CS-7491-A: 3D Complexity Techniques for Graphics, Modeling, and Animation
Lecture 15

Nipun Kwatra

1 Discussion on the project 2

The project 2 consists of main 3 parts

1. Finding Boundaries from image. (Discussed in previous lectures)
2. Displaying
3. Clicking. Determine whether a current point is inside a region and if yes then which region.

We will mainly discuss how to solve part 3. Consider the figure 1, and think of the child analogy in which a child is placed on an edge and moves in one direction following the rightmost edge and painting his side of the edge. To make sure that we move around the boundaries of all the regions, we should have gone through each of the half edge, i.e. both directions of each edge should be painted by a child. One can think of placing a boy on one side of the edge and a girl on the other side, both moving in opposite directions, following the rightmost edge on their way and painting their sides of the edges.

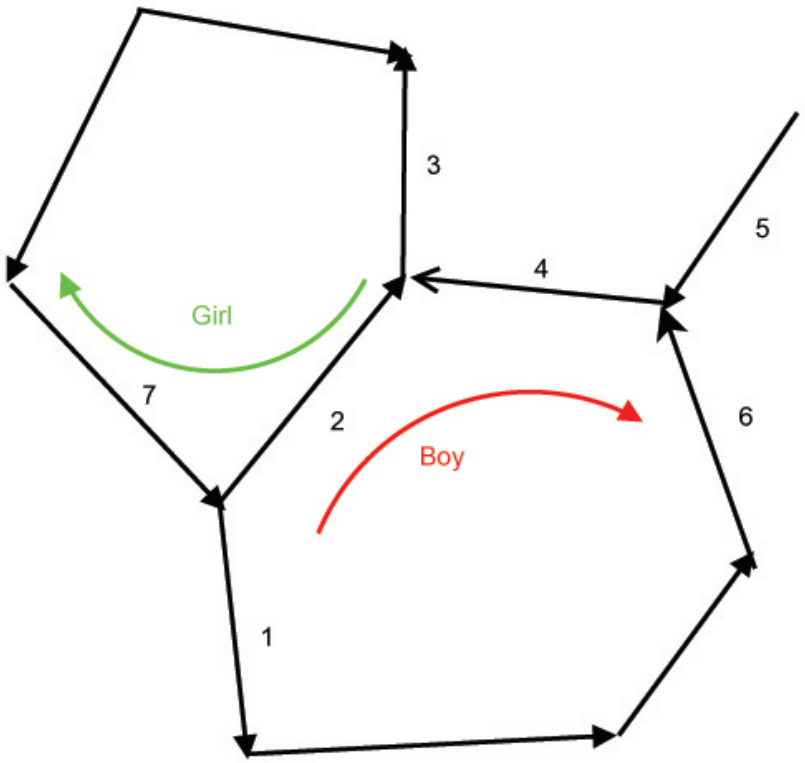

Figure 1: The boy and girl move start from edge 2, moving in opposite directions

1.1 Finding the next and previous edges

To follow the rightmost edges in both direction, we first precompute these rightmost edges for each edge. For and edge e, we call the rightmost turn going forward the next edge ($e.n$), and the rightmost turn going backwards the previous edge ($e.p$) (See figure 2). Now we describe how to compute $e.n$ and $e.p$.

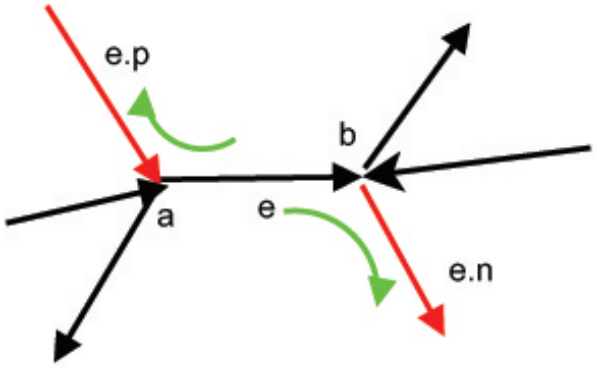

Figure 2: The next and previous edges for an edge

Let $e \equiv (a,b)$. To compute $e.n$, we first find all edges e_i s.t. one vertex of the e_i is b. We re-orient the edges, so that the starting vertex of each e_i is b, i.e. $e_i \equiv (b,x)$. Also reverse the edge e, so that all the edges now have the same starting point b. The right most edge can be found as shown in the figure 3 – take the dot product of e_i with e and $e.left$ (vector orthogonal to e) to give scalars c and s, respectively. The angle of e_i from e in the anticlockwise direction can then be found using the function atan2(s,c). The vector with the minimum angle, is the rightmost/next adjacent edge $e.n$. The edge $e.p$ can be found in a similar manner.

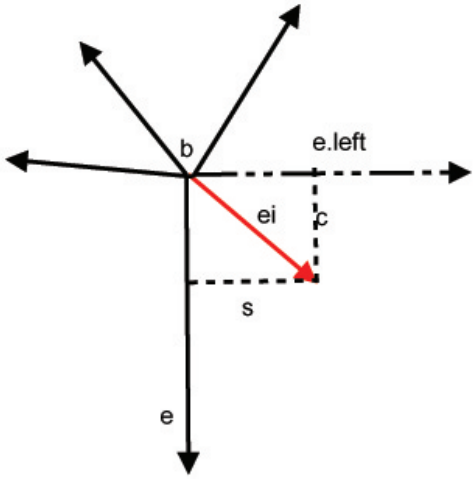

Figure 3: Computing the angle between vectors

One can keep this information in an edge table. e.g. the table for some of the edges in figure 4 is shown in table 1.

1.2 Creating Loops

Now since the orientation of each edge may be arbitrary, we need to keep track of our current direction while going around a loop (e.g. boy going around the loop). The following algorithm can be used to travel the loop starting at

Edge	V_start	V_end	Next	Prev
2	-	-	4	7
3	-	-	-	4
4	-	-	3	6

Table 1: Edge table with next and previous edge information for some edges

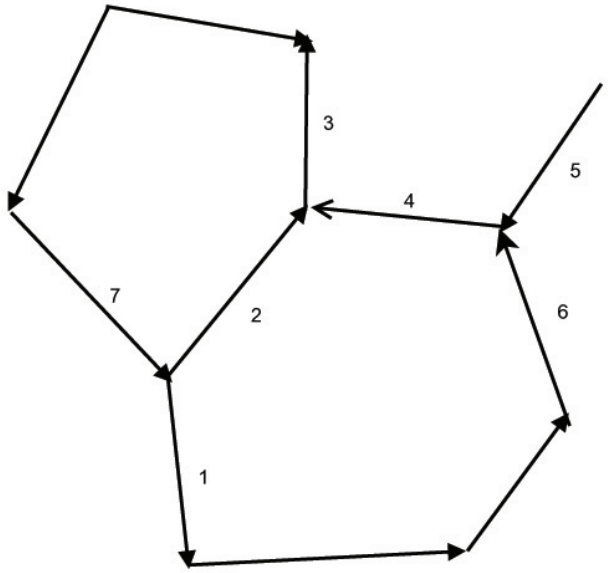

Figure 4: An example region

edge *e* in the direction *dir*

```
travelLoop(e, dir){
  while( initial edge not reached )
  {
    loop.push(e);
    if(dir==FORWARD)
    {
      e.markForward=true;
      if(e.end == e.n.end){
        dir=BACKWARD;
      }
      e=e.n;
    }
    else
    {
      e.markBackward=true;
      if(e.start == e.p.start){
        dir=FORWARD;
      }
      e=e.p;
    }
  }
```

```
  return loop;
}
```

Visiting All Loops: To visit all the loops, we use the following pseudo code.

```
foreach edge e{
  if(e.markForward==false){
    travelLoop(e,Forward);
  }
  if(e.markBackward==false){
    travelLoop(e,Backward);
  }
}
```

We can now store just the seed edges from which the traversals started and the start direction of the corresponding traversal. This information can be used to reconstruct any loop by calling the *travelLoop* routine.

1.3 Determining if a point is inside a loop

This can be determined by shooting a ray from the point P and computing the number of times it intersects with the loop. If the number of intersections is odd, the point is inside the loop, else the point is outside that loop. Note that we also need to check for special cases:

1. Ray intersects a corner of the loop. This can be handled by jittering the ray a little bit. Or one can use the trick suggested in a previous lecture by counting the number of edges to the right of the ray.

2. Point P lies on the loop. This needs to be checked separately, and a consistent convention could be followed.

Another way to determine whether a point is inside a poly-loop, would be to calculate the angles between line segments formed by jointing P to the corners of the polyloop. If these angles sum to 2π, the point is inside the loop, else not.

2 Triangulation of a Polygon

The aim is to *partition* a polygon into a set of triangles. One approach, as shown in figure 5 would be *Ear Cutting*. An *ear* is defined to be a triangle of vertices (A,B,C), s.t. B is convex, i.e. the triangle ABC should not contain any other vertex of the polygon. The algorithm would cut an ear in each step until the entire polygon is cut.

Question: Can we have a lock? i.e. is it possible that during the ear cutting algorithm, we don't have any ear even when the polygon is not fully cut?

It can be proved that if there is no hole in the polygon, an ear can always be found. i.e. the only case when an year cannot be found is when there is a hole in the polygon. An example is shown in figure 6

2.1 Handling polygons with holes

This can be handled in ways we have discussed before. e.g. just add dummy edges (bridge) as shown in figure 7

2.2 Detection holes

How do we detect that the polygon has any holes or not? It can be easily seen that there is a hole in the polygon if and only if the graph of the vertices and edges and the polygon is not simply connected. Thus we can just make a graph from the polygon and run an exhaustive search like DFS/BFS to check if the graph is connected or not.

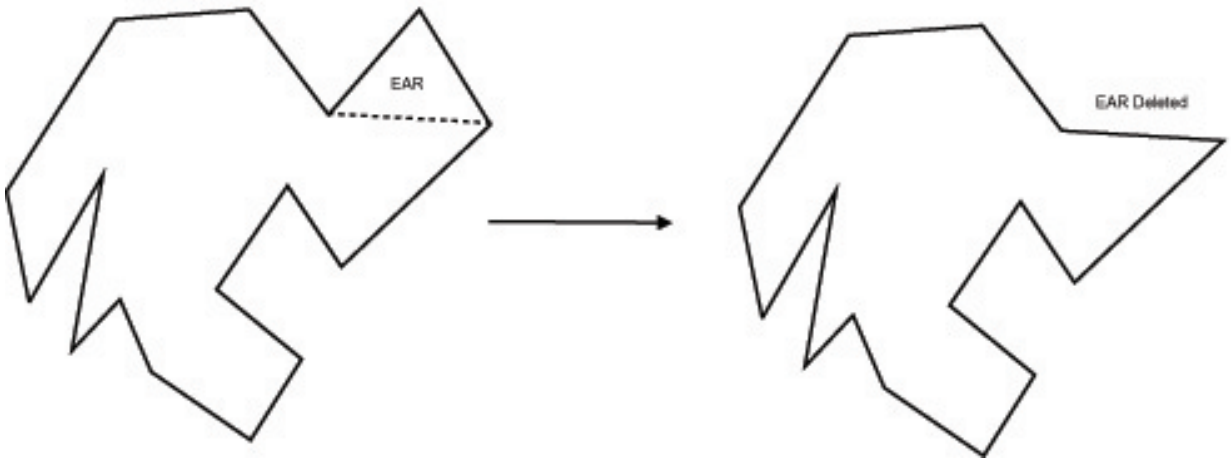

Figure 5: Ear cutting

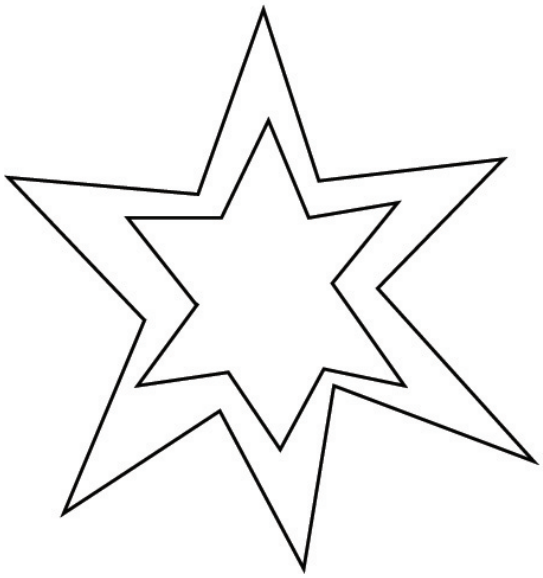

Figure 6: There is no ear in this case with a hole

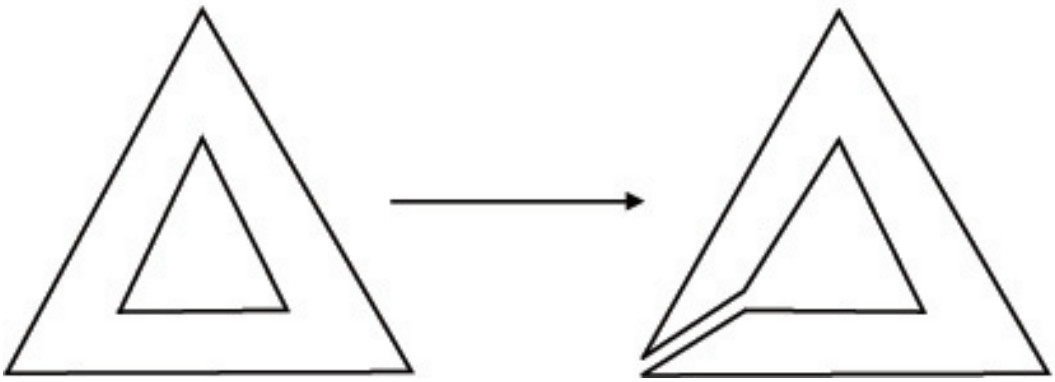

Figure 7: Holes can be removed by inserting a pseudo bridge

Question: Can we use the formula between faces, edges and vertices, which workes only for simply connected meshes?

We can, but determining the number of faces will be difficult unless we have the connectivity information.

Another way to determine if the loops are simply connected or not, would be to use our traversal graph in a slightly modified manner. We now start with any edge and traverse around as before. But for the other edges we start traversal only if the edge is marked on exactly one side (i.e. Forward XOR Backward). If after this process there are unpainted edges, it will mean that the regions are not simply connected.