Point location in planar subdivisions

Michiel Smid*

September 25, 2003

1 Introduction

In these notes, we consider one of the basic problems in computational geometry, the *planar point location problem*. Consider a plane embedding of a connected planar graph G. For simplicity, we also denote this embedding by G; it consists of vertices, edges, and faces. The edges are drawn as straight-line segments and any two of them do not cross in their interiors. We denote the number of vertices of G by n.

We want to store the embedding G in a data structure, such that for any query point $q \in \mathbb{R}^2$, we can find the face that contains q. If q is on the boundary of several faces, then it suffices to find one of them. Such queries are called *point location queries*, see Figure 1.

In these notes, we will give two solutions for the point location problem. The first solution is known as the *slab method* and is given in Section 2. Using this method, we can solve queries optimally, i.e., in $O(\log n)$ time. The data structure, however, uses $\Theta(n^2)$ space in the worst case. The second solution, the *triangulation refinement method*, is given in Section 3. It gives a data structure using only $O(n)$ space, that can be used to solve point location queries in $O(\log n)$ time.

Recall the following result.

Theorem 1 *Let G be a non-crossing embedding of a connected planar graph with $n \geq 3$ vertices. Then*

*School of Computer Science, Carleton University, Ottawa, Ontario, Canada K1S 5B6. E-mail: michiel@scs.carleton.ca.

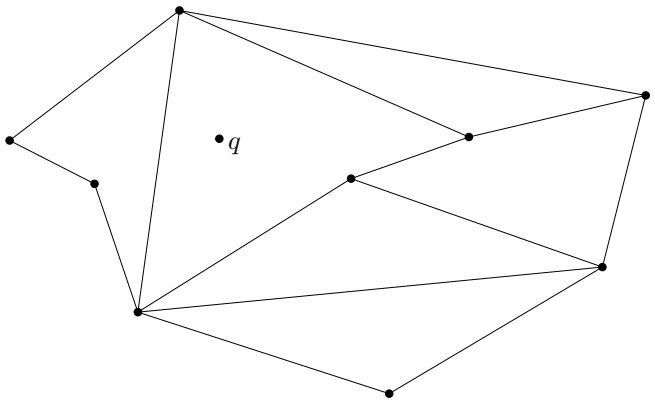

Figure 1: *In a point location query, we want to find the face of the embedding of the planar graph that contains the query point q.*

---

1. *G has at most $3n - 6$ edges and*
2. *this embedding has at most $2n - 4$ faces (including the unbounded face).*

2 The slab method

This method for solving the point location problem, due to Dobkin and Lipton (1976), is very simple. We draw a vertical line through each vertex of G. This divides the plane into at most $n + 1$ vertical slabs, which we sort from left to right.

Consider any slab s. Observe that s does not contain any vertex of G in its interior. Let E_s be the set of all edges in the graph G that cross s. The edge set E_s partitions s into (possibly unbounded) trapezoids and triangles. We sort the edges of E_s from bottom to top. With each edge, we store the names of the two faces of G that are immediately below and above it.

Consider any query point $q \in \mathbb{R}^2$. To find the face of G that contains q, we first use binary search with the x-coordinate of q to find the vertical slab s that contains q. Given s, we use binary search with the y-coordinate of q to find the edges of E_s between which q lies. This gives the name of the face of G that contains q.

It is clear that in this way, we correctly solve the point location problem. How much time do we need to locate a query point? Clearly, each slab is

crossed by at most $3n - 6$ edges, because the entire graph has at most this many edges. Hence, in a query, we make two binary searches, one in a set of size less than or equal to n, the other in a set of size less than or equal to $3n - 6$. As a result, it takes $O(\log n)$ time to locate a query point.

Since there are at most $n + 1$ slabs, each of which is crossed by at most $3n - 6$ edges, we need $O(n^2)$ space to store the entire data structure. Finally, it is easy to see that the data structure can be built in $O(n^2 \log n)$ time. We have proved the following result.

Theorem 2 *For the point location problem in a connected planar graph with n vertices, the slab method gives a data structure having query time $O(\log n)$, size $O(n^2)$, and that can be built in $O(n^2 \log n)$ time.*

Exercise 1 Give an example of a planar graph for which the above data structure has size $\Omega(n^2)$.

3 The triangulation refinement method

The second method, due to Kirkpatrick (1983), gives a data structure that solves the point location problem optimally.

Let G be (an embedding of) a connected planar graph with n vertices, whose edges are non-crossing straight-line segments. We assume that

1. each bounded face of G is a triangle and
2. the unbounded face of G is bounded by a triangle. We call this triangle the *outer triangle* of G.

Hence, G is a triangulation and the boundary of its convex hull is a triangle; see Figure 2. We will give an optimal data structure for solving the point location problem for this graph. In Section 3.1, we will show how to extend this to an optimal solution for any planar graph.

Exercise 2 Since G is a connected planar graph with n vertices, we know from Theorem 1 that it has at most $3n - 6$ edges and at most $2n - 4$ faces (including the unbounded face). Prove that G has exactly $3n - 6$ edges and $2n - 4$ faces. (*Hint:* Use the fact that the boundary of G's convex hull contains exactly three vertices.)

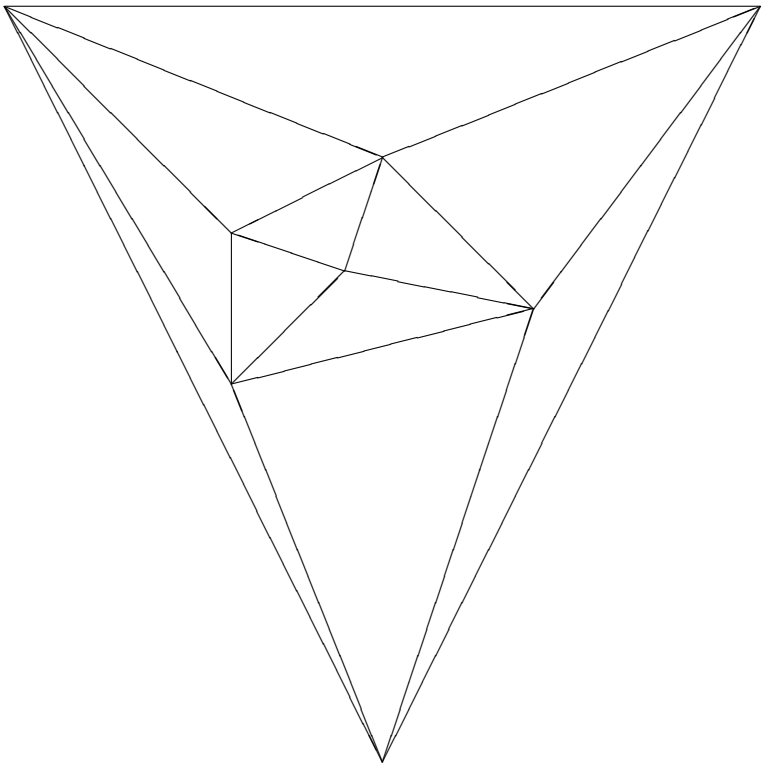

Figure 2: *An example of a triangulation.*

First, we describe the *basic idea* of the triangulation refinement method. We compute a sequence $S_1, S_2, \ldots, S_h$ of triangulations, such that

1. $S_1 = G$,
2. S_h is the outer triangle of G,
3. $h = O(\log n)$,
4. there is a constant α with $0 < \alpha < 1$, such that for each i with $1 \leq i < h$, the number of vertices of S_{i+1} is at most α times the number of vertices of S_i, and
5. for each i with $1 \leq i < h$, and any query point $q \in \mathbb{R}^2$, if we have located q in S_{i+1}, then we can locate q in S_i in $O(1)$ time.

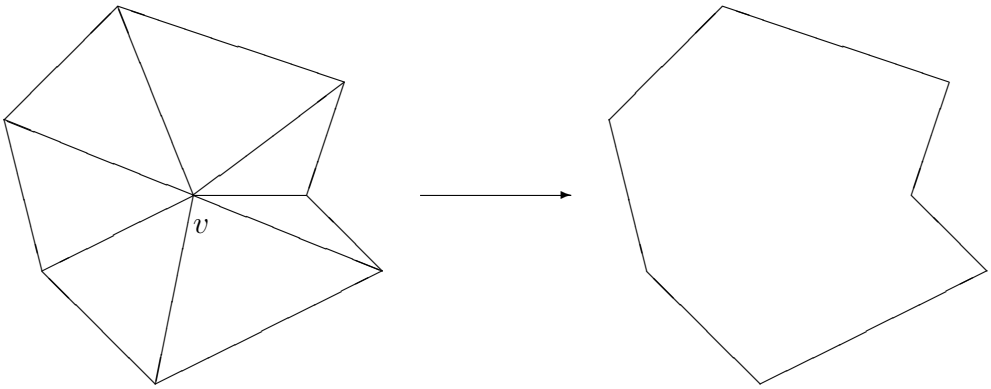

Figure 3: *Removing a vertex from a triangulation.*

Given this sequence, we can solve the point location problem for G as follows. First, we test if q is inside or outside the triangle S_h. If it is outside, then q is in the outer face of G, and we are done. Assume q is inside S_h. Then, we locate q in the sequence $S_{h-1}, S_{h-2}, \ldots, S_1 = G$ of triangulations (in this order). Since $h = O(\log n)$, the fifth property implies that this gives a query time of $O(\log n)$. By the fourth property, the total size of the entire sequence of triangulations is bounded by a geometric series whose sum is $O(n)$.

How do we compute the sequence of triangulations? Consider $S_1 = G$. The triangulation S_2 should be obtained by *removing* a constant fraction of the vertices of G. Let v be a vertex of G that is not a vertex of the outer triangle, and let it have degree d. Let us consider what happens if we remove v, together with its d adjacent edges, from G. These edges determine d triangles of G. Hence, if we remove v, these d triangles are replaced by a simple polygon D having d vertices. We call D a *d-gon*. (Refer to Figure 3 for an example with $d = 7$. In this figure, only the part of G that is incident to v is shown.)

Let q be any point in D. Then, by considering each of the d triangles of G whose union forms D, we can find the triangle that contains q in $O(d)$ time. Hence, if we only remove vertices v of *small* degree, then we can guarantee that the fifth property above holds. For the fourth property to hold, we must remove *many* such vertices.

This leads to the question whether there always exist *many* vertices of

small degree. Moreover, if they exist, how do we find them? The answers are given in the following two lemmas.

Lemma 1 *Consider the triangulation G having n vertices. In G, there are at least $2 + n/2$ vertices of degree less than or equal to eight.*

Proof. For any vertex v of G, let $deg(v)$ denote its degree. Since G is a triangulation, we have $deg(v) \geq 3$ for each vertex v. Also, $\sum_v deg(v)$ is equal to twice the number of edges of G. Hence, by Exercise 2,

$$\sum_v deg(v) = 6n - 12. \tag{1}$$

Let A be the subset of those vertices of G having degree less than or equal to eight. Then

$$\sum_v deg(v) = \sum_{v \in A} deg(v) + \sum_{v \notin A} deg(v) \geq 3|A| + 9(n - |A|) = 9n - 6|A|.$$

Combining this with (1), it follows that $9n - 6|A| \leq 6n - 12$, which implies that $|A| \geq 2 + n/2$. ∎

A subset I of the vertices of G is called an *independent set*, if no two vertices of I are connected by an edge of G.

Lemma 2 *The graph G contains an independent set I of size at least $\lceil \frac{1}{9}(\frac{n}{2} - 1) \rceil$, such that each vertex of I (i) has degree at most eight and (ii) is not a vertex of the outer triangle. Such a set I can be computed in $O(n)$ time.*

Proof. The following algorithm computes the set I.

mark the three vertices of the outer triangle;
$I := \emptyset$;
while there are unmarked vertices of degree at most 8
do let v be any vertex of degree at most 8 that is not marked;
    $I := I \cup \{v\}$;
    mark v and all its neighbors
endwhile

Observe that the degree of the vertices does not change because of the marking. It is not difficult to see that this algorithm can be implemented such that it runs in $O(n)$ time. Also, it is easy to see that the algorithm computes an independent set (the set I) of vertices that are not vertices of the outer triangle, each one having degree less than or equal to 8. Therefore, it remains to show that the size of I is greater than or equal to $\lceil \frac{1}{9}(\frac{n}{2} - 1) \rceil$.

By Lemma 1, G contains at least $2 + n/2$ vertices of degree at most 8. The algorithm starts by marking the three vertices of the outer triangle. At that moment, there are at least $n/2 - 1$ unmarked vertices of degree at most 8. The algorithm takes one of these vertices and marks it, together with its at most 8 neighbors. This is repeated as long as there are unmarked vertices of degree at most 8. Since at most 9 vertices are marked during one iteration, there must be at least $\lceil \frac{1}{9}(\frac{n}{2} - 1) \rceil$ iterations. During each of them, one vertex is added to I. ∎

We state one more property of the independent set I of Lemma 2. The proof follows from the discussion above.

Lemma 3 *Let I be the independent set that is constructed in the proof of Lemma 2. Let G' be the graph that is obtained from G by removing all vertices of I, together with their incident edges. Then*

1. *each face of G' is a simple polygon having at most 8 vertices,*
2. *each face of G' intersects at most 8 triangles of G, and*
3. *the outer faces of G and G' are the same.*

Exercise 3 Why do we remove an independent set from the triangulation G and not just an arbitrary set of vertices of degree at most 8? Why don't we remove any of the three vertices of the outer triangle of G?

Now we can give the algorithm for constructing the sequence $S_1, S_2, \ldots, S_h$ of triangulations, and the corresponding data structure for solving point location queries. The data structure will be a directed acyclic graph. The nodes of this graph represent triangles of the triangulations $S_1, S_2, \ldots, S_h$. The node representing triangle t will be denoted by $n(t)$. The number of vertices of triangulation S_i will be denoted by $|S_i|$.

One remark about the terminology. Each triangulation in the sequence $S_1, S_2, \ldots, S_h$ consists of *vertices*, *edges*, and *faces* (which are triangles). The

point location data structure is a directed acyclic graph consisting of *nodes* and *arcs*.

The algorithm is given in Figure 4. The query algorithm follows easily. The final triangulation S_h consists of one triangle. Its node in the data structure has no incoming arcs, and this is the only node having in-degree zero. We call it the *root* of the data structure. Let $q \in \mathbb{R}^2$ be any query point. The algorithm that finds the face of G containing q is given in Figure 5.

Lemma 4 *The query algorithm is correct.*

Proof. The while-loop maintains the invariant that q is contained in the triangle represented by node v. If v has no outgoing arcs, then this triangle is a face of G. ∎

Hence, our data structure correctly solves the point location problem. It remains to analyze its complexity.

Lemma 5 *Let h be the number of triangulations constructed by the algorithm in Figure 4. For each i with $1 \leq i \leq h$, let n_i be the number of vertices of the triangulation S_i. Then*

1. *$n_1 = n$ and*
2. *for each i with $1 \leq i < h$, $n_{i+1} \leq \frac{17}{18} n_i + \frac{1}{9}$.*

Proof. Lemma 2 implies that

$$n_{i+1} \leq n_i - \left\lceil \frac{1}{9} \left(\frac{n_i}{2} - 1 \right) \right\rceil \leq n_i - \frac{1}{9} \left(\frac{n_i}{2} - 1 \right) = \frac{17}{18} n_i + \frac{1}{9}.$$

∎

Lemma 6 *The algorithm in Figure 4 computes a sequence of $O(\log n)$ triangulations. The data structure constructed by this algorithm is a directed acyclic graph and each path in this graph has length $O(\log n)$.*

Proof. This follows from the algorithm and Lemma 5. Observe that $n_1 = n$ and $n_h = 3$. ∎

Lemma 7 *The data structure has size $O(n)$ and can be built in $O(n)$ time.*

$S_1 := G$;
for each triangle t of G
do create a node $n(t)$ representing t
endfor;
$i := 1$;
while S_i has more than three vertices
do compute an independent set I in S_i containing at least
$\lceil \frac{1}{9}(\frac{|S_i|}{2} - 1) \rceil$ vertices that are not vertices of the outer triangle,
each one having degree at most 8;
remove the vertices of I from S_i, together with their incident
edges;
triangulate the resulting graph and call the result S_{i+1};
($*$ we triangulate because then we can apply Lemma 2
during the next iteration $*$)
for each triangle t of S_{i+1} that is not a triangle of S_i
($*$ hence, t is a new triangle $*$)
do create a node $n(t)$ representing t;
for each node $n(t')$ such that triangle t' belongs to S_i and
intersects t
do add a directed arc from $n(t)$ to $n(t')$
endfor
endfor;
$i := i + 1$
endwhile

Figure 4: *Constructing the sequence of triangulations and the corresponding point location data structure.*

```
if q is outside the triangle represented by the root
then output "q belongs to the unbounded face of G"
else v := root;
     while v has outgoing arcs
     do for each node u such that there is an arc from v to u
        do if q is inside the triangle represented by u
           then v := u
           endif
        endfor
     endwhile;
     output "q belongs to the face of G represented by v"
endif
```

Figure 5: *Finding the face that contains the query point q.*

Proof. By Exercise 2, triangulation S_i has less than $2n_i$ faces, $1 \leq i \leq h$. Therefore, the total number of nodes of the data structure is bounded from above by $\sum_{i=1}^{h} 2n_i$. This summation is bounded from above by a geometric series with sum $O(n_1) = O(n)$. Observe that we also have to count the number of arcs of the data structure. Consider the independent set I in S_i. By Lemma 3, each face of $S_i \setminus I$ intersects at most 8 triangles of S_i. Since S_{i+1} is obtained by triangulating $S_i \setminus I$, it is clear that each triangle of S_{i+1} intersects at most 8 triangles of S_i. This proves that each node in the data structure has at most 8 outgoing arcs. As a result, the total number of arcs is $O(n)$. Hence, the entire data structure has size $O(n)$.

The bound on the building time follows in a similar way. We know already that the independent set I in S_i can be computed in $O(n_i)$ time. To triangulate $S_i \setminus I$, we must triangulate $O(n_i)$ simple polygons, each one having at most 8 vertices. One such polygon can be triangulated in constant time. Hence, all polygons can be triangulated in $O(n_i)$ total time. This proves that the total building time is $O(\sum_i n_i) = O(n)$. ∎

Lemma 8 *The query time of the data structure is $O(\log n)$.*

Proof. The query algorithm follows a path from the root to a node without outgoing arcs. By Lemma 6, this path has length $O(\log n)$. In each node on this path, the algorithm checks all outgoing arcs and then takes the one

whose triangle contains the query point. Since each node has at most 8 outgoing arcs, and checking one arc is just checking if a point lies inside or outside a triangle, we spend $O(1)$ time in each node on the search path. ∎

This completes the analysis of the data structure. We summarize our result.

Theorem 3 *Let G be a connected planar graph with n vertices such that each bounded face of G is a triangle and the outer face of G is bounded by a triangle. In $O(n)$ time, we can store G in a data structure of size $O(n)$ such that point location queries can be answered in $O(\log n)$ time.*

3.1 Extension to arbitrary graphs

Let G be an arbitrary connected planar graph with n vertices. Of course, we reduce the point location problem for G to the same problem on a graph that satisfies the conditions of Theorem 3. We do the following.

First, we find three points $a, b, c \in \mathbb{R}^2$, such that all vertices of G are contained in the triangle T defined by these points. Let G' be the union of G and T. We triangulate G'; denote the triangulation by G'_t. Then we store G'_t in the data structure of Theorem 3. With each face f of G'_t, we store the name of the face in our original graph G that contains f.

Let $q \in \mathbb{R}^2$ be a query point. To find the face of G that contains q, we do the following. First we test if q is inside or outside the triangle T. If q is outside, then this point is contained in the unbounded face of G. Assume that q is inside T. Then we find the face of G'_t that contains q. This also gives us the face of G that contains q.

Theorem 4 *For the point location problem in a connected planar graph with n vertices, there exists a data structure having query time $O(\log n)$, size $O(n)$, and that can be built in $O(n)$ time.*

Proof. The bounds on the query time and the size follow from the above discussion. Consider the algorithm for building the data structure. The three new vertices a, b, and c can be found in $O(n)$ time. Given these vertices, we can create the union of G and triangle T. Each face of this union is a simple polygon. Since a simple polygon with m vertices can be triangulated in $O(m)$ time, the entire graph G' can be triangulated in $O(n)$ time. ∎

Remark 1 The linear-time algorithm for triangulating a simple polygon with m vertices is very complicated. For practical applications, we can use an $O(m \log m)$-time plane sweep algorithm. This increases the building time of the point location data structure to $O(n \log n)$.