CS-7491-A: 3D Complexity Techniques for Graphics, Modeling, and Animation
Lecture 24

Nipun Kwatra

1 Representing Shells around surface

Given a curve, we want to represent a shell of some thickness around it (figure 1). We can use normal maps for representing such shells, with the details represented by an elevation field. One problem with constructing such shells is shown in the image – the shell may self intersect where two regions of the curve pass close to each other. However, this problem is usually acceptable.

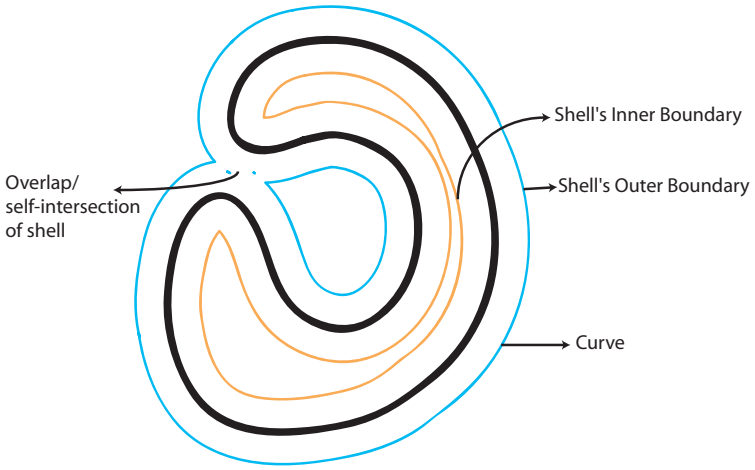

Figure 1: A curve and its shell

To fill the shell we can triangulate the area inside the shell. The main problem is how to construct these triangles. We can also pose the same problem for 3D. Here we will have a surface, and we will have to represent a 3D shell around it. Such a shell can be represented by tetrahedras.

An example of constructing these triangles using triangle strips is shown in figure 2.

However the triangulation may not be that trivial when the curvature of the curve is really large. As shown in figure 3, when there is a high curvature the triangulation may be fan-like in which all the trianles are coming out of a single vertex. We need to identify such cases and triangulate accordingly.

1.1 Scheme for Triangulate shell from curve

The student groups suggested different methods for triangulating the shell around a curve:

1.1.1 Method 1 - Intersecting circles:

This method involves the following steps and is illustrated in figure 4:

1. Distribute centers around the curve and draw circles.
2. Compute intersection between the consecutive circles.

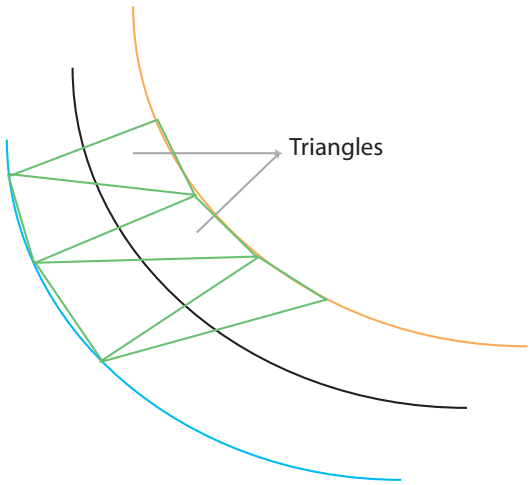

Figure 2: Triangulation using triangle strips

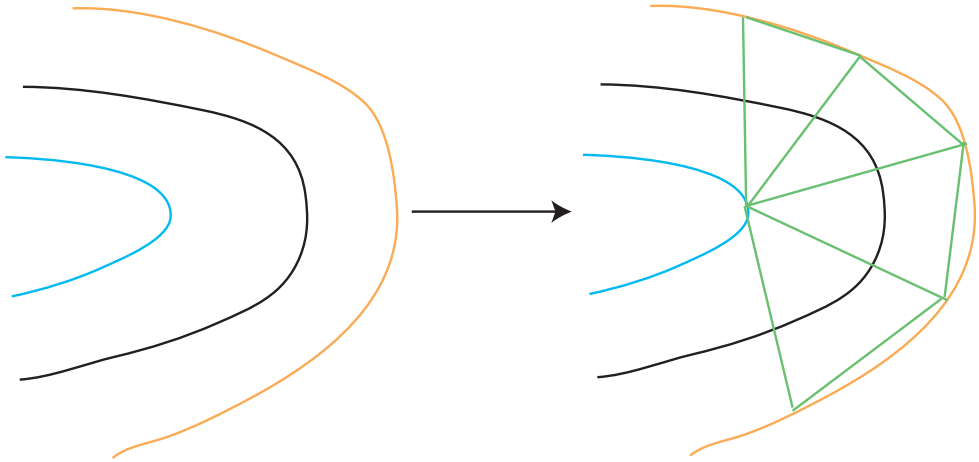

Figure 3: Triangulation using triangle strips

3. Push these intersections on a stack.
4. Check if the point on top of the stack is inside the next circle and if it does, mark it.
5. Replace all these *consecutive marked* intersections with an average point or one of the intersection point.
6. Finally triangulation can be done systematically by using the consecutive points on the stack.

This method will handle the high curvature case shown in figure 3. All conflicting points will be moved to one, and we will get the fan-like triangulation. Also not that this method will allow self-intersections of shell, which we don't mind.

1.1.2 Method 2 - Normal offsetting and triangulation :

This method involves the following steps and is illustrated in figure 5:

1. Distribute points uniformly around the curve
2. For each point create two points by offsetting by a fixed distance along the normal direction of the curve.

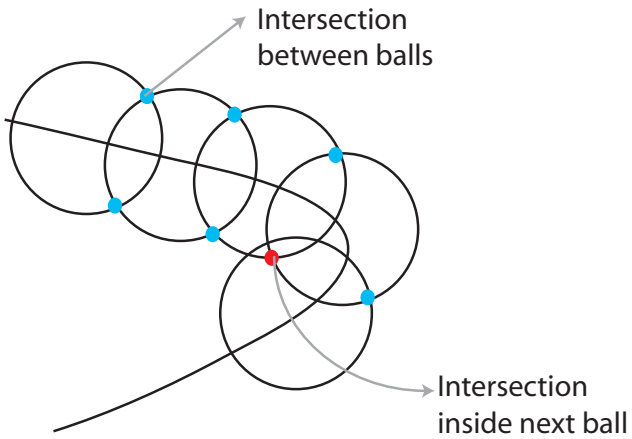

Figure 4: Intersecting circles method

3. Triangulate systematically as shown in figure 5

As shown in the figure, this method will fail near high curvature regions. But it can be handled similarly as in the previous method, by identifying offset points which lie in the next triangle and replacing them by a single point.

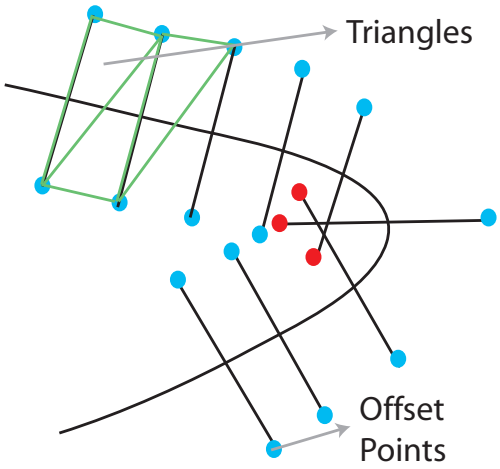

Figure 5: The blue dots show the offset points. The red dots will cause overlapping triangles and must be replaced be a single point. The systemic triangulation is shown in green.

1.1.3 Method 3 - Distance Field

One can also construct a distance field (on some grid, say) around the curve. Pairs of polygonal iso-surfaces in and out of the curve will form a shell around the surface (figure 6).

The problem now boils down to triangulating these polygonal regions. One way to triangulate would be to superimpose a grid on the region and triangulate each cell. Then we can use edge-collapsing techniques to simplify the triangular mesh. Another method would be to use Delaunay triangulation. However as shown in figure 7 Delaunay triangulation may create unwanted triangles like the one shown in red. So we would have to do some kind of *constrained* Delaunay triangulation in which we could put a constraint that the edges corresponding to the boundary of the region are always there in the triangulation.

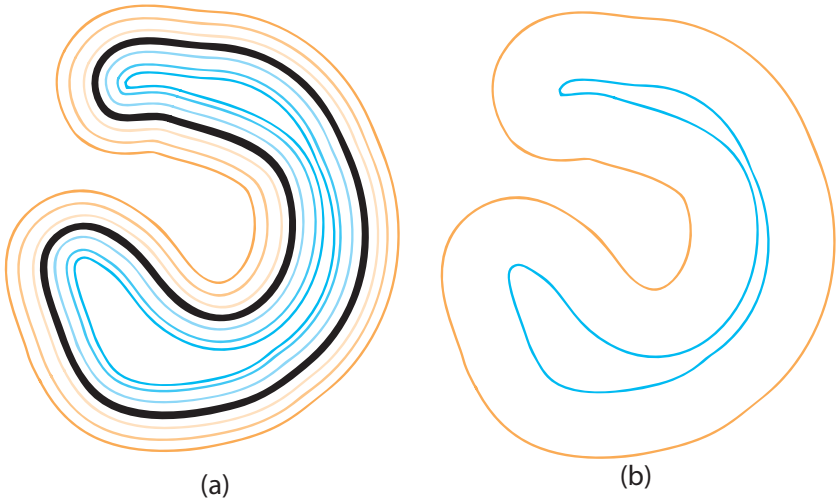

Figure 6: (a) The figure shows iso-surfaces shown in different colors. (b) Polygonal region which needs to be triangulated.

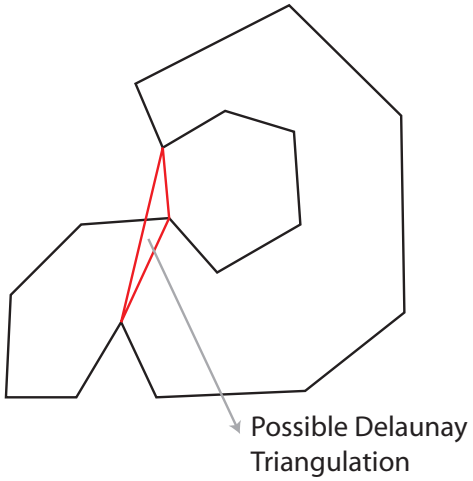

Figure 7: Delaunay triangulation may created unwanted triangles like the one shown in red.

1.1.4 Method 4 - Using ideas from Animation

One could also use ideas from animation literature to solve this problem. We can treat triangles as objects and use collision detection and resolution to avoid intersections near high curvature regions.

1.1.5 Method 5 - Topology preserving evolution

Another method would be to evolve the curve farther and farther from the original position, but prevent any topology changes. To do this we can triangulate a discrete grid around the curve and mark the vertices with its distance from the original curve (figure 8). Then we can use the transformations used in figure 9 to evolve the curve, but make sure to preserve the topology during the process.

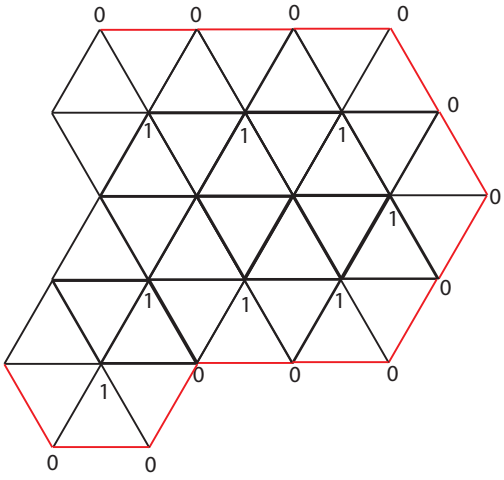

Figure 8: Triangulated grid with vertices marked with distance from curve.

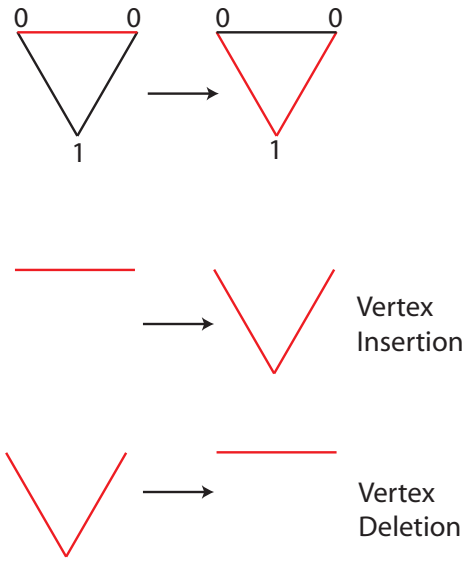

Figure 9: Transformations for evolving the curve.