Assignment 5: Bézier Curves
CS180 Winter 2020

Professor: Lingqi Yan

University of California, Santa Barbara

Assigned on Feb 11, 2021 (Thursday)

Due at 23:59 on Feb 17, 2021 (Wednesday)

Notes:

- Be sure to read the Programming and Collaboration Policy on course website.
- Any updates or correction will be posted on Piazza, so check there occasionally.
- You must do your own work independently.
- Read through this document carefully before posting questions on Piazza.
- Submit your assignment on GauchoSpace before the due date.

---

1 Overview

Bézier curves are parametric curves that are used in computer graphics. In this assignment, you are expected to implement the **de Casteljau's** algorithm for drawing a Bézier curve, defined by 4 control points (although once you've implemented the algorithm correctly, there's no reason why it shouldn't work with more).

The functions you should modify are given in the **main.cpp** file.

- `bezier`: This function should draw the Bezier curve. It takes a set of points in pixels and an `OpenCV::Mat` object as inputs, and doesn't output anything. It iterates through the range 0 to 1, starting at `t = 0` and increasing it just a little in each iteration. Look at `naive_bezier` function to see how small these iterations should be. For each `t` in this loop, the other function `recursive_bezier` is called, which should return the point the Bezier curve for that `t`. Finally, that returned point should be painted on the OpenCV::Mat object in `green` color (again look at `naive_bezier` for help, which paints the point in red color).
- `recursive_bezier`: This function takes a set of points (control points), and a float `t`, and implements the **de Casteljau**'s algorithm to find the point that's on the Bezier curve.

2 Algorithm

De Casteljau algorithm is explained as following:

1. Consider a Bezier curve with control points p_0 ... p_n. You first connect the consecutive points to have line segments.
2. Subdivide each line segment with the ratio $t : (1 - t)$, and find the points at the division.
3. These points are your new control points, and their number will be one fewer than the previous step.
4. If you have only one point, then return that point and terminate. Otherwise, go to step 1 with current points.

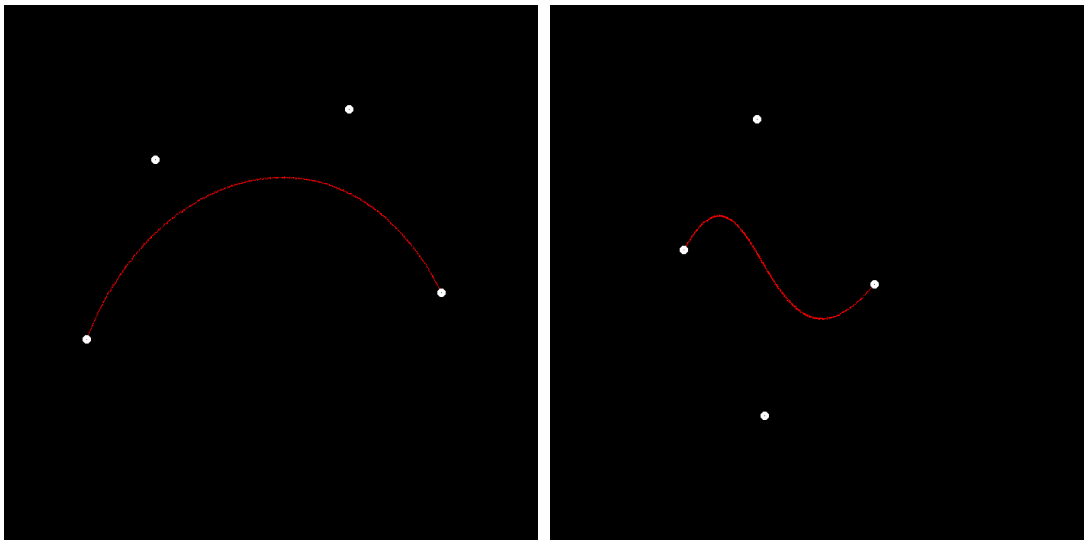
If you apply this algorithm for different **t**'s between [0, 1], you will get the corresponding Bezier curve.

3 Getting started

This assignment will be based on a different program from the previous assignments, which is much smaller. Similar to the previous assignments, you can choose to work either on your own computers or on the virtual machine. Download the project's skeleton code, and build the project like we did previously by using the following commands:

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

Now you should be able to run the given code by `./BezierCurve`. When run, a black window will be opened. Now, you can click on the window to place points. The program waits for you to place 4 points and then it will draw a Bezier curve based on them. The given implementation computes the curve by using its polynomial equation and draws it in red. The following are examples of how this would look:



After making sure that this is working, you can do your implementation. Comment out the line invoking `naive_bezier` function inside the `while` loop in the `main` function, and uncomment the `bezier` function instead. Now you must implement the `bezier` and `recursive_bezier` functions. After finishing your implementation,

if you follow the above steps to compile, run and draw the curve, it should be drawn in `green` color. Don't forget to make it green.

If you want to make sure your program is working correctly, call both the `naive_bezier` and `bezier` functions, by uncommenting both. If your implementation is correct, both should write to the approximately to the same pixels, thereby producing the line in **yellow**. Remember that for this, you should increment `t`, in your loop for each iteration, by the same amount as `naive_bezier` does, and also you should draw the curve in green. If you verify so, you can be sure that your implementation is correct.

You can also try to modify the code and experiment with different numbers of control points and see different Bezier curves.

4 Grading and Submission

4.1 Grading:

- `[5 points]` Submission is in the correct format, include all necessary files. Code can compile and run.
- `[20 points] De Casteljau's Algorithm:`
  Given the control points, your code should be able to produce the correct Bezier curve.
- `[5 points] Bonus points:`
  Implement anti-aliasing on the Bezier curve. (When you find a point that lies on the curve, instead of putting it directly to a single pixel, you need to splash it to the neighboring pixels according to the distances to the pixel centers.)

4.2 Submission:

After you finish the assignment, delete the build folder and leave both CMakeLists.txt and main.cpp in the folder. Now add a short Report in pdf format to the directory, write down your full name and perm number inside it, add images of the Bezier curves you obtained (if you did for more than 4 points include them too!) in both colors. State whether you did the bonus, and add images of anti aliased images if you did. Then compress the entire folder and rename it with your name, like "Kalyan.zip". Submit the **.zip** file on GauchoSpace.