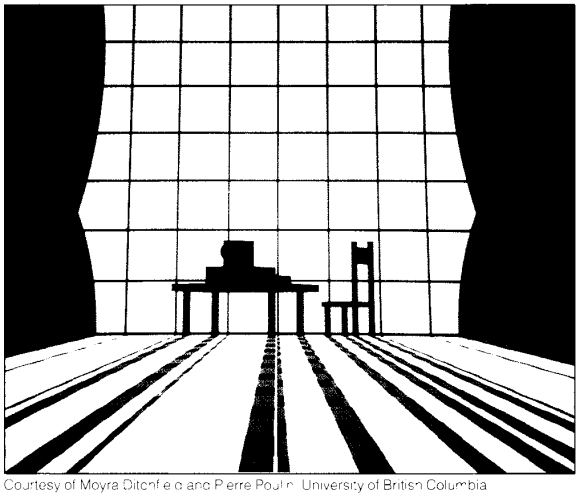
Survey

A Survey of Shadow Algorithms

Andrew Woo, Pierre Poulin, and Alain Fournier
University of Toronto



Courtesy of Moyra Ditchfield and Pierre Poulin, University of British Columbia

Essential to realistic and visually appealing images, shadows are difficult to compute in most display environments. This survey characterizes the various types of shadows. It also describes most existing shadow algorithms and discusses their complexities, advantages, and shortcomings. We examine hard shadows, soft shadows, shadows of transparent objects, and shadows for complex modeling primitives. For each type, we examine shadow algorithms within various rendering techniques.

This survey attempts to provide readers with enough background and insight on the various methods to allow them to choose the algorithm best suited to their needs. We also hope that our analysis will help identify the areas that need more research and point to possible solutions.

A shadow—a region of relative darkness within an illuminated region—occurs when an object totally or partially occludes the light. A transparent object does not necessarily attenuate the light it occludes. In fact, it can concentrate light. However, as is traditional in image synthesis, we will consider a region to be in

0272-17-16 90 1100-0013$01.00 © 1990 IEEE

Table 1. Common notation used in this survey.

Symbol	Definition
E	Average number of edges per polygon
P	Number of points simulating an extended light source
R	Linear resolution of some buffer used
n	Number of primitives in the scene
$p \times q$	Resolution of the image in pixels

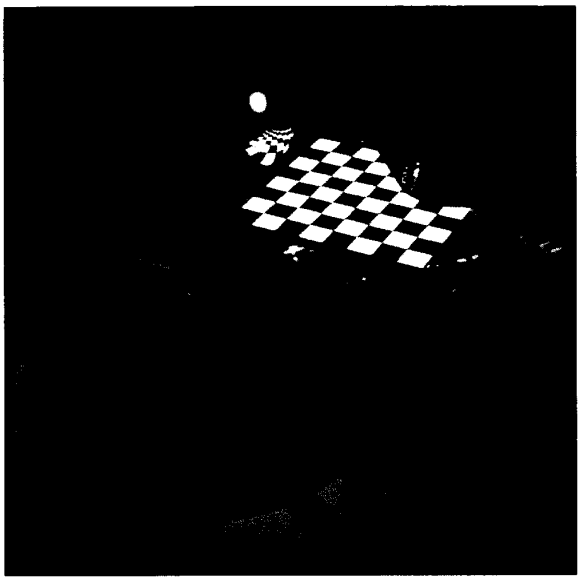


Figure 1. Hard shadows in ray tracing. Ray tracing offers a simple technique to produce sharp shadows. In this image, we used a single directional light source. Notice how the shadow of the table looks sharp and well defined over the objects in shadow.

shadow if there exists an occluding surface between light and the region of interest, whether transparent or not.

A shadow does not necessarily look like a darker region; colored shadows can appear in different ways in a scene. For example, let's say you used two different colored light sources. A region occluded from only one light source by an opaque object will lie in the shadow of this light source, but the color of the second light source can influence the region. An object can also act as a filter, with the color of the shadow region depending on the wavelength spectrum filtered through the object. Moreover, diffraction of light through a transparent object might provide color within a shadow.

Using almost any measure of the quality of an image, the computation of shadows is essential. They cause some of the highest intensity contrasts in images; they provide strong clues about the shapes, relative positions, and surface characteristics of the objects; they can indicate the approximate location, intensity, shape, and size of the light source(s); and they represent an integral part of the total effect in architecture with many objects in the environment. In fact, in some circumstances the shadows constitute the only components of the scene, as in shadow-puppet theater and in pin screen animation (developed by Alexander Alexeieff).

In the only previous comprehensive discussion of shadow algorithms, Crow[1] discussed shadow generation by classifying the type of algorithms used. He distinguished three general categories of shadow algorithms. This useful distinction has since inspired many developments. However, the intervening years have seen the advent of many new modeling primitives and rendering techniques. Within each one, new shadow algorithms have been developed. Amanatides,[2] Thalmann and Thalmann,[3] and Foley et al.[4] described some of the existing shadow algorithms within the general context of image synthesis. However, they dealt mainly with shadows created by opaque objects, and even within this framework their discussion was incomplete.

In this survey, we examine the algorithms according to the type of shadows produced: hard shadows and soft shadows caused by opaque objects, shadows caused by transparent objects, and shadows caused by more complex modeling primitives, such as parametric and implicit surfaces, particle systems, volume-filling media, and surface self-shadowing. Within each type, we examine algorithms as they relate to the specific rendering techniques for which they were developed. We hope that after reading this survey, implementors will be aware of nearly all the existing shadow algorithms and will have sufficient information to choose an algorithm given the type of rendering, primitives, and effects desired.

Of course, many problems related to shadows have not been solved (or solved well). With this in mind, we have tried to identify gaps and suggest improvements to known algorithms, as well as pointing out directions for new ones.

Complexity of algorithms

As a consistent basis for comparison, all shadow complexity order statistics assume a single light source and polygonal surfaces. The complexity consists mainly of three components: storage usage, preprocessing runtime complexity, and runtime complexity during actual rendering (referred to as shadow rendering complexity from here on). The storage complexity is stated with respect to the total storage requirements for shadow determination. The runtime complexities represent the additional cost of shadow computation over implementations that do not compute shadows. The shadow rendering complexity is also stated with respect to each pixel unless otherwise indicated.

Note that, in general, we give worst-case complexity. Whenever practical, we also give an estimate of average complexity, but this is risky without an analysis of scene statistics.

Table 1 summarizes common notation used throughout this survey.

Hard shadow generation

This section discusses shadow algorithms for hard shadows, requiring the display of the umbra section alone (as illustrated in Figure 1). Calculation of hard shadows involves only the determination of whether or not a point in the scene lies in the shadow of opaque objects. This is a binary decision problem on top of the shading model. In other words, multiply a value of either 0 or 1 by the light intensity, indicating in shadow or not in shadow, respectively.

The domain of light sources truly generating hard shadows is restricted to point[5,6] and directional light (see Figure 2).

In general, we can consider hard-shadow determination just as difficult as visibility determination from the eye, because shadow determination is visibility determination with respect to the light source. However, hard-shadow determination is actually simpler to deal with. You do not need to calculate the closest visible object; just determine the existence of an object between the light source and the point of interest.

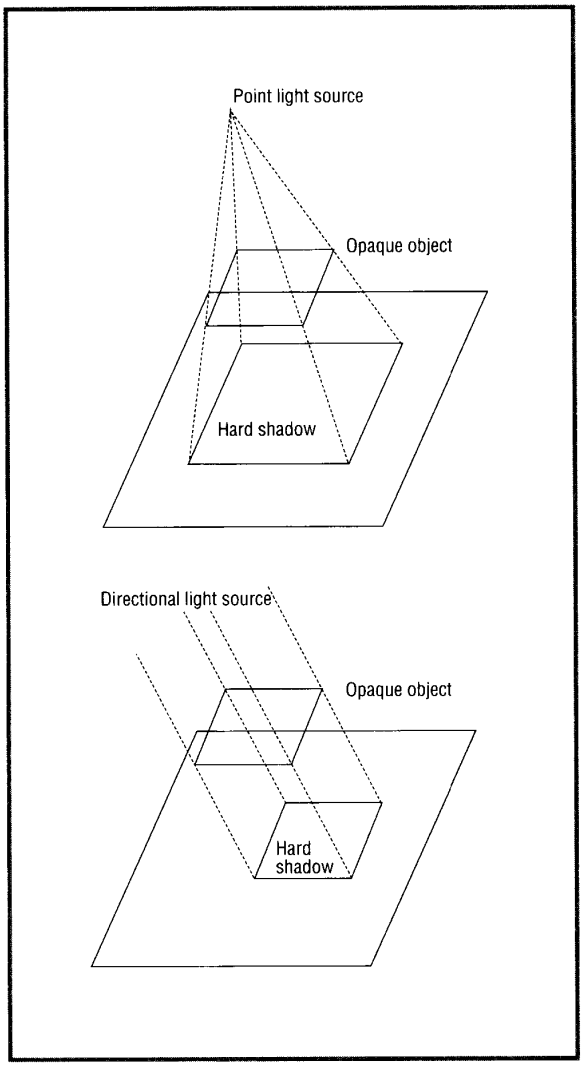


Figure 2. Hard shadows.

Fake shadows

The simplest approach employs special-purpose shadow algorithms that work only under certain circumstances. For example, you might use a real-time shadow generator that takes into account only shadows projected on a floor.[7] Such short-cut algorithms tend to be faster computationally than "honest" algorithms (discussed in the upcoming subsections) and can be very effective in the appropriate context, such as video games.

Shadow generation during the scanning phase

Appel[8] and Bouknight and Kelley[9] suggested shadow generation during the display phase using an extended scan-line approach. During the display phase, polygonal boundaries are projected down onto the scanned object to form shadow boundaries, clipped within the boundaries of the scanned object,

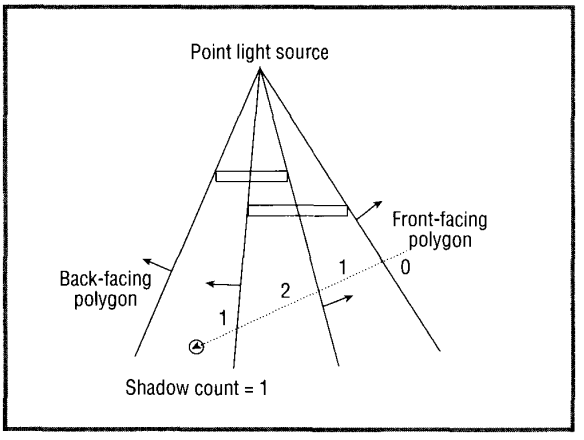


Figure 3. Shadow volume.

then projected onto the viewing screen. The intensity of a scanned segment changes as it crosses the shadow boundaries.

The storage usage and preprocessing for Bouknight and Kelley's method are $O(n^2)$ and $O((En)^2)$, respectively, since they perform some preprocessing to determine candidate occluding objects. However, Appel's method does not perform the preprocessing step, instead checking all objects during the scanning phase. The shadow rendering complexity for both methods is $O(En)$ per scanned segment (not per pixel). This shadow determination approach only suits polygons. In addition, it has not acquired the popularity of the shadow determination approaches discussed in the upcoming subsections.

Shadow volumes

Crow[1] proposed an approach to generate polygonal shadow umbrae from the objects in the scene, then place them into the rendering data structure as invisible objects. Many others[10-18] later applied some variation of the general principle to frame-buffer and scan-line approaches.

To compute shadow determination, use a shadow count. Calculate an initial shadow count by counting the number of shadow volumes that contain the viewing position. Then increment the shadow count by 1 whenever there exists a shadow front-facing polygon (that is, entering the shadow umbrae) crossing in front of the nearest visible object. Decrement the shadow count by 1 whenever there exists a shadow back-facing polygon (that is, exiting the shadow umbrae). If the final shadow count is 0, then the visible object does not lie in shadow; if positive, it does (see Figure 3).

The raw approach[1,12] requires $O(En)$ for storage and preprocessing, where En represents the number of shadow polygons. Shadowing rendering complexity requires depth processing of shadow polygons per pixel, which may get up to $O(En)$. A binary space partitioning (BSP) tree variation[15] requires $O(En)$ storage, with preprocessing complexity of $O(En^2)$ and rendering complexity of perhaps $O(\log(En))$ for BSP tree traversal. Other algorithms, such as presented by Fournier and Fussell,[14] require storage of $O(pq)$ to keep the shadow count updates. The shadow rendering complexity is constant, with preprocessing complexity of $O(Enpq)$.

Area subdivision

Nishita and Nakamae[19] and Atherton, Weiler, and Greenberg[20] used clipping transformations for polygon shadow generation. In this two-pass hidden surface algorithm, the first pass transforms the image to the view of the light source and separates shadowed and unshadowed portions of the polygons via a hidden surface polygon clipper. It then creates a new set of polygons, each marked as either completely in shadow or not. The second pass encompasses visible determination from the eye and shading of the polygons, taking into account their shadow flag.

The storage complexity might be quite high, depending on the spatial complexity of the scene. In the worst case, clipping one polygon over another results in the creation of E^2 polygons. Thus, the overall storage complexity considering the polygons in the scene is $O(E^2n)$. The preprocessing complexity of the algorithm due to the polygon clipping process is $O((En)^2)$. Although the shadow rendering complexity involves no additional cost, we might need to consider many more shadow and nonshadow polygons (compared to the original set of polygons without consideration of shadowing).

Note that the hidden surface polygon clipper must possess sufficient sophistication to handle convex and concave polygons with holes as a result of the clipping. In addition, it is algorithmically difficult to come up with a numerically robust polygon clipper as well as a clipper that deals with modeling primitives other than polygons. However, since this algorithm stores the shadow boundaries internally as polygons, the information can be sent to hardware shaders (hardware that processes and shades polygons). It can produce real-time shadows if the lights and polygonal database do not change.

Depth buffer

Williams[21] used a z-buffer depth map algorithm to generate shadows. The z-buffer approach to determine visibility and intensity with respect to the eye is repeated for the light source. Thus, it creates a buffer with respect to the point of view of the light source, except that the buffer contains only z depth values, not shading values or object information. The light source depth map is created before visible surface processing. During rendering, if the surface being shaded is not the nearest visible object with respect to the light source buffer view (meaning the z value in the buffer), then it lies in shadow; otherwise, it does not.

The storage complexity of this algorithm is $O(pq)$. The preprocessing time equals $O(Enpq)$, and shadow rendering complexity remains constant.

This algorithm has the obvious disadvantage that it can handle a maximum 180 degree shadow frustum per buffer. If you placed a light source inside the scene, then you might need six such buffers to handle all shadow cases. Thus, the algorithm deals most effectively with spotlights and directional lights. In addition, it has aliasing problems due to discretized depth map cells and orientation of the shadow z-buffer (a cell on the buffer with respect to the eye view differs in orientation and size from a cell with respect to the light source). However, the attractiveness of this approach is that it can trivially support primitives other than polygons.

Hourcade and Nicolas[22] and Reeves, Salesin, and Cook[23] attempted to improve on both aliasing problems by applying stochastic sampling and some prefiltering to the z-buffer approach. Reeves, Salesin, and Cook also claimed that their algorithm could generate soft shadows. However, note that this results from the sampling and filtering done and is not physically accurate because a point light source should only generate hard shadows (assuming no inter-reflections).

Some proposed algorithms use the idea of keeping the shadow information provided by projecting the scene in the light source direction.[24-26] Consider, for example, Robertson's proposed optimization of the scan-line algorithm for surface shadowing.[26] He suggested combining rotation of the scene along the light direction, alignment of the points that can occlude themselves along vertical scan lines, determination of the shadowed points, and a return to the original orientation. This optimized the shadow determination process because Robertson could simply use a composition of one-dimensional operations. However, his method, like many of the others based on projections, is highly subject to aliasing problems as the light source gets closer to the scene.

Ray tracing

Ray casting[8,27] was introduced as a method for visibility calculation and shadow determination. Ray casting involves shooting (or casting) a ray from the eye to each pixel, performing ray-surface intersections, and declaring the surface with the minimum hit distance to be the visible surface. This approach generalizes to the rendering technique known as ray tracing, popularized by Whitted.[28] This technique also models reflection and refraction and generates shadows.

During rendering, if the surface being shaded is not the nearest visible object with respect to the light source buffer view (meaning the z value in the buffer), then it lies in shadow; otherwise, it does not.

A simple principle underlies ray tracing for hard shadow determination: a shadow ray is shot from the intersection point to the light source. If the ray intersects any object between its origin and the light source, then it lies in shadow; otherwise, it does not.

Note that basic ray tracing requires no additional storage and preprocessing for shadow determination; shadow determination is lazily evaluated as needed. However, shadow determination complexity is very expensive and uses no coherence information: the cost equals $O(En)$ per ray shot, since ray-surface intersections are required for all surfaces. One shadow ray is shot per pixel initially, but this number may increase due to the need for reflection, refraction rays, or additional sampling for antialiasing.

The main advantage of ray tracing is that it handles shadow determination in an (almost) identical manner for cast, reflected, and refracted rays. Because it is the simplest among the hard shadow algorithms to implement, some software packages automatically switch the rendering to ray tracing when shadows are required.

Note that the ray tracing process is very floating-point intensive. Thus, the visibility and shadow tests

Figure 4. Soft shadows in cone tracing. Cone tracing can produce soft shadows. (Courtesy of J. Amanatides.)

might give incorrect results due to numerical errors, usually seen as little holes in the images. (Some refer to it as surface acne.) Amanatides and Mitchell[29] discussed further numerical error problems and offered solutions in their work.

Intersection culling algorithms for ray tracing

Since each ray requires intersection checks with all objects in the scene, research has gone into intersection culling. Intersection culling algorithms limit checking for intersection with all ray types to a small candidate set of objects. Many such culling algorithms exist. They include hierarchical bounding volumes,[30-33] variable size voxel traversal,[34,35] uniform size voxel traversal,[36-39] hybrid uniform-variable voxel traversal,[40,41] ray classification,[42] spatial coherence,[43,44] and ray coherence.[45,46] Analyzing the complexities of most of the approaches proves difficult. They usually much improve over the brute force $O(En)$ per ray shot. Typically they require some preprocessing but substantial storage.

Shadow culling algorithms for ray tracing

While the above algorithms perform culling with respect to all ray types including shadows, additional work has focused on culling shadow rays even further. These works take advantage of the realization that neighboring shadow binary decisions are usually the same. Preprocessing can potentially save a great deal of time in shadow intersection tests.

The light buffer,[47] hybrid shadow testing,[16] voxel occlusion testing,[17] and *zz*-buffer[48] exemplify such shadow cullers. The light buffer,[49,50] voxel occlusion testing,[17] and *zz*-buffer[48] have been extended to model soft shadows as well.

Soft shadow generation

Another type of shadow algorithm deals with soft shadows (illustrated in Figure 4), meaning the inclusion of the penumbra region along with the umbra for a higher level of visual quality (see Figure 5). Full occlusion from the light causes the umbra region, and partial occlusion from the light causes the penumbra region. The degree of partial occlusion from the light results in different intensities of the penumbra region.

Determining soft shadows requires calculating the fraction of opaque occlusion (which results in the different intensities of the penumbra region), not just a binary decision as for hard shadows. A fraction in the range (0, 1) is multiplied by the light intensity, where 0 indicates umbra, 1 indicates no shadow, and all other values indicate penumbra.

Note also that the resultant shadow region has a shape depending both on the occluding object and on the light source (see Figure 5). In addition, the types of light sources modeled are linear and area light sources.[51,6,50] Here, we consider only the soft shadow algorithms due to opaque objects. Assume the intensity is identical for every point on the light source.

Some have done work to generate soft shadows due to inter-reflections of diffuse light. Such work takes into account global illumination issues. It considers surfaces in the scene as secondary light sources, making possible the generation of soft shadows.[52-54]

Frame buffer algorithm

Brotman and Badler[10] stochastically chose points to model higher dimensional light sources. Their algorithm generates shadow umbra polygons for each such point source in the same manner as in Crow's algorithm[1] (see the section "Shadow volumes" above). This shadow polygon generation occurs during preprocessing. A 2D depth buffer for visible surface determination is extended to store cell counters.

Calculate the cell count (slightly different from Crow's shadow count) as follows: The cell in which the intersected point resides is found, and the associated counter is incremented by 1 if the shadow polygons for that particular point source enclose the whole cell. If the corresponding cell count equals the number of chosen point light sources, then the point lies in the umbra region. If the cell count is less than the number of chosen point light sources but higher than zero, then the point lies in the penumbra region.

Theoretically, the storage complexity is $O(pq)$ to store the cell counters. The algorithm has preprocessing complexity of $O(PEnpq)$, where P is the number of

point sources simulating the light source, and constant shadow rendering complexity. In general, the algorithm requires many such point sources. (Brotman and Badler rendered several images using about 100 points to simulate an area light source.) Even with a reasonable number of point sources, the most significant points on the light source with respect to the shaded point might not be considered because of the prechosen point sources. Thus, the inconsistency might cause some sparkle-like artifacts for highly specular surfaces, although perhaps not noticeable in diffuse environments. Similarly, some artifacts such as aliasing can appear in the shadow areas.

Recent display architectures have features to help implement this type of algorithm. The accumulation buffer, as described by Haeberli and Akeley,[55] allows the rendering of soft shadows by accumulating in a frame buffer the weighted shadows from point sources. They did not discuss how to correctly sample the light sources.

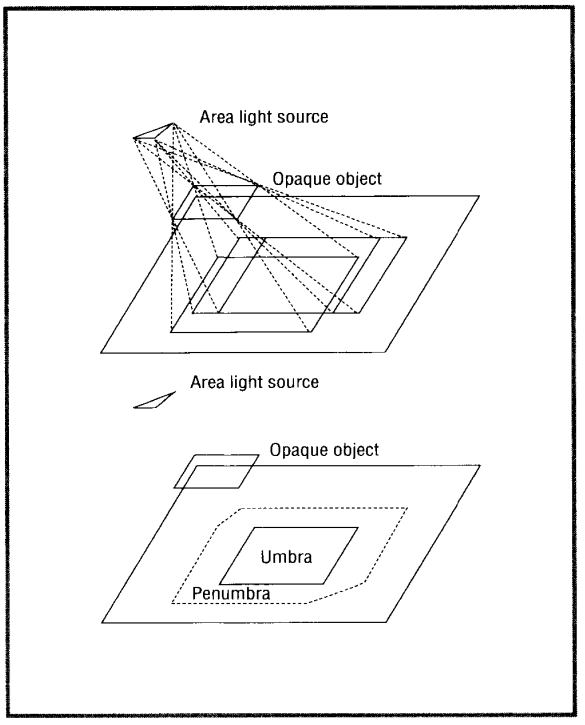


Figure 5. Soft shadows.

Distributed ray tracing

Cook, Porter, and Carpenter[56] proposed a distributed ray tracing method for soft shadow generation. Their method involves shooting a collection of shadow rays from the intersected point to randomly perturbed locations on the light source. The number of rays is proportional to the illumination of the region if it were completely unoccluded and to the projected area of the light source as seen from the surface. The intensity of the penumbra depends on the number of intersections of those rays with occluding objects.

As with traditional ray tracing, this method requires no storage or preprocessing. The shadow rendering complexity is $O(P\ En)$, where P is the number of shadow rays shot. The main attraction of this algorithm is that it deviates little from the traditional ray tracing implementation and allows for the generation of gloss (blurred reflections), translucency (blurred refractions), depth of field, and motion blur. However, because it is a point sampling approach, it might not always provide good approximations to the correct solution. Other works have discussed further stochastic schemes for distributed ray tracing.[57-60]

Cone tracing

Amanatides[61] extended the concept of a ray to a cone. Instead of point sampling, like previous approaches, cone tracing does area sampling. Achieving antialiasing requires shooting exactly one conic ray per pixel. Broadening the cone to the size of a circular (spherical) light source for shadow cones permits generation of soft shadows; a partial intersection with an object not covering the entire cone indicates penumbra.

Again, this method imposes no additional storage or preprocessing costs, and the shadow rendering complexity is $O(En)$. It has the benefit of shooting exactly one shadow ray, and the area sampling done should provide an acceptable approximation to the penumbra intensity. However, note that the approximation is physically valid only for circular light sources. Since the resultant shadow region depends on the shape of both the occluding object and the light source, cone tracing proves less suitable for light source shapes that cannot be closely approximated by one or more spheres.

This approach is very powerful in that it can provide antialiasing, soft shadows, and gloss, but the cone geometry calculations involved are more difficult than dealing with just rays, as in traditional ray tracing.

Area subdivision approach

Generally, you would compute soft shadows in point sampling ray tracing by shooting a set of distributed shadow rays to determine occlusions as well as

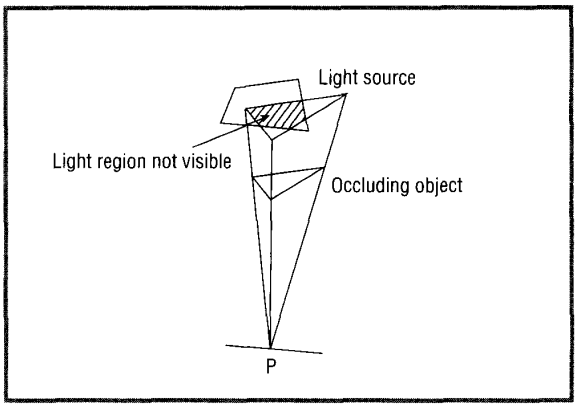


Figure 6. Area subdivision shadow tracing.

illumination. However, these rays are expensive because of ray-surface intersections and may not even allow a good approximation to the true solution.

If the only modeling primitive available is the polygon, we propose another method for soft shadow generation. Given the light source and the point P to be shaded, we can apply in reverse the polygon clipper described by Atherton, Weiler, and Greenberg[20] (see the subsection "Area subdivision" in "Hard shadow generation" above).

Using any intersection culler, you can easily acquire a candidate set of objects lying between P and the light. Project the candidate objects onto the light source as viewed from P, then clip them using Atherton, Weiler, and Greenberg's algorithm. After clipping all the candidate objects, identify exactly the region of the light source visible from P (see Figure 6). This region is then passed to any intensity integral solver. Whether the solver uses an analytic[6,50] or point sampling approach[51] does not matter; the important result is that you can rely upon the shadow calculations. You could also apply many coherence optimizations here.

Bidirectional ray tracing

Chattopadhyay and Fujimoto[54] proposed a fast method named bidirectional ray tracing to generate soft shadows (as a result of inter-reflections) using a uniform voxel structure. Their method involves shooting shadow rays from the light sources to surfaces (highly diffuse only) considered a potential secondary light. This lets you calculate the intensity contribution to the surface. Then interpolate this contribution for the vertices of the voxels that contain the surface.

As a second step, add an $(R + 1)^3$-bit factor to each voxel vertex, where $R \times R \times R$ is the resolution of the voxel structure. Each bit factor indicates whether vertex i is visible from the current voxel vertex. This handles diffuse inter-reflections of light: for each vertex i visible from the current voxel, the secondary light source contribution from the vertex intensity contribution is redistributed to the current surface. However, for each vertex i, you must calculate the secondary contribution, and so on. The authors claim that one or two iterations of this process usually prove sufficient. The final illumination value at any shaded point thus results from an interpolation of the vertex information and retracing of the contributions of the vertices.

The storage complexity of this approach is $O(R^6)$. The preprocessing necessary measures about $O(EnR^6)$, where R^6 accounts for the outer loop of calculations of the vertex bits and En is the worst-case cost for determining occlusion between the vertex points. Finally, the shadow rendering complexity is $O(SR^3)$, where S stands for the number of iterations of secondary illumination calculations.

This approach for soft shadow generation has a problem: The image results differ significantly for different voxel sizes due to the interpolation process between voxel vertices. Moreover, it can result in poor approximations, since the method does not take the surface orientations into account.

Radiosity

Another method of computing soft shadows comes from the calculation of radiosity.[62] This class of algorithms can calculate diffuse inter-reflections between surfaces by determining an equilibrium energy balance within an enclosure. So far, only polygonal surfaces have been used for this rather expensive algorithm.

Each surface (including a light source, which the algorithm also considers a surface) is subdivided into patches, where each patch i is assumed to possess a constant radiosity B_i. Depending on the orientation of patches i and j, the fraction of the radiosity B_i reaching patch j is given by a geometric form factor F_{ij}. To determine these form factors requires a determination of interpatch visibility. Once you know the form factors, solving a system of linear equations gives the radiosity, hence the shading, of each patch.

Hemicube

Two approaches described in the literature specifically handle soft shadows using the radiosity approach. The first approach, proposed by Cohen and

Greenberg,[52] computes form factors by attaching a hemicube to each patch (see Figure 7). The hemicube consists of five grid planes surrounding the center of a patch. Each grid pixel (referred to as a delta form factor) contributes to the overall form factor. It contains a pointer to the closest projected surface visible from the patch and intersecting the given grid pixel. Other surfaces are then considered occluded from the reflected light of the current surface at that pixel.

For a given grid pixel p, the closest patch projected onto this pixel might not cover it completely. Patches lying beyond this closest patch within the frustum of p are considered to lie completely in shadow, but they might not. Thus, the method might generate inaccurate shadows if there exist many visible patches with respect to the pixel, caused by the insufficiency of the hemicube resolution.

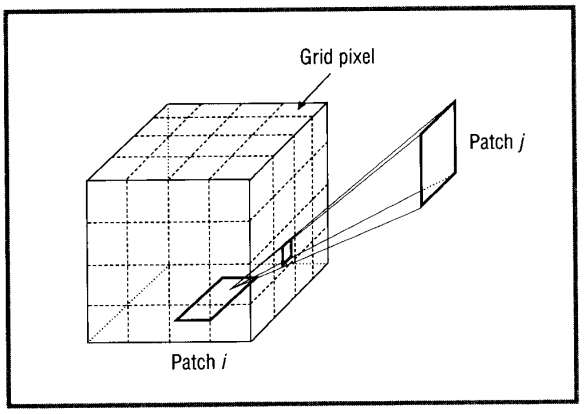


Figure 7. Hemicube.

Shadow polygons

Nishita and Nakamae[6,53] proposed another approach for soft shadow generation within radiosity. In the hemicube approach, the B_i values are calculated at the center of the patch. Nishita and Nakamae proposed calculating B_i at the vertices of the patches and doing shadow testing only at these points. To do so, they added two additional terms to the form factors: a weighting function taking into account the location of a point on a patch, and a shadowing function indicating the fraction of the patch k hidden from patch i by the other patches. They used shadow polygons to determine this shadowing function and identified the umbra and penumbra regions[6] (see Figure 5).

For each vertex of patch i, project shadow polygons towards all other patches j so that j now acts as the occluding object and i as the area light source. If a patch k lies inside the shadow umbra region, then no light can reach patch k from patch i; if patch k lies inside the shadow penumbra region, then light can partially reach patch k from patch i. Interpolating the luminance of each patch from the luminance at the vertices generates smoothed soft shadows.

Complexity analysis

The preprocessing complexity of the hemicube is $O(En^2R^2)$, where n represents the number of patches in the scene, $R \times R$ is the resolution of the hemicube, and EnR^2 accounts for the scan-conversion cost per hemicube. Nishita and Nakamae's approach has a cost of $O((En)^2I)$, where I is the cost of identifying the umbra and penumbra regions.

The calculation of the form factors resulting from the hemicube approach tends to be faster than that of Nishita and Nakamae's shadow polygons approach. However, the hemicube approach has the undesirable feature that the image results differ significantly with variations in the resolution of the hemicube.

Both approaches only approximate the level of shadowing. Which method provides more accurate results remains unknown.

Other radiosity approaches

Many others have written about radiosity. We do not list those works, since most do not specifically address shadow determination. However, note one exception: the use of ray tracing to replace the hemicube approximations,[63] which tends to be much more expensive though more accurate in shadowing effects.

Another exception appears in the work done by Campbell and Fussell.[64] They adaptively subdivided the scene along the shadow boundaries in a progressive refinement scheme. Their method subdivides a light emitter until each element can be treated as a point light source. It then uses BSP shadow volume generation, similar to that described by Chin and Feiner,[15] to efficiently subdivide the receiver polygons for each element on the emitter.

The quality of the shadows (especially sharpness) thus created greatly exceeds that of the previous techniques. However, you must seriously consider the time and storage complexity of their approach if you expect many iterations of the progressive refinement. Assume P points approximate an emitter and k is the number of iterations. Then the number of edges used to build a shadow BSP-tree on a single receiving polygon is $O(kPEn)$, giving a worst case size of $O((kPEn)^2)$ for the shadow BSP-trees. For $O(n)$ receiving polygons, this gives a total size of $O(k^2P^2E^2n^3)$. Keep in mind that P can be large and that k can be $O(n)$ for a large number of strongly emitting or reemitting poly-

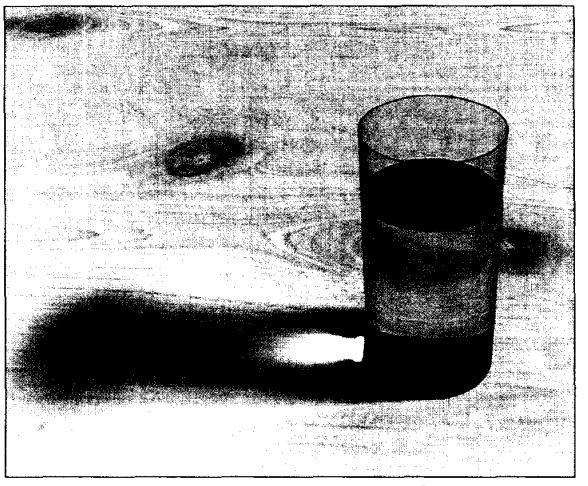


Figure 8. Shadows from transparent objects. Correct shadows produced by transparent objects are hard to calculate. This image illustrates the shadow and the concentration of light produced when the light goes through a glass of water. Pencil tracing was used to simulate this effect. (Courtesy of M. Shinya of NTT.)

gons. The above complexity is only for the storage; processing is even larger.

Skylight illumination

Nishita and Nakamae[65] contributed the concept of adding skylight to soft shadows. With this method, for each point you want to shade, generate a hemisphere representing the sky and define band sources longitudinally on this hemisphere. By assuming a uniform luminance of the sky within each band, you can sample the luminance only over the center line of each band. You can interpolate the luminance of the sky from clear to overcast. Every object in the scene is projected onto the sample lines to compute the obstructed factor. If the point you want to shade lies on a tilted surface (with respect to the scene), this method tilts the hemisphere by the same angle and considers the light reflection from the ground for this tilted angle. The method also extends to take into account the skylight entering interiors.

Shadows from transparent objects

One of the difficult problems encountered while generating realistic shadows is dealing correctly with shadows from occluding transparent objects. When light goes through a transparent object, the object can change the properties of the incoming light, for instance, its direction and color. You must take these effects into account to generate correct shadows.

One aspect in determining the shadows from transparent objects is the presence of concentration or diffusion of light passing through this type of object. One of the most noticeable effects is the presence of caustics. Caustics in optics are defined as the envelope of the rays going through an area. This creates highlights when light passes through a convex lens (illustrated in Figure 8) or reflects from a concave mirror. However, this effect, which many algorithms try to capture, tends to be computationally expensive.

Determining shadows from transparent objects proves more complex in that you need to find not only the first occluding object (if transparent), but all possible occluding objects in the direction of the light source(s). Determining shadows from transparent objects is also more complex than the visibility problem, since transparent objects not located between the point to shade and the light source(s) can contribute to the final intensity within the shadow region.

We must mention that the colored region produced from occluding transparent objects is not really a shadow. A shadow results from occlusion from light, but the colored region results from transmitted light. However, since previous works have considered these regions to fall in the class of shadows, and many of the techniques are similar, we will also study the algorithms that generate such effects.

Note that, with the exception of the discussion in the next section, all the algorithms discussed for transparent shadow generation can also generate soft shadows.

Shadow ray

Traditionally in ray tracing, shadows that result from occluding transparent objects are handled by detecting all occluding objects and returning the fraction of light transmitted through the objects.[28] This won't work for transparent objects because the shadowing function is wavelength dependent. Moreover, it does not take into account the refraction of incoming light.

Hall and Greenberg[66] (and likewise Lee, Redner, and Uselton[58] in images they generated) dealt with the display of colored transparent objects using the spectrum absorption model, which accounts for scattering of refracted light sources in ray tracing. They did not explicitly mention the shadowing effects of transparent objects. However, one of their images showed a green transparent object projecting a green shadow on an opaque surface. The implementation used approx-

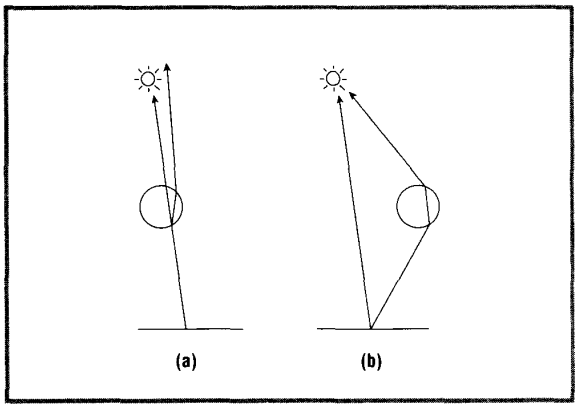


Figure 9. Incorrect shadowing of transparent objects.

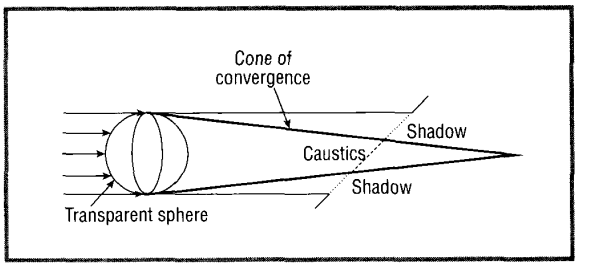


Figure 10. Concentration from parallel rays.

imate transparency without considering refraction: They shot a single shadow ray to the light source, and all transparent objects that intersected the ray contributed to the final shadow color through color filtering of all the occluding transparent surfaces. They considered the Fresnel reflectance at the intersection of the ray with the surface to decide the percentage reflected and refracted into the object.

A simple trick to simulate the light attenuation from the shadow of transparent objects was described by Pearce.[67] He attenuated the light as a function of the angle between the shadow ray and the normal at the intersection between the shadow ray and the object.

Backward ray tracing

Correctly generating transparent shadows in ray tracing just by applying a single shadow ray proves very difficult. Because of refraction, the shadow ray aimed towards the light source bends as it enters and exits transparent objects, and it might not even reach the light source (see Figure 9a). Similarly, other ray paths that go through transparent objects might contribute to the shading value but are not part of the shadow ray (see Figure 9b). The above two scenarios will result in incorrect shadowing if a shadow ray is shot to the light source.

This suggests you need the tracing of rays from the light source, so that all refracted light reaching any point is considered. This general approach should theoretically deal with diffuse inter-reflections and generate concentration and diffusion of light as well. This approach resembles the backward ray tracing approach proposed by Arvo[68] (assuming only point light sources). Stochastically chosen rays are shot from the light sources as a preprocessing step, and the amount of light that reaches the surfaces is recorded via a texture map storage scheme.

When shooting rays from the light, you need to shoot a very large set of rays in all directions to accurately account for diffuse inter-reflections. Thus, Arvo's approach, and that of some others,[69-71] requires a great deal of storage and processing. In addition, the generated images usually exhibit aliasing and sparkle-like artifacts due to the insufficiency of the sampling done. Watt[72] used a technique similar to beam tracing from the light, but the concentration of light in the resulting images betrayed their polygonal origin.

Cone of convergence

Inakage[73] used ray tracing to model the concentration of light passing through a convex lens. He illustrated a limited effect where a spherical lens projects a shadow with a concentration of light within the shadow area. Given parallel rays of light striking a perfectly spherical lens (that is, given a directional light source), he assumed a resulting convergence of light intensity at a focal point beyond the object.

The cone of convergence (see Figure 10) is defined as the volume swept by the joining of the focal point and the silhouette of the sphere. If the point to shade falls within this cone of convergence, then it lies within caustics, and intensity at that point increases. Note that you would calculate the cone of convergence during preprocessing and must do so for each transparent sphere per light source.

This approach has many problems. First, Inakage did not consider the light concentration beyond the cone of convergence and assumed a uniform concentration of light within the cone of convergence. Second, the algorithm does not deal with occlusion of these light rays, which results in incorrect shadowing

effects. Finally, assuming a parallel set of incoming rays means the resultant convergence is valid only for a single thin lens, but not for a sphere of arbitrary radius.

Paraxial approximation theory, presented in the next section, deals more effectively with the generation of caustics.

Pencil tracing

Applying fundamentals of paraxial approximation theory, Shinya, Takahashi, and Naito[74] proposed another method for handling transparency: concentration and diffusion of light. The axial ray surrounded by nearby paraxial rays defines a pencil.

Pencil tracing—basically an intermediary between cone tracing and ray tracing—shoots a group of rays in a small solid angle.

Pencil tracing is basically an intermediary between cone tracing and ray tracing. Ray tracing traces lines, so only point-sampled information can be gathered. Cone tracing traces a single conic pencil to get a better approximation. Pencil tracing shoots a group of rays in a small solid angle, and sampled information can be grouped together to get an even more precise solution than previous attempts. Divergence (Figure 11a) and convergence of light (Figure 11b) result naturally from this algorithm.

To generate shadows from transparent objects, this method shoots pencils from the lights to the bounding volumes of all transparent objects, then propagates them to the surfaces. This takes refraction into account, but still only first-generation transmitted light.

This processing requires shadow rendering complexity of $O(QEn)$, where Q stands for the average number of pencils shot from the light source to each transparent object.

Shinya, Saito, and Takahashi proposed an extension of the approach including wavelength dependency.[75]

The rendering equation and path tracing

Kajiya[76] proposed an approximation to the rendering equation that uses ray tracing. Named path tracing, it only traces certain secondary rays for computational speedup. The rays traced must maintain the correct proportion of secondary ray types contributing to each pixel. These ray types include reflection, inter-reflection, refraction, and shadow, where inter-reflection rays are shot stochastically into the environment. However, choosing only certain secondary rays can conceivably result in erroneous shadows and undesirable light effects.

Light-driven global illumination

Proposed as a light-driven approach to global illumination (for both the specular and diffuse components), Fiat[77] handles inter-reflections, inter-refractions, shadows, and concentration and diffusion of light by balancing the power of regions of space. Basically, a hierarchical data structure envelops the scene, in this case an octree. The goal is for the region occupied by each octree cell to achieve an approximate power balance. Thus, in each cell, the power emitted minus the power absorbed must approximately equal the power distributed to all the other cells minus the power acquired from all the other cells.

To keep track of the traversal of light power within regions, each octree is surrounded by six $R \times R$ arrays of cells called sexells. Light traversing an octree region must pass through the sexells. The same goes for reflections and refractions of light within the octree cell calculated using ray-surface intersections. Each sexell is associated with a hemisphere of discretized directions that carries information on the power transported in each direction. When some power is transferred to an adjacent cell, that particular octree cell contains new information on the power inheritance and must be balanced. Selecting the least balanced cell to adjust first allows the use of a progressive refinement approach in this balancing. Iterations of these power adjustments continue until the octree regions are sufficiently close to a power balance. An initial rendering step allows for sharp shadows from point and directional light sources.

Fiat, like other approaches handling inter-reflections, requires a great deal of memory and computational resources. The other disadvantage of the algorithm is that it uses discrete directions to store information in the sexell. Conceivably, illumination might be slightly distorted or shifted if large amounts of activity (such as many objects, many inter-reflections) lie between neighboring sexell directions.

Shadows of complex surfaces

The shadows dealt with so far in this article include the shadowing on and from objects without concern for the surface definition of these objects. In this sec-

tion, we discuss problems related to the shadowing evaluation of parametric and implicit surfaces. In addition, we present shadowing on and of texture-mapped surfaces, as well as self-shadowing from facets.

Shadows of parametric and implicit surfaces

Parametric and implicit surfaces can be rendered either by direct numerical techniques or by polygonization. Unfortunately, neither method does a totally satisfactory job in rendering them accurately.

Numerical iteration techniques

Numerical iteration techniques solve for the visible surface directly and should therefore theoretically provide more accurate results. However, they are expensive to evaluate and difficult to implement robustly. We present only brief descriptions of a few approaches here to give an insight into the difficulties encountered. This section is by no means thorough.

Various methods exist to evaluate implicit surfaces directly. As an example, Blinn[78] used a combination of Newton's iteration and *regula falsi* to render algebraic surfaces. However, convergence to a proper solution cannot be guaranteed unless you can generate an accurate start point; divergence shows up as holes in the visible surface and shadows. Hanrahan[79] derived analytic solutions for low-order polynomial surfaces, but not general enough to handle other formulations. Two recent works used interval analysis to guarantee correct intersection results for certain classes of implicit surfaces,[80,81] which also improves shadow determination.

Rendering parametric patches can be just as difficult. Kajiya[82] ray traced parametric bicubic patches by combining two sets of equations into an 18th degree polynomial. In his original paper, Kajiya used Laguerre's root-finding algorithm, and the overall performance was rather poor. However, the method improves with a faster root finder and has become a little more competitive.[83]

Sweeney and Bartels[84] used a multivariate Newton's method to calculate roots, but it tends to be numerically unstable. This method does not converge if the ray is perpendicular to the surface normal at the intersection point (since it needs the Jacobian of the functions). This problem appears at the silhouette of the shadow.

Toth[85] used interval arithmetic to arrive at suitable roots. This tends to be more stable than most other methods, but is quite slow (except compared to Kajiya's method).

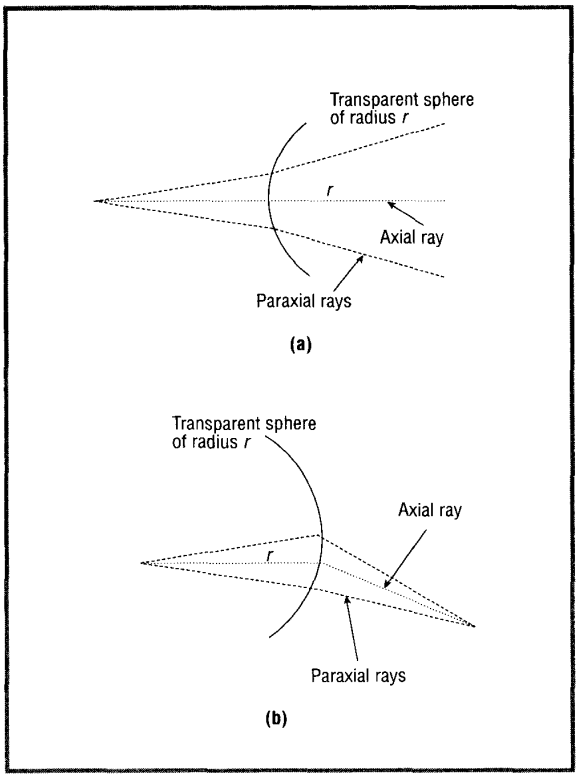


Figure 11. Paraxial approximation theory.

Fournier and Buchanan[86] used the properties of Chebyshev polynomials as bases both to get improved boxing and to get better criteria to stop the subdivision. At the end of the subdivision, a bilinear subpatch is intersected with a guaranteed tolerance on parametric values.

Polygonization

Parametric and implicit surfaces are often rendered by polygonization (decomposition of the surfaces into polygons). Polygonal decomposition has the advantage of simplicity in rendering polygons efficiently. However, a uniform subdivision scheme may either subdivide too much for low curvature regions or too little for high curvature regions. Adaptive subdivision of patches is sometimes used, but you must take care to avoid cracks between polygons resulting from different neighboring polygon subdivision levels. A common practice involves subdividing until the decomposed polygons meet certain flatness criteria. Two adjacent polygons meet the flatness criteria if the angle between their normals or tangents is sufficiently small.

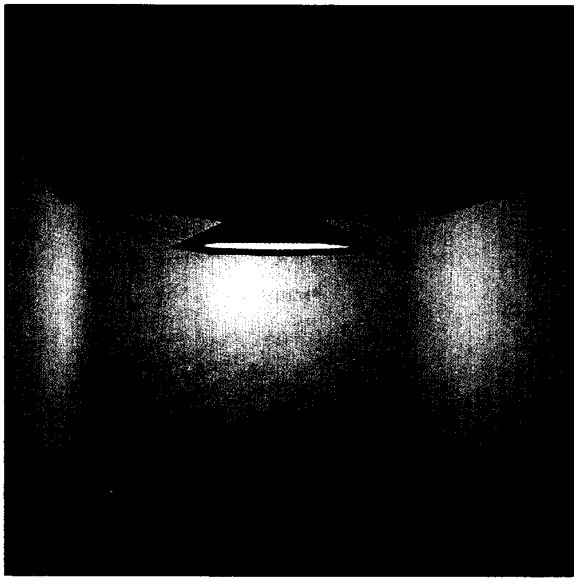


Figure 12. Polygonization. Polygonization offers one answer to the problem of rendering some primitives. However, it has some drawbacks, some of them illustrated in this image. Notice the silhouette of the sphere and the shadows of the lamp shade on the walls.

One problem with the polygonization of surfaces is that objects do not appear smooth. Within each polygon, interpolating the surface normals can reduce this effect. Unfortunately, shadow calculations do not have sufficient interpolation information, thus resulting in polygonal shadow silhouettes (illustrated in Figure 12).

Snyder and Barr[38] proposed an improvement by subdividing more at the silhouette of the surface (a function of the light source position) to reduce this artifact. That had some success. But multiple light sources introduce the danger of severe surface subdivision.

Max[87] also attempted to smooth polygonal silhouettes as well as shadows from polygonal edges.

Polygonization also has the terminator problem, where incorrect shadow determination results from the polygonal approximation. A point on a convex surface is lit if the angle between the normal at this point and the light direction is less than $\pi/2 (\vec{N} \cdot \vec{L} > 0)$. In the case of concave surfaces, a point can lie in shadow even if $\vec{N} \cdot \vec{L} > 0$. Then testing determines whether a ray shot in the light direction will intersect the object again. A simple method to avoid checking this intersection consists of tessellating the concave surface into many polygons. Figure 13 illustrates the terminator problem.

To keep smooth self-shadowing while interpolating normals (the terminator problem) in ray tracing, Snyder and Barr[38] proposed starting the shadow ray further from the intersection point. However, this can cause further artifacts if poorly approximated.

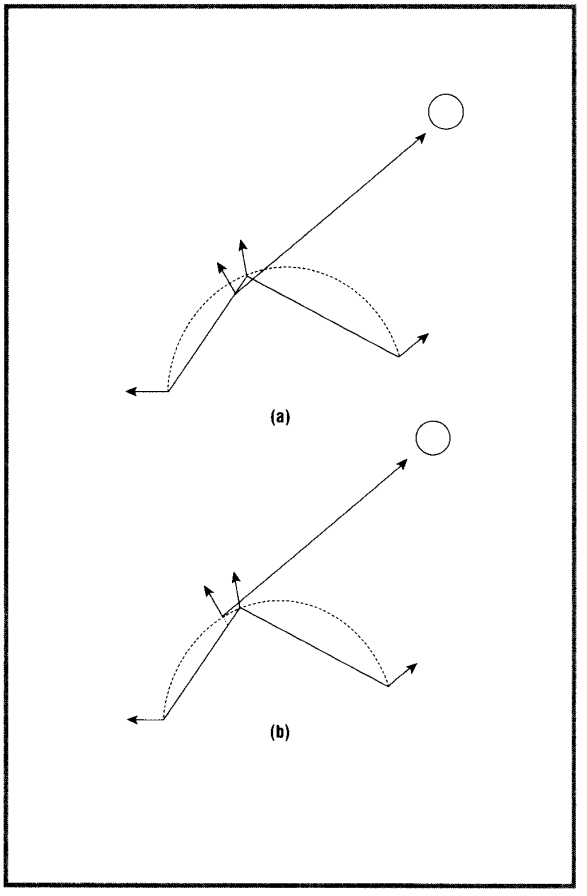


Figure 13. Terminator problem.

Shadows on texture-mapped surfaces

Texture mapping offers a very powerful approach for providing detail and realism to surfaces at a low cost for modeling and rendering. Some texture mapping aims to give the appearance of a bumpy surface. For example, in bump mapping[88] the surface normal is perturbed to make the surface appear bumpy, without altering the surface itself.

Two problems arise with shadows using bump mapping: the shadow cast on the surface by other surfaces and the shadow from bumps onto other surfaces and

on itself. Another form of texture mapping, transparency mapping, also often exhibits incorrect shadowing effects if care is not taken.

Shadows on bumpy surface

Perturbing the surface normal makes the surface appear bumpy. This, in effect, also implicitly perturbs the visible surface. However, shadow determination as applied to the bumpy surface has been dealt with, assuming the visible surface region is not perturbed (since the shadow determination is done independently of the surface normal). See Figure 14. This problem proves a difficult one because reconstruction of the surface bumpiness from surface normals is necessary for an accurate solution of the shadowing. No good solutions have yet been proposed.

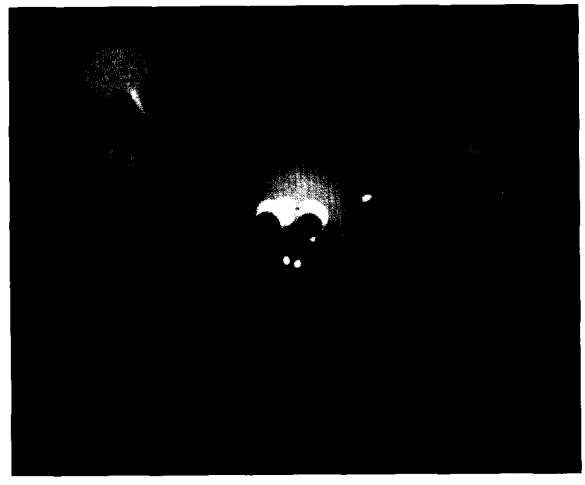


Figure 14. Shadows on bump-mapped surfaces. The normals of the surface plane have been modified by a bump map function. Notice the shadows of the algebraic surfaces onto the plane; the shadows are sharp and regular, as if the normals have not been modified on the plane.

Self-shadowing of bumpy surfaces

In displacement mapping, instead of the surface normal, the height of the surface is modified. For each point on a surface, there exists a corresponding height value by which this point would be raised on the surface. Max[89] approximated the shadows cast by these bumps on the same surface by introducing horizon mapping. Interpreting the bump function as a 2D table of height values, Max computed and stored, for each height value, the angle between the horizon and the surface plane at eight azimuthal directions on the surface plane. During the rendering stage, the horizon angle at the intersection point is interpolated from the light direction and the horizon map. If the horizon angle from the surface normal exceeds the light angle, then this point lies in shadow.

Max also proposed a correction of the horizon angle for curved surfaces and penumbra generation from circular area light sources. The aliasing of the shadows (due to discrete sampling) can be reduced using coarser bump functions, chosen depending on how much of the bumps are covered by a pixel. Although the technique might quite likely miss isolated narrow peaks, the overall self-shadowing effect is greatly improved.

In some cases of analytic surface maps, the function defining the horizon angle can be evaluated analytically. Similarly, for statistically defined surface maps, the horizon angle can be approximated based on probabilistic theory. In these cases, many problems that occur because of the discrete sampling just disappear.

Transparency mapping

Transparency mapping modifies the level of opaqueness of the surface. However, just about all software that provides both features of shadows and transparency mapping ignore the combination of opaque and transparency shadow effects generated from such surfaces. This effect is difficult to generate using solely preprocessing shadow algorithms. It appears that evaluation of mapping is required on the fly, which proves easy for some forms of ray tracing. But the evaluation of the mapping for shadow determination increases computations.

Self-shadowing of facets

Smaller and denser bumps over a surface prevent us from perceiving the bumps and the shadows. However, the reflection of light behaves differently, and the local reflection model should capture this.

In a reflection model proposed by Torrance and Sparrow,[90,91] the surface bumps are represented by a randomly oriented collection of mirror-like facets. Assuming that the facets take the form of V-shaped grooves, the model can approximate the self-shadowing effect on them.

However, for many types of surfaces, the facets exhibit preferred orientations. Then the shadowing effect does not depend only on the angle between the light and the surface normal, but also on the orientations of the facets relative to the surface. In the case of simple distributions of facets, the attenuation factor

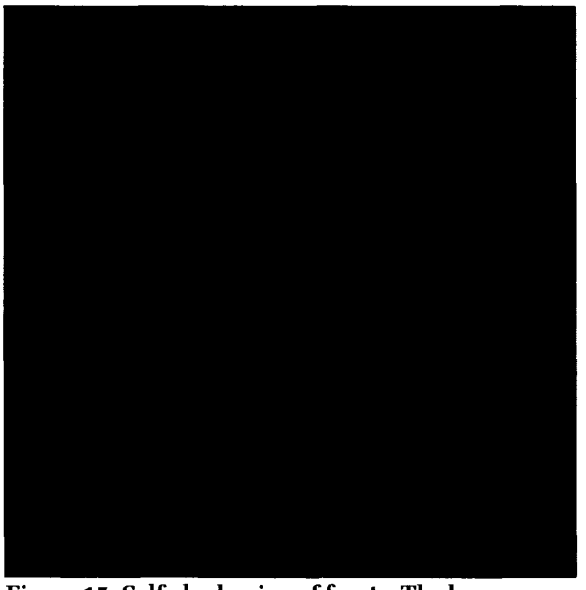


Figure 15. Self-shadowing of facets. The bumps over a surface can be so small that you cannot see them. Depending on the orientation of these facets, reflection of the light and shadowing differs. This image shows a simulated Christmas ball. The facets are defined as threads going from one pole of the sphere to the other. Notice the darker area near one pole.

can be computed analytically. For instance, Poulin and Fournier[92] computed the self-shadowing caused by adjacent cylinders simulating an anisotropic surface (illustrated in Figure 15). Cabral, Max, and Springmeyer[93] used an approach similar to that of Max[89] to compute a bidirectional reflection map from a height field applied to the facets distribution.

Shadows for particle-based objects

To render realistic images from nature, the modeling of scenes would require a tremendous amount of work just to specify the scene description. Systems have been developed to procedurally and stochastically transform a small set of simple constraints into a complete description of very complex objects (such as fire,[94,95] trees, and grass[24]). A simple graphics primitive, the particle, is used to create this complexity.

Such systems introduce new dimensions to the shadowing problem. Scenes composed of millions of particles become a formidable task to shade exactly. Probabilistic schemes must then be extended to the shadowing of each particle.

Particle systems

Reeves and Blau[24] used a particle system to generate images of trees (see Figure 16). The self-shadowing of branches within a tree is associated with the ambient term of the illumination model. The value of this term decreases exponentially as the particle location gets deeper within the tree. To determine the shadows cast by other trees onto one tree, Reeves and Blau defined a horizon line using the locations of the tops of the trees positioned between the light source and the chosen tree. This line and the light source direction define a shadow plane. If a particle lies under this plane, it has a higher probability of being shaded with only the ambient term.

The shadows cast from trees onto the ground are determined with a shadow mask. First, compute an orthographic view of the trees from the light. When rendering a particle on the ground, invoke a test with the shadow mask to determine if the particle lies in the shadow of trees. However, calculating the true shadowing requires tests with objects other than trees.

In reality, for each shadowing particle system you must determine the shadowing stochastically, but in close relation with the modeling process of the particles. Thus, to some extent, each particle system must have its own shadowing determination algorithm.

Volume densities

Blinn[96] simulated the interaction of light with clouds (and dusty surfaces) by using statistical simulation of light passing through and reflected by clouds composed of small particles, considering only single scattering. The shadowing results from other particles' blockage of light. If you assume that all particles have constant radius, you can use any statistical process (in this case, a Poisson process) to model the probability that a particle is completely illuminated and that the reflected light being in the view direction does not intersect any other particle. Thus, an analytic solution to shadow determination is available.

Kajiya and Von Herzen[97] proposed a more general model and applied it to ray tracing. Their single scattering model resembles Blinn's. As a preprocessing step, create a 3D grid of voxels. Each voxel represents a point in 3D space and contains the contribution of each light to the brightness of each point in space (the density distribution). During actual ray tracing, determine the brightness of the ray by summing the contribution of each voxel element that the ray pierces.

The main computational cost of this approach comes from the integral evaluation of the ray bright-

ness. Note also that the 3D grid of density distribution information requires a great deal of memory.

Participating medium

The treatment of volume densities easily extends to media that cause emission, scattering, and absorption of the light.

Max[13] modeled atmospheric illumination assuming a volume density model like Blinn's[96]: low albedo (reflective power) and single scattering. Thus, an analytic solution for the scattered energy at each point is possible. Computation of scattered energy requires information about the portions of the viewing ray that is shadowed, and an extended shadow polygon approach[1] can be used here. If the ray falls completely in shadow, then the illumination value is a function of the surface color and ambient illumination. If the ray is partially illuminated, then the illumination value is also a function of the scattered energy reaching the illuminated portion of the ray.

Efficient implementations of this approach using regular grid subdivisions were described by Woo and Amanatides[17] and Ebert and Parent.[98] More general models of density distribution appear in work by Nishita, Miyawaki, and Nakamae[99] and Ebert and Parent.[98]

Rushmeier and Torrance[100] applied the zonal method used in heat transfer to the computation of radiosity. Their method discretizes the medium into small volumes for which the form factors volume/volume and volume/surface are calculated. The shadows are generated with the hemicube (described earlier) extended to a full cube. The complexity of the algorithm is the same as the general radiosity approach, taking into consideration the increase of volume/volume and volume/surface form factor calculations.

Conclusions

The many algorithms examined in this survey give a common impression: that shadow determination is an expensive process. This is not, of course, very surprising when you consider that the simplest shadow problem is a version of the visibility problem and that while a scene has only one viewpoint, it can have many lights.

Another unfortunate conclusion is that once you have determined that a region lies in shadow, you have not reached the end of the problem. You then have to decide how to modify the illumination accordingly. This proves especially difficult in the case of soft shadows, where the real answer results from a convolution between the occluding object and the light. The survey also makes clear that generating accurate shadows from transparent objects, which can diffuse and concentrate the light, proves extremely difficult.

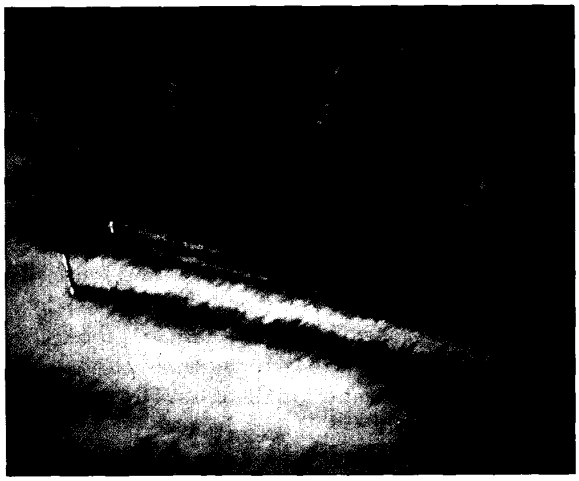


Figure 16. Andre's forest. This scene of a forest was generated by a particle system. Notice the self-shadowing of branches of a same tree and the shadows of the trees projected onto the grass. (Courtesy of W. Reeves of Pixar.)

The three basic factors to consider in the choice of a shadow algorithm are (1) the rendering technique used, (2) the modeling primitives used, and (3) the degree of physical accuracy needed. Other constraints usually decide the first two factors, leaving only the third to be decided by the shadow maker. To some extent, one can question the importance of generating physically exact shadows in imagery for many application fields. Some kludges, like smoothing the polygonal shadows via interpolating curves or slowly increasing the intensity near the edges of the cast shadows (similarly with the penumbra region), could improve the quality of the images and preserve information about the scene. On the other hand, such kludges require some additional intelligence or heuristics from the system. This is worth considering before involving more research and computing time in finding an exact solution to the problem of shadow determination.

Several directions for future research emerge from this survey. While many algorithms exist for shadows of polygonal objects, very few of the high-performance display systems in laboratories or available on the market incorporate shadows, even though almost all are based on polygonal primitives. A notable exception, Pixel-Planes,[11] can compute shadows with a

version of the shadow-count algorithm. Some version of the algorithms described in the section "Shadow volumes" above should be applicable to most other systems as well, since they all use z-buffers for visibility. Another area needing more results involves shadows of complex primitives, such as parametric and implicit surfaces. While subdividing such surfaces into polygons seems reasonable and efficient for rendering their visible parts, it looks rather wasteful for shadows, since you only need the outline of the shadow area and since each polygon costs $O(E)$ for processing shadows in most algorithms. In the same category, we see room for improvement in shadows of textured and bump-mapped surfaces.

Finally, in this survey we have only discussed shadow determination within a single image. Are additional optimizations achievable during an animation? While some of the algorithms discussed here do not require (or require little) additional processing for a fly-by animation, few animations can rely on the assumption of a static scene. Other works have discussed optimization of shadow determination for certain animations.[33,101,102]

We hope this survey helps cast an even larger shadow on computer graphics. ■

Acknowledgments

We would like to thank Mikio Shinya of Nippon Telegraph and Telephone (NTT) and Marie-Claire Forgue of the National Research Institute for Computer and Robotics Science (INRIA) for carefully reading an early draft of our paper and suggesting some improvements. Thanks also go to Andrew Pearce and Maria Raso of Alias for checking over the later drafts of the paper and to Rick Speer of Animation Research for suggesting additional references. We are also grateful to the National Science and Engineering Research Council (NSERC), the Information Technology Research Center (ITRC), and the University of British Columbia for their financial support.

References

1. F. Crow, "Shadow Algorithms for Computer Graphics," *Computer Graphics* (Proc. SIGGRAPH), Vol. 11, No. 3, Aug. 1977, pp. 242-248.
2. J. Amanatides, "Realism in Computer Graphics: A Survey," *IEEE CG&A*, Vol. 7, No. 1, Jan. 1987, pp. 44-56.
3. N. Thalmann and D. Thalmann, *Image Synthesis, Theory and Practice*, Springer-Verlag, New York, 1987, pp. 156-169.
4. J. Foley et al., *Computer Graphics, Principles and Practice*, 2nd ed., Addison-Wesley, Reading, Mass., Aug. 1990, pp. 745-814.
5. D. Warn, "Lighting Controls for Synthetic Images," *Computer Graphics* (Proc. SIGGRAPH), Vol. 17, No. 3, 1983, pp. 13-21.
6. T. Nishita, I. Okamura, and E. Nakamae, "Shading Models for Point and Linear Sources," *ACM Trans. on Graphics*, 4(2), April 1985, pp. 124-146.
7. J. Blinn, "Jim Blinn's Corner: Me and My (Fake) Shadow," *IEEE CG&A*, Vol. 8, No. 1, Jan. 1988, pp. 82-86.
8. A. Appel, "Some Techniques for Shading Machine Renderings of Solids," *Proc. AFIPS JSCC*, Vol. 32, 1968, pp. 37-45.
9. W. Bouknight and K. Kelley, "An Algorithm for Producing Half-Tone Computer Graphics Presentations Shadows and Movable Light Sources," *AFIPS Conf. Proc.*, Vol. 36, 1970, pp. 1-10.
10. L. Brotman and N. Badler, "Generating Soft Shadows with a Depth Buffer Algorithm," *IEEE CG&A*, Vol. 4, No. 10, Oct. 1984, pp. 71-81.
11. H. Fuchs et al., "Fast Spheres, Shadows, Textures, Transparencies, and Image Enchancements in Pixel-Planes," *Computer Graphics* (Proc. SIGGRAPH), Vol. 19, No. 3, July 1985, pp. 111-120.
12. P. Bergeron, "A General Version of Crow's Shadow Volumes," *IEEE CG&A*, Vol. 6, No. 9, Sept. 1986, pp. 17-28.
13. N. Max, "Atmospheric Illumination and Shadows," *Computer Graphics* (Proc. SIGGRAPH), Vol. 20, No. 4, Aug. 1986, pp. 117-124.
14. A. Fournier and D. Fussell, "On the Power of the Frame Buffer," *ACM Trans. on Graphics*, Vol. 7, No. 2, April 1988, pp. 103-128.
15. N. Chin and S. Feiner, "Near Real-Time Shadow Generation Using BSP Trees," Computer Graphics (Proc. SIGGRAPH), Vol. 23, No. 3, July 1989, pp. 99-106.
16. D. Eo and C. Kyung, "Hybrid Shadow Testing Scheme for Ray Tracing," *Computer Aided Design*, Vol. 21, No. 1, Jan. 1989, pp. 38-48.
17. A. Woo and J. Amanatides, "Voxel Occlusion Testing: A Shadow Determination Accelerator for Ray Tracing," *Graphics Interface 90*, May 1990, pp. 213-220.
18. F. Jansen and A. van der Zalm, "A Shadow Algorithm for CSG," *Eurographics 90*, Aug. 1990, pp. 51-61.
19. T. Nishita and E. Nakamae, "An Algorithm for Half-Tone Representation of Three-Dimensional Objects," *Information Processing in Japan*, Vol. 14, 1974, pp. 93-99.
20. P. Atherton, K. Weiler, and D. Greenberg, "Polygon Shadow Generation," *Computer Graphics* (Proc. SIGGRAPH), Vol. 12, No. 3, Aug. 1978, pp. 275-281.
21. L. Williams, "Casting Curved Shadows on Curved Surfaces," *Computer Graphics* (Proc. SIGGRAPH), Vol. 12, No. 3, Aug. 1978, pp. 270-274.
22. J. Hourcade and A. Nicolas, "Algorithms for Antialiased Cast Shadows," *Computer and Graphics*, Vol. 9, No. 3, 1985, pp. 259-265.
23. W. Reeves, D. Salesin, and R. Cook, "Rendering Antialiased Shadows with Depth Maps," *Computer Graphics* (Proc. SIGGRAPH), Vol. 21, No. 4, July 1987, pp. 283-291.
24. W. Reeves and R. Blau, "Approximate and Probabilistic Algorithms for Shading and Rendering Structured Particle Systems," *Computer Graphics* (Proc. SIGGRAPH), Vol. 19, No. 3, July 1985, pp. 313-322.
25. G. Miller, "The Definition and Rendering of Terrain Maps," *Computer Graphics* (Proc. SIGGRAPH), Vol. 20, No. 4, Aug. 1986, pp. 39-48.
26. P. Robertson, "Spatial Transformations for Rapid Scan-Line Surface Shadowing," *IEEE CG&A*, Vol. 9, No. 2, March 1989, pp. 30-38.

27. R. Goldstein and R. Nagel, "3-D Visual Simulation," *Simulation*, Jan. 1971, pp. 25-31.

28. T. Whitted, "An Improved Illumination Model for Shaded Display," *CACM*, Vol. 23, No. 6, June 1980, pp. 343-349.

29. J. Amanatides and D. Mitchell, "Some Regularization Problems in Ray Tracing," *Graphics Interface 90*, May 1990, pp. 221-228.

30. S. Rubin and T. Whitted, "A 3-Dimensional Representation for Fast Rendering of Complex Scenes," *Computer Graphics* (Proc. SIGGRAPH), Vol. 14, No. 3, July 1980, pp. 110-116.

31. T. Kay and J. Kajiya, "Ray Tracing Complex Scenes," *Computer Graphics* (Proc. SIGGRAPH), Vol. 20, No. 4, Aug. 1986, pp. 269-278.

32. J. Goldsmith and J. Salmon, "Automatic Creation of Object Hierarchies for Ray Tracing," *IEEE CG&A*, Vol. 7, No. 5, May 1987, pp. 14-20.

33. A. Glassner, "Spacetime Ray Tracing for Animation," *IEEE CG&A*, Vol. 8, No. 2, March 1988, pp. 60-70.

34. A. Glassner, "Space Subdivision for Fast Ray Tracing," *IEEE CG&A*, Vol. 4, No. 10, Oct. 1984, pp. 15-22.

35. M. Kaplan, "Space Tracing: A Constant Time Ray Tracer," Tutorial Notes on the State of the Art in Image Synthesis, SIGGRAPH 85, July 1985, pp. 149-158.

36. A. Fujimoto, T. Tanaka, and K. Iwata, "ARTS: Accelerated Ray Tracing System," *IEEE CG&A*, Vol. 6, No. 4, April 1986, pp. 16-26.

37. J. Amanatides and A. Woo, "A Fast Voxel Traversal Algorithm for Ray Tracing," *Eurographics 87*, Aug. 1987, pp. 1-10.

38. J. Snyder and A. Barr, "Ray Tracing Complex Models Containing Surface Tessellations," *Computer Graphics* (Proc. SIGGRAPH), Vol. 21, No. 4, July 1987, pp. 119-128.

39. J. Cleary and G. Wyvill, "Analysis of an Algorithm for Fast Ray Tracing Using Uniform Space Subdivision," *Visual Computer*, July 1988, pp. 65-83.

40. O. Devillers, "The Macro-regions: An Efficient Space Division Structure for Ray Tracing," Tech. Report of the Computer Science Laboratory of Ecole Normale Superieure (LIENS), Paris, Nov. 1988.

41. D. Jevans and B. Wyvill, "Adaptive Voxel Subdivision for Ray Tracing," *Graphics Interface 89*, June 1989, pp. 164-172.

42. J. Arvo and D. Kirk, "Fast Ray Tracing by Ray Classification," *Computer Graphics* (Proc. SIGGRAPH), Vol. 21, No. 4, July 1987, pp. 55-64.

43. P. Heckbert and P. Hanrahan, "Beam Tracing Polygonal Objects," *Computer Graphics* (Proc. SIGGRAPH), Vol. 18, No. 3, July 1984, pp. 119-127.

44. P. Hanrahan, "Using Caching and Breadth-First Search to Speed up Ray Tracing," *Graphics Interface 86*, May 1986, pp. 51-61.

45. R. Speer, R. DeRose, and B. Barsky, "A Theoretical and Empirical Analysis of Coherent Ray Tracing," *Graphics Interface 85*, May 1985, pp. 11-25.

46. M. Ohta and M. Maekawa, "Ray Coherence Theorem and Constant Time Ray Tracing Algorithm," *Computer Graphics* (Tokyo), 1987, pp. 303-314.

47. E. Haines and D. Greenberg, "The Light Buffer: A Shadow-Testing Accelerator," *IEEE CG&A*, Vol. 6, No. 9, Sept. 1986, pp. 6-16.

48. D. Salesin and J. Stolfi, "Rendering CSG Models wiht a zz-Buffer," *Computer Graphics* (Proc. SIGGRAPH), Vol. 24, No. 4, Aug. 1990, pp. 67-76.

49. G. Fossum and D. Fussell, "Generating Soft Shadows Efficiently," Technical Report 78712-1188, Dept. of Computer Sciences, University of Texas at Austin, June 1987.

50. P. Poulin and J. Amanatides, "Shading and Shadowing with Linear Light Sources," *Eurographics 90*, Aug. 1990, pp. 377-386.

51. C. Verbeck and D. Greenberg, "A Comprehensive Light Source Description for Computer Graphics," *IEEE CG&A*, Vol. 4, No. 7, July 1984, pp. 66-75.

52. M. Cohen and D. Greenberg, "The Hemi-Cube: A Radiosity Solution for Complex Environments," *Computer Graphics* (Proc. SIGGRAPH), Vol. 19, No. 3, July 1985, pp. 31-40.

53. T. Nishita and E. Nakamae, "Continuous Tone Representation of Three-Dimensional Objects Taking Account of Shadows and Interreflection," *Computer Graphics* (Proc. SIGGRAPH), Vol. 19, No. 3, July 1985, pp. 23-30.

54. S. Chattopadhyay and A. Fujimoto, "Bidirectional Ray Tracing," *Computer Graphics 1987*, Springer-Verlag, New York, 1987, pp. 335-343.

55. P. Haeberli and K. Akeley, "The Accumulation Buffer: Hardware Support for High-Quality Rendering," *Computer Graphics* (Proc. SIGGRAPH), Vol. 24, No. 4, Aug. 1990, pp. 309-318.

56. R. Cook, T. Porter, and L. Carpenter, "Distributed Ray Tracing," *Computer Graphics* (Proc. SIGGRAPH), Vol. 18, No. 3, July 1984, pp. 137-145.

57. M. Dippe and E. Wold, "Antialiasing Through Stochastic Sampling," *Computer Graphics* (Proc. SIGGRAPH), Vol. 19, No. 3, July 1985, pp. 69-78.

58. M. Lee, R. Redner, and S. Uselton, "Statistically Optimized Sampling for Distributed Ray Tracing," *Computer Graphics* (Proc. SIGGRAPH), Vol. 19, No. 3, July 1985, pp. 61-67.

59. R. Cook, "Stochastic Sampling in Computer Graphics," *ACM Trans. On Graphics*, Vol. 5, No. 1, Jan. 1985, pp. 51-72.

60. C. Bouville et al., "Monte-Carlo Integration Applied to an Illumination Model," *Eurographics 88*, Aug. 1988, pp. 483-498.

61. J. Amanatides, "Ray Tracing with Cones," *Computer Graphics* (Proc. SIGGRAPH), Vol. 18, No. 3, July 1984, pp. 129-135.

62. C. Goral, K. Torrence, and D. Greenberg, "Modelling the Interaction of Light Between Diffuse Surfaces," *Computer Graphics* (Proc. SIGGRAPH), Vol. 18, No. 3, July 1984, pp. 213-222.

63. A. Campbell and D. Fussell, "Adaptive Mesh Generation for Global Diffuse Illumination," *Computer Graphics* (Proc. SIGGRAPH), Vol. 24, No. 4, Aug. 1990, pp. 155-164.

64. J. Wallace, K. Elmquist, and E. Haines, "A Ray Tracing Algorithm for Progressive Radiosity," *Computer Graphics* (Proc. SIGGRAPH), Vol. 23, No. 3, July 1989, pp. 315-324.

65. T. Nishita and E. Nakamae, "Continuous Tone Representation of Three-Dimensional Objects Illuminated by Sky Light," *Computer Graphics* (Proc. SIGGRAPH), Vol. 20, No. 4, Aug. 1986, pp. 125-132.

66. R. Hall and D. Greenberg, "A Testbed for Realistic Image Synthesis," *IEEE CG&A*, Vol. 3, No. 8, Nov. 1983, pp. 10-20.

67. A. Pearce, "Shadow Attenuation for Ray Tracing Transparent Objects," in *Graphics Gems*, A. Glassner, ed., Academic Press, Cambridge, Mass., Aug. 1990, pp.397-399.

68. J. Arvo, "Backward Ray Tracing," Tutorial Notes on the Developments in Ray Tracing, SIGGRAPH 86, August 1986.

69. G. Ward, R. Rubinstein, and R. Clear, "A Ray Tracing Solution for Diffuse Inter-reflection," *Computer Graphics* (Proc. SIGGRAPH), Vol. 22, No. 4, Aug. 1988, pp. 85-92.

70. P. Shirley, "A Ray Tracing Method for Illumination Calculation in Diffuse-Specular Scenes," *Graphics Interface 90*, May 1990, pp. 205-212.

71. P. Heckbert, "Adaptive Radiosity Textures for Bidirectional Ray Tracing," *Computer Graphics* (Proc. SIGGRAPH), Vol. 24, No. 4, Aug. 1990, pp. 145-154.

72. M. Watt, "Light-Water Interaction Using Backward Beam Tracing," *Computer Graphics* (Proc. SIGGRAPH), Vol. 24, No. 4, Aug. 1990, pp. 377-385.

73. M. Inakage, "Caustics and Specular Reflection Models for Spherical Objects and Lens," *The Visual Computer*, Vol. 2, No. 6, Feb. 1986, pp. 279-383.

74. M. Shinya, T. Takahashi, and S. Naito, "Principles and Applications of Pencil Tracing," *Computer Graphics* (Proc. SIGGRAPH), Vol. 21, No. 4, July 1987, pp. 45-54.

75. M. Shinya, T. Saito, and T. Takahashi, "Rendering Techniques for Transparent Objects," *Graphics Interface 89*, June 1989, pp. 173-182.

76. J. Kajiya, "The Rendering Equation," *Computer Graphics* (Proc. SIGGRAPH), Vol. 20, No. 4, Aug. 1986, pp. 143-150.

77. A. Fournier et al., "FIAT Lux: Light Driven Global Illumination," DGP Technical Memo DGP89-1, Dynamic Graphics Project, University of Toronto, 1989.

78. J. Blinn, "A Generalization of Algebraic Surface Drawing," *ACM Trans. On Graphics*, Vol. 1, No. 3, July 1982, pp. 235-256.

79. P. Hanrahan, "Ray Tracing Algebraic Surfaces," *Computer Graphics* (Proc. SIGGRAPH), Vol. 17, No. 3, July 1983, pp. 83-89.

80. D. Kalra and A. Barr, "Guaranteed Ray Intersection with Implicit Surfaces," *Computer Graphics* (Proc. SIGGRAPH), Vol. 22, No. 4, July 1989, pp. 297-306.

81. D. Mitchell, "Robust Ray Intersection with Interval Arithmetic," *Graphics Interface 90*, May 1990, pp. 68-74.

82. J. Kajiya, "Ray Tracing Parametric Patches," *Computer Graphics* (Proc. SIGGRAPH), Vol. 16, No. 3, July 1982, pp. 245-254.

83. T. Nishita, T. Sederberg, and M. Kakimoto, "Ray Tracing Trimmed Rational Surface Patches," *Computer Graphics* (Proc. SIGGRAPH), Vol. 24, No. 4, Aug. 1990, pp. 337-345.

84. M. Sweeney and R. Bartels, "Ray Tracing Free-Form B-Spline Surfaces," *IEEE CG&A*, Vol. 6, No. 2, Feb. 1986, pp. 41-49.

85. D. Toth, "On Ray Tracing Parametric Surfaces," *Computer Graphics* (Proc. SIGGRAPH), Vol. 19, No. 3, July 1985, pp. 171-179.

86. A. Fournier and J. Buchanan, "Chebyshev Polynomials for Boxing and Intersections of Parametric Curves and Surfaces," Technical Memo Imager 90-1, Imager, University of British Columbia, 1990.

87. N. Max, "Smooth Appearance for Polygonal Surfaces," *The Visual Computer*, Vol. 4, 1989, pp. 160-173.

88. J. Blinn, "Simulation of Wrinkled Surfaces," *Computer Graphics* (Proc. SIGGRAPH), Vol. 12, No. 3, Aug. 1978, pp. 286-292.

89. N. Max, "Horizon Mapping: Shadows for Bump-Mapped Surfaces," *The Visual Computer*, Vol. 4, 1988, pp. 109-117.

90. K. Torrance and E. Sparrow, "Theory for Off-Specular Reflection from Roughened Surfaces," *J. Optical Society of America*, Vol. 57, No. 9, 1967.

91. J. Blinn, "Models of Light Reflection for Computer Synthesized Pictures," *Computer Graphics* (Proc. SIGGRAPH), Vol. 11, No. 2, July 1977, pp. 192-198.

92. P. Poulin and A. Fournier, "A Model for Anisotropic Reflection," *Computer Graphics* (Proc. SIGGRAPH), Vol. 24, No. 4, Aug. 1990, pp. 273-282.

93. B. Cabral, N. Max, and R. Springmeyer, "Bidirectional Reflection Functions from Surface Bump Maps," *Computer Graphics* (Proc. SIGGRAPH), Vol. 21, No. 4, July 1987, pp. 273-282.

94. W. Reeves, "Particle Systems—A Technique for Modeling a Class of Fuzzy Objects," *ACM Trans. On Graphics*, Vol. 2, No. 2, April 1983, pp. 359-376.

95. K. Sims, "Particle Animation and Rendering Using Data Parallel Computation," *Computer Graphics* (Proc. SIGGRAPH), Vol. 24, No. 4, Aug. 1990, pp. 405-413.

96. J. Blinn, "Light Reflection Functions for Simulation of Clouds and Dusty Surfaces," *Computer Graphics* (Proc. SIGGRAPH), Vol. 16, No. 3, July 1982, pp. 21-29.

97. J. Kajiya and B. Von Herzen, "Ray Tracing Volume Densities," *Computer Graphics* (Proc. SIGGRAPH), Vol. 18, No. 3, July 1984, pp. 165-174.

98. D. Ebert and R. Parent, "Rendering and Animation of Gaseous Phenomena by Combining Fast Volume and Scan-line A-buffer Techniques," *Computer Graphics* (Proc. SIGGRAPH), Vol. 24, No. 4, Aug. 1990, pp. 357-366.

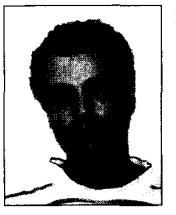
99. T. Nishita, Y. Miyawaki, and E. Nakamae, "A Shading Model for Atmospheric Scattering Considering Luminous Intensity Distribution of Light Sources," *Computer Graphics* (Proc. SIGGRAPH), Vol. 21, No. 4, July 1987, pp. 303-310.

100. H. Rushmeier and K. Torrance, "The Zonal Method for Calculating Light Intensities in the Presence of a Participating Medium," *Computer Graphics* (Proc. SIGGRAPH), Vol. 21, No. 4, July 1987, pp. 293-302.

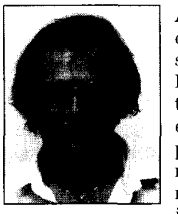
101. W. Seales and C. Dyer, "Shaded Rendering and Shadow Computation for Polyhedral Animation," *Graphics Interface 90*, May 1990, pp. 175-182.

102. P. Hsu and J. Staudhammer, "Superposing Images with Shadow Casting," to appear in *Visualization 90*, Oct. 1990.

Andrew Woo currently works at Alias Research, Toronto, Canada. His research interests include general rendering issues, particularly acceleration techniques and illumination models. Woo received a BS in computer science and commerce in 1987 and an MS in computer science in 1989, both at the University of Toronto. He is a member of ACM and IEEE and currently serves as the treasurer of the ACM Toronto SIGGRAPH Group.



Pierre Poulin is a PhD candidate in the department of computer science of the University of British Columbia. He received his MS in 1989 from the University of Toronto and his BS in 1986 in computer science from the University Laval, Quebec. His research involves illumination models, color, and stochastic processes. He is member of IEEE and ACM.



Alain Fournier is an associate professor in the department of computer science at the University of British Columbia, and co-director of Imager, its computer graphics laboratory. For the past 10 years he has taught computer science and has been active in research in computer graphics, especially in the areas of modeling and geometric and display algorithms. His publications in computer graphics include papers on the stochastic modeling of terrain, on the realistic modeling of ocean waves, on various antialiasing schemes, and on theoretical models of computer graphics devices to better analyze the complexity of basic algorithms.

Fournier received a Masters in chemistry from the University of Montreal and a PhD in mathematical sciences from the University of Texas at Dallas. He is a member of ACM and is currently an associate editor of *ACM Transactions on Graphics*.

Readers can contact Woo at Alias Research, 110 Richmond St. E., Toronto, Ontario, Canada, M5C 1P1, and Poulin and Fournier at Imager, Department of Computer Science, University of British Columbia, Vancouver, B.C., Canada, V6T 1W5. Woo can be reached via e-mail at awoo@alias.UUCP, Poulin at poulin@cs.ubc.ca, and Fournier at fournier@cs.ubc.ca.