International Journal of Computational Geometry & Applications
© World Scientific Publishing Company

A SIMPLIFIED TECHNIQUE FOR HIDDEN-LINE ELIMINATION IN TERRAINS*

FRANCO P. PREPARATA†
Department of Computer Science, Brown University
Providence, Rhode Island 02912–1910, U. S. A.

and

JEFFREY SCOTT VITTER‡
Department of Computer Science, Brown University
Providence, Rhode Island 02912–1910, U. S. A.

ABSTRACT

In this paper we give a practical and efficient output-sensitive algorithm for constructing the display of a polyhedral terrain. It runs in $O((d+n)\log^2 n)$ time and uses $O(n\alpha(n))$ space, where d is the size of the final display, and $\alpha(n)$ is a (very slowly growing) functional inverse of Ackermann's function. Our implementation is especially simple and practical, because we try to take full advantage of the specific geometrical properties of the terrain. The asymptotic speed of our algorithm has been improved upon theoretically by other authors, but at the cost of higher space usage and/or high overhead and complicated code. Our main data structure maintains an implicit representation of the convex hull of a set of points that can be dynamically updated in $O(\log^2 n)$ time. It is especially simple and fast in our application since there are no rebalancing operations required in the tree.

Keywords: display, hidden-line elimination, polyhedral terrain, output-sensitive, convex hull.

1. Introduction

A large number of scenes in graphics applications can be modeled efficiently and effectively by polyhedral terrains. A terrain is a three-dimensional closed polyhedron having the property that, for each location (x, y) in the plane, the values z for which (x, y, z) is on the polyhedron form either the empty set, a single point, or a closed interval.

Reif and Sen recently did pioneering work on the generation of displays of terrains.[1] They gave an $O((d+n)\log^2 n)$-time and $O(n\alpha(n)\log n + d)$-space output-sensitive algorithm for the hidden-line elimination of an n-edge terrain whose dis-

*An earlier version of this work appeared in the Proceedings of the 1992 Symposium on Theoretical Aspects of Computer Science (STACS '92), Paris, France, February 1992, published in Lecture Notes in Computer Science, **577**, Springer-Verlag, Berlin, 135–146.

†Electronic mail can be addressed to franco@cs.brown.edu.

‡Electronic mail can be addressed to jsv@cs.brown.edu.

play consists of d segments. (The notation $\alpha(n)$ denotes a functional inverse of Ackermann's function.) Their technique resorts to general-purpose primitives such as ray shooting and is therefore ponderous and not immediately likely to lead to practical implementation. They also showed that dynamic fractional cascading techniques can further reduce their theoretical time and space bounds to $O((d+n)\log n \log\log n)$ time and $O(n\alpha(n)+d)$, respectively. Katz, Overmars, and Sharir more recently developed an elegant algorithm that improved the time bound to $O((n\alpha(n)+d)\log n)$ time, but still used $O(n\alpha(n)\log n)$ space.[2]

Output-sensitive algorithms have the advantage that their running time depends upon the complexity d of the final display. In the worst case, d can be $\Theta(n^2)$, but in real-life scenes it is typically less.

The purpose of this paper is essentially methodological, since it shows how to take advantage of the specific nature of the objects involved in order to get a very simple and, therefore, practical algorithm, with the same asymptotic time complexity as that of Reif and Sen's original algorithm, and using less space than both algorithms mentioned above. Our emphasis rests on the viewpoint that the analysis of asymptotic performance does not exhaust the objectives of algorithmic design, and that simplicity and ease of implementation are equally important criteria.

In the next section we define the display problem for terrains and discuss useful geometrical features of terrains. In Section 3, we describe our main data structure. It can be viewed as an efficient implicit representation of the lower convex hull of a semidynamic set of points (by which we mean that the universe of points is known beforehand). In Section 4 we give the $O((d+n)\log^2 n)$-time algorithm for terrain display, based upon a $O(\log^2 n)$-time algorithm for dynamically maintaining the implicit representation of the lower hull.

In Section 5 we describe the following result of independent interest: If we remove the semidynamic assumption, we can replace our semidynamic data structure with a more general balanced tree and get an alternative dynamic data structure for the lower convex hull that is simpler than the one proposed in (Ref. 3). Our new data structure uses only one balanced tree, as opposed to the linear number of balanced trees required by Overmars and van Leeuwen. The representation of the lower hull is implicit rather than explicit, but it can support all the applications described by Overmars and van Leeuwen with the same performance guarantees.

2. Preliminaries

Let R be a planar subdivision in the (x,y) plane defined by n vertices. We assume for simplicity and with essentially no loss of generality that R has a unique unbounded region. For each open bounded region (simple polygon) r of R, we have an affine function $f_r(x,y)$ defined for all (x,y) in region r; for the unbounded region r^* we have $f_{r^*}(x,y)=0$.

A *polyhedral terrain* (or simply *terrain*) τ is a polyhedron given by the following function $z(x,y)$ defined on the interiors $\bigcup_{r\in R} r$ of all the regions of R of the plane:

$$z(x,y) = f_r(x,y), \qquad \text{where } (x,y)\in r.$$

As stated, z is a function defined only in the interiors of the regions of R. We extend the definition of z to the edges of R, so that if edge e is shared by regions r' and r'', then for each point u of e, we define the following z-coordinate interval:

$$z(u) = \Big[\min\{f_{r'}(u), f_{r''}(u)\},\ \max\{f_{r'}(u), f_{r''}(u)\}\Big]. \tag{1}$$

Frequently, for each point u of e, we have $f_{r'}(u) = f_{r''}(u)$; however, in general, the polyhedron defined by the terrain τ can have as many as $2n$ vertices. Each vertical line intersects τ in a single connected domain, either a point or an interval.

Let $v_0 = (x_0, y_0, z_0)$ be the *observation point* of the viewer and let $\mathbf{n} = (n_x, n_y, n_z)$ be the normal to the display plane P. Informally, the observer's eye is placed at v_0 and points in direction $\mathbf{n}$; the terrain can be imagined as being projected onto P. Without loss of generality, we let $n_x = 0$, and we assume that the planar subdivision R is *regular*,[4] in that R admits a complete ordered set $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_p\}$ of separators in the (x, y) plane monotone with respect to the x-axis. In particular, we assume that no edge of R is perpendicular to the x axis. Let p_{min_x} and p_{max_x} be the unique two points of the boundary of R that have the minimum and maximum x values, respectively. Each edge in R is contained in at least one monotone separator σ_j. Each separator σ_j is a monotone chain of edges of R that starts at p_{min_x} and ends at p_{max_x}. By "monotone," we mean that any line perpendicular to the x axis intersects σ_j at most once. According to standard convention, if $i < j$ then a line perpendicular to the x-axis and intersecting both σ_i and σ_j intersects σ_i closer to $y = y_0$ than it intersects σ_j. The number p of separators is $O(n)$.

The set of separators Σ naturally induces a complete set of three-dimensional polygonal chains $\Sigma' = \{\sigma'_1, \sigma'_2, \ldots, \sigma'_q\}$ of τ, which are ordered so that whenever two polygonal chains $\sigma'_i, \sigma'_j \in \Sigma'$, for $i < j$, have points $u_i \in \sigma'_i$, $u_j \in \sigma'_j$ that project to the same point in the (x, y) plane, we have $z(u_i) \le z(u_j)$ (cf. (1)). We let $\overline{\sigma_j}$ denote the central projection of $\sigma'_j \in \Sigma'$ to the display plane P, and we define $\overline{\Sigma} = \{\overline{\sigma_1}, \overline{\sigma_2}, \ldots, \overline{\sigma_q}\}$. For efficiency of representation, we eliminate from each polygonal chain $\overline{\sigma_j}$ those edges that appear in an earlier polygonal chain $\overline{\sigma_i}$, for $i < j$.

As in (Ref. 1), the construction of the display of the terrain τ is done incrementally, by successively processing the polygonal chain of $\overline{\Sigma}$, starting from $\overline{\sigma_1}$ closest to the observation point and proceeding away to $\overline{\sigma_q}$. The construction maintains in the display plane P an x-monotone polygonal chain ρ, called the *silhouette*, that separates the "clear" from the "opaque."

Specifically let $\overline{\sigma_j}$ be the currently processed polygonal chain of $\overline{\Sigma}$. The advancing mechanism of the construction intersects $\overline{\sigma_j}$ with ρ. The portion of $\overline{\sigma_j}$ lying below ρ is eliminated; the portion of ρ lying below $\overline{\sigma_j}$ is reported (as part of the display). The upper envelope of $\overline{\sigma_j}$ and ρ forms the new updated silhouette ρ:

$$\rho := \sup(\rho, \overline{\sigma_j}),$$

where sup will also denote a function that implements the silhouette update.

Our approach makes use of several interesting geometric features of the silhouette:

- The silhouette is an x-monotone chain (in the display plane P) consisting of lower-convex subchains separated by vertices that are the display-images of original vertices of the terrain. By "lower-convex," we mean that the subchains are counterclockwise from left to right, or equivalently, the line segment between two points on a subchain lies above the subchain.
- Each of the lower-convex subchains introduced above contains at most N segments, where $N = O(n)$ is the number of edges of τ, since each edge contributes at most one segment to each convex chain.
- The worst-case total number of edges of the silhouette is $\Theta(n\,\alpha(n))$, where $\alpha(n)$ is a functional inverse of Ackermann's function, which follows by the well-known results on the one-sided envelope of a collection of segments.[5,6,7]
- A chain $\overline{\sigma_j}$ is a concatenation of edges and gaps. (Gaps correspond to edges appearing in previously handled chains.) Thus each time the left extreme p' of an edge of $\overline{\sigma_j}$ belongs to the display (that is, it lies above ρ), then the abscissa of p' is the left extreme of the range of some leaf of T.

3. Silhouette Data Structure

The data structure we use to store the silhouette ρ is an implicit recursive representation of the lower convex hull of the vertices of ρ. From now on, whenever we refer to a point, segment, edge, terrain, etc., we refer to their projections in the display plane P. Since the abscissæ of the vertices of τ are known *a priori*, we order them and denote by X their ordered set. We store them as leaves in a *contracted binary tree* (CBT),[8,9] which we call T. We recall here for the reader's convenience that the nodes in a CBT T, which we call the *active nodes*, are a subset of the nodes of a "skeletal" balanced binary tree T^*, whose leaves correspond 1–1 to the members of a fixed ordered universe X (which in our case is the set of abscissæ of the vertices). For simplicity we assume that $|X|$ is a power of 2. More formally, the *active nodes* are defined as follows:

(i) The root is always active.
(ii) A leaf is active if it is stored in T and otherwise it is inactive.
(iii) Any internal nonroot node is active if both of its subtrees contain active nodes.

The number of nodes in the CBT T is clearly bounded by twice the number of leaves in T. The CBT has depth at most $\log|X|$ and no update ever involves rebalancing since the skeletal tree T^* has depth $\log|X|$. Insertions and deletions in T can be done in constant time, when a pointer is given to the element's neighbor; otherwise they require $O(\log|X|)$ time. Moreover, adjacent leaves and adjacent nodes in symmetric order are accessible from one another in constant time. For more details, the reader is referred to (Ref. 9).

We store the silhouette in the following manner: Our universe X is an ordered multiset of abscissæ, one per vertex of τ. (If $k > 1$ vertices have the same abscissa, we order the k copies of the abscissa according to the following ordering of the corresponding vertices: the vertices whose edges have τ to their left are ordered from top to bottom, and are followed by the vertices whose edges have τ to their right, ordered from bottom to top.) Let $\{\lambda_1, \lambda_2, \ldots \lambda_p\}$, $p \leq n$, be the active leaves of the CBT T, each corresponding to a vertex of τ appearing in the silhouette ρ in left-to-right order. With a slight abuse of notation, we use λ_j to represent both the leaf in the CBT and the vertex in τ it corresponds to. With each λ_j, we associate a secondary structure S_{λ_j} (to be discussed below) that stores the convex subchain, denoted $\rho(\lambda_j)$, terminating at the vertex λ_j on the right. Each leaf λ_j, for $j \geq 2$, identifies an interval *range*(λ_j), defined to be $[x(\lambda_{j-1}), x(\lambda_j)]$. The range of an internal node is defined to be the union of the ranges of its leaves.

In each internal node w of T we store two additional items:

(i) We store a segment called the *bridge* (denoted *bridge*(w)), which is the common supporting segment of the lower convex hulls of the portions of the silhouette corresponding to the left and right subtrees of w, respectively. (A lower convex hull of a set of points P in the plane is the convex hull of $P \cup \{(0, +\infty)\}$.) If the supporting segment is not well defined (because the concatenation of the nonvertical portions of the two convex hulls is convex), we make the convention that *bridge*(w) is the first segment of the right convex hull. For example, in Figure 1, the bridge spanning the convex subchains $\rho(\lambda_{17})$ and $\rho(\lambda_{20})$ is the first segment ℓ in $\rho(\lambda_{20})$'s convex subchain.

(ii) We store the abscissa *rightmost*(w) of the rightmost vertex of the subtree rooted at w.

An example of T is pictured in Figure 1a.

The rest of this section describes the general organization of each secondary data structure S_{λ_j}, which stores the convex subchain $\rho(\lambda_j)$ of ρ that terminates at λ_j. The internal vertices of $\rho(\lambda_j)$ are not vertices of τ, but rather they are defined implicitly by intersections of polygonal chains in $\overline{\Sigma}$. Fortunately we can represent $\rho(\lambda_j)$ in such a way that it can itself be stored efficiently in a CBT (which always requires a fixed universe). We denote the CBT by S_{λ_j}.

We set the universe of CBT S_{λ_j} to be the set A of edges in $\overline{\Sigma}$ ordered according to their slope. The convex subchain $\rho(\lambda_j)$ is represented by the edges of A that form it. More specifically, let $e_{i_1}, e_{i_2}, \ldots, e_{i_t}$, where $e_{i_k} = \overline{(p_{k-1}, p_k)}$, be the edges of $\overline{\Sigma}$ that contain the segments of $\rho(\lambda_j)$. The marked leaves of the CBT S_{λ_j} are $e_{i_1}, e_{i_2}, \ldots, e_{i_t}$. Edge e_{i_k} represents the vertex p_k of $\rho(\lambda_j)$, and p_0 is represented implicitly; for $1 \leq k < t$, p_k is the intersection of e_{i_k} and $e_{i_{k+1}}$, and $p_t = \lambda_j$.

We associate the root of each CBT S_{λ_j} with the leaf λ_j of the CBT T. We denote the resulting "combined" tree by $\widehat{T}$. Because $\rho(\lambda_j)$ is a convex subchain, all the bridges in the recursive decomposition of $\rho(\lambda_j)$ (based on the structure of the CBT S_{λ_j}) are simply segments in $\rho(\lambda_j)$ and thus do not need to be stored explicitly. The bridge *bridge*(w) for an internal node w in S_{λ_j} is the segment whose leaf (edge) in S_{λ_j} is the symmetric-order successor of w in S_{λ_j}, which can be found in constant time. A typical CBT S_{λ_j} is pictured in Figure 1b.

4. Dynamic Update of the Silhouette

The dynamic update of the silhouette ρ caused by processing polygonal chain $\overline{\sigma_j}$

Figure 1: (a) The current silhouette ρ consists of the non-bold solid lines, which comprise the top border of the figure. The lower convex hull of ρ consists of the segments d, b, a, m. The two subhulls whose bridging (by segment a) gives the full hull are pictured in bold lines. All the bridges in the recursive decomposition of the full hull are indicated by dashed lines, except for bridge ℓ, which is also a segment of the silhouette. (b) The combined tree data structure $\widehat{T}$ consists of the upper CBT T and the lower CBTs S_{λ_1}, S_{λ_2}, ... Upper CBT T stores the vertices of the silhouette ρ that are display-images of vertices of the terrain τ. Each internal node w in the diagram is labeled by the segment $bridge(w)$. For simplicity, the only lower CBT pictured is S_{λ_7}, which stores λ_7's convex subchain e_{i_1}, e_{i_2}, e_{i_3}, e_{i_4}.

consists of three tasks:

Task 1. Computation of the intersections of ρ and $\overline{\sigma_j}$. We define an intersection of ρ and $\overline{\sigma_j}$ to be a changeover point in which $\overline{\sigma_j}$ changes from being on or above ρ on one side of the intersection to being strictly below ρ on the other side.

Task 2. Dynamic update of the display: We output to the display the portions of ρ lying below $\sup(\rho, \overline{\sigma_j})$.

Task 3. Dynamic update of the data structure $\widehat{T}$: We remove from $\widehat{T}$ the portions of ρ lying on or below σ_j, and we insert into $\widehat{T}$ the portions of $\overline{\sigma_j}$ lying on or above ρ.

The three tasks can be implemented concurrently left-to-right on σ_j and ρ. Tasks 1 and 2 are handled in Section 4.1 in $O(d' + n' \log^2 n)$ time, where n' is the number of edges of $\overline{\sigma_j}$ and d' is the number of segments output to the display during the processing of $\overline{\sigma_j}$. In Section 4.2 we show that $\widehat{T}$ can be updated in $O((d'+1)\log^2 n)$ time. The size of the data structure is bounded by the size of the largest silhouette ever encountered, which is $O(n\alpha(n))$. This gives us our main theorem:

Theorem 1 *Hidden-line elimination for terrains can be done without the need for balanced-tree data structures in $O((d+n)\log^2 n)$ time and in $O(n\alpha(n))$ space, where d is the number of segments in the resulting display, and $\alpha(n)$ is a (very slowly growing) functional inverse of Ackermann's function.*

4.1. Computation of the Intersections and Dynamic Update of the Display

We do Tasks 1 and 2 by handling $\overline{\sigma_j}$ edge-by-edge from left to right, as indicated in the following loop. Let $e = \overline{(p', p'')}$ denote the current edge of $\overline{\sigma_j}$ being processed, starting with the point $p \in e$. That is, segment $e' = \overline{(p, p'')}$ is the portion of e currently being processed. We distinguish two cases:

(i) *p and the portion of e' immediately to its right are on or above ρ.* In this case we march along the silhouette ρ until either we find an intersection q of e' with ρ or we reach the abscissa of p'' (in which case we set $q := p''$) (Task 1). The portion of ρ lying below $\overline{(p, q)}$ is reported as appearing in the display (Task 2). The above loop continues with $p := q$.

(ii) *The portion of e' immediately to the right of p is below ρ.* In this case the display does not change in a neighborhood of p (Task 2), and we have to search for the leftmost intersection q of e' with ρ; if none exists, we set $q := p''$ (Task 1). The loop then continues with $p := q$.
The search for q begins with the node w of T that has the smallest level (leaves of T have level 0) among those whose range contains that of e'. We use the segment $\mathit{bridge}(w)$ and test it against the line $\mathit{line}(e')$ containing e'. We denote the line containing $\mathit{bridge}(w)$ by $\mathit{line}(\mathit{bridge}(w))$. We distinguish three cases, as illustrated in Figure 2:

(a) *$\mathit{bridge}(w)$ does not extend below $\mathit{line}(e')$.* If $\mathit{line}(\mathit{bridge}(w))$ intersects $\mathit{line}(e')$ to the right (respectively, left) of $\mathit{bridge}(w)$, then w's left sub-

Figure 2: The different cases in the recursive search for the first intersection between edge segment e' and the silhouette ρ.

tree (respectively, right subtree) can be excluded from the search. (See Figure 2a.)
Justification: If, for example, $\mathit{line}(\mathit{bridge}(w))$ intersects $\mathit{line}(e')$ to the right of $\mathit{bridge}(w)$, then all vertices of ρ pertaining to the left child of w are confined to the shaded region in Figure 2a obtained by intersecting the closed halfplane above $\mathit{line}(\mathit{bridge}(w))$ and the closed halfplane to the left of the vertical line $x = x(\mathit{rightmost}(\mathit{left\_child}(w)))$. Clearly none of this region's edges can intersect e'.

(b) *$\mathit{bridge}(w)$ lies entirely below $\mathit{line}(e')$.* In this case, w's right subtree can be excluded from the search. (See Figure 2b.)
Justification: By assumption, the part of e' immediately to the right of p is below ρ. Since ρ contains the left endpoint of $\mathit{bridge}(w)$, which is below $\mathit{line}(e')$, there must be an intersection between $\mathit{line}(e')$ and the portion of ρ pertaining to w's left child.

(c) *$\mathit{bridge}(w)$ lies partly on or above $\mathit{line}(e')$ and partly below $\mathit{line}(e')$.* In this case, there is certainly at least one intersection of e' with ρ, but we may not know which subtree of w contains the portion of ρ having the first intersection with e', as in Figure 2c. The search is continued in the left subtree, in accordance with the left-to-right update policy. If no intersection is found in the left subtree, processing continues with the right subtree. The search for q halts whenever an intersection is found.

Finding the intersections of $\overline{\sigma_j}$ with ρ and the dynamic update of the display in Case (i) requires $O(n' + d')$ time, where n' is the number of edges of $\overline{\sigma_j}$ and d' is the number of segments output to the display during the processing of $\overline{\sigma_j}$.

We now show that the time required for Case (ii) is $O(n' \log^2 n)$. If e' does not intersect ρ, we traverse a path from node w to a leaf of $\widehat{T}$, each time under Case (a) or Case (b) spending constant time at each node, for a total of $O(\log n)$ time.

Figure 3: (a) The silhouette before the processing of $\overline{\sigma_j} = e_1, e_2, \ldots, e_4$. (b) The silhouette after the processing.

Suppose instead that e' intersects ρ. We consider a path in $\widehat{T}$ from w to the leaf pertaining to the segment containing the intersection q. In the worst case, each node on this path is a right child and at each such node we launch an unsuccessful search through the left sibling as specified by Case (c). Clearly there are $O(\log n)$ nodes from which such unsuccessful searches can be taken, each search taking $O(\log n)$ time, before q is found.

4.2. Dynamic Update of $\widehat{T}$

In the algorithm in Section 4.1 for computing intersections between the polygonal chain $\overline{\sigma_j}$ being processed and the silhouette ρ, the polygonal chain $\overline{\sigma_j}$ alternates between being on or above ρ and being below ρ. The dynamic update of the combined CBT data structure $\widehat{T}$ (Task 3) is concerned with the portion of $\overline{\sigma_j}$ that lies on or above ρ (which corresponds to Case 1 of Tasks 1 and 2, described in Section 4.1).

Let p denote the last processed intersection point of $\overline{\sigma_j}$ with ρ. Immediately after p, $\overline{\sigma_j}$ is on or above ρ. Let $e_1, e_2, \ldots, e_k$, with $e_i = \overline{(p_{i-1}, p_i)}$, denote a maximal subchain of $\overline{\sigma_j}$ such that $p \in e_1$, $p \neq p_0$, and edges $e_2, \ldots, e_{k-1}$ lie completely on or above ρ, as illustrated in Figure 3a for the case $k = 4$. Let λ_{left} be the first vertex of τ whose convex subchain $\rho(\lambda_{left})$ intersects e_1 at p, and let λ_{right} be the last vertex of τ whose convex subchain $\rho(\lambda_{right})$ intersects e_k as close to p along $\overline{\sigma_j}$ as possible. We modify T by deleting the leaf λ_{left} and inserting the leaves $p_1, p_2, \ldots p_{k-1}$. The convex subchain $\rho(p_i)$, for $2 \leq i \leq k-1$, is initialized to contain the singleton edge e_i. These updates to $\widehat{T}$ require $O(k)$ time. The convex subchain $\rho(p_1)$ is initialized to contain the edges of $\rho(\lambda_{left})$ up to its intersection with e_1, followed by the portion of e_1 after the intersection. Similarly, in the convex subchain $\rho(\lambda_{right})$, the initial portion, up to the intersection with e_k, is deleted and replaced with the portion of e_k up to the intersection. If $k = 1$, then $\rho(\lambda_{right})$ in addition stores the initial part of $\rho(\lambda_{left})$ up to its intersection with e_1. This is illustrated in Figure 3b. These updates involve splits of the CBTs $S_{\lambda_{left}}$ and $S_{\lambda_{right}}$, which, if $k = 1$, are spliced together; this can be done easily in $O(\log n)$ time. The total time for this process is thus $O(\log n + k)$.

We mark the leaves $p_1, \ldots, p_{k-1}, \lambda_{right}$ for later processing. The next portion of the polygonal chain $\overline{\sigma_j}$, starting with the intersection of e_k and $\rho(\lambda_{right})$, lies below the silhouette ρ and is processed as in Case (ii) in Section 4.1, seeking the subsequent intersection of $\overline{\sigma_j}$ and ρ. The portion of $\overline{\sigma_j}$ that remains after Case (ii) is again on or above ρ, so we repeat the above process.

After the above processing of the polygonal chain $\overline{\sigma_j}$ is completed, there may be several marked leaves in the CBT T. They represent the vertices of τ whose convex subchains were created or modified as a result of processing $\overline{\sigma_j}$. In the remainder of the dynamic update, we perform a postorder traversal of the portion of T that lies above the marked leaves, in order to update the bridges stored in those nodes.

At each internal node w in T lying on a rootward path from each of the marked leaves (on the way up in the postorder traversal), we compute in $O(\log n)$ time the updated bridge segment $bridge(w)$ for node w by processing the bridge information in the nodes below w in T, as follows: We initialize ℓ and r to be the left and right children of w. We use $line(bridge(\ell))$ and $line(bridge(r))$ to denote the straight lines containing the segments $bridge(\ell)$ and $bridge(r)$. Let λ denote the rightmost leaf $rightmost(\ell)$ in the subtree rooted at ℓ. We carry out the procedure below until ℓ and r are each leaves in $\widehat{T}$. The resulting ℓ and r are made the endpoints of $bridge(w)$.

As long as ℓ and r are both internal nodes of $\widehat{T}$, we repeatedly do the actions corresponding to the following two cases:

(i) *Some portion of $bridge(r)$ (respectively, $bridge(\ell)$) lies below $line(bridge(\ell))$ (respectively, $line(bridge(r))$).* We set ℓ to be ℓ's left child (respectively, we set r to be r's right child). (See Figure 4a.)
Justification: If, for example, some portion of $bridge(r)$ lies below $line(bridge(\ell))$, then all vertices of ρ pertaining to the right child of ℓ are confined to the shaded region pictured in Figure 4a, obtained by intersecting the closed halfplane above $line(bridge(\ell))$ and the strip delimited by the two vertical lines $x = x$(left extreme of $bridge(\ell)$) and $x = x(\lambda)$. Clearly none of this region's vertices can support $bridge(w)$.

(ii) *Both $bridge(\ell)$ and $bridge(r)$ lie on or above the line extension of the other.* We denote by $int(line(bridge(\ell)), \lambda)$ and $int(line(bridge(r)), \lambda)$, respectively, the ordinates of the intersections of $line(bridge(\ell))$ and of $line(bridge(r))$ with the vertical line $x = x(\lambda)$.

  (a) If $int(line(bridge(\ell)), \lambda) \leq int(line(bridge(r)), \lambda)$, then we set ℓ to be ℓ's right child. (See Figure 4b.)

  (b) If $int(line(bridge(\ell)), \lambda) \geq int(line(bridge(r)), \lambda)$, then we set r to be r's

Figure 4: Recursive descent needed to determine *bridge*(w) for a node w in $\widehat{T}$. The bridges for the left child ℓ and the right child r of w are shown. (a) The left anchor point for *bridge*(w) must be in ℓ's left subtree. (b) The left anchor point for *bridge*(w) must be in ℓ's right subtree.

left child. (This case is symmetric to Case (a).)

Justification: Since we are negating Case 1, the wedge formed by $line(bridge(\ell))$ and $line(bridge(r))$ is lower-convex. Thus if, for example, we have $int(line(bridge(\ell)), \lambda) \leq int(line(bridge(r)), \lambda)$, then the vertices of ρ pertaining to the left child of ℓ are confined to the shaded region pictured in Figure 4b, obtained by intersecting the closed halfplane above $line(bridge(\ell))$ and the closed halfplane to the left of the vertical line $x = x(\text{right extreme of } bridge(\ell))$. None of this region's vertices can support *bridge*(w).

When this loop terminates, either ℓ or r is a leaf in the combined CBT $\widehat{T}$. We repeatedly do the following step until ℓ and r are both leaves: If ℓ (respectively, r) is a leaf and lies on or below $line(bridge(r))$ (respectively, $line(bridge(\ell))$), then we set r to be r's right child (respectively, ℓ to be ℓ's left child); otherwise we set r to be r's left child (respectively, ℓ to be ℓ's right child).

At this point, both ℓ and r are leaves of $\widehat{T}$. We set *bridge*(w) to be the segment whose endpoints are represented by the leaves ℓ and r.

The time to compute each bridge by the above procedure requires $O(\log n)$ time. The number of bridges to be updated as a result of processing $\overline{\sigma_j}$ is $O(n' \log n)$, where n' is the number of edges in $\overline{\sigma_j}$. Thus the time required for all the bridge updates caused by $\overline{\sigma_j}$ is $O(n' \log^2 n)$. Theorem 1 follows by summing the running times needed for the three tasks in the processing of $\overline{\sigma_j}$.

5. Dynamic Representation of the Convex Hull

In this section we describe how to use our lower convex hull data structure of Sections 3 and 4 in a variety of dynamic applications described by Overmars and van Leeuwen.[3] In these applications the universe of points is not necessarily known in advance, so we use a balanced tree instead of a CBT to implement L. The points S stored in L are ordered from left to right, and those with the same abscissa are ordered from top to bottom. There is no need for the secondary data structures that we described in Sections 3 and 4; all the points of S are stored as leaves of L. The resulting data structure makes use of a single balanced tree L as opposed to the data structure of (Ref. 3), which uses a linear number of balanced trees; there, each node in the tree has a secondary data structure represented by a balanced tree.

The following theorem shows that the dynamic applications given by Overmars van Leeuwen[3] can be implemented within the same time and space bounds using our data structure L for the lower convex hull.

Theorem 2 *Our lower convex hull data structure L and the symmetrically defined upper convex hull data structure U support the following operations:*

(i) *A set of n points in the plane can be "peeled" in $O(n \log^2 n)$ time.*
(ii) *The convex layers of a set of n points in the plane can be determined in $O(n \log^2 n)$ time.*
(iii) *A connecting spiral of a set of n points in the plane can be determined in $O(n \log^2 n)$ time.*
(iv) *A set of points in the plane can be maintained at the cost of $O(\log^2 n)$ time per insertion or deletion so that the query "does p lie on the convex hull of the set of points?" can be answered in $O(\log n)$ time.*
(v) *Two sets A and B of points in the plane can be maintained at the cost of $O(\log^2 n)$ time per insertion or deletion so separability can be determined in constant time.*

The following theorem is useful for determining the convex layers:

Theorem 3 *Using our lower convex hull data structure L, the lower convex hull of a set of points $S = \{p_1, p_2, \ldots p_n\}$ can be output in $O(h \log \frac{n}{h})$ time, where h is the current size of the lower convex hull of S.*

Let $x1(bridge(w))$ and $x2(bridge(w))$ denote the left and right abscissae, respectively, of segment $bridge(w)$. We prove Theorem 3 by giving the pseudocode that outputs the lower convex hull of the points in left-to-right order. Let λ_1 and λ_n denote the abscissæ of the leftmost and rightmost points; in case of a tie, we choose the bottommost point. The printing of the lower hull is done via the procedure call *print_hull*(root of L, λ_1, λ_n). In the following pseudocode, w is a node of L, and *left* and *right* are abscissæ satisfying *left* $\leq$ *right*.

```
$print\_hull(w, left, right)$;
  **begin**
    **if** $left < x1(bridge(w))$ **then**
      $print\_hull(left\_child(w), left, x1(bridge(w)))$;
    **if** $[x1(bridge(w)),\ x2(bridge(w))] \subseteq [left, right]$ **then**
      Output $bridge(w)$;
    **if** $x2(bridge(w)) < right$ **then**
      $print\_hull(left\_child(w), x2(bridge(w)), right)$
  **end**;
```

The running time is bounded by the path length to the nodes in the tree where the segments of the lower convex hull are stored as bridges. If there are h segments on the lower hull, we can show by convexity arguments that the path length is $O(h \log \frac{n}{h})$. The justification for the pseudocode is given in the following lemma:

Lemma 1 *The segment $bridge(w)$ belongs to the lower convex hull of the set of points S represented in L if and only if*

$$\Big(x1(bridge(w)),\ x2(bridge(w))\Big) \quad \bigcap \quad \Big(x1(bridge(v)),\ x2(bridge(v))\Big) = \emptyset \quad (2)$$

for each ancestor v of w in L.

Proof. ($\Longrightarrow$) Suppose that $bridge(w) = \overline{(p_{left}, p_{right})}$ is part of the lower convex hull. Then the lower hull lies in the halfplane $H^+(w)$ on or above $line(bridge(w))$. For any ancestor v of w, all the points stored in the subtree of L rooted at v, including the endpoints of $bridge(w)$, are contained in $H^+(v)$. Therefore, $line(bridge(v))$ intersects the vertical line $x = x(p_{left})$ (respectively, $x = x(p_{right})$) at point q_{left} (respectively, q_{right}) at or below p_{left} (respectively, p_{right}). If $bridge(w)$ and $bridge(v)$ are colinear, condition (2) must hold, since the bridges can intersect only at their endpoints. Otherwise if the bridges are not colinear and condition (2) does not hold, then the segment $\overline{(q_{left}, q_{right})}$ contains a point of $bridge(v)$ not in $H^+(w)$, which contradicts the assumption that $bridge(w)$ belongs to the lower convex hull.

($\Longleftarrow$) Suppose that $bridge(w) = \overline{(p_{left}, p_{right})}$ does not belong to the lower convex hull. Then there is a point $p \in S$ not in $H^+(w)$. By definition of bridge, p cannot lie directly under $bridge(w)$, but must be either to the left or the right of the vertical strip spanned by $bridge(w)$. Without loss of generality, let us assume that (1) $p \in S$ is to the right of such strip, (2) p is not in $H^+(w)$, and (3) p is the leftmost point in S satisfying (1) and (2). Let v be the lowest common ancestor in L of the leaves for p_{right}, p_{left} and p. By the definition of $bridge(w)$, it follows that node w is in the left subtree of v and the leaf for p is in the right subtree of v. This implies that the left endpoint q of $bridge(v)$ precedes p in S. Assume, for a contradiction, that $x(p_{right}) \leq x(q)$. By definition of p, there is no point in S between p_{right} and p that is below $line(bridge(w))$. Since q is either between p_{right} and p in S or else on or directly above p_{right}, we have $q \in H^+(w)$. Since both q and p are in $H^+(v)$, while $q \in H^+(w)$ and $p \notin H^+(w)$, it follows that $line(bridge(v))$ intersects $line(bridge(w))$ at or to the right of p_{right} in S, and thus $p_{left} \notin H^+(v)$, which contradicts the definition of $bridge(v)$. Therefore, we must have $x(q) < x(p_{right})$, and condition (2) is violated. $\square$

The rest of this section is devoted to proving Theorem 2. The first application of peeling can be implemented easily by observing that the bridge stored at the root of L must be on the lower convex hull. Hence we can "peel" off the two endpoints of the bridge, dynamically update L in $O(\log^2 n)$ time, and continue until no points remain.

For the second application, we need to peel off an entire layer before updating L and U. Theorem 3 can be invoked to output the ith layer in $O(h_i \log n)$ time, where h_i is the size of the ith layer, and $\sum_i h_i = n$. Deleting the points in a layer takes $O(\log^2 n)$ time per point, resulting in a total time bound of $O(n \log^2 n)$.

The third application can be done in a similar way, except that after each layer is removed, the last point on the previous layer is left temporarily in L and U. The first segment on the convex hull starting at that point, which is the desired spiral segment that connects the previous layer to the next, can be found by executing the procedure for Theorem 3 up until the first segment is output, which takes $O(\log n)$ time.

In the fourth application, we can determine whether a point p lies on the convex hull in $O(\log n)$ time by traversing L and U from the leaf for p upward to their respective roots. By Lemma 1, the point p is on the lower (respectively, upper) convex hull if and only if every bridge in L (respectively, in U) encountered either contains p or has an x-range to the left or right of $x(p)$.

For the last application, we use lower and upper convex hull data structures for each of the sets A and B. We can separate A and B if and only if one of the sets has a lower convex hull that does not intersect the upper convex hull of the other. For simplicity let us consider whether the lower convex hull of A (including its interior) intersects the upper convex hull of B (including its interior); the other case is symmetrical. We store the lower convex hull of A in data structure L_A and the upper convex hull of B in data structure U_B.

Let a and b denote the roots of L_A and U_B. We carry out the actions corresponding to the following cases repeatedly until either it is determined that there is or isn't an intersection or else either a or b is a leaf. If a is a leaf, for example, a corresponds to a single segment, and we test whether a's endpoints are on or inside the upper convex hull of B using a procedure like that used for the previous application. We leave the justification to the reader.

(i) *The vertical strip lying on or directly above $bridge(a)$ intersects the vertical strip lying on or directly below $bridge(b)$.* In this case there is an intersection.

(ii) *$bridge(a)$ is on or below $line(bridge(b))$ or $bridge(b)$ is on or above $line(bridge(a))$.* If $bridge(a)$ is to the left (respectively, right) of $bridge(b)$, then we set a to be a's right (respectively, left) child and b to be b's left (respectively, right) child.

(iii) *Cases (i) and (ii) do not hold.*

  (a) If $bridge(a)$ and $bridge(b)$ are parallel, there can be no intersection.

(b) If $\mathit{int}(\mathit{line}(\mathit{bridge}(a)), \mathit{line}(\mathit{bridge}(b)))$ is to the right (respectively, left) of $\mathit{bridge}(a)$, then we set a to be a's right (respectively, left) child.

(c) If $\mathit{int}(\mathit{line}(\mathit{bridge}(a)), \mathit{line}(\mathit{bridge}(b)))$ is to the right (respectively, left) of $\mathit{bridge}(b)$, then we set b to be b's right (respectively, left) child.

In each iteration we descend one level in L_A or U_B. Thus we can determine separability in $O(\log n)$ time.

Acknowledgements

Support for Franco Preparata was provided in part by National Science Foundation grant CCR–8906419. Support for Jeff Vitter was provided in part by a National Science Foundation Presidential Young Investigator Award CCR–9047466 with matching funds from IBM Corporation, by National Science Foundation grant CCR–9007851, and by the U.S. Army Research Office under grant DAAL03–91–G–0035.

References

1. J. H. Reif and S. Sen, "An efficient output-sensitive hidden-surface removal algorithm and its parallelization," *Proc. 4th Annual ACM Symp. on Computational Geometry*, Urbana-Champaign, IL, June 1988, pp. 193–200.
2. M. J. Katz, M. H. Overmars, and M. Sharir, "Efficient hidden surface removal for objects with small union size," *Proc. 7th Annual Symp. on Computational Geometry*, North Conway, NH, June 1991, pp. 31–40.
3. M. H. Overmars and J. van Leeuwen, "Maintenance of configurations in the plane," *J. Comput. System Sci.* **23** (1981) 166–204.
4. F. P. Preparata and M. I. Shamos, *Computational Geometry* (Springer-Verlag, New York, 1985).
5. R. Cole and M. Sharir, "Visibility problems for polyhedral terrains," *J. Symbolic Computation* (1989) 11–30.
6. A. Wiernik and M. Sharir, "Planar realization of nonlinear Davenport-Schinzel sequences," *J. Discrete Computational Geometry* **3** (1988) 15–47.
7. S. Hart and M. Sharir, "Nonlinearity of Davenport-Schinzel sequences and of a generalized path compression scheme," *Combinatorica* **6** (1986) 151–177.
8. D. T. Lee and F. P. Preparata, "Parallel batch planar point location on the CCC," *Information Processing Letters* **33** (December 1989) 175–179.
9. F. P. Preparata, J. S. Vitter, and M. Yvinec, "Computation of the axial view of a set of isothetic parallelepipeds," *ACM Transactions on Graphics* **9** (July 1990) 278–300.