Visible Surface Ray-Tracing of Stereoscopic Images

Stephen J. Adelson and Larry F. Hodges
Graphics, Visualization and Usability Center
College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332
hodges@cc.gatech.edu *steve1@cc.gatech.edu*

Abstract

Ray-tracing is a well-known method for producing realistic images. If we wish to view a ray-traced image stereoscopically, we must create two distinct views of the image: a left-eye view and a right-eye view. The most straight-forward way to do this is to ray-trace both views, doubling the required work for a single perspective image. We have developed a reprojection algorithm that produces stereoscopic images efficiently with little degradation in image quality. In this paper, we derive the necessary stereoscopic separation technique required to transform a ray-traced left-eye view into an inferred right-eye view. Minor changes in the technique needed to expand from single to multiple rays per pixel are noted. Finally, results from evaluating several random scenes are given, which indicate that the second view in a stereo image can be computed with less than 16 percent of the effort of computing the first.

INTRODUCTION

Ray-tracing, while widely-used to produce realistic scenes, is notorious for the amount of computation required. Even if we ignore reflection and transparency, the algorithm is computationally intensive. If we desire to create a stereoscopic ray-traced image, we must have two slightly different perspective views of the same scene, potentially doubling the required work. Therefore, any savings which can be derived for stereoscopic ray-traced images would be useful. Badt has described a speedup technique for animation of ray-traced scenes.[1] As will be shown, we can think of the two views of a stereoscopic image as two animation frames where the second viewing position differs from the first position only in the horizontal axis. Given this model, the Badt technique will be refined in this paper for use with stereoscopic images and achieve large computational savings in creating the second view of a ray-traced stereo pair. Some preliminary work has been performed previously in this area [4], but only in implementing Badt's algorithm for stereoscopic views, not using the modifications implied by stereo pair creation.

PERFORMANCE MEASUREMENTS

Our purpose is to develop an algorithm that simultaneously operates on the two different images of a scene by extending a single-image algorithm. Using the single-image algorithm, if we can complete an operation on a scene in time, *t*, then the upper bound on completing that operation simultaneously on two slightly different views is approximately *2t*. The lower bound is *t* (i.e., the algorithm operating on two images can be done as efficiently as one image). We have chosen to measure computational savings in the text of the paper by comparing the simultaneous algorithm with the time beyond that needed to compute a single view. If we view the time *t* to compute one eye view as our benchmark and the increase in time beyond *t* as the time needed to compute the second view, then the decrease in computing time for the second view would range from 0% (total computing time for the second view = *t*) to 100% (total computing time for the second view = *0*).

ELEMENTARY RAY-TRACING

In ray-tracing, a light ray is traced backward from the viewpoint through each pixel of the screen. If an object is struck by a ray, several rays are sent off from the intersection point: shadow rays toward the light sources to indicate shadows, a reflection ray at an equal but opposite angle relative to the plane perpendicular to the intersection normal, and a transparency ray through the object. An illustration of these rays appears in figure 1. The reflection and transparency rays may strike other objects and send out yet more rays, and from all these rays the correct color is synthesized. We will use a simple model of ray-tracing and assume that all objects are diffuse reflectors with specular highlights and that there is a single light source. This means that we ignore the reflection and transparency rays and deal only with the shadow ray. The color of a pixel can then be determined using only two rays: the initial intersection ray and one shadow ray.

STEREOSCOPIC VIEWS

A standard single view perspective projection is to project the three-dimensional scene onto the x-y plane with a center of projection at (0, 0, -d). Given a point P

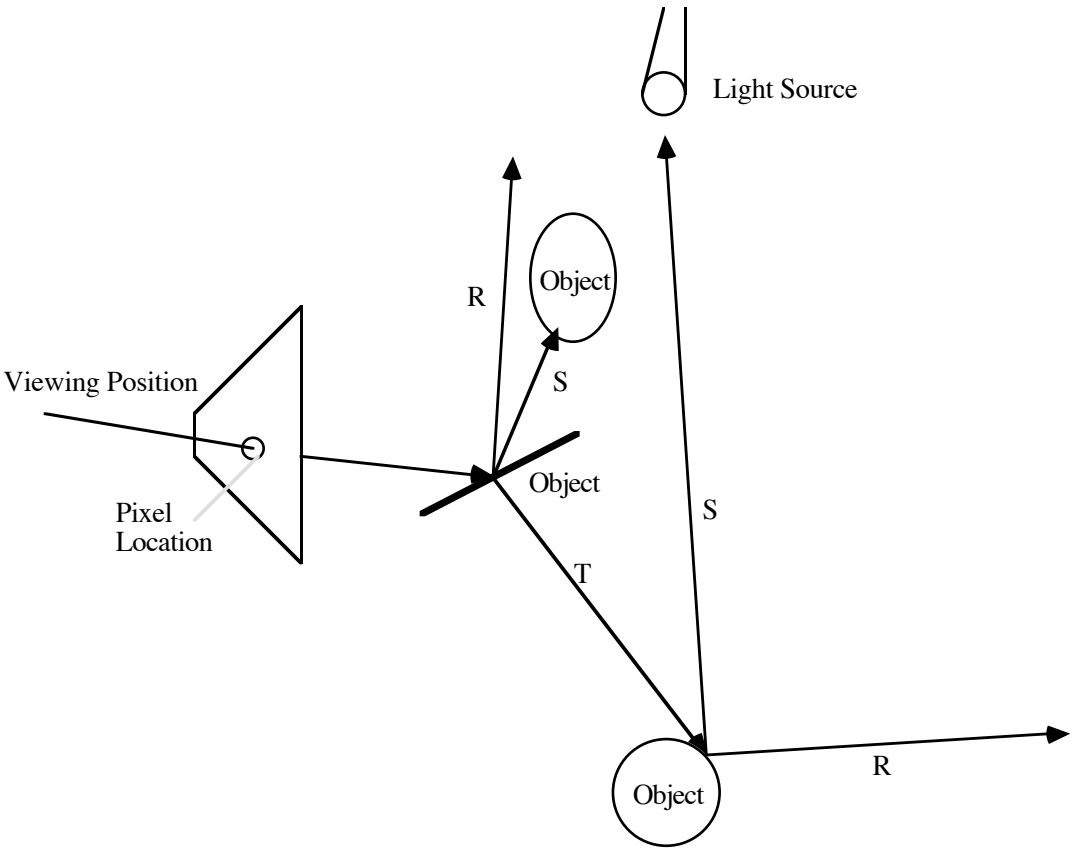

Figure 1. Ray-Tracing Rays, S (shadow), R (reflection), and T (transparency)

= (x_p,y_p,z_p), the projected point is (x_s, y_s) where

$$x_s = x_p d/(z_p+d) \quad (1)$$

and

$$y_s = y_p d/(z_p+d). \quad (2)$$

For stereoscopic images we require two differently projected views of the image. Each of the observer's eyes is presented with a view of a scene from different (usually horizontally displaced) positions. For our purposes we will derive these views from two different centers of projection, but other methods are possible.[1, 5, 6]

Assume a left-handed coordinate system with viewplane located at z=0 and two centers of projection: one for the left-eye view (LCoP) located at (-e/2, 0, -d) and another for the right-eye view (RCoP) located at (e/2, 0, -d) as shown in figure 2. A point $P = (x_p, y_p, z_p)$ projected to LCoP has projection plane coordinates $P_{sl} = (x_{sl}, y_{sl})$ where

$$x_{sl} = (x_p d - z_p e/2)/(d + z_p) \quad (3)$$

and

$$y_{sl} = y_p d/(d + z_p). \quad (4)$$

The same point projected to RCoP has projection plane coordinates $P_{sr} = (x_{sr}, y_{sr})$ where

$$x_{sr} = (x_p d + z_p e/2)/(d + z_p) \quad (5)$$

and

$$y_{sr} = y_p d/(d + z_p). \quad (6)$$

Note that the $y_{sl} = y_{sr}$ so that this value need be computed only once. Also note that the terms in x_{sr} and x_{sl} are identical and differ only by whether the terms in the numerator are added or subtracted. Assuming that we have already computed the left-eye view and have access to its component terms, only one addition and one multiplication are necessary to compute the right-eye view:

$$x_{sr} = x_{sl} + e(z_p / (d + z_p)). \quad (7)$$

In other words, the point will move horizontally between the views by a distance dependent on the depth **z** of the point to be projected, the distance **d** from the viewing position to the projection plane, and the distance **e** between the two viewing positions.

The illumination intensity of an object is dependent only upon the position of the object and light sources,

not the viewing position. However, specular highlights are dependent on the position of the viewer. If we use the familiar Phong highlighting term [3] then the highlight can be computed using the formula:

$$F(\mathbf{N}^{o}\mathbf{B})^{\alpha}$$

where

F is a constant for the object,

N is the unit normal vector at the intersection point (x_n, y_n, z_n),

B is the normalized sum **V+L**, where **V** is the vector to the viewing position (x_v, y_v, z_v) and **L** is the vector to the light source (x_l, y_l, z_l),

α is the Phong glossiness factor,and

^{o} is the dot product.

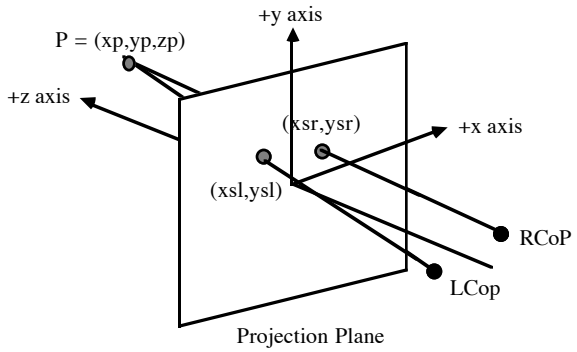

Figure 2. Stereoscopic Perspective Projection

The term (**N**^{o}**B**) is therefore

$$\{x_n(x_v+x_l) + y_n(y_v+y_l) + z_n(z_v+z_l)\} / \sqrt{[(x_v+x_l)^2 + (y_v+y_l)^2 + (z_v+z_l)^2]} \quad (8)$$

Let **V** be the vector from the intersection point to the left center of projection, then the Phong term for the right view is

$$\frac{\{ex_n + x_n(x_v+x_l) + y_n(y_v+y_l) + z_n(z_v+z_l)\}}{\sqrt{[(x_v+x_l)^2 + (y_v+y_l)^2 + (z_v+z_l)^2 + e(2(x_v+x_l)+e)]}} \quad (9)$$

If we represent the numerator of equation *(8)* as P_1, the denominator squared of the same equation as P_2, and $(x_v + x_l)$ as P3, then the Phong highlighting term for the left view is

$$P_1 / \sqrt{P_2} \quad (10)$$

and the right view term is

$$(ex_n + P_1) / \sqrt{[P_2 + 2eP_3 + e^2]}. \quad (11)$$

We can compute the right-eye specular term from the left-eye term with only seven additional operations.

PREVIOUS WORK

Badt describes a method for using *reprojection* to create a ray-traced scene in which the viewpoint has moved some small distance.[2] Reprojection involves moving the pixels in one view to their position in the second and cleaning up the image by recalculating only those pixels whose value is unknown or in question after the viewpoint has translated. Since a stereo pair is equivalent to two images in which the viewpoint moves a small horizontal distance, it is instructional to see the original algorithm before it is modified for stereoscopic images.

All pixels from the original frame, or view, are reprojected to their new position in any order. Simultaneously, a *hit-point count image* is generated. This image represents the number of "hits" or reprojections to a given pixel-position by adding one to the pixel-position for every reprojection to that pixel. Also, some pixels may receive no hits. Both the pixel image and the hit-count image are stored to disk.

The algorithm is concerned with four kinds of pixels, as represented by the hit-point count image: missed pixels, or those that received no hits; overlapped pixels, which received multiple hits; bad pixels, whose hit count is one and yet do not match their surroundings; and good pixels, whose hit count is one and do match. The first two types of pixels are easy to find in the hit-point count image and can be repaired by again ray-tracing, or *re-tracing*, the pixel. The second two are more difficult to distinguish. Previous papers have used the idea of a filter centered on a pixel to determine whether a pixel is good or bad. The hit count of all pixels under the filter are added together. If the combined value differs from the expected number of hits by a fixed value, then the pixel is considered bad and re-traced.

Badt shows that his algorithm could save a significant amount of effort when creating the second image; in his tests, almost 62% of the reprojected pixels were judged to be good and did not require re-tracing. Badt's technique can be used as is to create stereo pairs. A paper by Ezell and Hodges reported the results of such a endeavor; they showed that an average of 73% of the reprojected pixels were good.[4] However, when we use a modified version of this technique on stereoscopic images, the bad and overlapped pixel problems can be eliminated, as can the necessity for a filter, and the resulting performance will be dramatically improved.

ELIMINATING THE BAD AND OVERLAPPED PIXEL PROBLEMS

As stated previously, we must consider the image artifacts in a reprojected image caused by overlapped pixels, missed pixels, and bad pixels. It will now be shown that an algorithm can be derived for stereoscopic reprojection so that we only need concern ourselves with missed pixels, whose solution entails re-tracing the pixel in question.

If we calculate the pixels of the left image of a stereo pair first, the second image will have each corresponding pixel at a point equal or to the right of the original pixel, as per equation *(7)*. We also know from equations *(4)* and *(6)* that the pixel will not change its y value. We can therefore calculate both views of the image simultaneously, scan-line by scan-line.

Eliminating the Overlapped Pixel Problem

The overlapped pixel problem occurs because more than one pixel in the first image is mapped to the same pixel in the second image. However, since we are able to create this image a scan-line at a time, a small data structure can be created to hold the points reprojected from the left image to the right. In its simplest form, this structure need only have an entry for each pixel in a scan-line with each entry containing a Boolean flag representing a reprojection to the corresponding pixel and a value for its color. Using this data structure, we can overwrite reprojected pixel values, resulting in the correct value at the end of processing.

Consider the ray from the RCoP through a pixel PIX_1 at $(x_1, y, 0)$. The equation of the z position of this ray is

$$z = -d + T * d = d * (T - 1) \qquad (12)$$

where $T \geq 1$ means that the ray is in image space. An object which intersects this ray in image space will appear in PIX_1 in the right-eye view.

Consider another ray from the LCoP through PIX_2 at $(x_2, y, 0)$. This ray will intersect the previous ray when the x values of the rays coincide:

$$\begin{aligned} x_l = -e/2 + T(x_2 + e/2), \quad x_r &= e/2 + T(x_1 - e/2) \\ -e/2 + T(x_2 + e/2) &= e/2 + T(x_1 - e/2) \\ T(x_2 + e/2) - T(x_1 - e/2) &= e \\ T(x_2 - x_1 + e) &= e \\ T &= e / (x_2 - x_1 + e) \end{aligned} \qquad (13)$$

As illustrated in Figure 3, a point on the surface of an object will appear in PIX_2 in the left-eye view and in PIX_1 in the right-eye view if and only if there is an object surface at $T = e / (x_2 - x_1 + e)$, $T \geq 1$.

Suppose we have another pixel, PIX_3, at $(x_3, y, 0)$, $x_2 < x_3 \leq x_1$. A ray from the LCoP will intersect our ray from RCoP through PIX_1 at

$$T_0 = e / (x_3 - x_1 + e). \qquad (14)$$

If there is an object surface at this point, it will appear in PIX_3 in the left-eye view and PIX_1 in the right-eye view. We now are faced with the overlapped pixel problem. However, note that the denominator of equation *(8)*, $x_2 - x_1 + e$, is smaller than that of equation *(14)*, $x_3 - x_1 + e$. Hence, $T_0 < T$ and the object surface at T_0 is closer to the viewing plane than the surface at T.

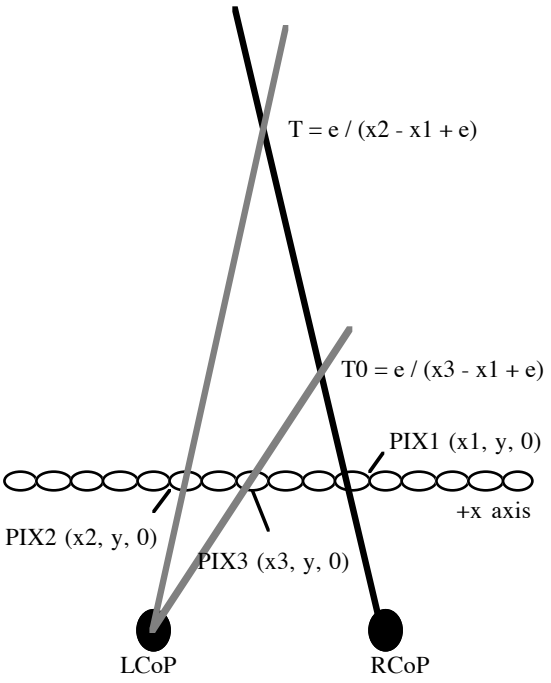

Figure 3. Solving the Overlapped Pixel Problem

Therefore, if we process pixels in a left to right fashion and overwrite the scan-line record as we go, we will always end up with the correct object reprojected to the pixels in the right-eye view.

When pixels are reprojected from the left-eye view to the right-eye view, they move according to equation *(7)*. This distance is dependent upon the z value of the image in the original pixel. It is possible, however, that the z values of two adjacent pixels in the left-eye view are such that the second will be reprojected further than the first, leaving a gap of 1 to e-1 pixels. Further, it is possible that other pixels may have reprojected into this gap; these gap-filling pixel values may or may not belong in the right-eye view. It is these questionable pixels that cause the bad pixel problem.

The bad pixel problem is illustrated in figures 4a and 4b. In figure 4a, we can see that there is indeed a true gap and the farther object A should be reprojected into this area. In figure 4b, however, there is an intervening object which would block A were the right-eye view fully ray-traced. This problem was previously solved by Badt by using the hit-point count image and a filter to test each pixel. It is hoped that the bad pixels will not totally fill the gap and surrounding missed pixels will disclose the location of improper pixels. Unfortunately, it is fairly easy to imagine a case where improper pixels are missed, or where they are caught but the filter causes many good pixels to be re-traced.

This problem can be solved for stereoscopic images by eliminating the projected values in intervening pixels when two adjacent pixels in the left-eye view, x and

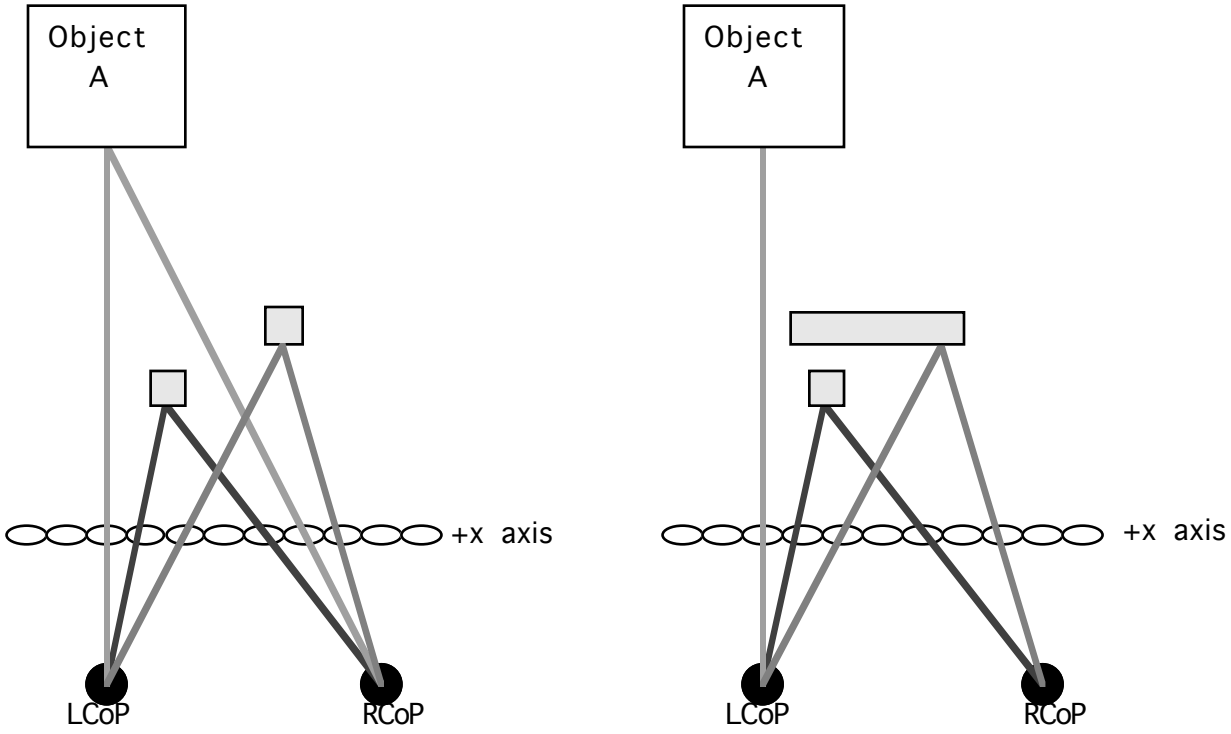

Figure 4a. Object A should appear in right-eye view.

x+1, are reprojected to the right-eye view such that **newx**(x+1) - **newx**(x) > 1, where **newx** is the reprojection equation *(1)*. Since pixels are often small relative to the objects they are displaying, the case of figure 4b seems much more likely than the case of 4a. Certainly, it is possible that this solution eliminates several good pixels, but the possibility of waste when displaying spheres or other smooth surfaces appears slight. We shall see later that the number of pixels reset in this process, on average, is small. In any case, it is absolutely impossible for bad pixels to exist in the right-eye view when using this method.

FINAL ALGORITHM

We now define the final algorithm for creating stereo pairs of ray-traced images. Assume we are creating both views in tandem or have access to pixel information of the left image in left to right fashion for each scan-line. For M by M resolution, we need a scan-line record line_rec of length M containing the following information for each pixel j, $0 \leq j < M$:

HIT : flag which is TRUE if there is a reprojection to pixel j, FALSE otherwise

COLOR : the color for the right -eye view of pixel j.

When we are creating images in tandem or we know the value of e in advance, COLOR can be computed as we go. Otherwise, we must also have access to information which will allow us to create a color for the right-view view; this would include light-point position, Phong