BEAM TRACING POLYGONAL OBJECTS

Paul S. Heckbert
Pat Hanrahan

Computer Graphics Laboratory
New York Institute of Technology
Old Westbury, NY 11568

Abstract

Ray tracing has produced some of the most realistic computer generated pictures to date. They contain surface texturing, local shading, shadows, reflections and refractions. The major disadvantage of ray tracing results from its point-sampling approach. Because calculation proceeds *ab initio* at each pixel it is very CPU intensive and may contain noticeable aliasing artifacts. It is difficult to take advantage of spatial coherence because the shapes of reflections and refractions from curved surfaces are so complex.

In this paper we describe an algorithm that utilizes the spatial coherence of polygonal environments by combining features of both image and object space hidden surface algorithms. Instead of tracing infinitesimally thin rays of light, we sweep areas through a scene to form "beams." This technique works particularly well for polygonal models since for this case the reflections are linear transformations, and refractions are often approximately so.

The recursive beam tracer begins by sweeping the projection plane through the scene. Beam-surface intersections are computed using two-dimensional polygonal set operations and an occlusion algorithm similar to the Weiler-Atherton hidden surface algorithm. For each beam-polygon intersection the beam is fragmented and new beams created for the reflected and transmitted swaths of light. These sub-beams are redirected with a 4x4 matrix transformation and recursively traced. This beam tree is an object space representation of the entire picture.

Since the priority of polygons is pre-determined, the final picture with reflections, refractions, shadows, and hidden surface removal is easily drawn. The coherence information enables very fast scan conversion and high resolution output. Image space edge and texture antialiasing methods can be applied.

CR Categories: I.3.3 **[Computer Graphics]**: Picture/Image Generation - *display algorithms;* I.3.7 **[Computer Graphics]**: Three-Dimensional Graphics and Realism - *visible line/surface algorithms.*

General Terms: algorithms.

Additional Key Words and Phrases: ray tracing, refraction, polygon, object space, coherence.

1. Introduction

Two of the most popular methods used to create frame buffer images of three-dimensional environments are ray tracing and scan line algorithms. Ray tracing generates a picture by casting a ray of light from the eye point through each pixel of the image and into the scene. Visible surfaces are determined by testing for line-surface intersections between the ray and each object in the scene. By recursively tracing reflected and refracted rays, considerable realism can be added to the final image. In contrast, a scan line rendering program generally takes advantage of coherence to draw surfaces incrementally. Comparing the two approaches we find:

Advantages of ray trace:
- Uses a global lighting model that calculates reflections, refractions, shadows.
- Can handle a variety of geometric primitives.

Disadvantages of ray trace:
- Often slow since the intersection calculations are floating point intensive.
- Point sampling the environment causes aliasing.

Advantages of scan line algorithms:
- Incremental calculation of geometry is very efficient.

Disadvantages of scan line:
- Local lighting model not as realistic.

Scan conversion of polygons is particularly popular, especially in computer animation, where many images must be produced and image generation time must be kept to a minimum [Crow, 1978]. Ray tracing has generated some very realistic still images, but generally has been impractical for animation. The only extensive ray traced animation made to date was done with the aid of special-purpose hardware [Kawaguchi, 1983; Nishimura *et. al.*, 1983].

The differences between many hidden surface algorithms depends on the techniques used to exploit coherence in the scene. Most attempts at exploiting coherence in ray tracing have concentrated on techniques to limit the number of ray-surface intersections that are to be tested. This can be done by hierarchically decomposing the scene into a tree of enclosing volumes [Clark 1976; Rubin and Whitted, 1980; Dadoun, Kirkpatrick and Walsh, 1982]. The ray-surface intersection calculation proceeds by testing the outermost enclosing volume first and searches the subvolumes only if the ray pierces that volume. Another approach is to form a cellular decomposition of the scene, keeping track of which surfaces are within or border a given cell. If the ray is assumed to be in a particular cell then only the surfaces inside that cell need be tested for intersections [see for example Jones, 1971]. If no intersections are found the ray passes through that cell, enters a neighboring cell and the search continues. Another technique, applicable if rays are only traced to one level, is to enclose each surface with a bounding box in image space [Roth, 1982]. As the image is scanned, surfaces become active or inactive depending on whether the current raster location is within the surface's bounding box.

A different type of coherence results from the observation that in many scenes, groups of rays follow virtually the same path from the eye to the light source and thus can be bundled into coherent beams of light (see Figure 1) [Hanrahan and Heckbert, 1984]. This observation can be exploited by attempting to trace a beam of rays in parallel rather than an individual ray. Exploiting this coherence is advantageous since it reduces the number of intersection. Also, once such coherence is identified it allows incremental techniques to be used for drawing a homogeneous region (a region over which the ray tree is constant). This would increase the speed of the rendering algorithm, especially at high resolutions. Finally, it is just this lack of coherence which causes many of the aliasing artifacts in ray traced images. It is possible to use the coherence to antialias textures and shading calculations within a homogeneous region and potentially, by identifying the boundaries between regions, to antialias their edges.

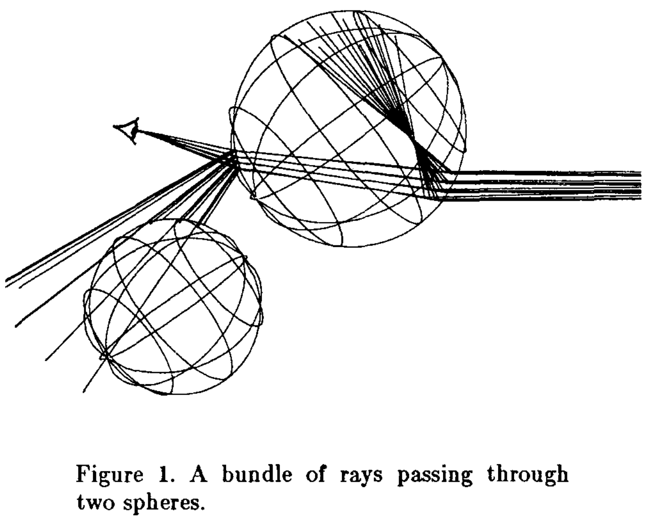
Figure 1. A bundle of rays passing through two spheres.

We propose an algorithm for tracing beams through scenes described by planar polygonal models. Beam tracing polygons is much simpler because of the large body of knowledge regarding both object space hidden surface calculations and image space display algorithms. Also, unlike the general case of a beam reflecting from a curved surface, beams formed at planar boundaries can be approximated by pyramidal cones. The algorithm we describe is similar in principle to a technique developed by Dadoun, Kirkpatrick and Walsh [1982] to trace sound beams from audio sources to a receiver. They noted that this problem is equivalent to the hidden surface problem and proposed computationally efficient algorithms for performing rapid hidden surface removal in static scenes. Our algorithm differs in that it is patterned closely after the classic ray trace and creates output that can directly drive image space rendering programs.

2. Beam Tracing

The beam tracer is a recursive polygon hidden surface algorithm. The hidden surface algorithm is designed to find all visible polygons within an arbitrary two dimensional region. The procedure begins with the viewing pyramid as the initial beam. The beam tracer builds an intermediate data structure, the *beam tree*, which is very similar to the ray tree [Whitted, 1980]. Like the ray tree whose links represent rays of light and whose nodes represent the surfaces those rays intersect, the beam tree has links which represent cones of light and nodes which represent the surfaces intersected by those cones. But unlike a link in a ray tree which always terminates on a single surface, the beam link may intersect many surfaces. Each node under the beam represents a visible surface element as seen along the beam axis. This is illustrated abstractly in figure 2 and a simple example is shown in figure 3. The beam tree is computed in object space and then passed to a polygon renderer for scan conversion to form the final shaded image.

2.1. Object Space Beam Tracing

The beam tree could be formed using any of several object space hidden surface algorithms. The pictures generated for this paper used an algorithm modeled along the lines of the classic ray tracing program. We now outline the procedure:

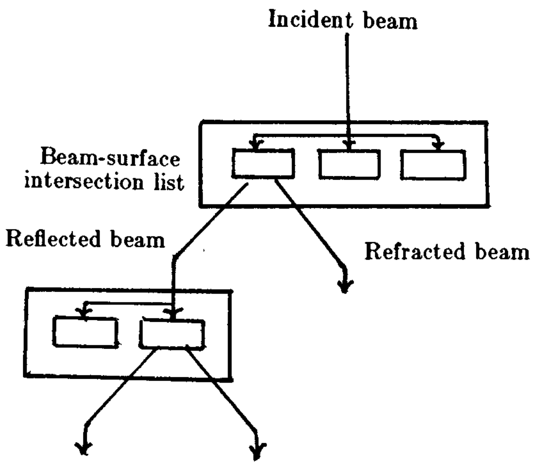

Figure 2. A schematic representation of the beam tree. Notice that the incident beam is fragmented into several pieces each of which may give rise to a reflected and refracted beam.

```
function Beam-Trace( Beam : Polygon; Ctm : Matrix ) : PolygonList;
begin
 {Transform scene into the beam coordinate system
  using the current transformation matrix (Ctm)}
 {Priority sort all polygons in the scene (ScenePolygonList)}
 For each Polygon in the ScenePolygonList do begin
   P = Intersection( Beam, Polygon );
   if P ≠ nil then begin
     if RecurseFurthur( Depth, P, ... ) then begin
       if reflective(Polygon) then begin
         NewCtm = Ctm * ReflectionMatrix( Polygon );
         P.ReflectiveTree = Beam-Trace( P, NewCtm );
       end
       if refractive(Polygon) then begin
         NewCtm = Ctm * RefractionMatrix( Polygon );
         P.RefractiveTree = Beam-Trace( P, NewCtm );
       end
   end
   {Add P to the Beam polygon intersection list (FragmentList)}
   Beam = Difference( Beam, Polygon );
 end
 Beam-Trace = FragmentList;
end
```

Most ray tracers perform all their calculations in the world coordinate system. The beam tracer performs all calculations in a transformed coordinate system, initially the viewing coordinate system, called the beam coordinate system. In the beam coordinate system, beams are defined as the volume swept out as a two-dimensional polygon in the x-y plane is translated along the z-axis. Since the transformation into world space may contain perspective, the most general beam is a polygonal cone in world space. In a ray tracer, the ray is redirected after a reflection or refraction whereas in the beam tracer the scene is transformed into the beam coordinate system.

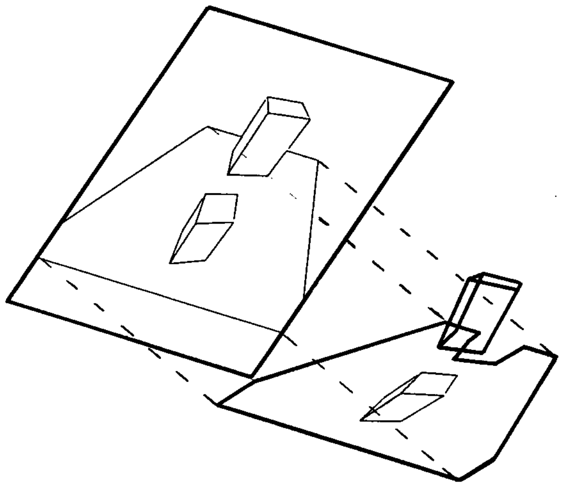

Figure 3. A beam tree corresponding to a cube on a mirrored surface. The bold lines indicate the beam fragments at that level in the tree.

This technique is analogous to forming the virtual image of an optical system.

The closest beam-surface intersection is determined by searching a depth-sorted list of polygons using two dimensional set operators. The depth ordered list is formed by priority sorting the polygons [Newell, Newell, and Sancha, 1972]. In our implementation, intersecting polygons and cyclic dependencies are not allowed although this is not a theoretical limitation of the approach. Such sorting is not required during a ray trace but is characteristic of object space hidden surface algorithms [Sutherland, Sproull, and Schumacker, 1974] and implies that the worst case running time could be $O(n^2)$, which is worse than a ray trace which is $O(n)$. Since we must depth-sort the polygons after every beam intersection, algorithms that preprocess the scene so that priority ordering can be quickly determined from any viewpoint could be used here [Sutherland, Sproull, and Schumacker, 1972; Fuchs, Kedem, and Naylor, 1980].

To find the first visible polygon, we intersect the beam with the first polygon in the list. If the result is *nil* then the polygon is outside the beam, otherwise it is visible and is added to the list of visible surface elements within this beam. To ensure that no other polygon is classified as visible within the area of this visible polygon, we subtract it from the beam before continuing through the depth ordered list. The set operators used in this algorithm must be able to handle concave polygons containing holes. Different methods for performing polygonal spatial set operations (union, intersection, and difference) are discussed in [Eastman and Yessios, 1972; Weiler and Atherton, 1977].

To simulate reflection and refraction we call the beam tracer recursively by generating new beams whose cross-sections are the intersection polygon. The transformations for reflection and refraction are discussed further in the next section and the appendix.

Recursion of the beam tracer can be terminated by several criteria:

(1) Maximum tree depth: 5 levels is common.

(2) Threshold of insignificance: determine if the intensity contributed by this branch of the tree will make a perceptible difference (called "adaptive tree depth control" in [Hall and Greenberg, 1983]).

(3) Polygon size: terminate when polygon area is below some threshold, such as one pixel.

2.2. Reflection and Refraction Transformations

Reflection in a plane, which maps each point to its mirror image, is a linear transformation, and can be represented by a 4x4 homogeneous matrix. Refraction by a plane, however, is not a linear transformation in general. Figures 4 and 5, which were made with a standard ray tracer, show the distortion of an underwater checkerboard viewed from above, and an above-water checkerboard viewed from underwater, respectively. The second shows the effect of the *critical angle* which occurs when a ray is refracted from the denser material into the sparser one. Rays incident at the critical angle are refracted parallel to the surface (toward the horizon in our "fish-eye" view). Outside the circle, when the incident angle is greater than the critical angle, there is no refraction, and one has *total internal reflection*. (Humans do not see this phenomenon when swimming because our eyes are not adapted to focus underwater [Walker, 1984]).

Since refraction bends lines, it cannot always be expressed as a linear transformation. There are two situations under which it is linear, however. For orthographic projections the incident angle is constant, and refraction is equivalent to a skew or shear. The other situation is for rays at near-perpendicular incidence, known as *paraxial rays* in geometrical optics. The latter corresponds to the centers of figures 4 and 5, where lines are approximately linear. Derivations of the matrix formulas for these transformations are given in the appendix.

Since beam tracing, as outlined here, is limited to linear transformations, we must choose one of these approximations in order to simulate refraction. The consequences are that beam traced perspective pictures exhibiting refraction will not be optically correct. There will be no critical angle, and lines will never become bent. Beam traced approximations to figures 4 and 5 would look like normal checkerboards. The error of the approximation is highest for refraction from dense materials to sparse ones, but fortunately for beam tracing, humans are normally on the sparser side (e.g. in air, looking into water). This explains why refraction's curvature of lines is not widely known.

2.3. Image Space Rendering

Associated with each screen-space polygon is the tree of face fragments which are projected onto that polygon by the reflection and refraction transformations. Final intensities can be computed by scan converting all of the faces covering a given area in parallel, and applying the recursive intensity formula

$$I = c_d I_d + c_s I_s + c_r I_r + c_t I_t$$

to the tree of faces. This blends the diffuse, specular, reflected, and transmitted intensities according to the coefficients c_d, c_s, c_r, and c_t to compute a color for each pixel.

Faceted or Gouraud shading of beam traced scenes can be done using a modified painter's algorithm. This was the technique used to produce the beam traced pictures in this paper (figures 6-9). Polygons are drawn into a frame buffer one-by-one as the beam tree is traversed. For each face or vertex, the diffuse and specular intensities I_d and I_s are computed using the Phong lighting model [Newman and Sproull, 1979]. Since the polygons in the fragment lists of the beam tree are defined in the beam coordinate system, they must be transformed back

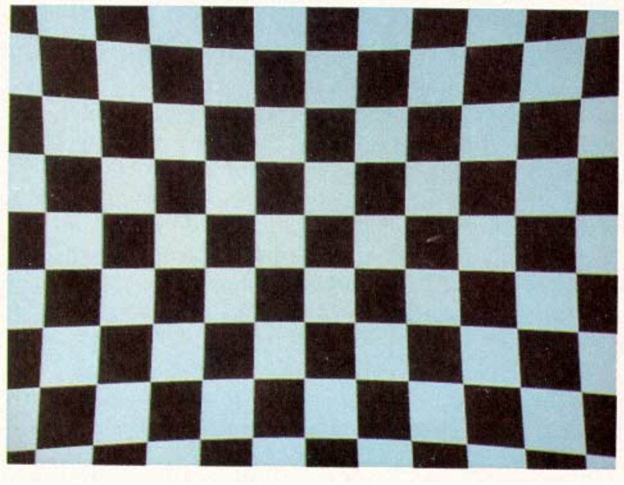

Figure 4. View of an underwater checkerboard from air.

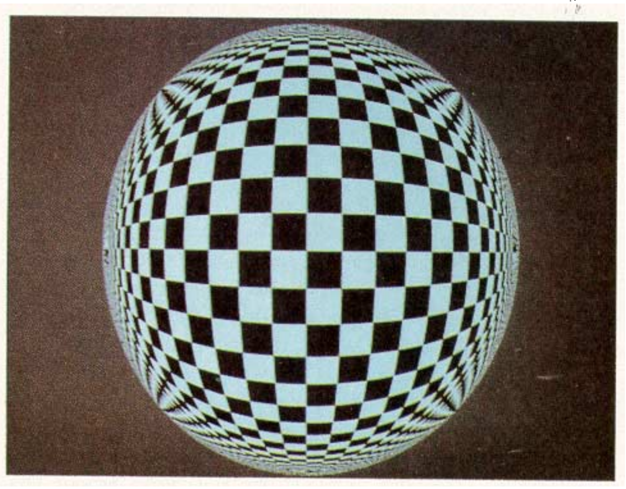

Figure 5. View of an above-water checkerboard from underwater.

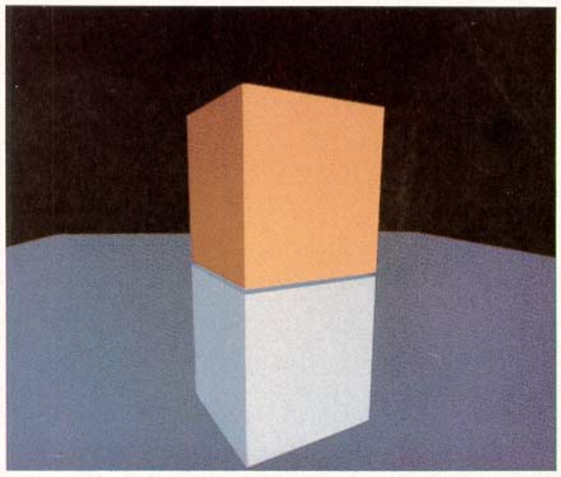
Figure 6. A diffusely shaded cube resting on a mirror.

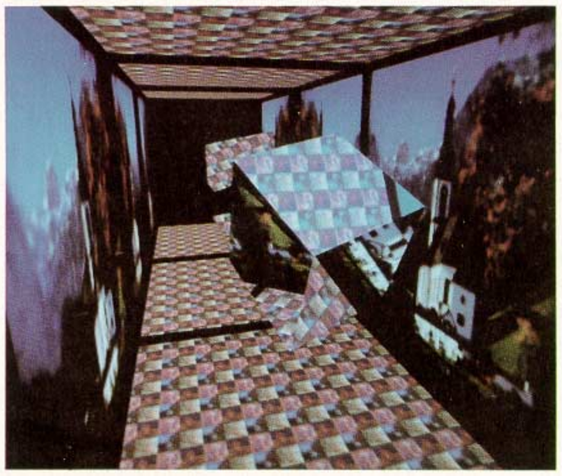
Figure 8. A reflective cube in the interior of a texture mapped, reflective cube.

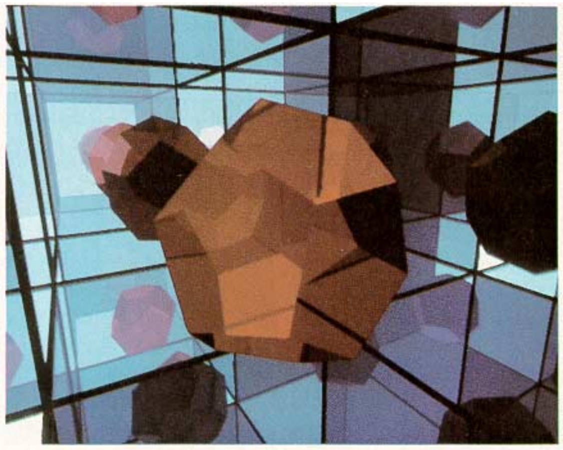
Figure 7. A translucent dodecahedra within a cube of mirrors.

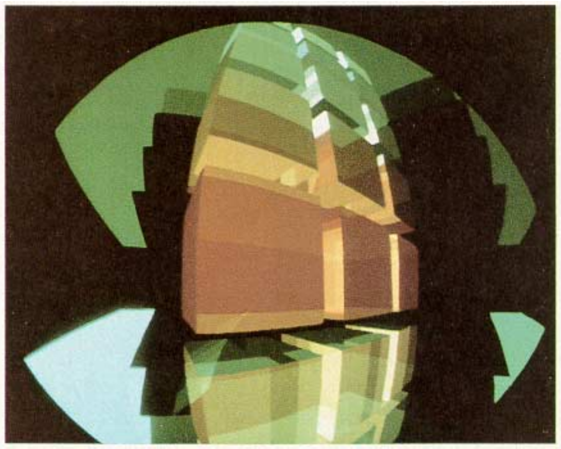
Figure 9. An example of post-processing a beam tree to simulate an Omnimax projection system.

to "world space" for all shading calculations; shading should not be done in a virtual or perspective space. If the coefficients c_d, c_s, c_r, c_t, and the material transparency are constant over each polygon, rendering can be done by simply adding the intensities into a frame buffer as each polygon is drawn. Since addition is commutative, the beam tree can be traversed in any order. This method requires a fairly robust polygon tiler, since the polygons are potentially concave with holes, and a single-pixel gap or overlap between polygons will result in an edge which is too dark or too light [Heckbert, 1983]. If the polygon tiler is limited to convex polygons, concavities and holes can be eliminated by polygon subdivision. Note that the pixel to pixel coherence allows antialiased texture mapping (figure 8).

3. Discussion

A useful measure of any ray traced image is average ray tree size. This statistic is similar to the notion of depth complexity used in analyzing polygon display algorithms. With the beam tree this is easily calculated since

$$\textit{average ray tree size} = \frac{\textit{total area (in screen space) of the beam tree polygons}}{\textit{screen area}}$$

The total time spent ray tracing an image is equal to the average ray tree size, multiplied by the time needed to determine the nearest ray-polygon intersection, times the resolution of the image. In contrast, the beam tracer is resolution independent and therefore the relative efficiency of beam tracing versus ray tracing increases linearly with resolution.

Beam tracing, however, is more complicated then ray tracing and may not always be worth the extra expense. The expected improvement depends more on the intrinsic coherence of the imaged scene rather than resolution. We can define coherence as

$$\textit{coherence} = \frac{\textit{average ray tree size}}{\textit{total beam tree size}}$$

This measure indicates to what degree the beam is fragmented–the more it is fragmented the lower the scene's coherence. If the coherence is very high then many rays are being traced in parallel and therefore a beam tracer will be faster than a ray tracer. In summary, beam tracing is most efficient when there are large homogeneous regions in the picture.

Figures 6 and 7 took 30 seconds and 5 minutes, respectively, to generate. We estimate a standard ray tracer would take 20-100 times longer.

One nice feature of the beam tracer is that it generates a reasonably compact object space representation of the image. Because it produces object space coordinates, precise line drawings like figure 3 can be made. These are difficult to make with a ray tracer [Roth, 1982]. The intermediate representation can also be further manipulated. For example, the same tree can be rendered at different resolutions or a series of images can be produced which differ in their coloring and lighting parameters. Since the output of the beam tracer is in object space, fisheye projections such as Omnimax can be made by beam tracing in perspective with a very wide camera angle and distorting the result (figure 9). Methods for mapping and subdividing lines for the Omnimax projection are given in [Max, 1983].

There are, however, some rather severe limitations to the technique as we've developed it here. We exploit coherence by assuming that reflections and refractions form virtual images within a polygonal window and that these virtual images can be formed by linear transformations. As we've shown, refraction under perspective is not a linear transformation thus the pictures are not physically correct. In practice it is very difficult for "non-experts" to detect the discrepancy in the pictures we've generated. This linearity assumption is also not correct for curved surfaces since their normals vary from point to point.

We now discuss several possibilities for future extensions of beam tracing.

3.1. Light Beam Tracing

In standard ray tracing, diffuse shading of surface points is done by aiming rays toward each point light source and determining if any objects are blocking the path of direct illumination. Blocked lights create shadows; unblocked lights contribute to the reflected intensity using Lambert's law. Since rays are aimed only at the lights, and not in all directions, this does not realistically model indirect illumination. It is also intellectually unappealing because it creates an asymmetry between the light source and the eye point, contrary to the laws of physics.

Diffuse shading and shadows for a beam-traced model can be computed as a pre-process to the polygon database. We propose a recursive extension of the Atherton-Weiler shadow algorithm [1978]. Beams are traced from each light source just as they are from the viewpoint ("light beam" tracing as opposed to "eye beam" tracing). The first order intersections are the directly illuminated surfaces, (not shadowed) and higher order intersections are illuminated surfaces resulting from the reflected and refracted light sources, an effect difficult to achieve with a standard ray tracer. The illuminated polygons thus formed can be built into the model database as "surface detail", that is, polygons which do not affect the shape of the objects, but only their shading. If the light sources are infinitely distant, each face will have a constant diffuse intensity, which can be compactly saved in the database on this pre-process pass, for use during the rendering pass.

This algorithm has several other advantages over diffuse shading during rendering. Light sources can be directional and have a polygonal cross-section. They need not be point sources. The depth of the eye beam tree can also be reduced, since light beam tracing propagates shading information through several bounces (one could say that the light beams meet the eye beams halfway). Finally, if the model and lights are stationary during an animation, light beam pre-processing need be done only once.

3.2. Antialiasing

In classical ray tracing, antialiasing is usually done by adaptive subdivision of pixels near large intensity changes or small objects [Whitted, 1980; Roth, 1982]. The method attempts to use heuristic criteria to probe the image frequently enough that small details will not be overlooked. Depending on the criteria, it will sometimes subdivide too little, resulting in aliasing, or too much, in which case processing time is wasted.

Before rendering, it is possible to subdivide the beam tree into non-overlapping polygonal regions and form an adjacency graph which indicates which regions are neighbors. Given this information, antialiasing edges is straightforward. Since the beam tracer resolves all hidden-surface questions, all that is needed is a polygon scan converter with a pixel integrator. Pixel integration can be done by sub-sampling or with analytic methods [Catmull, 1978]. This same information might be useful when light beam tracing. The symmetry between eye and light allows us to relate partially-covered pixels to partially-obscured lights: the former suggests antialiasing, the latter suggests soft shadows. Consequently, if a region adjacency graph is made during light beam tracing, this can assist in the creation of soft-edged shadows.

3.3. Rendering Options

There are many other interesting variations to rendering the beam tree. If, as mentioned in the previous section, the output is divided into non-overlapping regions and we are doing faceted shading, the recursive shading formula can be calculated once for the entire region and a single polygon rendered. At the other extreme, it might be worth modifying the polygon renderer so that it simultaneously tiles all the polygons in a region. This would allow the simulation of light scattering through translucent materials, since in this case the intensity is an exponential function of material thickness [Kay and Greenberg, 1979] which varies over the polygons. As we've mentioned, because of the additive nature of the shading formula, the polygons can be rendered in any order but the overall intensity is modulated depending on the shading coefficients and the depth in the tree. If, on the other hand, we always render the scene in back-to-front order, one can accommodate spatially varying reflection and refraction coefficients. For example, cut glass could be simulated by modulating these parameters with a texture map. Finally, the coherence in the beam tree often allows certain aspects of the shading calculation to be done once per region which allows a more complicated shading model to be used. For example, we have used constant values for the four intensity coefficients, but more realistic results could be obtained using Fresnel's equations [Longhurst, 1967].

4. Acknowledgements

We would like to thank to Kevin Hunter and Jules Bloomenthal for proofreading and Jane Nisselson for assistance with the writing.

5. Appendix: Reflection and Refraction Transformations

We derive the homogeneous 4x4 matrix form for the reflection and refraction transformations.

Figure 10 shows the geometry of an incident ray I hitting a plane and generating a reflected ray R and refracted (transmitted) ray T. The three rays and the surface normal N all lie in a plane. The index of refraction changes from η_1 to η_2 at the boundary. The angle of incidence is θ_1 and the angle of refraction is θ_2. We wish to find the transformations which map the real reflected and refracted points P_r and P_t to their virtual image P.

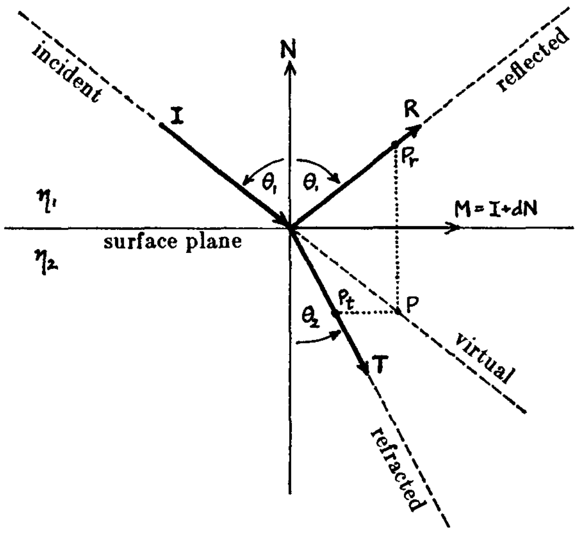

Figure 10. Geometry of reflection and refraction in the plane of incidence.

Notation:

$I =$ *incident ray direction,* $|I| = 1$

$N = (A\ B\ C\ 0)^T =$ *normal to plane*

$|N| = \sqrt{A^2+B^2+C^2} = 1$

$L = (A\ B\ C\ D) =$ *coefficients of plane equation:*

$LP = Ax+By+Cz+D = 0$

$P = (x\ y\ z\ 1)^T =$ *any point*

The direction of the reflected ray is given by:

$R = I\text{-}2(N{\cdot}I)N = I+2dN$ *(a unit vector)*

where: $d = \cos\theta_1 = -N{\cdot}I > 0$

The reflected point can be found by noting that LP_r gives the distance of a point P_r from the plane. The point formula has a form similar to the direction formula:

$$P = P_r\text{-}2(LP_r)N = P_r\text{-}2NLP_r = \mathbf{M}_rP_r$$

where $\mathbf{M}_r$ is the homogeneous 4x4 matrix for the reflection transformation:

$$\mathbf{M}_r = \mathbf{I}\text{-}2NL = \begin{pmatrix} 1\text{-}2A^2 & -2AB & -2AC & -2AD \\ -2AB & 1\text{-}2B^2 & -2BC & -2BD \\ -2AC & -2BC & 1\text{-}2C^2 & -2CD \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

and $\mathbf{I}$ is a 4x4 identity matrix.

Snell's law relates the incident and refracted angles:

$$\eta_1\sin\theta_1 = \eta_2\sin\theta_2$$

The direction of the refracted ray is:

$T = \eta I\text{-}(c\text{-}\eta d)N$ *(a unit vector)*

where: $\eta = \eta_1/\eta_2 =$ *relative index of refraction*

and $c = \cos\theta_2 = \sqrt{1\text{-}\eta^2(1\text{-}d^2)}$

There is no refracted ray (total internal reflection) if $1\text{-}\eta^2(1\text{-}d^2) < 0$. Our formula for T is equivalent to, but simpler than, the one in [Whitted, 1980].

For orthographic projections, the incident direction I is independent of object position, and refraction is a skew transformation parallel to the plane which maps a point P_t as follows:

$P = P_t+(\tan\theta_1\text{-}\tan\theta_2)\hat{M}LP_t$

where: $M = N\times(I\times N) = I\text{-}(N{\cdot}I)N = I+dN,$

$M =$ *vector tangent to plane,* $|M| = \sin\theta_1$

and $P = \mathbf{M}_tP_t$, where $\mathbf{M}_t$ is a 4x4 matrix:

$\mathbf{M}_t = \mathbf{I}+\alpha(I+dN)L$

$$\textit{where: } \alpha = \frac{\tan\theta_1\text{-}\tan\theta_2}{\sin\theta_1} = \sec\theta_1\text{-}\eta\sec\theta_2 = \frac{1}{d} - \frac{\eta}{c}$$

If viewing along the Z axis, then $I = (0\ 0\ 1\ 0)^T$, $d = -C$, and

$$\mathbf{M}_t = \begin{pmatrix} 1-\alpha A^2C & -\alpha ABC & -\alpha AC^2 & -\alpha ACD \\ -\alpha ABC & 1-\alpha B^2C & -\alpha BC^2 & -\alpha BCD \\ \alpha A(1-C^2) & \alpha B(1-C^2) & 1+\alpha C(1-C^2) & \alpha D(1-C^2) \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

This formula is exact for orthographic projections. If the eye is local, however, I and d vary from point to point on the surface, and there is no linear transformation from P_t to P.

Observing figure 11, however, we see that rays with small incidence angles (paraxial rays) produce virtual refracted rays which nearly come to a focus. Thus, for paraxial rays,

$$\frac{D_1}{D_2} = \frac{\tan\theta_2}{\tan\theta_1} = \frac{\tan\phi_2}{\tan\phi_1} = constant$$

If we take the constant to be η_1/η_2, then

$$\eta_1\tan\theta_1 = \eta_2\tan\theta_2$$

We call this the *tangent law.* For paraxial rays, $\sin\theta \approx \tan\theta \approx \theta$, so Snell's law is in agreement with the tangent law. A graphical comparison of the two laws is shown in figure 12.

The virtual focus can be interpreted as follows: When looking across a boundary with relative index of refraction η, objects appear to be at η times their actual distance [Feynman, 1963]. Recall that light travels slower in denser materials by precisely this factor η. Within the paraxial approximation, then, refraction is equivalent to a scaling transformation perpendicular to the plane:

$$P = P_t + (\eta-1)(LP_t)N = \mathbf{M}_t P_t$$

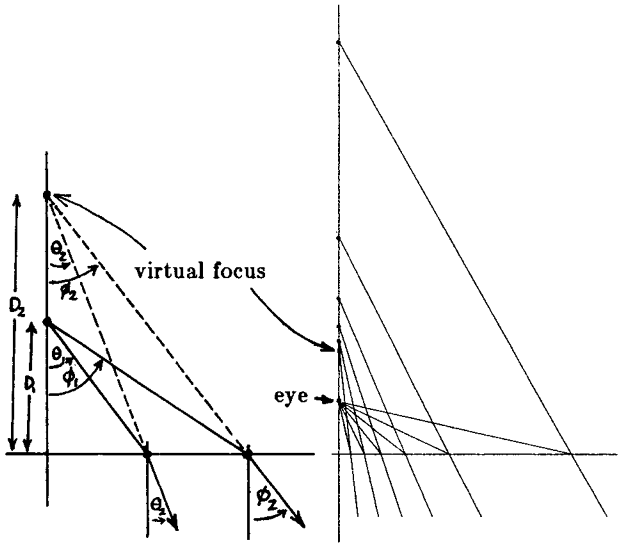

Figure 11. Refracted paraxial rays come to a virtual focus in the first medium at a distance $D_2 = \frac{1}{\eta}D_1$ from the plane. Left diagram illustrates tangent law, right illustrates Snell's law.

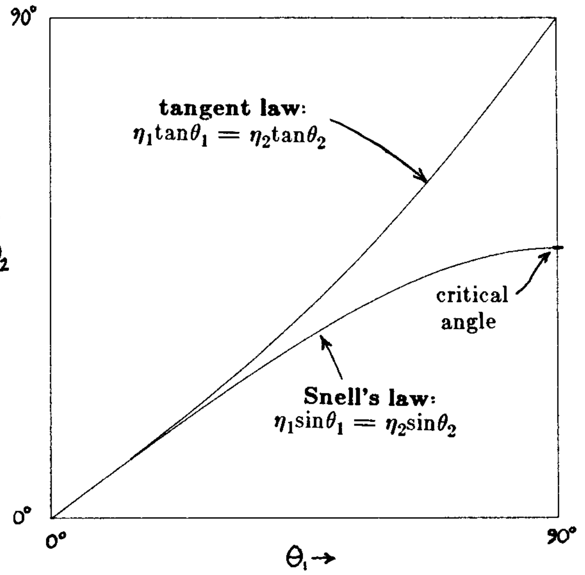

Figure 12. Refraction angle as a function of incidence angle using Snell's law and tangent law. For this graph, $\eta_2/\eta_1 = 1.33$.

$$\mathbf{M}_t = \mathrm{I} + \lambda NL = \begin{pmatrix} 1+\lambda A^2 & \lambda AB & \lambda AC & \lambda AD \\ \lambda AB & 1+\lambda B^2 & \lambda BC & \lambda BD \\ \lambda AC & \lambda BC & 1+\lambda C^2 & \lambda CD \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

where: $\lambda = \eta - 1$

Note the similarity between this and the reflection formula. Reflection is simply paraxial refraction with $\eta = -1$.

6. References

Atherton, Peter K., Kevin Weiler, and Donald Greenberg, "Polygon Shadow Generation." *Computer Graphics (SIGGRAPH '78 Proceedings),* vol. 12, no. 3, Aug. 1978, pp. 275-281.

Catmull, Edwin, "A Hidden-Surface Algorithm with Anti-Aliasing." *Computer Graphics (SIGGRAPH '78 Proceedings),* vol. 12, no. 3, Aug. 1978, pp. 6-11.

Clark, James, "Hierarchical Geometric Models for Visible Surface Algorithms." *C.A.C.M.* vol. 19, no. 10, 1976, pp. 547-554.

Crow, Franklin C., "Shaded Computer Graphics in the Entertainment industry." *Computer,* vol. 11, no. 3, March 1978, p. 11.

Dadoun, Norm, David G. Kirkpatrick, and John P. Walsh, "Hierarchical Approaches to Hidden Surface Intersection Testing." *Proceedings of Graphics Interface '82,* May 1982, pp. 49-56.

Eastman, C. M., and C. I. Yessios, "An Efficient Algorithm for Finding the Union, Intersection and Differences of Spatial Domains." Technical Report 31, Institute of

Physical Planning, Carnegie-Mellon University, Sept. 1972.

Feynman, Richard P., Robert B. Leighton, and Matthew Sands, *The Feynman Lectures on Physics.* Addison-Wesley, Reading, Mass., 1963, vol. I, pp. 27-3, 27-4.

Fuchs, Henry, Zvi M. Kedem, and Bruce F. Naylor, "On Visible Surface Generation by A Priori Tree Structures." *Computer Graphics (SIGGRAPH '80 Proceedings),* vol. 14, no. 3, July 1980, pp. 124-133.

Hall, Roy A., and Donald P. Greenberg, "A Testbed for Realistic Image Synthesis." *IEEE Computer Graphics and Applications,* vol. 3, no. 8, Nov. 1983, pp. 10-20.

Hanrahan, Pat, and Paul S. Heckbert, "Introduction to Beam Tracing." *Proc. Intl. Conf. on Engineering and Computer Graphics,* Beijing, China, Aug. 1984.

Heckbert, Paul, *PMAT and POLY User's Manual.* New York Inst. of Tech. internal document, Feb. 1983.

Jones, C. B., "A New Approach to the 'Hidden Line' Problem." *The Computer Journal,* vol. 14, no. 3, Aug. 1971, pp. 232-237.

Kawaguchi, Yoichiro, "Growth: Mysterious Galaxy." *SIGGRAPH '83 Film & Video Shows,* p. 5.

Kay, Douglas S., and Donald Greenberg, "Transparency for Computer Synthesized Images." *Computer Graphics (SIGGRAPH '79 Proceedings),* vol. 13, no. 2, Aug. 1979, pp. 158-164.

Longhurst, R. S., *Geometrical and Physical Optics.* Longman, London, 1967.

Max, Nelson, "Computer Graphics Distortion for IMAX and OMNIMAX Projection." *Nicograph '83 Proceedings,* Dec. 1983, pp. 137-159.

Newell, M. E., R. G. Newell, and T. L. Sancha, "A New Approach to the Shaded Picture Problem." *Proc. ACM Nat. Conf.,* 1972, p. 443.

Newman, William M., and Robert F. Sproull, *Principles of Interactive Computer Graphics, 2nd ed.* McGraw-Hill, New York, 1979.

Nishimura, Hitoshi, Hiroshi Ohno, Toru Kawata, Isao Shirakawa, and Koichi Omura, "Links-1: A Parallel Pipelined Multimicrocomputer System for Image Creation." *IEEE 1983 Conf. Proc. of the 10th Annual Intl. Symp. on Computer Architecture.*

Roth, Scott D., "Ray Casting for Modeling Solids." *Computer Graphics and Image Processing,* vol. 18, no. 2, Feb. 1982, pp. 109-144.

Rubin, S.W., and Turner Whitted, "A 3-dimensional Representation for Fast Rendering of Complex Scenes." *Computer Graphics (SIGGRAPH '80 Proceedings),* vol. 14, no. 3, July 1980, pp. 110-116.

Sutherland, Ivan E., Robert F. Sproull, and Robert A. Schumacker, "A Characterization of Ten Hidden-Surface Algorithms." *Computing Surveys,* vol. 6, no. 1, March 1974, p. 1.

Walker, Jearl, "The Amateur Scientist: What is a fish's view of a fisherman and the fly he has cast on the water?" *Scientific American,* vol. 250, no. 3, March 1984, pp. 138-143.

Walsh, John P., and Norm Dadoun, "What Are We Waiting for? The Development of Godot, II." presented at the 103rd meeting of the Acoustical Society of America, Chicago, April 1982.

Weiler, Kevin, and Peter Atherton, "Hidden Surface Removal Using Polygon Area Sorting." *Computer Graphics (SIGGRAPH '77 Proceedings),* vol. 11, no. 2, Summer 1977, pp. 214-222.

Whitted, Turner, "An Improved Illumination Model for Shaded Display." *C.A.C.M.* vol. 23, no. 6, June 1980, pp. 343-349.