CS 465 Homework 6

(revised December 4, 2003)

out: Tuesday 2 December 2003
due: Monday 8 December 2003

For this assignment you'll implement cubic Bézier splines in the context of a spline editor. Your final program will let the user edit a curve made out of Bézier spline segments and build a surface of revolution from it. Your job is just to implement the spline class; the user interface and drawing are handled by the framework.

This assignment is to be done individually.

Principles of operation

This program is based on cubic Bézier splines, as discussed in the lecture slides and in Shirley. The spline editor maintains a list of spline segments and displays them along with the control polygon and handles at the control points. The user can grab these handles and move them to edit the spline, and the editor enforces appropriate continuity conditions between adjacent segments.

All operations with the spline are based on adaptively spaced lists of points, which are generated by recursively splitting the spline. There are no explicit polynomial evaluations anywhere in the code! The fundamental spline operation, on which all others are built, is splitting a segment into two parts.

The surface of revolution is stored as an indexed triangle mesh, as discussed in the lecture slides. The spline class is responsible for computing the vertex positions and normals and the triangle indices.

Requirements

You must implement the methods `divide`, `getPoints`, and `getMesh` of the class `BezierSegment`.

The `divide` method returns a pair of spline segments, which together should describe exactly the same curve as the original segment. The junction between the two segments

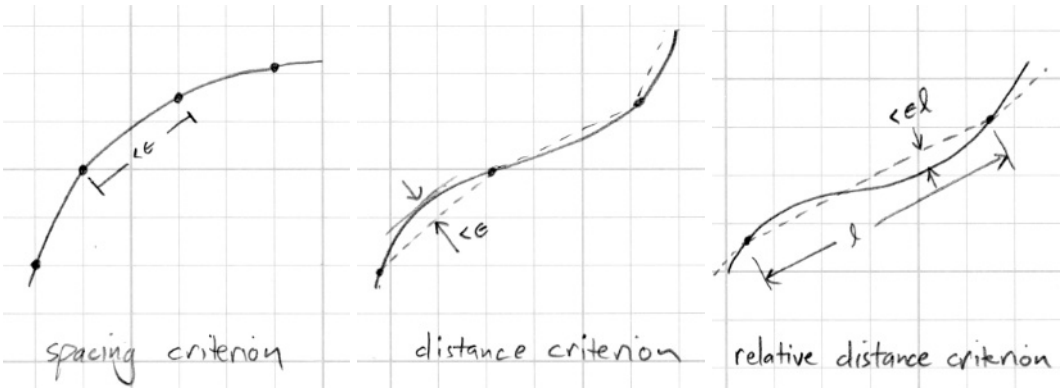


Figure 1: The three subdivision criteria.

should occur at the specified t value along the original segment. See Shirley 13.2 and 13.3. You can tell if it works by using the framework to split a segment: if the curve does not move and a new junction is introduced exactly where you clicked, it is working.

The `getPoints` method generates an ordered list of points that are on the curve, optionally along with their tangents and parameter values. It should do this using recursive subdivision, splitting using `divide` with $t = 0.5$ at each level. You should implement three different termination criteria (see figure), each using a tolerance ϵ that is given by the caller:

1. *Uniform spacing.* The recursion should terminate when the distance between adjacent points is less than ϵ. This option is used to generate a set of points to test when finding the closest point on the spline to where the user clicked.

2. *Absolute distance.* The recursion should terminate when the spline's convex hull property guarantees that the spline is within ϵ of the polygon defined by the points. This option is used to generate points to display the 2D spline.

3. *Relative distance.* The recursion should terminate when the spline's convex hull property guarantees that the segment of spline under consideration is within ϵl of the polygon defined by the points, where l is the distance from the first to the last endpoint. This option is used to tessellate the surface of revolution.

You should limit the recursion depth to 8, so that no more than 256 points are ever generated for one segment. Take care to ensure that each point is only output once, and that the spline goes all the way to the ends.

The `getMesh` method generates a triangle mesh for the surface of revolution. This is a

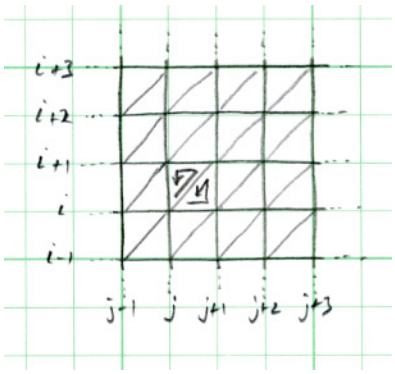


Figure 2: Diagram for surface triangulation.

parametric surface defined by:

$$s_x(u,v) = c_x(v)cos(2\pi u)$$
$$s_y(u,v) = c_y(v)$$
$$s_z(u,v) = -c_x(v)sin(2\pi u)$$

where (c_x, c_y) are the coordinates of the 2D spline curve. The mesh should be built so that the surface normals face outward and the surface is oriented outward when $c_x > 0$ and c_y is increasing. The surface should be tessellated finely enough to satisfy the relative distance criterion along the u and v directions.

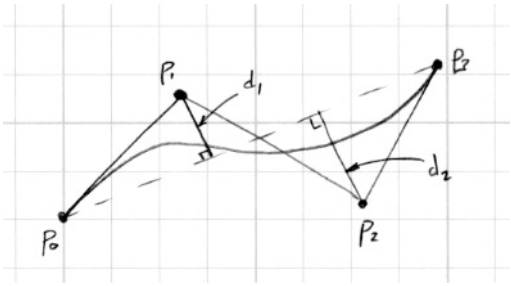
You should build your mesh as a grid that's regularly spaced in u and spaced according to the points generated by `getPoints` in v. The diagram in Figure 2 illustrates this idea. The vertices and triangles both come in 2D grids, so you'll need to lay them out in the 1D vertex and triangle arrays row by row, in the same way that you'd store a 2D image in a 1D array.

Framework code

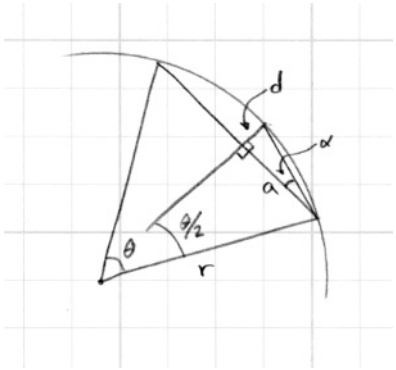
The framework has two main parts: the spline editor window, and the mesh viewer window. The mesh viewer, controlled by the `MeshView` class, is very similar to the viewer in Homework 2b.

The spline editor window shows you a 2D drawing area with your current spline (a piecewise linear one as the framwork ships). The following features are available: (But note that most of them will not work until you can generate points for the spline.)

- To move the control points, click and drag the handles.
- To convert a point from smooth (tangent continuity enforced) to corner (no tangent



(a)



(b)

Figure 3: Diagrams for preparation questions.

continuity) or back, click on it (without dragging). Note that corner points are drawn with special square handles.

- To split a segment, shift-click directly on the spline curve itself.
- To delete a junction, combining two segments into one, shift-click on a junction point. Shift-clicking on an end point deletes the end segment.

By hitting the space bar (you may need to click in the window once first, to ensure it has the keyboard focus) you can cycle through three debug modes: one that shows you the points used to draw the curve (from the absolute distance criterion), one that shows the set of points generated by the uniform spacing criterion (which are used to project to the spline when splitting a segment), and one that shows the points and tangents generated by the relative distance criterion (which you should use to build the mesh). The text display in the lower left indicates the current debug mode.

There are two sliders in the framework, which allow you to adjust the tolerances used to subdivide the curve and tesselate the surface. The left-hand slider adjusts the threshold for the distance and spacing criteria (the spacing threshold is always 5 times the distance threshold). The right-hand slider adjusts the threshold for the relative distance criterion, which controls the fineness of the surface tesselation in both u and v. Setting this tolerance to a small number results in a lot of triangles, which can slow things down.

Preparation questions

You do not need to turn these in, but you should answer them nonetheless, because they provide helpful formalizations of some problems you'll run into during implementation.

1. Looking at Figure 3a, write down expressions for d_1 and d_2.

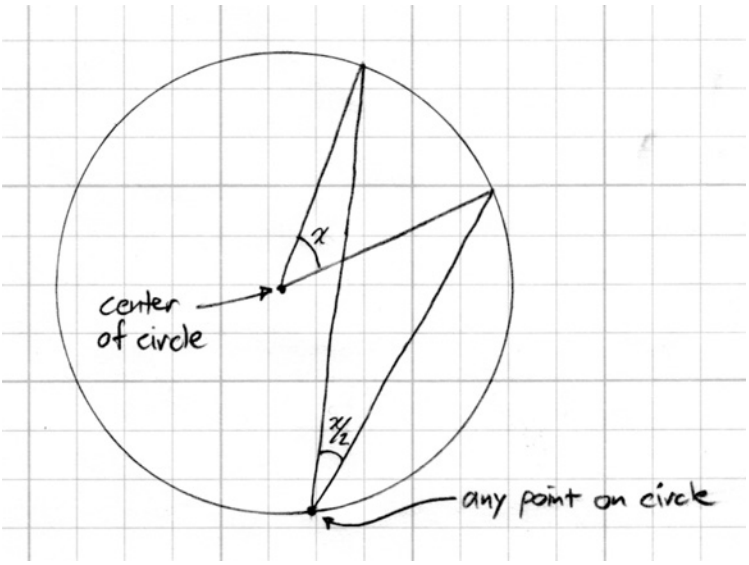


Figure 4: An angle based at a point on a circle is equal to half the arc subtended by that angle.

2. What does the convex hull property have to say about the relationship between d_1, d_2, and the absolute distance criterion defined above, if $\mathbf{p}_0$ and $\mathbf{p}_3$ are adjacent points in our list of points? What about the relative distance criterion?

3. Figure 3b concerns the error involved in approximating a circle with a polygon. Write an expression for d in terms of θ that is valid for small angles (that is, you can assume that $\sin\theta = \tan\theta = \theta$). Remember that an angle based at a point on a circle is equal to half the arc subtended by that angle. Figure 4 illustrates this.

4. Into how many segments does a circle need to be divided in order to satisfy the relative distance criterion for a given ϵ?

Handing in

Hand in in the usual way via CMS: a flat `.zip` file with the Java source files only—particularly, no `.class` files. If you did extra credit, include any additional data files that are needed.

Extra credit

For extra credit you could extend the program to handle more kinds of splines. Some examples:

1. non-cubic Bézier splines
2. B-splines

If you do extra credit, you will probably want to extend or modify the scenes we have provided, in order to demonstrate your extra feature.

We recommend talking to us about your proposed extra credit first so we can steer you towards interesting mappings and make sure we agree that it would be worth extra credit.

Let us re-emphasize that, as always, extra credit is only for programs that correctly implement the basic requirements.