1989

Polygon Nesting and Robustness

Chanderjit Bajaj
Tamai Dey

Report Number: 89-918

http://docs.lib.purdue.edu/cstech/782

This document has been made available through Purdue e-Pubs, a service of the Purdue University Libraries. Please contact epubs@purdue.edu for additional information.
POLYGON NESTING AND ROBUSTNESS

Chanderjit Bajaj
Tamal Day

CSD-TR-918
October 1989
Polygon Nesting and Robustness

Chanderjit Bajaj*,
Tamal Dey†,

Computer Sciences Department
Purdue University
CSD-TR-918
October, 1989

* Supported in part by NSF Grant MIP 88-16286, ARO contract DAAG29-85-C-0018 under Cornell MSI and ONR contract N00014-88-K-0402.
† Supported in part by a David Ross Fellowship.
Polygon Nesting and Robustness

Chanderjit L. Bajaj*  
Tamal Dey†

Department of Computer Science  
Purdue University  
West Lafayette, IN 47907

1 Introduction

We consider the problem of computing the nesting structure of a set of \( m \) simple, planar polygons with \( n \) vertices and \( N \) notches (reflex angles). The polygons are mutually nonintersecting, that is they do not intersect along their boundary. This problem arises as a fundamental subproblem in our robust polyhedral decomposition algorithm [2] as well as in the algorithms of [4] and [12].

Problem: Let \( \mathcal{P} \) be a set of \( m \) simple polygons \( P_i, i = 1, \ldots, m \). Corresponding to each polygon \( P_i \) we define \( \text{ancestor}(P_i) \) as the set of polygons containing \( P_i \). The polygon \( P_k \) in \( \text{ancestor}(P_i) \) is called the parent of \( P_i \) if \( \text{ancestor}(P_k) = \text{ancestor}(P_i) - P_i \). Notice that there may not exist any such \( P_k \) since \( \text{ancestor}(P_i) \) may be empty. In that case we say that the parent of \( P_i \) be \( \text{null} \). Any polygon whose parent is \( P_k \) is called the child of \( P_k \). See Figure 1.1. The nesting structure \( G \) of \( \mathcal{P} \) is an acyclic directed graph (A forest of trees) in which there is a node \( n_i \), corresponding to each polygon \( P_i \) in \( \mathcal{P} \), and there is a directed edge from a node \( n_i \) to \( n_j \) if \( P_j \) is the parent of \( P_i \). The polygon nesting problem is to compute the nesting structure of a set of simple nonintersecting polygons.

Related Work: In [4] Chazelle gives an \( O(n \log n) \) algorithm to detect the outermost polygons and their children, given a set of simple nonintersecting polygons with \( n \) vertices. However his algorithm does not compute the nesting structure of the given set of polygons.

Results: In section 3 we give an algorithm which computes the polygon nesting structure in \( O(n + (m + N) \log (m + N)) \) time where \( n \) is the total number of vertices in \( m \) polygons and \( N \) is the total number of notches. Since in practice \( m \) and \( N \) are much less than \( n \), this algorithm runs much faster than any \( O(n \log n) \) algorithm. In section 4 we give a robust algorithm for the same problem restricted to a class of polygons called fleshy polygons. Our robust algorithm has a worst-case time bound of \( O(n \log n + m + N) \).

2 Preliminaries

Definitions: Let \( P \) be a simple polygon with vertices \( v_1, v_2, \ldots, v_n \) in clockwise order. A vertex \( v_i \) is a notch of \( P \) if the inner angle between the edge \((v_{i-1}, v_i)\) and \((v_i, v_{i+1})\) is \( > 180^\circ \). Between any two consecutive notches \( v_i, v_j \) in the clockwise order, the sequence of vertices \((v_i, v_{i+1}, \ldots, v_j)\) is called a convex polygonal-line. Each polygonal-line can be partitioned into convex-chains, which are maximal pieces of a polygonal-line, with the property that its vertices form a convex polygon. Each convex-chain can be further partitioned into at most three \( z \)-monotone maximal pieces called subchains, i.e., vertices of a subchain have \( x \)-coordinates in either strictly increasing or decreasing order. See Figure 2.1.

A vertex or an edge is said to lie inside a polygon if it completely lies inside the polygonal region restricted by the boundary of the polygon. A vertex or an edge is said to be contained in a polygon if it lies on the boundary of the polygon.

Let \( L \) be a line drawn through a set of polygons. Let \( E \) be the set of edges which intersect \( L \) in the following two ways. An edge \( e \in E \) either properly intersects \( L \) (i.e., two vertices of \( e \) lies on the opposite sides of \( L \)) or \( e \) intersects \( L \) at a vertex and the other vertex of \( e \) lies to the right of \( L \). The third possible case of \( e \) intersecting \( L \) is ignored as the information related to that edge would already be recorded in a plane sweep. Finally, degenerate intersection (\( e \) is collinear with \( L \)) is handled in section 3.

An edge \( e_1 \) in \( E \) is said to be “above” the edge \( e_2 \) in \( E \) if the point of intersection of \( L \) and \( e_1 \) lies above the point of intersection of \( L \) and \( e_2 \). If \( e_1 \) and \( e_2 \) have a common vertex through which \( L \) passes, \( e_1 \) is “above” \( e_2 \) if the other vertex of \( e_1 \) lies above the line containing \( e_2 \). \( L \) induces a total order \( R \) on the edges in \( E \) with respect to the “above” relation. If \( L \) passes through a vertex \( v_i \), we define \( \text{above}(v_i) \) as the set of edges whose point of intersection with \( L \) is above \( v_i \). The lowest edge in \( \text{above}(v_i) \) is called the neighbor of \( v_i \). Between \( v_i \) and its neighbor there is no other edge intersecting \( L \). See Figure 1.1 and 3.2. Note that there may not exist any neighbor of \( v_i \) since \( \text{above}(v_i) \) may be empty.
Order $R$ naturally extends to another order $O$ of subchains associated with the edges in $R$. If edges $e_1$, $e_2$ of subchains $C_1$ and $C_2$ are intersected by $L$, then $C_1$ is “above” $C_2$ if the point of intersection of $L$ and $e_1$ is above the point of intersection of $L$ and $e_2$.

**Lemma 2.1.** Let $P$ be a simple polygon with $N_p$ notches. No line can intersect $P$ in more than $\max(1,2N_p)$ segments or $\max(2,2N_p+1)$ points.

**Proof:** See [2]

**Lemma 2.2.** Let $P$ be a simple polygon with $N_p$ notches. The number of subchains $S_p$ in $P$ is bounded as $S_p \leq 6(1 + N_p)$.

**Proof:** See [2].

**Lemma 2.3.** Let $L$ be any line through a vertex $v_i$ of a polygon $P_i$. Let the edge $e$ be the neighbor of $v_i$. Parent of $P_i$ is either the polygon $P_j$ containing $e$ or $P_j$’s parent (possibly null).

**Proof:** If the neighbor edge $e$ of $v_i$ is an edge of $P_i$, which is a parent of $P_j$, the lemma holds trivially. Suppose the neighbor edge $e$ of $v_i$ is an edge of $P_j$ which is not the parent of $P_i$. We claim that $v_i$ lies inside polygon $P_j$ iff $e$ lies inside it. Suppose $e$ lies inside $P_j$ and $v_i$ does not. Then the region between $v_i$ and $e$ on $L$ contains a part which is outside $P_j$. Hence there must be an edge of $P_j$ between $e$ and $v_i$ on $L$. But this is impossible since $e$ is the neighbor edge of $v_i$. Similarly, we can argue that if $v_i$ lies inside polygon $P_j$, so does $e$. Hence $e$ lies inside the same set of polygons, within which $v_i$ lies. Hence if $P_k$ is the parent of $P_i$ it is a parent of $P_j$ and vice versa.

**Lemma 2.4.** Let $L$ be any line passing through $v_i$ of $P_i$. If $v_i$ is contained in the polygon $P_k$ iff the number of edges of $P_k$ which are in $\text{above}(v_i)$ is odd.

**Proof:** Since any edge demarks the region which is “inside polygon $P$” and “outside polygon $P” on $L$ the above proposition is obvious.

**Lemma 2.5.** Let $L$ be any line passing through $v_i$ of $P_i$. Let edge $e$ of polygon $P_i$ be the neighbor of $v_i$ on $L$. If the number of edges of $P_k$ in $\text{above}(v_i)$ is odd and $k \neq i$ then $P_k$ is the parent of $P_i$. Otherwise, $P_k$’s parent (possibly null) is the parent of $P_i$.

**Proof:** Combine Lemma 2.3 and Lemma 2.4.

### 3 Polygon Nesting Structure

**Plane Sweep:** Each polygon $P_i$ consists of subchains $C_{i1}, C_{i2}, ..., C_{ik}$. We sweep a line $L$ in the plane through all the polygons, while maintaining the ordering $O$ of the subchains $C$ induced by $L$. To maintain this ordering we stop only at the endpoints of the subchains, while sweeping say from left to right. We break all the boundaries of the polygons into subchains in no more than $O(n)$ time where $n$ is the total number of vertices of all the polygons. We sort only the endpoints of all the subchains on a line perpendicular to $L$. At each subchain endpoints we update the ordering $O$ as follows.

**Update at a Vertex:** if $v_i$ is a vertex such that both subchains $C_1$ and $C_2$ connected to $v_i$ have not yet been encountered by the sweep line $L$, we insert $C_1$ and $C_2$ in the ordering $O$ on $L$ by a simple binary search. The search is based on a procedure for determining the position of $v_i$ w.r.t. the edge intersected by $L$ on a subchain $C_i$ already present in the ordering $O$ on $L$.

For the latter purpose, we keep a last visited edge associated with each subchain $C_i$ in $O$, as we now detail. This is reminiscent of the topological sweep of [7]. Let the edge associated with $C_i$ initially be $e_1$, the first edge of the subchain $C_i$. We visit the sequence of edges $e_1, e_2, ..., e_k$ of $C_i$ stopping at the first edge $e_k$ which intersects $L$. We determine the “above” relation of $v_i$ w.r.t. $e_k$ and associate edge $e_k$ with $C_i$. Later, when we need to classify any other vertex w.r.t. $C_i$ we start from edge $e_k$. See Figure 3.1. Obviously, the edges $e_1, ..., e_{k-1}$ are visited only once, while $e_k$ and $e_k$ are visited more than once throughout a sweep. Now, for each vertex-edge classification, there will be at most two edges similar to $e_1$ and $e_k$ of a subchain which will be visited more than once. Since in the binary search for determining the position of a vertex in the order $O$, we encounter only $O(\log S)$ subchains (where $S$ is the total number of subchains) there will be at most $O(\log S)$ edges, for each line position, which will be visited more than once. Hence, for each update at $v_i$ (where we insert subchains) we visit $t_i$ edges which are visited only once throughout the sweep and $O(\log S)$ edges which are visited more than once. If $v_i$ is a vertex such that both subchains connected to $v_i$ have been encountered then we delete both these subchains from the ordering $O$. This again takes at most $O(\log S)$ time. Hence the total time taken for all updates is $O \left( \sum_{i=1}^{S} t_i \right) + O(\log S)$ where $S$ is the total number of subchains and $t_i$ is the number of edges visited at each update, which is visited only once throughout the sweep. Certainly, $\sum_{i=1}^{S} t_i = O(n)$ where $n$ is the total number of vertices. Hence updates take $O(n) + O(\log S)$ time.

**Detecting parent of a polygon:** At the vertex $v_i$ of $P_i$, when we insert the subchains in the ordering $O$ on $L$ we determine the parent of $P_i$ as follows. If parent of $P_i$ has already been determined then we are done. If it has not we find the neighbor edge $e$ of $v_i$ on $L$ (Actually, $e$ is found while inserting the subchains connected to $v_i$). Let $P_j$ be the polygon containing $e$ on the boundary. We determine $k$, the number of edges or equivalently the number of subchains of the polygon $P_j$ which are in $\text{above}(v_i)$. Maintaining the ordering of subchains of each polygon separately, this number can be obtained in $O(\log S)$ time where $S$ is the number of subchains in that polygon. If $k$ is even and $P_j \neq P_i$, we set $P_j$ as the parent of $P_i$. Otherwise we set the parent of $P_j$ to be the parent of $P_i$ (Lemma 2.5).

Certainly parent determination at each update add up to at most
Algorithm:
Input: A set of m simple, nonintersecting polygons.
Output: A directed acyclic graph G, called the nesting structure, in which there is a directed edge from a node ni corresponding to a polygon P_i to the node nj corresponding to the polygon P_j if P_j is the parent of P_i.
Step 1: Detect the endpoints of subchains in all polygons.
Step 2: Sort the x-coordinates of these endpoints. If two points have same x-coordinates, the one with higher y-coordinate is sorted before the other. Let this sorted sequence W be v_1, v_2, ..., v_w.
Step 3: Create a node for each polygon in G. Enter two subchains, connected to the leftmost vertex of W, in the ordering O by inserting the two polygon edges connected to that vertex in O. Note O is initially empty.
Step 4: Sweep a pseudo-line from left to right, taking steps at each vertex v_i of W as follows. Let v_i be on the boundary of the polygon P_i. If both subchains connected to v_i have already been visited, delete them from the ordering O and skip steps from 4(a) to 4(d).
Step 4(a): Detect the position of v_i with respect to the subchains intersected by the sweep line. For this, carry out a binary search in the ordering O of these subchains. To detect the position of v_i with respect to a subchain C_i during binary search, find the edge e_i of this subchain kept in O and then follow the linked sequence of edges e_1, e_2, ..., e_k until the edge e_k is found which intersects L.
Step 4(b): Let e'_i of polygon P_j be the neighbor edge of v_i found by step 4(a). Determine the number k of subchains of P_j which are in above(v_i). This is done by a similar binary search, as in step 4(a), in the ordering of chains maintained separately for each polygon.
Step 4(c): Insert two subchains connected to v_i in O and in the ordering of subchains maintained for polygon P_i. In degenerate case, insert or delete the subchain which does not correspond to the edge, collinear with the sweep line, from O.
Step 4(d): If k is odd, then create a directed edge in the nesting structure from the node n_i corresponding to the polygon P_i, to the node n_j corresponding to the polygon P_j. If k is even, create a directed edge from n_j to the node n_k (if any), to which n_i is connected through a directed edge.

Theorem 2.1: The problem of polygon nesting for m polygons can be solved in O(n + (m + N)log(m + N)) time where n is the total number of vertices and N is the total number of notches of all polygons.
Proof: Detecting the endpoints of the subchains takes O(n) time. Sorting these endpoints requires O(SlogS) time. Updating and determining parent takes O(n + SlogS) time. Hence, computing the nesting structure for all polygons takes O(n + SlogS) time. By lemma 2.2, S, the total number of subchains is bounded as S \leq 6(m + N) where m is the total number of polygons and N is the total number of notches. Hence, total time spent is O(n + (m + N)log(m + N)).

4 Robustness under Finite Precision Arithmetic

In the algorithm given in the previous section we assumed arbitrary precision arithmetic in all our computations. In this section we give an algorithm for polygon nesting problem which is robust in that it never fails due to finite precision arithmetic. It correctly yields the nesting structure of a set of simple nonintersecting polygons, possessing a minimum feature with respect to their "skinniness".

We first assume that all our polygons are bounded by a square box, -B < x < B and -B < y < B. We define a polygon P to be "fleshy" if there is a point inside P such that a square with center (intersection of square's diagonals) at that point and with sides of length 2\epsilon B lies inside P. Here, \epsilon is machine precision. In our implementation we set B = 2^{16}, \epsilon = 2^{-24}. Hence the area of the square is 784 \times 2^{-16}. The polygons which are not fleshy are thus extremely skinny for most practical purposes.

Related Work: Robust computations under finite precision arithmetic have recently taken added importance because of the increasing use of geometric manipulations in computer-aided design, and solid modeling, see for e.g. [3]. Edelsbrunner and Mucke [6], and Yap [17], suggest using expensive symbolic perturbation techniques for handling geometric degeneracies. Sugihara and Iri [16], and Dobkin and Silver [5], describe an approach to achieving consistent computations in solid modeling, by ensuring that computations are carried out with sufficiently higher precision than used for representing the numerical data. There are drawbacks however, as high precision routines are needed for all primitive numerical computations, making algorithms highly machine dependent. Furthermore, the required
precision for calculations is difficult to a priori estimate for complex problems. Segal and Sequin [14] require estimating various numerical tolerances, tuned to each computation, to maintain consistency. Milenkovic [13] presents techniques for computing the output for a modified input which preserves some basic topological constraints. Green and Yao [9] present a method for drawing line segment arrangements on a discrete grid which alters the input symbolic data. Hoffmann, Hopcroft and Karasick [11], and Karasick [12], propose a proof of correctness. Sugihara [15] uses geometric reasoning and apply it to the problem of triangulating a planar point set. In this paper we use the methods of topological reasoning with a minimum feature assumption on the skinniness of the polygons.

Assumptions and Definitions: A binary predicate $\text{CONT}$ is defined as $\text{CONT}(P_1, P_2)$ iff $P_1$ contains $P_2$. $\text{NOT}(\text{CONT}(P_1, P_2))$ denotes the negation of $\text{CONT}(P_1, P_2)$. A point $p_1$ is said to be vertically visible from another point $p_2$ if the vertical line through $p_2$ also passes through $p_1$ and the vertical segment between $p_1$ and $p_2$ does not intersect any other edge. Similarly, we define an edge to be vertically visible from a point $p_1$ if the vertical line through $p_1$ intersects the edge and does not intersect any other edge in between. The numerical computations in our algorithm are carried out in two places.

1. Sorting the vertices:
   Sorting can be carried out without any error as the comparison of two floating point numbers is exact to within machine precision. (This is true on most of the machines available today.) Here we assume that given input data (coordinates of polygon vertices) is accurate.

2. Computing the points of intersection of a vertical sweep line with the edges:
   In Lemma 3.1 we will develop a bound on the maximum error which can occur during this computation. Actually, this bound leads us to the estimate of a square box with side $28\varepsilon B$, to define a fleshy polygon.

Results: We present a robust algorithm for computing the nesting structure of a set of simple, nonintersecting, fleshy polygons. Our algorithm runs in $O(n(m + N + \log n) + m^3)$ time.

Lemma 3.1: Given an edge $e$ at a point $p$, the absolute error $e_{abs}$ in the computed position of $p$ is bounded as $e_{abs} < 7\varepsilon B$, where $\varepsilon$ is the machine precision and $B$ is the largest value of any of the coordinates.

Proof: Let us consider a vertical line $x = x_0$ which intersects $e$ at $p$. Obviously, $x$ coordinate of $p$ is $x_0$. Let the $y$ coordinate of $p$ be $y_0$ and $y_e$ be the computed value of $y_0$. By simple geometry,

$$\frac{x_2 - x_1}{x_0 - x_1} = \frac{y_2 - y_1}{y_0 - y_1}$$

and

$$y_0 = \frac{(y_2 - y_1)(x_0 - x_1)}{x_2 - x_1} + y_1$$

With finite precision the computed value $y_e$ of $y_0$ is given by

$$y_e = \frac{(y_2 - y_1)(x_0 - x_1)(1 + \varepsilon^*)}{x_2 - x_1} + y_1(1 + \varepsilon_0)$$

where $1 + \varepsilon^* = \frac{1 + \varepsilon + \varepsilon_0(1 + \varepsilon + \varepsilon_0)}{1 + \varepsilon_0}$ and $|\varepsilon_0| < \varepsilon$. Let $t_0 = \frac{(y_2 - y_1)(x_0 - x_1)}{x_2 - x_1}$. We can write

$$y_e = t_0(1 + \varepsilon^*) + y_1(1 + \varepsilon_0)$$

$$y_e - y_0 = t_0\varepsilon^* + y_1\varepsilon_0$$

$$e_{abs} \leq |t_0\varepsilon^*| + |y_1\varepsilon|$$

Neglecting higher order terms in $\varepsilon$, we get $|\varepsilon^*| \leq 6\varepsilon$

Since $\frac{|x_0 - x_1|}{|x_2 - x_1|} < 1$, we have

$$|t_0| < |y_2 - y_1|$$

$$|t_0| < B$$

$$e_{abs} < 6\varepsilon B + \varepsilon B$$

$$e_{abs} < 7\varepsilon B$$
coinciding with two sides of the square as shown in the Figure 4.1.

Case(i): There is a vertex \( v \) of \( P \) within the two vertical lines. W.l.o.g, \( v \) can be assumed to be above \( ab \). Consider a vertical line \( L \) passing through \( v \). Let \( L \) intersect the line passing through \( q \) and parallel to \( ab \) at \( q' \). The set of intersecting points of the edges of the polygons with this vertical line can be partitioned into three sets \( I_{\text{above}}(v) \), \( I_{\text{close}}(v) \), \( I_{\text{below}}(v) \) based on the closeness of intersecting points to \( v \). \( I_{\text{above}}(v) \) is the set of all points of intersection whose distance from \( v \) is greater than equal to \( 14eB \) and which are above \( v \). Similarly, define \( I_{\text{below}}(v) \). The rest of the points of intersection constitute \( I_{\text{close}}(v) \). Corresponding to each set \( I_{\text{above}}(v) \), \( I_{\text{below}}(v) \), \( I_{\text{close}}(v) \), we define \( E_{\text{above}}(v) \), \( E_{\text{below}}(v) \), \( E_{\text{close}}(v) \) as the set of edges on which those points of intersection lie.

Let \( s \) be the point of intersection of \( L \) with the edge of \( P \) which is vertically visible from \( q' \) and which below \( cd \). Any polygon containing \( P \) cannot have an edge intersecting \( L \) in between \( v \) and \( q' \) and \( q' \) and \( s \). Since the distance between \( v \) and \( s \) must be greater than equal to \( 28eB \), the computed distance between them must be at least \( 21eB \). Hence, \( s \) cannot be in \( I_{\text{close}}(v) \). For any polygon \( P_i \), let \( K_{\text{above}}(v), K_{\text{close}}(v), K_{\text{below}}(v) \) be the number of edges of \( P_i \) in \( E_{\text{above}}(v), E_{\text{close}}(v), E_{\text{below}}(v) \) respectively. Polygon \( P_i \) contains the portions of \( L \) which is in between \( I_{\text{close}}(v) \) and \( I_{\text{below}}(v) \) if \( K' = K_{\text{above}}(v) + K_{\text{close}}(v) \) is odd. This portion also lies in \( P \). Hence, if \( K' \) is odd either \( P \) contains \( P_i \) or \( P_i \) contains \( P \). But using Lemma 3.2 one of these two possibilities is omitted by checking the leftmost vertex of each polygon. Hence, it can be determined correctly whether \( P_i \) is an ancestor of \( P \) or not.

Case(ii): There is no vertex \( v \) which lies in between two vertical lines \( L_1 \) and \( L_2 \). In this case, only two edges of \( P \) will be vertically visible from \( q \). Let these two edges be \( e_1, e_2 \) as shown in Figure 4.1(b). Let \( r \) (resp.) be the first vertex which is hit by a vertical line \( L \) if we sweep \( L \) from the position of \( L_2 \) (resp.) to right (left resp.). Consider a vertical line through \( r \) which intersects \( e_1 \) and \( e_2 \) at \( b' \) and \( c' \) respectively. Similarly, consider the vertical line through \( r \) which intersects \( e_1 \) and \( e_2 \) at \( a' \) and \( d' \) respectively. Certainly, the quadrilateral \( ab'c'd' \) lies inside \( P \). Since \( abcd \) lies inside \( ab'c'd' \), one of the edges \( b'c' \) and \( a'd' \) must be greater than equal to \( 28eB \). W.l.o.g let us assume \( b'c' \) is that edge. Certainly, \( r \) is at a distance of at least \( 14eB \) either from \( b' \) or \( c' \). W.l.o.g let us assume the distance between \( r \) and \( c' \) is greater than equal to \( 14eB \). Following the same logic as in Case (i) we can determine the ancestors of \( P \) by counting the number of edges in \( E_{\text{above}}(r) \) and \( E_{\text{close}}(r) \) for each polygon \( P_i \).

**Algorithm:**

**Input:** A set of simple, nonintersecting, fleshy polygons.

**Output:** A acyclic directed graph, called the nesting structure, in which each node \( n_i \) represent a polygon \( P_i \). There is a directed edge from \( n_i \) to \( n_j \) iff \( P_j \) is the parent of \( P_i \).

**Step 1:** Sort the vertices of the polygons on the x axis. Let this sorted sequence be \( v_1, v_2, ..., v_n \).

**Step 2:** Sweep a vertical line from left to right taking the following steps at each vertex \( v_i \).

**Step 2(a):** Let \( P \) be the polygon having \( v_i \) on the boundary and \( E \) be the set of edges which were intersected by \( L \) when the sweep line stopped at \( v_{i-1} \). Compute the intersection point of \( L \) with each edge in \( E \). Construct the sets \( E_{\text{above}}(v_i), E_{\text{close}}(v_i), E_{\text{below}}(v_i) \) for \( v_i \).

**Step 2(b):** Count the number of edges of \( P \) in \( E_{\text{above}}(v_i) \) and \( E_{\text{close}}(v_i) \). If this number is odd then take step 2(c) otherwise skip 2(c).

**Step 2(c):** For each polygon \( P_i \) intersected by \( L \) count the number of edges in \( E_{\text{above}}(v_i) \) and \( E_{\text{close}}(v_i) \). Compute \( K' = E_{\text{above}}(v_i) + E_{\text{close}}(v_i) \). If \( K' \) is odd, then check the leftmost vertices of \( P_i \) and \( P \) to determine whether \( \text{NOT}(\text{CONT}(P, P_i)) \) or \( \text{NOT}(\text{CONT}(P_i, P)) \). If \( \text{NOT}(\text{CONT}(P, P_i)) \) then create a directed edge from the node corresponding to \( P_i \) to the node corresponding to \( P \) in the nesting structure. Note that this will create a directed edge from \( n_i \) to \( n_j \) iff \( P_j \) is an ancestor(not merely parent) of \( P_i \). This nesting structure is refined in Step 3.

**Step 2(d):** If \( v_i \) is a vertex such that both edges adjacent to \( v_i \) were not in \( E \), then include them in \( E \). If \( v_i \) is a vertex such that both edges adjacent to it were in \( E \), then delete them from \( E \). If \( v_i \) is a vertex such that one of the edges were in \( E \) then delete that edge from \( E \) and include the other edge adjacent to \( v_i \) in \( E \).

**Step 3:** In the nesting structure computed by Step 2(c) determine the longest path from each node \( n_i \) to every other node. If no node is reachable from \( n_i \) then \( P_i \) is the corresponding polygon \( P_i \) is null. Otherwise, the polygon \( P_j \), corresponding to the node \( n_j \) with the longest path length of \( l \) is the parent of \( P_i \).

**Time Analysis:** Step 1 takes \( O(n \log n) \) time. Since a vertical line intersects at most \( O(m + N) \) edges (Lemma 2.1), Step 2 takes \( O(m + N) \) time for each stop while sweeping. Hence, total time spent for Step 2 is \( O(n(m + N)) \). The longest path determination in step 3 for each node takes \( O(m^2) \). Since the underlying graph of the nesting structure with \( m \) nodes is directed and acyclic, we can apply the well known Dijkstra's shortest path algorithm (See for e.g. [1]) with negative weight of -1 on every edge, to determine the longest path from a source to every other node. Hence, step 3 takes \( O(m^2) \) time for \( m \) nodes. Combining these, the time complexity \( T \) of the robust algorithm for polygon nesting of a set of simple, nonintersecting, fleshy polygons is given by, \( T = O(n \log n + n(m + N) + m^3) = O(n \log n + m + N) + m^3) \).
References


Figure 1.1
Above(v) = \{e_1, e_3, e_5, e_6\}
norther(v) = e_4
Parent of p_3 = p_2, Parent of p_3 = Null
Ancestor of p_3 = \{p_1, p_2\}

Figure 3.1 (a)
e_1 is kept with c_1 before the step at v_1.

Figure 3.1 (b)
e_1 is kept with c_2 after computing position of v_1 with respect to c_2.

Figure 3.2
above(v_4) = \{e_4, e_7, e_8, e_9\}
e_5 is a degenerate case.
Figure 4.1 (a)  
Case (i)

Figure 4.1 (b)  
Case (ii)