Original Lecture #7: 30 January 1992
Topics: Seidel's Trapezoidal Partitioning Algorithm
Scribe: Michael Goldwasser

Seidel's Trapezoidal Partitioning Algorithm

Overview

A Randomized Incremental Algorithm.

Idea: incremental algorithms are usually simple. We build a structure by adding objects one at a time. The problem is that they often lead to poor performance. As a somewhat simplified example, suppose we build a search tree for a set of keys by inserting the keys one at a time, without rebalancing. If the keys are inserted in order, the tree will be unbalanced and the construction will take $\Theta(n^2)$ time. Randomization can help with this problem: if we insert keys in random order, we expect to get a balanced search tree in $O(n \log n)$ time. By randomizing the order in which we add objects, we speed up the algorithm.

The randomization in randomized incremental algorithms has two key properties. First, the randomization is internal to the algorithm; we make no assumptions about the input. Instead, the algorithm does its own "coin flips." Second, the randomization is of the Las Vegas type: the algorithm is guaranteed to give a correct answer (whatever that means in the context of the problem), but the running time is not guaranteed in any particular instance. Instead, we talk about the *expected* running time, based on the algorithm's internal randomization.

Content of Lecture

1. Basic algorithm and data structures
2. Analysis
3. Randomized Analysis – $O(n \log n)$
4. Tracing and analysis of it
5. Extensions

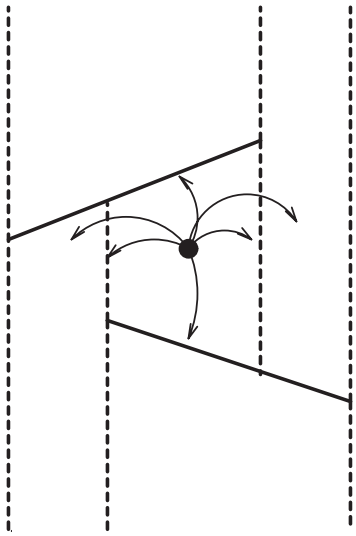

Figure 1: The mappings in $T(S)$.

Data structures

Let S be a set of line segments (with the usual non-degeneracy conditions, such as no two endpoints directly above and below one another, and no vertical line segments).

Let $T(S)$ be a trapezoidal partitioning (also called a "trapezoidation"). It should let us map any trapezoid τ to

- its bounding segments; and
- its horizontally neighboring trapezoids (there are at most four).

These mappings are illustrated in fig. 1.

This representation lets us trace a polygonal curve C through $T(S)$ in time $O(|C| + k)$, where k is the number of trapezoids C intersects, provided $C \cap S = \emptyset$.

Let $Q(S)$ be a point location structure for $T(S)$. That is, given a query point $q = (q_x, q_y)$, it allows us to determine what trapezoid that point is in. $Q(S)$ is a directed acyclic graph with one source, and with one sink per trapezoid of $T(S)$. The internal nodes are of two types:

1. X-nodes. (See fig. 2.) When we get to these nodes, we ask, is point q to the left or right of a particular X-value? We then branch accordingly.
2. Y-nodes. (See fig. 3.) Here we ask, is point q above or below a particular line?

So to answer a query (*i.e.* identify the trapezoid containing point q), we follow a directed path in $Q(S)$, branching to one of the two descendants at each internal node,

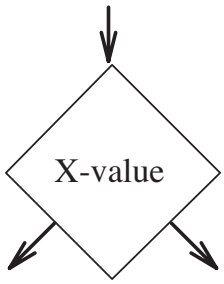

Figure 2: An X-node.

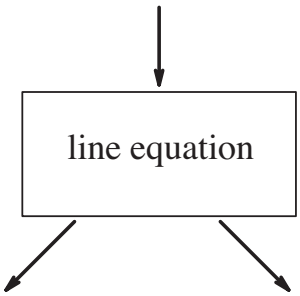

Figure 3: A Y-node.

and stopping at a sink (*i.e.* a trapezoid). (See fig. 4.)

Now, let $e \notin S$ be a segment $\overline{ab}$ ($a_x < b_x$) such that e crosses no segment of S. Define $S' = S \cup \{s\}$.

Question: How do we produce $T(S')$ from $T(S)$, and $Q(S')$ from $Q(S)$? See fig. 5. (Note: in this figure, we have $S = \{e_1\}$ and the new segment is e_2.)

The basic steps are: insert left endpoint, insert right endpoint, and insert the segment between them.

The first thing to ask is, does the new segment share an endpoint with any of the existing segments? If not, we add new vertical threads corresponding to the new endpoints. This splits trapezoid 3 into new trapezoids 5 and 6, and splits 4 into 7 and 8 (see fig. 6). But, of course, e_2 itself further splits 6 into 9 and 10. It also splits 7, but 7's bottom half is then merged with 10 (see fig. 7).

We perform similar operations on $Q(S)$, as shown in the figures. (Indeed, since we need $Q(S)$ to help us locate trapezoids in $T(S)$, we should do these operations in parallel.)

Analysis of running time

The running time has two parts:

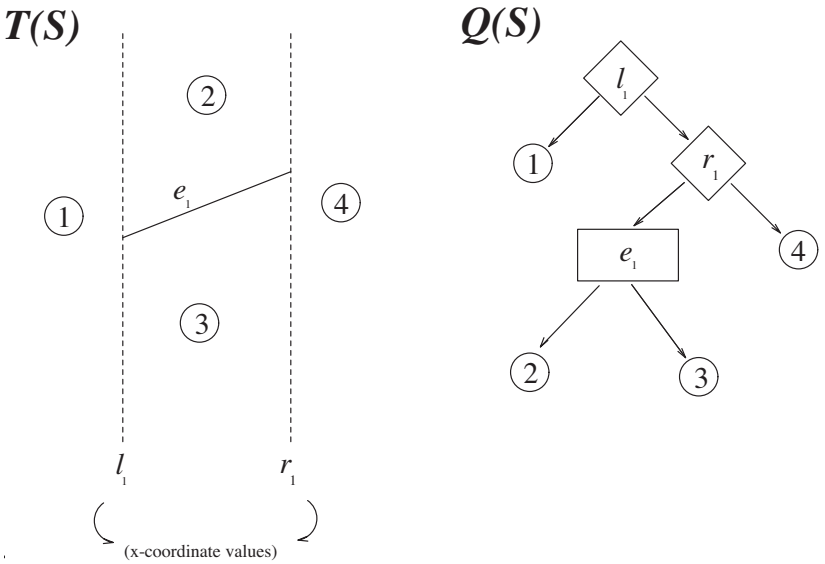

Figure 4: An example of a trapezoidal partitioning $T(S)$, and the corresponding point location structure $Q(S)$.

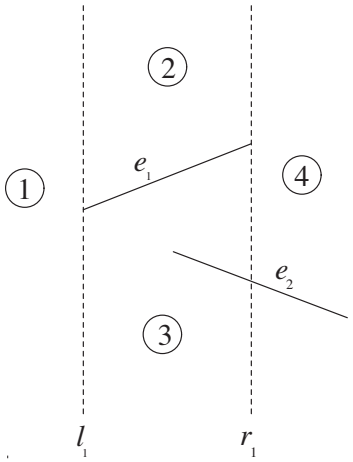

Figure 5: Adding a new segment.

- query time, *i.e.* point location in $Q(S)$; and
- tracing time through the trapezoids in $T(S)$, including the time to perform splitting and merging operations.

Now, the tracing time is $O(1)$ plus a term proportional to the number of threads in the new partitioning that "abut" upon e. Note that this does not include the threads running through the endpoints; for instance, the segment in fig. 8 has only one thread abutting on it.

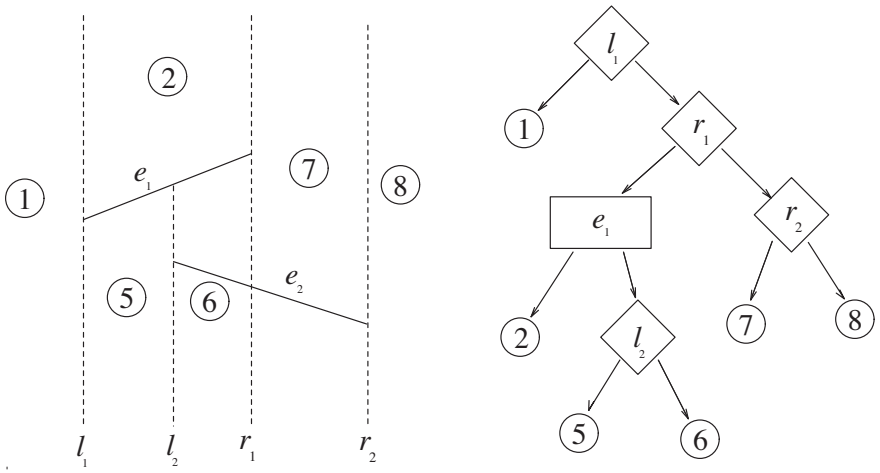

Figure 6: Intermediate stage in adding a segment

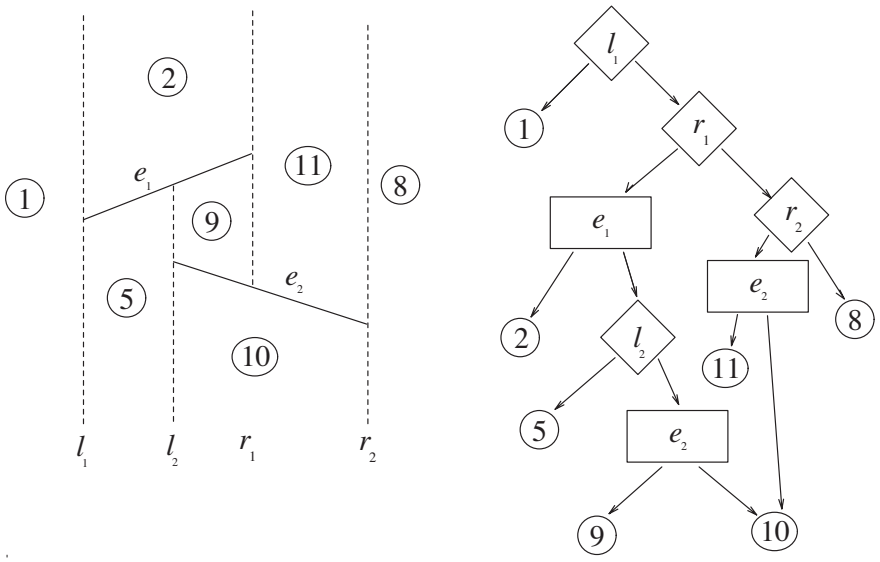

Figure 7: Adding a segment. $T(S)$ and $Q(S)$ have now been fully updated.

In the worst case, this could make for a lot of tracing time. For instance, in the polygon shown in fig. 11, if we insert the segments in counterclockwise polygon order starting at the upper right, the trace time is $\Theta(|S|^2)$. The point location cost may also be excessive. If the segments shown on the left side of fig. 9 are inserted bottom to top, the query data structure is very unbalanced, as shown on the right side of fig. 9, and the total query time is $\Theta(|S|^2)$.

But what if we look at the algorithm's average or expected performance, rather than its worst-case performance?

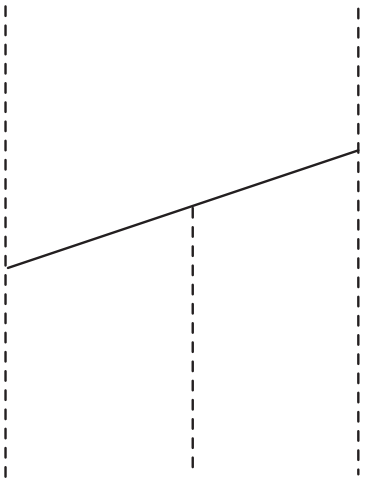

Figure 8: A segment with just one thread abutting on it.

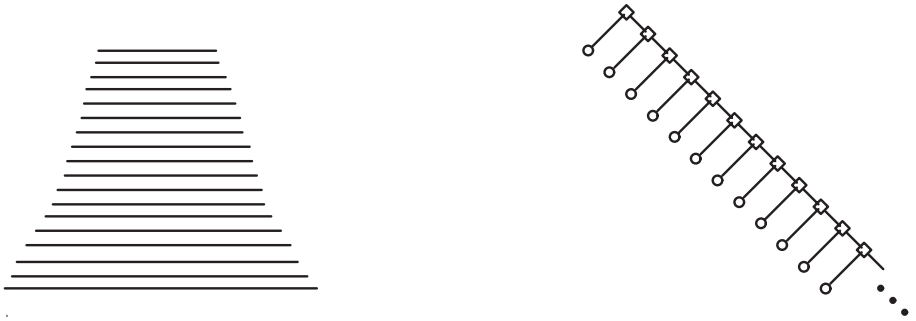

Figure 9: (Left) A set of segments in T which, if inserted bottom to top, leads to a worst-case query time. (Right) The resulting query structure.

Randomized analysis of running time

Let $e_1, e_2, \ldots, e_n$ be a random ordering of the segments of S, and let $S_i = \{e_1, e_2, \ldots, e_i\}$ (a prefix of S), for $0 \leqslant i \leqslant n$.

Lemma 1. *The expected cost of tracing e_i through $T(S_{i-1})$ is $\leqslant 4$.*

Proof: Define the **degree** of a segment in a partitioning, $\deg(e, T(S))$, as the number of vertical threads that abut upon e ($e \in S$).

Then

$$\sum_{e \in S_i} \deg(e, T(S_i)) \leqslant 4i \tag{1}$$

This is because each thread (there are $\leqslant 2i$ of them) hits $\leqslant 2$ segments. So for a "random" e_i, the expected degree $E(\deg(e_i, T(S_i)))$ is at most 4. □

Now, recall that $H_n = \sum_{i=1}^{n} \frac{1}{i} = \Theta(\log n)$.

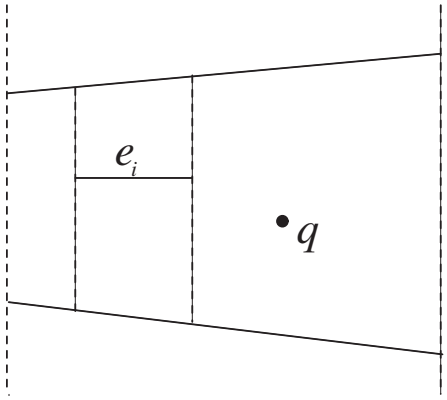

Figure 10: Here, the addition of e_i creates two new decision levels (instead of one) in the search path for q.

Lemma 2. *For any query point q, the expected number of key comparisons (over all orderings of S) to locate q in $Q(S_n)$ is $\leqslant 5H_n = O(\log n)$.*

In other words, the number of decision nodes in $Q(S)$ that we'll cross to get to q is $O(\log n)$.

Proof: We use "backward analysis." We will first find how many extra decision nodes we have to cross, on average, as a result of adding a single edge. Then we add up all those incremental costs to get the total cost (effectively, building up $Q(S)$ from scratch).

Suppose, then, that q lies in trapezoid τ_{i-1}, where $\tau_{i-1} \in T(S_{i-1})$, and q also lies in $\tau_i \in T(S_i)$. If we know τ_{i-1} (and its corresponding sink in $Q(S_{i-1})$), what is the expected cost E_i of finding τ_i?

Well, under what circumstances will we have to go through an extra decision level because of the new edge e_i? That is, when is $\tau_i \neq \tau_{i-1}$? This occurs

- if τ_i's upper bounding edge is e_i (an event with probability $\leqslant \frac{1}{i}$);
- if τ_i's lower bounding edge is e_i (an event with probability $\leqslant \frac{1}{i}$);
- if τ_i's left bounding thread was generated by e_i (an event with probability $\leqslant \frac{1}{i}$); or
- if τ_i's right bounding thread was generated by e_i (an event with probability $\leqslant \frac{1}{i}$).

One more decision level is possible. For the situation shown in fig. 10, the addition of e_i caused *two* new comparisons to be made, not one. This situation also has probability $\leqslant \frac{1}{i}$.

Therefore, the expected incremental cost E_i is $\leqslant \frac{5}{i}$, and the expected total cost of locating q is $\sum_{i=1}^{n} E_i = 5H_n$. □

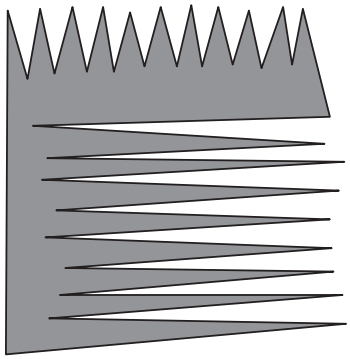

Figure 11: A polygon whose boundary we would not want to trace.

With these lemmas, we can now state the following theorem, which applies to arbitrary sets of disjoint line segments.

Theorem 3. *Let S be a set of n segments, disjoint except for their endpoints. Then*

1. *We can build $T(S)$ and $Q(S)$ in expected $O(n \log n)$ time.*
2. *Expected $|Q(S)| = O(n)$. $\langle$the expected number of trapezoids split is $O(n)\rangle$*
3. *Expected query time for any point q in $Q(S)$ is $O(\log n)$.*

The second part of this theorem follows from Lemma 1. The third part is a restatement of Lemma 2. The first part is a consequence of Lemmas 1 and 2.

Where does all that time go? It goes to point location. How, then, can we avoid point location, in the special case in which the segments form a polygon? Well, we could start from the endpoints of earlier segments, *i.e.* insert the segments in polygon order. But then we would lose the benefits of randomization. For example, with the polygon in fig. 11, the cost of tracing around it goes up as $\Omega(n^2)$. So let us use a compromise solution. We will insert the segments in random order, but periodically trace the polygonal boundary to speed up point location for later insertions. (Details below.)

Lemma 4. *For any q, if we know that $q \subset \tau_j$ (where $\tau_j \in T(S_j)$), then the expected cost of locating q in $T(S_k)$, $k \geqslant j$, is $\leqslant 5(H_k - H_j) = O(\log \frac{k}{j})$.*

Proof: The proof is almost exactly the same as that of Lemma 2, above. □

Lemma 5. *Let R be a random subset of S, with size $|R| = r$. Let Z be the number of intersections between $T(R)$ and $S \backslash R$. [Recall that $S \backslash R$ denotes the segments of S that are not in R.] Then the expected value of Z is $\leqslant 4(n - r)$.*

Proof: For any $e \in (S \backslash R)$, the number of intersections between e and $T(R)$ is $\deg(e, T(R \cup \{e\}))$. So

$$\begin{aligned} E(Z) &= \frac{1}{\binom{n}{r}} \sum_{\substack{R \subset S \\ |R|=r}} \sum_{e \in (S \backslash R)} \deg(e, T(R \cup \{e\})) \\ &= \frac{1}{\binom{n}{r}} \sum_{\substack{R' \subset S \\ |R'|=r+1}} \sum_{e \in R'} \deg(e, T(R')) \end{aligned}$$

from (1),

$$\begin{aligned} &\leqslant \frac{1}{\binom{n}{r}} \sum_{\substack{R' \subset S \\ |R'|=r+1}} 4|R'| \\ &= 4(r+1)\frac{\binom{n}{r+1}}{\binom{n}{r}} \\ &= 4(n-r) \end{aligned}$$

□

Now, define $\log^{(i)} n = \overbrace{\log \log \cdots \log}^{i} n$, and $\log^* n = \max(h | \log^{(h)} \geqslant 1)$. Also let $N(h) = \lceil \frac{n}{\log^{(h)} n} \rceil$.

With these definitions, we now present the final version of Seidel's algorithm.

Seidel's Trapezoidal Partitioning Algorithm

1. Generate a random order $s_1, s_2, \ldots, s_n$ of the segments of P.
2. Generate $T(S_1)$ and $Q(S_1)$ [initialization].
3. for $h = 1$ to $\log^* n$ do
   a. for $i = N(h-1)+1$ to $N(h)$ do
      Insert segment e_i, producing $T(S_i)$ and $Q(S_i)$ from $T(S_{i-1})$ and $Q(S_{i-1})$.
   b. Trace P through $T(S_{N(h)})$ to locate the endpoints of all e_j, for $j > N(h)$.
4. for $i = N(\log^* n)+1$ to n do
   Insert e_i, producing $T(S_i)$ and $Q(S_i)$ from $T(S_{i-1})$ and $Q(S_{i-1})$.

Analysis

Let us examine the expected running times of each step:
1. $O(n)$
2. $O(1)$
3a.

$$N(h) \cdot \left(O(1)\langle \text{from Lemma 1} \rangle + O(\log(\frac{N(h)}{N(h-1)}))\langle \text{from Lemma 2} \rangle \right)$$

Now,

$$\begin{aligned}\log(\frac{N(h)}{N(h-1)}) &\leqslant \log(\frac{n}{N(h-1)}) \\ &\leqslant \log(\log^{(h-1)} n) \\ &= \log^{(h)} n.\end{aligned}$$

Therefore, step (3a) takes $\leqslant N(h) \cdot O(\log^{(h)} n) = O(n)$ time.
3b. $O(n)$, by Lemma 5.
4. $O(n) \cdot \underbrace{\langle \log(\frac{n}{N(\log^* n)}) = \log\log^{(\log^* n)} n \leqslant \log 2 = O(1) \rangle}_{\text{(as in 3a)}}$

The total running time is $O(n)$ per phase. There are $O(\log^* n)$ phases, so the final running time of the algorithm is $O(n \log^* n)$. This completes the proof of the following improved version of Theorem 3.

Theorem 6. *Let S be a set of n segments that form a simple polygon. Then*

1. *We can build $T(S)$ and $Q(S)$ in expected $O(n \log^* n)$ time.*
2. *Expected $|Q(S)| = O(n)$.*
3. *Expected query time for any point q in $Q(S)$ is $O(\log n)$.*

Implementation

The algorithm was run on a computer as a demonstration at the end of class, which served to emphasize the basic steps involved in this algorithm: (1) add a segment at random; (2) locate its endpoints; (3) insert trapezoids.

Every now and then, though, the algorithm would search around the boundary of the polygon. As it did so, it located the vertices that had not yet been placed, in terms of the existing trapezoids. Then later, when it needed to locate those endpoints, it was able to use that information to start its searches of $Q(S)$ from further down the DAG (thereby reducing its search time). This is the compromise discussed earlier on page 8.

Extensions

Seidel's algorithm also applies to more general objects than simple polygons. Theorem 6 applies to any collection of line segments, disjoint except for their endpoints, that form a straight-line embedding of a connected planar graph—we replace the polygon tracing step by a graph traversal, but all other steps remain unchanged. Combining this observation with Theorem 3, we get the following theorem.

Theorem 7. *Let S be a set of n segments that form a straight-line embedding of a planar graph with k connected components. Then*

1. *We can build $T(S)$ and $Q(S)$ in expected $O(k \log n + n \log^* n)$ time.*
2. *Expected $|Q(S)| = O(n)$.*
3. *Expected query time for any point q in $Q(S)$ is $O(\log n)$.*