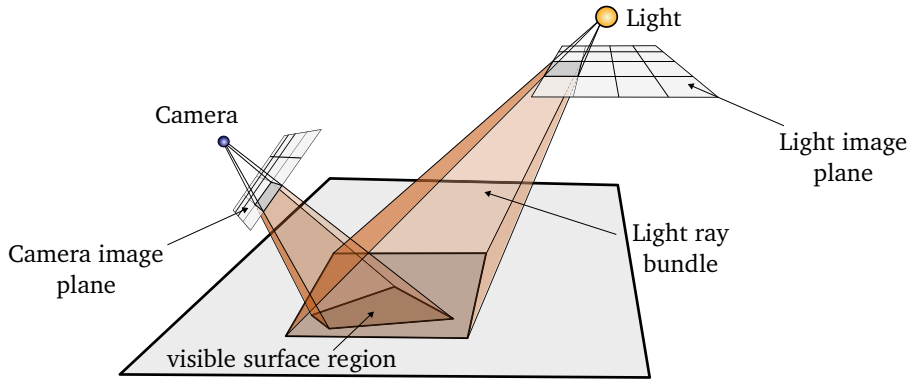


Figure 2.4: Light space undersampling.

These aliasing can be reduced by globally increasing the shadow map resolution. In fact, this obvious solution is both inadequate and inefficient. Providing higher resolution does not address the overall aliasing while it increases both the computational cost and the memory requirements of the shadow computation. As an illustration, in an natural environment, a high resolution for the sun shadow map is useless for the shadows of the far away big trees. On the contrary, its resolution is still insufficient to generate un-aliased shadows casted by the small grass in front of the camera.

In practice, this undersampling issue is particularly difficult to handle. There are principally three approaches that reduce these artifacts: the warping of the shadow map, the local increase of its resolution and its filtering.

Shadow map warping

Warping algorithms reduce the aliasing by deforming the scene as seen by the light viewpoint. The resulting light Z-buffer stores the closest surfaces of the deformed scene. Despite the uniformity of the shadow map, its depth sampling resolution is thus increased in stretched regions and decreased in squeezed parts.

Based on this observation, the perspective shadow map algorithm [SD02] adapts the shadow map resolution according to the projected size of the shadows onto the image plane. First, it transforms the scene and the light by the perspective transformation of the camera. The shadow map is then generated in this space by rasterizing the scene from the transformed light viewpoint. Despite its neatness, this approach exhibits singularities when the lights lie behind the viewpoint. Furthermore, in order to achieve visual pleasant hard shadows, it must pay attention to the relationship between the camera and the light positions [Koz04].

Light space perspective [WSP04] and trapezoidal [MT04] shadow maps generalize the previous warping method. In contrast to perspective shadow maps, these algorithms do not exhibit singularities. Furthermore they equalize the error over the visualized depth range and so they give an equivalent hard shadow quality on all the visible scene distance.

Several shadow map investigations outline that a logarithmic shadow map parametrization would limit the undersampling aliasing [WSP04, LTYM06, FZSXL06]. Thus, by using a logarithmic perspective parametrization, one can drastically reduce the shadow map aliasing [LGT$^+$06, Llo07]. Unfortunately, due to the lack of logarithmic rasterization hardware [LGMM07], such approach is still inefficient.

Non uniform shadow map resolution

In order to adaptively reduce the light Z-buffer undersampling, one can use several shadow maps and define their resolution according to an aliasing error metric. The first algorithm based on multiple shadow maps, partition the visible depth range in order to compute accurate shadows due to the sun-light [TQJN99]. The number of the view frustum partitions depends of the relationship between viewing and solar directions as well as the required shadow accuracy. The parallel split [FZSXL06, ZSN07] and cascaded [Eng07] shadow map algorithms are based on the same overall idea. According to the aliasing error, they split the view frustum and generate several smaller shadow maps for each split part. In the same way, Lloyd *et al.* [LTYM06] propose a metric for evaluating the perspective aliasing error over the view frustum. Finally, they limit the aliasing error of directional lights by using existing partitioning and shadow map warping algorithms.

Rather than partitioning the view frustum, the adaptive shadow maps [FFBG01] use a hierarchical quadtree in order to refine the light Z-buffer in the scene regions with high aliasing error. In the same way, the tiled shadow maps algorithm [Arv04] adapts the resolution of each tiles according to an aliasing measurement heuristic.

By using a general projective transformation one can ensure the one-to-one correspondence between the image plane pixels and the shadow map texels. On the one hand, Chong and Gortler [CG04] use the partitioning of warped shadow maps to achieve this

exact correspondence for few surfaces. On the other hand, the irregular shadow map [AL04, JMB04] is a general purpose data structure that addresses the aliasing error in all situations. This algorithm distributes the shadow map samples according to the projection of visible surface points onto the light image plane. Despite its accuracy, this method requires specific software or hardware [SEA08] implementation still not efficiently generalized to common renderers.

Shadow map filtering

In computer graphics, textures are filtered in order to limit minification or magnification aliasing. In practice, shadow maps are nothing more than textures that store the scene depth as seen by light sources. Thus, reducing their aliasing by using filtering seems particularly attractive. However, a common filtering (bi-linear, tri-linear, anisotropic) is prohibited since it would generate new depth values rather than reducing shadow aliasing.

The percentage closer filtering [RSC87] was specifically designed for the filtering of the shadow maps. It anti-alias the casted shadows by filtering the result of the shadow test for near shadow map texels. In fact, this is very close to standard texture filtering techniques [Hec89]. The main difference is that filtered data are the binary result of the shadow map test of near texels rather than their value. Thanks to this similarity, common graphics hardwares accelerate the percentage closer bilinear filtering. Nevertheless, high shadow quality requires a large number of samples. Unfortunately, since the filtering is performed after the shadow test, it is impossible to use prefiltered mipmaps [Wil83] in order to accelerate this process.

Variance shadow maps [DL06] are quite similar to standard shadow maps. They store both the depth and the squared depth of the environment as seen from the light viewpoint. The *two* first moments of the depth distribution are then evaluated by locally convolving the variance shadow map. From these values, the mean and the variance of the distribution are computed in order to define the probability that visible surfaces are in shadow. The clue of the algorithm is that the computation of the moments is a linear process that can be evaluated with common texture filtering hardware. As a consequence, performances can be greatly enhanced with hardware accelerated bi-linear, tri-linear or anisotropic filtering. Nevertheless, this probabilistic algorithm gives an upper bound on the probability that the surfaces are in shadow. In fact, this upper bound becomes an equality only for planar and parallel occluders but leads to strong light leaking issues when the depth complexity increases. Unfortunately, even though this light leakages can be reduced [Lau07, LM08], they cannot be completely removed.

Convolution shadow maps [AMB+07] do not exhibit light leaking artifacts but keep the desirable linear property of variance shadow maps. This algorithm encodes the visibility

function with respect to a Fourier series expansion. A low order expansion is however quite inefficient to handle function discontinuities. Thus, this algorithm suffers of light bleeding near occluders. In addition, visual pleasant results require a high truncation order that leads to a prohibitive memory consumption. One can note that convolution shadow maps share some similarities with deep [LV00] and opacity [KN01] shadow maps. In the same way, they encode the visibility function with respect to a basis that allows its linear filtering.

Exponential shadow maps [Sal08, AMS+08] are a specialization of convolution shadow maps. This algorithm drastically reduces light bleeding while it improves the performances and reduce the memory requirements. Rather than using the Fourier series expansion, the shadow map values are projected onto exponential basis. As previously, the projection is then stored into basis textures naturally filtered by common texture filtering techniques. Nevertheless, this approach is based on the assumption that filtered depth values are lower than the distance from the light to the receiver. When this assumption is violated, percentage closer filtering has to be used, limiting the performances and the quality of the resulting shadows.

Handling shadow acne and omni-directional lights

A shadow casted by a surface onto itself leads to shadow acne artifacts. Such incorrect self-shadowing is partially due to the finite precision of the computations. It is however emphasized by the discrete nature of the shadow maps. Indeed, each shadow map texel stores the depth of the scene for the direction from the light to its center. Due to this discrete representation, shadow acne appears on surfaces which its distance from the light is greater than its corresponding shadow map value (figure 2.5). This artifact can be limited by increasing the depth discretization resolution [ZSN07] or using a specific depth offset [AMB+07, Lau07]. Nevertheless, no general solution exists.

The shadow map algorithm captures the scene depth according to a given linear projection. Explicit treatments are thus required to handle omni-directional lights. Multiple shadow maps can be captured by performing several linear projection of the scene. A more sophisticated solution consists in using non linear projections reducing the number of shadow maps [BAS02]. However, these projections are not supported by graphics hardware and are approximated using high tessellated objects in conjunction of a non linear projection of their vertexes. In practice, omni-directional shadow maps are simply generated from *six* linear projections of the scene [Die01]. Despite its simplicity, this solution significantly increases the rendering cost as well as the memory requirements of the shadows. In addition, the algorithms addressing the shadow map issues (undersampling, shadow acne, *etc.*) are designed for directional lights. When dealing with omni-directional

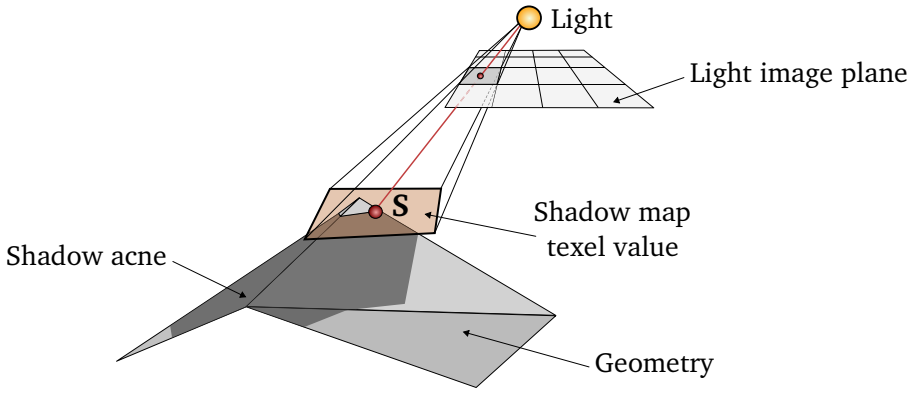


Figure 2.5: For each texel, the shadow map saves the distance from the light to a scene sample S. Due to this discretization, shadow acne artifacts occur when the surfaces are occluded by their own shadow map texel.

lights, their computational cost is thus drastically increased due to the multiple shadow maps used to compute the omni-directional visibility.

2.3 Soft shadows

Real environments are illuminated by extended lights that generate a soft transition between the light and the shadow. The receivers lying into this penumbra region are partially illuminated, *i.e.* they see only a part of the light. In computer graphics, approximating or evaluating the "amount of light" that reaches a surface (equation 2.6) is far more time consuming than solving the binary hard shadow problematic.

Due to the recent impressive increase of both the horsepower and the programmability of the graphics hardware, interactive soft shadows are today widely investigated [HLHS03]. Nevertheless, rasterizing realistic soft shadows in real time is still challenging. One can accumulate several hard shadow renderings in order to numerically evaluate the direct illumination according to a set of light samples [BB84, HH97, Her97]:

$$\begin{aligned} L\left(x' \rightarrow x''\right) &= L_e\left(x' \rightarrow x''\right) + I\left(x' \rightarrow x''\right) \\ &+ \sum_{k \in N} \frac{1}{|\mathcal{L}_k|} \sum_{x \in \mathcal{L}_k} \frac{1}{p\left(x\right)} L_e\left(x \rightarrow x'\right) f_s\left(x \rightarrow x' \rightarrow x''\right) V\left(x \leftrightarrow x'\right) \frac{cos\theta_o cos\theta_i}{||x - x'||^2} \end{aligned} \tag{2.8}$$

where N is the number of extended light sources, $\mathcal{L}_k$ the set of samples distributed onto the light source k, $p\left(x\right)$ the probability distribution function for the sample x and V the visibility evaluated with a hard shadow algorithm. This solution is however particularly inefficient since, as we saw previously, rasterizing hard shadows is not obvious. In order

to efficiently compute soft shadows, specific approaches must be designed.

Many interactive soft shadow algorithms approximate the direct lighting (equation 2.6) by decorrelating the shadow generation from the lighting evaluation. In this situation, the direct illumination of the light is modulated by a visibility coefficient (V_{coef}) representing the visibility integral for the direction towards the area light source:

$$\begin{aligned} L\left(x' \rightarrow x''\right) &= L_e\left(x' \rightarrow x''\right) + I\left(x' \rightarrow x''\right) \\ &+ \sum_{x \in \mathcal{L}'} L_e\left(x \rightarrow x'\right) f_s\left(x \rightarrow x' \rightarrow x''\right) V_{coef}\left(k \leftrightarrow x'\right) \frac{cos\theta_i}{||x - x'||^2} \end{aligned} \tag{2.9}$$

with $\mathcal{L}'$ the set of the positions of the area light centers and k the area light corresponding to x. The V_{coef} represents the percentage of visible light and is given by:

$$V_{coef}\left(k \leftrightarrow x'\right) = \frac{1}{|\mathcal{L}_k|} \sum_{x \in \mathcal{L}_k} \frac{1}{p\left(x\right)} V\left(x \leftrightarrow x'\right) \tag{2.10}$$

Equation 2.8 is a simplification of the direct lighting. It extends the hard shadow problematic (equation 2.7) by replacing the binary visibility V by the V_{coef} attenuation factor. Despite its simplicity, this formulation exhibits some limitations. Indeed, the BSDF of the receiver x'' is evaluated only for the light direction starting from the light center while it must be computed for the bundle of directions between x'' and the area light (equation 2.6). In addition, the emitted radiance is assumed to be constant for each light source. However, this assumption does no remain valid when dealing with lights emitting a spatial varying radiance (*i.e.* textured lights). Despite its drawbacks, the accuracy of the V_{coef} formulation is commonly considered as acceptable when targeting efficient soft shadows. Nevertheless, the use of V_{coef} lead to an approximative direct illumination.

In the following we first present visually plausible soft shadow algorithms. Based on heuristics or the filtering of hard shadows, they provide a visual pleasant approximation of the V_{coef}. We then discuss about more accurate solutions. In contrast to visually plausible approaches, their goal is to explicitly evaluate the V_{coef} (equation 2.10) or the direct lighting integral.

We point out that the goal of this analysis is to give a brief overview of the advantages and drawbacks of the more relevant techniques that rasterize soft shadows in general situations. Thus, we do not investigate the algorithms designed for specific purpose as low frequency shadows [RWS+06], shadows casted by height fields [SN08], *etc.*

2.3.1 Visually plausible soft shadows

In the following, we briefly describe some methods providing a visual pleasant approximation of the V_{coef}. These algorithms are based on the visual aspect of the shadows rather

than a rigorous mathematical and physically plausible background.

We propose to separate this soft shadow algorithms in *three* categories. The first group includes the techniques based on heuristics evaluated using one or several shadow maps. The second class of algorithms approximates the V_{coef} by rasterizing additional shadow primitives. Finally, the last category includes the approaches based on the convolution of hard shadows.

Soft shadow mapping heuristics

The layered attenuation map algorithm [KM99, ARHM00] directly extends the shadow map approach. This algorithm is based on a set of shadow maps generated from several positions onto light sources. These light Z-buffers are used to identify multiple depth layers of the scene (typically $four$) as seen from the light viewpoint. In each layer, each texel stores the distance from the light to the scene as well as a percentage of occlusion (*i.e.* the complementary of the V_{coef}) computed from the underlying samples that it occludes. Soft shadows are finally obtained by modulating the direct illumination by the layered attenuation maps. Despite its similarity with the shadow map approach, in practice, this algorithm is not well suited for real time rendering. Indeed, in order to obtain a visual pleasant V_{coef}, this algorithm requires a high number of shadow map renderings that limits its overall performances.

Heidrich *et al.* [HBS00] computes the V_{coef} using few shadow maps captured from the end vertexes of a linear light. The shadow maps are then analyzed in order to separate the receivers from the occluders. The V_{coef} is approximated by the Gouraud shading of the polygons linking the occluders to the receivers. The performances of the proposed algorithm is however directly linked to the complexity of the casted shadows. Furthermore, despite its extension to polygonal lights [YTD02], it imposes restrictive conditions onto the type of light sources that can be treated.

Unlike the previous approaches, Brabec and Seidel [BS02] use a single shadow map to approximate soft shadows. Their method is in fact very similar to standard shadow mapping. Based on the work of Parker *et al.* [PSS98], they use the depth information encoded into the shadow map to extend the shadow region and create the penumbra. For each visible surface point, the algorithm retrieves its corresponding position into the light image plane. It then identifies the nearest shadow map texel that is illuminated or that is closer to the light, whether the surface is respectively in the shadow or not. The relative position of the receiver, the light and the occluder are finally used to approximate the V_{coef}. Even though this approach is more efficient than using multiple shadow maps, its performances are directly linked to the size of the search region into the shadow map. Finally it suffers of several quality drawbacks due to its inherent assumptions.

The flood fill soft shadow algorithm [AHT04, RT06] approximates the V_{coef} using an heuristic directly evaluated into the camera image plane. First, it generates hard shadows by using shadow maps. Then, it detects the shadow borders by filtering the hard shadows as seen from the camera. Finally it judiciously spreads out the initial binary V_{coef} from the shadow boundary to the neighbor pixels. Pixels lying into shadows are brighten while pixels close to hard shadow boundary are darken. Nevertheless, the multiple rendering passes used to spread out the occlusion information limits the performance. Furthermore, it exhibits strong artifacts that reduce the overall shadow quality.

Object-based approaches

As a first step toward real time object-based soft shadows, Haines [Hai01] rasterizes specific shadow primitives directly onto the planar receivers. First, the umbra region is computed using the shadow volume approach. Plateaus are then extracted from the silhouette edges detected during the shadow volume step and projected onto the planar receivers. The rendering of the projected plateaus finally gives an approximation of the penumbra region. Nevertheless, common environments are particularly difficult to handle since this algorithm only efficiently deals with planar receivers.

On the contrary, the penumbra map [WH03] and the smoothie [CD03] algorithms approximate a V_{coef} independently of the receiver shape. They first evaluate the umbra region using shadow maps. The silhouette edges of the occluders with respect to the light center are then extended perpendicularly to the corresponding occluding surface. Finally, they use these extruded extensions to evaluate a gradual variation of the shadow border that simulates the V_{coef}. Despite their overall efficiency, these approaches can only approximate the outer penumbra part of the soft shadows, *i.e.* the penumbra region lying outside the original hard shadows. As a consequence, the occluders will always cast an umbra even for very thin objects that should cast only penumbra.

Filtering hard shadows

One can observe that convolving the visibility as captured from the light source, would provide a very good approximation of the V_{coef} when the light, the occluder and the receiver lie in parallel planes. From this starting point, Soler and Sillion [SS98] convolve the hard shadows in order to approximate soft shadows. The proposed algorithm recursively subdivides the occluders with respect to a specific error metric. The soft shadows evaluated from each subset of occluders are then combined to obtain the final soft shadows. Due to its error driven nature, the computational cost of this approach is controlled. Unfortunately, the recursive subdivision of the occluders drastically reduces the performances

while the accuracy of the solution is certified only in very specific cases.

Shadow map filtering techniques limit the undersampling artifacts by softening the shadow borders. Following this observation, the percentage closer soft shadow algorithm [Fer05] simulates soft shadows by simply extending the percentage closer filtering algorithm [RSC87]. It simulates the variation of the penumbra size by defining the filter kernel width according to the distance from the light source to the occluders lying between the emitter and the receiver. Due to its simplicity, this approach is widely used and extended in real time applications. In order to improve its efficiency, one can filter only the shadow border [Ura05]. Finally, better shadow quality is obtained by using non correlated Monte Carlo sampling patterns [Ura05, Mit07].

Dong *et al.* [DAM+08] follow the same overall idea. In the same way, they approximate the penumbra size variation by defining the filter kernel width from the average of the distances from the light to the occluders. The main difference with the previous approach relies on the shadow filtering technique: they use the convolution shadow maps [AMB+07] rather than the percentage closer filtering. Despite real time performances, this algorithm has to deal with the high memory consumption required by the convolution shadow maps.

The occlusion texture algorithm [ED06b] is based on a layered discretization of the scene as seen from the light center point of view. In contrast to layered attenuation maps [KM99], this algorithm directly generates the sliced representation of the environment in one rendering pass [ED06a]. For a receiver point, it then approximates a visual pleasant V_{coef}, by filtering adaptively each occlusion texture lying between the light and the receiver. The filter kernel width is defined according to the distance from the light to the receiver as well as the size of the light source. This method allows the real time rendering of visual pleasant soft shadows. Nevertheless, due to the discrete sliced representation of the scene it cannot handle properly near self shadowing. In addition, the discretization leads to light leakages that have to be explicitly addressed for thin occluders.

2.3.2 Physically plausible soft shadows

In contrast to visually plausible soft shadows, physically plausible solutions are based on the explicit evaluation of the V_{coef} (equation 2.10) or on the direct lighting integration (equation 2.6). According to the environment, they explicitly evaluate the area light visibility rather than approximate a V_{coef} from a specific visual heuristic.

Due to the complexity of the problematic and its computational requirement, few algorithms target the interactive generation of physically plausible soft shadows. In this section we present *three* investigations toward the interactive rasterization of realistic soft shadows. The first class of algorithms precomputes the visibility information and numerically solve the direct lighting at runtime. The second category extends the shadow

map framework in order to evaluate a physically plausible V_{coef}. The last one regroups object-based techniques, *i.e.* algorithms that explicitly lie onto the underlying geometry representation of the occluders.

Precomputed visibility

The on line evaluation of the visibility is particularly time consuming. As an alternative, the visibility informations can be precomputed in order to reduce the runtime computational cost. Several approaches capture the whole radiance transfer for static scenes illuminated by distant lights [SKS02, SLSS03, Leh04, NRH04]. The radiance informations are then highly compressed and finally reconstructed in real time using graphics hardware. Despite convincing global illumination results including shadows, dynamic scenes are difficult to handle with such approaches since the visibility cannot be precomputed anymore. Many algorithms are designed to address this drawback. In practice, these propositions are often either too expensive in terms of memory requirement and computation time [RHCB05], or not well suited for all frequency shadows, local illumination, *etc.* [ZHL+05, SM06].

As a straightforward alternative, one can pre-capture several shadow maps on a per-object basis with respect to a set of directions [MSW04]. This discretized representation is then used to solve the visibility queries at runtime. Based on the same overall idea, the coherent shadow map algorithm [RGKM07] precomputes the direct visibility information for rigid objects. In order to reduce the memory cost, it exploits the coherence of the captured values to compress the resulting set of shadow maps. It finally uses this visibility representation to define the visibility informations required by the numerical integration of the direct lighting. This approach has to deal with strong drawbacks. The visibility reconstruction is time consuming while, despite the compression of the visibility, the memory requirement is still prohibitive. In addition, deformable meshes as well as common light sources (dynamic point or spot lights) are not supported. Finally, even though the objects can be animated, artifacts appear when their convex hull are intersecting.

Soft shadow mapping

A shadow map can be considered as a surjective discrete representation of the scene as seen from the light viewpoint. In this representation, each shadow map texel stores the distance from the light source of a quad parallel to the light image plane. This is the clue behind the soft shadow mapping algorithm [ASK06, AHL+06, BCS06, GBP06]. This algorithm evaluates the V_{coef} by back projecting [DF94] the shadow map samples onto the light plane. This back projection can be performed from the light point of view by

explicitly separating the occluders from the receivers [AHL+06]. Despite its apparent simplicity, this approach has to deal with undersampling and light leaking artifacts. In addition, for each receiver the whole occluding samples are back projected onto the light. Consequently, the performances are inversely proportional to the shadow map resolution.

Better results are obtained by evaluating the V_{coef} from the camera viewpoint [GBP06]. Indeed, this avoids the explicit distinction between occluders and receivers and reduce the soft shadow undersampling artifact. With this strategy, each receiver has to define the shadow map region affecting its area light visibility. The shadow map samples lying into this area are finally back projected onto the light plane in order to integrate the percentage of visible light, *i.e.* its V_{coef}. Thanks to a hierarchical organization of the shadow maps, this approach limits the number of the back projected shadow map samples. Nevertheless, the surjective nature of the shadow map leads to light leakages that must be addressed with a gaps filling heuristic.

This latter back projection algorithm was widely extended to improve its efficiency and its visual quality [GBP07, SS07, SS08, SS]. However, the inherent discretization of the shadow map used to represent the scene, limits the overall accuracy of the proposed solutions.

Object-based physically plausible soft shadows

As a direct extension of the shadow volumes, Akenine-Möller and Assarsson [AMA02] introduce the penumbra wedge algorithm. Its implementation onto graphics hardware [AAM03, ADMAM03] relies on *two* rendering passes. First, the hard shadows are computed using shadow volumes and are used to initialized the V_{coef}. Then, a penumbra wedge is extruded from each silhouette edge detected during the shadow volume step. This primitive conservatively includes the scene volume which area light visibility is influenced by its associated edge. For each receiver lying into the penumbra wedge, its V_{coef} is then updated with respect to the percentage of light occluded by the corresponding edge.

In fact, this algorithm computes the V_{coef} analytically . However, it relies on the assumption that silhouette edges are not overlapping as seen from the light center. This leads to penumbra overestimation artifacts when this constraint is not satisfied. In addition, while the silhouette edge detection from the light center is accurate for point light, it does not remain correct when dealing with surfacic light sources.

Lately, Sintorn *et al.* sample the light visibility by rasterizing shadow primitives from the light point of view [SEA08]. First they generate an alias free shadow map [AL04] by storing for each shadow texel the list of visible receivers as seen from the light viewpoint. For each triangle, they build a conservative primitive that includes the scene region influenced by the shadow of the triangle. This primitive is then conservatively

rasterized [HAMO05] from the light viewpoint into the alias free shadow map. For each shadow map texels covered by the primitive, the algorithm tests which receivers effectively lie into the shadows of the triangle. For these receivers it finally defines the occluded light sample in order to numerically solve its light visibility. As a main advantage, this approach provides an accurate evaluation of the visibility integral. In contrast to the penumbra wedge approach, the shadow computations are performed per triangle rather than per silhouette edge. This avoids the silhouette detection step but on the counter part increases the soft shadow computation requirements. The per light computation cost is furthermore linear to the number of alias free shadow maps used to capture the light influence. Thus, handling omni-directional lights is time consuming. Finally, one can note that the conservative rasterization increases the overall rendering cost. Due to these performance overheads, this algorithm provides an interactive solution only in simple situations (few directional lights, low geometric complexity, *etc.*).

2.4 Discussion

Despite many investigations very few algorithms generate robust shadows. Many favor efficiency rather than accuracy. While they can offer good performances [Ura05, DAM+08], such approximations are not suitable for direct lighting simulation. Others precompute the visibility in order to accelerate the runtime direct lighting evaluation [RGKM07]. However, due to strong limitations, they are not well suited for general environments. Some others target the explicit evaluation of the V_{coef}. Despite convincing results [AAM03, GBP07], they exhibit robustness issues and still rely on the V_{coef} approximation of the direct lighting.

In the following we first define what should be a robust shadow algorithm. From these observations and the previous analysis of existing solutions, we then discuss why we choose the object-based framework as a starting point toward the robust shadow simulation.

2.4.1 Designing a robust shadow algorithm

To be widely used, realistic shadow algorithms must be independent of the underlying environment. Thus, hand fixed scene dependent parameters have to be prohibited. In addition it may have to pay attention to neither the light type nor the model representation or shape.

In the same way, the desired accuracy of the solution has to be independent of the scene: realistic soft shadows must be generated whatever the light positions, the object relationship or the camera point of view. In other words, the algorithm must be robust.

Nevertheless, exact soft shadows are not required all the time, in all situations. Some applications prefer efficient solutions to accurate computations. Others use approximations as a fast overview of the accurate case. An unified shadow algorithm that can evaluate both a fast approximation and an efficient accurate solution would thus be a very attractive and powerful tool.

2.4.2 Image-based VS object-based framework

Due to its generality, the image-based framework seems to be a good starting point toward a general purpose robust shadow algorithm. Indeed, image-based approaches lie on the same background than the rasterization rendering algorithm. Thus, they naturally fit to current graphics hardware and take benefit of its efficiency (texture access, filtering, *etc.*). In addition, their performances are not explicitly influenced by the geometric complexity of the scene or the underlying object representation. Nevertheless, the evaluation of generic and artifact free image-based shadows is particularly difficult. Several investigations specifically target shadow acne, undersampling artifacts or omni-directional light sources. In fact, despite the apparent simplicity of image-based shadows, it turns out that no practical general purpose solution avoids their initial pathological issues.

Due to their geometric nature, object-based frameworks rely on geometric constraints [AMA03] and their performances are drastically influenced by the complexity of the models. Nevertheless, unlike image-based methods, their robust implementation does not exhibit discretization artifacts (shadow acne, undersampling aliasing, shadow popping) and does not require specific treatments for omni-directional lights. The basic reason for preferring an object-based framework is that it is far less fragile than an image-based approach. This robustness property seems particularly attractive when targeting a general and accurate solution.

The conventional wisdom in real time rendering is that object-based approaches are too expensive, and that acceptable shadows can be achieved by using the simplest image-based methods. Moreover, image-based shadows are considered more general than object-based ones. However, there is little research to support this claim. While there has been many works targeting the drawbacks of the shadow maps very few researches investigate the limitations of object-based techniques. In our view, more researches are necessary before judging of the capabilities of the object-based shadow framework.

2.4.3 Conclusion

We argue that object-based shadow approaches are well suited for the interactive evaluation of physically plausible shadows. Due to their object-based nature, their accuracy

rely on the explicit geometry of the meshes rather than an intermediary discrete data structure. Thus, object-based algorithms have to pay attention to neither shadow undersampling artifacts nor hand fixed discretization parameters. In practice, object-based shadows are far more predictable than image-based shadows. For instance, the shadow volumes generate pixel accurate hard shadows while the shadow maps have to deal with aliasing and shadow acne. In fact, the object-based shadow framework provides a very strong algorithmic background particularly well suited for the robust simulation of direct shadows.

3

Penumbra wedge blending

In this chapter we first detail the penumbra wedge framework. This object-based algorithm directly extends the shadow volume algorithm in order to compute realistic soft shadows. It analytically evaluates a physically plausible V_{coef} by assuming that the penumbra of the occluders are not overlapping. The violation of this assumption leads to V_{coef} underestimation and thus to over shadowed regions.

In the second part of the chapter we explicitly address the penumbra overlapping artifacts. The presented algorithm is based on a new blending heuristic that estimates the overlapped error in order to propose a realistic penumbra blending that drastically reduces the over shadowed artifacts.

The rest of the chapter is more practical. We detail our implementation of the penumbra wedge framework improved by our blending heuristic. We then present the visual results of our blending process and its impact onto the performances. As a conclusion, we discuss the overall limitations of the penumbra wedge algorithm as well as our improvement on both theoretical and implementation point of view.

3.1 The penumbra wedge algorithm

Due to hardware limitations, the original penumbra wedge algorithm [AMA02] was based on a combination of hardware and software rasterization. This implementation exhibits some robustness issues. Its later formulation onto graphics hardware [AAM03] is more practical. It allows real time performances and avoid the main drawbacks of its initial formulation.

This section describes this latter implementation. We first give an overview of the algorithm before describing its *two* main steps. We finally outline its main limitations.

3.1.1 Overview

Algorithm 1 render_scene_pwedge(world, view)

```
1: set_up_camera(view);
2: (color-buffer, Z-buffer) ⇐ draw_ambient_and_emissive_lighting(world);
3: for all light ∈ world do
4:    compute_vbuffer(world, light, V-buffer, Z-buffer);
5:    color-buffer + = direct_lighting(world, light, V-buffer);
6: end for
```

The hardware implementation of the penumbra wedge algorithm approximates the rendering equation in *three* steps (algorithm 1). First, it initializes the color buffer from the emissive and the indirect illumination of the scene (line 2). Then, for each light source, it computes a visibility buffer (V-buffer) encoding for each view sample its associated V_{coef}. The direct lighting contribution of the light source is then attenuated according to the V-buffer, and finally accumulated into the color buffer (line 5).

Algorithm 2 compute_vbuffer(world, light, V-buffer, Z-buffer)

```
1: edge ⇐ detect_silhouette_edges(light, world);
2: V-buffer ⇐ render_shadow_volumes(edges, world, Z-buffer);
3: wedges ⇐ build_wedges(edges);
4: render_pwedges(wedges, light, V-buffer, Z-buffer);
```

The clue of this soft shadow algorithm is the computation of the V-buffer (algorithm 2). This buffer is first initialized by the robust shadow volume rendering (line 2). Then a *penumbra wedge* is extruded (line 3) from each silhouette edge previously detected (line 1). The V-buffer is finally compensated by the rendering of the penumbra wedges (line 4).

The robustness of this algorithm relies on *two* major steps: the construction of the penumbra wedge primitive and its rendering.

3.1.2 The penumbra wedge primitive

A penumbra wedge includes the penumbra volume of its associated silhouette edge. Computing the exact penumbra region of an edge is however particularly time consuming. Nevertheless, a robust soft shadow evaluation does not require a tight approximation of the penumbra volume. In fact, the penumbra wedge defines a conservative bounding volume of the "edge penumbra influence".

A penumbra wedge is composed of $five$ planes (figure 3.1(a)). Each penumbra primitive is extruded independently of the others. Its robust construction is performed as follows. Considering a silhouette edge e defined by *two* vertexes v_0 and v_1. First, the

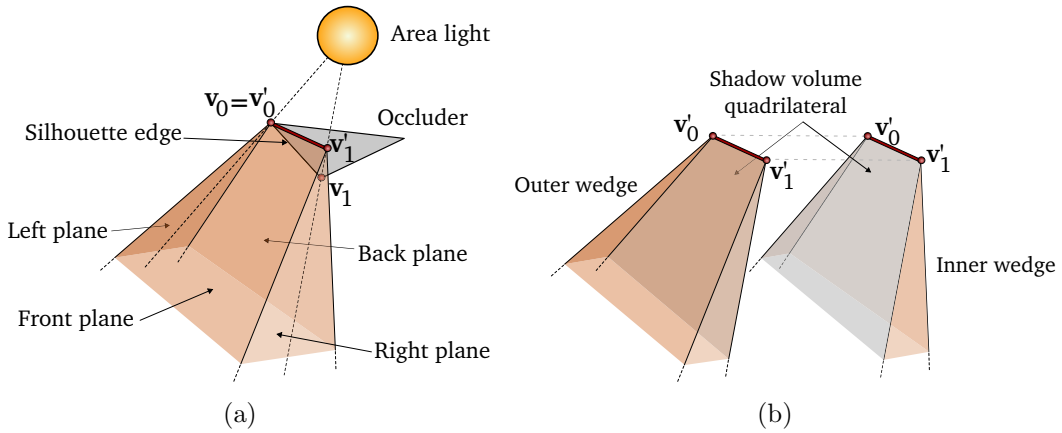


Figure 3.1: (a) The wedge planes. In practice, in addition of the front, back, right and left planes, a plane caps the penumbra wedge. Due to the infinite/semi infinite projection of the wedge, this plane is not illustrated here. (b) Explicit separation of the penumbra wedge into an outer and an inner wedge.

algorithm defines the vertex that is closest to the center of the area light source l_c. The other vertex is moved toward l_c until it is at the same distance of the light center than the first vertex. This results in *two* vertexes v'_0 and v'_1. Note that this new edge is used to ensure that the corresponding wedge contains the entire penumbra region of e. The original silhouette edge is still associated to the penumbra wedge and it is used for the penumbra computation.

The front and back planes of the penumbra wedge are defined by rotating in opposite directions the shadow volume quadrilateral of e around the axis $\overrightarrow{v'_0v'_1}$ until they are tangent to the light source. The left plane contains the vertex v'_0 and it is rotated around the axis perpendicular to the vector $\overrightarrow{v'_0v'_1}$ and the vector $\overrightarrow{v'_0l_c}$ until it becomes tangent to the light source. Note that this axis is in fact the normal of the shadow volume quadrilateral. The right plane is constructed similarly; it contains v'_1 and it is rotated around the same axis in the opposite direction until it is tangent to the light. The fifth plane is the back caped of the penumbra wedge and it is required by the robust rendering of the penumbra wedge (section 3.1.3).

We point out that each penumbra wedge is in fact split in *two* parts by the shadow volume quadrilateral (figure 3.1(b)). The outer and the inner wedges include the penumbra region lying outside or inside the associated shadow volume quadrilateral, respectively. This explicit distinction between the outer and the inner penumbra is the clue of the efficient evaluation of the V_{coef} (section 3.1.3).

The penumbra wedge extrusion can be performed with either an infinite or a semi infinite projection of its vertexes. In order to avoid robustness issues, one have to prefer the infinite projection. We refer to the appendix 3.B for a practical implementation of

such robust construction.

3.1.3 Rendering the penumbra wedge

The V-buffer is first initialized by the robust shadow volume algorithm. The penumbra wedge rasterization generates the penumbra regions by modulating this hard shadow result. The following algorithm (algorithm 3) describes this compensation step. Its basic idea relies on the analytical integration for each visible receiver of the occluded light area with respect to the silhouette edges.

Algorithm 3 render_pwedge(wedges, light, V-buffer, Z-buffer)

```
 1: for all half_wedge ∈ wedges do
 2:    receivers ⇐ fail_zbuffer_visibility(Z-buffer, half_wedge);
 3:    for all p ∈ receivers do
 4:       edge ⇐ half_wedge.silh_edge;
 5:       edge^p ⇐ project(edge, light, p);
 6:       edge^p_c ⇐ clip(edge^p, light);
 7:       occ ⇐ occluded_percentage(edge^p_c, light);
 8:       if is_outer_wedge(half_wedge) then
 9:          V-buffer[p] + = occ;
10:       else
11:          V-buffer[p] − = occ;
12:       end if
13:    end for
14: end for
```

Each half wedge is rendered independently (line 1). Its rendering is based on the failure of the visibility test with respect to the Z-buffer of the scene (line 2) [EK02]. This visibility strategy defines the surface points **p** potentially included into the half wedge (line 3). For such receivers, the silhouette edge associated to the penumbra wedge (line 4) is projected onto the light source as seen from **p** (line 5). This projected edge is then clipped against the light border (line 6). The percentage of the light that is occluded by the clipped edge is evaluated according to the light center (line 7). Finally this occluded percentage is added or subtracted to the V-buffer whether the half wedge is either an outer (line 9) or an inner wedge (line 11). The figure 3.2 illustrates this algorithm.

In summary, the V-buffer is initialized by the shadow volume stencil test. The result represents the number of shadow volumes in which the receivers lies. The wedge rendering finally compensates these values in order to generate the penumbra region. In fact, the data stored into the V-buffer are not V_{coefs}. The penumbra wedge algorithm assimilates them as the complementary of V_{coefs}, *i.e.* the percentage of the light that is occluded. The V_{coef} of the receivers is retrieved by computing the complementary of the V-buffer

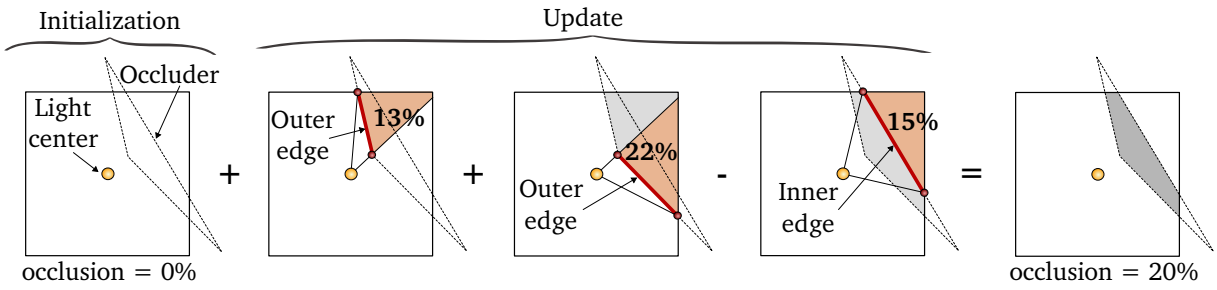


Figure 3.2: Integration of the light occlusion percentage for a given receiver. The rectangular light and the occluder is represented as seen from the receiver. The shadow volume step first initializes the percentage of occlusion that is then updated by the penumbra wedge integration rules.

clamped between *zero* and *one*. Due to the simplicity of this V_{coef} derivation, it is however common to define this intermediary data structure as a V-buffer.

3.1.4 Discussion and limitations

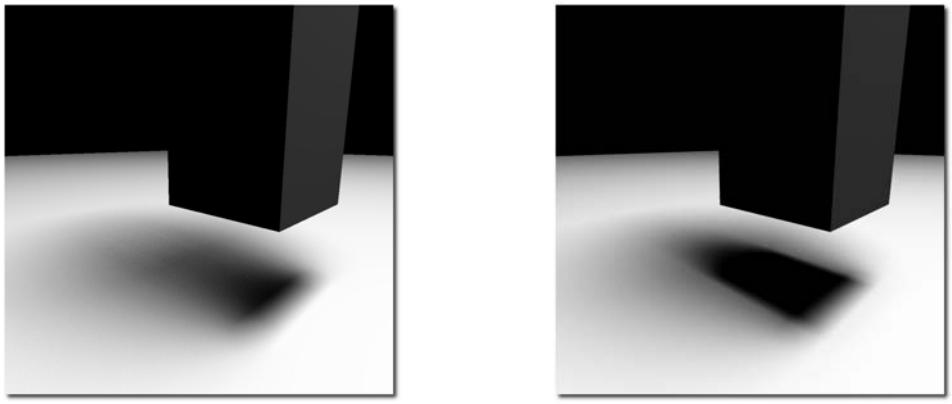
Despite the neatness of the analytical evaluation of the V_{coef}, the penumbra wedge algorithm has to deal with robustness issues.

The single light sample artifact

The penumbra wedge algorithm uses the silhouette edges of the occluders detected with respect of a single point onto the light source, typically its center. While such detection is accurate with point lights, it does not remain correct when dealing with area light sources. Indeed, the silhouette may deffer by using different positions onto the light source. In fact, the detection of the silhouette edges has to be performed according to the shape of the area light [LAA+05]. Due to this single light sample simplification, occluding silhouette edges are missed leading to light bleeding artifacts (figure 3.3).

The silhouette overlapping artifact

One of the main advantage of the penumbra wedge algorithm is that the construction of the wedges as well as their rendering are independent. This independence relies on the assumption that the silhouette edges are not overlapping as seen from the receivers. Nevertheless, this assumption is seldom verified in practice. When silhouettes are overlapping, the overlapped occluded region is take into account several times (figure 3.4). In such situation, the resulting V_{coef} is underestimated resulting in over shadowed areas (figure 3.5(a)).



(a) (b)

Figure 3.3: (a) Illustration of the single light sample artifact. (b) Ray-traced reference [mi].

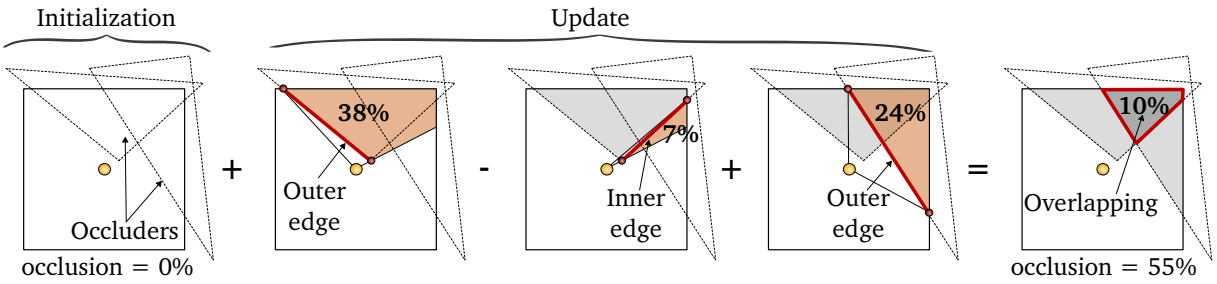


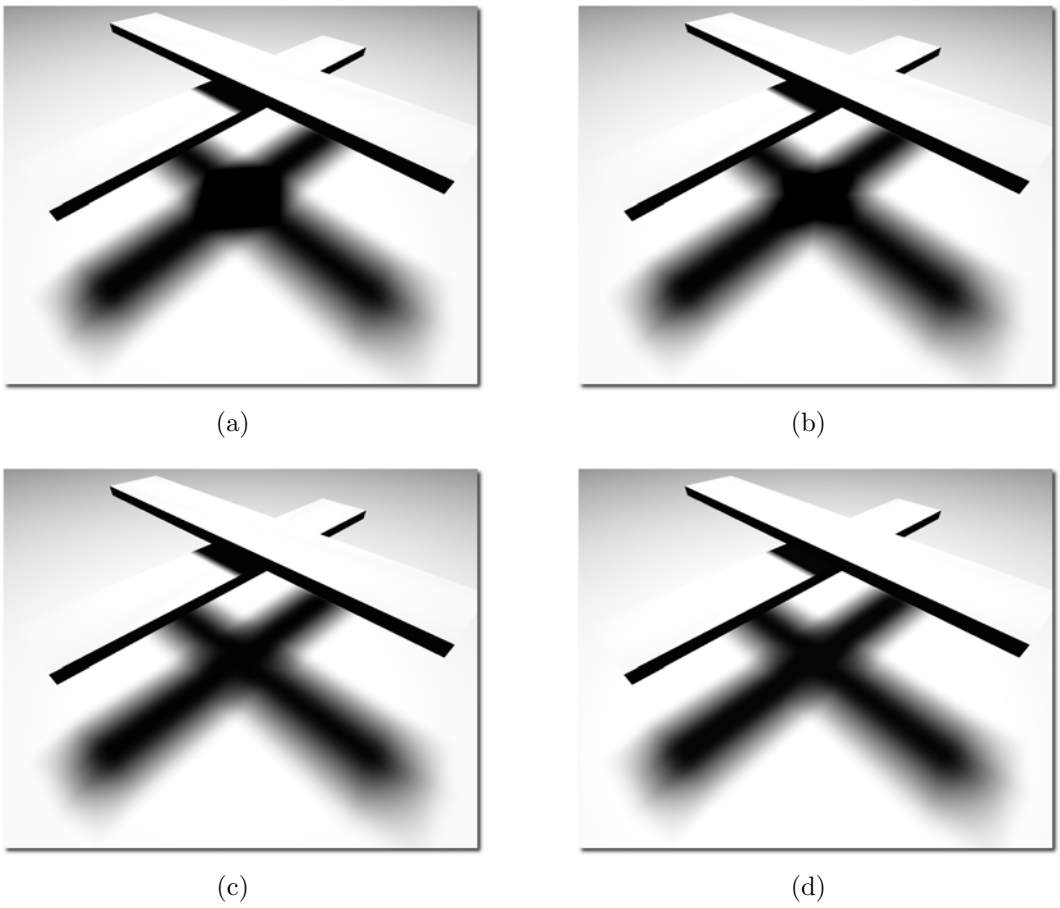
Figure 3.4: When occluders are overlapping as seen from the receiver, the overlapped occluded region is counted several times. This results in an over estimation of the occlusion percentage.

3.2 The penumbra wedge blending

The silhouette overlapping issue is the major drawback of the penumbra wedge framework. This artifact occurs for every receivers lying in the shadow of multiple occluders. Such situation is very common in real environments and one can exhibit over shadowed region even on very simple scenes (figure 3.5).

In order to avoid the penumbra overlapping drawback, the integration of the V_{coef} must be performed with respect to the light area already occluded by previous silhouette edges. This update strategy requires a global knowledge of all the silhouette edges that affect the visibility of the receiver. Nevertheless, the efficiency of the penumbra wedges relies on the locality of the computations and the independence between the occluders. Using a global constraint would consequently reduce the overall performances.

In the following we present a new technique that blends the soft shadows cast by several silhouette edges. In order to keep the local property of the penumbra wedge



(a) (b)

(c) (d)

Figure 3.5: Penumbra blending comparison. (a) standard penumbra wedge algorithm [AMA02], (b) penumbra blending by a probabilistic approach [AAM03], (c) our penumbra blending heuristic, (d) reference image using 1024-sample shadows

computations, we propose a per receiver compact representation of the occluded part onto the light source. We use this approximation to blend the penumbra wedge contribution according to a new blending heuristic that drastically reduces the overlapping artifacts.

The proposed approach (algorithm 4) relies on two main steps: a per silhouette loop visibility buffer evaluation (SV-buffer) (line 6), and its blending with the final V-buffer (line 7). These buffers encode an approximation of the geometry that occludes the light and the corresponding percentage of occlusion. In the following section, we define how we compute the SV-buffer. Then, we describe the blending step of the SV-buffer into the V-buffer.

Algorithm 4 compute_enhanced_vbuffer(world, light, V-buffer, Z-buffer)

```
1: silhouette_loops ⇐ detect_non_overlapped_silhouette_loops(light, world);
2: for all loop ∈ silhouette_loops do
3:    edges ⇐ get_edges(loop);
4:    SV-buffer ⇐ render_silhouette_shadow_volumes(edges, world, Z-buffer);
5:    wedges ⇐ build_wedges(edges);
6:    render_silhouette_pwedges(wedges, light, SV-buffer, Z-buffer);
7:    blend_vbuffers(SV-buffer, V-buffer, light);
8: end for
```

3.2.1 The silhouette visibility buffer

The detection of the silhouette edges with respect to the light center leads to a set of silhouette loops. We decompose them into silhouette loops that are not overlapping. Consequently, the evaluation of the per silhouette loop visibility buffer (SV-buffer) does not exhibit overlapping artifacts.

Algorithm 5 render_silhouette_pwedge(silh_wedges, light, SV-buffer, Z-buffer)

```
1: for all half_wedge ∈ silh_wedges do
2:    receivers ⇐ fail_zbuffer_visibility(Z-buffer, half_wedge);
3:    for all p ∈ receivers do
4:       edge ⇐ half_wedge.silh_edge;
5:       edge^p ⇐ project(edge, light, p);
6:       for all light_part ∈ light do
7:          edge^p_c ⇐ clip(edge^p, light_part);
8:          occ ⇐ occluded_percentage(edge^p_c, light);
9:          brect ⇐ compute_brect(edge^p_c, light);
10:         if is_outer_wedge(half_wedge) then
11:            SV-buffer[p][light_part].occ + = occ;
12:            increase_brect(brect, SV-buffer[p][light_part].brect);
13:         else
14:            SV-buffer[p][light_part].occ − = occ;
15:            decrease_brect(brect, SV-buffer[p][light_part].brect);
16:         end if
17:      end for
18:   end for
19: end for
```

The proposed evaluation of the SV-buffer is quite similar to the original V-buffer computation [AAM03]. First, we initialize the SV-buffer by the hard shadows generated from the shadow volume of the silhouette loop (algorithm 4 line 4). Then we compensate it by the rendering of its associated wedges (algorithm 5). As with the original penumbra wedges, for each half wedge (line 1), we define the potential receivers according to the

failure of the Z-buffer visibility test (line 2). For such surface points **p** (line 3), we retrieve the silhouette edge associated to the half wedge (line 4) that we project onto the light source as seen from **p** (line 5). In contrast to the common wedge rendering, the light source is subdivided into several parts with respect to the viewpoint of **p**. This subdivision allows a better approximation of the occluding geometries. We choose to split the light source by its center, in *four* parts with equal area. We treat, each light part separately (line 6). We clip the projected edge against the border of the subdivided region (line 7). Then, we compute its corresponding percentage of occlusion (line 8). We also evaluate a bounding rectangle that includes the occluded part (line 9). Finally, we update the SV-buffer according to the type of the half wedge (line 10). We compensate the percentage of occlusion as defined by the penumbra wedge integration rules (line 11 and 14) while the bounding rectangle is used to adjust the approximation of the occluded region into the light part (line 12 and 15). This algorithm is summarized in the figure 3.7.

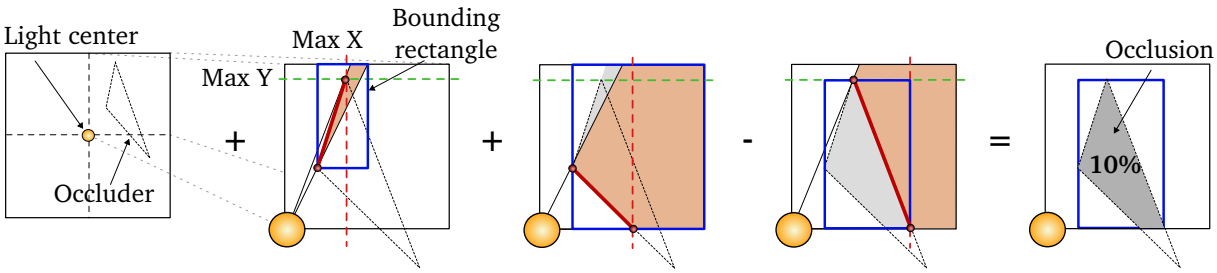


Figure 3.6: Per light part computation of the occluded light region as well as its associated bounding rectangle.

3.2.2 The penumbra blending

The blending of the SV-buffer into the V-buffer is performed as follows (algorithm 6). For the visible surface points (line 1), we treat each light part independently (line 2). The percentage of occluded light saved into the SV-buffer is first accumulated with the one stored into the V-buffer (line 7). In a second step, we compensate the potential overlapping error. We define the intersection between the bounding rectangles of the occluded area encoded in the *two* buffers (line 8). If the intersection area is not null then there is a potential overlapping issue (line 9). We use an Effective Overlapped Area Heuristic (EOAH) in order to approximate the overlapping error (line 10). This value represents the percentage of occlusion that are overlapping. It is given by:

$$\varepsilon = \frac{A_v}{B_v} \cdot \frac{A_s}{B_s} \cdot \beta \tag{3.1}$$

with ε the effective overlapped area, β the area of the intersection of the bounding rectangles, A_x the light occlusion area and B_x the area of the bounding rectangles. The indexes s and v identify respectively the SV-buffer and the V-buffer. We subtract the effective overlapped area from the previously accumulated percentage of occlusion (line 11). We finally combine the bounding rectangles to update the localization of the occluding geometry (line 14), used as the input for the next blending computations.

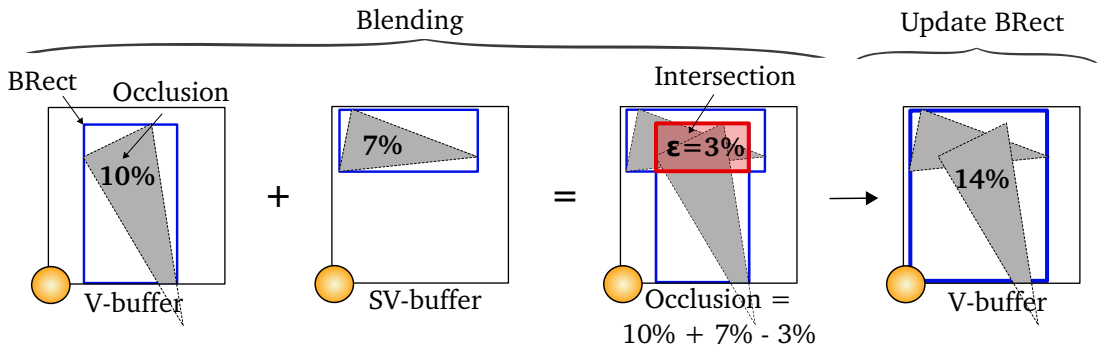


Figure 3.7: Per light part penumbra blending step.

Algorithm 6 blend_vbuffers(SV-buffer, V-buffer, light)

Require: SV-buffer.length() == V-buffer.length();

```
 1: for all p ∈ SV-buffer do
 2:   for all light_part ∈ light do
 3:     occ_s ⇐ SV-buffer[p][light_part].occ;
 4:     brect_s ⇐ SV-buffer[p][light_part].brect;
 5:     occ_v ⇐ V-buffer[p][light_part].occ;
 6:     brect_v ⇐ V-buffer[p][light_part].brect;
 7:     blended_occ ⇐ occ_s + occ_v;
 8:     brect_i ⇐ brect_intersection(brect_s, brect_v);
 9:     if area(brect_i) ≠ 0 then
10:       overlapped_occ ⇐ EOAH(occ_s, occ_v, brect_s, brect_v, brect_i, light);
11:       blended_occ -= overlapped_occ;
12:     end if
13:     V-buffer[p][light_part].occ ⇐ blended_occ;
14:     V-buffer[p][light_part].brect ⇐ merge_brect(brect_s, brect_v);
15:   end for
16: end for
```

3.3 Implementation

This section presents an OpenGL [SA06] implementation of our blending algorithm. The proposed algorithm targets common programmable graphics hardware and uses both vertex (VP) [AAA+02] and fragment programs (FP) [BBC+02]. Since our algorithm extends the penumbra wedges, we first investigate a robust and efficient implementation of this object-based framework. Then we outline the penumbra wedge rendering strategy that we are based on. We finally expose how our blending heuristic is implemented onto common Graphic Processor Units (GPU).

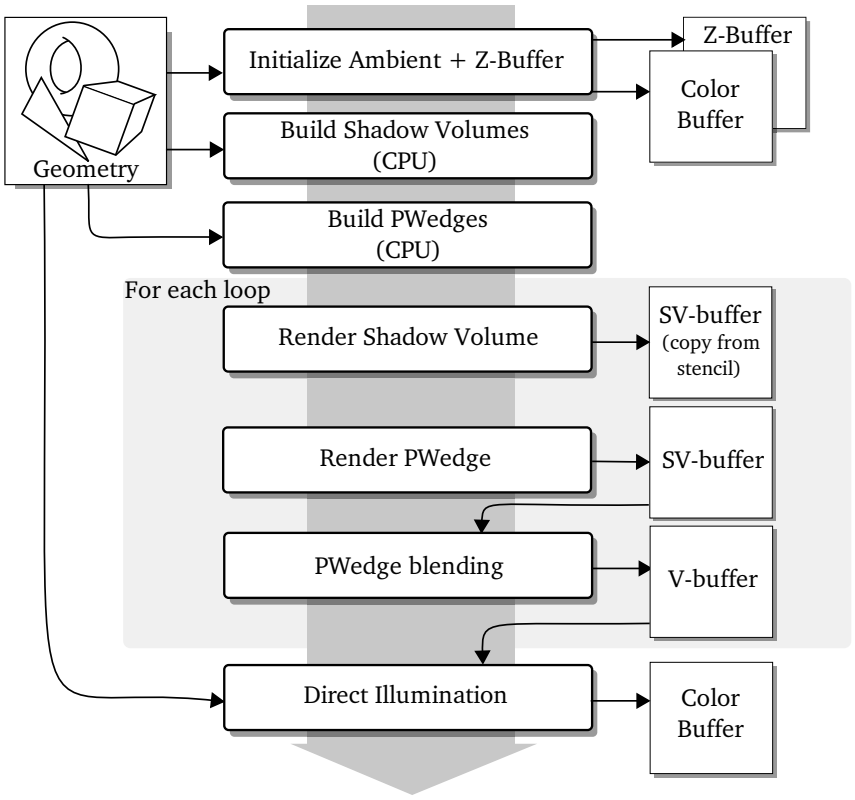


Figure 3.8: Overview of our GPU implementation of the proposed penumbra wedge blending algorithm.

3.3.1 The shadow volume framework

Our shadow volume implementation is based on its robust formulation, *i.e.* infinite projection and Z-fail stencil update strategy [Car00, EK02]. Because the silhouette edge selection is CPU intensive, we use a half edge structure [Ket99] both accelerating the edge selection and providing the connectivity of the silhouettes. Thanks to this explicit connectivity, the shadow volume quadrilaterals of the silhouette loops are encoded as

triangle strips in order to optimize the post transform cache efficiency of the GPU. The stripped quads are saved on GPU side using buffer objects [SA06]. Their infinite extrusion is performed by a specific VP with respect to the homogeneous coordinate of the vertexes (appendix 3.A).

We perform the Z-fail stencil update in one rendering pass by using the `EXT_stencil_two_side` extension [Kil05]. Due to API limitations, we cannot explicitly access to the stencil buffer. Thus, we use a specific render target in which we copy the stencil values.

3.3.2 The penumbra wedge framework

As the shadow volume algorithm, the penumbra wedge algorithm is based on the rendering of non existing primitives. The penumbra wedges are extruded on the CPU with respect to the silhouette edges previously detected. For attenuated lights, we reduce the fill rate requirements with scissor and depth bound tests as defined during the shadow volume rendering pass. We still reduce the fill rate consumption by using a set of specific Z-buffer tests with respect to the half wedge type as well as their orientation. For front facing wedges (figure 3.9 left), the back faces of their outer wedges are rasterized if they *fail* the Z-buffer test while the front face of their associated inner wedges are rendered if they *pass* the Z-buffer visibility test. This process is reversed for back facing wedges (figure 3.9 right).

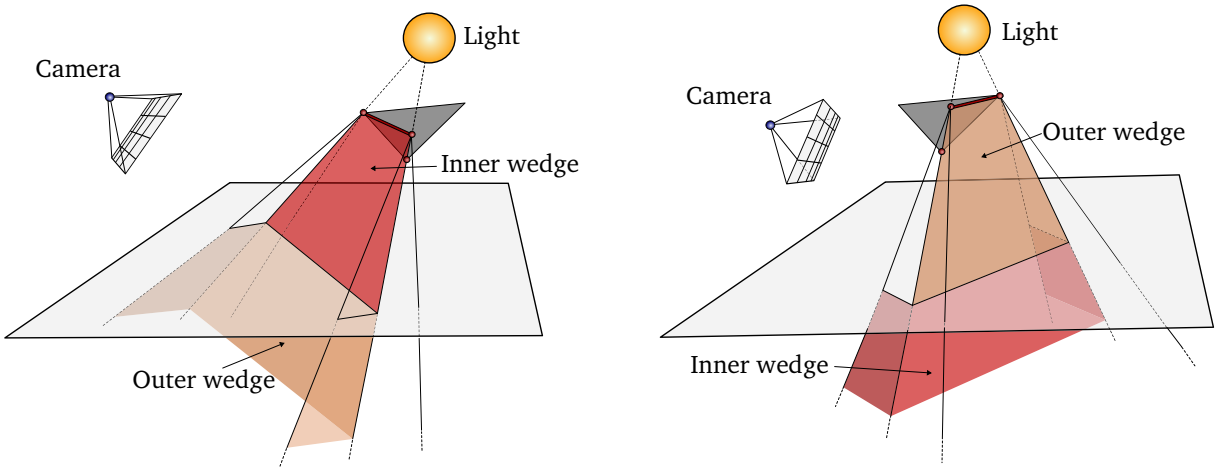


Figure 3.9: Z-buffer visibility test used during the penumbra wedge rasterization. The test is performed according to the orientation of the shadow volume quadrilateral with respect to the camera. Left: front facing wedge; Right: back facing wedge.

The compensation of the V-buffer is particularly time consuming. We reduce the computational cost by limiting the penumbra wedge evaluation step to a tight approximation of the receivers effectively occluded by the associated edge. To do this, we compute the

three outside bounding plane (*i.e.* left, right and front/back plane) of each half wedge during the software penumbra wedge construction [Len05]. These planes are transformed in screen space by a VP and sent to the FP used for the wedge shading. For each fragment influenced by the wedge rendering, we access the Z-buffer of the scene in order to retrieve the z screen space coordinate of the potential receiver. Note that the x and y screen coordinates of the fragment is given by its position attribute. Thus, with respect to the previous plane equations, we simply perform a quick screen space test to define whether the surface point is outside the half wedge or not:

```
!!ARBfp1.0
OPTION NV_fragment_program2;

ATTRIB f_pos = fragment.position;
ATTRIB plane0 = fragment.attrib[0];     # outside planes in screen spaces
ATTRIB plane1 = fragment.attrib[1];
ATTRIB plane2 = fragment.attrib[2];
TEMP r0, r1;

# texture[0] == Z-buffer
SWZ        r0.xyw, fPos_sp, x, y, 0, 1; # r0 = fragment in screen space
TEX        r0.z,   fPos_sp, texture[0], RECT;
DP4.CC0    r1.x,   r0,      plane0;
DP4.CC0    r1.y,   r0,      plane1;
DP4.CC0    r1.z,   r0,      plane2;
RET (LT0.xyzz);
# [...]
END
```

For the surface points $\mathbf{p}$ that effectively lie into the rendered half wedge, we have to project its corresponding edge from $\mathbf{p}$ onto the light source. Firstly we clip the edge to a local "near plane" [Len05]. This is necessary for the silhouette edges that does not lie entirely between the fragment and the light source (figure 3.10). These edges are then projected and clipped onto the light source. Earlier implementations [AAM03, ADMAM03] avoid the use of a local near plane by clipping the edges in *three* dimensions. Nevertheless, by performing the clipping step after the edge projection we drastically reduce the computational cost.

3.3.3 The silhouette visibility buffer evaluation

The proposed blending heuristic is based on the evaluation of the percentage of occlusion and the bounding rectangle of the occluded region for a set of light parts. We thus simply replace the last step of the penumbra wedge shading by our per light part computation.

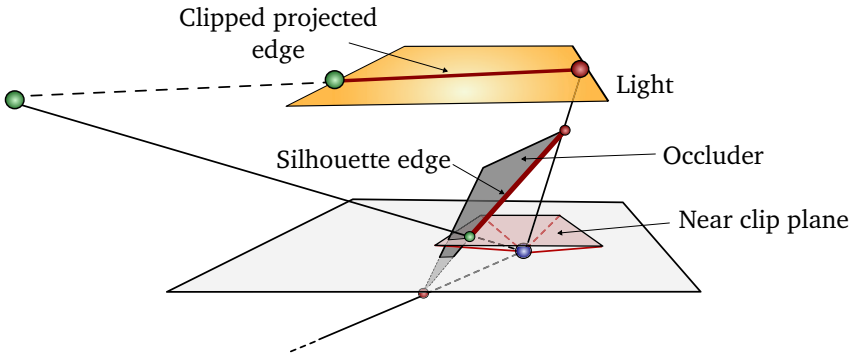


Figure 3.10: Local clipping of the silhouette edge.

We typically use *four* light parts. In order to simplify the explanation, in the following, we only refer to the upper right part evaluation. The other light parts are treated by symmetry.

In contrast to the penumbra wedge approach, we clip the edge against each light part rather than the light borders. We use a conditional test [BW04] to save computation time when the edge does not intersect the treated region. The occluded light percentage is computed as described in the hardware implementation of the penumbra wedges [AAM03]. It is finally added or subtracted into the SV-buffer with respect to the wedge type.

The bounding rectangle of the occluded light area is evaluated by the same FP. Our evaluation of the per light part bounding rectangle is *dependent* on the rendering order of the half wedge type. Indeed, the outer wedges has to be treated uppermost to the inner ones. This is not problematic since the wedge rendering is already batched with respect to their type: inner or outer (section 3.3.2). Thus, we first treat all the outer wedges in order to define an upper bounding rectangle of the occluded light region. The rendering of the inner wedges finally adjusts the bounding rectangle according to the maximum in X and Y of the occluding outer edges (figure 3.7).

In summary, for each visible surface point $\mathbf{p}$, the per light part evaluation of the SV-buffer requires the following parameters:

- the percentage of light that is occluded
- the top, bottom, left and right coordinates of the bounding rectangle
- the maximum in X and Y of the occluding edges

This results in *seven* parameters for a light part, *i.e. twenty eight* parameters for each visible surface point. In order to reduce the memory consumption, we use a precision of *one* byte per parameters. With such precision we can pack [BW04] the bounding rectangle

attributes into *one* 32-bit scalar. The *three* others parameters are packed in another 32-bit scalar. However, common render targets support at most $four$ 32-bit channels, *i.e.* Red, Green, Blue and Alpha (RGBA) format. We use the Multi Render Target (MRT) capability of the GPU to write the required *eight* 32-scalars into *two* single precision RGBA render targets.

3.3.4 Computing the penumbra blending

The blending of the SV-buffer with the V-buffer is straightforward. For each light part, this consists in unpacking the values saved into the buffers and then performing the blending operation presented in section 3.2.2. This blending step is performed by a specific FP. It is evaluated by simply drawing a full screen quad. According to the proposed algorithm 6 (line 9), we use a dynamic branching [BW04] to save computation time when there is no potential overlapping error.

We point out that the V-buffer uses the same encoding than the SV-buffer excepted for the X and Y parameters that are not saved into the V-buffer. Indeed, these outer edge parameters are not required since no bounding rectangle adjustment is performed on the V-buffer.

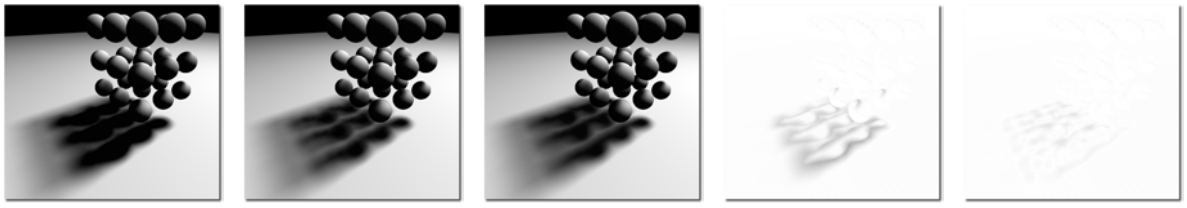
3.4 Results

In this section we analyze the performances of the penumbra wedge algorithm improved by our blending stage. First, we give the memory cost of our blending strategy. We finally analyze its penalty onto the performances.

3.4.1 Memory requirement

Due to the hardware limitations, we cannot regroup all the required render targets in *one* render context. Indeed, in addition of the depth/stencil buffer, the targeted GPU [NVi05] supports at most $four$ render targets. Our implementation relies on *two* render contexts. Each is encoded into a frame buffer object. The first one saves the color and the depth/stencil buffer. The latter stores the $four$ render targets require by the V-buffer and the SV-buffer, and shares the depth/stencil buffer with the first render context.

The SV-buffer requires *two* RGBA 32-bit buffers. The V-buffer needs *one* RGBA 32-bit render target to encode the bounding rectangles of each light part. In addition it uses *one* RGBA 8-bit buffer for the associated percentage of occlusion. Thus, with a



(a) PWedge (b) Our algorithm (c) Reference (d) (a) - (c) (e) (b) - (c)

Figure 3.11: A cube composed of 9×9 spheres. This scene illustrates the penumbra blending generated by different soft shadow algorithms: the penumbra wedges, our method and a reference image.

1024×1024 frame buffer resolution, the memory cost of the shadow buffers is:

$$1024 \times 1024 \times 4\,channels\,(3\,buffers \times 4\,bytes + 1\,buffer \times 1\,byte) = 52\,MB$$

Nevertheless, the targeted GPUs does not support render contexts composed of render targets with different encodings. As a consequence we have to use $four$ RGBA 32-bit buffers. With such format, the memory cost of the visibility render context reaches 64 MB.

3.4.2 Performances

The performances were measured on a Linux 64-bit workstation composed of *two* Opterons dual core 2Ghz with 4GB of memory. Our graphics system is based on *two* NVIDIA 7800GTX. We take benefit of the *two* GPUs by using the Scalable Link Interface (SLI) technology [NVi05] configured in the Alternate Frame Rendering mode, *i.e.* the frames are rendered alternatively on the *two* GPUs. We also bench our implementation with only *one* graphics card in order to outline the influence of the graphics system.

The test scene is composed of a collection of spheres that we organize in order to exhibit strong overlapping artifacts (figure 3.11). Each of them is discretized by 762 triangles. The impact of the geometric complexity is evaluated by increasing progressively the number of spheres. The figures 3.12 and 3.13 illustrates the results.

With very few triangles, the original penumbra wedge algorithm runs twice as fast as our approach. However, when we use more polygons the penumbra wedge performances decrease faster than our method. Indeed, we see that with $17,526$ triangles and the SLI enabled, evaluating our blending heuristic increase the penumbra wedge rendering cost by only 7%. This is due to the silhouette detection as well as the primitive extrusion that are both performed onto the CPU. This bottleneck can be exhibited by comparing the performances of the penumbra wedges with and without the SLI enabled. With only $3,048$ polygons, the SLI gain is already negligible. Our algorithm is more GPU intensive

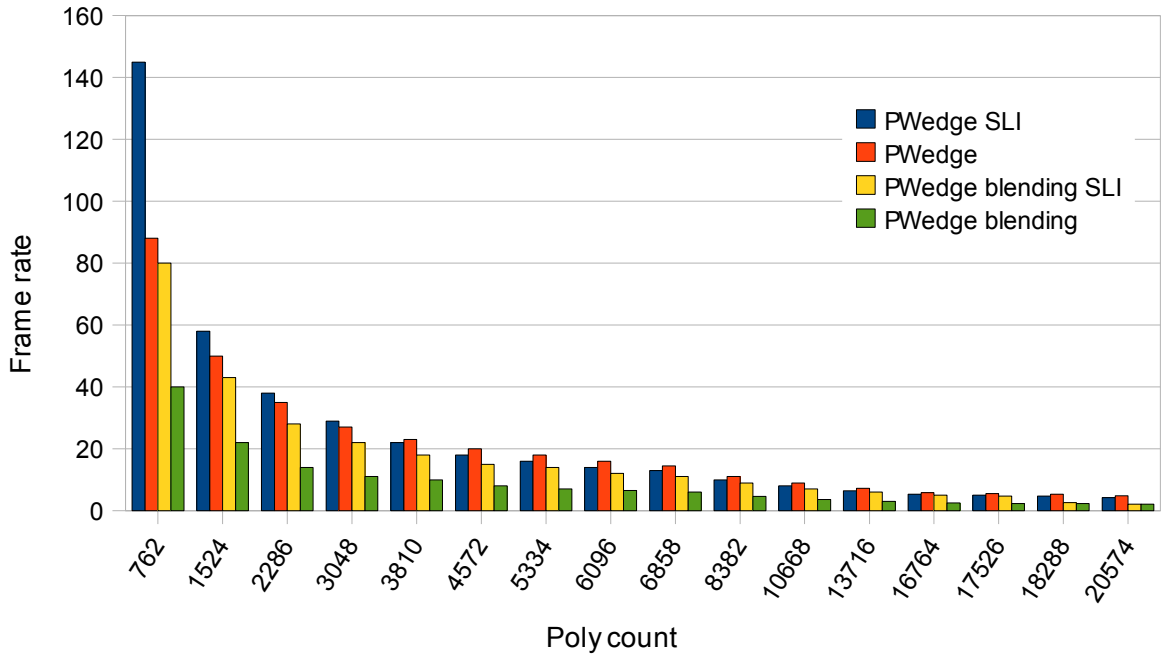


Figure 3.12: Frame rate comparison between the penumbra wedge algorithm and the proposed improvement.

and thus it takes a strong advantage of a SLI configuration. However, when we increase the number of polygons over 17,482 our algorithm becomes CPU limited and the SLI gain is quasi-null.

3.5 Discussion

The proposed algorithm drastically improves the overall shadow quality of the penumbra wedge approach (figure 3.11). It takes benefit of the robust penumbra wedge framework to limit the costly soft shadow evaluation to the penumbra regions. Nevertheless, despite convincing results (figure 3.14), we can observe several inherent limitations.

First, the obtained performances outline the main issue of object-based algorithms: their efficiency is drastically influenced by the geometric complexity of the environment. Following our initial implementation, the silhouette detection and the penumbra wedge extrusion is CPU intensive. Interactive performances are thus obtained only on simple scenes. Additional efforts has to be made in order to improve the efficiency of this step. One can note that more recent graphics hardware supports geometry programs. Thanks to this functionality we can expect better performances by performing the silhouette detection and the shadow volume/penumbra wedge generation directly onto the GPU.

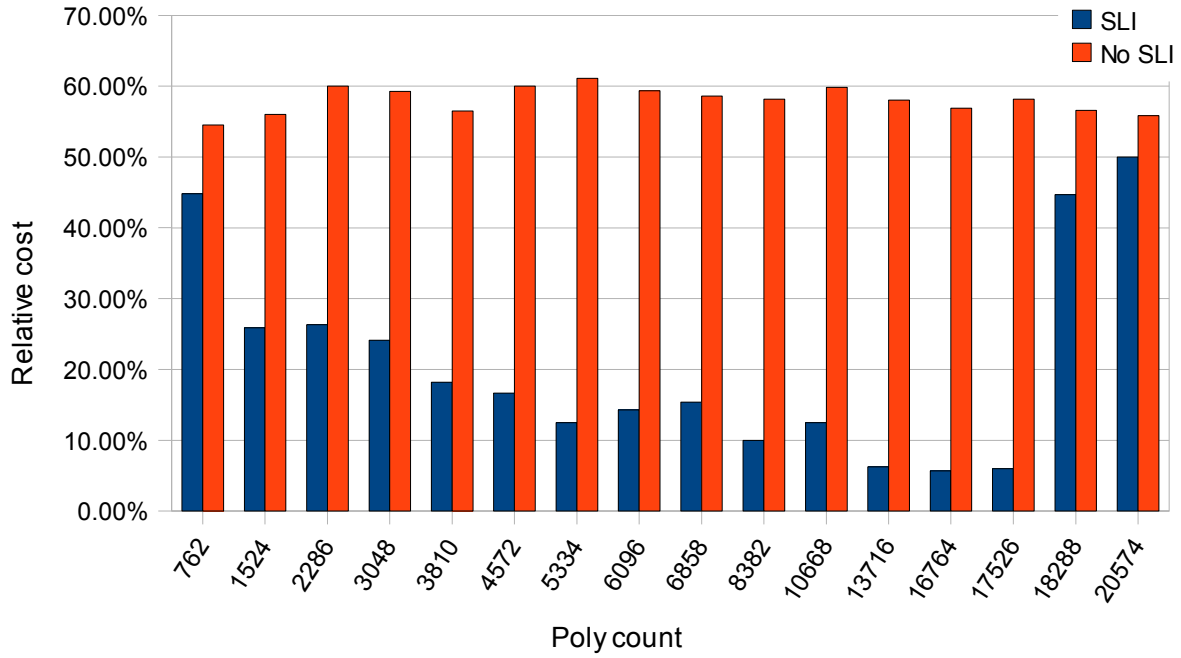


Figure 3.13: Relative cost of the penumbra wedge blending compare to the original penumbra wedge algorithm.

Even though our blending heuristic improves the resulting shadow quality, we cannot assume the correctness of the result. Our algorithm is based on an approximation of the occluded light area. This approximation is of course insufficient when targeting realistic shadows. We can improve the blending heuristic by using the center of gravity of the occluded light part. This new information would allow a better approximation of the position of the occluded region into the bounding rectangle. Nevertheless, the resulting shadows would still lies on an approximative heuristic that could not pretend to accurate solutions.

Finally, despite its intuitive use, computing a V_{coef} inevitably leads to wrong shadows. Indeed, the V_{coef} formulation is based on an approximation of the direct lighting. In fact, accurate direct shadow simulation should be evaluated according to the direct light transport problematic. Thus, in the following chapter, we will explicitly focus on the visibility problematic of the direct light transport in order to address the accuracy issues of a V_{coef}-based direct illumination.

3.A Infinite shadow volume extrusion

This `ARBvp1.0` vertex program performs the infinite extrusion of the shadow volume vertexes with respect to a point light.

Figure 3.14: Our algorithm applied on a scene composed of 6662 polygons. This scene is based on the models and materials of Half-life[2]. Objects and textures copyright Valve Corporation: used with permission.

```
!!ARBvp1.0
PARAM lpos = program.env[0];            # {light position.xyz, 0}
PARAM mvp[4] = {state.matrix.mvp};
ATTRIB vpos = vertex.position;
OUTPUT opos = result.position;

TEMP r0;

SUB    r0,        vpos,   lpos;
MAD    r0,        r0.w,   lpos,     r0;
DP4    opos.x,    r0,     mvp[0];
DP4    opos.y,    r0,     mvp[1];
DP4    opos.z,    r0,     mvp[2];
DP4    opos.w,    r0,     mvp[3];
END
```

3.B Infinite penumbra wedge construction

The following C++ listing computes the vertexes of an infinite penumbra wedge from a given silhouette edge with respect to a spherical light source. The class Vector3D and Vector4D

are internally defined. They overload the common per component vector operators as `+`, `-`, `*=` , *etc.* Furthermore, one can simply cast a `Vector4D` object into a `Vector3D` one by using the `Vector3D& Vector4D::to_Vector3D(void)` method. Finally, the constructor `Vector4D::Vector4D(const Vector3D& v, float w)` creates a `Vector4D` instance from the `Vector3D v` and the homogeneous value `w`.

```
class InfinitePWedge {
public:
    void build_from_silhouette_edge
      (const Vector3D& v0, const Vector3D& v1,
       const Vector3D& light_pos, float light_radius);

protected:
    // Associated silhouette edge
    Vector3D edge_v0;
    Vector3D edge_v1;
    // Homogeneous coordinates of the penumbra wedge vertexes
    Vector4D wedge_edge_v0, wedge_edge_v1;
    Vector4D wedge_front_v0, wedge_front_v1;
    Vector4D wedge_back_v0, wedge_back_v1;
    Vector4D wedge_center_v0, wedge_center_v1;
};

void
InfinitePWedge::build_from_silhouette_edge
  (const Vector3D& v0, const Vector3D& v1,
   const Vector3D& light_pos, float light_radius) {

    Vector3D light_to_v0, light_to_v1;
    Vector3D axis_x, axis_y;
    Vector3D tmp0, tmp1;
    float sqr_light_to_v0, sqr_light_to_v1;

    this->edge_v0 = v0;
    this->edge_v1 = v1;
    light_to_v0 = this->edge_v0 - light_pos;
    light_to_v1 = this->edge_v1 - light_pos;
    sqr_light_to_v0 = dot_product(light_to_v0, light_to_v0);
    sqr_light_to_v1 = dot_product(light_to_v1, light_to_v1);

    if(sqr_light_to_v0 < sqr_light_to_v1) {
        light_to_v1 *= sqrtf(sqr_light_to_v0 / sqr_light_to_v1);
        this->wedge_edge_v1 = Vector4D(light_to_v1 + light_pos, 1.f);
        this->wedge_edge_v0 = Vector4D(this->edge_v0, 1.f);
```

```
    } else {
        light_to_v0 *= sqrtf(sqr_light_to_v1 / sqr_light_to_v0);
        this->wedge_edge_v0 = Vector4D(light_to_v0 + light_pos, 1.f);
        this->wedge_edge_v1 = Vector4D(this->edge_v1, 1.f);
    }
    axis_x = (light_to_v1 - light_to_v0).normalize() * light_radius;
    axis_y = (cross_product(light_to_v1, axis_x)).normalize() * light_radius;

    tmp0 = this->wedge_edge_v0.to_Vector3D() - light_pos - axis_x;
    tmp1 = this->wedge_edge_v1.to_Vector3D() - light_pos + axis_x;

    this->wedge_front_v0 = Vector4D(tmp0 + axis_y, 0.f);
    this->wedge_front_v1 = Vector4D(tmp1 + axis_y, 0.f);
    this->wedge_back_v0 = Vector4D(tmp0 - axis_y, 0.f);
    this->wedge_back_v1 = Vector4D(tmp1 - axis_y, 0.f);
    this->wedge_center_v0 = Vector4D(tmp0, 0.f);
    this->wedge_center_v1 = Vector4D(tmp1, 0.f);
}
```

4

Accurate shadows by depth complexity sampling

We present a new object-based soft shadow algorithm that targets the robust simulation of the direct lighting. The proposed algorithm is based on a new efficient evaluation of the number of occluders between *two* points, *i.e.* the depth complexity. We use the depth complexity information in order to define which light samples are visible for each receiver (*i.e.* with a depth complexity equal to *zero*). Then, we either modulate the direct lighting or numerically simulate the direct illumination of the environment according to an user controllable trade off between quality and performances.

Our algorithm combines the efficiency of the real time penumbra wedge framework [AAM03] with the accuracy of the offline soft shadow volume approach [LAA+05, LLA06]. The soft shadow volumes generalize the penumbra wedges in order to compute accurate offline soft shadows for planar lights. This algorithm uses the penumbra wedge framework to define a list of edges that potentially affect the visibility of the planar light from the receiver point $\mathbf{p}$. For each $\mathbf{p}$, the occluding edges are then projected onto the light source as seen from $\mathbf{p}$. A set of edge rules are then used to evaluate the depth complexity for a set of light samples. Finally, to determine whether the samples with the lowest depth complexity are occluded, a single shadow ray is cast to one of them. In our algorithm, neither data structures storing silhouette edges for each $\mathbf{p}$, nor explicit edge rules for depth complexity computation, are necessary. In addition, our approach is not limited to planar area light sources and does not lie onto any ray tracer.

We start in section 4.1 with a description of our new efficient depth complexity evaluation. Then we discuss about the different sampling strategies that we used in order to reduce the variance of our depth complexity computation (section 4.2). Section 4.3 outlines how we use the depth complexity in order to compute soft shadows. Section 4.4 describes the GPU implementation of this new soft shadow algorithm. Finally, we present

results in section 5.4 and we end with a discussion.

4.1 Local depth complexity computation

This section details our streamed evaluation of the depth complexity function. In contrast to previous approaches [LAA+05, LLA06], the proposed algorithm incrementally computes the depth complexity without any explicit knowledge of the overall scene organization. In a first step, we compute the silhouette edges E from the entire environment as seen from a light l. Then, we initialize the depth complexity between $\mathbf{p}$ and a set of light samples from the pool of edges E (section 4.1.1). We finally update the depth complexity of the receivers lying in the penumbra region of the silhouette edges E (section 4.1.2).

4.1.1 Depth complexity initialization

The depth complexity between $\mathbf{p}$ and a light sample $\mathbf{s}$ was originally defined as the number of surfaces that a ray from $\mathbf{p}$ to $\mathbf{s}$ intersects. A depth complexity of n (n greater than *zero*) means that n surfaces occlude $\mathbf{s}$ as seen from $\mathbf{p}$. However, potential changes in the visibility function can occur only on the silhouette edges. It is therefore sufficient to track the occluding silhouette loops rather than the occluding surfaces. In consequence, we reformulate the depth complexity function as the number of silhouette loops occluding $\mathbf{s}$ from $\mathbf{p}$.

This reinterpretation addresses a limitation of previous algorithms. Indeed, as explained by Laine *et al.* [LAA+05], each silhouette edge generates a local change in the depth complexity function and the set of all silhouette edges represents its derivative. Integrating over the local changes results in integrating the derivative of the depth complexity function, and gives the depth complexity without the constant of integration. In order to define this constant, Laine *et al.* cast a shadow ray towards the light sample with the lowest depth complexity, thus limiting the performances on dynamic scenes. However, the shadow volume algorithm computes for each visible point $\mathbf{p}$ the number of occluding silhouette loops. Using the previous reformulation, this defines the constant in the integration of the depth complexity function derivative. As a result we can avoid the ray casting by initializing the constant of integration with the result of the shadow volume step. This is in fact quite similar to the initialization step of the penumbra wedge approach [AAM03].

4.1.2 Update of the depth complexity

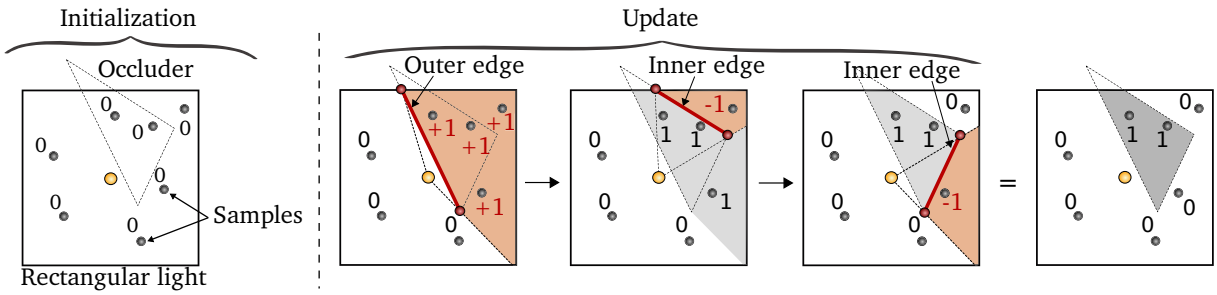


Figure 4.1: Update of the depth complexity of a set of light samples seen from a point **p**. Each occluding edge is projected from **p** onto the light source. The covered samples are then incremented or decremented according to the wedge type (outer or inner).

Algorithm 7 update_depth_complexity(wedges, light)

```
1:  for all half_wedge ∈ wedges do
2:    for all p in wedge do
3:      edge ⇐ half_wedge.silhouette_edge;
4:      edge^p ⇐ project(edge, light, p);
5:      edge^p_c ⇐ clip(edge^p, light);
6:      for i ⇐ 0 to NBR_LIGHT_SAMPLES − 1 do
7:        if p.sample[i] is covered by edge^p_c then
8:          if is_outer_wedge(half_wedge) then
9:            p.sample[i].depth_complexity + = 1;
10:         else
11:           p.sample[i].depth_complexity − = 1;
12:         end if
13:       end if
14:     end for
15:   end for
16: end for
```

The depth complexity integration for opaque meshes is performed as in algorithm 7. The depth complexity between a point **p** and a set of light samples is updated by processing the silhouette edges which projection from **p** overlaps the light source (line 3). Due to its linear nature, the integration can be performed separately for each projected edge and it is independent for each light sample (line 6). Thus, we update the depth complexity counter independently for each light sample as follows.

Following the penumbra wedge framework, the shadow volume quadrilateral splits each wedge into *two* parts: the inner part and the outer part. In both cases, the edge is projected onto the light source (line 4) to define the covered light samples (line 7). Their corresponding depth complexity counters are then incremented (line 9) or decremented (line 11) if the wedge is respectively outer or inner (figure 4.1). Since projected edges are clipped (line 5), silhouette edges crossing the light border are also naturally handled

(standard offline integrations require specific treatments here [LAA+05]).

4.1.3 The counter packing encoding

The depth complexity is evaluated with edges encoded as a streamed data set rather than a common static list. Instead of iterating over the list of edges to globally compute the depth complexity between a sample **s** and a point **p**, we progressively update it for each edge that potentially affects the visibility. This avoids the use of a costly pre-computed static space partitioning data structure referencing silhouette edges [LAA+05, LLA06]. For each visible point **p** we maintain a set of depth complexity counters corresponding to the set of light samples. Therefore the number of available counters must be equal to the number of light samples, and the precision available has to be sufficient to store the maximum number of occluders.

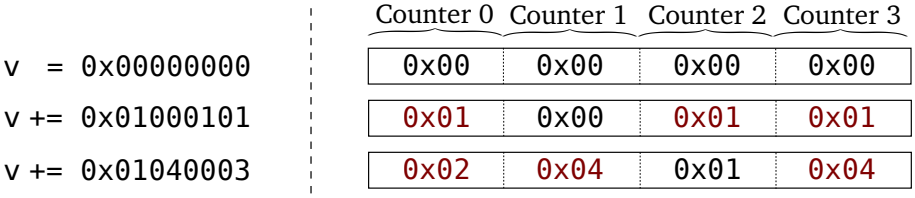


Figure 4.2: Simultaneous update of $four$ counters $\in$ [0..255] packed in a $four$-byte value. Left: operations on the packed representation; right: resulting value of the packed counters.

However, common renderers can use render buffers with at most $four$ values (red, green, blue and alpha). In order to provide a sufficient number of counters being both efficiently stored and updated, we pack several counters into a single value as illustrated in the following example. Considering a value v encoded in a 32-bit unsigned integer. The value v can store a single counter ranging from 0 to $2^{32} - 1$ or, for instance, $four$ values ranging from 0 to 255 in $four$ consecutive bytes. This packed representation is particularly well suited for a vectorized incrementation/decrementation. Indeed, assuming that the resulting values do not overflow the domain of the counters, the addition of a $four$-byte value with correctly positioned bits performs a vectorized update of the counters as shown in figure 4.2.

We point out that the proposed counter packing representation is designed to address the format limitation of current application programming interfaces [SA06, Mic08]. Software renderers does not care about such issue since they are not limited to RGBA render targets.

4.1.4 Advantages and drawbacks

The counter packing allows an efficient update of the depth complexity counters and increases their quantity. However, only unsigned integers can be used. Even though depth complexity cannot be negative, inner wedges can affect the depth complexity counters before the outer wedges, resulting in temporary negative values. Thus, the depth complexity update must be split in *two* batches, where outer wedges are treated before the inner ones.

(a) Standard (b) Accurate (c) RT reference

Figure 4.3: Comparison of silhouette edge determination methods for soft shadows generation. Detecting the silhouette edges from the light center (a) leads to an under-estimated penumbra while considering the whole surfacic light (b) avoids this single light sample artifact and produces shadows identical to the ray-traced reference [mi] (c).

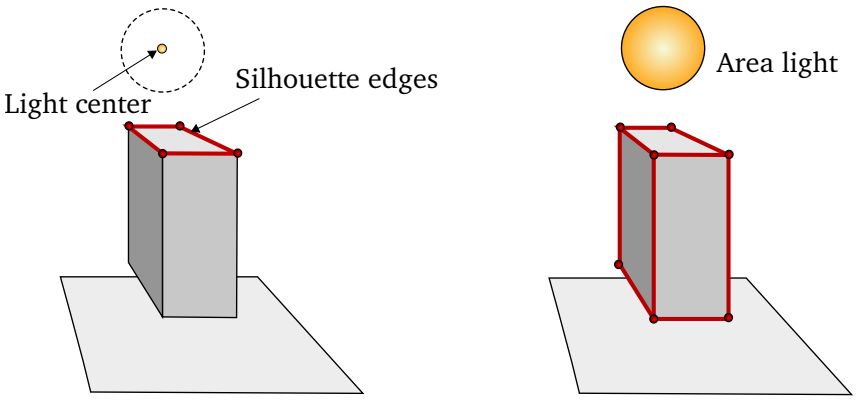


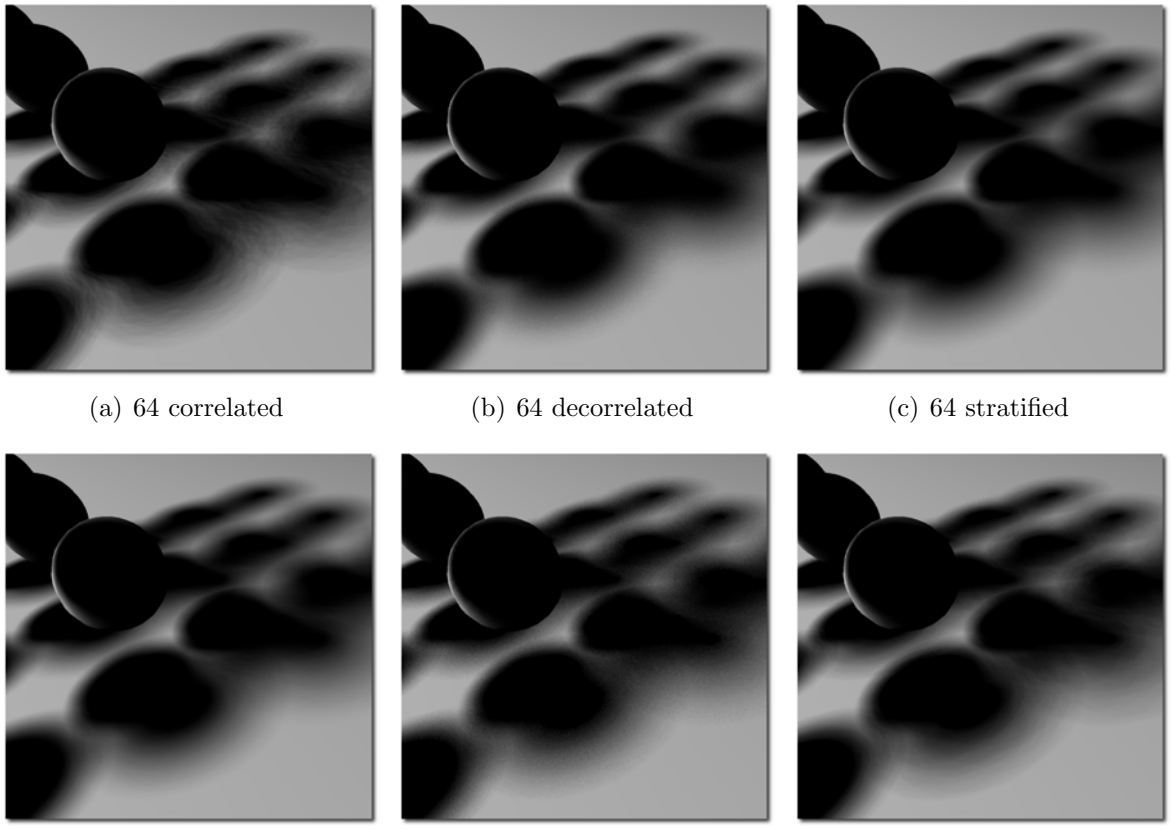
Figure 4.4: Left: detection of the silhouette edges from the light center. Right: detection of the silhouette edges with respect to the whole area light.

Standard silhouette detection from the center of the light leads to single light sample artifact (figure 4.3(a)). We avoid this using a more accurate approach (figure 4.3(b)). Considering the planes of two triangles connected to an edge, the edge is a silhouette if a part of the light source is in the positive side of one plane and in the negative side of

the other [LAA+05]. Even though this detection is more accurate, it is also more time consuming since it generates several silhouette loops where only one is produced by a standard detection (figure 4.4).

Finally, the presented algorithm generalizes the penumbra wedge approach [AAM03] and it can be easily integrated into the penumbra wedge framework.

4.2 Light sampling strategy



(a) 64 correlated (b) 64 decorrelated (c) 64 stratified

(d) 64 interleaved (e) 16 stratified (f) 16 interleaved

Figure 4.5: Comparison of the soft shadow quality according to the number of samples and the sampling strategy. Shadows are generated with either 64 (a, b, c, d) or 16 (e, f) samples using a correlated (a), decorrelated (b), decorrelated and stratified (c, e) or interleaved sampling pattern (d, f).

The quality of the reconstructed visibility function between a point $\mathbf{p}$ and an extended light source depends on both the number of samples and their distribution over the light (figure 5.3). In this section we detail our sampling strategy and propose a method to

dynamically adjust the balance between the precision of the depth complexity and the number of samples.

4.2.1 Sample distribution

In order to reduce the variance of the depth complexity function evaluation for $\mathbf{p}$, it is necessary to distribute random samples onto the light source using an adapted probability distribution function [PH04]. However, a fixed sampling pattern for all $\mathbf{p}$ may become visible (figure 4.5(a)), even with several light samples. In order to alleviate this drawback, one can vary the set of samples for each $\mathbf{p}$ by randomly rotating the initial sampling pattern (figure 4.5(b)). In addition to this deccorelation, we reduce the variance by using a stratified sampling strategy. (figures 4.5(c) and 4.5(e)).

4.2.2 Interleaved sampling

Correlated samples generate visible sampling patterns and their deccorelation generates noise. Interleaved sampling [KH01] takes advantage of both distributions. The main idea is that for neighbor points, a measured signal may produce the same result. Thus, the signal for nearby points is sampled with an interleaved sampling distribution and finally merged into a single sampling pattern. Segovia *et al.* [SIMP06] proposed a real time interleaved sampling framework for deferred renderers. To conserve the coherence between neighbor pixels, they split the rendered image into sub-buffers containing the pixels that use the same distribution of samples. After sampling the desired signal, they gather the sub-buffers and filter the result according to the normal and depth discontinuities.

This real time interleaved sampling formulation can be efficiently used in our framework (figure 4.5(d) and 4.5(f)). However, our object-based algorithm works in a purely forward manner. Thus, in order to avoid multiple rendering of shadow volumes and wedges, we use neither the image splitting nor the gather step. In spite of the loss of near points coherence, no drop of performance occurs since no coherent memory access is required (section 4.4).

4.2.3 Adaptive distribution

The maximum depth complexity of the rendered scene depends on the scene complexity and the viewpoint. While rendering a plain leads to a globally low depth complexity, a forest rendering generates many occlusions. Given a fixed amount of available memory for a visible point $\mathbf{p}$, we propose a simple way to dynamically adjust the precision and the number of packed depth complexity counters.

The maximum depth complexity between the light samples and $\mathbf{p}$ is equal to the maximum number of occluding silhouette loops. Thus, in a first step, we track the maximum integration constant m_c resulting from the depth complexity initialization of the current visible points. Then we use this value to define the counter packing encoding for the current frame. As an illustration, consider the previous example where $four$ bytes are available for each visible point. With m_c in $[256, 65535]$, *two* counters of *two*-byte ($\in [0, 65535]$) can be packed without overflow, while m_c in $[128, 255]$ allows the use of $four$ counters of *one* byte ($\in [0, 255]$).

4.3 Depth complexity for shadow computation

This section describes the computation of soft shadows by using the depth complexity information. We first outline how an artifact free V_{coef} can be derived from the depth complexity (section 4.3.1). Targeting the robust simulation of direct shadows, we then show how we use the depth complexity to numerically solve the direct lighting integral (section 4.3.2). Finally, we describe an extension that handles the soft shadows cast by occluders having a specific transmittance property (section 4.3.3).

4.3.1 From depth complexity to visibility coefficient

The *depth complexity function* returns the number of occluders between two points. Considering a light sample $\mathbf{s}$ and a surface point $\mathbf{p}$, $\mathbf{s}$ is visible from $\mathbf{p}$ if its depth complexity from $\mathbf{p}$ is equal to *zero*. Thus, a V_{coef} for $\mathbf{p}$ can be simply computed from a set of N samples uniformly distributed over a light l as:

$$V_{coef}\left(l \leftrightarrow \mathbf{p}\right) = 1 - \frac{1}{N}\sum_{i=0}^{N-1} Sat\left(D\left(\mathbf{s}_{i,l} \leftrightarrow \mathbf{p}\right)\right) \tag{4.1}$$

where $D\left(\mathbf{s}_{i,l} \leftrightarrow \mathbf{p}\right)$ returns the depth complexity $\in \mathbb{N}$ between the i^{th} light sample $\mathbf{s}_{i,l}$ and $\mathbf{p}$. $Sat\left(x\right)$ is a saturating function that clamps x into $[0, 1]$.

Area light sources such as a fire or a TV screen have a spatially varying luminance. Using depth complexity, such textured light sources can be simply taken into account during the V_{coef} computation by multiplying the visibility query (*i.e.* the saturated depth complexity) by the sample luminance L.

$$V^{rgb}_{coef}\left(l \leftrightarrow \mathbf{p}\right) = \frac{1}{N}\sum_{i=0}^{N-1} L^{rgb}_{i,l} - L^{rgb}_{i,l} \cdot Sat\left(D\left(\mathbf{s}_{i,l} \leftrightarrow \mathbf{p},\right)\right) \tag{4.2}$$

The sample luminance can be encoded with an arbitrary color space. However, since the red, green, blue (RGB) representation is well suited for common rendering engines, we define sample luminance and V_{coef} as RGB values $\in [0,1]^3$. In addition, sample luminance can vary over time. We take into account these animated textured lights by adding time dependence to the luminance.

4.3.2 Numerical integration of the direct lighting

Previously, the depth complexity function is used to compute a *visibility coefficient* of a point $\mathbf{p}$. The V_{coef} is then used to modulate the direct illumination of $\mathbf{p}$ computed from the center of the light. Hence, considering a set of j lights and its center position x, the lighting according to a given point of view x' is computed as:

$$\begin{aligned} L\left(\mathbf{p} \rightarrow xi'\right) &= L_e\left(\mathbf{p} \rightarrow x'\right) + I\left(\mathbf{p} \rightarrow x'\right) \\ &+ \sum_j L_e\left(x \rightarrow \mathbf{p}\right) f_s\left(x \rightarrow \mathbf{p} \rightarrow x'\right) V_{coef}\left(j \leftrightarrow \mathbf{p}\right) \frac{cos\theta_i}{||x-\mathbf{p}||^2} \end{aligned} \tag{4.3}$$

Even though the visibility is evaluated for an area light source, the direct lighting is still computed using point lights. In order to compute an accurate *direct* illumination for extended light sources, we have to solve the direct lighting part of the rendering equation:

$$\begin{aligned} L\left(\mathbf{p} \rightarrow x'\right) &= L_e\left(\mathbf{p} \rightarrow x'\right) + I\left(\mathbf{p} \rightarrow x'\right) \\ &+ \int_{\mathcal{M}} L_e\left(x \rightarrow \mathbf{p}\right) f_s\left(x \rightarrow \mathbf{p} \rightarrow x'\right) V\left(x \leftrightarrow \mathbf{p}\right) \frac{cos\theta_o cos\theta_i}{||x-\mathbf{p}||^2} dA\left(x\right) \end{aligned} \tag{4.4}$$

Note that the binary visibility $V\left(x \leftrightarrow \mathbf{p}\right)$ can be very simply evaluated from the depth complexity function as:

$$V\left(x \leftrightarrow \mathbf{p}\right) = H\left(D\left(x \leftrightarrow \mathbf{p}\right)\right) \tag{4.5}$$

where $H(x)$ is the Heaviside function that returns *one* if x is strictly positive and *zero* otherwise. Thus, we can numerically solve the equation 4.4 using a set of N samples uniformly distributed over each extended light source, and their corresponding depth complexity:

$$\begin{aligned} L\left(\mathbf{p} \rightarrow x'\right) &= L_e\left(\mathbf{p} \rightarrow x'\right) + I\left(\mathbf{p} \rightarrow x'\right) \\ &+ \sum_j \frac{1}{P_j N} \sum_{k=0}^{N-1} L_e\left(x_{k,j} \rightarrow \mathbf{p}\right) f_s\left(x_{k,j} \rightarrow \mathbf{p} \rightarrow x'\right) H\left(D\left(x_{k,j} \leftrightarrow \mathbf{p}\right)\right) \frac{cos\theta_o cos\theta_i}{||x-\mathbf{p}||^2} \end{aligned} \tag{4.6}$$

where $x_{k,j}$ is the k^{th} sample of the light j and P_j the uniform probability for the samples of the j^{th} light source. This equation 4.6 naturally handles textured lights since the emitted

radiance $L_e(x_{k,j} \rightarrow \mathbf{p})$ from the light samples is explicitly defined.

4.3.3 Handling semi opaque occluders

Our depth complexity algorithm is designed to generate shadows cast by fully opaque occluders. In this section, we propose an extension that handles the transmittance property of the occluders.

The transmittance integration

Targeting meshes with a transmittance property, we separate opaque objects from semi transparent meshes. The shadows from opaque objects are evaluated with our depth complexity sampling algorithm (section 4.1). For the visible light samples (*i.e.* with a depth complexity equal to *zero*), we then evaluate the amount of light that effectively reaches the receiver with respect to the transmittance of the semi transparent occluders. This computation is performed as follows.

In a first step, for each visible receiver $\mathbf{p}$, we evaluate an initial percentage of transmitted light (algorithm 8). To do that, we simply extends the Z-fail shadow volume strategy in order to initialize both the transmittance and the depth complexity between the light samples and the receivers. Indeed, we evaluate the constant of the depth complexity integration (lines 9 and 15) as well as the percentage of radiance that is transmitted (lines 12 and 19), with respect to the orientation of the rendered primitives (lines 7).

In a second step, we use the penumbra wedge framework to update the initial transmittance (algorithm 9). This update step is very close to the algorithm 7. The main difference is that we perform additional treatments to handle semi transparent occluders (lines 12 and 19). We modulate the transmitted light percentage (lines 14 and 21) according to the type of the wedges (lines 9 and 16). The figure 4.6 summarizes this transmittance integration process.

The transmitted direct lighting

Following the previous algorithm, we modulate the direct illumination of the scene by using a V_{coef} evaluated from the depth complexity D and the transmittance T:

$$V_{coef}^{rgb}(l \leftrightarrow \mathbf{p}) = \frac{1}{N} \sum_{i=0}^{N-1} T(\mathbf{s}_{i,l} \leftrightarrow \mathbf{p}) \left(L_{i,l}^{rgb} - L_{i,l}^{rgb} \cdot Sat(D(\mathbf{s}_{i,l} \leftrightarrow \mathbf{p},)) \right) \tag{4.7}$$

One the other hand, when targeting accurate direct lighting, we numerically solve the direct illumination formulation as follow:

Algorithm 8 initialization(sh_volumes, camera, Z-buffer, stencil-buffer, trans-buffer)

```
 1: clear(stencil-buffer, 0);
 2: clear(trans-buffer, 1.f);
 3: for all triangle ∈ sh_volumes do
 4:    receivers ⇐ Z-fail(Z-buffer, triangle);
 5:    mesh ⇐ get_associated_mesh(triangle);
 6:    for all p ∈ receivers do
 7:       if is_back_facing(triangle, camera) then
 8:          if is_opaque(mesh) then
 9:             stencil-buffer[p] + = 1;
10:          else
11:             transmittance ⇐ get_transmitted_percentage(mesh);
12:             trans-buffer[p] ∗ = transmittance;
13:          end if
14:       else
15:          if is_opaque(mesh) then
16:             stencil-buffer[p] − = 1;
17:          else
18:             transmittance ⇐ get_transmitted_percentage(mesh);
19:             trans-buffer[p] / = transmittance;
20:          end if
21:       end if
22:    end for
23: end for
```

Algorithm 9 update_DC_and_transmitted_light(wedges, light)

```
1: for all half_wedge ∈ wedges do
2:   for all p in wedge do
3:     edge ⇐ half_wedge.silhouette_edge;
4:     edge^p ⇐ project(edge, light, p);
5:     edge^p_c ⇐ clip(edge^p, light);
6:     mesh ⇐ get_associated_mesh(half_wedge);
7:     for i ⇐ 0 to NBR_LIGHT_SAMPLES − 1 do
8:       if p.sample[i] is covered by edge^p_c then
9:         if is_outer_wedge(half_wedge) then
10:          if is_opaque(mesh) then
11:            p.sample[i].depth_complexity + = 1;
12:          else
13:            transmittance ⇐ get_transmitted_percentage(mesh);
14:            p.sample[i].transmittance ∗ = transmittance;
15:          end if
16:        else
17:          if is_opaque(mesh) then
18:            p.sample[i].depth_complexity − = 1;
19:          else
20:            transmittance ⇐ get_transmitted_percentage(mesh);
21:            p.sample[i].transmittance / = transmittance;
22:          end if
23:        end if
24:      end if
25:    end for
26:  end for
27: end for
```

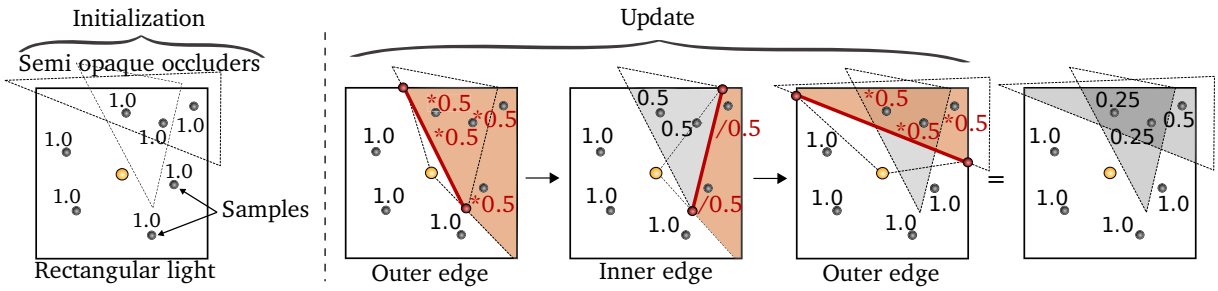


Figure 4.6: Transmittance update for a set of light samples as seen from a receiver $\mathbf{p}$ and occluded by *two* triangles with a transmittance factor equal to 0.5. Each occluding edge is projected from $\mathbf{p}$ onto the light source. The covered samples are then modulated according to the wedge type (outer or inner).

$$
\begin{aligned}
L\left(\mathbf{p} \rightarrow x'\right) &= L_e\left(\mathbf{p} \rightarrow x'\right) + I\left(\mathbf{p} \rightarrow x'\right) \\
&+ \sum_j \frac{1}{P_j N} \sum_{k=0}^{N-1} \Bigg[L_e\left(x_{k,j} \rightarrow \mathbf{p}\right) f_s\left(x_{k,j} \rightarrow \mathbf{p} \rightarrow x'\right) \\
&\quad T\left(x_{k,j} \leftrightarrow \mathbf{p}\right) H\left(D\left(x_{k,j} \leftrightarrow \mathbf{p}\right)\right) \frac{cos\theta_o cos\theta_i}{||x - \mathbf{p}||^2} \Bigg]
\end{aligned} \tag{4.8}
$$

In both situations, the binary visibility between the light samples and the receivers is retrieved with respect to the depth complexity. The radiance that effectively reaches the surface points is then evaluated from the transmitted light percentage of the visible light sample.

Analysis

The proposed transmittance evaluation is performed according to primitives extruded from the silhouette edges; in other words, no explicit access to the occluding surfaces is done. Thus, we keep the desirable independence property of the penumbra wedge framework. However, this precludes the use of surfaces with spatially varying transmittance.

The transmitted light percentage cannot be evaluated with integer operations:. It needs floating point capabilities. Even though GPU-based renderers support the floating point representation, they can store at most $four$ channels per pixel. The counter packing representation addresses this limitation for the depth complexity integration by packing several counters in one integer value. Nevertheless, this encoding cannot be used for the transmittance evaluation due to its limitation to integer operations.

We propose an alternative transmittance evaluation that takes benefit of the counter

packing formating. We replace the depth complexity function $D(\mathbf{s}_{i,l} \leftrightarrow \mathbf{p})$ (equation 4.2) by a light attenuation function $Dr(\mathbf{s}_{i,l} \leftrightarrow \mathbf{p})$ computed by *summing* the occluders' percentage of opaqueness. We approximate the floating point values by discretizing and mapping the transmittance domain into integers. This limits the precision of both counters and opaqueness but allows the use of the counter packing formatting for the transmittance evaluation. Functions D and Dr are equal for opaque occluders and in fact we do not differentiate, instead denoting D and Dr as a real depth complexity function.

We point out that this alternative is an approximation producing accurate results only when a single semi opaque surface occludes $\mathbf{p}$. Indeed, the light contribution is derived from the sum of opaqueness factors while it should rather be modulated by the transmittance of *all* occluding surfaces. Nevertheless, this algorithm is designed in order to address current graphics API limitations. For software renderers [SCS+08], our original approach can be naturally used.

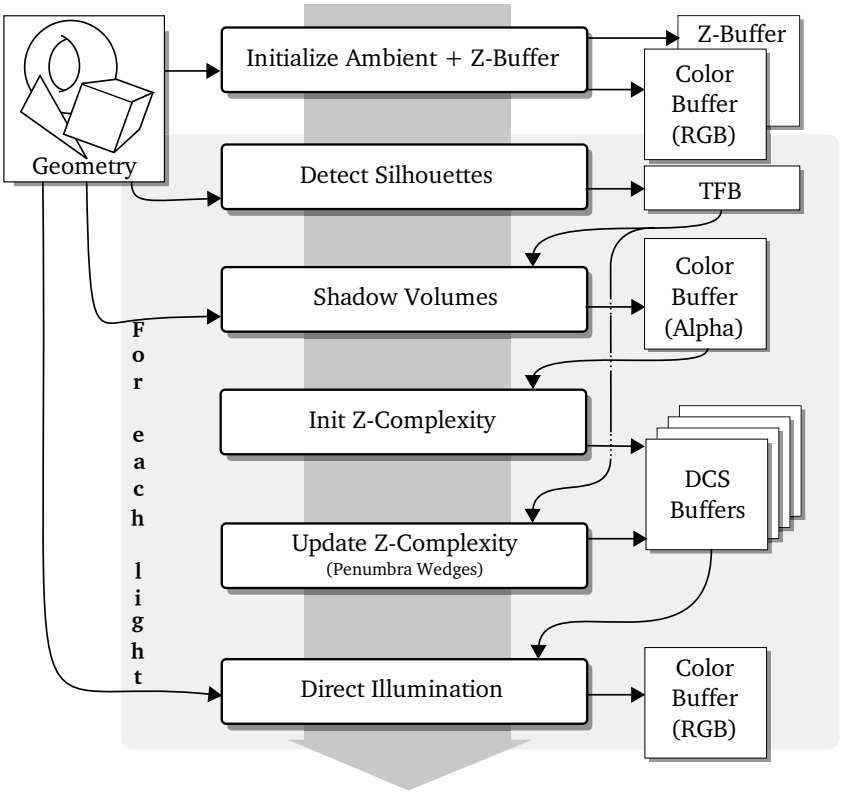


Figure 4.7: Overview of our algorithm for GPU implementation of the proposed depth complexity evaluation. All computations are performed on the GPU without transferring data to main memory.

4.4 Implementation

In this section, we present a GPU-intensive implementation of our algorithm. The proposed rendering algorithm (algorithm 10) is developed for graphics hardware that supports fragment programs (FP), vertex programs (VP) and geometry programs (GP). Figure 4.7 summarizes the GPU implementation and the render context organization, which we elaborate in the following explanations.

Algorithm 10 render_scene(world, view)

```
Require: Pre-computed samples pattern
  set_up_camera(view);
  (color-buffer, Z-buffer) ⇐ draw_ambient_lighting(world);
  if Interleaved Sampling then
    init_discontinuity_buffer();
  end if
  for all light ∈ world.lights do
    clear_shadow_buffers();
    edges ⇐ detect_silhouette_edges(light, world);
    init_depth_complexity(edges);
    if adaptive_sampling then
      define_max_depth_complexity();
    end if
    wedges_o ⇐ build_outer_wedges(edges);
    wedges_i ⇐ build_inner_wedges(edges);
    DC-buffer ⇐ update_depth_complexity(wedges_o, wedges_i);
    if V-buffer is used then
      V-buffer ⇐ vcoef_from_depth_complexity(DC-buffer);
      if interleaved_sampling then
        filter_vbuffer();
      end if
      color-buffer+ =draw_modulated_direct(light, world, V-buffer);
    else
      if interleaved_sampling then
        color-buffer+ =filter(direct_illumination(light, world, DC-buffer));
      else
        color-buffer+ =direct_illumination(light, world, DC-buffer);
      end if
    end if
  end for
```

4.4.1 Sample distribution

Depending on the sampling strategy and light type we pre-compute, and store into textures, the corresponding 2D sample distribution (we assume that omni-directional light

sources can be parametrized in 2D). In order to limit the memory requirements and the number of texture accesses, we pack several 2D sample positions per texel. Since the sample positions are computed in the normalized texture space, a precision of *one* byte per coordinate does not introduce a significant error. This precision allows us to encode *two* sample positions in a texture channel of $four$-byte. Hence, using $four$ channels (*i.e.* a RGBA texture), we obtain *eight* sample positions in *one* texture fetch. Note that textured lights require the luminance of each sample. Thus, in addition to their packed position, we pre-compute for these lights an $N \times M$ 24-bit RGB texture that stores the N sample luminances for M sets of correlated sample patterns.

4.4.2 Soft shadow volume framework

Silhouettes detection

In order to avoid the CPU bottleneck of object-based shadow algorithms, we perform the silhouette detection onto the GPU using a specific GP. The detected silhouettes are stored on the GPU in a *Transform Feedback Buffer* (TFB) [SA06]. Since writing to the TFB is asynchronous, CPU computations such as scissor rectangle definition or depth bounds evaluation [Len05] are performed in parallel. Note that according to the desired quality and performance trade off, the silhouette determination can be computed with either accurate or standard detection. The accurate detection is obviously more time consuming while it is required only in specific situations. Thus, we prefer the common silhouette detection, leaving the accurate one to unbiased solutions. Even though it is easy to exacerbate the differences between the *two* methods, using standard detection rarely leads to distinguishable visual issues in common environments.

Shadow volumes

Silhouette edges are then used in the shadow volume pass to define the initial value of the depth complexity function. We perform silhouette edge extrusion with a GP. In order to avoid a costly access to the stencil buffer, the Z-fail stencil test [Car00] is performed in a single pass in a simple FP. The resulting value is then cumulatively blended in the alpha channel of the color buffer:

```
# face.x = fragment is front-facing ? 1 : -1
ATTRIB face = fragment.facing;
ATTRIB fPos = fragment.position;
TEMP r0;

# texture[0] == zBuffer
TEX    r0.x, fPos,     texture[0], RECT;
```

```
SGE    r0.x, fPos.z,   r0.x;
MUL    result.color.w, r0.x, -face.x;
```

For a robust Z-fail stencil update, the shadow volumes have to be capped. Thus, we use a specific GP to compute the shadow volumes capping in an additional geometric pass.

Wedges

In order to reduce memory consumption, wedges are robustly extruded to infinity during the depth complexity evaluation, rather than being pre-computed and stored. We perform this construction with a GP. However, the performance of a GP is influenced by the number of generated data. Thus, the number of required vertexes is minimized using a single triangle strip per wedge and only *ten* vertexes per half-wedge. Figure 4.8 illustrates this generation of wedges. Also, common fill-rate optimizations and fragment rejection are used during the wedge rendering (scissor test, depth bound tests, *etc.*)

Figure 4.8: Triangle strip used for the half-wedge generation. This strip implicitly defines coherent normals for each face. Therefore, common optimizations based on the face orientations can be used [Len05].

4.4.3 The depth complexity sampling step

Counter packing representation

Since reading and writing in the same buffer is prohibited, we use the blending operations of the GPU to perform the counter updates. Unfortunately, even though the targeted GPUs support integers, their blending is not yet supported. This avoids a naive GPU implementation of the depth complexity update. On the one hand, the counter packing corresponds to a base decomposition where each base factor encodes the value of a counter with a precision up to the base-1 *e.g. two* counters c_0 and $c_1 \in [0 \cdot\cdot 255]$ packed in a *two*-byte value k is decomposed in base 256 as $k = c_0 * 256^0 + c_1 * 256^1$. On the other hand the simple precision floating point values are supported for both buffer format and blending

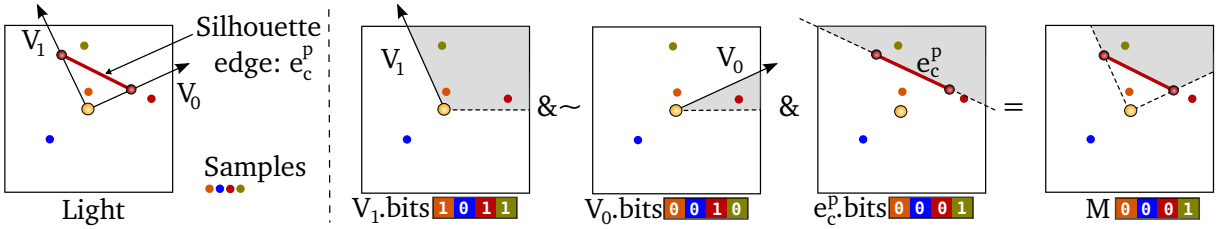


Figure 4.9: Determination of the samples covered by the projected clipped edge e_c^p. A cube map encodes the covered samples from the origin to v_x ($x \in \{0,1\}$) while a 2D texture stores the samples covered by the e_c^p line. The effective covered samples are then simply defined by a logical combination of the fetched bitfields.

operations. With this representation, one can count up to $2^{24}-1$ without missing an integer value. This value can be expressed in the following base decomposition:

$$2^{24}-1 = 255 * \left(256^0 + 256^1 + 256^2\right) \quad (4.9)$$
$$= 63 * \left(64^0 + 64^1 + 64^2 + 64^3\right) \quad (4.10)$$
$$= 15 * \left(16^0 + 16^1 + 16^2 + 16^3 + 16^4 + 16^5\right) \quad (4.11)$$
$$= 7 * \left(8^0 + 8^1 + 8^2 + 8^3 + 8^4 + 8^5 + 8^6 + 8^7\right) \quad (4.12)$$

Thus, we use the floating point representation and the base decompositions exposed in equations 4.9, 4.10, 4.11 and 4.12, to pack the depth complexity counters.

Depth complexity initialization

In order to simplify the explanations, we present the depth complexity initialization with a fixed Base 64 (B64) counter packing representation (equation 4.10). However, its generalization to the other encodings is straightforward. A B64 counter packing provides *sixteen* depth complexity counters per 128-bits RGBA buffer. We then use *Multi Render Target* (MRT) to increase the number of counters. Even tough the targeted GPUs support up to *eight* MRT, we limit the memory bandwidth and its consumption by using at most *four* MRTs (*i.e.* 64 counters). The depth complexity initialization of the 64 counters is then very simply performed via an FP as:

```
ATTRIB fPos = fragment.position;
TEMP r0;

# texture[0].w == stencil value
# 266305 == 64^0 + 64^1 + 64^2 + 64^3
TEX    r0.w, fPos, texture[0], RECT;
```

```
MUL    r0.w, r0.w, 266305;
MOV    result.color[0], r0.w;
MOV    result.color[1], r0.w;
MOV    result.color[2], r0.w;
MOV    result.color[3], r0.w;
```

(a) (b) (c)

Figure 4.10: (a) Illustration of the artifacts introduced by a discretization of the back projected edges and the corresponding covered samples in a 1024×1024 4D texture (16 MB). (b) Our discretization using a $64 \times 64 \times 6$ cube map and a 512×512 2D texture (4.375 MB). (c) Determination of the covered samples without discretization.

Depth complexity update

One of the main challenge we face while updating the depth complexity is efficiently determining samples covered by the projected clipped edge. Despite having access to *eight* sample positions in one texture fetch, a naive search leads to a complexity in $O(N)$ where N is the number of samples. We therefore propose an efficient discrete approach to find the covered samples. A pre-computed 4D texture [AAM03] uses large amount of memory and produces discretization artifacts (figure 4.10(a)), while the Hough transform, as in [ED07], requires that we either approximate, or pre-compute, the unsupported arc-cosine function. This leads to either additional texture fetches or heavy computation. Hence, we propose a new discrete representation (figures 4.9 and 4.17(b)).

First, covered samples lying in a sector defined by the origin and vector v_x ($x \in \{0, 1\}$) are encoded in a bit field and stored in a cube map. Then, a 2D texture indexed by the orthogonal projection of the light center onto the line e_c^p stores the bit field of samples covered by this line. The final bit mask M is then simply defined as $M = v_1.bits \ \& \ (\sim v_0.bits) \ \& \ e_c^p.bits$. Finally, we iterate through the bits of M to update the corresponding counters. Despite the use of integer operations, which are less efficient than

floating point, this method requires only *three* texture fetches and a logical combination to define all the covered samples.

Note that this discretization is robust even though the light center passes through the line e_c^p. Indeed, in this case, the effective covered samples are just defined by the cube map and so the 2D texture has to store a field of bits set at *one*.

4.4.4 Evaluating the direct illumination

Finally, the depth complexities associated with a fragment are used to solve either the equation 4.6 or to compute the corresponding V_{coef} (equation 4.2). The first approach merges the direct illumination computation with the visibility queries while the second performs an additional step to compute the V_{coef} used to modulate the direct lighting. The resulting lighting contribution is then cumulatively blended. Note that in both cases, the interleaved sampling strategy requires an additional filtering step combining the interleaved sampling patterns [SIMP06].

4.5 Results

In this section we analyze both the memory consumption and the performances of our implementation onto graphics hardware of the depth complexity algorithm.

4.5.1 Memory cost

Table 4.1 shows the memory used by data structures in our implementation. We do not apply any texture compression in order to present significant memory cost without the influence of a specific/subjective compression method. In addition, we reserve 5MB of memory for the silhouettes Transform Feedback Buffer, allowing us to store up to $655,360$ silhouette edges.

The memory requirement of our algorithm is *independent* of the scene complexity and depends only on the algorithm parametrization and the light types of the scene. Using only one depth complexity buffer (DC-buffer) for direct illumination without textured light and covered samples discretization requires only $(a)+(b)+(e)+(f)+(k) \approx 32MB$ of memory. On the other hand, computing an image modulated by the V-buffer, with *ten* textured lights, $four$ DC-buffers, an interleaved sampling strategy and the discrete covered samples representation requires $(a)+(b)+(c)+10*(d)+(e)+(f)+(g)+(h)+(i)+(j)+4*(k) \approx 105.575MB$ of memory. However, this memory requirement is not a limitation on current high-end GPUs.

		Format	Memory cost
(a)	**Silhouettes TFB**	1,310,720 × 32F	5MB
(b)	**Samples position**	64 samples packed in RGBA 32F	128B
(c)	**Edge LUT**	$(512^2 + 6 * 64^2) *$ RGBA 32F	4.375MB
(d)	**Per light LUT**	64×64 RGB 8UB	12KB
(e)	**Color + Stencil buffer**	RGBA 16F	8MB
(f)	**Z-buffer**	DEPTH_COMPONENT24	3MB
(g)	**V-buffer**	RGB 8UB	3MB
(h)	**Discontinuity buffer**	Alpha 8UB	1MB
(i)	**Temp filtered buffer**	RGBA 8UB	4MB
(j)	**Deferred buffer**	RGB 32F	12MB
(k)	**DC-buffer(s)**	RGBA 32F	16MB

Table 4.1: Detailed memory costs of our algorithm implementation with a 1024^2 image resolution. An edge LUT (c) stores our discreet covered samples representation while a per light LUT (d) stores the luminance texture lookup defined per textured light. The V-buffer (g) is not used when the equation 4.6 is solved. Finally, discontinuity (h), temp filtered (i) and deferred (j) buffers are only necessary with an interleaved sampling strategy.

4.5.2 Performance analysis

We present the performances of our algorithm on a complete 1024×1024 image rendering. Indirect lighting is approximated with an irradiance map [RH01] and the Blinn BRDF [Bli77] is used for the materials' appearance. Our renderer is based on the OpenGL API and all the shaders are written in the pseudo assembly language of the `NV_gpu_program4` extension [Bro06]. The results are measured on a 64-bits Linux workstation with a Core™ 2 Duo 2.4Ghz, 4GB of DDR2 800Mhz and a Geforce GTX−280. Our benchmarks measure the global rendering time (figure 4.11) and the cost of the different steps (figure 4.12) on *three* test scenes (figure 4.14, 4.15 and 4.16) with varying algorithm parametrization. To stress our approach, the light sources are not attenuated and so they do not take advantage of per-light scissor and depth bound optimizations. In addition, we do not perform any culling optimization. Finally, we use a counter packing representation where each DC-buffer stores 16 depth complexity counters with a precision up to 63 (equation 4.9).

Figure 4.11 illustrates that the Depth Complexity Sampling with 16 samples (DCS16) is approximately 3 times faster than using 64 samples (DCS64). This performance improvement is explained by the limitation on memory bandwidth and texture fetches, in addition of the reduction in working load of the raster operation unit. To reduce the noise of the shadows computed with 16 depth complexity counters, the interleaved sam-

pling strategy can be used with a negligible performance cost (DCSi16). Approximating shadows for semi-opaque occluders (t-DCS64) is less than 4% slower than the common depth complexity sampling. Note that compared to a direct lighting modulated by a V-buffer, solving the equation 4.6 (DLS64) leads to a performance drop between 7% and 10%, except for the game scene, where the rendering time is 64% slower than using a V-buffer. This result is explained by the effectiveness of the V-buffer on low-polygon scenes. Indeed, in these scenes the simple direct lighting computation step is very fast, since very few triangles have to be transformed. However, solving the equation 4.6 is more computationally intensive and results in a more important performance gap between the two direct lighting approaches than on a high-polygon scene (figure 4.12). Finally, we observe in figure 4.12 that our discrete covered samples representation (Discrete DCS64) improves performances by a percentage between 39% and 63% with respect to a naive search of the covered samples.

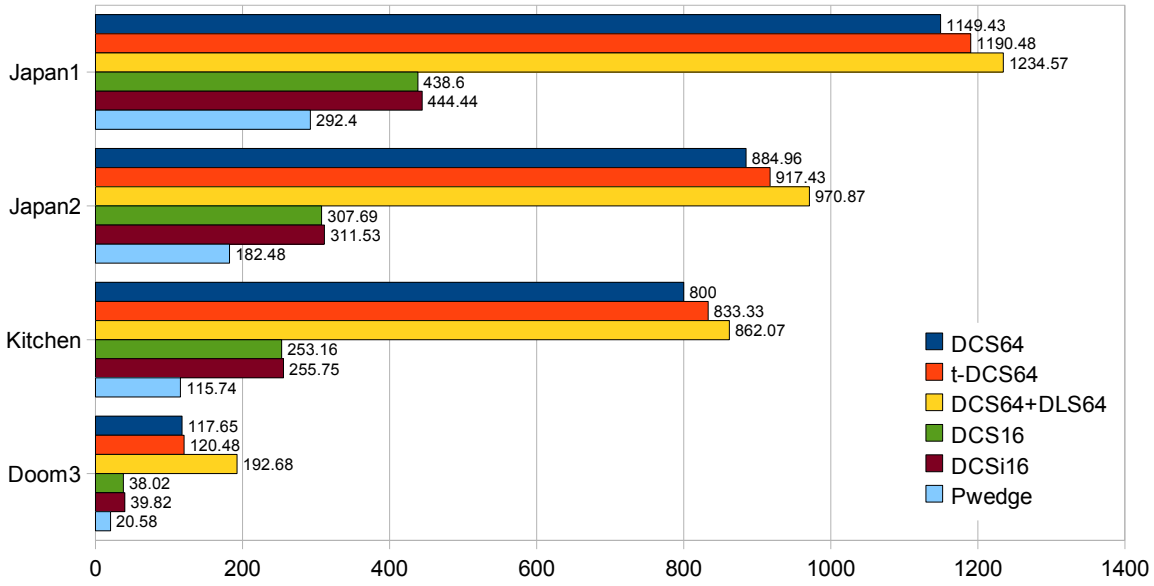


Figure 4.11: Rendering time in milliseconds for a 1024^2 image. Japan1: 4 lights, 592,047 polygons; Japan2: Japan1 scene with a reduced number of polygons (293,272 triangles); Kitchen: 2 lights, 179,383 polygons; Doom3: 2 lights, 22,451 polygons.

4.6 Discussion

The presented depth complexity sampling algorithm allows the generation of fast and accurate shadows. However, since it uses the soft shadow volume framework, it handles only polygonal models and is submitted to fill rate bottleneck. Furthermore, the splitting

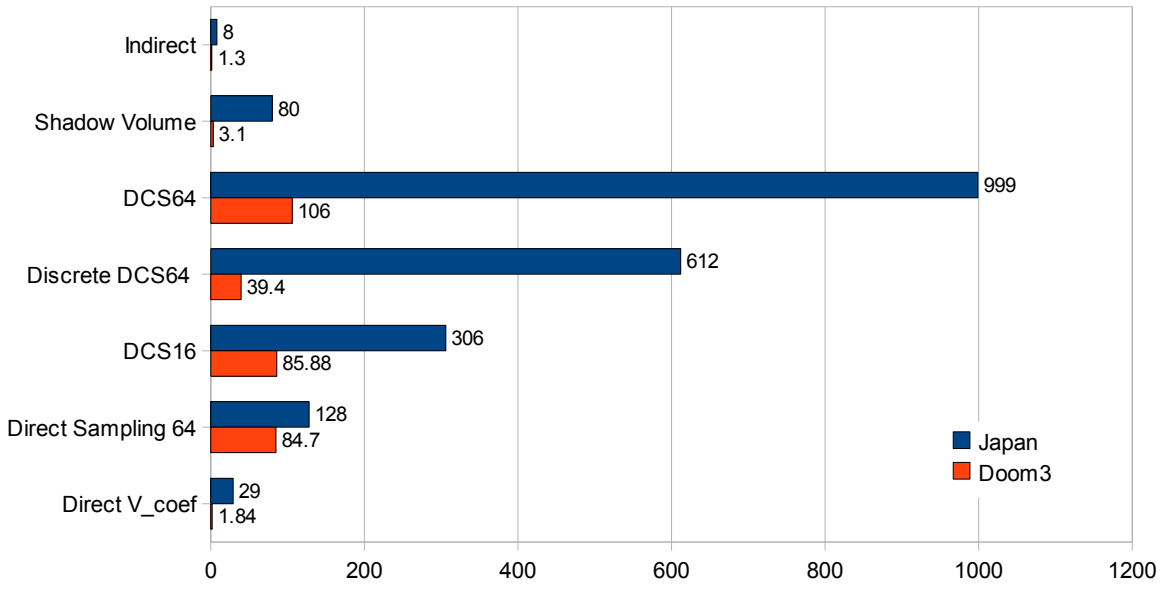


Figure 4.12: Time in milliseconds of the rendering steps according to the algorithm parametrization. Image resolution: 1024^2; Japan: 4 lights, 592,047 polygons; Doom3: 2 lights, 22,451 polygons.

of wedges in inner and outer parts generates aliasing due to precision errors. Nevertheless, our approach is orthogonal to the previous object-based methods and thus it may take advantage of all their current and future improvements.

Due to its sampling nature, and despite the use of our adaptive sample distribution, the proposed algorithm is still exposed to the sub-sampling artifacts. Nevertheless, since we use a Monte Carlo sample distribution, averaging the result of several runs would give a solution that would be statistically very close to the exact solution. One can therefore imagine a progressive rendering or a multi-GPU system where each picture is computed according to different distributions and then averaged in the final image.

The adaptive sampling algorithm defines the format of the counters for a given point of view. Note that this encoding can be locally defined per pixel rather than globally set for the current rendered image. A simple idea is to track the number of wedges bounding each pixel.

The lack of integer support in the blending stage limits both the precision of the counters and the efficiency of memory usage. In addition, despite the use of an adaptive sampling strategy, the counter packing robustness is still compromised by an eventual overflow error. These issues are due to subjective limitations of common graphics APIs [Mic08, SA06] that support only a specific set of render target formats. The current evolution toward a software graphics programming model [NVi08, SCS+08] will avoid such arbitrary limitations.

4.7 Conclusion

We have presented the depth complexity sampling soft shadow algorithm. It addresses the limitations of both the penumbra wedge approach [AAM03] (figure 4.13) and the offline soft shadow volume methods [LAA+05, LLA06]. Depending on the desired trade off between performance and quality, the resulting shadows are either very close to, or as accurate as, a ray-traced reference. Thus, it is well suited to many domains ranging from quality sensitive to performance critical applications.

The proposed implementation solved the direct lighting problematic onto the GPU, only. No a-priori knowledge about the mesh properties (animation, deformation, *etc.*) is required and no intermediary CPU $\leftrightarrow$ GPU transfer is performed. Thus, in contrast to common object-based frameworks, this implementation is totally independent of the underlying scene. In fact, it is a black box that provides the visibility information between a set of light samples and the visible receivers. Finally, thanks to the efficiency and the functionalities of the targeted graphics hardware, we demonstrate that despite its object-based nature, this algorithm can efficiently deal with complex animated models lit by a complex direct illumination (figure 5.12).

(a) PWedge (b) DCS (c) RT

Figure 4.13: Comparison of the shadows generated by the penumbra wedge algorithm (a), our Depth Complexity Sampling (b) and a Ray-Traced reference [mi] (c). In addition of the overlapping penumbrae artifact, the penumbra wedge algorithm produces an incorrect lighting due to the single light sample direct lighting computation.

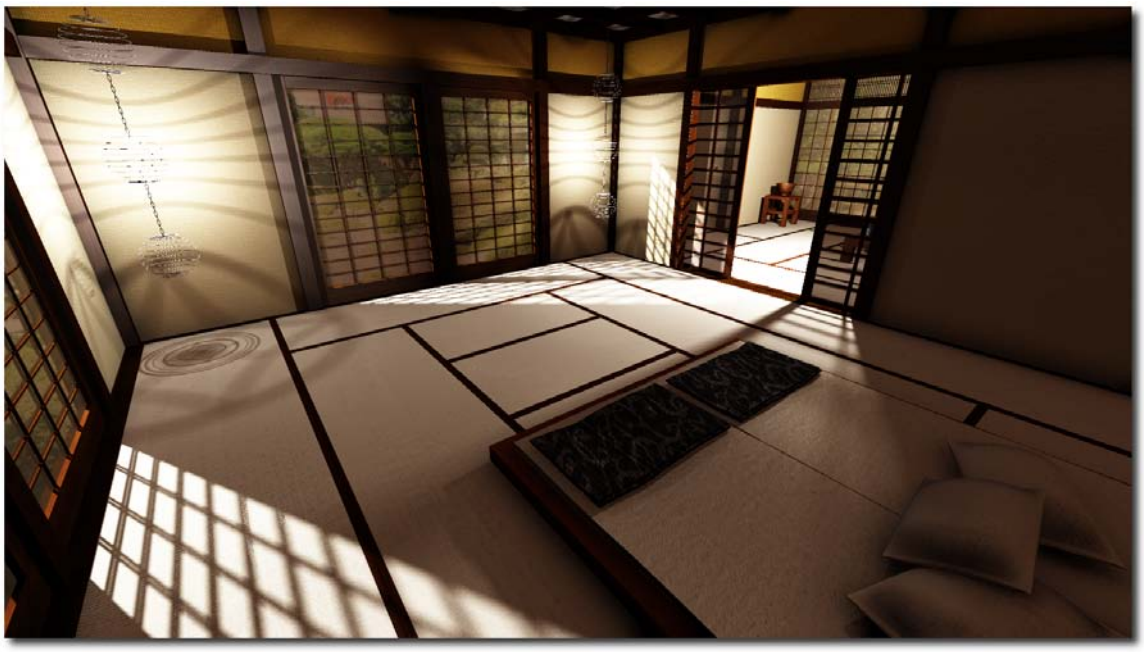


Figure 4.14: Japan test scene.

Figure 4.15: Kitchen test scene.

Figure 4.16: Doom3 test scene.

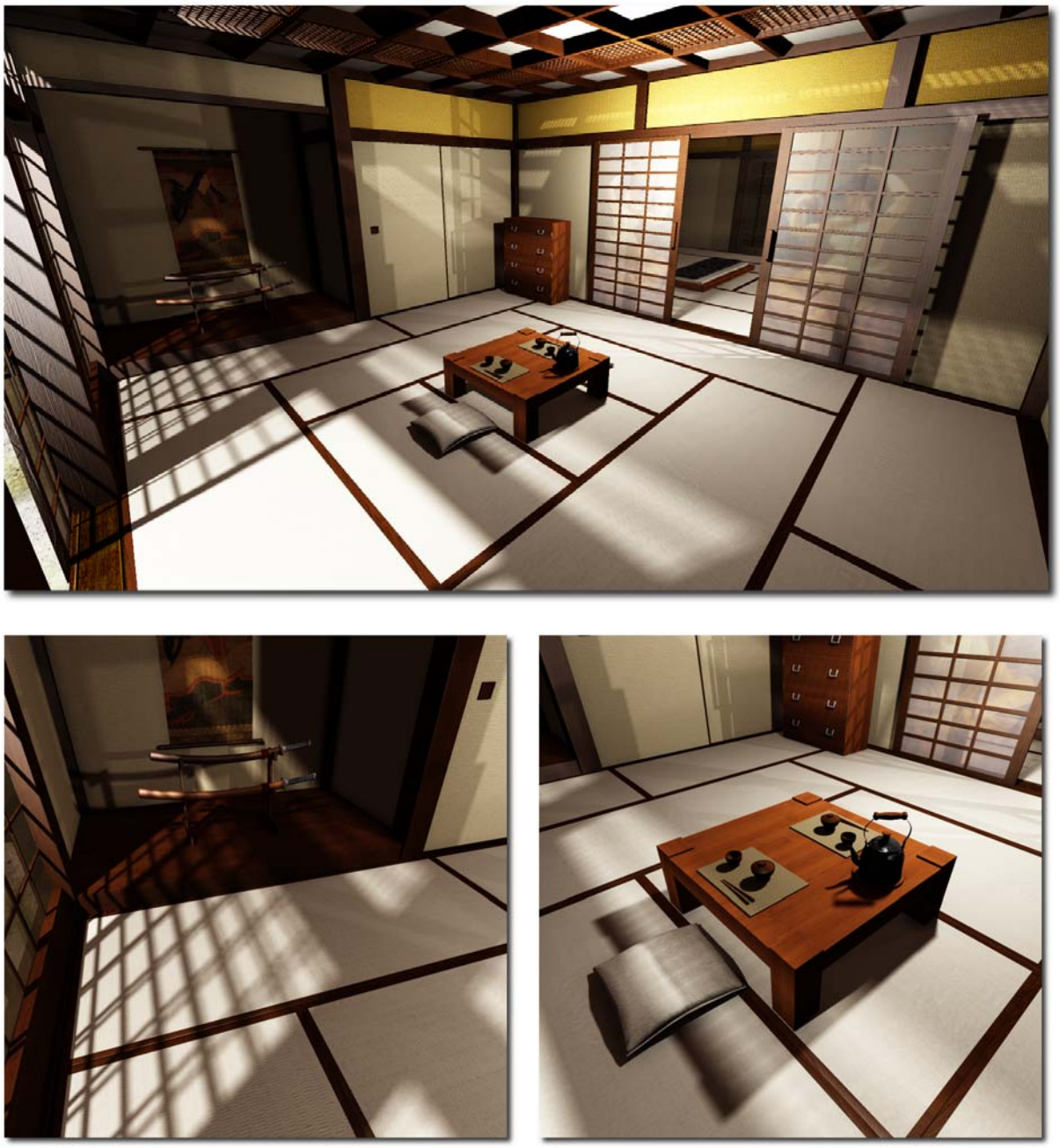


Figure 4.17: High quality shadows produced by our algorithm.

5

Soft textured shadow volumes

Object-based algorithms are well suited for the robust simulation of direct shadows. However, they exhibit an important drawback: they can only deal efficiently with triangle based geometries having a constant transmittance factor. Thus, unlike image-based shadows, object-based algorithms cannot handle perforated triangles. Such geometry is widely used in real time and offline applications in order to define highly perforated and thin objects with few triangles and alpha textures encoding their binary opacity (*e.g.* a wire fence).

As a first step, Hasselgren and Akenine-Möller [HAM07] propose an extension of the shadow volumes computing *pixel exact hard shadows* cast by triangles with a spatially varying transmittance (including perforated triangles). Eventually, this technique could also produce a coarse approximation of soft shadows by pre-filtering the transmittance texture.

This chapter addresses the next step. We propose a general algorithm computing robust and accurate *soft shadows* for triangles with a spatially varying transmittance denoted as S-triangles. We also show how this technique can be included into object-based soft shadow frameworks, and we present an efficient and practical GPU implementation with binary transmittance textures (a texel is fully transparent or full occluder). In general, the use of perforated triangles allows a significant reduction of the geometric overhead. Our approach takes benefit of this property since our soft shadow computation is accelerated by a factor varying between 25 and 35 when compared with object-based soft shadows generated from an equivalent geometry represented by meshes.

In section 5.1 we detail our algorithm. In order to provide an unified and robust shadow framework, we show in section 5.2 how our approach can be naturally and easily integrated into object-based soft shadow algorithms. Section 5.3 describes our GPU implementation. Finally, we present our results in section 5.4 and we conclude with a discussion and directions for future work.

5.1 Soft textured shadow volumes

In this section we present our algorithm that allows the generation of *accurate soft shadows* cast by S-triangles (section 5.1.1). In practice, this boils down to the computation of the influence of S-triangles on the visibility between the receivers and the area light source.

The proposed approach merges the properties of both soft [AAM03, FBP08] and textured [HAM07] shadow volumes: it extrudes conservative volumes from S-triangles in order to define their soft shadow influence. We thus call our technique *Soft Textured Shadow Volumes* (STSV).

5.1.1 The algorithm

Targeting objects with spatially varying transmittance, we divide the occluding geometries into *two* sets of triangles. The shadows from opaque triangles are evaluated in a conventional way (using any soft shadow algorithm) while S-triangles are treated separately using the algorithm 11.

Algorithm 11 STSV(Triangles T, Light l)

Require: samples distributed onto the light source

```
 1: clear_light_sample_buffer(l.LS_buffer);
 2: for all t ∈ T do
 3:    stsv_t ⇐ build_soft_textured_shadow_volume(t, l);
 4:    for all visible points p ∈ stsv_t do
 5:       for all light samples s as seen by p do
 6:          tmp ⇐ texture_access( t, t.tex_transmittance, s, p);
 7:          update_light_sample(l.LS_buffer[p][s], tmp);
 8:       end for
 9:    end for
10: end for
```

For a light l, a light sample buffer (LS_buffer) stores the information necessary to derive the visibility interaction between a visible point **p** and a set of light samples. We point out that the visible points **p** are already defined (for instance using a first rendering pass initializing the Z-buffer). For each S-triangle **t** (line 2), we extrude a *Soft Textured Shadow Volume* (STSV) (line 3 and section 5.1.2). The STSV conservatively includes the region in which light visibility is influenced by **t**. For all **p** lying in the STSV (line 4 and section 5.1.3), we finally update the LS_buffer according to the influence of **t** on the visibility between **p** (section 5.1.4) and a set of light samples (line 6, 7 and section 5.1.5).

Note that we do not explicitly define the information stored into the LS_buffer yet. Indeed, the LS_buffer can encode any data allowing the computation of the visibility interactions. For instance, the LS_buffer could store the percentage of light intensity

attenuated by the S-triangles lying between any visible point **p** and a set of light samples **s**.

5.1.2 Soft textured shadow volume extrusion

We define the shadow influence of the S-triangle **t** by extruding a volume that conservatively includes both the umbra and the penumbra regions of **t** (figure 5.1). We use the robust Z-fail strategy [Car00, EK02] to rasterize the resulting primitive and thus we have to close the Soft Textured Shadow Volume with front and back caps.

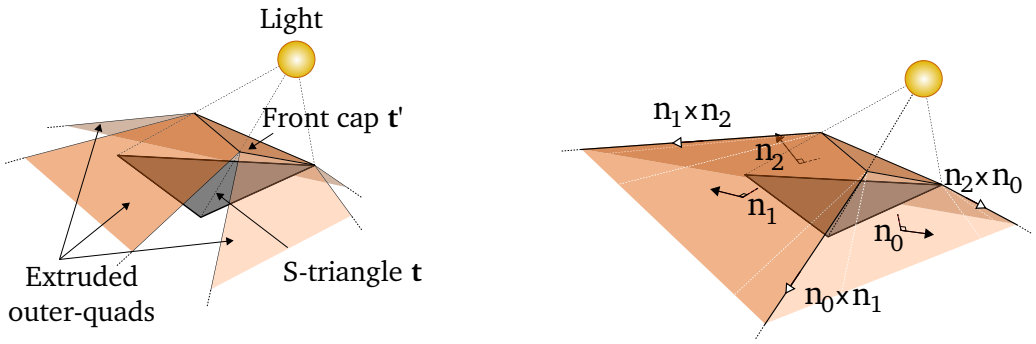


Figure 5.1: Construction of the Soft Textured Shadow Volume. We first extrude the penumbra wedge outer-quads from the front cap triangle edges (left). Then we close the volume by connecting the adjacent outer-quads (right).

For a robust construction that fully encloses the shadow region of **t**, we define which vertex of **t** is the closest to the light center [AAM03]. Then, we move the others towards the light center until the distance is the same for all the vertexes. The resulting vertexes describe a new triangle $\mathbf{t}'$ that defines the front cap of the STSV. For each edge of $\mathbf{t}'$ we extrude the outer-quad of its corresponding penumbra wedge primitive (figure 5.1 left). At this step we connect these quads in order to build a closed volume defining an upper bound of the shadow region of $\mathbf{t}'$ (figure 5.1 right). This is done by first evaluating the cross product of the normals of *two* adjacent outer-quads to define the direction of their intersection. Then, we extrude the vertex belonging to $\mathbf{t}'$ and shared by the quads along this direction. Finally, we define the back cap triangle of the STSV with the set of extruded vertexes.

5.1.3 Points into soft textured shadow volume

A given point **p** lies inside a STSV if it is on the same side of the *four* STSV planes (figure 5.2 left). This test is done efficiently using the interpolation algorithm of [HAM07].

In order to introduce notations and the technical background, we briefly present this algorithm.

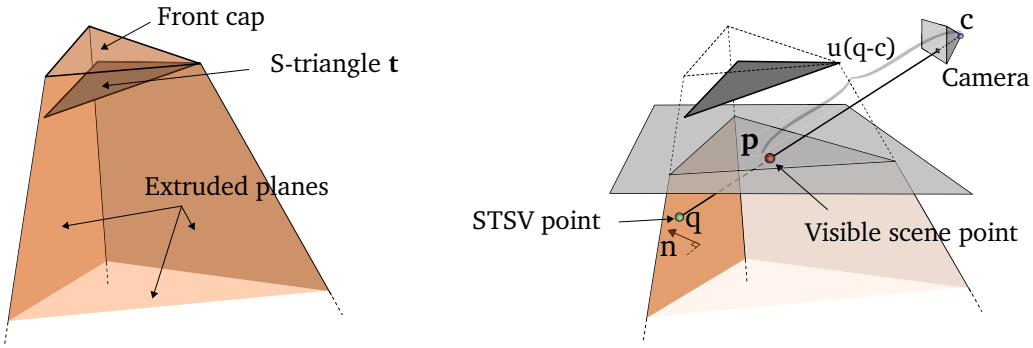


Figure 5.2: Z-fail rasterization of an extruded STSV plane defining if the visible surface point $\mathbf{p}$ is in the inside side of the plane (equation 5.1).

Each STSV plane $\mathcal{P}$ is defined by its implicit equation $\mathcal{P}(v) := n \cdot v + d = 0$ where n is the plane normal, d the orthogonal distance from the origin to the plane and v a point in space. A visible point $\mathbf{p}$ is inside the STSV if $\mathcal{P}(\mathbf{p}) < 0$ for all STSV planes. In order to avoid per $\mathbf{p}$ time consuming computations, the implicit equation is reformulated as follows:

$$\begin{aligned} \mathcal{P}(\mathbf{p}) &= n \cdot \mathbf{p} + d \\ &= n \cdot (c + u(q - c)) + d \\ &= (n \cdot c + d) + u(n \cdot (q - c)) \end{aligned} \tag{5.1}$$

where c is the camera position, q the projection of $\mathbf{p}$ onto $\mathcal{P}$ with respect to c, and u the linear interpolation parameter that defines $\mathbf{p}$ according to q and c (figure 5.2 right). Using this formulation the scalar $(n \cdot c + d)$ is constant for each $\mathcal{P}$ and it is then computed per STSV plane. The scalar $(n \cdot (q - c))$ is first computed per STSV vertex with q being the vertex position and then retrieved for any q onto the STSV plane by interpolation. This interpolation is efficiently performed by rasterization. The point q coordinates are also given by the rasterization and u is computed as $u = (\mathbf{p}.z - c.z) / (q.z - c.z)$. Once efficiently implemented [HAM07], the evaluation of $\mathcal{P}(\mathbf{p})$ only requires *two* multiplications and *one* addition per visible point $\mathbf{p}$.

5.1.4 Accessing the transmittance texture

The access to the transmittance texture of the occluding triangle $\mathbf{t}$ can be performed by shooting a ray from $\mathbf{p}$ to a light sample. The barycentric coordinates of the ray↔triangle

intersection [Bad90, MT97] is then used to interpolate the texture coordinates of $\mathbf{t}$ and access the transmittance value.

However, in the context of *hard shadow* computation, a more efficient approach has been proposed [HAM07]. This technique is called texture projection and it is summarized as follows.

Texture projection

Given a S-triangle $\mathbf{t}$, its normal n, its world space vertex coordinates v_0, v_1, v_2, their associated textured coordinates t_0, t_1, t_2 and the orthogonal distance k from the point light to $\mathbf{t}$. The surface points $\mathbf{p}$ included into the *hard textured shadow volume* of $\mathbf{t}$, are transformed into the homogeneous texture space as follows:

$$\begin{aligned} \mathbf{p}'_h &= \overbrace{S \cdot K \cdot M}^{T} \cdot \mathbf{p} \\ &= T \cdot (c + u\,(q - c)) \\ &= (T \cdot c) + u\,(T \cdot (q - c)) \end{aligned} \tag{5.2}$$

where M is a transformation matrix from the world to the homogeneous triangle space having the point light as origin,

$$M = \begin{pmatrix} \overrightarrow{v_0v_1}.x & \overrightarrow{v_0v_2}.x & -k * n.x \\ \overrightarrow{v_0v_1}.y & \overrightarrow{v_0v_2}.y & -k * n.y \\ \overrightarrow{v_0v_1}.z & \overrightarrow{v_0v_2}.z & -k * n.z \end{pmatrix}^{-1} \tag{5.3}$$

K is the matrix that translates the first triangle vertex to the origin, defining the homogeneous barycentric space,

$$K = \begin{pmatrix} 1 & 0 & -(M \cdot v_0).x \\ 0 & 1 & -(M \cdot v_0).y \\ 0 & 0 & 1 \end{pmatrix} \tag{5.4}$$

and S is the transformation from the homogeneous barycentric space to the homogeneous texture space.

$$S = \begin{pmatrix} \overrightarrow{t_0t_1}.x & \overrightarrow{t_0t_2}.x & t_0.x \\ \overrightarrow{t_0t_1}.y & \overrightarrow{t_0t_2}.y & t_0.y \\ 0 & 0 & 1 \end{pmatrix} \tag{5.5}$$

Equation 5.2 does not perform any projection. It evaluates an affine transformation of $\mathbf{p}$ into $\mathbf{p}'_h$. Thus, the position $(\ T \cdot (\ q - c)\ )$ is computed per hard textured shadow volume vertex and then interpolated across the surface. In addition, the position $(\ T \cdot c\ )$

is evaluated per S-triangle since it is constant for the triangle $\mathbf{t}$. The texture transmittance coordinate $\mathbf{p}'$ of a point $\mathbf{p}$ is finally retrieved by first evaluating the equation 5.2 according to the per $\mathbf{p}$ parameter u, and then performing the projection:

$$\mathbf{p}' = \frac{1}{\mathbf{p}'_h.z} \begin{pmatrix} \mathbf{p}'_h.x \\ \mathbf{p}'_h.y \end{pmatrix} \tag{5.6}$$

Transmittance sampling

The use of an area light requires the access to the set of transmittance values affecting the direct lighting of a point $\mathbf{p}$. When sampled, an area light can be considered as a set of point lights. Thus, for each light sample, the equation 5.2 could be directly used in order to access the transmittance texture of a S-triangle. However, the per S-triangle position ($T \cdot c$) and the per STSV vertex position ($T \cdot (q - c)$) must be computed per light sample. This would be particularly inefficient in terms of number of parameters to compute and to transmit when sampling the transmittance of the S-triangle.

Instead, we reduce the per light sample computations by reversing the texture projection problematic: we project each light sample onto the triangle plane as seen by $\mathbf{p}$ rather than projecting $\mathbf{p}$ from the light samples onto $\mathbf{t}$. Let denote as e the orthogonal distance from $\mathbf{p}$ to $\mathbf{t}$. We retrieve the texture transmittance value affecting the visibility of $\mathbf{p}$ and $\mathbf{s}$ by computing the homogeneous texture transmittance coordinates $\mathbf{s}'_h$ as follows:

$$\mathbf{s}'_h = S \cdot \overbrace{W \cdot M'}^{J} \cdot \mathbf{s} \tag{5.7}$$

where M' is the transformation from the world to an homogeneous triangle space,

$$M' = \begin{pmatrix} \overrightarrow{v_0v_1}.x & \overrightarrow{v_0v_2}.x & -n.x \\ \overrightarrow{v_0v_1}.y & \overrightarrow{v_0v_2}.y & -n.y \\ \overrightarrow{v_0v_1}.z & \overrightarrow{v_0v_2}.z & -n.z \end{pmatrix}^{-1} \tag{5.8}$$

W is the matrix setting $\mathbf{p}$ as the projective center and performing the transformation into the homogeneous barycentric space of the S-triangle,

$$W = \begin{pmatrix} 1 & 0 & -\left(M' \cdot (v_0 - \mathbf{p})\right).x/e \\ 0 & 1 & -\left(M' \cdot (v_0 - \mathbf{p})\right).y/e \\ 0 & 0 & 1 \end{pmatrix} \tag{5.9}$$

and S is the transformation from the homogeneous barycentric triangle space to the homogeneous texture space (equation 5.5). In a first step, we evaluate the S and M'

matrices per S-triangle since they are constant for a triangle. Then, for each visible point $\mathbf{p}$ lying into the STSV, we compute its corresponding transformation W. We multiply the W and M' matrices to define the transformation J from world space to homogeneous barycentric coordinates (equation 5.7). Due to the conservative nature of the STSV, some light samples as seen by $\mathbf{p}$ may not be occluded by $\mathbf{t}$. Let $\mathbf{s}_h^* = J \cdot \mathbf{s}$. A light sample s is occluded if $(\mathbf{s}_h^*.x \geqslant 0)$, $(\mathbf{s}_h^*.y \geqslant 0)$ and $(\mathbf{s}_h^*.z - \mathbf{s}_h^*.x - \mathbf{s}_h^*.y \geqslant 0)$. We transform the homogeneous barycentric coordinates $\mathbf{s}_h^*$ of the occluded light samples into homogeneous texture coordinates $\mathbf{s}'_h$ by applying the matrix S. We finally retrieve the texture transmittance position $\mathbf{s}'$ by simply performing the projection:

$$\mathbf{s}' = \frac{1}{\mathbf{s}'_h.z} \begin{pmatrix} \mathbf{s}'_h.x \\ \mathbf{s}'_h.y \end{pmatrix} \tag{5.10}$$

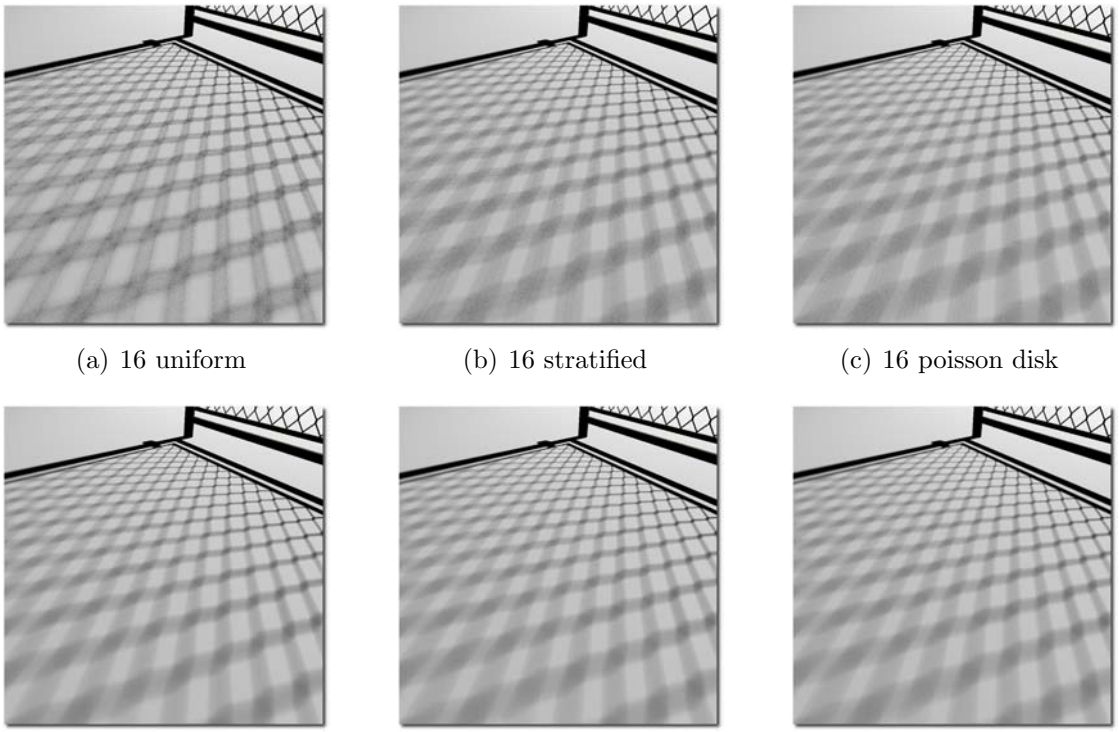
Even though several computations are performed for each visible point $\mathbf{p}$, the use of the equation 5.7 is still more efficient than a ray-traced approach to access the transmittance texture (appendix 5.A). Indeed, the matrix J is computed once for a given $\mathbf{p}$ (whatever the number of light samples). Thus, the computations of $\mathbf{s}_h^*$ only requires 9 `MUL` and 6 `ADD` and for the occluded light samples, the evaluation of the texture transmittance coordinates is obtained with 6 `MUL`, 4 `ADD` and 2 `DIV`.

5.1.5 Light sampling strategy

In order to compute the visibility interaction between a visible point $\mathbf{p}$ and an extended light source accurately, it is necessary to distribute the light samples using an adapted probability distribution function [PH04]. Following the depth complexity sampling algorithm (chapter 4) we avoid visible sampling patterns by using decorrelated sampling strategy, *i.e.* a sample set is generated independently for each visible point $\mathbf{p}$ according to a specific sample distribution. A good sample distribution is given by the stratified sampling approach (figure 5.8(b)). However, better results are obtained with more sophisticated methods such as the Poisson disk [Coo86] (figure 5.3(c) and 5.3(e)) or the low discrepancy [Nie92] (figure 5.3(d)) sampling strategy. Finally, any of these sample distributions can be combined with the interleaved sampling strategy [KH01, SIMP06] (figure 5.3(f)) in order to reduce the decorrelation noise.

5.2 Unified object-based soft shadow framework

Theoretically our STSV approach can be used for the generation of soft shadows from both standard meshes and S-triangles. However, it requires unnecessary computations for



(a) 16 uniform (b) 16 stratified (c) 16 poisson disk

(d) 64 low discr. (e) 64 poisson disk (f) 64i poisson disk

Figure 5.3: Impact of the sampling strategy on the shadow quality. Shadows are generated with either 16 (a, b, c) or 64 (d, e, f) decorrelated light samples using a uniform (a), stratified (b), low-discrepancy (d) or Poisson disk (c, e) sampling strategy. Any of these sampling techniques can be combined with any interleaved sampling pattern. For instance, figure (f) illustrates shadows generated with a 4×4 interleaved sampling pattern of 64 samples distributed with the Poisson disk strategy.

triangles without a transmittance property and conventional object-based approaches are far more efficient. Thus, we show how our STSV algorithm can be naturally integrated in object-based soft shadow frameworks.

Most S-triangles with spatial varying transmittance are used to represent perforated triangles, *i.e.* the transmittance texture encodes its spatially varying binary opacity (fence, branch, grass, *etc.*). Indeed, such representation is naturally and efficiently integrated in real time renderers. Even though the STSV algorithm performs with any transmittance values, our implementations thus focus on perforated triangles.

(a) PWedge+STSV (b) DCS+STSV (c) RT

Figure 5.4: Comparison between the shadows computed by the Penumbra Wedge+Soft Textured Shadow Volume algorithm (a), the Depth Complexity Sampling+Soft Textured Shadow Volume technique (b) and a Ray Traced reference [mi] computed onto the fences represented by meshes (c).

5.2.1 Penumbra wedge

This section describes how we combine the penumbra wedges [AAM03] (chapter 3) with our STSV algorithm. For common meshes, a V_{coef} is computed per visible point $\mathbf{p}$, using the penumbra wedge algorithm. The shadows cast by perforated triangles are then evaluated with the STSV approach (algorithm 11). For each visible point $\mathbf{p}$, the light sample buffer (LS_buffer) stores a bit mask where each bit encodes the visibility between $\mathbf{p}$ and a light sample. The LS_buffer is computed as follows. For a given perforated triangle $\mathbf{t}$, we have to compute a bit mask for each point $\mathbf{p}$ lying into its extruded STSV. For each of these $\mathbf{p}$, the value of each bit is derived from the transmittance value affecting its light sample visibility (equation 5.7). Then, we perform a logical `OR` of the resulting bit mask with the corresponding mask stored in the LS_buffer. When all perforated triangles are treated, we evaluate the per $\mathbf{p}$ STSV V_{coef} with the bit mask of the LS_buffer. We add this V_{coef} with the one computed during the penumbra wedge pass in order to evaluate the final V_{coef} used to modulate the direct illumination.

In the case of overlapping S-triangles, the STSV V_{coef} does not exhibit the underestimation artifact. However, the inherent shadow overlapping artifact of the penumbra wedges still occurs when perforated triangles overlap common meshes (figure 5.4(a)).

5.2.2 Depth complexity sampling

The integration of the STSV algorithm into our Depth Complexity Sampling (DCS) framework is straightforward (figure 5.4(b)). For common occluders, the DCS algorithm evaluates the depth complexity between each visible point $\mathbf{p}$ and the light samples. Then,

the STSV algorithm updates the depth complexity for the visible points **p** as follows. First, for each **p** lying into the STSV of a perforated triangle **t**, the equation 5.7 allows us to retrieve the transmittance values affecting its visibility from the set of light samples. Then the depth complexity counter of the corresponding light sample **s** is incremented if the retrieved transmittance value indicates that **s** is occluded by **t**. After the depth complexity evaluation we finally perform the direct lighting step as proposed in the DCS framework, *i.e.* by numerically solving the direct illumination or by modulating the direct lighting with a V_{coef} (chapter 4).

5.3 Implementation

In this section we detail the GPU implementation of our Soft Textured Shadow Volume algorithm. We propose an algorithm (algorithm 12) targeting the generation of graphics processors that takes benefit of *Vertex Programs* (VP), *Geometry Programs* (GP) and *Fragment Programs* (FP).

Algorithm 12 render_scene(Scene w, RenderView v)

Require: Pre-computed sample pattern

```
 1: set_up_camera(v);
 2: (color-buffer, Z-buffer) ⇐ draw_ambient_lighting(w);
 3: for all l ∈ w.lights do
 4:    clear_shadow_buffers();
 5:    tri_o ⇐ w.get_opaque_triangles();
 6:    tri_p ⇐ w.get_perforated_triangles();
 7:    if Depth Complexity Sampling then
 8:       DCS(tri_o, l);
 9:       STSV_DCS(tri_p, l);
10:       color_buffer + = DCS_direct_lighting(w, l);
11:    else
12:       PWedge(tri_o, l);
13:       STSV_bitmask(tri_p, l);
14:       add_v_coef();
15:       color_buffer + = PWedge_direct_lighting(w, l);
16:    end if
17: end for
```

As in the shadow volume algorithm [Cro77], we first render the scene to compute an approximation of the indirect illumination and to initialize the Z-buffer (line 2). Then we separate common occluders from perforated triangles to perform the direct lighting passes (lines 5 and 6). Full opaque meshes are treated with object-based soft shadow algorithms (lines 8 and 12) while we generate the shadows cast by perforated triangles with our STSV algorithm (lines 9 and 13).

5.3.1 Sample distribution

Depending on the light type and the sampling strategy (section 5.1.5), we store in a texture a set of pre-computed light samples. We reduce the memory consumption and the number of texture fetches using the data representation of the sample positions described in the Depth Complexity Sampling algorithm of the previous chapter (section 4.4.1). The decorrelated sample distributions are generated by randomly rotating per pixel the pre-computed sample positions. According to the desired performance/quality ratio, we propose to distribute either 16 or 64 samples onto the light sources. Note that any number of samples can be chosen with respect to hardware limitations. The low discrepancy sample distribution is based on the (0, 2)-sequence while for the Poisson disk sampling strategy we experimentally fix the minimum distance constraint to 0.2456139 or 0.1076681 for respectively 16 or 64 samples.

5.3.2 Soft textured shadow volume extrusion

The Soft Textured Shadow Volume of each perforated triangle is robustly extruded onto the GPU with a geometry program implementing the procedure presented in section 5.1.2. In this GP, we compute the per STSV vertex parameters required by the interpolation algorithm (section 5.1.3). In order to ensure their correct interpolation we perform a *semi infinite* extrusion of the STSV rather than an infinite extrusion. We also transmit per STSV vertex, the constants computed per S-triangle: the per STSV plane scalar ($n \cdot c + d$) in equation 5.1 and the matrices S and M' in equation 5.7.

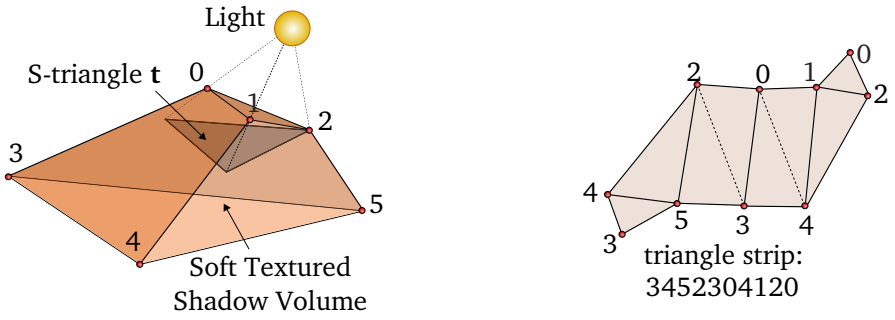


Figure 5.5: Triangle strip used for the Soft Textured Shadow Volume extrusion. The resulting extruded volume is capped and it implicitly defines a coherent normal orientation required for its robust Z-fail rendering [EK02].

Our STSV construction (section 5.1.2) minimizes the number of generated vertexes rather than build a tight bounding volume of the shadow cast by the S-triangle. This reduces both the computational complexity of its generation and the geometric amplification bottleneck of the geometry processors. The STSV is in fact a wedge that we generate

with a GP as described in the DCS framework (chapter 4 section 4.4.2). The figure 5.5 summarizes this construction.

After its extrusion, the STSV is sent to the rasterization stage where it is rendered according to the failure of the Z-buffer visibility test.

5.3.3 Transmittance sampling

For each fragment q resulting from the rasterization of the STSV, we retrieve its corresponding visible scene point $\mathbf{p}$ from the Z-buffer. In order to know if $\mathbf{p}$ is included into the STSV, we evaluate equation 5.1 for each STSV plane. Then for each light sample $\mathbf{s}$, the equation 5.7 allows us to retrieve the texture transmittance value affecting its visibility from $\mathbf{p}$ (appendix 5.A). We use these transmittance values to compute the visibility properties according to the chosen implementation (penumbra wedge or DCS). In both cases the fix Raster OPeration stage (ROP) updates the light sample buffer with respect to these visibility informations.

5.3.4 Direct illumination

On the one hand, in the STSV+penumbra wedge algorithm, we have to add the penumbra wedge and the STSV V_{coefs}. The resulting V_{coef} is evaluated for each pixel in a single full screen quad rendering pass. In this pass, we efficiently compute the STSV V_{coef} in a FP (appendix 5.B). This V_{coef} is then added to the penumbra wedge V_{coef} by the blending stage. Finally, in an additional rendering pass we modulate the direct lighting as in the penumbra wedge algorithm.

On the other hand, the STSV+DCS algorithm does not require any specific treatment for the direct lighting computation. Indeed, the light sample buffer (LS_buffer) is in fact a depth complexity buffer as defined in the DCS algorithm. Thus, either the computation of the V_{coef} modulating the direct lighting or the numerical integration of the direct illumination is directly evaluated by the DCS algorithm without any STSV specific treatment (we refer to the chapter 4 for additional details).

5.4 Results

5.4.1 Memory consumption

The treatment of S-triangles by our STSV algorithm requires neither pre-computed parameters nor additional per mesh texture. Indeed, the Soft Textured Shadow Volumes

are generated "on the fly" by the GPU while the transmittance textures are already associated to the S-triangles for their appearance. Thus, the memory consumption of our algorithm is independent of the geometric complexity of the scene.

When it is included in the penumbra wedge framework, the STSV algorithm requires both a set of pre-computed light samples and a light sample buffer storing the visibility bit mask. Thus, for a 1024×1024 frame buffer and 64 samples distributed onto the light source, the memory cost of the STSV implementation is:

$$\overbrace{1024^2 \times 64 \times 1bit}^{LS\_buffer} + \overbrace{64 \times 16bits}^{precomputed\,sample\,pattern} \approx 8MB$$

Note that the STSV algorithm does not require any specific memory allocation when it is included in the DCS framework. Indeed, the precomputed light samples are shared with those of the DCS and the light sample buffer is a reference to the depth complexity buffer.

5.4.2 Performances

The performances have been measured on a 64 bits Linux workstation with a Core^{TM} 2 Duo 3Ghz and 4GB of DDR2 at 800Mhz . The graphics hardware is a GeForce GTX280. Our implementation is based on the OpenGL API and the shaders are written using the pseudo assembly presented in the `NV_gpu_program4` extension. Our work targets direct soft shadow generation and the indirect lighting is orthogonally evaluated with any existing technique. Here, it is approximated using an irradiance map [RH01] and a screen space local ambient occlusion [SA07]. Finally, we use the Blinn BRDF [Bli77] for the materials' appearance.

Our benchmarks are based on *three* test scenes (figure 5.9, 5.10 and 5.11). We first compute the average rendering time with respect to a camera path defined for each scene (figure 5.6). We then evaluate the cost of the different rendering steps for a representative frame chosen into each camera path (figure 5.7). In order to avoid "biased" results, each frame is rendered from scratch, in a purely forward manner, without frustum culling, precomputed visibility set or scissor/depth bound test.

The HL2 scene (figure 5.11) is composed of $4,002$ perforated triangles casting large visible shadows. In such situation, the rendering time is computationally bounded by the Soft Textured Shadow Volume evaluation (from 50% to 75% of the whole rendering time). For instance, 70% of the rendering time is consume by the STSV when we numerically solve the direct lighting integral with 64 samples per light source. On the other hand, in the high poly-count factory scene (figure 5.9), highly detailed and thin geometries are represented with few S-triangles (14 perforated triangles). Thus, even though they cast

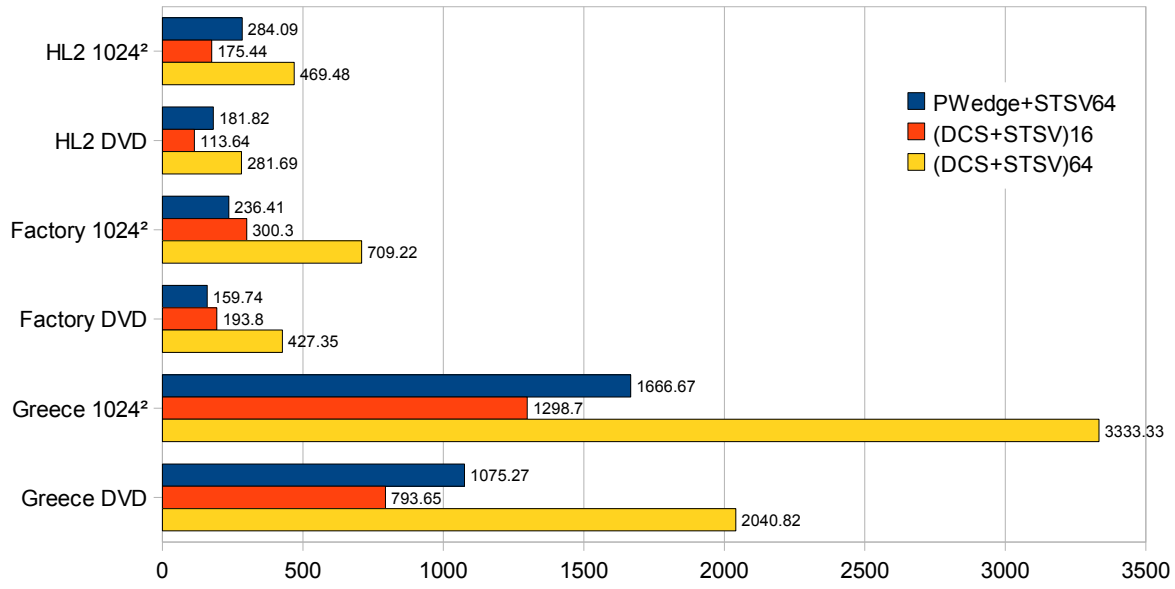


Figure 5.6: Rendering time in milliseconds on our *three* test scenes. The shadows are computed with our Soft Texture Shadow Volume algorithm (STSV) combined with either the Penumbra Wedge (PWedge) or the Depth Complexity Sampling (DCS) approach. The performances are reported for *two* image resolution: 1024×1024 or 720×576 (DVD).

large visible shadows, their computational cost is negligible (between 2% and 6% of the overall rendering time). The Greece scene (figure 5.10) is composed of $26,150$ S-triangles (*two* per leaf of the plants) and it is thus far more aggressive for our algorithm. However, in contrast to common wisdom and despite the horsepower required by the STSV evaluation, our object-based framework can generate robust direct soft shadows on this complex fully dynamic environment in about a second ((DCS+STSV)16).

5.5 Conclusion and discussion

We have presented the Soft Textured Shadow Volume algorithm. This algorithm addresses one of the main issues of the object-based shadow generation: the *accurate* computation of soft shadows cast by triangles with a spatial varying transmittance. In this section, we analyze its main advantages and discuss future improvements.

We have demonstrated that our approach can be naturally integrated into common object-based soft shadow algorithms. This provides efficient object-based frameworks computing soft shadows on fully animated scenes. Thanks to its object-based nature, our algorithm does not deal with shadow discretization aliasing nor shadow popping. In addition, it handles omni-directional area lights without any additional performance

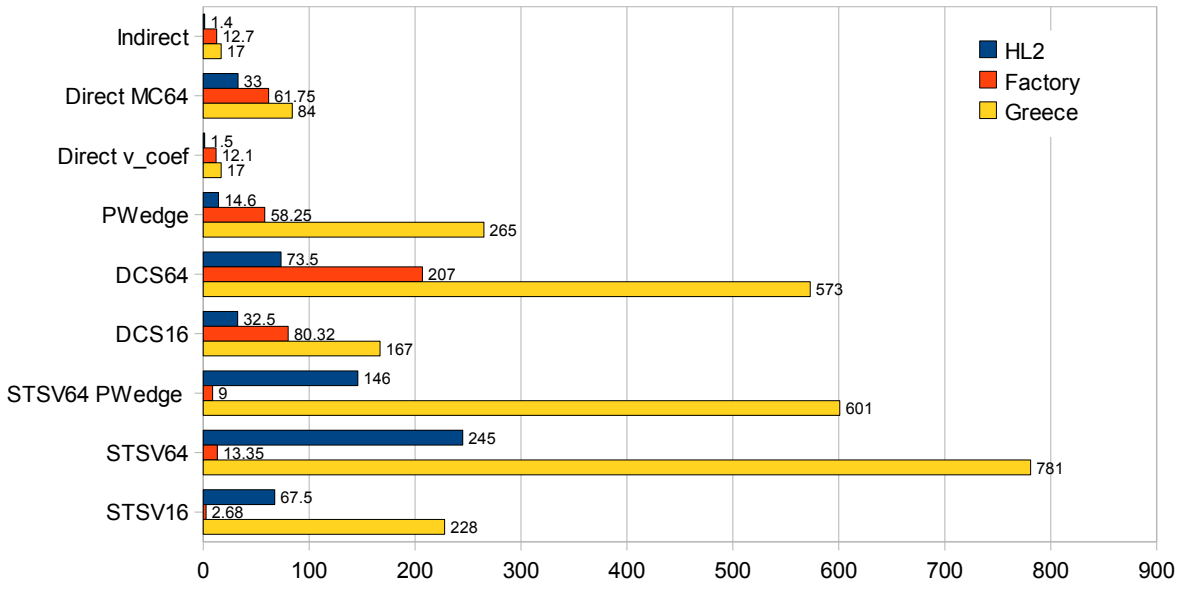


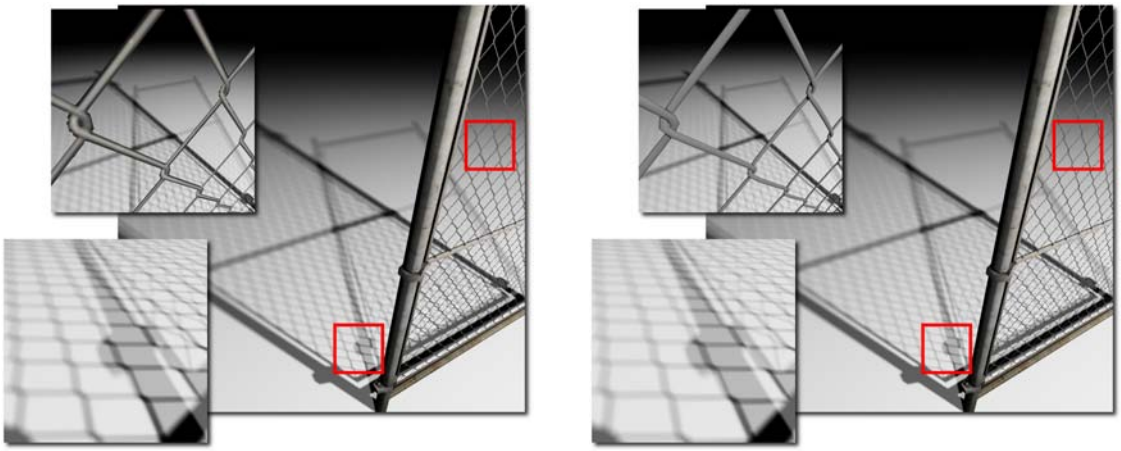
Figure 5.7: Time in milliseconds of the rendering steps for an image resolution of 1024 × 1024. The numbers nearby the acronyms give the number of per light samples used for each pixel. The direct lighting can be either attenuated with a V_{coef} or numerically solved by Monte Carlo sampling with the DCS framework (Direct MC64). Excepted for our indirect lighting pass (that is independent of the number of lights), the given times are the average of the per light computation times.

penalty or specific treatment.

Common object-based techniques extrude primitives only at the silhouette edges of the occluders whereas our STSV algorithm generates a STSV for *each* triangle. This could be seen as a bottleneck. However, perforated triangles are used as an alternative representation for detailed geometries (leafs, fence, *etc.*) and the generation of an analogous shadow quality cast by an explicit mesh representation would require much more horsepower (figure 5.8).

Also, perforated triangles are useful in a wide range of applications such as sprite/billboards [DDSD03, PMDS06, Ris07] or distant objects in a LOD hierarchy. The STSV algorithm is an efficient object-based approach allowing the generation of robust soft shadows on such representations.

We build the Soft Textured Shadow Volume according to the orientation of the corresponding S-triangle as seen from the light center. This leads to the well known single light sample artifact when the *two* sides of the triangle are lit by the area light. Indeed, in this situation, each side of the triangle has to cast a shadow since it occludes a part of the light. Nevertheless, this artifact can be very simply corrected by extruding a STSV for each side of the S-triangle when this case occurs.



(a) STSV (b) DCS

Figure 5.8: Comparison between the shadows cast by a fence represented by perforated triangles (2 perforated triangles, STSV: 90FPS) (a) and by its corresponding mesh (233, 358 triangles, DCS16: 3.3FPS) (b).

In order to get a fair performance/quality ratio, we distribute at most 64 light samples with sampling strategies that drastically reduce the variance. We have empirically set this maximum number of samples while the real limitation is given by the hardware constraints of the implementation. Nevertheless, thanks to the Monte-Carlo sampling nature of the STSV algorithm, the average of several runs would lead to a result statistically very close to the exact solution.

Figure 5.9: Representative frame of the factory test scene (582, 510 triangles, 14 perforated triangles and 4 omni-directional lights).

Figure 5.10: Representative frame of the Greece test scene (1, 396, 078 triangles, 26, 150 perforated triangles and 3 omni-directional lights).

Figure 5.11: Representative frame of the HL2 test scene (84,712 triangles, 4,002 perforated triangles and 2 omni-directional light).

5.A Transmittance value

`NV_gpu_program4` fragment program sub-routine. Retrieves the transmittance value of the perforated triangle affecting the visibility of the light sample `s_pos` as seen by the treated pixel.

```
# texture[0].w = transmittance texture of the STSV
get_transmittance_value:

    DP3.CC0 r0.x,     J_row0,  s_pos; #r0.xyz=s*_h=(W·M')s
    DP3.CC0 r0.y,     J_row1,  s_pos;
    DP3     r0.z,     J_row2,  s_pos;
    MOV     transm.w, 0;
    ADD     r1.z,     r0.x,    r0.y;  #r1.z=s*_h.z-s*_h.x-s*_h.y
    SUB.CC0 r1.z,     r0.z,    r1.z;
    RET     (LT.xyzz);                 #if(s*_h.z-s*_h.x-s*_h.y<0 ||
                                       #   s*_h.xy<0) return;
    DP3     r1.x,     S_row0,  r0;    #r1.xy,r0.z=s'_h=S·s*_h
    DP3     r1.y,     S_row1,  r0;
    DIV     r0.xy,    r1,      r0.z;  #r0.xy= s'=s'_h.xy / s'_h.z
    TXL     transm.w, r0,      texture[0], 2D;
RET;
```

5.B V_{coef} from visibility bit mask

`NV_gpu_program4` fragment program. Fast vectorized and parallel bit count routine. Compute the V_{coef} from the bit mask visibility of a set of 64 uniformly distributed light samples.

```
# texture[0] = RGBA16UI bit mask texture
ATTRIB f_pos = fragment.position;
TEX.U    bitmask, f_pos,  texture[0], RECT;

AND.U    r0,      bitmask, 0x5555;
SHR.U    bitmask, bitmask, 1;
AND.U    bitmask, bitmask, 0x5555;
ADD.U    bitmask, r0,      bitmask;
AND.U    r0,      bitmask, 0x3333;
SHR.U    bitmask, bitmask, 2;
AND.U    bitmask, bitmask, 0x3333;
ADD.U    bitmask, r0,      bitmask;
AND.U    r0,      bitmask, 0x0F0F;
SHR.U    bitmask, bitmask, 4;
AND.U    bitmask, bitmask, 0x0F0F;
ADD.U    bitmask, r0,      bitmask;
AND.U    r0,      bitmask, 0x00FF;
SHR.U    bitmask, bitmask, 8;
AND.U    bitmask, bitmask, 0x00FF;
ADD.U    bitmask, r0,      bitmask;

I2F      bitmask, bitmask;
DP4      v_coef,  bitmask, 0.015625;   # 0.015625 = 1/64
```

Figure 5.12: Illustration of shadows cast by perforated triangles (fences) using our soft textured shadow volume algorithm.

6

Conclusion

This chapter concludes this dissertation by discussing our technical choices (section 6.1) and summarizing our main results (section 6.2 and 6.3). We finally discuss future works on the accurate direct lighting simulation (section 6.4).

6.1 Rasterizing accurate soft shadows

In this thesis we have first outlined that the rasterization of robust shadows is a widely studied and challenging problem. We have observed that very few algorithms target the robust simulation of direct shadows. Several algorithms approximate shadows by modulating the direct lighting with a V_{coef}. Following the explicit formulation of the direct lighting, we have pointed out that the use of a V_{coef} leads to approximative results. In practice this rough solution may be well suited when targeting real time applications. Unfortunately, popular V_{coef}-based shadows exhibit strong inherent limitations that can be addressed with several concurrent techniques. In fact, no general purpose shadow algorithm exists, neither for the generation of fast approximative shadows nor for a robust direct lighting simulation. In other words, computing direct shadows efficiently remains an open problem.

Despite its initial "weakness", the object-based shadow framework provides a very strong algorithmic background [EK02, AAM03]. In spite of geometric constraints and performance bottlenecks, we have outlined that object-based shadows seem to be a right direction to go when designing robust shadow algorithms.

6.2 The penumbra wedge blending

Starting from the previous observations we have first investigated the object-based penumbra wedge algorithm. This approach computes very convincing soft shadows by analytically evaluating the percentage of light that is occluded. It is however based on the assumption that penumbrae are not overlapping. This leads to over shadowed artifacts when such assumption is not satisfied.

We have proposed an extension to the penumbra wedge framework that adds a penumbra blending stage in order to reduce the overlapping artifacts. We evaluated the penumbra wedge algorithm independently for each silhouette loop that are not overlapping. We also defined for each visible receiver an approximation of the shape of the occluders. We finally used these informations to blend the shadow contribution of each silhouette loop according to a proposed blending heuristic.

We have presented an implementation of the penumbra wedge framework improved by our blending stage. We have used common programmable graphics hardware that at this time did not support the generation of geometric primitives. Thus, following common object-based frameworks, we performed the shadow volume and the penumbra wedge construction on the CPU side while the GPU was used to efficiently evaluate the shadow contribution.

On the resulting shadows, our penumbra blending stage drastically reduces the penumbra overlapping artifacts. However, the obtained performances outline the popular limitation of the object-based approaches: their efficiency is greatly influenced by the geometric complexity of the scene. Such performances are essentially due to the silhouette detection and the primitive generation both performed onto the CPU. In our view, this bottleneck is not pathological. Better performances would be achieved with a careful implementation using software multi threading or graphics hardware. The core of the proposed soft shadow evaluation is far more problematic than this initial performances. Indeed, despite convincing shadows, we could not guarantee the correctness of the solution.

6.3 Robust unified object-based framework

The second contribution of this dissertation concerns a new object-based shadow framework that addresses the previous limitations.

6.3.1 The depth complexity sampling

We have proposed an approach that merges the efficiency of the real time penumbra wedges with the accuracy of its offline generalization [LAA+05]. The presented algorithm

evaluates the number of occluders, *i.e.* the depth complexity, lying between the emitter and the receiver. First, we use the penumbra wedge framework to define which silhouette edges affect the depth complexity between the area light and a visible receiver $\mathbf{p}$. Then we update the depth complexity between $\mathbf{p}$ and a set of light samples according to each silhouette edge. Unlike offline depth complexity approaches [LAA+05, LLA06], all our computations are performed locally. We thus avoid the use of a global data structure that would increase the computation time and the memory consumption.

Due to its sampling nature, the quality of the proposed depth complexity evaluation depends on both the number of samples and their distribution over the light. We have thus investigated several sampling strategies in order to obtain a good accuracy with few samples. We have also proposed a method dynamically adjusting the balance between the precision of the depth complexity and the number of samples.

Then, we have described how we use the depth complexity to generate soft shadows. First, we derive an artifact free V_{coef} from the depth complexity informations. This formulation provides a fast convincing approximation of the direct illumination. Targeting the robust simulation of direct shadows, we then use the depth complexity to solve the visibility queries in the direct lighting integral. This allows us to numerically solve the direct illumination with an accuracy depending on the light sampling strategy. Finally, we have presented an extension of our initial method that computes shadows cast by semi opaque occluders.

We have then presented a GPU intensive implementation of this object-based soft shadow algorithm. All the computations are performed onto graphics hardware, including silhouette detection and primitive generation. According to the desired trade off between performances and quality, the resulting shadows are computed either with respect to a V_{coef} or by numerically solving the direct illumination. We have described the implementation details that address the current hardware limitations and optimize the efficiency of our algorithm. As previously expected, this fully GPU implementation addresses the main performance bottlenecks of object-based shadows. Thus, complex fully animated environments can be efficiently handled by our object-based shadow algorithm.

6.3.2 Soft textured shadow volumes

We have studied the generation of soft shadows cast by occluders with a spatial varying transmittance. In real time rendering, this representation is mainly used to represent highly detailed and thin objects with few triangles and a transmittance texture encoding their binary opacity (*e.g.* a fence). Object-based soft shadow frameworks do not naturally handle such perforated triangles. We have presented an algorithm addressing this strong limitation.

Following object-based shadow approaches, the proposed algorithm is based on the extrusion of a volume that conservatively includes the scene region lying in the shadow of the occluder. We have detailed the construction of such volume. Then, we have described how we compute shadows by sampling the visibility with respect to the area light and the varying transmittance of the occluding triangle.

We have shown that this algorithm can be efficiently integrated into object-based frameworks. We first detail its integration into the original penumbra wedge approach. We show that our soft textured shadow volume evaluation does not exhibit penumbra overlapping errors. However, the inherent shadow overlapping artifact of the penumbra wedges still occurs when perforated triangles overlap common meshes.

We have outlined that using the soft textured shadow volumes with the depth complexity sampling framework avoids the previous drawbacks. We have presented this straightforward combination. Following the depth complexity sampling algorithm, we are able to either evaluate an artifact free V_{coef} or numerically solve the direct lighting for both opaque meshes and perforated triangles. In fact, this object-based framework answers the problematic of this dissertation: it provides an efficient robust soft shadow framework that naturally handles dynamic environments.

6.4 Conclusion and future works

In this dissertation we focus on a specific problematic of computer graphics. Our investigation targets the robust shadow generation or more generally the simulation of direct illumination for dynamic environments. We illustrates that object-based approaches can provide a fast and robust solution for this problematic.

The strength of the presented framework relies on its capability to define the reciprocal visibility between any visible receiver and the light samples. Thanks to this information some global problematics could be now investigated in rasterization. For instance, we have defined the sampling strategy according to the importance of the light. Better real time glossy highlight could be achieved by distributing the samples with respect to the importance of both the light and the bidirectional scattering distribution function [VG95]. The depth complexity sampling framework would be used to finally define the visibility between the receiver and its associated samples.

We outline that the unbiased direct lighting evaluation requires the detection of the silhouette edges with respect to the *shape* of the light. Such brute force approach drastically increases both the computation time and the fill-rate requirements. One can expect better performances by proposing new silhouette detection techniques that address this performance bottleneck.

We have pointed out that unlike image-based approaches, object-based shadow algorithms do not have to deal with magnification artifacts. However, as any rendering technique, the viewport discretization is still prone to image aliasing. This cannot be naively corrected by a multi-sampling approach since the shadow has to be evaluated for each sub-pixel sample. Thus, the shadow computation must be super-sampled. A more efficient solution would consist in combining a conservative rasterization [HAMO05] and a judicious anti-aliasing algorithm that super-samples the shadow only where the aliasing occurs.

More generally, it seems particularly interesting to propose a software implementation of our framework. This would avoid several current limitations of graphics API and would propose attractive perspectives. One can argue that the resulting performances would be quite poor. Even though the efficiency would inevitably decrease, we think that the performances would be acceptable. In fact, today, very few parts of the graphics pipeline are hardware accelerated. While it is common to define a hardware implementation as an implementation onto graphics hardware, the evolution towards a fully programmable graphics pipeline [SCS+08] is breaking down this misconception. Thus, we think that one can already develop an efficient software rasterizer onto current graphics hardware [NVi08] in order to propose a fully programmable pipeline without API constraints.

Finally, from a more general point of view, the presented results illustrate the efficiency of the rasterization for the direct illumination simulation. Despite promising perspectives it is still difficult to define the right direction to go for realistic real time renderers. On the one hand, the rasterization is robust, extremely fast for local effects and it efficiently deals with dynamic scenes. In addition, this thesis has illustrated that it can be also efficiently used for the robust simulation of direct shadows/illumination. On the other hand, we observe impressive improvements towards real time ray tracing. Ray tracers naturally handle global effects and thus these recent performance improvements open attractive real time perspectives. For instance, it seems particularly interesting to propose an hybrid real time renderer combining the efficiency of the rasterization with the generality of a ray tracer.

Bibliography

[AAA+02] Kurt Akeley, Allen Akin, Ben Ashbaugh, Bob Beretta, John Carmack, Matt Craighead, Ken Dyke, Steve Glanville, Michael Gold, Evan Hart, Mark Kilgard, Bill Licea-Kane, Barthold Lichtenbelt, Erik Lindholm, Benj Lipchak, Bill Mark, James McCombe, Jeremy Morris, Brian Paul, Bimal Poddar, Thomas Roell, Jeremy Sandmel, Jon Paul Schelter, Geoff Stahl, John Stauffer, and Nick Triantos. OpenGL extension: ARB_vertex_program. `http://www.opengl.org/registry/specs/ARB/vertex_program.txt`, 2002.

[AAM03] Ulf Assarsson and Tomas Akenine-Möller. A geometry-based soft shadow volume algorithm using graphics hardware. *ACM Transactions on Graphics*, 22(3):511–520, 2003.

[AAM04] Timo Aila and Tomas Akenine-Möller. A hierarchical shadow volume algorithm. In *Proc. SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 15–23. ACM Press, 2004.

[ADMAM03] Ulf Assarsson, Michael Dougherty, Michael Mounier, and Tomas Akenine-Möller. An optimized soft shadow volume algorithm with real-time performance. In *Proc. SIGGRAPH/EUROGRAPHICS Conference on Graphics hardware*, pages 33–40. Eurographics, 2003.

[AHL+06] Lionel Atty, Nicolas Holzschuch, Marc Lapierre, Jean-Marc Hasenfratz, Chuck Hansen, and François Sillion. Soft shadow maps: Efficient sampling of light source visibility. *Computer Graphics Forum*, 25(4):725–741, 2006.

[AHT04] Jukka Arvo, Mika Hirvikorpi, and Joonas Tyystjärvi. Approximate soft shadows with an image-space flood-fill algorithm. *Computer Graphics Forum, Proc. EUROGRAPHICS*, 23(3):271–280, 2004.

[AL04] Timo Aila and Samuli Laine. Alias-free shadow maps. In *Proc. EG Symposium on Rendering*, pages 161–166. Eurographics, 2004.

[AMA02] Tomas Akenine-Möller and Ulf Assarsson. Approximate soft shadows on arbitrary surfaces using penumbra wedges. In *Proc. EG Workshop on Rendering Techniques*, pages 297–306. Eurographics, 2002.

[AMA03] Thomas Akenine-Möller and Ulf Assarsson. On the degree of vertices in a shadow volume silhouette. *Journal of Graphics Tools*, 8(4):21–24, 2003.

[AMB+07] Thomas Annen, Tom Mertens, Philippe Bekaert, Hans-Peter Seidel, and Jan Kautz. Convolution shadow maps. In *Proc. EG Symposium on Rendering*, volume 18, pages 51–60. Eurographics, 2007.

[AMS+08] Thomas Annen, Tom Mertens, Hans-Peter Seidel, Eddy Flerackers, and Jan Kautz. Exponential shadow maps. In *Proc. of Graphics Interface*, volume 34, pages 155–161, 2008.

[ARHM00] Maneesh Agrawala, Ravi Ramamoorthi, Alan Heirich, and Laurent Moll. Efficient image-based methods for rendering soft shadows. In *Proc. SIGGRAPH*, pages 375–384. ACM Press, 2000.

[Arv04] Jukka Arvo. Tiled shadow maps. *Computer Graphics Forum*, 00:240–247, 2004.

[ASK06] Barnabás Aszódi and László Szirmay-Kalos. Real-time soft shadows with shadow accumulation. In *EUROGRAPHICS short papers*, 2006.

[Bad90] Didier Badouel. An efficient ray-polygon intersection. *Graphics gems*, pages 390–393, 1990.

[BAS02] Stefan Brabec, Thomas Annen, and Hans-Peter Seidel. Shadow mapping for hemispherical and omnidirectional light sources. *Computer Graphics International*, pages 397–408, 2002.

[BB84] L.S. Brotman and N.I. Badler. Generating soft shadows with a depth buffer algorithm. *Computer Graphics Application*, 4(10):5–12, 1984.

[BBC+02] Bob Beretta, Pat Brown, Matt Craighead, Cass Everitt, Evan Hart, Jon Leech, Bill Licea-Kane, Bimal Poddar, Jeremy Sandmel, Jon Paul Schelter, Avinash Seetharamaiah, and Nick Triantos. OpenGL extension: ARB_fragment_program. `http://www.opengl.org/registry/specs/ARB/fragment_program.txt`, 2002.

[BCS06] Louis Bavoil, Steven P. Callahan, and Claudio T. Silva. Robust soft shadow mapping with depth peeling. Technical Report UUSCI-2006-028, University of Utah, 2006.

[Ber86] Philippe Bergeron. A general version of crow's shadow volumes. *Computer Graphics Application*, 6(9):17–28, 1986.

[BJ] H. Batagelo and I. Junior. Realtime shadow generation using bsp trees and stencil buffers. In *Proc. SIGRAPI*, volume 12, pages 93–102.

[Bli77] James F. Blinn. Models of light reflection for computer synthesized pictures. In *Proc. SIGGRAPH*, pages 192–198. ACM Press, 1977.

[Bro06] Pat Brown. OpenGL extension: NV_gpu_program4. `http://www.opengl.org/registry/specs/NV/gpu_program4.txt`, 2006.

[BS99] Bill Bilodeau and Mike Songy. Real time shadows. Creative Labs sponsored Game developer Conference, unpublished slides, May 1999.

[BS02] Stefan Brabec and Hans-Peter Seidel. Single sample soft shadows using depth maps. In *Proc. Graphics Interface*, pages 219–228, 2002.

[BW04] Pat Brown and Eric Werness. OpenGL extension: NV_fragment_program2. `http://www.opengl.org/registry/specs/NV/fragment_program2.txt`, 2004.

[Car00] John Carmack. An email from john carmack to mark kilgard outlining z-fail algorithm - `http://developer.nvidia.com/attach/6832`. Id-Software, 2000.

[CD03] Eric Chan and Frédo Durand. Rendering fake soft shadows with smoothies. In *Proc. EG Workshop on Rendering Techniques*, pages 208–218. Eurographics, 2003.

[CG04] Hamilton Chong and Steven J. Gortler. A lixel for every pixel. In *Proc. EG Symposium on Rendering*, pages 167–172. Eurographics, 2004.

[Coo86] Robert L. Cook. Stochastic sampling in computer graphics. *ACM Trans. Graph.*, 5(1):51–72, 1986.

[Cro77] Franklin C. Crow. Shadow algorithms for computer graphics. In *Proc. SIGGRAPH*, pages 242–248. ACM Press, 1977.

[DAM+08] Zhao Dong, Thomas Annen, Tom Mertens, Philippe Bekaert, Hans-Peter Seidel, and Jan Kautz. Real-time, all-frequency shadows in dynamic scenes. *ACM Transactions on Graphics, Proc. SIGGRAPH*, 0(0):to appear, 2008.

[DDSD03] Xavier Décoret, Frédo Durand, François X. Sillion, and Julie Dorsey. Billboard clouds for extreme model simplification. In *Proc. SIGGRAPH*, pages 689–696. ACM Press, 2003.

[DF94] George Drettakis and Eugene Fiume. A fast shadow algorithm for area light sources using backprojection. *Computer Graphics Forum*, 28:223–230, 1994.

[Die01] Sim Dietrich. Shadow techniques. NVIDIA Cooperation, Game Developer Conference, slides, `http://developer.nvidia.com/object/gdc2001_shadow_techniques.htm`, 2001.

[DL06] William Donnelly and Andrew Lauritzen. Variance shadow maps. In *SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pages 161–165. ACM Press, 2006.

[ED06a] Elmar Eisemann and Xavier Décoret. Fast scene voxelization and applications. In *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pages 71–78. ACM SIGGRAPH, 2006.

[ED06b] Elmar Eisemann and Xavier Décoret. Plausible image based soft shadows using occlusion textures. In *Proc. SIGRAPI*, pages 155–162. IEEE Computer Society, 2006.

[ED07] Elmar Eisemann and Xavier Décoret. Visibility sampling on gpu and applications. *Computer Graphics Forum, Proc. EUROGRAPHICS*, 26(3):535–544, 2007.

[EK02] C. Everitt and M. Kilgard. Practical and robust stenciled shadow volumes for hardware-accelerated rendering. Technical report, NVIDIA Cooperation, 2002.

[Eng07] Wolfgang Engel. *ShaderX^5 - Advanced Rendering Techniques*, chapter Cascaded Shadow Maps, pages 197–206. Charles River Media, 2007.

[FBP06] Vincent Forest, Loïc Barthe, and Mathias Paulin. Realistic soft shadows by penumbra-wedges blending. In *Proc. SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 39–48. Eurographics, 2006.

[FBP08] Vincent Forest, Loïc Barthe, and Mathias Paulin. Accurate shadows by depth complexity sampling. *Computer Graphics Forum, Proc. EUROGRAPHICS*, 27(2):663–674, 2008.

[FBP09] Vincent Forest, Loïc Barthe, and Mathias Paulin. Soft textured shadow volume. Submitted to EUROGRAPHICS, 2009.

[Fer05] Randima Fernando. Percentage-closer soft shadows. In *SIGGRAPH Sketches*, page 35. ACM Press, 2005.

[FF88] Alain Fournier and Donald Fussell. On the power of the frame buffer. *ACM Transactions on Graphics*, 7(2):103–128, 1988.

[FFBG01] Randima Fernando, Sebastian Fernandez, Kavita Bala, and Donald P. Greenberg. Adaptive shadow maps. In *Proc. SIGGRAPH*, pages 387–390. ACM Press, 2001.

[FGH+85] Henry Fuchs, Jack Goldfeather, Jeff P. Hultquist, Susan Spach, John D. Austin, Jr. Frederick P. Brooks, John G. Eyles, and John Poulton. Fast spheres, shadows, textures, transparencies, and imgage enhancements in pixel-planes. In *Proc. SIGGRAPH*, pages 111–120. ACM Press, 1985.

[FZSXL06] Zhang-hang Fan Z, Hanqiu Sun, Leilei Xu, and Lee Kit Lun. Parallel-split shadow maps for large-scale virtual environments. In *Proc. of the Virtual reality continuum and its applications*, pages 311–318. ACM press, 2006.

[GBP06] Gael Guennebaud, Loïc Barthe, and Mathias Paulin. Real-time soft shadow mapping by backprojection. In *Proc. EG Symposium on Rendering*, pages 227–234, http://www.eg.org/, 2006. Eurographics.

[GBP07] Gael Guennebaud, Loïc Barthe, and Mathias Paulin. High-quality adaptive soft shadow mapping. *Computer Graphics Forum, Proc. EUROGRAPHICS*, 26(3):525–533, 2007.

[Hai01] Eric Haines. Soft planar shadows using plateaus. *Journal of Graphics Tools*, 6(1):19–27, 2001.

[HAM07] Jon Hasselgren and Thomas Akenine-Möller. Textured shadow volumes. *Journal of Graphics Tools*, 12(4):59–72, 2007.

[HAMO05] Jon Hasselgren, Tomas Akenine-Möller, and Lennart Ohlsson. *GPU Gems 2 - Programming Techniques for High-Performance Graphics and General-Purpose Computation*, chapter Conservative Rasterization, pages 677–694. Addison Wesley, 2005.

[HBS00] Wolfgang Heidrich, Stefan Brabec, and Hans-Peter Seidel. Soft shadow maps for linear lights. In *Proc. EG Workshop on Rendering Techniques*, pages 269–280. Eurographics, 2000.

[Hec89] Paul Heckbert. Fundamentals of texture mapping and image warping. Master's thesis, 1989.

[Hei91] Tim Heidmann. Real shadows, real time. *Iris Universe*, 18:28–31, 1991.

[Her97] Michael Herf. Efficient generation of soft shadow textures. Technical Report CMU-CS-97-138, Carnegie Mellon University, 1997.

[HH97] Paul S. Heckbert and Michael Herf. Simulating soft shadows with graphics hardware. Technical Report CMU-CS-97-104, Jan. 1997.

[HHLH05] Samuel Hornus, Jared Hoberock, Sylvain Lefebvre, and John Hart. Zp+: correct z-pass stencil shadows. In *I3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games*, pages 195–202. ACM Press, 2005.

[HLHS03] Jean-Marc Hasenfratz, Marc Lapierre, Nicolas Holzschuch, and François Sillion. A survey of real-time soft shadows algorithms. In *State-of-the-Art Report, Proc. EUROGRAPHICS*. Eurographics, 2003.

[JMB04] Gregory S. Johnson, William R. Mark, and Christopher A. Burns. The irregular z-buffer and its application to shadow mapping. Technical report, University of Texas, Austin, 2004.

[Kaj86] James T. Kajiya. The rendering equation. In *Proc. SIGGRAPH*, pages 143–150. ACM Press, 1986.

[Ket99] Lutz Kettner. Using generic programming for designing a data structure for polyhedral surfaces. *Computational Geometry: Theory and Applications*, 13(1):65–90, 1999.

[KH01] Alexander Keller and Wolfgang Heidrich. Interleaved sampling. In *Proceedings of the 12th Eurographics Workshop on Rendering Techniques*, pages 269–276. Eurographics, 2001.

[Kil05] Mark J. Kilgard. OpenGL extension: EXT_stencil_two_side. `http://www.opengl.org/registry/specs/EXT/stencil_two_side.txt`, 2005.

[KM99] Brett Keating and Nelson Max. Shadow penumbras for complex objects by depth-dependent filtering of multi-layer depth images. In *Proceedings of the 10th Eurographics Workshop on Rendering*, pages 205–220. Springer-Verlag, 1999.

[KN01] Tae-Yong Kim and Ulrich Neumann. Opacity shadow maps. In *Proc. EG Workshop on Rendering Techniques*, pages 177–182, 2001.

[Koz04] Simon Kozlov. *GPU Gems - Programming Techniques, Tips, and Tricks for Real-Time Graphics*, chapter Perspective Shadow Maps: Care and Feeling, pages 217–244. Addison Wesley, 2004.

[LAA+05] Samuli Laine, Timo Aila, Ulf Assarsson, Jaakko Lehtinen, and Tomas Akenine-Möller. Soft shadow volumes for ray tracing. volume 24, pages 1156–1165. ACM Press, 2005.

[Lai05] Samuli Laine. Split-plane shadow volumes. In *Proc. SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 23–32. Eurographics, 2005.

[Lau07] Andrew Lauritzen. *GPU Gems 3*, chapter Summed-Area Variance Shadow Maps, pages 157–182. Addison Wesley, 2007.

[Leh04] Jaakko Lehtinen. Foundations of precomputed radiance transfer. Master's thesis, 2004.

[Len02] Eric Lengyel. The mechanics of robust stencil shadows. Gamasutra website, `http://www.gamasutra.com/features/20021011/lengyel_01.htm`, 2002.

[Len05] Eric Lengyel. Advanced stencil shadow and penumbra wedge rendering. Game developer Conference, slides, `www.terathon.com/gdc_lengyel.ppt`, 2005.

[LGMM07] Brandon Lloyd, Naga K. Govindaraju, Steven E. Molnar, and Dinesh Manocha. Practical logarithmic rasterization for low-error shadow maps. In *Proc. SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 17–24, 2007.

[LGT+06] Brandon Lloyd, Naga K. Govindaraju, David Tuft, Steve Molnar, and Dinesh Manocha. Practical logarithmic shadow maps. In *ACM SIGGRAPH Sketches*, page 103. ACM Press, 2006.

[LLA06] Jaakko Lehtinen, Samuli Laine, and Timo Aila. An improved physically-based soft shadow volume algorithm. *Computer Graphics Forum, Proc. EUROGRAPHICS*, 25(3):303–312, 2006.

[Llo07] D. Brandon Lloyd. *Logarithmic perspective shadow maps.* PhD thesis, Chapel Hill, NC, USA, 2007. Adviser-Dinesh Manocha.

[LM08] Andrew Lauritzen and Michael McCool. Layered variance shadow maps. In *Proc. of Graphics Interface*, pages 139–146, 2008.

[LTYM06] Brandon Lloyd, David Tuft, Sung-eui Yoon, and Dinesh Manocha. Warping and partitioning for low error shadow maps. In *Proc. EG Symposium on Rendering*, pages 215–226. Eurographics, 2006.

[LV00] Tom Lokovic and Eric Veach. Deep shadow maps. In *Proc. SIGGRAPH*, pages 385–392. ACM Press, 2000.

[LWGM04] Brandon Lloyd, Jeremy Wendt, Naga Govindaraju, and Dinesh Manocha. Cc shadow volumes. In *SIGGRAPH Sketches*, page 146. ACM Press, 2004.

[McC00] Michael D. McCool. Shadow volume reconstruction from depth maps. *ACM Transactions on Graphics*, 19(1):1–26, 2000.

[MHE+03] M. McGuire, J. Hughes, K. Egan, M Killgard, and C. Everitt. Fast, pratictical and robust shadows. Technical report, NVIDIA Cooperation, 2003.

[mi] mental images. mental ray renderer. `http://www.mentalimages.com/products/mental-ray.html`.

[Mic08] Microsoft. Direct3D 9. SDK documentation - `http://msdn.microsoft.com/en-us/library/bb219837%28VS.85%29.aspx`, 2008.

[Mit07] Martin Mittring. Finding next gen: Cryengine 2. In *SIGGRAPH courses*, pages 97–121. ACM Press, 2007.

[MSW04] Chunhui Mei, Jiaoying Shi, and Fuli Wu. Rendering with spherical radiance transport maps". *Computer Graphics Forum*, 23(3):281–290, 2004.

[MT97] Tomas Möller and Ben Trumbore. Fast, minimum storage ray-triangle intersection. *Journal of Graphics Tools*, 2(1):21–28, 1997.

[MT04] Tobias Martin and Tiow-Seng Tan. Anti-aliasing and continuity with trapezoidal shadow maps. In *Proc. EG Symposium on Rendering*, pages 153–160. Eurographics, 2004.

[Nie92] Harald Niederreiter. *Random number generation and quasi-Monte Carlo methods*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1992.

[NRH04] Ren Ng, Ravi Ramamoorthi, and Pat Hanrahan. Triple product wavelet integrals for all-frequency relighting. In *Proc. SIGGRAPH*, pages 477–487. ACM Press, 2004.

[NVi05] NVidia. NVidia GPU Programming Guide. `http://developer.download.nvidia.com/GPU_Programming_Guide/GPU_Programming_Guide.pdf`, 2005.

[NVi08] NVidia. NVIDIA CUDA, Compute Unified Device Architecture. Programming guide Version 2.0 - `http://developer.download.nvidia.com/compute/cuda/2.0-Beta2/docs/Programming_Guide_2.0beta2.pdf`, 2008.

[PH04] Matt Pharr and Greg Humphreys. *Physically Based Rendering: From Theory to Practice*, chapter Monte Carlo Integration I: Basic Concepts, pages 631–660. Morgan Kaufmann, 2004.

[PMDS06] Voicu Popescu, Chunhui Mei, Jordan Dauble, and Elisha Sacks. Reflected-scene impostors for realistic reflections at interactive rates. *Computer Graphics Forum, Proc. EUROGRAPHICS*, 25(3):313–322, 2006.

[PSS98] Steven Parker, Peter Shirley, and Brian Smits. Single sample soft shadows. Technical Report UUCS-98-019, 1998.

[RGKM07] Tobias Ritschel, Thorsten Grosch, Jan Kautz, and Stefan Muller. Interactive illumination with coherent shadow maps. In *Proc. EG Symposium on Rendering*, volume 18, pages 61–72. Eurographics, 2007.

[RH01] Ravi Ramamoorthi and Pat Hanrahan. An efficient representation for irradiance environment maps. In *Proc. SIGGRAPH*, pages 497–500. ACM Press, 2001.

[RHCB05] Zhong Ren, Wei Hua, Lu Chen, and Hujun Bao. Intersection fields for interactive global illumination. *The Visual Computer*, 21(8-10):569–578, 2005.

[Ris07] Eric Risser. *GPU Gems 3*, chapter True Imposters, pages 481–490. Addison Wesley, 2007.

[RSC87] William T. Reeves, David H. Salesin, and Robert L. Cook. Rendering antialiased shadows with depth maps. In *Proc. SIGGRAPH*, volume 21, pages 283–291. ACM Press, 1987.

[RT06] Guodong Rong and Tiow-Seng Tan. Utilizing jump flooding in image-based soft shadows. In *In Proc. Symposium on Virtual Reality Software and Technology*, pages 173–180. ACM, 2006.

[RWS+06] Zhong Ren, Rui Wang, John Snyder, Kun Zhou, Xinguo Liu, Bo Sun, Peter-Pike Sloan, Hujun Bao, Qunsheng Peng, and Baining Guo. Real-time soft shadows in dynamic scenes using spherical harmonic exponentiation. *ACM Transactions on Graphics*, 25(3):977–986, 2006.

[SA06] Mark Segal and Kurt Akeley. The OpenGL ® Graphics System: A Specification. `http://www.opengl.org/registry/doc/glspec21.20061201.pdf`, 2006.

[SA07] Perumaal Shanmugam and Okan Arikan. Hardware accelerated ambient occlusion techniques on gpus. In *SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pages 73–80. ACM Press, 2007.

[Sal08] Marco Salvi. *ShaderX^6 - Advanced Rendering Techniques*, chapter Rendering Filtered Shadows with Exponental Shadow Maps, pages 257–274. Charles River Media, 2008.

[SCS+08] Larry Seiler, Doug Carmean, Eric Sprangle, Tom Forsyth, Michael Abrash, Pradeep Dubey, Stephen Junkins, Adam Lake, Jeremy Sugerman, Robert Cavin, Roger Espasa, Ed Grochowski, Toni Juan, and Pat Hanrahan. Larrabee: a many-core x86 architecture for visual computing. *ACM Transactions on Graphics, Proc. SIGGRAPH*, 27:1–15, 2008.

[SD02] Marc Stamminger and George Drettakis. Perspective shadow maps. In *Proc. SIGGRAPH*, pages 557–562. ACM Press, 2002.

[SEA08] Erik Sintorn, Elmar Eisemann, and Ulf Assarsson. Sample based visibility for soft shadows using alias-free shadow maps. In *Proc. EG Symposium on Rendering*, page to appear. Eurographics, 2008.

[SIMP06] Benjamin Segovia, Jean-Claude Iehl, Richard Mitanchey, and Bernard Péroche. Non-interleaved deferred shading of interleaved sample patterns. In *Proc. SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 53–60. Eurographics, 2006.

[SKS02] Peter-Pike Sloan, Jan Kautz, and John Snyder. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. *ACM Transactions on Graphics*, 21(3):527–536, 2002.

[SKvW+92] Mark Segal, Carl Korobkin, Rolf van Widenfelt, Jim Foran, and Paul Haeberli. Fast shadows and lighting effects using texture mapping. *Proc. SIGGRAPH*, 26(2):249–252, 1992.

[SLSS03] Peter-Pike Sloan, Xinguo Liu, Heung-Yeung Shum, and John Snyder. Bi-scale radiance transfer. In *Proc. SIGGRAPH*, pages 370–375. ACM Press, 2003.

[SM06] Weifeng Sun and Amar Mukherjee. Generalized wavelet product integral for rendering dynamic glossy objects. In *Proc. SIGGRAPH*, pages 955–966. ACM Press, 2006.

[SN08] John Snyder and Derek Nowrouzezahrai. Fast soft self-shadowing on dynamic height fileds. In *Proc. EG Symposium on Rendering*, page to appear. Eurographics, 2008.

[SS] Michael Schwarz and Marc Stamminger. Quality scalability of soft shadow mapping. In *Proc. of Graphics Interface*.

[SS98] Cyril Soler and François Sillion. Fast calculation of soft shadow textures using convolution. In *Proc. SIGGRAPH*, pages 321–332. ACM Press, 1998.

[SS07] Michael Schwarz and Marc Stamminger. Bitmask soft shadow. *Computer Graphics Forum, Proc. EUROGRAPHICS*, 26(3):515–524, 2007.

[SS08] Michael Schwarz and Marc Stamminger. Microquad soft shadow mapping revisited. In *EUROGRAPHICS short papers*, pages 295–298, 2008.

[TQJN99] Katsumi Tadamura, Xueying Qin, Guofang Jiao, and Eihachiro Nakamae. Rendering optimal solar shadows using plural sunlight depth buffers. *Computer Graphics International*, pages 166–173, 1999.

[Ura05] Yury Uralsky. *GPU Gems 2 - Programming Techniques for High-Performance Graphics and General-Purpose Computation*, chapter Efficient Soft-Edged Shadows Using Pixel Shader Branching, pages 269–282. Addison Wesley, 2005.

[VG95] Eric Veach and Leonidas J. Guibas. Optimally combining sampling techniques for monte carlo rendering. In *Proc. SIGGRAPH*, pages 419–428. ACM Press, 1995.

[WH03] Chris Wyman and Charles Hansen. Penumbra maps: approximate soft shadows in real-time. In *Proc. EG Workshop on Rendering Techniques*, pages 202–207. Eurographics, 2003.

[Wil78] Lance Williams. Casting curved shadows on curved surfaces. In *Proc. SIGGRAPH*, pages 270–274. ACM Press, 1978.

[Wil83] Lance Williams. Pyramidal parametrics. *Proc. SIGGRAPH*, 17(3):1–11, 1983.

[WSP04] Michael Wimmer, Daniel Scherzer, and Werner Purgathofer. Light space perspective shadow maps. In *Proc. EG Symposium on Rendering*, pages 143–151. Eurographics, 2004.

[YTD02] Zhengming Ying, Min Tang, and Jinxiang Dong. Soft shadow maps for area light by area approximation. In *In Proc. Pacific Graphics*, page 442, Washington, DC, USA, 2002. IEEE Computer Society.

[ZHL+05] Kun Zhou, Yaohua Hu, Stephen Lin, Baining Guo, and Heung-Yeung Shum. Precomputed shadow fields for dynamic scenes. *ACM Transactions on Graphics*, 24(3):1196–1201, 2005.

[ZSN07] Fan Zhang, Hanqiu Sun, and Oskari Nyman. *GPU Gems 3*, chapter Parallel-Split Shadow Maps on Programmable GPUs, pages 203–238. Addison Wesley, 2007.

Abstract

Direct shadow algorithms generate shadows by simulating the direct lighting interaction in a virtual environment. The main challenge with the accurate direct shadow problematic is its computational cost. In this dissertation, we develop a new robust object-based shadow framework that provides realistic shadows at interactive frame rate on dynamic scenes. Our contributions include new robust object-based soft shadow algorithms and efficient interactive implementations.

We start, by formalizing the direct shadow problematic. Following the light transport problematic, we first formalize what are robust direct shadows. We then study existing interactive direct shadow techniques and outline that the real time direct shadow simulation remains an open problem. We show that even the so called physically plausible soft shadow algorithms still rely on approximations. Nevertheless we exhibit that, despite their geometric constraints, object-based approaches seems well suited when targeting accurate solutions.

Starting from the previous analyze, we investigate the existing object-based shadow framework and discuss about its robustness issues. We propose a new technique that drastically improve the resulting shadow quality by improving this framework with a penumbra blending stage. We present a practical implementation of this approach. From the obtained results, we outline that, despite desirable properties, the inherent theoretical and implementation limitations reduce the overall quality and performances of the proposed algorithm.

We then present a new object-based soft shadow algorithm. It merges the efficiency of the real time object-based shadows with the accuracy of its offline generalization. The proposed algorithm lies onto a new local evaluation of the number of occluders between *two* points (*i.e.* the depth complexity). We describe how we use this algorithm to sample the depth complexity between any visible receiver and the light source. From this information, we compute shadows by either modulate the direct lighting or numerically solve the direct illumination with an accuracy depending on the light sampling strategy. We then propose an extension of our algorithm in order to handle shadows cast by semi opaque occluders. We finally present an efficient implementation of this framework that demonstrates that object-based shadows can be efficiently used on complex dynamic environments.

In real time rendering, it is common to represent highly detailed objects with few trian-

gles and transmittance textures that encode their binary opacity. Object-based techniques do not handle such perforated triangles. Due to their nature, they can only evaluate the shadows cast by models whose their shape is explicitly defined by geometric primitives. We describe a new robust object-based algorithm that addresses this main limitation. We outline that this method can be efficiently combine with object-based frameworks in order to evaluate approximative shadows or simulate the direct illumination for both common meshes and perforated triangles. The proposed implementation shows that such combination provides a very strong and efficient direct lighting framework, well suited to many domains ranging from quality sensitive to performance critical applications.