Original Lecture #6: 28 January 1991
Topics: Triangulating Simple Polygons
Scribe: Michael Goldwasser

Algorithms for triangulating polygons are important tools throughout computational geometry. Many problems involving polygons are simplified by partitioning the complex polygon into triangles, and then working with the individual triangles. The applications of such algorithms are well documented in papers involving visibility, motion planning, and computer graphics. The following notes give an introduction to triangulations and many related definitions and basic lemmas. Most of the definitions are based on a simple polygon, P, containing n edges, and hence n vertices. However, many of the definitions and results can be extended to a general arrangement of n line segments.

1 Diagonals

Definition 1. *Given a simple polygon, P, a* **diagonal** *is a line segment between two non-adjacent vertices that lies entirely within the interior of the polygon.*

Lemma 2. *Every simple polygon with $|P| > 3$ contains a diagonal.*

Proof: Consider some vertex v. If v has a diagonal, it's party time. If not then the only vertices visible from v are its neighbors. Therefore v must see some single edge beyond its neighbors that entirely spans the sector of visibility, and therefore v must be a convex vertex. Now consider the two neighbors of v. Since $|P| > 3$, these cannot be neighbors of each other, however they must be visible from each other because of the above situation, and thus the segment connecting them is indeed a diagonal. (See figure 1) □

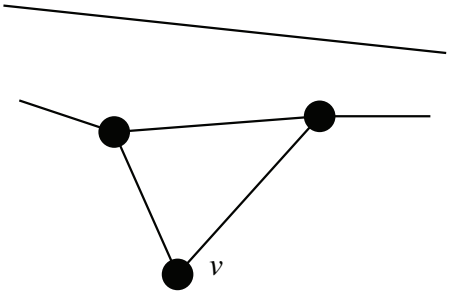

Figure 1: Existence of diagonal

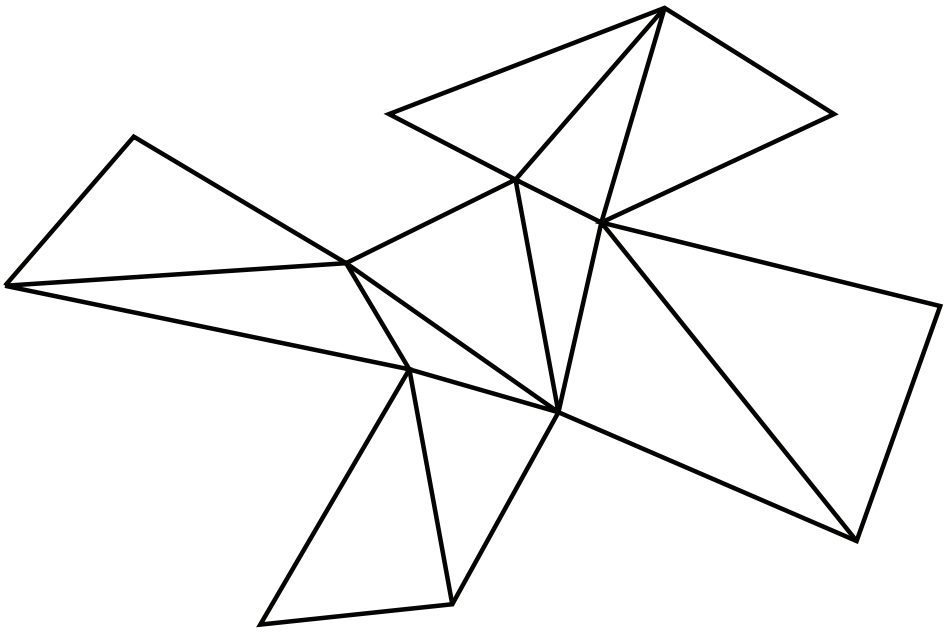

Figure 2: Example of a triangulated polygon

2 Triangulations

Now we can use these diagonals to prove the existence of triangulations for any simple polygon. First, we must formalize the concept of triangulating a polygon.

Definition 3. *Given a simple polygon P, a* **triangulation** *of P is a partition of the interior of P into triangles.*

An example of such a triangulation is given in Figure 2. Again, a similar definition can be given for a set of line segments, where a triangulation is a partition of the plane respecting the line segments. Often a less strict definition of triangulation is used, requiring the polygon to be partitioned into polygons of some bounded degree. In fact, later in these notes we will discuss the concept of a trapezoid partition of a simple polygon.

Lemma 4. *Every simple polygon has a triangulation.*

Proof: We will show this by induction on the number of sides. If the polygon has only 3 sides, then it is already triangulated. Otherwise, by Lemma 2, the polygon has a diagonal. Since the diagonal spans the interior, we can break the polygon into two pieces, adding the diagonal to each part. The size of each polygon is reduced by at least one, so by the inductive hypothesis, each part has a triangulation, and thus combining these gives a triangulation for the original polygon. □

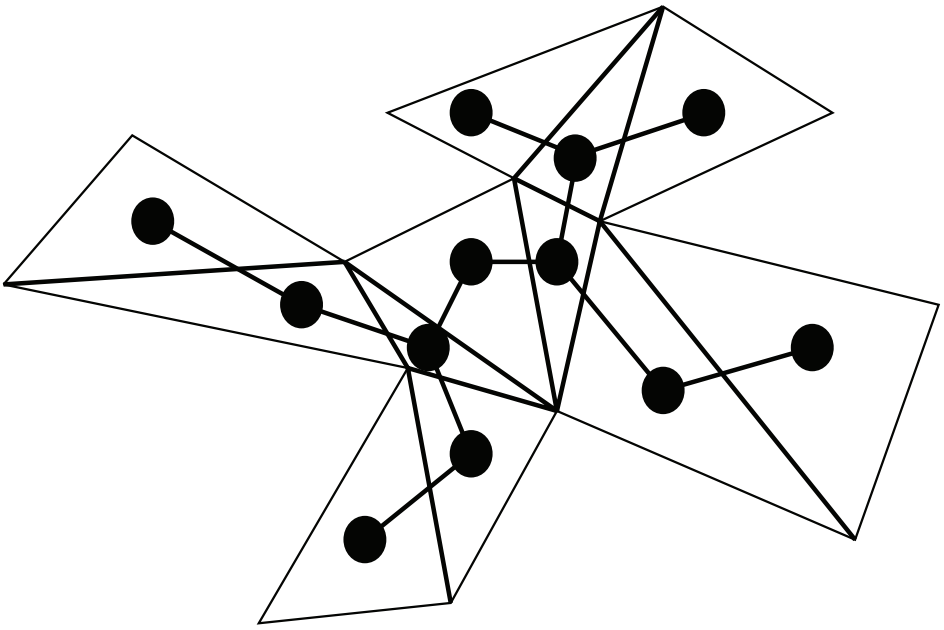

Figure 3: Example of dual graph

3 Dual Graph

Given a specific triangulation of a polygon, we can talk about the dual graph of the triangulation.

Definition 5. *Given a polygon P and a triangulation T for that polygon, the* **dual graph** *is defined as $\mathcal{D}(T) = (V, E)$, where $v_i \in V$ corresponds to a specific triangle in T, and $(v_a, v_b) \in E$ if the two corresponding triangles share an edge.*

Figure 3 provides an example of a dual graph. Notice that the edges in the graph are in one-to-one correspondence with the diagonals of the triangulation. Let's examine the properties of the dual graph.

Lemma 6. *Given a triangulation of a simple polygon, the resulting dual graph is a tree with maximum degree three.*

Proof: To see that the graph is a tree, recall that each edge corresponds to a diagonal of the triangulation. Since each diagonal breaks the polygon into two disjoint pieces, deleting an edge from the graph breaks the graph into two connected components. Thus the graph is a tree.

Since every triangle can have at most three neighbors, any node in the dual graph can have at most degree three. □

4 Counting

We consider some basic counting arguments involving various properties of triangulations and dual graphs. First, let $T(n)$ denote the number of triangles in a triangulation of an n-vertex polygon. Therefore $T(n)$ also denotes the number of nodes in the dual graph. Since each diagonal breaks the polygon into two smaller polygons who sizes add to $(n+2)$, we get the recurrence $T(n) = T(k) + T(n+2-k)$, with the base case $T(3) = 1$. Solving this we get that $T(n) = (n-2)$.

Let $D(n)$ denote the number of diagonals introduced for this triangulation, and therefore the number of edges in the dual graph. Counting the individual sides of the triangles, and noting that any diagonal will be counted twice, we have $T(n) = (n-2)$ triangles accounting for $(3n-6)$ sides. Since only n of those are the original sides, there must be exactly $(n-3)$ diagonals. Notice we could also realize that $D(n) = (n-3)$ by considering $D(n)$ to be the number of edges in a tree with $T(n)$ nodes.

A more interesting function to consider is how many different triangulations exist for a given polygon. This quantity is not the same for all n-vertex polygons; the maximum value is achieved for convex polygons. Let $F(n)$ denote the number of distinct triangulations of an n-vertex convex polygon. We show that $F(n) = b_{n-2}$, the number of binary trees on $n-2$ nodes, by showing that there is a one-to-one correspondence between triangulations of the polygon and $(n-2)$-node binary trees.

We already know that a triangulation defines an $(n-2)$-node binary tree, namely the dual graph. To make this tree unique, we must pick a root for it. Consider the triangle incident to a given edge of the polygon, say $\overline{p_1 p_2}$. Because one of the triangle's edges is $\overline{p_1 p_2}$, it has at most two triangle neighbors. Therefore, the corresponding node in the dual tree has degree at most 2, and we can make that node the root of the tree.

To prove the other direction of the correspondence, we must show that for every binary tree, there is a triangulation with that tree as its dual graph. We create a triangulation by a recursive process that triangulates a convex polygon with a distinguished *top edge*. The top edge for P is $\overline{p_1 p_2}$. Suppose that the vertices of the current convex polygon are $p_1, p_2, \ldots, p_m$ in counterclockwise order, that $\overline{p_1 p_2}$ is the top edge, and that the root of the given tree has k nodes in its left subtree. We introduce triangle $\triangle p_1 p_2 p_{k+3}$, partitioning the polygon into a triangle and two subpolygons of sizes $k+2$ and $m-k-1$. Recursively, we find a triangulation of the polygon $p_2, \ldots, p_{k+3}$, with top edge $\overline{p_{k+3} p_2}$, corresponding to the left subtree. Similarly, we find a triangulation of the polygon $p_1, p_{k+3}, \ldots, p_m$, with top edge $\overline{p_1 p_{k+3}}$, corresponding to the right subtree.

What is the value of $b_n = F(n+2)$? From Knuth, vol. 1, Sec 2.3.4.4, we get that

$$b_n = \frac{1}{n+1}\binom{2n}{n} = \frac{4^n}{n\sqrt{\pi n}}\left(1 + O\left(\frac{1}{n}\right)\right),$$

the n^{th} Catalan number, which is also the number of ways to parenthesize a formula.

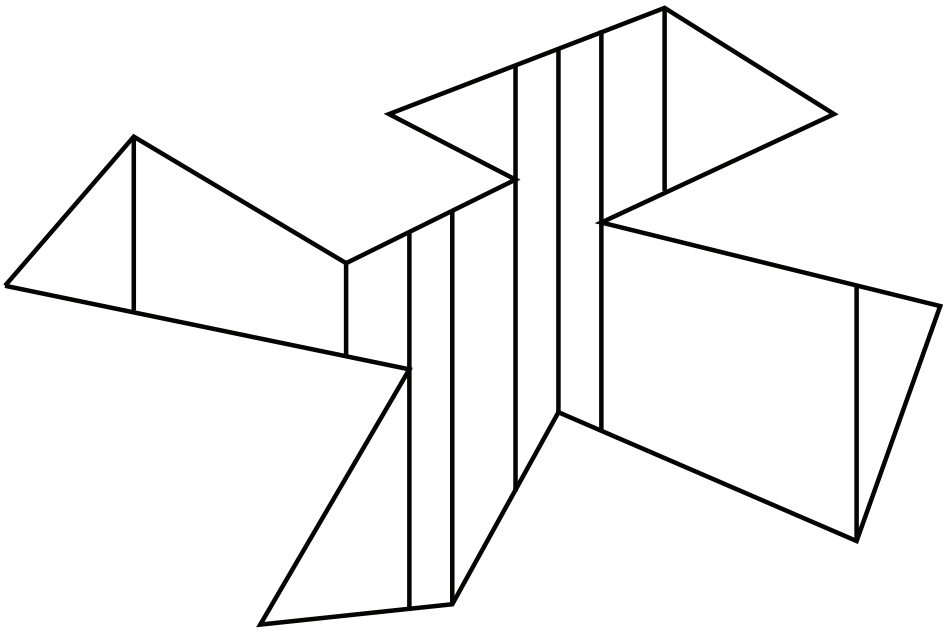

Figure 4: Example of canonical trapezoid partition

5 Ears

Definition 7. *Given a simple polygon P, an* **ear** *of the polygon is a vertex whose neighbors are visible from each other.*

Lemma 8. *Every simple polygon with $|P| > 3$ has at least two non-adjacent ears.*

Proof: From lemma 4, consider a triangulation of P and its dual graph. Since $|P| > 3$, the dual graph is not a single node, and since the dual graph is a tree, it must have at least two nodes of degree one. Our claim is that these two nodes correspond to non-adjacent ears. If a node in the dual has degree one, then two of the edges defining that triangle must be adjacent edges of the original polygon, and the third edge is the diagonal closing the triangle. Therefore the common vertex of the two polygon edges is an ear. Furthermore, since no polygon edge is used in more than one triangle, the second degree-one node cannot involve either of the polygon edges from the first, and therefore the ear corresponding to that node cannot be adjacent to the first. □

6 Trapezoid Partitions

In an earlier lecture we discussed the fact that adding vertical threads to an arrangement of lines partitioned the plane into trapezoids. We now review this process for simple polygons.

Definition 9. *Given a simple polygon, a* **trapezoid partition** *of the polygon is a partition of the interior of the polygon into trapezoids.*

Lemma 10. *Every simple polygon admits a trapezoid partition.*

Proof: We will give a constructive proof. Given a polygon, add in all of the vertical threads that are in the interior of the polygon. Namely, from each vertex, if that vertex is below the interior of the polygon, extend a vertical thread up from that vertex until it reaches the nearest edge. Similarly, if the vertex is above the interior of the polygon, extend a thread vertically down from the vertex.

Our claim is that the original edges of the polygon with these additional threads partitions the interior into trapezoids. Each face of our partition has one or two vertical threads as sides. If there are two threads, then they both must extend upward until each hits an edge of the original polygon. Our claim is that both sides must hit the same edge, and therefore the top of the face is a single segment. If the two sides hit two different edges, then there must be some vertex along the top chain visible, but if this were the case, there would have been a thread drawn from this vertex. A similar argument holds if the face was bounded by only one thread. A symmetric argument shows that each face has a single bottom segment, and therefore the face is a trapezoid. □

We will refer to the specific trapezoid partition constructed above as the *canonical* trapezoid partition. Figure 4 shows the canonical trapezoid partition of a simple polygon. Notice that every trapezoid is delimited on the left and right by an original vertex of the polygon since each thread is anchored to an original vertex. However, these left and right vertices can be on the top, middle, or bottom of the trapezoid. Also notice that the number of trapezoids is linear in the degree of the polygon since every polygon vertex has at most two incident threads, and the number of trapezoids is one greater than the number of threads.

7 Triangulating Unimonotone Polygons

We begin by providing the definitions for monotone chains and polygons, and for unimonotone polygons.

Definition 11. *A polygonal chain is* **monotone** *(w.r.t. the x-axis) if any vertical line intersects it in at most one point. A polygon is* **monotone** *if it is the union of two monotone chains. A polygon is* **unimonotone** *if it is the union of two monotone chains, one of which consists of a single edge.*

Lemma 12. *We can triangulate a unimonotone polygon in linear time.*

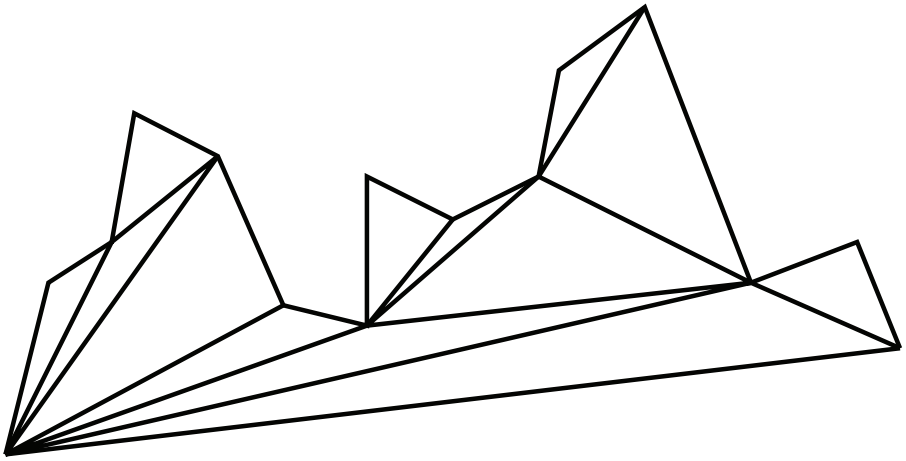

Figure 5: Triangulated unimonotone polygon

Proof: We give a constructive proof by presenting an algorithm satisfying these bounds. The algorithm creates an ear decomposition by cutting off ears. Notice that any convex vertex in the longer monotone chain must be an ear. Our algorithm processes the vertices starting with the leftmost and following the longer monotone chain until the rightmost vertex is reached. The algorithm is similar in design to the Graham-Yao algorithm for computing the convex hull of a simple polygonal chain. The first vertex is placed on a stack. If the next vertex is an ear, then the diagonal of that ear is added to our triangulation and the ear is popped from the stack. Then the algorithm reconsiders the vertex at the top of the stack and proceeds. To check whether a vertex is an ear is simply a constant time check of whether the angle at that vertex is convex. Each vertex is pushed and popped from the stack exactly once, so the algorithm runs in linear time. Figure 5 shows the resulting triangulation of a unimonotone polygon. □

8 Trapezoid Partitions and Triangulations

The problems of constructing a triangulation of a simple polygon and constructing the canonical trapezoid partition are very similar. In fact there is a linear time equivalence between the two problems. We will only take time to show the more useful half of this claim, namely that given the canonical trapezoid partition, we can in linear time compute a triangulation.

Theorem 13. *Given the canonical trapezoid partition for a polygon P, we can in linear time compute a triangulation for P.*

Proof: We give a constructive proof by presenting an algorithm creating the triangulation. Given the canonical trapezoid partition, there are two steps.

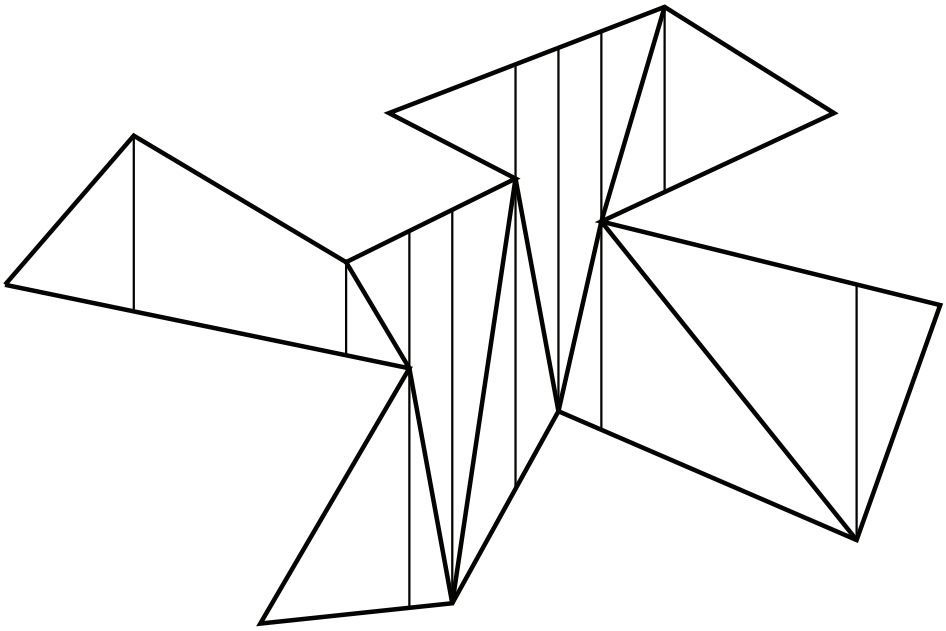

Figure 6: Trapezoid partition with "obvious" diagonals

1. Add "obvious" diagonals to P.
2. Triangulate the resulting (unimonotone) polygons.

First we must explain the interpretation of "obvious" diagonals. Second, we will claim that the polygons that result by adding these diagonals to P are unimonotone, and thus can be triangulated by the algorithm given in lemma 12.

Recall that each trapezoid in the canonical partition was delimited by exactly two vertices, one on the left side and one on the right side. For each trapezoid consider these two vertices. If the vertices do not already share an edge in the original polygon, then these two vertices define what we call an "obvious" diagonal. Notice that since the trapezoids are convex, the line segment connecting these vertices remains inside the trapezoid, and since the trapezoid is contained in the polygon, this is a valid diagonal for the original polygon. Also since the trapezoids are disjoint, we can add all such diagonals without fear of having any of them intersect each other. Figure 6 shows the obvious diagonals of a trapezoid partition. Adding all of these diagonals clearly takes linear time, since there are linearly many trapezoids, and checking and adding the diagonal for each one is a constant time operation.

Now consider the components created by breaking up the original polygon with these "obvious" diagonals. We claim that each such component is in fact a unimonotone polygon. The key fact is that the canonical trapezoid partition for the original polygon, when limited to one of these components, gives us the canonical trapezoid partition for that component. Our first claim is that each component is x-monotone. Assume a non-monotone component exists, and therefore has a cusp at some point.

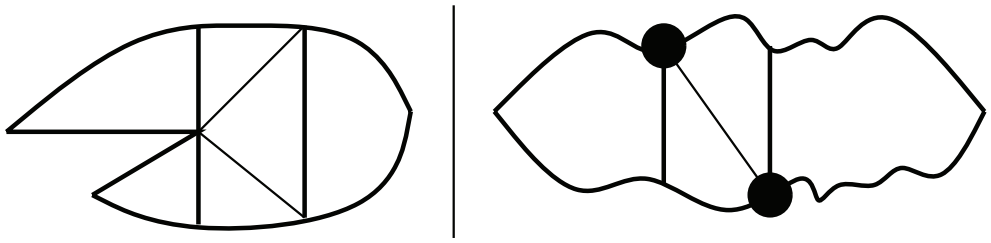

Figure 7: Proof of unimonotone condition

(See figure 7.) Consider the trapezoid immediately adjacent to that cusp, and its two defining vertices. One of these vertices will be the cusp vertex, and the other will be on the opposite side of the trapezoid. However, since the cusp is in the middle of the side, then no matter where the opposite vertex is, there will be an "obvious" diagonal between the two. Such a diagonal would have broken this component, and so there could not have been such a cusp. Therefore we know that each component is x-monotone. Now, assume that we have such an x-monotone component that is not unimonotone. Then both the upper and lower chains must have at least one internal vertex each. Therefore there must exist some trapezoid where one side contained a vertex from the top chain and the other side contained a vertex from the bottom chain. This trapezoid would contain an "obvious" diagonal, a contradiction. Therefore, one of the chains must be a single edge. (See figure 7.) Thus we have proven that every component created by step (1) is a unimonotone polygon. By lemma 12 we can triangulate each such component in linear time, and since we added only a linear number of diagonals to the original polygon, the combined size of all of these components is still linear.

We have shown that both steps of our algorithm run in linear time and produce the desired triangulation. □

9 Building the Trapezoid Partition

The previous section implies that, given any algorithm to construct the canonical trapezoid partition, we immediately have a triangulation algorithm with the same time bound. In this section we present a simple sweepline algorithm that builds the trapezoid partition in $O(n \log n)$ time. In the following lecture, a randomized, incremental algorithm by Seidel will be presented which creates the trapezoid partition in $O(n \log^* n)$ time.

The following algorithm takes a simple polygon P and creates the canonical trapezoid partition of P in $O(n \log n)$ time. The algorithm is a standard left-to-right sweepline method. The events are the vertices of the polygon. We maintain the intersections with the sweep line and the segments of the polygon. Also with each segment in the sweepline structure, if the interior of the polygon is immediately below that segment,

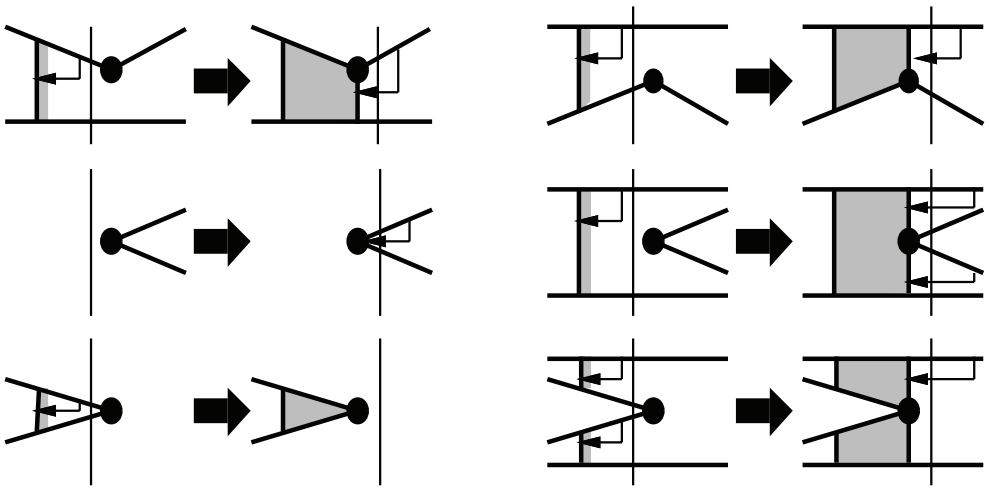

Figure 8: Sweepline operations for trapezoid partition

we store the rightmost vertical trapezoid edge directly below it. This pointer stores the left side of the currently half-open trapezoid with that segment as the top side. There are six possible event types that occur in the algorithm, and the operations for each of these cases is demonstrated in figure 8. Although not given formally in these notes, the implementation details of this algorithm are quite simple.

The proof of correctness is not given in these notes. In general, our invariant claim is that all trapezoids completely to the left of the sweepline have already been processed and reported.

The running time analysis for this algorithm is simple. Since all the events are known beforehand, the event list can be sorted beforehand in $O(n \log n)$ time. The sweepline is maintained as a balanced tree, and since the segments are all from the polygon, there can be at most n segments crossing the sweepline at once. Therefore locating an event in the sweepline can be done in $O(\log n)$ time, and the operations and reporting are all done in constant time for each event. It follows that the n events can be processed in $O(n \log n)$ time, which gives the desired bound on the overall time of the algorithm.

Notice also that the working storage used is linear. The event queue consists of the n vertices, and thus is linear. Also the sweepline structure contains at most n segments, and for each segment, the data stored uses constant space.