Real-Time Hidden-Line Elimination for a Rotating Polyhedral Scene Using the Aspect Representation*

Harry Plantinga
Department of Computer Science
University of Pittsburgh

Charles R. Dyer
W. Brent Seales
Department of Computer Science
University of Wisconsin

Abstract

In this paper we address the problem of the real-time display of line drawings of a polyhedral scene with hidden lines removed, as the scene rotates or the viewpoint moves. We use frame-to-frame coherency to formulate an algorithm for efficiently computing and displaying the rotation along a path of viewpoints. The algorithm precomputes the viewpoints at which the appearance changes and the changes in appearance that occur, using the *aspect representation*. It then displays the views in real time by updating the current view to get a new view. The on-line display phase is about as fast as displaying a rotating wire-frame model of the scene. Thus high-speed animation with hidden line removal can be achieved for scenes of moderate size and visual complexity on general-purpose workstations.

1. Introduction

The three-dimensional structure of an object is easier to perceive when the object is rotating or when the viewer moves to see the object from a range of viewpoints. Thus, CAD systems are usually able to display an object as it rotates. Displaying a scene from a series of viewpoints (or "animating rotation") is also important to graphical simulations such as flight simulators. In problems such as these, real-time display and hidden-line or hidden-surface removal for complex scenes are desirable but sometimes incompatible goals.

In this paper, we discuss the problem of displaying from a moving viewpoint a series of line-drawing images of a polyhedral object or scene with hidden lines removed. We will refer to this as the problem of animating rotation. We present an algorithm for animating the rotation of a polyhedral scene that displays frames at a rate roughly equivalent to the rate at which the same hardware could display line-drawings of the scene without hidden-line removal. We consider here only the case of a rotation about one axis of the coordinate system under orthographic projection.

The naive approach to animating rotation is to treat frames independently. For each frame, hidden lines are removed and the frame is displayed. The real-time goal is to do this at video rates, such as 30 frames per second. However, since the viewpoints are closely spaced, there will be little difference between two successive frames and repeated hidden-line removal may be superfluous. This *frame-to-frame coherence* makes it possible to devise a more efficient algorithm.

The algorithm presented here takes advantage of frame-to-frame coherence by computing the initial appearance of the scene in the first frame and the viewpoints at which the scene changes substantively, that is, at which the topological structure of the image changes. The viewpoints at which the appearance changes substantively are computed through the construction of the *aspect representation* for the scene, a representation that makes explicit exactly which vertices, edges, and faces are visible from all viewpoints. The algorithm has two phases: a preprocessing phase, in which the initial appearance of the polyhedron and the events are computed and the visible edge graph is constructed; and an on-line phase, in which a sequence of frames is displayed in real time.

Section 2 discusses frame-to-frame coherence and the sorts of events that change the edge graph of the image. Section 3 shows how to compute and store these events, and Section 4 describes how to use this information to animate rotation. Section 5 describes the implementation and results, and Section 6 discusses related work.

2. Coherence and Events

We will restrict the discussion to the rotation of an object or scene by θ about the world coordinate system y-axis (see Figure 1), viewed under orthographic projection. Other rotations can be handled by a coordinate transformation. We will assume that no object edge is parallel to the plane of rotation.

*The support of the National Science Foundation under Grant No. IRI-8802436 is gratefully acknowledged.

(In practice, such edges can be handled by changing the axis of rotation infinitesimally.) We will say that a vertex and an edge *appear to intersect* from a given viewpoint when their projections are visible and intersect in an image from that viewpoint. We will refer to the apparent intersection point of two edges in an image as a *T-junction.*

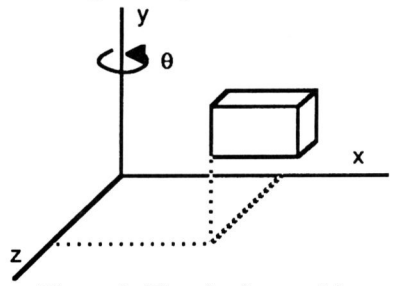


Figure 1. The viewing model.

Frame-to-frame coherence means that images from nearby viewpoints are "similar." In order to formalize this notion of similarity, we define the *Edge Structure Graph* (ESG) of an image from a given viewpoint. The ESG from a viewpoint is the graph with edges and vertices corresponding to edge segments and intersection points of edge segments, respectively, in an image from that viewpoint. In addition, for each edge in the ESG, the coordinates in $\mathbf{R}^3$ of the endpoints of the corresponding object edge are stored. If the image edge ends in a T-junction, a pointer to the occluding edge is stored.

As the viewpoint changes between two values or, more generally, traces out a path in viewpoint space, the appearance of the object changes. These changes may consist of a linear change in the location of edges and vertices or a structural (topological) change, that is, a change in the ESG. We are interested in the viewpoints that have the property that an arbitrarily small change in viewpoint along the path will result in a change in the ESG. These viewpoints we call *events* after the visual events of Koenderink and van Doorn [1979].

The event viewpoints are the viewpoints from which either an object vertex and a non-adjacent edge appear to intersect (*EV-events*) or three object edges appear to intersect at an image point (*EEE-events*). To see that this is true, suppose that a given viewpoint is an event and that no non-adjacent vertex-edge pair appears to intersect. The change in the ESG must occur at an image vertex. Image vertices are a result of object vertices or T-junctions, and if the T-junction is the apparent intersection of only two object edges, the structure doesn't change with a sufficiently small change in viewpoint. Since the image structure does change with an arbitrarily small viewpoint change, at least three object edges must appear to intersect at a point somewhere in the image.
Now suppose that an object vertex and non-adjacent edge appear to intersect from a given viewpoint. Since we are assuming that none of the object edges is parallel to the plane of rotation, rotating slightly in some direction will separate the edge and the vertex, resulting in one of the following cases. If the vertex is in front of the edge from the given viewpoint, one of the events shown in Figure 2 occurs. If the vertex is behind the edge from the given viewpoint, one of the events shown in Figure 3 occurs. Suppose that three object edges appear to intersect. Let the first and second be arbitrary. Depending on the location and orientation of the third, one of the events shown in Figure 4 occurs. If more edges or vertices meet at a viewpoint, the event can be treated as two more events from infinitesimally different viewpoints.

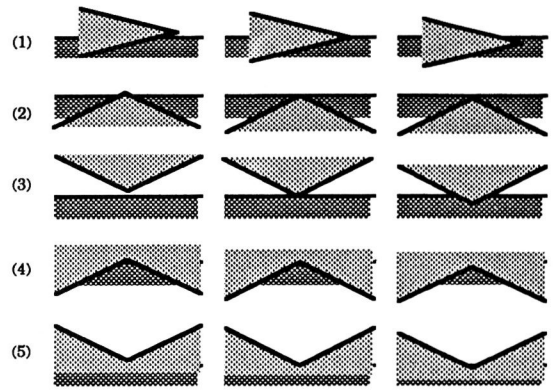


Figure 2. EV events with the vertex in front of the edge. In each row, the center figure shows the image from the event viewpoint and the left and right figures show the appearance on either side.

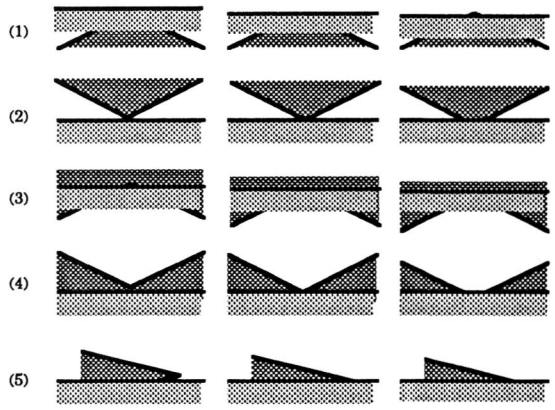


Figure 3. EV events with the vertex behind the edge.

3. Computing Events

In order to animate rotation, the viewpoints at which these changes occur must be computed, and the change in the ESG that occurs must be represented. To that end, note that all of the events shown above arise from the apparent intersection of three object edges. In order to find events, it is therefore sufficient to take every set of three edges and find the viewpoints on the path from which they appear to intersect. We will call these viewpoints *potential events*; they are events if they are visible in the image. It remains to determine whether the apparent intersection point is visible in the image. This

could be computed for each potential event individually. However, it is usually true that most of the potential events are not visible. We now describe a more efficient approach.

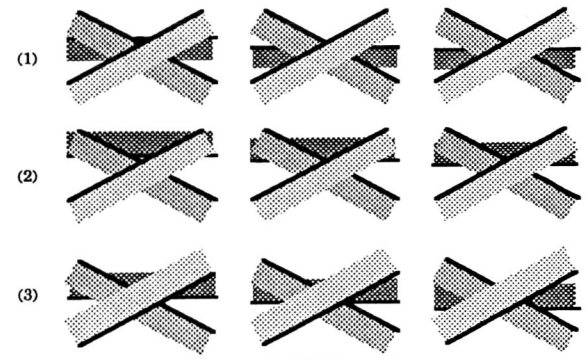


Figure 4. EEE events.

The approach we take is to define *aspect space* to be image space × viewpoint space. In the present case, in which the rotation is about a single axis, we can consider viewpoint space to be S^1. Since image space is a plane, aspect space is $R^2 \times S^1$. We can represent this as a subspace of R^3, specifically $R^2 \times (-\pi,\pi]$. A point in aspect space then corresponds to a point in the image plane from a particular viewpoint. In addition, a point (x_0,y_0,z_0) in object space is represented by a 1-D locus in aspect space since it is visible from all viewpoints. This locus can be computed from the equations for the location of the point in image space after a rotation by θ about the y-axis. Denoting coordinates in the image plane (u,v), the result is

$$u = x_0 \cos\theta - z_0 \sin\theta \quad (1)$$

$$v = y_0 \quad (2)$$

Thus, a point in object space is represented in aspect space by the 1-D locus $(u(\theta),v,\theta)$, $-\pi < \theta \leq \pi$.

An edge of the object connecting vertices $\mathbf{p}_1 = (x_1,y_1,z_1)$ and $\mathbf{p}_1 + \mathbf{a}_1 = (x_1 + a_1, y_1 + b_1, z_1 + c_1)$ can be represented parametrically as $\mathbf{p}(s) = \mathbf{p}_1 + s\,\mathbf{a}_1$, $0 \leq s \leq 1$. It appears in the image at the points

$$u = (x_1 + a_1 s)\cos\theta - (z_1 + c_1 s)\sin\theta \quad (3)$$

$$v = y_1 + b_1 s \quad (4)$$

Thus, an edge in object space is represented in aspect space by the 2-D locus $(u(s,\theta),v(s),\theta)$.

We are interested in computing the appearance of an object face from all viewpoints. If the face is not occluded from any viewpoint, it can be modelled as volume of aspect space. We call that volume the *aspect representation* or *asp* for the face [Plantinga and Dyer, 1986, 1990; Plantinga, 1988]. The asp for a polygon is a volume bounded by surfaces of the form of Eqs. (3) and (4) and edges of the form of Eqs. (1) and (2). Figure 5 shows a part of the asp for a triangle in the xy-plane. If the face is occluded, not all of the face is visible from all viewpoints. Then to represent the appearance of the face from all viewpoints part of the asp for the face must be removed. We would like to determine the appropriate volumes to subtract for each occluding face.

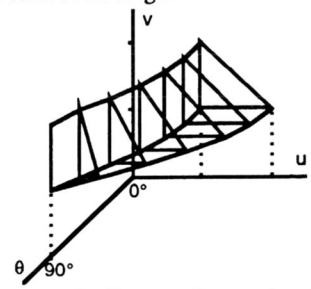


Figure 5. The asp for a triangle.

We make use of the following property of occlusion. We will assume that the faces of the scene are oriented, with the *front* of the face visible (on the outside of the object) and the rear of the face invisible. We will say that a face occludes another face when there is a line of sight that passes through the front of the occluding face and hits the front of the occluded face. Let **f** be an oriented face of a scene. Let **P** be the plane containing **f**, and let the halfspace with respect to **P** on the front side of **f** be $\mathbf{f}^+$ and the other halfspace $\mathbf{f}^-$. The faces that **f** occludes from some viewpoint are precisely the faces (or parts of faces) in $\mathbf{f}^-$, and no face or part of a face in $\mathbf{f}^-$ occludes **f** from any viewpoint.

The algorithm for computing the events rests on the following observation: the appearance of a face **f** *as occluded by* the faces in front of the plane containing **f** is characterized by the subtraction of the asps for the occluding faces from the asp for **f**. This is true since if a point in aspect space is a part of the asp for **f** and an occluding face, the occluding face occludes **f** at that image point and viewpoint since it is in $\mathbf{f}^+$.

To construct the asp for a face **f** of a scene as occluded by the other faces, the asp for each face (by itself) is first constructed. The form of the asp for a face is displayed graphically in Figure 5. It consists of a surface for each edge of the face, represented with constants determining the surface and pointers to the bounding curves. The curves are represented with constants determining the curve and pointers to the bounding vertices. In addition, the surfaces intersect in a line at the viewpoint from which the face appears edge-on. Since a face by itself is not occluded from any viewpoints, there are no other bounding edges or vertices.

The next step in constructing the asp for **f** is to subtract the asps for the faces in front of **f** from the asp for **f**. (If a face intersects the plane containing **f**, the asp for the part of the face in $\mathbf{f}^+$ is subtracted.) The remaining volume of the asp for **f** consists of image points and viewpoints at which the face **f** is not occluded, i.e. appears in the image.

Note that the faces of the asp for a polygon are not planar (see Figure 5). They are ruled quadric surfaces [Gigus *et al.*, 1988]. Asps can be represented as polyhedra except that the

constants for the curved surfaces must be represented. In order to find the intersection of volumes bounded by such surfaces, we must also know how to find the intersection of the surfaces. Below we solve for the intersections of asp surfaces. Then finding the intersections of asps is similar to finding the intersections of polyhedra, except that intersections of ruled surfaces must be found. These intersections are found by substituting the stored constants into the equations below, yielding new constants.

A volume of aspect space corresponds to a polygon in the image, and it is bounded by surfaces corresponding to edges in the image. The surfaces are of the form given by Eqs. (3) and (4) above. The intersection of two of these surfaces corresponds to the apparent intersection of two edges in the image, that is, a T-junction. If the two edges are $\mathbf{p}_1 + s_1\mathbf{a}_1$ and $\mathbf{p}_2 + s_2\mathbf{a}_2$, solving for s_2 as a function of θ and for s_1 as a function of s_2 yields

$$s_1 = \frac{b_2 (x_2-x_1) - a_2 (y_2-y_1) - (b_2 (z_2-z_1) - c_2 (y_2-y_1)) \tan\theta}{(b_2 a_1 - a_2 b_1) - (b_2 c_1 - c_2 b_1) \tan\theta} \quad (5)$$

Substituting this expression into Eqs. (3) and (4) yields the image point at which the edges appear to intersect as a function of θ. Thus, $(u(\theta),v(\theta),\theta)$ is the curve in aspect space corresponding to the intersection of two asp surfaces, and it is the general form for 1-D boundaries of the asp.

EEE visual events result from the apparent intersection of three object edges. Three edges can appear to intersect in a single point from two viewpoints and their polar opposites along the path. The viewpoints can be found by noting that if the line of sight passes through the point $\mathbf{p}_1 + s_1\mathbf{a}_1$ and the other two edges, it must be the intersection of the planes defined by that point and the other two edges (see Figure 6).

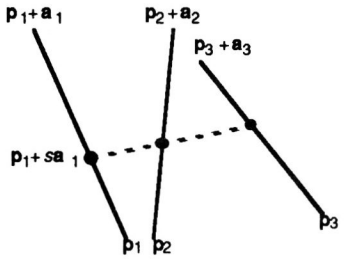


Figure 6. The viewpoints from which three object edges appear to intersect in a single image point.

Thus, the direction of the line of sight from which these edges appear to intersect at the given point is

$$\mathbf{d} = ((\mathbf{p}_1 + s\,\mathbf{a}_1 - \mathbf{p}_2) \times \mathbf{a}_2) \times ((\mathbf{p}_1 + s\,\mathbf{a}_1 - \mathbf{p}_3) \times \mathbf{a}_3) \quad (6)$$

This is a quadratic equation in s. Since lines of sight in this problem are parallel to the xz-plane, the viewing direction is constrained to lie in the xz-plane. Thus, we can set the y-component of Eq. (6) to 0 and solve for s to get the two viewpoints (and their polar opposites) from which three lines appear to intersect in a single point. Substituting s into Eqs. 1-3 yields the asp vertices resulting from the intersection of three general asp surfaces.

EV visual events result from the apparent intersection of an object edge and a non-adjacent object vertex. Since the plane containing the vertex and the edge intersects the plane of rotation in a line, there is a single viewpoint (and its polar opposite) from which the vertex and edge appear to intersect. Since they intersect in a single image point, the event is represented by a point in aspect space. This kind of event is generated by the intersection in aspect space of the surface corresponding to the edge and the curve corresponding to the point, yielding a point in aspect space. Solving Eqs. (1)-(4) for s and θ we see that the point is given by

$$s = \frac{y_0 - y_1}{y_2 - y_1} \quad (7)$$

$$\tan\theta = \frac{x_0 - (x_1 + a_1 s)}{z_0 - (z_1 + c_1 s)} \quad (8)$$

together with Eq. (1). This event is a special case of EEE events in which two of the edges intersect.

Since there is a constant number of asp vertices for every set of 3 object edges, the number of vertices is bounded by $O(\mathbf{n}^3)$ where $\mathbf{n}$ is the number of edges in the polyhedral scene. These are the vertices of a 3-manifold in 3-space, so the number of edges, faces, and cells is also bounded by $O(\mathbf{n}^3)$ by Euler's formula. Let $\mathbf{q}$ be the size of the asp for a face $\mathbf{f}$ and note that the asp for a face of size $\mathbf{r}$ by itself is $O(\mathbf{r})$. The time to subtract the asp for a face from the asp for $\mathbf{f}$ is $O(\mathbf{qr})$. Thus, the time to construct the asp for the face is $O(\mathbf{q}\ (\Sigma\mathbf{r})) = O(\mathbf{qn})$. The time for the construction of the asp for the whole scene is $O(\Sigma\mathbf{qn}) = O(\mathbf{n}^4)$. These are worst case times; we will argue that for many polyhedra the times are much better. For example, the convex case requires less time: there are no faces in front of any other face, so there is no subtraction of cells to be done. If the polyhedron is not known in advance to be convex, the naive algorithm for finding all of the faces in front of each face takes time $\Theta(\mathbf{n}^2)$; thus, the asp construction algorithm takes time $\Theta(\mathbf{n}^2)$.

4. Animating Rotation

The procedure for animating rotation involves computing and storing all events as well as the changes in the ESG that occur at each event. For example, suppose that the object in Figure 7 is to be rotated. To display the first frame the ESG must be computed from the initial viewpoint using a standard hidden-line removal algorithm. The algorithm must be modified slightly to report occluding edges at T-junctions. Subsequent images are then displayed by determining whether any events are crossed by the change in viewpoint, and if so, updating the ESG.

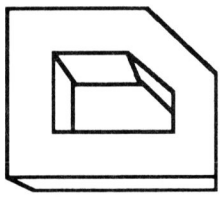


Figure 7. The object to be rotated.

In order to perform these operations, two data structures are maintained: the Event List and the ESG. The ESG consists of a graph of object edges, vertices, and T-junctions as described above. The Event List is a list of events sorted by viewpoint along the viewpoint path. With each event is stored a list of pointers to edges that appear in the ESG and a list of pointers to edges that disappear. Figure 8 shows the ESG for the object (with T-junctions circled) together with a representation of the sorted list of event viewpoints. Pointers from an event to the edges that appear in the ESG are represented pictorially. Figure 9 shows the ESG after the viewpoint has passed the event and the two edges have been added to the ESG.

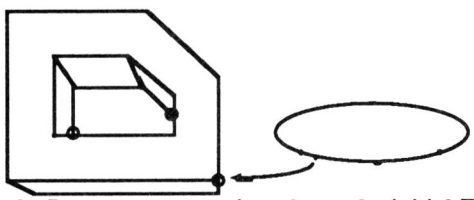


Figure 8. In the preprocessing phase, the initial ESG of an object is computed and the visual events along the path of viewpoints are computed and stored together with pointers to edges that appear and disappear in the ESG. The circled vertices are T-junctions.

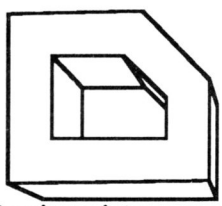


Figure 9. When the viewpoint passes events, the image is updated and displayed from the new viewpoint. In this case, two new edges have been added to the ESG.

To construct the event list, we first construct the asp for the polyhedral scene. The asp vertices correspond to events, so for each vertex in the asp we add an event for the viewpoint of that vertex and add appropriate lists of edges to be removed and added from the ESG. When all of the events are found, they are sorted by viewpoint along the path and put into a list.

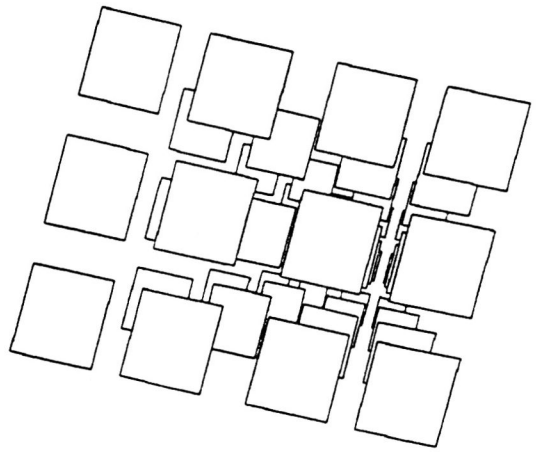
The on-line phase of animating rotation involves keeping track of the current ESG as the viewpoint changes. Each time the viewpoint crosses an event, the ESG is updated by deleting and adding the specified edges. Each update requires linear time in the number of edges added and removed, which is usually less than a small constant. Displaying a frame is then about as fast as displaying a wire-frame object; the only difference is that the image location of T-junctions must be computed by finding the intersection point of the two (rotated) lines. However, there are usually fewer visible lines to display for simple objects, since as many as about half of the lines may be hidden and are not drawn (though for very complex objects there may be *more* line segments to display than edges in the object). Thus, if the number of events is no more than the order of the number of frames to be displayed (as is the case for scenes of moderate complexity and moderate rotation rates), the display time is roughly equal to the time required to display a series of wire-frame views of the scene. Furthermore, the algorithm can take full advantage of 2-D and 3-D vector display hardware.

The preprocessing time required is essentially the time to construct the asp for the scene and to sort the viewpoints at which events occur. Sorting the events takes less time than constructing the asp. Constructing the asp takes time $\Omega(\mathbf{n}^2)$ to $O(\mathbf{n}^4)$, depending on the number of visual events in the scene, where $\mathbf{n}$ is the number of faces in the scene. A large class of scenes (probably most models of real-world scenes) have $O(\mathbf{n}^2)$ visual events, and the asp in this case can be constructed in $\Omega(\mathbf{n}^2)$ to $O(\mathbf{n}^3)$ time. In the results section below, we show that for a number of scenes of moderate size and visual complexity, the number of events is less than 7 times the number of edges.

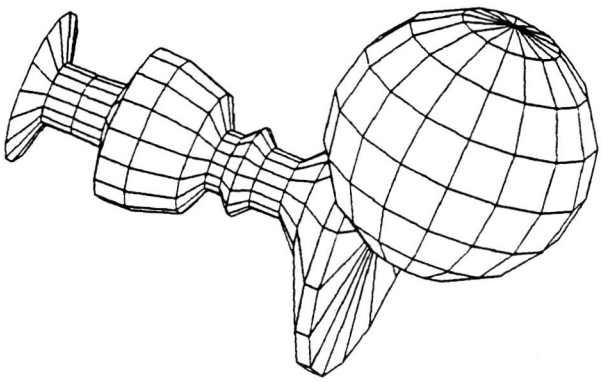
This algorithm requires storage for every visual event along the path of viewpoints. Thus, the amount of storage required for events will be between $\Omega(\mathbf{n})$ and $O(\mathbf{n}^3)$. The actual amount required varies according to the visual complexity of the scene. For convex objects, the amount of storage required is linear in the number of faces of the object. For the worst case of visual complexity, the storage requirements may become impractical when the number of faces is less than 1000; as few as 600 faces may result in about 1,000,000 visual events for scenes of the highest-possible visual complexity [Plantinga, 1988]. In practice, common objects have a visual complexity much closer to the convex case than to the worst case, unless the scene modelled is visually very complex, as for example a picket fence in front of a trellis. The results of the implementation below suggest that the storage requirements are reasonable for scenes of moderate size and visual complexity. In fact, some of the storage may be slower than main memory because the memory in active use at one time is only about as much as the memory required for one image and a small number of events.

5. Results

A prototype of both the preprocessing and the on-line portions of this algorithm has been implemented. The prototype program is written in C and was tested on a DECstation 3100 workstation. The algorithm was tested on several scenes; two examples are shown in Figure 10.



Model 1



Model 2

Figure 10. Two polyhedral scenes of moderate visual complexity.

Timing values and sizes for the construction phase of the algorithm for the six models are shown in Figure 11. The numbers of faces, edges, and vertices of the original models are recorded in the first three columns of the tables. For each of the selected model orientations, the construction time (in CPU seconds) and the resulting size (in Kbytes) are shown.

Model	Vertices	Edges	Faces
1	192	192	48
2	388	800	416

Model	orient.	Time (sec)	Size (KB)	Max events per edge	Ave events per edge	Total Events
1	-30	11	35	32	16.7	3214
	0	9	31	31	14.9	2870
	30	12	36	39	17.2	3300
2	-30	413	168	33	6.8	5479
	0	437	176	32	7.2	5761
	30	268	144	32	5.8	4650
	base	95	47	16	2.3	1866

Figure 11. Asp construction information

Several conclusions can be drawn from the data shown in Figure 11. First, the construction times and sizes indicate that the construction of the asp representation for models with many more faces is indeed tractable. Although the off-line computation time for the largest model was approximately seven minutes, this absolute time is highly dependent on both the prototype program coding efficiency and the hardware configuration. We have estimated that the asp representation for models with at least an order of magnitude more faces can be computed and stored by using more efficient program coding. Second, the number of visual events that occurs in a typical scene is only a small constant times the number of edges in the model. The models in the upper table were constructed to illustrate the potential worst-case behavior of the asp construction algorithm. In all models, the largest number of events per edge encountered was 17.2. Extrapolating from the models tested, 10 MB is sufficient to store the events for a scene with 20,000 to 180,000 edges. However, since the number of events may be worse than linear in scene size, the maximum possible scene size for 10 MB of storage will depend on the visual complexity of the scene and may be much smaller for complex scenes. Still, the storage requirements appear to be practical for scenes of moderate size and visual complexity.

In order to measure the display rate of each of the test models, we measured the time needed to display a sequence of 360 frames. The timing results include the time to update the event structure as well as the time spent on computing the image coordinates of each of the vectors to be displayed. A complete asp and animation sequence was computed for the models in each of three orientations: -30°, 0°, and 30°. The 0° orientation, chosen to illustrate a significant amount of occlusion, is the orientation pictured in Figure 10. For the second model a baseline orientation was also chosen, in which no occlusion occurs throughout the animation sequence.

The results of these experiments for 360-frame rotations in which the angle between frames varies from 1° to 17° are as follows: for the first model between 1 and 2 seconds were required in all orientations, or 180-360 frames were computed per second. For the second model, between 6 and 8 seconds were required in all orientations, or 45-60 frames per second. These results lead to several observations. First, a change in the number of degrees between frames in the animation sequence does not greatly affect the display rate. This indicates that the time needed to compute the image coordinates of the segments to be displayed greatly dominates the time needed to update the event structure. Thus processing a larger number of events between frames has little effect on the frame rate. Second, the display times for each of the three orientations of each model are very similar. Although the amount of occlusion varies from the base orientation to the other orientations, the stable display time suggests again that the large number of visible segments to be computed dominates the overall frame rate.

In order to quantify these observations we have isolated the percentage of total display time spent on processing visual events. The total time for visual event processing includes time spent both scanning the event list and maintaining the ESG. For each model the time spent on event processing is less than 5% of the total time for increments of less than 15° and less than 1% of the total time for increments of less than 2°. As expected, larger increments between frames imply less coherence and greater time spent processing events.

The above observations indicate that the dominant part of the display process is the computation of the image coordinates of the endpoints of segments, and not the processing of visual events. With 3-D rotation and vector-drawing hardware this segment coordinate computation can be done quickly for large wire-frame models. Since the added computational cost of processing visual events is small, the animation of much larger scenes can be achieved using the aspect representation.

Thus, the aspect representation can be used to achieve the real-time display of scenes containing at least 1,000-10,000 faces on a general-purpose workstation. From the prototype implementation of this algorithm we have found that for scenes of moderate size and visual complexity

- The number of visual events is in practice a relatively small constant times the number of edges in the scene
- Frame display time is stable for less than 15° increments between frames
- Visual event processing requires less than 5% of the total display time for increments of less than 15° and 1% for increments of less than 2°.

These results suggest that the preprocessing times for typical larger models will be much lower than the theoretical worst-case bounds and that the resulting size of the event structure will be a relatively small constant times the number of edges in the model. In addition, the small computational cost associated with visual event processing shows that the interactive, real-time on-line display of large models can be achieved when preprocessing time is available.

6. Related Work

In computing successive frames of an animation sequence, one technique is to compute each frame independently and store the results. Animated displays of moderate complexity may thus be computed in advance and displayed in real time. Denber and Turner described a method of compressing the data in an animated sequence and increasing the speed of their replay [Denber and Turner, 1986]. The technique involves storing and displaying only the difference between successive images. The technique is raster-based, and it does not involve a faster method for computing the successive images. Glassner described a method for faster ray-tracing of a sequence of images in an animation [Glassner, 1988]. The technique uses a space subdivision algorithm for decreasing the time required for ray tracing. The novel part of the method is that it uses the subdivision techniques in 4-D *spacetime* rather than 3-D space, achieving approximately a 50% decrease in ray-tracing runtime for the examples given.

Hubschman and Zucker introduced the idea of using frame-to-frame coherence to decrease the time required for hidden-line removal [Hubschman and Zucker, 1981]. They work in a world with one or two stationary convex polyhedra, and they find a number of frame-to-frame coherence constraints. The result is a partition of the scene such that "the movement of the viewing position across a partition boundary results in an occlusion relationship becoming active or inactive." The scene is updated when one of these "change boundaries" is crossed. As a result, the storage requirements are $\Theta(n^3)$ even in the case of a single, non-degenerate convex polyhedron. A generalization of this technique to multiple non-convex polyhedra would result in worst-case storage requirements of $\Theta(n^9)$ for a scene with n faces [Plantinga and Dyer, 1990].

Shelley and Greenberg introduced the idea of using frame-to-frame coherence for animation with a viewpoint moving along a path of viewpoints in viewpoint space [Shelley and Greenberg, 1982]. They use a number of culling and sorting rules to reduce the work involved. For example, they find the box bounding the path of viewpoints; any faces that point away from all viewpoints in the box can be removed from consideration for all viewpoints along the path.

Fuchs *et al.* address the problem of displaying a set of polygons from an arbitrary viewpoint in near-real time [Fuchs *et al.*, 1983], with an approach that involves constructing the BSP-tree (Binary Space Partition tree) [Fuchs, *et al.*, 1980] for a scene in an off-line preprocessing stage. Then the display of a frame from some viewpoint with hidden surfaces removed involves traversing the tree to get a list of faces in an approximation of back-to-front order. The faces are drawn on the screen in that order, and the result is an image with hidden surfaces removed. In this approach, displaying an image involves a tree-traversal and the display of all of the faces of the scene.

This approach works well for the problems for which it applies. However, the BSP-tree approach only applies to hidden-surface removal. In cases where hidden-line removal is a desirable or acceptable alternative, our approach may be used to achieve greater frame-rates since hidden-line removal makes it possible to draw the outline of polygons rather than filling them, which is usually slower. And even assuming that filling a polygon and drawing its outline require the same amount of time, our algorithm may have better on-line performance since it only draws the visible polygons in a scene, usually less than the total number of polygons in the scene. The BSP-tree approach requires drawing all of the polygons in the BSP-tree, which is more than the number of polygons in the scene since polygons may have to be split in constructing the BSP-tree.

7. Concluding Remarks

This paper presents an algorithm for efficiently animating rotation by taking advantage of the frame-to-frame coherence inherent in an animated sequence. The appearance from all viewpoints along a given path is computed in a preprocessing phase that works by constructing the aspect representation for the scene. The preprocessing phase also involves computing the initial appearance of the scene with a standard hidden-line removal algorithm. The on-line phase involves the display of views of the scene with hidden lines removed at a real-time rate. It is about as fast as displaying a wire-frame model of a polyhedral scene as it rotates, without removing hidden lines.

Another approach to this problem is to precompute all of the frames of the animation sequence. This approach will require considerably more storage than the method presented above, since each view is stored rather than one view and the differences between views. In our test images above, the number of events between frames was a small constant. In addition, pre-computation time is likely to be greater than for the method presented above, although general comparisons are difficult to make. However, this simple approach may have slightly higher replay speed since the representation is in 2-D rather than 3-D vectors, and 2-D vectors may require less time to display. Of course, the asp approach is more flexible in that the number of degrees per frame is not determined in advance.

The only previous algorithm for efficiently animating rotation of general polyhedra with hidden lines or surfaces removed is that of Fuchs *et al.* [1983]. After some pre-computation time, it is efficient at computing the faces of the scene in an order that approximates back-to-front. This is sufficient for hidden-surface removal by drawing all of the (shaded) faces from back to front. However, when hidden-line removal is a desirable or acceptable alternative to hidden-surface removal, and when pre-computation time is available per viewpoint path (rather than per scene), our algorithm may provide better performance for a class of scenes, at the expense of greater storage requirements.

The algorithm presented here is practical only for a certain class of scenes. When the number of faces in the scene becomes large and the scene is visually complex, the number of visual events will eventually become too large to store. In the convex case, the number of visual events is $O(\mathbf{n})$ where $\mathbf{n}$ is the number of faces in the scene, but in the worst case the number of visual events is $O(\mathbf{n}^3)$. However, the prototype implementation shows that for polyhedral scenes of moderate size and visual complexity, the number of visual events is a relatively small constant times the number of edges in the scene. In practice, depending on the visual complexity of the scene, a typical workstation has enough memory to store the events for polyhedral scenes containing up to approximately 1,000 to 100,000 edges.

In addition, there must be sufficient preprocessing time per rotation and sufficient processing speed to handle the on-line phase. Perhaps a good rule of thumb is that a workstation will be able to animate rotation for a polyhedral scene in real time if it can display a rotating wire-frame model of the scene in real time. Since only a small portion of the on-line display time is used in updating the ESG this rule applies even if 3-D rotation and vector-drawing hardware is available.

References

Denber, M. and P. Turner, "A differential compiler for computer animation," *ACM Computer Graphics* **20**(4), 1986, pp. 21-27.

Fuchs, H., Z. M. Kedem, and B. F. Naylor, "On visible surface generation by a priori tree structures," *ACM Computer Graphics* **14**(3), 1980, pp. 124-133.

Fuchs, H., G. Abram, and E. Grant, "Near real-time shaded display of rigid objects," *ACM Computer Graphics* **17**(3), 1983, pp. 65-72.

Gigus, Z., J. Canny, and R. Seidel, "Efficiently computing and representing aspect graphs of polyhedral objects," *Proc. Second Int. Conf. on Computer Vision*, 1988, pp. 30-39.

Glassner, A., "Spacetime ray tracing for animation," *IEEE Computer Graphics and Applications* **8**(2), 1988, pp. 60-70.

Hubschman, H. and S. Zucker, "Frame-to-frame coherence and the hidden surface computation: constraints for a convex world," *ACM Computer Graphics* **15**(3), 1981, pp. 45-54.

Koenderink, J. and A. van Doorn, "The internal representation of solid shape with respect to vision," *Biol. Cybernetics* **32**, 1979, pp. 211-216.

Plantinga, H. and C. R. Dyer, "An algorithm for constructing the aspect graph," *Proc. 27th IEEE Symp. Foundations of Computer Sci.*, 1986, pp. 123-131.

Plantinga, H., "The Asp: A Continuous, Viewer-Centered Object Representation for Computer Vision," Ph.D. dissertation, University of Wisconsin – Madison, August 1988.

Plantinga, H. and C. R. Dyer, "Visibility, Occlusion, and the Aspect Graph," to appear in *International Journal of Computer Vision*, 1990.

Shelley, K. and D. Greenberg, "Path specification and path coherence," *ACM Computer Graphics* **16**(3), 1982, pp. 157-166.