Exploiting 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. An adaptation of the basic ray tracing algorithm is presented which exploits this temporal (image space) coherence to reduce the computational cost of image generation in sequences. A theoretical and empirical analysis of the proposed algorithm is given. In addition a statistical analysis of 'real world' image sequences is presented.

Keywords: ray tracing, image space, coherence, temporal

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.

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. This is perhaps the most obvious form of coherence to exploit in generating an image sequence. We are thus led to ask if we can construct an algorithm where prior knowledge of (only) the pixel values of frame l and frame n can be used to reduce the rendering time of frame m ($l < m < n$). The algorithm described in this paper is an attempt to address this problem.

2. An Analysis of Image Coherence in Commercial Animation

One immediate question is whether or not there is sufficient image space coherence in typical animation sequences to warrant investigation of an algorithm which exploits this coherence. Since the types of images usually produced in a graphics research environment tend to be atypical when compared to those generated by production houses it was decided to aquire animation sequences from a commercial environment. Six sequences of animation, each consisting of sixty frames (two seconds of animation), were obtained from a local production house and subjected to simple statistical analysis. In order to prevent experimenter bias the only instructions given to the donors (other than number of frames and sequences desired) were to select sequences which they felt were typical of their work and to avoid including initial and final portions of a sequence (to prevent bias created by boundary conditions).

Figure 1 shows four frames from one such sequence. The results of the analysis are shown in figures 3 to 7. A 'pixel-event' is the time period over which the value of a pixel remains constant. Figure 3 is a plot of the number of pixel-events versus the duration. The mean duration of a pixel-event was 7.23 frames with a standard deviation of 16.27 . The results shown in Figure 3 indicate that the pixel-events fall into three general categories: very short duration pixel-events (the overwhelming majority), pixel events which span the entire sequence of frames, and the remaining pixel-events which are more or less uniformly distributed by duration. While this may appear to imply that there are too few pixel-events of sufficiently long duration to produce significant savings by exploiting image-space temporal coherence, the next figure provides a different view.

A pixel-event can be visualized as a three dimensional volume having one temporal and two spatial dimensions. Figure 4 is a graph of the volume of pixel-events (as a percentage of the total image-space volume) versus the duration. Figure 4 shows that even though the vast majority of pixel-events are of very short duration (figure 3) they account for a very small portion of the image-space volume and that more than half the image-space volume is accounted for by the relatively few (less than 10%) long term pixel-events.

The frame-to-frame coherence is defined as the number of pixels which maintain the same value between two adjacent frames, i.e. the number of pixel-events which span both frames. The average frame-to-frame coherence was found to be 88% with a minimum of 43% and a maximum of 99%. Figure 5 shows which pixels have changed between the frames shown in Figures 1b and 1c. White areas indicate pixels whose value has not changed and black areas indicate those pixels which have changed. The (long term) coherence was also measured between the first frame of each sequence and all succeeding frames in the same sequence; Figure 6 shows the coherence between the first frame and all succeeding frames averaged over all sequences analyzed. The average long term coherence was 77% with a minimum of 43% and a maximum of 99%. Figure 7 shows which pixels have changed value at some point in the animation from which the frames in Figure 1 were taken; again white areas indicate no change and black indicates change. It is clear from these results that there is sufficiently high image space coherence, both short and long term, that an algorithm which efficiently exploits either or both types of coherence should run significantly faster than a traditional ray tracer.

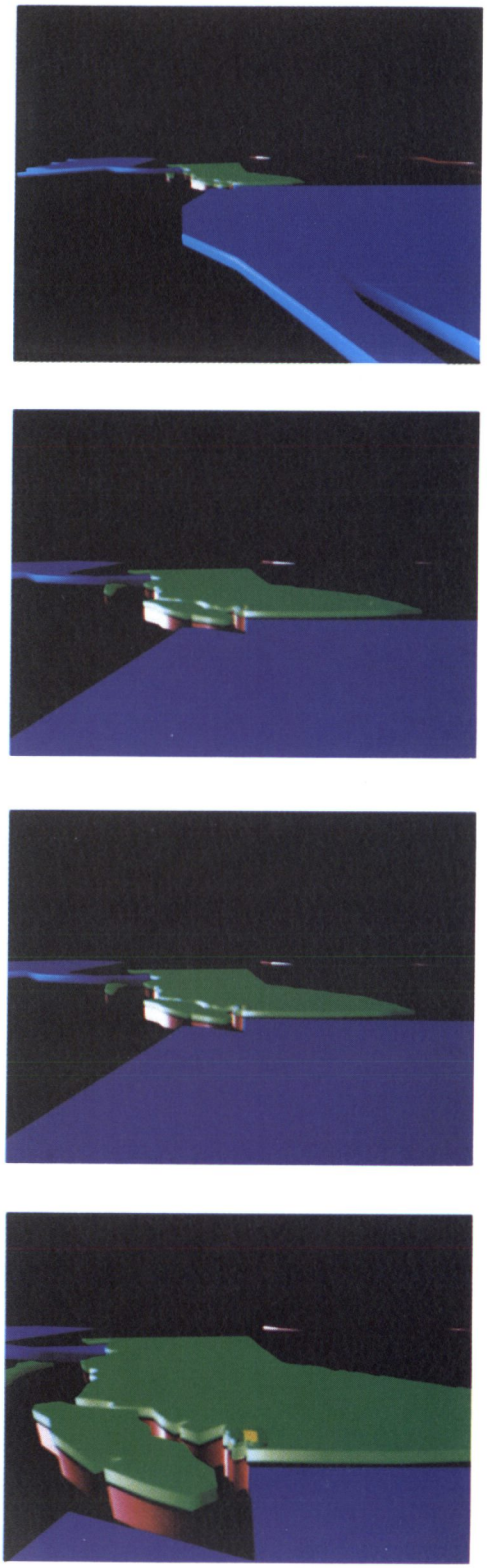

Figure 1. Figures 1a (top) to 1d (bottom) show frames 1, 30, 31 and 60 from a commercial animation sequence.

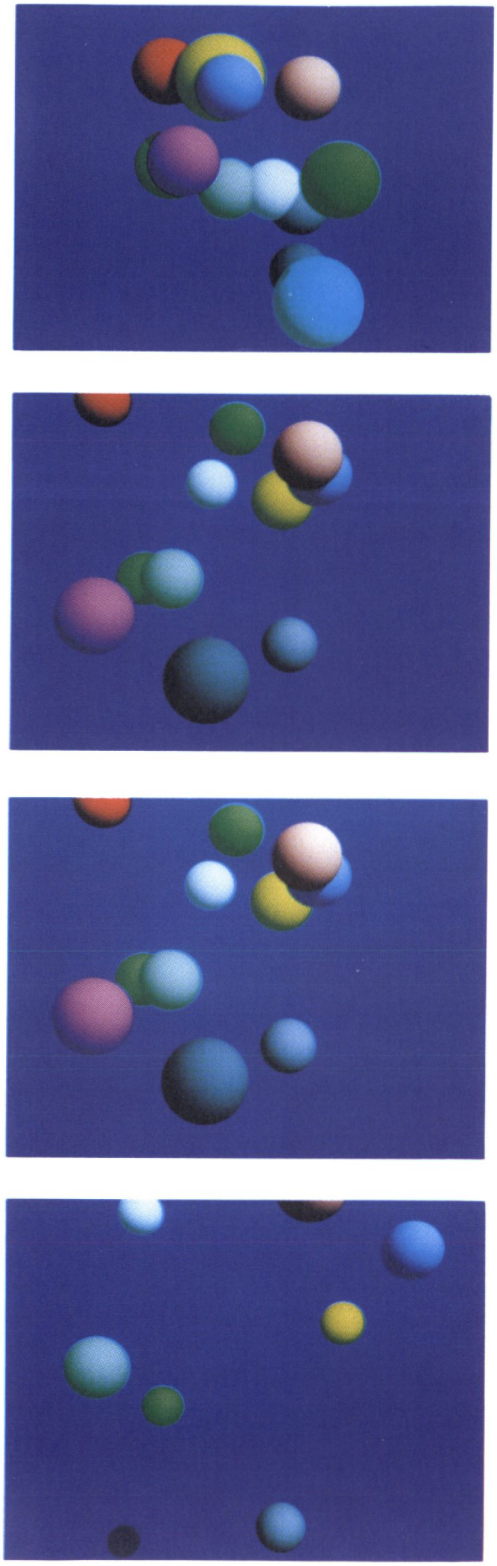

Figure 2. Figures 2a (top) to 2d (bottom) show frames 1, 30, 31 and 60 from a test animation (described in section 5).

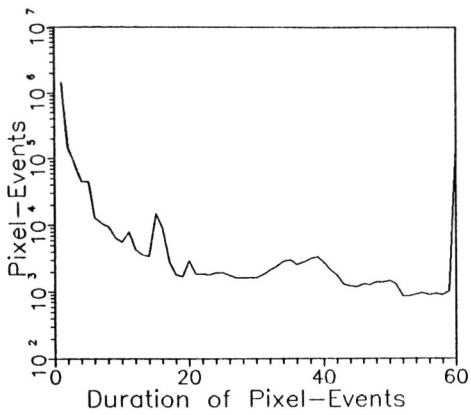

Figure 3. Number of pixel-events .vs. duration.

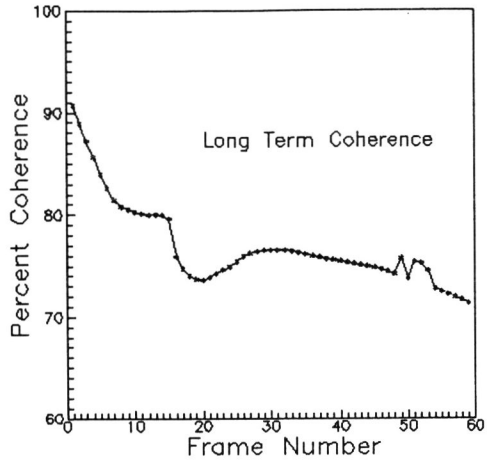

Figure 6. Coherence between first and succeeding frames in commercial animation.

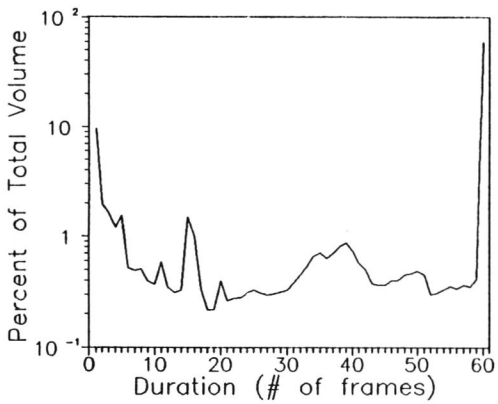

Figure 4. Percent of pixel-event volume .vs. duration.

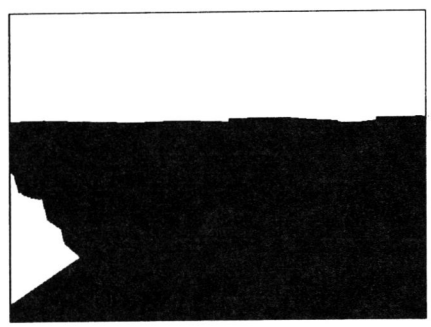

Figure 7. All pixels which change during sample animation sequence depicted in figure 1.

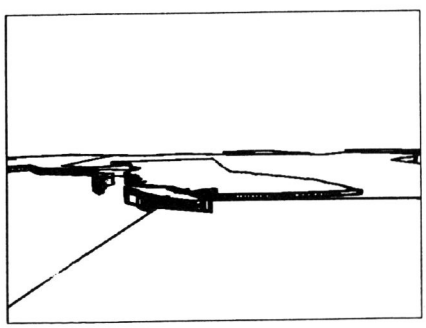

Figure 5. Pixels which differ between figures 1b and 1c.

3. Image Coherence: Frame Differencing

One immediate consequence of viewing ray tracing as a four dimensional rendering process is that generating pixels primarily in the spatial dimension(s) in a temporally sequential manner is no longer a constraint; image generation can occur sequentially in time first and secondarily in space. In other words we may concentrate on the complete rendering of a given pixel at all points in time, exploiting the temporal coherence of the image, before turning to the rendering of spatially adjacent pixels. If we are able to place an upper limit on the maximum visual frequency of phenomena in a sequence of frames then we may make the following assertion: if a pixel **P** has the same value in both frame i and frame $i+n$ (where n, the bin width, is determined from the maximum visual frequency) then **P** must have

the same value in all intervening frames. So only those pixels not meeting this criterion need have their values calculated by actual ray tracing, while the remainder merely have their previous values replicated in frames *i+1* through *i+n-1*. We already make a similar assertion when we select the frame rate at which a given sequence is to be produced. For a given sequence of frames $\mathbf{f}^1$, $\mathbf{f}^2$, we compute the values $\mathbf{p}^1_{i,j}$, $\mathbf{p}^2_{i,j}$, ... , $\mathbf{p}^n_{i,j}$ of pixel $\mathbf{p}_{i,j}$. The time dependent values of all pixels, for n frames, are calculated as follows:

notation:

P_i is the collection of all pixel values for frame i.
P_i [x] is the value of pixel x in frame i.

algorithm:

```
Procedure RenderSequence(n)
1. S1 = { x| x is a pixel in frame 1 }
2. Sn = { x| x is a pixel in frame n }
3. P1 = RayTrace(1, S1)
4. Pn = RayTrace(n, Sn)
5. Search(S1, 1, n)

Procedure Search(S, l, r)
1. IF l=r THEN return
2. S' = S - { x| x∈ S and Pl[x] ≠ Pr[x] }
3. FOR i=l+1 to r-1 DO
4.    FOR each s ∈ S' DO Pi[s] = Pl[s]
5. S'' = S-S'
6. j = (l+r)/2
7. Pj[S'] = RayTrace(j,S'')
8. Search(S'', l, j)
9. Search(S'', j, r)
10. return

Procedure RayTrace(n, S)
1. Ray trace all pixels for frame n that are in set S
2. return
```

Procedure RenderSequence first ray traces every pixel in the first and last frames of the sequence (lines 1-4) and then invokes Search to render intervening frames in the sequence. Search is passed a set S denoting the pixels it may need to render and the numbers of the 'left' and 'right' frames which bound the sequence of frames to be generated. It then constructs the set S' which denotes all the pixels in S whose values do not differ in the left and right frames (line 2). Then these unchanging pixel values are merely copied into the appropriate positions of the intervening frames (lines 3-4). S'', the set of all pixels in S whose values differ in the left and right frames, is constructed (line 5) and these pixels are ray-traced for a frame chosen to lie mid way between the left and right frames (lines 6-7). Search is then recursively invoked, only for those pixels differing between the left and right frames, on the frames between the left and middle frames (line 8) and the frames between the middle and right frames (line 9). In effect we are performing a binary search, in the temporal domain, for the points at which each pixel changes value. Since it is entirely image dependent/driven this method will function correctly even with/during changes to lighting and viewing parameters.

There are implications for an implementation of this algorithm. Care must be taken in choosing the bin width, n; if n is very small then the computational savings may be unnecessarily minimized while if it is too large high frequency phenomena may go undetected, e.g. $P_1=P_nP_i$, $1<i<n$. Additionally, the scene data structure must allow data extraction for any time (frame) in the span to be rendered. This will clearly exact some space penalty although not necessarily one which is significant or prohibitive (see empirical results below). In practice it may be more efficient to treat a whole row, or area, of pixels at a time in this manner, rather than a single pixel at a time, to provide some optimisation in the same spirit as loop unrolling.

4. Algorithm Analysis

A natural question to ask is 'what savings can be expected from this algorithm?'. Performance will be maximized if every frame is identical to its predecessor. In this case the performance ratio P is given by:

$$P = \frac{nt}{2t} = \frac{n}{2} \qquad (1)$$

where t is the time required to render a single frame in it's entirety. However, normally the bin width will be less than the total number of frames to be rendered. In this case the performance ratio will be:

$$P = \frac{Nt}{\left(1+\frac{N-1}{n}\right)t} = \frac{N}{1+\frac{N-1}{n}} \qquad (2)$$

assuming N, the total number of frames, is *one more than* an exact multiple of n. P will tend to n as N grows large. It is easily seen that even for small n the performance increase is significant. A more realistic assumption is that all frames are not identical. If we let d denote the mean value of the fraction of pixels which differ between any pair of frames then the relative work for n frames is $1+(n-2)d+1$, i.e. we ray-trace all pixels in the first and n^{th} frames and an average of d of the pixels in each of the intervening frames. If we also remove the constraint of $N-1$ being an integral multiple of n, equations 1 and 2 become:

$$P = \frac{nt}{(2+(n-2)d)t} = \frac{n}{2+(n-2)d}$$

and

$$P = \frac{Nt}{(a+b)t} = \frac{N}{a+b} \qquad (3)$$

where $a = 1 + \left\lfloor \frac{N-1}{n-1} \right\rfloor (1+(n-2)d)$

and

$$b \equiv \begin{cases} 0 & \text{if } rem\left(\frac{N-1}{n-1}\right) = 0 \\ (rem\left(\frac{N-1}{n-1}\right) - 1)d + 1 & \text{if } rem\left(\frac{N-1}{n-1}\right) \neq 0 \end{cases}$$

respectively, giving the expected average performance ratio. The part of the denominator in equation 3 involving the floor function accounts for the portion of the sequence which is an integral multiple of the bin width; the remaining portion of the sequence must be rendered at a smaller bin width which is accounted for by the term involving the remainder function. Figure 8 is a graph of equation 3 for several bin widths using a fixed value of N=100; the rapid change of slope at some points is produced by the floor term described above.

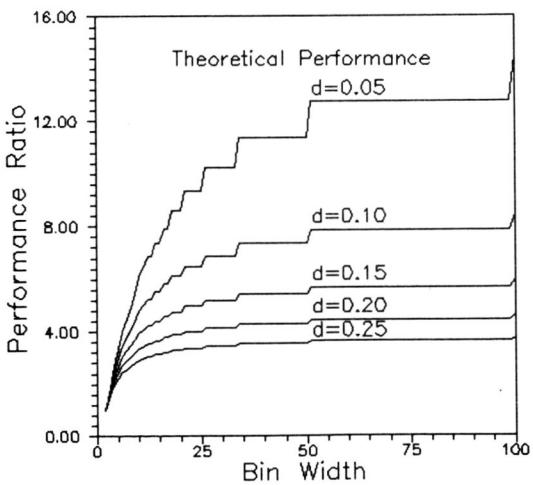

Figure 8. Performance ratio .vs. bin width for 100 frame sequence. d = 1 - (average coherence between any two frames).

An obvious question is how to choose n, the bin width. Its value depends on the visual frequencies in the image sequence to be generated. If n is too large visual phenomena may not be rendered at all. A pathological case is that of an object occupying a field of view equivalent to a single pixel in the generated images. Suppose the object occupies pixel $\mathbf{P}_{i,j}$ in the first frame, pixel $\mathbf{P}_{i+k,j+l}$ in frame n, and moves linearly between them in the intervening frames. Using a bin width of n the algorithm will detect the changes at pixels $\mathbf{P}_{i,j}$ and $\mathbf{P}_{i+k,j+l}$ and so (after searching) will correctly render $\mathbf{P}_{i,j}$ and $\mathbf{P}_{i+k,j+l}$ in the intervening frames. However, while all the pixels along the path from $\mathbf{P}_{i,j}$ to $\mathbf{P}_{i+k,j+l}$ will (should) change value at some time between frame 1 and frame n the algorithm will find that these pixels have the same value at the boundary frames and so will not search the intervening frames thus producing an incorrect result. This is precisely an aliasing problem produced by too low a sampling rate of the image sequence.

There are several feasible solutions to this problem. At video frame rates it may be possible to use small values for the bin width and accept the errors since they will occur over such a short period of time that they may well be unnoticed by the viewer. Another possible solution is to try and construct an envelope around each (spatial) group of changing pixels and re-render all pixels within the envelope. A more promising alternative is to determine which parts of the image are changing rapidly and to use different bin widths for different portions of the image. The images could first be rendered using a simple, but rapid, scan line algorithm and then efficiently compared to determine rates of change in the image sequence. Alternatively the images could first be rendered using a standard ray tracer (or beam tracer) but on a coarser raster and again a determination would be made as to rates of change for sub-regions of the image sequence. If the images were ray traced on a raster with one eighth the horizontal and vertical resolution of the desired final images (one pixel corresponding to an eight by eight tile of the original raster) most phenomena of the type described above would be recognized (subject to normal ray tracing aliasing problems) and yet the total rendering time would be increased by less than 2%. This is similar in nature to the multi-grid method, used to find numerical solutions for systems of differential equations, due to Brandt.[17]

5. Empirical Results

The basic algorithm described above has been implemented on a variety of machines including Sun workstations and a Silicon Graphics IRIS 4D. A hierarchical data structure is employed to represent the model. Each primitive object, or group of objects, which moves has three associated polynomials, one for each spatial dimension, describing the movement of the object(s) with respect to time. The coefficients of these polynomials are stored in the appropriate positions of the data structure. When it is desired to render pixels for time t a procedure traverses the data structure evaluating the polynomials and adjusting the data structure appropriately. In this manner the model is

input only once, regardless of the number of frames to be generated, and multiple copies of the data structure are not required. Even if every primitive object in a scene moves independently (thus requiring a unique set of coefficients for each object) the space penalty is neither prohibitive nor a function of the number of frames to be rendered.

A series of three test animations were constructed using three different models in three-space: a) a sphere, background polygon and single light source; a dozen spheres of varying colour and size with a background polygon and single light source; c) a terrain composed of 20,000 polygons and a single background polygon. The three resulting test animation sequences were: a) the single sphere moving toward the viewer and diagonally right and downward with a mean frame-to-frame coherence of 95.06 (minimum of 91.57; maximum of 97.48) and a mean long term coherence of 92.55 (minimum of 88.41; maximum of 97.48); b) the dozen spheres moving with random trajectories and velocities from a loose cluster near the centre of the modelled space (see figure 2) with a mean frame-to-frame coherence of 86.03 (minimum of 81.71; maximum of 93.77) and a mean long term coherence of 70.47 (minimum of 60.11; maximum of 84.4); c) the 'camera' position following a trajectory over the terrain with a mean frame-to-frame coherence of 81.77 (minimum of 80.29; maximum of 82.4) and a mean long term coherence of 40.99 (minimum of 38.48; maximum of 80.29);

These sixty frame animation sequences were repeatedly rendered using a number of bin widths. Figure 9 shows the results given as a function of bin width. Column two shows the relative (compared to traditional frame-by-frame ray tracing) cpu time required. Column three is the theoretical lower bound on the relative cpu time as determined by equation 4 with d derived from the animations; column four is the ratio of actual performance ratio to the theoretical performance ratio given by equation 4. The difference between the actual and predicted performance ratios can be accounted for, at least in part, by the fact that equation 4 makes the (clearly invalid) assumptions that the manipulation of sets and replication of pixel values require zero time and that the time required to ray-trace a pixel is constant over all pixels in the scene. Column five is the fraction, of the total number of pixels, that were actually ray traced; column six is the total number of pixels whose value differs from that produced by the traditional method expressed as a percentage of the total number of pixels in the sequence - note that, even for the worst case (in figure 9c), less than three percent of the pixels differ from those in the "correct" sequence. Figure 10 plots both the measured and predicted relative cpu times as well as the measured percentage of pixels in error (columns two, three and six of figure 9) as a function of bin width.

In an informal experiment the animation sequences for bin widths up to ten were videotaped and shown to an audience. Each sequence was paired with the "correctly" rendered sequence, i.e. rendered with a bin width of two. The order of the sequences within each pair was chosen randomly and the audience viewed the resulting twenty-seven pairs in a random order to prevent subjects from discerning any trend. The subjects were told nothing about the the sequences (other than that they would be shown in pairs) and were asked to (individually) indicate on a questionaire whether the two animation sequences in each pair appeared to be identical. Figure 11 is a table of the results obtained from this experiment; the second, third and fourth columns correspond to the first, second and third animation sequences. The first number in each entry denotes the number of individuals reporting the sequences as identical while the second denotes the number reporting a perceived difference; the number of responses reported in each cell of figure 11 is not constant (over the table) due to the fact that some individuals reported no choice for a particular pair. It is of interest to note the number of subjects reporting the three pairs of identical sequences (bin width of 2) as being non-identical.

Bin #	Rel. CPU	Theor-etical	CPU/ Theo.	Pixels Traced	% Error
2	1.0	1.0	1.0	1.0	0.0
3	0.43	0.54	0.79	0.54	0.02
4	0.31	0.38	0.81	0.39	0.042
5	0.26	0.3	0.85	0.31	0.05
6	0.21	0.26	0.84	0.26	0.063
7	0.2	0.22	0.89	0.23	0.075
8	0.18	0.21	0.87	0.22	0.081
9	0.18	0.19	0.94	0.2	0.089
10	0.17	0.18	0.95	0.19	0.1

Figure 9a. Results for first animation (d = 0.05).

Bin #	Rel. CPU	Theor-etical	CPU/ Theo.	Pixels. Traced	% Error
2	1.0	1.0	1.0	1.0	0.0
3	0.58	0.58	1	0.6	0.036
4	0.45	0.44	1	0.47	0.074
5	0.39	0.37	1.1	0.4	0.1
6	0.37	0.33	1.1	0.36	0.14
7	0.33	0.3	1.1	0.34	0.18
8	0.32	0.28	1.1	0.33	0.23
9	0.33	0.27	1.2	0.31	0.29
10	0.29	0.25	1.2	0.3	0.59

Figure 9b. Results for second animation (d = 0.14).

Bin #	Rel. CPU	Theor-etical	CPU/ Theo.	Pixels Traced	% Error
2	1.0	1.0	1.0	1.0	0.0
3	0.74	0.6	1.2	0.68	0.1
4	0.7	0.47	1.5	0.6	0.26
5	0.67	0.4	1.7	0.55	0.49
6	0.67	0.36	1.9	0.53	0.77
7	0.65	0.33	2	0.52	1.2
8	0.64	0.32	2	0.5	1.6
9	0.65	0.3	2.2	0.49	2
10	0.63	0.29	2.2	0.48	2.4

Figure 9c. Results for third animation (d = 0.18).

Figure 9. Test animation measurements.

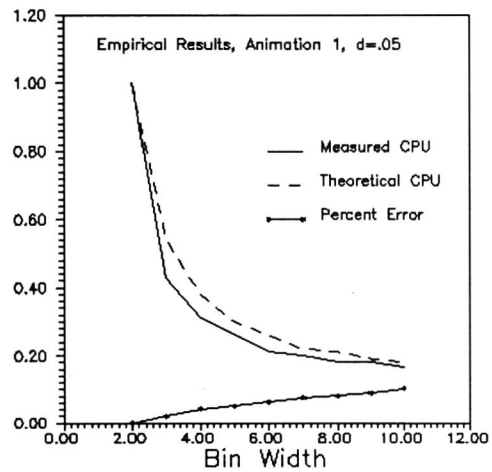

Figure 10a. Results for first animation.

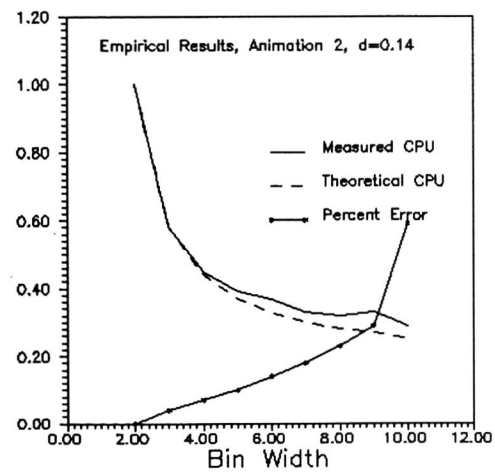

Figure 10b. Results for second animation.

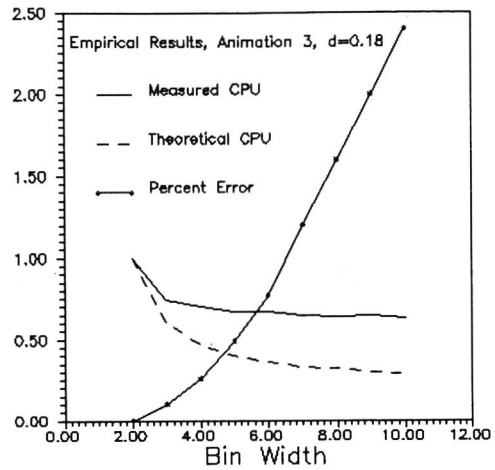

Figure 10c. Results for third animation.

Figure 10. A comparison of empirical and predicted performance increase for three test animations.

Bin #	Animation 1	Animation 2	Animation 3
2	14,3	12,4	11,6
3	12,4	13,4	14,3
4	14,3	12,5	8,8
5	12,5	8,9	12,5
6	12,5	14,3	12,5
7	14,3	10,7	10,7
8	13,4	10,7	10,7
9	11,6	9,8	9,8
10	5,12	8,9	14,3

Figure 11. Audience survey results.

6. Discussion and Summary

Concurrent with the research presented in this paper, Badt[18] described an image space algorithm similar in spirit to the algorithm presented here but stochastic in nature. Unfortunately, since Badt provides no theoretical analysis and did not implement his proposed algorithm (and thus provides no empirical data) it is difficult to compare the two approaches. However, one feature of his algorithm is that it would seem to require that both the values of all pixels for all frames and the data structure representing the model be capable of being sampled/examined on an essentially random basis with respect to both spatial and temporal coordinates. This implies a rather large space penalty for an actual inplementation of Badt's algorithm. In contrast the algorithm presented here imposes negligible space penalties since, while frames are not generated in sequential order, all the pixels for any given frame are generated as a group before pixels for another frame are produced.

Empirical evidence has been presented which demonstrates that substantial performance increases may be obtained by an algorithm which can successfully exploit image space coherence. A candidate algorithm has been presented and an empirical and theoretical analysis performed. Ray-tracing a sequence of images has been divided into two disjoint tasks: 1. ray-tracing a minimal set of pixels and 2. determining which pixels can be excluded from that set. The latter problem is potentially of lower computational complexity than simply determining the correct value of the pixels concerned. The algorithm presented can also be applied to rendering schemes other than ray tracing. When applied to ray tracing the results show that significant performance increases can be obtained with this algorithm. Although it is possible for the algorithm, as implemented, to produce "incorrect" results, in some situations, several potential solutions have been postulated. Ongoing research also includes the investigation of algorithms designed to directly exploit spatio-temporal coherence in ray-object intersection calculations.

7. Acknowledgements

The authors wish to express their gratitude to Marcus Tessman and Icon Computer Graphics Ltd. of Vancouver for providing the authors with raw data in the form of sequences of commercial animation produced by Icon Computer Graphics.

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 CGA*, 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. Brandt, "Multi-level adaptive solutions to boundary value problems," *Mathematical Computation*, vol. 31, no. 138, pp. 333-390, 1978.
18. Sig Badt Jr., "Two algorithms for taking advantage of temporal coherence in ray tracing," *The Visual Computer*, no. 4, pp. 123-132, 1988.