Exact Ray-Traced Animation Frames Generated by Reprojection

by

Stephen J. Adelson and Larry F. Hodges

GIT-GVU-93-30
July 1993

Graphics, Visualization & Usability Center

Georgia Institute of Technology
Atlanta GA 30332-0280

Exact Ray-Traced Animation Frames Generated by Reprojection

Stephen J. Adelson and Larry F. Hodges
Graphics, Visualization, and Usability Center
College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332-0280

Abstract

The majority of computer graphics images are generated as part of a sequence of animation frames. The usual approach for producing these images is to render each frame as if it were a single, isolated image. A technique is described which exploits spatial-temporal coherence between frames to speed up the generation of a ray-traced animation sequence. The concept is that the information gained when ray-tracing a particular frame in a sequence may be used to speed up the generation of adjacent frames, reprojecting samples from one frame to the next. This approach promises significant savings when ray-tracing animations which have moving and rotating viewpoints, objects and/or light sources. The algorithm is tested on an animation with many reflective surfaces, yet it still attains up to 96% savings over full ray-tracing in terms of the number of rays cast.

Key Words: Display algorithms, Ray-traced animation, Spatial Coherence, Temporal Coherence.

Introduction

Ray-tracing is an attractive method of generating images because of its simplicity and its ability to produce many realistic optical effects such as refraction, reflection, and shadowing. Ray-tracing's major drawback is its large computational cost, due mainly to calculating ray-object intersections [Wh80]. The computational time for complex images can run from several minutes to several hours on current workstations, depending on the machine and efficiency of algorithm implementation. General speedup techniques for ray-tracing usually concentrate on improving ray-object intersection algorithms by using bounding volumes or some type of hierarchical structuring of the object space. For ray-tracing of animation sequences, a further speedup approach is to take advantage of spatio-temporal coherence from one frame to the next. The idea is that each frame is usually very similar to the frames that immediately precede and succeed it. Therefore, the information gained when ray-tracing a particular frame in a sequence may be used to speed up the ray-tracing of these neighboring frames. In this paper an algorithm is presented which exploits spatio-temporal coherence between frames to significantly

decrease the rendering time of ray-traced animations.

This new method produces inferred ray-traced images of any scene which can be ray-traced using a point-sampled method. The images created by the algorithm are not approximated frames created from weighted averages of other frames (e.g. [FL90]), nor are they frames patched together from near-frame pixels values (e.g. [CC90]). The algorithm guarantees that a color seen in a subpixel would be returned by a ray passing somewhere through that subpixel, but not necessarily though the center. Additionally, the algorithm will never be slower than a traditional point-sampled ray-tracer, and usually much faster. In some test cases, the technique described here has generated animation frames with as few as 3% of the primary rays cast as compared with a standard renderer.

This algorithm will efficiently create frames of any view which can be ray-traced. While the savings of the technique will increase with the complexity of the rendered objects and a preponderance of diffuse objects, a large degree of savings can still be achieved with reflective and refractive objects. However, the animation requires that the ray-tracing method is point-sample oriented. "Pure" ray-tracing involves following lines through the scene, and these lines are considered infinitesimally thin. Since samples are reprojected based upon their three-dimensional intersection, these point samples must be available. Methods such as beam tracing [HH84] or cone tracing [Am84] are therefore forbidden in the first level of ray-tracing. Anti-aliasing can still be accomplished by a point-source oriented method such as adaptive super-sampling or distributed ray-tracing [CP84], and other methods of ray-tracing may be used for the higher-order rays involved with reflection and refraction.

Previous Work

Ray-Tracing and Animation

Many attempts have been made to speed the ray-tracing of individual frames (images), but most fall into only two categories. In the first, the algorithm seeks to decrease rendering time by reducing the number of objects for a given intersection test. Examples of this category include Rubin's nested bounding volumes for testing against increasing levels of detail [RW80], and space subdivision methods of both Glassner's Octrees [Gl84] and Fujimoto's ARTS system [Fu86]. In the second category, time is saved by providing more efficient intersection tests for each object. Often, this involves more efficient bounding volumes, such as the bounding volume work by Bouville [Bo85] and Kay's work with intersections with convex hulls [KK86].

Several algorithms also attempt to use frame-to-frame coherence to speed up animation. Most of these have used object space coherence for their speed-up method. Hubschman's object visibility tests process static convex objects based on when they will become visible [HZ82]. Glassner created 4-D bounding volumes extending through both time and space to reduce both bounding volume creation and duplicated intersection tests [Gl88]. Chapman designed a method which finds the intersection of

"continuous intersections" between rays and polygons through time, so that a pixel need only be retraced when the current continuous intersection ends [CC91].

Sequin stored the ray tree at each pixel so that image attributes could be changed in the image without having to cast any additional rays [SS89]. The method does not work when objects move, since the algorithm cannot determine when visible surfaces move. This method was extended for moving objects by Murakami [MH90], by subdividing space into voxels and storing a list of voxels for every ray in each ray tree. Moving objects are noted in the voxels, and all rays which pass through the changed voxels are retraced. However, memory requirements for storing the lists are very large and the computational load is heavy because of the many tree traversals. Further, ray trees are completely lost if only the first ray is moved.

Jevans stored in each voxel a tag to the original generating pixel which spawned the ray passing through the voxel [Je92]. Changed voxels caused the appropriate pixels to be re-traced. Jevans' approach is limited in that it only works for static cameras and a change in a ray of any level causes the entire pixel to be retraced. Also, his voxel tags represent large blocks of pixels rather than individual pixels because of the memory constraints.

A method was developed by the authors for using spatio-temporal coherence in animations where the only movement allowed is translations and rotations of the camera [AH93]. The technique had many limitations: the animations created were limited to "fly-overs" of the environment, complex clean-up techniques were required, and in some situations the algorithm would work very inefficiently, such as in the case of an object which moves rapidly across the screen. The work embodied by this paper is both an extension of that work and an alleviation of the previous problems.

Image Space Coherence and Stereoscopic Ray-Tracing

Badt in 1988 proposed a method of generating frames using image space coherence by reusing the pixels from one frame of a diffuse-object animation to determine many of the pixels in the next frame [Ba88]. His method led to gaps in the new frame and unoccluded pixels which should have been occluded. These problems were repaired to a large degree by clean-up routines, but not completely nor consistently. Ezell, realizing that a stereoscopic pair is the equivalent to two animation frames in which the objects remain motionless and the viewpoint moves some small distance, implemented Badt's method for the generation of stereoscopic ray-traced images [EH90]. However, this implementation suffered from the same limitations and image problems as Badt had in his animation frames.

The causes of the image problems in Badt's method have previously solved for stereo pairs [AH92a]. Using the geometry of stereoscopic viewing, the image problems were eliminated, as well as the tests needed by Badt and Ezell when attempting to fix their images. Further, a method was developed to allow full ray-tracing of stereoscopic images. It was demonstrated

that even though the higher levels of ray-tracing must be carried out individually for each frame (as they are eye-point dependent), the majority of the savings will remain [AH92b]. The algorithm presented in this paper is the result of taking the technique back to its original application, the generation of animation frames.

Reprojection and Volume Visualization

The algorithm in its developed form has some superficial resemblance to the certain volume visualization techniques. Specifically, this algorithm looks somewhat like the dividing cubes algorithm [CD88] and Gudmundsson's incremental projection method [GR90].

The dividing cubes algorithm uses voxelized data to create a set of surface points and normals which may then be projected to the viewing position. This is similar to the method used on ray-traced samples in the technique described in this paper. However, there are significant differences between the two:

Dividing cubes keeps no color information; instead a depth shading or a very simple color model is used. This is in keeping with the visualization application; realism is sacrificed for speed.

Dividing cubes renders one static surface. Multiple surfaces can be composited using a depth buffer, but the hundreds or thousands of surfaces in some animations is unreasonable for the technique.

There are no shadows, reflection, or true refraction (although there is translucency).

Finally, the voxelization of the data assumes that the viewing position is outside of the entire set of data; this is not a reasonable assumption for most animations.

Incremental projection is a method for increasing the speed of ray-cast volume visualization animations. Again, the viewpoint is assumed to be outside of the data set, and the set moves only by rotating along the y axis. By transforming the viewing positions about the axis, the technique attempts to identify newly hidden positions and cast rays in the image holes which appear due to changing occlusion.

The method works sufficiently well in its limited context, but it can easily generate incorrect frames and falters when more than one surface is rendered. As the goal of this paper is to develop exact frames of any image and for any movement, the incremental projections method is insufficient.

A Special Case of Animation: Stereoscopic Ray-Tracing

In stereoscopic rendering, there must be two centers of projection so that a different perspective view is produced for each eye. Figure 1 illustrates that the two viewing positions are separated parallel to the X axis by a distance **e** and that they are both a distance **d** from the projection plane [Ho92]. Stereoscopic ray-tracing is really a special case of animation, with the small exception that the viewing window remains fixed in three-space while in animation it is set at a certain distance and orientation to the viewpoint. Therefore certain issues in stereo are first examined before extending them to the full animation case. The

following is a summary of work which can be found in [AH92b].

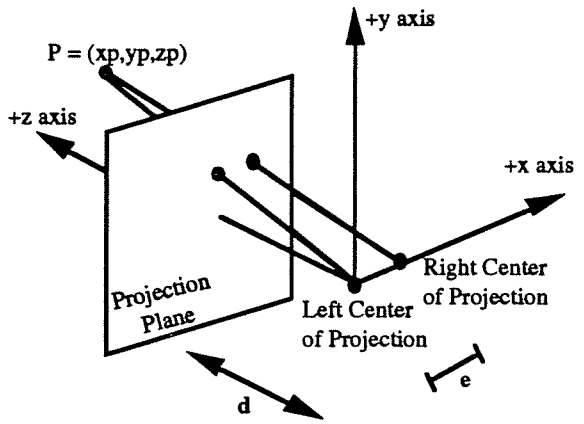

Figure 1. The Stereoscopic Viewing Geometry

Given the particulars of the stereoscopic viewing geometry, a ray intersection at (x_p, y_p, z_p) projecting to position (X, Y) in the left-eye view will appear on the same scan-line or *reproject* to the position (X + **e** - **e** * **d** / z_p, Y) in the right eye view, requiring only one addition and one multiplication (The term **e** * **d** is a precomputed constant). Reprojected (sub)pixels have the same diffuse color, shadowing, three-dimensional intersection, normal, and texture as the (sub)pixel from which the sample originated. While reflected components, refracted components, and highlights will differ, first level ray-tracing savings amount to 50-95% in most images.

When samples reproject in the stereoscopic geometry, one of four things can happen in the second view. First, a sampling position may have exactly one sample value reproject into it. These are called *good pixels* and their color and position values may be used directly. Positions which receive no reprojections are called *missed pixels* and must be fully ray-traced. *Overlapped pixels* are those in which two or more previous sample values reproject to the same sampling position (figure 2). It is important that we chose which of several projections is the correct one.

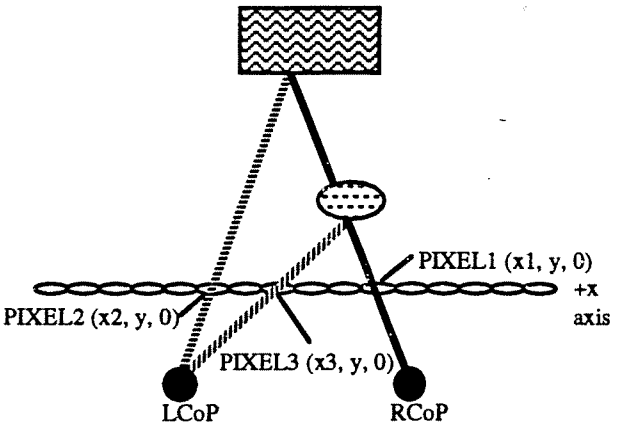

Figure 2. The Overlapped Pixel Problem. Both PIXEL2 and PIXEL3 in the left view will reproject to PIXEL1 in the right view. Which is the correct projection?

Last, samples which were consecutive on the original scan-line may reproject with a gap between them, allowing other samples to be viewed through the gap even though they should be obscured, as in figure 3. These last two problems are solved in the stereoscopic geometry by taking advantage of processing order and the relative positions of the viewpoints.

Animation has two-dimensional equivalents of the overlapped and bad pixel problem, must they are made more complex by the extra dimensional of sample movement and no a priori knowledge of consecutive viewpoints can be assumed.

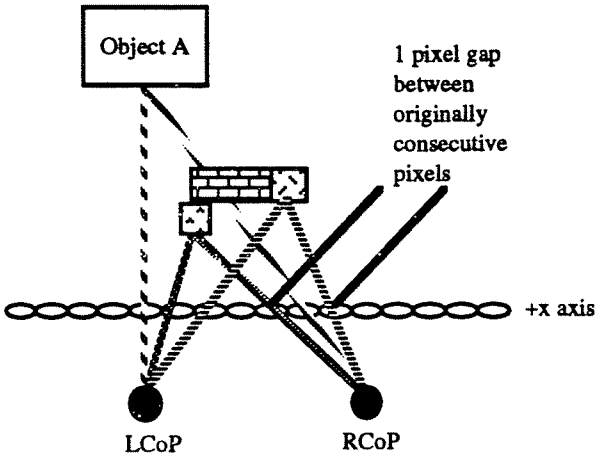

Figure 3. The Missed Pixel Problem. A gap opens in the right-eye view through which Object A can been seen. If the "brick" object exists, Object A should instead be obscured.

A General Animation Technique

As in stereoscopic ray-tracing, it is possible to make reprojection equations for a viewpoint movement in three-space based on the original projection location in the sampling grid. Unfortunately, these movements are far more computationally intensive and they do not take into account rotations, which must be provided separately. Instead, it is more efficient to preserve the three-dimensional intersections between rays and objects and project them to the new viewing position.

Technique Overview

The algorithm requires two frames to operate: the *base frame* which has been previously rendered (by full ray-tracing or by previous application of the algorithm itself), and the *inferred frame* which will be the new frame generated from reprojected samples of the base frame. It is best that the base frame and inferred frames be sequential; there are fewer differences between succeeding frames. The algorithm will work if the two are not adjacent, but the savings will diminish as the images in the frames diverge.

The initial base frame of the animation must be completely ray-traced. For each subpixel sampled in the base frame, the three-dimensional intersection point, normal vector, diffuse color, and id-tags of the intersected object and the shadowing object(s) are saved. The sampling may be adaptive, if desired.

Succeeding inferred frames will be created in four steps. The first step, reprojection, takes the samples from the base frame and projects them to the new view point, accounting for object movement or transformation as well as camera movement and rotation. The second step is *verification*, in which reprojected samples are tested to see if they are now obscured due to a moving object or point of view. Third, the positions are *enhanced* by reflection, refraction, and specular highlights, phenomena which generally cannot be guaranteed to have remained unchanged since the previous frame. Finally, the samples are filtered down to image resolution.

If the sampling is adaptive, the verification and enhancement steps should be performed only as needed in order to save time. It is possible that some reprojections will not be utilized in a given inferred frame. Nonetheless, they may considered valid samples until verification shows otherwise, and any unused data is written out with the verified data and reprojected in later frames.

New frames may be created in which the camera position has been rotated, translated, or zoomed; objects have moved or undergone a transformation which can be described in a function or matrix; and/or the light sources have moved or changed characteristics. In the description to follow, camera or object "movement" should be read to mean any of the above transformations. A single light source is discussed, but the same algorithm applies to multiple light sources when the appropriate steps are taken for each source.

New Frame - Reprojection Phase

The first step in generating a new frame is to reproject the positions from the previous frame. A file of object movements is provided which contains the incremental movement since the last frame of all objects which have moved. The object data file must be rewritten to take into account these movements as well. These two steps insure that previous samples will be moved in three space to the new object location, and that any rays which must be cast will strike the objects in their shifted position. Any affine transformation can thus be modeled by creating the appropriate four by four matrix **A** and "hanging" it off of the object datastructure so that samples of the objects can be properly transformed.

The camera in the new frame is given to be **d** units from the projection plane, which need not be the same **d** as the base frame. A four by four matrix **T** is created which rotates the axes to match the orthogonal world axes and translates the new viewing position to (0, 0, 0). The samples are read in one at a time. If it is a sample from a moving object (which can be determined by the existence of an **A** matrix associated with the sample's object), the projection is transformed to the new object specifications. All samples are passed through **T** to derive the three-dimensional point (x_t, y_t, z_t). This point is projected to the new viewing position by the equations

$$X = x_t * d / z_t,$$
$$Y = y_t * d / z_t$$

in the usual situation where the view plane is orthogonal to the direction of view and centered on the viewing direction vector. When this is not the case, some other projection may be used, but the transformation steps still hold. This process is illustrated in figure 4.

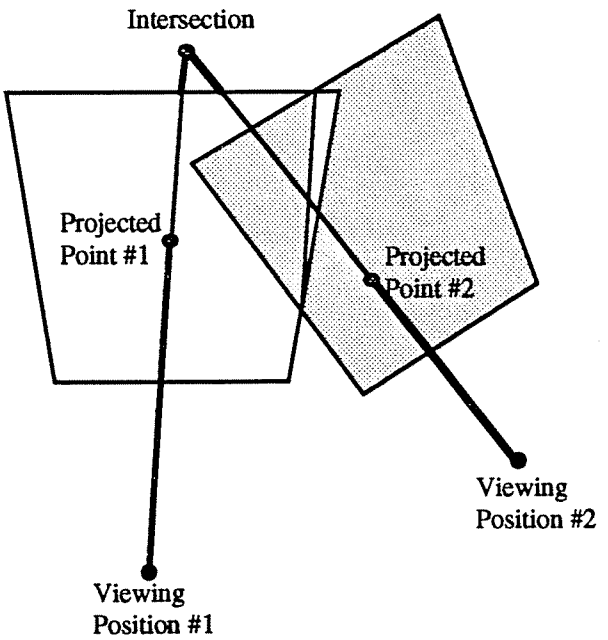

Figure 4. Reprojection. A three-dimensional intersection point which appeared in Projected Point #1 from Viewing Position #1 will *reproject* to Projected Point #2 in a frame whose eye point is Viewing Position #2.

Since samples may move in any direction because of a camera or object movement, a full-size (window sized) data structure to hold the reprojected samples is required. Each sampling cell will retain the world-coordinate intersection point, normal vector, color information, and id tags to the intersected object and the closest shadowing object. This data will be used in calculating specular highlights, reflection, refraction, checking shadows if needed, as well as being written to disk for use in creating the next frame.

The overlapped pixel problem, where more than one sample from the base frame projects to the same sampling grid position in the new frame, is an artifact of changing occlusion, both interobject and self-occlusion. In most cases, the value which is physically closest to the viewpoint will be the correct value. Therefore, in such a case the closest position will remain in the sampling grid, and it will be later verified.

Verification Phase

After reprojecting all values, there still may be a two-dimensional equivalent to the bad pixel problem, where holes in the image (caused by the samples of the visible objects covering a larger image area in the new frame) may allow positions which should be obscured to remain visible. Additionally, objects may have moved, obscuring positions which otherwise would remain visible. To preclude this, all bounding boxes (or other bounding structure) between the intersection and the new view point are checked for intersections. In most cases, only the intersected object will be hit. If another object is struck, however, the intersection with the intervening object must be calculated. Even the most efficient ray-tracer would need to check this minimum number of objects, and in most cases this algorithm will allow the object intersection routine to be eliminated since the intersection, normal, and often a portion of the color calculation can be reused.

In Figure 5 the verification step can be seen. The line from the intersection point to the viewing position does not intersect the bounding boxes of objects 1 and 2, and ignores them. It does hit the bounding box of object 3, checks the intersection with 3, discovers that in fact there is no intersection, and retains the original object information without further calculations.

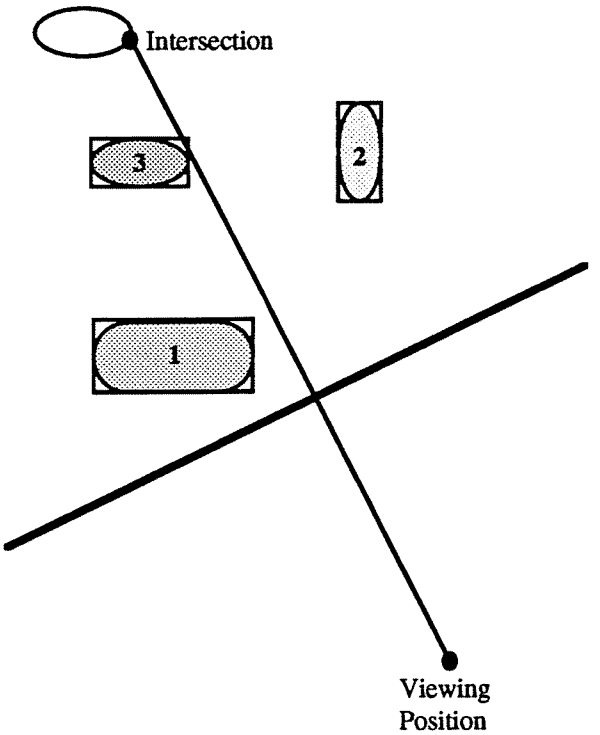

Figure 5. Verification Step. In most cases, there is no intersection with other objects.

Note that if the camera is set in a three-dimensional position, motionless objects need to

be verified only against the moving or transformed objects. Rotation of the camera or a change in the field of view will not change the occlusion of objects. Jevans claimed that many animations spend most of their time with a static camera [Je92]. If this is true, then the verification phase will be much faster than the best ray-tracer as the projected positions will need to be verified against a subset of the data.

If the camera does move, some portion of the image will be unique to the inferred frame: the part which was not visible in the base frame. These positions must be ray-traced in a normal fashion, but they obviously will not need to be verified.

If the light source or objects move, the first shadow ray of reprojected samples must also be verified. If the light source is static, positions which are self-shadowed and unmoving will remain self-shadowed, and need not be checked. Likewise a shadowed static position whose shadowing object has not moved will still be shadowed. Unshadowed static samples need check against only the moving objects, while a shadowed moving position first checks against its shadowing object and then against all other possible objects.

When the light source moves, all positions first check for self-shadowing or current shadowing object (if any), and then against all other objects.

Color Calculation

Under most circumstances at least part of the color calculation can be retained. Certainly the ambient term remains unchanged. The diffuse term will be retained if the light source is static, the object in question is static, and its shadowing information remain unchanged. It may also be kept if it was previously shadowed and remains so. Note that it is possible for an object to be found to have a different shadowing object in the verification phase and yet retain the same diffuse color. Likewise if a position stops being self-shadowed but becomes shadowed by another object or vice versa. If there is foreknowledge that the light sources, camera, and the object in question are all static, the specular term may also be retained from frame to frame. Otherwise, the normal and intersection are already calculated and can be utilized to determine the specular highlight.

Enhancement Phase

During the enhancement phase, all eye-point dependent phenomena are added. This includes the specular term (if not retained in the color calculation) and any reflective or refractive rays which may be cast by the object. The number of rays cast in the enhancement phase will be approximately equal to those cast during a full ray-tracing on the image.

Filtering

Once the samples are enhanced, the image can be generated by using any desired filter.

Summary

Rays are checked during verification phase only as needed; in the best case, the color of a sample is used directly from the base frame without any need for verification. In the worst, verification will reveal that the reprojected position has been

obscured in the inferred frame, but this is no worse than casting a ray through that position using an efficient ray-tracer.

Overall, there is some overhead involved with projecting the previous samples, but this is small as compared to the total time (it accounts for less than 2% of the total). Verification is a subset of the necessary steps of ray-tracing, and the algorithm will therefore usually be much faster than ray-tracing.

The Significance of Savings

This algorithm saves time when calculating the first level of ray-tracing, also known as ray-casting. The exact quantity of time saved will be the function of the number of samples which reproject into the inferred frame and survive verification, and the relative cost of casting rays. The cost is dependent on the implementation of the ray-tracer, but in general if C is the cost function, C(primary ray) $\leq$ C(shadow ray) $\leq$ C(reflection ray) $<$ C(refraction ray). In naive ray-tracers the four cost functions are nearly identical, but the possibility of complete internal reflection makes refractive rays somewhat more costly. In an ideal ray-tracer, the costs would also be nearly identical, though with a much lower expense than the naive one. Most real-world ray-tracers have a strictly increasing order of the four costs above.

It is possible to utilize some special techniques such as reflection mapping, refraction mapping, or environment mapping [BN76, Gr86] so that the cost of the higher level rays are significantly less than primary and shadow rays. This algorithm, used with such methods, would demonstrate higher savings than when rendering images with true reflective or refractive rays.

Performance Tests

To test the algorithm, three short animations were generated. The first had a length of 20 seconds (600 frames) and consisted of 755 polygons and 41 quadric surfaces with a single light source. Of these surfaces, 578 of the polygons and all 41 quadrics were partially reflective; additionally, four of the quadrics were also partially refractive. The animation was rendered with one light source at 640 by 480 resolution and one ray per pixel. Camera movement included rotations about all three axes.

The second animation used a single overhead view of the above scene with a spinning four-sided reflective object, for a total of 759 polygons and 41 quadrics. The camera was static throughout the entire animation, which was 10 seconds long (300 frames). Two levels of adaptive super-sampling were used.

The third animation used the original camera movements and object files, but added a moving blimp and large weight which was dropped to the ground (789 polygons and 43 quadrics). The total length of the animation was slightly over 32 seconds (932 frames). Two levels of adaptive super-sampling were used.

The images displayed a large degree of interreflection, from which no reduction in rendering time is attained. Therefore the savings generated is called an informal lower bound.

There are two ways of measuring performance of this algorithm. The first is the number of total rays saved. This gives an indication of how much time is saved, but it does not take into account the overhead of the reprojection. If instead the actual time required to generate frames is taken as the metric, the implementation method as well as the load on the rendering machine at any given moment will affect the measurements. Therefore, both measures of savings are indicated, with the claim that the actual savings is somewhere between the two.

Frame 0 of animations 1 and 3 (seen in figure 6) was completely ray-traced to serve as the base data frame. It would have been more efficient to fully ray-trace frame 300, which would allow frames to be inferred towards both ends of the animation simultaneously. However, frame 0 contains nothing except the ground and sky planes, meaning that the "meat" of the animation must be inferred by this algorithm. For demonstration purposes, this method best illustrates the abilities of the method.

In the first animation, despite an average of over 196,000 secondary rays per frame, over 60% of the total number of rays cast were retained. Ignoring machine load effects on rendering time, a 53% speedup in rendering time was achieved over a standard ray-tracer .

Many algorithms which purport to speed animation generation time use diffuse images for testing. These measurements did not do so in order to show the power of the technique. It is a simple matter to count rays, however, and were this a strictly diffuse animation, almost 80% of the ray casting would be saved by using this method. Even if the same amount of time for reprojection and verification overhead is assumed, a diffuse image would save over 70% in rendering time.

Figure 6. Frame 0 of animations 1 and 3. Two infinite planes.

In the second animation, the average number of primary rays actually computed per pixel was 0.05, or about 3% of the primary rays in the entire image, and the time savings was on the order of 65 - 70%. The static camera of the animation meant that no position checking needed to be done except against the moving object. However, this also meant that all unshadowed positions must consider the moving object in the shadow ray verification; hence, the difference between the percentage of rays cast and the rendering time savings. One frame of the second animation is shown in figure 7.

The final animation contained moving objects and a moving camera, so that most position and shadow rays needed to be verified. Despite this, 55 - 60% fewer rays were cast and the image

was generated 40 - 50% faster than completely ray-tracing the image.

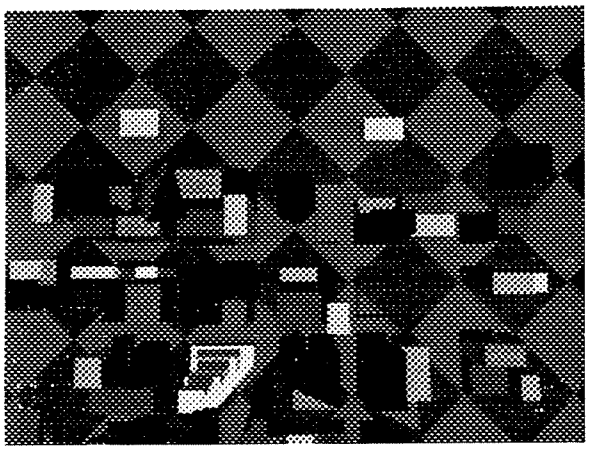

Figure 7. A frame of the second animation.

Other Concerns

Image Quality

No quantity of ray savings is significant if the animation frames produced are not of quality. The algorithm as described is equivalent to ray-tracing an image which has been jittered by some random value within the sampling cell. This effect reduces aliasing by introducing noise into the image.

In figures 8-10 are inferred frame 150 from animation 3, the same frame fully ray-traced, and the difference image between the two with enhanced pixel values for easier viewing. The same has been provided in figures 11-13 for frame 450.

Visually, there is little difference between the inferred frames and the fully ray-traced frames. The difference images reveal that the disparity occurs on edge boundaries. This exists because the boundaries in the inferred frames have reprojected from other frames and, while they can be seen if a ray is cast somewhere through that pixel, they do not always match up with the rays sent through a different portion of the pixels in the fully ray-traced view.

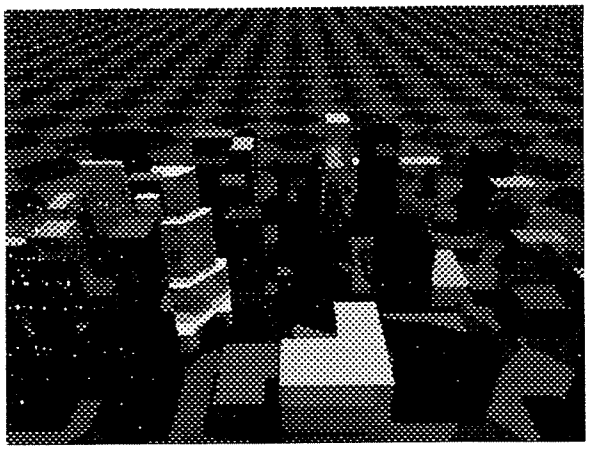

Figure 8. Inferred frame 150.

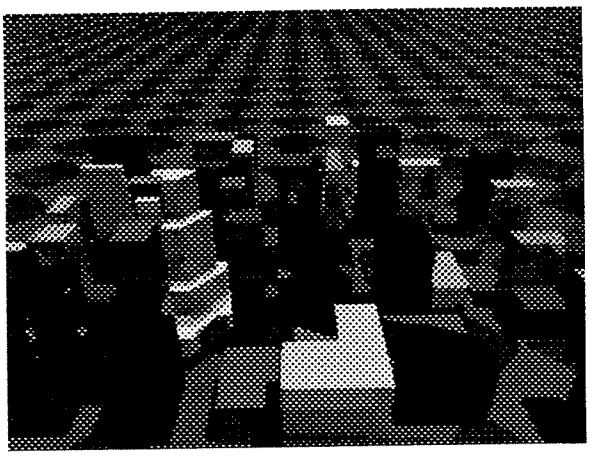

Figure 9. Fully ray-traced frame 150

Finally, figure 14 is presented. This is inferred frame 599 of animation 1, inferred through 598 steps from the initial frame. No visible image quality is lost by the algorithm no matter the number of continuous inferred frames.

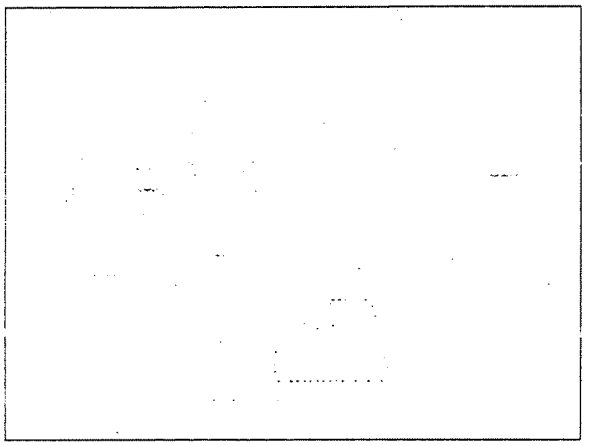

Figure 10. Difference image of figures 8 and 9. Pixel values in the range of 0-50 have been mapped to 0-255.

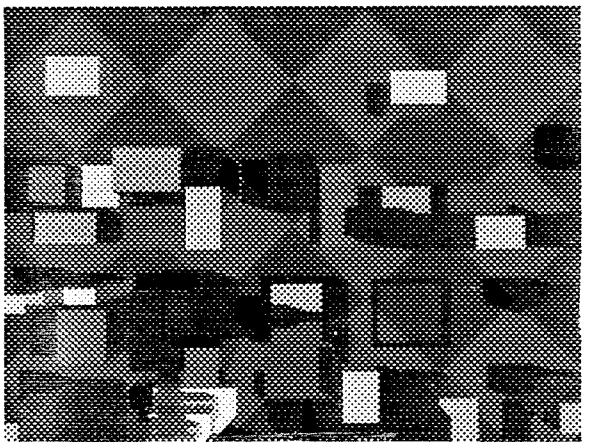

Figure 11. Inferred frame 450.

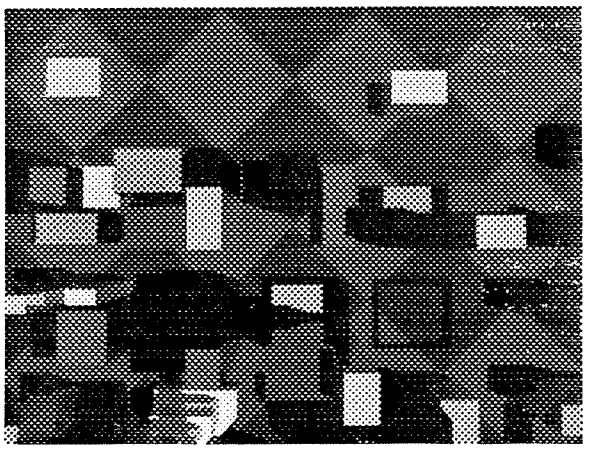

Figure 12. Fully ray-traced frame 450.

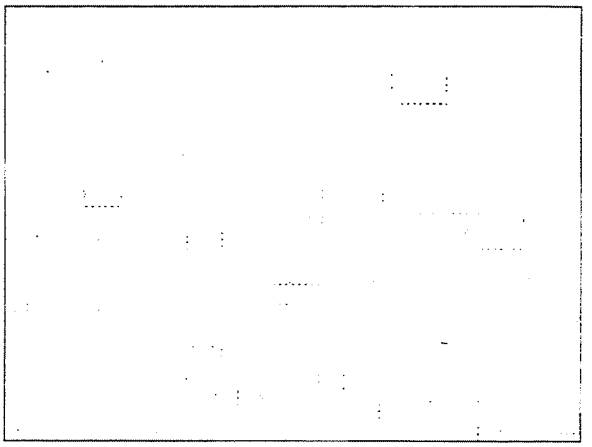

Figure 13. Difference image of figures 11 and 12. Again, pixel values have been enhanced.

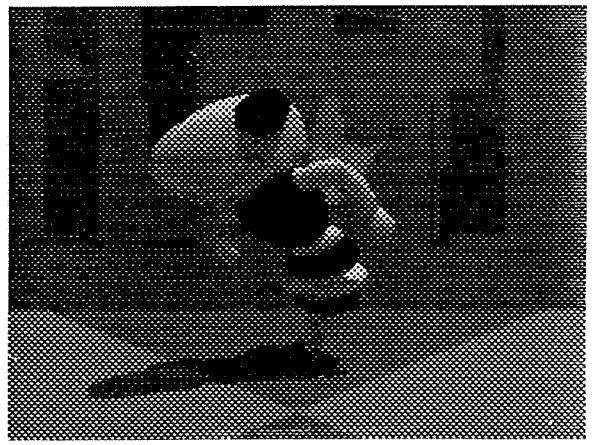

Figure 14. Inferred frame 599.

Ray-Tracing Efficiencies

Most methods for increasing the speed of generating ray-traced images can be used to generate the rays of inferred images on both the missed positions and the high-level rays which need to be generated. For example, hierarchical bounding boxes, shadow buffers, space partitioning, pixel row planes [Me93], pixel boxes (more efficient two-dimensional pixel row planes), and octrees can all be used without alteration in the algorithm. Any technique which allows point sampling at the first level of

ray-tracing can be used in generating the inferred animation frames.

Moving Light Sources

Light source movement requires that most shadow rays be recast and that the diffuse color be recalculated. In the algorithm as implemented, movement of the light sources allows only about 10% savings over full ray-tracing. This percentage, however, might be larger with a more efficient shadow ray algorithm, and will not be worse than completely ray-tracing the image.

Motion Blur

One method of generating motion blur is to use distributed ray-tracing, where the samples are jittered both spatially and temporally. The inferred frames algorithm can be used with distributed ray-tracing and motion blur except that the new viewing positions may vary spatially, requiring a different projection for each sample. A random spread of temporal points can be expected in an inferred motion blurred image, but it cannot be guaranteed that the inferred temporal locations will cover all of the time quantum represented in the frame.

A small problem with motion blur and inferred frames is that it is possible for several samples at different temporal locations to reproject to the same position spatially. Since the time coordinate is random, any of these points may be used, and in any case the sample will be verified at its temporal location during the verification phase.

Storage Details

The RAM needed when calculating frames is not insignificant. Each position must hold the three-dimensional intersection and normal vector, 3 bytes for the diffuse color, an object id tag, and a shadow tag for each light source. The intersection and normal can also be kept as floats, but all math should convert them to doubles. The id tags may be 2 byte integers if the number of objects in the image is less than 2^{16}.

This requires a minimum of 29 bytes per sample. For a 640 by 480 image rendered at 16 samples per pixel, over 142 M are required. This number does not include the memory requirements of the image object file. While this quantity will not be problematic in the future, for now images must be created which are smaller, sampled less frequently, or created in sections.

Given an arbitrarily large memory, this algorithm could be extended to retain information for the entire ray-tree, effectively verifying each ray as needed. Moving cameras would preclude using the information after the first level, but it would provide for an estimate of which object is intersected. Static camera animations could use the information to reuse rays and colors to any depth.

Parallelization of Frame Generation

The creation of animation frames using this technique is a linear process; the previous frame must be completed before the next frame can be begun. It may be desirous in long animations to render frames on different machines.

Obviously, camera cuts are natural points of breaking the animation. Otherwise, it is suggested that if an anti-aliasing technique is used that every kth frame be fully ray-traced (k an arbitrarily large odd integer, but based on the number of machines available and their relative speed), and infer the $\lfloor k/2 \rfloor$ frames on either side of the fully ray-traced image.

If anti-aliasing is not being used, the aliasing artifacts will not match at the join points, so the matching points should be placed in sections of fast camera movement.

Conclusions

Decreasing animation frame generation time by exploiting spatio-temporal coherence with a moving camera position has been largely ignored because of the general perception that all information is lost when the camera moves. It has been shown not only that data can be saved from the first level of ray-tracing, but that the shadow ray information may also be retained in many cases. While some savings are lost to the eye-point dependent reflection and refraction rays, it should be noted that most of the world is not reflective and refractive but predominately diffuse.

In the animation performance test with static light sources, the algorithm saved 50 - 90% of the rays, and 45 - 66% in rendering time. A similar animation with diffuse-only objects would have resulted in an 60-96% reduction in rays cast. The animation consisted of polygons and quadric surfaces, but the technique is not limited to these objects. Any scene which can be ray-traced in a point-sampled manner is usable material for the algorithm.

References

[AH92a] Adelson SJ and Hodges LF Visible Surface Ray-Tracing of Stereoscopic Images. Proc SE ACM: 148-157

[AH92b] Adelson SJ and Hodges LF Stereoscopic Ray-Tracing. Georgia Tech TR GIT-GVU-92-17. To appear in The Visual Computer.

[AH93] Adelson SJ and Hodges LF Exploiting Spatio-Temporal Coherence in Ray-Traced Animation Frames. Georgia Tech TR GIT-GVU-93-01.

[Am84] Amanatides J Ray Tracing with Cones. Comput Graph 18 (3): 129-135.

[Ba88] Badt, Jr. S Two algorithms taking advantage of temporal coherence in ray tracing. Vis Comput 4 (3): 123-132

[BN76] Blinn JF and Newell ME Texture and Reflection in Computer Generated Images. Commun ACM 19 (10): 542-547.

[Bo85] Bouville C Bounding Ellipsoids for Ray-Fractal Intersection. Comp Graph 19 (3): 45-52

[CC90] Chapman J, Calvert TW and Dill J Exploiting Temporal Coherence in Ray Tracing. Proc Graph Interface 90: 196-204

[CC91] Chapman J, Calvert TW and Dill J Spatio-Temporal Coherence in Ray Tracing. Proc Graph Interface 91: 101-108

[CD88] Cline HE, Lorenson WE, Ludke S, Crawford CR, and Teeter BC. Two Algorithms for the Three-Dimensional Reconstruction of Tomograms. Med Phys 15 (3): 320-327.

[CP84] Cook RL, Porter T, and Carpenter L Distributed Ray Tracing. Comput Graph 18 (3): 137-145.

[EH90] Ezell JD and Hodges LF Some preliminary results on using spatial locality to speed up ray tracing of stereoscopic images. SPIE Proc 1256: 298-306

[FL90] Foley TA, Lane DA and Nielson GM Towards Animating Ray-Traced Volume Visualization. Jour Vis Comput Anim 1 (1): 2-8

[Fu86] Fujimoto A ARTS: Accelerated Ray Tracing System. IEEE Comput Graph Appl 6 (4): 16-26

[Gl84] Glassner AS Space Subdivision for Fast Ray Tracing. IEEE Comput Graph Appl 4 (10): 15-22

[Gl88] Glassner AS Spacetime Ray-Tracing For Animation. IEEE Comput Graph Appl 8 (2): 60-70

[Gr86] Greene N Environment Mapping and Other Applications of World Projections. IEEE Comput Graph Appl 6 (11): 21-29.

[GR90] Gudmundsson B and Randen M Incremental Generation of Projections of CT-

Volumes. Proc First Conf Vis Biomed Comput: 27-34.

[HH84] Heckbert PS and Hanrahan P Beam Tracing Polygonal Objects. Comput Graph 18 (3): 119-127.

[Ho92] Hodges LF Time-multiplexed stereoscopic computer graphics. IEEE Comput Graph Appl 12 (2): 20-30

[HZ82] Hubschman H and Zucker SW Frame to Frame Coherence and the Hidden Surface Problem: Constraints for a Convex World. ACM Trans Graph 1 (2): 129-162

[KK86] Kay TL and Kajiya JT Ray Tracing Complex Surfaces. Comput Graph 19 (3): 269-278

[Je92] Jevans DA Object Space Temporal Coherence for Ray Tracing. Proc Graph Interface 92: 176-183

[Me93] Meyer TC Efficiency Techniques for Ray Tracing Computer Graphics. Undergraduate research project for Dr. Larry F. Hodges, College of Computing, Georgia Tech.

[MH90] Murikami K and Hirota K Incremental Ray Tracing. Eurograph Workshop on Photosimulation, Realism, and Phys Comput Graph 23 (3): 15-29

[RW80] Rubin SM and Whitted T A 3-Dimensional Representation for Fast Rendering of Complex Scenes. Comput Graph 14 (3): 110-116

[SS89] Sequin CH and Smyr EK Parameterized Ray Tracing. Comput Graph 23 (3): 307-314

[Wh80] Whitted T An Improved Illumination Model for Shaded Display. Commun ACM 23 (6): 343-349.

Acknowledgments

"3D Buzz" was modeled by Thomas Meyer using the Wavefront render. The city model was created by Augusto Opdenbosh using AutoCAD.