HIDDEN-SURFACE REMOVAL IN POLYHEDRAL-CROSS-SECTIONS

by

Peter Egyed

School of Computer Science
McGill University
Montreal, Quebec, Canada
July 1987

A thesis submitted to the
Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of
Master of Science

Abstract

One of the fundamental problems in computer graphics is determining which portions of a scene are visible from a given viewing position. The problem is known as the hidden-line or hidden-surface problem depending on whether edges or faces are displayed. One approach to the hidden-surface problem involves assigning priorities to the faces of a scene. A realistic image is then rendered by displaying the faces with the resulting priority ordering. Although priority orderings have been researched, very little effort has gone into the development of a mathematical theory. In this paper we develop a new formalism for describing priority orderings and propose efficient algorithms for dealing with a variety of inputs. As well, we present insertion and deletion algorithms for maintaining a priority ordering in a dynamic environment.

Résumé

Un des problèmes fondamentaux dans le domaine des graphiques par ordinateur est de déterminer les portions d'une scène qui sont visibles à partir d'un point de vue donné. Ce problème est connu sous le nom de problème des lignes cachées ou de surface cachées, tout dépendant si l'on présente à l'écran des arêtes ou des surfaces. Une des approches au problème de surfaces cachées consiste à attribuer des priorités aux faces d'une scène. Une image réaliste est ensuite obtenue en affichant les faces par ordre de priorité. Bien que cette méthode ait été étudiée, très peu d'efforts ont été fournis pour développer une théorie mathématique. Dans cette thèse nous développons un nouveau formalisme pour décrire les classements par priorité et proposons des algorithmes performants pour diverses classes de scènes. Egalement, nous présentons des algorithmes d'insertion et d'élimination pour maintenir un classement par priorité dans un environnement dynamique.

Table of Contents

Chapter 1

Introduction

When displaying objects, one of the most challenging problems encountered involves removing the portions of the objects obscured by others nearer to the viewing position. Depending on whether edges or faces are displayed, the problem is commonly referred to as the *hidden-line* or *hidden-surface* problem.

Due to the variety of applications, many different algorithms employing various approaches have been proposed. In general, differences between the algorithms arise from different variables such as, the complexity of the scene model, and, the required realism of the image. Despite their great diversity, the algorithms all share one common characteristic: each performs some kind of geometric sorting. The use of geometric sorting stems from the need to distinguish those portions of the scene that are visible from those that are hidden. Those parts that are hidden lie further from the viewing position than the parts that obscure them. The difficulty then, of the hidden-line and hidden-surface problems, arises from the complex nature of orderings of objects in space.

Algorithms for hidden-line and hidden-surface removal can be broadly classified into two groups. *Image-space* algorithms perform depth comparisons at each pixel of the display device. Their resulting time complexities are thus dependent on the resolution of the display device. *Object-space* algorithms on the other hand, perform geometric comparisons directly on the objects in some abstract space, and so their time complexities are strictly object dependent.

Much of the motivation for the development of hidden-line and hidden-surface algorithms stems from their ever increasing importance in computer graphics. As a result, a considerable portion of the total research effort in the field has been guided by the practitioner's viewpoint. For an overview of the algorithms designed from this point of view see [1-4]. Only recently, spurred by developments in the new and flourishing field of computational geometry, has the theoretical nature of the problems begun to be investigated.

The notion of visibility amongst geometric objects has been intensively investigated in two dimensions. Many different variations on the hidden-line or *visibility-polygon* problem, have been considered. El Gindy and Avis [5], as well as Lee [6], each describe a linear and thus optimal algorithm for the case of a single polygonal object. An efficient algorithm for determining visibility amongst a collection of disjoint polygons is proposed by Asano [7]. By restricting the input to a single star-shaped polygon, Rappaport and Toussaint [8] are able to exhibit a very simple linear algorithm for the problem. Introduced by Avis and Toussaint [9], *edge-visibility* problems consider polygonal visibility from an edge. The *strong hidden-line* problem, which involves determining the region of a polygon visible from a specified edge, is one such problem. Different solutions to the problem can be found in papers by Lee and Lin [10], El Gindy [11], Chazelle and Guibas [12], and Toussaint [13]. Edelsbrunner et. al [14] consider various visibility problems associated with scenes composed of simple convex objects. One involves the maintenance of a view during the insertion and deletion of objects, and the other considers frame-to-frame coherence while walking around a scene.

A vast amount of research in computational geometry has been devoted to intersection problems. In order to display a three-dimensional image, the scene must be projected onto the viewing plane and any conflicts between components must be resolved. It is only natural then, that the techniques discovered during the investigation of intersection problems be applied to visibility problems in three dimensions.

Many different solutions, having various time and space requirements, have been proposed for the general hidden-line problem. In order to put the various results into perspective, let us first consider a few definitions. Let n denote the number of edges, in the scene and let k

and r respectively denote the number of intersections of edges and the number of times an edge is contained by a polygon, both in the viewing plane. Schmitt [15] has demonstrated a worst-case $\Omega(n^2)$ lower bound for the problem. Devai [16] has established $\Theta(n^2)$ time bound by presenting an optimal $O(n^2)$ time algorithm. The algorithm requires $O(n^2)$ space and relies on existing methods for computing line arrangements in the plane. Some output-sensitive algorithms that depend on plane-sweep techniques also exist. Schmitt [15] presents such an algorithm with a worst-case runtime of $O(r+(n+k)logn)$ and space requirements of $O(n+k)$. Also using the plane-sweep paradigm is an algorithm proposed by Ottmann et. al [17] that requires $O((n+k)log^2n)$ time and $O(nlogn)$ space. A modification to this algorithm, due to Nurmi [18], reduces the time requirements to $O((n+k)logn)$ but also increases the space requirements to $O((n+k)logn)$. Note that these algorithms all require more than $O(n^2)$ time in the worst case. By restricting the class of input considered, other authors have obtained improved results. Rappaport [19] for example, presents a linear algorithm for the case of a single monotone slab. For finitely-oriented sets of polygons, Guting and Ottmann [20] are able to obtain an algorithm which runs in $O(nlogn+k)$ time and requires $O(nlogn)$ space.

Some theoretical results have also been obtained in the area of hidden-surface removal. Schmitt [15] has demonstrated a worst-case $\Omega(n^2)$ lower bound for the problem. As well, Mckenna [21] has presented an optimal $O(n^2)$ time algorithm thus establishing an $\Theta(n^2)$ time bound. The algorithm requires $O(n^2)$ space and depends on existing techniques for computing line arrangements in the plane. One method that shows great promise is the *priority approach*. This technique involves assigning depth priority numbers to the faces of a scene. The desired obscuring effect is then achieved by displaying the faces using the resulting priority ordering. This type of procedure is commonly known as the *painter's* algorithm. Suppose for a given viewing position some face f_1 obscures another face f_2. This relationship between the pair must then be correctly reflected by their assigned priorities. Unfortunately, it is not always possible to compute priority orderings since cyclic constraints may exist. On the other hand, many scenes exhibit a remarkable property in that it is possible to compute priority orderings for them before a viewing position is specified. This of course leads to significant time savings during image generation. Although several papers [22-26] have considered various aspects of the problem, they fail to develop any significant theoretical insight into the problem. In contrast, Yao [27] investigates the underlying mathematical nature of priority orderings, and proposes efficient algorithms for a restricted class of input. For the class, Yao proves that for a given view point the priority ordering can be computed in $O(n \log n)$ time using $O(n)$ space As well, Yao demonstrates an $\Theta(n)$ bound for the required number of priority orderings.

The purpose of this thesis is to extend the work of Yao. In particular, we consider a new formalism for describing priority orderings and present efficient algorithms for dealing with a variety of inputs. As well, we propose algorithms for maintaining a priority ordering during a series of insertions and deletions. We now briefly describe the remainder of this thesis. In chapter two, the class of scenes to be considered is defined and some basic properties of the objects comprising the scene are deduced. A new formalism for describing priority orderings is introduced in chapter three. Also, an existing algorithm due to Yao, and a modification of the algorithm, are presented for a subclass of scenes that is predominantly two-dimensional. In chapter four the most general class of scenes is considered. These scenes do not in general admit priority orderings. To remedy this situation, different decompositions of the scene are proposed, and algorithms for solving the problem are presented. Although finding a minimum decomposition appears difficult, a heuristic is presented that uses at most twice the minimum number of cuts. In chapter five, algorithms for maintaining a priority ordering through a series of insertions and deletions are developed. Finally, possible future research is discussed in the last chapter.

Chapter 2

The Scene

The complexity of a two-dimensional scene is dependent on the class of objects chosen to represent the scene. In general, choosing a class of objects appropriate for a specific application involves a trade-off between scene complexity and processing efficiency. We introduce in this chapter, a three-dimensional scene of moderate complexity, whose two-dimensional properties afford an efficient solution to the hidden-surface problem.

In the sections of this chapter, we first introduce some basic definitions and notation, then define the scene, and conclude by proving some properties of the scene.

2.1. Basic Definitions

As is standard in computational geometry, points are termed *vertices*, and pairs of points defining line segments are termed *edges*. A *simple polygon P* is a simply connected subset of the plane whose boundary is a closed chain of edges linked by their endpoints, with no two nonadjacent edges intersecting. We represent such polygons by a clockwise sequence of vertices, $v_1, v_2, ..., v_n$, where each vertex v_i is described by its cartesian coordinates (x_i, y_i). The sequence is assumed to be in *standard form*, i.e., the vertices are distinct and no three consecutive vertices, indices taken modulo n, are collinear. A pair of consecutive vertices, say v_i, v_{i+1}, indices taken modulo n, termed the *tail* and *head* respectively, define the i^{th} edge and is represented by e_i. The sequence $e_1, e_2, ..., e_n$ of edges forms the *boundary* of a polygon P, is denoted by bnd(P), and partitions the plane into two open regions: one bounded, termed the *interior* of P and denoted by *int*(P), and the other unbounded, termed the *exterior* of P and denoted by *ext*(P).

2.2. Defining the Scene

A polyhedron is a solid bounded by simple polygons, termed *faces*, so that each edge is shared by a pair of adjacent faces and no two nonadjacent faces intersect. We define a *scene*, the class of input to be considered, as a collection $S = (PX_1, PX_2, \ldots, PX_m)$ of nonintersecting *polyhedral-cross-sections*. A polyhedral-cross-section is a polyhedron of restricted form that is enclosed by *base-faces*, simple polygons $P_{b_i} = (v_{b_i 1}, v_{b_i 2}, \ldots, v_{b_i n_{b_i}})$ and $P_{t_i} = (v_{t_i 1}, v_{t_i 2}, \ldots, v_{t_i n_{t_i}})$ that lie in parallel planes $z = z_{b_i}$ and $z = z_{t_i}$ respectively, and also by a collection $F_i = (f_{i1}, f_{i2}, \ldots, f_{in_i})$ of simple polygons, termed *lateral-faces*, that connect P_{b_i} and P_{t_i}. The base-faces P_{b_i} and P_{t_i} are named with the convention $z_{t_i} > z_{b_i}$, and termed the *top* and *bottom* base-face respectively. Note that a vertex v_i of a base-face is described by its planar cartesian coordinates (x_i, y_i) and the plane $z = z_i$ in which it lies. Given a three-dimensional object G, let its projection onto the x-y plane, termed the *x-y projection*, be denoted by G'. P_{b_i} and P_{t_i} are restricted so that either $P_{b_i}' \subseteq P_{t_i}'$ or $P_{t_i}' \subseteq P_{b_i}'$. Alternate symbols for the base-faces are derived from the containment relation: if $P_{b_i}' = P_{t_i}'$, then the *minor* base-face, denoted by P_{m_i}, is P_{t_i}, and the *superior* base-face, denoted by P_{s_i}, is P_{b_i}, otherwise P_{m_i} is the properly contained base-face and P_{s_i} is the other. For simplicity we shall denote $int(P_i) \cap int(P_j)$ by $\Gamma(P_i, P_j)$. The placement of the polyhedral-cross-sections is restricted so that given any pair PX_i, PX_j of S, if $\Gamma(P_{s_i}', P_{s_j}') \neq \emptyset$ and $z_{b_i} < z_{b_j}$ then $z_{t_i} \leq z_{b_j}$, i.e., if the x-y projections of two polyhedral-cross-sections intersect, then one lies above the other. A polyhedral-cross-section is composed of *base-edges*, those that form the base-faces, and *lateral-edges* which together form the lateral-faces. Let Δ, a binary operator on simple polygons, be defined so that $P_i \,\Delta\, P_j = P_i - int(P_j)$. A lateral-edge links a vertex of each of P_{m_i} and P_{s_i}, i.e., of the type $v_{m_{ij}}, v_{s_{ik}}$, is denoted by $\overline{e_{ijk}}$, and is restricted so that

$e'_{i_{jk}} \in P'_{s_i} \Delta P'_{m_i}$. We represent a polyhedral-cross-section PX_i by $(P_{b_i}, z_{b_i}, P_{t_i}, z_{t_i}, F_i)$, and denote the complexity of the scene, $\sum_{i=1}^{m} |P_{b_i}| + |P_{t_i}| = \sum_{i=1}^{m} n_{b_i} + n_{t_i}$, by n.

2.3. Properties of the Scene

We present in this section a theorem pertaining to the two-dimensional properties of a polyhedral-cross-section. In order to prove the theorem we first propose several lemmas. Note that the reader may skip the proofs without any loss of continuity.

Lemma 2.1. A lateral-face f_i of a polyhedral-cross-section PX is either a triangle bounded by two lateral-edges and a base-edge, or a convex quadrilateral bounded by two lateral-edges and a pair of parallel base-edges, one from each of the base-faces.

Proof: Let V_b and V_t be the subsets of the vertices of P_b and P_t respectively that define f_i. Consider the plane B in which f_i lies and its intersection with the parallel planes B_b and B_t, defined as $z = z_b$ and $z = z_t$ respectively. The intersection between a pair of parallel planes and a third plane not parallel to the pair, is a pair of parallel lines. With respect to the intersection of B with B_b and B_t, we denote the pair of parallel lines by l_b and l_t. Referring to figure 2.1, since the vertices of V_b and V_t lie in l_b and l_t, and also since l_b and l_t are parallel, the vertices of each of V_b and V_t are consecutive vertices of f_i, and so must also be consecutive vertices of P_b and P_t. But the vertices of P_b and P_t are in standard form and so $|V_b| \leq 2$ and $|V_t| \leq 2$. In the case where $|f_i| = 3$, f_i is a triangle and so each pair of vertices define an edge, with the result that f_i is bounded by two lateral-edges and a base-edge. If on the other hand $|f_i| = 4$, then V_b and V_t determine the parallel lines l_b and l_t and so f_i is a convex quadrilateral bounded by two lateral-edges and a pair of parallel base-edges, one from each of the base-faces. Q.E.D.

Corollary. There are $O(n)$ lateral-faces in a scene.

Proof: Each edge of a polyhedron is common to two faces. Since each lateral-face is bounded by one or two base-edges, and each base-edge bounds the top or the bottom base-face, the number of lateral-faces is $O(n)$.

Define a *polygonal-chain* C as a chain of convex polygonal faces in which each link in the chain is an edge common to two adjacent faces and no two nonadjacent faces intersect.

Lemma 2.2. The set $F = (f_1, f_2, ..., f_k)$ of lateral-faces of a polyhedral-cross-section PX, form a closed polygonal-chain C linked by lateral-edges.

Proof: We know that each lateral-face f_i is convex and bounded in part by two lateral-edges. As well, each lateral-edge is shared by a pair of adjacent lateral-faces. Now, since the number of faces in a polyhedron is finite, the lateral-faces form a closed chain C with the lateral-edges as the links. Finally, no two lateral-faces intersect unless they are adjacent faces, and so C is a closed polygonal-chain. Q.E.D.

Lemma 2.3. Given two lateral-faces f_i and f_j of a polyhedral-cross-section PX, $\Gamma(f_i', f_j') = \varnothing$.

Proof: Let C denote the polygonal-chain of the lateral-faces of PX. If $\Gamma(f_i', f_j') \neq \varnothing$, then C' must be properly self-intersecting. However, if this is true, then either P_b' or P_t' self-intersects, or $bnd(P_m') \cap ext(P_s') \neq \varnothing$, both of which lead to contradictions. Q.E.D.

We are now ready to prove the main result of this chapter in which the general shape of a polyhedral-cross-section PX is deduced.

Theorem 2.1. The x-y projection of the set F of lateral-faces of a polyhedral-cross-section PX, represents a convex non-overlapping decomposition of $P_s' \Delta P_m'$.

Proof: This follows directly from lemmas 2.1 - 2.3. Q.E.D.

Referring to figure 2.2, consider the two-dimensional properties of a polyhedral-cross-section as proved in the theorem of this chapter; that the lateral-faces are convex and that their x-y projections decompose the difference between the x-y projections of the superior and minor base-faces, are used extensively in the development of an efficient solution to the hidden-surface problem.

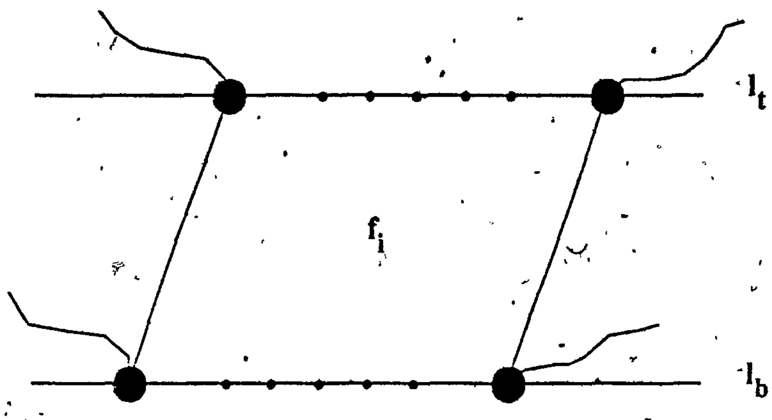

figure 2.1

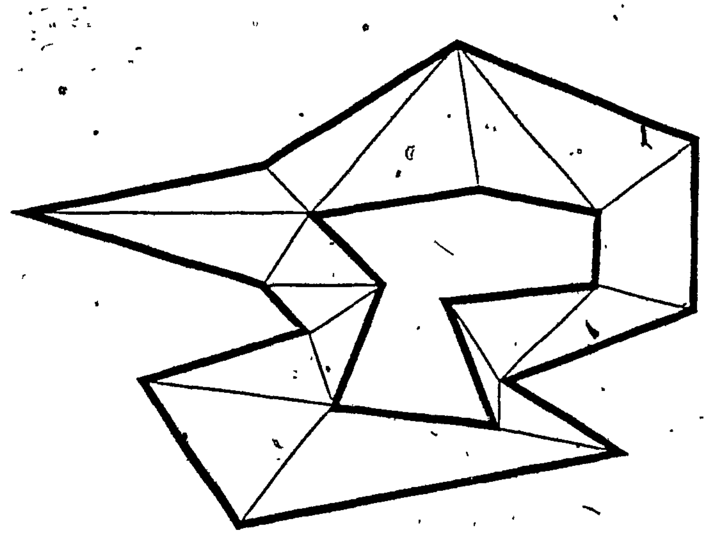

figure 2.2

Chapter 3

Elementary Scenes

In this chapter we consider the priority approach to hidden-surface removal with respect to scenes that, although comprised of polyhedral-cross-sections, are predominantly two dimensional. The polyhedral-cross-sections of these scenes are restricted so that the set of top base-faces and the set of bottom base-faces each lie in a fixed z-plane and each pair of base-faces is congruent.

In the first section of this chapter, we formalize the problem of computing priority orderings for a restricted class of scenes. In the second and last section, we reproduce, due to their importance with respect to this thesis, the results of Yao [27] on the priority approach to hidden-surface removal. The algorithm proposed by Yao involves two passes of the data set: the first determines a partial ordering of the faces of the scene and the second topologically sorts the ordering yielding a linear ordering of the faces. In addition, we present a new formalism for describing priority orderings which leads to a simple modification to Yao's algorithm, eliminating the need for the second pass. We have recently learned that this modification was independently discovered by Ottmann and Widmayer [28] within the context of line segment translation. We note however that our method of proof, on which another chapter of this thesis depends, is of a completely different flavor than that of Ottmann and Widmayer.

3.1. Problem Description

Consider a scene $S = (PX_1, PX_2, \ldots, PX_m)$ of polyhedral-cross-sections. Referring to figure 3.1, consider a class of input restricted so that for each polyhedral-cross-section PX_i, $P'_{b_i} = P'_{t_i}$, $z_{b_i} = z_b$, and $z_{t_i} = z_t$ where z_b and z_t are each a constant. Furthermore, lateral-edges link the similar vertices of each base-face. We shall refer to each polyhedral-cross-section PX_i

of such a scene, as a *prism*. Since $P'_{b_i} = P'_{l_i}$, we refer to each as P_i.

To define a dominance relation between the faces of a scene, requires that a viewing model be chosen. We choose the parallel viewing model since it affords a simple analysis and is of practical importance in many applications. In the parallel model, refer to figure 3.2, parallel rays emanate from an observer at infinity and head towards the scene. The observer's view is then completely determined by the pair of angles (θ, ϕ), $0 \le \theta \le 2\pi$ and $\frac{-\pi}{2} \le \phi \le \frac{\pi}{2}$, formed by the projections of a ray r onto the x-y and x-z planes respectively.

Define the *outward normal vector* of a face as the unit normal of the face directed away from the interior of the polygon. Given an observer, each face whose outward normal vector has no component in the direction of the observer, is invisible. We call such invisible faces, *back-faces*, and describe the remaining potentially visible faces as *visible*. Having discarded the back-faces, displaying the remaining visible faces with a valid priority ordering, results in a correctly rendered scene.

Consider a scene S of prisms and suppose $\phi \ne 0$, then either all the top base-faces or all the bottom base-faces are visible, otherwise $\phi = 0$, and then no base-faces are visible. Any ray r that intersects a visible base-face must do so before intersecting any visible lateral-face, and also may not intersect any other visible base-face. Thus each of the visible base-faces has an equal and highest priority. Solving the hidden-surface problem for a scene of prisms, using a priority based approach, is then a matter of determining a valid priority ordering for the visible lateral-faces. Consider the cases in which $\phi = \frac{-\pi}{2}$ or $\phi = \frac{\pi}{2}$. The solution in these cases is trivial since the lateral-faces are all back-faces. We assume therefore that $\frac{-\pi}{2} < \phi < \frac{\pi}{2}$.

Let F be a set of faces. Define $\Psi(F, r)$ to be the partial ordering of the faces of F induced by their order of intersection with a ray r. Let $F = (f_1, f_2, \ldots, f_n)$ be the lateral-faces of S. Consider any ray r in a fixed direction $(\theta, 0)$ that lies parallel to and between the planes $z = z_b$ and $z = z_t$. Let R be the family of rays for which for each $s \in R$, $s' = r'$. Since for each ray $s \in R$, $\Psi(F, s)$ and $\Psi(F, r)$ are consistent, the problem of determining a valid priority ordering for the visible lateral-faces of F is independent of ϕ. As a result, the problem can be further simplified: determining the required ordering is equivalent to determining, in two dimensions, a valid priority ordering for the visible edges of $F' = (f'_1, f'_2, ..., f'_n)$ in the direction θ. As a matter of convenience, an edge of F' will be referred to by its corresponding face in F.

Consider a clockwise *view-interval* $\omega = [\rho_1, \rho_2]$, defined so that $|\omega|$ is maximized with the condition that if f_i is visible for any angle $\theta \in \omega$, then f_i is visible for all angles $\theta \in \omega$. Since a face f_i is visible over an interval of length π, the complete interval $[0, 2\pi]$ is properly divided into at most n view-intervals, each of which contains $O(n)$ faces in general.

In the next section, two important results are discussed: one, that there exists a static priority ordering for a scene of prisms within any view-interval, and two, that this ordering can be computed quickly.

3.2. Problem Solution

We now present Yao's approach to computing priority orderings with some improvements. While the worst-case complexity remains the same, some simplifications to the algorithm are obtained. The simplifications arise due to the introduction of a new formalism for describing priority orderings. The new orderings introduced will be used in subsequent chapters.

Before a solution can be proposed, it is necessary to discuss the abstract representation of a scene. The problem of representing a scene S is easily resolved since it clearly suffices to represent S by S' and the bounding planes $z = z_b$ and $z = z_t$. Each polygon P_i of S' can be suitably represented by a doubly-linked-list of its vertices.

Given a scene S and a view-interval $\omega = [\rho_1, \rho_2]$ we can, without loss of generality, rotate the scene so that the view-interval can be expressed as $\omega = [0, \rho]$. Let F_ω, a subset of F, be the faces of the view-interval ω. If for a view-interval ω, a face f_j must be assigned a higher priority than an face f_i, we say that f_j dominates f_i and denote the relationship by f_j *dom* f_i. Referring to figure 3.3, consider an edge f_i and define the region R_i to include the two half-lines determining its boundary, but exclude the portion of f_i not lying on the half-lines. Suppose for view-interval ω that f_j *dom* f_i, then f_j must intersect the region R_i. Of the two vertices determining a face f_i, the tail is denoted by $v_{t_i} = (x_{t_i}, y_{t_i})$ and the head is denoted by $v_{h_i} = (x_{h_i}, y_{h_i})$. Referring to figure 3.4, suppose f_j *dom* f_i, then either f_j intersects the half-line boundary of R_i containing v_{t_i}, or it does not; these cases are denoted respectively by f_j *leftdom* f_i and f_j *rightdom* f_i.

Theorem 3.1. For any view-interval $\omega = [0, \rho]$ of a scene composed of prisms, there exists a priority ordering on the faces of F_ω (Yao [27]).

Proof: Referring to figure 3.5, consider the following three facts: (i) the relation *leftdom* is acyclic; (ii) if f_i *rightdom* f_j, then $x_{t_i} > x_{t_j}$; (iii) if f_i *leftdom* f_j and f_j *rightdom* f_k, then f_i *dom* f_k. Of the maximal faces with respect to *leftdom*, i.e., those that are not left-dominated, consider the one whose tail has the largest x-coordinate and denote it by f_m. We will now show that f_m is not dominated by any other face. First, suppose that the tails of two faces have the same x-coordinate, then one must left-dominate the other, consequently f_m is unique. Let f_i *dom* f_m, then since f_m is maximal with respect to *leftdom*, f_i *rightdom* f_m. By fact (i), there exists a noncyclic sequence f_k *leftdom* f_j *leftdom* $\cdots$ *leftdom* f_i such that

f_k is maximal with respect to *leftdom*. Note, if f_i is maximal with respect to *leftdom* then $f_k = f_i$. Applying fact (iii) to the sequence repeatedly yields f_k *dom* f_m. Now, since f_m is maximal with respect to *leftdom*, f_k *rightdom* f_m, and by fact (ii), $x_{t_k} > x_{t_m}$. But f_m was chosen so that, of the maximal elements with respect to *leftdom*, its tail had the largest x-coordinate, thus we have a contradiction. Since for any F_ω there exists an element f_m that is maximal with respect to *dom*, there exists a priority ordering on F_ω. **Q.E.D.**

Consider a relation *ileftdom*, defined so that f_j *ileftdom* f_i if and only if f_j left-dominates f_i immediately from above, i.e., no face intersects the left half-line of R_i below f_j. A face is of course maximal with respect to *leftdom* if and only if it is maximal with respect to *ileftdom*. Suppose we add a face $f_{\max}$ that left-dominates all other faces, then $f_{\max}$ is the only face maximal with respect to *ileftdom*. Since each face, with the exception of $f_{\max}$, is immediately left-dominated by only one face, the relation *ileftdom* can be represented by a tree T rooted by $f_{\max}$. Let T be arranged so that the children of a node f, those immediately left-dominated by f, are ordered from left to right by the value of the x-coordinate of their tail. Suppose the subtrees of a tree T, ordered from left to right, are $T_1, T_2, ..., T_r$. Consider the following recursive definition of the left to right *postorder* traversal of T: list the nodes of $T_1, T_2, ..., T_r$ in postorder all followed by the root of T. Thus, if the children of a node h, ordered from left to right, are $h_1, h_2, ..., h_s$, then in the postorder listing of T the nodes $h_1, h_2, ..., h_s, h$ appear in the given order.

Theorem 3.2. The left to right postorder traversal of the tree T yields a priority ordering on F_ω, which can be optimally calculated in $O(nlogn)$ time using $O(n)$ space.

Proof: Let f be a face of F_ω, then referring to figure 3.6, let T_f be the subtree of T in which the faces occurring before f in a left to right postorder traversal of T, have been eliminated. Also, let L_f be the left most path, from root to leaf, of T_f and note that the leaf of L_f is f. It is sufficient to show, by theorem 3.1, that given f and the faces of $T - T_f$, of the faces maximal with respect to *leftdom*, the tail of f has the largest x-coordinate. Referring to figure 3.7, consider the partition of the faces in F_ω induced by the *ileftdom* sequence represented by L_f. Denote the partitioning line by C_f and note that C_f is either piecewise linear and descends from left to right, or is vertical. Also, by the definition of *ileftdom* no two partitioning lines may cross. Clearly then, given a face of $T - T_f$, either its tail lies left of C_f or it is a descendant of f in T. Therefore, given f and the faces of $T - T_f$, f is maximal with respect to *leftdom*, and of those faces that are maximal with respect to *leftdom*, the tail of f is rightmost.

It now remains to show that the postorder listing can be computed in $O(nlogn)$ time. Suppose the faces are processed so that a face f and those faces immediately left-dominated by f, $f_1, f_2, ..., f_k$, ordered by the x-coordinate of their tail, are encountered in the order $f, f_1, f_2, ..., f_k$. This enables the construction of a doubly-linked-list in which a face is inserted before the face that immediately left-dominates it, achieving the desired suborder of $f_1, f_2, ..., f_k, f$.

The problem is solved with a plane sweep technique similar to that used by Bentley and Ottmann [29]. Referring to figure 3.8, consider a vertical line l through v_t, the tail of a face f, and its intersection with the elements of F_ω. The face of F_ω that intersects l directly above v_t, immediately left-dominates f. Since no two faces intersect, the ordering of the intersections of the faces with l as it is swept from left to right, changes only as an end point of a face is encountered. The ordering, as l is swept from left to right, can therefore be maintained in a balanced tree in which a face f is inserted when its tail is processed, and deleted when its head is processed. The face that immediately left-dominates f is found when f is inserted. Since the tails are encountered from left to right, a face f and $f_1, f_2, ..., f_k$, those faces immediately left-dominated by f, are encountered in the order $f, f_1, f_2, ..., f_k$ as desired.

In $O(n)$ time the back-faces can be eliminated and the scene rotated so that $\omega = [0, \rho]$. Computational details concerning back-face elimination and scene rotation can be found in [3]. The end points can be sorted according to their x-coordinate, with special attention paid to points with the same x-coordinate, in $O(nlogn)$ time. Each insertion into and each deletion from the balanced tree can be done in $O(logn)$ time. In addition, $O(logn)$ time is required following each insertion in order to determine which face immediately left-dominates the inserted face. Thus the total time spent manipulating the balanced tree is $O(nlogn)$. Since each insertion into the doubly-linked-list representing the priority ordering requires $O(1)$ time, its construction requires $O(n)$ time. The priority ordering can therefore be determined in $O(nlogn)$ time, and since each data structure used has $O(n)$ space requirements, using $O(n)$ space.

The optimality of the algorithm follows simply since sorting is linear time transferable to the priority ordering problem. Consider a set $X = (x_1, x_2, \ldots, x_n)$ of n distinct integers. Of the elements of X, let $x_{\max}$ be the largest. Suppose we wish to sort X in descending order. Referring to figure 3.9, map each x_i to a horizontal line segment as follows: $x_i \rightarrow \left[(x_i, x_{\max} - x_i), (x_{\max} + 1, x_{\max} - x_i)\right]$. Clearly, the resulting set of line segments has a unique priority ordering for the direction $\theta = 0$, and this ordering corresponds to the sorted values of X. Since the transformations obviously require linear time, the algorithm is optimal. **Q.E.D.**

A scene is said to be *k-regular* if number of view intervals is k. In general, no two faces will be visible over the same interval of length π and so $k = n$, however, there exists scenes in which $k \ll n$. By angular sorting, the k view-intervals can be calculated in $O(nlogn)$ time. The k priority orderings, which are sufficient for all views, and the corresponding k lists can be calculated in $O(knlogn)$ time and stored in $O(kn)$ space. Displaying the scene from a given view point (θ, ϕ) requires an $O(logk)$ time search to locate the required view-interval, $O(n)$

time to project the scene, and $O(n)$ display commands to render an image. The computational particulars regarding scene projection and displaying can be found in [3].

In the next chapter we extend the theory so far developed to include more general classes of scenes. In these scenes the base-faces are no longer restricted to two z-planes and each pair of base-faces are not necessarily congruent.

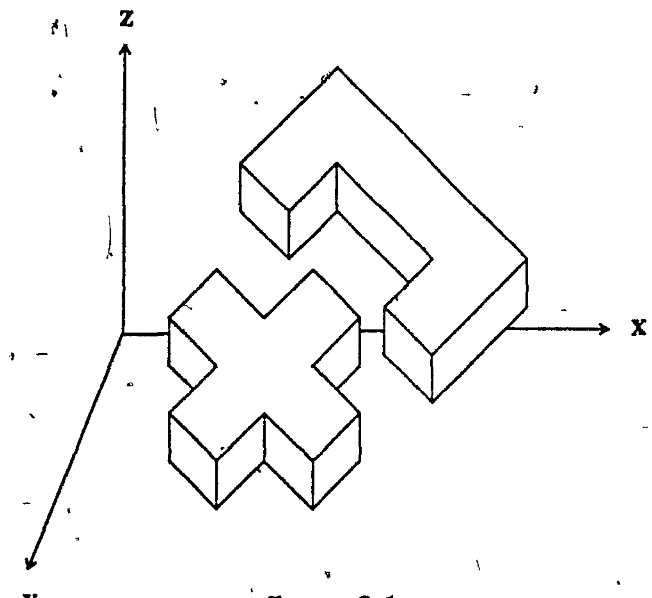

figure 3.1

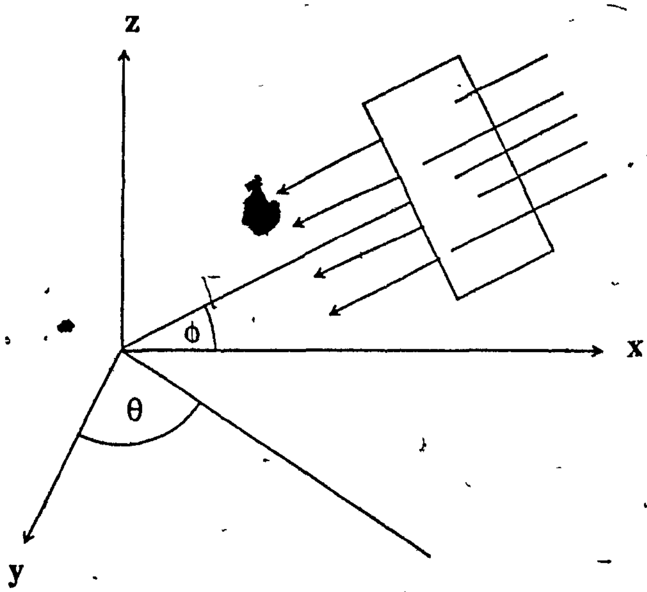

figure 3.2

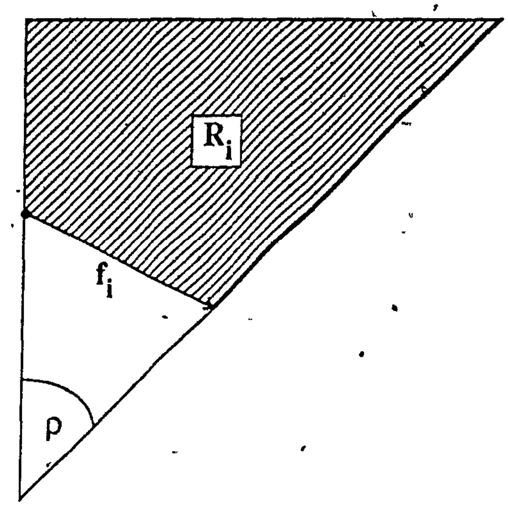

figure 3.3

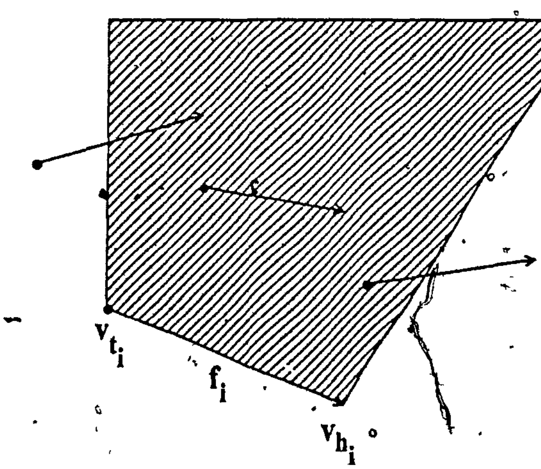

figure 3.4

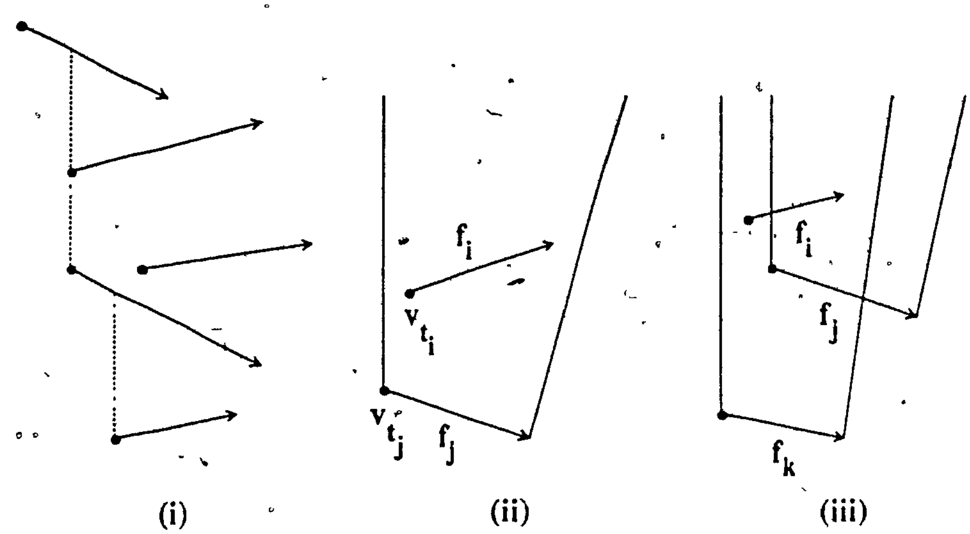

figure 3.5

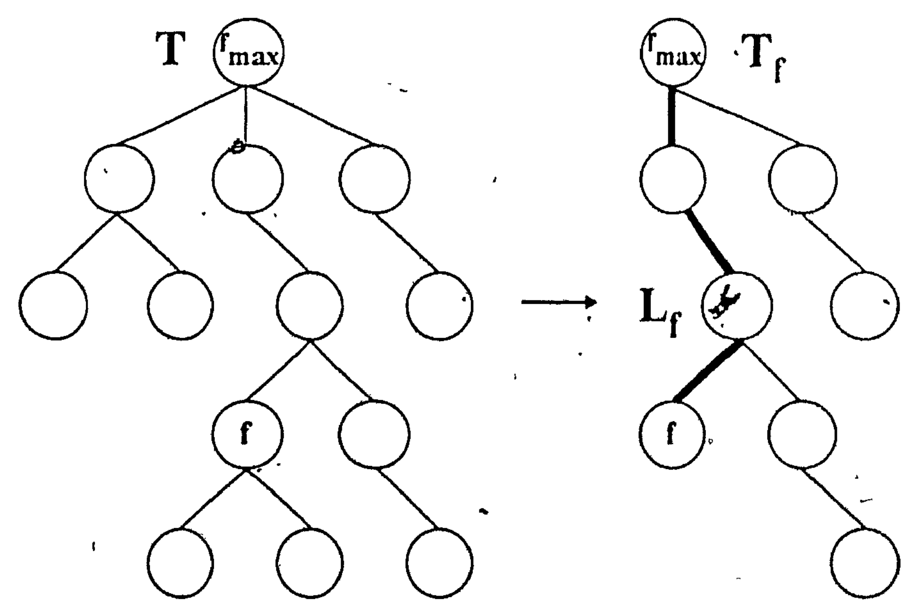

figure 3.6

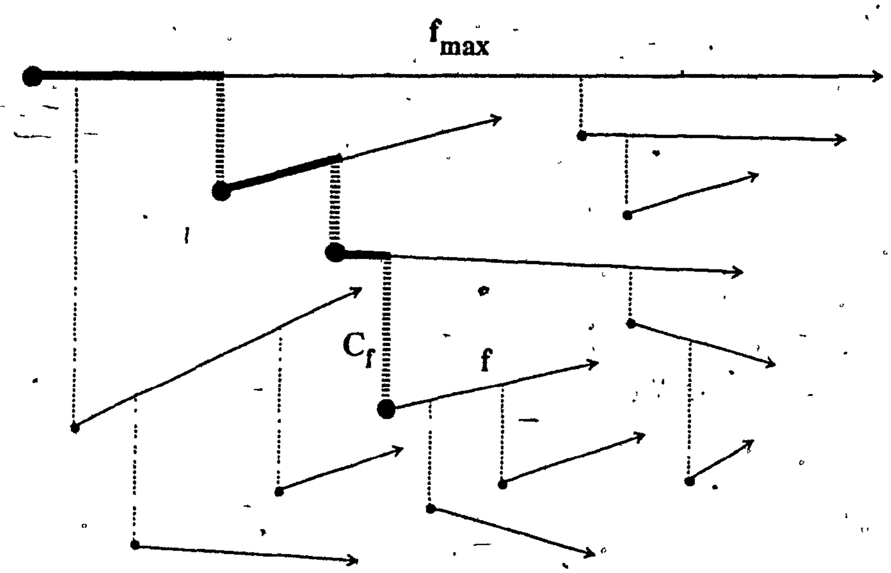

figure 3.7

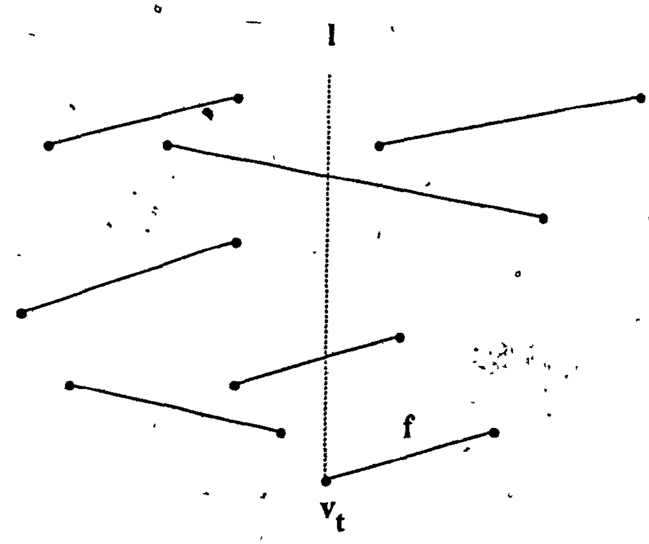

figure 3.8

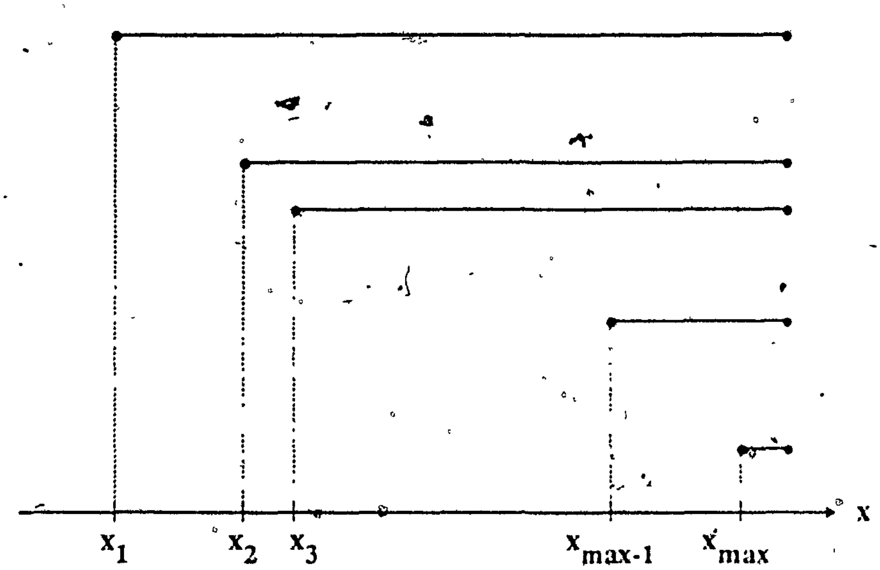

figure 3.9

Chapter 4

Complex Scenes

So far we have considered scenes of polyhedral-cross-sections whose two-dimensional properties afford an efficient solution to the hidden-surface problem. These two-dimensional properties resulted largely from the placement of the base-faces. In this chapter we examine a more general class of scenes, in which the placement of base-faces is not so rigidly confined. In general, these scenes do not admit priority orderings on their faces, i.e., the corresponding *leftdom* relation is cyclic. To remedy this situation, a scene is decomposed so as to eliminate potential problem areas.

In the first section of this chapter, we discuss nonoverlapping scenes, those for which no two x-y projections of superior base-faces intersect. For these scenes we consider vertical decompositions in order to avoid problem situations. The most general class of scenes is treated in the last section. These scenes require both vertical and horizontal decompositions to eliminate potential problem areas.

4.1. Nonoverlapping Scenes

Consider a scene $S = (PX_1, PX_2, ..., PX_m)$ of polyhedral-cross-sections and let $F = (f_1, f_2, ..., f_n)$ be the corresponding lateral-faces. Restrict S so that for any pair PX_i, PX_j of polyhedral-cross-sections, $\Gamma(P_{s_i}', P_{s_j}') = \varnothing$. Call each polyhedral-cross-section of such a scene a *column*. As remarked in the previous chapter, an observer's view-point in the parallel viewing model is completely determined by the pair of angles (θ, ϕ), $0 \le \theta \le 2\pi$ and $\frac{-\pi}{2} \le \phi \le \frac{\pi}{2}$. We assume $\frac{-\pi}{2} < \phi < \frac{\pi}{2}$ since we will consider the special cases in which $\phi = \frac{-\pi}{2}$ or $\phi = \frac{\pi}{2}$ in section 4.2.

Unlike in a scene composed of prisms, the top and bottom base-faces of a scene constructed from columns do not necessarily lie in respective z-planes. Consequently, referring to figure 4.1, for a fixed viewing position (θ, ϕ), the visible base-faces do not necessarily have equal and highest priority. Thus it is no longer sufficient to simply determine a valid priority ordering for the set of lateral-faces. The question then is, is it even possible to compute a priority ordering for the combined set of lateral-faces and base-faces? In general, the answer is no. Referring to figure 4.2, it is simple to construct a scene of columns in which for any viewing position, there exist a base-face and lateral-face that determine a cycle, i.e., neither can have a higher priority than the other. To remedy this situation, we will introduce a vertical decomposition of the scene which easily adapts to the existing framework.

It is of course desirable to render the problem independent of ϕ, and hence reduce it to two dimensions. Note however, that a lateral-face in general position may not, for a fixed value of θ, be visible through the complete range of ϕ, yet may, for specific values of ϕ, be visible through the entire range of θ. Clearly, this disqualifies from consideration any method that computes a priority ordering after the elimination of the back-faces. Instead, we adopt a strategy that computes a view-interval dependent total ordering of the faces in a scene. Given a viewing position, the back-faces can then be quickly eliminated.

Suppose each minor base-face P_{m_i} of S is triangulated. Euler showed that a planar graph on n vertices has $O(n)$ edges and faces. Consequently, referring to figure 4.3, the decomposition of the minor base-faces yields $O(n)$ *triangular-faces* and induces a vertical decomposition of S. Redefine $F = (f_1, f_2, ..., f_n)$ to include both the lateral-faces and triangular-faces of S. Note that the elements of $F' = (f_1', f_2', ..., f_n')$ are edges and convex polygons, and $\Gamma(f_i', f_j') = \varnothing$ for any pair f_i, f_j of F.

Lemma 4.1. Any priority ordering on the elements of F' for a fixed direction θ, is valid on the faces of F for every direction (θ, ϕ).

Proof: Let r be any ray with direction θ in the x-y plane. Define R to be the family of rays for which for each ray $s \in R$, $s' = r$. In order to establish the required result, it is sufficient to demonstrate that for any ray $s \in R$, $\Psi(F, s)$ and $\Psi(F', r)$ are consistent. Let f_i' be any face of F, and s any ray of R. Since f_i is convex, s intersects f_i and r intersects f_i' at most once. Then, referring to figure 4.4, since for any pair f_i, f_j of F, $\Gamma(f_i', f_j') = \varnothing$, $\Psi(F, s)$ and $\Psi(F', r)$ are consistent. Q.E.D.

All that remains, is to bring the superior base-faces into the argument. Let F_i be the set of lateral-faces and triangular-faces of a column PX_i. F_i' defines a non-overlapping decomposition of P_{s_i}' in that $F_i' = P_{s_i}'$. Since each non vertical face of F_i is associated with a unique component of the decomposition of P_{s_i}, it suffices to compute a priority ordering solely on the faces of F'.

In order to process a scene S, a suitable representation of each column of S is required. From such a representation, the base-faces and lateral-faces must be immediately available. To satisfy this condition a planar graph structure, such as the doubly-connected-edge-list of Muller and Preparata [30], is used. Note that due to vertical lateral-faces, the x-y projections of two edges may overlap. However, due to the superior and minor labels of the base-faces, conflicts can be resolved and so a planar graph structure is still appropriate. The main component of the doubly-connected-edge-list is the *edge-node.* There is a one-to-one correspondence between the edge-nodes and the edges of the graph. Each edge-node consists of six fields named V_1, V_2, F_1, F_2, P_1, and P_2. The fields V_1 and V_2 contain the names of the head and tail of the edge respectively, effectively orienting the edge. Given this orientation, the fields F_1 and F_2 contain the names of the faces which lie to the left and to the right of the edge respectively. The field P_1 is a pointer to the edge-node containing the next edge about V_1 in the counterclockwise direction. P_2 is defined analogously. This representation is suitable since for any edge it is possible to start walking clockwise around either of its adjacent faces. In addition, each step of the walk involves only a constant amount of work, and so the faces can be retrieved in $O(n)$ time. So that the lateral-faces and base-faces can be distinguished, a type indicator field is included with each edge-node.

Before attempting to compute a priority ordering on F, it is necessary to triangulate the minor base-faces of F. Since the triangulation of a minor base-face does not effect the planarity of the corresponding column, the doubly-connected-edge-list remains a suitable representation. So that the triangular-faces can be distinguished, the type indicator of each edge-node representing a triangulation edge is appropriately set.

Lemma 4.2. The set $M = (P_{m_1}, P_{m_2}, ..., P_{m_m})$ of minor base-faces can be triangulated, and the corresponding doubly-connected-edge-lists updated, in $O(nlogn)$ time using $O(n)$ space.

Proof: Many algorithms [31-34] exist for triangulating a simple polygon in $O(nlogn)$ time and $O(n)$ space. With respect to this thesis, these results are sufficient. However, it should be mentioned that Tarjan and Van Wyk [35] have recently discovered an $O(nloglogn)$ time algorithm. Each minor base-face is a simple polygon. Since there are $O(n)$ vertices determining the m minor base-faces, the m minor base-faces can be extracted from the doubly-connected-edge-lists in $O(n)$ time, and subsequently triangulated in $O(nlogn)$ time and $O(n)$ space. Given a list of the $O(n)$ triangulation edges, it remains to show that the doubly-connected-edge-lists can be quickly updated. Consider a minor base-face P_m and the corresponding doubly-connected edge-list. Allocate an edge-node for each triangulation edge, and arbitrarily designate the head and tail. Referring to figure 4.5, sort the the triangulation edges so that for each vertex v of P_m, the edges with v as an endpoint are sorted counterclockwise between the bounding edges

e_+ and e_- of P_m at v. From this information, in constant time it is possible to update the P_1 and P_2 fields of any edge-node with P_m as a bordering face. With equal ease, the P_1 and P_2 fields of the new edge-nodes can be set. To update the F_1 and F_2 fields, the information contained in the P_1 and P_2 fields is used. Consider a triangulation edge e. Referring to figure 4.6, it is possible, in constant time, to determine the edges bounding the two triangles bordering e and subsequently update their F_1 and F_2 fields. The sort step dominates the updating procedure, and so, since there are $O(n)$ triangulation edges, updating the doubly-connected-edge-lists can be accomplished in $O(nlogn)$ time using $O(n)$ space. Q.E.D.

Ideally, the treatment of convex polygons, with respect to computing priority orderings, would not differ from that for edges. We shall now show that, with only a few extra considerations, this is in fact true. Let P be a convex polygon. A line l is a *line of support* of P if the interior of P lies completely to one side of l. A pair of vertices v_i, v_j of P is an *antipodal pair* if it admits parallel lines of support. Call the edge e determined by an antipodal pair, a *shadow-edge*.

Lemma 4.3. When computing a priority ordering for a fixed direction θ, it suffices to replace each polygon of F' by an appropriate shadow-edge.

Proof: Referring to figure 4.7, consider the parallel lines of support of a polygon f_i' of F' in the direction θ, and let e denote the corresponding shadow-edge determined by the antipodal pair v_j, v_k. Since f_i' is convex, e lies within f_i', and, as remarked by Guibas and Yao [36], f_i' and e sweep the same area when translated in the direction θ. Furthermore, for any pair of faces f_i, f_j of F, $\Gamma(f_i', f_j') = \emptyset$, and so e_i and e_j, the shadow-edges of f_i and f_j with respect to θ, do not intersect. However, e_i and e_j may overlap. Fortunately, this is not a problem since each face of F is either a triangle or a quadrilateral, and so in constant time the ordering of f_i' and f_j' with respect to θ can be computed. Finally, since no edge and shadow-edge of F' intersect (overlap is handled as above), it suffices to replace each polygon of F' by its shadow-edge for the direction θ. Q.E.D.

Lemma 4.4. The polygons of F' have $O(n)$ shadow-edges, each valid through some range of θ, which can be computed in $O(n)$ time.

Proof: Shamos [37] showed, for a convex polygon P on n vertices, that the $O(n)$ antipodal pairs of of P can be computed in $O(n)$ time. In addition, referring to figure 4.8, each antipodal pair defines a family of parallel lines of support through a clockwise *angular-interval* $\alpha = [\sigma_1, \sigma_2]$ and its reflection $\alpha_r = [\sigma_1+\pi, \sigma_2+\pi]$. Note that $|\alpha| = |\alpha_r| < \pi$. The result then follows simply since each antipodal pair defines a shadow-edge, and also since the polygons of F' are determined by a toatl of $O(n)$ vertices. Q.E.D.

For a scene S, there are then $O(n)$ edges and shadow-edges. Associated with each edge e are two nonoverlapping intervals of length π, reflecting the distinct sides of e. The visibility of each side of e will be associated with the corresponding interval. Likewise, the two angular-intervals α and α_r of a shadow-edge e, define the visibility of the two sides of e Let $E = (e_1, e_2, ..., e_n)$ be the edges and shadow-edges of F'. A view-interval $\omega = [\rho_1, \rho_2]$, is redefined so that $|\omega|$ is maximized with the condition that if e_i is visible for any angle $\theta \in \omega$, then e_i is visible for all angles $\theta \in \omega$. The visibility of each edge $e_i \in E$ is defined with respect to two equal but opposite intervals. As a result, each view-interval $\omega = [\rho_1, \rho_2]$ has a mirror image $\omega_r = [\rho_1+\pi, \rho_2+\pi]$. Since $\theta \in \omega$ if and only if $\theta+\pi \in \omega_r$, reversing the priority ordering determined for ω yields a valid priority ordering for ω_r. Therefore, rather than considering the complete interval $[0, 2\pi]$, it is sufficient to determine priority orderings over the interval $[0, \pi]$. Without loss of generality, S can be rotated so that a view-interval $\omega = [\rho_1, \rho_2]$ can be expressed as $\omega = [0, \rho]$. Clearly, the interval $[0, \pi]$ is properly divided into $O(n)$ view-intervals, each of which contains $O(n)$ edges.

Theorem 4.1. For any view-interval $\omega = [0, \rho]$ of a scene composed from columns, there exists a priority ordering on F' which can be optimally calculated in $O(nlogn)$ time and $O(n)$ space.

Proof: The proof follows directly from lemmas 4.1-4.4 and theorem 3.2. Q.E.D.

Given a *k-regular* scene composed of columns, the minor base-faces can be triangulated and the k view-intervals computed in $O(nlogn)$ time and $O(n)$ space. The corresponding k priority orderings can be determined in $O(knlogn)$ time and $O(kn)$ space. In order to display the scene from a view-point (θ, ϕ), the appropriate view-interval, which can be computed in $O(logk)$ time, must first be determined. Next, in $O(n)$ time, the back-faces can be eliminated and the scene projected. Since each non vertical face has a portion of a major base-face associated with it, the relative ordering of the pair must be considered in the case where neither is a back-face. Suppose this is the case, their relative ordering will then be arbitrary since otherwise a ray in the direction (θ, ϕ) must intersect both, with the result that one must be a back-face. Finally, $O(n)$ display commands are needed to render an image. Note that if the base-faces are confined to two z-planes as in the previous chapter, then the results simplify since the triangulation of the minor base-faces is not required.

In the next section the most general class of scenes is considered. In these scenes the x-y projections of two polyhedral-cross-section may intersect.

4.2. General Scenes

We now consider the most general class of scenes. Let $S = (PX_1, PX_2, ..., PX_m)$ be a scene of polyhedral-cross-sections. The placement of the polyhedral-cross-sections is restricted so that given any pair PX_i, PX_j, if $\Gamma(P'_{s_i}, P'_{s_j}) \neq \emptyset$ and $z_{b_i} < z_{b_j}$, then $z_{t_i} \leq z_{b_j}$. This restriction limits the placement of the polyhedral-cross-sections so that if the x-y projections of any pair intersect, then there exists a z-plane which separates the pair. We now ask the following question: is it possible to compute a priority ordering on the faces of such a scene? In general, the answer is no. Referring to figure 4.9, Yao [27] showed that it is possible to construct scenes in which for any viewing position (θ, ϕ), $\frac{-\pi}{2} < \phi < \frac{\pi}{2}$, there exists a set of lateral-faces that determine a cycle. In order to avoid such a situation, we introduce a horizontal decomposition of the scene.

First consider the cases in which $\phi = \frac{-\pi}{2}$ and $\phi = \frac{\pi}{2}$. For any two lateral-faces $f_j, f_k \in PX_i$, $\Gamma(f_j', f_k') = \varnothing$. Also, if given a pair of polyhedral-cross-sections PX_i, PX_j for which $\Gamma(P_{s_i}', P_{s_j}') \neq \varnothing$, then PX_i and PX_j are separable by a z-plane. Consequently, if the top base-faces are sorted and renamed so that $z_{t_1} \leq z_{t_2} \leq \cdots \leq z_{t_m}$, then assigning each face of a polyhedral-cross-section PX_i the priority i, induces a priority ordering on the faces for $\phi = \frac{\pi}{2}$. A similar result holds for $\phi = \frac{-\pi}{2}$. Since this process amounts to simple sorting, we will assume that $\frac{-\pi}{2} < \phi < \frac{\pi}{2}$.

Consider partitioning space into $t + 1$ horizontal *slabs* with a series of t z-planes $z = z_1 < z = z_2 < \cdots < z = z_t$. Suppose a scene S is decomposed by such a partitioning into $t + 1$ subscenes so that within each subscene $\Gamma(P_{s_i}', P_{s_j}') = \varnothing$ for any pair PX_i, PX_j of polyhedral-cross-sections. Any ray r in a fixed direction $(0, \phi)$ either passes through a single slab $(\phi = 0)$ or traverses the slabs in a fixed order. In the case where $\phi < 0$, r passes through the slabs bottom-up intersecting the z-planes in the order $z = z_1, z = z_2, \ldots, z = z_t$. The ordering is simply reversed if $\phi > 0$. It therefore suffices to process and display the subscenes independently. For each subscene the priority orderings are computed as in section 4.1. Rendering an image from a fixed viewing position $(0, \phi)$, involves displaying the subscenes individually based on the order of intersection of a ray r in the direction $(\theta, \phi+\pi)$ with the corresponding slabs. Note that this strategy may decompose a scene even though no cycles are present.

Determining where to cut a scene is a major consideration since it could adversely effect the complexity of the scene. Minimizing the complexity of the scene, i.e., minimizing the number of lateral-faces cut by the z-planes, is a difficult problem. Instead, we concentrate on minimizing the number of cuts. A scene S is said to be *t-cuttable* if t is the minimum number of z-planes required to decompose S so that within each subscene, no two x-y projections of superior base-faces intersect. We now present an algorithm that decomposes a scene S as required. The algorithm determines at most $2t$ z-planes and so minimizes within a constant factor.

The problem of deciding where to cut a scene is basically one of determining two-dimensional intersections. Given two polyhedral-cross-sections PX_i and PX_j such that $\Gamma(P_{s_i}', P_{s_j}') \neq \varnothing$ and $z_{b_i} < z_{b_j}$, the scene must be cut with some z-plane $z = z_c$, $z_{t_i} \leq z_c \leq z_{b_j}$. Suppose the scene is cut with a series of z-planes $z = z_{t_1}, z = z_{t_2}, ..., z = z_{t_m}$. Clearly, such a decomposition always appropriately cuts the scene, and so $t \leq m$. It is easy to realize scenes in which m cuts are necessary simply by stacking polyhedral-cross-sections one on top of another. Consider the x-y projection of a scene. In the worst case as many as $O(n^2)$ intersections will exist between the x-y projections of the superior base-faces, and so any algorithm that computes all the intersections will require $O(n^2)$ time in the worst case. Since at most $O(n)$ cuts are required to decompose a scene, it would be advantageous to eliminate the excess from consideration. Consider a polyhedral-cross-section PX_i and let $I_i = \{j \mid \Gamma(P_{s_i}', P_{s_j}') \neq \varnothing \text{ and } z_{b_i} < z_{b_j}\}$. Also, let $\min_i = \min(z_{b_j}),\ j \in I_i$. Clearly, cutting the scene with the z-plane $z = z_c$, $z_{t_i} \leq z_c \leq \min_i$, eliminates the intersections above, and in part due to, PX_i.

The key to the quickness of our algorithm will lie in its ability to locate the intersections between polyhedral-cross-sections in close proximity. The algorithm uses a divide-and-conquer scheme. During the divide phase, the scene is decomposed with a set of $O(n)$ z-planes This is followed by the conquer phase which then selects at most $2t$ of the z-planes. At the heart of the algorithm is intersection testing, determining whether or not any pair of x-y projections of superior base-faces intersect. In general the superior base-faces are simple polygons, a class of polygons which do not lend themselves to the existing, fast algorithms. To remedy this situation we assume the superior base-faces have been decomposed. Consider the decomposition of each superior base-face, induced by its lateral-faces and the triangulation of its minor base-face. As explained in section 4.1, such a decomposition requires $O(nlogn)$ time and $O(n)$ space to compute, and yields $O(n)$ components. Of the components, which are line segments, triangles, and convex quadrilaterals, the line segments are redundant with respect to the relevant intersection testing, and so are ignored. The plane-sweep algorithm of Shamos and Hoey [38] is used to detect intersections. Given a set of n triangles and quadrilaterals, the algorithm can detect whether any pair of objects intersects in $O(nlogn)$ time using $O(n)$ space. Using this algorithm, a 0-*cuttable* scene could be quickly detected.

Theorem 4.2. For any scene S that is *t-cuttable*, a set of at most $2t$ z-planes that properly decompose S, can be computed in $O(nlognlogm)$ time using $O(n)$ space.

Proof: For each polyhedral-cross-section PX_i, let t_i and b_i denote z_{b_i} and z_{t_i} respectively, and let D_i denote the set of components of the decomposition of P_{s_i}. Sort the t_i's and b_i's separately, and rename the polyhedral-cross-sections so that $t_1 \le t_2 \le \cdots \le t_m$ Merge the sorted sequences of t_i's and b_i's using the convention that if $t_i = b_j$, then in the ordering t_i comes before b_j. Call the resultant sequence Q and append to it, as its bottommost symbol, the dummy symbol t_0. Now each intersection can be characterized as follows: suppose $i < j$, then $t_i \leq b_j$ and $\Gamma(D_i, D_j) \neq \varnothing$. To complete the divide phase, consider the triple $G_i = (Q_i, B_i, T_i)$. Q_i is the subsequence of Q above t_{i-1}, up to and including t_i. B_i and T_i, which denote the bottom and top search boundaries within G_i, are respectively set equal to the first and last symbols of Q_i. Note that by the definition of a scene, each G_i initially defines a slab within which there are no intersections.

At each level of the conquer phase adjacent pairs of G_i's are merged, and any intersection between the pair is detected. If any intersection is detected, then a *cut* splitting the pair is introduced and any intersections straddling the cut are eliminated. Let r denote the number of G_i's at the current level of the conquer phase, thus initially $r = m$. At each level, for all odd i, $1 \leq i \leq r$, let $j = \frac{i+1}{2}$. If $i+1 \leq r$ then G_i and G_{i+1} are merged into G_j, otherwise G_i is simply renamed G_j. After each level, r is updated as follows: if r is odd $r = \frac{r+1}{2}$, otherwise $r = \frac{r}{2}$.

If at each level the intersections between the merged pairs are detected and eliminated, then clearly the resulting set of cuts will appropriately decompose S. Once an intersection has been detected, and a cut made, it would be senseless to search for intersections straddling the cut. To prevent this from happening, when G_i and G_{i+1} are merged, only intersections between B_i and T_{i+1} will be considered. Note that from B_i to the topmost symbol of Q_i, and from the bottommost symbol of Q_{i+1} to T_{i+1}, there are no intersections. Suppose G_i and G_{i+1} are about to be merged, then any intersection between the pair can be characterized as follows: if $j < k$ then $t_j \in Q_i$, $t_j \geq B_i$, $b_k \in Q_{i+1}$, $b_k \leq T_{i+1}$, and $\Gamma(D_j, D_k) \neq \varnothing$. Let $V_i = \{j \mid t_j \in S_i\}$ and let $W_i = \{j \mid b_j \in S_i\}$, then detecting an intersection involves determining for any pair D_j, D_k, $j \in V_i$ and $k \in W_{i+1}$, whether $\Gamma(D_j, D_k) \neq \varnothing$. For this purpose, we use the algorithm of Shamos and Hoey. If an intersection is detected, then cutting at t_j, the topmost symbol of Q_i, eliminates all intersections between G_i and G_{i+1}. What remains is to merge G_i and G_{i+1} into G_j. There are two cases to consider depending on whether or not an intersection is detected. In both cases Q_j is determined by concatenating Q_i and Q_{i+1}. Referring to figure 4.10, if an intersection is detected then $T_j = T_i$ and $B_j = B_{i+1}$. Note that if $B_k < T_k$ then Q_k has not been cut. Referring to figure 4.11, consider the case in which an intersection is not detected. If $B_i < T_i$ then $T_j = T_{i+1}$, otherwise $T_j = T_i$. On the other hand, if $B_{i+1} < T_{i+1}$ then $B_j = B_i$, otherwise $B_j = B_{i+1}$.

Let us consider the complexity of the algorithm. The space requirements are clearly $O(n)$. In the divide phase the running time is dominated by the sorting, and so $O(mlogm)$ time is required. Since at each level of the conquer phase $\left\lfloor \frac{r}{2} \right\rfloor$ merges occur, there are $O(logm)$ levels. At each level the intersection detection computations dominate the running time. Since the sum of the number of components of the D_i's is $O(n)$, and since each component is considered at most twice, once for each of t_i and b_i, the total time time spent detecting intersections at each level is $O(nlogn)$. Therefore, the running time of the algorithm is $O(nlognlogm)$.

What remains to be shown is that at most $2t$ cuts are made. Referring to figure 4.12, suppose that while merging G_i and G_{i+1}, an intersection is detected. Let j and k, $j < k$, denote the intersection pair, then $t_j \in Q_i$ and $b_k \in Q_{i+1}$. Also, let c denote the topmost symbol of Q_i. Clearly, the line segment $l = (t_j, b_k)$ must be cut. Choosing c achieves this and ensures all intersections straddling c are eliminated, it does not however guarantee minimality. Let d denote the number of cuts made. It is possible that an intersection will be detected between G_i and what is below G_i, and between G_{i+1} and what is above G_{i+1}. Still referring to figure 4.12, let l_b and l_t denote the line segments that would need to be cut. Clearly, l and l_b, and, l and

l_t may overlap, however, l_b and l_t will not. Thus, if we consider the sequence of d cuts in bottom-to-top order, then of the corresponding d segments, every second segment is non-overlapping. Hence at least $\left\lceil \frac{d}{2} \right\rceil$ cuts are required and so at most $2t$ cuts have been made.

Q.E.D.

Given a *t-cuttable* scene, the minor base-faces can be triangulated in $O(nlogn)$ time using $O(n)$ space. A set of at most $2t$ cuts, which appropriately decompose S, can be determined in $O(nlognlogm)$ time and $O(n)$ space. Cutting a polyhedral-cross-section PX is simple since each of the resultant objects has the same topology as PX. In order to determine which polyhedral-cross-sections are cut, sort the cuts and denote the resulting list by $C = (c_1, c_2, ..., c_t)$. Next, merge Q and C, ordering t_i before c_j if $t_i = c_j$. Now, scan the resultant list, inserting PX_i into an active list when b_i is encountered, and deleting it when t_i is encountered. Further, when c_i is encountered, output it and the active list. Therefore, the scene can be cut in $O(tn+tlogt)$ time and stored in $O(tn)$ space. Let us say a scene is *k-regular* if the maximum number of view-intervals in any slab, is k. In total, $O(tnlogn)$ time and $O(tn)$ space are required to determine the $O(kn)$ view-intervals. The corresponding priority orderings can be computed in $O(tknlogn)$ time and stored in $O(tkn)$ space. In order to display a scene from a view-point (θ, ϕ), the appropriate view-intervals, which can be determined in $O(tlogk)$ time, must first be determined. Then in $O(tn)$ time the back-faces must be eliminated and the scene projected. Finally, $O(tn)$ display commands render an image.

In the next chapter we consider the insertion and deletion of edges from priority orderings. These problems are a fundamental concern when maintaining dynamic scenes.

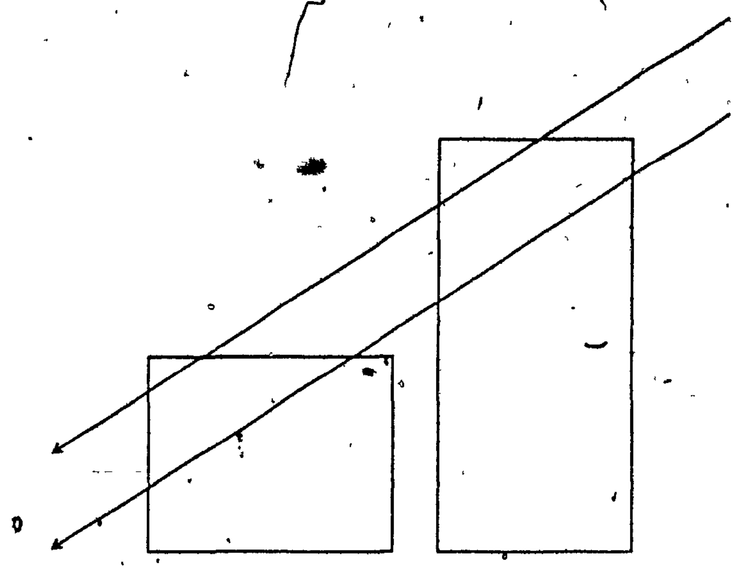

figure 4.1

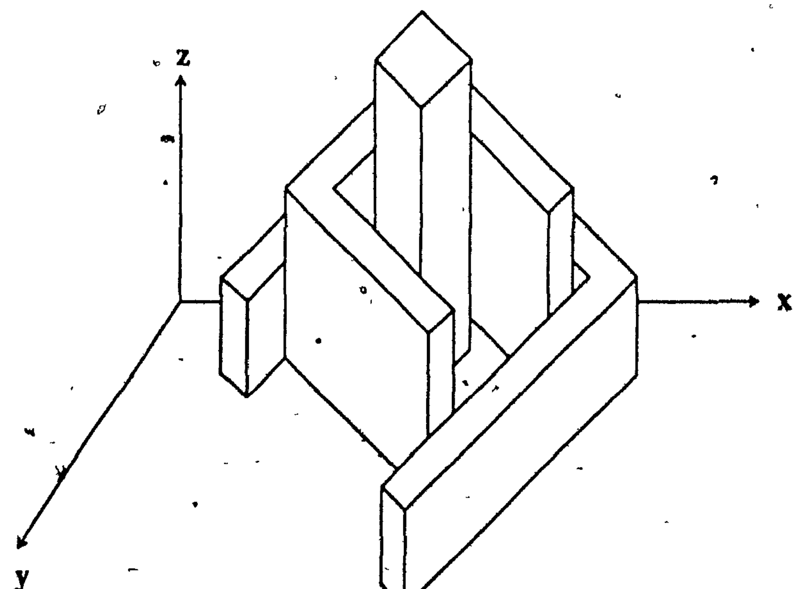

figure 4.2

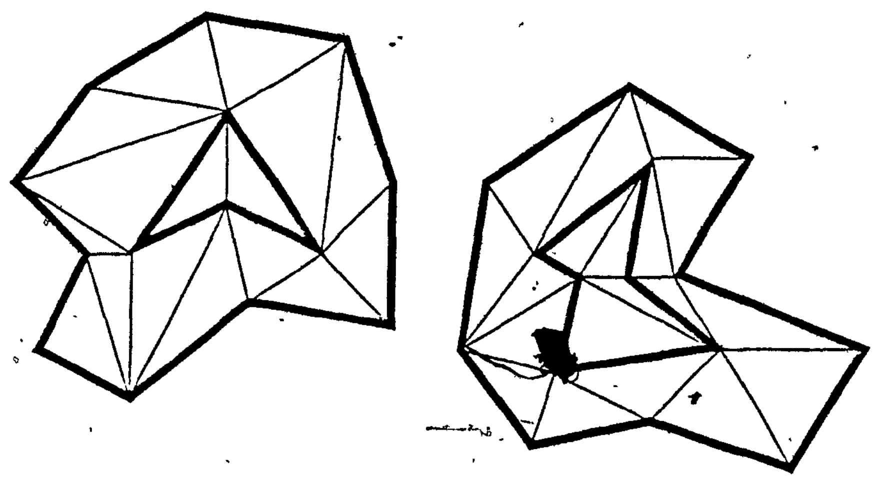

figure 4.3

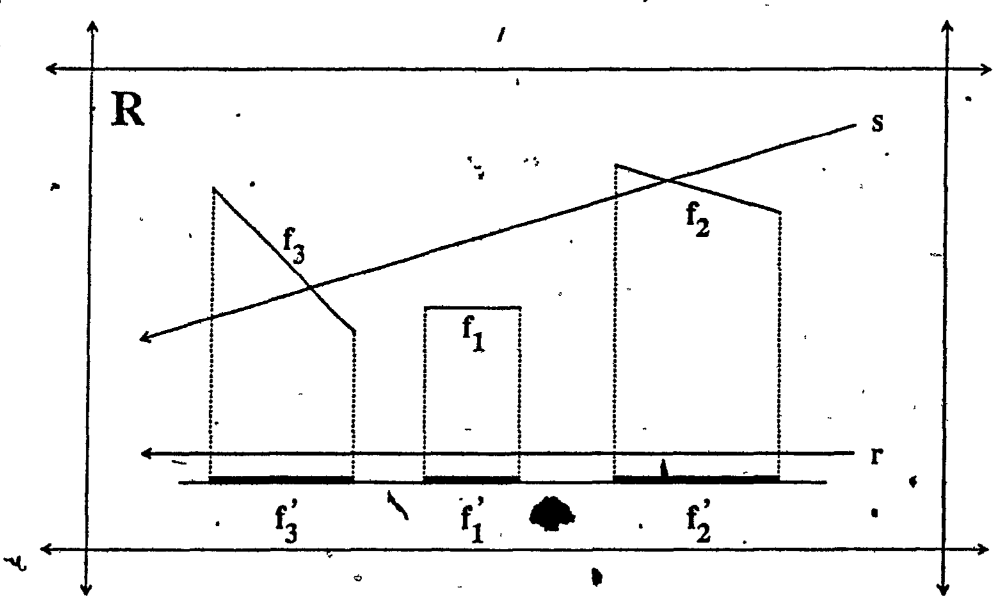

figure 4.4

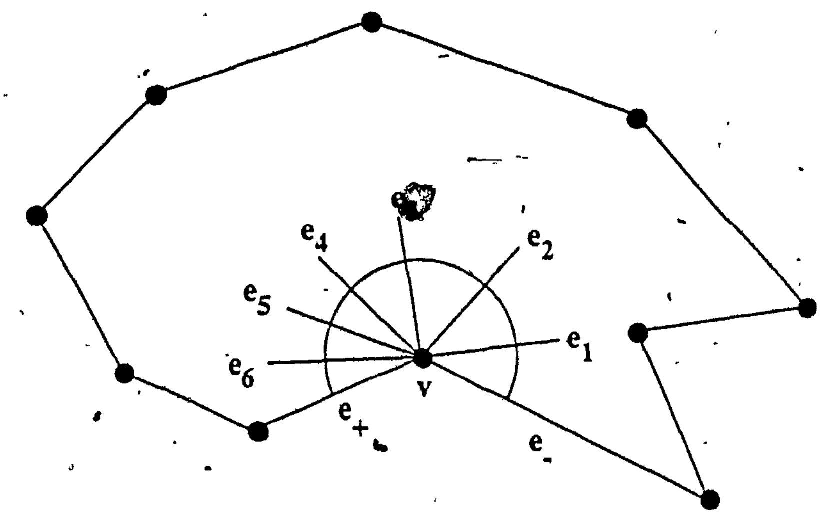

figure 4.5

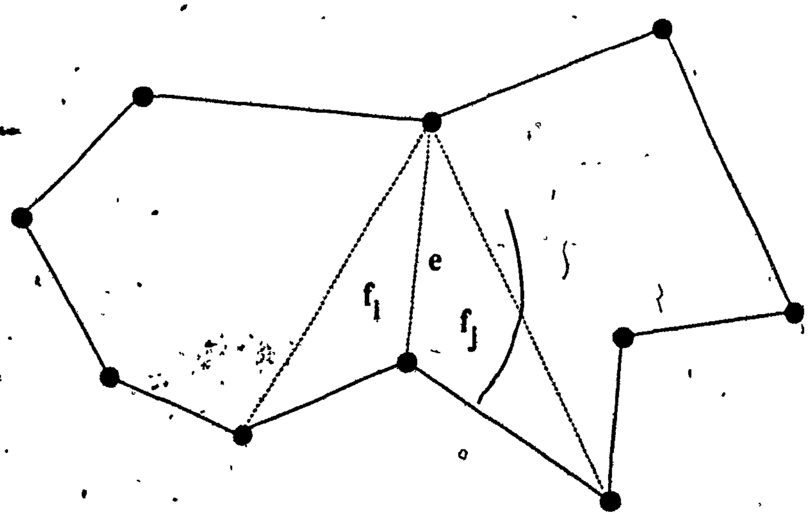

figure 4.6

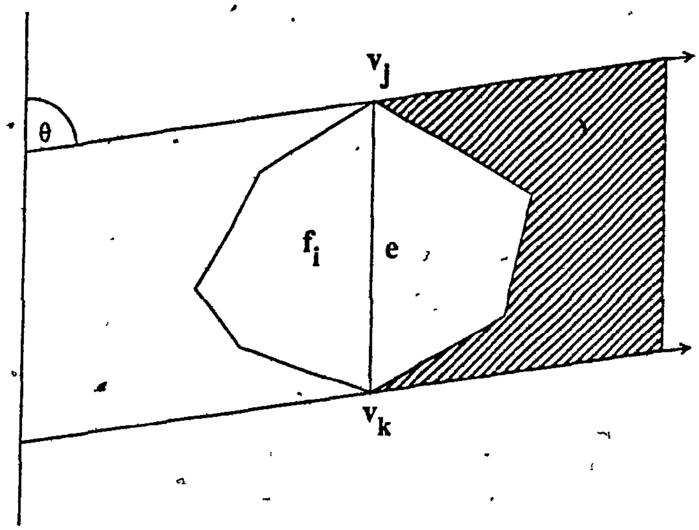

figure 4.7

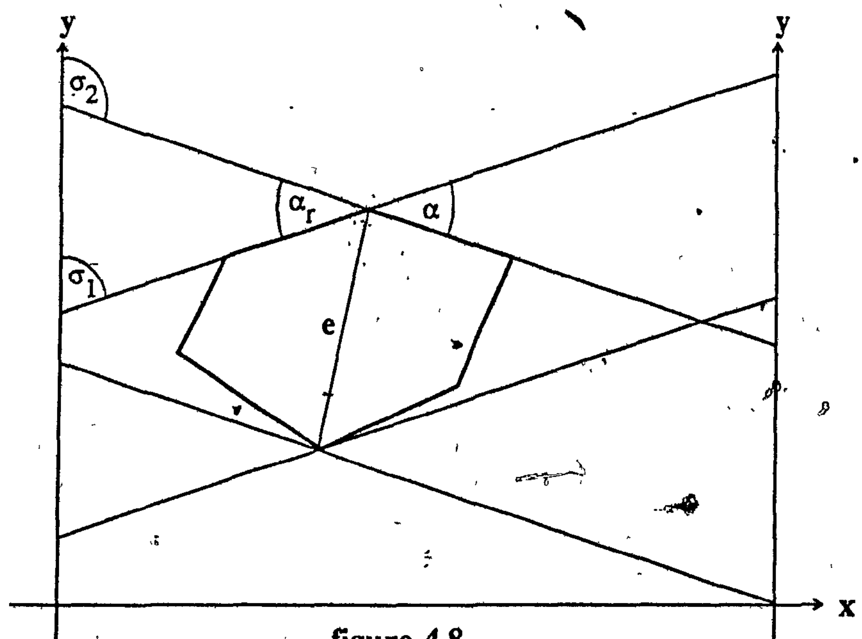

figure 4.8

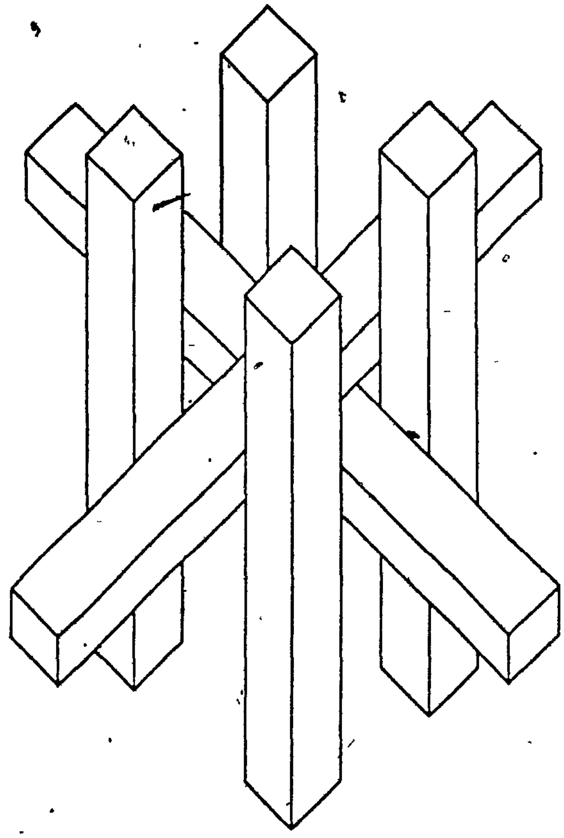

figure 4.9

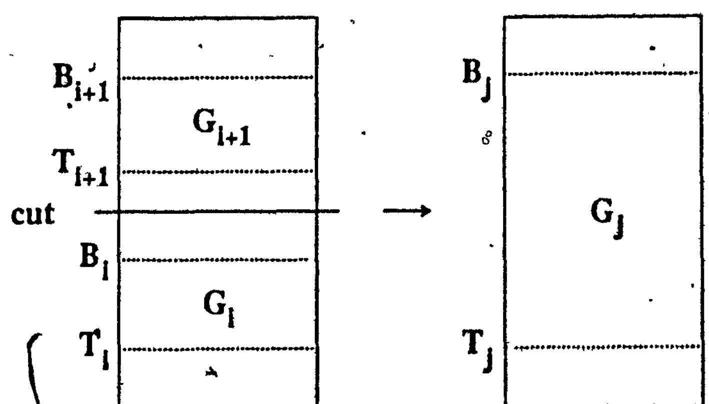

figure 4.10

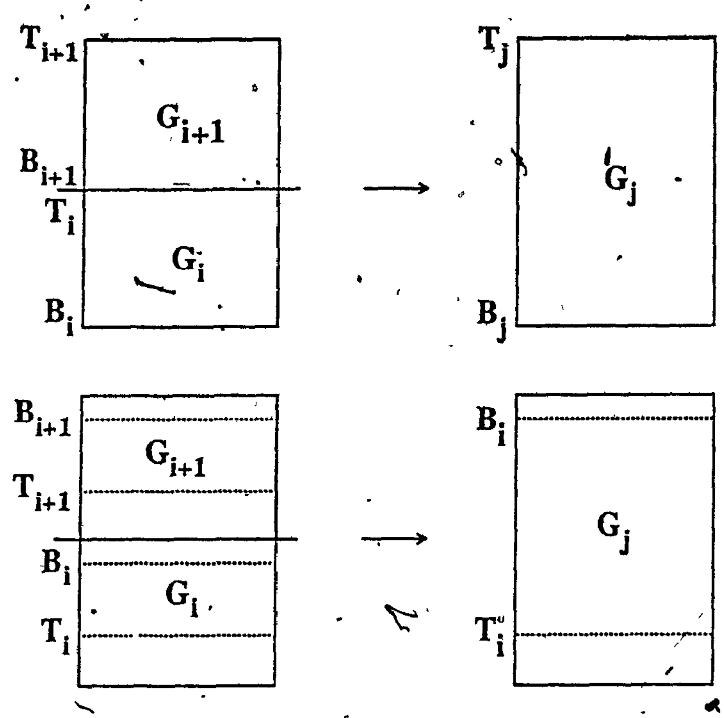

figure 4.11

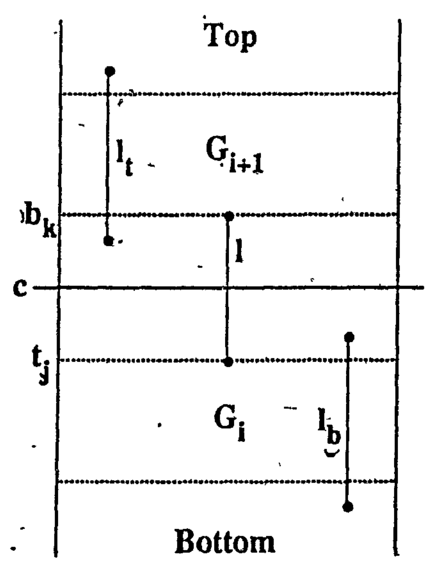

figure 4.12

Chapter 5

Dynamic Priority Orderings

Up until now we have only considered static scenes. In this chapter we examine the fundamental problem encountered when objects are allowed to be inserted into and deleted from a scene. The problem involves updating a priority ordering in order to reflect the insertion or deletion of a face. Consider a set F of faces (edges), a view-interval ω, and let $F_\omega = (f_1, f_2, \ldots, f_n)$ denote the faces of ω. As usual, we assume the view-interval $\omega = [\rho_1, \rho_2]$ has been rotated so that $\omega = [0, \rho]$. Suppose we add an extra face f_{max}, which left-dominates all other faces, including any that will be inserted. As shown in section 3.2, the *ileftdom* relation can be represented by a tree T which is rooted by f_{max}. In T the children of a node f are ordered from left to right by the value of the x-coordinate of their tail. We know from theorem 3.2 that the left to right postorder traversal of T yields a priority ordering on F_ω. Maintaining a correct priority ordering through a series of insertions and deletions will amount to updating T in order to reflect the changes in the *ileftdom* relation.

In the first section of this chapter an appropriate data structure and search technique are introduced. The insertion problem is considered in the second section, and in the last section, the deletion problem is investigated.

5.1. The Data Structure

In order to represent a tree T, an appropriate data structure is required. For our purposes the *leftmost-child, right-sibling* representation [39] is adequate. The main component of this representation is the *edge-node*. Each edge-node consists of four fields whose names, which are *first-child, next-sibling, previous-sibling*, and *parent*, describe their function. The main reason for using this representation is that for a given edge-node f, the edge-node which immediately left-dominates f, and those immediately left-dominated by f, can be quickly determined. Also, inserting into and deleting from T are simple operations. Finally, postorder and preorder traversals of T, which are crucial in the maintenance of priority orderings, can be performed in $O(n)$ time.

In some applications a large database is constructed before any general insertions or deletions are processed. In these cases it will often be beneficial, due to the time complexity of a single insertion, to construct T directly rather than considering the construction as a series of insertions. Consider the algorithm proposed in theorem 3.2. Let f be a face and let $f_1, f_2, .., f_k$ be the faces, ordered from left to right, immediately left-dominated by f. Since the algorithm encounters these faces in the order $f, f_1, f_2, ..., f_k$, we can, provided we store for each face its last child detected, use the algorithm to construct T in $O(nlogn)$ time using $O(n)$ space.

When a face is inserted or deleted it is necessary to reconfigure T in order to reflect the changes in the *ileftdom* relation. To do this quickly, T must be systematically traversed so that any changes in the *ileftdom* relation can be reported in some orderly manner. For this purpose, we introduce a search of the space containing F_ω, which corresponds to a combination preorder-postorder traversal of T. Suppose the subtrees of T, ordered from left to right, are $T_1, T_2, ..., T_r$. Consider the following recursive definition of the left to right *prepostorder* traversal of T: list the root of T, followed by the prepostorder listings of $T_1, T_2, .., T_r$, all followed by the root of T. Each node of T then is visited twice, once before its descendants, and once after. Such a traversal can be completed in $O(n)$ time.

Let f_i be a face of F_ω and let L_i denote the path in T from the root to f_i. As described in section 3.2, L_i induces a partition of the faces in F_ω. As well, C_i, the line representing the partition, which we shall call a *chain*, is either piecewise linear and descends from left to right, or vertical. Referring to figure 5.1, let C_t' denote the chain which results when f_i and C_t are combined. A chain is said to be *monotone* with respect to a direction θ, if when traversed, it yields a monotonically increasing projection onto a line in the direction θ. Clearly, C_t and C_i' are monotone with respect to the x-direction. Suppose we wish to determine which face of F_ω immediately left-dominates some face f with tail v_t. To solve the problem we modify the prepostorder traversal so that at every step it is determined whether a particular interval of a face lies directly above v_t. Let f be any face of F_ω, and let f_p and $f_1, f_2, ..., f_k$ respectively denote, provided they exist, the parent and children of f. Referring to figure 5.2, we now modify the prepostorder traversal of T as follows: when f is first encountered, consider the interval of f_p left of v_t; during the second encounter, consider the interval of f right of v_{t_k}. The two special cases must also be examined: if $f = f_{\max}$, then no interval is considered during the first encounter; if f is a leaf, then all of f is considered during the second encounter. To summarize, the interval(s) of f left of v_{t_k} are examined when $f_1, f_2, ..., f_k$ are first encountered, and the remainder of f is examined when f is encountered for the second time.

Lemma 5.1. The first face discovered during the modified prepostorder traversal of T that lies directly above v_t, immediately left-dominates f.

Proof: Clearly, all portions of all faces are considered and so some solution will be found. Suppose the algorithm stopped when f_t was encountered, however the correct solution f_x, was not reported. Referring to figure 5.3, the algorithm will have reported either f_j, the parent of f_i, or f_t itself, depending on whether it was the first or second encounter of f_t. If f_j was reported, then v_{t_x} lies left of C_t, otherwise, v_{t_x} lies left of C_t'. Whichever the case may be, denote the chain by C. Now, C and C_x do not cross, and, each is monotone with respect to the x-direction. Therefore, C_x lies left of C and so the appropriate interval of f_x will already have been considered. We thus have a contradiction. Q.E.D.

In the following sections, we consider the insertion and deletion problems in priority orderings. At the heart of the algorithms that are proposed, is the modified prepostorder traversal described above.

5.2. The Insertion Problem

Consider the following problem: given a tree T representing the *ileftdom* relation on a set $F_\omega = (f_1, f_2, ..., f_n)$ of faces, insert a new face f into F_ω and update T in order to reflect the changes in the *ileftdom* relation. To realize the changes, we must determine the face f_p that immediately left-dominates f, and the faces $f_1, f_2, ..., f_k$, ordered from left to right, immediately left-dominated by f.

As proved in lemma 5.1, the modified prepostorder traversal of T will compute f_p. As well, the traversal examines the intervals of f_p from left to right, and so the position of f amongst the children of f_p can easily be determined.

All that remains then is to calculate $f_1, f_2, ..., f_k$, preferably in their natural order. Once found, removing $f_1, f_2, ..., f_k$ from their old positions in T is a simple matter. As well, note that the subtrees which they root do not change. Suppose the subtrees of a tree T, ordered from left to right, are $T_1, T_2, ..., T_r$. Consider the following recursive definition of the left to right *preorder* traversal of T: list the root of T followed by the preorder listings of $T_1, T_2, ..., T_r$. Thus, if the children of a node h, ordered from left to right, are $h_1, h_2, .., h_s$ then in the preorder listing of T the nodes $h, h_1, h_2, ..., h_s$ appear in the given order. Referring to figure 5.4, determining which faces are immediately left-dominated by f is equivalent to determining which of the relevant vertical sections of the chains are cut by f. Let f be any face of F_ω and let f_p be the face that immediately left-dominates f. Suppose we modify the preorder traversal of T so that when f is encountered, we determine, referring to figure 5.5, if the vertical interval of C between v_i and f_p is cut by f. For the special case in which $f = f_{max}$, no interval is examined.

Lemma 5.2. The faces $f_1, f_2, ..., f_k$, those immediately left-dominated by f, are discovered in order during the modified prepostorder traversal of T.

Proof: Clearly, all the relevant vertical intervals are considered, and so $f_1, f_2, ..., f_k$ will be found. We need to show then that if $x_{t_i} < x_{t_j}$, then f_i is found before f_j. Since f does not intersect any faces of F_ω, and also since each chain is monotone with respect to the x-direction, f may intersect a given chain at most once. Referring to figure 5.6, $x_{t_i} < x_{t_j}$ and so f cuts C_i left of C_j with the result that f_i will have been considered before f_j. Q.E.D.

Since the face $f_1, f_2, ..., f_k$ are found in order, they can be inserted as the children of f as they are found. Once the traversal is completed, f can then be inserted into its proper position amongst the children of f_p.

Theorem 5.1. The priority ordering on the faces of F_ω can be maintained at a cost of $O(n)$ time per insertion.

Proof: The cost of updating T is dominated by the time required to execute the modified prepostorder and preorder traversals on T, each of which requires $O(n)$ time. Since determining the resulting priority ordering amounts to computing the postorder traversal of T, which itself requires $O(n)$ time, the priority ordering can be maintained at a cost of $O(n)$ time per insertion Note that since the face to be inserted may immediately left-dominate $O(n)$ faces, any method which explicitly maintains the *ileftdom* relation will require $O(n)$ time in the worst case. Q.E.D.

5.3. The Deletion Problem

Consider the following problem: given a tree T, representing the *ileftdom* relation on a set $F_\omega = (f_1, f_2, ..., f_n)$ of faces, delete a face f from F_ω and update T in order to reflect the changes in *ileftdom* relation. Suppose the faces immediately left-dominated by f, ordered from left to right, are $f_1, f_2, ..., f_k$. To update T requires that we determine for each f_i, $1 \le i \le k$, f_{p_i} the face which immediately left-dominates f_i when f is deleted.

Removing f from T is a simple matter. As well, note that the subtrees rooted by $f_1, f_2, ..., f_k$ will remain intact, and so can be removed before we search for $f_{p_1}, f_{p_2}, ..., f_{p_k}$. Given f_i, $1 \le i \le k$, we know, from lemma 5.1, that the modified prepostorder traversal of T can be used to determine f_{p_i}. Suppose that in the traversal f_{p_i} would be found before f_{p_j} if $x_{t_i} < x_{t_j}$. Then a single traversal is sufficient to compute $f_{p_1}, f_{p_2}, ..., f_{p_k}$.

Lemma 5.3. The faces $f_{p_1}, f_{p_2}, ..., f_{p_k}$, those immediately left-dominated by $f_1, f_2, ..., f_k$, are found in order during the modified prepostorder traversal of T.

Proof: We need to show that f_{p_i} is found before f_{p_j} if $x_{t_i} < x_{t_j}$. Extend a vertical half line upwards from each of x_{t_i} and x_{t_j}, denoting them by l_i and l_j respectively. Since each chain is monotone with respect to the x-direction, each of l_i and l_j may cross a given chain at most once. Clearly, if $C_{p_i} \subseteq C_{p_j}$, then since l_i lies left of l_j, f_{p_i} will have been considered before f_{p_j}. Otherwise, referring to figure 5.7, since no pair of chains can cross, and also since l_i lies left of l_j, C_{p_i} lies left of C_{p_j} and so the same result holds. Q.E.D.

During the traversal, the intervals of f_{p_i}, $1 \le i \le k$, are considered in order from left to right, and so the position of f_i amongst the children of f_{p_i}, can be easily determined.

Theorem 5.2. The priority ordering on the faces of F_ω can be maintained at a cost of $O(n)$ time per deletion.

Proof: The cost of updating T is dominated by the time required to execute, at a cost of $O(n)$ time, the modified prepostorder traversal on T. Since determining the resulting priority ordering demands only a postorder traversal of T, which also requires $O(n)$ time, the priority ordering can be maintained at a cost of $O(n)$ time per deletion. Note that since the face to be deleted may immediately left-dominate $O(n)$ faces, any method which explicitly maintains the *ileftdom* relation will require $O(n)$ time in the worst case. Q.E.D.

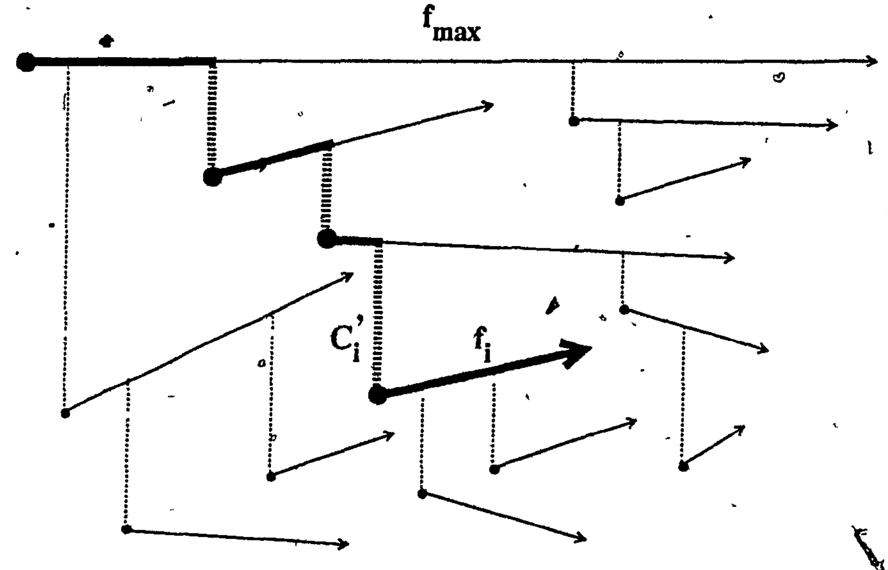

figure 5.1

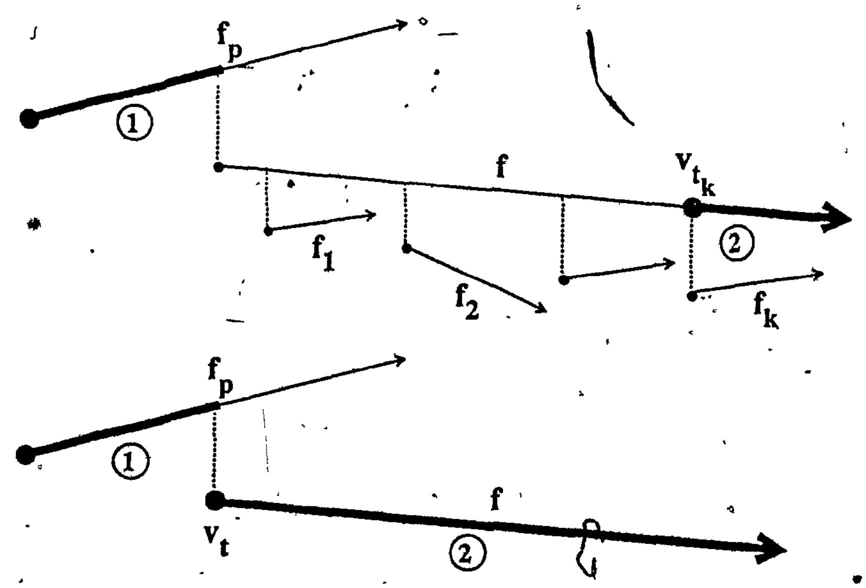

figure 5.2

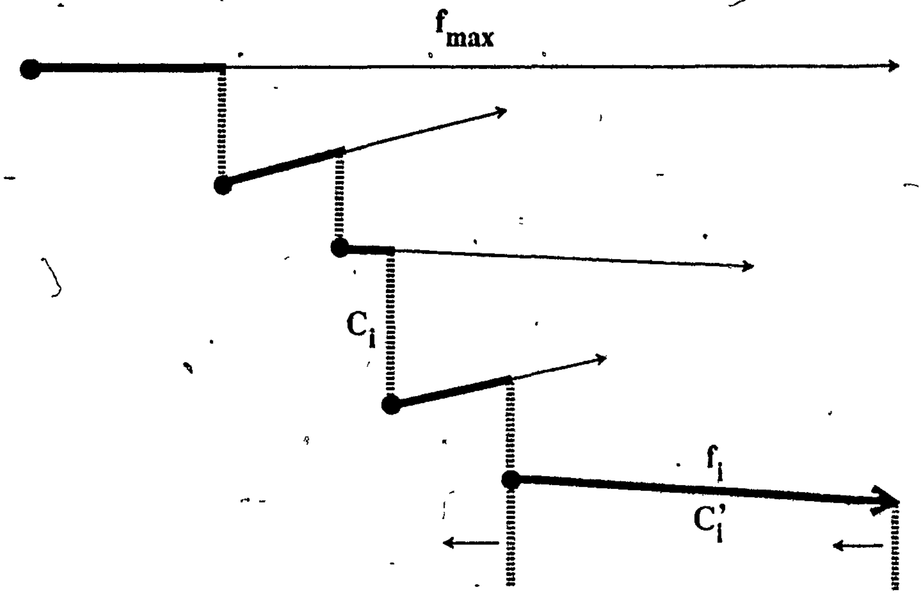

figure 5.3

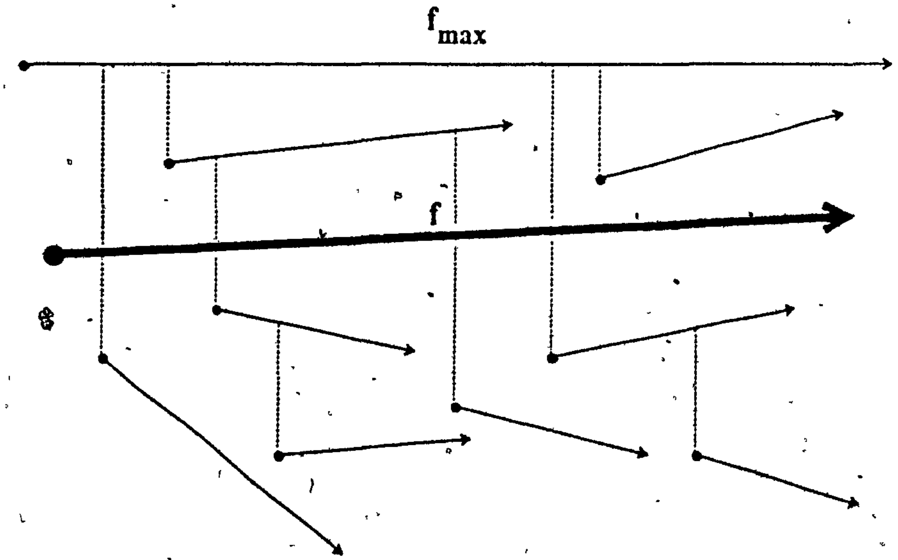

figure 5.4

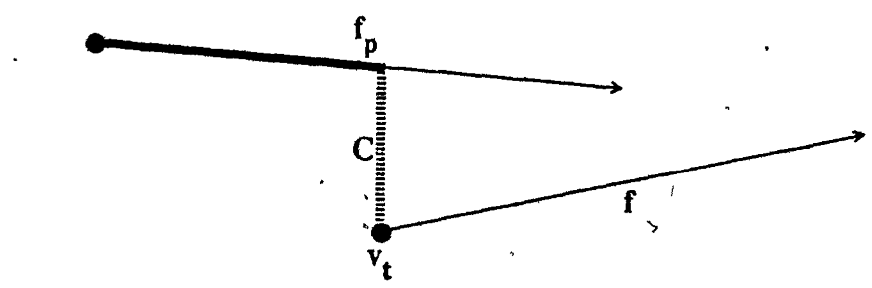

figure 5.5

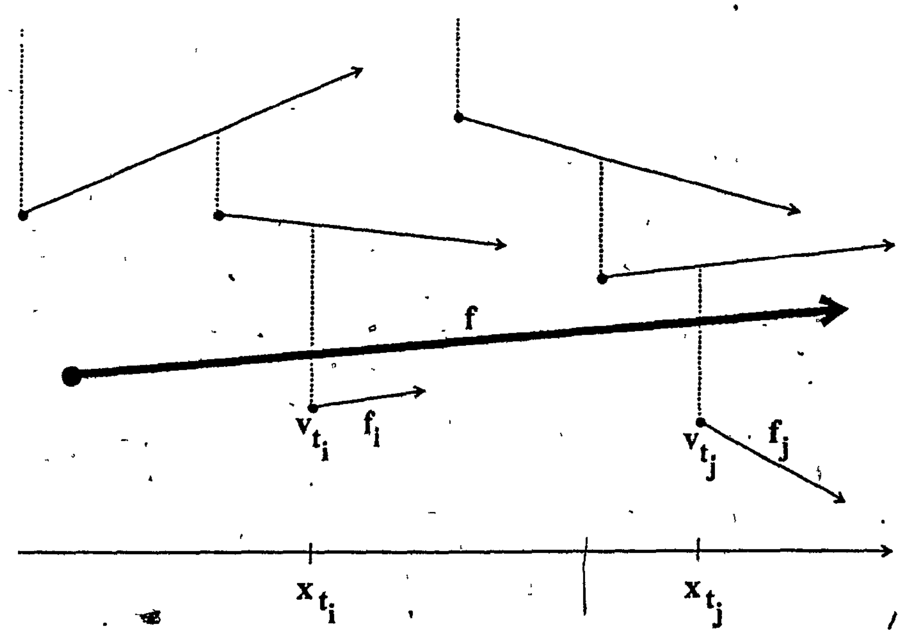

figure 5.6

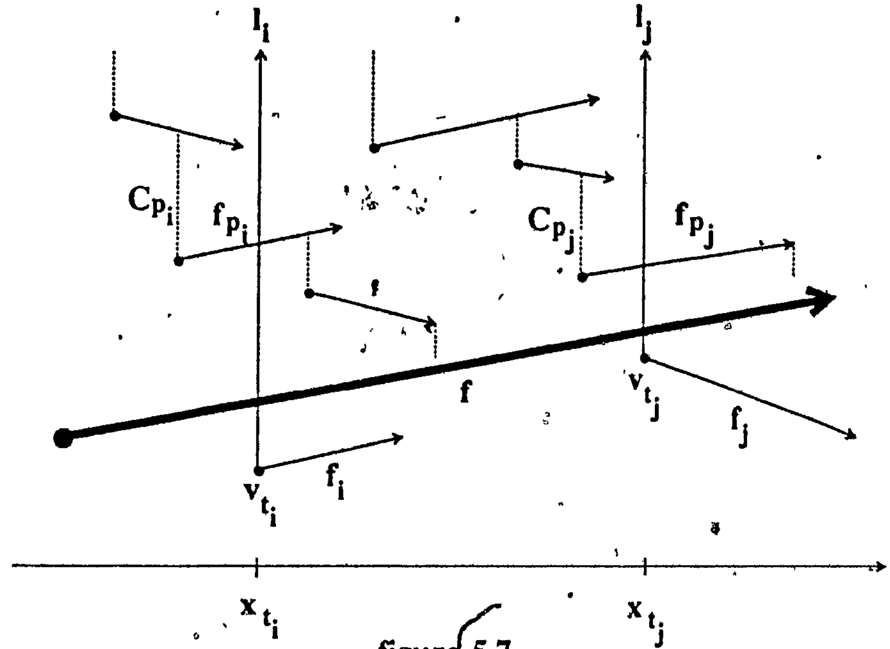

figure 5.7

Chapter 6

Conclusion

Several new results pertaining to the priority approach to hidden-surface removal have been introduced. In particular, a new formalism, in which priority orderings are described as trees, has been proposed. As well, efficient algorithms have been presented for solving the hidden-surface problem in various restricted classes of polyhedra. Note that with only minor modifications, the algorithms presented could be adapted to include the degeneration of a minor base-face to an edge or a vertex. Finally, the maintenance of a priority ordering in a dynamic environment has been investigated, and efficient algorithms for the problem have been introduced.

Future research includes the development of algorithms for more complex polyhedra. With respect to this thesis, several areas could be investigated. We have considered decomposing a scene in order to avoid potential problem areas. A better approach would eliminate only actual cyclic constraints. Another consideration when decomposing, is minimizing the number of faces cut as opposed to simply minimizing the number of cuts. Lastly, of interest is whether within some framework different from that presented, there exists sublinear algorithms for the insertion and deletion problems.

References

[1] I.E. Sutherland, R.F. Sproull and R.A. Schumaker, *A characterization of ten hidden-surface algorithms*, Computing Surveys 6 (1974), 1-55.

[2] J.G. Griffiths, *Bibliography of hidden-line and hidden-surface algorithms*, Computer Aided Design 10, No. 3 (1978), 203-206.

[3] W. Newman and R. Sproull, *Principles of Interactive Computer Graphics*, McGraw-Hill, New York, 1979.

[4] J.D. Foley and A. Van Dam, *Fundamentals of Interactive Computer Graphics*, Addison-Wesley, Reading, 1982.

[5] H. El Gindy and D. Avis, *A linear algorithm for computing the visibility polygon from a point*, Journal of Algorithms 2 (1981), 186-197.

[6] D.T. Lee, *Visibility of a simple polygon*, Computer Vision, Graphics, and Image Processing 22 (1983), 201-221.

[7] T. Asano, *Efficient algorithms for finding the visibility polygon from a polygonal region with holes*, Tech. Rept., Osaka Electro-Communication University, 1985.

[8] D. Rappaport and G.T. Toussaint, *A simple linear hidden-line algorithm for star-shaped polygons*, Tech. Rept. TR-SOCS-83.23, McGill University, 1983.

[9] D. Avis and G.T. Toussaint, *An optimal algorithm for determining the visibility of a polygon from an edge*, IEEE Transactions on Computers C-30, No. 12 (1981), 910-914.

[10] D.T. Lee and A. Lin, *Computing the visibility polygon from an edge*, Tech. Rept., Northwestern University, 1984.

[11] H. El Gindy, *Hierarchical decomposition of polygons with applications*, Ph.D. thesis, McGill University, 1985.

[12] B. Chazelle and L.J. Guibas, *Visibility and intersection problems in plane geometry*, Proc. ACM Symposium on Computational Geometry (1985), 135-146.

[13] G.T. Toussaint, *An O(nloglogn) algorithm for solving the strong hidden-line problem in a simple polygon*, Pattern Recognition Letters 4, No. 6 (1986), 449-451.

[14] H. Edelsbrunner, M.H. Overmars and D. Wood, *Graphics in flatland*, in Advances in Computing Research 1 (1983), F.P. Preparata, ed., 35-49.

[15] A. Schmitt, *Time and space bounds for hidden-line and hidden-surface algorithms*, Proc. Eurographics'81 (1981), 43-56.

[16] F. Devai, *Quadratic bounds for hidden-line elimination*, Proc. ACM Symposium on Computational Geometry (1986), 269-275.

[17] T. Ottmann, P Widmayer and D. Wood, *A worst-case efficient algorithm for hidden-line elimination*, International Journal of Computer Mathematics 18 (1985), 93-119.

[18] O. Nurmi, *A fast line-sweep algorithm for hidden-line elimination*, BIT 25 (1985), 466-472.

[19] D. Rappaport, *Eliminating hidden lines from monotone slabs*, Proc. 20th Allerton Conference on Communication, Control, and Computing, (1982), 43-52.

[20] R.H. Guting and T. Ottmann, *New algorithms for special cases of the hidden-line elimination problem*, Tech. Rept. 184, University of Dortmund, 1984.

[21] M. McKenna, *Worst-case optimal hidden-surface removal*, Tech. Rept., The Johns Hopkins University, 1986.

[22] R.A. Schumaker, B. Brand, M Gilliland and W. Sharp, *Study for applying computer-generated images to visual simulation*, Tech. Rept. AFHRL-TR-69-14, U.S. Air Force Human Resources Lab, 1969.

[23] M.E. Newell, R.G. Newell and T.L. Sancha, *A new approach to the shaded picture problem*, Proc. ACM National Conference (1972), 443.

[24] W.R. Franklin and H.R. Lewis, *3-d graphic display of discrete spatial data by prism maps*, Proc. ACM Siggraph'78 (1978), 70-75.

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

[26] H. Hubschman and S.W. Zucker, *Frame-to-frame coherence and the hidden-surface computation*, Computer Graphics **15** (1981), 45-54.

[27] F.F. Yao, *On the priority approach to hidden-surface algorithms*, Proc. 21st Annual IEEE Symposium on the Foundations of Computer Science (1980), 301-307.

[28] T. Ottmann and P. Widmayer, *On translating a set of line segments*, Computer Vision, Graphics, and Image Processing 24 (1983), 382-389.

[29] J. Bentley and T. Ottmann, *Algorithms for reporting and counting geometric intersections*, IEEE Transactions on Computers **C-28**, No. 9 (1979), 643-647.

[30] D.E. Muller and F.P. Preparata, *Finding the intersection of two convex polyhedra*, Theoretical Computer Science **7** (1978), 217-236.

[31] B. Chazelle, *A theorem on polygon cutting with applications*, Proc. 23rd Annual IEEE Symposium on Foundations of Computer Science (1982), 339-349.

[32] B. Chazelle and J. Incerpi, *Triangulating a polygon by divide-and-conquer*, Proc. 21st Annual Allerton Conference on Communication, Control, and Computing (1983), 447-

456.

[33] M.R. Garey, D.S. Johnson, F.P. Preparata and R.E. Tarjan, *Triangulating a simple polygon*, Information Processing Letters 7 (1978), 175-179.

[34] S. Hertel and K. Mehlhorn, *Fast triangulation of simple polygons*, Proc. Conference on Foundations of Computing Theory (1983), 207-218.

[35] R.E. Tarjan and C.J. Van Wyk, *An O(nloglogn) algorithm for triangulating a simple polygon*, SIAM Journal on Computing, to appear.

[36] L.J. Guibas and F.F Yao, *On translating a set of rectangles*, Proc. 12th Annual ACM Symposium on Theory of Computing (1980), 154-157.

[37] M.I. Shamos, *Computational Geometry*, Ph.D thesis, Yale University, 1977.

[38] M.I. Shamos and D. Hoey, *Geometric intersection problems*, Proc. 17th Annual IEEE Symposium on Foundations of Computer Science (1976), 208-215.

[39] A.V Aho, J.H. Hopcroft and J.D. Ullmann, *Data Structures and Algorithms*, Addison-Wesley, Reading, 1983.