Australian National University

COMP4610/8610 Computer Graphics

Homework Assignment #4, S2 2023

Topic: Bezier Curve

Date Issued: see Wattle page
Due Date : see Wattle page

Weighting: 10%

Instruction:

All homework assignments must be completed individually. We encourage you to discuss the assignments with other students. However, you should not share any of your codes with anyone else. Each student is responsible for implementing the assignment on their own. You may assist other in debugging their codes, but you should not copy and paste. ANU is using TurnItIn to detect possible duplications. Consulting with previous year students who enrolled in this course on specific assignment is also not allowed. You may use the internet as a resource for learning the materials, but you should not borrow any existing codes found online.

The homework assignments involve a significant amount of C++ programming. However, for most cases, a skeletal code base is provided, and you only need to fill in the missing parts, and/or fix bugs if any.

You will submit a single ZIP file as your submission, which must contain the following files:

(1) All source codes (ending in .h, or .hpp, or .cpp), and your Makefile, CMakeLists.txt. Please include all needed source codes for successful compilation. Please also remove all intermediate files (such as ./vs. . /build.) that are not needed for the compilation – Failing to do so will lead to penalty to the marks.

(2) A written HW4 Report (minimum 10-point font size, single column, in PDF format.)

Your ZIP file must be named as “COMPX610_2023_HW4_UID.zip”. Replace ‘X’ with 4 or 8. Replace the UID with your Uxxxxxxxx; Please submit your ZIP file to Wattle before the deadline. Late submission will lead to penalty as per the ANU policy. Later-than-one-week submission will not be accepted, which may result zero mark, unless a pre-approval for special consideration is obtained in written before the submission deadline.

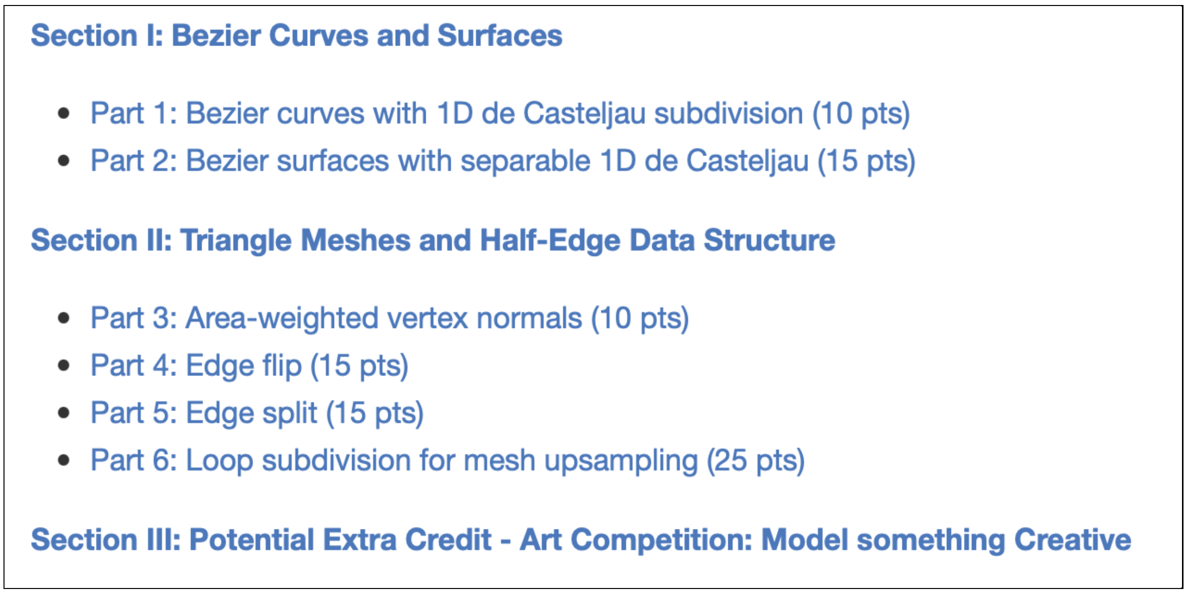
Tasks for Hw4:

There are in total two optional tasks for HW4. Your only need to choose one task to work on. The marking criterion for each of the two tasks will be consistent.

YOU ONLY NEED TO CHOOSE ONE TASK TO WORK ON AS YOUR HW4 SOLUTION.

Acknowledgement: these two tasks of this HW4 assignment are borrowed/adapted from project assignment from the University of California Berkeley's CS184 Computer Graphics course.
For interested student, you may refer to the original task descriptions here at https://cs184.eecs.berkeley.edu/sp23/docs/proj2#p2 .

While Berkeley's original homework assignment has seven parts (7 tasks, see the framed text-box below), in contrast, in our COMP4610/8610, you only need to complete one single task (chosen from part-1, and part-2 below.)



Section I: Bezier Curves and Surfaces

- Part 1: Bezier curves with 1D de Casteljau subdivision (10 pts)
- Part 2: Bezier surfaces with separable 1D de Casteljau (15 pts)

Section II: Triangle Meshes and Half-Edge Data Structure

- Part 3: Area-weighted vertex normals (10 pts)
- Part 4: Edge flip (15 pts)
- Part 5: Edge split (15 pts)
- Part 6: Loop subdivision for mesh upsampling (25 pts)

Section III: Potential Extra Credit - Art Competition: Model something Creative

Task-1: Bezier Curves

1.1 Overview

The Bézier curve is a parametric curve used in computer graphics. In this assignment, you need to implement the *de Casteljau* algorithm to plot a Bézier curve represented by 4 control points (after you have implemented the algorithm correctly, you can begin plotting Bézier curves controlled by more points).

The functions you need to modify are in the main.cpp file provided.

- *bezier* - This function implements the function that draws the Bézier curve.

It uses a sequence of control points and an OpenCV:: Mat object as the input, with no return value. It causes t to change in value over a range from 0 to 1, increasing t by a tiny value in each iteration. For each t that needs to be computed, another function *recursive_bezier* is called for, which then returns the point at t on the Bézier curve. Finally, the returned points are plotted on the OpenCV::Mat object.

Hint: If you encounter library compatibility issues with using OpenCV in your virtual machine, you may replace the OpenCV libraries with alternative/equivalent libraries, or use your own implementation.

Alternatively, for doing HW4 (which is a standalone task, not relates to other HW assignments), you may implement your codes on your native Operating System (rather than via the Virtualbox).

- *recursive_bezier* - This function implements the *de Casteljau* algorithm to return the coordinates of the corresponding point on the Bézier curve, using a sequence of control points and a floating-point number t as input.

1.2. Algorithm

The *De Casteljau* algorithm is described below:

1. Consider a Bézier curve with a sequence of control points p0, p1, ..., pn. First, join the adjacent points to form line segments.

2. Subdivide each line segment by the ratio t : (1 - t) and find the segmentation point

3. The obtained segmentation points are used as a new sequence of control points, and the length of the new sequence is reduced by one.

4. If the sequence contains only one point, it returns that point and terminates. Otherwise, use the new sequence of control points and go to step 1.

   By using several different t's in the range [0,1] to perform the above algorithm, you will get the corresponding Bézier curves.

5. [Advanced feature] When you draw your Bezier curve on screen, please test with and without anti-aliasing, and include your results in the HW report.

1.3 Start Writing

In this assignment, you will write on a new code framework, which is much smaller than the previous one. Please download the skeleton code for the project and build the project as before using the following command:

```
$ mkdir build
$ cd build
$ cmake . .
$ make
```

After that, you can run the given code by using the following command: *./BezierCurve*

When running, the program will open a black window. Now, you can click on the screen to select points to control the Bézier curve. The program will wait for you to select 4 control points in the window and then it will automatically draw the Bézier curve based on the control points you selected. The naive Bezier implementation provided in the skeleton code calculates the Bézier curve and plots it in red by using a polynomial equations.

1.4 After making sure everything is in order with the skeleton code, it's time to start completing your own implementation. Comment out those naïve_bezier code in the main function. Use your implementation to plot the Bézier curve in green.

To make sure the implementation is correct, call both the naïve_bezier and bezier functions, and if the implementation is correct, then both should write approximately the same number of pixels, so the curve will appear yellow. If this is the case, the implementation is correct.

You can also try to modify the code and use a different number of control points to plot different Bézier curves.

[Advanced feature] When you draw the Bezier curve on screen, please use an anti-aliasing technique so that the curve can be rendered properly without colour aliasing. Please compare the two rendering results, one is without anti-aliasing, and one is with anti-aliasing.

Task-2: Bezier surface patch

In task-2, you will implement a Bezier surface using separable 1D de Casteljau algorithm.

2.1 Overview:

The functions you mainly need to modify are in the main.cpp file provided.

- *evaluate1DBezier* - This function implements the 1D de Casteljau algorithm to return the coordinates of the corresponding point on the Bézier curve, using a sequence of control points and a floating-point number t as input. It serves a foundational role in generating Bézier patches.

- *generateBezierPatch* - This function takes a grid of control points as input and transforms it into a list of triangles representing a tessellated Bézier surface patch. It achieves this by computing points on the Bézier surface across both the u and v dimensions using *evaluate1DBezier.* Then the resulting surface points are tessellated into triangles.

2.2 Specification:

- Bezier Patches:
  - Compute the surface points using separable 1D de Casteljau algorithm from a set of given control points. Use an iterative, not recursive, evaluation technique to calculate vertices on the patch.
  - Tessellate the surface points into triangles and calculate the surface for each triangle. Initially the patch should be sampled with a 10 x 10 resolution.
  - Set at least one light source, which is near and connected to the camera, and render the patch using Blinn-Phong shading model.

- Interaction:
  - Draw the X, Y and Z axes. They should be drawn at the origin of the patch's coordinate system. X axis should be drawn as a short red line. Y axis as short green line, and Z axis as a short blue line.
  - The patch should be in the middle of the window when your program starts.
  - The user should be able to change the view of the patch, and the camera should always look at the patch.
  - Include a command to reset the view.
  - The user should have the capability to choose a control point to modify, via keyboard interaction.
  - Change the colour and/or size of the selected control point.
  - The user can then change the location of the selected point via 6 keyboard buttons (increase/decrease x/y/z position).
  - The user should be able to increase and decrease the sampling of the patch with a keyboard interaction.

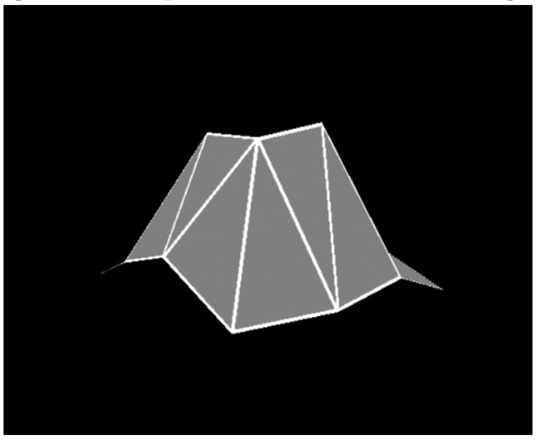
2.3 Start Writing:

In this assignment, you will write on a new code framework, which is based on the one in HW3. Please download the skeleton code for the project, read through it. Before build, complete *rasterize_triangle(const Triangle& t)* in rasterizer.cpp. You can just copy and paste your implementation from HW3. Then build the project as before using the following command:

```
$ mkdir build
$ cd build
$ cmake . .
$ make
```

After that, you can run the given code by using the following command: *./BezierSurface*

When running, the program will open a window as the image below:

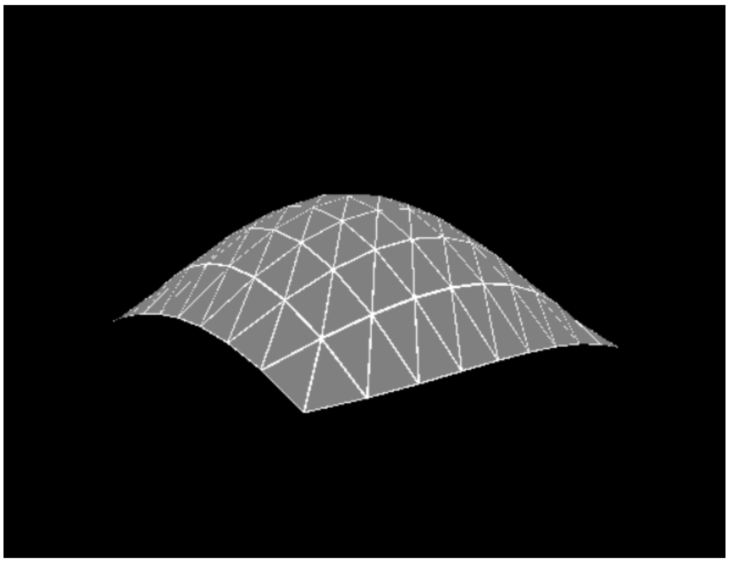


It's a hard surface mesh generated from the given 4*4 points. You can use left and right arrow keys to rotate it. By default, it is using a wireframe fragment shader, so you can see all the edges of your surface. You can switch to normal and Blinn Phong shaders using command *./BezierSurface normal* and *./BezierSurface phong*.

2.4 Use your implementation

After making sure everything is in order with the skeleton code, it's time to start completing your own implementation. Comment out those *createHardSurface()* code in the main function and uncomment *generateBezierPatch()* to test your implementation.

If your Bezier surface is correctly made using separable 1D de Casteljau algorithm, you may get similar results as the image below using wireframe shader:



Submission requirement and Marking criteria:

Please refer to "Homework#4 submission template and marking criteria" posted on Wattle.

Your ZIP file must be named as "COMPX610_2023_HW4_UID.zip".

==END ===