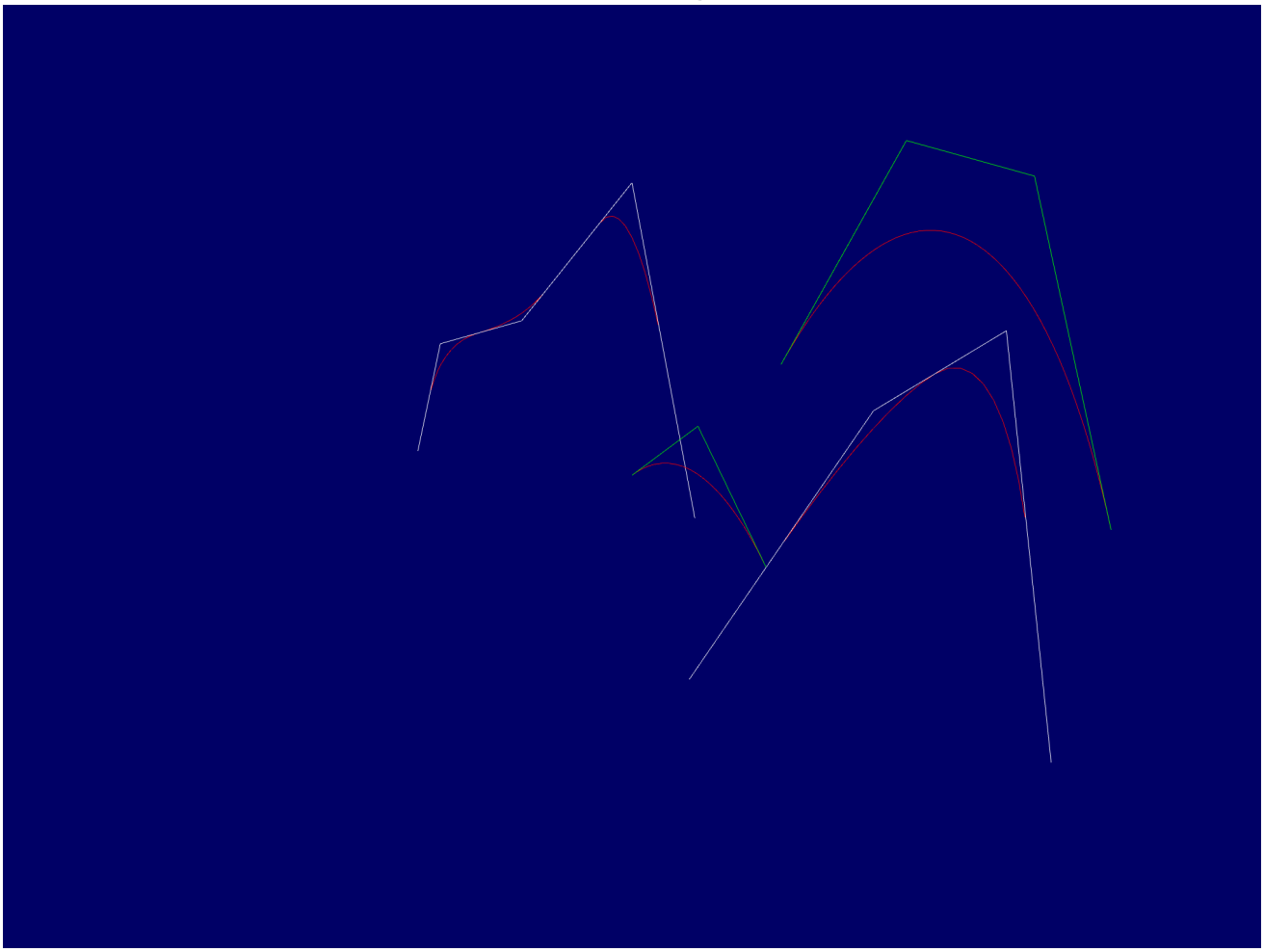
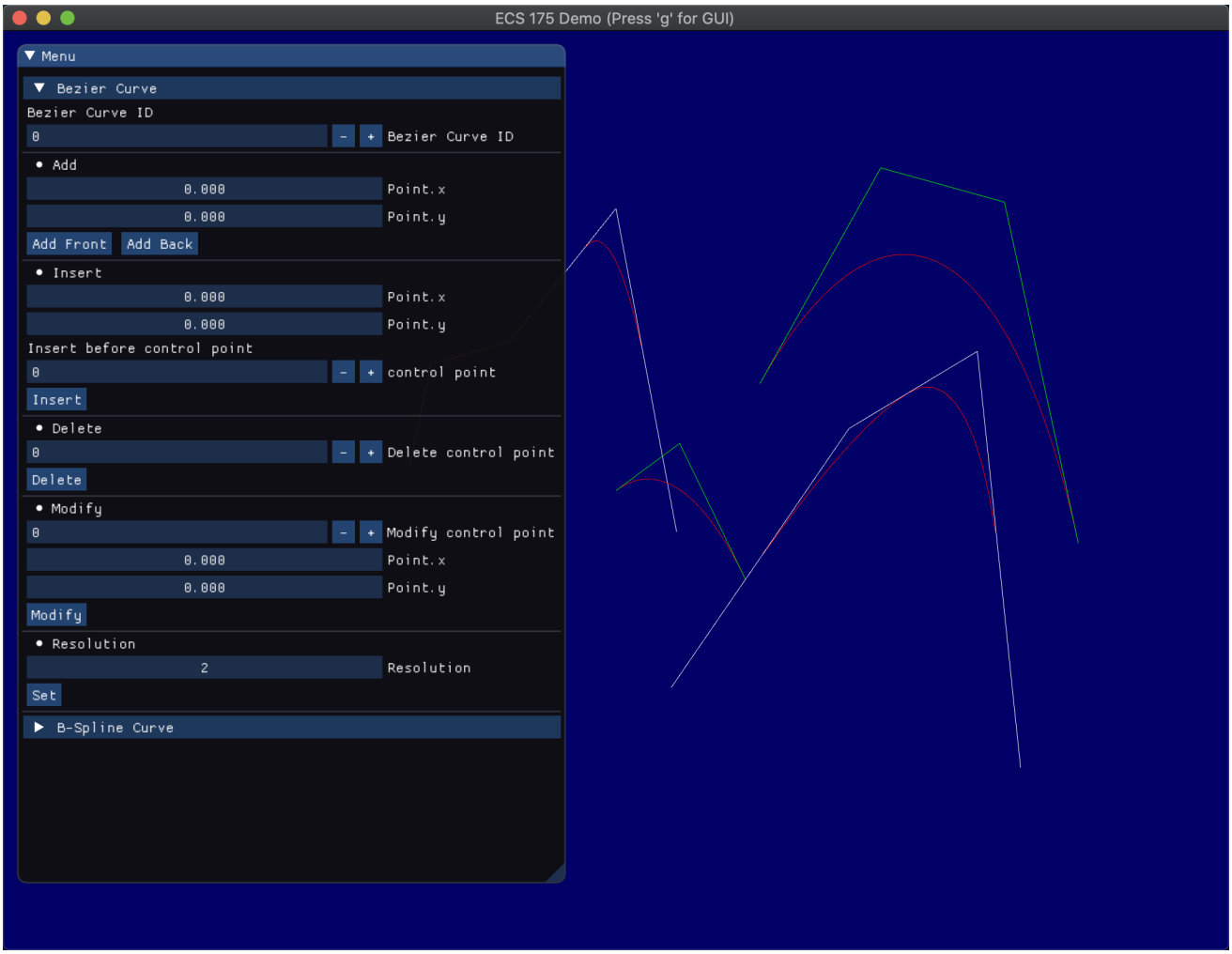
README.md

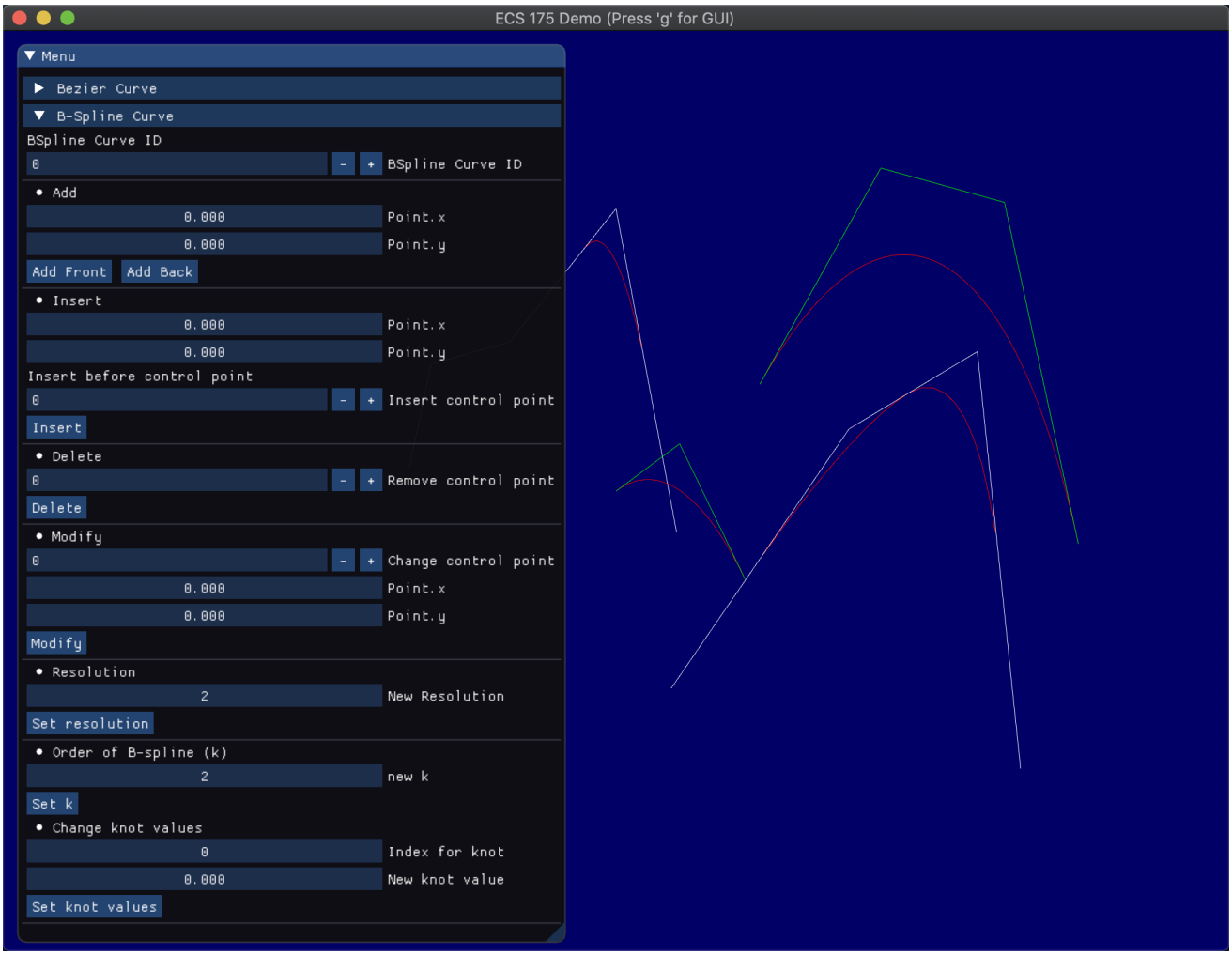
# General info

This project supports the implementation of a 2D Bezier and B-spline curve editor. The project first draw curves from the input file, and the user can press `G` to interact with the GUI and curve editor will display. The user can add, insert, delete, and modify the control points in any curve, and can specify the number of points (resolution) to be drawn and the order (k) of any curve through the GUI.

The bezier curve and bspline curves will be drawn in red. However, the control points for bezier curve is green, and the control points for b-spline is white. In the following image, two bezier and two bspline curves are drawn along with their control points connected.




ECS 175 Demo (Press 'g' for GUI)
Menu
Bezier Curve
Bezier Curve ID
0
Bezier Curve ID
Add
0.000
Point.x
0.000
Point.y
Add Front
Add Back
Insert
0.000
Point.x
0.000
Point.y
Insert before control point
0
control point
Insert
Delete
0
Delete control point
Delete
Modify
0
Modify control point
0.000
Point.x
0.000
Point.y
Modify
Resolution
2
Resolution
Set
B-Spline Curve


ECS 175 Demo (Press 'g' for GUI)
Menu
Bezier Curve
B-Spline Curve
BSpline Curve ID
0
BSpline Curve ID
Add
0.000
Point.x
0.000
Point.y
Add Front
Add Back
Insert
0.000
Point.x
0.000
Point.y
Insert before control point
0
Insert control point
Insert
Delete
0
Remove control point
Delete
Modify
0
Change control point
0.000
Point.x
0.000
Point.y
Modify
Resolution
2
New Resolution
Set resolution
Order of B-spline (k)
2
new k
Set k
Change knot values
0
Index for knot
0.000
New knot value
Set knot values

All the GUI display function can be found at `gui.cpp` and `gui.h` , and where as the computation function can be found at `helper.cpp` and `helper.h` file.

## Usage

To run the program, for the first command line argument please specify either "Both" or "Bezier" or "Bspline" to specify which curve to draw, whether or not these three words are capitalized does not matter, but please do spell it correctly.

If the first command line argument is `Bezier` , please specify the path of the input file for Bezier Curve as the second command line argument.

If the first command line argument is `Bspline` , please specify the path of the input file for Bspline Curve as the second command line argument.

If the first command line argument is `both` , please specify the path of the input file for Bezier Curve as the second command line argument, and the path of the input file for Bspline Curve as the third command line argument. The program will not continue if the number of command line argument is less than 2.

```
./run_p4 Bezier /projects/p4/2BezierCurve.txt
./run_p4 Bspline /projects/p4/2Spline.txt
./run_p4 Both /projects/p4/2BezierCurve.txt /projects/p4/2Spline.txt
```

To interact with the editor, press `G` after the window has shown up.

To exit the program, press `ECS` and all the work will be saved to the input file.

## Input File Format

### Bezier Curve

The bezier curve file have the following format:
2 (number of bezier curve)

3 (number of 2D points for 1st bezier curve)
0.0 0.0 (1st point)
0.5 0.5 (2nd point)
1.0 0.0 (3rd point)

4 (number of 2D points for 2nd bezier curve)
-1.0 -1.0 (1st point)
0.0 0.0 (2nd point)
0.5 0.0 (3rd point)
0.5 -1.0 (4th point)

### Bspline curve

The b-spline curve file have the following format:
2 (number of b-spline curve)

3 (k order of the first b-spline curve)
4 (number of control points)
0.5 -1.0 (1st control point)
1.5 1.0 (2nd control point)
2.0 1.5 (3rd control point)
2.5 0.0 (4th control point)
0 (1st knot)
0.5 (2nd knot)
1.0 (3rd knot)
1.5 (4th knot)
2.0 (5th knot)
2.5 (6th knot)
3.0 (7th knot)

3 (k order of the first b-spline curve)
6 (number of control points)
-2.5 -1.0 (1st control point)
-2.0 0.0 (2nd control point)
-1.0 0.5 (3rd control point)
-0.5 1.0 (4th control point)
0.0 1.5 (5th control point)
0.5 0.0 (6th control point)
0 (1st knot)
0.5 (2nd knot)
1.0 (3rd knot)
1.5 (4th knot)
2.0 (5th knot)
2.5 (6th knot)
3.0 (7th knot)
3.5 (8th knot)
4.0 (9th knot)

## Input File Function

There are two types of input file format: one for bezier curve and one for bspline curve, and there are 6 files already provided: `1BezierCurve.txt`, `2BezierCurve.txt`, `3BezierCurve.txt`, `1BsplineCurve.txt`, `2BsplineCurve.txt`, `3BsplineCurve.txt`.

The function responsible for parsing input can be found at line 21 in `helper.cpp` and is

```
void Readfile(int argc, char** argv);
```

The function responsible for reading bezier curve can be found at line 47 in `helper.cpp` and is

```
void ReadFileBezierCurve(const std::string& input);
```

The function responsible for reading b-spline curve can be found at line 75 in `helper.cpp` and is

```
void ReadFileBSplineCurve(const std::string& input);
```

## Output File Function

The final scene will be written to the same input file replacing the original content. This part is handled at line 109 in `helper.cpp` by

```
void Outputfile(int argc, char** argv);
```

However this function uses two smaller function: one for output for bezier curve, and one for output for b-spline curve.

The function responsible for outputting bezier curve can be found at line 130 in `helper.cpp` and is

```
void OutputFileBezierCurve(const std::string& input);
```

The function responsible for outputting bezier curve can be found at line 148 in `helper.cpp` and is

```
void OutputFileBSplineCurve(const std::string& input);
```

## Data Structure

A bezier curve is defined at line 18 in `helper.h` and as

```
struct BezierCurve {
  // Output {c0, c1, c2, ...}
  std::vector<Point> C;

  // Input {b0, b1, b2 , ...}
  std::vector<Point> B;

  // Interval for time, t \in [0, 1]
  std::vector<float> domain;

  // Resolution for smoothness of curve
  int res = 50;

  // n  = number of control point - 1
  int n;

  // VBO object
  std::vector<float> g_vertex_buffer_lines;
  GLuint vertex_buffer_id_lines;

  // VBO object for Control points
  std::vector<float> b_vertex_buffer_lines;
  GLuint b_vertex_buffer_id_lines;
};
```

A b-spline curve is defined at line 51 in helper.h and as

```
struct BSplineCurve{
  // Output {c0, c1, c2, ...}
  std::vector<Point> C;

  // Knots {u0, u1 , ...} where u0 < u1 < u2 < u3 < ... < u_{n + k}
  std::vector<float> U;

  // Control points {d0, d1, d2, ...} as input
  std::vector<Point> D;

  // Domain
  std::vector<float> domain;

  // Order
  int k;

  // n  = number of control point - 1
  // Note that n >= k - 1
  int n;

  // Resolution for smoothness of curve
```

```
    int res = 10;

    // VBO object for Bezier Curve
    std::vector<float> g_vertex_buffer_lines;
    GLuint vertex_buffer_id_lines;

    // VBO object for Control points
    std::vector<float> b_vertex_buffer_lines;
    GLuint b_vertex_buffer_id_lines;
  };
```

For both curves, I also want to draw the control points and thus for each curve there is a VBO object for the control point.

The program uses two global variable to store all the bezier and bspline curves and they are

```
std::vector<BSplineCurve> BSplineCurve_vec;
std::vector<BezierCurve> BezierCurve_vec;
```

The ID of each bezier curve is defined as the order that they are read in. For instance, the first bezier curve and the first bspline curve will have ID 0, and so on. There are two sections in the GUI for each type of curve, so both type of curve having the same ID won't affect each other.

## Control flow in fragment shader

There are two variable `original` and `bezier` that are sent to the fragment shader. They are used to control the color of control points and curves through if statements in `p4_shader_color.glsl`. They can be found at line 93 in `main.cpp`.

The effects are that the bezier curve and bspline curves will be drawn in red. However, the control points for bezier curve is green, and the control points for b-spline is white.

## Implementation

### VBO

Before generating VBOs, the bezier curve and b-spline curve is calculated. The function for creating VBO for both bezier curve, b-spline curve, and their corresponding control points can be found at line 208 in `helper.cpp` and is

```
void CreateVBO();
```

There are also 4 smaller function in `CreateVBO()` and they are

```
// Create VBO for Bspline
// line 233 in helper.cpp
void VBOSingleBSplineCurve(BSplineCurve& BSpline);

// Create VBO for control point of Bspline
// line 251 in helper.cpp
void VBOSingleBSplineControlPoint(BSplineCurve& BSpline);

// Create VBO for Bezier curve
// line 270 in helper.cpp
void VBOSingleBezierControlPoint(BezierCurve& Bezier);

// Create VBO for control point of Bezier curve
// line 289 in helper.cpp
void VBOSingleBezierCurve(BezierCurve& Bezier);
```

## DeCasteljau algorithm

The DeCasteljau algorithm can be found at line 376 in `helper.cpp` . The algorithm follows the procedure from the lecture slide.

The function for calculating Bezier curve and using DeCasteljau algorithm numerously can be found at line 308 in `helper.cpp` and is

```
void GenerateBezierCurve();
```

## DeBoor algorithm

The DeBoor algorithm can be found at line 401 in `helper.cpp` . The algorithm follows the procedure from the lecture slide.

The function for caculating Bezier curve and using De Bour algorithm numerously can be found at line 343 in `helper.cpp` and is

```
void GenerateBSplineCurve();
```

The `I` value is also calculated in this function at line 424.

## Domain for Bspline and Interval for Bezier curve

The function for setting the domain for each bspline curve can be found at line 182 in `helper.cpp` and is

```
void BSplineSetDomain(BSplineCurve& BSplineCurve);
```

The function for setting the interval for each bezier curve can be found at line 170 in `helper.cpp` and is

```
void SetResolution(BezierCurve& curve);
```

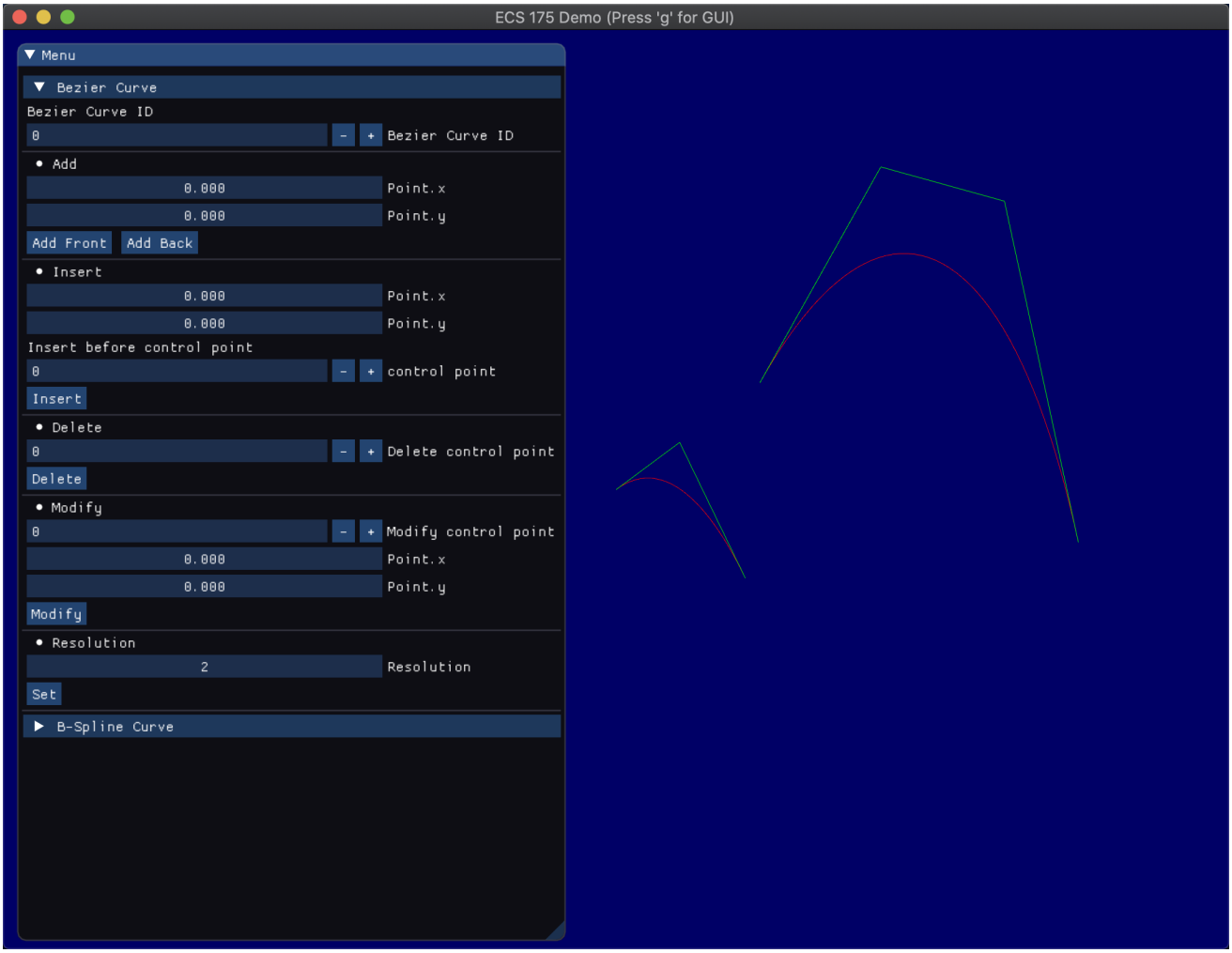
## Editor

### Bezier

The corresponding execution for bezier curve editor can be found at line 46 in `gui.cpp` and the function is

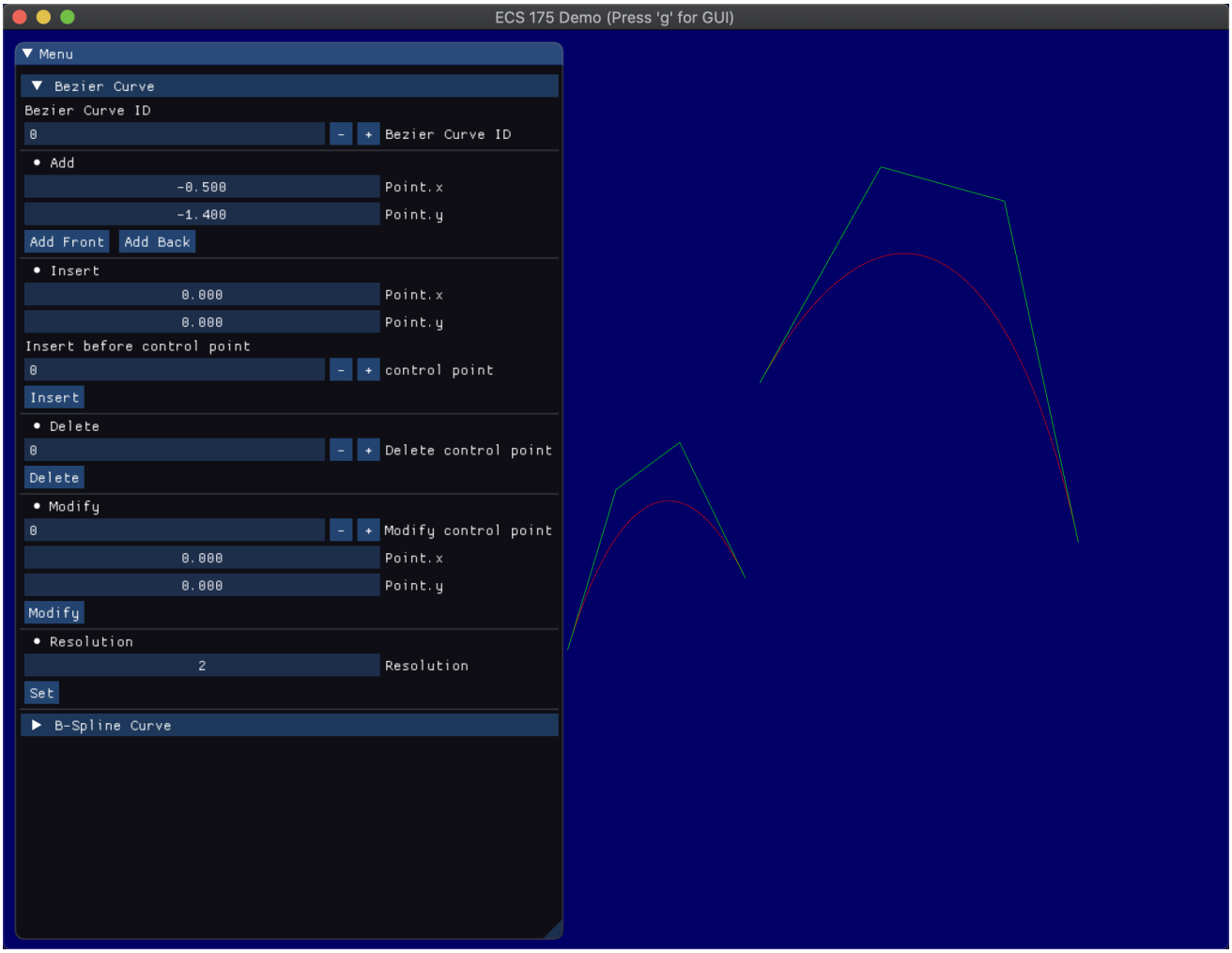
```
void BezierMenu(bool* p_boolean, GLFWwindow* window)
```

Now I want to demonstrate how to use the editor. In the following picture, there are two bezier curve generated from `2BezierCurve.txt`. The bezier curve with ID 0 is the one on the left that has 3 control points, and the bezier curve with ID 1 is the one on the right that has 4 control points.

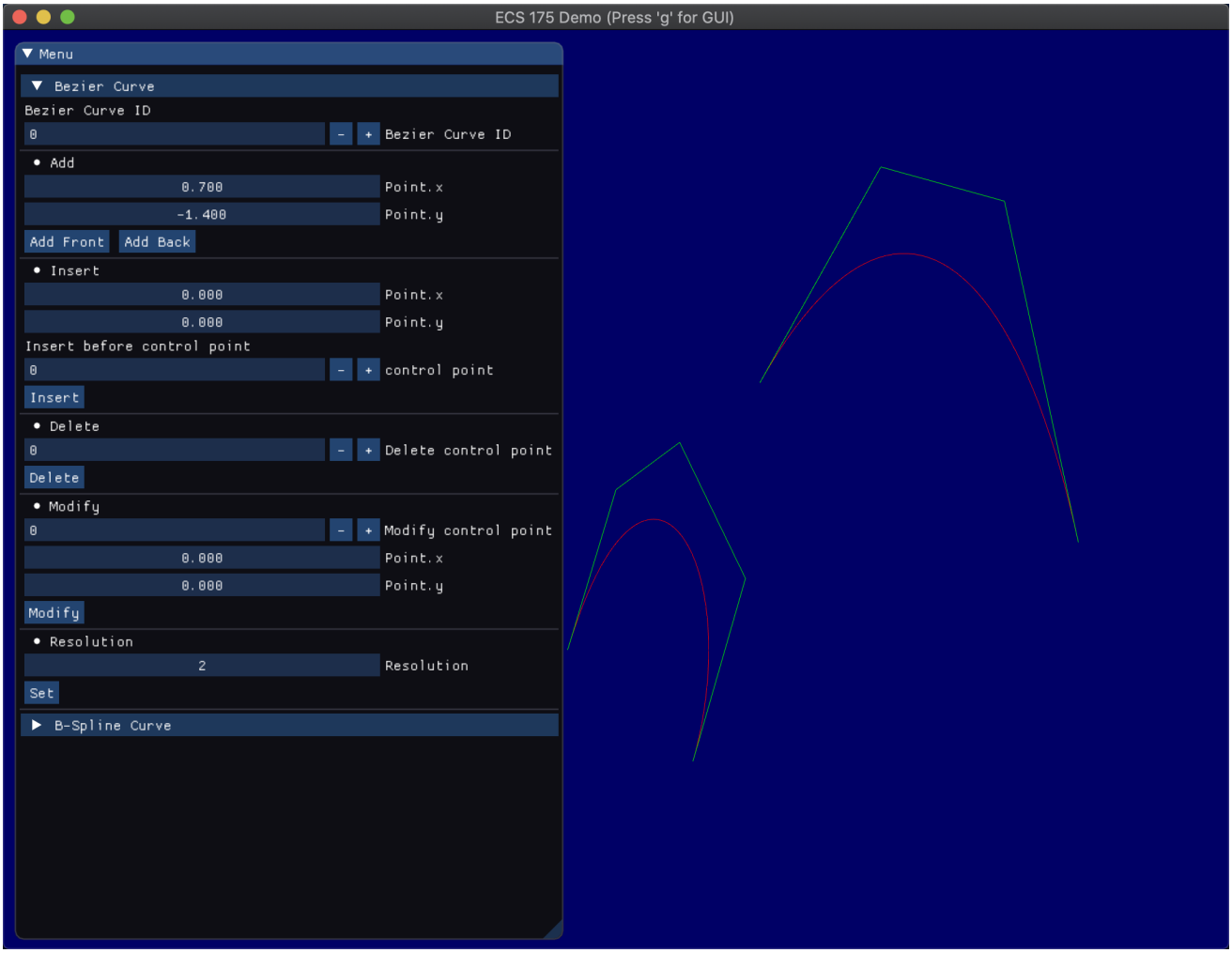



The following execution are all done on the bezier curve with ID 0, the one on the left.

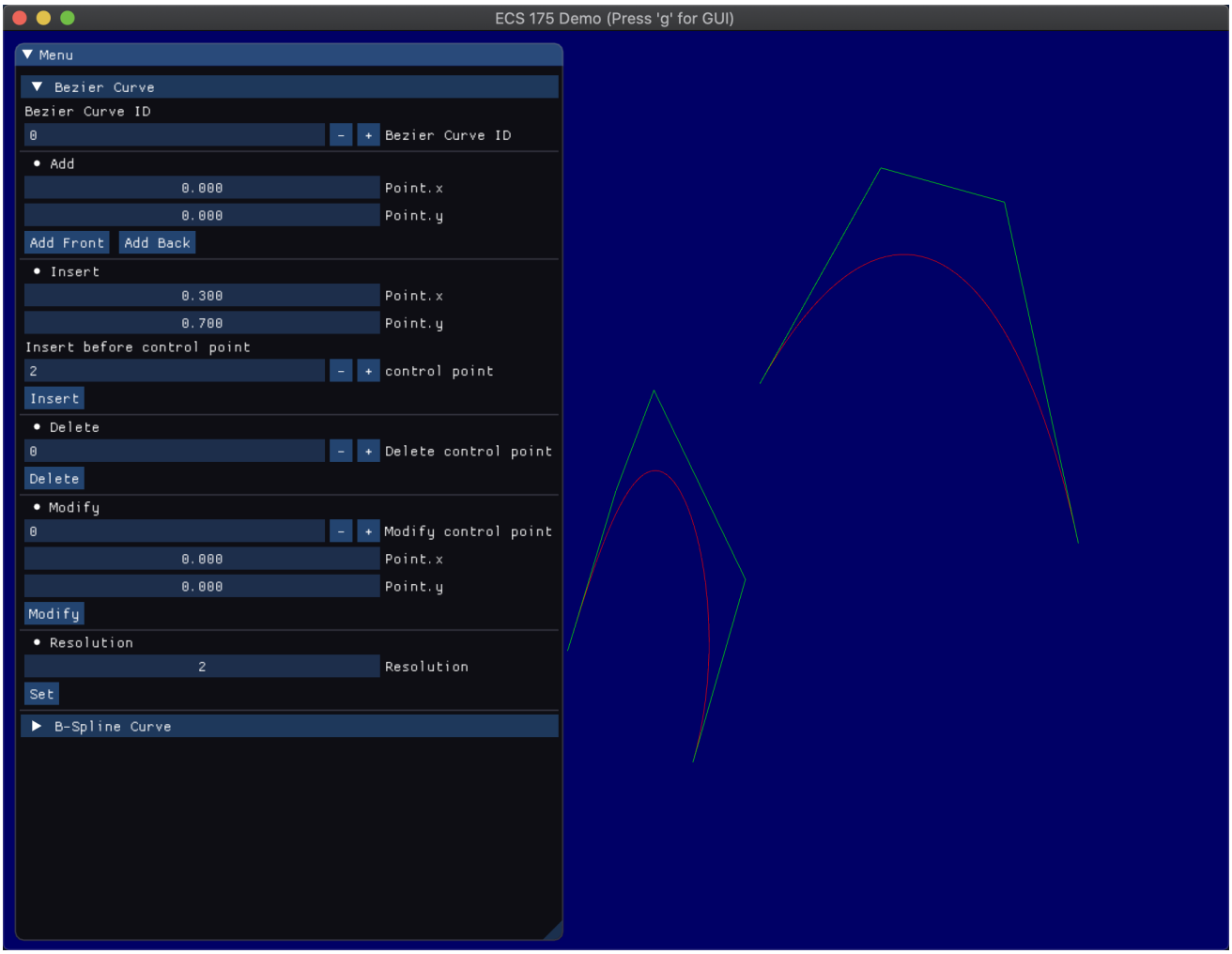
Add front with (-0.5, -1.4) will have the following effect.



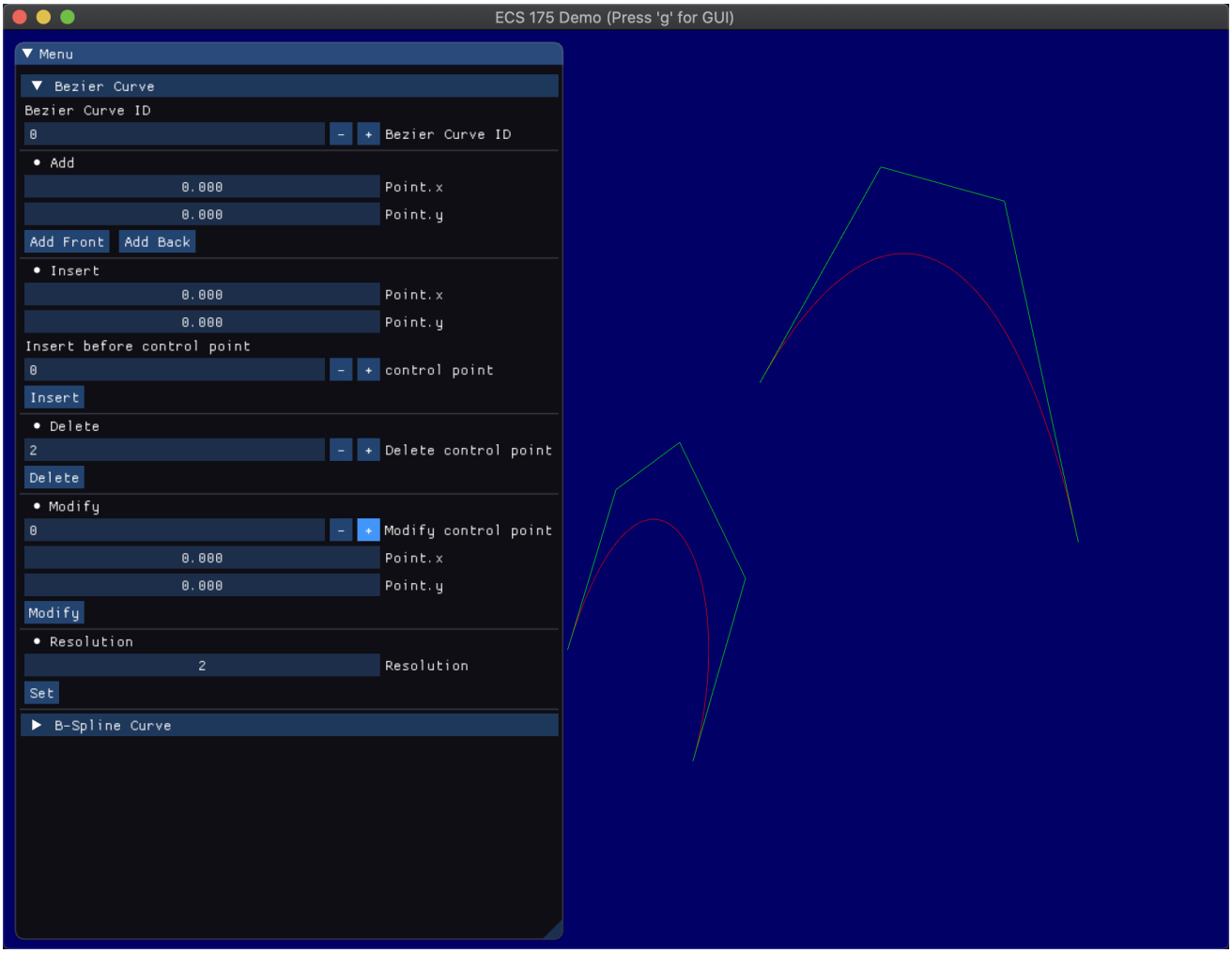
After that add front, add back with (0.7, -1.4) will have the following effect.



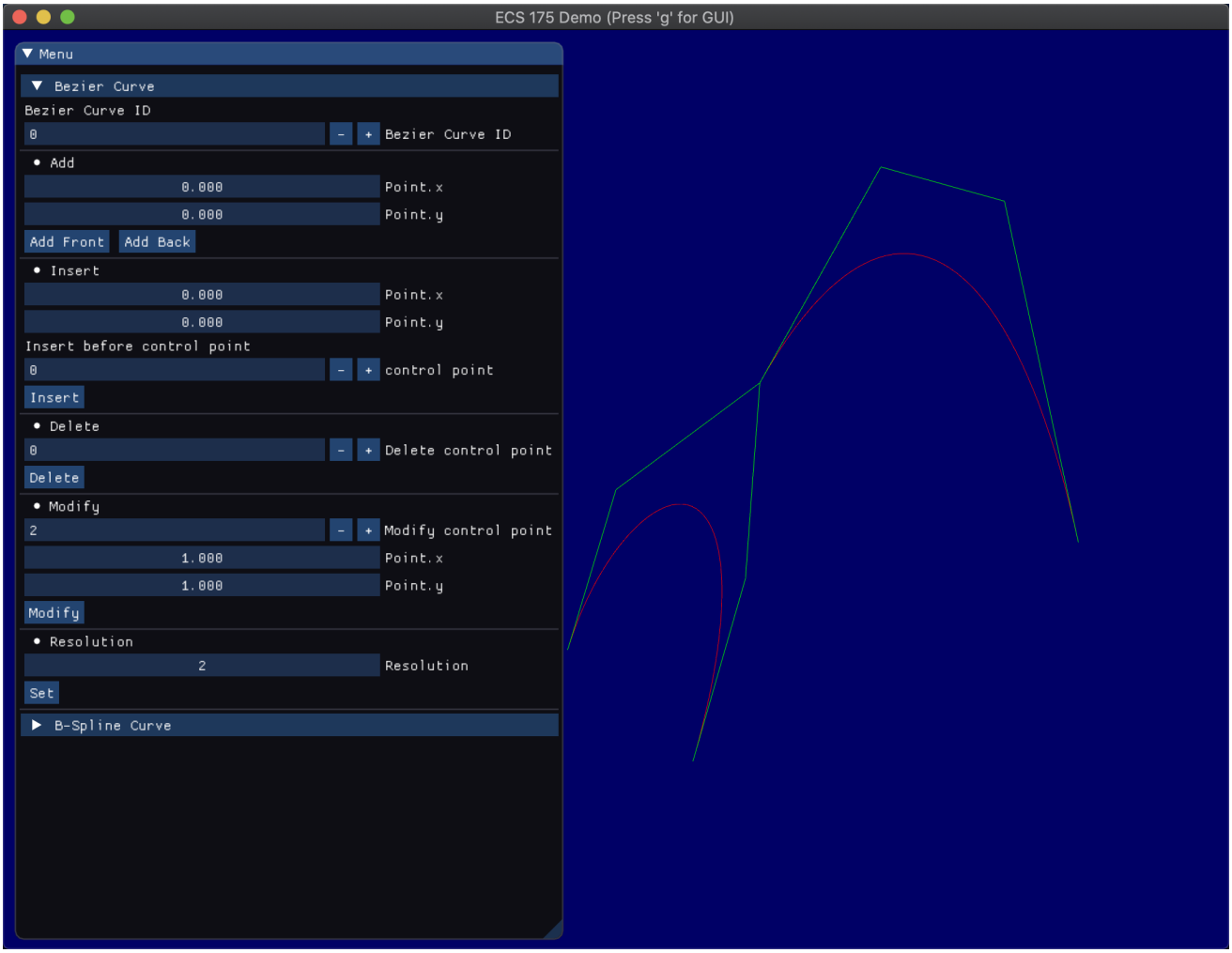
After that add back, insert with (0.3, 0.7) before the 2nd control point will have the following effect. Note that there are now 6 control points, but it's hard to see from the picture.



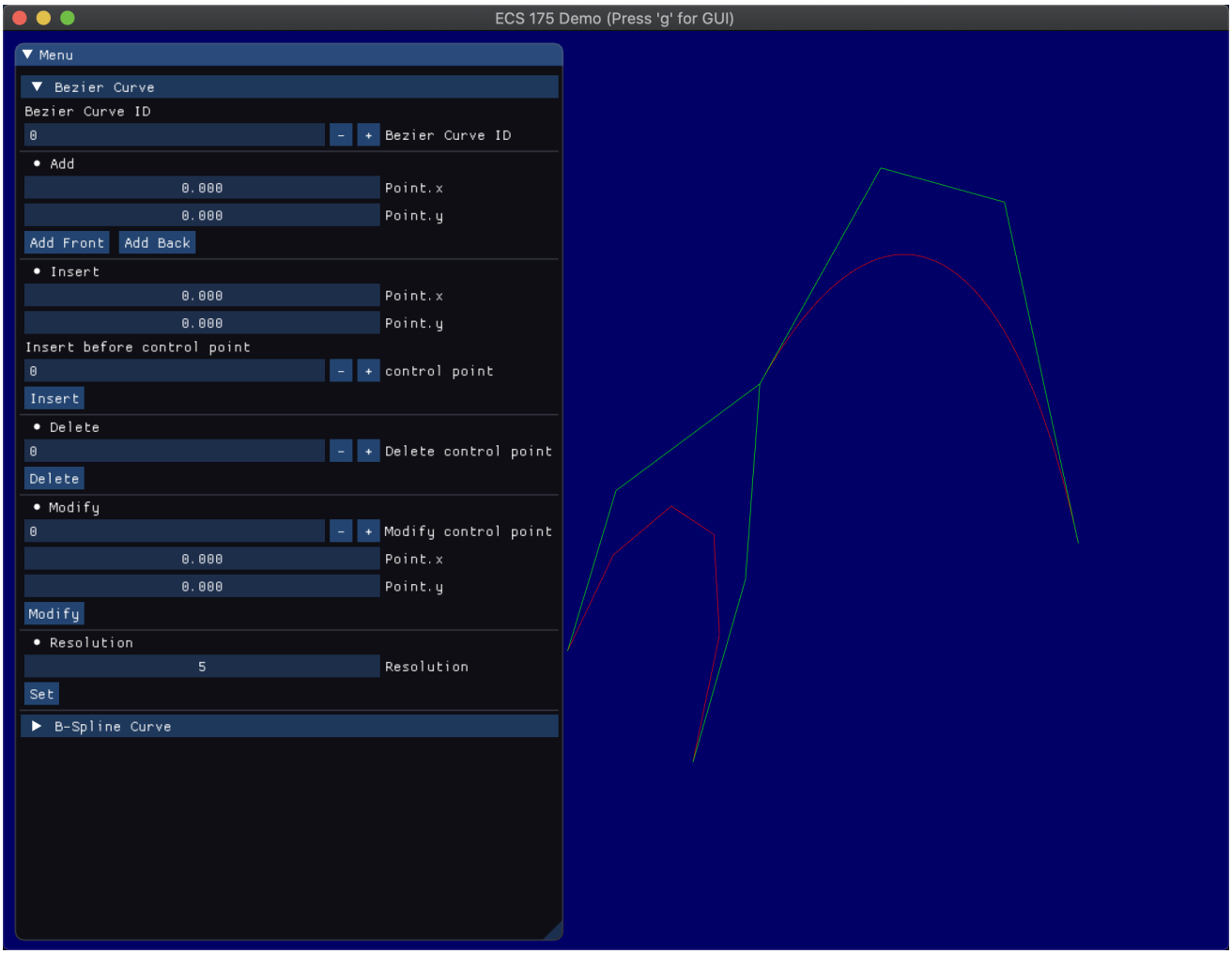
Now we delete the 2nd control point, and get the same image before we do the add back.



Now we modify the 2nd control point from (0.5, 0.5) to (1.0, 1.0) and we get this image.



The initial resolution is 50 for all bezier curve. Now we change it to 5 for the left bezier curve.



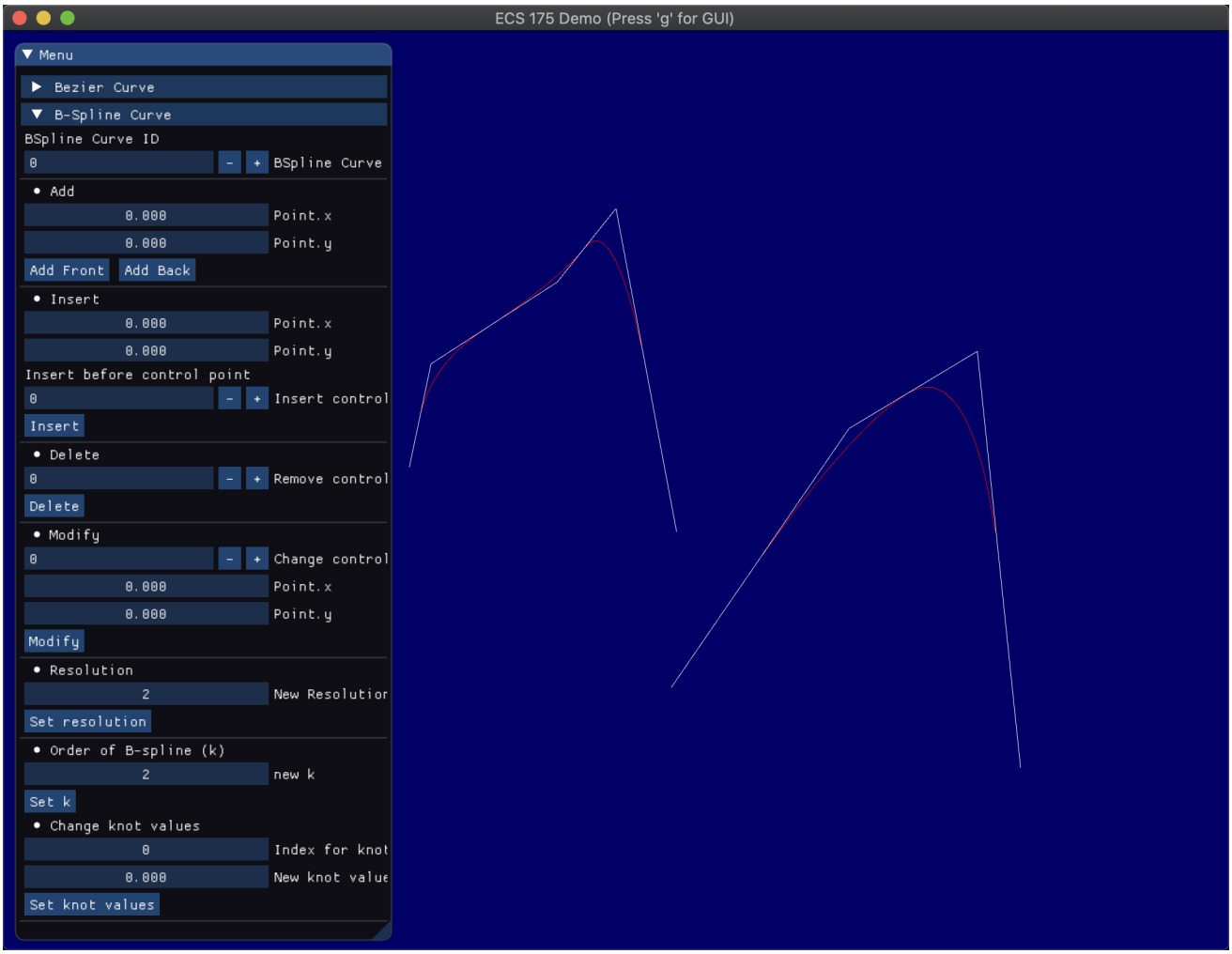

We can do the same operation to the other bezier curve by changing the ID from 0 to 1.

## Bspline

The corresponding execution for bspline curve editor can be found at line 195 in `gui.cpp` and the function is

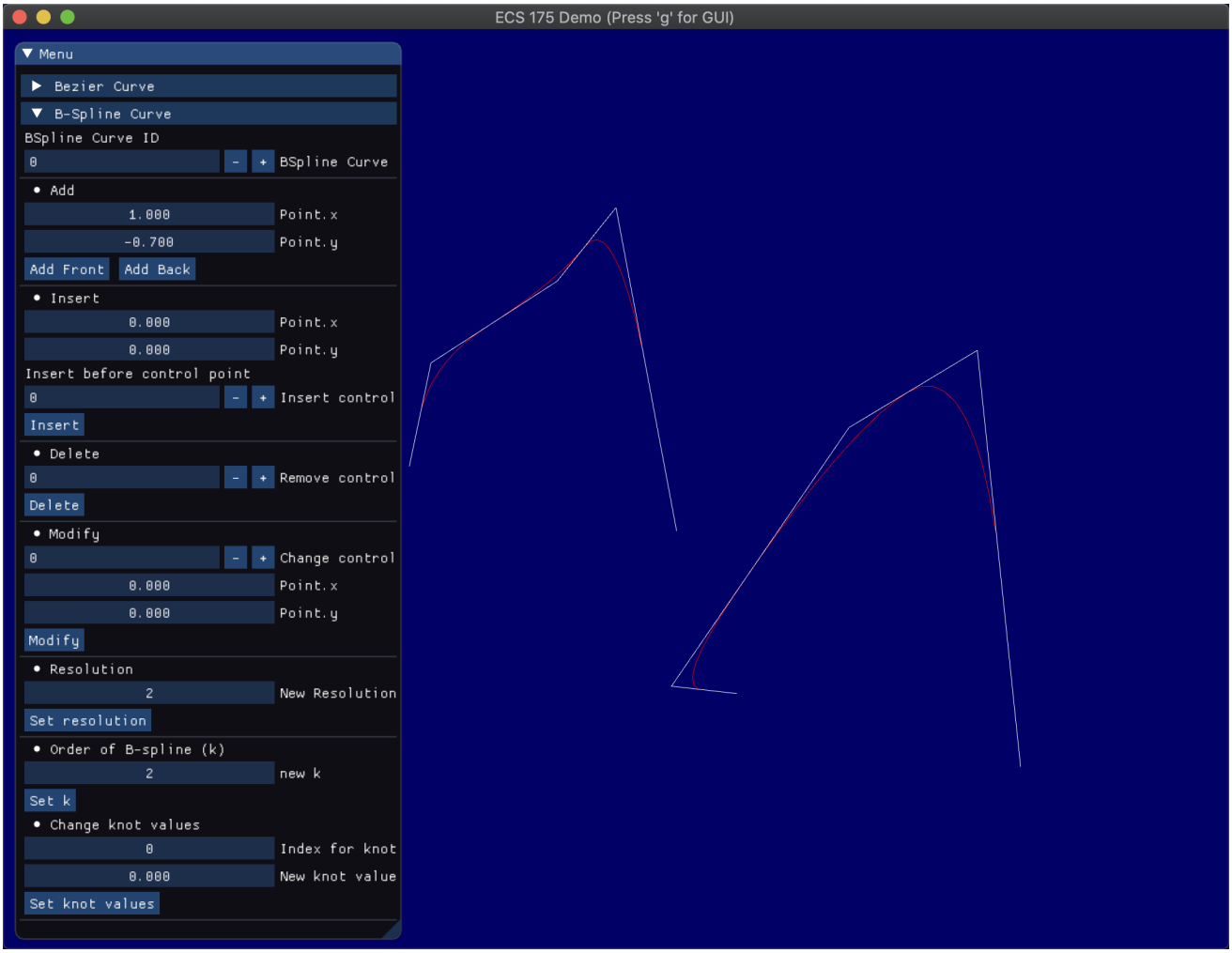
```
void BSplineMenu(bool* p_boolean, GLFWwindow* window);
```

In the following picture, there are two bspline curve generated from `2BsplineCurve.txt`. The bspline curve with ID 0 is the one on the right that has 4 control points, and the bspline curve with ID 1 is the one on the left that has 5 control points.

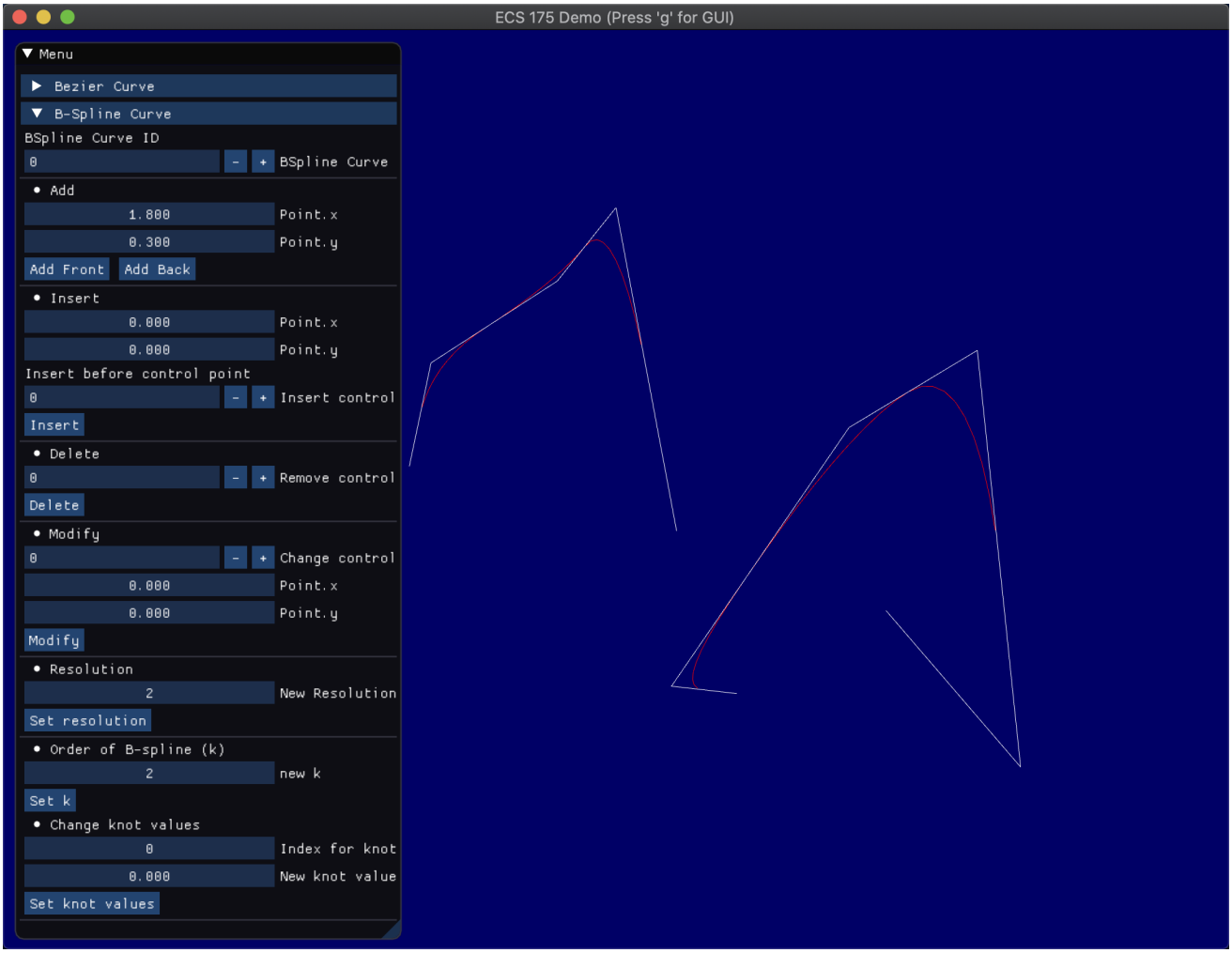



The following procedure are done on the bspline curve with ID 0, the one on the right.

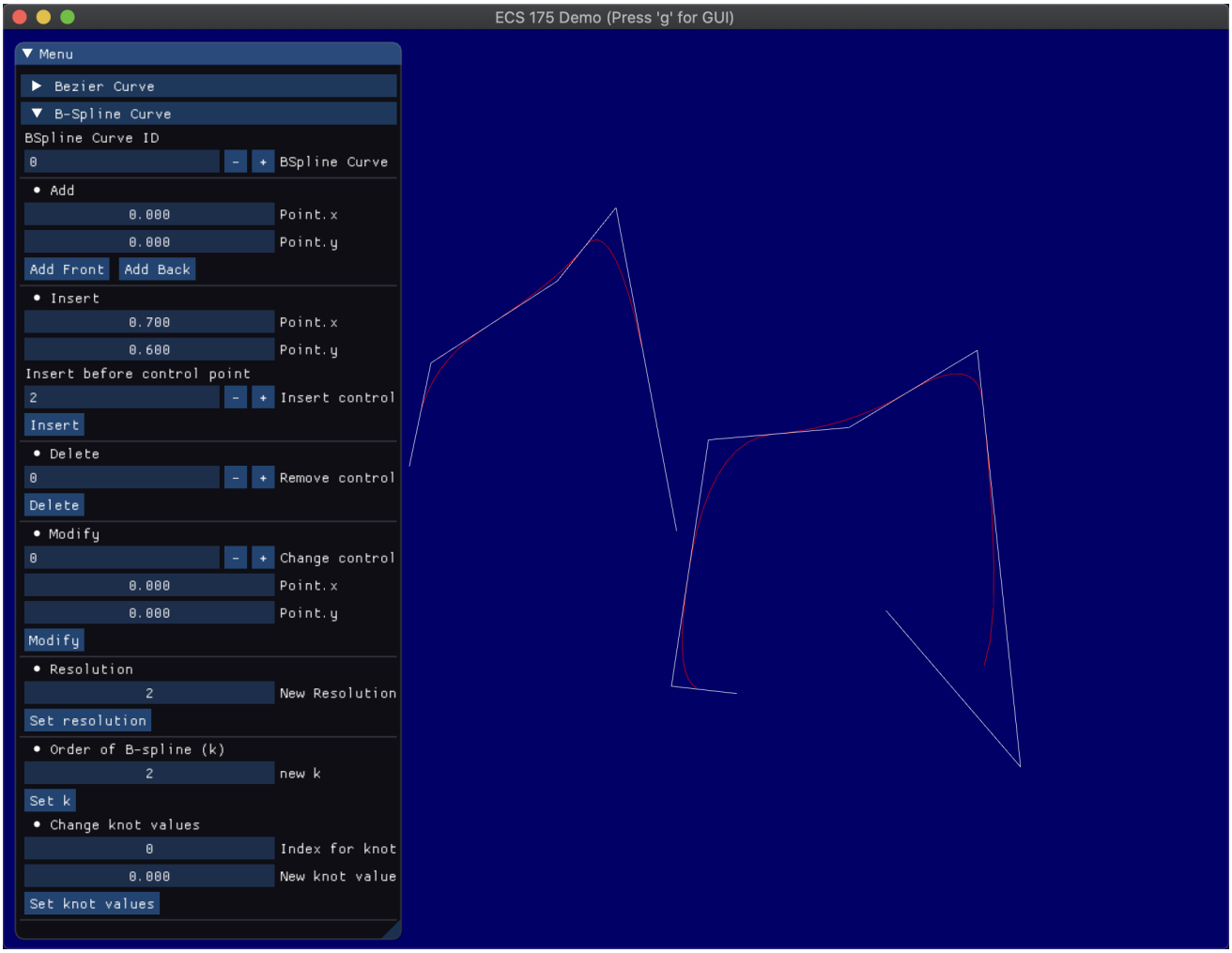
We add front (1.0, -0.7) to get



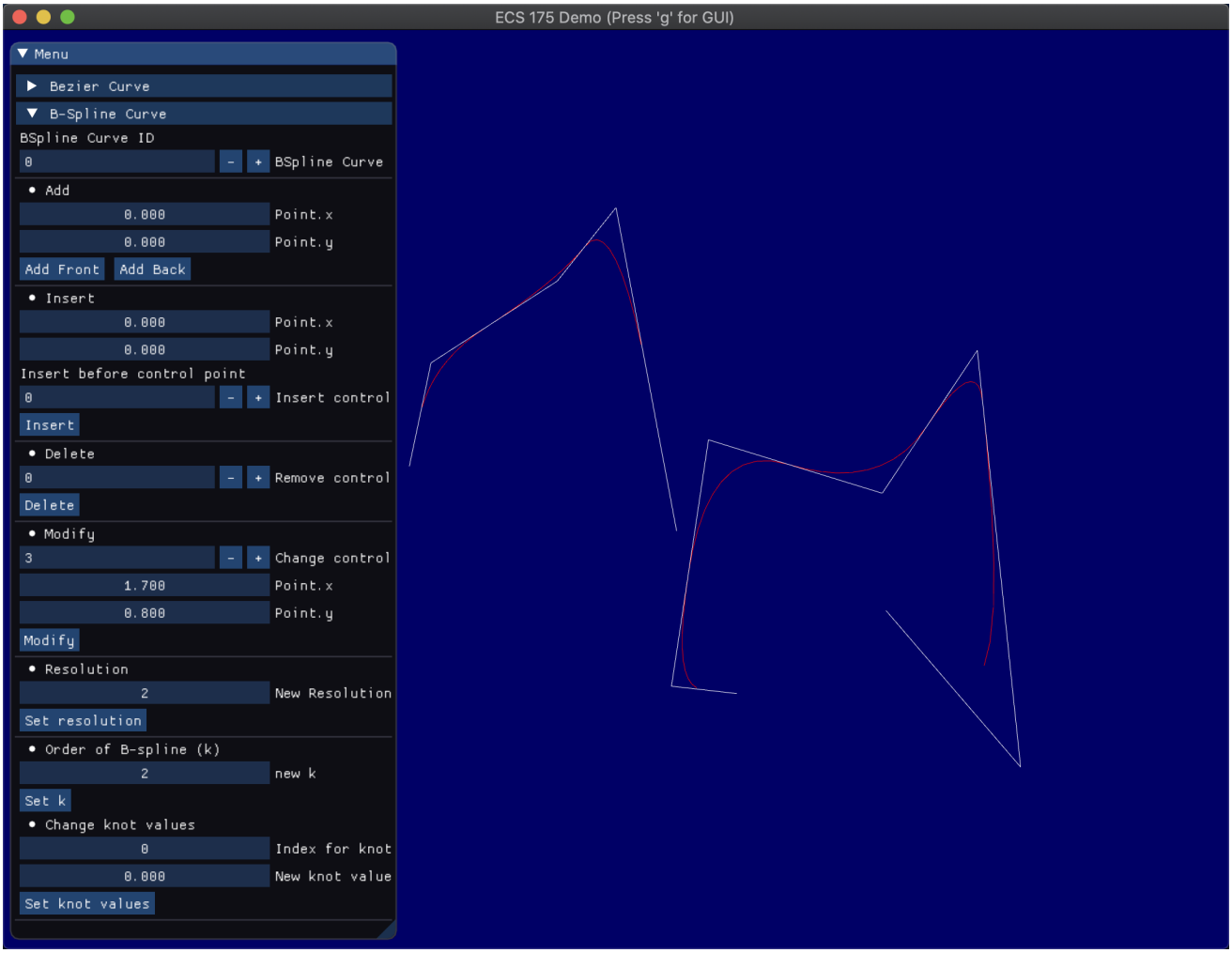
We add back (1.8, 0.3) to get



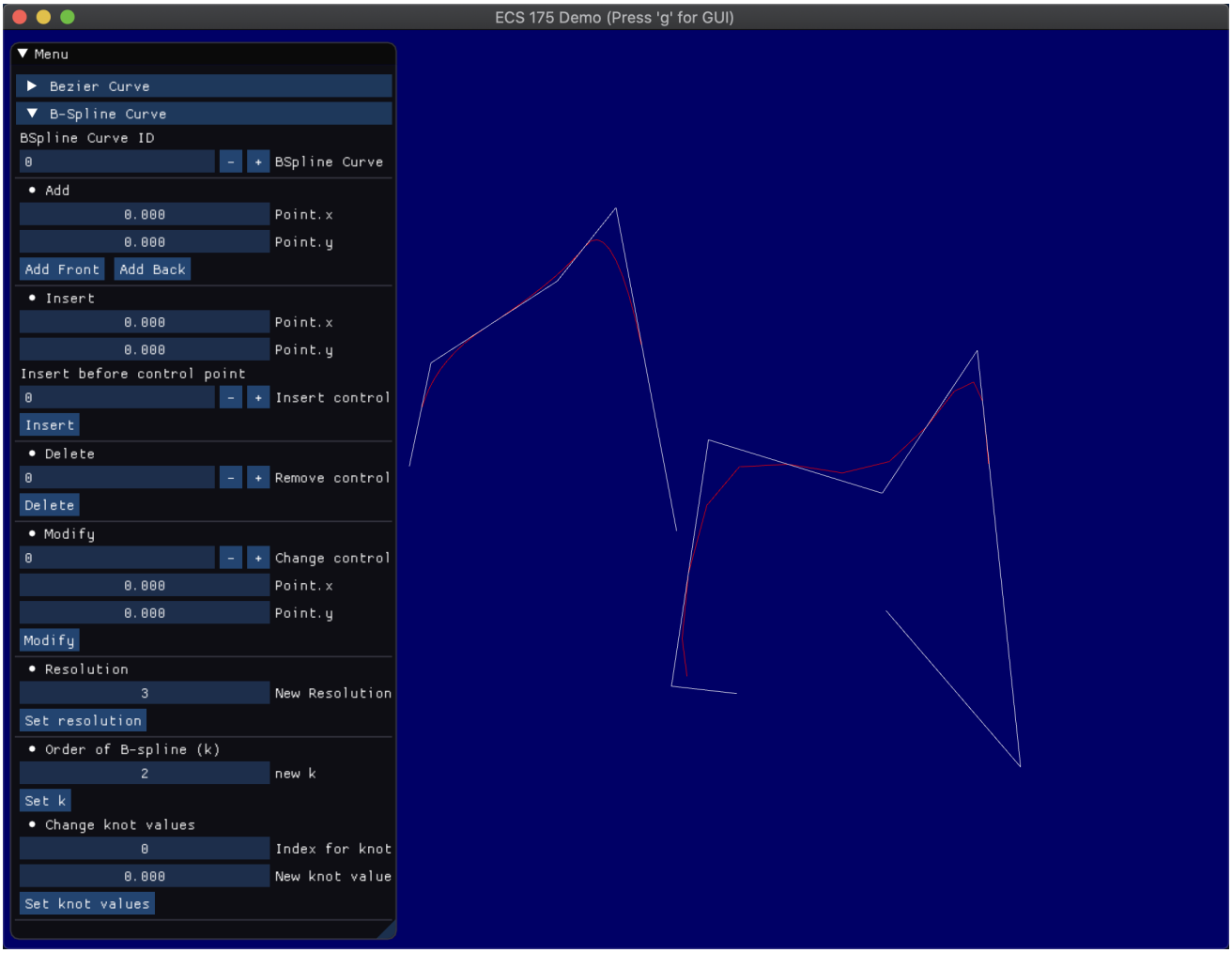
We insert (0.7, 0.6) before the 2nd control point (indexing from 0) to get



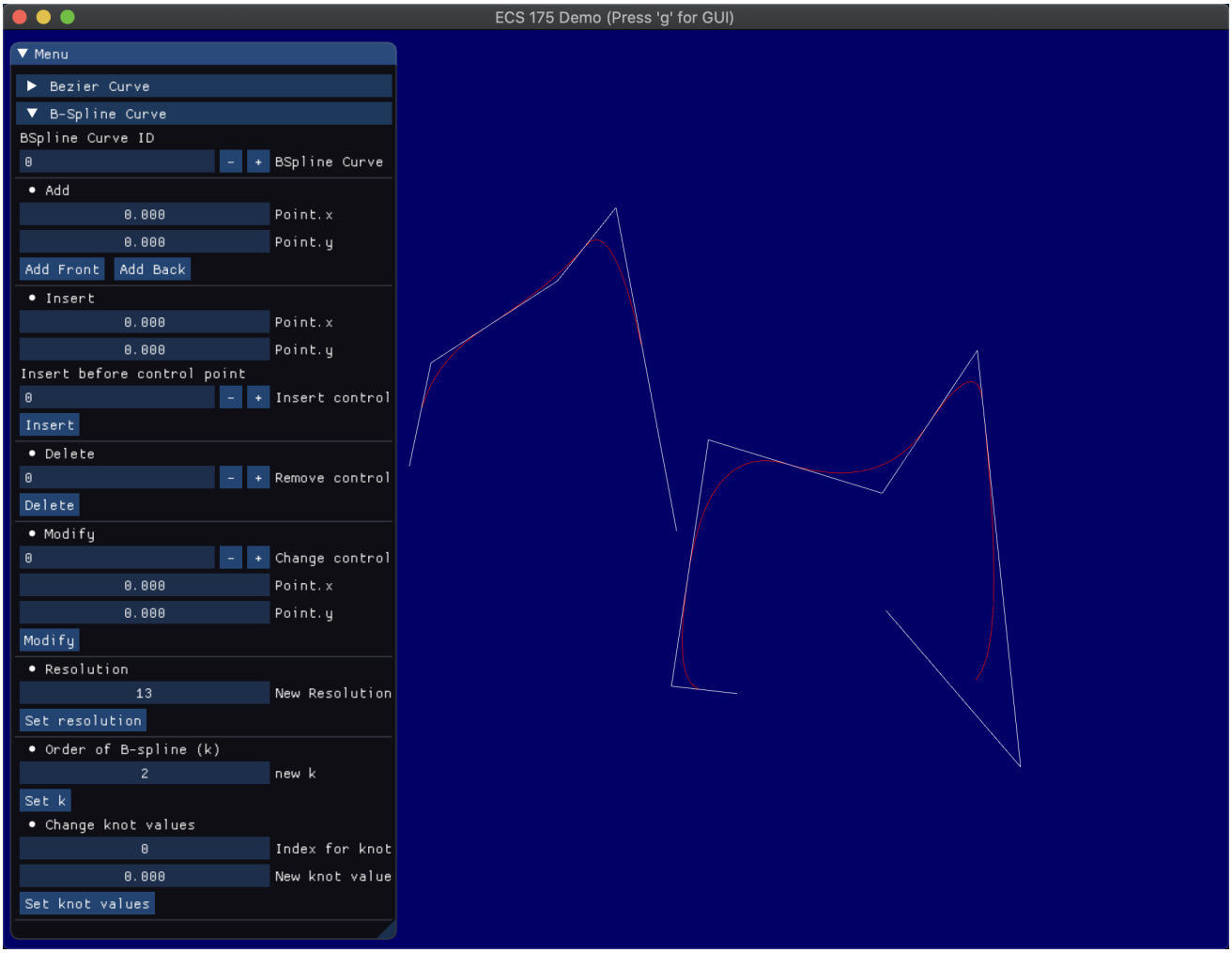
We modify 3rd control point to (1.7, 0.8)



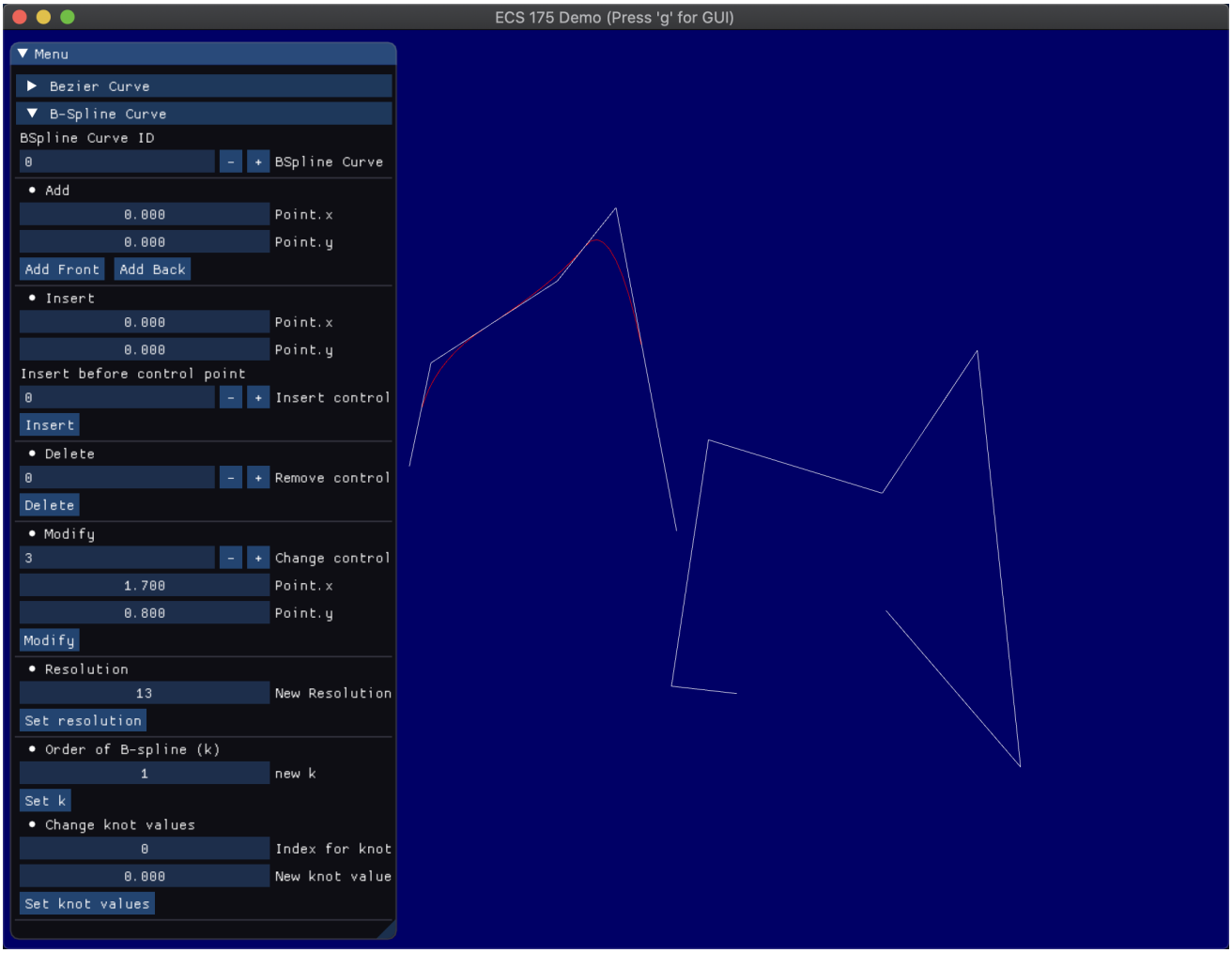
We modify resolution to 3. Note that the original resolution is 10.



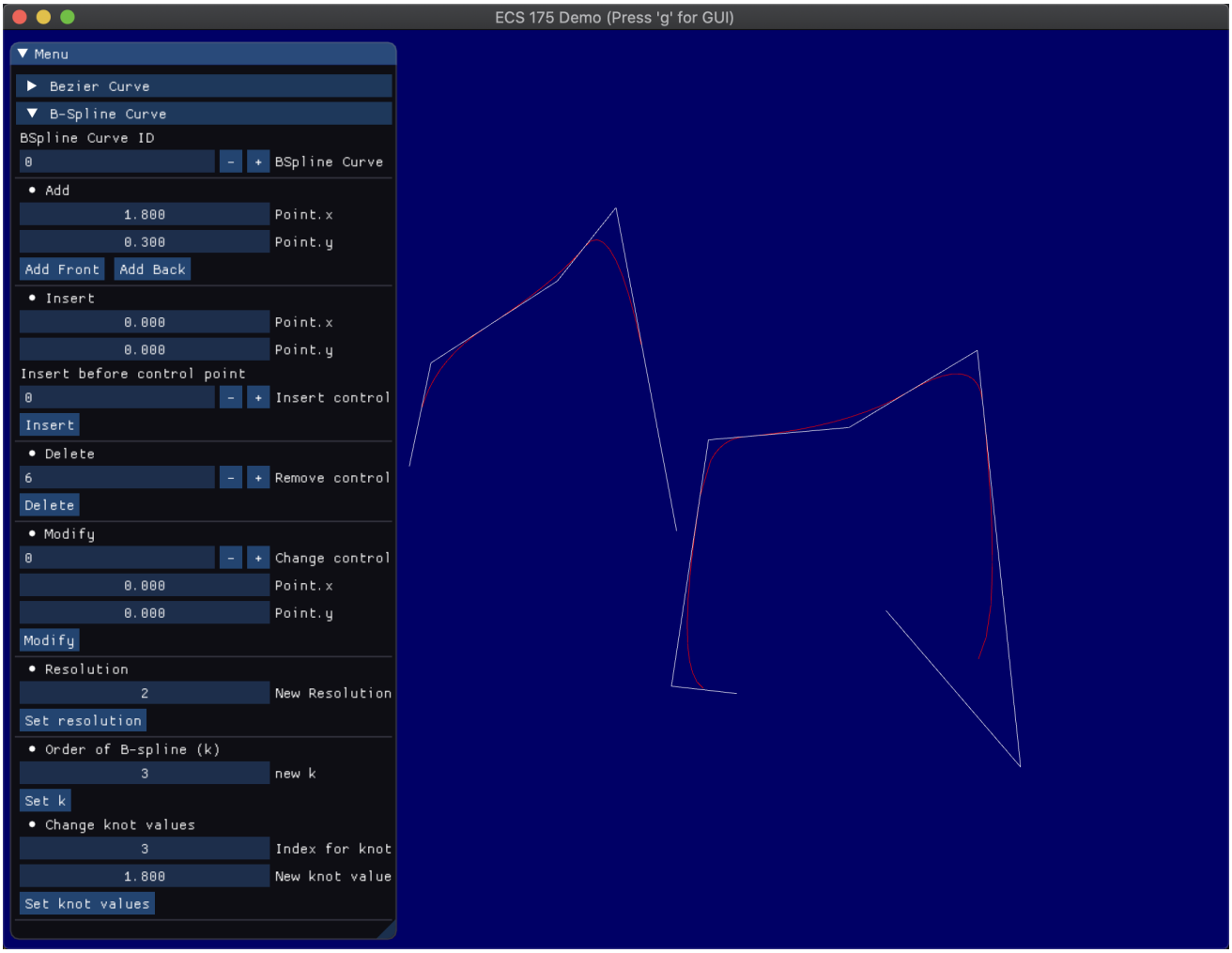
Now we modify resolution to 13 and get a smoother curve



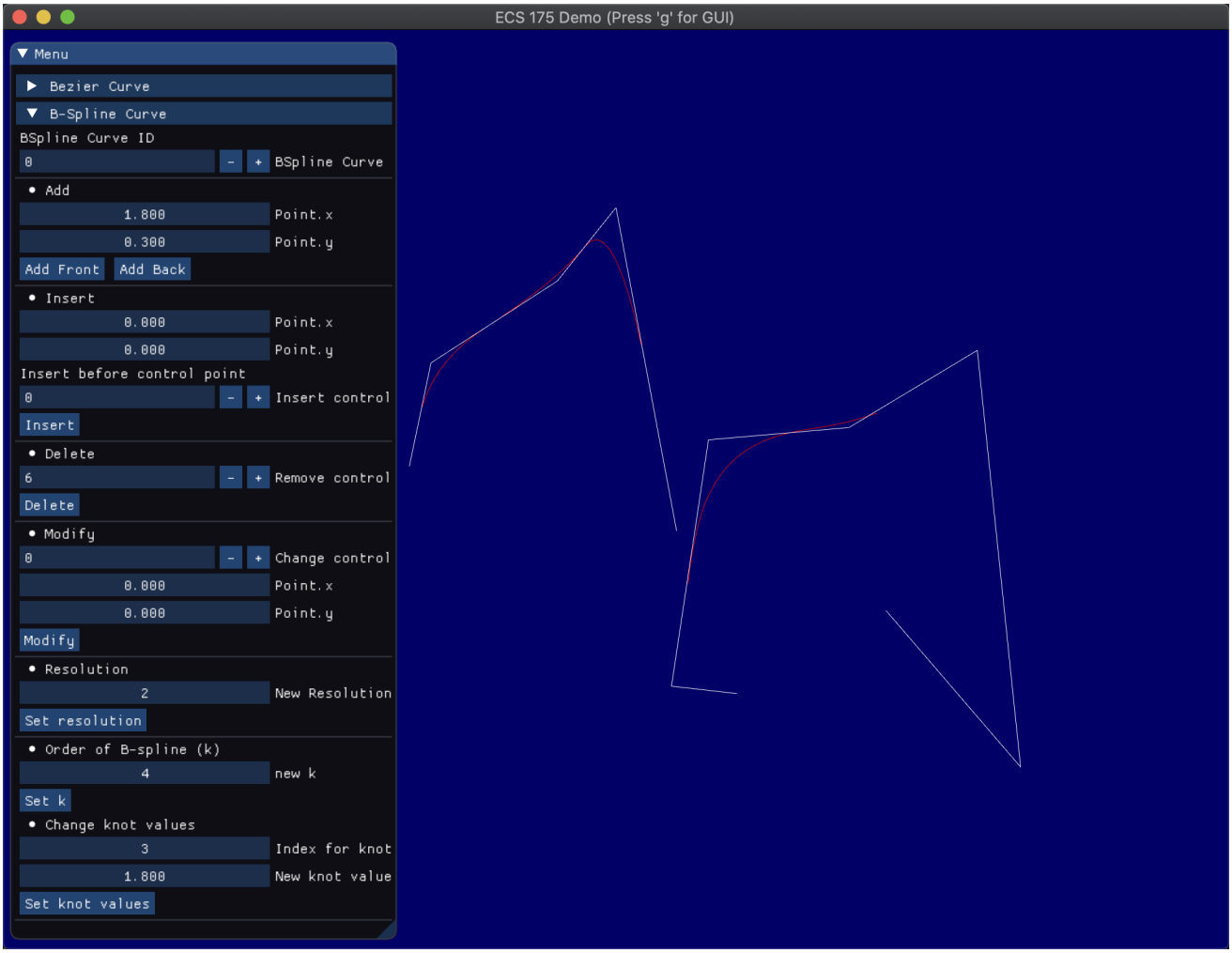
We change the order of k to 2 and get



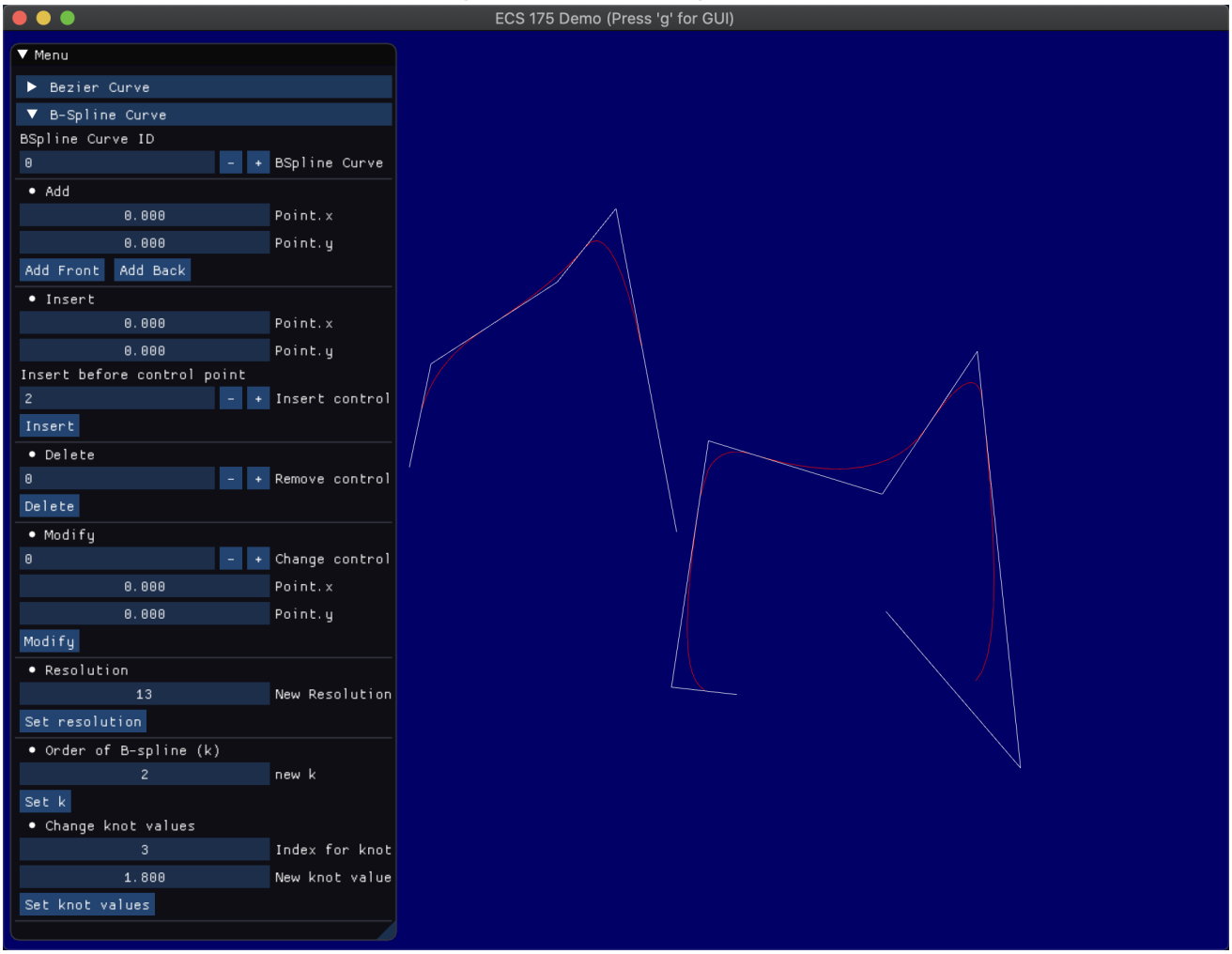
We change the order of k to 3 and get



We change the order of k to 4 and get



We then change k back to 3 and change the knot value for the 2nd control point to 1.8, and notice that the bspline curve gets close the edges around the 2nd control point.



Now we delete the last control point (the 6th one) and get