Volume 22 (2003), number 4 pp. 753–774 COMPUTER GRAPHICS *forum*

A Survey of Real-time Soft Shadows Algorithms

J. -M. Hasenfratz[†], M. Lapierre[‡], N. Holzschuch[§] and F. Sillion[§]

Artis GRAVIR/IMAG-INRIA**

Abstract

Recent advances in GPU technology have produced a shift in focus for real-time rendering applications, whereby improvements in image quality are sought in addition to raw polygon display performance. Rendering effects such as antialiasing, motion blur and shadow casting are becoming commonplace and will likely be considered indispensable in the near future. The last complete and famous survey on shadow algorithms — by Woo et al. *[52] in 1990 — has to be updated in particular in view of recent improvements in graphics hardware, which make new algorithms possible. This paper covers all current methods for real-time shadow rendering, without venturing into slower, high quality techniques based on ray casting or radiosity. Shadows are useful for a variety of reasons: first, they help understand relative object placement in a 3D scene by providing visual cues. Second, they dramatically improve image realism and allow the creation of complex lighting ambiances. Depending on the application, the emphasis is placed on a guaranteed framerate, or on the visual quality of the shadows including penumbra effects or "soft shadows". Obviously no single method can render physically correct soft shadows in real time for any dynamic scene! However our survey aims at providing an exhaustive study allowing a programmer to choose the best compromise for his/her needs. In particular we discuss the advantages, limitations, rendering quality and cost of each algorithm. Recommendations are included based on simple characteristics of the application such as static/moving lights, single or multiple light sources, static/dynamic geometry, geometric complexity, directed or omnidirectional lights, etc. Finally we indicate which methods can efficiently exploit the most recent graphics hardware facilities.*

Keywords: shadow algorithms, soft shadows, real-time, shadow mapping, shadow volume algorithm.

ACM CSS: I.3.3 Computer Graphics Picture/Image Generation—*Bitmap and framebuffer operations*

1. Introduction

Cast shadows are crucial for the human perception of the 3D world. Probably the first thorough analysis of shadows was Leonardo Da Vinci's [48] (see Figure 1), focusing on paintings and static images. Also of note is the work of Lambert [35] who described the geometry underlying cast shadows (see Figure 1), and more recently the paper from Knill *et al.* [34].

[†] University Pierre Mendès France – Grenoble II
[‡] University Joseph Fourier – Grenoble I
[§] INRIA
**Artis is a team of the GRAVIR/IMAG laboratory, a joint research unit of CNRS, INPG, INRIA, UJF.

With the emergence of computer graphics technology, researchers have developed experiments to understand the impact of shadows on our perception of a scene. Through different psychophysical experiments they established the important role of shadows in understanding:

- the position and size of the occluder [27,30,31,38,49];
- the geometry of the occluder [38];
- the geometry of the receiver [38].

Wanger [49] studied the effect of shadow quality on the perception of object relationships, basing his experiments on shadow sharpness. Hubona *et al.* [27] discuss the general role and effectiveness of object shadows in 3D visualization.

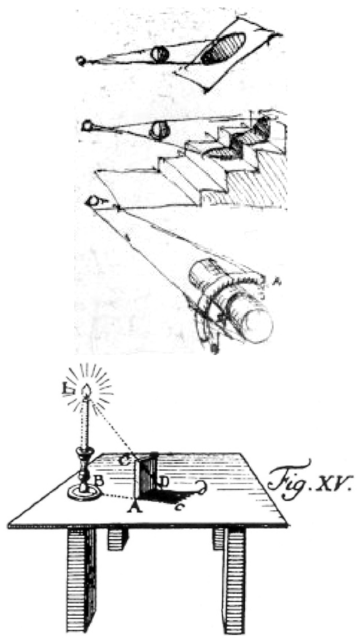


Figure 1: *Left: Study of shadows by Leonardo da Vinci [48] — Right: Shadow construction by Lambert [35].*

In their experiments, they put in competition shadows, viewing mode (mono/stereo), number of lights (one/two), and background type (flat plane, "stair-step" plane, room) to measure the impact of shadows.

Kersten *et al.* [30,31] and Mamassian *et al.* [38] study the relationship between object motion and the perception of relative depth. In fact, they demonstrate that simply adjusting the motion of a shadow is sufficient to induce dramatically different apparent trajectories of the shadow-casting object.

These psychophysical experiments convincingly establish that it is important to take shadows into account to produce images in computer graphics applications. Cast shadows help in our understanding of 3D environments and soft shadows take part in realism of the images.

Since the comprehensive survey of Woo *et al.* [52], progress in computer graphics technology and the development of consumer-grade graphics accelerators have made real-time 3D graphics a reality [3]. However incorporating shadows, and especially realistic soft shadows, in a real-time application, has remained a difficult task (and has generated a great research effort). This paper presents a survey of shadow generation techniques that can create soft shadows in real time. Naturally the very notion of "real-time performance" is difficult to define, suffice it to say that we are concerned with the display of 3D scenes of significant complexity (several tens of thousands of polygons) on consumer-level hardware *ca.* 2003. The paper is organized as follows:

We first review in Section 2 basic notions about shadows: hard and soft shadows, the importance of shadow effects showing problems encountered when working with soft shadows and classical techniques for producing hard shadows in real time. Section 3 then presents existing algorithms for producing soft shadows in real time. Section 4 offers a discussion and classifies these algorithms based on their different abilities and limitations, allowing easier algorithm selection depending on the application's constraints.

2. Basic concepts of hard and soft shadows

2.1. What is a shadow?

Consider a light source L illuminating a scene: *receivers* are objects of the scene that are potentially illuminated by L. A point P of the scene is considered to be in the *umbra* if it can not see any part of L, *i.e.* it does not receive any light directly from the light source.

If P can see a part of the light source, it is in the *penumbra*. The union of the umbra and the penumbra is the shadow, the region of space for which at least one point of the light source is occluded. Objects that hide a point from the light source are called *occluders*.

We distinguish between two types of shadows:

attached shadows, occuring when the normal of the receiver is facing away from the light source;

cast shadows, occuring when a shadow falls on an object whose normal is facing toward the light source.

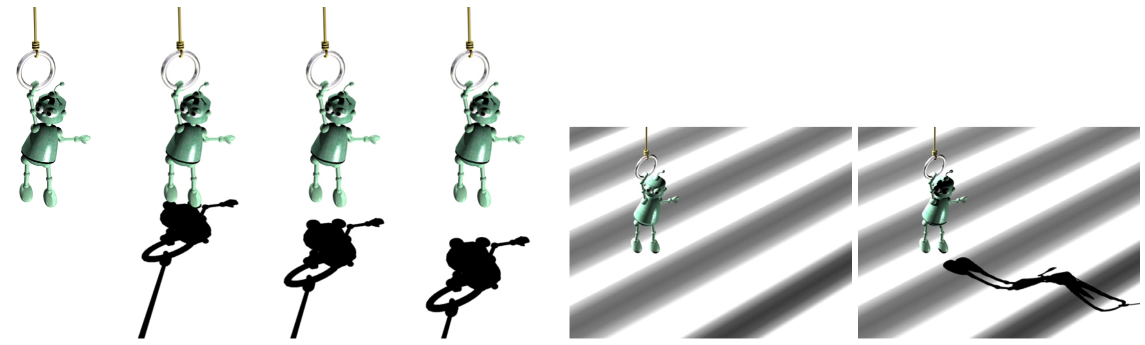
Self-shadows are a specific case of cast shadows that occur when the shadow of an object is projected onto itself, *i.e.* the occluder and the receiver are the same.

Attached shadows are easy to handle. We shall see later, in Section 4, that some algorithms cannot handle self-shadows.

2.2. Importance of shadow effects

As discussed in the introduction, shadows play an important role in our understanding of 3D geometry:

- Shadows help to **understand relative object position and size** in a scene [27,30,31,38,49]. For example, without a cast shadow, we are not able to determine the position of an object in space (see Figure 2(a)).
- Shadows can also help us **understanding the geometry of a complex receiver** [38] (see Figure 2(b)).
- Finally, shadows provide useful visual cues that help in **understanding the geometry of a complex occluder** [38] (see Figure 3).



(a) Shadows provide information about the relative positions of objects. On the left-hand image, we cannot determine the position of the robot, whereas on the other three images we understand that it is more and more distant from the ground.

(b) Shadows provide information about the geometry of the receiver. Left: not enough cues about the ground. Right: shadow reveals ground geometry.

Figure 2: *Shadows play an important role in our understanding of 3D geometry.*

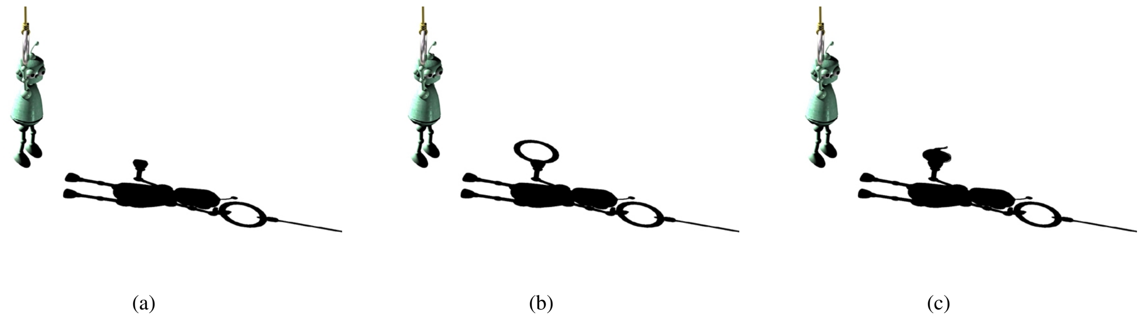


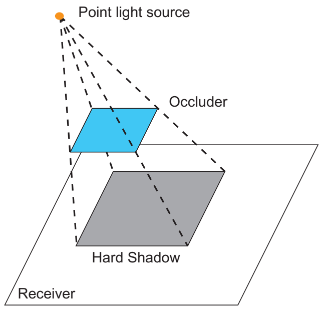
Figure 3: *Shadows provide information about the geometry of the occluder. Here we see that the robot holds nothing in his left hand on Figure 3(a), a ring on Figure 3(b) and a teapot on Figure 3(c).*

2.3. Hard shadows *vs.* soft shadows

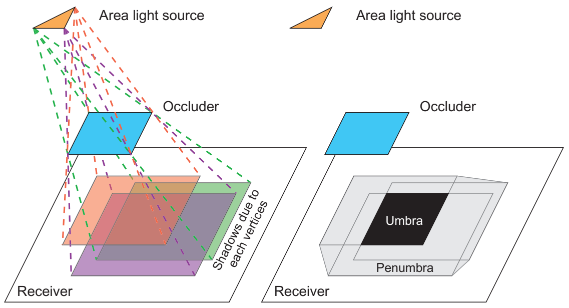
The common-sense notion of shadow is a binary status, *i.e.* a point is either "in shadow" or not. This corresponds to *hard shadows*, as produced by point light sources: indeed, a point light source is either visible or occluded from any receiving point. However, point light sources do not exist in practice and hard shadows give a rather unrealistic feeling to images (see Figure 4(c)). Note that even the sun, probably the most common shadow-creating light source in our daily life, has a significant angular extent and does not create hard shadows. Still, point light sources are easy to model in computer graphics and we shall see that several algorithms let us compute hard shadows in real time.

In the more realistic case of a light source with finite extent, a point on the receiver can have a partial view of the light, *i.e.* only a fraction of the light source is visible from that point. We distinguish the *umbra* region (if it exists) in which the light source is totally blocked from the receiver, and the *penumbra* region in which the light source is partially visible. The determination of the umbra and penumbra is a difficult task in general, as it amounts to solving visibility relationships in 3D, a notoriously hard problem. In the case of polygonal objects, the shape of the umbra and penumbra regions is embedded in a discontinuity mesh [13] which can be constructed from the edges and vertices of the light source and the occluders (see Figure 4(b)).

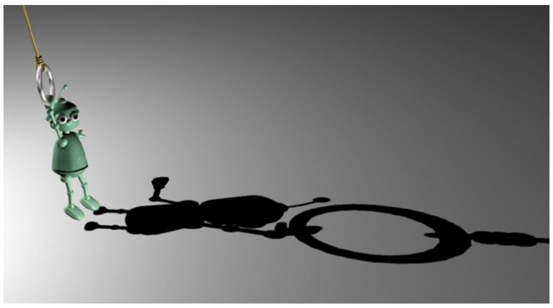
Soft shadows are obviously much more realistic than hard shadows (see Figures 4(c) and 4(d)); in particular the degree of softness (blur) in the shadow varies dramatically with the distances involved between the source, occluder, and receiver. Note also that a hard shadow, with its crisp



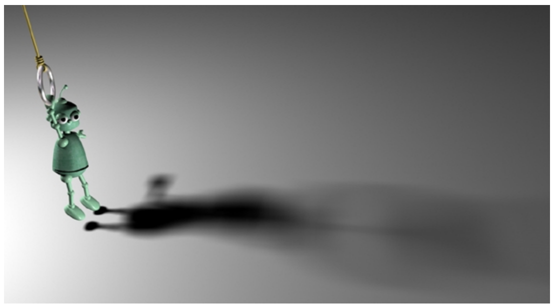
(a) Geometry of hard shadows



(b) Geometry of soft shadows



(c) Illustration of hard shadows



(d) Illustration of soft shadows

Figure 4: *Hard vs. soft shadows.*

boundary, could be mistakenly perceived as an object in the scene, while this would hardly happen with a soft shadow.

In computer graphics we can approximate small or distant light source as point sources only when the distance from the light to the occluder is much larger than the distance from the occluder to the receiver, and the resolution of the final image does not allow proper rendering of the penumbra. In all other cases great benefits can be expected from properly representing soft shadows.

2.4. Important issues in computing soft shadows

2.4.1. Composition of multiple shadows

While the creation of a shadow is easily described for a (light source, occluder, receiver) triple, care must be taken to allow for more complex situations.

Shadows from several light sources: Shadows produced by multiple light sources are relatively easy to obtain if we know how to deal with a single source (see Figure 5). Due to the linear nature of light transfer we simply sum the contribution of each light (for each wavelength or color band).

Shadows from several objects: For point light sources, shadows due to different occluders can be easily combined since the shadow area (where the light source is invisible) is the union of all individual shadows.

With an area light source, combining the shadows of several occluders is more complicated. Recall that the lighting contribution of the light source on the receiver involves a partial visibility function: a major issue is that no simple combination of the partial visibility functions of distinct occluders can yield the partial visibility function of the set of occluders considered together. For instance there may be points in the scene where the light source is not occluded by any object taken separately, but is totally occluded by the set of objects taken together. The correlation between the partial visibility functions of different occluders cannot be predicted easily, but can sometimes be approximated or bounded [5,45].

As a consequence, the shadow of the union of the objects can be larger than the union of the shadows of the objects (see Figure 6). This effect is quite real, but is not very visible on typical scenes, especially if the objects casting shadows are animated.

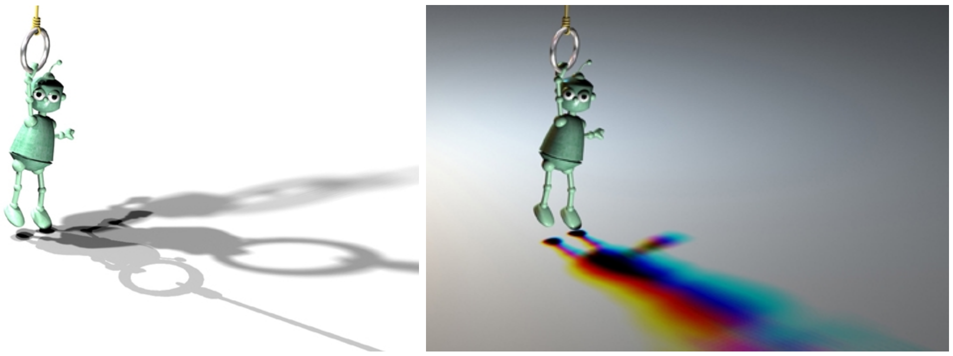


Figure 5: *Complex shadow due to multiple light sources. Note the complex interplay of colored lights and shadows in the complementary colors.*

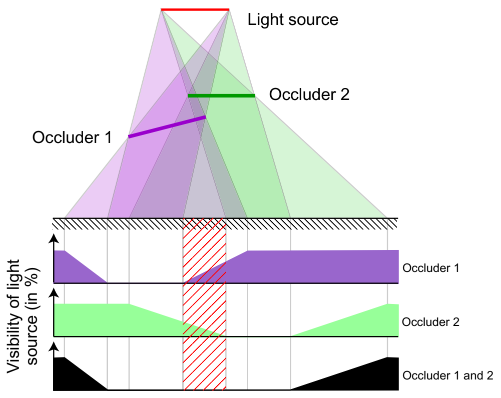


Figure 6: *The shadow of two occluders is not a simple combination of the two individual shadows. Note in particular the highlighted central region which lies in complete shadow (umbra) although the light source is never blocked by a single occluder.*

2.4.2. Physically exact or fake shadows

Shadows from an extended light source: Soft shadows come from spatially extended light sources. To model properly the shadow cast by such light sources, we must take into account all the parts of the occluder that block light coming from the light source. This requires identifying all parts of the object casting shadow that are visible from at least one point of the extended light source, which is algorithmically much more complicated than identifying parts of the occluder that are visible from a single point.

Because this visibility information is much more difficult to compute with extended light sources than with point light sources, most real-time soft shadow algorithms compute visibility information from just one point (usually the center of the light source) and then simulate the behavior of the extended light source using this visibility information (computed for a point).

This method produces shadows that are not physically exact, of course, but can be close enough to real shadows for most practical applications. The difference between the approximation and the real shadow is harder to notice if the objects and their shadow are animated — a common occurrence in real-time algorithms.

The difference becomes more noticeable if the difference between the actual extended light source and the point used for the approximation is large, as seen from the object casting shadow. A common example is for a large light source, close enough from the object casting shadow that points of the light source are actually seeing different sides of the object (see Figure 7). In that case, the physically exact shadow is very different from the approximated version.

While large light sources are not frequent in real-time algorithms, the same problem also occurs if the object casting shadow is extended along the axis of the light source, *e.g.* a character with elongated arms whose right arm is pointing toward light source, and whose left arm is close to the receiver.

In such a configuration, if we want to compute a better looking shadow, we can either:

- Use the complete extension of the light source for visibility computations. This is algorithmically too complicated to be used in real-time algorithms.
- Separate the light source into smaller light sources [5,24]. This removes some of the artefacts, since each light source is treated separately, and is geometrically closer to the point sample used to compute the silhouette. The speed of the algorithm is usually divided by the number of light sources.
- Cut the object into slices [45]. We then compute soft shadows separately for each slice, and combine these

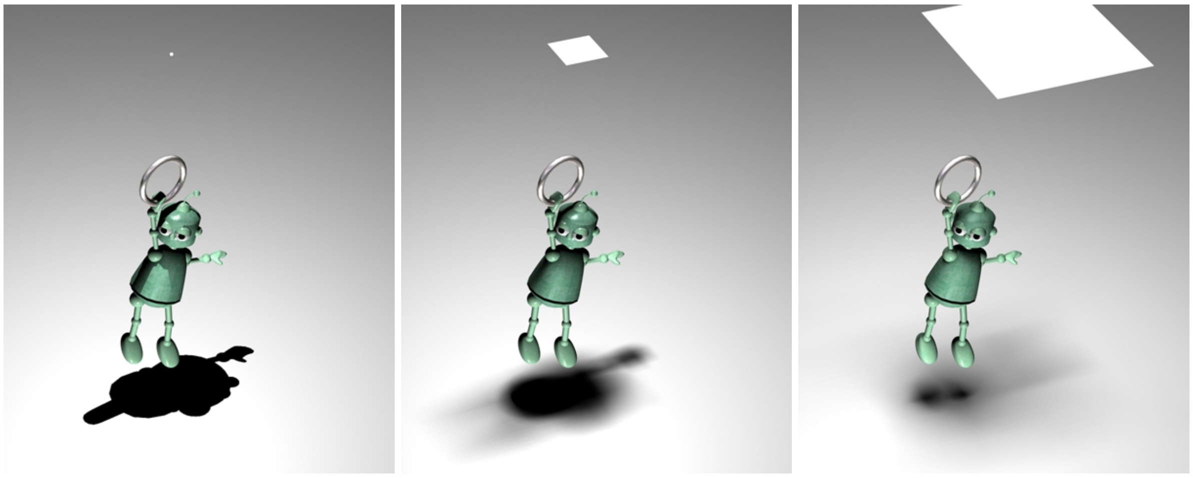


Figure 7: *When the light source is significantly larger than the occluder, the shape of the shadow is very different from the shape computed using a single sample; the sides of the object are playing a part in the shadowing.*

shadows. By slicing the object, we are removing some of the visibility problems, and we allow lower parts of the object — usually hidden by upper parts — to cast shadow. The speed of the algorithm is divided by the number of slices, and combining the shadows cast by different slices remains a difficult problem.

Approximating the penumbra region: When real-time soft shadow algorithms approximate extended light sources using points, they are in fact computing a hard shadow, and extending it to compute a soft shadow.

There are several possible algorithms:

- extend the umbra region outwards, by computing an *outer penumbra* region,
- shrink the umbra region, and complete it with an *inner penumbra* region,
- compute both inner penumbra and outer penumbra.

The first method (outer penumbra only) will always create shadows made of an umbra and a penumbra. Objects will have an umbra, even if the light source is very large with respect to the occluders. This effect is quite noticeable, as it makes the scene appear much darker than anticipated, except for very small light sources.

On the other hand, computing the inner penumbra region can result in light leaks between neighboring objects whose shadows overlap.

Illumination in the umbra region: An important question is the illumination in regions that are in the umbra — completely hidden from the light source. There is no light reaching theses regions, so they should appear entirely black, in theory.

However, in practice, some form of ambient lighting is used to avoid completely dark regions and to simulate the fact that light eventually reaches these regions after several reflections.

Real-time shadow methods are usually combined with illumination computations, for instance using the simple OpenGL lighting model. Depending on whether the shadow method operates before or after the illumination phase, ambient lighting will be present or absent. In the latter case the shadow region appears completely dark, an effect that can be noticeable. A solution is to add the ambient shading as a subsequent pass; this extra pass slows down the algorithm, but clever re-use of the Z-buffer on recent graphics hardware make the added cost manageable [40].

Shadows from different objects: As shown in Section 2.4.1, in presence of extended light sources, the shadow of the union of several objects is larger than the union of the individual shadows. Furthermore, the boundary of the shadow caused by the combination of several polygonal objects can be a curved line [13].

Since these effects are linked with the fact that the light source is extended, they can not appear in algorithms that use a single point to compute surfaces visible from the light source. All real-time soft shadow algorithms therefore suffer from this approximation.

However, while these effects are both clearly identifiable on still images, they are not as visible in animated scenes. There is currently no way to model these effects with real-time soft shadow algorithms.

2.4.3. Real-time

Our focus in this paper is on real-time applications, therefore we have chosen to ignore all techniques that are based on an expensive pre-process even when they allow later modifications at interactive rates [37]. Given the fast evolution of graphics hardware, it is difficult to draw a hard distinction between real-time and interactive methods, and we consider here that frame rates in excess of 10 fps, for a significant number of polygons, are an absolute requirement for "real-time" applications. Note that stereo viewing usually require double this performance.

For real-time applications, the display refresh rate is often the crucial limiting factor, and must be kept high enough (if not constant) through time. An important feature to be considered in shadowing algorithms is therefore their ability to guarantee a sustained level of performance. This is of course impossible to do for arbitrary scenes, and a more important property for these algorithms is the ability to parametrically vary the level of performance (typically at the price of greater approximation), which allows an adaptation to the scene's complexity.

2.4.4. Shadows of special objects

Most shadowing algorithms make use of an explicit representation of the object's shapes, either to compute silhouettes of occluders, or to create images and shadow maps. Very complex and volumetric objects such as clouds, hair, grass etc. typically require special treatment.

2.4.5. Constraints on the scene

Shadowing algorithms may place particular constraints on the scene. Examples include the type of object model (techniques that compute a shadow as a texture map typically require a parametric object, if not a polygon), or the necessity/possibility to identify a subset of the scene as occluders or shadow receivers. This latter property is important in adapting the performance of the algorithm to sustain real-time.

2.5. Basic techniques for real-time shadows

In this State of the Art Review, we focus solely on real-time soft shadows algorithms. As a consequence, we will not describe other methods for producing soft shadows, such as radiosity, ray-tracing, Monte-Carlo ray-tracing or photon mapping.

We now describe the two basic techniques for computing shadows from *point light sources*, namely *shadow mapping* and the *shadow volume algorithm*.

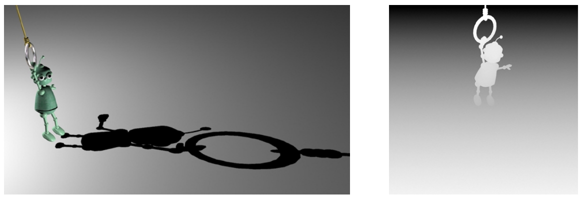


Figure 8: *Shadow map for a point light source. Left: view from the camera. Right: depth buffer computed from the light source.*

2.5.1. Shadow mapping

Method: The basic operation for computing shadows is identifying the parts of the scene that are hidden from the light source. Intrisically, it is equivalent to visible surface determination, from the point-of-view of the light source.

The first method to compute shadows [17,44,50] starts by computing a view of the scene, from the point-of-view of the light source. We store the z values of this image. This Z-buffer is the *shadow map* (see Figure 8).

The shadow map is then used to render the scene (from the normal point-of-view) in a two pass rendering process:

- a standard Z-buffer technique, for hidden-surface removal.
- for each pixel of the scene, we now have the geometrical position of the object seen in this pixel. If the distance between this object and the light is greater than the distance stored in the shadow map, the object is in shadow. Otherwise, it is illuminated.
- The color of the objects is modulated depending on whether they are in shadow or not.

Shadow mapping is implemented in current graphics hardware. It uses an OpenGL extension for the comparison between Z values, `GL_ARB_SHADOW`[†].

Improvements: The depth buffer is sampled at a limited precision. If surfaces are too close from each other, sampling problems can occur, with surfaces shadowing themselves. A possible solution [42] is to offset the Z values in the shadow map by a small bias [51].

If the light source has a cut-off angle that is too large, it is not possible to project the scene in a single shadow map without excessive distortion. In that case, we have to replace

[†] This extension (or the earlier version, `GL_SGIX_SHADOW`), is available on Silicon Graphics Hardware above Infinite Reality 2, on NVidia graphics cards after GeForce3 and on ATI graphics cards after Radeon9500.

the light source by a combination of light sources, and use several depth maps, thus slowing down the algorithm.

Shadow mapping can result in large aliasing problems if the light source is far away from the viewer. In that case, individual pixels from the shadow map are visible, resulting in a staircase effect along the shadow boundary. Several methods have been implemented to solve this problem:

- Storing the ID of objects in the shadow map along with their depth [26].
- Using deep shadow maps, storing coverage information for all depths for each pixel [36].
- Using multi-resolution, adaptative shadow maps [18], computing more details in regions with shadow boundaries that are close to the eye.
- Computing the shadow map in perspective space [46], effectively storing more details in parts of the shadow map that are closer to the eye.

The last two methods are directly compatible with existing OpenGL extensions, and therefore require only a small amount of coding to work with modern graphics hardware.

An interesting alternative version of this algorithm is to warp the shadow map into camera space [55] rather than the usual opposite: it has the advantage that we obtain a modulation image that can be mixed with a texture, or blurred to produce antialiased shadows.

Discussion: Shadow mapping has many advantages:

- it can be implemented entirely using graphics hardware;
- creating the shadow map is relatively fast, although it still depends on the number and complexity of the occluders;
- it handles self-shadowing.

It also has several drawbacks:

- it is subject to many sampling and aliasing problems;
- it cannot handle omni-directional light sources;
- at least two rendering passes are required (one from the light source and one from the viewpoint);

2.5.2. The Shadow Volume Algorithm

Another way to think about shadow generation is purely geometrical. This method was first described by Crow [12], and first implemented using graphics hardware by Heidmann [23].

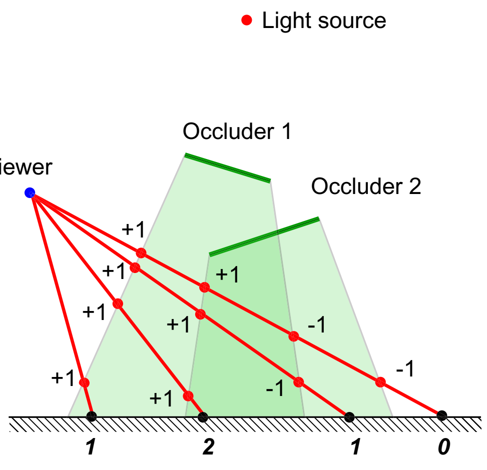


Figure 9: *Shadow volume.*

Method: The algorithm consists in finding the silhouette of occluders along the light direction, then extruding this silhouette along the light direction, thus forming a *shadow volume*. Objects that are inside the shadow volume are in shadow, and objects that are outside are illuminated.

The shadow volume is calculated in two steps:

- the first step consists in finding the silhouette of the occluder as viewed from the light source. The simplest method is to keep edges that are shared by a triangle facing the light and another in the opposite direction. This actually gives a superset of the true silhouette, but it is sufficient for the algorithm.
- then we construct the shadow volume by extruding these edges along the direction of the point light source. For each edge of the silhouette, we build the half-plane subtended by the plane defined by the edge and the light source. All these half-planes define the shadow volume, and knowing if a point is in shadow is then a matter of knowing if it is inside or outside the volume.
- for each pixel in the image rendered, we count the number of faces of the shadow volume that we are crossing between the view point and the object rendered. Front-facing faces of the shadow volume (with respect to the view point) increment the count, back-facing faces decrement the count (see Figure 9). If the total number of faces is positive, then we are inside the shadow volume, and the pixel is rendered using only ambient lighting.

The rendering pass is easily done in hardware using a stencil buffer [15,23,32]; faces of the shadow volume are rendered in the stencil buffer with depth test enabled this way: in a first pass, front faces of the shadow volumes are rendered incrementing the stencil buffer; in a second pass, back faces are rendered, decrementing it. Pixels that are in shadow are "captured" between front and back faces of the shadow volume, and have a positive value in the stencil buffer. This way to render volumes is called *zpass*.

Therefore the complete algorithm to obtain a picture using the Shadow Volume method is:

- render the scene with only ambient/emissive lighting;
- calculate and render shadow volumes in the stencil buffer;
- render the scene illuminated with stencil test enabled: only pixels which stencil value is 0 are rendered, others are not updated, keeping their ambient color.

Improvements: The cost of the algorithm is directly linked to the number of edges in the shadow volume. Batagelo and Júnior [7] minimize the number of volumes rendered by precalculating in software a modified BSP tree. McCool [39] extracts the silhouette by first computing a shadow map, then extracting the discontinuities of the shadow map, but this method requires reading back the depth buffer from the graphics board to the CPU, which is costly. Brabec and Seidel [10] reports a method to compute the silhouette of the occluders using programmable graphics hardware [14], thus obtaining an almost completely hardware-based implementation of the shadow volume algorithm (he still has to read back a buffer into the CPU for parameter transfer).

Roettger *et al.* [43] suggests an implementation that doesn't require the stencil buffer; he draws the shadow volume in the alpha buffer, replacing increment/decrement with a multiply/divide by 2 operation.

Everitt and Kilgard [15] have described a robust implementation of the shadow volume algorithm. Their method includes capping the shadow volume, setting $w = 0$ for extruded vertices (effectively making infinitely long quads) and setting the far plane at an infinite distance (they prove that this step only decreases Z-buffer precision by a few percents). Finally, they render the shadow volume using the *zfail* technique; it works by rendering the shadow volume *backwards*:

- we render the scene, storing the Z-buffer;
- in the first pass, we increment the stencil buffer for all back-facing faces, but only if the face is behind an existing object of the scene;
- in the second pass, we decrement the stencil buffer for all front-facing faces, but only if the face is behind an existing object;
- The stencil buffer contains the intersection of the shadow volume and the objects of the scene.

The *zfail* technique was discovered independently by Bilodeau and Songy and by Carmack.

Recent extensions to OpenGL [15,16,21] allow the use of shadow volumes using stencil buffer in a single pass, instead of the two passes required so far. They also [15] provide *depth-clamping*, a method in which polygon are not clipped at the near and far distance, but their vertices are projected onto the near and far plane. This provides in effect an infinite view pyramid, making the shadow volume algorithm more robust.

The main problem with the shadow volume algorithm is that it requires drawing large polygons, the faces of the shadow volume. The fillrate of the graphics card is often the bottleneck. Everitt and Kilgard [15,16] list different solutions to reduce the fillrate, either using software methods or using the graphics hardware, such as scissoring, constraining the shadow volume to a particular fragment.

Discussion: The shadow volume algorithm has many advantages:

- it works for omnidirectional light sources;
- it renders eye-view pixel precision shadows;
- it handles self-shadowing.

It also has several drawbacks:

- the computation time depends on the complexity of the occluders;
- it requires the computation of the silhouette of the occluders as a preliminary step;
- at least two rendering passes are required;
- rendering the shadow volume consumes fillrate of the graphics card.

3. Soft shadow algorithms

In this section, we review algorithms that produce soft shadows, either interactively or in real time. As in the previous section, we distinguish two types of algorithms:

- Algorithms that are based on an image-based approach, and build upon the shadow map method described in Section 2.5.1. These algorithms are described in Section 3.1.
- Algorithms that are based on an object-based approach, and build upon the shadow volume method described in Section 2.5.2. These algorithms are described in Section 3.2.

3.1. Image-Based Approaches

In this section, we present soft shadow algorithms based on shadow maps (see Section 2.5.1). There are several methods to compute soft shadows using image-based techniques:

1. Combining several shadow textures taken from point samples on the extended light source [22,25].
2. Using layered attenuation maps [1], replacing the shadow map with a Layered Depth Image, storing depth information about all objects visible from at least one point of the light source.
3. Using several shadow maps [24,54], taken from point samples on the light source, and an algorithm to compute the percentage of the light source that is visible.
4. Using a standard shadow map, combined with image analysis techniques to compute soft shadows [9].
5. Convolving a standard shadow map with an image of the light source [45].

The first two methods approximate the light source as a combination of several point samples. As a consequence, the time for computing the shadow textures is multiplied by the number of samples, resulting in significantly slower rendering. On the other hand, these methods actually compute more information than other soft shadow methods, and thus compute more physically accurate shadows. Most of the artefacts listed in Section 2.4.2 will not appear with these two methods.

3.1.1. Combination of several point-based shadow images [22,25]

The simplest method [22,25] to compute soft shadows using image based methods is to place sample points regularly on the extended light source. These sample points are used to compute binary occlusion maps, which are combined into an attenuation map, used to modulate the illumination (calculated separately).

Method: Herf [25] makes the following assumptions on the geometry of the scene:

- a light source of uniform color,
- subtending a small solid angle with respect to the receiver,
- and with distance from the receiver having small variance.

With these three assumptions, contributions from all sample points placed on the light source will be roughly equal.

The user identifies in advance the object casting shadows, and the objects onto which we are casting shadow. For each

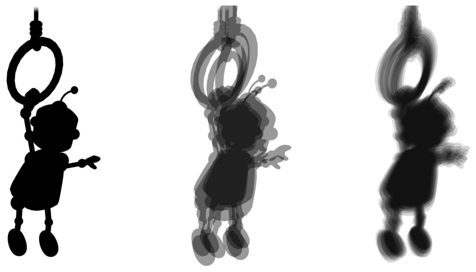


Figure 10: *Combining several occlusion maps to compute soft shadows. Left: the occlusion map computed for a single sample. Center: the attenuation map computed using 4 samples. Right: the attenuation map computed using 64 samples.*

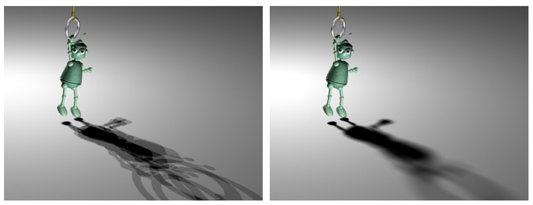


Figure 11: *With only a small number of samples on the light source, artefacts are visible. Left: soft shadow computed using 4 samples. Right: soft shadow computed using 1024 samples.*

object receiving shadow, we are going to compute a texture containing the soft shadow.

We start by computing a binary occlusion map for each sample point on the light source. For each sample point on the light source, we render the scene into an auxiliary buffer, using 0 for the receiver, and 1 for any other polygon. These binary occlusion maps are then combined into an attenuation map, where each pixel stores the number of sample points on the light source that are occluded. This attenuation map contains a precise representation of the soft shadow (see Figures 10 and 11).

In the rendering pass, this soft shadow texture is combined with standard textures and illumination, in a standard graphics pipeline.

Discussion: The biggest problem for Herf [25] method is rendering the attenuation maps. This requires $N_p N_s$ rendering passes, where N_p is the number of objects receiving shadows, and N_s is the number of samples on the light source. Each pass takes a time proportionnal to the number of polygons in the objects casting shadows. In practice, to make this method run in real time, we have to limit the number of receivers to a single planar receiver.

To speed-up computation of the attenuation map, we can lower the number of polygons in the occluders. We can also lower the number of samples (n) to increase the framerate, but this is done at the expense of image quality, as the attenuation map contains only $n-1$ gray levels. With fewer than 9 samples (3×3), the user sees several hard shadows, instead of a single soft shadow (see Figure 11).

Herf's method is easy to parallelize, since all occlusion maps can be computed separately, and only one computer is needed to combine them. Isard *et al.* [28] reports that a parallel implementation of this algorithm on a 9-node Sepia-2a parallel calculator with high-end graphics cards runs at more than 100 fps for moderately complex scenes.

3.1.2. Layered Attenuation Maps [1]

The Layered Attenuation Maps [1] method is based on a modified layered depth image [29]. It is an extension of the previous method, where we compute a layered attenuation map for the entire scene, instead of a specific shadow map for each object receiving shadow.

Method: It starts like the previous method: we place sample points on the area light source, and we use these sample points to compute a modified attenuation map:

- For each sample point, we compute a view of the scene, along the direction of the normal to the light source.
- Theses images are all warped to a central reference, the center of the light source.
- For each pixel of these images:
  - In each view of the scene, we have computed the distance to the light source in the Z-buffer.
  - We can therefore identify the object that is closest to the light source.
  - This object makes the first layer of the layered attenuation map.
  - We count the number of samples seeing this object, which gives us the percentage of occlusion for this object.
  - If other objects are visible for this pixel but further away from the light they make the subsequent layers.
  - For each layer, we store the distance to the light source and the percentage of occlusion.

The computed Layered Attenuation Map contains, for all the objects that are visible from at least one sample point, the distance to the light source and the percentage of sample points seeing this object.

At rendering time, the Layered Attenuation Map is used like a standard attenuation map, with the difference that all

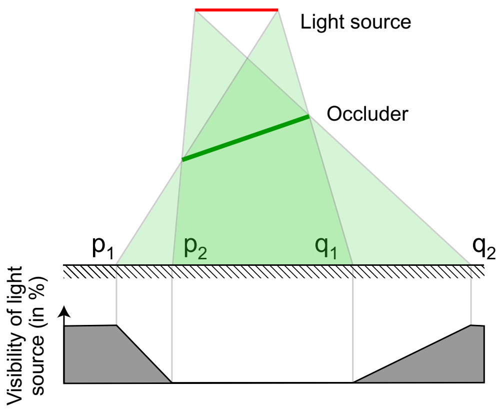


Figure 12: *Percentage of a linear light source that is visible.*

the objects visible from the light source are stored in the map:

- First we render the scene, using standard illumination and textures. This first pass eliminates all objects invisible from the viewer.
- Then, for each pixel of the image, we find whether the corresponding point in the scene is in the Layered Attenuation Map or not. If it is, then we modulate the lighting value found by the percentage of occlusion stored in the map. If it isn't, then the point is completely hidden from the light source.

Discussion: The main advantage of this method, compared to the previous method, is that a single image is used to store the shadowing information for the entire scene, compared to one shadow texture for each shadowed object. Also, we do not have to identify beforehand the objects casting shadows.

The extended memory cost of the Layered Attenuation Map is reasonable: experiments by the authors show that on average, about 4 layers are used in moderately complex scenes.

As with the previous method, the speed and realism are related to the number of samples used on the light source. We are rendering the entire scene N_s times, which precludes real-time rendering for complex scenes.

3.1.3. Quantitative Information in the Shadow Map [24]

Heidrich *et al.* [24] introduced another extension of the shadow map method, where we compute not only a shadow map, but also a visibility channel(see Figure 12), which encodes the percentage of the light source that is visible. Heidrich *et al.* [24]'s method only works for linear light sources, but it was later extended to polygonal area light sources by Ying *et al.* [54].

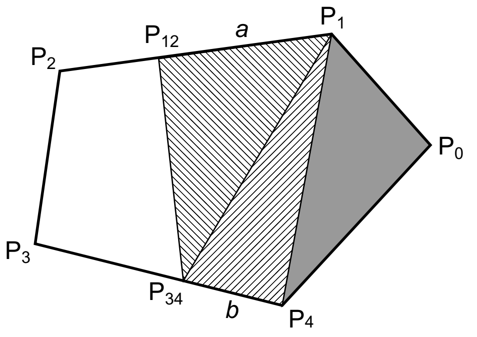


Figure 13: *Using the visibility channel to compute visibility from a polygonal light source. The shadow maps tell us that vertices P_0, P_1 and P_4 are occluded and that vertices P_2 and P_3 are visible. The visibility channel for edge $[P_1P_2]$ tells us that this edge is occluded for a fraction a; similarly, the visibility channel for edge $[P_3P_4]$ tells us that this edge is occluded for a fraction b. The portion of the light that is occluded is the hatched region, whose area can be computed geometrically using a and b.*

Method: We start by rendering a standard shadow map for each sample point on the linear light source. The number of sample points is very low, usually they are equal to the two end vertices of the linear light source.

In each shadow map, we detect discontinuities using image analysis techniques. Discontinuities in the shadow map happen at shadow boundaries. They are separating an object casting shadow from the object receiving shadow. For each discontinuity, we form a polygon linking the frontmost object (casting shadow) to the back object (receiving shadow). These polygons are then rendered in the point of view of the other sample, using Gouraud shading, with value 0 on the closer points, and 1 on the farthest points.

This gives us a visibility channel, which actually encodes the percentage of the edge linking the two samples that is visible.

The visibility channel is then used in a shadow mapping algorithm. For each pixel in the rendered image, we first check its position in the shadow map for each sample.

- if it is in shadow for all sample points, we assume that it is in shadow, and therefore it is rendered black.
- if it is visible from all sample points, we assume that it is visible, and therefore rendered using standard OpenGL illumination model.
- if it is hidden for some sample point, and visible from another point, we use the visibility channel to modulate the light received by the pixel.

Ying *et al.* [54] extended this algorithm to polygonal area light sources: we generate a shadow map for each vertex of the polygonal light source, and a visibility channel for each edge. We then use this information to compute the percentage of the polygonal light source that is visible from the current pixel.

For each vertex of the light source, we query the shadow map of this vertex. This gives us a boolean information, whether this vertex is occluded or not from the point of view of the object corresponding to the current pixel. If an edge links an occluded vertex to an non-occluded one, the visibility channel for this edge gives us the percentage of the edge that is occluded (see Figure 13). Computing the visible area of the light source is then a simple 2D problem. This area can be expressed as a linear combination of the area of triangles on the light source. By precomputing the area of these triangles, we are left with a few multiplications and additions to perform at each pixel.

Discussion: The strongest point of this algorithm is that it requires a small number of sampling points. Although it can work with just the vertices of the light source used as sampling points, a low number of samples can result in artefacts in moderately complex scenes. These artefacts are avoided by adding a few more samples on the light source.

This method creates fake shadows, but nicely approximated. The shadows are exact when only one edge of the occluder is intersecting the light source, and approximate if there is more than one edge, for example at the intersection of the shadows of two different occluders, or when an occluder blocks part of the light source without blocking any vertex.

The interactivity of the algorithm depends on the time it takes to generate the visibility channels, which itself depends on the complexity of the shadow. On simples scenes (a few occluders) the authors report computation times of 2 to 3 frames per second.

The algorithm requires having a polygonal light source, and organising the samples, so that samples are linked by edges, and for each edge, we know the sample points it links.

3.1.4. Single Sample Soft Shadows [9,33]

A different image-based method to generate soft shadows was introduced by Parker *et al.* [41] for parallel ray-tracing and later modified to use graphics hardware by Brabec and Seidel [9].

This method is very similar to standard shadow mapping. It starts by computing a standard shadow map, then uses the depth information available in the depth map to extend the shadow region and create a penumbra. In this method, we distinguish between the inner penumbra (the part of the penumbra that is inside the shadow of the point sample) and

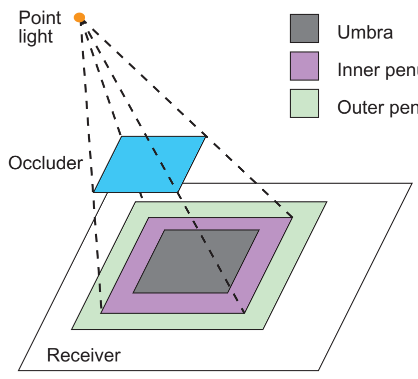


***Figure 14:** Extending the shadow of a point light source: for each occluder identified in the shadow map, we compute a penumbra, based on the distance between this occluder and the receiver.*

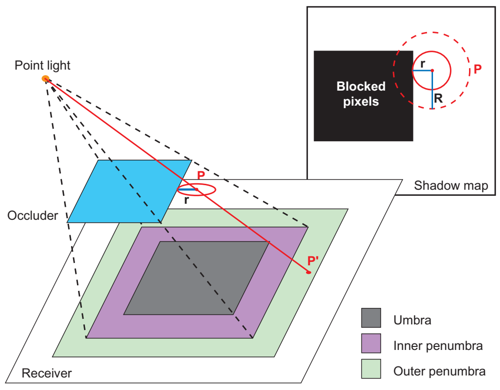


***Figure 15:** Extending the shadow of a single sample: For each pixel in the image, we find the corresponding pixel P in the shadow map. Then we find the nearest blocked pixel. P is assumed to be in the penumbra of this blocker, and we compute an attenuation coefficient based on the relative distances betwen light source, occluder and P.*

the outer penumbra (the part of the umbra that is outside the shadow of the point sample, see Figure 14). Parker *et al.* [41] compute only the outer penumbra; Brabec and Seidel [9] compute both the inner and the outer penumbra; Kirsch and Doellner [33] compute only the inner penumbra. In all cases, the penumbra computed goes from 0 to 1, to ensure continuity with areas in shadow and areas that are fully illuminated.

Method: In a first pass, we create a single standard shadow map, for a single sample — usually at the center of the light source.

During rendering, as with standard shadow mapping, we identify the position of the current pixel in the shadow map. Then:

- if the current pixel is in shadow, we identify the nearest pixel in the shadow map that is illuminated.
- if the pixel is lit, we identify the nearest pixel in the shadow map that corresponds to an object that is closer to the light source than the current pixel (see Figure 15).

In both cases, we assume that the object found is casting a shadow on the receiver, and that the point we have found is in the penumbra. We then compute an attenuation coefficient based on the relative positions of the receiver, the occluder and the light source:

$$f = \frac{\text{dist}(\text{Pixel}_{\text{Occluder}}, \text{Pixel}_{\text{Receiver}})}{RSz_{\text{Receiver}}|z_{\text{Receiver}} - z_{\text{Occluder}}|}$$

where R and S are user-defineable parameters. The intensity of the pixel is modulated using [8]:

- $0.5 * (1 + f)$, clamped to $[0.5, 1]$ if the pixel is outside the shadow,
- $0.5 * (1 - f)$, clamped to $[0, 0.5]$ if the pixel is inside the shadow.

For pixels that are far away from the boundary of the shadow, either deep inside the shadow or deep inside the fully lit area, f gets greater than 1, resulting in a modulation coefficient of respectively 0 or 1. On the original shadow boundary, $f = 0$, the two curves meet each other continuously with a modulation coefficient of 0.5. The actual width of the penumbra region depends on the ratio of the distances to the light source of the occluder and the receiver, which is perceptually correct.

The slowest phase of this algorithm is the search of neighbouring pixels in the shadow map, to find the potential occluder. In theory, an object can cast a penumbra than spans the entire scene, if it is close enough to the light source. In practice, we limit the search to a maximal distance to the current pixel of $R_{max} = Rz_{\text{Receiver}}$.

To ensure that an object is correctly identified as being in shadow or illuminated, the information from the depth map is combined with an item buffer, following Hourcade and Nicolas [26].

Discussion: The aim of this algorithm is to produce perceptually pleasing, rather than physically exact, soft shadows. The width of the penumbra region depends on the ratio of the respective distances to the light source of the occluder and the receiver. The penumbra region is larger if the occluder is far from the receiver, and smaller if the occluder is close to the receiver.

Of course, the algorithm suffers from several shortcomings. Since the shadow is only determined by a single sample shadow map, it can fail to identify the proper shadowing edge. It works better if the light source is far away from the occluder. The middle of the penumbra region is placed on the boundary of the shadow from the single sample, which is not physically correct.

The strongest point of this algorithm is its speed. Since it only needs to compute a single shadow map, it can achieve framerates of 5 to 20 frames per second, compared with 2 to 3 frames per second for multi-samples image-based methods. The key parameter in this algorithm is R, the search radius. For smaller search values of R, the algorithms works faster, but can miss large penumbras. For larger values of R, the algorithm can identify larger penumbras, but takes longer for each rendering.

A faster version of this algorithm, by Kirsch and Doellner [33], computes both the shadow map and a shadow-width map: for each point in shadow, we precompute the distance to the nearest point that is illuminated. For each pixel, we do a look-up in the shadow map and the shadow-width map. If the point is occluded, we have the depth of the current point (z), the depth of the occluder (z_{occluder}) and the shadow width (w). A 2D function gives us the modulation coefficient:

$$I(z, w) = \begin{cases} 1 \quad \text{if} \quad z = z_{\text{occluder}} \\ 1 + c_{bias} - c_{scale} \frac{w}{z_{\text{occluder}} - z} \quad \text{otherwise} \end{cases}$$

The shadow-width map is generated from a binary occlusion map, transformed into the width map by repeated applications of a smoothing filter. This repeated filtering is done using graphics hardware, during rendering. Performances depend mostly on the size of the occlusion map and on the size of the filter; for a shadow map resolution of 512×512 pixels, and a large filter, they attain 20 frames per second. Performance depends linearly on the number of pixels in the occlusion map, thus doubling the size of the occlusion map divides the rendering speed by 4.

3.1.5. Convolution technique [45]

As noted earlier, soft shadows are a consequence of partial visibility of an extended light source. Therefore the calculation and soft shadows is closely related to the calculation of the visible portion of the light source.

Soler and Sillion [45] observe that the percentage of the source area visible from a receiving point can be expressed as a simple convolution for a particular configuration. When the light source, occluder, and receiver all lie in parallel planes, the soft shadow image on the receiver is obtained by convolving an image of the receiver and an image of the light source. While this observation is only mathematically valid in this very restrictive configuration, the authors describe how the same principle can be applied to more general configurations:

First, appropriate imaging geometries are found, even when the objects are non-planar and/or not parallel. More importantly, the authors also describe an error-driven algorithm in which the set of occluders is recursively subdivided according to an appropriate error estimate, and the shadows created by the subsets of occluders are combined to yield the final soft shadow image.

Discussion: The convolution technique's main advantages are the visual quality of the soft shadows (not their physical fidelity), and the fact that it operates from images of the source and occluders, therefore once the images are obtained the complexity of the operations is entirely under control. Sampling is implicitly performed when creating a light source image, and the combination of samples is handled by the convolution operation, allowing very complex light source shapes.

The main limitation of the technique is that the soft shadow is only correct in a restricted configuration, and the proposed subdivision mechanism can only improve the quality when the occluder can be broken down into smaller parts. Therefore the case of elongated polygons in th direction of the light source remains problematic. Furthermore, the subdivision mechanism, when it is effective in terms of quality, involves a significant performance drop.

3.2. Object-Based Approaches

Several methods can be used to compute soft shadows in animated scenes using object-based methods:

1. Combining together several shadow volumes taken from point samples on the light source, in a manner similar to the method described for shadow maps in Section 3.1.1.
2. extending the shadow volume [11,19,53] using a specific heuristic (Plateaus [19], Penumbra Maps [53], Smoothies [11]).
3. computing a penumbra volume for each edge of the shadow silhouette [2,4,5].

3.2.1. Combining several hard shadows

Method: The simplest way to produce soft shadows with the shadow volume algorithm is to take several samples on the light source, compute a hard shadow for each sample and average the pictures produced. It simulates an area light source, and gives us the soft shadow effect.

However, the main problem with this method, as with the equivalent method for shadow maps, is the number of samples it requires to produce a good-looking soft shadow, which precludes any real-time application. Also, it requires the use of an accumulation buffer, which is currently not supported on standard graphics hardware.

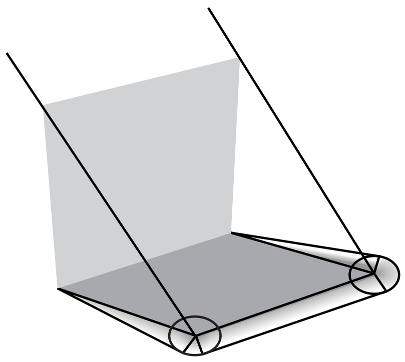


Figure 16: *Extending the shadow volume of an occluder with cones and planes.*

An interesting variation has been proposed by Vignaud [47], in which shadow volumes from a light source whose position changes with time are added in the alpha buffer, mixed with older shadow volumes, producing a soft shadow after a few frames where the viewer position does not change.

3.2.2. Soft Planar Shadows Using Plateaus

The first geometric approach to generate soft shadows has been implemented by Haines [19]. It assumes a planar receiver, and generates an attenuation map that represents the soft shadow. The attenuation map is created by converting the edges of the occluders into volumes, and is then applied to the receiver as a modulating texture.

Method: The principle of the plateaus method [19] is to generate an attenuation map, representing the soft shadow. The attenuation map is first created using the shadow volume method, thus filling in black the parts of the map that are occluded.

Then, the edges of the silhouette of the objects are transformed into volumes (see Figure 16):

- All the vertices of the silhouette are first turned into cones, with the radius of the cone depending on the distance between the occluder vertex and the ground, thus simulating a spherical light source.
- then edges joining adjacent vertices are turned into surfaces. For continuity, the surface joining two cones is an hyperboloid, unless the two cones have the same radius (that is, if the two original vertices are at the same distance of the ground), in which case the hyperboloid degenerates to a plane.

These shadow volumes are then projected on the receiver and colored using textures: the axis of the cone is black, and the contour is white. This texture is superimposed with the shadow volume texture: Haines' algorithm only computes the outer penumbra.

One important parameter in the algorithm is the way we color the penumbra volume; it can be done using Gouraud shading, values from the Z-buffer or using a 1D texture. The latter gives more control over the algorithm, and allows penumbra to decrease using any function, including sinusoid.

Discussion: The first limitation of this method is that it is limited to shadows on planar surfaces. It also assumes a spherical light source. The size of the penumbra only depends on the distance from the receiver to the occluders, not from the distance between the light source and the occluders. Finally, it suffers from the same fillrate bottleneck as the original shadow volume algorithm.

A significant improvement is Wyman and Hansen [53]'s Penumbra Map method: the interpolation step is done using programmable graphics hardware [6,14,20], generating a penumbra map that is applied on the model, along with a shadow map. Using a shadow map to generate the umbra region removes the fill-rate bottleneck and makes the method very robust. Wyman and Hansen report framerate of 10 to 15 frames per second on scenes with more than 10,000 shadow-casting polygons.

The main limitation in both methods [19,53] is that they only compute the outer penumbra. As a consequence, objects will always have an umbra, even if the light source is very large with respect to the occluders. This effect is clearly noticeable, as it makes the scene appear much darker than anticipated, except for very small light sources.

3.2.3. Smoothies [11]

Chan and Durand [11] present a variation of the shadow volume method that uses only graphics hardware for shadow generation.

Method: We start by computing the silhouette of the object. This silhouette is then extended using "smoothies", that are planar surfaces connected to the edges of the occluder and perpendicular to the surface of the occluder.

We also compute a shadow map, which will be used for depth queries. The smoothies are then textured taking into account the distance of each silhouette vertex to the light source, and the distance between the light source and the receiver.

In the rendering step, first we compute the hard shadow using the shadow map, then the texture from the smoothies is projected onto the objects of the scene to create the penumbra.

Discussion: As with Haines [19], Wyman and Hansen [53] and Parker [41], this algorithm only computes the outer

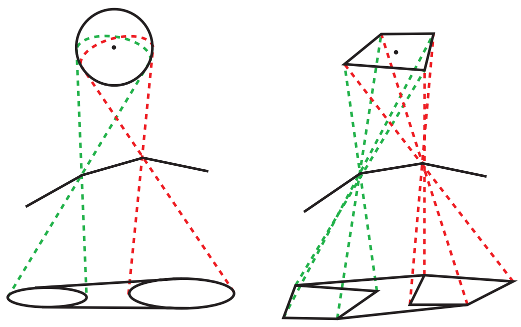


Figure 17: Computing the penumbra wedge of a silhouette edge: the wedge is a volume based on the silhouette edge and encloses the light source.

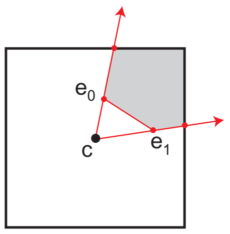


Figure 18: Computing the area of the light source that is covered by a given edge. The fragment program computes the hatched area for each pixel inside the corresponding wedge.

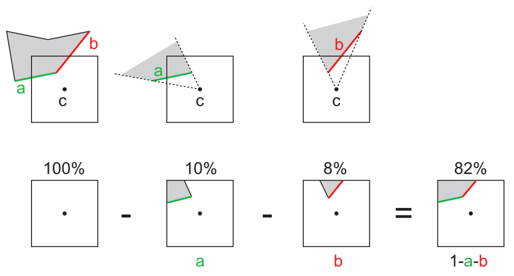


Figure 19: Combining several connected edges. The portion of the light source that is occluded is equal to the sum of the portions of the light source occluded by the different edges.

penumbra. As a consequence, occluders will always project an umbra, even if the light source is very large with respect to the occluders. As mentionned earlier, this makes the scene appear much darker than anticipated, an effect that is clearly noticeable except for very small light sources.

The size of the penumbra depends on the ratio of the distances between the occluder and the light source, and between receiver and light source, which is perceptually correct.

Connection between adjacent edges is still a problem with this algorithm, and artefacts appear clearly except for small light sources.

The shadow region is produced using the shadow map method, which removes the problem with the fill rate bottleneck experienced with all other methods based on the shadow volume algorithm. As with the previous method [53], the strong point of this algorithm is its robustness: the authors have achieved 20 frames per second on scenes with more than 50,000 polygons.

3.2.4. Soft Shadow Volumes [2,4,5]

Akenine-Möller and Assarsson [2], Assarsson and Akenine-Möller [4] and Assarsson *et al.* [5] have developed an algorithm to compute soft shadows that builds on the shadow volume method and uses the programmable capability of modern graphics hardware [6,14,20] to produce real-time soft shadows.

Method: The algorithm starts by computing the silhouette of the object, as seen from a single sample on the light source. For each silhouette edge, we build a *silhouette wedge*, that encloses the penumbra caused by this edge (see Figure 17). The wedge can be larger than the penumbra, that is we err on the safe side.

Then, we render the shadow volume, using the standard method (described in Section 2.5.2) in a visibility buffer. After this first pass, the visibility buffer contains the hard shadow.

In a subsequent pass, this visibility buffer is updated so that it contains the soft shadow values. This is done by rendering the front-facing triangles of each wedge. For each pixel covered by these triangles, we compute the percentage of the light source that is occluded, using fragment programs [20]. For pixels that are covered by the wedge but in the hard shadow (as computed by the previous pass), we compute the percentage of the light source that is visible, and add this value to the visibility buffer. For pixels covered by the wedge but in the illuminated part of the scene, we compute the percentage of the light source that is occluded and substract this value from the visibility buffer (see Figures 18 and 19).

After this second pass, the visibility buffer contains the percentage of visibility for all pixels in the picture. In a third pass, the visibility buffer is combined with the illumination computed using the standard OpenGL lighting model, giving the soft shadowed picture of the scene.

Discussion: The complexity of the algorithm depends on the number of edges in the silhouette of the object, and on the number of pixels covered by each penumbra wedge. As a consequence, the easiest optimisation of the algorithm is to compute tighter penumbra wedges [5].

The main advantage of this algorithm is its speed. Using programmable graphics hardware for all complex computations, and tabulating complex functions into pre-computed textures, framerates of 150 frames per second are obtained on simple scenes, 50 frames per second on moderately complex scenes (1,000 shadow-casting polygons, with a large light source), with very convincing shadows. Performance depends mostly on the number of pixels covered by the penumbra wedges, so smaller light sources will result in faster rendering.

It should be noted that although a single sample is used to compute the silhouette of the object, the soft shadow computed by this algorithm is physically exact in simple cases, since visibility is computed on the entire light source. More precisely this happens when the silhouette of the occluder remains the same for all points on the light source, *e.g.* for a convex object that is distant enough from the light source.

The weak point of the algorithm is that it computes the silhouette of the object using only a single sample. It would fail on scenes where the actual silhouette of the object, as seen from the area light source, is very different from the silhouette computed using the single sample. Such scenes include scenes where a large area light source is close to the object (see Figure 7), and scenes where the shadows of several objects are combined together (as in Figure 6). In those circumstances, it is possible to compute a more accurate shadow by splitting the light source into smaller light sources. The authors report that splitting large light sources into 2×2 or 3×3 smaller light sources is usually enough to remove visible artefacts. It should be noted that splitting the light source into n light sources does not cut the speed of the algorithm by n, since the rendering time depends on the number of pixels covered by the penumbra wedges, and smaller light sources have smaller penumbra wedges.

One key to the efficiency of the algorithm is its use of fragment programs [20]. The fragment programs take as input the projections of the extremities of the edge onto the plane of the light source, and give as output the percentage of the light source that is occluded by the edge (see Figure 18). If several edges are projecting onto the light source, their contributions are simply added (see Figure 19) — this addition is done in the framebuffer. The authors have implemented several fragment programs, for spherical light sources, for textured rectangular light sources and for non-textured rectangular light sources.

4. Classification

4.1. Controlling the time

Algorithms used in real time or interactive applications must be able to run at a tuneable framerate, in order to spend less time for rendering at places where there is a lot of computation taking place, and more time when the processor is available.

Ideally, soft shadow methods used in real-time applications should take as input the amount of time available for rendering, and return a soft shadow computed to the best of the algorithm within the prescribed time limit. Since this review focuses on hot research algorithms, this feature has not been implemented in any of the algorithms reviewed here. However, all of these algorithms are tunable in the sense that there is some sort of parameter that the user can tweak, going from soft shadows that are computed very fast, but are possibly wrong, to soft shadows that can take more time to compute but are either more visually pleasing or more physically accurate.

Several of these parameters are available to a various degree in the methods reviewed:

- The easiest form of user control is the use of a different level-of-detail for the geometry of the occluders. Simpler geometry will result in faster rendering, either with image-based methods or with object-based methods. It can be expected that the difference in the shadow will not be noticeable with animated soft shadows.
- Another form of user control is to add more samples on the light source [1,22,25], or to subdivide large light sources into a set of smaller ones [2,4,5,24,54]. It should be noted that the order of magnitude for this parameter is variable: 256 to 1024 samples are required for point-based methods [1,22,25] to produce shadows without artefacts, while area-based methods [2,4,5,24,54] just need to cut the light source into 2×2 or 3×3 smaller sources. Either way, the rendering time is usually multiplied by the number of samples or sources.
- All image-based methods are also tuneable by changing the resolution of the buffer.
- Other parameters are method-specific:
  - the single sample soft shadows [9] method is tuneable by changing the search radius;
  - Convolution [45] is tuneable by subdividing the occluders into several layers;
  - Plateaus [19] are tuneable by changing the number of vertices used to discretize the cones and patches;
  - Smoothies [11] are tuneable by changing the maximum width of the smoothies;

4.2. Controlling the aspect

Another important information in chosing a real-time soft shadow algorithm is the aspect of the shadow it produces. Some of the algorithms described in this review can produce a physically exact solution if we allow them a sufficient

Table 1: *Comparison of soft shadows algorithms (see Section 4 for details)*

Method	Time	Quality	Tunable	Light	Scene	Required Hardware
Image-based						
Multi-samples [22,25]	I	*	Y	Polygon	1 planar receiver	
Distributed Multi-samples [28]	RT	**	Y	Planar		ShadowMap
Single sample [9,33]	RT	*	Y	Sphere		ShadowMap
Convolution [45]	I	**	Y	Polygon		2D Convol.
Visibility Channel [24,54]	I	**	Y	Linear, Polygon		2D Convol.
Geometry-based						
Plateaus [19]	I	**	Y	Sphere	1 planar receiver	
Penumbra Map [53]	RT	**	Y	Sphere		Vertex & Frag. Programs
Smoothie [11]	RT	**	Y	Sphere		Vertex & Frag. Programs
Soft Shadow Volumes [2,4,5]	RT	***	Y	Sphere, Rect.		Fragment Programs

rendering time. Other methods produce a physically exact solution in simple cases, but are approximate in more complex scenes, and finally a third class of methods produce shadows that are always approximate, but are usually faster to compute.

Physically exact (time permitting): Methods based on point samples on the light source [1,22,25] will produce physically exact shadows if the number of samples is sufficient. However, with current hardware, the number of samples compatible with interactive applications gives shadows that are not visually excellent (hence the poor mark these methods receive in Table 1).

Physically exact on simple scenes: Methods that compute the percentage of the light source that is visible from the current pixel will give physically exact shadows in places where the assumptions they make on the respective geometry of the light source and the occluders are verified. For example, soft shadow volumes [4,5] give physically exact shadows for isolated convex objects, provided that the silhouette computed is correct (that the occluder is far away from the light source). Visibility channel [24,54] gives physically exact shadows for convex occluders and linear light sources [24], and for isolated edges and polygonal light sources [54] . Convolution [45] is physically exact for planar and parallel light source, receiver and occluder.

Always approximate: All methods that restrict themselves to computing only the inner- or the outer-penumbra are intrisically always approximate. They include single-sample soft shadows using shadow-width map [33], plateaus [19] and smoothies [11]. The original implementation of single sample soft shadows [9] computes both the inner- and the outer-penumbra, but gives them always the same width, which is not physically exact.

The second class of methods is probably the more interesting for producing nice looking pictures. While the conditions imposed seem excessively hard, it must be pointed out that they are conditions for which it is *guaranteed* that the shadow is exact in *all* the points of the scene. In most places of a standard scene, these methods will also produce physically exact shadows.

4.3. Number and shape of the light sources

The first cause for the soft shadow is the light source. Each real-time soft shadow method makes an assumption on the light sources, their shapes, their angles of emission and more importantly their number.

Field of emission: All the methods that are based on an image of the scene computed from the light source are restricted with respect to the field of emission of the light source, as a field of emission that is too large will result in distortions in the image. This restriction applies to all image-based algorithms, plus smoothies [11] and volume-based algorithms if the silhouette is computed using discontinuities in the shadow map [39].

On the contrary, volume-based methods can handle omnidirectional illumination.

Shape: For extended light sources, the influence of the shape of the light source on a soft shadow is not directly perceptible. Most real-time soft shadow methods use this property by restricting themselves to simple light source shapes, such as spheres or rectangles:

- Single-sample soft shadows [9,33], plateaus [19] and smoothies [11] assume a spherical light source. Soft shadow volumes [5] also work with a spherical light source.
- Visibility channel [24] was originally restricted to linear light sources.
- Subsequent implementation of the visibility channel works with polygonal light sources [54].
- Other methods place less restriction on the light source. Multi-sample methods [1,25] can work with any kind of light source. Convolution [45] are also not restricted. However, in both cases, the error in the algorithm is smaller for planar light sources.
- Convolution [45] and soft shadow volumes [4,5] work with textured rectangles, thus allowing any kind of planar light source. The texture can even be animated [4,5].

Number: All real-time soft shadow algorithms are assuming a single light source. Usually, computing the shadow from several light sources results in multiplying the rendering time by the number of light sources. However, for all the methods that work for any kind of planar light source [1,4,5,25,45], it is possible to simulate several co-planar light sources by placing the appropriate texture on a plane. This gives us several soft shadows in a single application of the algorithm. However, it has a cost: since the textured light source is larger, the algorithms will run more slowly.

4.4. Constraints on the scene

The other elements causing shadows are the occluders and the receivers. Most real-time soft shadows methods make some assumptions on the scene, either explicit or implicit.

Receiver: The strongest restriction is when the object receiving shadows is a plane, as with the plateaus method [19]. Multi-sample soft shadow [22,25] is also restricted to a small number of receivers for interactive rendering. In that case, self-shadowing is not applicable.

Self-shadowing: The convolution [45] method requires that the scene is cut into clusters, within which no self-shadows are computed.

Silhouette: For all the methods that require a silhouette extraction — such as object-based methods — it is implicitly assumed that we can compute a silhouette for all the objects in the scene. In practice, this usually means that the scene is made of closed triangle meshes.

4.5. New generation of GPUs

Most real-time soft shadow methods use the features of the graphics hardware that were available to the authors at the time of writing:

Shadow-map: all image-based methods use the `GL_ARB_SHADOW` extension for shadow maps. This extension (or an earlier version) is available, for example, on Silicon Graphics hardware above the Infinite Reality 2, on NVIDIA graphics cards above the GeForce 3 and on ATI graphics above the Radeon9500.

Imaging subset: along with this extension, some methods also compute convolutions on the shadow map. These convolutions can be computed in hardware if the *Imaging Subset* of the OpenGL specification is present. This is the case on all Silicon Graphics machines and NVIDIA cards.

Programmable GPU: finally, the most recent real-time soft shadow methods use the programming capability introduced in recent graphics hardware. Vertex programs [14] and fragment programs [21] are used for single-sample soft shadows [33], penumbra maps [53], smoothies [11] and soft shadow volumes [4,5]. In practice, this restricts these algorithms to only the latest generation of graphics hardware, such as the NVIDIA GeForce FX or the ATI Radeon 9500 and above.

Many object-based algorithms suffer from the fact that they need to compute the silhouette of the occluders, a costly step that can only be done on the CPU. Wyman and Hansen [53] report that computing the silhouette of a moderately complex occluder (5000 polygons) uses 10 ms in their implementation. If the next generation of graphics hardware would include the possibility to compute this silhouette entirely on the graphics card [10], object-based algorithms [2,4,5,11,53] would greatly benefit from the speed-up.

5. Conclusions

In this State of the Art Review, we have described the issues encountered when working with soft shadows. We have presented existing algorithms that produce soft shadows in real time. Two main categories of approaches have been reviewed, based on shadow maps and shadow volumes. Each one has advantages and drawbacks, and none of them can simultaneously solve all the problems we have mentioned. This motivated a discussion and classification of these methods, hopefully allowing easier algorithm selection based on a particular application's constraints.

We have seen that the latest algorithms benefit from the programmability of recent graphics hardware. Two main directions appear attractive to render high-quality soft shadows in real time: by programming graphics hardware, and by taking advantage simultaneously of both image-based and object-based techniques. Distributed rendering, using for instance PC clusters, is another promising avenue although little has been achieved so far. Interactive display speeds can be obtained today even on rather complex scenes. Continuing improvements of graphics technology — in performance and programmability — lets us expect that soft shadows will soon become a common standard in real-time rendering.

Acknowledgments

The “Hugo” robot used in the pictures of this paper was created by Laurence Boissieux.

This work was supported in part by the “ACI Jeunes Chercheurs” *CYBER* of the French Ministry of Research, and by the “Région Rhône-Alpes” through the DEREVE research consortium.

We wish to express our gratitude to the authors of the algorithms described in this review, who have provided us with useful detailed information about their work, and to the anonymous reviewers whose comments and suggestions have significantly improved the paper.

Remark: All the smooth shadows pictures in this paper were computed with distributed ray-tracing, using 1024 samples on the area light sources.

References

1. Maneesh Agrawala, Ravi Ramamoorthi, Alan Heirich and Laurent Moll. Efficient image-based methods for rendering soft shadows. In *Computer Graphics (SIGGRAPH 2000)*, Annual Conference Series, ACM SIGGRAPH, pp. 375–384. 2000.
2. Tomas Akenine-Möller and Ulf Assarsson. Approximate soft shadows on arbitrary surfaces using penumbra wedges. In *Rendering Techniques 2002 (13th Eurographics Workshop on Rendering)*, ACM Press, pp. 297–306. 2002.
3. Tomas Akenine-Möller and Eric Haines. *Real-Time Rendering*. A K Peters Ltd, 2nd edition, 2002.
4. Ulf Assarsson and Tomas Akenine-Möller. A geometry-based soft shadow volume algorithm using graphics hardware. *ACM Transactions on Graphics (SIGGRAPH 2003)*, 22(3):2003.
5. Ulf Assarsson, Michael Dougherty, Michael Mounier and Tomas Akenine-Möller. An optimized soft shadow volume algorithm with real-time performance. In *Graphics Hardware*. 2003.
6. Smartshader^{TM} technology white paper. ATI, 2001 http://www.ati.com/products/pdf/smartshader.pdf
7. Harlen Costa Batagelo and Ilaim Costa Júnior. Real-time shadow generation using BSP trees and stencil buffers. In *SIBGRAPI*, vol. 12, pp. 93–102. 1999.
8. Stefan Brabec. Personnal communication. 2003.
9. Stefan Brabec and Hans-Peter Seidel. Single sample soft shadows using depth maps. In *Graphics Interface*. 2002.
10. Stefan Brabec and Hans-Peter Seidel. Shadow volumes on programmable graphics hardware. *Computer Graphics Forum (Eurographics 2003)*, 25(3):2003.
11. Eric Chan and Fredo Durand. Rendering fake soft shadows with smoothies. In *Rendering Techniques 2003 (14th Eurographics Symposium on Rendering)*. ACM Press, 2003.
12. Franklin C. Crow. Shadow algorithms for computer graphics. *Computer Graphics (SIGGRAPH 1977)*, 11(3):242–248, 1977.
13. George Drettakis and Eugene Fiume. A fast shadow algorithm for area light sources using backprojection. In *Computer Graphics (SIGGRAPH 1994)*, Annual Conference Series, ACM SIGGRAPH, pp. 223–230. 1994.
14. Cass Everit. OpenGL ARB vertex program. 2003 http://developer.nvidia.com/docs/IO/8230/GDC2003_OGL_ARBVertexProgram.pdf
15. Cass Everitt and Mark J. Kilgar. Practical and robust stenciled shadow volumes for hardware-accelerated rendering. 2002 http://developer.nvidia.com/object/robust_shadow_volumes.html
16. Cass Everitt and Mark J. Kilgard. Optimized stencil shadow volumes. 2003 http://developer.nvidia.com/docs/IO/8230/GDC2003_ShadowVolumes.pdf
17. Cass Everitt, Ashu Rege and Cem Cebenoya. Hardware shadow mapping. http://developer.nvidia.com/object/hwshadowmap_paper.html
18. Randima Fernando, Sebastian Fernandez, Kavita Bala and Donald P. Greenberg. Adaptive shadow maps. In *Computer Graphics (SIGGRAPH 2001)*, Annual Conference Series, ACM SIGGRAPH, pp. 387–390. 2001.
19. Eric Haines. Soft planar shadows using plateaus. *Journal of Graphics Tools*, 6(1):19–27, 2001.
20. Evan Hart. ARB Fragment Program: Fragment level programmability in OpenGL. 2003 http://www.ati.com/developer/gdc/GDC2003_OGL_ARBFragmentProgram.pdf
21. Evan Hart. Other New OpenGL Stuff: Important stuff that doesn’t fit elsewhere. 2003 http://www.ati.com/developer/gdc/GDC2003_OGL_MiscExtensions.pdf

22. Paul S. Heckbert and Michael Herf. Simulating soft shadows with graphics hardware. Technical Report CMU-CS-97-104, Carnegie Mellon University, 1997.

23. Tim Heidmann. Real shadows, real time. In *Iris Universe*, vol. 18, Silicon Graphics Inc., pp. 23–31. 1991.

24. Wolfgang Heidrich, Stefan Brabec and Hans-Peter Seidel. Soft shadow maps for linear lights high-quality. In *Rendering Techniques 2000 (11th Eurographics Workshop on Rendering)*, Springer-Verlag, pp. 269–280. 2000.

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

26. J.-C. Hourcade and A. Nicolas. Algorithms for antialiased cast shadows. *Computers & Graphics*, 9(3):259–265, 1985.

27. Geoffre S. Hubona, Philip N. Wheeler, Gregory W. Shirah and Matthew Brandt. The role of object shadows in promoting 3D visualization. *ACM Transactions on Computer-Human Interaction*, 6(3):214–242, 1999.

28. M. Isard, M. Shand and A. Heirich. Distributed rendering of interactive soft shadows. In *4th Eurographics Workshop on Parallel Graphics and Visualization*, Eurographics Association, pp. 71–76. 2002.

29. Brett Keating and Nelson Max. Shadow penumbras for complex objects by depth-dependent filtering of multi-layer depth images. In *Rendering Techniques 1999 (10th Eurographics Workshop on Rendering)*, Springer-Verlag, pp. 205–220. 1999.

30. Daniel Kersten, Pascal Mamassian and David C. Knill. Moving cast shadows and the perception of relative depth. Technical Report no 6, Max-Planck-Institut fuer biologische Kybernetik, 1994.

31. Daniel Kersten, Pascal Mamassian and David C. Knill. Moving cast shadows and the perception of relative depth. *Perception*, 26(2):171–192, 1997.

32. Mark J. Kilgard. Improving shadows and reflections via the stencil buffer. 1999 http://developer.nvidia.com/docs/IO/1348/ATT/stencil.pdf

33. Florian Kirsch and Juergen Doellner. Real-time soft shadows using a single light sample. *Journal of WSCG (Winter School on Computer Graphics 2003)*, 11(1):2003.

34. David C. Knill, Pascal Mamassian and Daniel Kersten. Geometry of shadows. *Journal of the Optical Society of America*, 14(12):3216–3232, 1997.

35. Johann Heinrich Lambert. *Die freye Perspektive*. 1759.

36. Tom Lokovic and Eric Veach. Deep shadow maps. In *Computer Graphics (SIGGRAPH 2000)*, Annual Conference Series, ACM SIGGRAPH, pp. 385–392. 2000.

37. Céline Loscos and George Drettakis. Interactive high-quality soft shadows in scenes with moving objects. *Computer Graphics Forum (Eurographics 1997)*, 16(3):1997.

38. Pascal Mamassian, David C. Knill and Daniel Kersten. The perception of cast shadows. *Trends in Cognitive Sciences*, 2(8):288–295, 1998.

39. Michael D. McCool. Shadow volume recontsruction from depth maps. *ACM Transactions on Graphics*, 19(1):1–26, 2000.

40. Steve Morein. ATI Radeon HyperZ technology. In *Graphics Hardware Workshop*. 2000.

41. Steven Parker, Peter Shirley and Brian Smits. Single sample soft shadows. Technical Report UUCS-98-019, Computer Science Department, University of Utah, 1998.

42. William T. Reeves, David H. Salesin and Robert L. Cook. Rendering antialiased shadows with depth maps. *Computer Graphics (SIGGRAPH 1987)*, 21(4):283–291, 1987.

43. Stefan Roettger, Alexander Irion and Thomas Ertl. Shadow volumes revisited. In *Winter School on Computer Graphics*. 2002.

44. Mark Segal, Carl Korobkin, Rolf van Widenfelt, Jim Foran and Paul Haeberli. Fast shadows and lighting effects using texture mapping. *Computer Graphics (SIGGRAPH 1992)*, 26(2):249–252, 1992.

45. Cyril Soler and François X. Sillion. Fast calculation of soft shadow textures using convolution. In *Computer Graphics (SIGGRAPH 1998)*, Annual Conference Series, ACM SIGGRAPH, pp. 321–332. 1998.

46. Marc Stamminger and George Drettakis. Perspective shadow maps. *ACM Transactions on Graphics (SIGGRAPH 2002)*, 21(3):557–562, 2002.

47. Sylvain Vignaud. Real-time soft shadows on geforce class hardware. 2003 http://tfpsly.planet-d.net/english/3d/SoftShadows.html

48. Leonardo Da Vinci. *Codex Urbinas*. 1490.

49. Leonard Wanger. The effect of shadow quality on the perception of spatial relationships in computer generated imagery. *Computer Graphics (Interactive 3D Graphics 1992)*, 25(2):39–42, 1992.

50. Lance Williams. Casting curved shadows on curved surfaces. *Computer Graphics (SIGGRAPH 1978)*, 12(3):270–274, 1978.

51. Andrew Woo. The shadow depth map revisited. In *Graphics Gems III*, Academic Press, pp. 338–342. 1992.

52. Andrew Woo, Pierre Poulin and Alain Fournier. A survey of shadow algorithms. *IEEE Computer Graphics and Applications*, 10(6):13–32, 1990.

53. Chris Wyman and Charles Hansen. Penumbra maps: Approximate soft shadows in real-time. In *Rendering Techniques 2003 (14th Eurographics Symposium on Rendering)*. ACM Press, 2003.

54. Zhengming Ying, Min Tang and Jinxiang Dong. Soft shadow maps for area light by area approximation. In *10th Pacific Conference on Computer Graphics and Applications*, IEEE, pp. 442–443. 2002.

55. Hansong Zhang. Forward shadow mapping. In *Rendering Techniques 1998 (9th Eurographics Workshop on Rendering)*, Springer-Verlag, pp. 131–138. 1998.