Minimum Link Path Within Simple Polygons

Lecture 22
Date: April 28, 1993
Scribe: Scott Sona Snibbe, Nisha Thatte, George Wynne

1 Introduction

Given a simple polygon P with n vertices, we want to find a minimum link path from a point x to a point y. The path has the characteristics that while it can touch edges, it cannot cross them. The basic strategy is to follow edges. We want to minimize the number of bends in the path and therefore the number of links in the path is the maximum of the number of edges. This problem is related to the visibility problem. There are many applications that can take advantage of this algorithm. For example, robots need to know how to walk so they can minimize the number of turns they take. Another use is in microwave communications. Everytime a signal needs to bend on its way from the transmitter to receiver, there must be a repeater to handle that. This problem minimizes the number of repeaters needed.

2 Method

We need to divide the polygon into visibility polygons so that each visibility polygon must go through at least one vertex of P and the resulting partitioning is still a planar graph. Starting with point x, we determine how much can be seen from that point by shooting rays in all directions and using the ones that go the farthest without being obstructed by an edge.

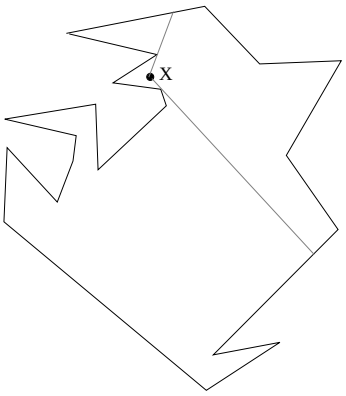


Figure 1: Determining the visible regions from x

Then from each following window, determine how much can be seen from there. In Fig. 2 Visibility Polygon B is easy to determine because all that is left there can be seen from the

window. However, there are several rays needed to create the next visibility polygon because there are many sections of the diagram that can only be partially seen from different points on the window.

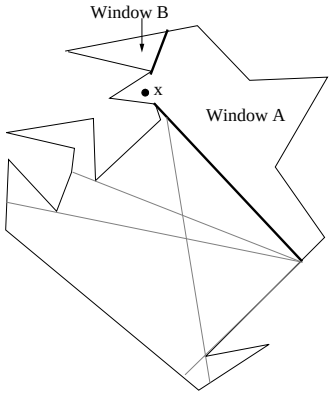


Figure 2: Determining the visible regions from Visibility Polygon A

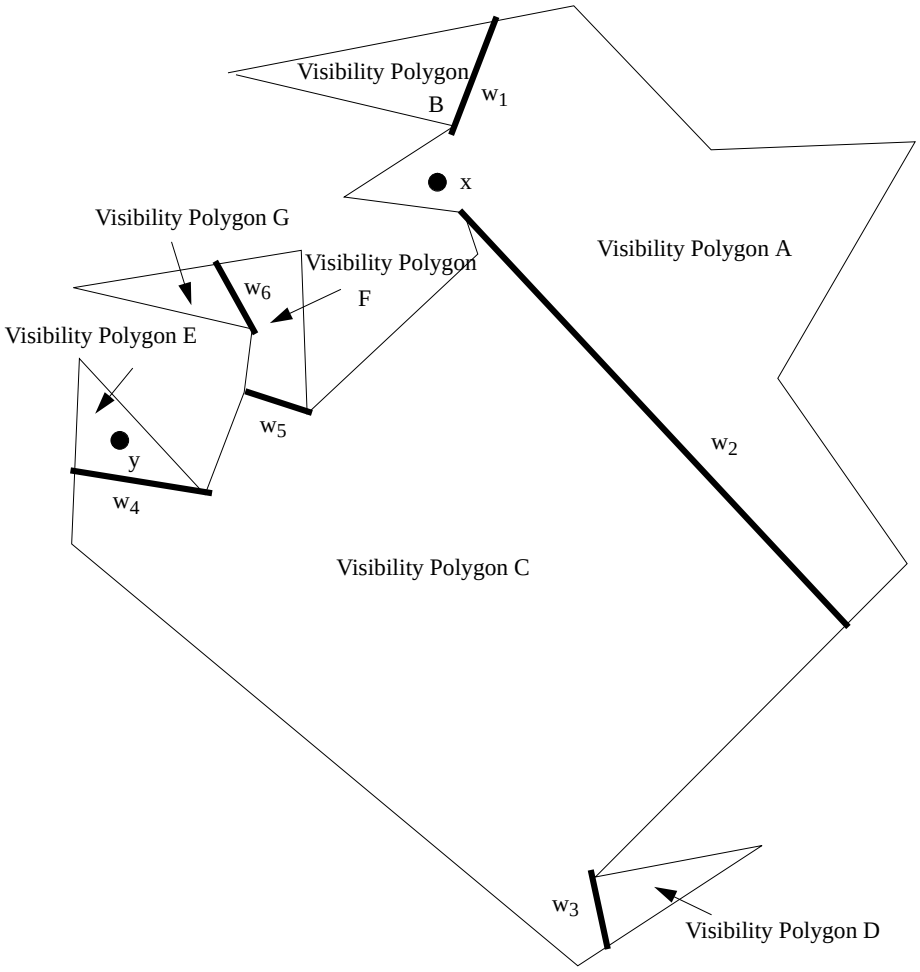


Figure 3: The final partitioning

If we continue with this trend, we end up with a window partitioning as in in Fig. 3. This will end up being $O(n)$ regions since in the worst case, we will have only one vertex to each region.

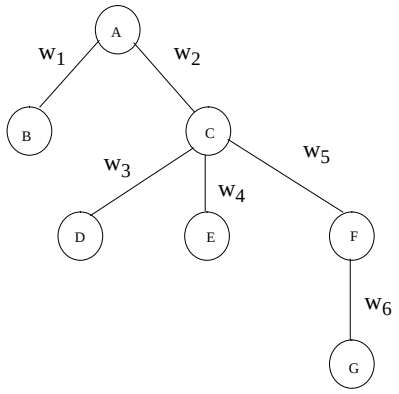


Figure 4: Window Tree

To determine the path to any point, we first need to create a tree. The root is the window that the starting point X is contained in. Each adjacent window is a child of the root. For each child, we create children from its adjacent windows. The height of the tree is $O(n)$ since in the worst case we can end up with $0(n)$ regions and each region is only adjacent to one other region.

3 Performing a Query

Given the fixed point x and a partition of polygon P into visibility polygons, we are now ready to perform a path query for an arbitrary point y. First, we perform a point location to identify the region R containing y. In our example this is region E. This can be performed in $O(\log n)$ time using Simple Polygon Inclusion with the window tree.

Once the region containing y is found we look up its corresponding node μ in the window tree. The number of links in the minimum link path $d_L(x, y)$ will be the depth of the tree at μ plus 1.

Now we trace from μ to the root and collect the encountered windows $w'_1, \ldots, w'_k$. In our example, this set consists of (w_4, w_2). For each $i, 1 \leq i < k$ we project w'_i to w'_{i+1}. The vertex v_i is found by intersecting the projection of w'_i with w'_{i+1}: $v_i = \vec{w'_i} \cap w'_{i+1}$ (figure 5).

To compute the last link we must find the line connecting y to w'_k that does not intersect and edges of P. The process is illustrated in figure 6. The last partition line w'_k is shown with endpoints a and b. First we compute the shortest path from y to a, $SP(a, y)$. This can be done in $O(\log n)$. Now we extend the first link $y\vec{v}$ to intersect w'_k. Where this intersection occurs we choose vertex v_k and connect it to y to complete our path. The intersection process is $O(1)$.

Theorem 1 *The intersection of the first segment of the shortest path $SP(y, a)$ and $w'_k = \overline{ab}$ is non-null.*

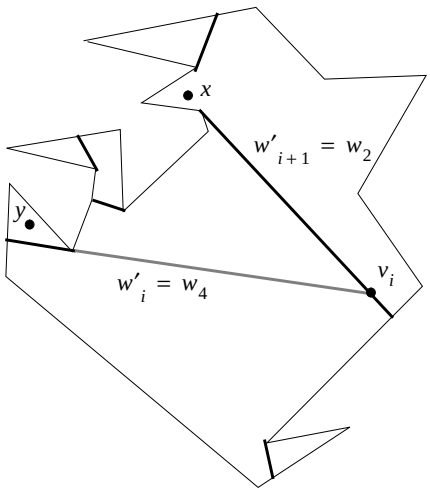


Figure 5: Projecting the a window partition to find v_k.

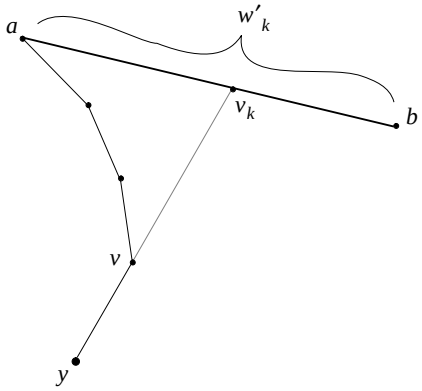


Figure 6: Computing the last link of the minimum link path.

Proof:

Let $\vec{s}$ be the first segment on $SP(y, a)$. If $\vec{s} \cap \overline{ab} = \emptyset$ then y would not be able to see $\overline{ab}$, and it would not be w'_k for the path from x to y. However, ab is by definition the last window w'_k. Therefore $\vec{s} \cap w'_k \neq \emptyset$. □

Now we have completed the computation of Minimum Link Path. The path is defined by the segments from x to y through v_1 to v_k:

$$LP(x, y) = (x, v_1, v_2, \ldots, y)$$

The total time complexity for a query is $O(\log n + k)$. The point location takes at most $\log n$ time and k time is used to compute the intermediate vertices.

4 Partitioning the Polygon

The partitioning of the polygon P is best done in the following steps:

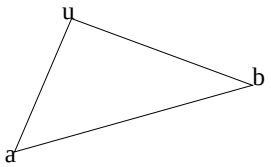
1. Triangulate P.

2. From x, compute the visibility polygon $V(x)$. Call the edges of $V(x)$ which are not edges of p windows $w_1, w_2, ...w_n$.

3. From each window w, compute the visibility polygon $V(w)$. Now we only need to describe how the visibility polygon $V(w)$ is calculated. (We can think of the computation of $V(x)$ as a special case, in which x is a window of length 0.)

4.1 Computing $V(w)$

The general idea is to recursively look at edges, steadily moving away from w, until w has "seen" all it can see. The following example demonstrates the basic idea.



Thus, $V(\overline{ab}) = V(\overline{au}) + V(\overline{bu})$. If $\overline{au}$ is an edge of p, then the recursive process stops and $V(\overline{au}) = \overline{au}$. Similarly for $\overline{bu}$.

But this is the simplest case. We now need to look at a more complicated example. Supposing that we have calculated the part of $\overline{cd}$ that is visible from $\overline{ab}$ (which we will represent as $Vis(\overline{cd} : \overline{ab})$), we will try to find $Vis(\overline{cg} : \overline{ab})$. This breaks down into the following two cases.

Case 1:

Because $SP(a, c) \cap SP(b, g) \neq \emptyset$
$\Rightarrow Vis(\overline{cg} : \overline{ab}) = \emptyset$

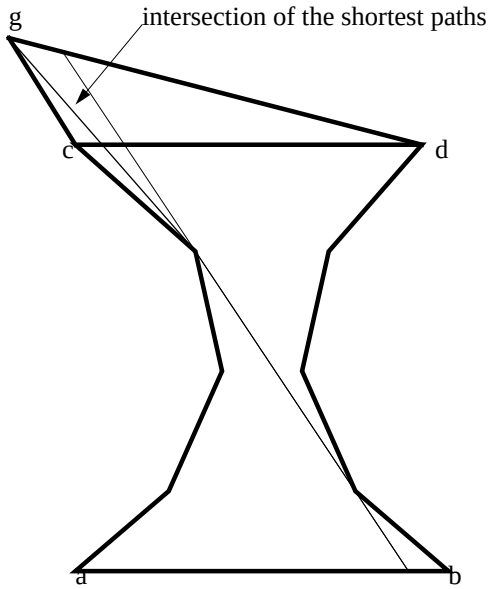


Figure 7: case 1

Case 2:
Because $SP(a,c) \cap SP(b,g) = \emptyset$
$\Rightarrow Vis(\overline{cg} : \overline{ab}) \neq \emptyset$
Now we have two options:

1. If $\overline{cg}$ is a diagonal, then recursively call $Vis(\overline{cg} : \overline{ab})$
2. Otherwise, $\overline{cg}$ is an edge of the polygon, and we have the following two subcases:

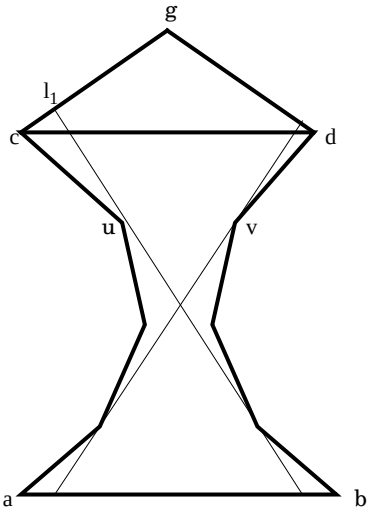


Figure 8: case 2a - $Vis(\overline{cg} : \overline{ab}) = (\overline{ul_1}, \overline{l_1 g})$

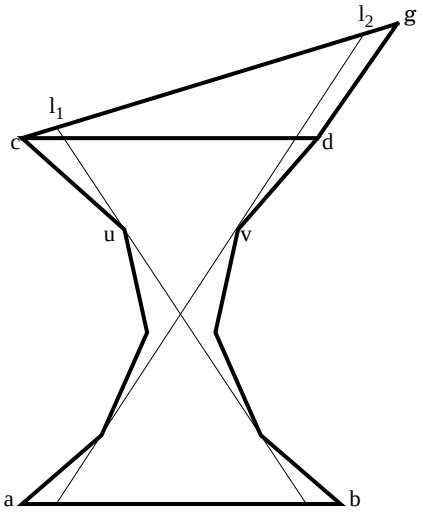
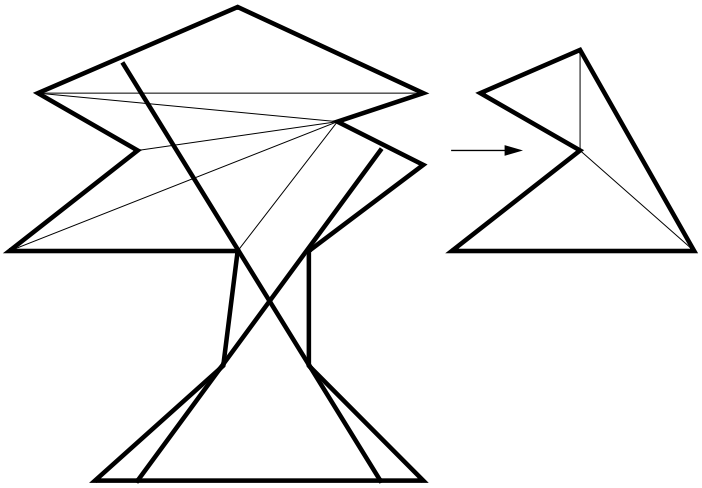


Figure 9: case 2b - $Vis(\overline{cg} : \overline{ab}) = (\overline{ul_1}, \overline{l_1 l2}, \overline{l_1 v})$

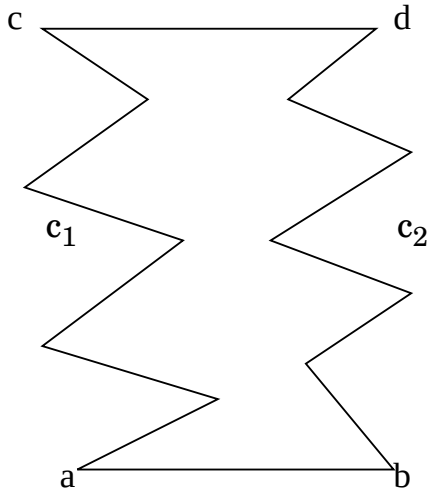
4.2 Retriangulation

The window $w = \overline{ab}$ must be an edge of a triangle. If it is not, the parts of the polygon outside of $V(\overline{ab})$ must be retriangulated. The two resulting polygons which must be retriangulated are unimonotone polygons, and they can easily be triangulated by just cutting off the convex corners. The time for this retriangulation = O(the number of triangles intersecting $V(\overline{ab})$.



4.3 The Convex Hull

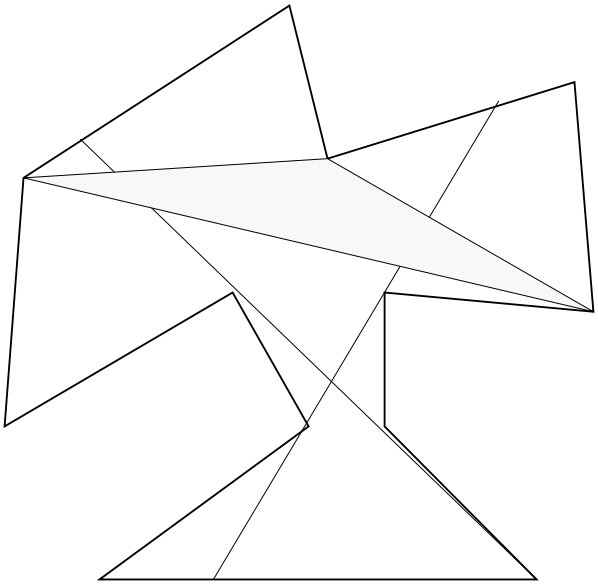
$SP(a, c)$ and $SP(b, d)$ and internal common tangents can be maintained in time O(the number of trianlges traversed). $SP(a, c)$ = the right convex hull of c_1 Use the algorithm for convex hull: visit the points in increasing y-coordinates and throw away the "concave" points. (Once they are thrown away, they are thrown away forever).



$\Rightarrow$ the time for $V(w)$ = the number of triangles intersected by $V(w)$.

Theorem 2 *Each triangle can intersect at most three visibility polygons in the window partition.*

Proof:



The above figure shows that a triangle can be in three visibility polygons. But since every point in the triangle is visible from any window that intersects the triangle, the triangle cannot be in more than three visibility polygons. □

$\Rightarrow$ the total time for constructing the window partition is $O(n)$.