Constrained Delaunay Triangulations for Polygons with Interior Points and Holes

by

Teofilo Gonzalez and Mohammadreza Razzazi‡
Department of Computer Science
University of California
Santa Barbara, CA 93106

ABSTRACT: We present an O(n log n) algorithm to construct a constrained Delaunay triangulation (CDT) for a simple polygon with interior points and holes. The main difference between our algorithm and the previous algorithms for this problem is that our algorithm reduces the problem to a set of line intersection problems plus finding Delaunay triangulations (DT) of several simple polygons plus finding a Delaunay triangulation of a set of points. Our algorithm is based on partitions and exploit useful properties of constrained Delaunay triangulations.

KEYWORDS: Computational Geometry, Constrained Delaunay Triangulations, Time Optimal Algorithms.

‡ Current address: Computer Engineering and Science Department, Case Western Reserve University, Cleveland, Ohio 44106.

July 27, 1991

I. INTRODUCTION

The problem of constructing a Delaunay Triangulation, and its dual (constructing a Voronoi Diagram), are fundamental problems in computational geometry. There are numerous application for these two problems which have been studied from the computational point of view for many years. Shamos and Hoey [SH] developed an $O(n \log n)$ algorithm to construct the Voronoi diagram for Q and showed that a Delaunay triangulation for Q may be subsequently obtained in $O(n)$ time. They also showed that any "decision tree" type algorithm must take at least $\Omega(n \log n)$ time to solve either of the two problems. Lee and Schacter [LS] developed an $O(n \log n)$ time algorithm to construct a Delaunay triangulations, without constructing the Voronoi diagram. Before we discuss our problem, let us formally define the problems just discussed. Let Q be a set of points in the plane. An *edge for Q (or simply an edge)* is a line segment that joins two points in Q and a *triangulation* for Q is a maximal set of edges for Q no two of which cross. A *Delaunay Triangulation (DT)* for Q is any triangulation for Q in which every edge satisfies the circle property. We say that edge e satisfies the *circle property* if there exists a circle C that passes through its endpoints and there is no vertex of Q in the interior of C. The edges (triangles) in a Delaunay triangulation are called *Delaunay edges (triangles)* and are referred to as *d_edges (d_triangles)*. The dual of the Delaunay triangulation problem is the problem of constructing a *Voronoi Diagram* [SH]. A Voronoi Diagram for a set of points $Q = \{q_1, q_2, ... \ q_n\}$ in the plane is a partition of the plane into a set of regions $R = \{r_1, r_2, ..., r_n\}$ with the property that for each i, $q_i \in r_i$ and for all $j \neq i$ each point in r_i is not farther from q_i than from q_j.

Generalization of these two problems have been extensively studied (see for example [WS], [Cw], [C], [S] and [PS]). In this paper we study a generalization of the Delaunay triangulation problem known as the *Constrained Delaunay Triangulation (CDT)* problem. This problem has many interesting applications, including the approximation of terrain surfaces ([LL]). Before we discuss known algorithms for this problem, let us formally define the *CDT* problem. Let P be a simple polygon, and let H be a set of pairwise disjoint simple polygons, called *holes,* defined inside P. The edges of P are called *boundary edges* and the edges of the holes in H are called *hole edges*. Let D be a set of points inside I_P-R_H, where I_P is the region inside P and R_H is the region inside and the boundary of H. We define $V(A)$, where A is a set of pairwise disjoint polygons, as the union of the set of vertices in each polygon in A. Let $Q = D \cup V(P) \cup V(H)$ and let n be the cardinality of Q. A *Constrained Delaunay Triangulation (CDT) for Q restricted by P and H* is a set of edges for Q that satisfy the circle property and partition the region I_P-R_H into triangles. An edge e is said to satisfy the *circle property* if there is a circle C that passes through its endpoints and which does not include inside it any other point in Q that is visible (when considering boundary and hole edges as obstacles) from both of the endpoints of e. It is simple to see that the constrained Delaunay triangulation problem reduces to the Delaunay triangulation for Q when P is a convex polygon and there are no holes. When each hole degenerates into a single line segment, we refer to the CDT problem as the *CDT'* problem. It is simple to see that the CDT' problem is a restricted version of the CDT problem.

A brute force algorithm for the CDT problem was developed by Nielson and Franke [NF]. Lee and Lin [LL] presented an $O(n^2)$ algorithm for the CDT' problem and an $O(n \log n)$ algorithm for the case when there are neither holes nor interior points. The latter algorithm is based on Chazelle's [Ch] divide-and-conquer partitioning rule for polygons. Chew [Cw] developed a divide-and-conquer algorithm for the CDT' problem that takes optimal time, i.e, $O(n \log n)$ time. Wang and Shubert [WS] introduced the notion of Bounded Voronoi diagram and showed how to find the Bounded Voronoi diagram and the CDT' in $O(n \log n)$ time. Jung ([J1], [J2]) adapted this algorithm to find the CDT' directly. As noted in [J1], the performance of the algorithm degrades as the number of hole edges increases. Seidel [S] developed the notion of an Extended Voronoi Diagram (EVD) and showed that it is the dual of the CDT' problem. Seidel's [S] algorithm constructs the extended Voronoi diagram and by duality the CDT' in $O(n \log n)$ time. All of these algorithms can be easily adapted to solve the CDT problem; therefore, the CDT and CDT' problems are computational equivalent problems. Let us explain these algorithms in more detail in order to compare them to our algorithm.

> Chew's [Cw] algorithm begins by sorting the set of points Q along their x-coordinate values and a box covering all its points is defined. The box is partitioned into vertical strips each containing exactly one point (this is possible since it is assumed that all points have distinct x-coordinate values). The CDT' is constructed for each strip and then the triangulations of adjacent strips are combined until the CDT' of the entire problem is obtained.
>
> Jung's [J1] and [J2] method, which is exactly the dual of Wang and Shubert's method [WS] is different. First, the DT of the set of points Q is constructed. If all the hole edges overlap with the Delaunay edges, then the CDT' is just the DT and the algorithm terminates. Otherwise, the edges in the DT that intersect hole edges are deleted. Because of this, some regions may need to be retriangulated. The area which is not triangulated is partitioned into polygons, called *difference polygons,* by introducing a set of auxiliary edges in such a way that each difference polygon contains exactly one hole edge. The CDT of each difference polygon is constructed via Lee and Lin's [LL] algorithm and the new edges are added to the previous Delaunay edges. Then the auxiliary edges are removed in certain order and the triangulation of certain adjacent regions needs to be modified.

Seidel's [S] method is based on the concept of Extended Voronoi Diagram and constructs the EVD using Fortune's sweep line technique. The CDT' is obtained by duality.

Our method is different, though it has many similarities to the one given by Jung [J1] and [J2], which is based on Wang and Shubert's method [WS]. The first step consists of finding the DT of all the points in Q. If the hole edges overlap with the Delaunay edges we have the CDT and the algorithm terminates. Otherwise we proceed as follows. Instead of deleting a Delaunay edge that intersects an h_edge (hole or boundary edge), as in Jung's method, we replace the edge (ab) by edges ai_a and $i_b b$ where i_a (i_b) is the closest point to vertex a (b) on edge ab that intersects an h-edge. As a result of this operation a new set of points is introduced. These points are called *E_points,* whereas the original points are referred to as *S_points*. All edges outside the boundary of P or inside the holes are deleted. Then the algorithm selects an h_edge at a time and deletes all the E_points on it as well as all the edges incident to these E_points. The resulting polygon is called the *CUT of the line.* We modify the CUT so that it satisfies some additional properties (which we define later on) and call it the *MCUT of the line.* A CDT for the MCUT is constructed via Lee and Lin's [LL] algorithm and then the CDT is transformed into another CDT that satisfies some additional properties. This procedure is applied to each h-edge with E_points on it. We claim that the resulting edges form a CDT and that procedure takes $O(n \log n)$ time..

The main difference between our procedure and Jung's procedure is that in our procedure each edge added inside an MCUT satisfies the circle property not only for the vertices in the MCUT, but also for the points in set Q. For the difference polygons, this is not necessarily true, i.e., the property is satisfied inside the difference polygon but not necessarily outside it. This is why he needs to merge adjacent difference polygons after deleting auxiliary edges. Because of this property we say that MCUTs are more natural than difference polygons. The implication of this property of MCUTs is that our algorithm reduces the CDT problem to a set of line intersection problems plus finding Delaunay triangulations of several simple polygons plus finding a Delaunay triangulation of a set of points. This is the first step in reducing the CDT problem to finding a DT of sets of points plus solving other simple problems. This is important since such a result would imply the "direct equivalence" of the DT and CDT problems. The interesting point is that some of the probabilistic analyses for the DT problem may translate directly to the CDT problem. Also, our algorithm is based on partitions and exploit useful properties of constrained Delaunay triangulations. This is similar in nature to the results of Guibas and Stolfi [GS] for finding a DT for a set of line segments.

In section II we define some terms and prove some basic properties for circles which will be used thought this paper. The algorithm is presented in section III. In section IV we establish correctness and in section V we show that the algorithm takes O(n log n) time.

II. DEFINITIONS AND BASIC CIRCLE PROPERTIES

As defined in the introduction, let P be a convex polygon, and let H be a set of pairwise disjoint simple polygons, called *holes* (note that a hole has nonzero area), defined inside P. Let D be a set of points inside I_P - R_H, let $Q = D \cup V(P) \cup V(H)$, and let n be the cardinality of set Q. An edge of a polygon representing a hole is called a *hole edge,* and an edge of P is called a *boundary edge.* A boundary edge, hole edge or part of a hole or boundary edge is called an *h_edge.* Edges and triangles resulting from a DT or CDT are called *Delaunay edges (d_edges) and Delaunay triangles (d_triangles).* Deleting zero or more edges from a triangulation results in an *s_triangulation* (or sub-triangulation).

The (infinite length) line that passes through points a and b ($a \neq b$) is referred to by *line(ab),* and the line that starts at point a, passes through b and continues to infinity is referred to as *halfline(ab).* Let p be a point and ab be a line segment. We say that point $p \in [a, b]$ if point p is a point in line segment ab. We say that point $p \in (a, b)$ if $p \in [a, b]$, $p \neq a$ and $p \neq b$. Let abc be a triangle. Then, *cir(abc)* is the circle passing through points a, b and c. The prefix R_<≠> attached to the name of a closed curve or polygon is used to represent the region inside and including that closed curve or polygon. The prefix *V_* (*E_*) attached to the name of a polygon is used to represent the vertices (edges) of the polygon. Let *B(c) (R(c))* represent the boundary (boundary and the region inside) of a polygon or a closed curve c. Note that R_cir(abc) = R(cir(abc)) and cir(abc) = B(R(cir(abc))). An edge is said to be a *crossing edge for triangle abc* if it intersects cir(abc) at two points, but does not intersect any side of triangle abc, unless it is a line that overlaps with line ab, bc, or ca. An h_edge of P is said to be a *passing edge for triangle abc* if it is a crossing edge for triangle abc. Let abc be a triangle such that no point in the interior of $R(abc)$ is inside a hole or outside the boundary of P. Let ef be an h_edge. We define R_cir_ignore(abc, ef) as the region (closed) bounded by the edge ef and cir(abc) that does not contain all of the three points a, b and c, if ef is a passing edge; otherwise it is defined as $\varnothing$. We define

$$\text{cons\_cir}(abc) = B(\text{R\_cir}(abc) - \bigcup_{ef \text{ is a passing edge for } abc} \text{R\_cir\_ignore}(abc, ef)).$$

Note that R_cons_cir(abc) consists of those points p inside and on cir(abc) visible from the center of triangle abc if we consider a passing edge as a wall (the undotted region inside the circle in figure 1). Similarly for circle C and a point $b \in R(C)$ we define cons(C, b) to be the boundary of the region containing all points $p \in R(C)$ which are visible from point b when considering any h_edge intersecting $R(C)$ as a wall.

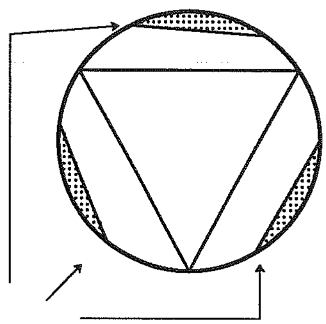


Figure 1: Passing edges.

Let C be a circle and let ab be a line that intersects circle C at the two distinct points $a \neq b$. We say that line ab *splits* $R(C)$ into two regions R_1 and R_2, with line ab belonging to both the regions. The circle centered at point t that passes through points a and b is denoted by *cirl(ab, t)*. A *crossing edge* for edge ab and point t is an edge which intersects cirl(ab, t) at two points but does not intersect edge ab unless it is a line that overlaps with line ab. We define cirl_ignore(ab, t), cons_cirl(ab, t), R_cirl_ignore(ab, t) and R_cons_cirl(ab, t) in the obvious way (note that in this case visibility is defined with respect to center of the line and the only points visible are points in I_P-R_H). Each point in the set Q is referred to as an *S_point* and it is represented in all figures by filled-in circles. The points introduced by the algorithm are referred to as *E_points* and are represented by non filled-in circles. An E_point on the h_edge st is referred to as an *E_point(st)*. We call an edge *SS_edge (EE_edge)* if its end points are two S_points (E_points). An edge is called an *SE_edge* if one of its end points is an S_point and the other is an E_point. We say that line ab satisfies the *circle property* if there is a circle C passing through a and b such that the set of all the points in $R(C)$ which are either visible from a or b (remember that h_edges are considered as walls) does not contain S_points, but may contain E_points. We say that a point m satisfies the circle property with respect to h_edge st, if edge mh satisfies the circle property, where h is the orthogonal projection of m on st.

The following three propositions give important properties of circles passing through two fixed points which will be used throughout our proofs.

Proposition 1: Let ab be a line segment that intersects circle C at points a' and b', and let c be any point which is not colinear with ab. Line ab splits $R(C)$ into regions R_1 and R_2. Then, $R_1 \subseteq$ R_cir(abc) and/or $R_2 \subseteq$ R_cir(abc).

Proof: Since ab intersects C at two points and $[a, b] \subseteq$ R_cir(abc), it must be that $R(C) \cap$ R_cir(abc) $\neq \emptyset$ and there are points on C which are inside R_cir(abc) (see figure 2).

Therefore, either $R(C) \subseteq$ R_cir(abc), or R_cir(abc) - $R(C) \neq \varnothing$ and $R(C)$ - R_cir(abc) $\neq \varnothing$. In the former case we know that $R_1 \subseteq$ R_cir(abc) and $R_2 \subseteq$ R_cir(abc), and the result follows. In the latter case it must be that C and cir(abc) intersect at two points. Let d and e be such points. Let de split C into two arcs, C_1 and C_2. Either $C_1 \subseteq$ R_cir(abc) and each point in C_2-$\{d, e\} \not\subseteq$ R_cir(abc), or $C_2 \subseteq$ R_cir(abc) and each point in C_1-$\{d, e\} \not\subseteq$ R_cir(abc). Since a' and b' are in R_cir(abc) and on circle C, they must both be in either C_1 or C_2. Therefore, all points in $C \cap R_1$ or $C \cap R_2$ are inside R_cir(abc), which implies that $R_1 \subseteq$ R_cir(abc) or $R_2 \subseteq$ R_cir(abc).

□

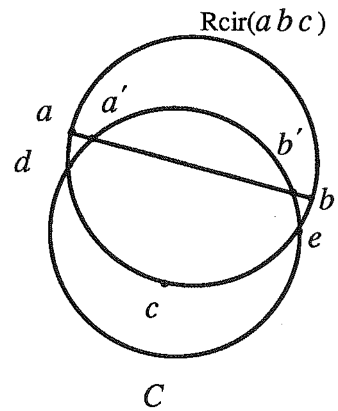


Figure 2: Proposition 1.

Proposition 2: If two circles D and D' intersect at two points a and b where line ab splits $R(D')$ into two regions R_1 and R_2, then $R_1 \subseteq R(D)$ and/or $R_2 \subseteq R(D)$.
Proof: The proof follows by substituting R_cir(abc) by $R(D)$, the $R(C)$ by $R(D')$, and line de by line ab in proposition 1.

□

Definition: Let ab be a line intersecting circle C at two points. Let c be a point not on line ab. Line(ab) divides the plane into two regions R_1 and R_2, and also divides $R(C)$ into two regions C_1 and C_2, where $C_1 \subseteq R_1$ and $C_2 \subseteq R_2$. If point $c \in R_1$, we use $C_{+c\,|ab}$ to represent $B(C_1)$ and $C_{-c\,|ab}$ to represent $B(C_2)$. For triangle abc we define similarly $cir_{+c\,|ab}(abc)$ and $cir_{-c\,|ab}(abc)$.

Proposition 3: Given two different circles C and C' both passing through points a and b and given point c not on line ab, then either $C_{+c\,|ab}$ is inside $C'_{+c\,|ab}$ or $C'_{+c\,|ab}$ is inside $C_{+c\,|ab}$.

Proof: Immediate from proposition 2.

□

III. Algorithm FIND_CDT.

In this section we present our algorithm, FIND_CDT, that generates the CDT for Q restricted by P and H. In subsequent sections we show that our algorithm generates a CDT for Q restricted by P and H in $O(n \log n)$ time.

Let E' be a triangulation for $Q \cup Q_e$ (where Q_e is a set of E_points) restricted by P and H. We say that E $\subseteq E'$ is an *s_triangulation* for $Q \cup Q_e$ restricted by P and H. We use E throughout the algorithm to represent an s_triangulation and we use E_edge to represent an edge in set E. Algorithm FIND_CDT consists of the following three steps.

Step 1: Construct the DT for Q. All the Delaunay edges just introduced are called *o_edges (original edges).* For each o_edge, e, let T_e be the set of triangles in the DT for Q having edge e as one of it sides. The triangles in set T_e are said to be *associated with edge* e. The set E consists of all the hole and boundary edges plus all the o_edges that do not completely overlap with a hole or boundary edge. Clearly, all edges in E are SS_edges. Note that set E might not be a triangulation or an s_triangulation for Q because some o_edges may cross hole and/or boundary edges.

Step 2: Invoke procedure DT_CE to transform E into an s_triangulation for $Q \cup Q_e$ restricted by P and H, where Q_e is a set of E_points. Procedure DT_CE considers each Delaunay edge at a time. When considering Delaunay edge ab the procedure checks if it crosses a hole or a boundary edge. When this is the case, the algorithm finds e_a (e_b), the hole or boundary edge that crosses edge ab at a point closest to a (b). Let i_a and i_b be the crossing points. An E_point is generated at i_a and if $i_a \neq i_b$ then another E_point is generated at i_b. Introducing an E_point, d, on an h_edge ef, has the effect of replacing h_edge ef by h_edges ed and df in E. When an E_point is deleted, it has the opposite effect. The o_edge ab is replaced by edges $a i_a$ and $b i_b$ (which will be referred to as *oc_edges)* in E. The triangles associated with edge ab are said to be *associated* with both of these new oc_edges. Once we have performed the above operation on all o_edges, we delete all the o_edges and oc_edges which are completely inside a hole or outside P, and delete each E_point with only h_edges incident to it. Clearly, after this step set E consists of SS_edges, SE_edges and EE_edges. All the o-edges are SS_edges; all the oc_edges are SE_edges; and the h_edges can be SS_edges, SE_edges or EE_edges.

Later on we show that after step 2 (i) - (iii) hold, and (i) - (iii) are loop invariants for step 3.

(i) Set E is an s_triangulation for $Q \cup Q_e$ restricted by P and H, where Q_e is the set of E_points.

(ii) Every non h_edge in E satisfies the circle property.

(iii) Every face with at least 4 vertices has at least one E_point on it.

In the next section we establish correctness from these loop invariants as well as with other facts that we establish later on.

Step 3: For each hole and/or boundary edge ab which is currently partitioned by E_points, we perform the following operations. First we delete all the E_points currently partitioning line ab together with all the edges incident to them (these edges are oc_edges, i_edges and g_edges [i_edges and g_edges are defined later on]). This operation is performed by procedure GET_CUT(ab) and it generates the polygon called the *CUT of line ab* or simply *CUT(ab)*. The dashed lines in figure 3 are the edges deleted for line ab when invoking procedure GET_CUT(ab). If we find a CDT for CUT(ab) it edges may not satisfy the circle property with respect to Q (loop invariant (ii)). This is why we transform the CUT into an MCUT.

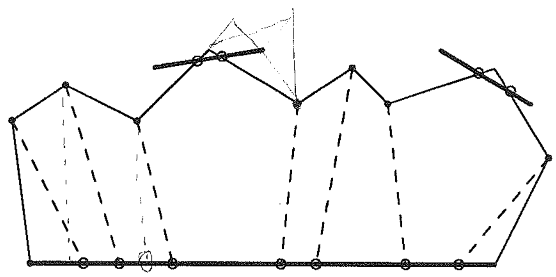
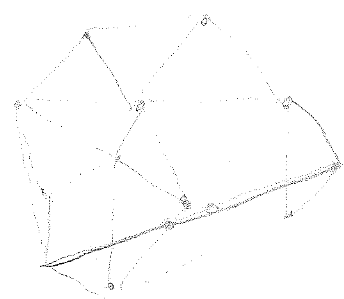


Figure 3: Cut of line ab.



We apply procedure GET_MCUT(ab). The procedure transforms the polygon CUT(ab) into the polygon called the *MCUT of line ab* which is denoted by *MCUT(ab)*. This procedure deletes some SE_edges (in lemma 1 we show that these SE_edges must be oc_edges) from E, and adds to E a new type of edge which we call *g_edges*. Every g_edge is an SE_edge which is perpendicular to the h_edge where its E_point is located. Let us now formally define procedure GET_MCUT.

```
Procedure GET_MCUT(ab);
begin
 let MCUT(ab) be CUT(ab);
 while there is an SE_edge which is not a g_edge on E_MCUT(ab) do
  begin
   Let se be an SE_edge which is not a g_edge on E_MCUT(ab);
   Let uv be the hole or boundary edge partitioned by E_point e;
   if s can be orthogonally projected on uv without intersecting another edge then
     let h be the projection of s on uv; /* sh is perpendicular to uv */
     if sh is inside cir(t), for some triangle t in the set of triangles associated with edge se then
          add E_point h (if not already there) to line uv, modify the corresponding
          edge which is in E_MCUT(ab) and is part of uv to extend to E_point h,
          and add g_edge sh to E and E_MCUT(ab).
     endif
   endif
   Delete SE_edge se from E and E_MCUT(ab);
   Delete E_point e if all the edges incident to it are h_edges.
  end;
end;
```

Let W be a copy of polygon MCUT. Change all the E_points in W to S_points and label all edges as boundary edges. We construct a CDT for W by invoking procedure XCDT (XCDT is any procedure that constructs the CDT of a given polygon without holes, e.g., the $O(n \log n)$ algorithm developed by Lee and Lin [LL] that is based on Chazelle's [C] divide and conquer partitioning rule for polygons). We add all the newly generated d_edges in the CDT for W to set E after transforming the CDT into another CDT without n_edges (EE_edges with endpoints on different h_edges). This is done by replacing n_edges by either SS_edges or SE_edges (later on we show that this is always possible) Later on we show that this is always possible, in part because the g_edges are perpendicular to the h_edge where its E_point is located. The replacement of n_edges is extremely important as otherwise the CDT edges introduced by procedure XCDT might not satisfy the circle property with respect to Q (loop invariant ii). After applying the above transformation it is simple to see that the d_edges in the CDT for W when added to set E could be: SS_edges which we call a_edges, or SE_edges which we call i_edges. The a_edges will end up in the final CDT; and the i_edges will eventually be deleted in procedure GET_CUT and GET_MCUT (actually only in GET_CUT(lemma 1)).

END OF PROCEDURE

Our algorithm FIND_CDT is summarized below.

```
procedure FIND_CDT
 Apply DT on Q and let E be the set of d_edges just introduced;
 Add all the hole and boundary edges to E;
 Apply routine DT_CE to transform E into an s_triangulation;
 for each hole or boundary edge ab which is partitioned by an E_point do
   begin
     find CUT(ab) and MCUT(ab);
     construct W from MCUT(ab);
     construct CDT of polygon W by invoking procedure XCDT; /* use Lee and Lin's procedure [LL] */
     after modifying the CDT for W so that no n_edges appear, add the resulting edges to E;
   end.
  return(E);
end of procedure
```

IV. Correctness of Algorithm FIND_CDT.

In this section we establish correctness, i.e., we show that for every problem instance algorithm FIND_CDT constructs a constrained Delaunay triangulation (theorem 1). This theorem is based on lemma 7 where loop invariants i - iii for procedure FIND_CDT are established, and lemma 1 where we show that when adding a g_edge at least one oc_edge is deleted. To prove lemma 7 we use lemma 1 where we show that only oc_edges are deleted by procedure GET_MCUT; lemma 2 where we show that g_edges satisfy the circle property; lemma 5 where we show that for each MCUT constructed by XCDT, there is a CDT without n_edges; and lemma 6 where we show that the d_edges in the CDT for an MCUT satisfy the circle property when considering all points in Q.

Definition: Suppose that in the process of constructing MCUT(st) from CUT(st) we find oc_edges np and $n'p'$ with E_points partitioning the hole or boundary edge $s't'$. Suppose that we remove SE_edge np ($n'p'$) and stop at S_point m (m'), such that point m (m') satisfies the circle property with respect to edge $s't'$ (figure 4). We call m (m') guard point of t' (s') for line $s't'$ or simply say m is g_point(t', $s't'$) and m' is g_point(s', $s't'$). Lines mh and $m'h'$ are called guard edges of line $s't'$ (g_edge(t', $s't'$) and g_edge(s', $s't'$) or in general g_edge). Note that it is possible for an h_edge $s't'$ to have less than two guard points, and if h_edge $s't'$ appears several times on the MCUT of an edge st, then it may have more than two guard points.

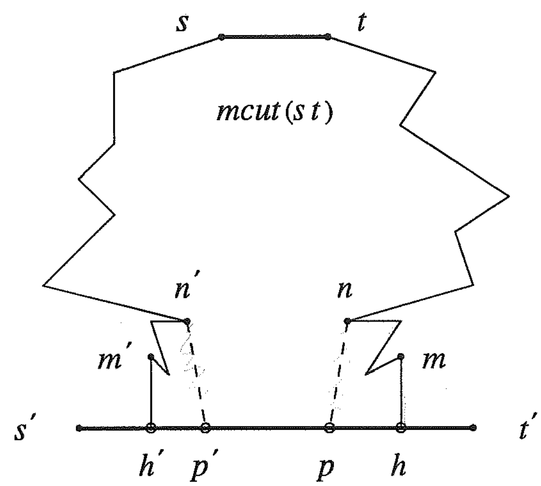


Figure 4: g-points.

Lemma 1: All the SE_edges deleted by procedure GET_MCUT are oc_edges and for each g_edge added at least one oc_edge is deleted.

Proof: Since the only type of SE_edges which are not g_edges are oc_edges and i_edges, we only need to show that i_edges are never removed by procedure GET_MCUT. Consider now the i_edges. After triangulating an MCUT (figure 5 a and c), suppose that the resulting polygon is of the form given in figure 5 (b and d).

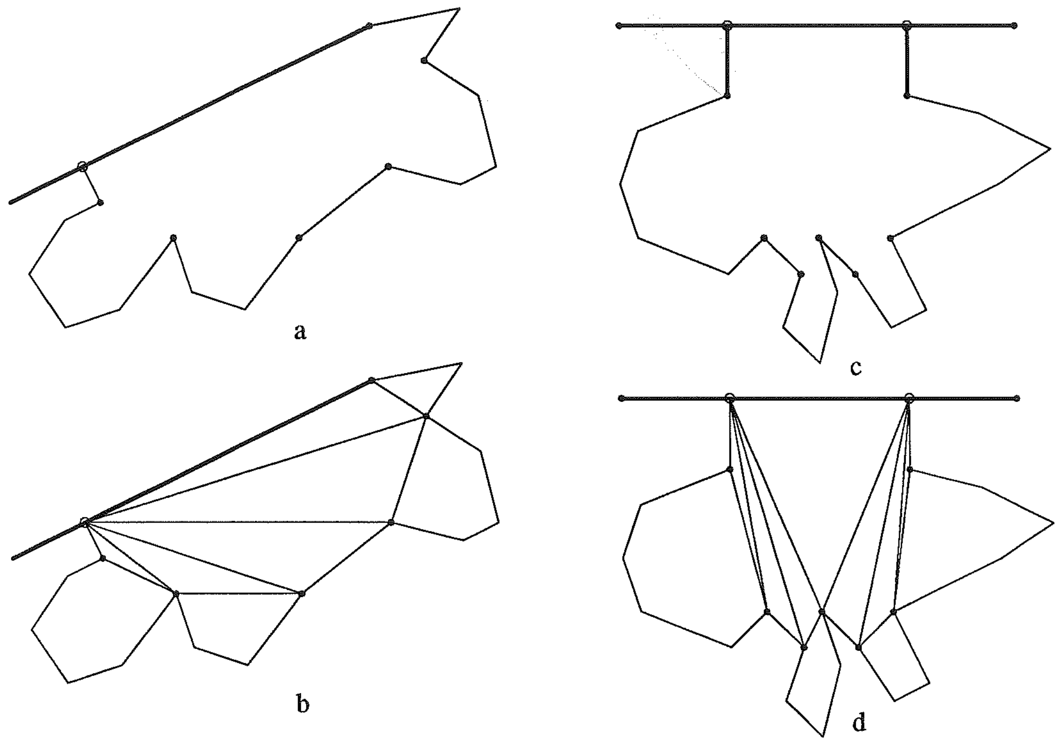


Figure 5: Proof of lemma 1.

It is simple to show that each i_edge introduced at step 8 is inside a polygon, which we call *bounding polygon for i_edges,* formed by SS_edges, h_edges and g_edges; and all the edges of the bounding polygon are i_edges. Since procedure GET_MCUT(ab) never deletes SS_edges nor g_edges, and when it removes an h_edge it also deletes an SE_edge which is not a g_edge, it then follows that no i_edges will ever be removed by GET_MCUT(ab). Since at each iteration at least one oc_edge is deleted and at most one g_edge is introduced, then for each g_edge added at least one oc_edge is deleted. This competes the proof of the lemma.

□

Lemma 2: Let mh be a g_edge introduced by procedure GET_MCUT(ab). Then edge mh satisfies the circle property.
Proof: Clearly, it is only required to show that there is a circle, C_1, such that the set of points in $R(C_1)$ visible from m and h may contain E_points but may not contain S_points. From procedure GET_MCUT and lemma 1, we know that the edge deleted when edge mh was added is an oc_edge. This oc_edge is a part of a d_edge (say edge mf), where mf intersects the h_edge which is partitioned by the E_point h. Let C be the circle that procedure GET_MCUT calls cir(t) when adding line mh. Clearly C passes through m and includes f, and the set of all

points visible from m and inside C contains no S_points. From procedure GET_MCUT we know that circle C includes completely line mh. Continue line mh until it intersects C, let us say that it intersects it at point k. Let o be the center of circle C, and let me be tangent to C at m (see figure 6). Let the intersection of the bisector of mh with line om be point o_1. Clearly, the circle C_1 with center o_1 and radius o_1h, passes through m and h. Since o_1m is perpendicular to line me, me is a tangent line to both C and C_1 at point m. Thus C_1 is completely inside C. Each point in C_1 which is visible from m is also in C and since C_1 is inside C, it is visible from m in C. Therefore, edge mh satisfies the circle property. This completes the proof of the lemma.

□

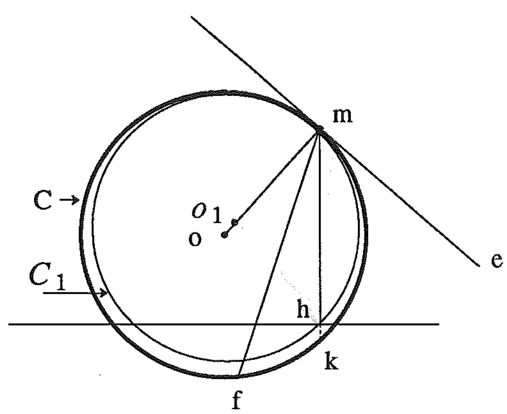


Figure 6: Proof of lemma 2.

Lemma 3: Let ae be an oc_edge generated by procedure DT_CE then edge ae satisfies the circle property.

Proof: Line ae is a part of a d_edge af. Since line af is a d_edge, we know it satisfies the circle property. Using arguments similar to the ones in the proof of lemma 2, it is simple to show that edge ae also satisfies the circle property. This completes the proof of the lemma.

□

We now show (lemma 5) that for every MCUT there is a CDT without n_edges. To prove this we need some basic properties of right angles (lemma 4) and additional definitions.

Definition: Vertex b is defined as the *focal point* of angle abc. We say that point v is visible from angle abc if there is a line vb that divides angle abc and does not overlap with line ab or bc. We say that angle abc is visible from angle $a'b'c'$ (or vice-versa) iff the focal point b' is visible from angle abc and the focal point b is visible from angle $a'b'c'$. We say that point v is visible from angles abc and $a'b'c'$ if there is a line segment vb that divides angle abc and does not intersect line segments ab, bc, $a'b'$ or $b'c'$; and there is a line segment vb' that divides angle $a'b'c'$ and does not intersect line segments ab, bc, $a'b'$, or $b'c'$.

Lemma 4: Consider the two different right angles abc and $a'b'c'$. Assume that these angles do not intersect, except when points a and a' coincide, or points c and c' coincide. Suppose that angle abc is visible from angle $a'b'c'$, and let C be any circle passing through b and either passing through b' or just having point b' inside it. Then a point $p \in \{a, c, a', c'\}$ is inside C, or all four points in $\{a, c, a', c'\}$ are on C.

Proof: Since angles abc and $a'b'c'$ are visible, then halfline(ba) intersects either halfline($b'a'$) or halfline($b'c'$). Similarly, halfline(bc) intersects halfline($b'a'$) or halfline ($b'c'$). Assume without loss of generality that halfline(ba) intersects halfline($b'a'$). Let a'' (c'') be the intersection point of halfline(ba) and halfline($b'a'$) (halfline(bc) and halfline($b'c'$)) (see figures 7 and 8). The polygon $bc''b'a''$ is a quadrangle. Since angles $a''bc''$ and $a''b'c''$ are 90 degrees, there is a circle, C, that passes through these four points (see figure 7 and 8). Clearly, at least one of a, or a' (b, or b') is inside of C, or both a and a' (c and c') are on C. Also, halfline bb' divides circle C into two regions R_1 and R_2, where R_1 contains point a and/or point a', and R_2 contains point c and/or c'. Applying proposition 1, we know that any circle passing through b and b' is identical to C, or it either contains region R_1 or R_2. Therefore, any circle passing through b and b' has a point $p \in \{a, c, a', c'\}$ inside it, or all points in $\{a, c, a', c'\}$ are on it. Any circle C' passing through b and having b' inside it will intersect C at two points (one of which is b), or will intersect C at b and includes C. Applying arguments similar to the ones in the previous case, it is simple to show that a point $p \in \{a, c, a', c'\}$ is inside C'.

□

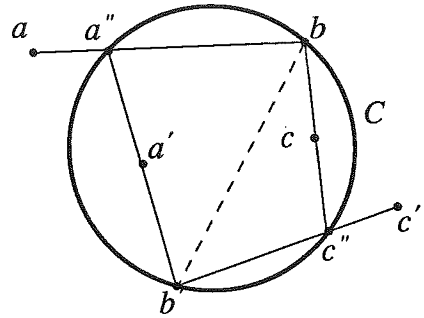


Figure 7: a' and c are inside C.

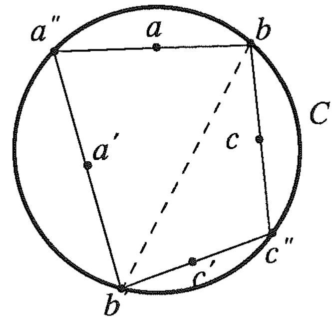


Figure 8: a, a', b and b' are inside C.

Lemma 5: For each MCUT constructed in FIND_CDT, there is a CDT without n_edges.

Proof: The proof is by contradiction. Suppose MCUT($s\,t$) has the property that all its CDTs have at least one n_edge. Let us now consider the CDT with the least number of n_edges. Clearly, there is at least one n_edge. Let edge $b\,b'$ be an n_edge in MCUT($s\,t$). From the algorithm it is simple to verify that each E_point in an MCUT is a vertex of V_MCUT($s\,t$) and is a focal point of a 90 degree angle. Let $a\,b\,c$ and $a'b'c'$ be any two such angles where points a, b, c, a, b' and c' belong to V_MCUT($s\,t$). One side of each right angle is an h_edge and the other side is a g_edge. Therefore, at least one of a and c (a' and c') is an S_point. In order for the CDT to contain the n_edge $b\,b'$ it must be that the two angles are visible and that there is a circle C that passes through b and b' such that cons(C, b) contains no S_points inside it, but may contain E_points. By construction it cannot be that the sides of these angles intersect, except that points a and a', or b and b' may coincide (or a and b' and a' and b). Since the conditions of lemma 4 are satisfied, we know every circle C passing through points b and b' has a point $p \in \{a, c, a', c'\}$ inside it; or all points in $\{a, a', c, c'\}$ are on circle C. Any such point, inside or on circle C, is visible from b so it is inside or on cons(C). In the former case we know that when the CDT of MCUT($s\,t$) was obtained, all the points in V_MCUT($s\,t$) were treated as solid points. Therefore, a CDT of MCUT($s\,t$) cannot contain $b\,b'$ as a d_edge. This contradicts the fact that $b\,b'$ is an n_edge. In the latter case it must be that a and a' coincide; and c and c'

coincide Therefore, b, b', a and c are on the circle and they form a quadrangle. Since at least one of a or c is an S_point, a triangulation for the MCUT($s\,t$) can be obtained by adding edge $a\,c$ and deleting edge $b\,b'$. This contradicts the fact that the CDT of the MCUT($s\,t$) that we started from has the least number of n_edges. This completes the proof of the lemma.

□

Now we are ready to prove a fundamental property of our algorithm. This property is that every d_edge in a CDT of an MCUT is also a d_edge of the whole problem.

Lemma 6: Let R be polygon MCUT($s\,t$) and assume that each non h_edge on it satisfies the circle property. Let X be a CDT of R without n_edges. Let $a\,b\,c$ be a triangle in X. Then cons_cir($a\,b\,c$) does not contain any point of Q but may contain E_points.
Proof: Suppose not. Assume there is a point $f \in Q$ which is inside cons_cir($a\,b\,c$) (i.e., f is visible from the center of triangle $a\,b\,c$ when considering the borders of P and the hole edges as walls). Triangle $a\,b\,c$ divides R_cons_cir($a\,b\,c$) in four regions, triangle $a\,b\,c$ and three other regions, each having a side of the triangle as its boundary edge. We refer to these regions as regions cc_region($b\,c$), cc_region($a\,b$) and cc_region($a\,c$) (see figure 9).

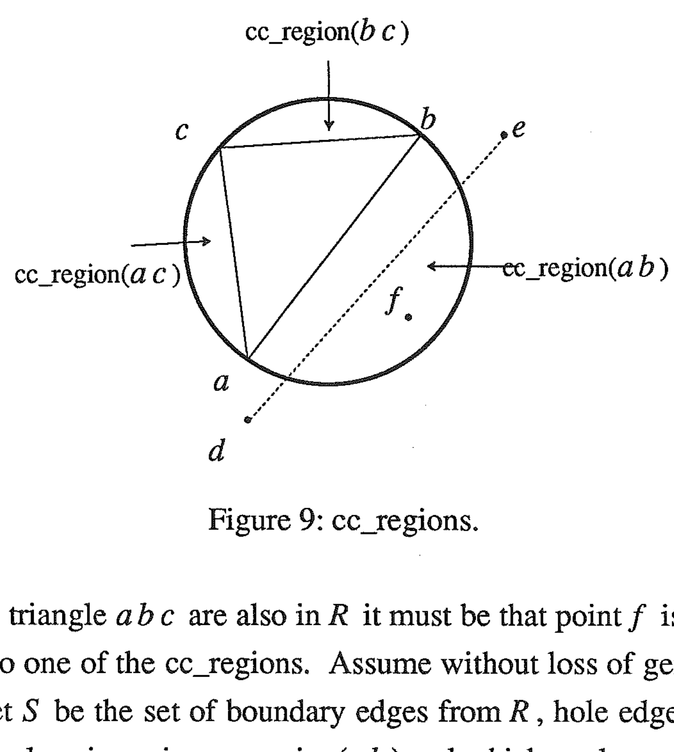


Figure 9: cc_regions.

Since all the points in triangle $a\,b\,c$ are also in R it must be that point f is outside triangle $a\,b\,c$, and point f belongs to one of the cc_regions. Assume without loss of generality that f belongs to cc_region($a\,b$). Let S be the set of boundary edges from R, hole edges, and boundary edges in P that are crossing edges in region cc_region($a\,b$) and which are located between line $a\,b$ and point f, i.e., point f is not located in the region formed by any one of these lines, line $a\,b$, and cir(abc). Since point f is in cons_cir(abc), it must be that set S does not contain a hole edge or an edge that is boundary of P. Since point f is outside polygon R, then there is at least one edge in S which is a boundary of R. Let de be one of such edges. Since de is a crossing edge, points d and e are not inside cir(abc). From the conditions of the lemma we know that de satisfies the circle property. Therefore, there is a point t such that cir(de, t) passes through the end points of line de and cir(de, t) does not contain any other point of Q. Since line de divides cir(abc) in two regions: R_1 containing f, and R_2 containing a, b and c (figure 9), then by proposition 1 cons_cir(de, t) contains either f or it contains a, b and c. Since X is a CDT of R without n_edges, we know that at least one of the points a, b or c is an S_point. This contradicts the fact that cons_cir(de, t) does not contain a point from Q inside it. Therefore, our assumption that f belongs to cons_cir(abc) is false and therefore no point in Q can belong to cons_cir(abc). This completes the proof of the lemma.

□

In the following lemma we prove three loop invariants for algorithm FIND_CDT which will help us establish correctness.

Lemma 7: Each time line 4 in procedure FIND_CDT is about to be executed set E satisfies (i) - (iii) below.

(i) Set E is an s_triangulation for $Q \cup Q_e$ restricted by P and H, where Q_e is the set of E_points.

(ii) Every non h_edge in E satisfies the circle property.

(iii) Every face with at least four vertices has at least one E_point vertex.

Proof: We prove by induction on k ($k \geq 1$) that the kth time line 4 is about to be executed (i) - (iii) hold.

Basis: We show that just before the first time line 4 is about to be executed (i) - (iii) hold. For convenience let us view algorithm DT_CE as the following procedure (note that this procedure is not O(n log n); however, it output is identical to the one generated by our procedure).

line 3

(3.1) Let E be the set of hole edges, boundary edges and d_edges introduced by procedure DT (note that if tw overlap, the d_edge is deleted);

for each pair of edges that intersect at a point other than their end points **do**

let edge ab and de intersect at point c (different from a, b, d and e);

add an E_point at c and replace edge ab (de) by lines ac (dc) and cb (ce);

endfor;

Delete all edges which are inside a hole or outside P;

(3.2) Delete each EE_edges which are not h_edges together with its E_points;

end of (3.1)-(3.2)

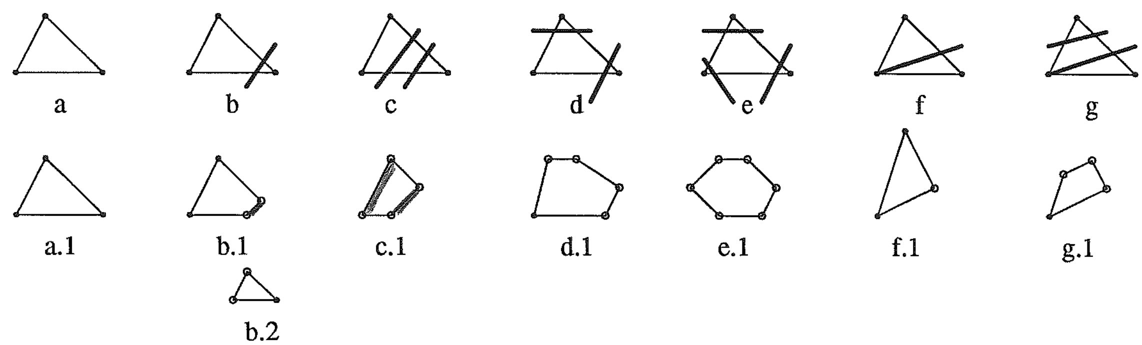


Figure 10: Proof of lemma 7.

Let us now prove that after step 3.1 set E satisfies (i) - (iii). It is simple to verify that after step 3.1 set E is an s_triangulation for $Q \cup Q_e$ restricted by P and H. Clearly, just after step 3.1 all the non h_edges are o_edges or sections of o_edges. Since all the o_edges are introduced in line 1 and they satisfy the circle property with respect to Q and by lemma 3 the circle property holds for each segment of an o_edge, then (ii) holds. Just after combining the hole and boundary edges with the d_edges in the DT for Q and replacing each crossing by four edges and an E_point, a hole or boundary edge either overlaps with a d_edge or it intersects at least one o_edge (see figure 10a-g, where the hole and boundary edges are represented by thick lines). Therefore, the faces resulting after these operations are shown in figure 10 (a.1, b.1-2, c.1, d.1, e.1, f.1, and g.1) and (iii) holds. It is simple to show that after deleting all edges inside a hole or outside P, (i) - (iii) hold. Also, all faces are of the form shown in figure 10 (a.1, b.1-2, c.1, d.1, e.1, f.1 and g.1) and just after step 3.1 all the E_points have exactly one oc_edge incident to it.

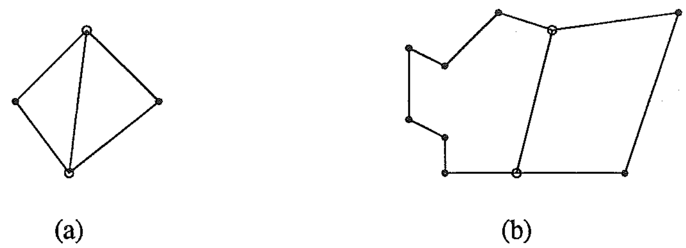


Figure 11.

Clearly, removing an EE_edge which is not an h_edge combines two faces together. Therefore, (i) and (ii) hold after line 3.2. Condition (iii) will not hold if two faces are combined and a face with at least four edges and no E_point is generated. This can happen if both of the initial faces have only two E_points and these E_points are adjacent to the EE_edge. Also, either both faces have three points (figure 11a), or at least one of the original faces must contain four points (see figure 11b). The case given in figure 11a does not arise since an E_point may not have two or more oc_edges incident to it nor it may join two different hole or boundary edges; and the case given in figure 11b does not arise since it would imply that at least one SE_edge in one of the faces must have been previously deleted. However, the procedure does not delete SE_edges inside these faces. So it must be that (i)-(iii) hold after line 3.2.

Induction Hypothesis: Suppose set E satisfies (i)-(iii) just before line 4 is about to be executed for the kth time.

Induction step: Set E satisfies (i)-(iii) just before line 4 is about to be executed for the kth time. Let us now consider the kth iteration. By the induction hypothesis we know that set E satisfies (i)-(iii). Procedure GET_CUT(ab) deletes a set of SE_edges which are not h_edges and deletes E_points on h_edges which are not endpoints of an SE_edge. The removal of all of these edges combines adjacent faces into a single face which we call polygon CUT(ab). Clearly, (i) and (ii) hold. Since we have not modified any face other than CUT(ab), E satisfies (iii) except possibly for face CUT(ab). Procedure GET_MCUT(ab) deletes some oc_edges and introduces additional g_edges (lemma 1) which satisfy the circle property (lemma 2). Therefore just after step 5, E satisfies (i)-(iii), except possibly for (iii) which might not hold for the face which we call MCUT(ab). As a result of executing steps 6-8 the face called MCUT(ab) is triangulated, the triangulation is slightly modified so that no n_edges appear and all the edges (which are called i_edges and a_edges) are added to E. By lemma 5 we know that it is always possible to transform a triangulation to one that does not contain n_edges. Therefore, (i) holds after these three steps. Since the conditions of lemma 6 hold, we know that all the new edges introduced in these three steps satisfy the circle property. Therefore, (ii) holds after the three steps. Since (iii) holds on all faces except possibly for the face called MCUT(ab) and we triangulate such face, it follows that (iii) hold after this step. Hence, (i)-(iii) hold true just before line 4 is about to be executed for the $k+1$st time. This completes the proof of the induction step and the lemma follows by induction.

□

The main theorem that establishes the correctness of procedure FIND_CDT is given below.

Theorem 1: Algorithm FIND_CDT constructs a CDT for Q restricted by P and H.
Proof: In the first three lines of procedure FIND_CDT E_points and oc_edges are introduced. We claim that each time loop 4-8 is executed the number of oc_edges plus the number of E_points decreases. The reason for this is that oc_edges are not introduced in the loop, for each oc_edge we delete in procedure GET_MCUT we introduce at most one E_point (lemma 1) and at least one E_point is deleted by procedure GET_CUT. Therefore, since the number of oc_edges and E_points introduced in lines 1-3 is finite, after a finite number of iterations of loop 4-8 there will be no E_points. From lemma 7(i), we know that the last time line 4 was executed E was an s_triangulation for $Q \cup Q_e$ restricted by P and H. By lemma 7 (iii) we know that each of the faces of E with more than four nodes on it must have an E_point. But when the algorithm terminates $Q_e = \emptyset$, i.e., there are no E_points. Therefore, none of the faces in E has more than three vertices. I.e., E is a triangulation for Q restricted by P and H. By lemma 7 (ii) we know that each E_edge which is not a hole or boundary edge in the triangulation E satisfies the circle property. Therefore, triangulation generated by algorithm FIND_CDT, E, is a CDT for Q restricted by P and H. This completes the proof of the theorem.

□

V. Complexity of Algorithm FIND_CDT.

In this section we establish the time complexity bound of O(n log n) for procedure FIND_CDT. To prove the result we establish three lemmas that relate edges in different MCUTs at different times.

It is simple to see that steps 1-3 take O(n log n) time. Let us break the loop in two parts. The first part (lines 4 - 6) take time O(k_i + er_i) where k_i is the number of points in the resulting MCUT polygon and er_i is the number of SE_edges removed. The second part (lines 7 and 8) takes time O(k_i log k_i) (remember that Lee and Lin's algorithm takes O(k log k) time when the polygon has k vertices). Since the number of S_points is n and no three consecutive vertices in the MCUT can be E_points, it must be that k_i is O(n). Therefore, the overall time complexity for both parts is O($\sum$ (k_i log n) + $\sum er_i$). We can establish our time complexity bound by showing that $\sum k_i$ is O(n) and $\sum er_i$ is O(n).

In lemma 9 we show that if consecutive S_points in an MCUT had an SE_edge removed when constructing the CUT, then in the CDT for the MCUT that we construct no two of these consecutive points can have an SE_edge to another E_point (see figure 12). This lemma together with other facts are then used in Theorem 2 to establish that $\sum k_i$ is O(n). If we show that each S_point appears in a constant number of MCUTs, then we can easily establish that $\sum er_i$ is O(n). However, it may be that some S_point is in a large number of MCUTs. In lemma 10 we essentially establish that every two times an S_point appears in an MCUT, at least one SS_edge is introduced. Since we prove in theorem 2 that the number of SS_edges introduced is O(n), it then follows that $\sum er_i$ is O(n).

Lemma 8: Let st be an h_edge and let points a and b belong to V_MCUT(st). Assume there are E_points h and h' belonging to R_MCUT(st) such that for triangles abh and abh' we know that $R(abh) \subseteq$ R_MCUT(st) and R(abh') $\subseteq$ R_MCUT(st). Furthermore, assume that S_1 = R_cir$_{+h\,|\,ab}(abh) \subseteq$ R_cir$_{+h'\,|\,ab}(abh')$ = S_2. Then, R_1 = R_cons_cir$_{+h\,|\,ab}(abh) \subseteq$ R_cons_cir$_{+h'\,|\,ab}(abh') = R_2$

Proof: We prove the lemma by contradiction. Suppose there exists a point $p \in R_1$ such that $p \notin R_2$. Since $S_1 \subseteq S_2$, $p \in R_1$ and $p \notin R_2$ there exists a passing edge cd for triangle abh' such that $p \in$ R_cir_ignore(abh', cd). Since $S_1 \subseteq S_2$ and $p \in R_1$, it must be that edge cd is also a passing edge for triangle abh. But then $p \in$ R_cir_ignore(abh, cd) and by assumption $p \notin$ R_cons_cir(abh), a contradiction. This completes the proof of the lemma.

□

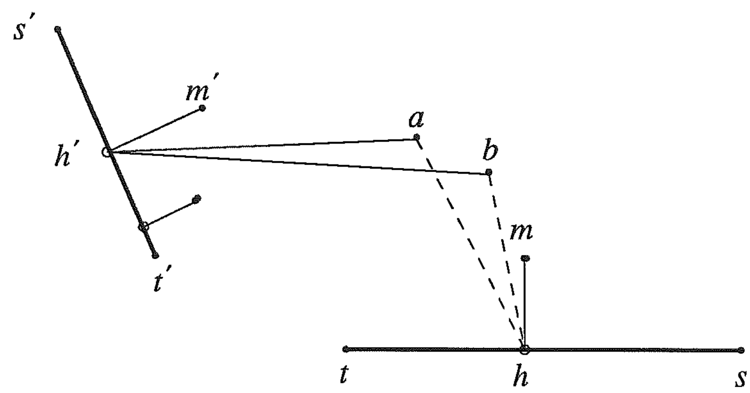


Figure 12: Proof of lemma 9.

Lemma 9: Assume two adjacent S_points a and b are connected directly by edges to point $h \in$ E_point(st) for h_edge st and suppose there is a g_edge mh (m is g_point(s,st)). In the CDT of MCUT(st) points a and b cannot be connected directly by edges to a point $h' \in$ E_point($s't'$) where $s't'$ is an h_edge with a section of it appearing as an edge in E_MCUT(st).

Proof: We prove this lemma by contradiction. Since h' is an E_point in MCUT(st), it must have a g_point. Let m' be g_point(s', $s't'$). Let $S_1 = \text{R\_cir}_{+h\,|ab}(abh)$, $S_2 = \text{R\_cir}_{+h'\,|ab}(abh')$, $R_1 = \text{R\_cons\_cir}_{+h\,|ab}(abh)$, and $R_2 = \text{R\_cons\_cir}_{+h'\,|ab}(abh')$. By proposition 2 we know that either $S_1 \subseteq S_2$ or $S_2 \subseteq S_1$. Therefore, the conditions of lemma 8 are satisfied, and we know that either $R_1 \subseteq R_2$, or $R_2 \subseteq R_1$. Let C be the larger of $cir_{+h\,|ab}(abh)$ and $cir_{+h'\,|ab}(abh')$. In each case we have two right angles mht and $m'h't'$ such that the focal point of one of these angles is inside or on circle C and the focal point of the other right angle is on the boundary of C. From lemma 4 either edge mh intersects line $m'h'$ or lines ht and $h't'$ intersect; or a point $p \in \{m, m', t', t\}$ is inside C. These lines cannot intersect at any point except at their end points, because st and $s't'$ are h_edges, $[m, h] \in$ R_CUT(st), and $m'h'$ is obtained by moving the E_point of an SE_edge ce, $ce \in$ E_CUT(st). Moving edge ce adds an extra region to CUT(st) and any g_edge introduced because of this operation to E_MCUT(st) is not inside R_CUT(st). None of points t', m', m and t can be inside C, otherwise, as we show in the following paragraph, they have to be inside cons(C) which contradicts our assumption.

In order to have one point e.g. point x inside C but not inside cons(C) there should exist a passing edge e dividing C into two regions c_1 and c_2 such that c_1 contains point x and c_2 contains points a and b, with the property that no edge can intersect edge e and connect any point of c_1 to a point of c_2. We prove that all points t', m', m and t if they belong to C they should belong to the c_2 region by showing for each of these points there is an edge which connects it to a point of region c_2. Assume these points belong to C. Points h and h' belongs to c_2 because of edges $h'a$ and ha. Point m belongs to c_2 because of edge mh, where h belongs to c_2. Similarly points t', m' and t belong to c_2 because of edges $t'h'$, $m'h'$ and th. This shows if any of points t',

m', m and t belongs to C it should belong to cons(C). This completes the proof of the lemma.

□

Lemma 10: Assume that in the process of executing the algorithm, we have the case where on MCUT of line st there are adjacent S_points a, b and c (a and c could be guard points), E_point h' on line $s't'$, E_point h on line st, and points a, b and c are connected by edges to h. Following the steps of the algorithm let's remove all E_points located on line st including point h and find the CDT of the MCUT of line st. Assume after this step point b is connected to h' where without loss of generality we assume that if there is another E_point q' on line $s't'$ such that it also is connected directly to b, going clockwise around b, bh' is before bq'. Because of line bh', b might appear in MCUT of line $s't'$. We prove each such additional appearance of solid point b on some MCUT will introduce at least one more SS_edge to the edges incident to b and for each such SS_edge introduced, point b can reappear on at most two other MCUTs.
Proof: From the arguments in the proof of lemma 7 we know that no edge of type EE where its end points belong to different h_edges exists. This together with the conditions given in the lemma we have figure 13 where b is connected directly to h. Assume E_points on line st are removed and the CDT of the MCUT of line st is obtained and there is an edge from b to h'.

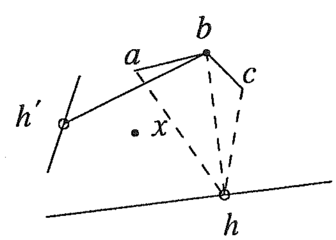


Figure 13: Proof of lemma 10.

By lemma 9, two adjacent solid points cannot be moved from one MCUT to another, so there cannot exist any edge connecting directly a or c to h'. Also since no edge of type EE where its end points belongs to different hole lines exists, the next spoke of point h' going clockwise around h' and after $h'b$ should be an edge connecting h' to a solid point x, where point x is not c. Since there is no other edge in clockwise order around h' and between $h'b$ and $h'x$, b has to be connected to x. Edge bx is type SS and is introduced in this step of the algorithm. To prove the second part of the lemma, note that each reappearance of point b on different MCUTs causes one SS_edge incident to b be added e.g. figure 14 where deleting bh and getting CDT of MCUT(st) causes SS_edges bx and by be introduced. Since bx and by can coincide, each SS_edge can contribute to at most two reappearances of an S_point in two different MCUTs.

□

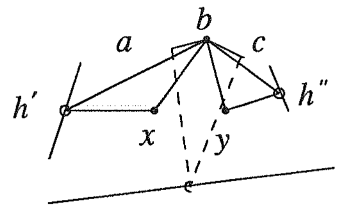


Figure 14: CDT of MCUT($s\,t$) causes SS_edges $b\,x$ and $b\,y$ to appear.

Theorem 2: Algorithm FIND_CDT constructs a CDT for Q restricted by P and H in $O(n \log n)$ time.

Proof: It is simple to see that steps 1-3 take $O(n \log n)$ time. Let us break the loop in two parts. The first part (lines 4 - 6) take time $O(k_i + er_i)$ where k_i is the number of points in the resulting MCUT polygon and er_i is the number of SE_edges removed. The second part (lines 7 and 8) takes time $O(k_i \log k_i)$ (remember that Lee and Lin's algorithm takes $O(k \log k)$ time when the polygon has k vertices). Since the number of S_points is n and no three consecutive vertices in the MCUT can be E_points, it must be that k_i is $O(n)$. Therefore, the total time complexity for both parts is $O(\sum (k_i \log n) + \sum er_i)$. Therefore, we can complete the proof of the theorem by showing that $\sum k_i$ is $O(n)$ and $\sum er_i$ is $O(n)$.

Before we prove these relations, it is convenient to first prove the following statements.

(i) The only place that oc_edges are introduced is in step 3.

(ii) Procedure GET_CUT($a\,b$) removes oc_edges, i_edges, and g_edges.

(iii) Procedure GET_MCUT($a\,b$) removes only oc_edges and for each g_edge it introduces it deletes an oc_edge (lemma 1).

(iv) The only place that i_edges are introduced is in line 8.

(v) SS_edges are never deleted after line 3.

(vi) Once an i_edge or a g_edge is removed, it will never be added again.

The proof of (i), (ii), (iv) and (v) is trivial and by lemma 1 (iii) holds. Let us now prove that (vi) holds. From (ii) and (iii) we know that the only place where i_edges and g_edges are removed is in procedure GET_CUT. When such a removal occurs the E_point is deleted and since E_points are never introduced after this step on line $a\,b$, it then follows that such an i_edge or g_edge will never be added again.

Since oc_edges are only introduced at step 3 (see (i) above), for each o_edge we introduce at most two oc_edges and the number of o_edges in DT of Q is $O(n)$, we know that the number of oc_edges is at most twice the number of o_edges which is $O(n)$. An oc_edge will be removed in CUT(ab) or in MCUT(ab). When it is removed in MCUT(ab) it may introduce a g_edge. Since the only place g_edges are introduced is in procedure GET_MCUT, the total number of g_edges is at most O(n). We say that an S_point appears in an MCUT(st) *directly* if an SE_edge incident to it was deleted by procedure GET_CUT(st). We say that it appears *indirectly* otherwise. When an i_edge is introduced (by step 8 of FIND_CDT) it must have been that its S_point appeared in MCUT(st) directly or indirectly. When the S_point appears directly in MCUT(st) then at least one oc_edge, g_edge or i_edge was deleted. Let us now analyze these cases separately.

Suppose that S_point p appears in MCUT(st) indirectly. Let sp and pt be the two edges in E_MCUT(st). Clearly, these two edges are SS_edges, h_edges or g_edges. Since p appears in MCUT(st) indirectly, then all the edges in the angle spt which were deleted at this step were oc_edges. When the algorithm introduces SE_edges (at step 8 of FIND_CDT) incident to p and located inside angle spt, we claim that if k of such edges are introduced, then at least (k/2)-1 SS_edges must be introduced in the angle spt. The reason for this is that we never introduce EE_edges (step 7 and 8 in FIND_CDT) and there can be at most two consecutive E_points in E_MCUT(st).

Suppose that an S_point b appeared directly in MCUT(st) because an i_edge was deleted. Let a and c be the nodes adjacent to b in V_MCUT(st). Suppose that a, b and c were joined to $h \in$ E_point(st) (the proof of the other cases is omitted since it is similar). Then from lemma 10 we know that an SS_edge is introduced when we construct the CDT of MCUT(st). If more than one i_edges incident to b are introduced, then a proof similar to the one in lemma 10 may be used to show that for each pair of them at least one SS_edge is introduced (at step 8 of FIND_CDT). The case when an S_point appears directly because of an oc_edge was deleted is treated as in the case of S_points that appear indirectly; and the case when an S_point appears directly because of a g_edge each side is treated as either an S_point that appears indirectly, or an S_point that appears directly and the edge deleted is an i_edge.

Therefore, the number of i_edges introduced is bounded by twice the number of SS_edges (which are at most O(n)) plus twice the number of g_edges (bounded by O(n)) plus twice the number of hole edges (which is bounded by O(n)). This together with the fact that the number of oc_edges is O(n) shows that $\sum er_i$ is O(n).

Let us now show that $\sum k_i$ is O(n). Each time we triangulate an MCUT with k_i nodes O(k_i) edges are introduced. The new edges introduced are SS_edges (a_edges), or SE_edges (i_edges), since there are no EE_edges (see the proof of lemma 7). Since a_edges are never removed and since i_edges will never be added after we delete their E_points, we know that $\sum k_i$ is bounded by the number of a_edges plus the number of i_edges introduced by the algorithm which we know is O(n). From previous arguments, it follows that FIND_CDT takes O(n log n)

time. This completes the proof of the theorem.

□

VI Discussion

We presented an O(n log n) algorithm to construct a constrained Delaunay triangulation for a simple polygon with interior points and holes. The main difference between our algorithm and the previous algorithms for this problem is that our algorithm reduces the problem to a set of line intersection problems plus finding Delaunay triangulations of several simple polygons plus finding Delaunay triangulations of several sets of points. This is the first step in reducing the CDT problem to finding a DT of sets of points plus solving other simple problems. This is important since such a result would imply the "direct equivalence" of the DT and CDT problems. The interesting point is that all the probabilistic analyses for the DT problem would directly translate to the CDT problem. Also, our algorithm is based on partitions and exploit useful properties of constrained Delaunay triangulations.

VII References

[C] Chazelle, B. M., "A Theorem on Polygon cuting with Applications," Proceedings of the 23rd IEEE Annual Symposium on Foundations of Computer Science, 1982, 339 - 349.

[Cw] Chew L., "constrained Delaunay Triangulations," Proceedings of the 3rd Annual Symposium on Computational Geometry, 1987, pp. 215 - 222.

[J1] Jung, D., "An Optimal Algorithm for Constrained Delaunay Triangulations," Proceedings of the 1988 Annual Allerton Conference on Communications, Control and Computing, pp. 83 - 84, October 1988.

[J2] Jung, D., "On Constructing the Constrained Delaunay Triangulation," Technical Report 100, Electrical, Computer and Systems Engineering Department, Rensselaer Polytechnic Institute, 1988.

[F] Fortune, S. J., "A Sweepline Algorithm for Voronoi Diagrams," *Algorithmica, 1987.*

[GS] Guibas, L. and Stolfi, J., "Primitives for the Manipulation of General Subdivision and the Computation of Voronoi Diagrams," *ACM Transactions on Graphics,* Vol. 4, No 2, pp 74 - 123, April 1985.

[L] Lawson, C. L., "Software for C_1 Surface Interpolation," in Mathematical Software III (J. R. Rice, Ed.), Academic Press, New York, 1977, pp. 161 - 194.

[LS] Lee D. T. and B. J. Schacter, "Two algorithms for Constructing Delaunay Triangulations," *International Journal of Computer and Information Sciences,* 9, 3, (1980), pp 219 - 242.

[LL] Lee D. T. and A. K. Lin, "Generalized Delaunay Triangulation for Planar Graphs," *Discrete Computational Geometry,* 1, (1986), 201 - 217.

[NF] Nielson G. N. and Richard Franke, "Surface Construction Based Upon Triangulations," *Surfaces in CAGD,* North-Holland Publishing Co., 1983, 163 - 177.

[PS] Preparata, F. P. and M. I. Shamos, *"Computational Geometry,"* Springer-Verlang, 1985.

[S] Seidel, R., "Constrained Delaunay Triangulations and Voronoi Diagrams with Obstacles," Computer Science Division, UC Berkeley, June 1989.

[SH] Shamos, M. I. and D. Hoey, "Closest Point Problems," Proceedings of the 16th Symposium on Foundations of Computer Science, October 1975, 151 - 162.

[WS] Wang C. and L. Schubert, "An Optimal Algorithm for Constructing the Delaunay Triangulation of a Set of Line Segments," Proceedings of the 3rd