Point Location using Trapezoidal Maps

Doug Zongker
CSE 523 project

December 18, 1996

When given a planar subdivision, a common task is locating the face that a given point lies in. The text gives an efficient algorithm for doing this using a *trapezoidal map*—the map of a set of segments is constructed using a randomized incremental algorithm in $O(n \log n)$ expected time, and point queries can be performed in $O(\log n)$ expected time.

The trapezoidal map of a set of noncrossing segments is constructed by placing the segments within an axis-parallel bounding box, and then extending the endpoint of each segment up and down until it hits another segment or the bounding box. Figure 1 shows a sample trapezoidal map for the set of thick line segments—the dotted lines represent the vertical extensions.

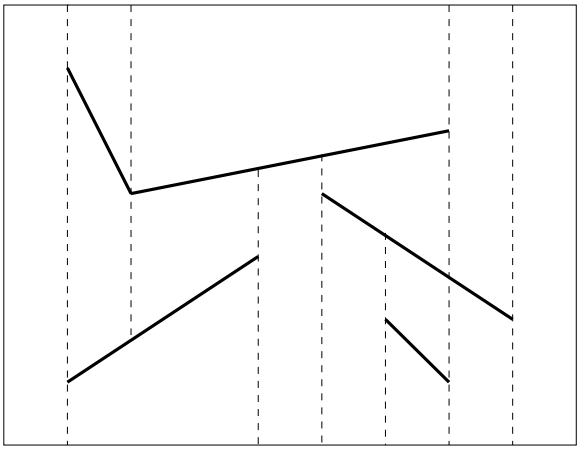

Figure 1: Trapezoidal map for a simple set of segments.

This divides the bounding box into a number of trapezoids whose parallel run in the vertical direction. Each trapezoid is uniquely specified by four objects: the segments that define its top and bottom (these will either be segments of the input set or the top or bottom of the bounding box) and the points whose vertical extensions define its left and right edges (endpoints of the input segments, or the corners of the bounding box).

Two trapezoids are *neighbors* if they share a common top or bottom segment. A given trapezoid can have up to four neighbors: upper left, upper right, lower left, and lower right. In Figure 2 below, trapezoid A possesses all four possible neighbors.

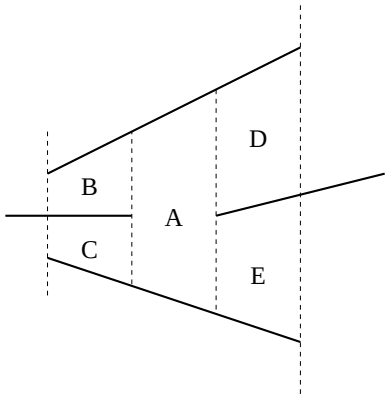

Figure 2: Neighbors of a trapezoid.

Degenerate zero-width trapezoids can be produced if multiple segment endpoints share an x-coordinate. The implementation specifies that if two points have the same x-coordinate, then the upper one (the one with the larger y-coordinate) lies to the "right." In cases like Figure 3, where it seems that the large trapezoid should have several neighbors on the right, this produces zero-width trapezoids so that each trapezoid has at most two right neighbors:

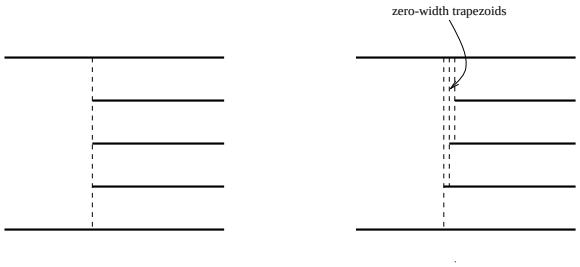

Figure 3: Zero-width trapezoids produced when vertical extensions coincide.

The algorithm works by constructing a search structure for performing point location in parallel with the trapezoidal map. It is an incremental algorithm, beginning with one trapezoid (the bounding box) and a trivial search structure. At each step, the search structure is used to locate the trapezoid(s) that the new segment falls in, then both the map and the search structure are updated.

The search structure takes the form of a directed acyclic graph. The search for the trapezoid a query point starts at the root and moves down until it reaches a leaf (which represents a trapezoid). There are two types of internal nodes:

point decisions and segment decisions. At nodes of the first kind, the query moves down to the left child if the query point is to the left of the decision point, or to the right child otherwise. Segment decision nodes cause the query to go left if the query point is above the segment, or to the right otherwise.

Adding a segment begins by querying the current search structure for the left endpoint of the new segment. Since the segment, if it leaves that trapezoid, must pass into either the upper or lower right neighbor trapezoid, we can determine in constant time what the next trapezoid is.

There are two cases for adding a segment. The first is when the new segment lies completely within an existing trapezoid T. In this case, T is shattered into four new trapezoids, and the leaf corresponding to T in the search structure is replaced with a small tree of depth 3 to select among the four new trapezoids, as in Figure 4.

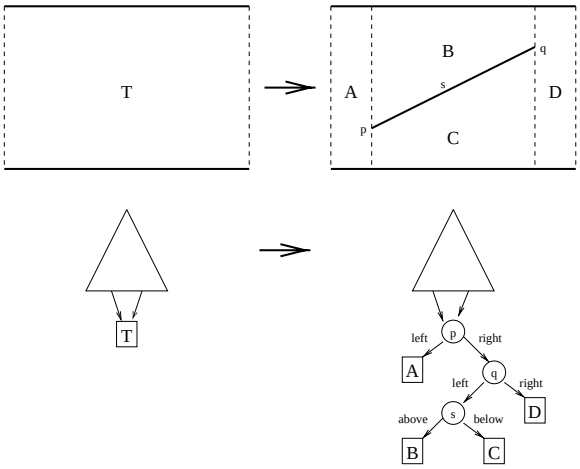

Figure 4: Updating the trapezoidal map and search structure when the new segment lies within a single trapezoid.

The implementation maintains the trapezoid map as a doubly-linked list of trapezoids—in addition to the *next* and *prev* fields, there are pointers to the trapezoid's neighbors, and its corresponding leaf in the search structure. This makes moving between the search structure and the map easy, and replacing portions of the map straightforward.

When the new segment crosses multiple existing trapezoids, as in Figure 5, the update is more complicated. First the trapezoid containing the left endpoint (A) is shattered into three trapezoids: the left one E, and the "current-above" trapezoid F and the "current-below" trapezoid G. E is placed on a temporary list of "new" trapezoids.

We now walk to the right along the new segments. Each trapezoid gets split into an upper and a lower trapezoid. If the lower one has a different bottom segment than the current-below trapezoid, then the current-below trapezoid

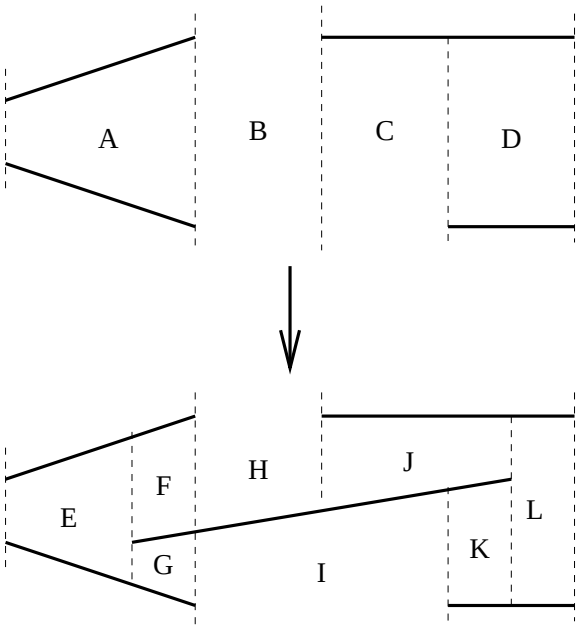

Figure 5: Updating the trapezoidal map when the new segment crosses multiple trapezoids.

gets pushed onto the new list and the lower trapezoid becomes the new current-below. (For instance, when B is split into H and I, G gets put on the new list and I becomes the new current-below trapezoid.) If the lower trapezoid shares its bottom segment with the current-below, then the trapezoids are merged: the current-below is extended to the right to subsume the lower trapezoid. A similar process occurs for the upper trapezoid and the current-above trapezoid, checking to see whether they share the top segment. After the last old trapezoid (D) is processed in this way, both the current-above and current-below trapezoids are pushed onto the new list, then the trapezoid to the right of the end point (L) is created and put on the new list.

Now all the old trapezoids can be deleted from the map and the entire new list spliced in. Each old trapezoid's search leaf is replaced with a segment decision node selecting the trapezoid above or below the new segment. (This is why the search structure is a dag: for instance, B and C are both replaced with segment nodes that point to trapezoid I.) The search structure for the end trapezoids also contain point decision nodes to select the trapezoids to the left and right of the new segment.