Spatio-Temporal Coherence in Ray Tracing

J. Chapman
T. W. Calvert
School of Computing Science

J. Dill
School of Engineering Science

Simon Fraser University
Burnaby, British Columbia

Abstract

The majority of images being rendered on systems today are portions of contiguous sequences of frames yet the usual practice is to render each frame as an isolated image. These frames exhibit a high degree of temporal coherence, i.e. each frame is usually very similar to immediately preceding and succeeding frames. This coherence is the result of an underlying spatio-temporal coherence in the model used to represent the scene to be animated. An adaptation of the basic ray tracing algorithm is presented which exploits this spatio-temporal coherence. The algorithm reduces the computational cost of image generation in sequences by calculating the continuous solution to ray-polygon intersections thus avoiding multiply redundant discrete intersection calculations. An implementation of the proposed algorithm is described and empirical results are presented.

Keywords: ray tracing, image space, coherence, temporal, spatial

1. Introduction

Ray tracing is an extension, of the Appel[1] ray casting technique, due to Whitted.[2] Ray tracing is an attractive method of rendering images because of its simplicity, elegance, and the realism of the images it is capable of producing. Ray tracing is based on a model of a pinhole camera; a ray is cast from a viewing point and passes through an element of a regular mesh overlaid on the image plane. The value given to the image plane at the mesh element is determined by using the ray to point sample as it interacts (reflecting, diffusing, refracting) with the environment being modelled. As rays are cast from the viewpoint through each of the mesh elements a raster image of the scene is generated. In addition it is often possible to extract clues to the realistic rendering of new phenomena from the physical method of image generation upon which the rendering process is modelled. As a rendering technique the major drawbacks to ray tracing are a large computational cost, due mainly to calculating ray-object intersections, and difficulties in the generation of realistic diffuse reflection phenomena, e.g. colour bleeding.

The majority of work on ray tracing has been either to expand the range of phenomena which it can successfully render, e.g. Amanatides,[3] Cook,[4] Kajiya,[5] Peachey,[6] and Fournier,[7] or to reduce the rendering time by reducing the cost of ray-object intersections. Reducing the cost of intersection calculations generally requires either substituting geometrically simpler primitive objects for more complex ones or restructuring the data in some way so as to eliminate unnecessary intersection tests. The former can range from the work of Kay[8] which has provided faster intersection algorithms for objects with convex hulls to Bouville's[9] work in finding more efficient bounding volumes for intersection testing. In the latter area significant results have been achieved in the hierarchical structuring of the data, notably the application of Octrees by Glassner[10] which employ space partitioning of the data so that a ray is tested against objects in the order in which it would encounter them, and the idea of hierarchically nested bounding volumes by Rubin[11] and Whitted[2] which employ object partitioning to provide bounding volume intersection tests at increasing levels of detail in the object. Weghorst[12] has further investigated the relative computational advantages of bounding volume selection, hierarchical environment descriptions and visible surface preprocessing.

As in scanline rendering algorithms, attempts have been made to exploit coherence of various types in ray tracing algorithms. Heckbert[13] has introduced the notion of 'beams' which exploit the image coherence of polygonal surfaces to perform antialiasing and reduce rendering time. Rather than cast individual rays Heckbert casts beams with the initial beam covering (corresponding to) the entire image plane. As the beam strikes objects it is subdivided and the process continues recursively in a manner reminiscent of Warnock.[14] Aliasing is reduced since we are no longer simply point sampling. Since coherence is maintained through reflection by a polygonal (planar) surface, duplicate reflection calculations are avoided (as compared to several discrete rays which strike the same object). A perhaps more subtle approach to using coherence is made by Joy[15] in the calculation of ray intersections with parametric surface patches. The calculation is made by a quasi-Newton iterative method and information from the previous ray intersection is used to provide the initial values for the iteration. Hubschman[16] has attempted to take advantage of frame-to-frame coherence in reducing calculations. In his model the only movement allowed is that of the view point and objects are required to be convex. Preprocessing occurs for the initial frame to determine object visibility and succeeding frames are then generated after determining which objects have changed their visibility status thus reducing redundant visibility tests. Glassner[17] has demonstrated modest performance increases, rendering simple (small numbers of spheres and polygons) motion blurred sequences, by employing 4D bounding volumes (Boyse,[18] Wang[19]) to reduce redundant intersection calculations. Korein[20] has employed image and object space coherence to reduce temporal aliasing artifacts, however this method is reported [Cook[21]] to suffer drawbacks, among them 'holes' in objects that change perspective severely during one frame.

Each of the above methods relies on some form of object space coherence. In its simplest form ray tracing generates a single image from a kinetically static, three dimensional, model. The sampling process proceeds in a regular manner with adjacent pixels being rendered sequentially. In any scene significant portions of the image exhibit coherence due simply to the "physical" coherence of the objects being modelled.

If a temporal dimension is added to the process the ray tracer can generate a sequence of frames; this is usually done by generating each frame sequentially to produce a contiguous sequence of frames. There is a great deal of image space (frame-to-frame) coherence in such a sequence. If this were not so the human viewer would be unable to make sense of the image sequence being presented. Further details on image space coherence in (commercial) animation and an associated algorithm can be found in Chapman.[22] This form of coherence is a direct result of the spatial and temporal coherence represented in the model used to produce the animation, i.e. each primitive object has spatial coherence and each trajectory has both spatial and temporal coherence and thus each combination of object and trajectory exhibits spatio-temporal coherence. We are thus led to ask if we can construct an algorithm which can successfully exploit this spatio-temporal coherence to reduce the rendering time of an animation. The algorithm described in this paper is an attempt to address this problem.

2. Spatio-Temporal Coherence

Consider a 'typical' ray-tracer used to produce a sequence of frames for an animation. A data structure, representing the model at time 0, is constructed and the ray-tracer is invoked to produce the first frame of the animation. Then the data structure is modified to represent the state of the model at the next frame (time) and the ray-tracer is invoked again to produce the next frame of the animation. Repetition of this process continues until each frame of the animation has been produced. When the ray-tracer is rendering frame n it employs none of the information determined during the rendering of frames $n-1$, $n-2$, ... 1, 0. Figure 1 illustrates this process for three contiguous frames; the large square represents the portion of the image plane being rendered and the central square represents an arbitrary pixel, $P_{i,j}$ of this image; the polygon moves from right to left as time increases.

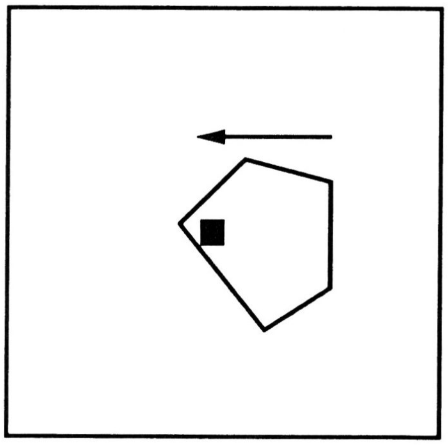
Figure 1a.

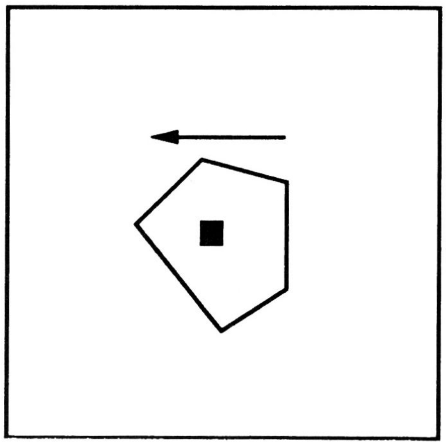
Figure 1b.

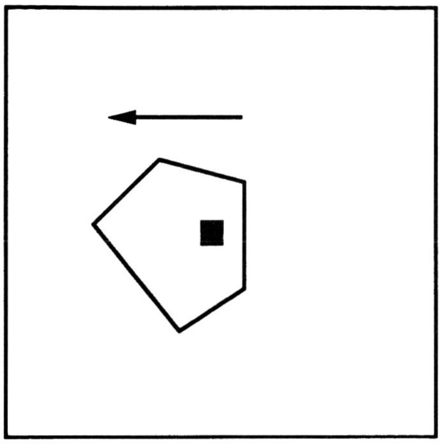
Figure 1c.

Figure 1. Intersection of a pixel with a moving polygon, in three successive frames.

As each frame is rendered in turn a ray $R_{i,j}$ is cast to point sample the value for $P_{i,j}$; each time this occurs $R_{i,j}$ must undergo an intersection test with the polygon (and surrounding bounding volumes if they are employed in the data structure) and, as can be seen, a different intersection point on the polygon's surface is determined for each frame. This process occurs in a manner which does not appear to recognize any relationship between the three intersection points generated however the three points are related to each other by a function of both the polygon's trajectoryand orientation and the origin and direction of $R_{i,j}$. This function can be determined and all the intersection points between $R_{i,j}$ and the polygon can thus be determined once for the entire duration of the animation.

The following is a description of an algorithm which operates in the suggested manner:

```
Procedure RayTrace
1. Read model descrption
2. Create data structure representing model.
3. FOR $i$=1 TO *number of rows*
4.   FOR $j$=1 TO *number of columns*
5.     Create primary ray $R_{i,j}$ for pixel $P_{i,j}$
6.     Trace($P_{i,j}$, $R_{i,j}$, $T_{start}$, $T_{end}$)

Procedure Trace($R$, $T_1$, $T_2$)
1. Find all object intersections $I_k$ with $R$ during ($T_1$, $T_2$)
2. Sort intersections by initial intersection time
3. FOR EACH $I_k$
4.   Discard portions of $I_k$ occluded by some $I_l$
5.   Let $I_r$ be remainder of $I_k$
6.   Let $m$ = start time of $I_r$
7.   Let $n$ = end time $I_r$
8.   FOR EACH secondary ray $S_{i,j}$ generated by $I_r$
9.     Trace($P_{i,j}$, $S_{i,j}$, $m$, $n$)
10.  Shade $P_{i,j}$ for the time period represented by $I_r$
```

There are several potential advantages to the above approach. The model is only read once and the internal data structure is created once regardless of the number of frames to be rendered; for large models and complex structuring schemes this could result in significant time reductions. If bounding volumes are employed then the ray for a given pixel $P_{i,j}$ will participate in an intersection test with any given bounding volume at most once (in contrast to the 'traditional' algorithm which may intersect the ray for $P_{i,j}$ with the same bounding volume once for each frame). As for the bounding volume intersections, the ray for $P_{i,j}$ will be intersected with any given primitive object at most once, regardless of the duration of the animation. Additionally, portions of the shading calculation for a surface can be calculated once regardless of the duration of the intersection.

3. Ray Intersections

The algorithm must be capable of performing two types of intersections: ray-bounding volume and ray-object. In the algorithm, as implemented, a typical hierarchical data structure is employed to represent the model with bounding volumes (axis aligned boxes) constructed from slabs [Kay[23]]. The bounding volumes are static, i.e. each bounding volume is fixed for all time. When an object is added to the data structure the trajectory (if any) associated with it is examined and the bounding volume is created so as to contain the object at all times during the animation sequence. This means that ray intersections with the bounding volumes can proceed in the usual manner. There are potential penalties for this approach however. If we make the simplifying assumption that the object exists in a flux of rays with both uniform density and a uniform distribution of ray direction, then the number of ray-bounding volume intersection tests required will increase as a function of the increase in surface area of the bounding volume (further discussion is available in Arvo[24]).

This surface area in turn depends on the relative (to the object size) motion of the object along each coordinate axis, e.g. if the object trajectory and object size are such that the component, of object motion, along each axis is twice the extent of the object in that direction then the surface area of the bounding volume will be eight times larger than if the object were static. This in turn means that eight times as many rays will pierce the bounding volume during the animation as would pierce the corresponding bounding volume during one frame of animation using a 'standard' ray tracer. Therefore, on average, more bounding volume intersection tests will be required by this algorithm if the average relative motion is greater than the number of frames to be rendered. Further investigation of alternative bounding volume techniques is clearly in order.

4. Ray-Polygon Intersections

The essential requirement of the algorithm is to be able to produce a description of all the intersection points between a ray and a (possibly) moving polygon for any given temporal interval. In the case of calculating the intersection of a ray with a static convex or concave polygon a common method (see also Snyder[25]) is to first calculate the intersection of the ray with the plane in which the polygon lies. Then the polygon and ray-plane intersection point are projected onto a plane, usually either the $X-Y$, $X-Z$, or $Y-Z$ plane for efficiency's sake. The coordinate system is then translated so that the ray-plane intersection point lies at the origin and the number of zero-crossings of the positive X-axis (in the case of projection onto the $X-Y$ or $X-Z$ plane) by polygon edges is counted. If an odd number of crossings is detected the intersection point is within the polygon else the ray does not intersect the polygon. In order to adapt this to the case of a moving polygon, in a straightforward way it would be necessary to calculate the *periods* during which an edge produces a zero crossing. These periods would have to be calculated for each polygon edge and then sorted and merged to produced a list of zero-crossing counts ordered by time which in turn determines at what periods the ray intersects the polygon (if at all). This still leaves the question of calculating the actual ray-polygon intersection points for the periods of intersection. A simpler approach is to note that intersecting a static ray with a dynamic (moving) polygon is equivalent to intersecting a dynamic (changing origin and/or direction) ray with a static polygon. In what follows it is assumed that each polygon has three translation functions and three rotation functions and that these can be represented as polynomial functions of time.

Assume we have a parametrically represented ray R,

$$R(s) = R_o + s\vec{R}_d$$

where $R_o = [x_o\ y_o\ z_o]$ and $\vec{R}_d = [x_d\ y_d\ z_d]$ and s is the ray parameter; a polygon P, with normal $\vec{N} = [ABC]$, which lies in the plane defined by

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

When P is static the intersection of R with P's plane is given by

$$s = -\frac{\vec{N}\cdot R_o + D}{\vec{N}\cdot\vec{R}_d}$$

Now if P is dynamic it will have three translation polynomials of the form

$$PT_x(t)=a_x+b_xt+c_xt^2\ldots ,$$

$$PT_y(t)=a_y+b_yt+c_yt^2\ldots \text{ and}$$
$$PT_z(t)=a_z+b_zt+c_zt^2 \quad (4)$$

and, similarly, three rotation polynomials $PR_x(t)$, $PR_y(t)$ and $PR_z(t)$. As stated above it is equivalent to move R rather than P, e.g. to apply $-P_x(T), -P_y(T)$ and $-P_z(T)$ to the ray origin. This results in a formulation of R in two parameters

$$R(s,t) = R_o(t) + s\vec{R}_d(t)$$

$$= [X_o(t)\ Y_o(t)\ Z_o(t)]+s[X_d(t)\ Y_d(t)\ Z_d(t)]$$

with

$X_o(t)$, $Y_o(t)$, $Z_o(t)$, $X_d(t)$, $Y_d(t)$ and $Z_d(t)$ being polynomial functions of time. As in the static case we wish to determine at what points, if any, R intersects the plane of P; this intersection is now given by

$$s(t) = -\frac{\vec{N}\cdot R_o(t) + D}{\vec{N}\cdot\vec{R}_d(t)}$$

and so we can evaluate s, and thus R, for any time t. Note that if the coefficients of t in $s(t)$ are all zero then the intersection point is fixed in space for all t, i.e. even though the polygon may be moving the intersection point is fixed in space for the duration of the intersection(s) of R with P; this only occurs if P is static or if P is only translated and the direction of translation is orthogonal to P's surface normal.

We now have a parameterized description of the intersection points of R and the projection plane as a function of time. Specifically, the intersection values of each coordinate are given by:

$$X(t) = X_o(t) + s(t)X_d(t) \quad (1)$$
$$Y(t) = Y_o(t) + s(t)Y_d(t) \quad (2)$$
$$Z(t) = Z_o(t) + s(t)Z_d(t) \quad (3)$$

The two functions which correspond to the axes of the projection plane (e.g. $X(t)$ and $Y(t)$ if the projection plane is $X-Y$) describe a curve of intersection in the plane. The next step is to determine which portions(s) of this curve, if any, lie within the projection of P onto the plane. Assume that the projection plane is the $X-Y$ plane and an edge of the projected polygon is coincident with the line

$$y(x) = g_0 + g_1x \quad (4)$$

then combining equations 1, 2 and 4

$$Y_o(t) + s(t)Y_d(t)=g_0+g_1[X_o(t) + s(t)X_d(t)] \quad (5)$$

If we let,

$$P_1(t) = \vec{N}\cdot R_o(t) + D$$

$P_2(t) = \vec{N} \cdot R_d(t)$
$P_3(t) = Y_o(t) - g_1 X_o(t) - g_0$
and
$p_4(t) = Y_d(t) - g_1 X_d(t)$

then equation 5 becomes

$$P_2(t)P_3(t) + P_1(t)P_4(t) = 0$$

The solution to this gives the time(s) at which the curve intersects the line and must still be checked, e.g. to see if an intersection is on the portion of the line corresponding to the polygon edge (Figure 2). If the rotation functions, $\vec{R_d}(t)$, are all degree 0 then solving $s(t)$ requires solving polynomials whose degree is at most the maximum degree of the polynomials comprising $R_o(t)$. Otherwise it may be necessary to solve polynomials of degree twice that of the maximum degree of the polynomials comprising $\vec{R_d}(t)$. As this intersection testing is repeated for each edge, a list of intersections, sorted by intersection time, is created. These points delineate sections of the curve which are inside/outside the polygon (see Figure 3); an inside/outside test, such as that described at the beginning of this section, must be applied to determine whether the first point is an entry or exit to the polygon.

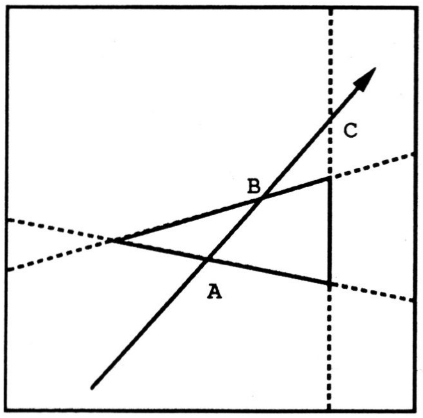

Figure 2. The false intersection (C) is culled.

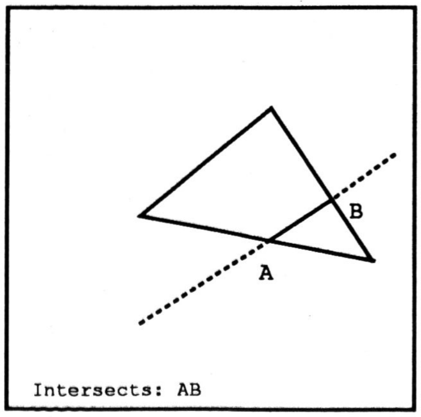

Figure 3a.

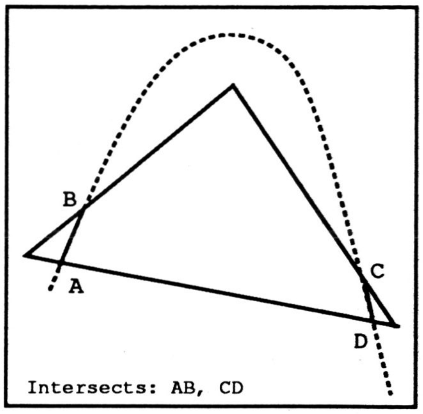

Figure 3b.

Figure 3. Examples of intersection curves.

When the resulting list of intersections between the ray and polygon has been generated a record is constructed for each which describes the duration of the intersection, the minimum and maximum values of s and the coefficients of the polynomial which generates s. These records are placed on a list in sorted (by initial intersection time) order; since a list head is kept for each frame this is a constant time operation. After all intersection records for the ray have been generated for all polygons which the ray intersects, the list is processed so that entries which overlap in time are split according to their respective values for s, e.g. if one record represents a partial occlusion of another the occluded record is truncated or split. The result is a time ordered list of intersections which are closest to the ray origin. The pixel's value for the duration of the animation can now be determined; shading for opaque surfaces can be done directly and as each record is processed reflected rays are generated as needed. In the case of a record resulting from the intersection of a ray with a moving polygon the origin of the secondary rays will only be constant under the conditions previously stated. Initially this may seem to imply that the capability to exploit coherence has now been lost, that it will be necessary to cast individual secondary rays for each frame when the primary ray intersects a moving surface. However we have already embedded the ability to deal with dynamic rays in order to treat moving polygons and since the functions describing the movement of the secondary ray are simply polynomials in t, of the same order as the original polygon's movement polynomials, we need merely include this information in the description of a ray - when the secondary ray is tested against a moving polygon the polynomials from both are combined prior to performing an intersection test.

5. Empirical Results

A partial (translation only) implementation of the algorithm has been tested on a Silicon Graphics Iris 4D platform. Three test cases were constructed and animations generated both with the algorithm described herein and with a "standard" ray tracer. In the first case the model consists of a single static polygon filling the entire image. In the second case the model consists of two polygons: one large "background" polygon which fills the entire image plane and a smaller polygon placed in front of the former, which moves from the lower left corner of the image to the upper right corner of the image. While these two cases are simple they provide information as to the relative performance of the algorithm under easily understood conditions. The first case is 'ideal' from the point of view of the algorithm and essentially defines the maximum speedup, for the particular implementation, for a sequence of that length. In order to assess the algorithm performance with a scene of more realistic complexity a model consisting of 2312 polygons was constructed from digital terrain data. In this case the animation consisted of a 'fly-by' of the terrain, i.e. every polygon was in motion (Figure 4 shows two frames from this animation). Each animation was 100 frames in length, was rendered at a frame rate of thirty frames per second and an image size of 400 (rows) by 512 (columns) pixels. The performance measurements resulting from these tests are given in Figure 5. Columns A1, A2, and A3 report results from the standard ray tracer for each of the three animations; similarly columns B1, B2 and B3 report results for the ray tracer described in this paper for each of the three animations. The first row of each table is the number (thousands) of bounding volume tests that occurred; the second row is the number (thousands) of bounding volume tests that resulted in an intersection with the bounding volume; the third row is the latter figure as a percentage; the fourth row is the number (thousands) of polygon intersection tests and the fifth row is the number (thousands) of these tests which produced an intersection between the ray and polygon being tested; the sixth row is this latter figure expressed as a percentage and the last row is the observed cpu time in seconds.

6. Conclusions

It can be seen that the program based on the proposed algorithm runs significantly faster and produces significantly fewer polygon and bounding volume intersection tests than the 'standard' ray tracing program. Work is currently under way both to add the polygon rotation capability and to investigate potential further speedups. Areas of interest in the latter case include intersection heuristics, bounding volumes and texture mapping. Calculating the continuous solution to a ray-polygon intersection can be relatively expensive in comparison to a single discrete intersection calculation; if a candidate polygon is sufficiently small or moving sufficiently rapidly the dynamic algorithm requires more time than repeated invocations of the static algorithm. It is expected that employing a simple (rapid) heuristic to select between which of the intersection algorithms to employ, for each case, will produce a hybrid rendering algorithm with increased performance. As previously mentioned the bounding volume scheme described in this paper is naive and suffers from inflation due to object motion. This is substantiated by the results shown in Figure 5 (columns A3 and B3) for the most complex test animation; the number of bounding volume intersections by rays has increased and the number of ray-polygon tests which actually produce an intersection has decreased by a proportional amount. Given that any given ray is only tested against a particular bounding volume once, independent of the number of frames, it may be worthwhile to use a bounding volume scheme which would normally be considered too expensive (slow) in a standard single frame ray tracer. In the area of texture maps it seems possible to exploit the information contained in the intersection records to access a map more efficiently, e.g. calculating the map index for an arbitrary quadrilateral can be a relatively expensive operation and it appears possible to eliminate many redundant calculations if the number and location of all intersections with a polygon are known in advance. The interested reader may also wish to refer to Catmull[26] which presents an image space solution to multiple intersections of a pixel by dynamic polygons, as part of an algorithm to produce motion-blurred animation from polygonal models.

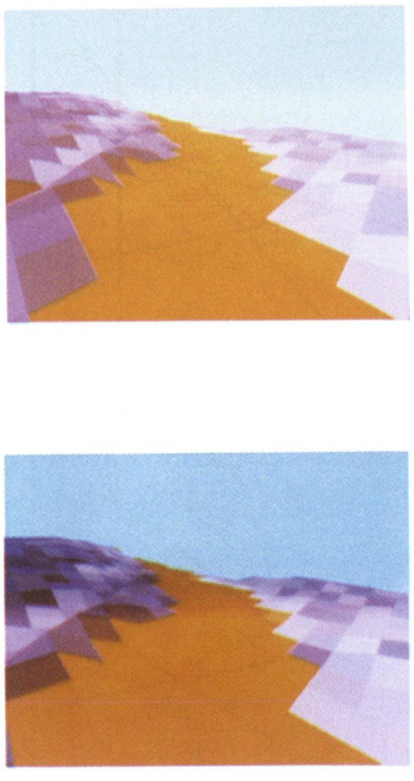

Figure 4. Two frames from the third test animation.

	A1	A2	A3
B.Vol. Test	20,480	61,440	422,532
B.Vol. Succ.	20,480	40,960	139,316
%	100	66	33
Poly. Test	20,480	20,480	28,913
Poly. Succ.	20,480	20,480	12,215
%	100	100	42
Seconds	4080	5160	16,020

Figure 5a. Discrete algorithm.

	B1	B2	B3
B.Vol. Test	204	614	17,309
B.Vol. Succ.	204	614	11,515
%	100	100	67
Poly. Test	204	409	5,620
Poly. Succ.	204	327	1,137
%	100	80	20
Seconds	349	411	2,025

Figure 5b. Continuous algorithm.

Figure 5. Experimental results of 'standard' ray tracer (5a) and ray tracer exploiting spatio-temporal coherence (5b) each rendering three animation sequences.

References

1. A. Appel, "Some Techniques for Shading Machine Renderings of Solids," *Proc. AFIPS JSCC*, vol. 32, pp. 37-45, 1968.
2. T. Whitted, "An Improved Illumination Model for Shaded Display," *CACM*, vol. 23, no. 6, pp. 343-349, June 1980.
3. J. Amanatides, "Ray Tracing with Cones," *Computer Graphics*, vol. 18, no. 3, pp. 129-135, July 1984.
4. R. L. Cook, T. Porter, and L. Carpenter, "Distributed Ray Tracing," *Computer Graphics*, vol. 18, no. 3, pp. 137-145, July 1984.
5. J. T. Kajiya and B. P. VonHerzen, "Ray Tracing Volume Densities," *Computer Graphics*, vol. 18, no. 3, pp. 165-175, July 1984.
6. D. R. Peachey, "Modelling Waves and Surf," *Computer Graphics*, vol. 20, no. 4, pp. 65-74, August 1986.
7. A. Fournier and W. T. Reeves, "A Simple Model of Ocean Waves," *Computer Graphics*, vol. 20, no. 4, pp. 17-27, August 1981.
8. T. L. Kay and J. T. Kajiya, "Ray Tracing Complex Surfaces," *Computer Graphics*, vol. 20, no. 4, pp. 269-278, Aug. 1986.
9. C. Bouville, "Bounding Ellipsoids for Ray-Fractal Intersection," *Computer Graphics*, vol. 19, no. 3, pp. 45-52, July 1985.
10. A. S. Glassner, "Space Subdivision for Fast Ray Tracing," *IEEE Computer Graphics & Applications*, vol. 4, no. 10, pp. 15-22, Oct. 1984.
11. S. M. Rubin and T. Whitted, "A 3-Dimensional Representation for Fast Rendering of Complex Scenes," *Computer Graphics*, vol. 14, no. 3, pp. 110-116, July 1980.
12. H. Weghorst, G. Hooper, and D. P. Greenberg, "Improved Computational Methods for Ray Tracing," *ACM TOG*, vol. 3, no. 1, pp. 52-69, January 1984.
13. P. S. Heckbert and P. Hanrahan, "Beam Tracing Polygonal Objects," *Computer Graphics*, vol. 18, no. 3, pp. 119-128, July 1984.
14. J. Warnock, "A Hidden-Surface Algorithm for Computer Generated Half-Tone Pictures," TR 4-15, University of Utah Computer Science Dept., 1969.
15. K. I. Joy and M. N. Bhetanabhotla, "Ray Tracing Parametric Surface Patches Utilizing Numerical Techniques and Ray Coherence," *Computer Graphics*, vol. 20, no. 4, pp. 279-285, Aug. 1986.
16. H. Hubschman and S. W. Zucker, "Frame to Frame Coherence and the Hidden Surface Computation: Constraints for a Convex World," *ACM TOG*, vol. 1, no. 2, pp. 129-162, April 1982.
17. A. S. Glassner, "Spacetime Ray Tracing For Animation," *IEEE Computer Graphics & Applications*, vol. 8, no. 2, pp. 60-70, March 1988.
18. J. W. Boyse, "Interference Detection Among Solids and Surfaces," *CACM*, vol. 22, no. 1, Jan. 1979.
19. W. P. Wang and K. K. Wang, "Geometric Modeling for Swept Volume of Moving Solids," *IEEE Computer Graphics and Applications*, vol. 6, no. 12, pp. 8-17, Dec. 1986.
20. J. Korein and N. Badler, "Temporal anti-aliasing in computer generated animation," *Computer Graphics*, vol. 17, no. 3, pp. 377-388, July 1983.
21. R. L. Cook, "Stochastic Sampling and Distributed Ray Tracing," in *An Introduction To Ray Tracing*, ed. A. S. Glassner, p. 181, Academic Press, 1989.

22. J. Chapman, T. W. Calvert, and J. Dill, "Exploiting Temporal Coherence in Ray Tracing," *Proc. Graphics Interface ' 90*, pp. 196-204, May 1990.

23. T. L. Kay and J. Kajiya, "Ray tracing complex scenes," *Computer Graphics*, vol. 20, no. 4, pp. 269-278, August 1986.

24. J. Arvo, "A Survey of Ray Tracing Acceleration Techniques," in *An Introduction To Ray Tracing*, ed. A. S. Glassner, pp. 209-213, Academic Press, 1989.

25. J. M. Snyder and A. H. Barr, "Ray Tracing Complex Models Containing Surface Tessellations," *Computer Graphics*, vol. 21, no. 4, pp. 119-128, July 1987.

26. E. Catmull, "An Analytic Visible Surface Algorithm for Indepenent Pixel Processing," *Computer Graphics*, vol. 18, no. 3, pp. 109-115, July 1984.