Constrained Delaunay Triangulations

L. Paul Chew
Department of Math and Computer Science
Dartmouth College
Hanover, NH 03755

Abstract

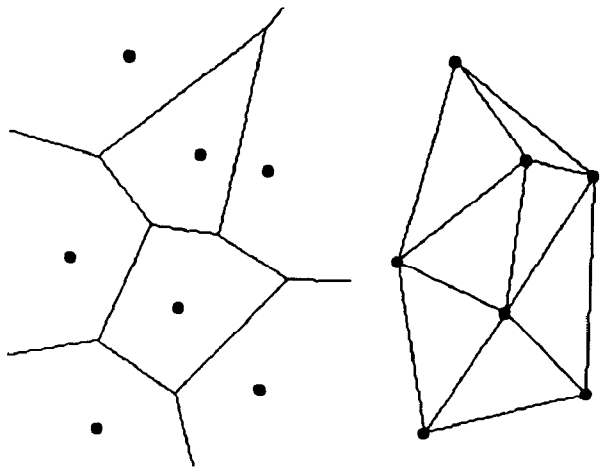
Given a set of n vertices in the plane together with a set of noncrossing edges, the *constrained Delaunay triangulation* (CDT) is the triangulation of the vertices with the following properties: (1) the prespecified edges are included in the triangulation, and (2) it is as close as possible to the Delaunay triangulation. We show that the CDT can be built in optimal O(n log n) time using a divide-and-conquer technique. This matches the time required to build an arbitrary (unconstrained) Delaunay triangulation and the time required to build an arbitrary constrained (nonDelaunay) triangulation. CDTs, because of their relationship with Delaunay triangulations, have a number of properties that should make them useful for the finite-element method. Applications also include motion planning in the presence of polygonal obstacles in the plane and constrained Euclidean minimum spanning trees, spanning trees subject to the restriction that some edges are prespecified.

Introduction

Assume we are given a planar straight-line graph G. A *constrained triangulation* of G is a triangulation of the vertices of G that includes the edges of G as part of the triangulation. See [PS85] for an explanation of how such a triangulation can be found in O(n log n) time. A *constrained Delaunay triangulation* (CDT) of G (called an obstacle triangulation in [Ch86] or a generalized Delaunay triangulation in [Le78]) is a constrained triangulation of G that also has the property that it is as close to a *Delaunay triangulation* as possible.

In this paper, we show that the CDT can be built in O(n log n) time, the same time bound required to build the (unconstrained) Delaunay triangulation, by using a method similar to that used by Yap [Ya84] for building the Voronoi diagram of a set of simple curved segments. Previously, the fastest algorithm for constructing the CDT required $O(n \log^2 n)$ time [Le78].

The *Delaunay triangulation* is the straight line dual of the *Voronoi diagram*. See [PS85] for definitions and a number of applications of Delaunay triangulations and Voronoi diagrams.



A Voronoi diagram and the corresponding Delaunay triangulation.

An important property of the Delaunay triangulation is that edges correspond to empty circles. Indeed, this property can be used as the definition of Delaunay triangulation.

Definition. Let S be a set of point in the plane. A triangulation T is a *Delaunay triangulation* of S if for each edge e of T there exists a circle C with the following properties:

1) the endpoints of edge e are on the boundary of C, and
2) no other vertex of S is in the interior of C.

If no 4 points of S are cocircular then the Delaunay triangulation is unique. For most cases in which there is not a unique Delaunay triangulation, any of them will do.

The following definition, equivalent to the definition given in [Le78], indicates what we mean when we say "as close as possible to the Delaunay triangulation". Compare this defintion with the definition of the (unconstrained) Delaunay triangulation given above.

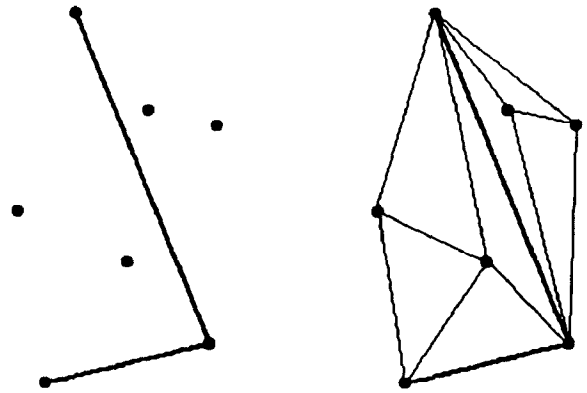
Definition. Let G be a straight-line planar graph. A triangulation T is a *constrained Delaunay triangulation* (CDT) of G if each edge of G is an edge of T and for each remaining edge e of T there exists a circle C with the following properties

1) the endpoints of edge e are on the boundary of C, and
2) if any vertex v of G is in the interior of C then it cannot be "seen" from at least one of the endpoints of e (i.e., if you draw the line segments from v to each endpoint of e then at least one of the line segments crosses an edge of G).

It follows immediately from the definition that if G has no edges then the constrained Delaunay triangulation is the same as the (unconstrained) Delaunay triangulation.

Intuitively, the definition of Delaunay traingulation and constrained Delaunay triangulation are the same except that, for the CDT, we ignore portions of a circle whenever the circle passes through an edge of G. Note that the CDT is *not* the same as the dual of the line-segment Voronoi diagram. (O(n log n) algorithms for constructing the line-segment Voronoi diagram appear in [Ki79] and [Ya84].)

We distinguish 2 types of edges that appear in a CDT: *G-edges*, prespecified edges that are forced upon us as part of G, and *Delaunay edges*, the remaining edges of the CDT.



A graph G and the corresponding constrained Delaunay triangulation.

One measure of the appropriateness of a definition is its utility. We demonstrate the utility of the definition of a CDT by presenting some applications. Just as the (unconstrained) Delaunay triangulation of S can be used to quickly determine the Euclidean minimum spanning tree (EMST) of S, the CDT can be used to find the constrained EMST of S, constrained in the sense that certain edges of the spanning tree are prespecified and may not be crossed by other edges of the spanning tree. See [PS85] for a proof that the EMST is a subgraph of the Delaunay triangulation. Virtually the same proof can be used to show that the constrained EMST is a subgraph of the CDT.

An additional application is presented in [Ch86] and [Ch87] where variations of the standard CDT are used for motion planning in the plane. These variations use a different distance function, in effect using a "circle" that is shaped like a square [Ch86] or a triangle [Ch87]. With some minor modifications, the results presented here are valid for CDTs based on the square-distance (the L_1 metric) or the triangle-distance or based on other *convex distance functions*. See [CD85] for more information on convex distance functions and their relation to Delaunay triangulations.

CDTs should also prove useful for the finite-element method. This is an area in which standard Delaunay triangulations have been shown to have desirable properties. CDTs inherit some of these desirable properties with the advantage that some edges can be prespecified.

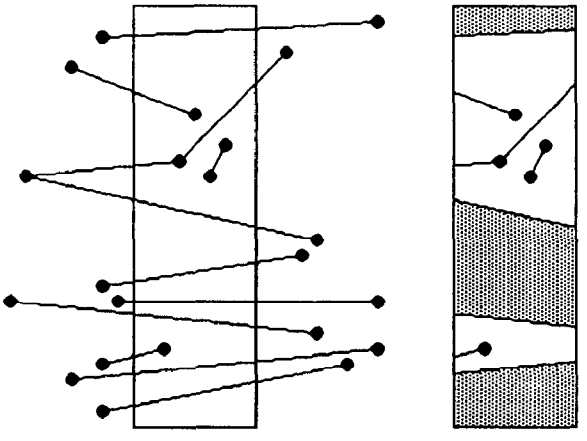
The algorithm

We use divide-and-conquer to build the CDT. For simplicity of presentation, we assume that the planar graph G is contained within a given rectangle. We start by sorting the vertices of G by x-coordinate; then we use this information to divide the rectangle into vertical strips in such a way that there is exactly one vertex in each strip. Of course, this cannot be done if some vertex is directly above another, but we can avoid this problem by rotating the entire graph G if necessary. Following the divide-and conquer paradigm, the CDT is calculated for each strip, adjacent strips are pasted together in pairs to form new strips, and the CDT is calculated for each such newly formed strip until the CDT for the entire G-containing rectangle has been built. The whole process takes time O(n log n) provided the CDT calculation for each newly formed strip can be done in time O(v) where v is the total number of G-vertices in the newly formed strip.

The trick used to ensure that this CDT pasting operation is done in reasonable time is

to avoid keeping track of too much edge information. Note that it is not possible to keep track of all places where G-edges intersect strip boundaries; if we do so then we could have as many as $O(n^2)$ intersections to work with. Edges that cross a strip, edges with no endpoints within the strip, are, for the most part, ignored. Such an edge is of interest only if it interacts in some way with a vertex that lies within the strip. Yap [Ya84] used the same technique to develop an algorithm for the line-segment Voronoi diagram.

Each strip, then, is divided into regions by *cross edges*, G-edges that have no endpoint within the strip. We do not keep track of all of these regions (there could be $O(n^2)$ of them). Instead we keep track of just the regions that contain one or more vertices.



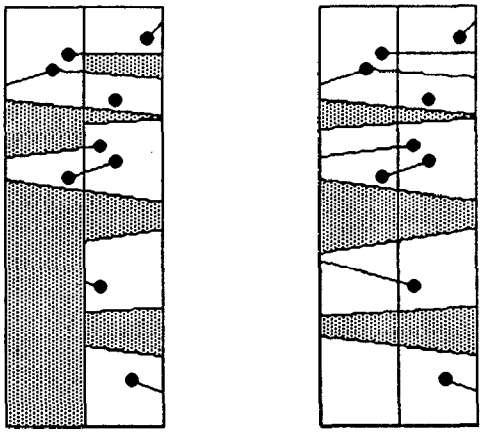
The contents of a strip and the contents that we keep track of.

At this point, two parts of the CDT algrothm require further explanation: (1) how to handle vertex-containing regions, initializing them and keeping track of them as adjacent strips are combined, and (2) how to stitch together CDTs as adjacent strips are combined.

Control of vertex-containing regions

Each initial strip has a single region containing its single vertex. To create these initial regions we need to know the edge immediately above and the edge immediately below each vertex. (Note that the edge immediately above (below) a vertex may be the top (bottom) edge of the rectangle that contains the entire graph G.) This information can be found for all vertices in O(n log n) time by using a vertical line as a sweep-line. See [PS85] for an explanation of this technique and a number of applications.

It is not difficult to determine appropriate regions when two adjacent strips are stitched together. Basically, we merge regions by running through the regions in their order along the strips.



Merging the regions of two adjacent strips.

As we move from top to bottom in the combined strip, we start a new region whenever either strip starts a region, we continue the region as long as a region continues in either strip, we stop a region only when we reach a point where neither strip has a vertex-containing region. It

is easy to see that this merge operation takes time proportional to the total number of regions in both strips; thus, it takes time O(v) where v is the number of vertices in both strips.

Stitching CDTs together

The method we use for stitching CDTs together is similar to one of the methods used by Lee and Schachter [LS80] to build (unconstrained) Delaunay triangulations. We assume the following operations can be done in unit time: (1) given point p and circle C, test whether p is in the interior of C; (2) given points p, q, and r, return the circle through these points. These functions clearly take constant time for any reasonable model of computation, even if we use nonstandard "circles" as in [CD85].

The major difficulty is to determine a place to start so that the CDTs of two adjacent strips may be stitched together. To do this, we create imaginary vertices at infinity. Each vertex-containing region has a set of 4 of these infinite vertices. For each region, the 4 infinite vertices are treated as if they are located at $(\pm\infty,\pm\infty)$. It is convenient to picture the infinite vertices for a region as if they are located at the 4 corners of the region; this is a convenience for drawing pictures - these vertices are treated in other respects as if they are located at one of the 4 infinite points. The CDT for a vertex-containing region is constructed using these infinite vertices as additional data points for the CDT. We specify that infinite vertices of two different regions do not interact; these vertices interact only with points within their own region. Note that with just 4 extra vertices per vertex-containing region the overall running time is affected by just a constant factor.

In the following algorithm, we create circles that go through these infinite vertices. To determine the proper form for such a circle, for instance a circle through (∞,∞), the reader should first consider a circle using the point (w,w), where w is a large number. Use the limiting circle as w approaches ∞. (The limiting circle will be a half-plane.)

The infinite vertices were introduced so that we always have a starting place for a CDT. Whenever we need access to a CDT we use an infinite vertex a starting point. The following lemmas show that the infinite vertices do not otherwise affect the CDTs.

Lemma 1. If we eliminate the infinite vertices of a region by removing the infinite vertices and their edges then the remaining vertices and edges form a CDT for the region.

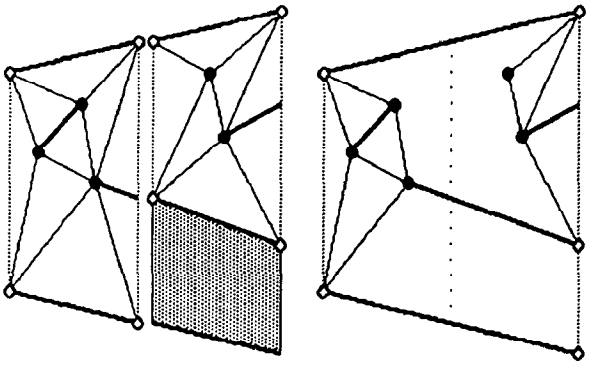
Proof. Infinite vertices interact only with vertices on the convex hull.

Lemma 2. Let S_1 and S_2 be adjacent strips that are combined to make strip S and let T be a CDT that is in S. (S_1 and S_2 contain CDTs that may be altered in the process of making S.) If e is a Delaunay edge of T and both endpoints of e are in S_1 then e is a Delaunay edge of a CDT in S_1.

Proof. By definition, there is a circle through the endpoints of e that contains no other vertices of S that can be seen from e. The same circle shows that e is a Delaunay edge in S_1.

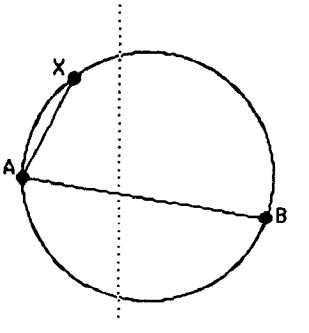
The process by which we combine the adjacent CDTs of the two strips is roughly equivalent to the process of constructing the dividing chain in the divide-and-conquer algorithm for building the Voronoi diagram. (See [PS85] for an explanation the Voronoi diagram algorithm.) CDTs are combined by executing the following steps: (1) eliminate infinite vertices along the boundary, leaving a partial CDT in each half; (2) stitch the partial CDTs together; and (3) eliminate any infinite vertices not stored at the region corners. More detailed versions of these steps are given below.

Step 1. Eliminate the infinite vertices along the boundary between the two strips. Also, eliminate any edges that use these vertices as endpoints. This leaves a partial CDT in each strip. Add a new infinite vertex for each region corner that does not have one. Lemma 2 implies that the only new edges that need to be added to complete the CDT for the combined strip are edges that cross the boundary between the two strips. (Some Delaunay edges of the partial CDTs may still need to be eliminated.)



Step 1: Eliminating infinite vertices as strips are combined. Open circles represent infinite vertices.

Explanation for step 2. Let A and B be vertices of G. Assume edge AB crosses the boundary between the two strips and that edge AB is known to be part of the desired CDT (AB may be either a new Delaunay edge added in an earlier stage or a G-edge). Consider a circle with points A and B on its boundary and with center well below AB. Change the circle by moving the center upward toward AB always keeping A and B on the boundary. Continue moving the center upward until the circle intersects the first point above AB that can be seen from both A and B. Call this point X. By definition, the edges AX and BX are Delaunay edges of the CDT.



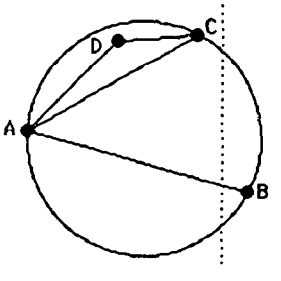
X is the first point that can be seen from A and B.

X is either in B's strip or in A's strip; we assume for the moment that X is in A's strip. By Lemma 2, we know AX exists in the lefthand partial CDT. So the next new edge above AB must go from B to a point X already connected to A; in other words, the possible points X are

limited to those connected to A. This gives us an efficient way to find the next edge above AB: if X is on the left then we only have to examine edges connected to A; if X is on the right then we only have to examine edges connected to B. Of course, we can't tell ahead of time whether the point we are looking for is on the left or the right, so we choose the best candidate point on each side, then we choose between them.

We still need to develop an efficient way to determine which of the points connected to A is the best candidate. We assume for the moment that X is on the same side of the boundary as A. We know from the argument above that the circle through A, B, and X shows AX to be a Delaunay edge, by definition. In other words, if AX is a good candidate then circle ABX will not contain a vertex that can be seen from A and X.

Let AC be the next edge counterclockwise around A from AB. We test if AC is a good candidate by examining triangle ADC (if it exists) where AD is the next edge around A from AC. If no such triangle exists (AC is on the edge of the partial CDT) or if AC is a G-edge then edge AC is automatically considered a good candidate. If triangle ADC does exist and AC is a Delaunay edge then we test to see if D is within circle ABC; if it is then AC is not a good candidate. Further, we know AC is not a valid edge in the CDT at all, so it can be eliminated; AD would then be tested to see if AD is a good candidate.



AC is not a good candidate and is eliminated.

Step 2. Each vertex containing region is divided into subregions by G-edges that cross the boundary of the strip. We consider only those G-edges that have endpoints within the strip, either G-vertices or the special infinite vertices. New Delaunay edges that cross the boundary (by Lemma 2 these are the only new edges we need) can be found be completing the following process for each subregion. Let A and B be the endpoints of the G-edge at the bottom of the current subregion.

loop
Eliminate A-edges that can be shown to be illegal because of their interaction with B;
Eliminate B-edges that can be shown to be illegal because of their interaction with A;
Let C be a candidate where AC is the next edge counterclockwise around A from AB (if such an edge exists);
Let D be a candidate where BD is the next edge clockwise around B from BA (if such an edge exists);
exit loop if no candidates exist;
Let X be the candidate that corresponds to the lower of circles ABC and ABD;
Add edge AX or BX as appropriate and call this new edge AB;
end loop;

Step 3. Eliminate any infinite vertices not stored at the region corners. Of course, we also eliminate any Delaunay edges (the ones that are not part of G) that use these infinite vertices. Because these vertices are located at infinity, their elimination does not affect CDT edges within the newly combined strip.

The process outlined above runs in time O(v) where v is the total number of vertices in the newly combined strip. To see this note that in each part of step 2 we either eliminate an edge from a partial CDT or we add an edge for the combined CDT. Since both the partial CDT and the combined CDT contain O(v) edges, the time bound follows.

Conclusions

We have shown how a divide-and-conquer algorithm can be used to produce the constrained Delaunay triangulation (CDT) in O(n log n) time. This time bound is optimal since it is easy to show that a CDT can be used to sort. There are two ideas that have been particularly important in reaching this time bound: (1) as in [Ya84], the only cross edges that we keep track of are those that bound vertex-containing regions; (2) infinite vertices are used so that partial CDTs are linked for efficient access. A useful property of these infinite vertices is that such a vertex can be eliminated with minimal effect on edges between noninfinite vertices.

It may be possible to develop a sweep-line algorithm for constructing the CDT. Such an algorithm would probably run faster than the divide-and-conquer algorithm presented here, although it would, of course, have the same asymptotic time bound. Not surprisingly, the CDT, a type of Delaunay triangulation, has a dual that is a type of Voronoi diagram. However, this dual graph can overlap itself, with different portions of the graph sharing the same portion of the Euclidean plane. Thus, the CDT does not lend itself to sweep-line techniques of the type used by Fortune [Fo86] to build the standard Voronoi diagram.

References

[CD85] L. P. Chew and R. L. Drysdale, Voronoi diagrams based on convex distance functions, Proceedings of the 1st Symposium on Computational Geometry, Baltimore (1985), 235-244. (Revised version submitted to Discrete and Computational Geometry.)

[Ch86] L. P. Chew, There is a planar graph almost as good as the complete graph, Proceedings of the Second Annual Symposium on Computational Geometry, Yorktown Heights (1986), 169-177.

[Ch87] L. P. Chew, Planar graphs and sparse graphs for efficient motion planning in the plane, manuscript.

[Fo86] S. Fortune, A sweepline algorithm for Voronoi diagrams, Proceedings of the Second Annual Symposium on Computational Geometry, Yorktown Heights (1986), 313-322.

[Ki79] D. G. Kirkpatrick, Efficient computation of continuous skeletons, Proceedings of the 20th Annual Symposium on the Foundations of Computer Science, IEEE Computer Society (1979), 18-27.

[Le78] D. T. Lee, Proximity and reachibility in the plane, Technical Report R-831, Coordinated Science Laboratory, University of Illinois (1978).

[LS80] D. T. Lee and B. Schachter, Two algorithms for constructing Delaunay triangulations, International Journal of Computer and Information Sciences, 9:3 (1980), 219-242.

[PS85] F. P. Preparata and M. I. Shamos, Computational Geometry, Springer-Verlag (1985).

[Ya84] C. K. Yap, An O(n log n) algorithm for the Voronoi diagram of a set of simple curve segments, Technical Report, Courant Institute, New York University (Oct. 1984).