Comp 163: Computational Geometry
Tufts University, Spring 2003

Professor Diane Souvaine
Scribe: Peter Waltman

Planar Point Location

1 Introduction

So far, in a previous lecture, we've seen a naive algorithm for determining a point's location within a single polygon (either convex or not). In the naive algorithm, we extend a horizontal line, h, through the query point, q, and count how many edges of the polygon, P, that h intersects.

Ignoring degeneracies, we can determine q's location relative to P such that:

- q is interior to P if on both q's left and right side, h intersects an **odd** number of P's edges.
- q is exterior to P if on both q's left and right side, h intersects an **even** number of P's edges.

However, as we need to examine all of P's edges to do this, this would yield an $O(n)$ query time for each query.

Can we do better?

2 Algorithm 1: Kirkpatrick's Hierarchical Decomposition

Kirkpatrick's planar point location algorithm improves upon the naive algorithm above, by allowing a $O(nlogn)$ query time by constructing a series of "S-trees" or Subdivision Trees of increasing granularity which can be searched in a manner similar to searching on standard binary search trees.

However, in order to assure this $O(n)$ query time, the space complexity **must** be linear, ($O(n)$). To do this, Kirkpatrick takes advantage of Euler's formula which shows that a planar graph with n vertices can have at most $2n - 4$

regions (and $3n - 6$ edges). Therefore, by not adding any new new vertices to the graph in his algorithm, Kirkpatrick assures that the space complexity is linear. More on this below.

First, however, the algorithm itself....

2.1 Algorithm Overview

Given a finite search domain consisting of one or more polygons, we:

1. Begin by creating a triangle large enough to enclose the graph
2. Triangulate this triangle and the contained polygon(s).
3. Create a new copy of this triangulation, in which we:
   (a) Find an independant set of vertices (no two vertices share and edge) *V'*
   (b) Delete these vertices along with the edges that are incident them
   (c) Create a new polygon for each region created by these deletions with the vertex in *V'* which begat it.
   (d) Re-triangulate these new regions
4. repeat steps *3a-d* until there is only one region left within the triangle.

Once completed with this pre-processing, each of the polygons created during each iteration corresponds to one level of the subdivision tree. Starting with the level of least granularity, we perform a constant time search to see which triangle in it's "child" S-tree contains the point and then successively continue down tree moving in direction of greater granularity.

Got that? Great! Let's move on... *Well*, maybe an example would be a little helpful....

2.2 Example

Ok, so, using the following example from [1]: We see that:

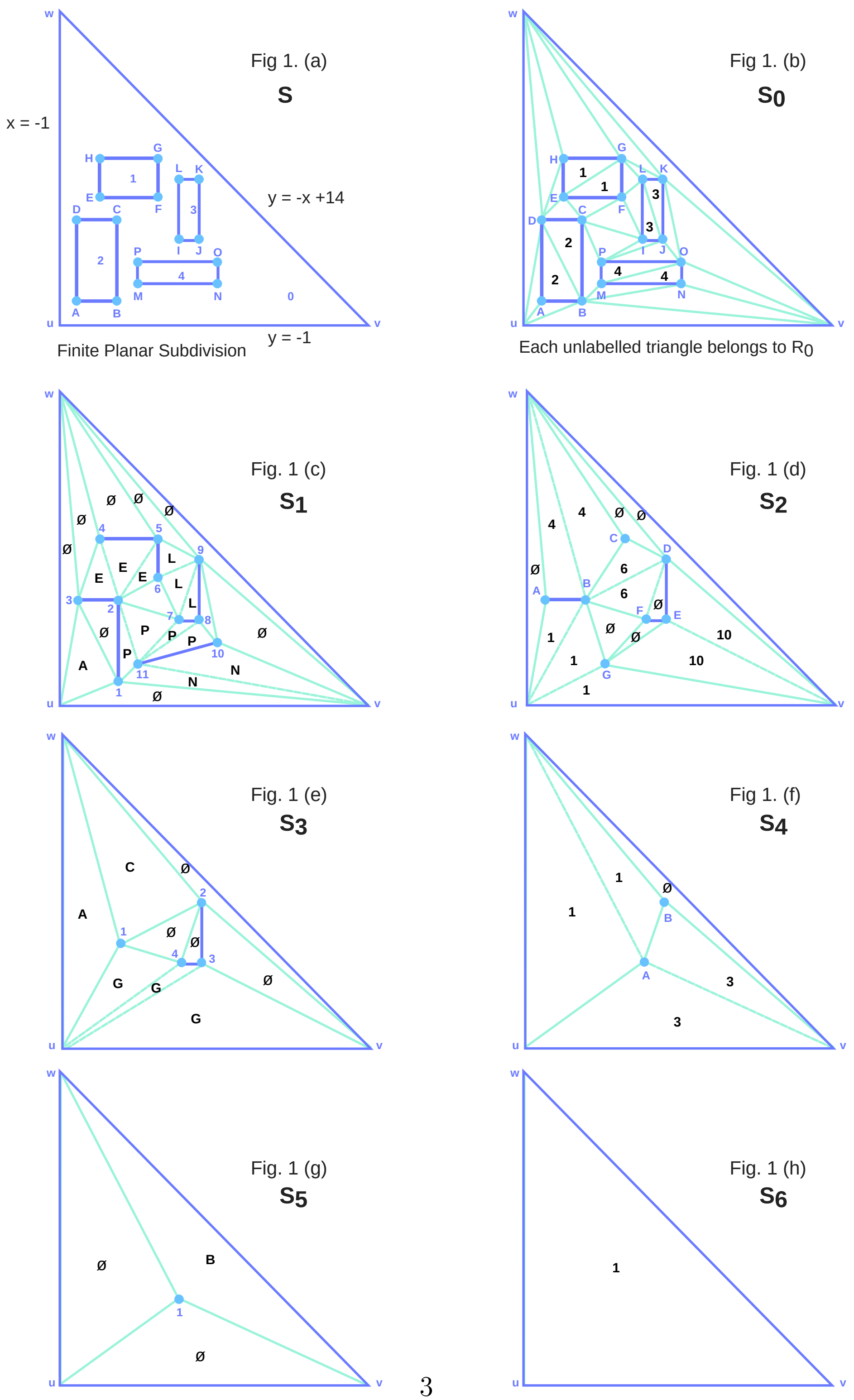


Figure 1: Figure1

- In Fig. 1a, we first begin with a planar graph containing 4 simple rectanglular polygons and the enclosing triangle.(**note**, this algorithm is not limitted to simple polygons like these, it just makes for an easier example.) The vertices are labelled alphabetically, the regions enclosed by the polygons are labelled appropriately, while the edges are left unlabelled.

- In Fig. 1b, you can see that we've completed the intitial triangulation of the initial graph. (step 2)

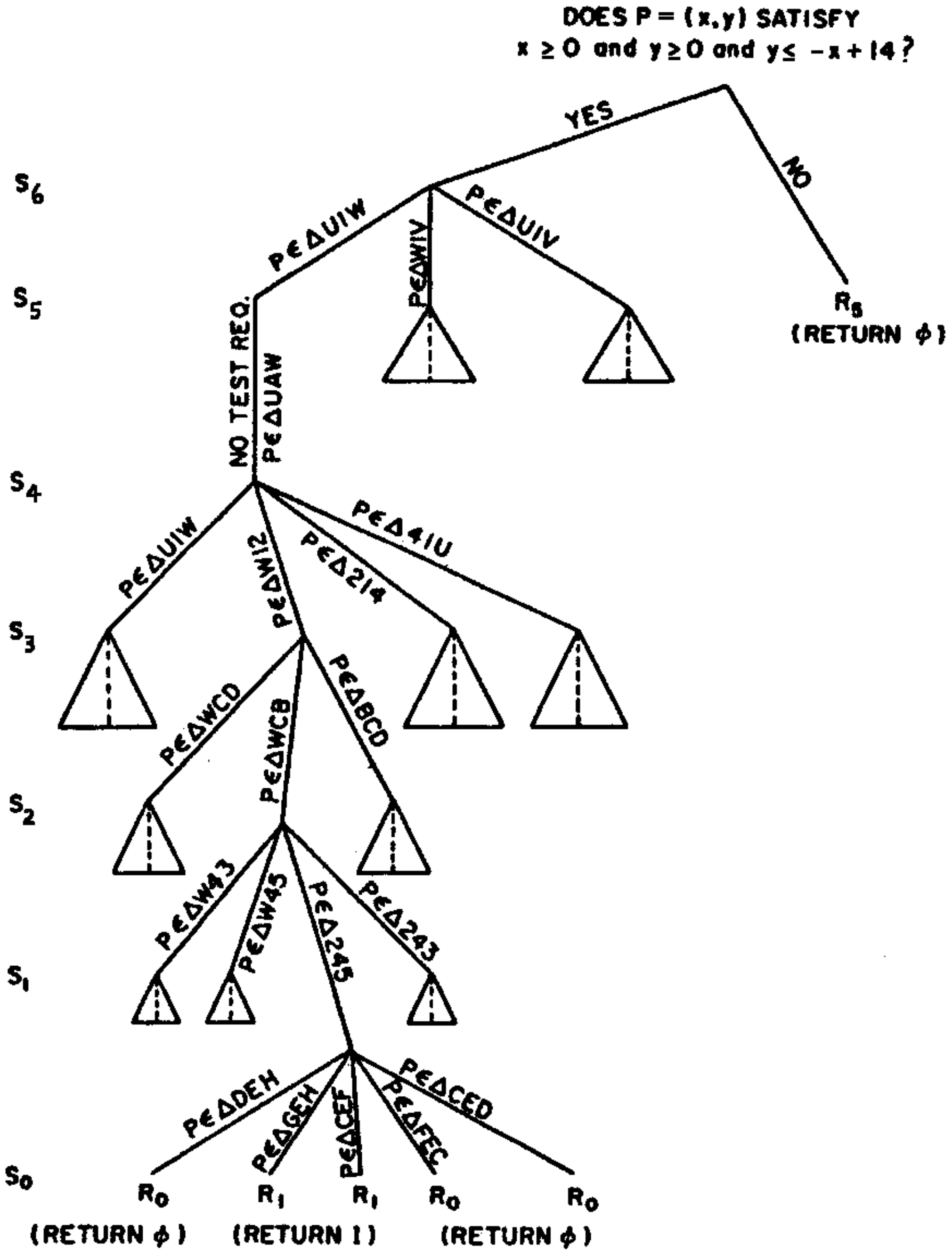
- In Fig. 1c,

  - vertices *A, E, L, N, P* been deleted along with any edges coming out of these.
  - Where these vertices and edges once were, 3 new polygons have been created and labelled with the appropriate vertex name and re-triangulated. (steps 3b-d). In terms of data structures, one can consider that there exists a pointer from each of these triangles to the appropriate vertex in S_0
  - Observe that regions in S_1 which are identical to triangles in S_0 are labelled with ϕ. In this case, each of these triangles has a pointer to the identical triangle in S_0

- In Fig's 1d-1h ($S_2 - S_6$), we see a continuation of this iterative process until in Fig. 1h, we see that S_6 only contains a single region.

Note The steps in Figure 1 alternate the naming of the vertices for ease-of-reading purposes and is not neccessary when implementing the actual algorithm.

In Fig. 2, we see a "stacking" of these subdivisions into a search tree. For each level of the tree, a constant time check can be performed to determine where a potential query point falls within the subdivision.



Search tree

FIG. 2

Figure 2: Figure1

For example, given a query point, $P = (3,\ 6)$, we can test on S_6 to see if the point falls within $\triangle UVW$. As it does fall within this triangle, we move to S_5 and test to the triangles tangent to point I, to see which contains P. In this case, P is interior to $\triangle UIW$. As $\triangle UIW$ is identical to it's "child" triangle (or parent, depending on how you look at it) in S_4, no test is neccessary in S_4 and we can jump to S_3 and continue from there.

2.3 Algorithm Complexity

As said above, this algorithm takes advantage of Euler's relation, which again states that for a planar graph with n vertices, there can be at most:

- $2n - 4$ regions
- $3n - 6$ edges

As we are fully triangulating the planar graph in S_0, we therefore know that there are $3n - 6$ edges in S_0. Furthermore, as every edges is incident to two vertices, the average vertex degree must be $(6n - 12)/n$, which is roughly equal to 6. Therefore, at least half of vertices must have degree less than 12, and if we let V be the vertices in S_0 with degree less than 12, then $|V| \geq n/2$.

By this relation, we also know that when choosing an independant subset, V', where $v \in V'$, that v has at most 11 neighbors at most 12 vertices are removed from V. Therefore, we can repeat this process (of finding $v \in V'$) at least $n/12$ times.

If we do this until V is empty, V' will contain $n/24$ independant vertices (or more), leaving at most $23n/24$ vertices in S_0

Therefore, by this relation, if, for every S_i we remove all independant vertices with degree ≤ 11 from S_i, this will yield the following sum to represent the total number of vertices within the hierarchy is:

$$|S_0|\ \textstyle\sum_{i=0}^{m}(\frac{23}{24})^i < n\ \sum_{i=0}^{\inf}(\frac{23}{24})^i = 24n$$

Hence, we know that the space complexity of the algorithm, $S(n)$ is linear$(O(n))$.

Moreover, we also know from the relation that $|S_i| = (\frac{23}{24})^i\ n$ that when $|S_i| = 1$ (the topmost S-tree), we get the following relation:

$|S_i| = (\frac{23}{24})^i\ n$
$1 = (\frac{23}{24})^i\ n$
$i \leq (logn - log1)/(log24 - log23)$
$i \leq logn$

Which shows us that there are at most $O(logn)$ levels per S-tree.

So, how many comparisons are there per level? Well, if we only delete those vertices in V, which are those vertices with degree less than 12, then we know that there are at most $O(12)$ or $O(1)$ constant time comparisons per each level of the S-tree.

As for pre-processing time, $P(n)$, the algorithm is linear (in terms of space) if the graph has been triangulated to start with and $O(nlogn)$ if not.

2.4 conclusion

Therefore, in summary, Kirkpatrick's Hierarchical Decompasition algorithm for planar point location has the following complexities:

- $S(n) = O(N)$
- $Q(n) = O(logn)$
- $P(n) = O(n)||O(nlogn)$

References

[1] David P. Dobkin, Diane L. Souvaine, *Computational Geometry - A User's Guide.*