CPSC 453 Assignment 3 Bezier, B-Spline Curves and Surfaces *

Fall 2021, University of Calgary

1 Overview and Objectives

The purpose of the third assignment in CPSC 453 is to provide you opportunities to familiarize yourself with 3D graphics in the OpenGL environment as well as splines for modelling curves and surfaces. Source code for a minimalistic and pedagogical boilerplate C++/OpenGL application is provided for you to use as a starting point, though you may create your own template if you prefer.

The most important outcome is to understand how to create Bezier curves and B-Splines in your C++ program and use them to construct 3D surfaces. These surfaces should then be viewable in 3D using the viewing pipeline taught in lecture. Some modification of the shaders and OpenGL function calls provided in the template will be necessary.

*Assignment specification taken from previous offering of CPSC 453 by Faramarz Samavati. Please notify drlarsen@ucalgary.ca of typos or mistakes.

2 Due Date

Nov 6th at 11:59 PM

3 Programming Assignment

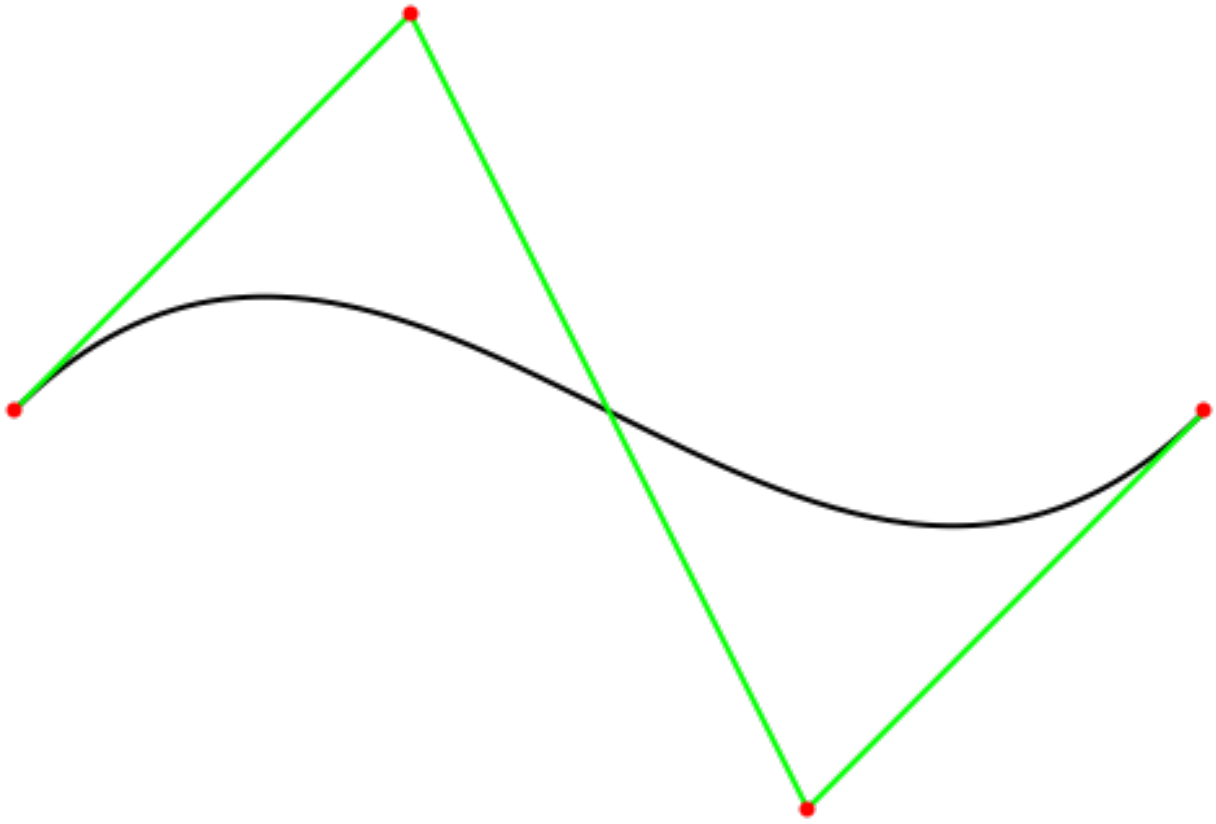
There are a total of four parts to this programming assignment with an additional requirement to integrate the scenes into a single application. This assignment is worth a total of 20 points, with the point distribution as indicated in each part, and an additional possibility of earning two "bonus" designations. Note that the bonuses may require going beyond what is presented in the lecture and tutorial components of the course and may require student initiative to find online resourses to complete successfully.

3.1 Part I: Bezier and B-Spline Curves (4 Points)

Create a program, or modify the provided template, to be able to draw draw a Bezier curve and a B-Spline curve, given a set of control points (You are required to do both).

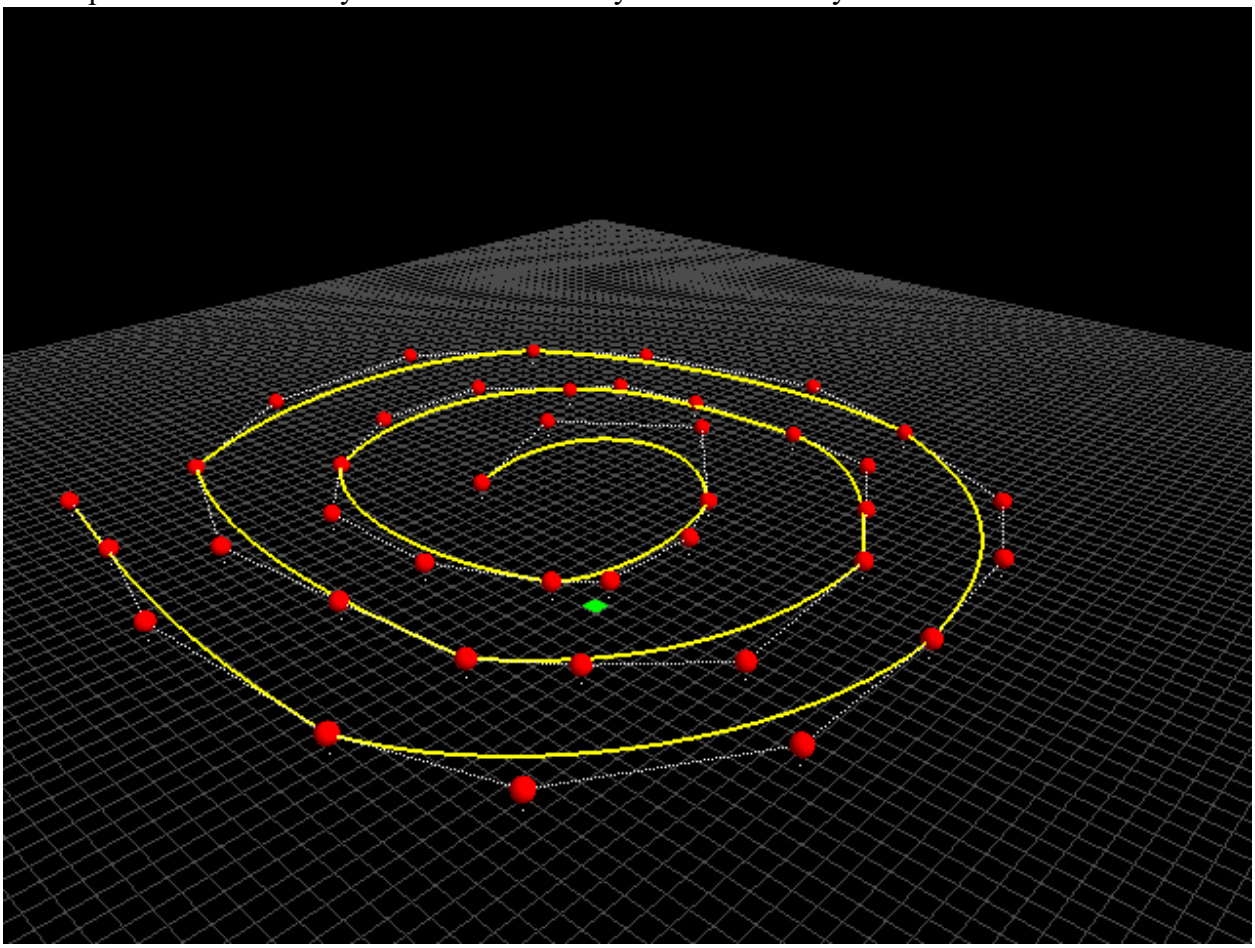
For creation of control points, the user should be able to click on the window at the location where they would like to create the point and it should appear there in the window. The user should also be able to modify the location of a currently existing control point using a click and drag operation. The user should be able to select and delete specific control points and should be able to reset the window, deleting all of the currently created control points. For implementing the selection, it may be useful to have all of the control points be the same color except for the currently selected control point which can be drawn in a different color. You should support addition of (within practical limits) an arbitrary number of control points. Note that the control polygon shown is green in the figure below does not need to be drawn (but could help for debugging of your program).

The Bezier curve should be constructed with the De Casteljau algorithm presented in lecture. The B-Spline should be an open quadratic B-Spline and can be constructed using the subdivision algorithm presented in lecture. Some kind of keyboard control should be provided to switch between which kind of curve that is drawn for the current set of control points. As control points are added to your scene, the curve should be updated automatically to take the newly created point into account.



3.2 Part II: Viewing Pipeline (4 Points)

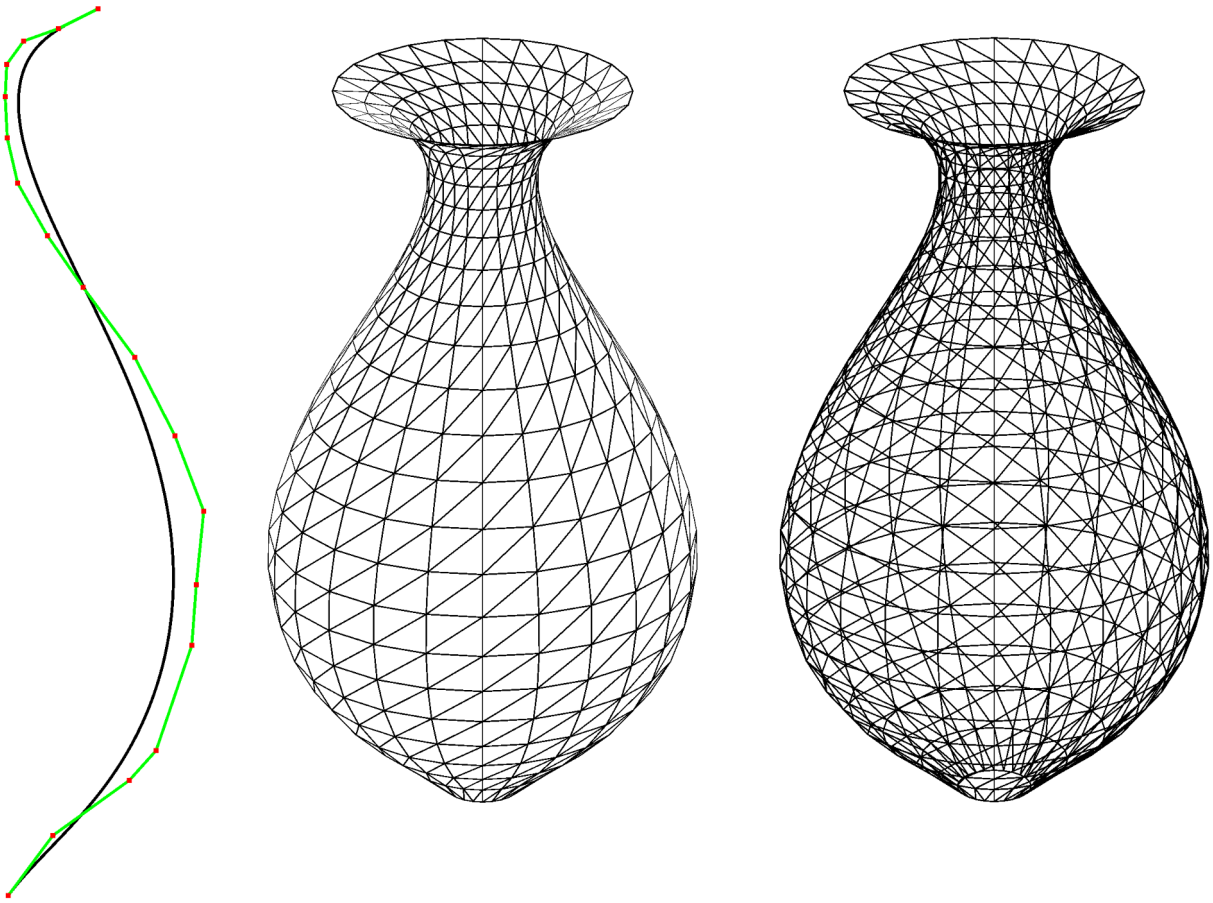
Add in the ability to view objects in 3D within your program. This will require the creation of a view and projection matrix which should be placed in your shader, similar to assignment 2. You should be able to toggle between viewing your scene in 2D for modifying your Bezier and B-Spline curves and viewing your scene in 3D. Note that you do not need to be able to add, remove or modify control points while viewing in 3D. Add in controls for moving the camera around in 3D so that your scene can be viewed from different angles. It is recommended that you provide a first person camera view seen in many modern games. You will likely want the camera to be controlled by the W, A, S and D keys as well as your mouse. The W key should translate the camera forward (along the direction it is currently looking), while the S key should translate backward. The A and D keys should translate the camera to the left and right respectively. By moving your mouse, you should be able to rotate your camera in the direction of the mouse movement. You can implement the mouse controls such that a click and drag is required to rotate the camera or you can implement it such that the camera always follows the mouse when in 3D view mode. Choose some reasonable values for the amount of translation and rotation created when you press these keys and move your mouse so that movement is smooth and controllable. It may be a good idea to develop Part II and Part III in parallel because they can be used to verify the functionality of each other.



3.3 Part III: Surface of Revolution (4 Points)

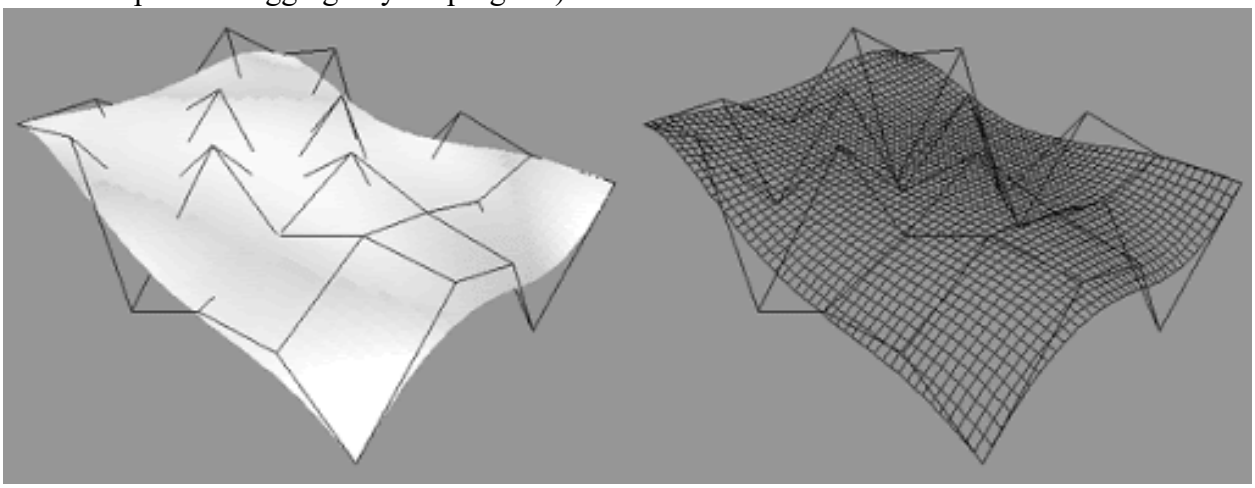
Use your control points and quadratic B-Spline curve from Part I to create a 3D surface of revolution. This surface should be viewable in 3D using your solution from Part II. Allow the user to be able to toggle the view of your surface between wireframe and solid surface (note that OpenGL has this functionality built in and it only require a single line of code to toggle it). Again note that you do not need to be able to add, remove or modify control points while viewing in 3D. Since you are not required to implement shading and lighting for this assignment, debugging your surface is probably much simpler in wireframe mode and if Part II is fully functioning, the ability to view your surface from different angles will be very useful. It may be a good idea to develop Part II and Part III in parallel because they can be used to verify the functionality of each other.

In the figures below the curve on the left was used to generate the surface of revolution in the middle. The one on the right is rendered using the wireframe mode that we expect you to use. Note that this results in all edges being drawn, including those from the triangles at the back of the scene. This is allowed and it is worth full points.



3.4 Part IV: Tensor Product Surface (4 Points)

Modify your program to allow the viewing of a tensor product surface using multiple sets of control points specifying multiple Bezier or B-Spline curves. If you wish, you can use your solution from Part I to generate the set of curves for creation of your surface. Similar to Part I and Part III, you should be able to view the surface in 3D (as a wireframe or as a solid surface) as well as the control points used to generate the surface. Provide 2 or 3 example surfaces to demonstrate that your solution to this part is working. Note that the control points for these surfaces can be hard coded into your program and you do not have to use your solution from Part I to generate the control points if you don't want to. Note that the black lines in the figure below do not need to be drawn (but could help for debugging of your program).



3.5 Integration and Control (4 Points)

Use keyboard input to provide a means for switching between the four scenes in your program. Ensure that your program does not produce rendering anomalies, or crash, when switching between scenes. 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.

3.6 Bonus Part I: Indexed Geometry (2 Points)

Modify your program so that the 3D geometry is indexed. This allows vertices to not be duplicated when they are sent to the GPU and is especially important for vertices of high valence in a surface or volume. Your glDrawArrays call should become a glDrawElements call for all of your 3D geometry to get full marks for this bonus.

3.7 Bonus Part II: Tesselation Shader (4 Points)

Modify your program so that the geometry for your curves and surfaces are generated in a tesselation shader (this requires OpenGL version 4.0). OpenGL optionally provides these shaders that can be inserted into your rendering pipeline for creating geometry. To do this, only the control

points for your curve should be passed to the GPU for rendering and the geometry for your curve or surface should be generated directly in the tesselation shader.

4 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. Include a "readme" text 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. In general, the onus is on you to ensure that your submission runs on your TA's grading environment for your platform! It is recommended that you submit a test assignment to ensure it works on the environment used by your TA for grading. Your TAs are happy to work with you to ensure that your submissions run in their environment before the due date of the assignment. Broken submissions may be returned for repair and may not be accepted if the problem is severe. Ensure that you upload any supporting files (e.g. makefiles, project files, shaders, data files) needed to compile and run your program. 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/`.