CMSC 754: Lecture 8
Trapezoidal Maps

Reading: Chapter 6 of the 4M's.

Trapezoidal Map: Many techniques in computational geometry are based on generating decomposing a complex arrangement of objects into a collection of simple objects. We have seen triangulations as one example, where the interior of an n-vertex simple polygon is subdivided into a collection of $n-2$ triangles. Today, we will consider a considerably more general technique for subdividing space in the midst of a collection of disjoint line segments.

Let $S = \{s_1, \ldots, s_n\}$ be a set of line segments in the plane such that the segments do not intersect one another, except perhaps that two segments can intersect at their endpoints. (We allow segments to share common endpoints so that our results can be generalized to planar graphs and planar subdivisions.) Let us make the general-position assumptions that no two endpoints have the same x-coordinate, and (hence) there are no vertical segments.

We wish to produce a subdivision of space that "respects" these line segments. To do so, we start by enclosing all the segments within a large bounding rectangle (see Fig. 1(a)). This is mostly a convenience, so we don't have to worry about unbounded regions. Next, imagine shooting a *bullet path* vertically upwards and downwards from the endpoints of each segment of S until it first hits another segment of S or the top or bottom of the bounding rectangle (see Fig. 1(b)). The combination of the original segments and these vertical bullet paths defines a subdivision of the bounding rectangle called the *trapezoidal map* of S (see Fig. 1(c)), denoted $\mathcal{T}(S)$.

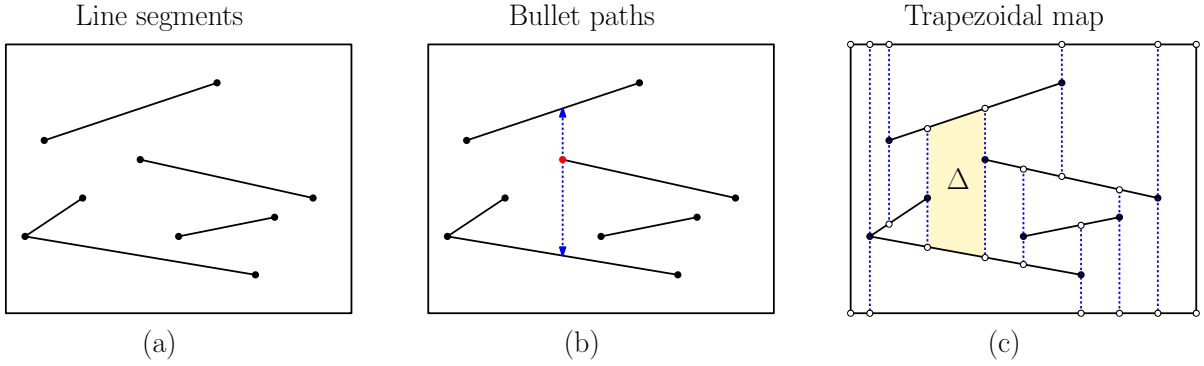

Fig. 1: A set of segments and the associated trapezoidal map.

The faces of the resulting subdivision are generally trapezoids with vertical sides, but they may degenerate to triangles in some cases. The vertical sides are called *walls*. Also observe that it is possible that the nonvertical side of a trapezoid may have multiple vertices along the interior of its top or bottom side. This was not the case for the triangulations that we discussed earlier, where adjacent triangles met only along complete edges. (In the terminology of topology, a trapezoidal map is *not* a *cell complex*, while a triangulation is.) Trapezoidal maps are useful data structures, because the provide a way to convert a possibly disconnected collection of segments into a structure that covers the plane.

We begin by showing that the process of converting an arbitrary polygonal subdivision into a trapezoidal decomposition increases its size by at most a constant factor. We derive the exact expansion factor in the next claim.

Lemma: Given set S of n line segments in the plane, the resulting trapezoidal map $\mathcal{T}(S)$ has at most $6n + 4$ vertices and $3n + 1$ trapezoids.

Proof: Each segment generates six vertices in the map, two for the segment's endpoints, two from the upward vertical rays, and two from the downward vertical rays (see Fig. 2(a)). In addition, there are four vertices for the bounding rectangle, resulting in a total of $6n + 4$ vertices.

Let us charge each trapezoid to the vertex along its left edge. Each segment is charged by three trapezoids, since the left endpoint of each segment is charged twice, one from the trapezoid above-right and one from the trapezoid below-right, and the right endpoint is charged once by the trapezoid to its right (see Fig. 2(b)). This yields a total of $3n$ trapezoids. There is one more trapezoid, namely the leftmost one, for a total of $3n + 1$.

(a) (b)

Fig. 2: (a) The six vertices associated with each segment and (b) the three left-bounding trapezoids associated with each segment.

An important fact to observe about each trapezoid is that its existence is *determined* by exactly four entities from the original subdivision: a segment on top, a segment on the bottom, a bounding vertex on the left, and a bounding vertex on the right. The bounding vertices may be endpoints of the upper or lower segments, or they may below to completely different segments. This simple observation will play an important role later in the analysis.

Construction: We could construct the trapezoidal map by a straightforward application of plane sweep. (By now, this should be an easy exercise for you. You might think about how you would do it.) Instead, we will build the trapezoidal map by a different approach, namely a *randomized incremental algorithm*.[1]

The incremental algorithm starts with the initial bounding rectangle (that is, one trapezoid) and then we add the segments of the polygonal subdivision one by one in random order. As each segment is added, we update the trapezoidal map. Let S_i denote the subset consisting of the first i (randomly permuted) segments, and let $\mathcal{T}_i$ denote the resulting trapezoidal map.

[1]Historically, the randomized incremental algorithm that we will discuss arose as a method for solving a more general problem, namely computing the intersection of a collection of line segments. Given n line segments that have I intersections, this algorithm runs in $O(I + n \log n)$ time, which is superior to the plane sweep algorithm we discussed earlier. The original algorithm is due to Ketan Mulmuley.

To perform this update, we need to know which trapezoid of the current map contains the left endpoint of the newly added segment. We will address this question later when we discuss point location. We then trace the line segment from left to right, by "walking" it through the existing trapezoidal map (see Fig. 3). Along the way, we discover which existing trapezoids it intersects. We go back to these trapezoids and "fix them up". There are two things that are involved in fixing process.

- The left and right endpoints of the new segment need to have bullets fired from them.
- One of the earlier created walls might hit the new line segment. When this happens the wall is trimmed back. (We store which vertex shot the bullet path for this wall, so we know which side of the wall to trim.)

The process is illustrated in Fig. 3, where we insert a new segment (red) into the trapezoidal map from Fig. 1.

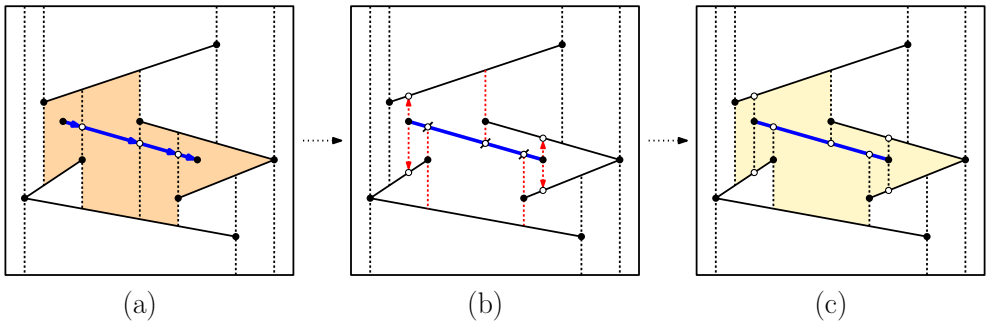

Fig. 3: Inserting a segment into the trapezoidal map: (a) Locate the left endpoint and trace the segment, (b) shoot bullet paths from endpoints and trim walls that have been crossed, (c) four original trapezoids (shaded red) have been replaced by seven new trapezoids (shaded yellow).

Observe that the structure of the trapezoidal decomposition does *not* depend on the order in which the segments are added. (This fact will be exploited later in the running time analysis, and it is one of the reasons that trimming back the walls is so important.) Ignoring the time needed to determine the trapezoid that contains a segment's left endpoint (which will be discussed later), the time needed to insert a segment is proportional to the number of bullet paths it crosses. For the sake of our later analysis, it will be useful to express the insertion time in terms of the number of trapezoids created.

Lemma: Ignoring the time spent to locate the left endpoint of an segment, the time that it takes to insert the ith segment and update the trapezoidal map is $O(k_i)$, where k_i denotes the number of newly created trapezoids.

Proof: Consider the insertion of the ith segment, and let w_i denote the number of existing walls that this segment intersects. We need to shoot four bullets (two from each endpoint) and then trim each of the w_i walls, for a total of $w_i + 4$ operations that need to be performed. If the new segment did not cross any of the walls, then we would get exactly four new trapezoids. For each of the w_i walls we cross, we add one more to the number of newly created trapezoids, for a total of $w_i + 4$. Thus, letting $k_i = w_i + 4$ be

the number of trapezoids created, the number of update operations is exactly k_i. Each of these operations can be performed in $O(1)$ time given any reasonable representation of the trapezoidal map as a planar subdivision, for example, a doubly connected edge list (DCEL).

Analysis: We will analyze the expected time to build the trapezoidal map, assuming that segments are inserted in random order. (Note that we make no assumptions about the spatial distribution of the segments, other than the fact they do not intersect.) Clearly, the running time depends on how many walls are trimmed with each intersection. In the worst case, each newly added segment could result in $\Omega(n)$ walls being trimmed, and this would imply an $\Omega(n^2)$ running time. (Too slow!)

We will show, however, that the expected running time is much smaller, in fact, we will show the rather remarkable fact that, each time we insert a new segment, the expected number of wall trimmings is just $O(1)$. (This is quite surprising at first. If many of the segments are long, it might seem that every insertion would cut through $O(n)$ trapezoids. What saves us is that, although a long segment might cut through many trapezoids, it shields later segments from cutting through many trapezoids.) As was the case in our earlier lecture on linear programming, we will make use of a backwards analysis to establish this result.

There are two things that we need to do when each segment is inserted. First, we need to determine which cell of the current trapezoidal map contains its left endpoint. We will not discuss this issue today, but in our next lecture, we will show that the expected time needed for this operation is $O(n \log n)$. Second, we need to trim the walls that are intersected by the new segment. The remainder of this lecture will focus on this aspect of the running time.

From the previous claim, we know that it suffices to count the number of new trapezoids created with each insertion. The main result is that the expected number of newly created trapezoids is a constant, no matter how many segments have already been inserted.

Lemma: Consider the randomized incremental construction of a trapezoidal map, and let k_i denote the number of new trapezoids created when the ith segment is added. Then for $1 \leq i \leq n$, $E[k_i] = O(1)$. (The expectation is taken over all $n!$ possible insertion orders of the segments.)

Proof: The analysis will be based on a backwards analysis. Recall that such an analysis involves analyzing the expected value assuming that the last insertion was random.

Let $\mathcal{T}_i$ denote the trapezoidal map resulting after the insertion of the ith segment. Because we are averaging over all permutations, among the i segments that are present in $\mathcal{T}_i$, each one has an equal probability $1/i$ of being the last one to have been added.[2] For each of the segments s we want to count the number of trapezoids that would have been created, had s been the last segment to be added.

We say that a trapezoid Δ of the existing map *depends* on an segment s, if s would have caused Δ to be created had s been the *last* segment to be inserted. (For example, in Fig. 4(a), the shaded trapezoids depend on s, and none of the others do.)

[2] An important feature of trapezoidal maps is that the structure of the map does not depend on the insertion order. So, any segment in the current diagram could have been the last.

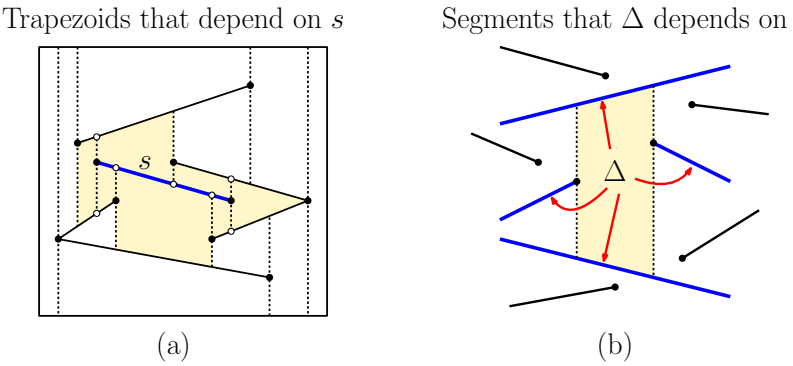

Fig. 4: Trapezoid-segment dependencies.

We want to count the number of trapezoids that depend on each segment, and then compute the average over all segments. Let $\delta(\Delta, s) = 1$ be an *indicator variable*, which is defined to be 1 if segment Δ depends on s, and 0 otherwise. The number of trapezoids that depend on segment s in $\mathcal{T}_i$ is $\sum_{\Delta \in \mathcal{T}_i} \delta(\Delta, s)$. Thus, the expected value is

$$E[k_i] = \sum_{s \in S_i} (\text{prob. that } s \text{ is last})(\text{num. of trapezoids that depend on } s)$$

$$= \sum_{s \in S_i} \frac{1}{i} \left(\sum_{\Delta \in \mathcal{T}_i} \delta(\Delta, s) \right) = \frac{1}{i} \sum_{s \in S_i} \sum_{\Delta \in \mathcal{T}_i} \delta(\Delta, s).$$

Some segments might have resulted in the creation of lots of trapezoids and others would have resulted in very few. How can we analyze such an unruly quantity? The trick is, rather than counting the number of trapezoids that depend on each segment, we swap the order of the two summations and instead count the number segments upon which each trapezoid depends. In other words we can express the above quantity as:

$$E[k_i] = \frac{1}{i} \sum_{\Delta \in \mathcal{T}_i} \sum_{s \in S_i} \delta(\Delta, s).$$

This quantity is much easier to analyze. In particular, each trapezoid is bounded by at most four sides. (The reason it is "at most" is that degenerate trapezoids are possible which may have fewer sides.) The top and bottom sides are each determined by a segment of S_i, and clearly if either of these was the last to be added, then this trapezoid would have come into existence as a result. The left and right sides are each determined by a endpoint of a segment in S_i, and clearly if either of these was the last to be added, then this trapezoid would have come into existence.[3]

In summary, each of the decomposition trapezoid is dependent on at most four segments, which implies that $\sum_{s \in S_i} \delta(\Delta, s) \leq 4$. Since $\mathcal{T}_i$ consists of at most $3i + 1$ trapezoids we

[3]There is a bit of a subtlety here. What if multiple segments share the endpoint? Note that the trapezoid is only dependent on the first such segment to be added, since this is the segment that caused the vertex to come into existence. Also note that the same segment that forms the top or bottom side might also provide the left or right endpoint. These considerations only decrease the number of segments on which a trapezoid depends.

have

$$E[k_i] \;\leq\; \frac{1}{i}\sum_{\Delta\in\mathcal{T}_i} 4 \;=\; \frac{4}{i}\,|\mathcal{T}_i| \;\leq\; \frac{4}{i}\,(3i+1) \;=\; O(1).$$

We know that the total number of trapezoids in the end is at most $3n+1 = O(n)$. Since the expected number of new trapezoids created with each insertion is $O(1)$, it follows that the total number of trapezoids that are created (and perhaps destroyed) throughout the entire process is $O(n)$. This fact is important in bounding the total time needed for the randomized incremental algorithm.

Unfinished Business: The only question that we have not considered in the construction is how to locate the trapezoid that contains left endpoint of each newly added segment. We will consider this question, and the more general question of how to efficiently determine the trapezoid that contains a given point in our next lecture. There, we will see that this can be done in expected time $O(\log n)$ per segment, which leads to an overall running time of $O(n \log n)$.

Line-Segment Intersection Revisited: We have assumed that the line segments do not intersect, but you might wonder whether we can generalize the algorithm to handle the case where they do intersect. The answer is that we can readily adapt the algorithm to handle this.

The trapezoidal map definition is modified so that whenever two segments intersect, we shoot a bullet path upwards and downwards from this intersection point (see Fig. 5(a)). We can think of the original n segments as being split by the intersection points into a total of $O(n+m)$ nonintersecting subsegments. We can then think of the process as that of constructing a trapezoidal maps for these subsegments.

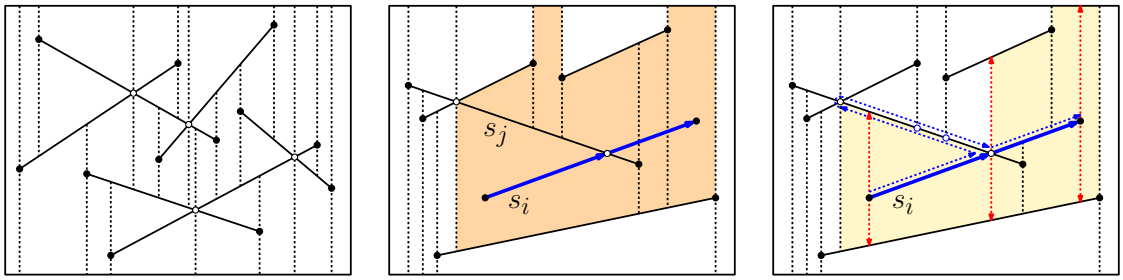

Fig. 5: Trapezoidal map of intersecting segments.

The principal new element of the algorithm involves the case when we are tracing a new segment s_i that intersects an existing subsegment s_j (see Fig. 5(b)). Observe that when this happens, the intersection point lies in two trapezoids, one above s_j and one below. We know the trapezoid that we arrived through, but we need to determine the trapezoid on the other side where the tracing continues. For concreteness, let's assume that the tracing of s_i hits the lower side of s_j. We walk along s_j, first tracing the bottom side and then wrapping around to the top side, until we reach the trapezoid containing the intersection point on the top side (see Fig. 5(c)). When we arrive at the intersection point on the top side of s_j, we shoot bullet paths up and down from the intersection point and continue with the tracing process.

How does the analysis change? The new element is the effort needed to process intersection events. The processing time is proportional to the number of new trapezoids created (same as the nonintersecting case) plus the number of bullet paths incident on the subsegments that are visited by the two-sided tracing process. While we will not show it here, it can be shown that in expectation, each subsegment contributes only a constant to this sum, which yields a total expected complexity of $O(n+m)$.[4] Thus, ignoring the time needed for computing the initial trapezoid containing the segment's left endpoints, the expected running time of the algorithm is $O(n+m)$.

Next time, we will show that this can be done in expected time $O(\log n)$ per segment endpoint, which leads to a total expected running time of $O(n \log n + (n+m)) = O((n \log n) + m)$. This beats the $O((n+m) \log n)$ running time of the plane sweep algorithm.

[4]The original analysis was done by Ketan Mulmuley, who observed that the trapezoidal map can be viewed as a planar graph with $O(n+m)$ vertices, and then exploiting the fact that the average degree of a face in any planar graph is $O(1)$.