CS 184: Assignment 4 — Bezier and B-Spline Curves

Ravi Ramamoorthi

Goals and Motivation

The purpose of this assignment is to do a mini-project on modeling with spline curves. *This homework is to be done individually.*

You will use the de Casteljau Algorithm, and variations of it, to draw Bezier curves and B-Splines. You will also be asked to implement a drawing of Bezier curves using recursion. For the most part, this assignment only asks you to do a 2D curve editor (and this assignment uses only no-op shaders, with most drawing done using old-style OpenGL). However, 2D curves can be used as elements of a 3D modeling package, for instance, to define surfaces of revolution. Thus, a second part of the assignment (mostly for extra credit) requests you to implement a simple modeling program in that direction.

Since some of the topics are advanced, the most helpful information will be found in the lecture material, and in the related handouts.

Post any questions you have to the newsgroup, since other students will want to see the answers too. Please use detailed subject headings. However, do not post anything resembling code.

Getting Started and Submission Instructions

START EARLY. It is very unlikely you will be able to do this homework in 1 or 2 days in any case, and most certainly not if you want to also write the 3D modeling package.

Download the skeleton code for the assignment (we will provide a link to the zip file on the class website). Unzip the assignment into its own directory. You can run *hw4_solution.exe* to see how your program should behave (we also include an executable for Linux, in addition to the visual Studio support).

The example solution is provided for reference as an executable. Make sure your code behaves identically to the solution (checking that detail levels match). If your code works except for slightly different behavior at level 1, don't worry about it. (Though it may be an indication of other errors).

Your job is simply to fill in the sections of *curves2.cpp* that say /* YOUR CODE HERE */
WorkingScene should be filled in first.

For the basic part of the assignment (the 2D curve drawing), *do not use any OpenGL calls or external libraries*. The one exception to the OpenGL calls rule above, is where it says "make sure the scene gets redrawn". The correct line to complete this operation is *glutPostRedisplay() ;*. Also, do not modify any files other than *curves2.cpp*.

For submission of the basic assignment, submit the file *curves2.cpp*, just like homework 1. That is, run "submit as4"from a directory containing the following:

- *curves2.cpp* (please also write your name as a comment in the file).
- *futurepartner.txt* This assignment is to be done individually. However, your submission should include your partner for assignment 5. The purpose of this is to get you to start thinking about assignment 5 soon; the first milestone on that will be due soon.

The implementation and submission issues for the (mostly extra credit) 3D modeling part are discussed separately, and should be submitted separately.

Basic Assignment Specifications

You will want to complete *WorkingScene* before anything else. Once you have done this correctly, you will be able to draw regular *Curve*s (just straight lines) by LEFT clicking on the screen to add points. You should be able to delete points by RIGHT clicking in them. You can also drag existing points around.

Thereafter, you will fill in code for *Bezier*, *Bspline* and *Bezier2*. For *Bezier*, you simply divide the curve into line segments (depending on the *detail* parameter, there should be *detail* segments). Hence, all you need to do is evaluate the curve at *detail+1* points, connecting these with line segments. The evaluation can be done by the deCasteljau algorithm as described in class, or you can use the explicit Bernstein-Bezier polynomial form. For *Bezier2*, you draw the curve by recursive subdivision, splitting it at its midpoint each time. The recursive subdivision of Bezier curves using the deCasteljau algorithm was discussed in class. Finally, for drawing cubic *Bspline*s, you can either use a variant of the deCasteljau algorithm, or the B-spline matrix formula discussed in class directly. This latter formula applies since the knot spacing is uniform and the B-splines are always cubic.

Notes for Skeleton Code: Please note that the program (even solution) will not do anything if you click on control points after startup. You first need to enter the type of curve, with 0 for a basic straight line. You should read the code to figure out how to switch to Bezier and Bspline curves. Also the + and - keys are used to change the *detail* parameter.

The assignment skeleton code uses the *vector* class from the Standard Template Library (STL). It is a very powerful class, but may have a slight learning curve. It is worth investing the time to learn how to use this. Microsoft's MSDN web site is a good source of information. There are also plenty of books and other web sites that discuss STL. If you really don't want to use *vector*, figure out just enough to convert the vector of points into a form you are comfortable with before you start your manipulations of the points.

For *Bezier2*, you are provided with a more complete skeleton. This code uses *vector*. You may choose to rewrite *draw()* and not use the skeleton at all, but we reccomend that you study this code and fill in the provided structure. This will help you learn the *vector* class, as well as the recursive algorithm for Bezier curves.

FAQ: Can I write other helper functions instead of using the given functions when implementing Bezier2? You can change whatever you want in *curves2.cpp*, but please don't change anything outside of that. Of course, you can add whatever helper functions you need inside of *curves2.cpp*.

Rules for Your Code: *Do not use outside (any) libraries for your computations.*

Helpful functions provided in the skeleton are:

- The class *Curve* has the function *drawLine( )*, which you should use to draw straight lines.
- The class *Point* has a function *draw( )*. For drawing knots in the B-Spline, you will want to create a point, and ask it to draw itself.

Hints and Dcoumentation: We briefly provide some hints and documentation that may be helpful.

- *Normalization*: You will likely want to normalize the coordinates returned by x and y for example, by dividing by the window width. This is true for both *WorkingScene::drag* and *WorkingScene::mouse*.
- *Curve Functions:* *WorkingScene* will benefit from the use of a number of functions of the *Curve* class (look at *Curve.cpp*). In particular, *moveActivePoint* (useful for drag) simply updates the current active point.
- *Adding and Deleting Points:* *WorkingScene::mouse* will require code for adding and deleting points. A number of functions in the *Curve* class are helpful here. *addPoint* adds a point with specified x and y coordinates. *updateActivePoint* takes the x and y coordinates, picking the relevant point within some radius (it can thus be used for selection). *deleteActivePoint* deletes the currently active point. These functions are likely to be quite useful to you.

Advanced Assignment and Extra Credit: 3D Modeling System

Warning: This part of the assignment should be attempted only after you have completed the basic assignment correctly. It should be submitted entirely separately, as *as4extra*. You still need to separately submit the basic assignment as noted above. Moreover, you can treat this part of the homework as optional.

Don't feel disappointed if you don't do it, or only do very little. You can still get almost full credit for this homework, since the advanced assignment will contribute mainly only to extra credit.

The advanced component will count only for a very small fraction of the total regular credit points for the homework, with extra credit being provided as needed, based on the quality of the demo and results.

Motivation and Specification: The idea here is to use the spline curves to implement a basic 3D modeling system. Since this is primarily for extra credit, it is specified and supported in much less detail than the main assignment.

The first thing to start out with is to use the spline curves to define a surface of revolution. This is a simple method that will allow you to define and draw interesting shapes. You will have to resolve details like how to tessellate the surfaces you output, and it would be nice to be able to change lighting and view on them, using much the same codebase as assignment 2. Ideally, you'd have two windows, one your standard curve editor, and the other the surface defined as a result. The second window would allow transformations or at least an interface analogous to assignments 1 and 2.

Beyond surfaces of revolution, you can define a number of other interesting shapes from curves, such as sweeps, extrusion, lofting and so on. You may want to look these up, and also take a look at a modeling program like Maya, to see if you can reproduce some of its curve modeling functionality. It is also possible to simply write your own code to build arbitrary generative models, with curves at the lowest level (see for example the curve models in "Creating Generative Models from Range Data" by Ramamoorthi and Arvo in SIGGRAPH 99). Note that more interesting shapes will likely require more than one curve at a time.

The restrictions on outside libraries/OpenGL calls and modifying the basic program framework or adding extra files no longer apply (within reason: outside libraries can be used only for basic routine functions, not for core functionality. If in doubt, ask).

Submission: Note that you must still submit *curves2.cpp* for the basic assignment though. *In addition*, your submission to the advanced assignment (a separate "submit as4extra") will include either your full codebase, or a zip thereof, including a README file and demo examples where appropriate. Some points (extra credit) may be given for good documentation. In addition, you should create a website, either on your class account or elsewhere, that briefly documents what you did, and has screenshots of your program in action, as well as some interesting 3D shapes you were able to construct with it. In your submission README file, include a link to this website, and do not modify the website after the due date. Extra credit will be given if you include on the webpage, movies showing your system for editing in action. Otherwise, we may request demos to see the interactive nature of 3D shapes you can create.