# CS 4621 PPA2: Modeling and Texturing

out: Wednesday 2 November 2011
**due: Friday 18 November 2011**

## 1 Introduction

There are two parts to this assignment. In the first part, you will generate a 2D B-spline based on several control points given by a 2D curve editor. In the second part, you will create a 3D surface of revolution using the generated curve with proper normal and texture coordinates for each vertex of the surface.

We have implemented a 2D curve editor for you with the following functionality:

- move a control point;
- select a control point;
- add a new control point;
- delete the selected control point;
- reset the curve to the start configuration;
- rebuild the mesh of the surface of revolution;
- change the epsilon that determines the termination of the curve rendering subdivision.

Your job is to implement:

1. rendering of the 2D spline curve by first converting from Bspline control points to equivalent Bezier curves as described in lecture, and then rendering the created Bezier curve(s) using de Castlejau's algorithm;
2. creating the geometry of the corresponding surface of revolution with correct normals and texture coordinates.

## 2 The Curve Editor

When you first start the program you will see a panel, on the left, where you can edit the 2D curve. As you drag points around you are changing the points of the spline. Always assume that you have a minimum of 5 points.

Your job is to update the spline curve as you drag the points around. A slider on the left is the epsilon or tolerance slider for the 2D spline curve. For lower values of epsilon you will implement a deeper subdivision of the de Castlejau rendering algorithm.

You must create a 3D surface of revolution that corresponds to the spline curve; the axis of rotation corresponds to the vertical line on the left edge of the 2D window. The 3D surface should be shown in the right panel. As the 2D curve is edited you have to update both the curve and the surface of revolution.

Initially, the associated 3D surface of revolution is a rounded cylinder (or capsule) for 5 control points. However, this surface will change as the user manipulates/adds/deletes points on the spline curve. As the user edits the spline curve the associated surface of revolution should automatically be updated. If your program slows down, you can uncheck the “interactive” button. In this case the surface will only be updated when the user selects the “rebuild mesh” option from the “Action” menu.

## 2.1 Editing the spline curve

Details about the functionality provided by the 2D curve editor are described below.

**Moving Control Points:** The user is able to move control points using the spline editor. This can be done by clicking on a control point and dragging the mouse. In addition, the first and last control point are “anchored” to the top and bottom edge of the panel, respectively.

As the user moves a point, the surface of revolution will be interactively updated to reflect the new spline curve (unless the “Interactive” box is unchecked).

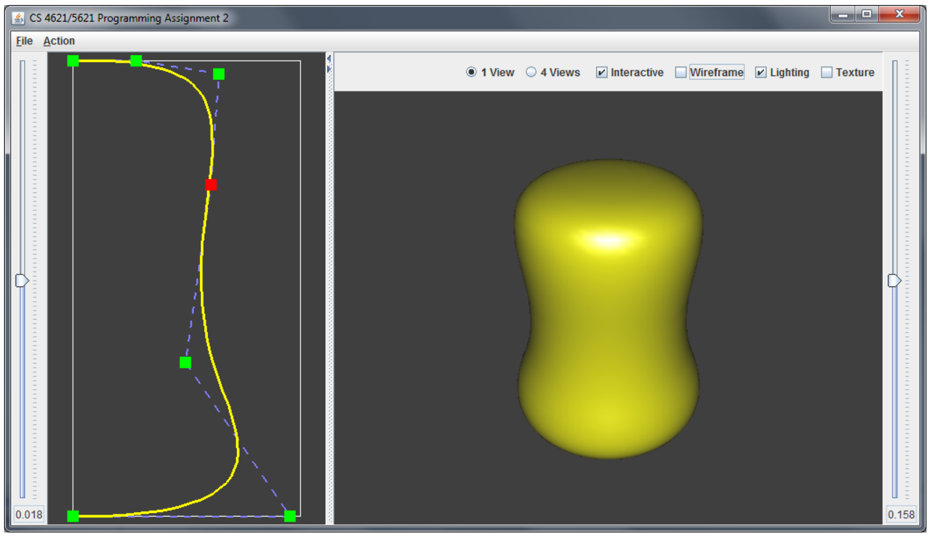
**Selecting Control Points:** When you click on a point, that point is set to be “selected” (and will be shown in red). You will see why this is useful when you add control points.

**Adding Control Points:** When the user selects the “Add Control Point” button in the “Action” menu, a new control point will be inserted into the list of control points after the selected one (defined above). The 2D location of the new control point will be at the midpoint of the selected control point and the next one in the sequence. If the last control point or no point is selected, the editor does nothing.

Originally, there are 5 control points. Since points can only be added after the selected one, the first point in the original set of 5 control points will always be the first point in the spline curve. Similarly, the last point in that set will always be the last point of the spline curve.

**Deleting Control Points:** When the user selects this button from the menu, the selected control point will be removed. If no point is selected, the editor does nothing. In addition, the first and last control points cannot be removed.

Every time control points are added/deleted/moved, the spline curve and surface of revolution should be updated appropriately.



[✓ 1 View] 4 Views [✓ Interactive] Wireframe [✓ Lighting] Texture

0.018

0.158

Figure 1: A reference screenshot. The red point has been “selected”.

**Reset:** This option resets the curve and the surface of revolution to the original configuration.

**Rebuild Mesh:** Updates the 3D mesh to correspond to the current spline curve. This is needed if the “Interactive” option is unchecked.

## 2.2 Surface of revolution

The surface of revolution is drawn in the right panel and is updated as the user edits the curve. The surface of revolution is created as follows: the spline curve is subdivided using the de Castlejau algorithm to produce the points that represent the curve. The epsilon setting determines the level of subdivision required. Each of the generated points is then rotated based on the tolerance value (as used in PPA1) to create the surface of revolution.

If you have a high epsilon setting, updating the mesh might be slow. In that case, you can uncheck the “Interactive” button to prevent interactive updates. To update the mesh you will then have to use the pulldown menu.

Figure 1 shows an example of the surface of revolution that could be produced by your program.

**Normals.** You will have to use the analytical normals of the curve (as described in lecture) when defining your polygons in the surface of revolution. We will evaluate the correctness of your normals by the correctness of shading and texturing, so make sure lighting works for the assignment.

**Texture Mapping.** Besides the normals, you will also have to assign the 2D texture coordinates for the vertices of the surface of revolution.

As shown in Figure 2, for a vertex $v$ with $t = t_0$ on the curve and rotated by $\phi$ around the axis of symmetry, the texture coordinates should be $(\frac{\phi}{2\pi}, t_0)$.

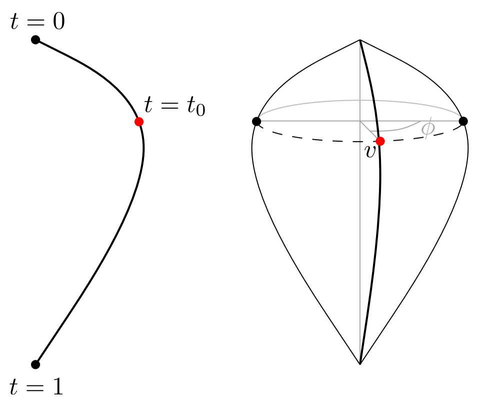


Figure 2: Assigning texture coordinates.

## 3 Resources

Design Mentor is a great program that shows some very nice visualizations of De Castlejau subdivision in the context of a curve editor and surface editor. It can be downloaded (with documentation) from http://www.cs.mtu.edu/ shene/COURSES/cs3621/LAB/curve/curve.html. We strongly urge you to try it out. Run curve.exe to play with the curve editor. Create a curve. Turn the DeCastlejau option on in the right. Change your parameter value by using the vertical slider. You will see all the intermediate points in the DeCastlejau algorithm interactively.

## 4 Implementation

You need to implement two classes: `BSpline` and `RevolutionVolume`. Both these classes belong to the package `cs4621.ppa2.shape`. The following is an overview of the required implementations.

- `DiscreteCurve` is an abstract class representing discretized curves in 2D. It has three fields:
  - `int nvertices`
  - `float[] vertices`
  - `float[] normals`

  `nvertices` indicates the total number of vertices; `vertices` and `normals` store the location and associated normal direction of every vertex in 2D, respectively. The $i$-th vertex lies at $(\mathtt{vertices}[2i], \mathtt{vertices}[2i+1])$ with the normal direction $(\mathtt{normals}[2i], \mathtt{normals}[2i+1])$ for $0 \leq i < \mathtt{nvertices}$.

  `DiscreteCurve` also contains an abstract method

  ```
  void build(float[] controlPoints, float epsilon)
  ```

  which takes an array of control points (in 2D) with an additional parameter `epsilon` controlling the level of subdivision, and constructs all three fields accordingly. The control points are given in a **clockwise** order.

- `BSpline` is a subclass of `DiscreteCurve`. You need to implement the abstract method `build` based on the technique described in class.
- `RevolutionVolume` is a subclass of `TriangleMesh`. Every revolution volume is associated with a discretized curve which is stored in its `curve` field. The `buildMesh` method constructs a 3D triangle mesh by rotating the 2D curve around the $Y$-axis. Besides the position and the normal, you also need to specify the texture coordinates $(\frac{\phi}{2\pi}, t)$ for each vertex where $t$ indicates the vertex's position on the curve, and $\phi$ is the angle by which the vertex has been rotated.