[LL86] D. T. Lee and A. K. Lin. Generalized delaunay triangulation for planar graphs. *Discrete & Computational Geometry*, (1):201–217, 1986.

[LP78] D. T. Lee and F. P. Preparata. The all nearest neighbor problem for convex polygons. *Information Processing Letter*, pages 189–192, June 1978.

[LS80] D. T. Lee and B. J. Schachter. Two algorithms for constructing a delaunay triangulation. *International Journal of Computer and Information Sciences*, 9(3):219–242, 1980.

[PS85] F. P. Preparata and M. I. Shamos. *Computational Geometry*. Springer-Verlag, New York, 1985.

[SH75] M. I. Shamos and D. Hoey. Closest point problems. In *Proceedings of the 16th IEEE Symposium on Foundations of the Computer Science*, pages 151–162, October 1975.

[Sib78] R. Sibson. Locally equiangular triangulations. *The Computer Journal*, 21(3):243–245, August 1978.

[YL79] C. C. Young and D. T. Lee. A note on all nearest neighbor problem for convex polygons. *Information Processing Letter*, 8:193–194, April 1979.

[ZSZZ90] Jianming Zhou, Keran Shao, Keding Zhou, and Qionghua Zhan. Computing constrained triangulation and delaunay triangulation: A new algorithm. *IEEE Transactions on Magnetics*, 26(2), march 1990.

References

[BEG90] Marshall Bern, David Eppstein, and John Gilbert. Provably good mesh generation. In *Proc. 31st IEEE Symposium Foundations of Computer Science*, pages 231–241, 1990.

[Che89] L. Paul Chew. Constrained delaunay triangulations. *Algorithmica*, (4):97–108, 1989.

[CT76] D. Cheriton and R. E. Tarjan. Finding minimum spanning trees. *SIAM J. Comput.*, pages 724–742, December 1976.

[DD82] Pierre A. Devijver and Michel Dekesel. Insert and delete algorithms for maintaining dynamic delaunay triangulations. *Pattern Recognition Letters*, 1(2):73–77, December 1982.

[Dwy87] Rex A. Dwyer. A fast divide-and-conquer algorithm for constructing delaunay triangulations. *Algorithmica*, (2):137–151, 1987.

[FB74] R. A. Finkel and J. L. Bentley. Quad trees a data structure for retrieval on composite keys. *Acta Informatica*, 4:1–9, 1974.

[FFP85] L. De Floriani, B. Falcidieno, and C. Pienovi. Delaunay-based representation of surfaces defined over arbitrarily shaped domains. *COMPUTER VISION, GRAPHICS, AND IMAGE PROCESSING*, pages 127–140, 1985.

[FP88] Leila De Floriani and Enrico Puppo. Constrained delaunay triangulation for multiresolution surface description. In *International Conference on Pattern Recognition*, pages 566–569, November 1988.

[Fri72] I. Fried. Condition of finite element metrics generated from nonuniform meshes. *AIAA Journal*, 10:219–221, 1972.

[GKS90] L. J. Guibas, D. E. Knuth, and M. Sharir. Randomized incremental construction of delaunay and voronoi diagrams. In *Proceedings of 17th International Colloq. – Automata, Languages, and Programming*, number 443, pages 414–431. Springer-Verlag LNCS, 1990.

[GS78] P. J. Green and R. Sibson. Computing dirichlet tessellations in the plane. *The Computer Journal*, 21(2):168–173, May 1978.

[GS85] Leonidas Guibas and Jorge Stolfi. Primitives for the manipulation of general subdivisions and the computation of voronoi diagrams. *ACM Transactions on Graphics*, 4(2):74–123, April 1985.

[KG89] J. Mark Keil and Carl A. Gutwin. The delaunay triangulation closely approximates the complete euclidean graph. *Springer-Verlag Lecture Notes on Computer Science*, 382:47–56, 1989.

[Law77] C. L. Lawson. Software for c^1 surface interpolation. *Mathematical Software III*, 1977.

[LD91] Yizhi Lu and Wayne Dai. A numerical stable algorithm for constructing constrained delaunay triangulation and application to multichip module layout. In *Proc. of 1991 International Conference on Circuits and Systems*, June 1991.

7. Summary and Future Work

7.0.3 Conclusion

Maintaining dynamic structures such as the constrained Delaunay triangulation is one of the research topics in computational geometry. Most existing algorithms are semi-dynamic in the sense that they only support insertions. This thesis presents a set of fully dynamic algorithms for constrained Delaunay triangulation. They support insertions of new points and new constrained edges as well as deletions of existing points and existing constrained edges. Some interesting properties of the constrained Delaunay triangulation have been shown by this thesis.

All these algorithms have been implemented and integrated into the SURF layout system. Although the algorithms are developed for a multichip module layout system, the ideas and algorithms should be applicable to other mesh generation.

The contributions of this thesis are:

- The algorithms are fully dynamic.
- The algorithms are concise and easy to implement.
- The algorithms are numerically stable even with a large amount of points.
- The experiments verifies that if all the points are inserted in random order, the number of new edges added for each new point in the process of locally updating CDT is $O(1)$.

7.0.4 Future work

Although our algorithms have been successfully implemented, it might be possible to get further improved. We are interested in developing a more efficient algorithm to locally updating a simple polygon, in which the boundary edges are not necessary be the constrained edges, and points and edges outside of this polygon may be triangulated already.

a time to apply LOP [Law77] of the quadrilateral. The time and space complexity is $O(N^2)$, or $O(N^2 \log N)$ time and $O(N)$ space. Also, they use the polygon cutting theorem and divide-and-conquer technique to make a Delaunay triangulation in a simple polygon. The time complexity is $O(N \log N)$. One drawback of this approach is that the algorithm couldn't handle the case in which the constrained edges are inserted incrementally.

Please note in [FP88] that the algorithm for insertion of a point or a constrained edge is very similar to the one presented in this thesis. The worst time complexity is $O(N^2)$, and for the point insertion algorithm, it runs in $O(N)$ time when the points are distributed uniformly. For the polygon resulted from deleting invalid edges, they first triangulate the polygon into a non-Delaunay triangulation, then optimized the portion of the triangulation inside the polygon by applying the circle criterion (the optimization procedure was not presented in the paper).

In this thesis, algorithms which dynamically update the constrained Delaunay triangulation are presented. We use the circle criterion instead of the LOP approach. The expected time complexity for point insertion is $O(N \log N)$ when points are inserted in random order, where N is the total number of points in the graph. The expected time complexity for point deletion is $O(N)$. The insertion and deletion algorithms for constrained edges are straightforward. There are no complex calculations. Although the worst time complexity is worse than others ($O(l|P|^3)$, where l is the number of constrained edges, and the $|P|$ is the number of points of the corresponding polygon, it is expected that the number of points of the resulting polygon at each stage will be small. In return for the higher worst case time complexity, we have a simple incremental algorithm.

two different types of constraints: boundary and prespecified line segments. A CDT provides a natural way to retain the boundary and line segment information while providing a "good" triangulation.

6.2.1 Divide-and-conquer approach

In [FFP85], the authors construct the constrained Delaunay triangulation within a polygon. They divide the polygon by connecting those points of the polygon that were both visible and Voronoi neighbors. Then, they incremently insert points and locally update the triangulation. It requires $O(N^2)$ time in the worst case.

L. Paul Chew [Che89] used a divide-and-conquer algorithm to construct the constrained Delaunay triangulation in a rectangle with points and constrained edges. It sorts all of the vertices in the graph by x-coordinate, then divides the rectangle into vertical strips that with exactly one vertex in each strip, calculates the constrained Delaunay triangulation for each strip, and then pastes them together in pairs. The time complexity is $O(N \log N)$.

The divide-and-conquer techniques may not be easy to implement in practice. They require the advance specification of all data points and constrained segments, and require more storage space, although they are generally more efficient in computation. The incremental techniques are usually easier to code and require limited amounts of storage. Our experiments show that when the points are distributed uniformly, the performance of the incremental techniques are quite promising. We are more interested in the incremental algorithm, because we need to dynamically update the CDT to meet the demands of this particular routing tool.

6.2.2 Incremental approach

D. T. Lee and A. K. Lin [LL86] proposed an algorithm to construct a constrained Delaunay triangulation with points and constrained edges. It computes the Delaunay edges incident with each vertex in the graph by first finding the visibility graph of each vertex, then for each one, it begins with a shortest edge and takes three consecutive vertices at

details in both the algorithms and the proof. The idea of the quad-edge data structure is that it keeps the records for vertices or faces in one data structure. One of the advantages of this data structure is it allows uniform access to the dual and mirror-image subdivisions. They used the circle criterion to construct Delaunay triangulation.

Rex A. Dwyer [Dwy87] gave an algorithm which ran in $O(N \log\log N)$ time for a large class of distributions that includes the uniform distribution in the unit square. The algorithm partitioned the whole graph into square cells. The Delaunay triangulation of the points within each cell was constructed with the Guibas-Stolfi algorithm. The triangulations within each row of cells were merged in pairs until the triangulation of the row has been completed. The row triangulations were merged in pairs to complete the triangulation of the entire set of points.

6.1.2 Incremental approach

D. T. Lee and B. J. Schachter [LS80] also provided an incremental algorithm to construct the Delaunay triangulation. It used the LOP approach developed by Lawson [Law77]. The algorithm requires $O(N^2)$ time in the worst case. If the points are distributed uniformly in the rectangle, then the algorithm is $O(N^{3/2})$ empirically.

Once again Guibas and Stolfi [GS85] had a similar algorithm but using the quad-edge data structure.

In [DD82], Pierre A. Devijver and Michel Dekesel presented two algorithms for a dynamic Delaunay triangulation. Their algorithms are similar to the ones presented in this thesis. The average case time complexity is both $O(N^{\frac{3}{2}})$ with the assumption that the resulting polygon from point deletion is a convex hull.

6.2 Constrained Delaunay triangulation

The methods above are restricted to Delaunay triangulation. Since there are some constraint factors in applications, many people have worked on the constrained Delaunay triangulation (also called generalized Delaunay triangulation [LL86]). Basically, there are

6. Related Work

In 1934, the mathematician Delaunay proved that for any set of points there was a unique triangulation which maximized the smallest angles in the mesh. In other words, given a set of points, the Delaunay triangulation is the one which provides as near an equilateral set of triangles as possible with the given points.

In [Law77], C. L. Lawson devised a *local optimal procedure* (LOP) to construct the Delaunay triangulation within the quadrilaterals for a set of points. This procedure implemented using the max-min angle criterion.

There are some implementations that generate Delaunay triangulations from Voronoi diagrams ([SH75,GS78]). The Delaunay triangulation is the dual graph of the Voronoi diagram if no four points are co-circular, this transformation can be done in $O(N)$ time. Since the Voronoi diagram can be generated in $O(N \log N)$ time, the Delaunay triangulation can also be generated in $O(N \log N)$ time.

The algorithms which generate this kind of triangulation directly can be divided as algorithms for Delaunay triangulation and algorithms for constrained Delaunay triangulation. They can be further divided by the strategies they used: divide-and-conquer technique versus incremental technique.

6.1 Delaunay triangulation

6.1.1 Divide-and-conquer approach

The algorithm proposed by D. T. Lee and B. J. Schachter [LS80] sorted the given set V of N points in lexicographically ascending order, then divided V into two subsets V_L and V_R. It recursively constructed the Delaunay triangulation of V_L and V_R, and afterwards, merged them to get the final Delaunay triangulation. The algorithm runs in $O(N \log N)$ time.

Leonidas Guibas and Jorge Stolfi [GS85] presented an algorithm which was very close to the one in [LS80], but they used a quad-edge data structure and gave more complete

Figure 5.5: The topological routing with CDT as the underlying data representation

Figure 5.4: The minimum spanning tree from the Delaunay triangulation with 1000 points

Figure 5.3: The Delaunay triangulation with 1000 points

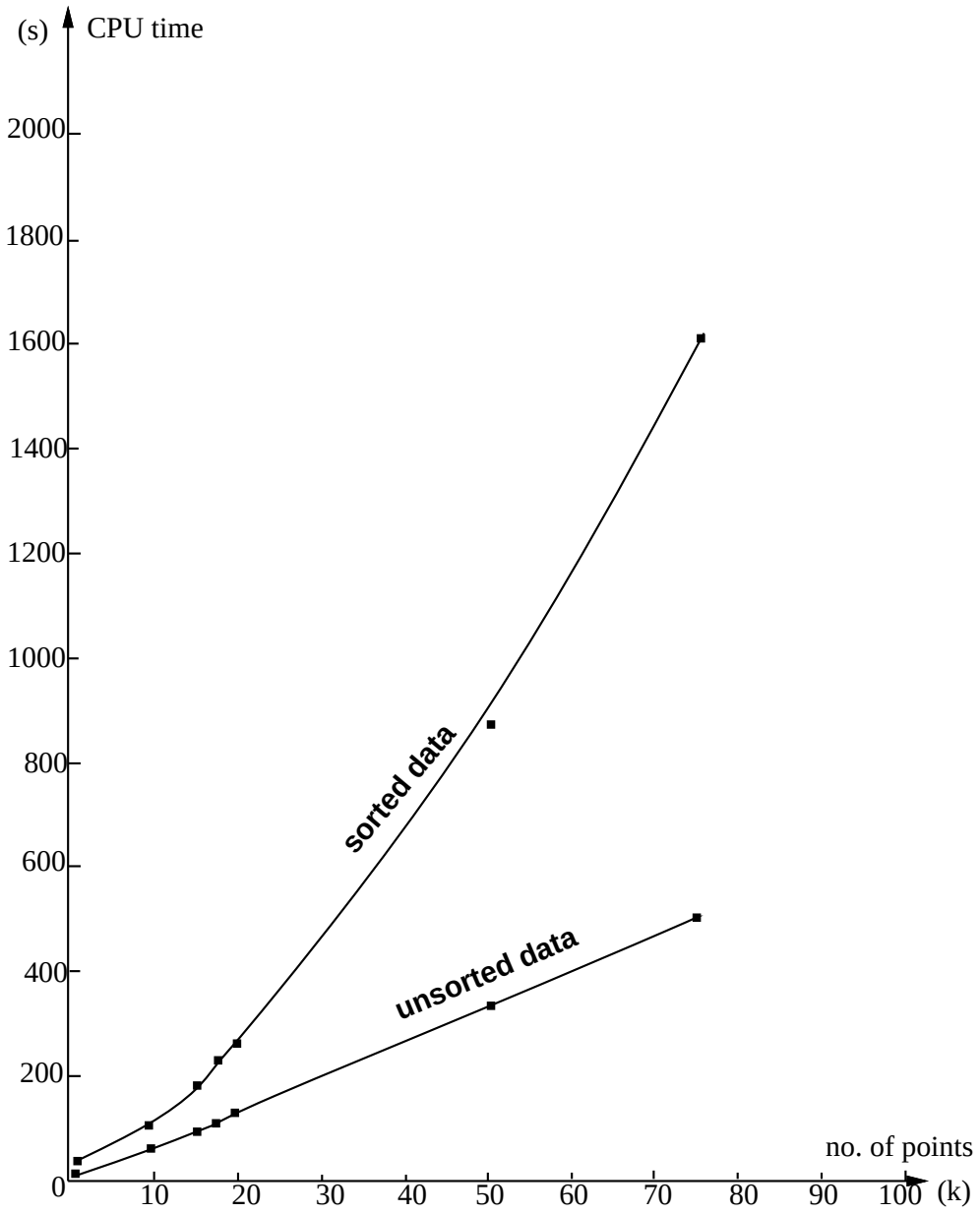


Figure 5.2: Running time of the Delaunay triangulation of sorted and unsorted data

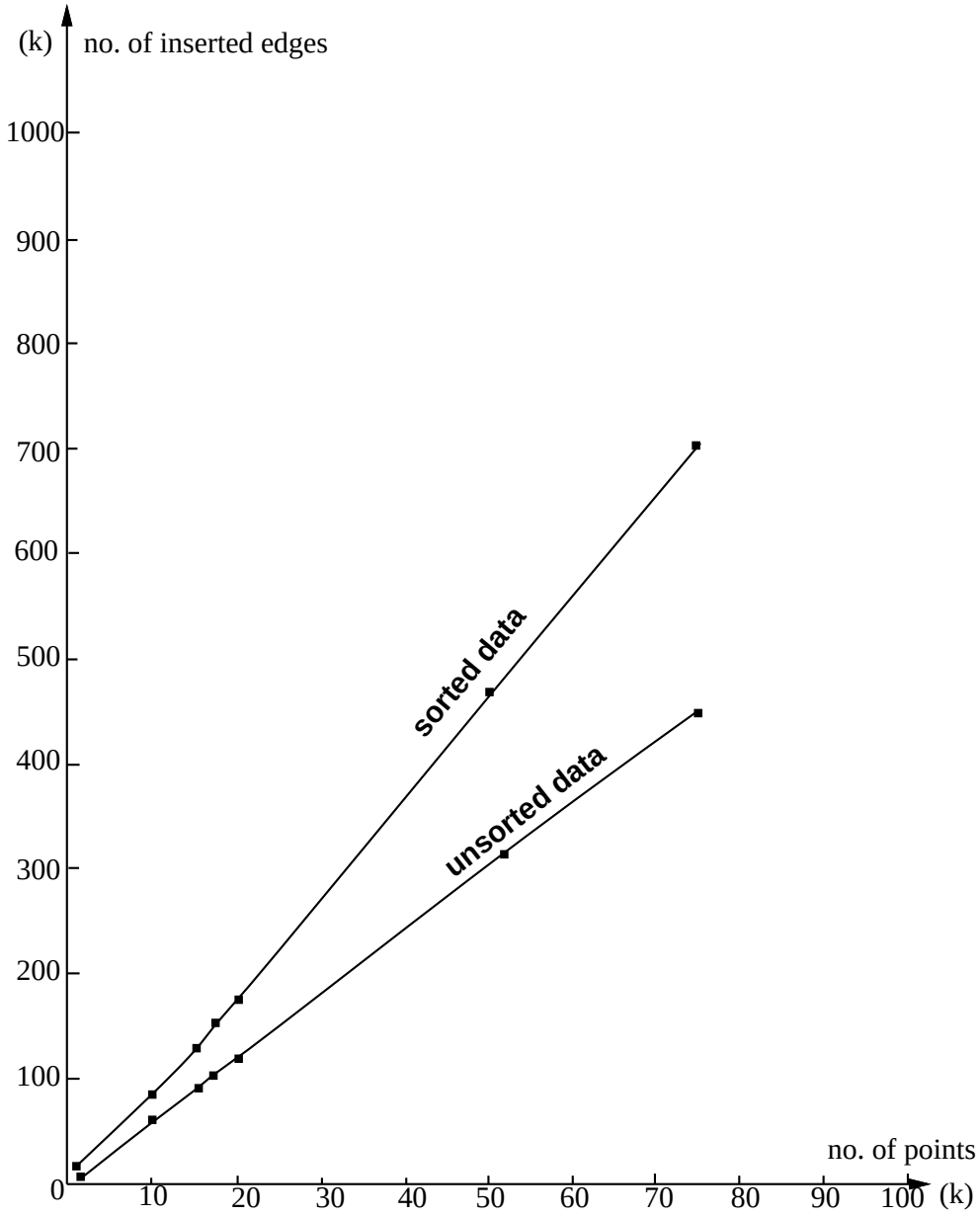


Figure 5.1: The number of inserted edges of sorted and unsorted data

points	sorted		unsorted	
size	time (ms)	no. of edges	time (ms)	no. of edges
1k	7970	7476	5910	5809
10k	105680	85814	63440	59758
15k	182730	132001	95780	89925
17.5k	212340	154608	112860	104996
20k	250110	176108	129000	119865
50k	872110	467698	331830	299483
75k	1604130	705938	511050	449355

Table 5.1: experiments with sorted and unsorted data

5. Implementation

The algorithms described in this paper have been implemented on Sun SPARCStation 1+ in C language. Randomly generated data has been used to test the algorithms. The experiments have been conducted by inserting randomly generated points in different orders. The "sorted" means adding points one by one in x-sorted order (using y-order to break ties), while "unsorted" is chosing points in random order. The experiments also calculate the total number of the new edges generated in the process of constructing Delaunay triangulation. The results (Figure 5.2) show that the running times for sorted data are worse than for the unsorted data, and more edges are updated for sorted data while they generated the same CDT from the same data in different orders.

Although the Delaunay triangulation is built with an incremental algorithm, the experimental results are promising when the points are inserted randomly. The expected number of new edges for each new point is $O(1)$. Our experiments also also this fact (see Table 5.1, Figure 5.1 and Figure 5.2). They show the average number of new edges ranges from 5.8090 to 5.9998. The example for a Delaunay triangulation with a thousand randomly generated points is appended (Figure 5.3) as well as the corresponding minimum spanning tree (Figure 5.4). An example of CDT is shown here as well (Figure 5.5).

$$d_3 = \begin{vmatrix} x_1^2 + y_1^2 & x_1 & 1 \\ x_2^2 + y_2^2 & x_2 & 1 \\ x_3^2 + y_3^2 & x_3 & 1 \end{vmatrix};$$

$$d_4 = \begin{vmatrix} x_1^2 + y_1^2 & x_1 & y_1 \\ x_2^2 + y_2^2 & x_2 & y_2 \\ x_3^2 + y_3^2 & x_3 & y_3 \end{vmatrix}.$$

So the center of the circle (x_0, y_0) is:

$$x_0 = \frac{d_2}{2d_1}; y_0 = -\frac{d_3}{2d_1}.$$

The radius is:

$$r = \frac{\sqrt{d_2^2/d_1^2 + d_3^2/d_1^2 + 4d_4/d_1}}{2}.$$

If the query point is in the circle, then the Euclidean distance between query point and the center of the circle should be less than the radius of the circle:

$$\sqrt{(x_q - x_0)^2 + (y_q - y_0)^2} - r < 0.$$

To avoid square root calculations, this inequality can be reformulated as:

$$d_1(x_q(x_q d_1 - d_2) + y_q(y_q d_1 + d_3) - d_4) < 0.$$

If all the coordinates of points are integers, as is the case in the SURF layout system, then these calculations completely avoid rounding errors. Of course, when the integers are very large, this may result in the integer overflow, but this can be avoided by using double precision arithmetic while avoiding the floating point calculations. This strategy successfully constructs the Delaunay triangulation with as many as one hundred fifty thousand points. It appears only to be limited by the amount of memory available.

4. Precision Problem in the Calculation

The problem of numerical errors is very important in constructing Delaunay triangulations. A number of algorithms for its construction are presented with the implicit assumption that there is no numerical error in the course of computation. In the real world, numerical errors cannot be ignored, and it is difficult to judge correctly, whether a point is inside, outside, or exactly on a line or circle. Misjudgement on geometric relations often results in topological inconsistency, and many cause a "theoretically correct algorithm" to fail. What we do here is to try to exclude the rounding errors in the course of calculations. The only numerical calculation in the algorithm is to check whether a point is in a circle or not. The following calculations are used to determine if a query point (x_q, y_q) is inside the circle defined by the three points (x_1, y_1), (x_2, y_2), and (x_3, y_3).

The Circle Equation is:

$$\begin{vmatrix} x^2+y^2 & x & y & 1 \\ x_1^2+y_1^2 & x_1 & y_1 & 1 \\ x_2^2+y_2^2 & x_2 & y_2 & 1 \\ x_3^2+y_3^2 & x_3 & y_3 & 1 \end{vmatrix} = 0.$$

We define d_1, d_2, d_3, and d_4 as:

$$d_1 = \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix};$$

$$d_2 = \begin{vmatrix} x_1^2+y_1^2 & y_1 & 1 \\ x_2^2+y_2^2 & y_2 & 1 \\ x_3^2+y_3^2 & y_3 & 1 \end{vmatrix};$$

can always find at least one triangulation edge in each traverse. And this guarantees that the algorithm will terminate. So the algorithm will triangulate the polygon correctly.

Time complexity

In the worst case, it takes $O(N)$ time to traverse the boundary of the polygon, and $O(N)$ time to check the visibility for each proposed edge. So the worst case time complexity is $O(|P|^3)$, where $|P|$ is the number of points of the polygon.

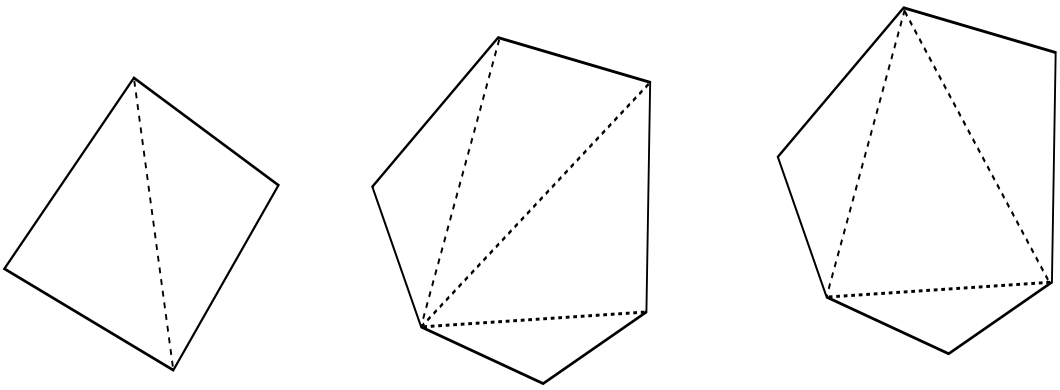


Figure 3.8: Two of the edges of the triangle are the edges of the polygon

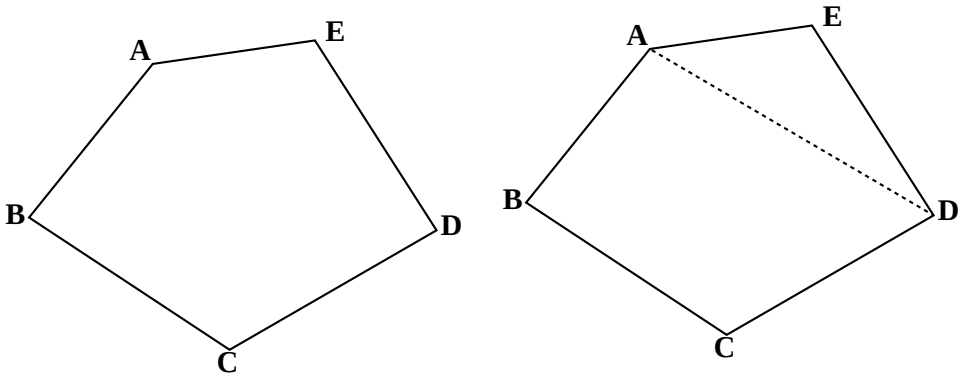


Figure 3.9: Remaining polygon

Correctness

To show the algorithm is correct, the main thing is to show that each time it traverses the remaining polygon, it always adds the correct triangulation edges. No new triangulation edges will be put outside the polygon. To add the triangulation edges, the algorithm needs to check if there are any points of the CDT which are visible from both the endpoints of the proposed triangulation edge. After these checks, the new triangulation edges will meet the circle criterion of CDT, and once these edges are added, they will not be deleted in this procedure. Since for each polygon with more than three points, there are two triangles in which two of its edges are the boundary of the polygon. That means with this algorithm, it

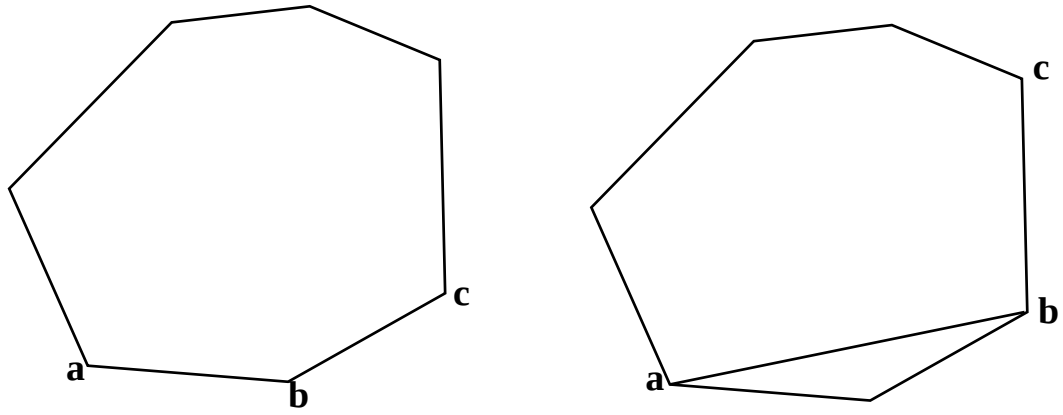


Figure 3.6: Successor point: case I

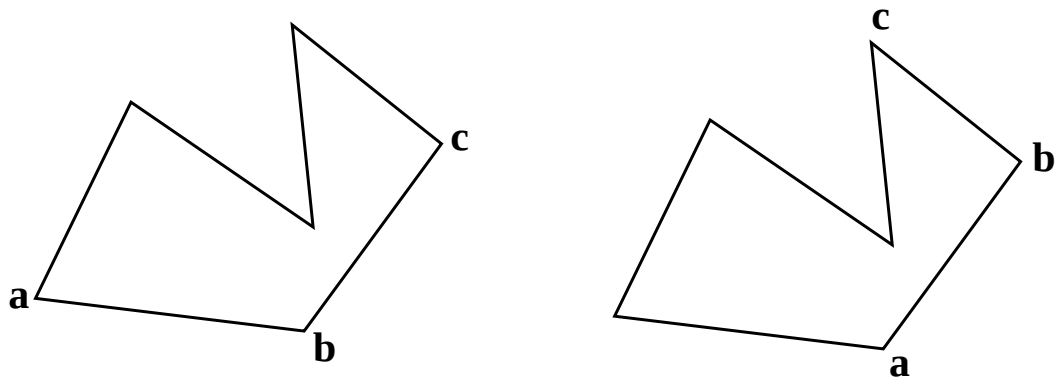


Figure 3.7: Successor point: case II

called the remaining polygon after adding $\overline{AD}$.

The number of triangles in a polygon is proportional to the number of points of the polygon. For the polygon triangulation algorithm, if every time it traverses the remaining polygon, it can add one more triangle, then the algorithm will terminate. So if the algorithm can guarantee that for each traversal, it triangulates at least one of the triangles in which two of the edges of the triangle are the edges of the polygon, the algorithm will terminate and the polygon will be fully triangulated at last.

TriangulatePolygon(T, P)

{

a and b are the first two points of the P which are sorted in the direction around polygon (for example: counter-clockwise).

c = successor of b;

while P is not decomposed into triangles completely

Triangulate the polygon with dummy triangulation edges;

if $\overline{ac}$ is inside P **and**

There is no point inside the checking circle that is visible from a and c

Delete all the dummy triangulation edges;

Connect a with c;

$b = c$;

c = successor of b; (Figure 3.6)

else

$a = b$;

$b = c$;

c = successor of b; (Figure 3.7)

Delete the dummy triangulation edge.

}

3.5.2 Analysis

Termination

Any triangulated polygon with $N(N > 3)$ points has $N - 2$ triangles. Among them there are at least two triangles which have the following characteristic: two of the edges of the triangle are the edges of the original polygon. (Figure 3.8)

To continue the discussion, let's introduce the concept of **remaining polygon**. In Figure 3.9, $ABCDE$ is called the remaining polygon before adding $\overline{AD}$, and $ABCD$ is

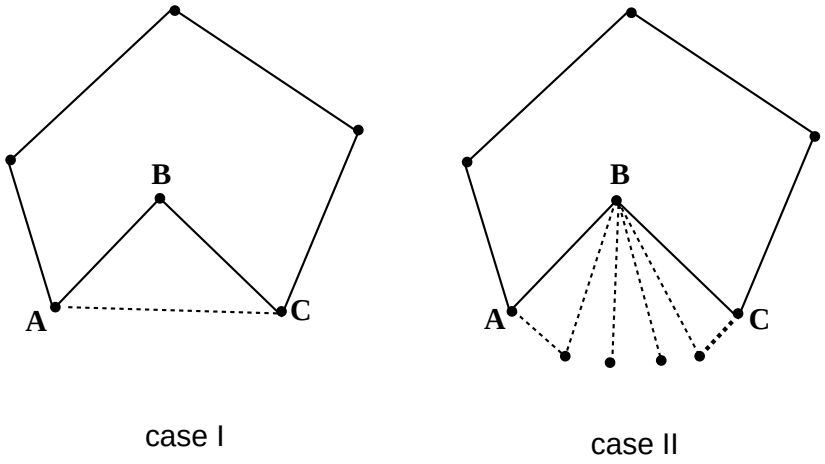


Figure 3.5: Triangulate polygon: special case I and II

Otherwise, it will pick the next point of c, such as d, to check these three points: b, c, and d. Once again, to efficiently check the visibility, the algorithm will first triangulate the polygon with the dummy triangulation edges in any way. After all these visibility checks are completed, the dummy triangulation edges will be removed.

Here is the algorithm for polygon triangulation.

3.5 Triangulating polygons

When triangulating a polygon, we first pick an edge, such as $\overline{AB}$ in Figure 3.4. Then, we consider another point of the polygon which is connected to one of the endpoints of the edge, such as C.

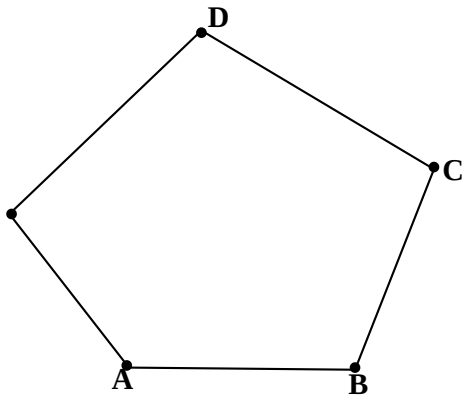


Figure 3.4: Triangulate polygon

Using the circle criterion of CDT, if there is a point of the CDT in the Circle(ABC) and it is visible from both points A and C in Figure 3.4, we will skip edge $\overline{AC}$ and go to check edge $\overline{BD}$, otherwise, we connect A with C.

Since the polygon may be non-convex, there are several cases we need to check. Such as these two cases in Figure 3.5:

In case I, A is already connected with C outside of the polygon; in case II, B is connected with a point which is outside the polygon, and $\overline{AC}$ will intersect with it.

So, when triangulating a polygon, all these cases should be checked.

3.5.1 Algorithm

Each time the algorithm picks three consecutive points of the polygon, such as a, b, and c. If there are no points of the CDT which are inside Circle(abc) and are visible from both a and c, and $\overline{ac}$ is inside the polygon, a will be connected with c with a triangulation edge.

3.4 Deleting a constrained edge from CDT

3.4.1 Algorithm

The steps for deleting a constrained edge from a CDT are: first delete the constrained edge, then delete all the triangulation edges that surround the deleted edge which violate the circle criterion, finally, retriangulate the resulting polygon.

Here is the algorithm for deleting a constrained edge.

DeleteEdge(T, v_a, e_b)

{

Delete the constrained edge e;

Get the new polygon P by deleting all the triangulation edges which violate the circle criterion of CDT;

TriangulatePolygon(T, P).

}

3.4.2 Analysis

Since there are finite number of edges in the CDT, the deletion of triangulation edges will terminate. In the process of checking the validation of triangulation edges, the algorithm will turn all the deleted edges including the constrained edges into dummy triangulation edges. This keeps the polygon resulting from the deletion of these edges fully triangulated. The triangulation is used to check the visibility of points, since visibility is the key in the circle criterion to check the validation of triangulation edges. After deleting all the invalid triangulation edges, all the dummy triangulation edges in the polygon will be deleted. Removing an edge takes $O(1)$ time. In the worst case, the time complexity to delete all the triangulation edges which violate the circle criterion of CDT is $O(N^2)$, where N is the number of triangulation edges in CDT. The polygon triangulation algorithm can be done in $O(|P|^3)$ in the worst case, where $|P|$ is the number of points of the resulting polygon. So the worst case time complexity is $O(N^2 + |P|^3)$.

3.3 Deleting a point from a CDT

3.3.1 Algorithm

It is assumed that the point to be deleted is not incident to any constrained edge. The algorithm first deletes all the triangulation edges that are connected to the point, then deletes the point and calls the polygon triangulation algorithm to triangulate the resulting polygon.

Here is the algorithm for deleting a point.

DeletePoint(T, Q)

{

Delete all the triangulation edges incident to point Q;

Delete point Q;

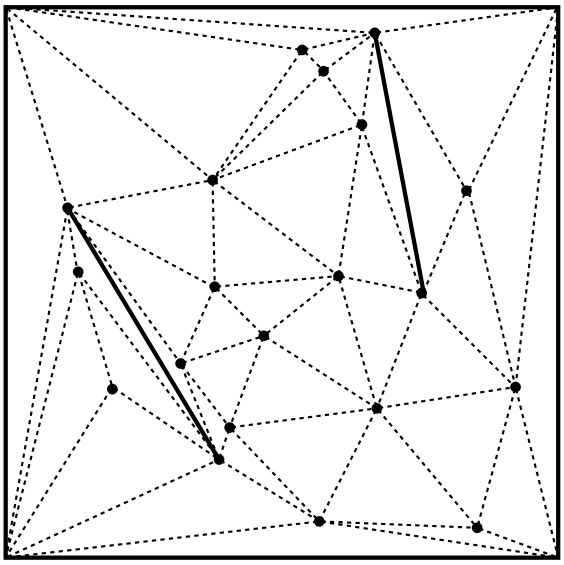
TriangulatePolygon(T, P).

}

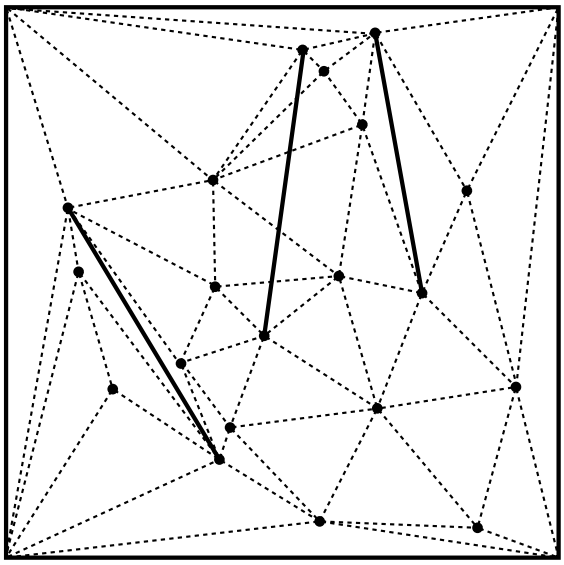
3.3.2 Analysis

Before the deletion, all the triangulation edges meet the circle criterion. After the deletion, the circle criterion will not be violated by the triangulation edges outside the polygon. So, to get the new CDT, it is sufficient to just triangulate the new polygon.

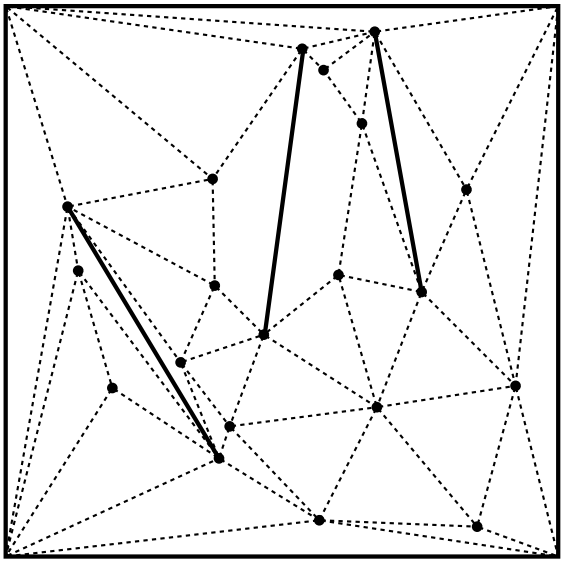
There are limit number of triangulation edges incident to any point in a graph, so the deletion of triangulation edges will terminate. The worst time complexity to delete all the triangulation edges incident to a point is $O(N)$. The average degree of a point in a triangulation is six (from Euler formuler), it takes constant time to get the polygon. For the polygon triangulation algorithm in this case, it will be shown later that the worst case time complexity is $O(|P|^3)$, where $|P|$ is the number of points of the polygon, but on average, it is $O(1)$. So, for average case the time complexity for deleting a point from CDT is $O(1)$.



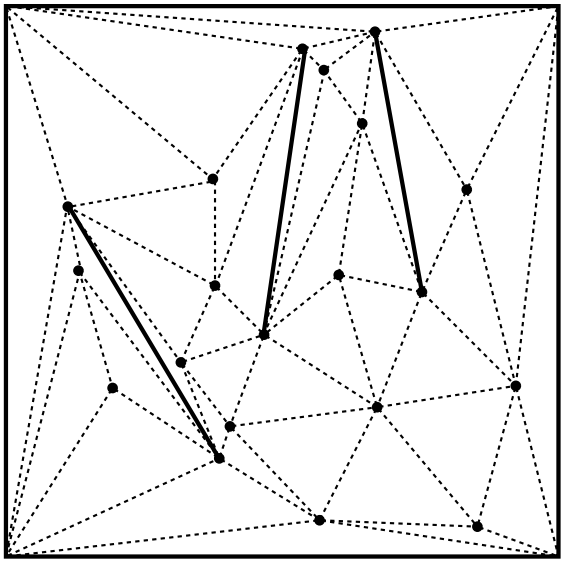
(a) Constrained Delaunay triangulation



(b) Insert a constrained edge into the CDT



(c) Delete all the triangulation edges that intersect with the constrained edge



(d) The new constrained Delaunay triangulation

Figure 3.3: Scenario of constrained edge insertion algorithm

simple polygons split by the constrained edge. After that, the algorithm calls the polygon triangulation algorithm to triangulate these two polygons.

Here is the algorithm for inserting a constrained edge.

AddEdge(T, v_a, v_b)
{
  AddPoint(T, v_a);
  AddPoint(T, v_b);
  Connect v_a and v_b by adding the constrained edge e;
  Delete all the triangulation edges which intersect with e;
  Get polygons $P1$ and $P2$;
  TriangulatePolygon(T, $P1$);
  TriangulatePolygon(T, $P2$).
}

Figure 3.3 shows snap-shots of the constrained edge insertion algorithm.

3.2.2 Analysis

Now we show the algorithm is correct. After placing the constrained edge into the CDT and deleting all the triangulation edges that intersect with the new constrained edge, the other triangles in the CDT still meet the CDT circle criterion. So, it is sufficient to just delete those triangulation edges that intersect with the new constrained edge. Since the number of these edges is limited, the deletion stage will terminate. The correctness of **TriangulatePolygon** will be discussed later.

For the **AddPoint**() routine, the running time in the worst case is $O(N)$, and when points are inserted in random order the time complexity is $O(\log N)$. To get the polygons, the time in the worst case is $O(N)$. For the polygon triangulation algorithm, the worst case time complexity could be $O(|P|^3)$, where $|P|$ is the number of points of the resulting polygon. So the time complexity of the algorithm for adding a constrained edge into a CDT is $O(N + |P|^3)$

3.1.3 Analysis

The correctness of the algorithm is established by **Theorem 1** and **Theorem 2**.

During the insertion of a point, the procedure may introduce some triangles that are later removed. However, a triangle once deleted will never be re-introduced into the triangulation. Let T be the CDT, Q be a new point, T_Q be the set of the triangles t of T such that the circumcircle of t contains Q and Q is visible from all the vertices of t. It has been proved by Guibas, Knuth and Sharir [GKS90] that for any collection of N points (no distribution assumptions), if randomizing over the sequence of their insertion by the incremental algorithm, then the expected sum of $|T_Q|$ is only $O(n)$.

To insert Q to a CDT, first delete all the invalid edges, and then create edges between Q and all the points of the resulting polygon. Among thoses newly created edges, three edges connect Q to the points of the triangle enclosing Q. For the remaining new ones, there is one-to-one correspondence between a created edge and a deleted one. Thus the total number of edges created, denoted by $|E_Q|$, is equal to:

$$|E_Q| = |T_Q| + 3 - 1 = |T_Q| + 2.$$

Therefore, the expected sum of $|E_Q|$ is also $O(N)$.

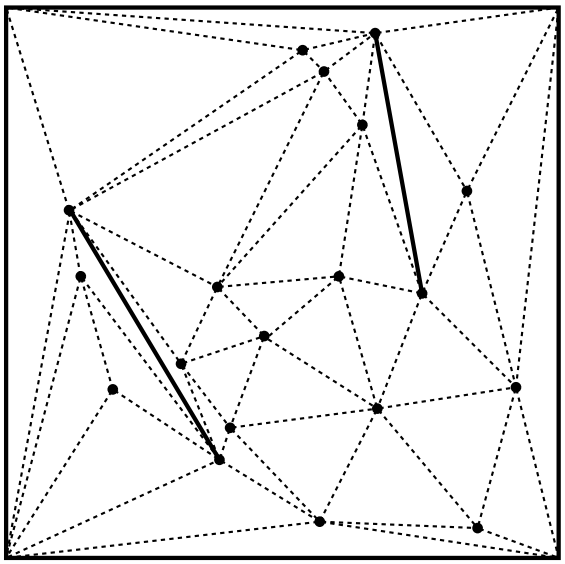
Since overall the total number of $|T_Q|$ is $O(n)$, the total number of $|E_Q|$ is $O(n)$, too. So, for each new point, the expected amortized number of new edges, or, the expected amortized number of edges deleted is $O(1)$.

Locating the triangle which encloses a point takes $O(\log N)$ time, the expected running time of our point insertion algorithm for constrained Delaunay triangulation is $O(N \log N)$, where N is the number of points in a CDT.

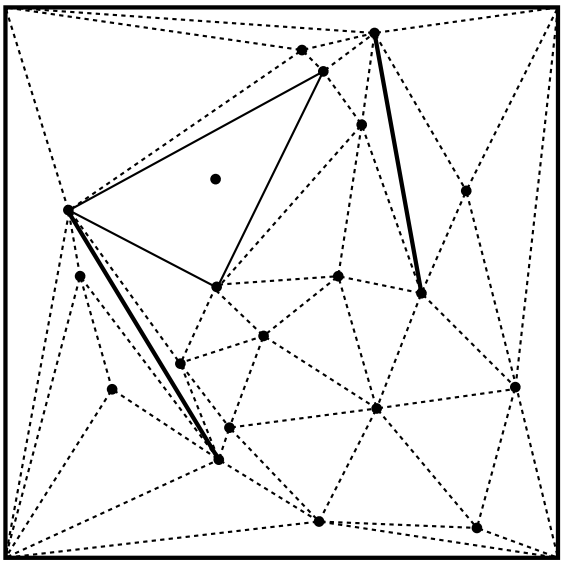
3.2 Inserting a constrained edge into a CDT

3.2.1 Algorithm

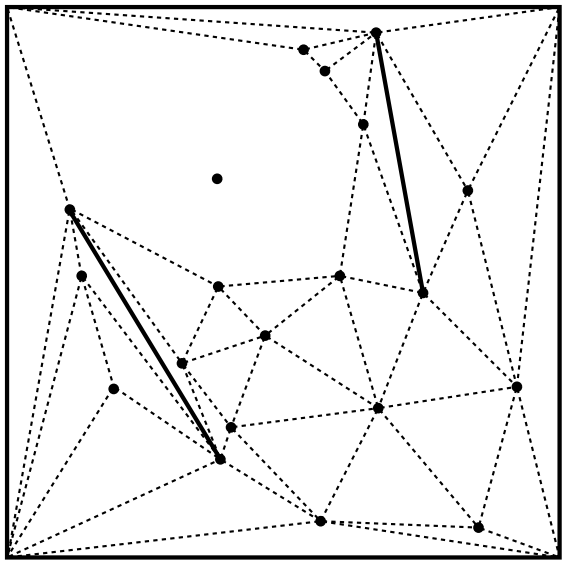
To insert a constrained edge, the two endpoints are inserted first, then the edge is added to the CDT and all the triangulation edges that intersect it are deleted. This results in two



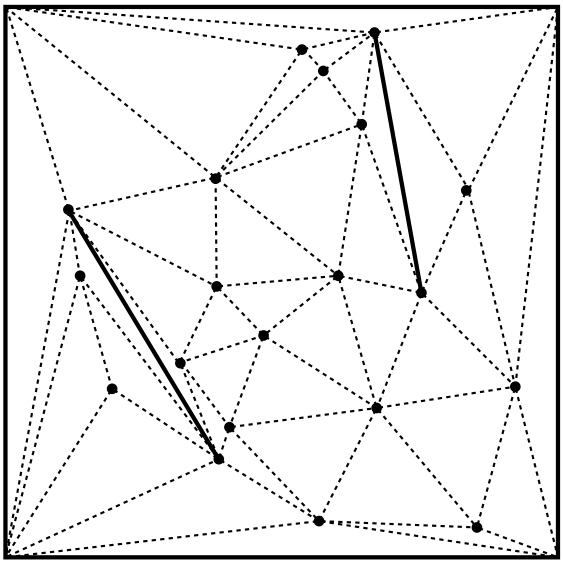
(a) Constrained Delaunay Triangulation



(b) Locate the triangle which encloses the new point



(c) Delete all the invalid edges



(d) The new constrained Delaunay triangulation

Figure 3.2: Scenario of point insertion algorithm

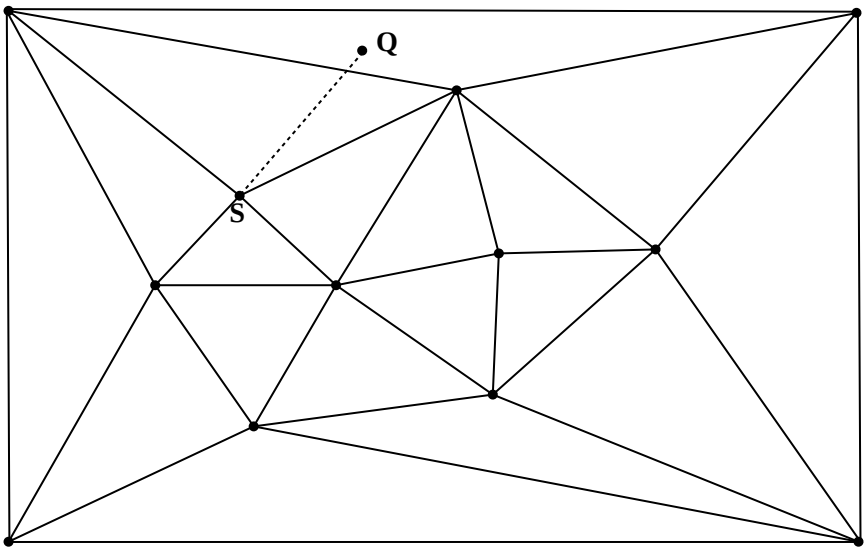


Figure 3.1: Point location

3.1.2 Inserting a point into a CDT

After the triangle that encloses the new point is found, the point insertion algorithm checks all the triangulation edges surrounding the new point and deletes those edges which violate the circle criterion of CDT (**Theorem** 1). Then, it connects the new point with all the points of the resulting polygon (**Theorem** 2).

The following algorithm updates a CDT to include the new point Q.

AddPoint(T, Q)
{
Locate the triangle which encloses the new point Q;
Get the new polygon P by deleting all the invalid triangulation edges;
Connect all the points of P to the new point Q.
}

Figure 3.2 shows snap-shots of various stages in the point insertion algorithm.

AddEdge: Given a CDT and a constrained edge, e, insert this new edge and update the CDT.

DeletePoint: Given a CDT containing a point, Q, delete Q and update the CDT.

DeleteEdge: Given a CDT containing a constrained edge, e, delete e and update the CDT.

In the following presentation, a CDT is denoted as T, a new point is denoted as Q, a constrained straight-line segment is denoted as $e = (v_a, v_b)$, and polygon is denoted as P. All the analysis that deals with polygon triangulation is postponed until the discussion of the algorithm which triangulates a polygon.

3.1 Inserting a point into a CDT

The point insertion algorithm consists of two steps: locating the triangle enclosing the new point and updating the CDT to include the new point.

3.1.1 Point location

The first step when adding a point to an existing CDT is to locate the triangle that encloses the new point. This triangle can be located by traversing the existing triangulation. In general, such a search can start from any point in the triangulation. However, the running time can be reduced by beginning the search at a point near the new point. In order to efficiently locate a starting point, an implementation stores the existing points in a quadtree. So the nearby points can be located in $O(\log N)$ time ([FB74]).

In figure 3.1, Q is the new point. Without loss of generality, assume we choose a starting point, S, in the vicinity of Q. Now, connect the new point Q and S with $\overline{SQ}$. By traversing the triangulation edges along line $\overline{SQ}$ we can finally locate the triangle which encloses the new point Q.

If Q is in the same location as a point in the triangulation, then Q will not be inserted into this triangulation again. If Q is on a triangulation edge, one of the two triangles which share the edge is returned.

3. Algorithms for Dynamic Constrained Delaunay Triangulation

This chapter describes the main algorithm for constructing a CDT. The algorithms use an incremental technique that constructs a CDT by incrementally inserting points and straight-line segments. These algorithms were developed with the CAD routing problem in mind, so the whole domain is bounded by a rectangle. The problem of incrementally computing a CDT is reduced to three subproblems: computing an initial triangulation of the rectangle, inserting a point, and inserting a constrained straight-line segment. Furthermore, to allow dynamic updating of the circuit layout, algorithms for deleting a point and deleting a constrained straight-line segment are also presented in this thesis. Every time a point or a constrained edge is inserted or deleted, the CDT is updated locally. An iterative triangulation algorithm is ideal for updating.

Since the algorithms presented in this thesis aim to improve the efficiency of the layout system, we are more interested in good average case performance for point insertion and deletion when the points are distributed almost uniformly.

Initially, the graph only consists of the bounding rectangle. This rectangle is triangulated by simply adding a diagonal edge.

The following algorithms will be described in detail in this chapter:

- Inserting a point into a CDT;
- Inserting a constrained straight-line segment into a CDT;
- Deleting a point from a CDT;
- Deleting a constrained straight-line segment from a CDT;
- Triangulating a polygon.

The main procedures are as follows:

AddPoint: Given a CDT and a new point, Q, insert Q and update the CDT.

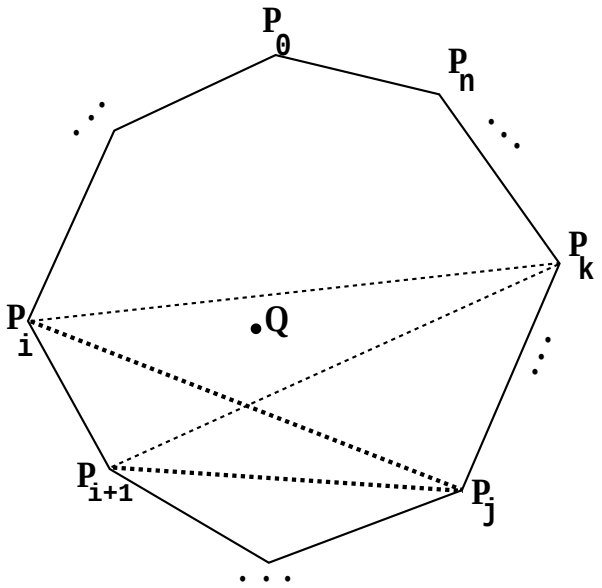


Figure 2.9: **Theorem 2**: connection

after Q is inserted, they are connected with point P_k $\{P_k | k \neq i,\ k \neq i+1,\ k \neq j\}$. Note here that no new constrained edge is introduced when point Q is inserted into the CDT. If P_i and P_{i+1} could be connected with P_k after Q is inserted, they could be so before Q is inserted as well. From **Lemma 4** we know, $\triangle P_i P_{i+1} P_j$ and $\triangle P_i P_{i+1} P_k$ can both be valid triangles if and only if points P_i, P_{i+1}, P_j, and P_k are co-circular. $\overline{P_i P_j}$ and $\overline{P_{i+1} P_j}$ were deleted because Q is in Circle$(P_i P_{i+1} P_j)$, that means Q is in Circle$(P_i P_{i+1} P_k)$. Hence, P_i and P_{i+1} can not be connected with P_k, otherwise the resulting triangulation won't satisfy the circle criterion of CDT.

Let T' be the CDT obtained from T by the insertion of Q, T'_Q be the triangulation of the resulting polygon P_Q. It is shown in [FP88] that $T' = T'_Q \cup T - T_Q$. So the only way to generate T_Q is connecting all the points of the P_Q with Q.

□

and D are not co-circular, at most one of $\triangle ABC$ and $\triangle ABD$ could be a valid triangle in CDT.

Now assume A, B, C and D are co-circular. C will be on the Circle(ABD) and D will be on the Circle(ABC). Since neither C is in Circle(ABD) nor D is in Circle(ABC), $\triangle ABC$ and $\triangle ABD$ could both be valid triangles in CDT (Note: In this case, each time only one of these two triangles will be in a CDT).

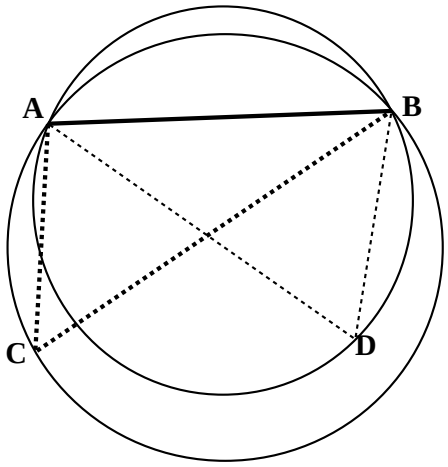


Figure 2.8: **Lemma 4**(continued)

□

From **Lemma 3** we know the new point Q could be connected with all the points of P_Q without crossing any other edges. Now, we will show that after connecting Q with all the points of P_Q, the resulting triangulation is CDT.

Theorem 2: If Q is the new point inserted in a CDT and P_Q is the resulting polygon after removing invalid edges, connecting Q to every point of P_Q will produce a valid CDT.

Proof: In Figure 2.9, assume that in the final CDT some of the points of P_Q are not connected with Q. Without loss of generality, assume P_i and P_{i+1} are the neighbor points of the polygon. Before Q is inserted, they were connected with point P_j $\{P_j | j \neq i,\ j \neq i{+}1\}$;

□

Lemma 3: After deleting the invalid edges as the result of adding a point, Q, to a CDT, Q is visible from all the points of the resulting polygon P_Q.

Proof:

The removal of the invalid edges is due to the fact that Q is in their checking circles. Thus, Q is visible from both endpoints of these edges. From **Lemma 2**, P_Q consists of exactly those points, so Q is visible from all the points of P_Q.

□

Lemma 4: Let $\overline{AB}$ be an edge in CDT. Points C and D are on the same side of $\overline{AB}$ (Figure 2.7), and they are visible from A and B. There are no points in Circle(ABC) and Circle(ABD). $\triangle ABC$ and $\triangle ABD$ are both valid triangles of CDT if and only if A, B, C, and D are co-circular.

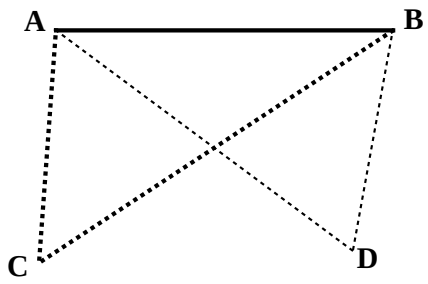


Figure 2.7: **Lemma 4**: co-circular

Proof:

Assume A, B, C, and D are not co-circular, and Circle(ABC) and Circle(ABD) intersect at $\overline{AB}$. On the side of C and D with respect to $\overline{AB}$, either Circle(ABC) encloses Circle(ABD), or Circle(ABD) encloses Circle(ABC) (Figure 2.8). If Circle(ABC) encloses Circle(ABD), then D is in Circle(ABC), $\triangle ABC$ could not be in CDT; if Circle(ABD) encloses Circle(ABC), then C is in Circle(ABD), $\triangle ABD$ could not be in CDT. So, if A, B, C

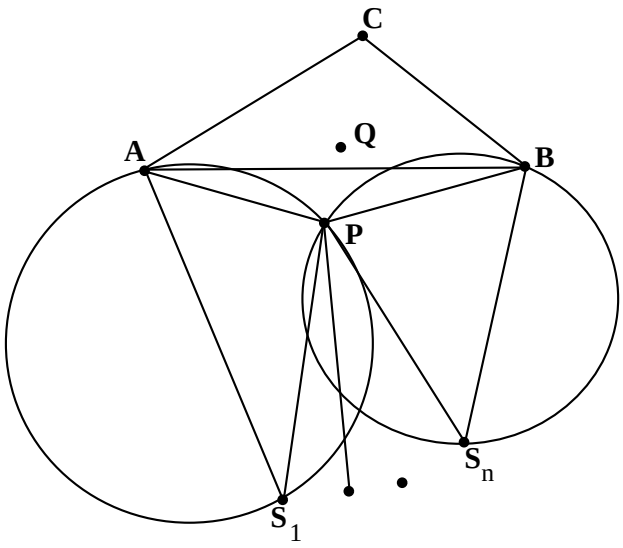


Figure 2.6: Empty polygon

Without loss of generality, assume edge $\overline{AB}$ is a triangulation edge, and Q is in the checking circle of $\overline{AB}$, $\overline{AB}$ will be deleted and the endpoints of $\overline{AB}$ will be the points of the new polygon $APBC$.

Any point in a triangulation has degree at least three, except the points on the boundary of the triangulation. The boundary of the triangulation is considered to consist of constrained edges, the boundary points cannot become isolated points.

Now $\overline{AP}$ and $\overline{BP}$ are the boundary edges of new polygon $APBC$. The checking circle of $\overline{AP}$ and the checking circle of $\overline{BP}$ must intersect at point P, and the intersecting area must be outside of polygon $APBC$. This means Q cannot be inside the intersecting area. If Q is inside the checking circle of $\overline{AP}$, Circle(APS_1), $\overline{AP}$ will be deleted, but $\overline{BP}$ and $\overline{PS_1}$ will not be deleted. So, at most one of the edges $\overline{AP}$ or $\overline{BP}$ could be deleted, P is still connected with other points, and this could not result in an isolated point. The same argument can be applied to other points (points other than Q).

Therefore there is only one isolated point, Q, in the resulting polygon P_Q after deleting all the invalid edges. The points of P_Q are the endpoints of the checked edges, such as $\overline{BP}$ (if $\overline{BP}$ is in P_Q).

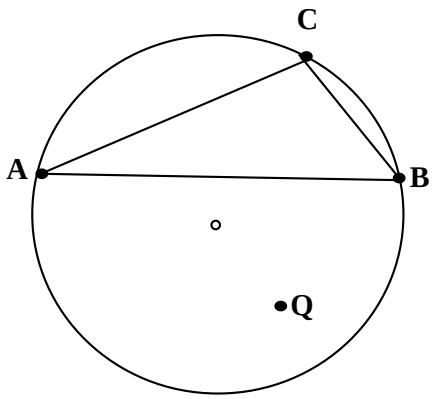


Figure 2.5: **Theorem 1**: circle criterion

□

All the invalid edges will be deleted after Q is added to T. Let T_Q be the set of the triangles t of T such that the circumcircle of t contains Q and Q is visible from all the vertices of t. The resulting polygon P_Q of point Q is formed by the external edges (the edges not shared by two triangles in T_Q) of the triangles in T_Q.

Checking Circle *Let $\overline{AB}$ be an edge of the polygon which encloses the query point Q. $\overline{AB}$ belongs to the unique triangle $\triangle ABC$ not containing Q. The circumcircle of $\triangle ABC$, Circle(ABC), is referred to as the checking circle of $\overline{AB}$.*

Lemma 2: After deleting the invalid edges as the result of adding a point Q to a CDT, the resulting polygon P_Q of Q contains no other isolated points aside from Q, and this polygon consists of the endpoints of all the checked edges which passed the circl criterion check in the process of deleting the invalid edges.

Proof:

In Figure 2.6, point Q is the new point inserted into a triangle $\triangle$ABC of a CDT T. If all the three edges of $\triangle ABC$ are constrained edges, then the polygon is $\triangle ABC$, and Q is the only point inside it.

$$a = |\overline{BD}|, \quad b = |\overline{OD}|.$$

Assume the perpendicular from C intersects the bisector at E,

$$c = |\overline{CE}|, \quad d = |\overline{EO}|.$$

At the very beginning,

$$a^2 + b^2 = c^2 + d^2.$$

Now assume after moving the Circle(ABC) toward C along the bisector line of $\overline{AB}$, the center of the new circle is O', and

$$\Delta = |\overline{OO'}|.$$

The new radius of the circle

$$r^2 = |\overline{BO'}|^2 = a^2 + (b - \Delta)^2,$$

$$|\overline{CO'}|^2 = c^2 + (d - \Delta)^2,$$

$$r^2 - |\overline{CO'}|^2 = 2(d - b)\Delta > 0.$$

So $|\overline{CO'}|$ is always shorter than r. C will be inside the new circle.

□

Theorem 1: Let $\overline{AB}$ be a triangulation edge of a CDT and point C is visible from both A and B. After adding a point Q on the side of edge $\overline{AB}$ opposite to C, $\overline{AB}$ is no longer a triangulation edge of the resulting CDT if and only if Q is in the Circle(ABC) and Q is visible from A and B.

Proof:

Assume as in Figure 2.5 that Q is in the Circle(ABC) and it is visible from A and B. If Q is on $\overline{AB}$, $\overline{AB}$ is no longer valid; otherwise, from **Lemma 1**, move the center of Circle(ABC) along the bisector of $\overline{AB}$ away from Q, C will be inside the new circle; or, move the center of Circle(ABC) along the bisector of $\overline{AB}$ away from C, Q will be inside the new circle. Therefore, either C or Q will be inside the circumcircle of $\overline{AB}$. From the definition of CDT, $\overline{AB}$ is no longer a triangulation edge now.

of CDT T are treated as triangulation edges. The circumcircle of points A, B, and C will be denoted as Circle(ABC). We assume that no four points of the original CDT are co-circular, hence, the CDT is unique. If this assumption is not true, inconsequential but lengthy details must be added to the statement and proofs.

After a new point Q is added to a CDT T, the triangle which encloses Q is located. Then all the edges surround Q are checked against the circle criterion.

Lemma 1: A, B, and C are three points on a plane. Let O be the center of Circle(ABC), F be the infinity point on the side of $\overline{AB}$ where C is on. If we move O any amount toward F along the bisector of $\overline{AB}$ while varying the radius of the circle in order to keep A and B on the boundary of the circle, point C will be inside the new circle.

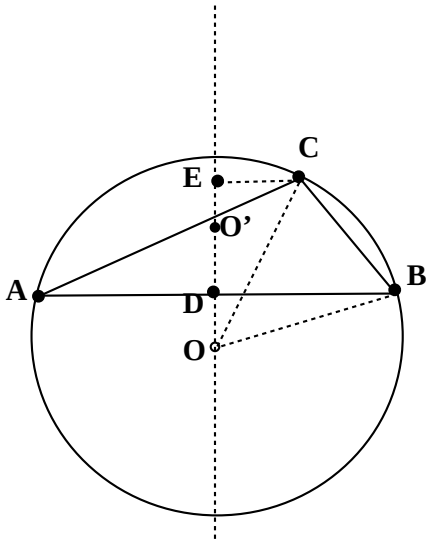


Figure 2.4: **Lemma 1**: C is inside Circle(ABD)

Proof:

In Figure 2.4, A, B and C are three points in a plane. Now move the center of the Circle(ABC) as described in the lemma. Assume the bisector intersects $\overline{AB}$ at D. Let's denote

Definition 2: (Constrained Delaunay Triangulation)(CDT): Let S be a set of points and a set of non-intersecting straight-line segments connecting some pairs of points in the plane. A triangulation T is a constrained Delaunay triangulation of S if each edge of S is an edge of T and for each remaining edge e of T there exists a circumcircle c with the following properties:

(1) The endpoints of edge e are on the boundary of c.

(2) If any point v of S is in the interior of c then it is not visible from at least one of the endpoints of e.

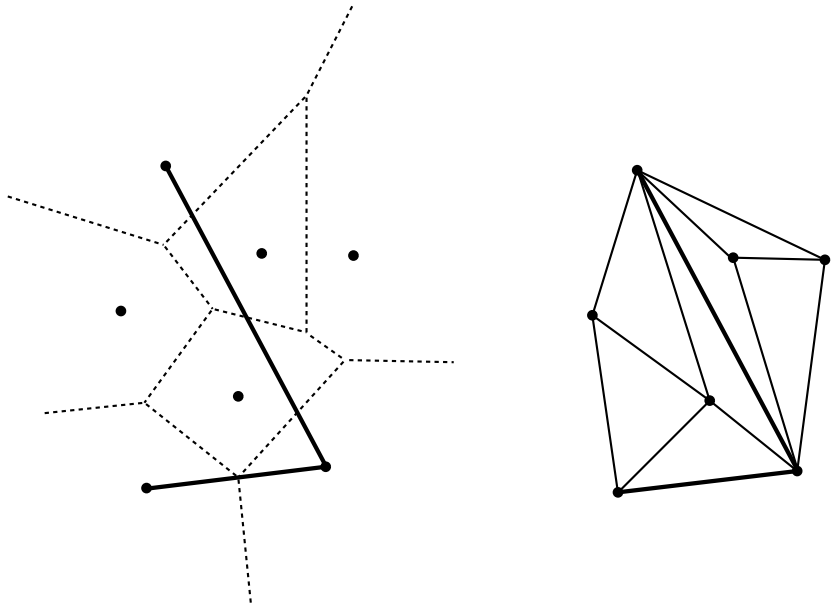


Figure 2.3: Constrained Delaunay triangulation

It has been shown that given a planar graph, it is always possible to construct a constrained Delaunay triangulation ([FFP85,LL86,FP88,Che89,LD91]). A set of dynamic CDT algorithms is presented in this thesis. Each time a point or a constrained edge is added or deleted, the algorithms will locally update the CDT. To show the algorithms update the CDT appropriately, we have the following lemmas and theorems. In the following presentation, the edges of S will be treated as constrained edges, and the remaining edges

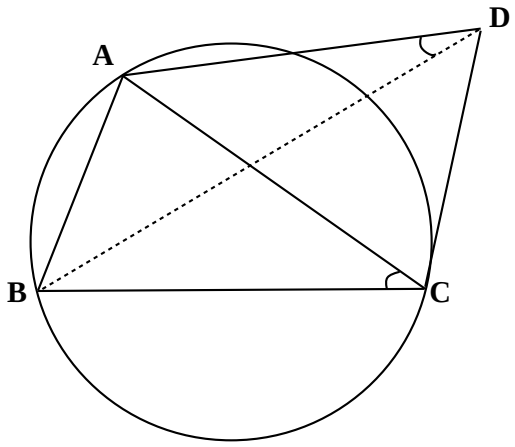


Figure 2.2: Max-min angle criterion and circle criterion

Only a Delaunay triangulation satisfies this criterion[Sib78]. An example is given in Figure 2.2, where $\triangle ABC$ is a triangle of Delaunay triangulation, and $\angle BCA$ is larger than $\angle BDA$.

The second criterion is the **circle criterion**:

> *A triangulation T of S is a Delaunay triangulation if and only if no point of S is inside the circumcircle of any triangle in T.*

We can construct the Delaunay triangulation based on these two criteria. It has been proved by R. Sibson [Sib78] that if the max-min angle criterion is used, the resulting triangulation must also satisfy the circle criterion, and vice versa. This can be illustrated by the simple example in Figure 2.2 where point D is not in Circle(ABC), $\angle BCA$ is larger than $\angle BDA$, and $\overline{AC}$ is a triangulation edge in Delaunay triangulation not $\overline{BD}$.

When there are prespecified edges, these constraints have to be incorporated into the triangulation (Figure 2.3). The triangulation is as close to the Delaunay triangulation as possible. For constrained Delaunay triangulation, the max-min angle criterion is not affected by the introduction of these constraints, but the definition of circle criterion must be changed slightly.

2. Preliminaries and Theorems

The concepts of Delaunay triangulation and constrained Delaunay triangulation were introduced in the last chapter. Their formal definitions are now presented.

Definition 1: (Delaunay Triangulation)(DT): Let S be a set of points in the plane (Figure 2.1). A triangulation T is a Delaunay triangulation of S if for each edge e of T there exists a circumcircle c with the following properties:

(1) The endpoints of edge e are on the boundary of c.

(2) There is no other point of S in the interior of c.

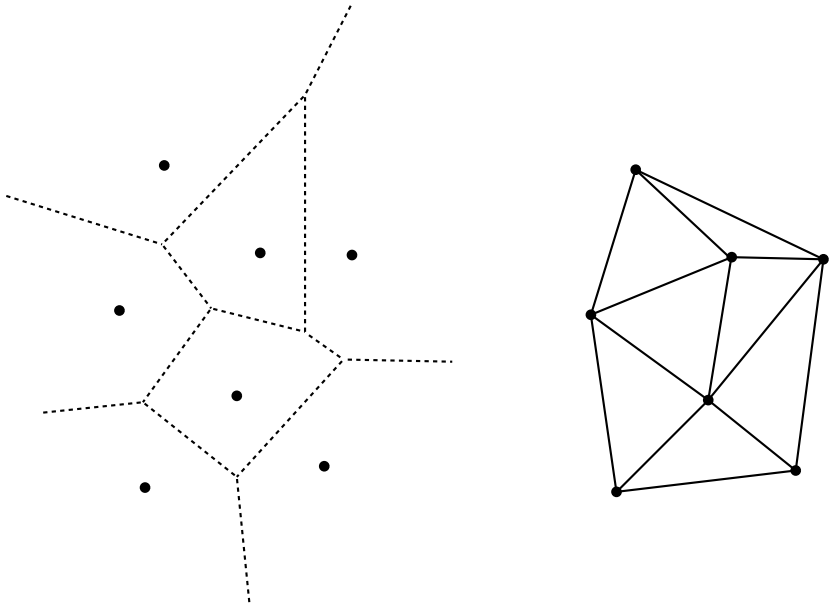


Figure 2.1: Delaunay triangulation

Delaunay triangulation satisfies two important criteria. The first is called **max-min angle criterion**:

> *The smallest angle of the triangles in the Delaunay triangulation is maximum among all possible triangulations.*

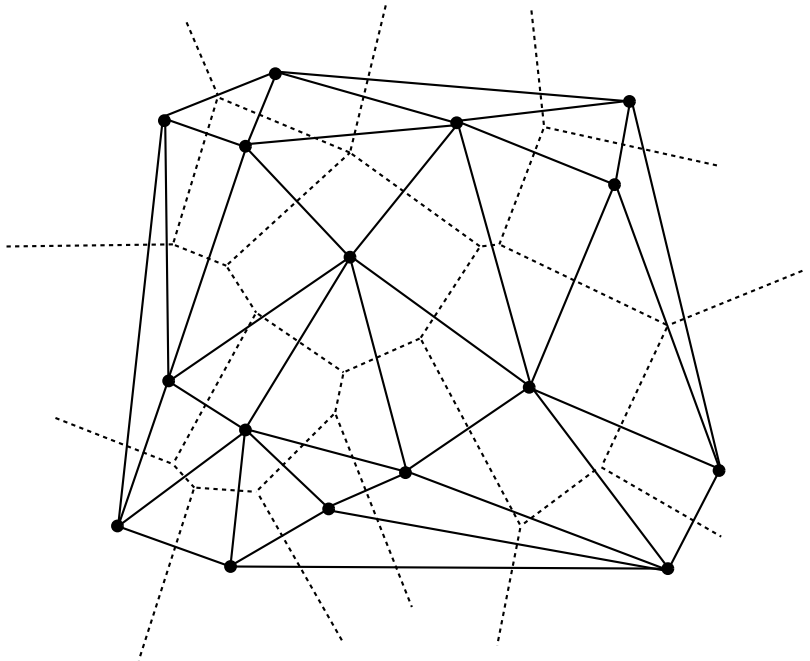


Figure 1.4: Delaunay triangulation

gorithm could fail in practical implementation. In this thesis, a technique is presented to avoid the rounding errors.

The thesis is organized as follows: Chapter 2 presents some important properties and theorems of constrained Delaunay triangulation, which are the basis of the algorithms. Chapter 3 describes the algorithms that incrementally construct and update the constrained Delaunay triangulation. The precision problem is discussed in Chapter 4. The results of the implementation are shown in Chapter 5. Chapter 6 discusses some related work in this area, and Chapter 7 summarizes the results and discusses the future work.

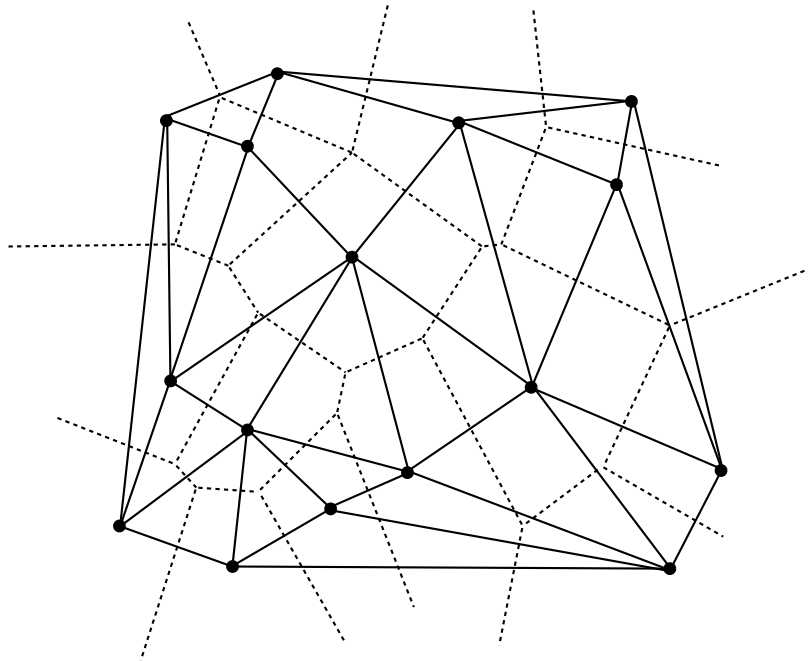


Figure 1.3: Pretriangulation

the Euclidean distance between p and q, $DT(p,q)$ be the length of the shortest path between p and q in the Delaunay triangulation of S. Then,

$$\frac{DT(p,q)}{d(p,q)} \leq \frac{2\pi}{3\cos\frac{\pi}{6}} \approx 2.42.$$

Triangles with very small angles produce a poor mesh ([Fri72,BEG90]). A Delaunay triangulation is an excellent choice for the finite element mesh generation ([ZSZZ90]). The Delaunay triangulation has many applications. It can be used to determine the closest pair of points and the closest neighbor of each point in linear time ([LP78,YL79]). Every minimum spanning tree is a subgraph of Delaunay triangulation ([SH75]), based on Delaunay triangulation we can also get the Euclidean minimum spanning tree of n points ([CT76]).

Since all the algorithms that build Delaunay triangulations use either angle or circle calculations[1], rounding errors should be considered; otherwise, a theoretically correct al-

[1]See Chapter 2 for the explanation of *max-min angle criterion* and *circle criterion*

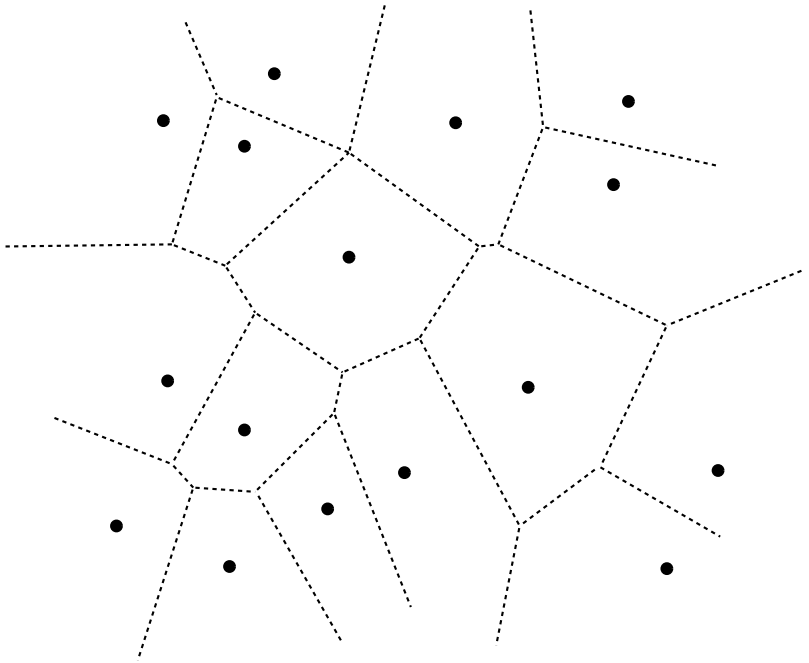


Figure 1.2: Voronoi diagram

(Figure 1.3). Given a Voronoi diagram V, the construction rules for its dual graph T (pre-triangulation) are as follows. A point of T is associated with each point of V. For each edge e_i of V there is an associated edge e_i^* of T. If e_i separates the faces f_i and f_k in V, then e_i^* connects the two points of T with f_i and f_k. If no four points are co-circular (on the same circle), the Delaunay triangulation is equivalent to the straight-line dual; otherwise, the dual contains non-triangular polygons, these polygons can easily be decomposed into non-overlapping triangles without violating the properties of the Delaunay triangulation (Figure 1.4). As an algorithmic tool, the Delaunay triangulation is one of the central topics in computational geometry.

It is known that if searching a shortest path between two points based upon the Delaunay triangulation, the ratio of the length of the shortest path between two points in the Delaunay triangulation to the Euclidean distance is bounded[KG89]:

Let p and q be a pair of points in a set S of N points in a plane. Let $d(p,q)$ be

and assigning cross points to the cut line for each crossing net. After global routing, a local router is applied to each bin. The local router determines a topological routing (rubber-band sketch) by using a minimum cost search to connect the points of each net within the bin.

A triangulation of a set of points in a plane is a planar, straight-line graph that includes the edges of the convex hull of the points and in which pairs of points are joined by straight-line segments. These segments may intersect only at their endpoints, and every region internal to the convex hull is a triangle. A triangulation as underlying data representation is preferred because of the ease with which it can be made to fit complex boundaries and obstacles.

The rubber-band sketch requires an efficient dynamic data representation, and the local routing requires the nearest neighbor information to find the minimum cost paths. The triangulation required here should have the nearest neighbors information, so the shortest path can be easily generated in $O(N \log N)$ time instead of $O(N^2)$ time, where N is the number of points. If there are obstacles (prespecified wires, pads, components, etc.) on the routing surfaces, they will be transformed to the straight-line wires in the rubber-band sketch, and these wires are called *constrained edges*. The constrained edges should be a part of the triangulation to keep the complete routing topology. In this case, the triangulation will have the nearest *visible* neighbors information. Here a point p is *visible* from point q if the open line segment $\overline{pq}$ does not intersect any constrained edges in the triangulation. The only triangulation which meets all these requirements is called the *constrained Delaunay triangulation*.

The classical definition of Delaunay triangulation is expressed in terms of the *Voronoi diagram* (also called the *Dirichlet tessellation* or the *Thiessen tessellation*) (Figure 1.2). Given a set of points, S, in a plane, the Voronoi diagram is a partition of the plane into *Voronoi* regions, each region being the locus of the points (x, y) closer to a point of S than to any other point of S [PS85].

The straight-line dual of the Voronoi diagram is called the *pretriangulation* by Sibson[Sib78]

1. Introduction

The SURF multichip module (MCM) routing system employs a gridless routing topology called the *rubber-band sketch*. Wires represented by rubber-bands are flexible and may be stretched and bent around objects. A rubber-band sketch represents a topological routing of one layer. It consists of a set of points, P, and a finite set of interconnecting *wires*, W. A *rubber-band wire* in W is a simple path between two points in P with minimum length. No two wires may intersect and no wire may cross itself (Figure 1.1).

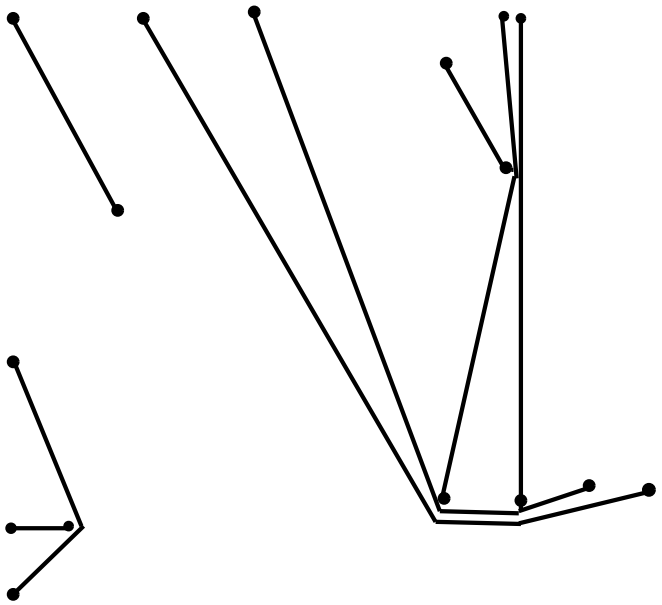


Figure 1.1: Rubber-band sketch

Given a set of interconnections (*nets*), a rubber-band sketch is generated by a topological routing process. This routing defines paths on the wiring surface of the carrier (printed circuit board, intergrated circuits or multichip module substrates), which realizes electrical connections necessary to form a given circuit function. It is performed in two steps: global routing and local routing. The global router begins with a rectangular bounding box which encloses all the points in a routing problem, then partitions the problem into a set of smaller rectangles, called *bins*. This partitioning is done by recursively splitting the bins in two,

List of Figures

Contents

Acknowledgements

I would like to express my gratitude to the criticisms, patience, resources and continued support I've received from professor Wayne W. Dai. I would also like to thank professor Martine Schlag and professor David Haussler for serving on my thesis reading committee and giving me their valuable comments. I thank David Staepelaere for his comments about my thesis, and Dr. Marshall Bern for the valuable references he gave to me. Special thanks go to my parents, my husband, Jian-Zhong, and my lovely daughter, Diane, for their consistent support, sacrifice, cooperation, and encouragement.

Abstract

The *Voronoi diagram* is a partition of a set S of N points in a plane, such that each region is the locus of the points (x, y) closer to a point of S than to any other point of S. If no four points are co-circular, the *Delaunay triangulation* is the straight-line dual of the Voronoi diagram. The triangulation may be *constrained*, that is, a set of straight-line segments may be prespecified.

This thesis presents some characteristics of constrained Delaunay triangulation and introduces a set of numerically stable algorithms for incremently constructing and updating constrained Delaunay triangulation.

The dynamic constrained Delaunay triangulation algorithms have been implemented in a layout system for multichip modules. It has been used as the underlying data representation for rubber-band sketch, a topological routing for one layer.

We have proved the $O(n \log n)$ expected running time for the Delaunay triangulation algorithm.

keywords: Voronoi diagram, Delaunay triangulation, constrained Delaunay triangulation, layout, multichip module, topological routing

University of California

Santa Cruz

Dynamic Constrained Delaunay Triangulation and Application to Multichip Module Layout

A thesis submitted in partial satisfaction

of the requirements for the degree of

Master of Science

in

Computer Engineering

by

Yizhi Lu

December 1991

The thesis of Yizhi Lu is

approved by:

Wayne Wei-Ming Dai

Martine Schlag

David Haussler

Dean of Graduate Studies and Research