Realism in Computer Graphics: A Survey

John Amanatides

Dept. of Computer Science
University of Toronto

Introduction

One of the most challenging problems in computer graphics is to generate images that appear realistic; that is, images that can fool a human observer when displayed on a screen. The quest for this "Holy Grail" began in earnest in the early 70's when memory prices dropped low enough to allow raster technologies to be cost-effective over the then prevailing calligraphic displays. Calligraphic displays could only draw a limited number of lines and even the most capable of these displays allowed for only a handful of colours. Previously, research work concentrated on removing "hidden lines" from objects drawn on these displays. The objects displayed were obviously not realistic but contained enough information for the task at hand, such as computer aided design. Raster technology, by subdividing the screen into pixels, allowed whole regions of the screen to be filled with colours, colours that had a wide variety of intensities and tints. This new technology, capable of displaying realistic images, opened up research in this direction and it is this research that we will outline.

This paper will survey most of the major issues that one must deal with when generating realistic images†. We begin with an overview of the rendering process and a quick review of visible surface determination algorithms. We then discuss, in more detail, shading, anti-aliasing, texture mapping, shadows, optical effects and close with a discussion of modeling primitives.

The Rendering Process

The rendering process takes a three dimensional description of a scene and generates a two dimensional array (typically 1024 by 1024) of intensities (pixels) that will be displayed on a CRT. The objects in the scene are usually described by polygons although higher order surfaces are sometimes used. The scene also contains light sources which are often point sources located outside the intended field of view. Finally, the position and viewing direction of the eye/camera is required to completely specify what is to be displayed on the CRT (A pinhole camera model is almost universally used.)

The rendering process begins by transforming the objects into the eye coordinate system. Objects that will be outside the field of view are clipped away. After performing the perspective transformation, the remaining objects must then be compared to see which and what portion of each of them are visible.

As the visible surfaces are found they must be broken down into pixels and shaded correctly. This process must take into account the position and colour of the light sources and the position, orientation and surface properties of the visible object.

Sorting is an integral part of the visible surface problem and typically algorithms try to capitalize on coherence properties in the final image to reduce the amount of sorting required.[1] Three popular visible surface algorithms are the z-buffer, priority, and the scan-line algorithms. In the z-buffer approach, a separate 2-D array is kept (called the z-buffer), indicating the depth (z value) of the pixel currently displayed in the frame buffer. As polygons are broken down into pixel-sized pieces, the depth of each piece (assuming a constant depth per piece) is compared with that in the z-buffer. If the piece is closer, it is written into the frame buffer and the z-buffer is updated to reflect the depth of this new piece. This algorithm, though memory intensive, is very simple and requires no presorting or storage of polygons.

The second visibility algorithm is the priority algorithm. The main idea here is that the polygons are sorted back-to-front and written in the frame buffer in that order. As more and more polygons are sent to

† This article surveys papers up to December 1985.

the frame buffer, they overwrite the polygons that are more distant. Unlike the previous algorithm, no z-buffer is required; however, the polygons must be sorted with respect to depth and this requires both space and time.

The final visible surface algorithm we will outline is the scan line algorithm. There are several variations, but each typically sorts the polygons in one direction (top-down, with a bucket sort) to reduce the number of polygons that must be considered for each horizontal scan line. The algorithm concentrates on finding the visible polygons for each scan line. This reduces the domain of the visible surface problem from polygons to lines, a much easier problem. The variants of this algorithm differ in their strategies for finding these visible lines, also called spans, and in propagating relevant information to neighboring scan lines to reduce subsequent computation (by capitalizing on spatial coherence).

Shading

In this section we start to deal with the interaction of light with matter. In particular, we will deal with point light sources shining on the surfaces of objects. The light reflected off a surface can be broken down into two components- diffuse and specular. When light hits an ideal diffuse surface it is re-radiated equally in all directions. Examples of real surfaces that radiate mostly diffuse light are chalk and flat paints. Ideal specular surfaces only re-radiate light in one direction, the reflected light direction. Examples of specular surfaces are mirrors and shiny surfaces on which highlights are visible. Physically, the difference between these two components is that specular light bounces off the surface of an object while diffuse light penetrates the surface and is scattered internally before emerging again.

The light reflected from real objects contains both diffuse and specular components and both must be modeled to create realistic images. Consider Figure 1. $\vec{E}$ and $\vec{L}$ are unit vectors that point to the eye and light source, respectively, $\vec{E}'$ is a unit vectorin the ideal light direction and $\vec{N}$ is the unit vector indicating the surface normal at the point P. Computing the diffuse component is very simple; it is $\vec{N} \cdot \vec{L}$ which is the well-known Lambert's Law. Note that since diffuse light is radiated in all directions the position of the eye is not required by the computation and the maximum intensity occurs when the surface is perpendicular to the light source.

The specular component is not as easy to compute. Real objects are non-ideal specular reflectors and some light is also reflected slightly off axis from the ideal light direction ($\vec{E}'$). This is because the surface is never perfectly flat but contains microscopic deformations.

The first reasonable approximation to the specular component in computer graphics was proposed by Bui Tuong Phong.[2] It was an empirical approximation and took the form:

$$W(\iota) \cos^n(\alpha)$$

where ι is the incident angle and α is the angle between $\vec{E}'$ and $\vec{L}$. For real objects, as the angle of incidence changes, the ratio of incident light to reflected light also changes and $W(\iota)$ is intended to model this. In practice, however, $W(\iota)$ has been ignored by most implementors.

The value n is the shininess factor. $\cos^n(\alpha)$ reaches a maximum when the light is in the $\vec{E}'$ direction ($\alpha = 0$). As n increases the function dies off much more quickly in the off-axis direction. Thus, a shiny surface with a concentrated highlight, would have a large value of n, while a dull surface with the highlight covering a larger area on the surface, would have a low value of n (Figure 2).

In 1977 Blinn[3] proposed a more accurate model of the specular component that was gleaned from the physics literature. This new model consisted of modeling the surface with a series of microscopic facets, each of which was a perfect reflector, and took the form:

$$\frac{DGF}{\vec{N} \cdot \vec{E}}$$

where D is the microfacet normal distribution function, G is the microfacet self-shadowing function and F is the Fresnel function. The $\vec{N} \cdot \vec{E}$ factor accounts for the increased subtended area when the surface is tilted. D is related to $\cos^n(\alpha)$ in the Phong model. The Fresnel function, which depends on the angle of incidence and index of refraction of the surface, indicates the fraction of the incident light that is reflected. It is related to $W(\iota)$ in the Phong model. Since the Blinn model is based on physics, measurements of real

objects can be used to fine tune the parameters to D, G and F to create more realistic highlights. When comparing images produced by the Phong and Blinn models one sees that they are essentially the same with the biggest difference occurring when the light hits the surface at grazing angles. Here the Blinn model is more accurate. However, since the computation of D, G and F is more expensive than Phong's approximations, many people continue to use the older model.

The most realistic model that is used today in computer graphics is one that was introduced by Cook and Torrance.[4] It is almost identical to Blinn's model with the exception being that the index of refraction parameter used in the Fresnel function is dependent on wavelength. This implies that the colour of the highlight is influenced by the colour of the surface, something that previous researchers had overlooked.

Though there are only two components to reflected light (diffuse and specular) a third component, termed ambient light, is used in computer graphics to model light reaching the surface from multiple reflections off other surfaces or the sky. Since an attempt to accurately model it would be computationally prohibitive, this ambient component is usually approximated with a constant. By using this ambient component, surfaces which do not face any light sources or lie in the shadow of other surfaces are not completely black (Figure 3).

Goral, Torrence and Greenberg[5] have attempted to model the ambient component of light by taking into account interreflections. Objects are modeled with very small polygons and are assumed to be perfectly diffuse. Algorithms gleaned from the physics and engineering literature (radiative heat exchange in enclosures) are used to solve this special case. It involves finding the relative visibility of each pair of polygons and then solving n equations in n unknowns where n is the number of polygons. Though the lack of any specular components and the limited number of polygons allowed make this expensive approach impractical by itself, it is a promising candidate for modeling environmental (ambient) lighting.

Once we know how to shade a point we can consider how to shade a surface. Most surfaces, including those that are curved, must be described by polygonal meshes when the perspective and visible surface calculations are to be performed by the majority of rendering algorithms. Since the normal of a polygon never changes, polygons will have just one shade, making the polygonal representation very evident‡. Gouraud[6] proposed a technique to overcome this: when the curved surface is being broken down into polygons, the true surface normals at the vertices of the polygons are retained. When the polygon is converted into pixels, the correct colours at each of the vertices are computed and these values are linearly interpolated across the polygon. This almost eliminates the impression of the underlying polygons. Unfortunately, Mach bands are sometimes produced and highlights are distorted because of the linear interpolation of the vertex colours. To get around this, Phong[2] proposed the linear interpolation of the surface normal instead, and performed the shading calculation at each pixel (Figure 4). Though this is much more expensive, it produces superior images, especially when the underlying polygons are still rather large†. Phong and Crow[8] reduce the extra shading computations required with normal interpolation by performing it only in polygons where highlights are expected to appear. In the remaining polygons Gouraud shading is performed.

To correctly calculate the intensity of a surface, we should also have a good model for the light source. Most graphics packages assume that the light source is either at infinity (parallel light rays) or is a point source near or within the scene. Unfortunately, these sources are difficult to use creatively. Warn[9] has tried to create more realistic light sources by mimicking the lights used by photographers. His extensions include making the intensity of the point light source a function of direction (to produce spotlights) and providing flaps that can cut off the light in certain directions. Nishita, Okamura and Nakamae[10] and Verbeck and Greenberg[11] extended this by providing more general intensity distributions for these point sources and more sophisticated lighting design tools.

---

‡ Actually, if the eye or the light source is very close to the surface, the shade of the pixels within the polygon will differ significantly.

†Another problem with linear interpolation is that it is not rotation invariant. This is most evident in animated sequences where large concave polygons are present. See Duff[7] for a more thorough discussion.

Sampling and Filtering

Many computer synthesized images exhibit annoying defects, such as jagged edges, distortions of very small objects and inconsistencies in areas of complicated detail (Figure 5). These distortions are the results of improper sampling of the original image and are called *aliasing* artifacts. To understand why we get these problems and how to solve them (perform *anti-aliasing*), we must look briefly at sampling theory.

Suppose we sample a continuous signal $I(x)$ at n regularly spaced points. Is it possible to reconstruct the original signal from the samples? This depends on the frequency components of the original signal. If it contains no frequencies greater than $n/2$ cycles per sampling period, we can reconstruct the original signal; if it contains frequencies greater than $n/2$ the reconstructed signal will always be incorrect.[12] That is, a discrete signal of n points can only represent frequencies below $n/2$ cycles per sampling period uniquely; all higher frequencies in the original signal will be represented (or, aliased) somewhere between 0 and $n/2$. If we look at the frequency components of this reconstructed signal, we cannot tell if they are legitimate or are distortions introduced by the undersampling of the original signal (Figure 6).

In computer graphics, the signal $I(x, y)$ is a two dimensional function that represents the intensity of light passing through the viewing screen. We sample this intensity function in order to obtain a pixel-based representation of intensity. These samples are stored in a frame buffer and the intensity function is reconstructed on the monitor by the display hardware. Unfortunately, $I(x, y)$ will typically contain high frequency components. Therefore, when we sample $I(x, y)$, aliasing problems are inevitable. What we need then, is a way of removing the offending high frequency components in $I(x, y)$ before sampling.

The simplest solution is to increase the sampling rate. As n increases, we can represent higher frequencies. Alas, computing time also increases in proportion to the number of samples. Also, display hardware limits the number of pixels we can display and thus the sampling rate. We can try to go around the hardware limitation by *supersampling* (sampling at higher than screen resolution and averaging) but this is also just as expensive. Consequently, researchers in computer graphics have looked at less expensive yet effective approaches to anti-aliasing.

Let us look again at sampling theory. Given an intensity function $I(x, y)$, is it possible to produce $I'(x, y)$ which has the same frequency components for frequencies less than some frequency ω_0 and has no frequency components greater than ω_0? This is desirable since we can now sample $I(x, y)$ to get the pixel intensities and know that no aliasing problems can occur. Sampling theory states that if we perform the following convolution:

$$I'(x, y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} I(\alpha, \beta)\, H(x - \alpha)\, H(y - \beta)\, d\alpha\, d\beta$$

with the appropriate sampling filter H:

$$H(u) = \frac{\sin(\omega_0 u)}{\pi u}$$

we get exactly what we want. This convolution is just a weighed average of the intensity function where H indicates the relative weight of a point as a function of distance (u) from the pixel center. Unfortunately, this convolution is in practice impossible. Computing an infinite double integral for each pixel is just too expensive. Also, the intensity function is too complicated to easily convolute analytically. Consequently, simplifications and approximations have been used in computer graphics to compute the convolution in reasonable time.

The two major simplifications are to use a simpler sampling filter and to capitalize on the coherence properties of the intensity function. The sampling filter is typically a box filter or a truncated Gaussian (Figure 7). In both cases, the filter usually does not spread over an area greater than three by two three regions (an approximation of the major lobe of the ideal sampling filter when the cut-off frequency is $n/2$) and many times is confined to the area of one pixel.

The intensity function approximation begins with the observation that as we sample along a polygon the signal will change relatively slowly. It is only when we cross polygon boundaries that great fluctuations in the intensity will occur. Thus we can assume that within a pixel the intensity of each polygon is

constant. This implies that we have to calculate the shade of each polygon within a pixel only once, a great saving in computation. Consequently, the area and position that a polygon covers within a pixel along with just one shade computation for that polygon is enough to calculate that polygon's contribution to the intensity function at that pixel.

The sampling theory outlined above was first applied to computer graphics by Crow[13] in the middle '70's. He computed the area coverage of a polygon and used a triangular filter to get a weight for the contribution of that polygon to the pixel. (Actually, he only performed the extra computations where the intensity was expected to change, such as object silhouettes or polygon boundaries when the surface colour changes. This adaptive filtering reduced the convolution computations, especially in scenes with large polygons). The weighting plus one shading computation for that polygon per pixel was sufficient to perform the anti-aliasing by Crow.

Catmull[14] later performed a visible surface computation at each pixel to produce polygon fragments that tessellated the pixel. Thus one could compute the exact area that each polygon occupied within the pixel. A simple box filter extending only to the edge of the pixel was used. Unfortunately, the visible surface computation at each pixel was too expensive to make the algorithm widely applicable.

Feibush, Levoy and Cook[15] extended Catmull's approach by adding a Gaussian filter By breaking down the fragments into triangles, they were able to use table lookup to approximate the weight of each of the fragments.

Fiume, Fournier and Rudolph[16] suggested a less accurate but much faster approximation of the area coverage of each polygon within each pixel by using 8 by 8 coverage masks. They also used a box filter. Variants of this approach can be seen in work by Carpenter[17] and Abram, Westover and Whitted.[18] The later is notable in that a better filter via table look-up is utilized.

Bloomenthal[19] has suggested a variation of the standard z-buffer that has a post-visible surface determination filtering step. It is fast but, in many cases, inadequate.

Another method of anti-aliasing that was suggested by Whitted[20] is adaptive sampling. In this scheme, the intensity is computed at the corners of the pixel. If the resulting values vary significantly, the pixel is subdivided into four sub-pixels and the process is performed recursively on each of the sub-pixels. Unfortunately, since this method uses no information other than the sampled intensities, it can fail.

In all the above approaches to anti-aliasing we have assumed that we would have problems only at the boundaries of polygons. Unfortunately this is not always true. For example, I(x,y) will contain high frequencies whenever a displayed object's surface normal changes significantly within a pixel. This variation in the surface normal will cause the specular component (which is the one that is most sensitive to orientation) to introduce very high frequencies when performing the shading computations (Figure 5). Aliasing errors can also occur in the interior of polygons when highly directional light sources introduce abrupt intensity changes.

There have been only two papers that have suggested solutions to this problem. Crow[21] has proposed computing the specular component at higher resolution in pixels where the surface normal changes significantly. A drawback to this expensive approach is that the user has to manually set a "threshold curvature" at which the highlight component was to be super-sampled. Williams[22] suggested replacing the highlight calculation with a spherical texture map (discussed in the next section). This approach requires a great deal of memory and will not generate as accurate highlights as the direct method.

In the above paragraphs we have been discussing ways of anti-aliasing static images. Animation, a sequence of still images, introduces a new dimension: time. The convolution must now occur in three dimensions to also filter out the aliasing introduced by sampling in the time dimension.

There has been relatively little work done on temporal anti-aliasing. A typical simplification that researchers have made is to assume a constant velocity of objects during each frame of the animation. Korein and Badler,[23] Potmesil and Chakravarty[24] and Max and Lerner[25] introduced simple approximations while Cook[26] used a more accurate though computationally expensive approach. Catmull[27] used a simplification that projected the temporal problem into the spatial domain by distorting the 2-D filter to account for

the velocity of objects. Grant[28] proposed an analytic solution by modeling moving polygons as 4 dimensional polyhedra. It unfortunately is an expensive and very involved procedure that does not look promising.

We have so far discussed sampling a signal at regularly spaced points. Recently, work by Lee, Redner and Uselton[29] and Dippe and Wold[30] has concentrated on stochastic sampling; that is, sampling the signal at irregularly spaced points. The motivation for this is as follows: A signal that causes aliasing when sampled at regular points can dissipate its energy as broadband noise when stochastically sampled. Instead of the "jaggies" we get noise. As we increase the sampling rate, the amount of noise in the image decreases. It is felt that this noise is less objectionable to the human observer than the aliasing present when sampling at regularly spaced points. The work so far has concentrated on finding good sampling distributions, filtering methods and adaptively increasing the number of samples in regions of the image where high frequencies are present. It has been used primarily in ray tracing (discussed in a later section) because that technique is inherently an expensive point sampling process that has proved to be difficult to anti-alias.

Texture

To provide the illusion of reality we must be capable of displaying complex scenes. For example, if we are modeling a room, we should be able to include portraits on the wall or Persian rugs on the fbor. These objects, rich in high frequencies, could be modeled by many individual polygons, but these resulting polygons could easily swamp the modeling and display programs as the number of polygons increases by several orders of magnitude. The technique of texture mapping was introduced by Catmull[31] to provide this illusion of complexity at a reasonable cost (Figure 8). Basically it is a method of "wallpapering" the existing polygons. Each vertex on a textured polygon contains coordinates in a two dimensional texture space. As each pixel is shaded, the texture coordinates are interpolated and a look-up is performed into a two dimensional array of colours containing the texture. The value in this array is used as the colour of the polygon at that pixel, thus providing the "wallpaper" (Figure 9).

Unfortunately, textures are very susceptible to aliasing since they contain high frequencies. Also, the polygon that is using the textures may occupy a small portion of the screen, forcing many texture points onto the same pixel. Blinn and Newell[32] refined the technique with the use of filters. Feibush, Levoy and Cook[15] suggested Gaussian filters, though the computation could be expensive. Williams[22] suggested filtering schemes that, though not as accurate as others, allowed for fast (constant time) computation of the texture. He kept copies of the texture at various resolutions and used the ones that were most appropriate. Crow[33] made use of a precomputed table storing the integral of the texture map so that filtering could also be done in constant time. Norton, Rockwood and Skolmoski[34] suggested that when the texture is an analytic function, precomputing the frequency spectrum and reconstructing the function without any of the offending frequencies could be an alternate way of computing the textures rapidly. This is useful for such fuzzy objects as clouds, waves and terrain where the texture does not have to contain many frequency components.

A problem with texture is that it is hard to "wrap" a two dimensional texture around complex three dimensional objects. One alternative is to define a function that maps the object's spatial coordinates into three dimensional texture space and use three dimensional textures. Thus no matter what the object's shape the texture on its surface is consistent. This is especially useful if the texture models the material out of which the object is made, such as wood or marble. Because of the extraordinary amount of space required to store a three dimensional array of pixels, procedural textures have been used by Peachey[35] and Perlin.[36] Procedural textures however, are, in general, difficult to anti-alias and thus the elegance of the technique is easily compromised.

Textures need not just contain the colour of a surface. Another way of using texture mapping to increase the illusion of complexity is to store other surface properties instead of just the surface colour. Blinn[37] suggested that surface normal perturbations could be stored in the texture map (bump mapping). The resulting variation in the highlights provides a very convincing simulation of wrinkled surfaces. This is even more apparent in animated sequences whenever the bump-mapped object moves since the highlights match the surface perturbations correctly. Gardner used analytic textures to modify the surface boundaries

of ellipsoids.[38, 39] for very convincing trees and clouds.

Another place where texture techniques have been used effectively is "environment mapping". If objects are shiny, objects in the surrounding environment should be reflected, however faintly, by the reflecting surface. Computing these reflections can be very expensive, especially in animated sequences. This is where textures can step in. We can project an approximation of the world (the environment) as seen from one viewpoint onto a sphere, cylinder or box and store the projection in a texture map.[32, 22, 40] When we later shade a point, the texture is sampled in the reflected direction. We have to be careful though for we have the same sampling problems here as with regular texture mapping.

Shadows

Shadows play an important role in conveying reality in computer synthesized images. They also facilitate the comprehension of spatial relationships between objects. Though there are situations where shadows are not required (eg. if the light source is behind the viewer or when modeling an overcast day), there are many other situations in which they are essential. This section will outline the research that has already performed.

The complexity of a shadow algorithm is related to the model of the light source. If it is a point source outside the field of view or at infinity, the problem is simplified. Finding which objects are in shadow is then equivalent to solving the visible surface problem as viewed from the light source. If the source is not a point source or is inside the field of view the problem becomes much more difficult.

Crow[41] has proposed a taxonomy of shadow algorithms which consists of three classes: shadow computation during scan-out;[42, 43, 44] division of object surfaces into shadowed and unshadowed areas prior to scanout;[45, 46, 47, 48] and inclusion of shadow volumes into the object data.[41]

Appel was one of the first computer graphics researchers to study shadows extensively.[42] He suggested three different solutions, two being variations of ray casting (to be discussed below), and the third a variation of a hidden line algorithm that he had previously used. He would create spans generated by the intersection of a plane passing through the eye, viewing screen and objects in the scene. These spans would represent the visible parts of an object. He would then use his hidden line algorithm to see what part of the span was visible from the light source. Bouknight and Kelly used a similar approach[43] where they projected possible shadowing polygons onto the plane of the polygon containing the span and then compared the span with the projected polygons. To reduce the number of polygons that had to be compared when the span was checked, they preprocessed the polygons by transforming them into a spherical coordinate space centered at the light source. Polygons that overlapped were marked, indicating possible shadowing. During scan-out only the marked polygons would be checked against the span.

Goldstein and Nagel used a visible surface algorithm called *ray casting*.[44] In it a ray is sent from the eye, through the pixel center and into the world (Figure 10). The points of intersection between the ray and the objects in the scene are found and the one closest to the eye represents the visible surface. An advantage of this algorithm is that objects such as ellipsoids or cones need not be broken down into polygons but can be intersected directly. To get shadow information, Goldstein and Nagel simply fired a ray from the intersection point towards the light source. If it intersected anything, the visible surface was in shadow. This method of finding shadows is very simple but the cost of computing the intersections is usually very high.

Nishita and Kakamae[45] introduced a two step shadow algorithm for convex polyhedra made up of convex polygons. The first step consisted of a visible surface determination from the light source. By clipping the polygons to the silhouettes of the polyhedra, they were able to subdivide them into two categories after the first step, visible and shadowed. The polygons were marked and a second visible surface determination, from the eye, was performed. Since the polygons were appropriately marked, the shadow computations were trivial.

Atherton, Weiler and Greenberg[46] extended this approach to handle more general environments by encorporating a more powerful polygon clipping algorithm that allowed for concave polygons.

Williams[48] also used a two pass process. However, he used a z-buffer algorithm in both steps. The advantage of this approach was that it was simple and that non-planar objects could easily be accommodated. Unfortunately, the z-buffer algorithm suffers from aliasing problems and this was exasperated by the

first pass from the light source. Williams suggested solutions to reduce the aliasing problem though they do not work well in many situations. Hourcade and Nicolas[47] modify Williams' basic approach by using a priority algorithm for the first pass with a better method of anti-aliasing so that the sampling problems are not as evident.

Crow[41] has advocated computing the volumes swept out by the shadows of objects and including them in the data base (in the form of "shadow" polygons). The shadow polygons defining these volumes are invisible but the visible surface algorithm (a scan line algorithm) uses these polygons to check if any of the visible polygons that it has found are within the shadow volumes. Thus a polygon is in shadow only if it is straddled by at least two shadow polygons, one indicating the front face of a shadow volume and another indicating the same shadow volume's rear face.

Solving the shadow problem when we have non point light sources is more difficult. Nishita and Nakamae have extended Crow's idea of using shadow volumes to permit the generation of penumbras that are cast by area[49] and linear[10] light sources. The environment is assumed to be made up of convex polyhedra.

When extending the work of Goral et. al.[5] on ambient light determination so that visible surfaces could be computed, Cohen and Greenberg[50] and Nishita and Nakamae[51] were also able to compute the shadows cast by non point sources. Unfortunately, the techniques are expensive and can handle only limited environments. Finally, more solutions to this problem will be discussed in the next section.

Optical Effects and Ray Tracing

Optical effects and ray tracing is an area in which significant research effort has been expended recently. This topic includes the modeling of transparency, reflection, refraction and camera models that have lenses instead of the pinhole camera model. This last element allows for visual effects such as focusing and depth-of-field.

There are two reasons for recent work in this field. First, these effects are very important as we try to model reality, and second, because the approach to solving these problems- ray tracing- is very simple both conceptually and algorithmically.

Ray tracing was developed by Whitted[20] and is an extension of the ray casting process used by Appel,[42] and Goldstein and Nagel.[44] As in ray casting, a ray is fired from the eye, through the pixel and into the world. The closest intersection between this ray and the objects in the world determines the visible surface. Shadows are determined by firing rays towards the light sources. Whitted extended this ray casting process into ray tracing by firing off two additional rays from the intersection point, one along the reflected direction and the other along the direction of transmission (Figure 11, 12). Using ray optics, he was able to model the distortions of reflecting and refracting surfaces accurately, thus producing stunning images.

There had been earlier attempts to model some of these effects. For example, reflection had been modeled by Blinn and Newell's "environment" texture mapping.[32] Unfortunately, the "environment" map sphere was at infinity and, thus, neighboring effects, such as the relative motion of objects, could not be modeled. Transparency had often been modeled by allowing the other surfaces behind the visible surface to show through. This was generally unconvincing since the refractive distortions were missing. Kay and Greenberg[52] tried approximating these distortions but their approach was not as accurate and was overshadowed by Whitted's work.

There are two drawbacks to ray tracing: computational expense and aliasing. Whitted's images took on the order of several VAX 780 CPU hours to compute. There are several reasons for the computational expense. First, all the coherence information is lost as each ray is traced independently of all the others. Second, the intersection calculations are floating point intensive and typically require root finding of polynomial functions. The aliasing problems result from the fact that since we are intersecting an object with a ray (line), we are forced to point sample at the intersection point. There is no way, for example, of area filtering as we cannot know how much of the pixel the intersected object occupies. Consequently, Whitted used super-sampling as his method of anti-aliasing. He did it in an adaptive manner though, by firing rays at the corners of a pixel and recursively subdividing and refiring rays if the intensity of the original corner rays differed significantly.

Ray tracing, with its beautiful images and its computationally straightforward approach, convinced many researchers to continue in this direction. They concentrated their research in four areas: finding intersection algorithms for various objects, extending the range of optical effects that could be captured using ray tracing, anti-aliasing, and reducing the total number of intersection tests. The majority of the early research concentrated on quickly finding the intersections of rays with more complicated objects.[53, 54, 55, 56, 57, 58, 59, 60] An advantage over the traditional approach to rendering was that, in general, the objects did not have to be broken down into polygons (with the resulting inaccuracies and extra intersection calculations), but could be directly rendered in their "natural" representation.

To speed up the intersection calculations and improve on anti-aliasing, Heckbert and Hanrahan worked with a cluster of rays at a time.[61] For simple scenes this reduced the computations dramatically, but the approach could not handle curved objects or refraction very well.

To help solve the aliasing problem when ray tracing, Amanatides generalized the concept of a ray from a line to a cone representing the cross-sectional area of a pixel.[62] Now, the intersection calculation could return the area of intersection between an object and a ray, thus providing a way of filtering. This approach also allowed for the computation of dull reflections and penumbras cast by non-point sources (Figure 13). Unfortunately, the intersection calculations become more complicated.

Potmesil and Chakravarty[63] used the information generated by ray tracing to compute the effects of using a camera with a lens and aperture. Cook, Porter and Carpenter[26] have also extended Whitted's original approach to ray tracing. By extending Whitted's original work with regards to highlight generation using ray tracing, they were able to model soft shadows, dull reflections, depth-of-field and temporal anti-aliasing. Their approach was to super-sample and distribute the samples effectively among the various dimensions to be sampled (stratified sampling). For example, when modeling a dull reflection, the reflected rays do not just follow the ideal reflected direction but are perturbed by an amount related to how dull the surface is to appear. Though not mentioned explicitly in the paper, they used stochastic sampling to reduce aliasing. As mentioned earlier, Lee, Redner and Uselton[29] and Dippe and Wold[30] have done similar work with regards to anti-aliasing.

Approaches to reducing the total number of intersections come in two flavors. The first is to envelop each complex object in a tree of bounding volumes.[20, 53, 64] Sub-objects within a branch of a tree are intersected only if the ray pierces the bounding volume of the branch. The second approach consists of subdividing space itself into regions and noting the objects that are in each region.[65, 66, 67] As a ray propagates from one region to the next, the objects in each region become candidates for ray intersection. Thus the nearest objects are the first candidates for intersection, leading to a quick determination of the closest object. To be really useful, a space subdivision scheme must not only work for primary rays (the original rays sent from the eye), but also for shadows and reflected and refracted rays.

A work that stands apart from those encountered above is that of Moravec.[68] Ray tracing depends on the particle model of light. Moravec suggested using a wave model instead. A wave front would be propagated through the volume occupied by the objects, and reflections and refractions would be modeled by new wave fronts that bounced off the objects in the scene. Though intriguing, the images generated were disappointing and the approach proved to be much too expensive.

Modeling

Most graphics packages, when performing transformations and modeling the visible surfaces of objects work with polygons. Unfortunately, determining complex objects with such simple, low level primitives is both time consuming, complicated and an unnecessary process for the user. Higher level modeling primitives that describe objects within a scene are required. This section covers several of the more popular and novel of these modeling primitives.

Some of the most popular modeling primitives, outside of polygons, are parametric patches or splines.[31, 69] These surfaces are parametric to allow orientation independent curves. They are typically cubic polynomials because these polynomials have been found to be easy to specify yet powerful enough to describe most curved surfaces that have been modeled in computer graphics (Figure 14). Numerous varieties of parametric surfaces have been proposed (see the Barsky paper[69] for a survey). These have different

ways of letting the user specify the shape of the curve (by choosing different basis functions), each trying to give the user controls that are simple, intuitive and yet powerful. One of the most popular at the present time is beta-splines.

For display, two strategies have been used: direct scan conversion of the patches and breaking down the patch into polygons by the graphics package just prior or during display.[70] The first approach has fallen into disfavor as the proposed algorithms have been found to be too cumbersome and difficult to incorporate in many visible surface algorithms.

Another approach to modeling, motivated by other constraints, is that of solid modeling.[71, 55] In this approach, elaborate objects are formed by the union, intersection and difference of simple solid volume primitives, such as spheres, cubes and cylinders. This approach is especially popular in CAD/CAM where machined objects must be modeled. Visible surface determination can easily be computed using either ray casting or variants of the scan line approach.[72]

Probably the hardest class of objects to model has been that of natural objects such as clouds, terrain, fire and the results of biological processes. Human generated objects are fairly regular but natural ones typically have a high degree of complexity that is both very hard to describe and store in a data base. Consequently, many researchers have looked into procedural, stochastic models. The general shape is defined by the user by specifying a few well-chosen parameters and a stochastic process is used to generate the required detail. Scientifically accurate processes are not as important as processes that create visually acceptable results which are also easily integratable into existing graphics methodologies. Examples of stochastic models are: for terrain (Figure 15),[73] fire[74] and fbra.[75, 76, 74, 77]

To create terrain, Fournier and Fussell[73] take a parametric patch and add to it a stochastic element. This is done by generating a two dimensional table of values that approximate fractional Brownian motion, and using these values to displace points on the surface patch in a direction perpendicular to the surface. This allows good global control (the shape of the parametric patch is user-defined) with the details being added by the stochastic process. A popular alternate approach, also in[73] by Carpenter, starts with a triangle and recursively subdivides it using a stochastic process to displace the endpoints of the sub-triangles.

To approximate fire, Reeves[74] used what he called "particle systems". A particle system is a procedural model that stochastically generates a series of moving points. The user defines the mean number of particles generated, their lifetime, velocity and colour and the stochastic process creates, typically, several thousand of these particles, assigning to each its own velocity, colour and lifetime. In the image, these moving particles are represented by straight lines. By having multiple particle systems and by summing up for each pixel the contribution from all the particles, convincing images of explosions or fire can be generated. In a similar manner, grass and trees have also been produced with the particle streaks forming the branches and leaves.[77]

Aono and Kunii[75] and Smith[76] both used a "grammar" approach to generate procedural models of trees. They both used a formal language describing the branching patterns of trees devised by biologists whose production rules generate a string representing the branching patterns of a tree. This string is then "interpreted" to generate an instance of a tree. The way the string is interpreted depends on the type of tree to be modeled and a stochastic element can be introduced to give trees individual characteristics.

Once we have modeled a surface we must be capable of animating it. Unfortunately, motion specification and control is still rather limited. It has primarily dealt with the motion of rigid objects such as skeletons or robots.[78, 79, 80, 81, 82, 83] Two general approaches researchers have taken are 1), parametric inbetweening of key frames and 2), more "intelligent" goal-directed animation. The fact that living things are not rigid but bend as they move has not been adequately addressed by current modeling packages.

Concluding Remarks

We have reached the stage where computer synthesized images are acceptable for some special effects in the movie industry (eg. **TRON, STAR TREK II** and **The Last Starfighter**) and in commercials on TV. But we still have not found our "grail". Overall, computer synthesized images have a "sanitized" look to them; no dirt or garbage in the corners; straight lines and perfectly flat faces instead of the familiar flaws; plain looking scenes with a distinct lack of detail; simple, rigid motions instead of motions which

flow and deform the object. It is only when we go past these limitations that we will approach reality.

I would like to thank Alain Fournier, Eugene Fiume, Dave Fleet and Darwyn Peachey for going over earlier drafts of this paper and giving me valuable suggestions. Thanks also to the reviewers who gave me many helpful comments.

References

1. Sutherland, I.E., Sproull, R.F., and Schumacker, R.A., "A Characterization of Ten Hidden-Surface Algorithms," *Computing Surveys*, 6(1), pp. 1-55 (March 1974).
2. Bui T. Phong, "Illumination for Computer Generated Pictures," *Comm. of the ACM*, 18(6), pp. 311-317 (June 1975).
3. Blinn, J.F., "Models of Light Reflection For Computer Synthesized Pictures," *Computer Graphics*, 11(2), pp. 192-198 (July 1977).
4. Cook, R.L. and Torrance, K.L., "A Reflectance Model for Computer Graphics," *ACM Trans. on Graphics*, 1(1), pp. 7-24 (January 1982).
5. Goral, C.M., Torrence, K.E., Greenberg, D.P., and Battaile, B., "Modeling the Interaction of Light Between Diffuse Surfaces," *Computer Graphics*, 18(3), pp. 213-222 (July 1984).
6. Gouraud, H., "Continuous Shading of Curved Surfaces," *IEEE Trans. on Computers*, C-20(6), pp. 623-629 (June 1971).
7. Duff, T., "Smoothly Shaded Renderings of Polyhedral Objects on Raster Displays," *Computer Graphics*, 19(2), pp. 270-275 (August 1979).
8. Bui T. Phong and Crow, F.C., "Improved Rendition of Polygonal Models of Curved Surfaces," *Proc. of the 2nd USA-Japan Computer Conference* (1975).
9. Warn, D.R., "Lighting Controls for Synthetic Images," *Computer Graphics*, 17(3), pp. 13-21 (July 1983).
10. Nishita, T., Okamura, I., and Nakamae, E., "Shading Models for Point and Linear Sources," *ACM Trans. on Graphics*, 4(2), pp. 124-146 (April 1985).
11. Verbeck, C.P. and Greenberg, D.P., "A Comprehensive Light-Source Description for Computer Graphics," *IEEE Computer Graphics and Applications*, 4(7), pp. 66-75 (July 1984).
12. Pratt, W.K., *Digital Image Processing*, Wiley-Interscience (1978).
13. Crow, F.C., "The Aliasing Problem in Computer-Generated Shaded Images," *Comm. of the ACM*, 20(11), pp. 799-805 (November 1977).
14. Catmull, E., "A Hidden-Surface Algorithm with Anti-Aliasing," *Computer Graphics*, 12(3), pp. 6-10 (August 1978).
15. Feibush, E.A., Levoy, M., and Cook, R.L., "Synthetic Texturing Using Digital Filters," *Computer Graphics*, 14(3), pp. 294-301 (July 1980).
16. Fiume, E., Fournier, A., and Rudolph, L., "A Parallel Scan Conversion Algorithm with Anti-Aliasing for a General Purpose Ultracomputer," *Computer Graphics*, 17(3), pp. 141-150 (July 1983).
17. Carpenter, L., "The A-buffer, an Antialiased Hidden Surface Method," *Computer Graphics*, 18(3), pp. 103-108 (July 1984).
18. Abram, G., Westover, L., and Whitted, T., "Efficient Alias-Free Rendering using Bit-Masks and Look-Up Tables," *Computer Graphics*, 19(3), pp. 53-59 (July 1985).
19. Bloomenthal, J., "Edge Inference with Applications to Antialiasing," *Computer Graphics*, 17(3), pp. 157-162 (July 1983).
20. Whitted, T., "An Improved Illumination Model for Shaded Display," *Comm. of the ACM*, 23(6), pp. 343-349 (June 1980).
21. Crow, F.C., "Computational Issues in Rendering Anti-Aliased Detail," *IEEE 1982 Spring COMP-CON*, pp. 238-244 (1982).

22. Williams, L., "Pyramidal Parametrics," *Computer Graphics*, 17(3), pp. 1-11 (July 1983).

23. Korein, J. and Badler, N., "Temporal Anti-Aliasing in Computer Generated Animation," *Computer Graphics*, 17(3), pp. 377-388 (July 1983).

24. Potmesil, M. and Chakravarty, I., "Modelling Motion Blur in Computer-Generated Images," *Computer Graphics*, 17(3), pp. 389-399 (July 1983).

25. Max, N.L. and Lerner, D.M., "A Two-and-a-Half-D Motion-Blur Algorithm," *Computer Graphics*, 19(3), pp. 85-93 (July 1985).

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

27. Catmull, E., "An Analytic Visible Surface Algorithm for Independent Pixel Processing," *Computer Graphics*, 18(3), pp. 109-115 (July 1984).

28. Grant, C.W., "Integrated Analytic Spatial and Temporal Anti-Aliasing for Polyhedra in 4-Space," *Computer Graphics*, 19(3), pp. 79-84 (July 1985).

29. Lee, M.E., Redner, R.A., and Uselton, S.P., "Statistically Optimized Sampling for Distributed Ray Tracing," *Computer Graphics*, 19(3), pp. 61-67 (July 1985).

30. Dippe, M.A.Z. and Wold, E.H., "Antialiasing Through Stochastic Sampling," *Computer Graphics*, 19(3), pp. 69-78 (July 1985).

31. Catmull, E., *A Subdivision Algorithm for Computer Display of Curved Surfaces*, PhD Thesis, University of Utah (1974).

32. Blinn, J.F. and Newell, M.E., "Texture and Reflection in Computer Generated Images," *Comm. of the ACM*, 19(10), pp. 542-547 (October 1976).

33. Crow, F.C., "Summed-Area Tables for Texture Mapping," *Computer Graphics*, 18(3), pp. 207-212 (July 1984).

34. Norton, A., Rockwood, A.P., and Skolmoski, P.T., "Clamping: A Method of Antialiasing Textured Surfaces by Bandwidth Limiting in Object Space," *Computer Graphics*, 16(3), pp. 1-8 (July 1982).

35. Peachey, D.R., "Solid Texturing of Complex Surfaces," *Computer Graphics*, 19(3), pp. 279-286 (July 1985).

36. Perlin, K., "An Image Synthesizer," *Computer Graphics*, 19(3), pp. 287-296 (July 1985).

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

38. Gardner, G.Y., "Simulation of Natural Scenes using Textured Quadratic Surfaces," *Computer Graphics*, 18(3), pp. 11-20 (July 1984).

39. Gardner, G.Y., "Visual Simulation of Clouds," *Computer Graphics*, 19(3), pp. 297-303 (July 1985).

40. Greene, N., "A Method of Modeling Sky for Computer Animation," *Proc. Intl. Conf. of Engineering and Computer Graphics, Beijing China*, pp. 297-300 (August 1984).

41. Crow, F.C., "Shadow Algorithms for Computer Graphics," *Computer Graphics*, 11(3), pp. 242-248 (July 1977).

42. Appel, A., "Some techniques for shading machine renderings of solids," *Proc. AFIPS JSCC*, 32, pp. 37-45 (1968).

43. Bouknight, W.J. and Kelly, K.C., "An algorithm for producing half-tone computer graphics presentations with shadows and movable light sources," *Proc. AFIPS JSCC*, 36, pp. 1-10 (1970).

44. Goldstein, R.A. and Nagel, R., "3-D Visual Simulation," *Simulation*, pp. 25-31 (January 1971).

45. Nishita, T. and Nakamae, E., "An Algorithm for Half-Tone Representation of Three-Dimensional Objects," *Information Processing in Japan*, 14, pp. 93-99 (1974).

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

47. Hourcade, J.C. and Nicolas, A., "Algorithms for Antialiased Cast Shadows," *Computers and Graphics*, 9(3), pp. 259-265 (1985).

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

49. Nishita, T. and Nakamae, E., "Half-Tone Representation of 3-D Objects Illuminated by Area or Polyhedron Sources," *Proc. of IEEE Computer Society's Seventh International Computer Software and Applications Conference (COMPSAC83)*, pp. 237-242 (Nov 7-11 1983).

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

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

52. Kay, D.S. and Greenberg, D., "Transparency for Computer Synthesized Images," *Computer Graphics*, 13(2), pp. 158-164 (August 1979).

53. Rubin, S.M. and Whitted, T., "A 3-Dimensional Representation for Fast Rendering of Complex Scenes," *Computer Graphics*, 14(3), pp. 110-116 (July 1980).

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

55. Roth, S.D., "Ray Casting for Modeling Solids," *Computer Graphics and Image Processing*, 18, pp. 109-144 (1982).

56. Hall, R.A. and Greenberg, D.P., "A Testbed for Realistic Image Synthesis," *IEEE Computer Graphics and Applications*, 3(8), pp. 10-20 (November 1983).

57. Hanrahan, P., "Ray Tracing Algebraic Surfaces," *Computer Graphics*, 17(3), pp. 83-90 (July 1983).

58. Kajiya, J.T., "New Techniques For Ray Tracing Procedurally Defined Objects," *Computer Graphics*, 17(3), pp. 91-102 (July 1983).

59. Van Wijk, J.J., "Ray Tracing Objects Defined By Sweeping Planar Cubic Splines," *ACM Trans. on Graphics*, 3(3), pp. 223-237 (July 1984).

60. Van Wijk, J.J., "Ray Tracing Objects Defined By Sweeping a Sphere," *Computers and Graphics*, 9(3), pp. 283-290 (1985).

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

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

63. Potmesil, M. and Chakravarty, I., "Synthetic Image Generation with a Lens and Aperture Camera Model," *ACM Trans. on Graphics*, 1(2), pp. 85-108 (April 1982).

64. Weghorst, H., Hooper, G., and Greenberg, D.P., "Improved Computational Methods for Ray Tracing," *ACM Trans. on Graphics*, 3(1), pp. 52-69 (January 1984).

65. Cleary, J.G., Wyvill, B., Birtwistle, G.M., and Vatti, R., "Multiprocessor Ray Tracing," *Research Report No. 83/128/7 Dept. of Computer Science University of Calgary* (1983).

66. Glassner, A.S., "Space Subdivision for Fast Ray Tracing," *IEEE Computer Graphics and Applications*, 4(10), pp. 15-22 (October 1984).

67. Fujimoto, A. and Iwata, K., "Accelerated Ray Tracing," *Proc. CG Tokyo '85*, pp. 41-65.

68. Moravec, H.P., "3D Graphics and the Wave Theory," *Computer Graphics*, 15(3), pp. 289-296 (August 1981).

69. Barsky, B.A., "A Description and Evaluation of Various 3-D Models," *IEEE Computer Graphics and Applications*, 4(1), pp. 38-52 (January 1984).

70. Lane, J.M., Carpenter, L.C., Whitted, T., and Blinn, J.F., "Scan Line Methods for Displaying Parametrically Defined Surfaces," *Comm. of the ACM*, 23(1), pp. 23-34 (January 1980).

71. Requicha, A.A.G., "Representations for Rigid Solids: Theory Methods and Systems," *Computing Surveys*, 12(4), pp. 437-464 (December 1980).

72. Atherton, P.R., "A Scan-Line Hidden Surface Removal Procedure for Constructive Solid Geometry," *Computer Graphics*, 17(3), pp. 73-82 (July 1983).

73. Fournier, A., Fussell, D., and Carpenter, L., "Computer Rendering of Stochastic Models," *Comm. of the ACM*, 25(6), pp. 371-384 (June 1982).

74. Reeves, W.T., "Particle Systems- A Technique for Modelling a Class of Fuzzy Objects," *Computer Graphics*, 17(3), pp. 359-376 (July 1983).

75. Aono, M. and Kunii, T.L., "Botanical Tree Image Generation," *IEEE Computer Graphics and Applications*, 4(5), pp. 10-34 (May 1984).

76. Smith, A.R., "Plants Fractals and Formal Languages," *Computer Graphics*, 18(3), pp. 1-10 (July 1984).

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

78. Reeves, W.T., "Inbetweening For Computer Animation Utilizing Moving Point Constraints," *Computer Graphics*, 15(3), pp. 263-269 (August 1981).

79. Korein, J.U. and Badler, N.I., "Techniques for Generating the Goal-Directed Animation of Articulated Structures," *IEEE Computer Graphics and Applications*, 2(9), pp. 71-81 (November 1982).

80. Reynolds, C., "Computer Animation with Scripts and Actors," *Computer Graphics*, 16(3), pp. 289-296 (July 1982).

81. Zeltzer, D., "Motor Control Techniques for Figure Animation," *IEEE Computer Graphics and Applications*, 2(9), pp. 53-59 (November 1982).

82. Girard, M. and Maciejewski, A.A., "Computational Modeling of the Computer Animation of Legged Figures," *Computer Graphics*, 19(3), pp. 263-270 (July 1985).

83. Steketee, S.N. and Badler, N.I., "Parametric Keyframe Interpolation Incorporating Kinetic Adjustment and Phrasing Control," *Computer Graphics*, 19(3), pp. 255-262 (July 1985).