A Survey of Shadow Algorithms

Andrew Woo[1]   Pierre Poulin   Alain Fournier[2]

Department of Computer Science
University of Toronto
Toronto, Ontario, Canada
M5S 1A4
{andreww | poulin | alain} @dgp.toronto.edu

[1]Andrew Woo's current address: Alias Research Inc., 110 Richmond Street East, Toronto, Ontario, Canada, M5C 1P1.

[2]Pierre Poulin and Alain Fournier's current address: Imager, Department of Computer Science, University of British Columbia, Vancouver, B.C., Canada, V6T 1W5.

Abstract

Shadows are essential to realistic and visually appealing images, but they are difficult to compute in most display environments. This survey will characterize the various types of shadows, describe most existing shadow algorithms, and for each one discuss their complexities, their advantages and their shortcomings. The types of shadows examined are *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.

The goal of the survey is to provide readers with enough background and insight on the various methods to allow them to choose the algorithm best suited to their needs. It is also hoped that our analysis will help identify the areas that need more research, and point to possible solutions.

CR Categories and Subject Descriptors: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism.
General Terms: Algorithms.
Additional Key Words and Phrases: survey, shadowing, hard shadow, soft shadow, transparency, caustics, parametric and implicit surfaces, umbra, penumbra, illumination, self-shadowing, particles, inter-reflections, scanline, ray tracing, radiosity.

1 Introduction

A shadow is a region of relative darkness within an illuminated region caused by an object totally or partially occluding the light. A transparent object does not necessarily attenuate the light it occludes, and in fact can concentrate it, but as is traditional in image synthesis, a region will be considered to be in shadow if there exists an occluding surface between light and the region of interest, whether it is transparent or not.

A shadow is not necessarily seen as a darker region; coloured shadows can appear in different ways in a scene. If two light sources of different colours are used, a region occluded from only one light source by an opaque object will be in the shadow of this light source but can be influenced by the colour of the second light source. An object can also act as a filter; the colour of the shadow region can then depend on the wavelength spectrum filtered through this object. Diffraction of light through a transparent object is another possible source of colour within a shadow.

By 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); they represent an integral part of the total effect in architecture with many objects in the environment. In fact, in some circumstances, the shadows are the only components of the scene, as in *shadow puppets* theater and in *pin screen* animation developed by Alexander Alexeieff.

In the only previous comprehensive discussion of shadow algorithms, Crow [1] discusses shadow generation by classifying the type of algorithms used. He distinguishes three general categories of shadow algorithms. This distinction has been very useful, and has inspired many developments since. In the intervening years, however, there have been many new modeling primitives and rendering techniques, and within each one, new shadow algorithms have been developed. Amanatides [2], Thalmann and Thalmann [3], Foley et al. [4] describe some of the existing shadow algorithms within the general context of image synthesis. However, they deal mainly with shadows created by opaque objects, and are not complete even within this framework.

In this survey, the algorithms will be examined 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, algorithms will be examined as they relate to the specific rendering techniques for which they were developed. Each algorithm will be described and then discussed in sufficient detail to be understood (but usually not in sufficient detail to be implemented). Its characteristics, its relationships with other algorithms, its physical accuracy and its limitations will be presented. Whenever possible, an indication of its complexity, in term of storage space and time, will be given.

It is our hope that after reading this survey, an implementor will be aware of nearly all the existing shadow algorithms, and will have sufficient information necessary to choose an algorithm given the type of rendering, primitives and effects desired.

Of course, many problems related to shadows are not solved (or well solved). In the course of this survey, we will try to identify gaps, and suggest improvements to known algorithms or directions for new ones.

2 Complexity of Algorithms

All shadow complexity order statistics are stated assuming a single light source and polygonal surfaces as a consistent basis for comparison. There are mainly three components to the complexity: storage usage, preprocessing runtime complexity, and runtime complexity during actual rendering (which will be referred to as shadow rendering complexity from now on in this survey).

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 stated.

The following table summarizes some common notation used throughout the 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

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

3 Hard Shadow Generation

This section discusses shadow algorithms for hard shadows, where the display of the umbra section alone is required (illustrated in image 10). Calculation of hard shadows involves only the determination of whether or not a point in the scene is in shadow of opaque objects. This is a binary decision problem on top of the shading model. In other words, a value of either 0 or 1 is multiplied with the light intensity, indicating in shadow or not in shadow, respectively.

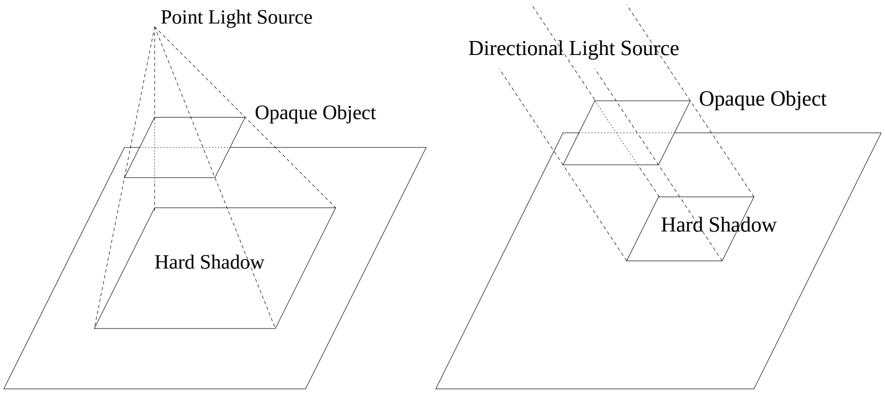

Figure 1: Hard Shadows

The domain of light sources truly generating hard shadows is restricted to point [5, 6] and directional light (see figure 1).

In general, hard shadow determination can be considered to be just as difficult as the visibility determination from the eye; i.e. shadow determination is just visibility determination with respect to the light source. However, it should be noted that hard shadow determination is actually a simpler issue to deal with. The closest visible object does not need to be calculated, just the existence of an object between light and the point of interest needs to be determined.

3.1 Fake Shadows

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

3.2 Shadow Generation During the Scanning Phase

Appel [8], and Bouknight and Kelley [9] suggest shadow generation during the display phase using an extended scanline 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, and 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 are $O(n^2)$ and $O((En)^2)$, respectively, since they perform some preprocessing to determine candidate occluding objects. However, Appel does not perform the preprocessing step; all objects are checked during the scanning phase. The shadow rendering complexity for both is $O(En)$ per scanned segment (not per pixel). This shadow determination approach is only suitable for polygons. In addition, it has not acquired the popularity of the shadow determination approaches to be discussed in the upcoming subsections.

3.3 Shadow Volumes

Crow [1] proposes an approach to generate polygonal shadow umbrae from the objects in the scene, and then placing them into the rendering data structure as invisible objects. Many others [10, 11,

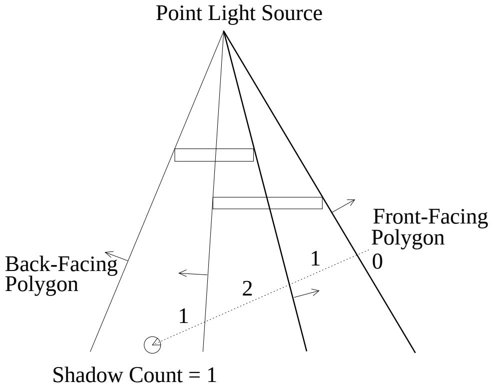

Figure 2: Shadow Volume

12, 13, 14, 15, 16, 17] later apply some variations of the general principle to frame buffer/scanline approaches. Shadow determination is computed by using a *shadow count*. An initial shadow count is calculated by counting the number of shadow volumes which contain the viewing position. The shadow count is then incremented by 1 whenever there exists a shadow back-facing polygon (i.e. entering the shadow umbrae) crossing in front of the nearest visible object, and decremented by 1 whenever there exists a shadow front-facing polygon (i.e. exiting the shadow umbrae). If the final shadow count is 0, then the visible object is not in shadow; if positive it is in shadow (see figure 2).

The raw approach [1, 12] requires $\mathrm{O}(En)$ 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 $\mathrm{O}(En)$. A BSP tree variation [15] requires $\mathrm{O}(En)$ storage, with preprocessing complexity of $\mathrm{O}(En^2)$ and rendering complexity of perhaps $\mathrm{O}(\log(En))$ for BSP tree traversal. Other algorithms, such as in Fournier et al. [14], requires storage of $\mathrm{O}(pq)$ to keep the shadow count updates, and the shadow rendering complexity is constant with preprocessing complexity of $\mathrm{O}(Enpq)$.

3.4 Area Subdivision

Nishita and Nakamae [18], and Atherton et al. [19] use clipping transformations for polygon shadow generation. It is a 2-pass hidden surface algorithm where 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. Then a new set of polygons is created, each marked as either completely in shadow or not. In the second pass, visible determination from the eye is done, and the polygons are shaded taking into account their shadow flag.

The storage complexity may 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 $\mathrm{O}(E^2n)$. The preprocessing complexity of the algorithm due to the polygon clipping process is $\mathrm{O}((En)^2)$. There is no additional cost for the shadow rendering complexity, but possibly many more shadow and non-shadow polygons need to be taken into consideration (as compared to the original set of polygons without consideration of shadowing).

Note that the hidden surface polygon clipper needs to be sufficiently sophisticated such that it can 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 the shadow boundaries are internally stored as polygons in this algorithm, the information can be sent to hardware shaders (hardware that processes and shades polygons) and produce real-time shadows if the lights and polygonal database are not altered.

3.5 Depth Buffer

Williams [20] uses 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 a buffer is created with respect to the point of view of the light source, except that the buffer only contains z depth values and not any shading values nor 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 (i.e. the z value in the buffer), then it is in shadow; otherwise it is not.

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

The obvious disadvantage of this algorithm is the maximum 180 degree shadow frustum that it can handle per buffer. If a light source were placed inside the scene, then six such buffers may be necessary 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 is different in orientation and size to a cell with respect to the light source). However, the attractiveness of this approach is that it can trivially support primitives other than just polygons.

Hourcade and Nicolas [21], and Reeves et al. [22] attempt to improve on both aliasing problems by applying stochastic sampling and some prefiltering to the Z-buffer approach. Reeves et al. also claim that soft shadows can be generated from their algorithm. However, it should be noted that this is a result of 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 algorithms that have been proposed use the idea of keeping the shadow information provided by projecting the scene in the light source direction [23, 24, 25]. An example is the optimization of the scanline algorithm for surface shadowing proposed by Robertson [25]. By using a combination of rotation of the scene along the light direction, alignment of the points that can occlude themselves along vertical scanlines, determination of the shadowed points and return back to the original orientation, Robertson optimizes the shadow determination process because he can now simply use a composition of one-dimensional operations. However, his method, as many of the others based on projections, is highly subject to aliasing problems as the light source gets closer to the scene.

3.6 Ray Tracing

Ray casting [8, 26] was introduced as a method for visibility calculation and shadow determination. A ray is shot (or cast) from the eye to each pixel, ray-surface intersections are performed and the surface with the minimum hit distance is declared the visible surface. This approach generalizes to the rendering technique popularized by Whitted [27] known as ray tracing. This technique also models reflection and refraction, and generates shadows.

The principle behind ray tracing for hard shadow determination is very simple: 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 is in shadow; otherwise it is 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 is $O(En)$ 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 anti-aliasing. The main advantage of ray tracing is that shadow determination is handled in the (almost) identical manner for cast, reflected and refracted rays. Since 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 might give incorrect results due to numerical errors. This is usually seen as little holes in the images - some refer to it as *surface acne*. Further numerical error problems are discussed and solutions offered in the work by Amanatides and Mitchell [28].

3.6.1 Intersection Culling Algorithms for Ray Tracing

Since for each ray, intersection checks with all objects in the scene are necessary, research has gone into intersection culling so that only a small candidate set of objects needs to be checked for intersection with all ray types. There are many such culling algorithms. They include: hierarchical bounding volumes [29, 30, 31, 32], variable size voxel traversal [33, 34], uniform size voxel traversal [35, 36, 37, 38], hybrid uniform-variable voxel traversal [39, 40], ray classification [41], spatial coherence [42, 43], and ray coherence [44, 45]. It is difficult to analyze the complexities of most of the approaches. They are usually much improved over the brute force $O(En)$ per ray shot, and require some preprocessing but substantial storage.

3.6.2 Shadow Culling Algorithms for Ray Tracing

While the above algorithms perform culling with respect to all ray types including shadows, additional work has been done to cull shadow rays even further. This is done in realization that neighbouring shadow binary decisions are usually the same. Preprocessing can potentially save a great deal of shadow intersection tests. The light buffer [46], hybrid shadow testing [16], voxel occlusion testing [17], and ZZ-buffer [47] are examples of such shadow cullers. The light buffer [48, 49], voxel occlusion testing [17], and ZZ-buffer [47] have been extended to model soft shadows as well.

4 Soft Shadow Generation

Another type of shadow algorithm deals with soft shadows (illustrated in image 11), i.e. the inclusion of the penumbra region along with the umbra, for a higher level of visual quality (see figure 3). The umbra region is due to full occlusion from the light, and the penumbra region is due to only partial occlusion from the light. The degree of partial occlusion from the light results in different intensities of the penumbra region.

Determining soft shadows requires the calculation of 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 with the light intensity, where 0 indicates umbra, 1 indicates no shadow and all other values indicate penumbra.

It is also noteworthy that the resultant shadow region has a shape depending both on the occluding object and on the light source (see figure 3). In addition, the types of light sources

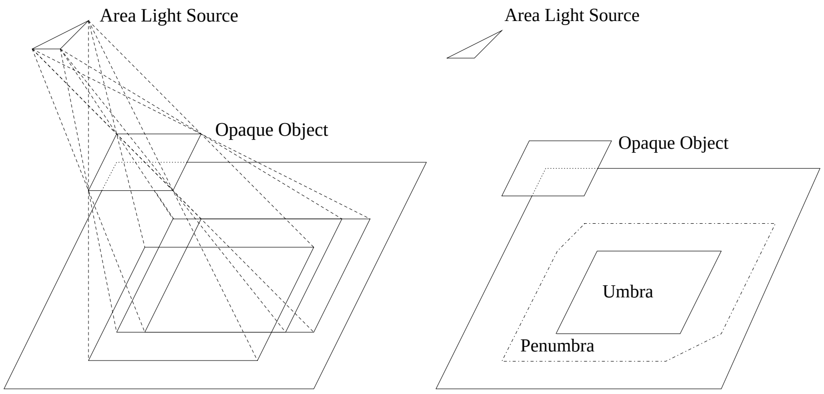

Figure 3: Soft Shadows

modeled are linear and area light sources [50, 6, 49]. The soft shadow algorithms considered here will be only due to opaque objects and the intensity is assumed identical for every point on the light source.

Some work has been done to generate soft shadows due to inter-reflections of diffuse light. Such work takes into account some of the global illumination issues. The surfaces in the scene are considered as secondary light sources and thus soft shadows can be generated [51, 52, 53].

4.1 Frame Buffer Algorithm

Brotman and Badler [10] stochastically choose points to model higher dimensional light sources. Shadow umbra polygons for each such point source are generated in the same manner as in Crow's algorithm [1] (see section 3.3). This shadow polygon generation is done during preprocessing. A 2D depth buffer for visible surface determination is extended to store cell counters. The cell count (slightly different from Crow's shadow count) is calculated 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 is equal to the number of chosen point light sources, then the point is 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 is 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, many such point sources are necessary (Brotman and Badler rendered several images using around 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 may not be considered because of the pre-chosen point sources. Thus the inconsistency may cause some sparkle-like artifacts for highly specular surfaces, although it may not be 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 [54], allows the rendering of soft shadows by accumulating in a frame buffer the weighted shadows from point sources. How to sample correctly the light sources is not discussed in the paper.

4.2 Distributed Ray Tracing

Cook et al. [55] propose a distributed ray tracing method for soft shadow generation. A collection of shadow rays is shot 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 proportional 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, no storage and preprocessing are necessary. The shadow rendering complexity is $O(PEn)$, 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, it is a point sampling approach, which may not always provide good approximations to the correct solution. Further stochastic schemes for distributed ray tracing have been discussed in other works [56, 57, 58, 59].

4.3 Cone Tracing

Amanatides [60] extends the concept of a ray to a cone. Instead of point sampling as in previous approaches, cone tracing does area sampling. Exactly one conic ray needs to be shot per pixel to achieve anti-aliasing. By broadening the cone to the size of a circular (spherical) light source for shadow cones, soft shadows can be generated: a partial intersection with an object not covering the entire cone indicates penumbra.

Again, there is no additional storage or preprocessing costs and the shadow rendering complexity is $O(En)$. It has the favourable property that exactly one shadow ray is shot and the area sampling done should provide a good enough approximation to the penumbra intensity. However, note that the approximation is only physically valid for circular light sources. Since the resultant shadow region is determined by the shape of both the occluding object and light source, cone tracing is less suitable for light source shapes that cannot be closely approximated by one or more spheres.

This approach is very powerful in that it is capable of providing anti-aliasing, soft shadows and gloss, but the cone geometry calculations involved are more difficult than dealing with just rays as in traditional ray tracing.

4.4 Area Subdivision Approach

Soft shadows in point sampling ray tracing are generally computed by shooting a set of distributed shadow rays to determine occlusions as well as 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 $\mathbf{P}$ to be shaded, the polygon clipper described by Atherton et al. [19] (section 3.4) can be applied in reverse.

A candidate set of objects lying between $\mathbf{P}$ and the light can be easily acquired through any intersection culler. Then the candidate objects are projected onto the light source as viewed from $\mathbf{P}$, and then clipped using the algorithm of Atherton et al. After clipping of all the candidate objects is done, exactly the region of the light source that is visible from $\mathbf{P}$ is identified (see figure 4). This region is then passed to any intensity integral solver. Whether the solver is an analytic

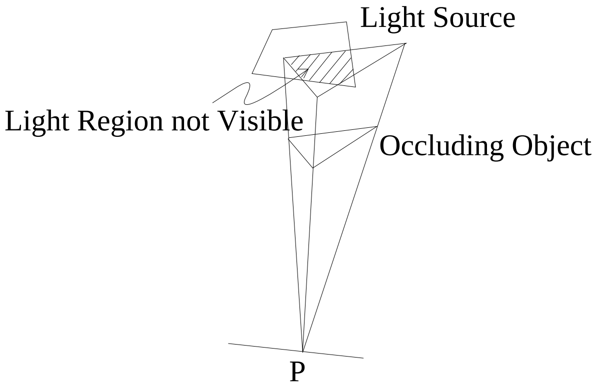

Figure 4: Area Subdivision Shadow Tracing

[6, 49] or point sampling approach [50] does not matter, the important result is that the shadow calculations can be relied upon. Many coherence optimizations can also be applied here.

4.5 Bidirectional Ray Tracing

Chattopadhyay and Fujimoto [53] propose a fast method named bidirectional ray tracing to generate soft shadows (as a result of inter-reflections) using a uniform voxel structure. Shadow rays are shot from the light sources to surfaces (highly diffuse only) that are considered to be a potential secondary light. This is to calculate the intensity contribution to the surface. This contribution is then interpolated for the vertices of the voxels that contain the surface.

As a second step, a $(R+1)^3$-bit factor is added 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 is done to handle diffuse inter-reflections of light: for each vertex i that is 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, the secondary contribution needs to be calculated, and so on. The authors claim that one or two iterations of this process are usually sufficient. Then the final illumination value at any shaded point is just an interpolation of the vertex information and retracing of the contributions of the vertices.

The storage complexity of this approach is $\mathrm{O}(R^6)$. The preprocessing necessary is about $\mathrm{O}(EnR^6)$, 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 $\mathrm{O}(SR^3)$, where S stands for the number of iterations of secondary illumination calculations.

A problem with this approach for soft shadow generation is that the image results differ significantly for different voxel sizes due to the interpolation process between voxel vertices. As well, this can also result in poor approximations since the surface orientations are not taken into account.

4.6 Radiosity

Another method of computing soft shadows comes from the calculation of radiosity [61]. This class of algorithms is capable of calculating diffuse inter-reflections between surfaces by determining an equilibrium energy balance within an enclosure. This is a rather expensive algorithm and only polygonal surfaces have been used so far.

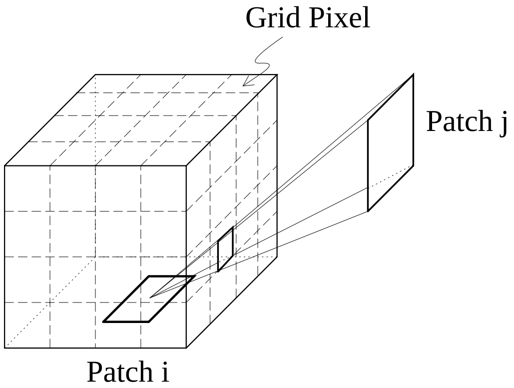

Figure 5: Hemi-Cube

Each surface (a light source is also considered a surface in this algorithm) 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 require a determination of inter-patch visibility. Once the form factors are known, solving a system of linear equations give the radiosity, hence the shading, of each patch.

4.6.1 Hemi-Cube

Two approaches have been described in the literature to specifically handle soft shadows using the radiosity approach. The first approach, proposed by Cohen and Greenberg [51], computes form factors by attaching an hemi-cube to each patch (see figure 5). The hemi-cube 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 that can be seen from the patch and that intersects the given grid pixel. Other surfaces will then be considered to be 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 may not cover it completely. Patches lying beyond this closest patch within the frustum of p are considered to be completely in shadow, but may not necessarily be. Thus inaccurate shadows may be generated if there exists many visible patches with respect to the pixel due to the insufficiency of the hemi-cube resolution.

4.6.2 Shadow Polygons

Another approach for soft shadow generation within radiosity is proposed by Nishita and Nakamae [6, 52]. In the hemi-cube approach, the B_i values are calculated at the center of the patch. Nishita and Nakamae propose that B_i be calculated at the vertices of the patches, and shadow testing only be done at these points. To do so, two additional terms are added 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 that is hidden from patch i by the other patches. This shadowing function is determined by using shadow polygons, and the umbra and penumbra regions are identified [6] (see figure 3). For each vertex of patch i, shadow polygons are projected 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 is inside the shadow umbra region, then no light can reach patch k from patch i; if patch k is inside the shadow penumbra region, then light can partially reach patch k from patch i. Smoothed soft shadows are generated when the luminance of each patch is interpolated from the luminance at the vertices.

4.6.3 Complexity Analysis

The preprocessing complexity of the hemi-cube is $O(En^2R^2)$, where n is the number of patches in the scene, $R \times R$ is the resolution of the hemi-cube, and EnR^2 accounts for the scan-conversion cost per hemi-cube. The approach of Nishita and Nakamae has a $O((En)^2I)$ cost, where I is the cost of identifying the umbra and penumbra regions. The calculation of the form factors resulting from the hemi-cube approach tends to be faster. However, the hemi-cube approach has the undesirable feature that the image results differ significantly with variations in the resolution of the hemi-cube. In addition, both approaches only approximate the level of shadowing. Which method provides more accurate results is unknown.

4.6.4 Other Radiosity Approaches

There are many other papers on radiosity, which will not be listed since most do not address specifically shadow determination. One exception is the use of ray tracing to replace the hemi-cube approximations [63], but it tends to be much more expensive though more accurate in shadowing effects.

Another exception is the work done by Campbell and Fussell [62]. They adaptively subdivide the scene along the shadow boundaries in a progressive refinement scheme. A light emitter is subdivided until each element can be treated as a point light source. A BSP shadow volume generation, similar to [15] is used to subdivide efficiently the receiver polygons for each element on the emitter.

The quality of the shadows (especially sharpness) thus created are greatly improved from the previous techniques. However the time and storage complexity of their approach have to be seriously considered if many iterations of the progressive refinement are expected. Assume P points approximate an emitter and k is the number of iterations, 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 $O(k^2P^2E^2n^3)$. It should be kept in mind that P can be large and that k can be $O(n)$ for a large number of strongly emitting or reemitting polygons. The above complexity is only for the storage and processing is even larger.

4.7 Skylight Illumination

Nishita and Nakamae [64] add the contribution of skylight to soft shadows. For each point to be shaded, an hemisphere representing the sky is generated and band sources are defined longitudinally on this hemisphere. By assuming a uniform luminance of the sky within each band, they can sample the luminance only over the centerline of each band. The luminance of the sky can be interpolated from clear to overcast sky. Every object in the scene is projected onto the sample lines in order to compute the obstructed factor. If the point to be shaded is on a tilted surface (with respect to the scene), the hemisphere is also tilted by the same angle and the light reflection from the ground is considered for this tilted angle. The method also extends to take into account the skylight entering into interiors.

5 Shadows from Transparent Objects

One of the difficult problems while generating realistic shadows is to deal correctly with shadows from occluding transparent objects. When the light goes through a transparent object, the object can change the properties of the incoming light, for instance, its direction and colour. These effects have to be taken into account in order to generate correct shadows.

An aspect in determining the shadows from transparent objects is the presence of concentration or diffusion of light when they pass 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 image 12) or reflects from a concave mirror. This is what many algorithms try to capture. This effect, however, tends to be computationally expensive.

Determining shadows from transparent objects is more complex in that not only the first occluding object (if it is transparent) needs to be found, but all the 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.

It is important to mention that the coloured region produced from occluding transparent objects is not really a shadow. A shadow is caused by occlusion from light, but the coloured region is the result of transmitted light. But since previous works have considered these regions to be in the class of shadows, and many of the techniques are similar, the algorithms that generate such effects will also be studied.

Note that, with the exception of the discussion in section 5.1, all the algorithms discussed for transparent shadow generation are also capable of generating soft shadows.

5.1 Shadow Ray

Traditionally in ray tracing, shadows which result from occluding transparent objects are handled by detecting all occluding objects and returning the fraction of light transmitted through the objects [27]. But this is not the correct answer for transparent objects because the shadowing function is wavelength dependent, and refraction of incoming light is not taken into account.

Hall and Greenberg [65] (and similarly with the images generated in work by Lee et al. [57]) deal with the display of coloured transparent objects using the spectrum absorption model, which accounts for scattering of refracted light sources in ray tracing. They do not explicitly mention the shadowing effects of transparent objects. However, one of their images shows a green transparent object projecting a green shadow on an opaque surface. The implementation used approximate transparency without consideration to refraction: a single shadow ray is shot to the light source and all transparent objects that intersect the ray contribute to the final shadow colour through colour filtering of all the occluding transparent surfaces. The Fresnel reflectance is considered at the intersection of the ray with the surface to decide the percentage that is reflected and refracted into the object.

A simple trick to simulate the light attenuation from the shadow of transparent objects is described by Pearce [66]. He attenuates 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.

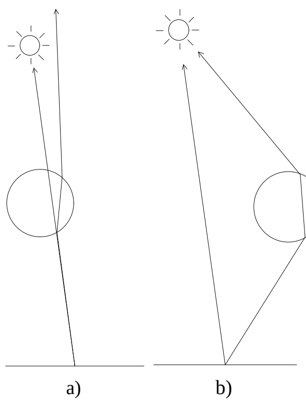

Figure 6: Incorrect Shadowing of Transparent Objects

5.2 Backward Ray Tracing

It is very difficult to generate correctly transparent shadows in ray tracing just by applying a single shadow ray. Because of refraction, the shadow ray aimed towards the light source bends as it enters and exits transparent objects, and it may not even reach the light source (see figure 6a). Similarly, other ray paths that go through transparent objects may contribute to the shading value but are not part of the shadow ray (see figure 6b). The above two scenarios will result in incorrect shadowing if a shadow ray is shot to the light source.

This suggests the tracing of rays from the light source is needed, 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 is similar to the *backward ray tracing* approach proposed by Arvo [67] (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.

By shooting rays from the light, an very large set of rays needs to be shot in all directions in order to accurately account for diffuse inter-reflections. The approach by Arvo, and similarly in some others [68, 69, 70], require a great deal of storage and processing. In addition, the generated images usually exhibits aliasing and sparkle-like artifacts due to the insufficiency of the sampling done. Watt [71] uses a technique similar to beam tracing from the light, but the concentration of light in the resulting images betray their polygonal origin.

5.3 Cone of Convergence

The concentration of light passing through a convex lens is modeled by Inakage [72] using ray tracing. Inakage illustrates a limited effect where a spherical lens projects a shadow with a concentration of light within the shadow area. Parallel rays of light are assumed to strike a perfectly spherical lens (i.e. assuming a directional light source), which he assumed would result in convergence of light intensity at a focal point beyond the object.

The cone of convergence (see figure 7) is defined as the volume swept by the joining of the focal point and the silhouette of the sphere. If the point to be shaded is within this cone of convergence, then it lies within caustics and intensity at that point is increased. Note that the cone of convergence is calculated during preprocessing and this needs to be done for each transparent sphere per light source.

There are many problems with this approach. First, the author does not consider the light

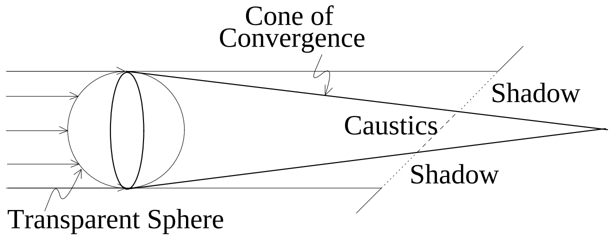

Figure 7: Concentration from Parallel Rays

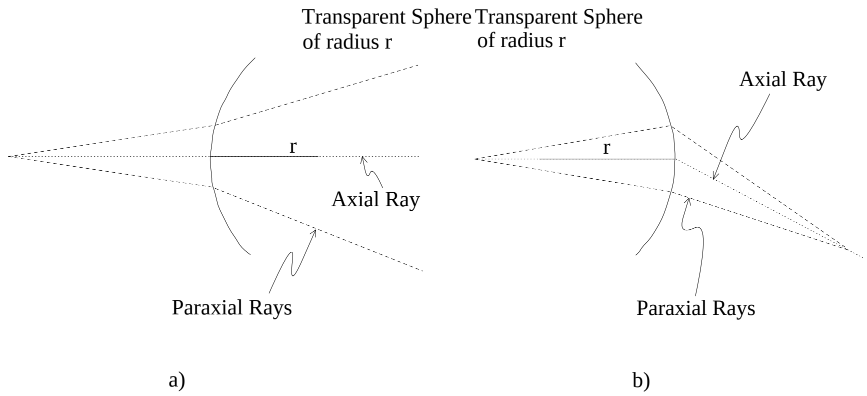

Figure 8: Paraxial Approximation Theory

concentration beyond the cone of convergence and assumes that the concentration of light is uniform within the cone of convergence. In addition, the algorithm does not deal with occlusion of these light rays, which results in incorrect shadowing effects. Finally, by assuming parallel set of incoming rays, the resultant convergence is valid only for a single thin lens, but not a sphere of arbitrary radius. Paraxial approximation theory, presented in the next section, deals more effectively with the generation of caustics.

5.4 Pencil Tracing

Another method handling transparency, concentration and diffusion of light is proposed by Shinya et al. [73], where they apply fundamentals of paraxial approximation theory. A pencil is defined by the axial ray surrounded by nearby paraxial rays.

Pencil tracing is basically an intermediary between cone tracing and ray tracing. Ray tracing traces lines so that 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. Thus divergence (figure 8a) and convergence of light (figure 8b) are natural results of this algorithm.

For generation of shadows from transparent objects, pencils are shot from the lights to the bounding volumes of all transparent objects, then propagated to the surfaces. Thus refraction is taken into account, but still only first-generation transmitted light is taken into account. This processing requires shadow rendering complexity of $\mathrm{O}(QEn)$, where Q stands for the average number of pencils shot from the light source to each transparent object. An extension of the approach including wavelength dependency is proposed in [74].

5.5 The Rendering Equation and Path Tracing

Kajiya [75] proposes an approximation to the rendering equation which uses ray tracing. It is named *path tracing* and 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, shadow, where inter-reflection rays are shot stochastically into the environment. However, choosing only certain secondary rays may conceivably result in erroneous shadows and undesirable light effects.

5.6 Light Driven Global Illumination

FIAT [76] is proposed as a light-driven approach to global illumination (for both the specular and diffuse components). Inter-reflections, inter-refractions, shadows, concentration and diffusion of light can be handled by balancing the power of regions of space. The scene is basically enveloped in an hierarchical data structure: 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 be approximately equal to 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 an hemisphere of discretized directions (θ, ϕ) that carries information on the power that is 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. A progressive refinement approach can be used in this balancing by selecting the least balanced cell to adjust first. Iterations of this 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 discrete directions are used to store information in the sexell. It is conceivable that illumination will be slightly distorted or shifted if large amount of activity (e.g. many objects, many inter-reflections) lie between neighbouring sexell directions.

6 Shadows of Complex Surfaces

The shadows dealt with so far in this paper include the shadowing on and from objects without any concern to the surface definition of these objects. In this section, problems related to the shadowing evaluation of parametric and implicit surfaces are discussed. In addition, shadowing on and of texture mapped surfaces, as well as self-shadowing from facets will be presented.

6.1 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.

6.1.1 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 it is difficult to implement them robustly. This section is by no means thorough. Only brief descriptions of a small number of approaches are presented to give an insight into the difficulties encountered.

There are various methods to evaluate implicit surfaces directly. As an example, Blinn [77] uses a combination of Newton's iteration and *regula falsi* to render algebraic surfaces. However, convergence to a proper solution cannot be guaranteed unless an accurate start point can be generated, and divergence can be seen as holes in the visible surface and shadows. Hanrahan [78] derives analytic solutions for low order polynomial surfaces, but not general enough to handle other formulations. Two recent works have used interval analysis to guarantee correct intersection results for certain classes of implicit surfaces [79, 80]. which also improves shadow determination.

Rendering parametric patches can be just as difficult. Kajiya [81] ray traces 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. The method, however can be improved by a faster root finder, and made a little more competitive [82]. Sweeney and Bartels [83] uses 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 the Jacobian of the functions is needed). This problem can be seen at the silhouette of the shadow. Toth [84] uses 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). Fournier and Buchanan [85] use 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.

6.1.2 Polygonization

Parametric and implicit surfaces are often rendered by polygonization (decomposition of the surfaces into polygons). The advantage of polygonal decomposition is mainly the simplicity of 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 care must be taken to avoid cracks between polygons resulting from different neighbouring polygon subdivision levels. A common practice is to subdivide until the decomposed polygons meet a *flatness* criteria - two adjacent polygons are considered to meet the flatness criteria if the angle between their normals or tangents is sufficiently small.

One problem with the polygonization of surfaces is that objects do not appear smooth. Within each polygon, this effect can be reduced by interpolating the surface normals. Unfortunately, shadow calculations do not have sufficient interpolation information, thus resulting in polygonal shadow silhouettes (illustrated in image 13). Snyder and Barr [37] propose an improvement by subdividing more at the silhouette of the surface (a function of the light source position) to reduce this artifact with some success. But with multiple light sources, there is the danger of severe surface subdivision. Max [86] also attempts to smooth polygonal silhouettes as well as shadows from polygonal edges.

Another problem with polygonization is the *terminator problem*, where shadow determination is wrong due to 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 $\frac{\pi}{2}$ ($\vec{N} \cdot \vec{L} > 0$). In the case of concave

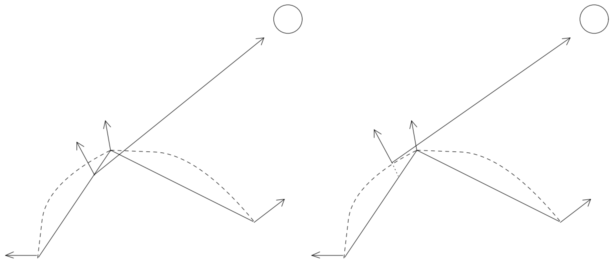

Figure 9: Terminator Problem

surfaces, a point can be in shadow even if $\vec{N} \cdot \vec{L} > 0$. Then whether a ray shot in the light direction will intersect the object again is tested. A simple method to avoid checking this intersection consists of tessellating the concave surface into many polygons. An illustration of the terminator problem is given in figure 9. In order to keep a smooth self-shadowing while interpolating normals (terminator problem) in ray tracing, Snyder and Barr [37] propose to start the shadow ray further from the intersection point, but this may cause further artifacts if poorly approximated.

6.2 Shadows on Texture Mapped Surfaces

Texture mapping is a very powerful approach to provide detail and realism to surfaces at a low cost for modeling and rendering. For some texture mapping, the aim is to give the appearance of a bumpy surface. One example is *bump mapping* [87], where the surface normal is perturbed to make the surface appear bumpy, without altering the surface itself. However, 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*, often exhibits incorrect shadowing effects if care is not taken.

6.2.1 Shadows on Bumpy Surface

By perturbing the surface normal, the surface appears bumpy. This, in effect, is also implicitly perturbing 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). This problem is a difficult one since reconstruction of the surface bumpiness from surface normals is necessary for an accurate solution of the shadowing. No good solutions have yet been proposed.

6.2.2 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 corresponds a height value by which this point would be raised on the surface. Max [88] approximates the shadows cast by these bumps on the same surface by introducing *horizon mapping*. Interpreting the bump function as a two dimensional table of height values, Max computes and stores, 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 is greater than the light angle, then this point is in shadow.

Max also proposes 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 may quite likely miss isolated narrow peaks, the overall self-shadowing effect is greatly improved.

In some cases of analytic surface map, the function defining the horizon angle may 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.

6.2.3 Transparency Mapping

Transparency mapping modifies the level of opaqueness of the surface. However, just about all software that provides both features of shadows and transpareny 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 is easy for some form of ray tracing. But the evaluation of the mapping for shadow determination increases computations.

6.3 Self-Shadowing of Facets

When the bumps are smaller and denser over a surface, neither the bumps nor the shadows can be perceived. However, the reflection of light behaves differently and this should be captured by the local reflection model.

In a reflection model proposed by Torrance-Sparrow [89, 90], the surface bumps are represented by a collection of mirror-like facets randomly oriented. Assuming that the facets are in form of V-shaped grooves, the self-shadowing effect on them can be approximated.

However, for many types of surfaces, the facets are oriented with 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 can be computed analytically. For instance, Poulin and Fournier [91] compute the self-shadowing caused by adjacent cylinders simulating an anisotropic surface (illustrated in 15). Cabral et al. [92] use an approach similar to Max [88] in order to compute a bidirectional reflection map from a height field applied to the facets distribution.

7 Shadows for Particle Based Objects

In order 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 transform procedurally and stochastically a small set of simple constraints into a complete description of very complex objects (e.g. fire [93, 94], trees and grass [23]). 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.

7.1 Particle Systems

Reeves and Blau [23] use a particle system to generate images of trees. 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 use the locations of the top of the trees positioned between the light source and the tree to define a horizon line. This line and the light source direction define a shadow plane. If a particle is under this plane, it receives 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, an orthographic view of the trees from the light is computed. When rendering a particle on the ground, a test with the shadow mask is invoked in order to determine if the particle is in shadow of trees. However, tests with objects other than trees have to be made to calculate the true shadowing.

The reality about shadowing particle systems is that for each system, the shadowing must be determined 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.

7.2 Volume Densities

Blinn [95] simulates 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 is a result of blockage of light from other particles. If it is assumed that all particles have constant radius, any statistical process (in this case, a Poisson process) can be used 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 [96] propose a more general model and apply it to ray tracing. Their single scattering model is similar to Blinn's. As a preprocessing step, a 3D grid of voxels is created. Each voxel represents a point in 3D space and contains the contribution of each light to the brightness of each point in space (i.e. density distribution). During actual ray tracing, the brightness of the ray is determined by summing the contribution of each voxel element that the ray pierces. The main computational cost of this approach is the integral evaluation of the ray brightness. Note also that a great deal of memory is needed for the three-dimensional grid of density distribution information.

7.3 Participating Medium

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

Max [13] models atmospheric illumination assuming a volume density model that is the same as Blinn's [95]: 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 - an extended shadow polygon approach [1] is used here. If the ray is completely in shadow, then the illumination value is a function of the surface colour 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 are described in [17, 97], and more general models of density distribution can be seen in the work by Nishita et al. [98] and Ebert and Parent [97].

Rushmeier and Torrance [99] apply the zonal method used in heat transfer to the computation of the radiosity. The medium is discretized into small volumes for which the form factors volume/volume and volume/surface are calculated. The shadows are generated with the hemi-cube (described earlier in section 4.6.1) 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.

8 Conclusions

A common impression given by the many algorithms examined in this survey is that shadow determination is an expensive process. It is, of course, not very surprising when one considers that the simplest shadow problem is a version of the visibility problem, and while there is only one viewpoint in a scene, there can be many lights.

Another unfortunate conclusion is that once a region is determined to lie in shadow, it is not the end of the problem. One then has to decide how to modify the illumination accordingly. This is especially difficult in the case of soft shadows, where the real answer is the result of a convolution between the occluding object and the light. It is also clear from the survey that generating accurate shadows from transparent objects, which can diffuse and concentrate the light, is especially difficult.

The three basic factors to consider in the choice of a shadow algorithm are: which rendering technique is used, which modeling primitives are used, and what degree of physical accuracy is needed. The first two factors are usually decided from other constraints, and only the third is 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, they require some additional intelligence or heuristics from the system. This is worth considering before involving more research and computing time in an exact solution to the problem of shadow determination.

Several directions for future research emerge from this survey. While there are many algorithms for shadows of polygonal objects, very few of the high performance display systems in laboratories or available on the market incorporate shadows, even though they are almost all based on polygonal primitives. A notable exception is *Pixel-Planes* [11], which can compute shadows with a version of the "shadow-count" algorithm. Some version of the algorithms described in section 3.3 should be applicable to most other systems as well, since they all use Z-buffers for visibility. Another area where more results are needed are with shadows of complex primitives, such as parametric and implicit surfaces. While subdividing such surfaces into polygons is reasonable and efficient for rendering their visible parts, it is rather wasteful for shadows, since only the outline of the shadow area is needed, and since each polygon cost $O(E)$ for processing shadows in most algorithms. In the same category, there is 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 there additional optimizations that can be achieved 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. Optimization of shadow determination for certain animations are discussed in some other works [32, 100, 101].

We hope this survey will help cast an even larger shadow on computer graphics.

9 Acknowledgements

We would like to thank Mikio Shinya (of NTT) and Marie-Claire Forgue (of INRIA) for carefully reading an early draft of our paper, and suggesting some improvements. Thanks also go to Andrew Pearce (of Alias) 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 NSERC, ITRC and the University of British Columbia for their financial support.

References

[1] F. Crow, "Shadow Algorithms for Computer Graphics", *Computer Graphics*, 11(3), August 1977, pp. 242-248.

[2] J. Amanatides, "Realism in Computer Graphics: A Survey", *IEEE CG&A*, 7(1), January 1987, pp. 44-56.

[3] N. Thalmann, D. Thalmann, "*Image Synthesis, Theory and Practice*", Springer-Verlag, 1987, pp. 156-169.

[4] J. Foley, A. Van Dam, S. Feiner, J. Hughes, "*Computer Graphics, Principles and Practice*", 2^{nd} edition, Addison-Wesley, August 1990, pp. 745-814.

[5] D. Warn, "Lighting Controls for Synthetic Images", *Computer Graphics*, 17(3), 1983, pp. 13-21.

[6] T. Nishita, I. Okamura, E. Nakamae, "Shading Models for Point and Linear Sources", *ACM Transactions on Graphics*, 4(2), April 1985, pp. 124-146.

[7] J. Blinn, "Jim Blinn's Corner: Me and my (fake) Shadow", *IEEE Computer Graphics and Applications*, 8(1), January 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, 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, N. Badler, "Generating Soft Shadows with a Depth Buffer Algorithm", *IEEE CG&A*, 4(10), October 1984, pp. 71-81.

[11] H. Fuchs, J. Coldfeather, J. Hultquist, S. Spach, J. Austin, F. Brooks, J. Eyles, J. Poulton, "Fast Spheres, Shadows, Textures, Transparencies and Image Enchancements in Pixel-Planes", *Computer Graphics*, 19(3), July 1985, pp. 111-120.

[12] P. Bergeron, "A General Version of Crow's Shadow Volumes", *IEEE CG&A*, 6(9), September 1986, pp. 17-28.

[13] N. Max, "Atmospheric Illumination and Shadows", *Computer Graphics*, 20(4), August 1986, pp. 117-124.

[14] A. Fournier, D. Fussell, "On the Power of the Frame Buffer", *ACM Transactions On Graphics*, 7(2), April 1988, pp. 103-128.

[15] N. Chin, S. Feiner, "Near Real-Time Shadow Generation Using BSP Trees", *Computer Graphics*, 23(3), July 1989, pp. 99-106.

[16] D. Eo, C. Kyung, "Hybrid Shadow Testing Scheme for Ray Tracing", *Computer Aided Design*, 21(1), January 1989, pp. 38-48.

[17] A. Woo, J. Amanatides, "Voxel Occlusion Testing: A Shadow Determination Accelerator for Ray Tracing", *Graphics Interface 90*, May 1990, pp. 213-220.

[18] T. Nishita, E. Nakamae, "An Algorithm for Half-Tone Representation of Three-Dimensional Objects", *Information Processing in Japan*, vol. 14, 1974, pp. 93-99.

[19] P. Atherton, K. Weiler, D. Greenberg, "Polygon Shadow Generation", *Computer Graphics*, 12(3), August 1978, pp. 275-281.

[20] L. Williams, "Casting Curved Shadows on Curved Surfaces", *Computer Graphics*, 12(3), August 1978, pp. 270-274.

[21] J. Hourcade, A. Nicolas, "Algorithms for Anti-aliased Cast Shadows", *Computer and Graphics*, 9(3), 1985, pp. 259-265.

[22] W. Reeves, D. Salesin, R. Cook, "Rendering Anti-Aliased Shadows with Depth Maps", *Computer Graphics*, 21(4), July 1987, pp. 283-291.

[23] W. Reeves, R. Blau, "Approximate and Probabilistic Algorithms for Shading and Rendering Structured Particle Systems", *Computer Graphics*, 19(3), July 1985, pp. 313-322.

[24] G. Miller, "The Definition and Rendering of Terrain Maps", *Computer Graphics*, 20(4), August 1986, pp. 39-48.

[25] P. Robertson, "Spatial Transformations for Rapid Scan-Line Surface Shadowing", *IEEE CG&A*, 9(2), March 1989, pp. 30-38.

[26] R. Goldstein, R. Nagel, "3-D Visual Simulation", *Simulation*, January 1971, pp. 25-31.

[27] T. Whitted, "An Improved Illumination Model for Shaded Display", *Communications of the ACM*, 23(6), June 1980, pp. 343-349.

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

[29] S. Rubin, T. Whitted, "A 3-Dimensional Representation for Rast Rendering of Complex Scenes", *Computer Graphics*, 14(3), July 1980, pp. 110-116.

[30] T. Kay, J. Kajiya, "Ray Tracing Complex Scenes", *Computer Graphics*, 20(4), August 1986, pp. 269-278.

[31] J. Goldsmith, J. Salmon, "Automatic Creation of Object Hierarchies for Ray Tracing", *IEEE CG&A*, 7(5), May 1987, pp. 14-20.

[32] A. Glassner, "Spacetime Ray Tracing for Animation", *IEEE CG&A*, 8(2), March 1988, pp. 60-70.

[33] A. Glassner, "Space Subdivision for Fast Ray Tracing", *IEEE CG&A*, 4(10), October 1984, pp. 15-22.

[34] 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.

[35] A. Fujimoto, T. Tanaka, K. Iwata, "ARTS: Accelerated Ray Tracing System", *IEEE CG&A*, 6(4), April 1986, pp. 16-26.

[36] J. Amanatides, A. Woo, "A Fast Voxel Traversal Algorithm for Ray Tracing", *EuroGraphics 87*, August 1987, pp. 1-10.

[37] J. Snyder, A. Barr, "Ray Tracing Complex Models Containing Surface Tessellations", *Computer Graphics*, 21(4), July 1987, pp. 119-128.

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

[39] O. Devillers, "The Macro-regions: an Efficient Space Division Structure for Ray Tracing", Rapport de Recherche du Laboratoire d'Informatique de l'Ecole Normale Supérieure, Paris, November 1988.

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

[41] J. Arvo, D. Kirk, "Fast Ray Tracing by Ray Classification", *Computer Graphics*, 21(4), July 1987, pp. 55-64.

[42] P. Heckbert, P. Hanrahan, "Beam Tracing Polygonal Objects", *Computer Graphics*, 18(3), July 1984, pp. 119-127.

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

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

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

[46] E. Haines, D. Greenberg, "The Light Buffer: A Shadow-Testing Accelerator", *IEEE CG&A*, 6(9), September 1986, pp. 6-16.

[47] D. Salesin, J. Stolfi, "Rendering CSG Models wiht a ZZ-Buffer", *Computer Graphics*, 24(4), August 1990, pp. 67-76.

[48] G. Fossum, D. Fussell, "Generating Soft Shadows Efficiently", Technical Report 78712-1188, Department of Computer Sciences, The University of Texas at Austin, June 1987.

[49] P. Poulin, J. Amanatides, "Shading and Shadowing with Linear Light Sources", to appear in *EuroGraphics 90*, August 1990.

[50] C. Verbeck, D. Greenberg, "A Comprehensive Light Source Description for Computer Graphics", *IEEE CG&A*, 4(7), July 1984, pp. 66-75.

[51] M. Cohen, D. Greenberg, "The Hemi-Cube: A Radiosity Solution for Complex Environments", *Computer Graphics*, 19(3), July 1985, pp. 31-40.

[52] T. Nishita, E. Nakamae, "Continuous Tone Representation of Three-Dimensional Objects Taking Account of Shadows and Interreflection", *Computer Graphics*, 19(3), July 1985, pp. 23-30.

[53] S. Chattopadhyay, A. Fujimoto, "BiDirectional Ray Tracing", *Computer Graphics 1987*, Springer-Verlag, pp. 335-343.

[54] P. Haeberli, K. Akeley, "The Accumulation Buffer: Hardware Support for High-Quality Rendering", *Computer Graphics*, 24(4), August 1990, pp. 309-318.

[55] R. Cook, T. Porter, L. Carpenter, "Distributed Ray Tracing", *Computer Graphics*, 18(3), July 1984, pp. 137-145.

[56] M. Dippe, E. Wold, "Antialiasing Through Stochastic Sampling", *Computer Graphics*, 19(3), July 1985, pp. 69-78.

[57] M. Lee, R. Redner, S. Uselton "Statistically Optimized Sampling for Distributed Ray Tracing", *Computer Graphics*, 19(3), July 1985, pp. 61-67.

[58] R. Cook, "Stochastic Sampling in Computer Graphics", *ACM Transactions On Graphics*, 5(1), January 1985, pp. 51-72.

[59] C. Bouville, J. Dubois, I. Marchal, M. Viaud, "Monte-Carlo Integration Applied to an Illumination Model", *EuroGraphics 88*, August 1988, pp. 483-498.

[60] J. Amanatides, "Ray Tracing with Cones", *Computer Graphics*, 18(3), July 1984, pp. 129-135.

[61] C. Goral, K. Torrence, D. Greenberg, "Modelling the Interaction of Light Between Diffuse Surfaces", *Computer Graphics*, 18(3), July 1984, pp. 213-222.

[62] A. Campbell, D. Fussell, "Adaptive Mesh Generation for Global Diffuse Illumination", *Computer Graphics*, 24(4), August 1990, pp. 155-164.

[63] J. Wallace, K. Elmquist, E. Haines, "A Ray Tracing Algorithm for Progressive Radiosity", *Computer Graphics*, 23(3), July 1989, pp. 315-324.

[64] T. Nishita, E. Nakamae, "Continuous Tone Representation of Three-Dimensional Objects Illuminated by Sky Light", *Computer Graphics*, 20(4), August 1986, pp. 125-132.

[65] R. Hall, D. Greenberg, "A Testbed for Realistic Image Synthesis", *IEEE CG&A*, 3(8), November 1983, pp. 10-20.

[66] A. Pearce, "Shadow Attenuation for Ray Tracing Transparent Objects" in *Graphics Gems*, ed. A. Glassner, Academic Press, August 1990, pp.397-399.

[67] J. Arvo, "Backward Ray Tracing", Tutorial Notes on the *Developments in Ray Tracing*, SIGGRAPH 86, August 1986.

[68] G. Ward, R. Rubinstein, R. Clear, "A Ray Tracing Solution for Diffuse Interreflection", *Computer Graphics*, 22(4), August 1988, pp. 85-92.

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

[70] P. Heckbert, "Adaptive Radiosity Textures for Bidirectional Ray Tracing", *Computer Graphics*, 24(4), August 1990, pp. 145-154.

[71] M. Watt, "Light-Water Interaction Using Backward Beam Tracing", *Computer Graphics*, 24(4), August 1990, pp. 377-385.

[72] M. Inakage, "Caustics and Specular Reflection Models for Spherical Objects and Lens", *The Visual Computer*, 2(6), February 1986, pp. 279-383.

[73] M. Shinya, T. Takahashi, S. Naito, "Principles and Applications of Pencil Tracing", *Computer Graphics*, 21(4), July 1987, pp. 45-54.

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

[75] J. Kajiya, "The Rendering Equation", *Computer Graphics*, 20(4), August 1986, pp. 143-150.

[76] A. Fournier, E. Fiume, M. Ouellette, C. Chee, "FIAT LUX: Light Driven Global Illumination", DGP Technical Memo DGP89-1, Dynamic Graphics Project, University of Toronto, 1989.

[77] J. Blinn, "A Generalization of Algebraic Surface Drawing", *ACM Transactions On Graphics*, 1(3), July 1982, pp. 235-256.

[78] P. Hanrahan, "Ray Tracing Algebraic Surfaces", *Computer Graphics*, 17(3), July 1983, pp. 83-89.

[79] D. Kalra, A. Barr, "Guaranteed Ray Intersection with Implicit Surfaces", *Computer Graphics*, 22(4), July 1989, pp. 297-306.

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

[81] J. Kajiya, "Ray Tracing Parametric Patches", *Computer Graphics*, 16(3), July 1982, pp. 245-254.

[82] T. Nishita, T. Sederberg, M. Kakimoto, "Ray Tracing Trimmed Rational Surface Patches" *Computer Graphics*, 24(4), August 1990, pp. 337-345.

[83] M. Sweeney, R. Bartels, "Ray Tracing Free-Form B-Spline Surfaces", *IEEE CG&A*, 6(2), February 1986, pp. 41-49.

[84] D. Toth, "On Ray Tracing Parametric Surfaces", *Computer Graphics*, 19(3), July 1985, pp. 171-179.

[85] A. Fournier, J. Buchanan, "Chebyshev Polynomials for Boxing and Intersections of Parametric Curves and Surfaces", Technical Memo Imager90-1, Imager, University of British Columbia, 1990.

[86] N. Max, "Smooth Appearance for Polygonal Surfaces", *The Visual Computer*, vol. 4, 1989, pp. 160-173.

[87] J. Blinn, "Simulation of Wrinkled Surfaces", *Computer Graphics*, 12(3), August 1978, pp. 286-292.

[88] N. Max, "Horizon Mapping: Shadows for Bump-Mapped Surfaces", *The Visual Computer*, vol. 4, 1988, pp. 109-117.

[89] K. Torrance, E. Sparrow, "Theory for Off-Specular Reflection from Roughened Surfaces", *J.Opt.Soc.Am.*, 57(9), 1967.

[90] J. Blinn, "Models of Light Reflection for Computer Synthesized Pictures", *Computer Graphics*, 11(2), July 1977, pp. 192-198.

[91] P. Poulin, A. Fournier, "A Model for Anisotropic Reflection", *Computer Graphics*, 24(4), August 1990, pp. 273-282.

[92] B. Cabral, N. Max, R. Springmeyer, "Bidirectional Reflection Functions from Surface Bump Maps", *Computer Graphics*, 21(4), July 1987, pp. 273-282.

[93] W. Reeves, "Particle Systems - A Technique for Modeling a Class of Fuzzy Objects", *ACM Transactions On Graphics*, 2(2), April 1983, pp. 359-376.

[94] K. Sims, "Particle Animation and Rendering Using Data Parallel Computation", *Computer Graphics*, 24(4), August 1990, pp. 405-413.

[95] J. Blinn, "Light Reflection Functions for Simulation of Clouds and Dusty Surfaces", *Computer Graphics*, 16(3), July 1982, pp. 21-29.

[96] J. Kajiya, B. Von Herzen, "Ray Tracing Volume Densities", *Computer Graphics*, 18(3), July 1984, pp. 165-174.

[97] D. Ebert, R. Parent, "Rendering and Animation of Gaseous Phenomena by Combining Fast Volume and Scanline A-buffer Techniques", *Computer Graphics*, 24(4), August 1990, pp. 357-366.

[98] T. Nishita, Y. Miyawaki, E. Nakamae, "A Shading Model for Atmospheric Scattering Considering Luminous Intensity Distribution of Light Sources", *Computer Graphics*, 21(4), July 1987, pp. 303-310.

[99] H. Rushmeier, K. Torrance, "The Zonal Method for Calculating Light Intensities in the Presence of a Participating Medium", *Computer Graphics*, 21(4), July 1987, pp. 293-302.

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

[101] P. Hsu, J. Staudhammer, "Superposing Images with Shadow Casting", to appear in *Visualization 90*, October 1990.

Images Illustrating the Results of Some Algorithms

Figure 10: Hard Shadows in Ray Tracing

Ray tracing is a simple technique to produce sharp shadows. In this image, a single directional light source is used. Notice how the shadow of the table is sharp and is well defined over the objects in shadow.

Figure 11: Soft Shadows in Cone Tracing

Cone tracing can produce soft shadows. Courtesy of J. Amanatides.

Figure 12: 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.

Figure 13: Polygonization

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

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.

Figure 15: Self-Shadowing of Facets

The bumps over a surface can be so small that they cannot be seen. Depending on the orientation of these facets, reflection of the light and shadowing is different. Here a Christmas ball is simulated. The facets are defined as threads going from one pole of the sphere to the other. Notice the darker area near one pole.

Figure 16: André's forest

This scene of a forest has been 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.