The Exploitation of Image Construction Data and Temporal/Image Coherence in Ray Traced Animation

by

Dana T. Marshall, M.A., B.A.

Dissertation

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

The University of Texas at Austin

May 2001

The Exploitation of Image Construction Data and Temporal/Image Coherence in Ray Traced Animation

Approved by
Dissertation Committee:

_____

_____

_____

_____

_____

To Someone

Acknowledgments

I'd like to thank Don Fussell for his continuing inspiration and patience. Thanks also go to Nina Amenta, A.T. Campbell, Harrick Vin, Dan Watkins, Denice Goldshmidt, Mrs. Miller, Mr. Bryant, and Arthur Marx.

Dana T. Marshall

The University of Texas at Austin

May 2001

The Exploitation of Image Construction Data and Temporal/Image Coherence in Ray Traced Animation

Publication No. _____

Dana T. Marshall, Ph.D.
The University of Texas at Austin, 2001

Supervisor: Donald S. Fussell

The similarity in successive images generated for computer graphic animated sequences is a source of great possibilities for accelerating rendering time by avoiding seemingly redundant calculations. There is a wealth of information that a renderer calculates or accesses in the course of creating an image that can be used to aid the creation of future frames. Previous solutions either incurred visibility errors or used methods to guarantee the accuracy of images that could be prohibitively time consuming.

This dissertation proposes an elegant and fast method that speeds the calculation of ray traced animated scenes constructed of convex objects by a combination of temporal and image coherence. The algorithm reuses rays

calculated from previous frames by transforming them to their new locations and updating their color from their shading and material properties.

The solution presented guarantees the accuracy of the visibility and shading of reused samples while minimizing calculation time. The method works with any existing bounding volume acceleration method and is compatible with jittered anti-aliasing schemes. It handles reflections and refractions and does not depend upon any part of the scene remaining static. It does assume the existence of some continuity in motion, which is usually found in animated sequences, but it will gracefully degrade to standard ray tracing for scenes with pathological animation design.

Contents

Chapter 1

Introduction

In the effort to speed the calculation of computer generated animated sequences there are a number of methods that take advantage of some form of coherence. Assuming that it is highly probable that a pixel in an image will not greatly differ from its neighbor leads to methods that exploit *image* coherence, while using the similarities between sequential frames exploits *temporal* coherence.

The approach introduced uses rendering information from previous frames. When rendering an image, any algorithm calculates a great deal of usable information: object motion, orientation and location as well as depth, part and shading information at pixel locations. There is a wealth of valuable information generated besides the color of individual pixels.

Re-using this information in generating the current frame leads to great savings in rendering time and avoids the problems incurred by re-using old imagery. New imagery calculated this way is accurate and never out of date. Information reuse works with objects like mirrors that interact with their environment. Visibility in future frames is solved faster and more accurately

than in previous approaches and scenes are allowed to be dynamic and not restricted to a kind whose visibility can be solved by a simple depth sort. Exploiting this information not only enhances the rendering time of a given renderer but has the capability to extend the capabilities of the renderer to include new techniques.

Rendering techniques other than image-based rendering have shown limited use of (or the need for) information recycling. In the use of a hierarchical image buffer to solve the visibility problem, the set of visible polygons from a previous frame are drawn first in the current frame in order to form an approximation of the current frame that is used to speed its calculation. The information used is the contents of the set of visible polygons - not their old locations. In another rendering acceleration technique, the use of directional coherence maps, the algorithm generates a hidden line drawing of a frame in order to aid it in determining the complex areas of the current image. This is an instance where information from a previous frame could be used to augment the visibility solution but is instead calculated from scratch.

This paper introduces this information reuse in the environment of ray traced animated sequences. Ray tracing is a technique that is known for creating images with sharp shadows and complicated reflections. It determines the color of a pixel by finding which object in the scene intersects the 'ray' from the viewer through the pixel. Once this object intersection is found, more rays may be shot to determine lighting, reflections, or refractions.

When a ray-object intersection has been successfully determined for a pixel, the approach introduced here exploits that information by re-using the intersection in subsequent frames thereby avoiding the need to re-trace the ray. By consulting the viewer motion and object motion, the intersection can

be transformed to its location in a future frame. The color can be accurately recalculated from the updated surface normal and the object's shading parameters. This avoids retracing the ray as long as it remains visible, avoiding the costly ray/scene comparisons that form the the bottleneck in ray tracing.

Previous approaches have updated the positions of ray-object intersections before, but only with primary rays. The method introduced here has the new advantages that it works with reflections and refractions and does not depend upon any part of the scene remaining static to incur savings. The method works with any standard bounding volume acceleration scheme commonly found in ray tracing and measures positive results against scenes already accelerated by bounding volumes.

Once an intersection point has been translated to its location in the new frame the visibility problem remains unsolved: there is no guarantee that another object hasn't moved in between the intersection and the viewer, or it is also possible for the object to become shaded or unshaded by any of the light sources.

Previous algorithms that exploited temporal or image coherence in ray tracing either didn't solve this problem, solved it by rendering a hidden line drawing of the next frame or solved it by retracing all of the primary rays; both solutions can easily achieve calculation times that approach that of rendering each frame from scratch. The method introduced here combines the accurately updated information from previous frames with an enhanced z-buffer to generate visibility-correct images for future frames. A *visibility correction* algorithm quickly detects image areas where changes in visibility may occur. The renderer can then calculate new imagery for these areas. This new approach quickly determines the areas of the image where visibility is correct-

the vast majority of time is spent calculating new rays in image areas where visibility is uncertain. The enhanced z-buffer treats reflections and refractions as if they were in their own image buffer, thus allowing fast, correct visibility to be calculated with reflected imagery.

This new approach also takes advantage of image coherence in ways similar to previous approaches but with two new enhancements. First, reflected or refracted imagery is determined first and then combined with primary imagery using alpha information. This allows image sub-sampling to appear in different pixels in the reflected and primary image. This, in turn, allows greater savings in ray/intersection reuse. Secondly, the method can determine the image area of unoccluded space by exploiting information gained when rendering the image. This unoccluded area information is used in connection with a hierarchical image buffer. to find large image areas where minimal sampling can occur. Both of these approaches are new and allow greater savings when used in connection with ray/intersection reuse.

Chapter 2 outlines previous work in this area and related areas. Chapter 3 describes the problem of maintaining correct visibility with this approach. Chapter 4 describes the problems and solutions related to reflections and refractions when re-using rays traced in previous frames. Chapters 5 and 6 show 2 solutions to the visibility problem. Chapter 7 tells how the approach is accelerated further by image sub-sampling. Chapter 8 explains the solution to solving the position of reflected and refracted rays in planes and spheres and sketches the solution to more complex surfaces. Chapters 9 and 10 show results and Chapter 11 briefly describes future directions.

Chapter 2

Previous Work

2.1 Ray Tracing

Ray tracing is a rendering algorithm that is known for creating pictures with complicated reflection and refraction effects and well defined shadows. [69, 24] A simple ray tracer is diagrammed in figure 2.1. Given a 3d scene geometry, shading parameters and a viewer's position and view direction, a ray tracer generates an image by 'shooting' imaginary rays from the viewer's location through an image plane towards the scene geometry. The image plane is broken up in to a grid corresponding to the image resolution. As each ray is shot through the interior of each pixel in the image, it is compared to all of the objects in the scene and tested for intersection. Of all the objects that the ray intersects, the intersection closest to the viewer is the visible surface in that direction. The surface normal and shading characteristics of the intersected surface are found and the correct color is calculated. If shadowing is desired, an additional ray is shot from the intersection location towards any

existing light sources. If an intersection is found with any object, then the surface is in shadow for that light source at that intersection point. If an intersected object is reflective or refractive, secondary rays are shot starting from the intersection point. Figure 2.1 shows a primary ray shot from the viewer that strikes a reflective sphere B. In order to find the correct pixel color, the reflection direction is found and a secondary ray is shot. The closest surface that the secondary ray intersects (sphere A) is then referenced for shading parameters, additional light rays are shot, and the final color is calculated. If a ray encounters more reflective surfaces additional rays are shot until the final pixel color is calculated or until the user-specified maximum number of reflections is reached. (Or until the available memory or user's patience is exceeded)

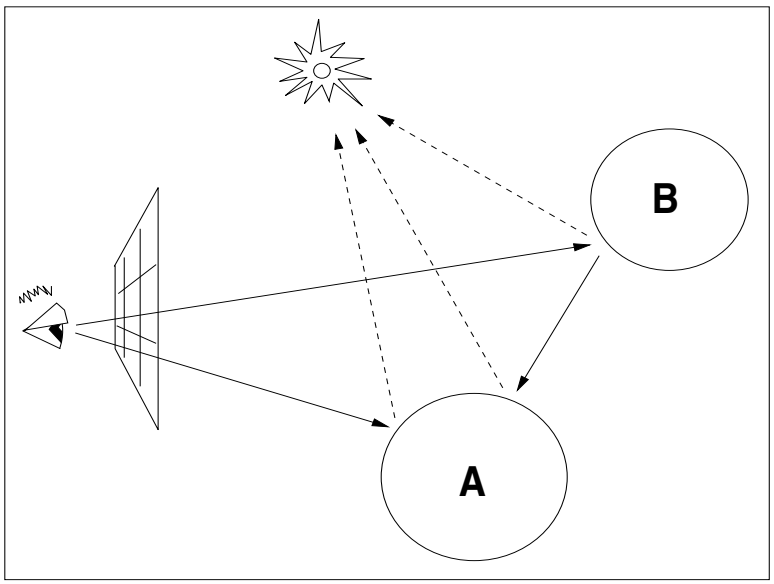

Figure 2.1: Ray Tracing

The simplest (and slowest) implementation would compare a ray against all of the polygons in the scene geometry for each pixel. This can be seen given that there are 262144 pixels in a 512x512 image and scene geometries with 100000 polygons are not uncommon, the execution time for each frame would include the time for 26 billion ray/polygon comparisons.

A family of acceleration techniques use some kind of space hierarchy to quickly eliminate most of the scene from intersection calculations. These fall into two categories: bounding volumes and space partitioning. A bounding volume hierarchy is created by first finding a simple volume bound (usually a sphere or box) that surrounds parts of the scene that are close together. The ray tracer can compare rays against that bound. If no intersection is found then all objects within the bounds can be eliminated from contention and the tracer can concentrate on the rest of the scene. Bounds themselves can be grouped together and bounded, creating a hierarchy that can be traversed by the tracer, thus hopefully eliminating large numbers of intersection tests.

A space partitioning scheme finds a bounding volume that encompasses the entire scene. It then subdivides the volume and determines which scene objects intersect the sub-volumes. Objects may overlap and be listed in more than one sub-volume (as opposed to a bounding volume scheme where the volumes themselves may overlap and occupy some of the same space.) There has been much work on bounding volume hierarchies and partitioning schemes and such approaches are a standard part of any raytracer. However, determining ray/object intersections is still the largest time consumer of the algorithm. The methods introduced here work with any bounding volume scheme and measure their progress against a raytracer using bounding volume acceleration.

2.2 Image-Based Techniques

There has been much recent work in using image-based techniques to accelerate rendering animated sequences - not necessarily ray traced [3, 10, 13, 4, 28]. All of these methods speed the calculation of an image frame by reusing portions of images rendered in preceding frames.

Macial et. al. [44] use image-based techniques with walkthroughs of static scenes. The scene is partitioned into an octree where the lowest nodes correspond to individual objects in the scene. An object may vary its rendering complexity by choosing whether to use texture-mapping, or varying the object's level of detail. Varying the level-of-detail (or LOD) allows the renderer to draw simpler (and therefore faster) versions of an object as it recedes from the viewer[15, 22, 33, 34, 43, 66, 60, 59, 67] But the new addition to this system was that axis-aligned views of the objects are precalculated, and these images may be used in lieu of rendering the objects if the estimated error of the resulting image is acceptable. Precalculated images of objects are combined at higher levels in the scene hierarchy and may be used to represent the entire contents of an octree node. This means a number of images need to be precalculated, but since the scene is static this is a viable option.

One system [57] avoids precalculating images by reusing parts of images that had already been rendered in previous frames of the animation. The static scene is preprocessed by dividing it up using a BSP tree[1, 50, 51, 52]. A Binary Space Partition (or BSP) tree is a binary tree where the root node represents the entire scene space. Given any plane that divides that space, the two resulting (possibly unequal) halves are represented by the two nodes that are the root's children. The children are then themselves divided (by possibly

different planes) to create further nodes. BSP trees are known for their use in fast polygon occlusion algorithms. In this system, the scene is rendered starting at the lowest nodes and images from lower nodes are composited to create images for higher nodes. The image generated from any node can be saved for future use as a texture map on a polygon whose screen size encompasses the node. Then, when rendering future frames, the algorithm decides whether to render the contents of the node or to reuse the texture map. The texture-mapped polygon is translated and warped according to viewer motion in order to match the deformations that would happen to the actual geometry. If the contents of the node are drawn, it may contain texture maps of its children.

The decision is made due to estimates of the work that would be saved by using the texture map and estimates of the accuracy of using the warped texture map. This is derived from the screen size of the node, the number of polygons in the node, and estimates of the viewer motion.

Lengyel et. al. extended the work to use image-based techniques in dynamic scenes. This approach [41, 64] divides a scene's parts into layers which will be rendered into their own frame buffer and later composited to make the final image. The layer's frame buffers, or sprites, contain alpha information and transformation variables that will warp them into position in the final image. The sprites' position is updated every frame, but the image need not be rendered as often. The system allows level-of-detail variation in underlying geometry and allows different renderers to be chosen depending on image quality and rendering speed. Scene parts and layers must be determined manually before hand. This predetermines the divisions of a scene that can be reused. If subparts of the divisions change visually, the entire division must

be re-rendered. If the objects that the layers represent intersect (or get close) in three space, then the layers must be combined, possibly creating one very large layer, which must be re-rendered as often as the most transient of the original layers.

None of these image-based approaches are designed to be used for ray traced imagery. They fare especially well if the objects in the scene have minimal visual interaction with their environment. Texture-mapped objects with minimal shading due to light sources are best because their images can be reused for longer periods of time - once an object's image is replaced with a newly rendered one there is less of a change in the object's appearance and less of a popping artifact in the animation than objects (like highly reflective ones) whose appearance interacts with the environment.

2.3 Temporal Coherence in Ray Tracing

Ray traced images in their simplest form are generated by defining a color for each 'ray' whose origin is the viewer location and whose direction is determined so that it goes through fixed positions in each pixel. Each ray is compared to all of the objects in the 3d scene, and if it intersects an object, the color can be calculated from the surface properties of the object. Further rays can be spawned from the intersection location in the direction of the light sources to discover if the object is shadowed. If the object is reflective or refractive further rays can be shot in the reflection/refraction direction to find the reflected/refracted color.

Animated sequences of ray traced imagery have usually been created one frame at a time, discarding all information calculated from the previous

frame. Recently there have been advances in the reuse of information on a frame to frame basis[36, 63, 29, 47, 9, 6, 49, 45, 11, 37, 7].

One early idea using temporal coherence to accelerate ray tracing involves rendering an animated sequence whose motion parameters are known[23]. The method extends the use of bounding volumes and space subdivision to accelerate ray tracing by bounding objects in time as well as space. Rays not only have x, y, and z coordinates, they also have a time coordinate and are compared against 4d bounding volumes using 4d plane equations. When a ray is shot for a particular time the question asked of the bounding volume hierarchy is not just 'Does this ray intersect this object?' but 'Does this ray intersect this object at this time?'.

This enables the bounding volume hierarchy to be calculated once at the beginning of the scene instead of at the beginning of each frame, and this is the primary source of the algorithm's advance in efficiency. Frame calculation itself is not accelerated much beyond the common effects of the use of bounding volumes except that the method introduces a new bounding volume/ space subdivision hybrid scheme which is not temporally based. The algorithm re-shoots all rays every frame and does not reuse any information gleaned from rendering the previous frame.

Another approach uses the idea of remembering pixel values and translating them to their subsequent location based upon their object's motion parameters [2]. An image is ray traced and all pixel values were translated to their new locations for the next frame. Determining if the samples are occluded is done by retracing the ray from the viewer to the intersection point, comparing it with the bounding volume hierarchy in its new location.

Savings are gained by the fact that the intersection point itself needn't