CS3388 - Assignment 3, 2016

Posted: 8th November 2016
Due: 23rd November 2016, 12:55 PM

Description

This assignment is about implementing a curve editor using OpenGL. You are provided with C++ source code which uses OpenGL to draw points on the screen by mouse click. You have to write the code to draw Bézier curve, B-spline and Beta-spline curves using these points. The assignment is worth **10%** of the final mark.

Unzip the files, compile and run. At this point you will see a drawing screen where you can click mouse and draw points. This is a simple user interface for manipulating curves. You should be able to create and move the control points by clicking and dragging with the mouse. This functionality is provided by the *PointCanvas* class. Right-clicking brings up a menu comprising of different types of curves that you have to implement: basic Bézier curve, cubic B-spline curve, Beta-spline curve, closed Beta-spline curve and cubic B-spline with phantom endpoints. Finally, last 2 menu items are for clearing the screen and exiting from the program, which are already implemented. You will need to write code at the specified places in *curves.cpp* (and *curves.h* if you add any new function).

Following are the details of the tasks for the assignment:

- **Bézier curve evaluation:** Implement simple evaluation and rendering of Bézier curve. You are provided with an incomplete *curveOperations* class in *curves.cpp*. Implement *curveOperations::evaluateBezier* to return the point along the Bézier curve at the given t-value. A function *curveOperations::binomialCoefficient()* is provided to compute $\binom{m}{i} = \frac{m!}{i!(m-i)!}$.

  Next, implement *curveOperations::drawBezier*. Draw the curve by evaluating it at uniformly-spaced t-values, and then drawing lines between those points. We have provided a function *curveOperations::drawLine()* to do the line-drawing for you. It is up to you to choose how many different points to evaluate in order to produce a reasonable-looking curve ($\Delta t = 0.01$ produces pretty good result).

- **Draw cubic B-spline curve:** Implement *curveOperations::drawCubicBspline* to draw a cubic B-spline curve from the clicked control points. Follow the course notes to get the details of cubic B-spline. There are different notations that are used, feel free to use any idea. However, the notation used in slide 43, 44 are pretty straightforward:

$$P(u) = \sum_{i=nu-2}^{nu+2} P_i B(nu - i)$$

Use the range of $(nu-2)$ from $floor(nu-2)$ to $ceil(nu+2)$. This notation will help you to implement rest of the parts of the assignment without any ambiguities from the course notes. Use *curveOperations::drawLine()* to draw the curve after generating the points.

- **Draw Beta-spline curve:** Implement *curveOperations::drawBetaspline* to draw a Beta-spline curve from the clicked control points. Follow the course notes, or this tutorial for detailed theories of Beta-spline. Use skew (s) = 1.0 and tension (t) = 10.0.
- **Draw closed Beta-spline curve:** Implement the function *curveOperations::drawBetasplineClosedCurve*. This is almost the same as the previous one, the only difference is that, you have to consider the endpoints in such a way that it draws a closed Beta-spline curve.
- **Draw cubic B-spline curve with phantom endpoints:** Implement *curveOperations::drawCubicBsplinePhantom*. In cubic B-spline, the curve does not pass through the start and end points. That's why we add two "phantom" nodes to force the curve to pass through the endpoints. These are denoted by P_{-1} and P_{n+1} in the course notes. Consider these two phantom points and draw the cubic B-spline. Now it should pass through the endpoints.
- **Write a short report:** Experiment with different values of skew and tension parameters of Beta-spline and write a report (less than one page) on your observations on the comparison of the curve behaviour on these parameters, and also compare with cubic B-spline curve.

You should not need to modify any files except for *curves.cpp*. However, feel free if you want to add any function to *curveOperations* class (say a separate method for blending function), and obviously you will need to modify *curves.h*. DO NOT change the names of the provided files. You are welcome to modify anything you like, but you must preserve all the functionalities of the assignment.

What to submit?

Submit the program file(s) you have implemented/modified and the report either in a text or PDF file. Put all the file(s) into a zip and submit via OWL (no files will be accepted by email). Please don't submit any unnecessary files (such as the whole project).

Late penalty

The late policy is a penalty of 5% per day up to 3 days of lateness. Saturday and Sunday count as one penalty day.

Plagiarism

Copying the code is a serious academic offence, which will be treated as per university norms. Remind that changing variable names and white spaces do not make your code unique, it's very easy to detect these cases using softwares.

General marking scheme

The marks will be distributed as follows:

- Working program: 70%
  - (will be divided into different parts of the assignment)
- Documentation: 10%
  - Main comment block identifying the student (name, student number, email address): 4%
  - Defining input and output parameters for a function: 3%
  - Purpose of functions/blocks of code: 3%
- Program style: 10%
  - Meaningful variable names: 3%
  - Constants instead "magic numbers": 2%
  - Readability (complete sentences, indentation, white spaces, etc): 2%
  - Code flows "nicely": 3%
- Program structure: 10%
  - Modular code: 4%
  - Uses appropriate data structure: 3%
  - Loops when needed/no loops when not needed: 3%