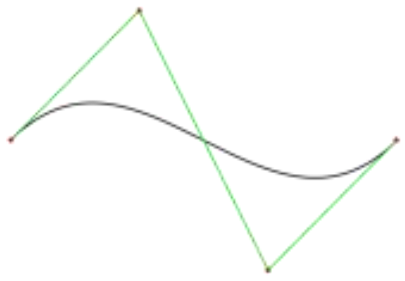
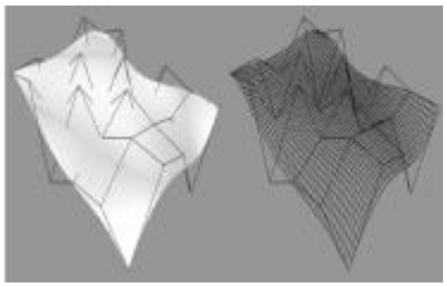
CPSC 453 Assignment 3 Bezier, B-Spline Curves and Surfaces [1]

Fall 2024, University of Calgary





1 Overview and Objectives

The purpose of the third assignment in CPSC 453 is to provide you with 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.

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.

2 Assignment Breakdown

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 resources to complete successfully. Be sure to cite all the resources used, especially for the bonuses.

[1] Assignment specification taken from previous offering of CPSC 453 by Faramarz Samavati and written by John Hall. Fall 2024 revision was updated by Jeffrey Layton. Please notify Jeffrey.layton@ucalgary.ca of typos or mistakes.

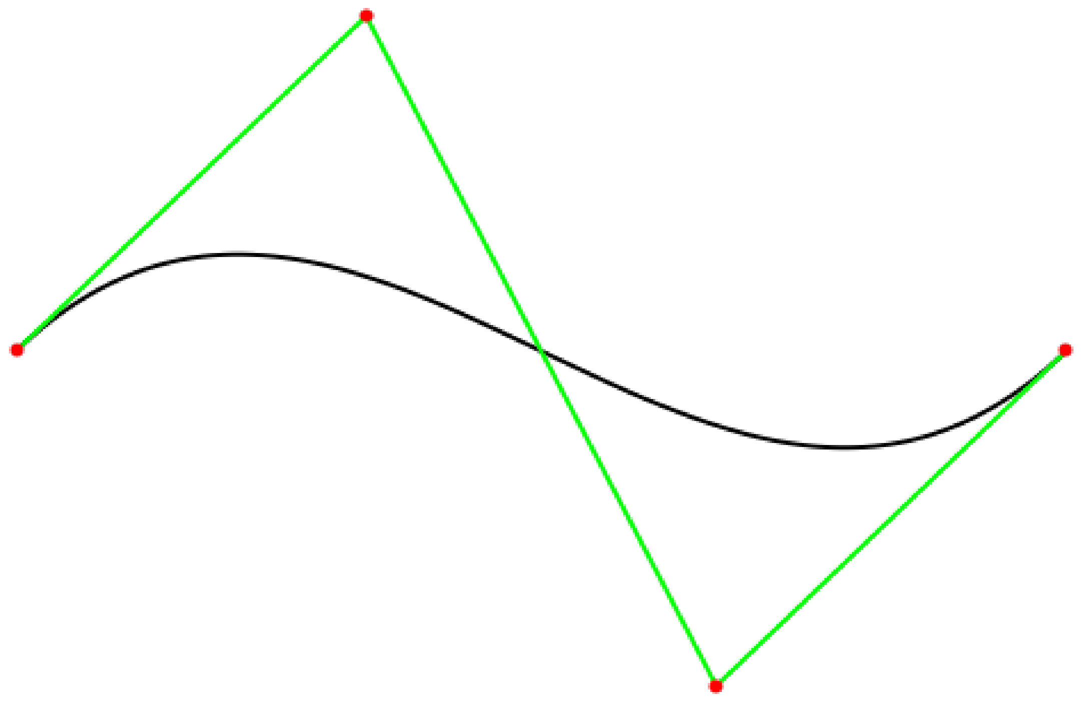
2.1 Part I: Bezier and B-Spline Curves (4 Points)

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

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 along with updating the curves with that additional control point.
- Modify the location of a currently existing control point using a click and drag operation.
- Select and delete specific control points.
- Reset the window, deleting all of the currently created control points.
- Switch between which kind of curve that is drawn for the current set of 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 the 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.



2.2 Part II: Viewing Pipeline (4 Points)

Add in the ability to view objects in 3D within your program with a ***perspective camera***. This will require the creation of a view and projection matrix which should be placed in your shader, similar assignment 2.

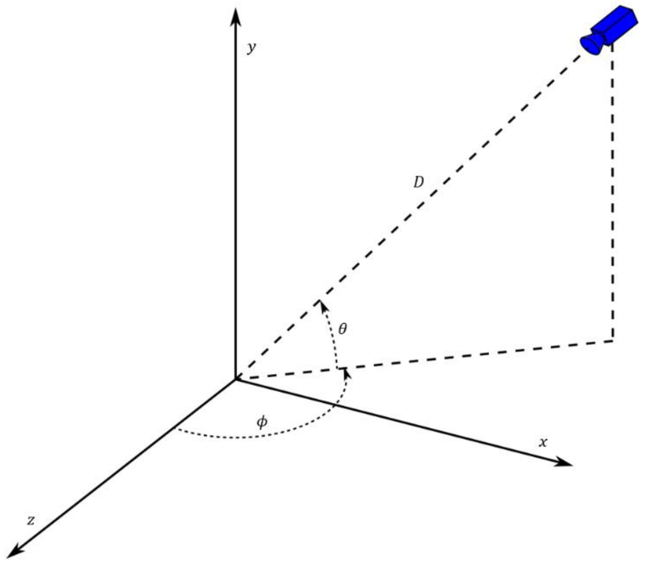
You should be able to:

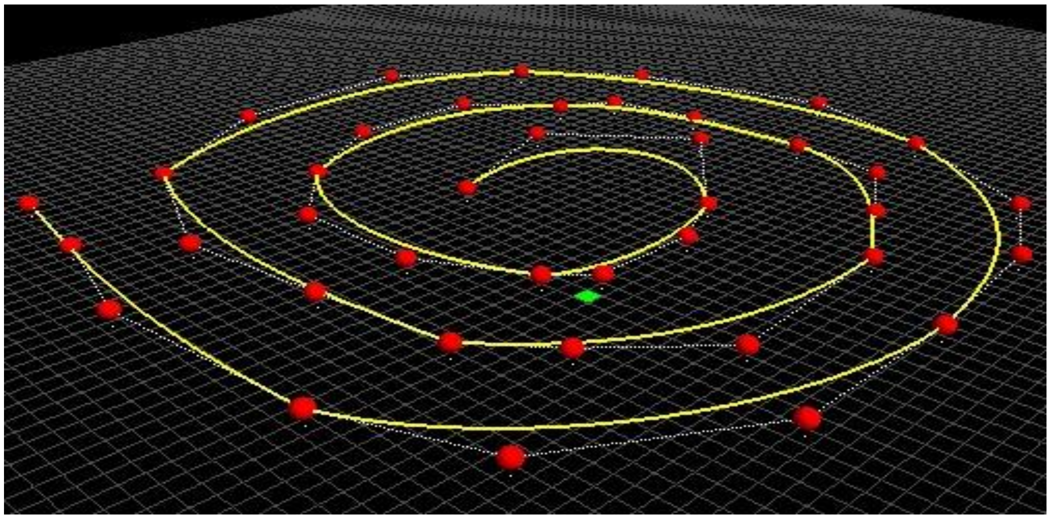
- Toggle between viewing your 2D curve editor for modifying your Bezier and B-Spline curves and your new 3D view containing your curve made in the editor centered at the origin. It may be orientated in any direction (example in the bottom figure).
- Control the camera in a so that we can move around (both position and direction) in 3D so that your scene can be viewed from different angles using an Orbit/Turn-Table Camera. Include a camera reset functionality in your code.
- Switch between which kind of curve that is drawn, along with their control points.

Note that you do not need to be able to add, remove or modify control points while viewing in 3D.

Orbit/Turn-Table Camera:

With your camera always pointing at the origin, the scroll wheel of the mouse will control the distance the camera is from the origin. Clicking and dragging the mouse will rotate the camera around the origin (opposite of your drag direction) at the aforementioned distance. See the diagram to the right for a representative diagram.





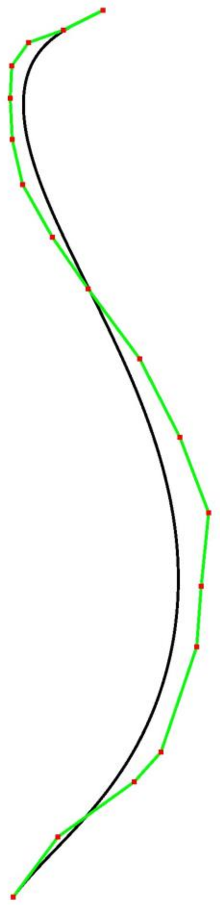
2.3 Part III: Surface of Revolution (4 Points)

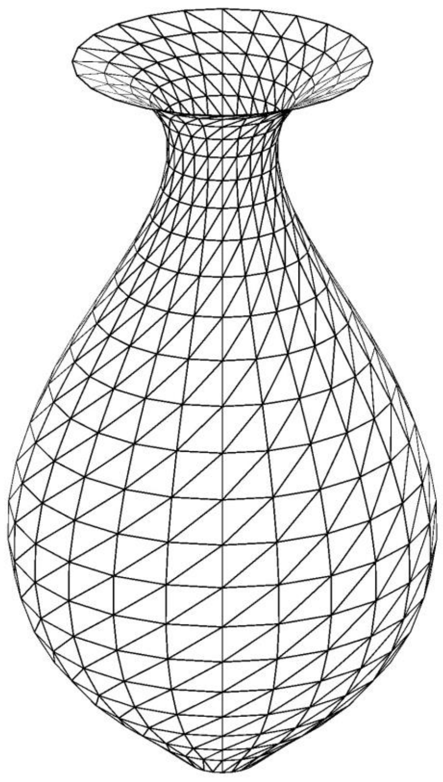
Use your control points and quadratic B-Spline curve from Part I to create a 3D surface of revolution. The user should be able to:

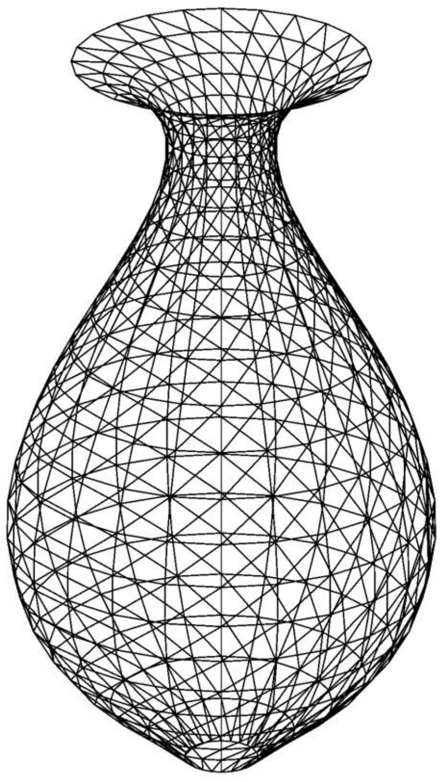
- View your surface just like in Part II.
- Toggle the view of your surface between the wireframe and solid surface.

OpenGL has wireframe functionality built in and it only requires a single line of code to toggle between wireframe and solid surface. 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.







2.4 Part IV: Tensor Product Surfaces (4 Points)

Modify your program to allow the viewing of a tensor product surface using multiple sets of control points specifying multiple B-Spline curves. Two surfaces need to be drawn:

- The surface described by the control points:

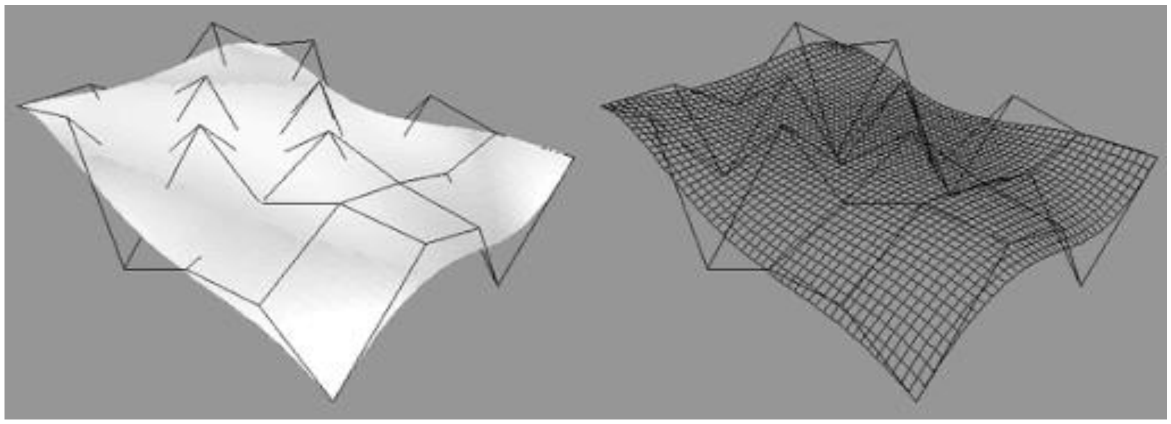
$(-2, 0, -2)$ ●	$(-1, 0, -2)$ ●	$(0, 0, -2)$ ●	$(1, 0, -2)$ ●	$(2, 0, -2)$ ●
$(-2, 0, -1)$ ●	$(-1, 1, -1)$ 🔴	$(0, 1, -1)$ 🔴	$(1, 1, -1)$ 🔴	$(2, 0, -1)$ ●
$(-2, 0, 0)$ ●	$(-1, 1, 0)$ 🔴	$(0, -1, 0)$ 🔵	$(1, 1, 0)$ 🔴	$(2, 0, 0)$ ●
$(-2, 0, 1)$ ●	$(-1, 1, 1)$ 🔴	$(0, 1, 1)$ 🔴	$(1, 1, 1)$ 🔴	$(2, 0, 1)$ ●
$(-2, 0, 2)$ ●	$(-1, 0, 2)$ ●	$(0, 0, 2)$ ●	$(1, 0, 2)$ ●	$(2, 0, 2)$ ●

- A surface of your choosing with non-equal UV dimensions (not square), like below.

The user should be able to:

- Switch between surfaces to render, along with their control points.
- Toggle the view of your surfaces between wireframe and solid surface.
- View your surface just like in Part II and Part III.

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 do not want to. Note that the black lines in the figure below do not need to be drawn (but could help with debugging of your program).



2.5 Integration and Control (4 Points)

Use keyboard input to provide a means for switching between the four scenes in your program (or using ImGui). 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.

2.6 Bonus Part I: Camera Extension (4 Points)

Note: The 4 points awarded to this bonus cannot be combined with the 4 points from the other for a total of 8 bonus points. A maximum of 4 bonus points, from completing one of the bonuses in full, will be awarded.

Modify your program to increase the camara capabilities for all your scenes. The user should be able to:

- Modify the window size so that all the cameras in the program adapt with the correct aspect ratio.
- ***2D Curve Editor***: Move the camera using click and drag mouse inputs.
- ***2D Curve Editor***: Change the zoom level of the camera using the scroll wheel.
- ***2D Curve Editor***: Reset the camera position and zoom level.
- ***3D Viewer:*** Modify the perspective camera FOV, near clip distance, and far clip distance. Be sure to include these in your reset functionality from part II.
- ***3D Viewer:*** Toggle between your perspective camera and a new 3D orthographic camera
- ***3D Viewer:*** Modify the orthographic camera scale of the horizontal and vertical clip distances, near clip distance, and far clip distance. Scaling your horizontal and vertical clip distances should preserve to the aspect ratio of the screen and will be the equivalent to the FOV + Zoom for the perspective camera. Be sure to include these parameters in your reset functionality from part II.

Be sure to add all controls to your "readme."

2.7 Bonus Part II: Tessellation Shader (4 Points)

Note: The 4 points awarded to this bonus cannot be combined with the 4 points from the other for a total of 8 bonus points. A maximum of 4 bonus points, from completing one of the bonuses in full, will be awarded.

Modify your program so that the geometry for your curves and surfaces are generated in a tessellation shader (this requires OpenGL version 4.0+, a version of GLAD that enables OpenGL 4.6 version of OpenGL has been included, inspect the CMakeLists.txt file to change the GLAD version). OpenGL optionally provides these shaders that can be inserted into your rendering pipeline for creating geometry. To do this, the control points for your curve, and any additional required data, should be passed to the GPU for rendering and the geometry for your curve or surface should be generated directly in the tessellation shader.

For full marks, the following curves/surfaces must be generated by the tessellation shader:

- Quadratic B-Spline rendering for Part I and II (ignore the Bezier curve)
- Your surface of revolution for Part III
- Both surfaces in Part IV

Completing one of the above items can get you up to an additional 2 points, with each of the remaining two adding up to an additional point. Marks will be assigned based on the quality of the implementation of the tessellation shader(s). Some pre-processing of your geometry is allowed to setup a sensible tessellation model, but this should be kept to a minimum. ***For each of the 3 parts, please provide a brief (3 to 5 sentences) design/implementation description for your tessellation inside your readme. Submissions missing rational will only receive partial credit for the bonus.***

3 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 3.3+ 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/.