Point Location using Hierarchical Decomposition

July 25, 2021 (newest version)

valentinpi
Freie Universität Berlin

Abstract

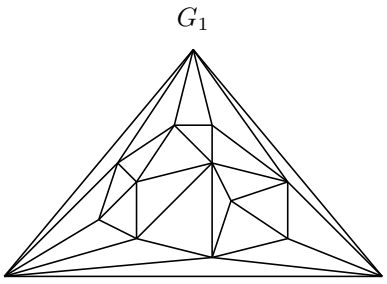
Planar Point Location has many useful applications, for instance in map apps or in video games. This short article presents the very elegant Hierarchical Decomposition Method for Point Location from my course on Algorithmic Geometry by giving an informal description and a runtime analysis. The resulting datastructure can be preprocessed in $O(n)$ time, using the same space complexity and a query time of $O(\log n)$.

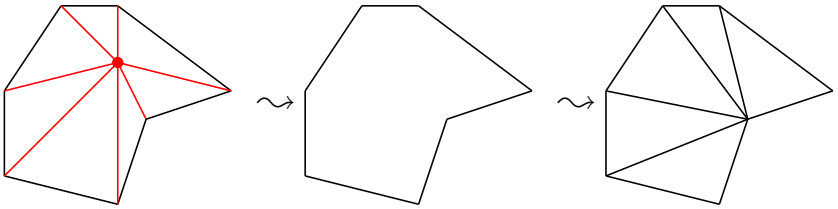
ATTENTION: I do not claim ownership of the contents here. All of the ideas are from my lecture notes. However, I find this construction to be fascinating.

Introduction Let $\mathbb{R}$ denote the set of real numbers. A *planar subdivision* is an embedding/drawing of a planar graph $G = (V, E)$ into the plane $\mathbb{R}^2$, consisting of vertices, edges and faces.
Basic knowledge of planar graphs and geometry is required for reading this paper.

Task: Given a planar subdivision $\mathcal{P}$ and a point $p \in \mathbb{R}^2$, find the face in which p falls.

Hierarchical Decomposition Method I will now present the idea. First, wlog put the planar subdivision inside of a gigantic triangle. Then, triangulate that subdivisions faces in $O(\log n)$ time with an algorithm of choice, excluding the outer face. Denote this graph as G_1.
It is clear that we can add three more such vertices for the triangle in $O(n)$ time by computing extremal points of the vertex set. We now pick one low-degree vertex, i.e. with $\leq c$ adjacent vertices, with c being some constant which we will later find out. This vertex will be removed with all its edges, and the resulting hole in the triangulation will be retriangulated.





This leads to the next graph G_2. A *hierarchical decomposition* of $\mathcal{P}$ is now composed of k such graphs $G_1, ..., G_k$. In each graph G_i, we augment the holes created from the vertex removal by storing which triangles of the previous graph G_{i-1} the hole embedded. It is important that the vertices chosen only have constantly high degree, as otherwise the face number could increase in a higher bound.

Queries With one vertex removed each time, we could obtain up to $k = n$ such graphs. Our goal is to obtain $k \in O(\log n)$ many, such that the query time becomes logarithmic. Speaking of the query time, to query a point p one has to start from $G_k =: G_i$, find the first hole that contains p. If that hole is part of the original graph, the point location is finished. If not, move to G_{i-1}. This continues until the root condition is satisfied.
To obtain logarithmically many such decisions, the idea is to remove several vertices of low degree each time. For that, we will need some more observations on the structure of planar graphs.

Definition 1. For a given graph $G = (V, E)$, a set of vertices $U \subseteq V$ is called an *independent set*, if none of the vertices are adjacent in G.

Lemma 1. Given a planar graph $P = (V, E)$ with $n \coloneqq |V|$, there are constants $c \geq 60$ and $0 < \alpha < 1$, s.t. there is an independent set of size αn with each vertex of degree $< c$.

Proof by Contradiction. We use a couple of facts: Planar graphs are 4-colorable, which can be done in $O(n^2)$ time. But especially, planar graphs are 5-colorable in $O(n)$ time. Also, for any planar graph with v vertices and e edges it holds that $e \leq 3v - 6$, which easily follows from Euler's Theorem.
We find such a 5-coloring, and pick a color class $U \in V$ with $\geq \frac{n}{5}$ vertices, as there has to exists at least one such class.
Set $\alpha \coloneqq \frac{1}{10}$. From U, we pick $\frac{n}{10}$ many vertices of smallest degree. We claim that these vertices all have degree $< c \coloneqq 60$. Suppose the opposite, then the *unpicked* vertices would have a total degree of $\frac{n}{10} \cdot 60 = 6n$, which corresponds to more than $3n$ edges inside the graph, as the coloring itself is an independent set. This contradicts the result from above $\lightning$. ■

Complexity Consequences Observe that with the existence of the constants c and α, we can now implement the algorithm. We now explore the complexity consequences of these results.
First, the size of the hierarchical decomposition. It is still open if $k \in O(\log n)$. For that, observe the following:

- Vertex count of G_1: n
- Vertex count of G_2: $n - \alpha n = (1 - \alpha)n$
- Vertex count of G_3: $(1 - \alpha)^2 n$
- ...
- Vertex count of G_k: $(1 - \alpha)^{k-1} n = 3$

In case of G_k, it is clear that the last graph will have a constant size, as the smallest remaining graph will be the outer triangle of vertex count 3. We never remove any vertices of that outer triangle, of course. With that in mind:

$$\begin{aligned} (1-\alpha)^{k-1} n &= 3 \\ \log(n) + (k-1)\log(1-\alpha) &= \log 3 \\ \Rightarrow k &= \frac{\log(n)}{\log \frac{1}{1-\alpha}} - \frac{\log 3}{\log 1 - \alpha} + 1 \in O(\log n) \end{aligned}$$

As for the space usage, observe:

$$\underbrace{n}_{G_1} + \underbrace{(1-\alpha)n}_{G_2} + \underbrace{(1-\alpha)^2 n}_{G_3} + ... + \underbrace{(1-\alpha)^{k-1} n}_{G_k} = \left(\sum_{i=0}^{k-1} (1-\alpha)^i \right) n = \frac{1-(1-\alpha)^k}{1-(1-\alpha)} n \longrightarrow_{k \to \infty} \frac{n}{\alpha} \in O(n)$$

Since $k \in O(\log n)$. With the previous assessment, we can also observe the preprocessing time:

$$\underbrace{\left(\sum_{i=0}^{k-1} (1-\alpha)^i \right) n}_{\text{Finding the independent set, removing vertices and retriangulating}} = O(n)$$

The retriangulations take $O(1)$ time for each hole, as they are of constant size, so even an inefficient triangulation algorithm can be used there.