The Visual Computer

Hidden-surface removal in polyhedral cross-sections

Peter Egyed

School of Computer Science, McGill University, 805 Sherbrooke St. West, Montreal, Quebec, Canada, H3A 2K6

Many of the fundamental problems in computer graphics involve the notion of visibility. In one approach to the hidden-surface problem, priorities are assigned to the faces of a scene. A realistic image is then rendered by displaying the faces with the resulting priority ordering. We introduce a tree-based formalism for describing priority orderings that simplifies an existing algorithm. As well, a decomposition-based algorithm is presented for classes of scenes that do not in general admit priority orderings. The algorithm requires $O(n \log n)$ time if $t=1$ and $O(tn \log n + n \log n \log m)$ time if $t>1$, where n and m are respectively the number of faces and polyhedra in the scene, and t is a minimum decomposition factor of the scene. Finally, the tree-based formalism is used in the development of $O(n)$ time insertion and deletion algorithms that solve the problem of dynamically maintaining a priority ordering.

Key words: Hidden-surface problems – Computational geometry – Priority orderings – Decomposition techniques – Dynamization techniques

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.

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. Only recently, spurred by developments in the new and fluorishing field of computational geometry, has the theoretical nature of the problems begun to be investigated. Many different solutions have been proposed for the general hidden-line problem (Devai 1986; Nurmi 1985; Ottmann et al. 1985; Schmitt 1981). By restricting the class of input considered, more efficient results have been obtained (Guting and Ottmann 1984; Rappaport 1986). Some theoretical results have also been presented in the area of hidden-surface removal (Schmitt 1981; McKenna 1987).

One method for solving the hidden-surface problem 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. 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 have considered various aspects of the problem, they fail to develop any significant theoretical insight into the problem. In contrast, Yao (1980) investigates the underlying mathematical nature of priority orderings, and proposes an efficient algorithm for a restricted class of input.

In this paper we extend the work of Yao (1980). In particular, a *tree-based formalism* for describing priority orderings is introduced. This formalism is used to simplify an existing algorithm due to Yao (1980). As well, a class of scenes, encompassing the class presented by Yao (1980), is introduced. Due to the possibility of cyclic constraints, a scene in this class will not in general admit a priority ordering. To remedy this situation, *decomposition tech-*

niques are used. Although finding a minimum decomposition appears difficult, a heuristic is presented that uses at most twice the minimum number of horizontal *cuts*. The resulting algorithm requires $O(n \log n)$ time if $t=1$ and $O(tn \log n) + n \log n \log m)$ time if $t > 1$, where n and m are respectively the number of faces and polyhedra in the scene, and t is the minimum number of horizontal cuts needed to decompose the scene. Finally, *dynamization techniques* are used to develop insertion and deletion algorithms for the problem of dynamically maintaining a priority ordering. These algorithms, which depend on the tree-based formalism, require $O(n)$ time.

We now briefly describe the organization of this paper. In section two, the class of scenes to be considered is defined, and some basic properties of the objects comprising the scene are deduced. The tree-based formalism and simplification of Yao's results are presented in section three. In section four, the decomposition techniques are considered, and the resulting algorithm is presented. The algorithms for dynamically maintaining a priority ordering are developed in section five. Finally, the last section concludes the paper and suggests some directions for further research.

2 Definitions and terminology

We present in this section the necessary definitions and terminology. As well, the class of scenes to be considered is introduced and some properties of these scenes are presented.

2.1 Basic definitions

We represent a *simple polygon* P by a clockwise sequence of vertices, $v_1, v_2, \ldots, v_n$, 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 ith edge and is represented by e_i. The sequence $e_1, e_2, \ldots, e_n$ 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 $\operatorname{int}(P)$, and the other unbounded, termed the *exterior* of P and denoted by $\operatorname{ext}(P)$.

2.2 Defining the scene

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 PX is a polyhedron of restricted form that is enclosed by *base-faces*, simple polygons $P_b=(v_{b1}, v_{b2}, \ldots, v_{bn_b})$ and $P_t=(v_{t1}, v_{t2}, \ldots, v_{tn_t})$ that lie in parallel planes $z=z_b$ and $z=z_t$ respectively, and also by a collection $F=(f_1, f_2, \ldots, f_k)$ of simple polygons, termed *lateral-faces*, that connect P_b and P_t. The base-faces P_b and P_t are named with the convention $z_t > z_b$, and termed the *top* and *bottom* base-face respectively. 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 and P_t are restricted so that either $P_b' \subseteq P_t'$ or $P_t' \subseteq P_b'$. Alternate symbols for the base-faces are derived from the containment relation: if $P_b'=P_t'$, then the *minor* base-face, denoted by P_m, is P_t, and the *superior* base-face, denoted by P_s, is P_b, otherwise P_m is the properly contained base-face and P_s is the other. If P_i and P_j are simple polygons, then for simplicity we shall denote $\operatorname{int}(P_i) \cap \operatorname{int}(P_j)$ 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 $\Gamma(P_{s_i}', P_{s_j}') \neq \varnothing$ and $z_{b_i} < z_{b_j}$ then $z_{t_i} \leq z_{b_j}$, i.e., if the x-y projections of PX_i and PX_j intersect, then they are separable by some z-plane. 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 $P_i \, \Delta P_j = P_i - \operatorname{int}(P_j)$. A lateral-edge links a vertex of each of P_m and P_s, is denoted by e_{jk}, and is restricted so that $e_{jk}' \in P_s' \, \Delta P_m'$. Finally, we 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.

A polyhedral cross-section PX, with lateral-faces $F=(f_1, f_2, \ldots, f_k)$ and base faces P_s and P_m, has several important properties with respect to the remainder of this paper. These properties are: (i) each lateral-face f_i is either a triangle or a convex quadrilateral; (ii) for every pair f_i, f_j of lateral-faces, $\Gamma(f_i', f_j')=\varnothing$; (iii) $F'=(f_1', f_2', \ldots, f_k')$ is a non-overlapping decomposition of $P_s' \, \Delta P_m'$. The proofs of the above properties, although easily derived, are omit-

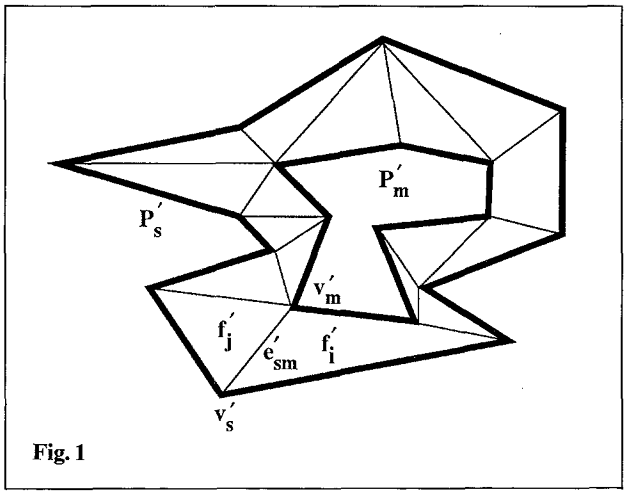

Fig. 1

ted. Some properties of the x-y projection of a polyhedral cross-sections PX are highlighted in Fig. 1.

3 Elementary scenes

In this section we consider hidden-surface removal with respect to restricted scenes in which 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. Yao (1980) considered this class of scenes and obtained results on computing priority orderings. We introduce a tree-based formalism for describing priority orderings that improves Yao's (1980) algorithm. While the worst-case complexity remains the same, a simplification of the algorithm, eliminating the need for a second pass of the data, is obtained. We have recently learned that this simplification was independently discovered by Ottmann and Widmayer (1983) within the context of line segment translation. We note however, that our method of proof, which relies on the tree-based formalism and on which another section of this paper depends, is of a completely different flavor.

3.1 Preliminary considerations

Consider a scene $S=(PX_1, PX_2, \ldots, PX_m)$ of polyhedral cross-sections, 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 $P'_{b_i}=P'_{t_i}$, 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 Fig. 2, parallel rays emanate from an observer at infinity and towards the scene. The observer's view is then completely determined by the pair of angles (θ, ϕ), $0 \leq \theta \leq 2\pi$ and $\frac{-\pi}{2} \leq \phi \leq \frac{\pi}{2}$, formed by the projections of a ray r onto the $x-y$ and $x-z$ planes respectively. We shall defer, until Sect. 4.2, the treatment of the special cases in which $\phi=\frac{\pi}{2}$ or $\phi=\frac{-\pi}{2}$. We therefore assume that $\frac{-\pi}{2} < \phi < \frac{\pi}{2}$.

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. Since each of the visible base-faces has an equal and highest priority, solving the hidden-surface problem for a scene of prisms involves computing a valid priority ordering for the visible lateral-faces.

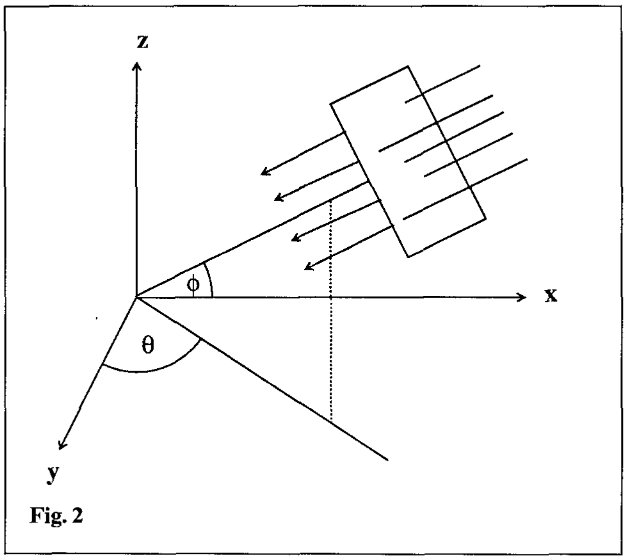

Fig. 2

Let $F=(f_1, f_2, \ldots, f_n)$ be the lateral-faces of S. Determining a priority ordering for the faces of F in the direction (θ, ϕ) is equivalent to determining, in two dimensions, a valid priority ordering for the visible edges of $F'=(f'_1, f'_2, \ldots, f'_n)$ 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.

3.2 Simplifying Yao's results with a tree structure

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_i 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 Fig. 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 $v_{t_i}=(x_{t_i}, y_{t_i})$

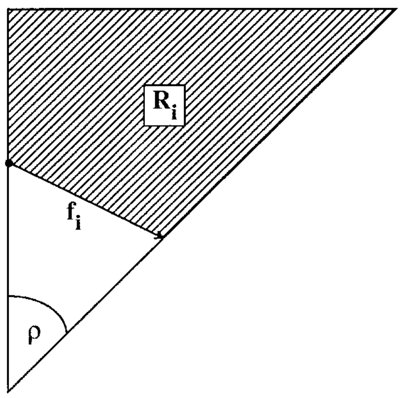

Fig. 3

and the head is denoted by $v_{h_i}=(x_{h_i}, y_{h_i})$. 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 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 1980).

The proof relies on the fact that, of the faces of F_ω that are maximal with respect to *left-dom*, the one whose tail has the largest x-coordinate, is not dominated by any other face.
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. Suppose we add a face $f_{\max}$ that left-dominates all other faces. The only face now maximal with respect to *ileftdom* is $f_{\max}$, and so 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 $T_1, T_2, \ldots, T_r$. Consider the following recursive definition of the *postorder* traversal of T: list the nodes of $T_1, T_2, \ldots, T_r$ in postorder all followed by the root of T. Thus, if the children of a node h, ordered from left to right, are $h_1, h_2, \ldots, h_s$, then in the postorder listing of T the nodes $h_1, h_2, \ldots, h_s, h$ appear in the given order.

Theorem 2. *The postorder traversal of the tree T yields a priority ordering on F_ω.*

Proof. Let f be a face of F_ω, then referring to Fig. 4, 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 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 Fig. 5, 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 monotone with respect to the x-axis, or 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

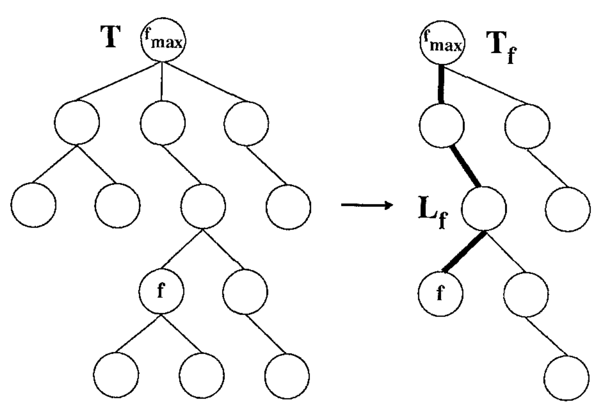

Fig. 4

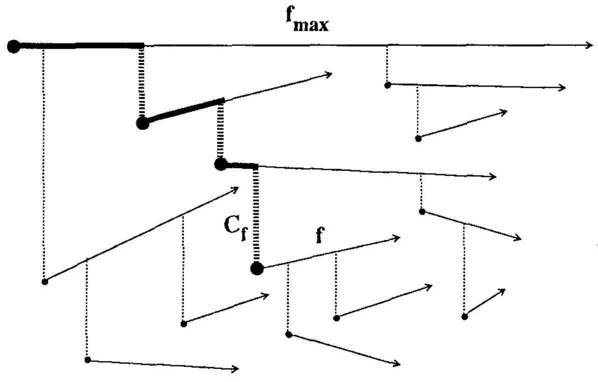

Fig. 5

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.

Theorem 3. *The postorder listing of the tree T can be optimally computed in $O(n \log n)$ time.*

Proof. Suppose the faces are processed so that a face f and those faces immediately left-dominated by f, $f_1, f_2, \ldots, f_k$, ordered by the x-coordinate of their tail, are encountered in the order f, f_1, $f_2, \ldots, f_k$. 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 $f_1, f_2, \ldots, f_k, f$. The problem is solved with a plane sweep technique identical to that described by Yao (1980). Consider a vertical line l through v_t, the tail of a face f, and its intersection with the elements of F_ω. The order of the intersections of the faces with l, as l is swept from left to right, can 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 $f_1, f_2, \ldots, f_k$, those faces immediately left-dominated by f, are encountered in the order f, f_1, $f_2, \ldots, f_k$ as desired. The optimality of the algorithm follows simply since, as noted by Yao (1980), sorting is linear time transferable to the priority ordering problem.

A scene is said to be *k-regular* if the number of view intervals is k. In general $k=n$, however, there exists scenes in which $k \ll n$. The k priority orderings, which are sufficient for all views, and the corresponding k lists can be calculated in $O(k\,n \log n)$ time. As well, $O(n)$ display commands are required to render an image.

4 Complex scenes

In this section we examine a more general class of scenes, in which the placement of base-faces is not so rigidly confined. These scenes, do not in general admit priority orderings. To remedy this situation, various decomposition techniques are introduced.

4.1 Nonoverlapping scenes

Consider a scene $S=(PX_1, PX_2, \ldots, PX_m)$ of polyhedral cross-sections and let $F=(f_1, f_2, \ldots, f_n)$ 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*.

Unlike in a scene composed of prisms, for a fixed viewing position (θ, ϕ), the visible base-faces of a scene constructed from columns, do not necessarily have equal and highest priority. Referring to Fig. 6, it is simple to construct a scene of columns in which for any viewing position, there exists a base-face, lateral-face pair, of which 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.

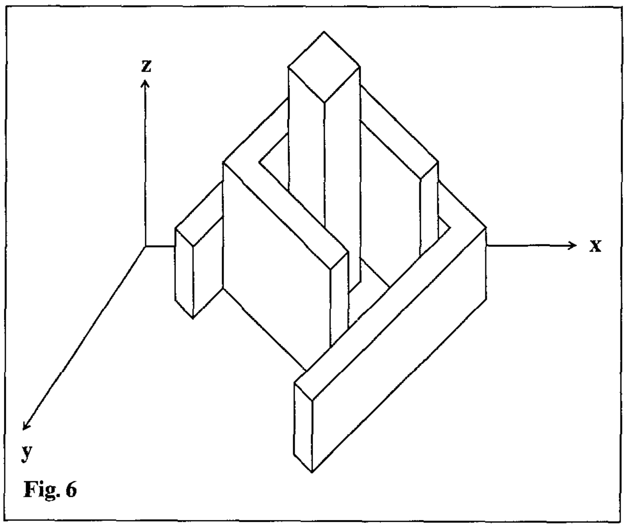

Fig. 6

It is of course desirable to render the problem independent of ϕ. With this in mind, 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 of S is triangulated. Euler showed that a planar graph on n vertices has $O(n)$ edges and faces. Consequently, the decomposition of the minor base-faces yields $O(n)$ *triangular-faces* and induces a vertical decomposition of S. Redefine $F=(f_1, f_2, \ldots, f_n)$ to include both the lateral-faces and triangular-faces of S. As well, define $\Psi(F, r)$ to be the partial ordering of the faces of F induced by their order of intersection with a ray r.

Lemma 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 Fig. 7, 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.

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. As well, the representation must support fast insertion of the triangular-faces. To satisy these conditions, a planar graph structure, such as the doubly-connected-edge-list of Muller and Preparata (1978), can be used.

Lemma 2. *The set $M = (P_{m_1}, P_{m_2}, \ldots, P_{m_m})$ of minor base-faces can be triangulated in $O(n \log n)$ time.*

Proof. Many algorithms (Chazelle 1982; Chazelle and Incerpi 1983; Garey et al. 1978; Hertel and Mehlhorn 1983) exist for triangulating a simple polygon in $O(n \log n)$ time. Each minor base-face is a simple polygon. Since there are $O(n)$ vertices

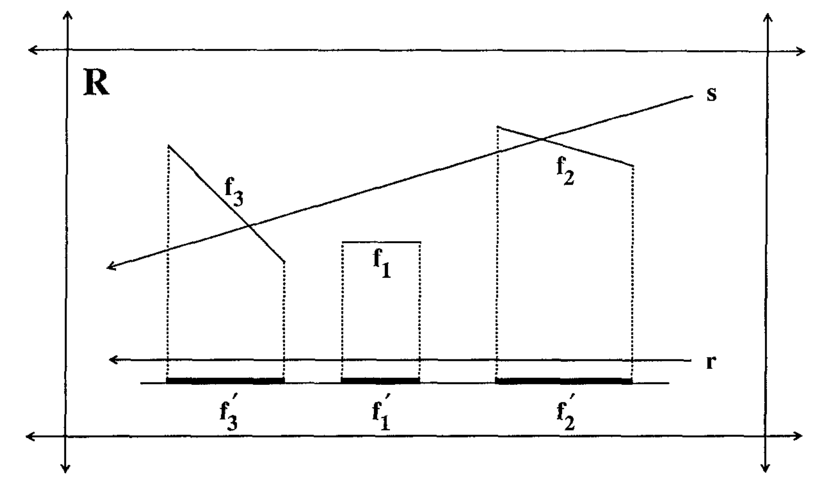

Fig. 7

determining the m minor base-faces, the m minor base-faces can be triangulated in $O(n \log n)$ time.

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

Proof. 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 (1980), 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.

Lemma 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 (1977) showed, for a convex polygon P on n vertices, that the $O(n)$ antipodal pairs of P can be computed in $O(n)$ time. In addition, 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 total of $O(n)$ vertices.

For a scene S, there are then $O(n)$ edges and shadow-edges. Associated with each edge e are two non-overlapping 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 $E = (e_1, e_2, \ldots, e_n)$ 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. *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 $O(n \log n)$ time.*

Proof. The proof follows directly from Lemmas 1–4 and Theorem 3.

Given a *k-regular* scene composed of columns, the corresponding k priority orderings can be determined in $O(k\, n \log n)$ time. 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.

4.2 General scenes

We now consider the most general class of scenes. Let $S = (PX_1, PX_2, \ldots, PX_m)$ 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 $z_{t_i} \leq z_{b_j}$. Yao (1980) showed that for such a class, 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 $\phi = \frac{\pi}{2}$. If the top base-faces are sorted and renamed so that $z_{t_1} \leq z_{t_2} \leq \ldots \leq z_{t_m}$, 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 $\phi = \frac{-\pi}{2}$.

Consider partitioning space into $t+1$ horizontal *slabs* with a series of t z-planes $z = z_1 < z = z_2 < \ldots < z = z_t$. Suppose a scene S is decomposed by such a partitioning into $t+1$ subscenes so that within each subscene $\Gamma(P'_{s_i}, P'_{s_j}) = \emptyset$ for any pair PX_i, PX_j of polyhedral cross-sections. Any ray r in a fixed direction (θ, ϕ) 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 Sect. 4.1.

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 $\Gamma(P'_{s_i}, P'_{s_j}) \neq \emptyset$ 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 $z = z_{t_1}$, $z = z_{t_2}, \ldots, z = z_{t_m}$. 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 $I_i = \{j \mid \Gamma(P'_{s_i}, P'_{s_j}) \neq \emptyset$ 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 at most $2t$ z-planes. This is followed by the conquer phase which then selects $O(t)$ of the z-planes. At the heart of the algorithm is intersection testing. By decomposing the superior base-faces as described in Sect. 4.2, we are able to make use of the intersection detection algorithm of Shamos and Hoey (1976). Given a set of n triangles and quadrilaterals, their algorithm can detect whether any pair of objects intersect in $O(n \log n)$ time. Using this algorithm, a 0-*cuttable* scene could be quickly detected.

Theorem 5. *For any scene S that is t-cuttable, a set of at most 2t z-planes that properly decompose S, can be computed in $O(n \log n \log m)$ time.*

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 $t_1 \leq t_2 \leq \ldots \leq t_m$. 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 $\Gamma(D_i, D_j) \neq \emptyset$. 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, $1 \le i \le r$, let $j=\frac{i+1}{2}$. If $i+1 \le r$ 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 $t_j \in Q_i$, $t_j \ge B_i$, $b_k \in Q_{i+1}$, $b_k \le T_{i+1}$, and $\Gamma(D_j, D_k) \neq \emptyset$. 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 $\Gamma(D_j, D_k) \neq \emptyset$. 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 Fig. 8 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 Fig. 9, 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. In the divide phase the running time is dominated

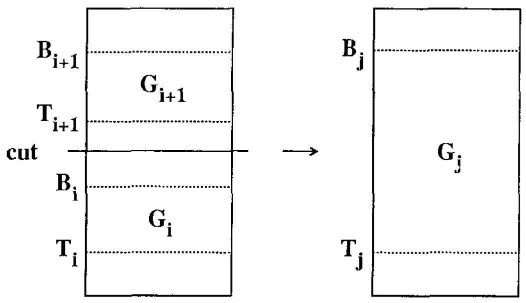

Fig. 8

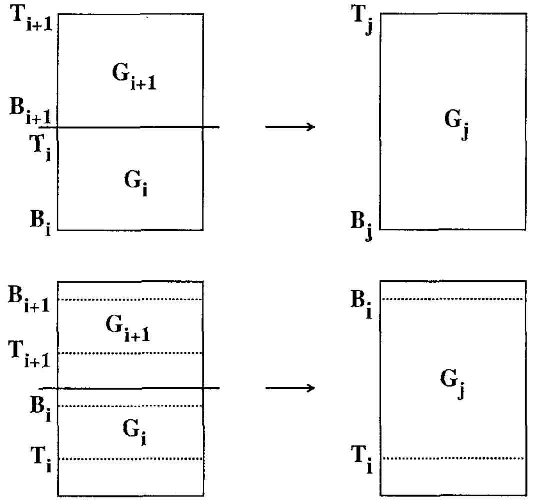

Fig. 9

by the sorting, and so $O(m \log m)$ time is required. Since at each level of the conquer phase $\left\lfloor \frac{r}{2} \right\rfloor$ merges occur, there are $O(\log m)$ 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 spent detecting intersections at each level is $O(n \log n)$. Therefore, the running time of the algorithm is $O(n \log n \log m)$.

What remains to be shown is that at most $2t$ cuts are made. Referring to Fig. 10, 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 $l=(t_j, b_k)$

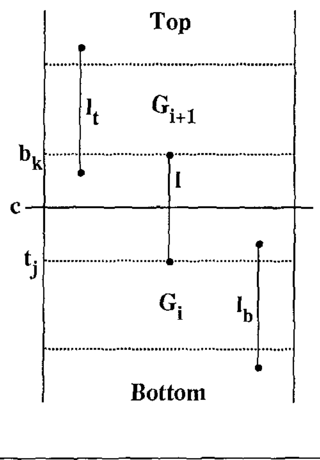

Fig. 10

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 Fig. 10, 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.

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 $C=(c_1, c_2, \ldots, c_t)$. 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 $O(tn + t\log t)$ time. Let us say a scene is *k-regular* if the maximum number of view-intervals in any slab, is k. In total, $O(tn\log n)$ time is required to determine the $O(kn)$ view-intervals. The corresponding priority orderings can be computed in $O(tkn\log n)$ time. Finally, $O(tn)$ display commands are required to render an image.

5 Dynamic priority orderings

In this section we present a dynamization technique that solves the problem of dynamically maintaining a priority ordering. 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 Sect. 3.2, the *ileftdom* relation can be represented by a tree T that is rooted by $f_{\max}$, and the 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.

5.1 A search technique

In order to represent a tree T, an appropriate data structure is required. For our purposes the *leftmost-child, right-sibling* representation (Aho et al. 1983) is adequate. Suppose we wish to construct T directly rather than consider the construction as a series of insertions. This can be done, in $O(n\log n)$ time, using the algorithm proposed in Theorem 3, provided we store for each face, its last child detected.

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. Suppose the subtrees of T, ordered from left to right, are $T_1, T_2, \ldots, T_r$. 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 $T_1, T_2, \ldots, T_r$, all followed by the root of T. Each node of T then is visited twice, once before its descendants, and once after.

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 Sect. 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 monotone with respect to the x-axis, or vertical. Referring to Fig. 11, let C_i' denote the chain which results when f_i and C_i are combined. Clearly, C_i' is monotone with respect to the x-axis. 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 $f_1, f_2, \ldots, f_k$ respectively denote, provided they exist, the parent and children of f. Referring to Fig. 12, 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

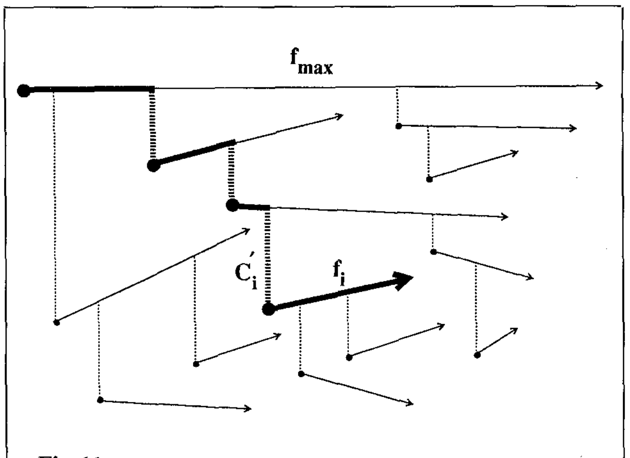
Fig. 11

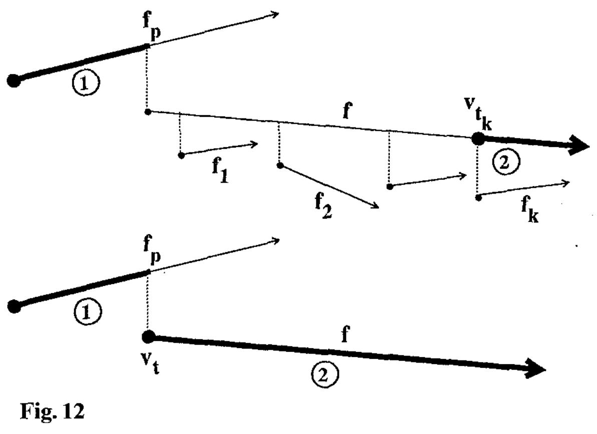
Fig. 12

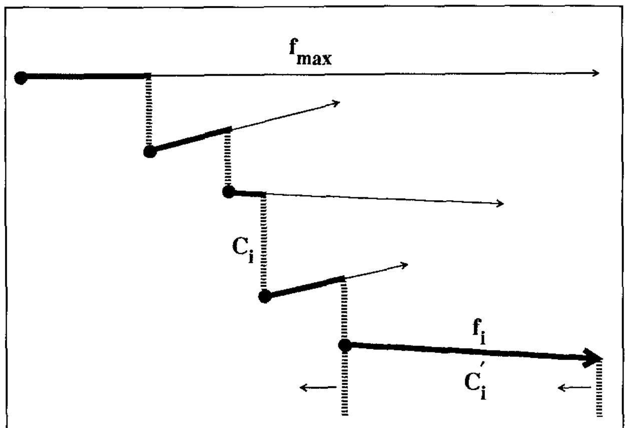

Fig. 13

examined when $f_1, f_2, \ldots, f_k$ are first encountered, and the remainder of f is examined when f is encountered for the second time.

Lemma 5. *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_i was encountered, however the correct solution f_x, was not reported. Referring to Fig. 13, the algorithm will have reported either f_j, the parent of f_i, or f_i itself, depending on whether it was the first or second encounter of f_i. If f_j was reported, then v_{t_x} lies left of C_i, otherwise, v_{t_x} lies left of C_i'. 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-axis. Therefore, C_x lies left of C and so the appropriate interval of f_x will aready have been considered. We thus have a contradiction.

5.2 The insertion problem

Consider the following problem: given a tree T representing the *ileftdom* relation on a set $F_\omega=(f_1, f_2, \ldots, f_n)$ 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 $f_1, f_2, \ldots, f_k$, ordered from left to right, immediately left-dominated by f.

As proved in Lemma 5, 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 $f_1, f_2, \ldots, f_k$, preferably in their natural order. Suppose the subtrees of a tree T, ordered from left to right, are $T_1, T_2, \ldots, T_r$. 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 $T_1, T_2, \ldots, T_r$. Thus, if the children of a node h, ordered from left to right, are $h_1, h_2, \ldots, h_s$ then in the preorder listing of T the nodes $h, h_1, h_2, \ldots, h_s$ appear in the given order. Referring to Fig. 14, determining which faces are immediately left-dominated by f is equivalent to determining which of the relevant 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 Fig. 15, if the vertical interval of C between v_t and f_p is cut by f. For the special case in which $f = f_{\max}$, no interval is examined.

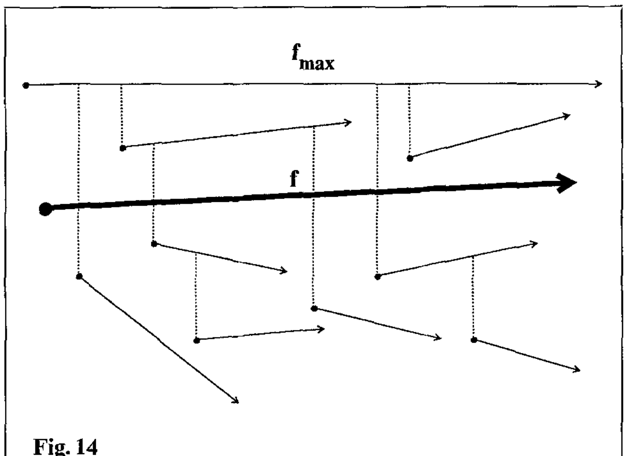

Fig. 14

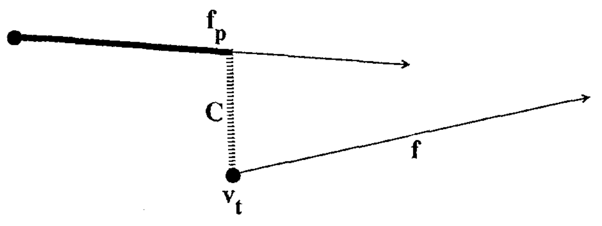

Fig. 15

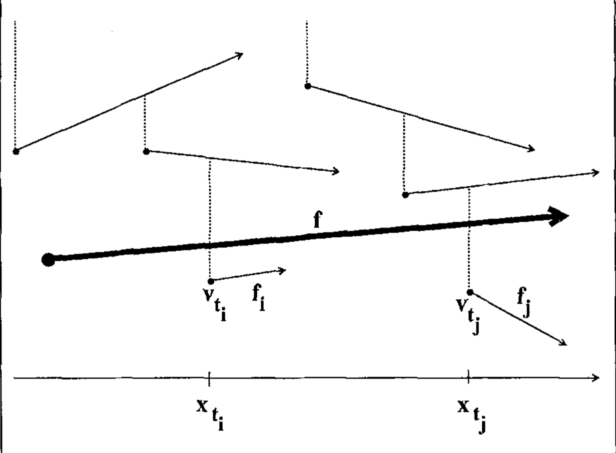

Fig. 16

Lemma 6. *The faces $f_1, f_2, \ldots, f_k$, 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 $f_1, f_2, \ldots, f_k$ 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-axis, f may intersect a given chain at most once. Referring to Fig. 16, $x_{t_i} < x_{t_j}$ and so f cuts C_i left of C_j with the result that f_i will have been considered before f_j.

Theorem 6. *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.

Refer to Fig. 17 for an illustration of the reconfiguration of a tree resulting from the insertion of a face.

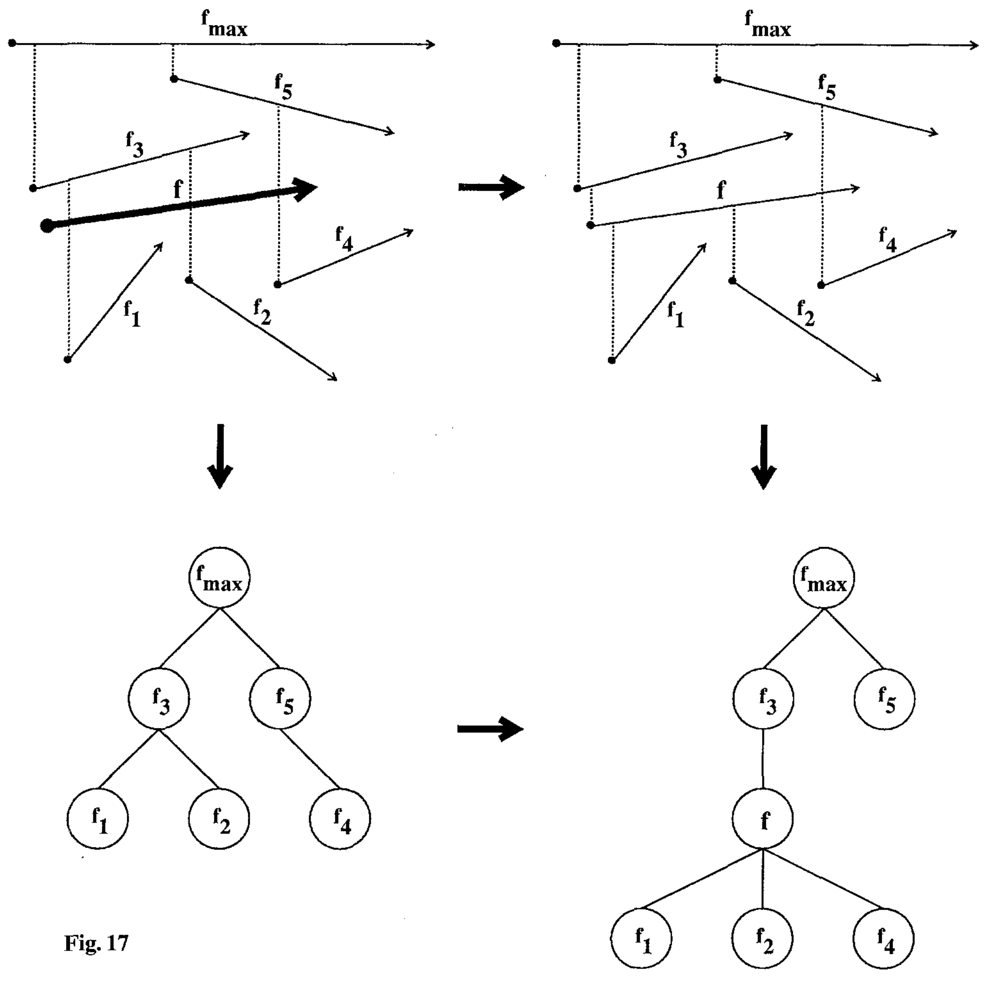

Fig. 17

5.3 The deletion problem

Consider the following problem: given a tree T, representing the *ileftdom* relation on a set $F_\omega = (f_1, f_2, \ldots, f_n)$ 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 $f_1, f_2, \ldots, f_k$. 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.

Note that the subtrees rooted by $f_1, f_2, \ldots, f_k$ will remain intact, and thus need not be considered in the search for $f_{p_1}, f_{p_2}, \ldots, f_{p_k}$. Given f_i, $1 \le i \le k$, we know, from Lemma 5, 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 $f_{p_1}, f_{p_2}, \ldots, f_{p_k}$.

Lemma 7. *The faces $f_{p_1}, f_{p_2}, \ldots, f_{p_k}$, those immediately left-dominated by $f_1, f_2, \ldots, f_k$, 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-axis, 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 Fig. 18, since no pair chains can cross, and also since l_i lies left

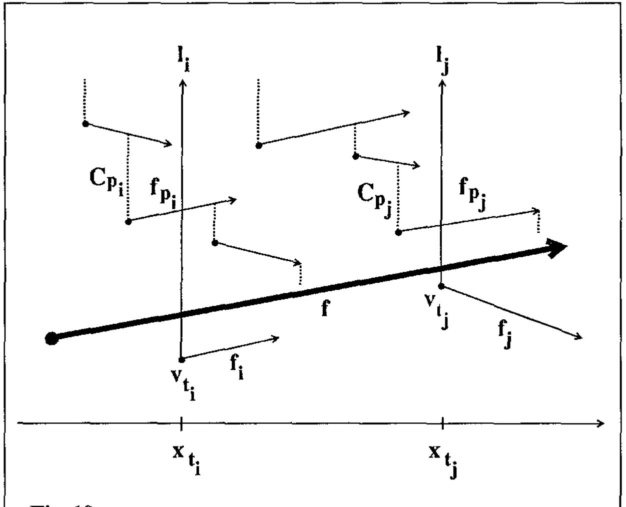

Fig. 18

of l_j, C_{p_i} lies left of C_{p_j} and so the same result holds.

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 7. *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.

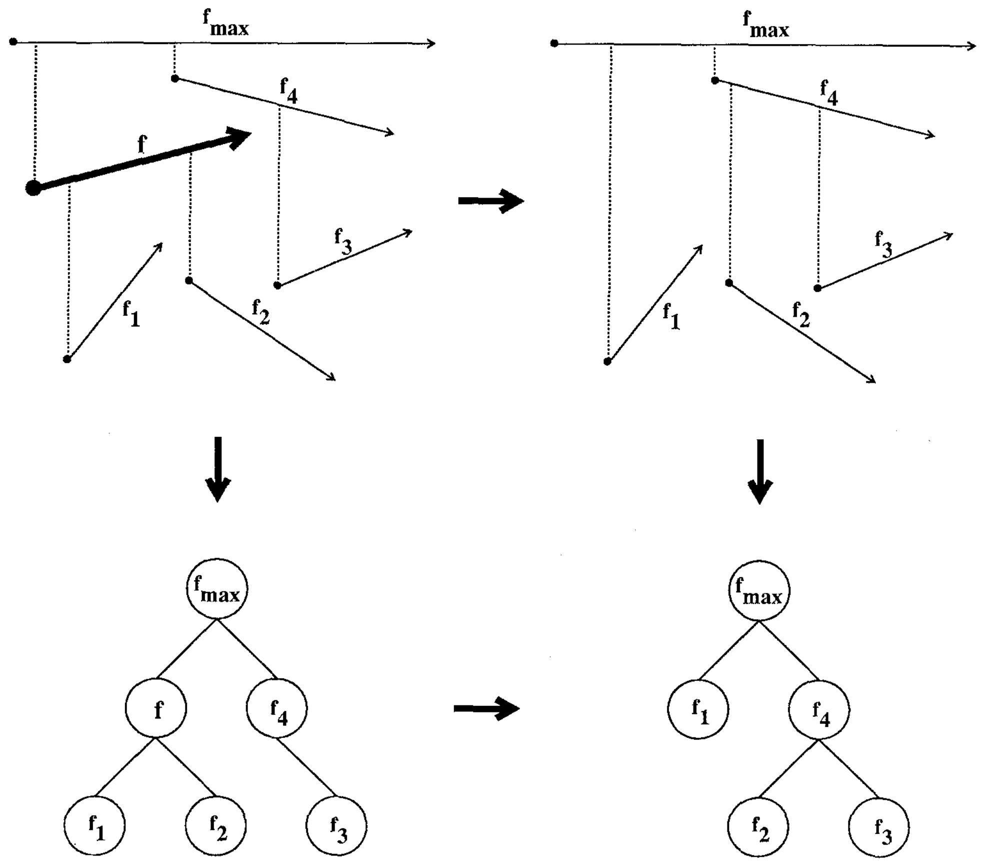

Fig. 19

Refer to Fig. 19 for an illustration of the reconfiguration of a tree resulting from the deletion of a face.

6 Conclusion and further research

Several new results pertaining to the priority approach to hidden-surface removal have been presented. In particular, a tree-based formalism for describing priority orderings has been introduced and used to simplify an existing algorithm (Yao 1980). As well, decomposition techniques have been considered for a variety of classes of scenes in order to eliminate the possibility of cyclic constraints. The resulting algorithm requires $O(n \log n)$ time if $t=1$ and $O(t\,n \log n + n \log n \log m)$ time if $t>1$. Note that with only minor modifications, the algorithm presented could be adapted to include the degeneration of a minor base-face to an edge or a vertex. Finally, $O(n)$ time insertion and deletion algorithms, which rely on the tree-based formalism, have been developed to solve the problem of maintaining a priority ordering in a dynamic environment.

There are several interesting and related research problems that remain unsolved. 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 other dynamization techniques could be used to obtain sublinear algorithms for the insertion and deletion problems.

References

Aho AV, Hopcroft JH, Ullmann JD (1983) Data structures and algorithms. Addison-Wesley, Reading, pp 88–93

Chazelle B (1982) A theorem on polygon cutting with applications. Proc 23rd Ann IEEE Symp Foundat Comput Sci, pp 339–349

Chazelle B, Incerpi J (1983) Triangulating a polygon by divide-and-conquer. Proc 21st Ann Allerton Conf Commun Control Comput, pp 447–456

Devai F (1986) Quadratic bounds for hidden-line elimination. Proc ACM Symp Comput Geom, pp 269–275

Garey MR, Johnson DS, Preparata FP, Tarjan RE (1978) Triangulating a simple polygon. Inf Proc Lett 7:175–179

Guibas LJ, Yao FF (1980) On translating a set of rectangles. Proc 12th Ann ACM Symp Theor Comput, pp 154–157

Guting RH, Ottmann T (1984) New algorithms for special cases of the hidden-line elimination problem. Tech Rep 184, Univ Dortmund

Hertel S, Mehlhorn K (1983) Fast triangulation of simple polygons. Proc Conf Foundat Comput Theor, pp 207–218

McKenna M (1987) Worst-case optimal hidden-surface removal. ACM Trans Graph 6(1):19–28

Muller DE, Preparata FP (1978) Finding the intersection of two convex polyhedra. Theor Comput Sci 7(2):217–236

Nurmi O (1985) A fast line-sweep algorithm for hidden-line elimination. BIT 25(3):466–472

Ottmann T, Widmayer P (1983) On translating a set of line segments. Comput Vision Graph Image Process 24(3):382–389

Ottmann T, Widmayer P, Wood D (1985) A worst-case efficient algorithm for hidden-line elimination. Int J Comput Math 18(2):93–119

Rappaport D (1986) A linear algorithm for eliminating hidden lines from a polygonal cylinder. The Visual Computer 2:44–53

Schmitt A (1981) Time and space bounds for hidden-line and hidden-surface algorithms. Proc Eurographics '81, pp 43–56

Shamos MI, Hoey D (1976) Geometric intersection problems. Proc 17th Ann IEEE Symp Foundat Comput Sci, pp 208–215

Shamos MI (1977) Computational Geometry. PhD Thesis, Yale Univ

Yao FF (1980) On the priority approach to hidden-surface algorithms. Proc 21st Ann IEEE Symp Foundat Comput Sci, pp 301–307