# Efficiently Computing and Representing Aspect Graphs of Polyhedral Objects *

Ziv Gigus, John Canny and Raimund Seidel
Computer Science Division, EECS
University of California, Berkeley, CA 94720

## Abstract

The *aspect graph* is one of the approaches to representing of 3-D shape for the purposes of object recognition. In this approach, the viewing space of an object is partitioned into regions, such that in each region the topology of the line drawing of the object does not change. The *viewing data* of an object is the partition of the viewing space together with a representative view in each region. We present an efficient algorithm for computing the viewing data for line drawings of polyhedral objects under orthographic projection. For an object of size $O(n)$ whose partition of size $O(m)$, the algorithm runs $O(n^4 \log n + m \log m)$ time. Using a novel data structure, we construct the set of all views in optimal $O(m)$ time and space.

*This research was supported by the Semiconductor Research Corporation grant number 82-11-008.

# Contents

# 1 Introduction

One of the approaches for representing three dimensional objects for the purpose of object recognition is to compute a finite set of two dimensional views of the object from different viewpoints, and match the image against this set. To generate the set of views, the viewpoint space is partitioned into a finite number of regions, and a representative view is selected in each region. There are two approaches to the partitioning of the viewpoint space: (1) a uniform, object independent partitioning, where the number of regions and their shape is fixed in advance and (2) the *aspect graph* approach. A more detailed description of these approaches, and a discussion of their advantages and disadvantages can be found in [GM88a].

In the aspect graph approach, we define a qualitative measure of the structure of the image - its *aspect*, and the viewpoint space is partitioned into maximal regions, such that the views from all viewpoints in a region have the same *aspect* (the definition of the aspect is dependent upon the particular application). At the boundary between adjacent regions, the aspect of the view changes and a *visual event* is said to occur. The partition of the viewpoint space for a given object, together with the view in each region is defined to be the *viewing data* of the object.

We present an algorithm for constructing the partition of the viewing space of polyhedral objects under orthographic projection, where the aspect is defined to be the topological structure of the line drawing. The algorithm runs in $O(n^4 \log n + m \log m)$ time, where $m$ is size of the partition and $n$ is the number of vertices of the object. We also present a method for computing all the views of the partition in time and space that are linear (and thus optimal) in the total number of changes that occur in the views, at the boundaries of the partition.

# 2 The Viewing Data of a Polyhedral Object

We describe how to compute the viewing data of polyhedral objects under orthographic projection. An object is assumed to have $n$ vertices, and therefore has $O(n)$ edge and $O(n)$ faces. We assume that an object has no surface markings, and therefore every line in a line-drawing of the object is the projection of a part of an edge.

The *image structure graph* (ISG) of a viewpoint is the labeled planar graph that corresponds to the line drawing of the object as seen from that viewpoint. For each junction in the line drawing there is a corresponding vertex in the graph, and for each line segment there is an edge between the vertices that correspond to the endpoints of the line segment. Each vertex in the graph is labeled by the names of the edges of the object whose projections meet at the vertex, and each edge is labelled by the labels of its endpoint vertices. We define the *aspect* of a viewpoint to be the topological structure of ISG of that viewpoint. That is, two different viewpoints have the same aspect if and only if the corresponding ISGs are isomorphic.

A *general* viewpoint is a viewpoint for which there exists an open neighborhood of viewpoints that have the same aspect. In other words, when the viewer moves inside this neighborhood, metric properties of the line drawing change, but its graph structure remains the same. A viewpoint that is not general is *accidental.*

The viewing space of the orthographic projection is the space of viewing direction. We represent this space by a cube that is centered around the origin, and whose edges are parallel to the major axes and are of length two. A point on the cube corresponds to the (non-normalized) viewing direction with the same coordinates. We refer to this cube as the *viewing cube*, and to points on the cube as the viewpoints of the orthographic projection.

The viewing cube is partitioned into open regions of general viewpoints, such that all viewpoints in a region have the same aspect, but the aspects of viewpoints in adjacent regions are different (the corresponding ISGs are not isomorphic). We use the term *view* to refer to the representative ISG for a given regions. The boundaries between these regions are curves of accidental viewpoints. All viewpoints on a curve segment between adjacent regions have the same aspect. Where several regions share a boundary point, two or more curves meet resulting in a vertex. The aspect at the vertex is different from that of any of the viewpoints in its neighborhood. In other words, this partition has the structure of a planar graph that is embedded on the viewing cube, where the vertices, arcs and faces of the graph are the vertices, curve segments and regions, respectively.

As the viewpoint moves from a region to its boundary, or from a boundary curve to an endpoint vertex, the view changes—a *visual event* occurs.

The partition of the viewing cube together with a view in each region is defined to be the *viewing data* of the object (this definition follows a similar

definition by Callahan and Weiss [CW85]).

# 3 Review of Past Work

The aspect graph approach was introduce by Koenderink and van Doorn [KvD79]. For smooth objects, some of the local visual events and the location of the corresponding accidental viewpoints were first described by Koenderink and van Doorn [KvD76]. A complete catalog of the local visual events and the location of the corresponding viewpoints is provided in related papers [Arn79,Arn83,Ker81]. Recently, Rieger [Rie87] has published a catalog of the visual events for piecewise smooth objects that contain no planar edges. Callahan and Weiss [CW85] suggested the viewing data representation and gave examples of the viewing data of a few simple smooth objects. There is no published algorithm for computing the viewing data for smooth or piecewise smooth objects.

Chakravarty and Freeman [CF82] used the aspect graph approach in the *characteristic views* representation of objects. They used heuristic constraints on the orientation of the object with respect to the camera, to select a subset of the set of views as the representation for the object. The views and the corresponding regions of the partition were computed manually.

Object recognition systems that were build by Ikeuchi [Ike87] and by Kanade and Hebert [HK85] used the aspect graph approach as the basis of the object representation. In the first system the aspect is defined by to the set of faces that are detectable by photometric stereo, and in the second system the aspect is defined by the set of occluding edges in the image. In both system, the partition of the viewing space was approximated by computing a set of views in a uniform partition, and then merging neighboring regions that have the same aspect.

Werman, Baugher and Gualtieri [WBG86] present an algorithm for constructing the aspect graph of a convex polygon as viewed from viewpoints in the plane of the polygon, using perspective projection.

Stewman and Boyer [SB87] describe an algorithm for constructing the viewing data of convex polyhedra under perspective projection.

In 1986 Plantinga and Dyer [PD86] presented an algorithm for computing the viewing data of polyhedral objects under orthographic projection, where an aspect is defined by the set of visible faces of the object. Be-

cause, under this definition, line drawings that are topologically different may correspond to viewpoints that share a common region in the partition (see Figure 1), this definition of the aspect is not appropriate for object recognition in line drawings, although it may be appropriate when range images are used.

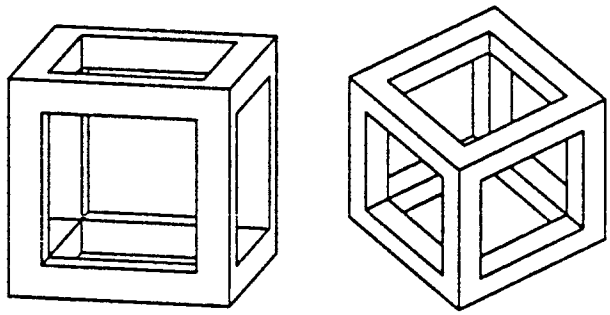

Figure 1: Two line drawings that are qualitatively different, but the same faces are visible in both views.

In [PD87] Plantinga and Dyer published a modified version of the same algorithm, and also described an extension of the algorithm for computing the viewing data of polyhedral objects under perspective projection. In this report the aspect is defined by the set of regions in the image. Visual events occur when the set of regions changes. The set changes when: (1) a current region disappears, (2) a new region appears, (3) a current region splits into two regions, or (4) two current regions merge into a single region. A change in the viewpoint that does not change the set of regions in the image but does change the boundary of some regions and the adjacency relation between regions is not considered a visual event. An example of such a change is illustrated in Figure 2. Under this definition the two images in Figure 1 do not have the same aspect. However, as this definition is not based on the topological structure of the line drawing, it still allows for viewpoints with different corresponding ISGs to be part of the same region. Thus, it is still inappropriate for object recognition in line drawings. Figure 3 is an example of line drawings of an object that are topologically different, and for which, under this definition of the aspect, the corresponding viewpoints belong to the same region in the partition.

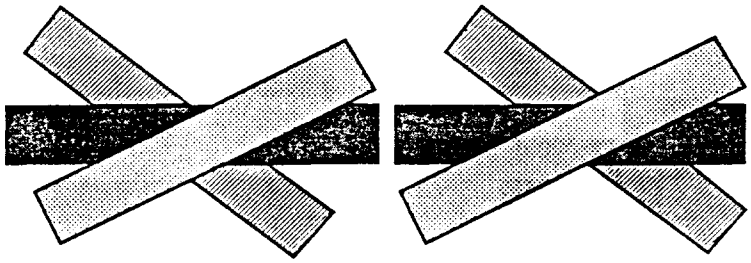

Figure 2: An example of a change in the image that is not considered a visual event under Plantinga and Dyer's definition of an aspect. This example is reproduced from [PD87].

In more recent versions of the same algorithm [Pla88], Plantinga and Dyer redefined the aspect according to graph isomorphism of the ISGs.

Plantinga and Dyer's algorithm computes the partition of the viewing space by computing a four dimensional "visibility volume" (5-D for the perspective case) for each face of the object, and intersecting these volumes to produce the visibility volume of the object. The boundaries of the partition are computed by projecting the boundaries of this visibility volume onto the viewing sphere and finding their intersection points. The algorithm computes the partition in $O(m \log m + T)$ time, where $m$ is the number of vertices in the partition, and $T$, which is bounded by $O(n^5)$, is the time to compute the visibility volume of the object. The views are computed in $O(n^2 m)$ time and space.

Gigus and Malik [GM88b] presented an algorithm for computing the viewing data for of polyhedral objects under orthographic projection, where the aspect is defined by the topological structure of the line drawing. They provided a full catalog of the changes that occur in the line drawing for each type of event. They also described how to generate the line drawing in a region, given the line drawing in an adjacent region and the event that occurs at the boundary between the regions. The algorithm performs all the computations in $\mathbb{R}^3$, however its running time is not sensitive to the size of the partition.

We describe how to compute the partition of the viewing space, using visibility tests that are performed directly in $\mathbb{R}^3$, without going into any higher dimensional spaces. The resulting algorithm is sensitive to the size

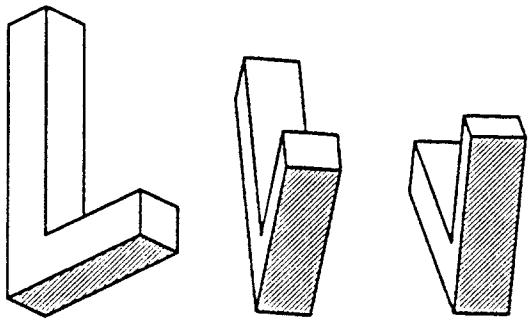

Figure 3: Line drawings of an L-shaped object that are qualitatively different but belong to the same region of the partition, when the aspect is defined by the set of regions in the image.

of the output, computing the partition in $O(n^4 \log n + m \log m)$ time. We compute all the views in optimal $O(m)$ time and space.

# 4 The Visual Events of Polyhedral Objects

In this section we describe the visual events of a polyhedral object. This is a review of a more detailed exposition that presented by Gigus and Malik in [GM88a].

For polyhedral objects there are two fundamental visual events: (1) the projections of an edge and a vertex coincide (an *EV event*), and (2) the projections of three non-adjacent edges intersect at one point in the image (a *EEE event*). The EV event can be considered as a special case of the EEE event, where we allow two of the edges to be adjacent, and thus think of the projection of the vertex as the intersection of the projections of these two edges. As the geometry of the locus of the accidental viewpoints for the EV event is much simpler than that of the EEE event, we prefer to consider them as different events and treat them separately.

# 5 The Potential Boundaries of The Partition

When a visual event occurs, a line of sight intersects all the features that participate in the event. We refer to this line as the *critical* line of sight. For the perspective projection, the accidental viewpoints of a given visual event lie on the family of critical lines the event. This family of lines defines parts of a ruled surface. Therefore, the accidental viewpoints also lie on this ruled surface. For the orthographic projection, the accidental viewpoints are the direction vectors of this family of lines.

A viewpoint $v$ that is on a critical line $L$ is potentially an accidental viewpoint of the visual event where the features that are intersected by $L$ interact. $v$ is an actual accidental viewpoints only when all the features are visible from $v$ along $L$. If any of the features is occluded then $v$ is not an accidental viewpoint of this event.

Given an edge and a vertex, or three edges, the locus of the corresponding potential accidental viewpoints defines a *potential boundary* of the partition. Some segments of the potential boundary (or all of it) may not participate in the partition of the viewing space. The segments that do not participate in the partition are those that contain viewpoints from which some of the features that participate in the corresponding visual event are occluded.

Below, we describe the locus of the potential boundaries that correspond to the two fundamental events. In subsequent sections we describe how to compute which parts of a potential boundary actually participate in the partition.

## 5.1 The Locus of Accidental Viewpoints for the EV event

Let the vertex and the edge be $v$ and $e = (a, b)$, respectively. The viewing directions from which the projections of $v$ and $e$ coincide are either a convex combination of $\overrightarrow{a - v}$ and $\overrightarrow{b - v}$ or of $\overrightarrow{v - a}$ and $\overrightarrow{v - b}$. On the viewing cube, these are two diametrically opposite polygonal curves that are part of the polygon that is defined by the intersection of the cube with a plane that

goes through the origin $\overrightarrow{a-v}$ and $\overrightarrow{b-v}$. We refer to the polygonal curve from which $e$ is in front of $v$ ($\overrightarrow{a-v} \rightarrow \overrightarrow{v-b}$) as the *front polygonal curve* and to the opposite polygonal curve as the *back polygonal curve*. We use the term *EV-boundary* to refer to either the front or the back polygonal curve. See Figure 4.

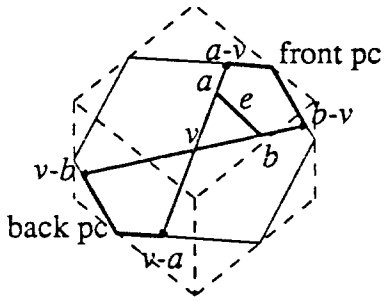


Figure 4: The polygonal curves of accidental viewpoints of an EV event. The dashed lines indicate the front faces of the viewing cube.

In the perspective case, the viewpoints from which the projections of $e$ and $v$ coincide, are on the plane that contains $v$ and $e$. We refer to this plane as the plane of the EV event. Not all viewpoints on this plane are accidental viewpoints of this events, only the viewpoints that lie on the critical lines, and are not between $v$ and $e$, are accidental viewpoints for this event.

For a face $f$, the collection of EV-boundaries for events between the vertices and the edges of $f$ forms a closed planar polygon on the viewing cube. The viewing directions that lie on this polygon are parallel to the plane of $f$. The polygon divides the viewing cube into two regions: the *southern* region of viewing directions that are below the plane of $f$ and therefore $f$ is invisible to viewpoints in this region , and the *northern* region, where $f$ is visible (provided that it is not occluded by other parts of the object). We refer to this polygon as a *boundary polygon*.

## 5.2 The Locus of Accidental Viewpoints for the EEE Event

The family of lines that go through three skew lines in $\mathbb{R}^3$ defines a ruled quadric surface. Therefore, for the perspective projection, the accidental viewpoints of a EEE event lie on a ruled quadric surface. As the edges are of finite extent, only parts of this surface actually contain points that are accidental with respect to this event.

For the orthographic projection, we are interested in the family of direction vectors of the corresponding family of critical lines. We can think of this family of vectors as defining a ruled surface of lines that go through the origin and have the same direction vectors. It turns out that this surface is either an elliptic cone or two intersecting planes. The locus of accidental viewpoints of the orthographic projection are the conic curves that result from intersecting the surface with the planes of the faces of the viewing cube. Similar to the perspective case, as the edges are of finite extent, only parts of these conic curves actually contain viewpoints that are accidental with respect to the EEE event. The details of how to compute the quadric surface and the conic curves are described in the appendix.

We use the term *EEE-boundary* to refer to the curves that are the locus of accidental viewpoints of a EEE event. The terms *boundary* or *boundary segment* will be used as general term that refers any boundary of the partition.

# 6 Boundary Pruning Using Local Information

In this section we describe simple tests for deciding whether a potential boundary is not part of the partition because the corresponding event is occluded by faces that are adjacent to the features that participate in the event. We use these tests to prune the set of boundaries and reduce the number of boundaries that have to be considered by the more time consuming global occlusion tests.

## 6.1 The Shadow Region of an Edge

For an oriented plane $p$ we define the negative and positive halfspaces of $p$ to be the halfspaces of $\mathbb{R}^3$ that are on the negative and positive sides of $p$, respectively. A viewpoint of the orthographic projection is said to be in the positive (negative) halfspace of $p$ when the dot product of the viewing direction with the normal to $p$ is positive (negative).

Assume that the plane normals of the faces of the object point to the outside of the object. Let the planes of the faces that share a given edge be $p_1$ and $p_2$. A convex edge can be visible only from points in $\mathbb{R}^3$ that are not in the intersection of the negative halfspaces of $p_1$ and $p_2$. A concave edge can be visible only from points that are in the intersection of the positive halfspaces of $p_1$ and $p_2$. We define the *shadow volume* of the edge to be the halfspace from which edge can not be visible. On the viewing cube we define the *shadow region* of edge in a similar manner. See figure 5.

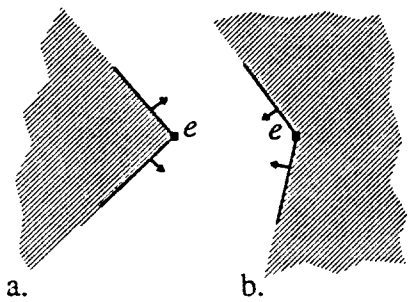


Figure 5: The *shadow volume* of an edge. The edge is perpendicular to the plane of the figure. The shadow volume is indicated by the shaded region and the plane normals of the faces are indicated by the arrows. a. a convex edge b. a concave edge.

## 6.2 Pruning EV-boundaries

Before we compute the boundary for an EV event, we check if $v$ is in the shadow volume of $e$. When this is the case, the event is never visible and we do not need to compute the boundary. When $v$ is not in the shadow volume of $e$, the event is potentially visible from the front polygonal curve

(of viewpoints from which $e$ is in front of $v$) only if the polygonal curve is not in the shadow region of $e$. A front polygonal curve for which this test fails, does not participate in the partition.

## 6.3 Pruning EEE-boundaries

The projections of two edges $e_1$ and $e_2$ will never intersect if either edge is fully contained in the shadow volume of the other. We create a table in which for each pair of edges we record whether the projections that pair may potentially intersect. We compute EEE-boundaries only for triplets of edges, $e_i$,$e_j$ and $e_k$, for which there is a potential intersection for each of the pairs $(e_i, e_j)$, $(e_j, e_k)$ and $(e_k, e_i)$.

## 6.4 The Effect of the Pruning on Convex Parts

In a convex part of the object, the shadow volume of an edge $e$ contains all edges and vertices that are not part of the two faces that meet at $e$. Thus, these tests prune away any event between features of convex parts of the object that are not part of the same face. Furthermore, for a convex object the EEE test takes $O(n^2)$ time, as the pairwise test is negative for all pairs of edges. Thus, in $O(n^2)$ time we prune away all boundaries except for the $O(n)$ boundary polygons of the faces of the object.

# 7 Computing The Active Boundaries of The Partition

After the initial pruning we are left with a set of potential boundaries. These boundaries are still only potential as we have not yet considered global occlusions, where some of the features that participate in the event are occluded by a face that is not adjacent to any of the features of the event. To compute the segments of the boundary that are part of the partition (*active* boundaries), we have to remove all the segments of the boundary from which the corresponding event is invisible.

When an event is occluded by a face $f$, a critical line of sight intersects $f$, such that the latter intersection point is closer to the viewpoint than at least one of the intersection points with the participating features. Suppose

that the viewpoint is moving along a boundary and the corresponding event is visible. At the point where the event becomes invisible the critical line of sight starts intersecting an occluding face. Therefore, when we consider the occlusion of an event by a single face, the viewpoints where the visibility of the event changes (event occlusion endpoints – *EOE points*), are viewpoints for which an edge of the face intersects the a critical line of sight.

This suggests the following algorithm for computing the active parts of a potential boundary whose endpoints are $b_1$ and $b_2$:

- Compute the EOE points of the boundary with respect to every face $f$ that is not adjacent to the features that participate in the event. The points are computed by finding the intersection points of each edge $e_f$ of $f$ with the ruled surface of the event (the surface on which the critical lines lie). For each intersection point $p$, check if the corresponding viewing direction is on the potential boundary (that is there is a critical line of sight that goes through $p$). If the answer is negative then $e_f$ does not contribute EOE points that are due to $p$[1]. Else, let $\vec{d}$ be a direction on the critical line of sight $l$ that intersects $e_f$ at $p$. At least one of $\vec{d}$ and $-\vec{d}$ is an EOE point (each lies on one of two diametrically opposite boundary segment of to the given event).

  If, along $l$, $p$ is between two of the features of the event, then both $\vec{d}$ and $-\vec{d}$ are EOE points. Else, when viewed from one of the diametrically opposite boundary segments, $e_f$ is in front of all the features of the event and contributes an EOE point due to $p$ on that boundary segment. From the other boundary segment, $f$ does not occlude the event at $e_f$, and therefore $p$ does not contribute an EOE point on that boundary segment.

- Classify each point as an *entry* or an *exit* point depending on whether the event becomes occluded or unoccluded by $f$, when the boundary is traversed in the direction from $b_1$ to $b_2$. See Figure 6.

- Sort the EOE points into a list that is ordered by their position along the boundary curve going from $b_1$ to $b_2$.

[1] $e_f$ may contribute another EOE point due to another intersection with the surface.

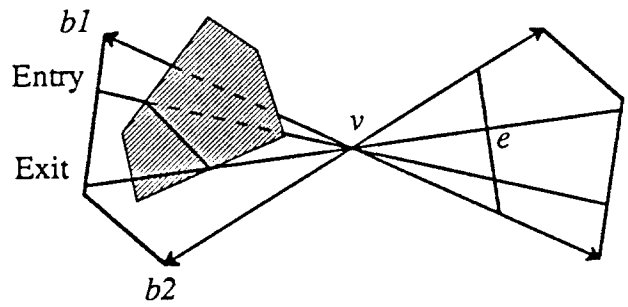


Figure 6: *Entry* and *exit* points of a face $f$ with respect to an EV-boundary

- For each face $f$, determine whether the event is occluded by $f$ at $b_1$. The event is occluded by $f$ at $b_1$ if the first EOE point of $f$ was determined to be an exit point. Compute the number of faces $F_o$ that occlude the event at $b_1$.

- The sorted list of EOE points and the endpoints of the boundary, defines a subdivision of the boundary into a set of segments. Scan the list of EOE points in the sorted order, incrementing $F_o$ by 1 at an *entry* point and decrementing it by 1 at an *exit* point. Using this procedure, $F_o$ keeps track of the total number of faces that occlude the event, from viewpoints that are on the boundary segment follow the point that has just been scanned. The *active* boundary segments are those for which $F_o$ is 0.

In the next two subsections, we describe how to compute and classify the EOE points.

## 7.1 Computing The EOE Points of an EV-boundary

To find the EOE points of an EV-boundary (for the event where the edge $e$ and the vertex $v$ interact) with respect to a face $f$, we compute the intersection points of the edges of $f$ with the plane of the EV event. Let $e_f$ be an edge of $f$ that intersects the plane at $p$. $\vec{d} = \overrightarrow{v-p}$ is a direction on the line $l$ that intersects both $v$ and $e_f$ and lies in the plane of the EV event. If $\vec{d}$ is neither on the front or the back polygonal curve of the event, then

$l$ does not intersect $e$, and therefore $e_f$ does not contribute any EOE point. Else, if $p$ is between $v$ and $e$ along $l$, then both and $\vec{d}$ and $-\vec{d}$ are EOE points. Else, if $\vec{d}$ is on the front polygonal curve, then $v$ and $e$ are in front of $f$ when viewed from the front polygonal curve, and therefore only $-\vec{d}$ is an EOE point (on the back polygonal curve). Else, only $\vec{d}$ is an EOE point. See figure 7.

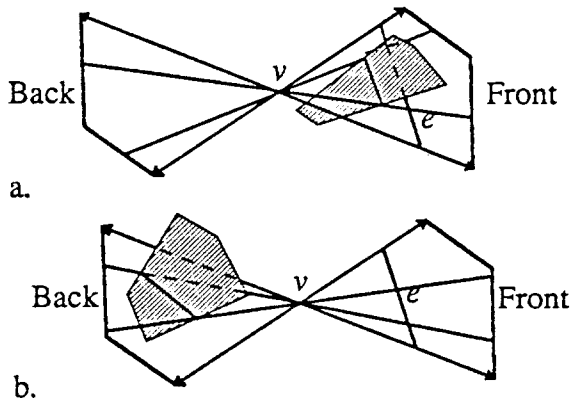


Figure 7: Occlusion of an EV event by a single face. a. The event is occluded from parts of both polygonal curves. b. The event is occluded only from part of the back polygonal curve.

To determine whether the EOE point of an edge $e_f$ is an *entry* or an *exit* point, we compute a vector $\vec{\xi}$ that points from $e_f$ into $f$, along the intersection line of $f$ with the plane of the event. For an EOE point that is the front (back) polygonal curve, if the sign of the dot product $< \vec{\xi}, \overrightarrow{b_2 - b_1} >$ is positive, then the point is an *entry* (*exit*) point, else it is an *exit* (*entry*) point.

## 7.2 Computing The EOE Points of a EEE-boundary

To find the EOE of a EEE-boundary (where $e_1$,$e_2$ and $e_3$ interact) that are generated by an edge $e_f$ of a face $f$, we compute the intersection points of the infinite line on which $e_f$ lies with the quadric surface $S_{eee}$ of the EEE event. For each intersection point $p$ that is contained in $e_f$, we compute

the a direction $d_p$ along the line $l_p$ that goes through $p$ (and hence $e_f$) and the infinite lines on which $e_1$, $e_2$ and $e_3$ lie (the details of this computation are given in the appendix). If $d_p$ is inside a segment of the EEE-boundary from which a line of sight actually goes through $e_1$, $e_2$ and $e_3$ then at least one of $d_p$ and $-d_p$ is an EOE point. If $p$ is between the intersection points of $l_p$ with $e_1$, $e_2$ and $e_3$ then both $d_p$ and $-d_p$ are EOE points. Else, if from $d_p$, $p$ is in front of the intersection points of $l_p$ with $e_1$, $e_2$ and $e_3$ then only $d_p$ is an EOE point. Else only $-d_p$ is an EOE point.

The classification of the points as entry or exit points is determined as follows:

- Let $\vec{\xi}$ be a vector that points into $f$ along the intersection line of $f$ with the tangent plane to $S_{eee}$ at $p$. This is the direction in $l_p$ moves locally when the event becomes occluded by the inside of $f$.
- Using the map that is described in the appendix, we compute the direction $\overrightarrow{d_{vc}}$ in which the viewpoint moves on the face of the cube when $l_p$ moves in the direction $\vec{\xi}$ .
- Let $\vec{t}$ be a vector that is tangent to the EEE-boundary at $d_p$ and is in the direction that goes from $b_1$ to $b_2$. If $\overrightarrow{d_{vc}}$ is in the same direction as $\vec{t}$ then $d_p$ is an *entry* point. Else it is an *exit* point. When both $d_p$ and $-d_p$ are found to be EOE points, then when $d_p$ is an *entry* point then $-d_p$ is an *exit* point and vice versa.

## 7.3 Computing the Active Segments of a Boundary Polygon

To compute the active segments of a boundary polygon of a face $f$, we compute the active segments of each of the EV-boundaries that are defined by the edges and vertices of $f$. We then merge overlapping active segments to get the active segments of the boundary polygon. These active segments are marked as being part of the boundary polygon of $f$.

## 7.4 The Complexity of Finding the Active Segments

For each potential boundary we spent constant time in computing the EOE points for each edge of the object. We also sort the EOE points along each

boundary. As the total number of boundaries is bounded by $O(n^3)$, and there are $O(n)$ edges, the number of EOE points is bounded by $O(n^4)$. As we sort these points along each boundary, the time complexity of finding the *active* boundary segments is $O(n^4 \log n)$. We may end up with $O(n^4)$ active segments. However, these segments will not intersect in more than $O(n^6)$ points, as the total number of intersections of the original potential boundaries is bounded by $O(n^6)$.

Note, that EOE points are the viewpoints from which lines of sight intersect four edges of the object and therefore we get $O(n^4)$ as a bound on the number of EOE points.

# 8 Computing the Partition of The Viewing Cube

After the global occlusion tests, we have the set of active boundary segments of the partition. To construct the partition of the faces of viewing cube, we need to find intersection points between the segments and create a data structure that represents the vertices, edges and regions of the partition. We construct the partition using a plane sweep algorithm [PS85] on each of the faces of the cube separately. In the discussion below we assume that we are sweeping a face of the cube that is parallel to the $xy$ plane and that the sweep line is parallel to the $y$ axis.

The plane sweep algorithm uses an event queue where the events are points where segments of the partition start or stop intersecting the sweep line, or intersection points between segments that are currently intersecting the sweep line. As a EEE-boundary segment is part of a conic curve, the points where the sweep line starts or stops intersecting the segment may either be the endpoints of the segment or points of maximal and minimal $x$ extent of the segment. Therefore, for each EEE-boundary segment we find the points that have the minimum and maximum x coordinates. If these points are not the endpoints of the segment, we divide the segment at these points into subsegments that are monotone in the x direction. Each of the subsegment will intersect any vertical sweep line at only one point. For the plane sweep we consider each subsegment separately. See Figure 8.

Once we have preprocessed the EEE-boundaries, we can execute a reg-

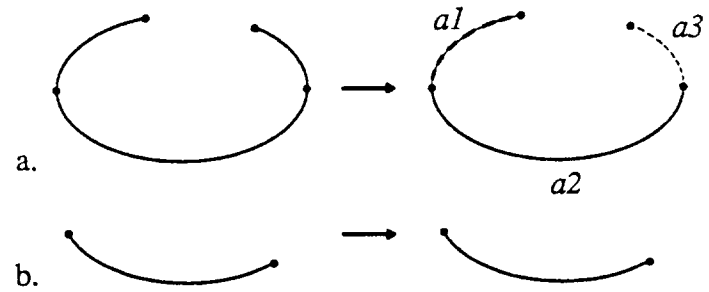


Figure 8: Preprocessing conic segments for the plane sweep. Segment a is divided into subsegments at the extremal points in the horizontal direction. Segment b is not divided as its endpoints are the extremal points in the horizontal direction.

ular plane sweep to construct the partition of the face of the cube. The algorithm runs in $O(m \log m)$ time, where $m$ is the number of vertices in the partition.

## 8.1 Viewpoint Location in Log Time and in Linear Space

For some applications we may need to be able to answer the query "Given a viewpoint $vp$, find the region of the partition that contains $vp$". This query can be answered in $O(\log m)$ time, if we save all the sweep lines at the events of the plane sweep algorithm (these are the sweep lines that go through the vertices of the partition). Using the persistent search tree data structure that was invented by Sarnak and Tarjan [ST86], we store all the sweep lines using only $O(m)$ space. The persistent search tree is build and updated during the plane sweep without increasing the time complexity of the algorithm.

# 9 Computing and Storing the Views

Once we have computed the partition, we compute the view of the object in one region of the partition, using a any suitable hidden line removal algorithm. We then traverse the partition in order of adjacent regions,

and compute the view in each region by updating the view of the adjacent region, according to the visual event that occurs at the boundary between the regions. The exact details of how to update the view according to the visual event that occurs at the boundary between regions are described by Gigus and Malik in [GM88a].

Using a naive approach to storing the views of the partition, we would explicitly represent the complete graph of the view in each region of the partition. As a single view may be of size $O(n^2)$ and the size of the partition may be as large as $O(n^6)$, the naive approach will use $O(n^8)$ space. As writing out a single view takes $O(n^2)$ time, the total time to generate all the views is also $O(n^8)$. Even if the average size of a view is $s$ and the size of the partition is $m$, the naive approach will use $O(sm)$ time and space.

When we generate a new view, we copy most of the old view with only a few modifications. If, instead of representing each view explicitly, we could use a persistent data structure that keeps track of the changes between adjacent views, we should be able to generate all the views in time and space that are close to being linear in the number of updates, which is bounded by $O(m)$ [GM88a]. Below, we describe a data structure for representing the views implicitly by keeping track of the changes between one view to the next. This data structure is constructed in $O(m)$ time and space, and requires $O(\log m + S)$ for retrieving a single view, where $S$ is the size of the view.

## 9.1 Representing the Views in Linear Space

The traversal of the partition can be abstracted as follows. The adjacency relation between the regions of the partition is represented by the dual of the graph of the partition, where we have a vertex for each region and an edge between any two vertices for which the corresponding regions are adjacent in the partition. This dual graph is the *aspect graph* of the object, as defined by Koenderink and van Doorn. With each edge of the aspect graph we keep a the list of visual events the occurs at the corresponding boundary. To generate the views, we compute the view at one vertex of the aspect graph, and then traverse the graph using a depth-first search algorithm. Whenever we visit a new vertex, we compute the view at that vertex according to the events that occur at the boundary that corresponds to the edge leading to the new vertex. The edges that are traversed in this

process form a spanning tree of the aspect graph.

In the naive approach, we store a view at each vertex of the graph. Whenever we reach a leaf $l$ of the spanning tree (a region for which all the neighbors have already been visited), we go back to a vertex $v$ that is the lowest ancestor of $l$ that has neighbors which have not been visited yet, retrieve the view at $v$, and continue the traversal process. Instead of retrieving the view at $v$ from storage, we could generate this view by walking back up the tree along the vertices on the path from $l$ to $v$, and maintaining the current view along the path by updating it according to the events that occur at boundaries that correspond to each edge on the path up the tree. As we never traverse any edge more than twice (once on the way down and once on the way up), the total number of updates that we perform in this process is bounded by twice the number of updates of the traversal method that uses view retrieval. Figure 9 illustrates the traversal of the partition in both methods.

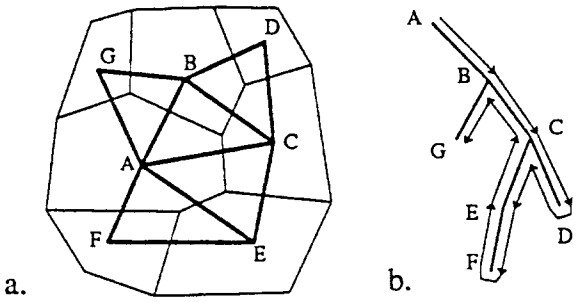


Figure 9: The traversal of the partition as a graph traversal process. a. A partition and the aspect graph (in bold lines). b. The spanning tree and the modified method of traversal. In the regular method the vertices are visited in the sequence ABCDEFG. In the modified method the sequence is ABCDCEFECBG.

If we consider a visit to a vertex of the aspect graph as a step in time, the modified version of the traversal generates a time sequence of ISGs, $SG = (G_0, G_1, \ldots, G_T)$, where $G_t$ (the ISG at time $t$) is generated from $G_{t-1}$ by deleting some of the its edges (and maybe some of its vertices) and adding some new edges and vertices. Some of the ISGs in this sequence are identical to each other. Assume that at time 0, each vertex in $G_0$ is given a

unique name, and whenever we create a new vertex, we give it a new unique name that has not been used previously. Under this naming scheme, we can associate with each vertex $v$, of the ISG, the interval $I(v) = [ti_v, td_v]$ which indicates when $v$ existed in the graph (it was inserted at time $ti_v$ and deleted at time $td_v$). Similarly with each edge $(v_i, v_k)$ we can also associate its existence interval. Instead of representing the sequence $SG$ explicitly, we store the sets $Ve$ and $Ee$ of the existence intervals of the vertices and edges in the sequence. Also, at each region of the partition we keep a record of the first time $t$ that we visited this region. Given the time label $t$ of a region we can retrieve the corresponding view $G_t$ by answering the query: "Find all the vertices and edges that existed at time $t$". This query is equivalent to the query: "Find all the existence intervals in $Ve$ and in $Ee$ that contain $t$", which is an *inverse range* query. For $m$ intervals that are sorted by their left endpoint (as our time intervals are), in $O(m)$ time we can construct a priority search tree of size $O(m)$, for answering inverse range queries in $O(\log m + K)$, where $K$ is the number of intervals that contain $t$ (For more details of this data structure see [MelAR, pages 199–201,211]).

As each update can only contribute one interval, the size of $Ve$ and the size of $Ee$ is bounded by $O(V+m)$, where $V$ is the size of the ISG at time 0. As, in the worst case, the ISG is of size $O(n^2)$ and there are $O(n^6)$ updates, the size of $Ve$ and $Ee$ is bounded by $O(n^6)$. In the worst case, the retrieval time is bounded by $O(n^2 + \log n) = O(n^2)$, which is optimal in the size of the view.

# 10 Conclusions

We have presented an algorithm that computes the viewing data of an object in near optimal time and in optimal space. The algorithm is based on understanding how the three dimensional geometry of the object is reflected in the visual interaction between the features of the object. We used two key ideas to reduce the complexity of the computation. The first idea is to prune boundaries that correspond to invisible events on the basis of local information at the edges and vertices of the object. The effectiveness of the initial pruning is demonstrated by the fact that, for convex objects, it eliminates all the potential boundaries that correspond to occluded events, and results in optimal computation time. The second idea is based on the

observation that the most time consuming part of the algorithm is computing the intersections between the boundaries that correspond to individual events. By computing the active boundaries of the partition, before we actually construct the partition, we are able to reduce the computation time to $O(m \log m + n^4 \log n)$ where $m$ is the actual size of the partition. All the computations of the algorithm are done in $\mathbb{R}^2$ or $\mathbb{R}^3$. These spaces are well behaved and we have a complete understanding of the geometry of the lines and surfaces that are involved in the computation.

The naive way of representing the set of views would take time and space that is the product of the size of a single view and the size of the partition. However, using a novel method of representing the views, we are able to generate all the views in time and space that are linear in the number of updates, and therefore optimal.

The current algorithm handles the class of polyhedral objects. As all the visibility computations of the algorithm are performed directly in $\mathbb{R}^3$, it might be possible to extend the algorithm to other classes of objects. We are currently investigating methods of extending the algorithm to the class of piecewise quadric objects.

# A Appendix - Computing the Boundaries for the EEE Event

## A.1 The Surface of the EEE Event

Let the three edges that participate in the event be $e_1$, $e_2$ and $e_3$, where $e_i$=($a_i$,$b_i$). Let $\overrightarrow{d_i} = \overrightarrow{b_i - a_i}$ be the direction vector of $e_i$ and let $l_i$ be the infinite line on which $e_i$ lies. Let $p$ be a viewpoint of a perspective projection. The normal to the plane $P_i$ through $p$ and $e_i$ is $\overrightarrow{(p - a_i)} \times \overrightarrow{d_i}$. The map

$$\overrightarrow{\mathcal{P}_{i,j}(p)} = (\overrightarrow{(p - a_i)} \times \overrightarrow{d_i}) \times (\overrightarrow{(p - a_j)} \times \overrightarrow{d_j}) \quad (1)$$

computes a direction along the intersection line of $P_i$ and $P_j$ which is the line that goes through $p$ and intersects both $l_i$ and $l_j$. In particular, if from $p$ there is a line of sight $l_c$ that actually intersects both edges, then $\overrightarrow{\mathcal{P}_{i,j}(p)}$ computes a direction along this line. If $l_c$ intersects all three edges then it is the common intersection line of all three planes (see Figure 10), and therefore:

$$< \overrightarrow{\mathcal{P}_{1,2}(p)}, \overrightarrow{(p - a_3)} \times \overrightarrow{d}_3 >= 0. \quad (2)$$

This is a quadric equation in the coordinates of $p$, which defines a quadric ruled surface on which the accidental viewpoints of this EEE event must lie.

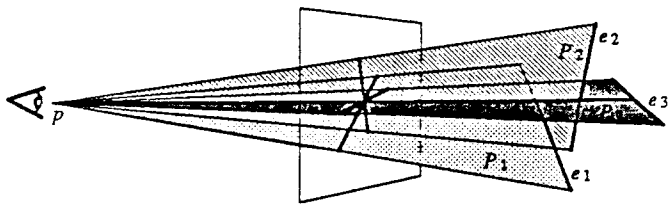


Figure 10: The line of sight that goes through the three edges as the common intersection of three planes.

Not all points on the surface are accidental viewpoints of the event where $e_1$, $e_2$ and $e_3$ interact. As the edges are of finite extent, some parts

of the surface do not contain points from which a straight line actually intersects all three edges and therefore the viewpoints that are contained in these parts of the surface are not accidental viewpoints of this event. In addition, the viewpoints that are contained in parts of the surface that are between the edges are not accidental viewpoints for this event, as not all three edges can be viewed simultaneously from these viewpoints.

## A.2 The Boundary Curve for the Orthographic Projection

For an orthographic projection with viewing direction $[x, y, z]$, the viewpoint is of the form $\alpha[x, y, z]$, with $\alpha$ going to infinity. Substituting the viewpoint into Equation 2 we get:

$$\lim_{\alpha \to \infty} (\alpha^2 (c_1 x^2 + c_2 y^2 + c_3 z^2 + c_4 xy + c_5 xz + c_6 yz) + \alpha(c_7 x + c_8 y + c_9 z) + c_{10}) = 0, \quad (3)$$

and therefore:

$$c_1 x^2 + c_2 y^2 + c_3 z^2 + c_4 xy + c_5 xz + c_6 yz = 0, \quad (4)$$

is the condition on the viewing direction of the EEE event in the orthographic projection. To get the curve of viewing directions on the viewing cube we intersect the surface described by Equation 4 with the planes of the faces of the cube: $x = \pm 1$,$y = \pm 1$ and $z = \pm 1$.

Similar to the perspective case not all points on the intersection curve of the surface with the cube are accidental viewpoints for the event. Only the segments of the curve from which a line of sight actually goes through $e_1$, $e_2$ and $e_3$ contain accidental viewpoints for this event, and are potential boundaries of the partition.

## A.3 Representing the EEE-boundary Segments

For the purposes of the algorithm, we need to assign a direction to the EEE-curve, represent continuous segments of curve, locate a segment of the curve that contains given point is on, and sort points along a given segment. We assign a direction to the curve by computing the normal to curve and

chosing the clockwise orthogonal vector as the tangent vector to the curve. This vector is unique at any point of a conic curve and it is continuous on a continuous segment of the curve. We partition the curve at points of maxima in the horizontal direction resulting in continuous segments for which each point has a unique $x$ coordinate. A segment is represent by an ordered pair of endpoints such that a traversal of the segment from the first endpoint to the second one follows the direction of the curve. When sorting points along the curve we also use the order that is implied by the direction of the curve. Given a point on the curve, we can locate the segment that contains it by comparing the tangent vector at the point to the tangent vectors at the endpoints of each segment of the curve.

## A.4 Computing the Potential Segments of the EEE-boundary

Let $l_c$ be the line the intersects $l_1$, $l_2$ and $l_3$. A motion of the viewpoint along the EEE-boundary correspond to a motion of $l_c$ in $I\!R^3$ which maintains its contact with $l_1$, $l_2$ and $l_3$, while its direction vector is changing to agree with the viewing direction (the viewpoint). At the endpoints of the potential segments of a EEE boundary, $l_c$ starts or stops intersecting one of the edges. In other words, the endpoints of the potential segments are viewpoints from which $l_c$ goes through one of the endpoint vertices of the edges. The endpoints of the segment $s_i$ from which $l_c$ intersects $e_i$ are $\mathcal{P}_{j,k}(a_i)$ and $\mathcal{P}_{j,k}(b_i)$. As at an accidental viewpoint $l_c$ intersect $e_1$, $e_2$ and $e_3$, the potential segments are the parts of the boundary where $s_1$, $s_2$ and $s_3$ overlap.

To fully determine $s_i$ we need to determine whether when traversing the boundary in the direction of the tangent vector, $l_c$ enters or exits $e_i$ when we reach $\mathcal{P}_{j,k}(a_i)$ (the second endpoint of $s_i$ has the opposite classification). Making this determination is equivalent to answering the following question: "Suppose that $l_c$ is at $a_i$ and it is moving into $e_i$, is the direction $\overrightarrow{d_{dl}}$ in which the direction vector of $l_c$ moves along the face of the cube the same as, or opposite to the tangent vector to the boundary at $\mathcal{P}_{j,k}(a_i)$ ?". In the next section we describe the map that computes the projection of local motions of $l_c$ onto the cube, $\overrightarrow{d_{dl}}$ is computed by applying this map to $\overrightarrow{b_i - a_i}$.

Once we have fully determined $s_1$, $s_2$ and $s_3$, we sort their endpoints along the boundary curve and determine the potential segments by computing the segments of the boundary where $s_1$, $s_2$ and $s_3$ overlap.

## A.5 Projecting Local Motions of the Line of Sight onto the Viewing Cube

In computing the potential and actual active segments of a EEE-boundary we need to be able to perform the following computation:

- Given:

  1. a point $p$ on the line $l_c$ that intersects $l_1$, $l_2$ and $l_3$

  and

  2. a direction $\vec{v_p}$ in which $l_c$ moves at $p$,

- find the direction $\vec{d_{cl}}$ in which the direction vector of $l_c$ moves on the face of the cube.

Assume that the point $p$ is not on either $l_i$ or $l_j$. Then the map $\overrightarrow{\mathcal{P}_{i,j}(p)}$ computes a direction vector $\vec{d_l}$ along $l_c$ when it goes through $p$. In general, $\vec{d_l}$ is not normalized. Assume that the point $c_l$ that is the representation of $\vec{d_l}$ on the viewing cube is on the face that is contained in the plane $z = 1$. Then $c_l$ is the perspective projection of $d_l$ onto the plane $z = 1$. That is, $c_l = \mathcal{Q}(\vec{d_l}) = \mathcal{Q}(\overrightarrow{\mathcal{P}_{i,j}(p)})$, where $Q(x, y, z) = (x/z, y/z, 1)$. If $l_c$ is moving in the direction $\vec{\xi}$ at $p$, then, as $l_c$ is always on the EEE quadric surface of the EEE event, $\vec{\xi}$ is on the tangent plane to the surface at $p$. The direction $\vec{d_{cl}}$ in which $c_l$ moves is on the tangent plane to the face of the cube at $c_l$ (which happens to be the plane of the face itself). To project a vector in the tangent plane at $p$ to the tangent plane at $c_l$ we need to apply the differential to the map that carries $p$ into $c_l$, and therefore:

$$\vec{d_{cl}} = d(\mathcal{Q}(\overrightarrow{\mathcal{P}_{i,j}}))|_p \ \vec{v_p} = d\mathcal{Q}|_{\mathcal{P}_{i,j}(p)} d\mathcal{P}_{i,j}|_p \ \vec{v_p} \tag{5}$$

where $dF$ denotes the Jacobian matrix of the map $F$.

# References


[Arn79] V. I. Arnol'd. Indices of singular points of 1-forms on a manifold with boundary, convolutions of invariants of reflection groups, and singular projections of smooth surfaces. *Russian Mathematical Surveys*, 34(2):1–42, 1979.

[Arn83] V. I. Arnol'd. Singularities of systems of rays. *Russian Mathematical Surveys*, 38(2):87–176, 1983.

[CF82] I. Chakravarty and H. Freeman. Characteristic views as a basis for three-dimensional object recognition. In *Proceedings of The Society for Photo-Optical Instrumentation Engineers Conference on Robot Vision*, pages 37–45, Vol. 336, SPIE, Bellingham, Wash., 1982.

[CW85] J. Callahan and R. Weiss. A model for describing surface shape. In *Proceeding of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 240–245, IEEE, N.Y., 1985.

[GM88a] Z. Gigus and J. Malik. Computing the viewing data for polyhedral objects. In *Proc. IEEE Comp. Soc. Conf. on Computer Vision and Pattern Recognition*, IEEE, N.Y, June 1988. To appear.

[GM88b] Z. Gigus and J. Malik. Computing the viewing data for polyhedral objects. In *Proc. 1988 IEEE Int. Conf. on Robotics and Automation*, IEEE, N.Y, April 1988.

[HK85] M. Hebert and T. Kanade. The 3d-profile method for object recognition. In *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 458–463, IEEE, N.Y., June 1985.

[Ike87] K. Ikeuchi. Precompiling a geometrical model into an interpretation tree for object recognition in bin-picking tasks. In *Proceedings of DARPA Image Understanding Workshop*, pages 321–339, February 1987.

[Ker81] Y. L. Kergosien. La famille des projections orthogonales d'une surface et ses singularitiés. *C. R. Acad Sc. Paris*, 292:929–932, 1981.

[KvD76] J. J. Koenderink and A. J. van Doorn. The singularities of visual mapping. *Biol. Cybern.*, 24:51–59, 1976.

[KvD79] J. J. Koenderink and A. J. van Doorn. The internal representation of solid shape with respect to vision. *Biol. Cybern.*, 32:211–216, 1979.

[MelAR] K. Melhorn. *Data Structures and Algorithms.* Volume 3: Multidimensional Searching and Computational Geometry, Springer-Verlag, YEAR.

[PD86] W. H. Plantinga and C. R. Dyer. An algorithm for constructing the aspect graph. In *Proceedings of the 27th. Symp. on the Foundation of Computer Science*, pages 123–131, IEEE, N.Y., 1986.

[PD87] W. H. Plantinga and C. R. Dyer. *Visibility, Occlusion and The Aspect Graph.* Tech. Report 736, University of Wisconsin, Madison, December 1987.

[Pla88] W. H. Plantinga. Personal communication, June 1988.

[PS85] F. P. Preparata and M. I. Shamos. *Computational Geometry, An Introduction.* Springer-Verlag, 1985.

[Rie87] J. H. Rieger. On the classification of views of piecewise smooth objects. *Image and Vision Computing*, 5(2):91–97, May 1987.

[SB87] J. Stewman and K. Bowyer. Aspect graphs for convex planar-faces objects. In *Proc. IEEE Comput. Soc. Workshop on Computer Vision*, pages 123–130, IEEE, N.Y, December 1987.

[ST86] N. Sarnak and R. E. Tarjan. Planar point location using persistent search trees. *Communications of the ACM*, 29(7):669–679, July 1986.

[WBG86] M. Werman, S. Baugher, and J. A. Gualtieri. *The Visual Potential: One Convex Polygon.* Tech. Report CAR-TR-212, Center for Automation Research, University of Maryland, College Park, MD, August 1986.