CPSC 516 Computational Geometry **Fall 2022**

Lecture 17: November 3

Instructor: William Evans *Scribe: Aryan Tajmir Riahi*

We will discuss the problem of Point Location in this lecture. Given a Polygon subdivision of the plane, we want to build a data structure that answer queries asking which polygon contains a specific point. We will analyze different methods for this problem and their space, query time, pre-processing time, etc complexity.

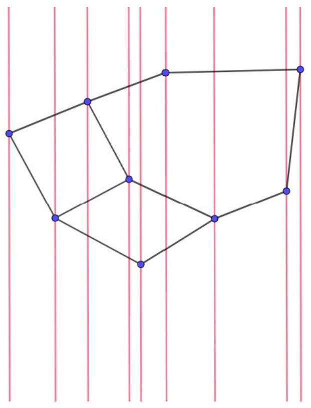
17.1 1D Case

To have a baseline of the complexity of this problem we will discuss the 1D case first. In the 1D case, subdivisions are intervals rather than polygons. In this simple case, the boundaries of subdivisions form $\mathcal{O}(n)$ points, and having these points sorted helps us answer queries efficiently. Storing these sorted points in an array would be problematic in the case that we want to handle update queries so we store the boundary points in a red-black tree. So in total, this method has pre-processing time complexity of $\mathcal{O}(n \log n)$ (it's just a sort), the space complexity of $\mathcal{O}(n)$, and query time complexity of $\mathcal{O}(\log n)$ (by using a binary search on sorted boundaries. Also, update queries can be handled in $\mathcal{O}(\log n)$.

17.2 Slab Method

Now we discuss the main problem in 2D. A trivial approach is to generalize the method for 1D into 2D. For this, we first search the point by its x-coordinate and then perform a search on y-coordinates. So this method has the following steps:

1. Draw vertical lines through all points
2. This leaves $n-1$ non trivial slabs
3. Perform a binary search to find the slab of the query point
4. Perform a binary search within the slab to find the region



This method performs two binary searches, hence it solves each query with $\mathcal{O}(\log n)$ time complexity. However, we have $n-1$ slabs and each one has $\mathcal{O}(n)$ complexity so we need $\mathcal{O}(n^2)$ space and pre-processing time complexity.

17.3 Kirkpatrick's Triangulation Refinement

Now we discuss a more efficient method in this section. The overall idea of Kapatrick's triangulation refinement is to have a sequence of triangulations $T_1, T_2, \dots, T_h$ which T_is are getting a coarser (T_{i+1} doesn't have some vertices in T_i), T_1 is the triangulated version of the input subdivision, while T_h is just a simple triangle. Using this sequence we can locate the point in T_h and then inductively by the location of the point in T_{i+1} find its location in T_i. In order to make this idea work we want to have:

- Small sequence. This idea requires performing for h iteration until it reaches T_1 so in order to have an efficient method we should build the small sequence. In other words in each step of building T_{i+1} from T_i we should delete many vertices.
- Simple iterations. In order to answer the query efficiently we should be able to go from T_{i+1} to T_i fast and easily. Notice that when building T_{i+1} we delete some vertices in T_i and re-triangulate the holes. So we should do this step in a way that each new triangle in T_{i+1} intersects with a few triangles in T_i. This means that we should delete vertices in T_i that leave small and independent holes. So we have two new criteria, we should only delete low-degree vertices and also delete an independent set of vertices in T_i.

To satisfy all three criteria for building T_{i+1} discussed above, we need the following Proposition.

Claim 1. Every triangulated planar graph with at least 4 vertices has independent set of size at least $n/18$ in which each vertex has degree equal or less than 8.
Proof. By Euler's formula we know that in a planar graph with n vertices there are $3n-6$ edges. Now we have

$$\sum_v \deg(v) = 2(3n-6) < 6n.$$

Now if there was at least $n/2$ vertices with degree greater than 8 we would have

$$\sum_v \deg(v) \geq 9 \times n/2 + 3 \times n/3 = 6n,$$

which is in contradiction with the previous inequality, proving there are at least $n/2$ vertices with degrees equal to or less than 8. Now among these $n/2$ vertices, we try to find an independent set by choosing an unmarked vertex and marking all its neighbors. Using this method each chosen vertex v will mark at most $\deg(v) \leq 8$ new vertices, so finally at least $n/2/9 = n/18$ vertices will be chosen. This proves the Claim.

Now for building T_{i+1} from T_i we can find the independent set described in Claim 1 and delete its vertices and re-triangulate holes. Using this method 1/18 of vertices will be deleted in each iteration so we have $h \in \mathcal{O}(\log n)$. Also as each deleted vertex has a degree at most 8 and they are not neighbors each triangle in T_{i+1} intersects with a finite number of triangles in T_i. So the time complexity of going from T_{i+1} to T_i is constant. Hence in total, the query time complexity of this method is $\mathcal{O}(\log n)$. For the space complexity we the summation of size of T_is is equal to $\sum_{i=0}^{h} n(\frac{17}{18})^i \in \mathcal{O}(n)$. The only challenging analysis is the pre-processing time complexity. Please take note that we can do the initial triangulation in $\mathcal{O}(n)$ and for building T_is every command we perform is amortized $\mathcal{O}(1)$ per deletion. Also, each vertex is deleted once so surprisingly (!) it's $\mathcal{O}(n)$ in total.

17.4 Summary

To summarise and compare all methods discussed in this lecture let's take a look at the following table.

	pre-processing time	space	query
1D	$\mathcal{O}(n \log n)$	$\mathcal{O}(n)$	$\mathcal{O}(\log n)$
Slab method	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(\log n)$
Kirkpatrick's method	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(\log n)$