Assignment G2

Due Date: March 5

Purpose

In this project, you will write an animation using splines of different order. To complete the program, you will need to code the solution to quadratic and cubic splines, as well as use transformations (possibly), double-buffering, and other OpenGL features.

Problem

Five Flags[1], an amusement park company based in Norton, MA, has hired you to model a new ride. This ride is similar to a roller coaster, but will be virtual. The new twist is that the rider can create her own coaster. Before they embark on this multi-million dollar investment, they want you to write an animation that shows how the ride will work.

The ride will be similar to the kind of roller coaster where the rider sits in a car beneath the track. The coaster will go up and down virtual hills before returning to the starting point. The hills are defined in a data file via control points; in the real ride, the rider will be able to "create" the hills. Thus, this data file is a simulation. In any event, the data file defines points that you will connect using a piecewise spline, either a linear (C^0), quadratic (C^1) or cubic (C^2). You will then draw these splines using only the `glVertex()` function along with GL_POINTS, GL_LINES, or GL_LINE_STRIP. You are free to use other OpenGL functions, such as `glColor()` and functions for text, polygons for the car, etc., **but** do not use other OpenGL functions to draw any part of your splines.

Input

Coordinates for the knots can be input two ways:

1. Data will be stored in a file specified by the user on the command line (what fun – you get to use those weirdo `argc` and `argv` thingies!), up to 20 characters long. For example, if your executable file is called `schmoeG2` and the input file is called `myInput.txt`, you would run the program on the command line by typing: `./schmoeG2 myInput.txt`

   The file will contain **any number of lines** each with the following format:

   $$n\ x_1\ y_1\ ...\ x_n\ y_n$$

   This defines n knots (control points) for a spline, where $2 \leq n \leq 10$. Knot values will fall between 0 and 1000 in x and y. The first and knot will be at (0, 0); successive knot x's should be in ascending (positive) order, and be defined properly by the user. The last knot will be at (1000, 0).

   The last line of the file contains just a 0, indicating there are no more points.

   Be sure that your program reads this file properly!

2. Data points can be picked simply by using the mouse. As above, successive knots should be in ascending order in x. If not, an error should be displayed and the program should stop. You must decide how the last knot in the set of points will be defined.

   The mouse input is optional. The correct implementation of mouse input will give you bonus points for the project.

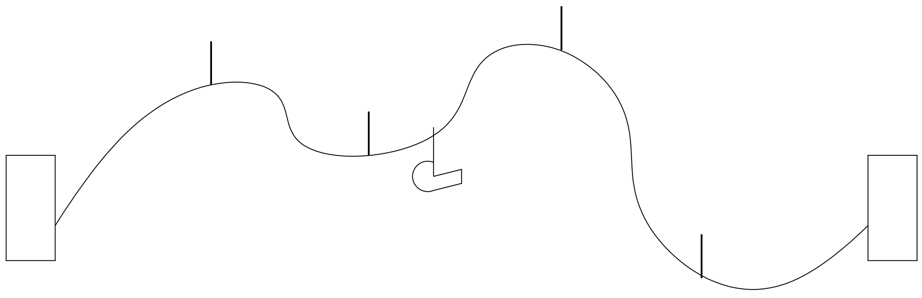
[1] Not nearly as good as Six Flags.

The program should also accept keyboard input:

- '0': Display a linear (C^0) piecewise spline
- '1': Display a quadratic (C^1) piecewise spline
- '2': Display a cubic (C^2) piecewise spline (if implemented)
- '4': Go slower
- '6': Go faster
- '+': Zoom in (things get bigger)
- '-': Zoom out (things get smaller)
- 'm': Choose new data points with mouse (if implemented)
- 'n': Display the next set of knots in file (if no more data, display a message in console)
- 'r': Reposition the car at the start of the current track
- 's': Start (or restart) the car running along the track
- 'p': Pause the car
- 'q': Quit

Output

The output should be an animation of a "car" hanging from and moving along the spline from left to right. Each time a new spline is generated, the car should have a default starting speed. If the user changes the speed, the new speed takes immediate effect and the ride continues at that speed until the user changes it or a new (different) spline is drawn. The car should stop when it reaches the last knot. The left and right ends should have a start and end "station." The knots (except the first and last) should be depicted by "hangers" or some icon that shows their location. Here is an (awful) version of the output with a version[2] of a C^1 spline:



A C^0 piecewise spline is created simply by joining line segments at the knots. You can draw these lines with OpenGL; however, you will still need individual coordinates along the line so that you can draw the car moving along the wire.

[2]This is not the way we are computing them!

In the case of a first- or second-order spline, each piece (between two successive x values) should be defined by a parabola, except the first, which is a line. The initial line for C^1 and C^2 splines can be drawn with OpenGL. Note that implementing C^2 splines is optional.

One problem you will encounter is that as the spline becomes smoother, it usually becomes more elongated in the y direction as well, especially as the x values get closer together. To combat this, you will want to "scale" your window size in the y direction; that is, define a much larger distance in y than in x. That will have the effect of "squishing" the y coordinates a little, but will make the entire output fit on the screen better.

The output should include a second viewport to display legal keyboard input. This viewport should be a relatively narrow vertical rectangle to the right of the main viewport.

Specifics

- As stated previously, you may only use `glVertex()` to draw points. However, you will want to draw line segments using **real** values between your knots. Otherwise, you may get large spaces between each of your x points.
- Be sure to give adequate (but short) instructions/menu in the right viewport regarding all keyboard input. If you have not implemented an option, do not include it in the instructions.
- I will not look at your code, including comments.[3]

Notes

Tackle this project in phases! One possible sequence towards a solution is as follows:

1. Be sure you understand the data and what you're trying to accomplish! Be sure you can figure out the splines on paper before attempting to code anything.
2. Write a program that reads the input file properly. Test this! If you can't read the data, there is no use continuing.
3. Add code that can display a linear spline through the control points. Add in the start and end stations and the indicators for the location of the knots.
4. Add in the second viewport with the instructions/menu of options.
5. Add in the animation and (some of) the keyboard functionality.
6. Add in the C^1 splines. If you've written modular code, then the animation should still work. If not, modify the code so that the animation works. Incrementally add keyboard functionality.
7. Finally, add in the C^2 splines, if desired. This should be relatively easy if you understood what you did in the previous step.
8. If there's time (and the will), add in the mouse input.

[3]The crowd erupts in applause!

Grading: You will earn up to 90 points if you implement a **smooth** animation with splines up to C^1 continuity. Adding C^2 continuity will give you up to 8 additional points. Adding the mouse input will earn you up to an additional 8 points. Thus, you can get a score of up to 106 points. You can do either or both of these optional portions.

Turn in your source code as an email attachment. Name this file *your_last_name*G2.`cpp` as in `GousieG2.cpp`. Send this to me via email before midnight on the due date.

The most painful thing about mathematics is how far away
from being able to use it after you have learned it.
– James R. Newman