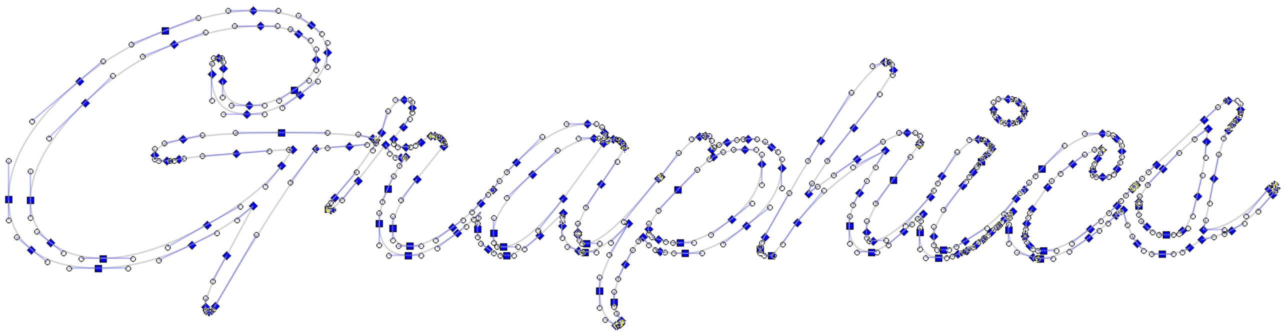
Curves & Splines

Assignment #3

CPSC 453 • Fall 2016 • University of Calgary



Overview & Objectives

The main objective of this third assignment in CPSC 453 is to learn to work with Bézier curves and splines. You will write a program to draw both quadratic and cubic Bézier curves in the context of what is likely to be their most prominent use in computer graphics, namely that of describing and rendering letterforms in computer fonts. You will also begin to explore real-time animated computer graphics by rendering text that scrolls across the screen.

A second objective of this assignment is to gain familiarity with the capabilities and programming of an additional shader stage in the OpenGL rendering pipeline. You will be writing OpenGL tessellation shaders to generate line segment primitives that approximate Bézier curves from their control points.

This assignment consists of a written component and a programming component. The programming part is described first, but **note that written answers to the assignment questions are due ahead of the program submission.**

Due Dates

Written component	**Thursday, October 20 at 11:59 PM**
Programming component	**Monday, October 31 at 11:59 PM**

Programming Component

There are a total of four parts to this programming assignment with an additional requirement to integrate the three scenes into a single application. This component is worth a total of 20 points, with the point distribution as indicated in each part, and an additional possibility of earning a "bonus" designation. Since the bonus is a binary state, it will only be awarded to submissions that do an exemplary job of meeting the requirements.

Part I: Bézier Curves (8 points)

Create a program, or modify the provided template, to draw a set of Bézier curves within an OpenGL window on the screen. You will want to create two distinct scenes for this part of the assignment: one that draws quadratic Bézier curves and another that draws cubic Béziers.

In your first scene, draw quadratic Bézier curves with the following sets of control points:

- (1, 1), (2, -1), (0, -1)
- (0, -1), (-2, -1), (-1, 1)
- (-1, 1), (0, 1), (1, 1)
- (1.2, 0.5), (2.5, 1.0), (1.3, -0.4)

In your second scene, draw cubic Bézier curves with the following sets of control points:

- (1, 1), (4, 0), (6, 2), (9, 1)
- (8, 2), (0, 8), (0, -2), (8, 4)
- (5, 3), (3, 2), (3, 3), (5, 2)
- (3, 2.2), (3.5, 2.7), (3.5, 3.3), (3, 3.8)
- (2.8, 3.5), (2.4, 3.8), (2.4, 3.2), (2.8, 3.5)

Scale your scenes (or equivalently, the control point coordinates) so that all the curves fit nicely within your window. Also draw the control points as coloured points on the screen and the control polygon (2 segments for quadratic, 3 for cubic) as line segments. Use different colours to distinguish between curve endpoints and off-curve control points. Choose colours for your curves and control polygons to give prominence to the curves themselves and de-emphasize the appearance of the polygon lines.

When drawing your curves, pass only the control point geometry through the vertex arrays, then write either *tessellation shaders* or *geometry shaders* to generate the curve geometry. Allow toggling between the two scenes in this part by use of keyboard input.

Notes & Hints:

- Recall that you already have a program from Assignment #1 that will draw a parametric curve on the screen. It may be helpful to use that same method for drawing these Bézier curves for initial testing and debugging. However, part of the purpose of this assignment is to learn to use tessellation or geometry shaders to generate primitives. **Submissions that do not make use of these shaders to create the curve will not receive full credit!**

- You may find that your scenes for this assignment will look better with multi-sampling (a form of anti-aliasing) turned on. Try `glfwWindowHint(GLFW_SAMPLES, 4)`. You can also use `glPointSize()` to create more contrast between control points and the segments.

Part II: Rendering Fonts (6 points)

Add a scene to your program that will draw your name on the screen using at least three different fonts. You may render your first name, last name, or whatever combination you like the best. Use two of the provided fonts, Lora and Source Sans Pro, and one or more fonts of your own choosing. You may use bold, black, or other variants depending on which you think looks best. Provide a means to interactively toggle between which of the fonts is used.

When setting up your scene, adjust the size of the letters so that your name nicely fills the window with each font. Ensure that no letters are clipped out of the window, and that the spacing between letters is appropriate.

You need not draw control points or polygons in this part. If you do wish to keep them, add a means to toggle whether or not the control points and polygons are shown. When they are hidden, you should see only the outlines of the letterforms.

Notes & Hints:

- We have provided some code for you that uses the FreeType library to extract spline control points from font files, but you must **use your own tessellation or geometry shader code to render the glyphs**. Naturally, only the glyph outlines will be drawn. You should not be using the font rendering features provided by FreeType in this assignment.
- TrueType fonts (.ttf) use only straight segments and quadratic Bézier curves, whereas OpenType fonts (.otf) use straight segments and cubic Bézier curves. Consider how this may help you set up your rendering code.
- Many great free fonts can be found on Font Squirrel: http://www.fontsquirrel.com

Part III: Scrolling Text (4 points)

Add one more scene to your program that displays an animated string of text, using three different fonts, that scrolls from right to left in your window. Use the following text:

The quick brown fox jumps over the lazy dog.

You may add to the above text if you'd like, or write whatever text you wish so long as every letter of the English alphabet is used at least once. After the last character in your text has scrolled off the left side of your window, repeat the text from the start.

Write your text using another two of the provided fonts, Alex Brush and Inconsolata, and one more different font of your own choosing. Provide a means to interactively toggle between which of the fonts is used. Set up your scene for each font so that the letterforms are nice and large, but one or two full words fit within the width of your window. Scroll the text at a speed that is comfortable to read. Finally, provide a means to interactively adjust the rate at which the text scrolls.

Notes & Hints:

- You will probably need to use `glfwPollEvents()` in your main loop, rather than the call to `glfwWaitEvents()` that was in the original template code, to run a smooth animation.

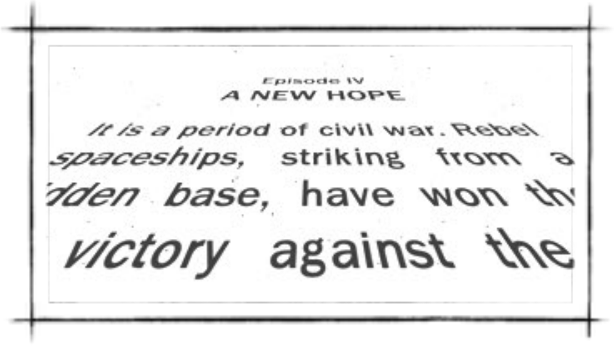
Integration & Control (2 points)

Use keyboard input to provide a means for switching between the three parts of this assignment. Ensure that your program does not produce rendering anomalies, or crash, when switching between scenes or fonts. Efficiency is important too! Programs that fail to clean up resources or generate excessive amounts of unnecessary geometry may bog down the system or eventually crash, and will not receive full credit.

Bonus Part: Advanced Rendering or Font Editing

There are two possibilities for earning a bonus in this assignment. Each is described in the paragraphs below. If you're so compelled, you may complete both to earn a "mega-bonus"!

Rendering solely the outlines of the glyphs is perhaps interesting and instructive, but not particularly practical for real applications. The first opportunity for the bonus is to devise or implement a method to render *filled* glyph outlines. Your method should ideally be efficient and support real-time rendering through the OpenGL pipeline, though we will still consider less efficient, but correct methods for bonus. The glyph rendering must be your own—you should not be using the FreeType features to rasterize glyph outlines. Once you achieve this, add another animated scene to your program to demonstrate your rendering. Be creative and make the letters spin, twirl, bounce, or dance to really showcase this feature. If you are having trouble thinking of an interesting effect, perhaps you'd just want to clone this effect, on the right,


Episode IV
A NEW HOPE
It is a period of civil war. Rebel
spaceships, striking from a
dden base, have won th
victory against the

from a certain major motion picture.

The second opportunity for a bonus in this assignment is to make the control points, and hence the outlines, editable. In essence, you will be creating a basic font editor. Provide a means for using mouse input to select and move control points, and update the Bézier spline outlines as the points are dragged. Also provide a means to add or remove control points. Once you achieve this, take a nice, thick marker and write the alphabet on a blank sheet of paper. Scan or photograph your "master" and use your code form Assignment #2 to load and render the image in your window, behind the curve outlines. Finally, use your program to trace these letters, one at a time, with spline outlines. At this point, you may want to create a means to save and load the outlines for each letter so that you don't lose all your work if your program misbehaves. If you made it all the way here, talk to the instructors about writing these outlines to a TrueType font file, and you will have a font of your own handwriting!

Submission

We encourage you to learn the course material by discussing concepts with your peers or studying other sources of information. However, all work you submit for this assignment must be your own, or explicitly provided to you for this assignment. *Submitting source code you did not author yourself is plagiarism!* If you wish to use other template or support code for this assignment, please obtain permission from the instructors first. Cite any sources of code you used to a large extent for inspiration, but did not copy, in completing this assignment.

Please upload your source file(s) to the appropriate drop box on the course Desire2Learn site, and indicate any late days used here. Include a "readme" file that briefly explains the keyboard controls for operating your program, the platform and compiler (OS and version) you built your submission on, and specific instructions for compiling your program if needed. If your program does not compile and run on the graphics lab computers in MS 239, *the onus is on you to ensure that your submission runs on your TA's grading environment for your platform!* Broken submissions will be returned for repair at a minimum penalty of one late day, and may not be accepted if the problem is severe. Ensure that you upload any supporting files (*e.g.* makefiles, shaders, data files) needed to compile and run your program, but please do not upload compiled binaries or object files. **Include extra font files that you used for this assignment**, or links to where we may be able to retrieve them.

Your program must also conform to the OpenGL 3.2+ Core Profile, meaning that you should not be using any functions deprecated in the OpenGL API, to receive credit for this part of the assignment. We highly recommend using the official OpenGL 4 reference pages as your definitive guide, located at: https://www.opengl.org/sdk/docs/man/.

Assignment Questions

The written component consists of three questions, each with several parts, corresponding to the parts of the programming assignment, and worth 10 points in total. When answering the questions, show enough of your work, or provide sufficient explanation, to convince the instructors that you arrived at your answer by means of your own.

Question 1: Bézier Curves (5 points)

A. Consider the quadratic Bézier curve formed by the three control points (1, 0), (1, 1), and (0, 1). What is the position of the point in the middle of this curve, *i.e.*, evaluated at parameter $u = 0.5$?

B. Now consider a quadratic Bézier spline consisting of four mirrored copies of the segment from 1A. The sequence of nine control points would then be (1, 0), (1, 1), (0, 1), (-1, 1), (-1, 0), (-1, -1), (0, -1), (1, -1), and (1, 0). What shape does this curve form?

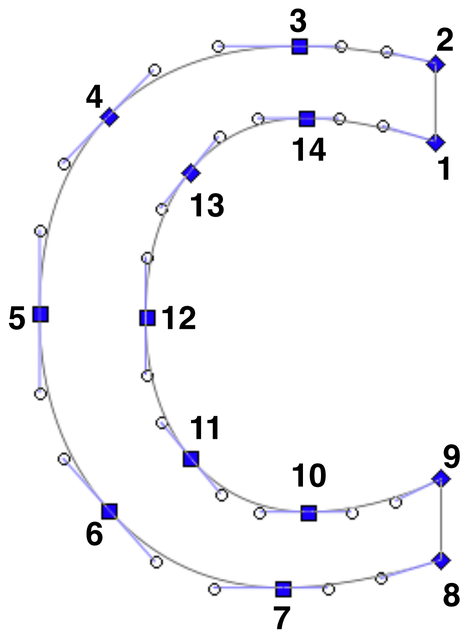
C. Write the four control points of a cubic Bézier curve with the following properties:

- endpoints are at (1, 0) and (0, 1)
- tangent at (1, 0) is vertical, or parallel to the vector [0, 1]
- tangent at (0, 1) is horizontal, or parallel to the vector [1, 0]
- midpoint is at $\left(\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}\right)$

D. Does the curve you created in 1C form part of a circular arc? Justify your answer.

E. Find the four control points of a cubic Bézier that form a curve identical to that of 1A.

Question 2: Rendering Fonts (3 points)



Examine the cubic Bézier spline shown on the right. On-curve control points (knots) are shown as solid squares or diamonds, and are numbered in the diagram. Off-curve control points are shown as hollow circles.

A. List the knots on the spline that are C^0 continuous.

B. List the knots on the spline that are G^1 continuous.

C. List the knots on the spline that are C^1 continuous.

Question 3: Scrolling Text (2 points)

A. Suppose you were to render your text in a window measuring 1000×1000 pixels at a refresh rate of 60 frames per second. If you scaled your text so that an average of 10 letters fit across the window, and you wanted to scroll the text at an average rate of 4 characters per second, how much would you have to move your text at every frame?

B. If you were to render the same text as in 3A, but now at 24 frames per second, how much would you have to move the text every frame to maintain the same scroll rate? Briefly describe how you might account for this difference in your program.

You may submit your answers in digital form (typed or scanned) in the appropriate drop box on the course Desire2Learn site, or as hard copy in the assignment boxes on the second floor of Math Sciences. Remember that late days may not be used for the written component!