USING THE ASP FOR THE INTERACTIVE VIEWING OF POLYHEDRAL SCENES

by

W. Brent Seales
and
Charles R. Dyer

Computer Sciences Technical Report #903

December 1989

Using the Asp for the Interactive Viewing of Polyhedral Scenes

W. Brent Seales
Charles R. Dyer

University of Wisconsin–Madison
Department of Computer Sciences
Madison, Wisconsin 53706

Abstract

In this paper we discuss an approach for solving the problem of interactively viewing a polyhedral scene. Interactive viewing is the computation and display of an interactively controlled sequence of views of a scene corresponding to a viewer's movement along a continuous viewpath. We present an algorithm for generating such views with hidden-lines removed, and consider extensions to solve the problem of generating views with hidden-surfaces removed. The method relies on a precomputation phase which constructs the *aspect representation*, or asp. This representation can be used to interactively view a polyhedral scene at video rates with hidden-lines or surfaces removed. The method exploits *viewpath coherence*, a form of frame-to-frame coherence present in such a sequence of views. The display of polyhedral line drawings with hidden lines removed makes use of the topology of the image line drawing and the pre-ordering of visual events which change that topology. This approach is extended to interactive viewing with hidden-surfaces removed and with shading, shadows, and multiple light sources. The set of object resolution polygons representing the visible faces and the shadow polygons for a single frame can be computed efficiently from the previous frame using the asp. The hidden-line and hidden-surface algorithms are closely related via the asp. Interactive viewing with hidden-lines removed is shown to be about as fast as the interactive display of a wire-frame scene. The primary on-line cost of hidden-surface interactive viewing is the cost associated with scan converting the visible surfaces and shadow polygons.

The support of NSF under Grant No. IRI-8802436 is gratefully acknowledged.

1. Introduction

Interactively viewing a static scene requires the computation and display of a sequence of images generated from a user-controlled continuously-changing viewpoint. The goal of a system for interactive viewing is to render a 3D scene as it would appear from a particular vantage point, and to update the appearance of the scene dynamically as the user interactively modifies the vantage point within the environment. There must be a balance, however, between the incompatible goals of detailed and realistic scene rendering, and the speed of the interactive display. Clearly it is important to render the scene as fast as possible to give the user the illusion of movement within the environment.

Computing and displaying such animation sequences has many applications. For example, computer-aided design of 3D objects requires the interactive construction and display of objects from a wide range of viewing directions. The animated display of a rotating 3D object gives the user a sense of depth and structure which is useful in design and visualization [Farr85]. The animated presentation of a scene which can be navigated dynamically has application in flight simulation [Yan85] and architectural walkthrough [Broo86].

Interactive viewing requires the computation and display of a sequence of images generated from a continuously changing viewpoint. There are two essential requirements for this type of animated display: realism in each image and video-rate display. Without an appropriate level of realism, the effect of interaction with an environment is lost. Perceptual continuity is lost when the display rate is too slow. In meeting the requirements of realism and video-rate display there is a fundamental trade-off between off-line and on-line solutions. The precomputation of information can be as extreme as complete image rendering off-line. The on-line animation phase is then reduced to a playback of the precomputed images [Denb86]. There are comparatively efficient ways to render realistic images off-line using, for example, ray tracing techniques [Glas88, Cook84], but the high per-frame cost makes interactive animated display using these approaches intractable. The main drawback of the total precomputation approach is the size and inflexibility of the resulting animation description. At the other end of the spectrum is the synthesis of the frames of the animation sequence on-line. Animation in this case depends on fast dynamic frame rendering [Fuch83]. Without some form of precomputation, however, video-rate display and realistic image synthesis become almost impossible.

In this paper, we discuss the problem of displaying a polyhedral scene interactively from a moving viewpoint. We will refer to this as *interactive viewing*; we consider in detail the restricted viewer motion of a great circle on the unit sphere under orthographic projection. A polyhedral scene can be rendered with varying degrees of detail, ranging from wire-frame display to hidden-surface display using a shading model. We present an algorithm for animating the display of a polyhedral scene with hidden-lines removed; we show that the same algorithm can be modified to allow hidden-surface display including the use of shading models, multiple light sources, and shadow computation. For hidden-line rendering, the

algorithm makes it possible to display the sequence at a rate roughly equivalent to the rate of display of wire-frame models without hidden-lines removed.

Our algorithm takes advantage of *viewpath* coherence, a form of frame-to-frame coherence, by computing the appearance of the scene in the first frame, and then computing the viewpoints on the viewpath 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 *asp* 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, interactive phase, in which a sequence of frames along a user-controlled viewpath is displayed in real time.

The asp quantifies explicitly how all possible kinds of *visual events* occur as a continuous function of viewpoint. Informally, a visual event can be thought of as any change in the scene as a result of occlusion. Thus the disappearance of a face as it turns away from the viewing direction is one such event. The partial occlusion of an edge by some other edge (the beginning or ending of a T-junction) is another kind of event. More formally, all topological changes in a polyhedral scene are the result of the apparent intersection in the image of three edges [Plan89]. In the case of a face which turns away from the viewing direction, the point at which the face is edge-on to the viewer is a degenerate form of the apparent intersection of three edges. The occlusion of an edge by another face creates a T-junction which begins and ends at two distinct viewpoints along a viewpath. The T-junction always begins and ends at a viewing direction where an edge and a vertex appear to intersect. This event (EV event) is another form of the apparent intersection of three edges where the two edges which meet at the vertex actually do intersect.

The orthographic viewing model can be characterized as the set of viewpoints on the surface of the unit sphere S^2. Any viewer movement can be described as a 1D path of viewpoints on the surface of S^2. In general, the apparent intersection of three edges (EEE event) occurs at a single point along a 1D view path on S^2, and corresponds to a topological change in the appearance of the scene. The location of all such events can be computed and ordered sequentially along the 1D viewpath. By precomputing and ordering the viewpoints where topological changes occur, the coherency between viewpoints is exploited. This coherence between frames is a result of the fact that for most small changes in viewpoint along a smooth viewpath, only linear changes in the appearance of the scene take place. Linear changes are those changes in the viewpoint which do not change the structure of the projected line drawing in the image. Changes in the structure, i.e., topological changes in the Edge Structure Graph (ESG), are characterized in the asp as visual events corresponding to the apparent intersection of edges in the image. In order to take advantage of the coherence between viewpoints and hence between frames, these topological changes are explicitly computed and stored in order to represent exactly the changes in appearance in viewpoint

space. At runtime a viewpath through this viewpoint space determines which events are relevant to updating each successive frame in an animation sequence.

Hubschman and Zucker introduced the idea of using frame-to-frame coherence to decrease the time required for hidden-line removal [Hubs81]. They worked in a world with a small number of stationary convex polyhedra and they found a number of frame-to-frame coherence constraints. The result was 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 boundaries is crossed. As a result, the storage requirements are $\Theta(\mathbf{n}^3)$ even in the case of a single, nondegenerate convex polyhedron. A generalization of this technique to multiple non-convex polyhedra would result in worst-case storage requirements of $\Theta(\mathbf{n}^9)$ for a scene with $\mathbf{n}$ faces [Plan87]. Our algorithm places no restrictions on the model polyhedra, allowing arbitrary nonconvexities. In addition, our algorithm extends to the hidden-surface case.

Shelley and Greenberg used frame-to-frame coherence for the generation of an animation sequence corresponding to a smooth viewpath through a 3D environment [Shel82]. A smooth viewpath is represented as a B-spline, and can be interactively defined. Path coherence (frame-to-frame coherence along a single viewpath) is exploited to reduce the expense of the sorting and culling operations for visible line/surface computation. Rendering makes use of a priority ordering of polygons in the environment, and changes in this ordering are computed when the B-spline viewpath crosses one of the separating planes between objects in the environment. Although the viewpath can be specified interactively, the computation of the appearance of the scene along the viewpath is done off-line and then displayed. Our approach generates the appearance of the scene on-line as the viewpath is interactively defined.

The Binary Space Partition Tree (BSP-tree) has been used to display polyhedral scenes in near real-time by precomputing a structure which gives relative depth-ordering for faces in the model [Fuch83]. The BSP-tree simplifies the hidden-surface computation for an arbitrary viewing direction by encoding the relative depth ordering of the model implicitly in a tree structure for all viewing directions. The display of a single frame from some viewpoint with hidden-surfaces removed involves traversing the BSP-tree to generate a list of polygons in back-to-front order. The faces are drawn on the screen in that order, and the result is an image with hidden-surfaces removed. The BSP-tree does not take advantage of the frame-to-frame coherence in animation sequences, however. Each frame of a viewpath is generated by a separate and complete traversal of the BSP-tree structure. Also, the BSP-tree approach only applies to hidden-surface removal. In cases where hidden-line removal is a desirable or acceptable alternative to raster display, our approach for hidden-line animation may be used to achieve greater frame-rates. Finally, 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. Our algorithm only draws the visible polygons, usually less than the total number of polygons in the

scene.

We concentrate in this paper on the efficient application of the asp to both the hidden-line and hidden-surface problems of interactive viewing. We define the asp and discuss its construction in Section 2. The use of the asp for the rendering of polyhedra with hidden-lines removed is discussed in Section 3. Section 4 discusses several shading models and their use with the asp for the rendering of polyhedra with hidden-surfaces removed. In Section 5 we present an object resolution method for computing the shadow regions of polyhedral scenes containing multiple point light sources based on the asp. Section 6 describes how these methods are combined in order to compute animation sequences for polyhedra with hidden surfaces removed. Section 7 presents experimental results and Section 8 concludes by discussing extensions.

2. The Aspect Representation

The aspect representation, or asp [Plan88, Plan87, Plan86] quantifies the visual events in a 3D polyhedral model which arise as a result of occlusion. In general, the visual events which are explicitly represented in the asp are the apparent intersections of edges. Under the orthographic projection model, the apparent intersection of two edges (a T-junction) corresponds to a 2D surface in aspect space. Similarly, the apparent intersection of three edges corresponds to a 1D curve, and four edges which appear to intersect form a single point in aspect space. An actual vertex in the scene can be thought of as a degenerate apparent intersection, where the edges actually do meet at the vertex. Since these various visual events are explicit in the asp, the appearance of an edge from a given viewing direction can be easily computed. The exact appearance of the edge is bounded by the visual events involving other occluding edges in the scene.

The asp for an object represents appearance for all viewpoints. Specifically, the asp for a single face is a 4D volume in the 4D space of image space $\times$ viewpoint space, i.e., a 4D volume in $\mathbb{R}^2 \times \mathbb{S}^2$. The boundaries of this volume are 3D volumes corresponding to the visibility of the edges bounding the face. The 2D surfaces bounding these 3D volumes correspond to the visibility of the vertices of the face. The 4D volume in aspect space which corresponds to the visibility of a face is not a polytope since the surfaces are not in general planar. The equations for the surfaces are well-behaved, however, and are algebraic of degree at most three. Therefore the asp for a face can be represented and manipulated in much the same way as a polyhedron. For example, the intersection of two volumes in aspect space can be computed exactly in closed form.

A fundamental property of aspect space is that occlusion in object space corresponds to set subtraction in aspect space. Thus, the visibility of a face from all viewing directions can be computed exactly by performing set subtraction in aspect space. The process of set subtraction in aspect space generates explicit equations for the 2D surfaces, 1D curves, and vertices mentioned above. The asp was initially studied in order to construct the aspect graph. The work by Plantinga [Plan88] includes a complete discussion of

the definition, construction, and properties of the asp.

The restricted problem of constructing the asp for one degree of freedom in viewpoint simplifies the form of the asp because the allowable viewpoints are constrained to lie on the equator of the view sphere (after an appropriate coordinate transform). By making this simplification, the general asp is reduced in complexity from four degrees of freedom to three. The asp for a polygonal face then becomes a 3D volume bounded by the visibility of its edges. The visibility of an edge becomes a 2D surface in aspect space. All of the visual events are simplified by this viewpoint constraint, so that the apparent intersection of two edges becomes a 1D curve of viewpoints, and is bounded by the apparent intersection of three edges which is a point in aspect space. The equations which generate these volumes, surfaces, curves and points are obtained directly from the general form of the asp by restricting the viewpoints to lie in a plane. This is equivalently done by setting one component of the viewing direction to 0.

The algorithm for constructing the asp for a face **F** is to subtract from the asp for **F** the asps for the faces in front of **F**. Subtraction is set subtraction, and faces in front of **F** are those faces which occlude **F** from some viewpoint. This algorithm hinges on the observation that the occlusion of **F** by other faces in the scene corresponds to the subtraction of the asps for the occluding faces from the asp for **F**. When the viewpoint is constrained to the equator of the view sphere, the subtraction of one asp from another corresponds to the subtraction of the region of intersection of two 3D volumes.

Specifically, the construction of the asp for a scene is done by defining the intersection of the boundaries of the 3D volumes in aspect space corresponding to the visibility of faces. The 2D surfaces in aspect space bounding a 3D volume correspond to the visibility of the edges of a face. The intersection of two 2D surfaces in aspect space is a 1D curve which lies on each of the 2D surfaces. Since each 2D surface in aspect space corresponds to the visibility of an edge in the model, this 1D curve represents the apparent intersection of the two edges in the image. For a specific value of the viewpoint parameter, the equation of the 1D curve which lies on the 2D surface can be used to compute the exact image coordinates of the apparent intersection point of the corresponding two edges.

The visibility of an edge is bounded by the other edges in the scene which occlude it from various viewing directions. The construction of the asp for an edge can be viewed as the computation of the equations of all of the 1D curves which lie on the 2D surface for that edge. Computing all such curves for the 2D surface involves finding the intersection of the 2D surface with all other 2D surfaces which correspond to the other edges in the scene. Consider the two edges $\mathbf{E}_1$ and $\mathbf{E}_2$ in $\mathbb{R}^3$. Let $\mathbf{E}_1$ be an edge from point $\mathbf{p}_1 = (x_1, y_1, z_1)$ to point $\mathbf{p}_1 + \mathbf{a}_1$ where $\mathbf{a}_1 = (a_1, b_1, c_1)$. Let $\mathbf{E}_2$ be an edge from point $\mathbf{p}_2 = (x_2, y_2, z_2)$ to point $\mathbf{p}_2 + \mathbf{a}_2$ where $\mathbf{a}_2 = (a_2, b_2, c_2)$. The edge $\mathbf{E}_1$ can be parametrized in $\mathbb{R}^3$ by the expression $\mathbf{p}_1 + s\ \mathbf{a}_1$, where $0 \leq s \leq 1$. The viewing direction $\mathbf{v}$ can be described in spherical coordinates as the vector $\mathbf{v} = (\sin\theta\ ,\ 0\ ,\ \cos\theta\ )$ which depends on the single viewpoint parameter θ. The equation of the image point where two edges $\mathbf{E}_1$ and $\mathbf{E}_2$ appear to intersect in the image plane (u, v) is a

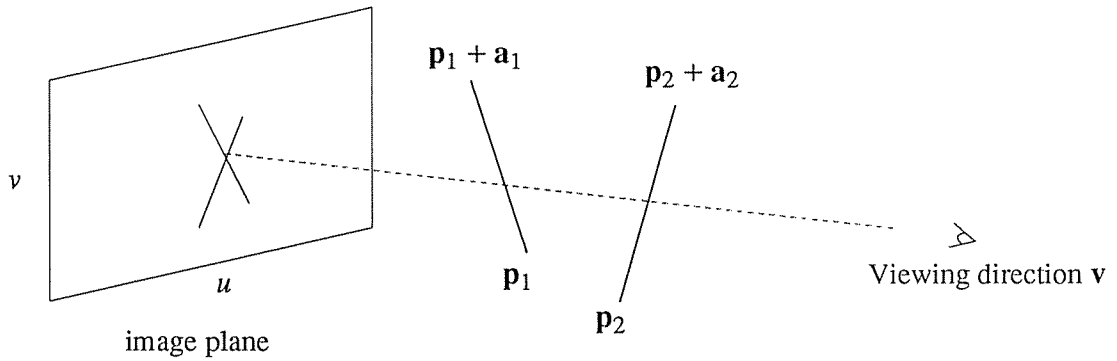


Figure 1. The apparent intersection of two edges in the image plane.

function of the viewpoint parameter θ and is given by [Plan88]

$$u = (x_1 + s\, a_1)\cos\theta - (c_1 + s\, z_1)\sin\theta \tag{1}$$

$$v = y_1 + s\, b_1$$

where the parameter s is expressed as

$$s = \frac{\mathbf{v}\cdot((\mathbf{p}_2 - \mathbf{p}_1) \times \mathbf{a}_2)}{\mathbf{v}\cdot(\mathbf{a}_1 \times \mathbf{a}_2)}$$

Figure 1 shows the edges $\mathbf{E}_1$ and $\mathbf{E}_2$ in $\mathbb{R}^3$ and the apparent intersection of $\mathbf{E}_1$ and $\mathbf{E}_2$ in the image plane. Given a fixed value of the viewpoint parameter θ, Eq. 1 gives the image coordinates of the apparent intersection of the edges $\mathbf{E}_1$ and $\mathbf{E}_2$.

The 1D curve described by Eq. 1 corresponds geometrically to the intersection in aspect space of the 2D surfaces generated by edges $\mathbf{E}_1$ and $\mathbf{E}_2$. The 1D curve of intersection extends over some range of viewpoints $[\theta_i, \theta_j]$. Figure 2(a) shows the edges $\mathbf{E}_1$ and $\mathbf{E}_2$ as they would appear in the image plane over a range of viewpoints. Notice that in the leftmost view shown there is no intersection of the edges in the image plane. As the viewpoint parameter changes along the rotation path, the edges appear to intersect. Figure 2(b) shows the 2D surface in aspect space which corresponds to $\mathbf{E}_1$. The 1D curve on this

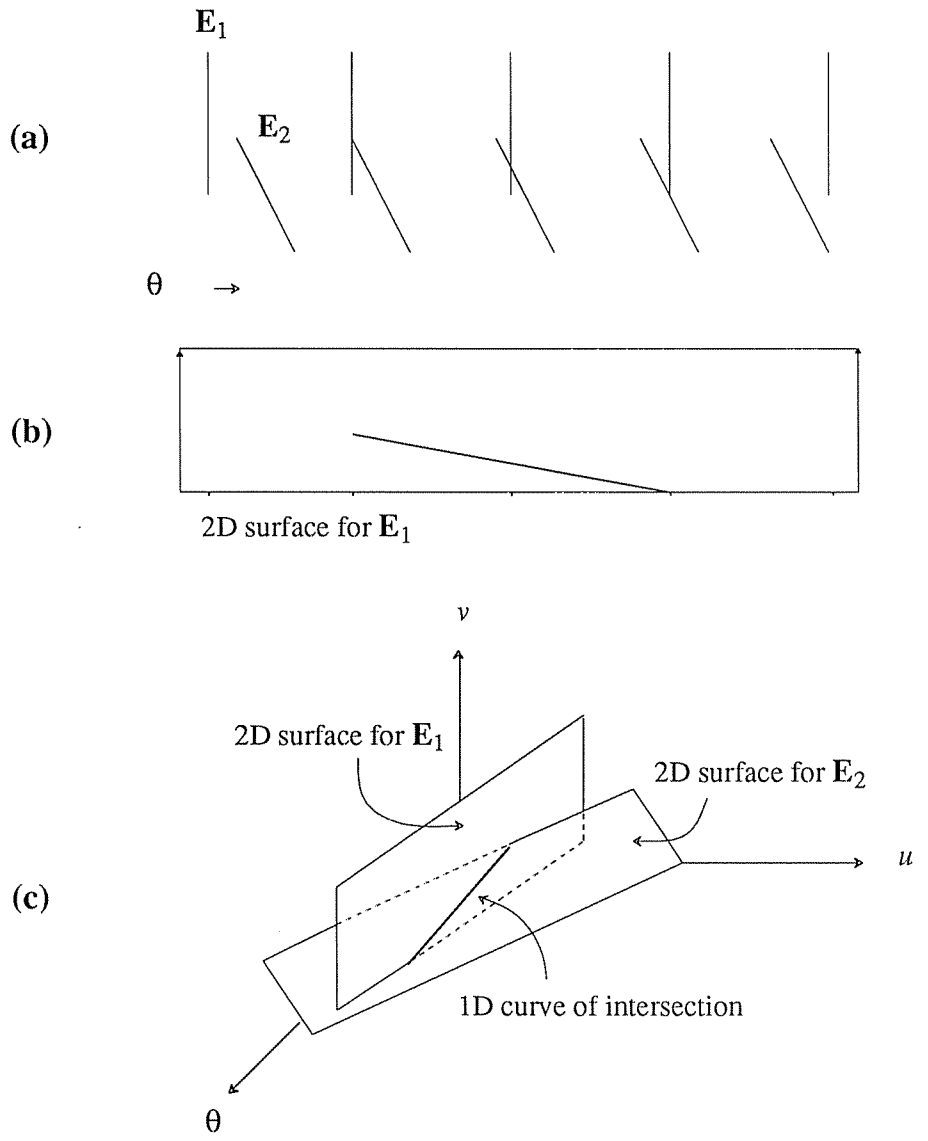


Figure 2. 1D curve generated by the intersection of two 2D surfaces in aspect space. **(a)** Appearance of two edges in the image plane at several points along a rotation. **(b)** 2D surface in aspect space for $\mathbf{E}_1$ showing 1D curve of intersection. **(c)** Intersection of two 2D surfaces and the resulting 1D curve in aspect space.

surface is the trace of the apparent intersection of $\mathbf{E}_1$ and $\mathbf{E}_2$. Figure 2(c) illustrates how the 1D curve which is the trace of the apparent intersection of $\mathbf{E}_1$ and $\mathbf{E}_2$ corresponds to the intersection of two 2D surfaces in aspect space. The figure shows two 2D surfaces, one corresponding to $\mathbf{E}_1$ and the other corresponding to $\mathbf{E}_2$. The intersection of the 2D surfaces is the 1D curve described by Eq. 1 and shown in Figure 2(b). The appearance of the two edges in the image plane is represented by a 2D cross-section for a fixed value of the viewpoint parameter θ. The figure is drawn for simplicity as planar, although the 2D surfaces and 1D curves in aspect space are governed by Eq. 1 and hence are not planar.

The most general visual event is the apparent intersection of three edges. This visual event corresponds to the intersection on a 2D surface of two distinct 1D curves. Computing the viewpoints where EEE events occur involves finding the intersection of 1D curves which were generated by edges which do not lie on the same face. The viewpoint of intersection where two curves on the same 2D surface coincide can be found directly by solving the equation for viewpoint given the equations of the curves and the 2D surface on which they lie. Consider three edges $\mathbf{E}_1$, $\mathbf{E}_2$, and $\mathbf{E}_3$ which lie in $\mathbb{R}^3$ as illustrated in Figure 3(a). $\mathbf{E}_1$ connects the points $\mathbf{p}_1$ and $\mathbf{p}_1 + \mathbf{a}_1$. Likewise, $\mathbf{E}_2$ connects the points $\mathbf{p}_2$ and $\mathbf{p}_2 + \mathbf{a}_2$ and $\mathbf{E}_3$ connects the points $\mathbf{p}_3$ and $\mathbf{p}_3 + \mathbf{a}_3$. Let $\mathbf{S}_i$ represent the 2D surface in aspect space corresponding to $\mathbf{E}_i$. The intersection of $\mathbf{S}_1$ and $\mathbf{S}_2$ in aspect space is a 1D curve $\mathbf{C}_1$ on $\mathbf{S}_1$. Similarly, the intersection of $\mathbf{S}_1$ and $\mathbf{S}_3$ in aspect space is a 1D curve $\mathbf{C}_2$ on $\mathbf{S}_1$. The intersection of the two curves $\mathbf{C}_1$ and $\mathbf{C}_2$ on $\mathbf{S}_1$ corresponds to the viewing direction where all three edges appear to intersect in the image plane. This viewing direction is equivalent to the direction of some vector through a point $\mathbf{p}_1 + s\mathbf{a}_1$ on $\mathbf{E}_1$, $0 \leq s \leq 1$, which intersects both $\mathbf{E}_2$ and $\mathbf{E}_3$. A vector parallel to this direction is given by

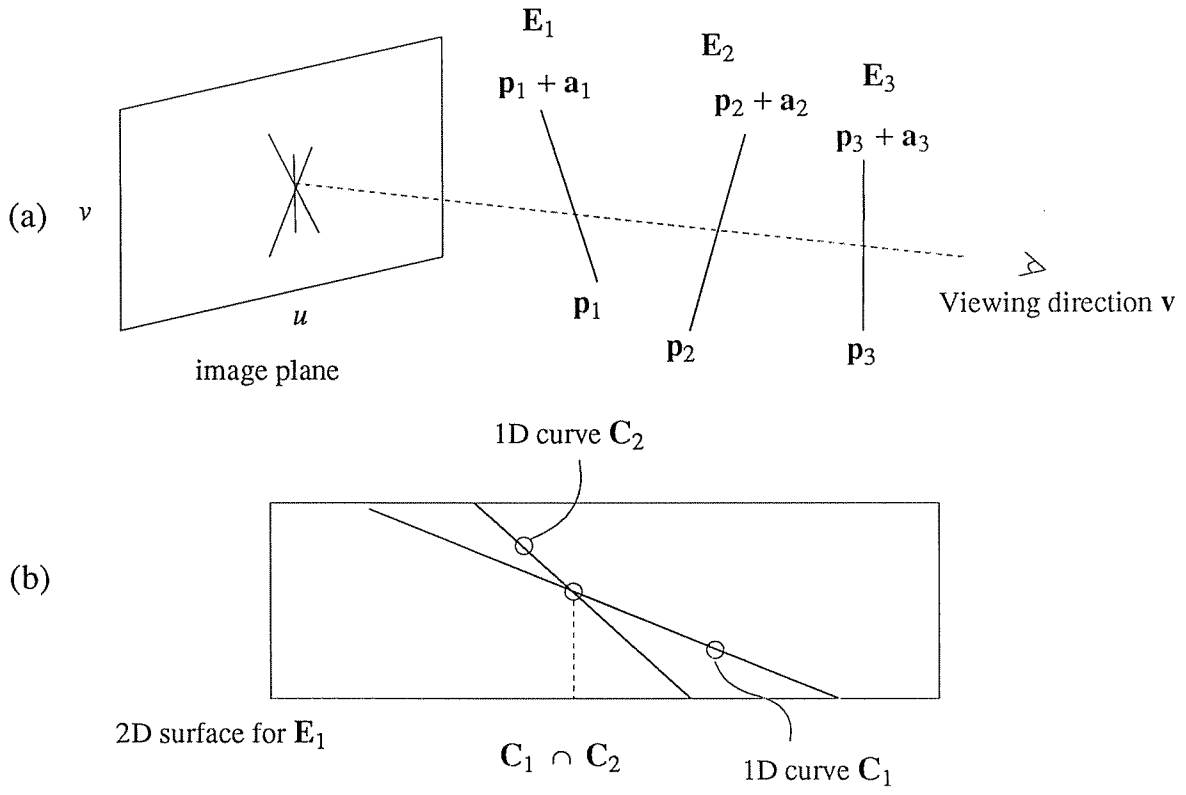
$$\mathbf{v} = ((\mathbf{p}_1 + s\mathbf{a}_1 - \mathbf{p}_2) \times \mathbf{a}_2) \times ((\mathbf{p}_1 + s\mathbf{a}_1 - \mathbf{p}_3) \times \mathbf{a}_3) \tag{2}$$

Since the y-component of the viewing direction is constrained to be 0, Eq. 2 has the form

$$\alpha s^2 + \beta s + \gamma = 0 \tag{3}$$

where α , β, and γ are linear functions of the endpoints of $\mathbf{E}_1$, $\mathbf{E}_2$ and $\mathbf{E}_3$. Eq. 3 can be solved for the parameter s, and then Eq. 2 can be solved for the desired vector $\mathbf{v}$. The value of the viewpoint parameter θ for vector $\mathbf{v}$ is the value of θ where the two 1D curves $\mathbf{C}_1$ and $\mathbf{C}_2$ intersect in aspect space on $\mathbf{S}_1$. This is shown in Figure 3(b).

Since the visibility of an edge is a function of the face to which the edge belongs, determining which edges in the scene occlude a given edge can be done quickly using simple tests. For example, edges which have non-overlapping y-coordinates in $\mathbb{R}^3$ cannot occlude each other. The extent of the curve of intersection between two 2D surfaces is bounded by the viewpoint extent of each of the 2D surfaces. In other words, the computed curve of intersection between two 2D surfaces must lie on both 2D surfaces. The viewpoint extent of the curve is therefore determined by the extent of the 2D surfaces in



$\mathbf{S}_i$: 2D surface corresponding to $\mathbf{E}_i$

$\mathbf{C}_1$: 1D curve generated from $\mathbf{S}_1 \cap \mathbf{S}_2$

$\mathbf{C}_2$: 1D curve generated from $\mathbf{S}_1 \cap \mathbf{S}_3$

Figure 3. (a) The apparent intersection of three scene edges in the image plane (b) The intersection of two 1D curves on a 2D surface in aspect space

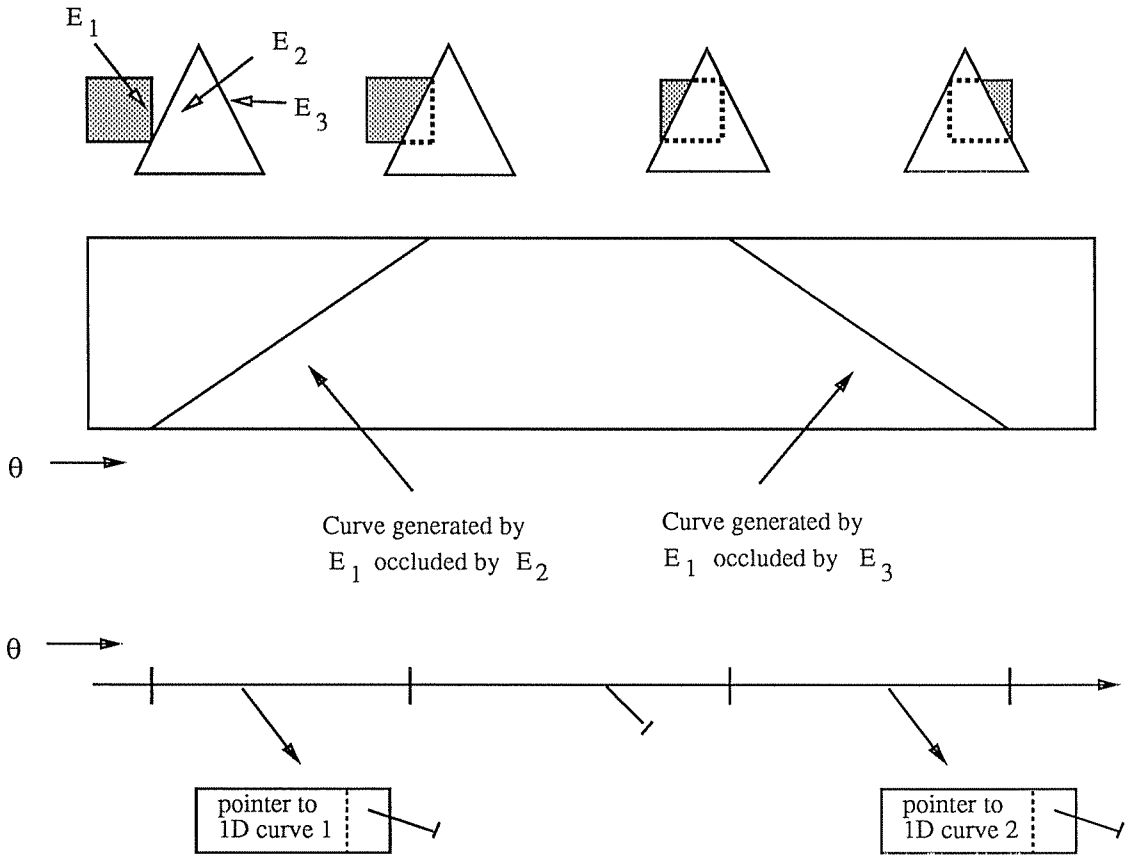


Figure 4. The occlusion of an edge of the square by the triangle corresponds to 1D curves on the 2D surface for that edge in aspect space.

aspect space, which is directly related to the visibility of the corresponding edge as a part of the face to which it belongs.

The goal of the construction of the asp is to compute exactly the viewpoints where visual events occur in the scene, and to use this list of visual events to represent how the scene will appear in the image plane. The complete asp for a polygonal face is the set of 2D surfaces corresponding to the edges of that face. Associated with each 2D surface is the set of 1D curves of intersection generated by all other occluding edges in the scene. Recall that each 1D curve on a 2D surface is the equation of an apparent intersection of two edges in the scene. The visual events of interest are the EEE events mentioned above: (1) the viewpoints where the edge appears or disappears completely as a part of the face of which it is a

part, (2) the viewpoints where a T-junction begins or ends, and (3) the most general EEE event where three non-intersecting edges appear to intersect at a single viewpoint. The viewpoints which represent these events can be obtained directly from the representation of 1D curves of intersection on a 2D surface. For each edge and its corresponding 2D surface, the viewpoints where the edge appears or disappears completely is the extent in viewpoint space of the 2D surface. This extent is given directly as the visibility of the face to which the edge belongs. The starting and ending of each T-junction for an edge is given by the viewpoint extent of each 1D curve of intersection on the 2D surface for that edge. These are found directly without additional computation, and can be stored in a list of events associated with the 2D surface.

All visual events affecting the appearance of an edge can be derived directly from the 2D surface for that edge and the 1D curves corresponding to the intersection of the 2D surface with all other 2D surfaces in the scene. Once computed, the visual events for each edge can be sorted and stored with the corresponding 2D surface. Figure 4 shows the event list which results for one edge $\mathbf{E}_1$ of a square which is being occluded by a triangle. The edges of the triangle cause two 1D curves to be generated on the 2D surface for the edge of the square. The viewpoint extent of the 1D curves on the 2D surface are the viewpoints where the T-junctions on $\mathbf{E}_1$ begin and end. The beginning and ending viewpoints correspond to EV events, and are the visual events in this example which are sorted and stored. As shown in the figure, the appearance of $\mathbf{E}_1$ over some interval $\mathbf{I}$ on the viewpath is described by a list of 1D curves. The set of 1D curves corresponds to T-junctions and hence determines the appearance of the edge over interval $\mathbf{I}$ in the sorted list of viewpoints.

More generally, a list of visual events $\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_k$ delimits all of the topological changes in the appearance of an edge corresponding to a 2D surface. For each particular interval $\mathbf{I} = [\varepsilon_i, \varepsilon_{i+1}]$ of the visual event list in the viewpath, the appearance of the edge is given by the set of 1D curves which extend over that interval. Figure 5 shows a portion of the visual event list for a single edge $\mathbf{E}_1$ being occluded by a face with five edges. For $\theta = \varepsilon_i$ there is an EV event where $\mathbf{E}_4$, $\mathbf{E}_5$ and $\mathbf{E}_1$ appear to intersect. At $\theta = \varepsilon_{i+1}$ the EV event involves $\mathbf{E}_2$, $\mathbf{E}_3$ and $\mathbf{E}_1$. Between viewpoints ε_i and ε_{i+1} the 1D curves in aspect space which determine the visibility of $\mathbf{E}_1$ are the curves generated by $\mathbf{E}_4$ and $\mathbf{E}_5$ with $\mathbf{E}_1$.

All of the visual events affecting the appearance of an edge over an interval in viewpoint space are represented by the set of 1D curves within that interval. Since no visual events occur *within* an interval $\mathbf{I}$, the set of 1D curves for an edge can be ordered within $\mathbf{I}$ based on the orientation of the edge. The 1D curves for interval $\mathbf{I}$ correspond to T-junctions along the edge. These junctions can be ordered relative to the designated *start* vertex of the directed edge. The ordering is based upon the *a priori* orientation of each edge, i.e., edges are directed from a start vertex to an end vertex, and T-junction locations along the edge can thus be ordered relative to one of these vertices. A change in the order corresponds to an EEE event of some type. Since no such event can occur *within* $\mathbf{I}$, the ordering of 1D surfaces cannot change

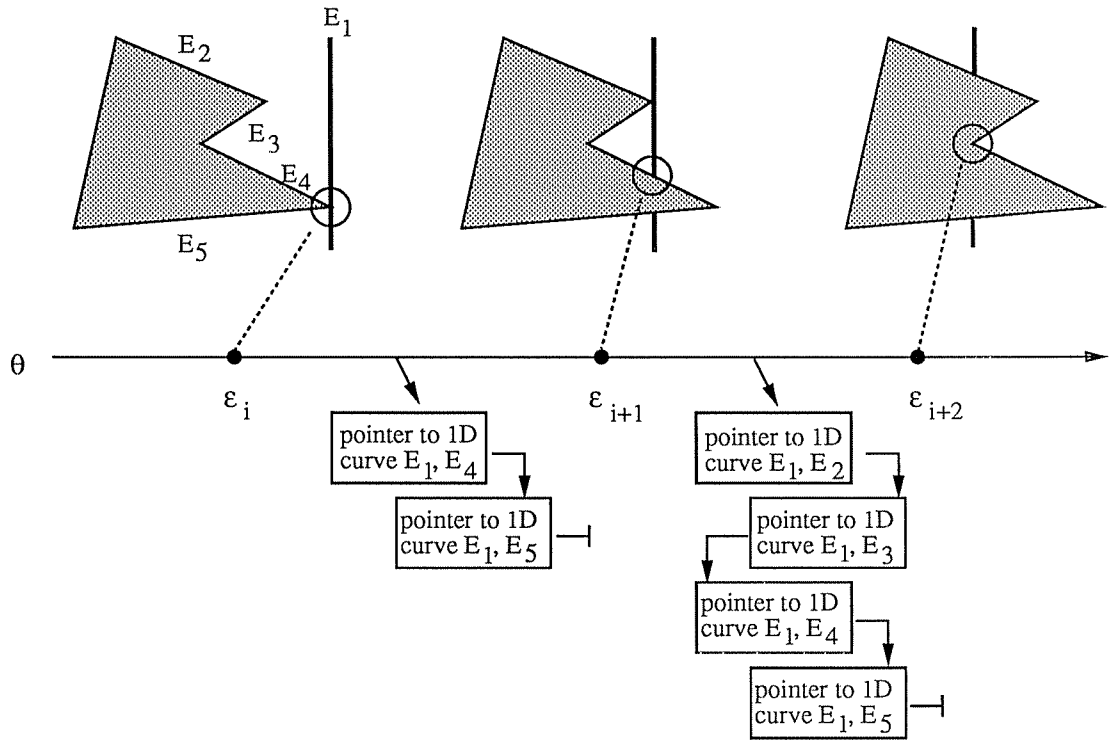


Figure 5. The 1D curves in aspect space which determine the appearance of an edge can be ordered for each interval in the list of visual events for the edge.

within **I**. This ordering can be done during the construction phase, making the on-line computation of the visible portions of the edge for interval **I** even faster.

The length of the list of pointers to 1D curves for any particular interval **I** varies depending on the amount of occlusion. For an unoccluded edge, the list of pointers to bounding curves is length two, i.e., a curve for each endpoint. Due to occlusion, however, a single edge can be fragmented into arbitrarily many disjoint pieces. An example of this type of fragmentation is shown by the interval $\mathbf{I} = [\varepsilon_{i+1}, \varepsilon_{i+2}]$ in Figure 5. Notice that the order of the list of 1D curves for interval $[\varepsilon_{i+1}, \varepsilon_{i+2}]$ in Figure 5 determines the appearance of $\mathbf{E}_1$ for all viewpoints within $[\varepsilon_{i+1}, \varepsilon_{i+2}]$. Because of the fragmentation of edges at various viewpoints, the resulting length of the list of pointers to curves bounding the visibility of the edge over a particular interval may be of length greater than two.

To analyze the complexity of the construction phase, let **e** represent the number of edges in a scene. The construction algorithm will form a 2D surface corresponding to each edge in the scene. Let **q** be the number of 1D surfaces on a particular 2D surface. The 1D surfaces are the curves which represent the intersection in aspect space of two 2D surfaces. For a particular 2D surface there are $2 + 2\mathbf{q} + \frac{\mathbf{q}^2 - \mathbf{q}}{2}$ events which can arise in the worst case. This follows by counting the number of each type of event which can occur given a 2D surface:

2	Appearance and disappearance of the entire edge
$2\mathbf{q}$	EV events (**q** 1D curves, each endpoint an EV event)
$\frac{\mathbf{q}^2-\mathbf{q}}{2}$	EEE events (intersection of **q** 1D surfaces on the 2D surface)

It follows that the complete construction of the asp for a single edge (a 2D surface) is bounded by $O(\mathbf{e}\,\mathbf{q}^2)$. In order to construct a complete 2D surface for each edge in the scene, the cost is $O(\mathbf{e}^2\mathbf{q}^2)$. In the worst case, $\mathbf{q} = \mathbf{e}$ for every 2D surface so that the construction time is $O(\mathbf{e}^4)$. In practice, **q** is only a small fraction of **e** for each 2D surface, and hence the overall construction time is somewhere between $O(\mathbf{e}^2)$ and $O(\mathbf{e}^3)$. Notice that, given a particular edge e_i, the computation of the edges which occlude e_i is in the worst case $O(\mathbf{e})$. This implies that the construction algorithm can do no better that $O(\mathbf{e}^2)$ unless other information is given about which edges are "in front of" other edges. For example, if the scene is known to be convex, the construction time of the asp is linear in **e**.

3. Interactive Display

The asp construction phase generates a representation off-line which can then be used to generate interactively the image sequences of a polyhedral scene from the position of a moving viewer with hidden-lines removed. The asp encodes the viewpoints within the space of all possible viewpoints where visual events occur. The interactive display process allows the user to interactively select a viewpath in the space of all viewpoints, and then uses the asp to determine where along that path changes in the appearance of the scene will occur. In the case where the viewspace is restricted to a great circle $\mathbf{S}^1$ on the view sphere $\mathbf{S}^2$, the user can interactively move along the great circle in either direction and at any speed.

The visual events which are computed from the asp are edge events and determine the appearance of the lines in the scene. The display phase makes use of the constructed asp for the scene as well as a list of global visual events and a list of scene elements which are currently visible. Global visual events are those viewpoints where an entire face either appears or disappears. This list is relatively small and is used to focus the display process on only those faces (and hence edges) which are visible at a given

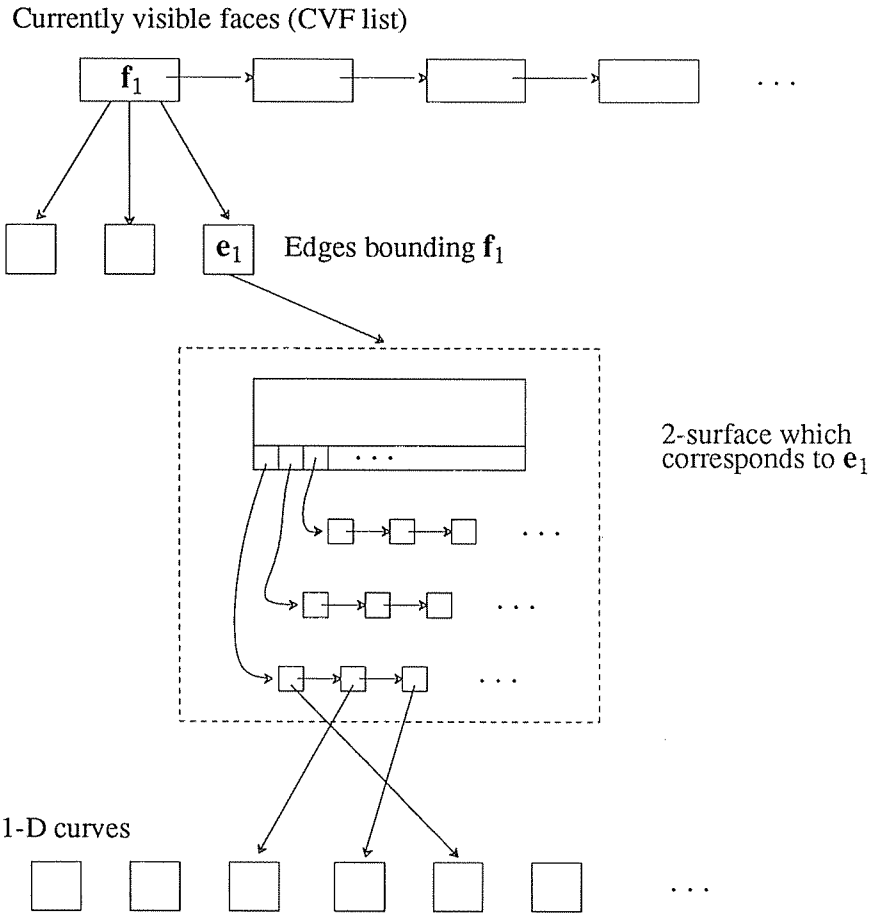
viewpoint. The list of currently visible faces is maintained in conjunction with the global event list. The list of currently visible faces (CVF list) must be computed for the initial viewpoint of chosen viewpath. At the viewpoints corresponding to subsequent frames to be rendered along the viewpath, updates to the list according to the global events which may have occurred (faces appearing and disappearing) are recorded in the event list.

With each edge in the scene is associated a fully constructed 2D surface in aspect space which is computed in the construction phase of the algorithm. Each edge is associated with its corresponding 2D surface. The 2D surface represents for its corresponding edge, the visual events along the view path which will affect its visibility. The 1D curves for each 2D surface provide the information needed to display the edge as it appears. The ordering of the 1D curves for each event interval in the 2D surface gives the appearance of the edge directly. Figure 6 shows the structure of the representation used to display the scene from the viewpath.

The algorithm for display based on the representation in Figure 6 can be divided into two main steps: (1) render the scene for a current viewing direction, and (2) update the structure based on a small change in viewpoint. These steps are iterated, once for each frame in the display. This assumes the existence of the list of currently visible faces for the initial viewpoint, and the global event list for the faces in the scene. These two lists can also be precomputed since they depend only upon the initial viewpoint of the viewpath. The computation of the list of currently visible features and the global event list will be described below.

The CVF list is a list of faces which are visible from the current viewing direction. Initially this list must be generated for the starting point of the viewpath. Since the normal to each face is fixed, this list can be precomputed for some starting viewpoint θ_0. The determination of those faces which are visible from viewing direction θ_0 can be computed using a simple dot product involving the normal to each face in $\mathbb{R}^3$ and the initial viewing direction θ_0. For example, let $\mathbf{n}$ be the normal vector in $\mathbb{R}^3$ to face $\mathbf{f}$. Let θ_0 be the initial viewing direction. Recall that the viewing directions for this restricted viewspace lie on a great circle of the view sphere and hence form a plane. The normal $\mathbf{n}$ can be divided into components so that $\mathbf{n}_\theta$ is the projection of $\mathbf{n}$ onto the view plane. The sign of $\mathbf{n}_\theta \cdot \theta_0$ determines the visibility of the face $\mathbf{f}$ from direction θ_0. This test is performed for each face in the scene. The resulting list is a list of all faces which are potentially visible from the initial viewing direction. These faces are only potentially visible since they may be occluded.

The global event list records the specific value of the viewpoint parameter where all faces in the scene appear and disappear. Faces are oriented and hence viewpoints in front of a face are those from which the face is visible. The viewpoints where the face becomes visible and where it becomes invisible are the two viewpoints where the face appears edge-on. These directions can be easily computed from the directed normal to each face. The global event list is sorted by viewpoint, and with each value is



Global event list

θ	2.29	3.1	3.21	. . .
Events	$\mathbf{f}_1$ appears	$\mathbf{f}_3$ disappears	$\mathbf{f}_4$ appears	. . .

Figure 6. The representation used for the display of the rotation sequence.

associated the type of event (*appear* or *disappear*) and the face affected.

The display phase begins after the CVF list and the global event list are computed. For each face on the CVF list the appearance is computed and displayed. The appearance of the edges for each face on the CVF list is computed from the 2D surfaces corresponding to those edges. Each 2D surface has a list of

event values which have been sorted. The interval which contains the current viewpoint is found, and the list of pointers to 1D curves in aspect space directly gives the appearance of the edge. A *nil* pointer signifies that the edge is entirely occluded. Hence the only computation involved is the scanning of the event list for a 2D surface and the computation of the image coordinates for each of the 1D curves affecting the appearance of the edge. Since the viewpoint typically changes by small increments along the viewpath, a complete scan of the event list within a 2D surface is not necessary at every frame. The position in the event list for the previous frame can be saved. The interval which contains the new value of the viewpoint parameter is usually within a single step or two of the previous frame's interval. For example, Figure 5 shows part of a 2D surface for an edge $\mathbf{E}_1$. Suppose that at frame k the value of the viewpoint parameter is θ_k, and that $\varepsilon_i \leq \theta_k \leq \varepsilon_{i+1}$. A pointer to interval $[\varepsilon_i, \varepsilon_{i+1}]$ is stored with the 2D surface for $\mathbf{E}_1$. If frame $k+1$ corresponds to the viewpoint $\theta = \theta_{k+1}$, the event interval for the 2D surface which contains θ_{k+1} must be found. The search, however, can begin at $[\varepsilon_i, \varepsilon_{i+1}]$. For small changes in θ, θ_{k+1} is likely to lie within or close to $[\varepsilon_i, \varepsilon_{i+1}]$.

The cost of the computation of the appearance of each frame along a viewpath is bounded by the number of faces visible for that frame. Let **f** be the number of faces on the CVF list for some frame, and assume that each face is consists of an average of **e** edges. In order to display a frame, the algorithm must compute the appearance of each edge for each face on the CVF list. Consider the following definitions:

s	Cost of scanning the event list within a 2D surface
a	Cost of computing the image points for a single 1D curve in aspect space
c	Average number of 1D curves determining the visibility of an edge
d = c s a	Cost of displaying one edge using the asp

Since the appearance of the scene for a single frame is the appearance of each of the edges on the CVF list, the cost of computing the appearance is **f e d**, i.e., linear in the number of edges to be displayed. In practice, **s** is small after the initial frame, and *a* involves only a small constant number of multiplication and addition instructions (and a single division). **c** is also, on the average, small and hence the cost of computing the appearance of an single edge is a small constant.

In addition to the cost of computing appearance, an update cost is incurred in order to keep the CVF list current. Let **u** be the cost of a single update to the CVF list. Let α be the average number of updates for a single frame (i.e., the average number of faces which appear and disappear between two frames). The cost to update the CVL for a single frame is α **u**. Assuming a uniform distribution of events and small increments in viewpoint per frame, α will be small. The update cost is bounded by the cost to update the CVF list. Using an appropriate data structure (e.g., a balanced binary tree) this cost is O(log **f**). Thus the total cost for a single frame is **f e d** + α **u** which is bounded by the linear cost of computing the appearance of the edges visible in a single frame.

4. Shaded Display of Polyhedral Scenes Using the Asp

The shaded display of polyhedral scenes conveys information about significant features including lighting, shading, surface markings and surface reflectance, texture, and so on. The information about visual events which affect polyhedral surface appearance is already encoded in the asp since the appearance of a face is determined by the appearance of the set of *edges* bounding the face. The description of the appearance of faces which can be extracted from the asp can be used to formulate a hidden-surface version of the previous algorithm for interactive display. The primary advantage of a hidden-surface rendering is the added realism gained from shading models, multiple light sources and shadows.

The asp is a face-based representation and encodes information about how edges are *connected.* Consequently, the asp represents the appearance of *faces* as a function of viewpoint. By representing the appearance of faces, the area enclosed by a particular face in the image plane is a closed (but not necessarily connected) region corresponding to the projection of a face in $\mathbb{R}^3$. The interior of a closed region can be filled according to shading information based on the reflectance of the surface, the position of the light source(s) with respect to viewpoint, and the effect of shadows cast by the light source(s) in the scene.

With arbitrary occlusion, the appearance of a single face may be a disjoint set of polygons. This situation is represented in the asp, however, because the asp allows the appearance of a face to be computed as a set of closed polygons each of which is bounded by an ordered list of edges. Thus the appearance of a face is a set of closed boundaries, even when a face is partially occluded. This set of closed boundaries in the image plane is guaranteed to be "empty" because the asp gives the appearance of the scene with hidden lines removed.

The positions of light sources must be incorporated into the shaded display process. It is significant to note, however, that the asp for a scene as it has been defined is independent of point light source positions. The asp encodes visual events arising from the geometry of the faces and edges in the polyhedral scene. Thus the asp is not dependent on the position in viewpoint space of light sources since the asp represents appearance from all viewpoints. This is significant since the light source can be moved and the scene rendered again without reconstructing the asp. The cost of constructing the asp is incurred only once in an off-line preprocessing phase. It is a natural result of the viewer-centered property of the asp that the position of a point light source defines another viewpoint, and visibility information from all viewpoints is included in the asp.

Section 4.1 discusses the shaded display of polyhedral scenes using a simple flat-shading model and the asp. Section 4.2 presents Phong shading in conjunction with the asp for shaded rendering.

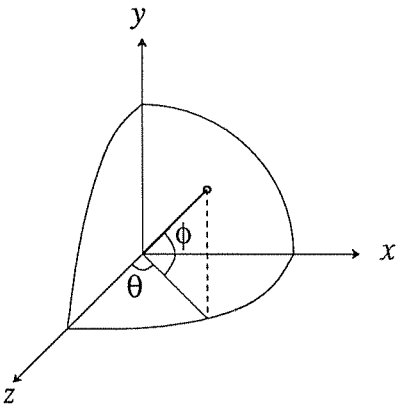


Figure 7. The viewpoint (θ, ϕ) on the unit sphere.

4.1. Flat Shading

The shaded display of polyhedral models in the simplest case assigns to each pixel belonging to a given face a fixed intensity value. Assuming a perfectly diffusing surface and point light sources at infinity, the intensity value at each point on the same planar surface is the same. This value is a function of the reflectance model which depends upon the normal to the face, the position of the light sources, and the reflectance properties of the surface. When considering perfectly diffusing surfaces, the viewer's position with respect to the light source does not matter since light is scattered uniformly in all directions. Since the viewing position does not affect the shading of a given face, and the angle between the surface normal and the light source is constant, the appropriate intensity value can be computed just once for each face. The intensity I for each pixel associated with a specific face given by

$$I = \sum_{i=1}^{N} I_{L_i} R(\phi_{L_i}, \theta_{L_i}) \tag{4}$$

where I_{L_i} is the radiance of the ith light source, ϕ_{L_i} and θ_{L_i} are the polar coordinates on the view sphere of the ith light source (see Figure 7), R is the surface reflectance function, and N is the number of light sources. For a single point light source the amount of reflected light at a point on the surface can be expressed as

$$R_d = k_d I_L(\mathbf{N} \cdot \mathbf{L}) \tag{5}$$

where $\mathbf{N}$ is the surface normal, $\mathbf{L}$ is the direction of the light source, I_L is the radiance of the light source, and k_d is the percentage of incident light reflected. Eq. 4 can be refined to include a constant term which accounts for reflected ambient light, and to model diffuse reflection at each point according to Eq. 5. Thus the reflected intensity at each point on a face is

$$I = I_a + \sum_{i=1}^{N} k_d I_{L_i}(\mathbf{N} \cdot \mathbf{L_i}) \tag{6}$$

where I_a is a constant to account for reflected ambient light [Whit88].

Under the assumptions of diffuse surfaces and point light sources, the shading model described by Eq. 6 can be used to precompute the shaded intensity of each of the faces in the scene given a fixed set of light source positions. This information is equivalent to the reflectance map [Horn77] for the scene, and can be used in conjunction with the asp to render a shaded display of the scene from any viewpoint. The appearance of each face can be computed with hidden lines removed directly from the asp. The shading information is constant (under the current surface and lighting assumptions) and as a result can be precomputed and stored in a lookup table.

The efficient scan conversion of a polygonal region which is described as an ordered list of vertices can be done with special-purpose graphics hardware [Fole82]. The asp provides the ordered segments bounding the appearance of a face as a result of the connectivity information generated during the construction process. The appearance of a face may be a single polygonal region which is shaded using a lookup table to determine the appropriate single intensity value for each pixel within the polygon. In general, the appearance of a face can be a set of disjoint polygons. This presents no complications, however, since the shading information is constant for all points on the same face. As a result, rendering a face which appears as a disjoint set of polygons can be done by filling each polygon separately with the appropriate intensity value.

4.2. Interpolated Shading

The shading model defined by Eq. 6 does not simulate the smoothness of curved surfaces which are approximated by polygons. Given a smooth surface and its corresponding polygonal approximation, the Phong shading model [Phon75] interpolates geometric information to estimate a normal at each point on the surface. This normal is used with an augmented shading function to shade each point on the surface individually, yielding a smoother shading within and across polygons. Notice that with the shading function of Eq. 6, each point on the same face receives the same intensity value. Using Phong shading, the correspondence between the approximated surface and the polygon is used to shade each point on the polygon independently. This gives a smoother, more realistic shading within each polygon. The Phong

model is given by the previous model plus a specular term so that the intensity at each point becomes

$$I = I_a + \sum_{i=1}^{N} k_d I_{L_i}(\mathbf{N} \cdot \mathbf{L_i}) + k_s \sum_{i=1}^{N} I_{L_i} R_s \tag{7}$$

where R_s is a reflectance model which approximates specularity and is also related to surface smoothness.

The application of the Phong shading model depends on two fundamental geometric assumptions: (1) a smooth, curved surface has been approximated by a set of planar faces, and (2) given a point on a planar face **f** contained in $\mathbb{R}^3$, a corresponding normal to the smooth surface can be estimated. Assumption (1) presents no additional difficulty since the asp deals directly with polygonal representations which may also be approximations of curved surfaces. Assumption (2) can be handled for a point on a face **f** contained in $\mathbb{R}^3$ using a bilinear interpolation scheme which makes use of the normal to the surface at the vertices of **f** [Phon75]. This scheme assumes that the 3D coordinates of the point on the face are known. The information derived from the asp, however, is the appearance in the image plane only, i.e., there is no 3D information. In order to shade a polygon in the image plane using this method, it must be possible to associate each point to be shaded in the image plane with a corresponding point on a face in $\mathbb{R}^3$. The

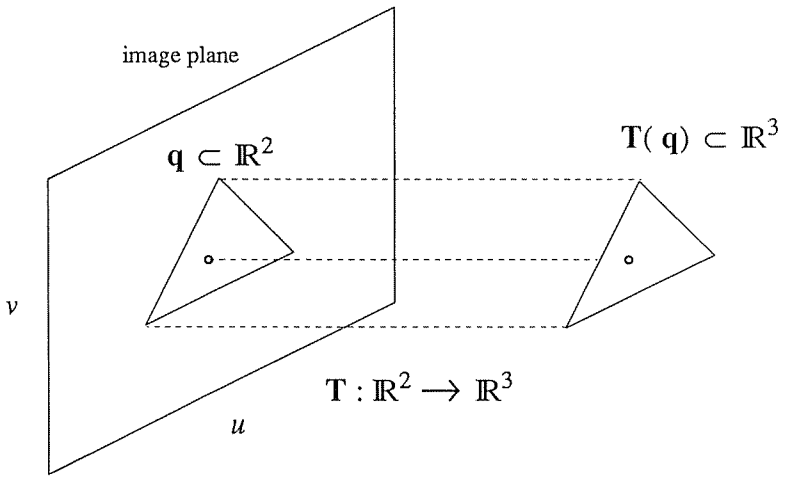


Figure 8. The transformation **T** recovers the 3D coordinates on a face in $\mathbb{R}^3$ corresponding to a point **q** within a polygon in the image plane.

transformation $\mathbf{T}$ which recovers the 3D point from its 2D projection depends upon the equation of the plane in $\mathbb{R}^3$ on which the face lies. Given this constraint, the 3D coordinates of a point can be computed.

The transformation $\mathbf{T}$ depends upon the description of the asp for a point in $\mathbb{R}^3$. Specifically, consider a fixed point $\mathbf{P} = (x, y, z) \subset \mathbb{R}^3$. The asp for $\mathbf{P}$ is computed by considering the image coordinates of $\mathbf{P}$ as a function of the viewpoint parameters θ and ϕ (see Figure 7). The projection onto the image plane can be separated into two parts: a rotation so that the viewing direction is along the z-axis, and the orthographic projection into the image plane. Each component of the rotation is described by an orthogonal 3×3 matrix. Under orthographic projection into the image plane (u, v) we have

$$u = x \cos\theta - z \sin\theta \tag{8}$$

$$v = x \sin\theta \sin\phi + y \cos\phi + z \cos\theta \sin\phi$$

These equations specify the image coordinates (u, v) of $\mathbf{P}$ as a function of the two viewpoint parameters θ and ϕ.

The problem of recovering the 3D coordinates of a point given only the value of the two viewpoint parameters for the orthographic projection and the image coordinates (u, v) is underconstrained; there are three equations and four unknowns. With the equation of the plane in $\mathbb{R}^3$ on which the recovered point is known to lie, the linear transformation $\mathbf{T}$ is fully determined.

Shading a polygon in the image plane using the Phong shading model involves applying the Phong shading equation (Eq. 7) at each point within the polygon. By using the transformation $\mathbf{T}$, each point in the image plane can be associated with a corresponding point which projects (orthographically) to it from a face in $\mathbb{R}^3$. Figure 8 shows the transformation applied to a point $\mathbf{q}$ in the image plane in order to recover the coordinates of the point $\mathbf{T}(\mathbf{q}) \subset \mathbb{R}^3$. Note that the equation of the plane in $\mathbb{R}^3$ which contains the triangular face must be known in order to apply the transformation. Bilinear interpolation can then be used to approximate the normal to the curved surface at that point. This approximated normal is the normal used in Eq. 7 to compute the appropriate intensity value for the point $\mathbf{q}$ in the image plane. Although the Phong model is more expensive than the flat shading model, there are fast approximations to the Phong shading model [Bish86, Duff79].

5. Shadows

When the position of the viewer is different from the position of the light source, regions in the scene can lie in shadows. A shadow on a surface results from the occlusion of the light source by some other surface in the scene. Assuming opaque objects, the light source illuminates the first surface it reaches. The surfaces hidden behind those which are illuminated lie in shadow. Note that when the light source position and the viewpoint coincide, there are no shadows. This is true since in that case the set of

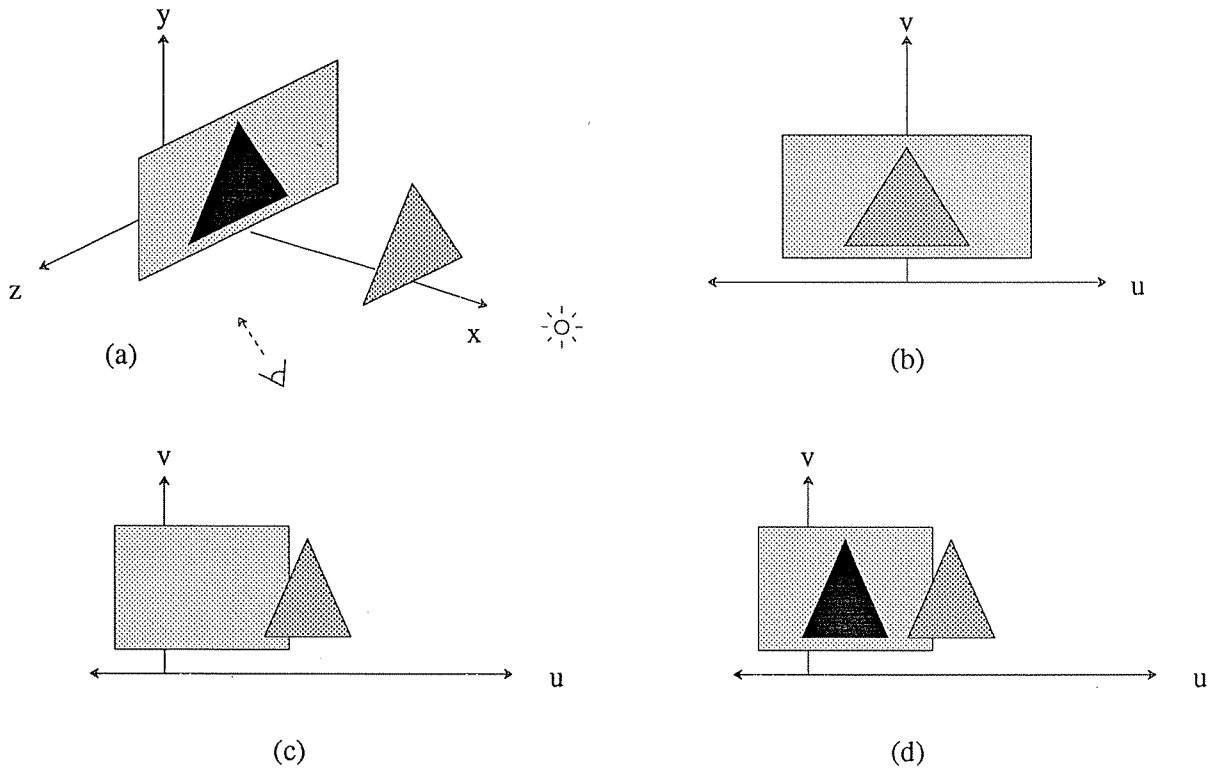


Figure 9. A 3D scene containing a single point light source. (a) The 3D scene. (b) The scene projected from the light source direction. (c) The scene as observed from the viewer's position without the shadow. (d) The scene observed by the viewer with the shadow.

visible surfaces corresponds identically to the set of illuminated surfaces.

The problem of accurately rendering a shaded scene which contains shadows involves finding the shadow regions in the scene given a viewpoint and the position of each light source. With scenes containing only a single light source, the problem is that of determining the set of shadow regions. Shadow region computation in polyhedral scenes has been incorporated into hidden surface algorithms using shadow volumes [Crow77] and polygon clipping [Athe78]. Most hidden-surface removal algorithms can compute shadow regions by performing hidden-surface removal from the position of the light source [Whit88]. Rendering can then be performed by integrating the shading of the shadow regions and the visible surfaces according to some shading model. These methods of removing hidden-surfaces and

finding shadow regions are not designed for interactive viewing. Using these algorithms for rendering a sequence of views of a scene would be very inefficient since these methods do not make use of viewpath coherence, and are not designed for interactive application.

Scenes which contain multiple light sources generate a set of shadow regions for each light source. The interaction between these sets of shadow regions must be computed, i.e., a surface can simultaneously lie in the shadow created by one light source and in the direct illumination of another. The shading of such a surface requires further computation according to the interaction of the sets of shadow regions associated with each of the individual light sources. Scenes of this complexity have been rendered using ray tracing techniques which create shadow regions as a by-product of light-source and ray intersections [Whit80, Cook84].

The asp represents the appearance of each face in the scene from all viewing directions. By treating each light source as an additional viewing direction, the asp can be used to obtain the appearance of a particular face from the light source position. The appearance of a face from the direction of the light source is the part of the face which is under direct illumination. A comparison of the appearances of a face from the viewing direction and from the light source direction gives the location of the shadows on the face which are cast from the light source. As an example consider the illustration in Figure 9. The 3D positions of the triangular and rectangular faces are illustrated in Figure 9(a). The triangular face casts a shadow on part of the rectangular face. Figure 9(b) shows the appearance of the scene from the light source direction. From the position of the viewer, however, the appearance of the scene is different. As shown in Figure 9(c), from direction of the viewer the rectangular face is only partially occluded. Regions visible from the light source are projected from the viewer's position thus identifying regions of full illumination and shadow (Figure 9(d)).

Consider the problem of using the asp to compute the regions of a scene which lie in the shadows created by a single light source. Let $\mathbf{L}$ and $\mathbf{V}$ represent the light source position and the viewer position in viewpoint space, respectively. Let $\mathbf{f}$ be a face in the scene which lies on plane $P \subset \mathbb{R}^3$. Let $A_{\mathbf{f}_V}$ be the appearance of $\mathbf{f}$ from the viewing direction $\mathbf{V}$, and let $A_{\mathbf{f}_L}$ be the appearance of $\mathbf{f}$ from the direction of the light source $\mathbf{L}$. The appearance of a polygon as encoded in the asp is always a set of closed polygons. Note that because of occlusion, a single polygon may actually appear to be a set of disjoint polygons. Thus $A_{\mathbf{f}_L}$ and $A_{\mathbf{f}_V}$ represent sets of polygons in the image plane. Furthermore, $A_{\mathbf{f}_L}$ represents the part of $\mathbf{f}$ which is illuminated by the light source $\mathbf{L}$, and $A_{\mathbf{f}_V}$ represents the part of $\mathbf{f}$ which is visible to the viewer from the viewpoint $\mathbf{V}$.

The computation of regions of the face $\mathbf{f}$ which lie in shadow hinges on a key observation: the polygons in each of the sets $A_{\mathbf{f}_V}$ and $A_{\mathbf{f}_L}$ can be thought of as the orthographic projection of disjoint polygons in $\mathbb{R}^3$ which lie on the same plane containing $\mathbf{f}$. That is, a transformation can be applied to the

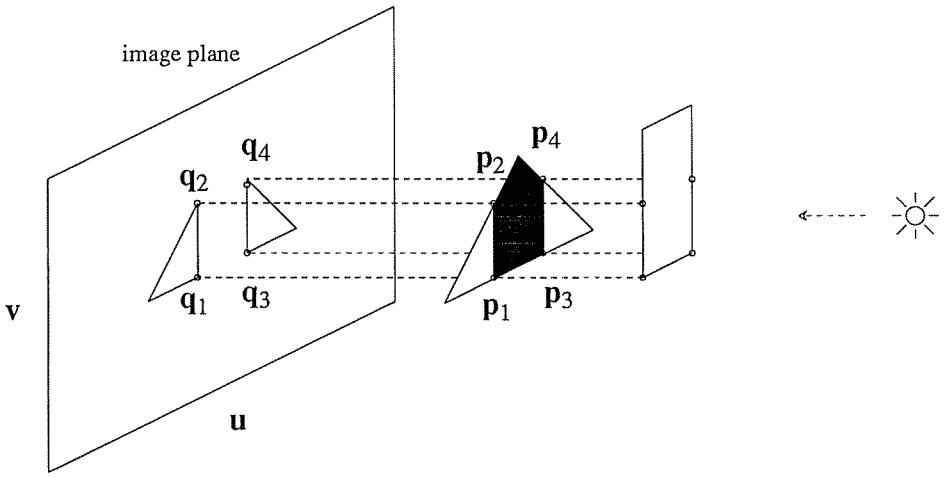


Figure 10. The regions of shadow and full illumination on the triangular face can be computed using the appearance of the face in the image and the transformation **T**.

edges and vertices in the image plane for both $A_{\mathbf{f}_V}$ and $A_{\mathbf{f}_L}$ in order to recover the corresponding 3D coordinates for the image edges. The 3D polygons which correspond to the transformation applied to both $A_{\mathbf{f}_V}$ and $A_{\mathbf{f}_L}$ are also sets of closed polygons which lie on plane P in $\mathbb{R}^3$. This is important in order to compare the regions of $\mathbf{f}$ which are visible, to the regions of $\mathbf{f}$ which are fully illuminated. Let $\mathbf{T}(A_{\mathbf{f}_V})$ and $\mathbf{T}(A_{\mathbf{f}_L})$ represent the sets of polygons in $\mathbb{R}^3$ on plane P corresponding to the necessary transformation applied to $A_{\mathbf{f}_V}$ and $A_{\mathbf{f}_L}$, respectively. Then the intersection $\mathbf{T}(A_{\mathbf{f}_L}) \cap \mathbf{T}(A_{\mathbf{f}_V})$ represents the regions in $\mathbb{R}^3$ on $\mathbf{f}$ which are both visible and illuminated. The regions described by $\mathbf{T}(A_{\mathbf{f}_V}) - \mathbf{T}(A_{\mathbf{f}_L})$ represent those areas on $\mathbf{f}$ which are visible but not illuminated, i.e., shadow regions.

The computation of the intersection and difference of these sets of polygons starts with the transformation **T**. **T** can be applied to every edge bounding the projection of a face in the scene from a particular viewpoint. This yields a set of polygons in $\mathbb{R}^3$ on the face. $\mathbf{T}(A_{\mathbf{f}_L})$ is the set of polygons on $\mathbf{f} \subset \mathbb{R}^3$. Figure 10 shows the appearance of a triangular face from the direction of the light source in the image plane. The points $\mathbf{q}_1$, $\mathbf{q}_2$, $\mathbf{q}_3$ and $\mathbf{q}_4$ are vertices in the image plane which are a result of the occlusion of the triangular face by the rectangular face. By applying **T** to the vertices $\mathbf{q}_i$ in the image plane, the points $\mathbf{p}_1$, $\mathbf{p}_2$, $\mathbf{p}_3$ and $\mathbf{p}_4$ are recovered on the triangular face in $\mathbb{R}^3$. Notice that the edge between $\mathbf{p}_1$ and $\mathbf{p}_2$ as well as the edge connecting $\mathbf{p}_3$ and $\mathbf{p}_4$ are the boundaries of the region on the triangular face which

separate the regions of shadow and full illumination.

In order to compute the intersection $\mathbf{T}(A_{\mathbf{f}_L}) \cap \mathbf{T}(A_{\mathbf{f}_V})$, the 2D appearance of the face $\mathbf{f}$ from the light source viewpoint can be recovered as a polygon in $\mathbb{R}^3$ which lies on the plane containing $\mathbf{f}$ in $\mathbb{R}^3$. This polygon can then be reprojected (orthographically) from the viewpoint of the viewer. The two sets of polygons can be compared directly in the image plane corresponding to the viewpoint of the viewer. Recall that for a single point $\mathbf{q} \subset \mathbb{R}^2$ the transformation $\mathbf{T}(\mathbf{q}) \subset \mathbb{R}^3$ is the image of $\mathbf{q}$ constrained to lie on plane P. Then the point $\mathbf{T}(\mathbf{q})$ can be reprojected so that it can be directly compared to the appearance of the face $\mathbf{f}$ from the viewing direction $\mathbf{V}$. Transforming the appearance of $\mathbf{f}$ from the direction of the light source in this way corresponds to projecting the way fully-illuminated regions of $\mathbf{f}$ will appear to the viewer from viewpoint $\mathbf{V}$.

For scenes with a single light source, the regions of full illumination and the regions of shadow which may divide the appearance of a face $\mathbf{f}$ can be found by applying the above transformation and then performing the intersection and difference operations

$$\mathbf{S}_1 = \mathbf{T}(A_{\mathbf{f}_V}) \cap \mathbf{T}(A_{\mathbf{f}_L}) \qquad \mathbf{S}_2 = \mathbf{T}(A_{\mathbf{f}_V}) - \mathbf{T}(A_{\mathbf{f}_L}) \tag{9}$$

The set of polygons $\mathbf{S}_1$ is the region of full illumination; $\mathbf{S}_2$ is the remaining part of the visible portion of $\mathbf{f}$, that is the part of $\mathbf{f}$ in shadow.

6. Interactive Viewing with Shadows and Multiple Light Sources

The precomputation of the visual events from the asp makes explicit all of the visual events which occur in viewpoint space. The intersection of an interactive viewpath with these precomputed visual events in viewpoint space allows the efficient computation of successive frames in the sequence with hidden-surfaces removed. This efficiency is a result of both the viewpath coherence and the visual event precomputation. Because the change in the visibility of the surfaces in the scene is represented, the dominant cost of hidden-surface interactive viewing becomes the cost of applying the shading model and the scan conversion of sets of visible surface and shadow polygons.

The on-line algorithm for interactive viewing computes the appearance of the scene from the starting viewpoint using a standard hidden-surface removal algorithm. As the viewpoint parameter changes, visual events along the viewpoint path occur. With each visual event is a description of how the appearance of visible faces in the scene change from the previous appearance. Hence, for every event an update to the current appearance of the visible faces is made. The second frame in the sequence is computed from the first by determining the visual events which occur under the small change in viewpoint, and then modifying the appearance of each face given the events. The viewpath coherence which results from smoothly changing viewpoints guarantees that only a small number of visual events will need to be processed between any two frames of the sequence.

These operations can be accomplished using two data structures: a list of visual events and an Edge Structure Graph (ESG). The edge structure graph is a graph of faces, edges and vertices which describes the appearance of the scene for a particular viewpoint. The ESG is constructed initially for the first frame of the viewpath, and is modified for each visual event thereafter. The visual event list is generated from the asp, and contains a complete, sorted list of visual events and corresponding updates to the ESG. The construction of both the visual event list and the initial state of the ESG is done off-line.

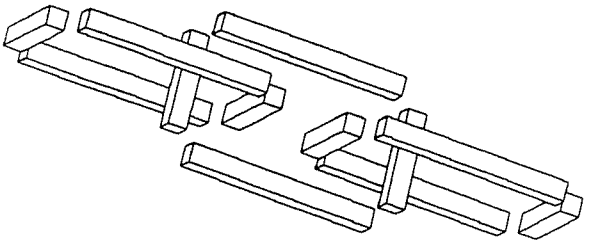
The interactive viewing of a shaded polyhedral scene is a direct application of the precomputed visual event list and the ESG. Consider a scene with fixed light sources and flat-shading. Intensities for each face are precomputed and stored in a lookup table, and shadow regions for each of the light sources can be similarly precomputed. For a single frame of the sequence the task of rendering the scene with shadows and with hidden surfaces removed is a three-step procedure. First, the appearance of each face in the scene from the current viewpoint is computed using the visual event list and the ESG. Second, each shadow set is transformed using the linear transformation discussed in Sections 4 and 5 according to the current viewpoint parameters. Finally, the appearance of the scene is rendered by combining the shadow sets and the flat-shading value for each face.

Interactive viewing which includes moving light sources amounts to the additional precomputation of the visual event list and ESG along each light source path. The changes in the polygons in the shadow set for a light source can then be computed efficiently by updating the ESG according to the visual event list. For a static viewer and moving light source, the algorithm is the same as above. For both moving viewer and moving light source, two distinct ESGs and visual event lists must be constructed.

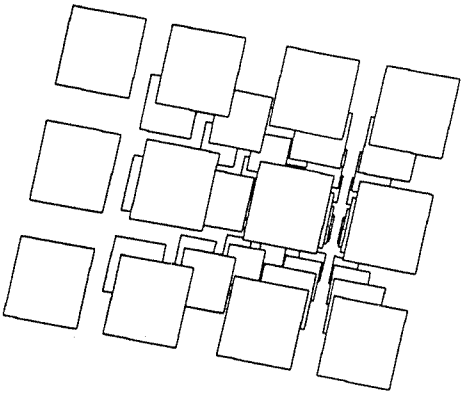
Interactive viewing using an interpolated shading model requires that a distinct intensity value be computed for each point inside a given face. Interpolated shading and fast Phong shading can be done relatively quickly [Swan86, Bish86], and a scan line algorithm can again be used to combine information from shadow sets and shaded visible surfaces in the image plane.

7. Results

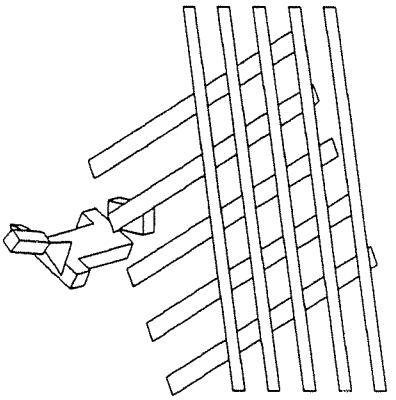
A prototype of both the preprocessing and the on-line portions of the hidden line algorithm has been implemented. The prototype program is written in C and was tested on a Vax 3100 workstation running Ultrix V2.1. The input to the algorithm is a polyhedral model of a 3D scene represented as a graph structure. A model is a collection of faces in which each face is bounded by a single closed cycle of edges and each edge is bounded by two vertices. The six test models used in the timing experiments are shown in Figure 11. These models were chosen to represent two basic categories of objects: those which approach the worst-case behavior of the asp construction algorithm, and those representing more realistic objects. The three chain links (Model 1), the layers of squares (Model 2), and the solids behind a grid (Model 3) contain a great deal of nonconvexity and are visually complex. The candlestick (Model 4), the



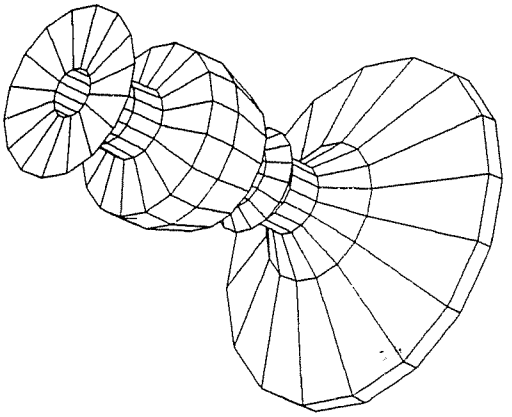
Model 1: Three Chain Links



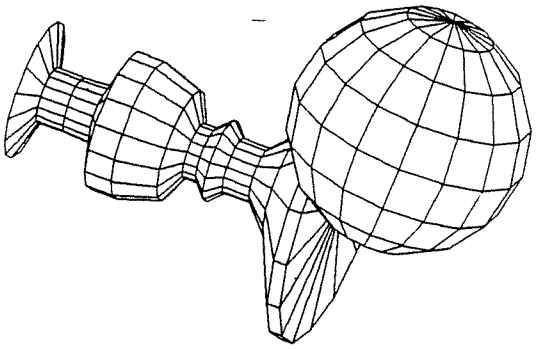
Model 2: Layers of Squares



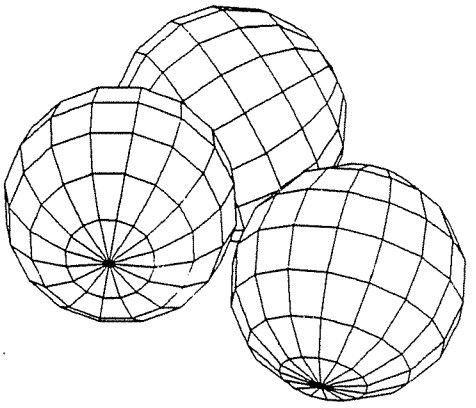
Model 3: Solids Behind Grid



Model 4: Candlestick



Model 5: Candlestick and Sphere



Model 6: Three Spheres

Figure 11. Six polyhedral models of moderate visual complexity.

candlestick and sphere (Model 5), and the three spheres (Model 6) illustrate the intra-object occlusion created by nonconvexity as well as inter-object occlusion from multiple objects in a scene.

Timing values and sizes for the construction phase of the algorithm for the six models are shown in Table 1. The numbers of faces, edges and vertices in each of the models are recorded in the first three columns of the table. For each of the selected model orientations, the construction time (in CPU seconds) and the resulting size (in Kbytes) are shown. The visual events computed for the edges of the models include three types: the appearance and disappearance of an edge, EE-events (two edges forming a T-junction), and EEE-events (three edges). The maximum, average and total number of events shown in Table 1 are the sum of these three types of events. Thus the minimum number of events that a model with $\mathbf{n}$ edges can have is $2\mathbf{n}$, i.e., two events for the appearance and disappearance of each edge.

Several conclusions can be drawn from the data shown in Table 1. First, the construction times and sizes indicate that the construction of the asp for models with many more faces is indeed tractable. Although the off-line computation time for the largest model, model 5, 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 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. Several models in Table 1 were constructed to illustrate the potential worst-case behavior of the asp construction algorithm. Note that the average number of events per edge for Models 1-3 (7.0 - 17.2) tends to be higher than the average number of events per edge for Models 4-6 (2.1 - 7.2). Even so, the number of visual events is only an order of magnitude larger than the number of edges. 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 the 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.

Timing measurements for the on-line interactive viewing are shown in Figure 12. 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. The viewpath was a great circle (rotation) about the y-axis. As shown in Figure 12, the display rates of Models 1-3 were measured for three distinct orientations. That is, a complete asp and animation sequence was computed for these models in each of three orientations. The -30 and +30 degree orientations were generated by rotating the model about the z-axis. The 0 degree orientation, chosen to illustrate a significant amount of occlusion, is the orientation pictured in Figure 11. These three orientations were also used for Models 4-6 in addition

Aspect Representation Construction Information									
Model	Faces	Edges	Vert.	Orient.	Time (Sec)	Size (Kbytes)	Max Events per edge	Ave Events per edge	Total Events
Model 1	72	144	96	-30	19	31	46	9.4	1360
				0	14	27	35	8.6	1238
				+30	9	23	26	7.0	1009
Model 2	48	192	192	-30	11	35	32	16.7	3214
				0	9	31	31	14.9	2870
				+30	12	36	39	17.2	3300
Model 3	131	105	82	-30	13	54	77	17.1	1799
				0	13	51	60	15.9	1674
				+30	12	46	57	15.2	1593
Model 4	288	560	274	-30	201	94	31	6.2	3450
				0	190	94	28	6.2	3457
				+30	206	95	32	6.3	3512
				Base	66	39	42	2.8	1586
Model 5	416	800	388	-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
Model 6	384	720	342	-30	218	136	26	6.1	4391
				0	196	128	24	5.4	3896
				+30	149	95	24	4.8	3420
				Base	59	37	9	2.1	1522

Table 1. The table shows construction information for models 1, 2, and 3 in three different orientations. The construction information for models 4, 5, and 6 includes four orientations, the base orientation being one with no occlusion. Visual events include the appearing and disappearing of an edge, edge-edge events (EE) and edge-edge-edge (EEE) events.

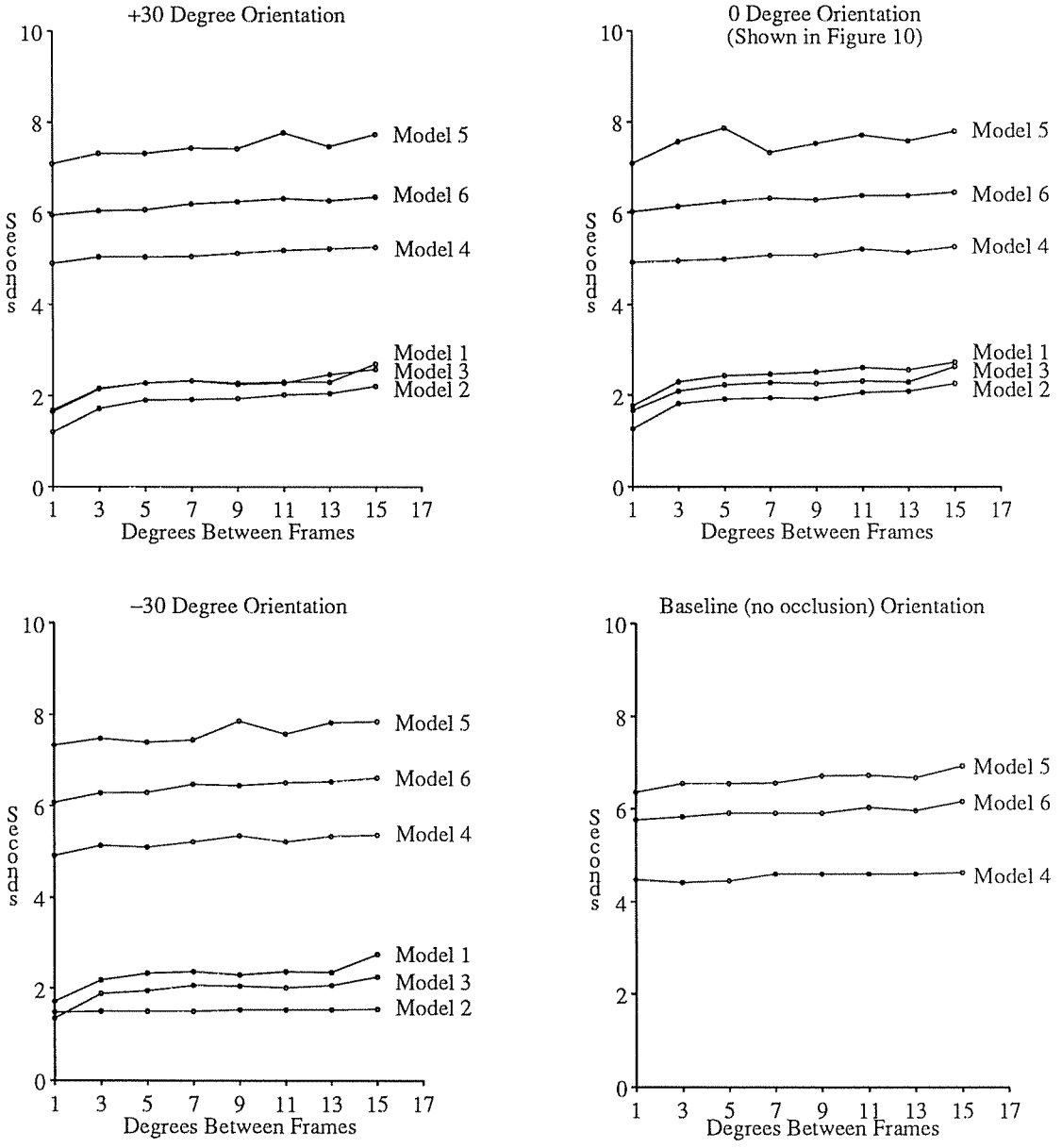


Figure 12. Each graph shows display times (in seconds) for 360 frames of a rotation sequence. The horizontal axis shows the degrees between each frame of the sequence. The graph in the lower right shows display times for models 4, 5, and 6 in orientations which produce almost no occlusion.

to a baseline orientation. The baseline orientation was chosen so that no occlusion occurs along the viewpath. Without occlusion, the number of visual events is minimized. Thus, as shown in Table 1, the average number of visual events for the base orientation is between 2 and 3, where 2 is the lower bound.

The results shown in Figure 12 lead to several observations. First, a small change in the number of degrees between frames in the 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. Third, in comparing the display times of Models 4-6 in the base orientation to the other orientations, the display times are similar. For example, the time to display a sequence of 360 frames for Model 5 in the -30 degree orientation is about 7.2 seconds compared to 6.2 seconds for the base orientation. This amounts to a frame rate of 58 frames/sec as opposed to 50 frames/sec. Yet there are almost three times as many visual events to be processed in the -30 degree orientation!

In order to verify and 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 Edge Structure Graph (ESG). Figure 13 shows the results of this analysis for Models 1, 4, 5 and 6. In particular, note that for each model the time spent on event processing is less than 5% of the total time for increments less than 15 degrees. As expected, larger increments between frames imply less viewpath coherence and therefore the time spent processing events increases. This behavior corresponds to the positive slope of the line representing the trends in each of the graphs in Figure 13.

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 fast 3D 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 so small, the interactive viewing of much larger scenes can be achieved using the asp.

Our results indicate that the asp can be used to achieve a display rate which is fast enough to allow interactive movement of the viewer position with scenes containing at least 2,000-10,000 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

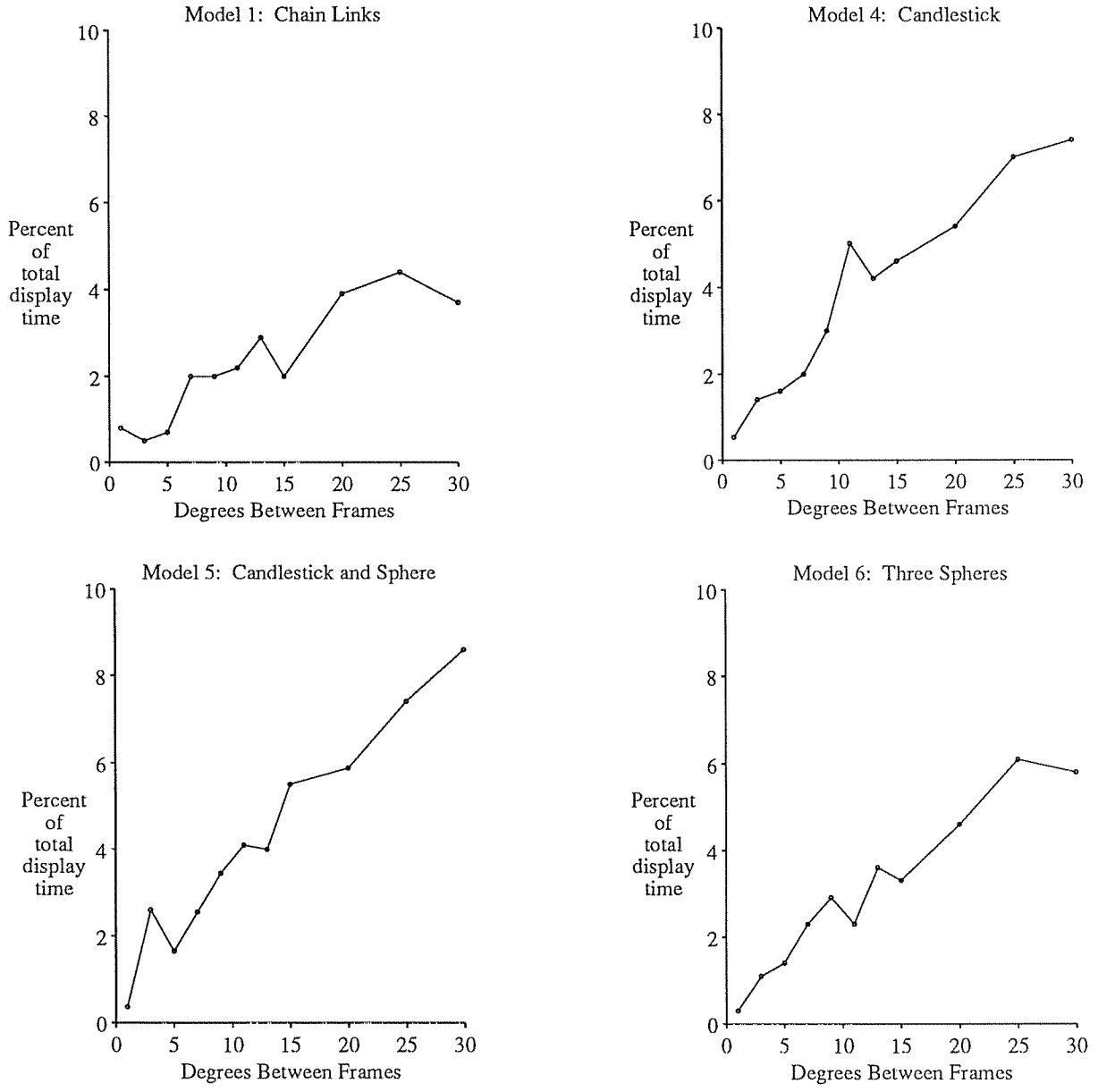


Figure 13. Each graph shows the percentage of the total display time spent on visual event processing. For small (less than 15 degrees) increments between frames in the sequence the event processing consumes less than 5 percent of the total display time.

- Frame display time is stable for less than 15 degree increments between frames
- Visual event processing requires less than 5 percent of the total display time.

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, on-line display of large models can be achieved when preprocessing time is available.

The interactive viewing of shaded scenes with multiple light sources and shadows is a direct extension of this implementation. The key feature of this approach is the separation of the hidden surface computation in the animation sequence from the application of the shading model and the scan conversion process. The set of object resolution polygons representing the visible faces and the shadow polygons is computed efficiently using the visual event list derived from the asp. Viewpath coherence guarantees that only a small percentage of the visual events need to be processed between a single pair of frames. The difference between hidden-line and hidden-surface computation in this approach is the edge connectivity information which must be preserved in the asp, as well as the cost of raster display. The primary cost of the shaded display is the cost associated with scan converting the visible surfaces and shadow polygons.

8. Extensions

The algorithm can be extended to viewpaths which are not simple great circles. Still using the orthographic model, it is possible to compute the asp along any viewpath such that the intersection of that path and the 4D volumes of the asp can be found in closed form. If asp surfaces and their intersections can be represented, the asp can be constructed.

The algorithm can also be extended to perspective projection. In the perspective case it is necessary to be able to find the intersection of the path of viewpoints and the surfaces in aspect space under a perspective projection, requiring a straightforward change in the equations listed above. This would be useful, for example, in interactively displaying the appearance of a scene generated by a viewpath moving through a workspace, such as a model of a building [Broo86]. However, depending on the problem, aspect space may be high dimensional and therefore the asp may require more space and time to compute. For related work, see [Plan86, Plan87, Plan88].

This paper presents an efficient algorithm for interactively viewing a polyhedral scene. The algorithm takes advantage of viewpath coherence, a form of frame-to-frame coherence, which is inherent in the sequence of images generated by a moving viewer along a continuous viewpath. The appearance from all viewpoints is computed in a preprocessing phase that works by constructing the asp for the scene. The preprocessing phase also involves computing the initial appearance of the scene with a standard hidden-line or hidden-surface removal algorithm. The on-line phase involves the display of views of

the scene with hidden lines or hidden surfaces removed as the user interactively specifies movement in viewpoint space.

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 1,000 to 100,000 edges. In addition, there must be sufficient preprocessing time and sufficient processing speed to handle the on-line phase.

References

[Athe78] Atherton, P., K. Weiler, and D. Greenberg, "Polygon shadow generation," *Proc. SIGGRAPH*, pp. 275-281 (1978).

[Bish86] Bishop, G. and D. M. Weimer, "Fast phong shading," *Proc. SIGGRAPH*, pp. 103-106 (1986).

[Broo86] Brooks, F., "Walkthrough – A dynamic graphics system for simulating virtual buildings," *Proc. Workshop on Interactive 3D Graphics*, pp. 9-21 (1986).

[Cook84] Cook, R. L., T. Porter, and L. Carpenter, "Distributed ray tracing," *Computer Graphics* **18** pp. 137-145 (July 1984).

[Crow77] Crow, F. C., "Shadow algorithms for computer graphics," *Proc. SIGGRAPH*, pp. 242-248 (1977).

[Denb86] Denber, M. and P. Turner, "A differential compiler for computer animation," *ACM Computer Graphics* **20**(4) pp. 21-27 (1986).

[Duff79] Duff, T., "Smoothly shaded renderings of polyhedral objects on raster displays," *ACM Computer Graphics* **13**(2) pp. 270-275 (1979).

[Farr85] Farrell, E., W. Yang, and R. Zappulla, "Animated 3D CT imaging," *IEEE Computer Graphics and Applications* **5**(12) pp. 26-32 (1985).

[Fole82] Foley, J. D. and A. VanDam, *Fundamentals of Interactive Computer Graphics*, Addison-Wesley, Reading, Mass. (1982).

[Fuch83] Fuchs, H., G. D. Abram, and E. D. Grant, "Near real-time shaded display of rigid objects," *Computer Graphics* **17** pp. 65-69 (July 1983).

[Glas88] Glassner, A., "Spacetime ray tracing for animation," *IEEE Computer Graphics and Applications* **8**(2) pp. 60-70 (1988).

[Horn77] Horn, B. K. P., "Understanding image intensities," *Artificial Intell.* **21** pp. 201-231 (1977).

[Hubs81] Hubschman, H. and S. W. Zucker, "Frame-to-frame coherence and the hidden surface computation: Constraints for a convex world," *Computer Graphics* **15** pp. 45-54 (1981).

[Phon75] Phong, B. T., "Illumination for computer generated pictures," *Comm. ACM* **18** pp. 311-317 (1975).

[Plan87] Plantinga, H. and C. R. Dyer, "The asp: A continuous viewer-centered representation for 3D object recognition," *Proc. 1st Int. Conf. Computer Vision*, pp. 626-630 (1987).

[Plan86] Plantinga, W. H. and C. R. Dyer, "An algorithm for constructing the aspect graph," *Proc. IEEE Symp. Foundations of Computer Science*, pp. 123-131 (1986).

[Plan88] Plantinga, W. H., "The asp: A continuous, viewer-centered object representation for computer vision," Technical Report 784, University of Wisconsin-Madison (August 1988).

[Plan89] Plantinga, W. H., C. R. Dyer, and W. B. Seales, "Real-time hidden-line elimination for a rotating polyhedral scene using the aspect representation," Technical Report 89-3, University of Pittsburgh, Pittsburgh, Pa. (1989).

[Shel82] Shelley, K. and D. Greenberg, "Path specification and path coherence," *Computer Graphics* **16**(3) pp. 157-166 (1982).

[Swan86] Swanson, R. W. and L. J. Thayer, "A Fast-shaded polygon renderer," *Proc. SIGGRAPH*, pp. 95-101 (1986).

[Whit80] Whitted, T., "An improved illumination model for shaded display," *Comm. ACM* **23** pp. 343-349 (1980).

[Whit88] Whitted, T. and R. Cook, "A comprehensive shading model," pp. 232-243 in *Computer Graphics: Image Synthesis*, ed. K. Joy, C. Grant, N. Max and L. Hatfield,IEEE Computer Society Press, Washington, D.C. (1988).

[Yan85] Yan, J., "Advances in computer-generated imagery for flight simulation," *Computer Graphics and Applications* **5**(8) pp. 37-51 (1985).