

Project 1 Report – Bezier Curve

D11315807

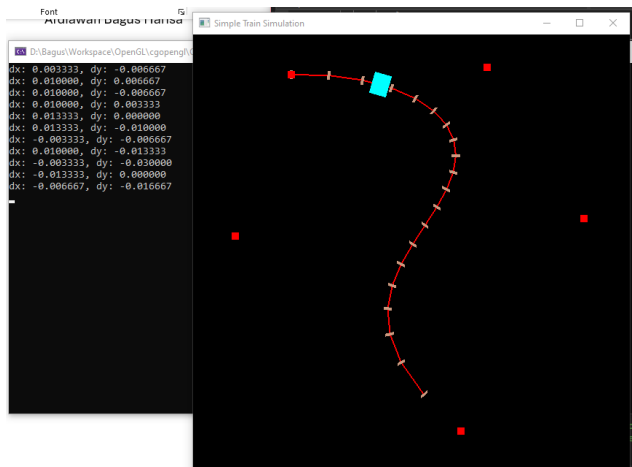
Ardiawan Bagus Harisa

Department of CSIE

Tasks:

1. Display four arbitrary control points.
2. Render Bezier curves.
3. Interactive control point adjustments.
4. Recursive Implementation.

1. Display four arbitrary control points.



Not only four, I let the user create control points anywhere on the frame using right click. The newly created control point is green-colored.

First, I created the point struct as the data type. Then, create list (vector) of control points. The variable *selectedPoint* define whether the specific control point is selected/ clicked.

```
OpenGL Train App.cpp  X
OpenGL Train  (Global Scope)
1  #include <GL/glew.h>
2  #include <GL/freeglut.h>
3  #include <vector>
4  #include <cmath>
5
6  struct Point {
7      float x, y;
8  };
9
10 std::vector<Point> controlPoints;
11 int selectedPoint = -1;
12 bool isTrainMoving = true;
13 float trainMovePos = 0.0f;
14
```

Secondly, we let the user arbitrarily create control points anywhere on the frame by using *right click*. It's done by saving the current location as a point $\{mx, my\}$, and then push it into the list of *controlPoints*.

```
164 void mouse(int button, int state, int x, int y) {
165     // Change the coordinate system from pixel to normalized range of -1 to 1
166     // Don't hardcode the window size. Use window width and height.
167     float mx = (float)x / (glutGet(GLUT_WINDOW_WIDTH) / 2) - 1;
168     float my = -(float)y / (glutGet(GLUT_WINDOW_HEIGHT) / 2) + 1;
169
170     if (button == GLUT_RIGHT_BUTTON && state == GLUT_DOWN) {
171         controlPoints.push_back({ mx, my });
172         selectedPoint = controlPoints.size() - 1;
173     }
```

Later on, on the display function call, we draw all *controlPoints* where its color is red if it's not selected, and green if it's selected. It is just to give the little UX decency for the user to know where has he clicked.

```
116 void drawControlPoints() {
117     for (int i = 0; i < controlPoints.size(); i++) {
118         if (i == selectedPoint) {
119             glColor3f(0.0f, 1.0f, 0.0f);
120         }
121         else {
122             glColor3f(1.0f, 0.0f, 0.0f);
123         }
124         drawPoint(controlPoints[i].x, controlPoints[i].y);
125     }
126 }
```

```
64
65 void drawPoint(float x, float y) {
66     glPointSize(10.0f);
67     glBegin(GL_POINTS);
68     glVertex2f(x, y);
69     glEnd();
70 }
```

2. Render Bezier curves.

To draw the curve, we need to define the points (or let's say segments) before we can really draw the curve line. In here, I implemented two methods on how to get the Bezier segments.

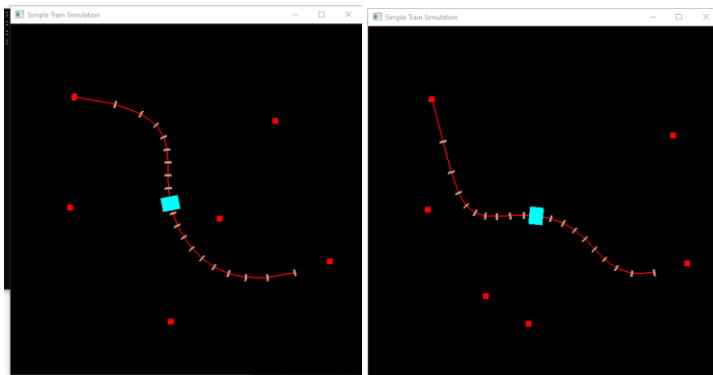
```
14 // Get the bezier points but using recursive function: de Casteljau's algorithm
15 Point deCasteljauBezierPoint(const std::vector<Point>& points, float t) {
16     std::vector<Point> temp = points;
17     int n = temp.size();
18     for (int r = 1; r < n; ++r) {
19         for (int i = 0; i < n - r; ++i) {
20             temp[i].x = (1 - t) * temp[i].x + t * temp[i + 1].x;
21             temp[i].y = (1 - t) * temp[i].y + t * temp[i + 1].y;
22         }
23     }
24     return temp[0];
25 }
26
27
28
29 Point getBezierPoints(float t) {
30     // Non-Recursive
31     /*int n = controlPoints.size() - 1;
32     Point p = { 0, 0 };
33
34     for (int i = 0; i <= n; i++) {
35         float binomial = tgamma(n + 1) / (tgamma(i + 1) * tgamma(n - i + 1)); // This is the binomial coefficient
36         float blend = binomial * pow(t, i) * pow(1 - t, n - i); // This is the blending function
37         p.x += controlPoints[i].x * blend;
38         p.y += controlPoints[i].y * blend;
39     }
40     return p;
41     */
42
43     // Recursive
44     return deCasteljauBezierPoint(controlPoints, t);
45 }
46 }
```

As you can see from the code, I applied recursive and non-recursive methods. However, I finally just use the recursive method from de Casteljau's algorithm, and comment the non-recursive one.

In the recursive method, first I prepare the temporary list of points to hold the point list argument. Then, do the nested iteration to calculate the segment. For each of control point, and its segment, we calculate the approximate position (x, y) by recursively calculate the last position to the initial position. Well, it is really just a recursive basic. In the non-recursive method, we do the segment calculation from beginning to the end. Using binomial factor.

3. Interactive control point adjustments.

As I mentioned earlier, the user can edit and add the control points as many as he wants. User can edit the curve by left clicking a control point.



From the code below, when the user click a point near where the control point is located at, it will change the state (*selectedPoint*) to the index of *controlPoints[i]*. Otherwise, the controlPoints will all have -1 value on *selectedPoint*.

```
164 void mouse(int button, int state, int x, int y) {
165     // Change the coordinate system from pixel to normalized range of -1 to 1
166     // Don't hardcode the window size. Use window width and height.
167     float mx = (float)x / (glutGet(GLUT_WINDOW_WIDTH) / 2) - 1;
168     float my = -(float)y / (glutGet(GLUT_WINDOW_HEIGHT) / 2) + 1;
169
170     if (button == GLUT_RIGHT_BUTTON && state == GLUT_DOWN) { ... }
171     else if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN) {
172         for (int i = 0; i < controlPoints.size(); i++) {
173             float dx = controlPoints[i].x - mx;
174             float dy = controlPoints[i].y - my;
175             if (fabs(dx) < 0.05f && fabs(dy) < 0.05f) { // This is not pixel unit. This is in the normalized range of -1 to 1.
176                 // Debug the dx and dy
177                 printf("dx: %f, dy: %f\n", dx, dy);
178                 selectedPoint = i;
179                 break;
180             }
181         }
182     }
183     else if (button == GLUT_LEFT_BUTTON && state == GLUT_UP) {
184         selectedPoint = -1;
185     }
186     glutPostRedisplay();
187 }
```

Finally, we can move the selected point according to the mouse x,y position.

```

193 void motion(int x, int y) {
194     if (selectedPoint != -1) {
195         // Change the coordinate system from pixel to normalized range of -1 to 1
196         // Don't hardcode the window size. Use window width and height.
197         controlPoints[selectedPoint].x = (float)x / (glutGet(GLUT_WINDOW_WIDTH) / 2) - 1;
198         controlPoints[selectedPoint].y = -(float)y / (glutGet(GLUT_WINDOW_WIDTH) / 2) + 1;
199         glutPostRedisplay();
200     }
201 }

```

4. Recursive Implementation.

Explained earlier.

5. Other

In this project, I created a simple 2D-train simulator. First, I drew the train, built up from a simple rectangle. Located initially in the first segment of the Bezier curve.

```

128 void drawTrain() {
129     if (controlPoints.size() < 2) {
130         return;
131     }
132     Point trainPos = getBezierPoints(trainMovePos);
133     Point trainTangent = getTangentPoints(trainMovePos);
134
135     float length = sqrt(pow(trainTangent.x, 2) + pow(trainTangent.y, 2));
136     if (length == 0) {
137         return;
138     }
139     trainTangent.x = trainTangent.x / length;
140     trainTangent.y = trainTangent.y / length;
141     Point normal = { -trainTangent.y, trainTangent.x };
142
143     glColor3f(0.0f, 1.0f, 1.0f);
144     glBegin(GL_QUADS);
145     glVertex2f(trainPos.x - normal.x * 0.05f - trainTangent.x * 0.04f, trainPos.y - normal.y * 0.05f - trainTangent.y * 0.04f);
146     glVertex2f(trainPos.x + normal.x * 0.05f - trainTangent.x * 0.04f, trainPos.y + normal.y * 0.05f - trainTangent.y * 0.04f);
147     glVertex2f(trainPos.x + normal.x * 0.05f + trainTangent.x * 0.04f, trainPos.y + normal.y * 0.05f + trainTangent.y * 0.04f);
148     glVertex2f(trainPos.x - normal.x * 0.05f + trainTangent.x * 0.04f, trainPos.y - normal.y * 0.05f + trainTangent.y * 0.04f);
149     glEnd();
150
151     // Update the train position
152 }
153
154

```

Then, update the position of the train with specific distance (0.005 point) in a loop.

```

154 void updateTrain(int value) {
155     if (isTrainMoving) {
156         trainMovePos += 0.005f;
157         if (trainMovePos > 1.0f) trainMovePos = 0.0f;
158         glutPostRedisplay();
159         glutTimerFunc(100, updateTrain, value);
160     }
161 }
162

```

```

89 void drawRails() {
90     if (controlPoints.size() < 2) {
91         return;
92     }
93
94     glColor3f(0.8f, 0.6f, 0.5f);    // Wooden color for sleepers
95     glLineWidth(4.0f);
96     glBegin(GL_LINES);
97     for (float t = 0; t <= 1; t += 0.05f) {
98         Point p = getBezierPoints(t);
99         Point tangent = getTangentPoints(t);
100
101         // Get the perpendicular vector by normalizing the tangent vector and then rotating it by 90 degrees
102         float length = sqrt(pow(tangent.x, 2) + pow(tangent.y, 2));
103         if (length == 0) {
104             continue;
105         }
106         tangent.x = tangent.x / length;
107         tangent.y = tangent.y / length;
108         Point normal = { -tangent.y, tangent.x };
109
110         glVertex2f(p.x - normal.x * 0.02f, p.y - normal.y * 0.02f);
111         glVertex2f(p.x + normal.x * 0.02f, p.y + normal.y * 0.02f);
112     }
113     glEnd();
114 }
115

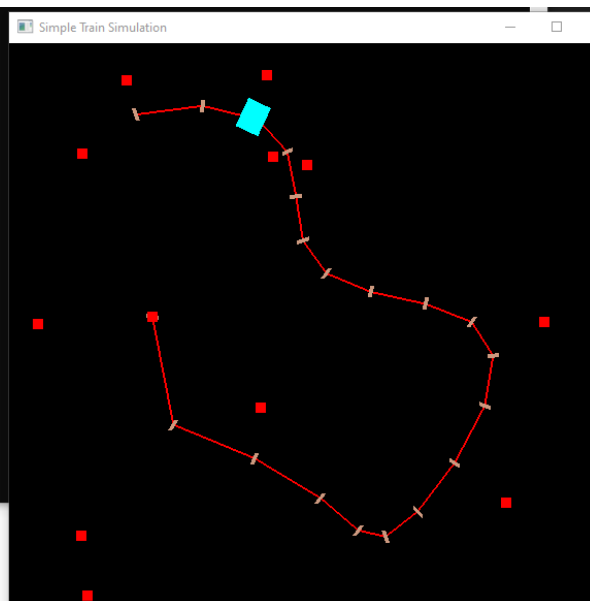
```

Finally, draw the chunk of rails wood perpendicular to the curve line of Bezier. Although, the implementation of line segmentation is still incorrect, because the length of each segment is differed among the control points.

```

dx: -0.003333, dy: -0.003333
dx: 0.020000, dy: -0.006667
dx: 0.000000, dy: -0.016667
dx: 0.006667, dy: -0.006667
dx: 0.020000, dy: -0.016667
dx: -0.010000, dy: -0.003333
dx: 0.013333, dy: -0.006667
dx: 0.000000, dy: 0.000000

```



Source code:

<https://github.com/ardiawanbagusharisa/cgopengl/tree/main/OpenGL%20Train%20Simulation>

```

#include <GL/glew.h>
#include <GL/freeglut.h>
#include <vector>
#include <cmath>

struct Point {
    float x, y;
};

std::vector<Point> controlPoints;
int selectedPoint = -1;
bool isTrainMoving = true;
float trainMovePos = 0.0f;

// Get the bezier points but using recursive function: de Casteljau's algorithm
Point deCasteljauBezierPoint(const std::vector<Point>& points, float t) {
    std::vector<Point> temp = points;
    int n = temp.size();
    for (int r = 1; r < n; ++r) {
        for (int i = 0; i < n - r; ++i) {
            temp[i].x = (1 - t) * temp[i].x + t * temp[i + 1].x;
            temp[i].y = (1 - t) * temp[i].y + t * temp[i + 1].y;
        }
    }
    return temp[0];
}

Point getBezierPoints(float t) {
    // Non-Recursive
    int n = controlPoints.size() - 1;
    Point p = { 0, 0 };

    for (int i = 0; i <= n; i++) {
        float binomial = tgamma(n + 1) / (tgamma(i + 1) * tgamma(n - i + 1));
        // This is the binomial coefficient
        float blend = binomial * pow(t, i) * pow(1 - t, n - i);
        // This is the blending function
        p.x += controlPoints[i].x * blend;
        p.y += controlPoints[i].y * blend;
    }

    return p;
}

// Recursive
return deCasteljauBezierPoint(controlPoints, t);
}

Point getTangentPoints(float t) {
    int n = controlPoints.size() - 1;
    if (n < 1) {
        return { 0, 0 };
    }

    Point tangent = { 0, 0 };
    for (int i = 0; i < n; i++) {
        float binomial = tgamma(n) / (tgamma(i + 1) * tgamma(n - i));
        float blend = binomial * pow(t, i) * pow(1 - t, n - 1 - i);
        tangent.x += (controlPoints[i + 1].x - controlPoints[i].x) * blend * n;
        tangent.y += (controlPoints[i + 1].y - controlPoints[i].y) * blend * n;
    }

    return tangent;
}

void drawPoint(float x, float y) {
    glPointSize(10.0f);
    glBegin(GL_POINTS);
    glVertex2f(x, y);
    glEnd();
}

```

```

}

void drawBezierCurve() {
    if (controlPoints.size() < 2) {
        return;
    }

    glLineWidth(2.0f);
    glBegin(GL_LINE_STRIP);
    for (float t = 0; t <= 1; t += 0.05f) {
        Point p = getBezierPoints(t);
        glVertex2f(p.x, p.y);
    }
    // Connect to the first point
    //Point p = getBezierPoints(0);
    //glVertex2f(p.x, p.y);
    glEnd();
}

void drawRails() {
    if (controlPoints.size() < 2) {
        return;
    }

    glColor3f(0.8f, 0.6f, 0.5f);          // Wooden color for sleepers
    glLineWidth(4.0f);
    glBegin(GL_LINES);
    for (float t = 0; t <= 1; t += 0.05f) {
        Point p = getBezierPoints(t);
        Point tangent = getTangentPoints(t);

        // Get the perpendicular vector by normalizing the tangent vector and then rotating
        it by 90 degrees
        float length = sqrt(pow(tangent.x, 2) + pow(tangent.y, 2));
        if (length == 0) {
            continue;
        }
        tangent.x = tangent.x / length;
        tangent.y = tangent.y / length;
        Point normal = { -tangent.y, tangent.x };

        glVertex2f(p.x - normal.x * 0.02f, p.y - normal.y * 0.02f);
        glVertex2f(p.x + normal.x * 0.02f, p.y + normal.y * 0.02f);
    }
    glEnd();
}

void drawControlPoints() {
    for (int i = 0; i < controlPoints.size(); i++) {
        if (i == selectedPoint) {
            glColor3f(0.0f, 1.0f, 0.0f);
        }
        else {
            glColor3f(1.0f, 0.0f, 0.0f);
        }
        drawPoint(controlPoints[i].x, controlPoints[i].y);
    }
}

void drawTrain() {
    if (controlPoints.size() < 2) {
        return;
    }

    Point trainPos = getBezierPoints(trainMovePos);
    Point trainTangent = getTangentPoints(trainMovePos);

    float length = sqrt(pow(trainTangent.x, 2) + pow(trainTangent.y, 2));
    if (length == 0) {
        return;
    }
}

```



```

        trainTangent.x = trainTangent.x / length;
        trainTangent.y = trainTangent.y / length;
        Point normal = { -trainTangent.y, trainTangent.x };

        glColor3f(0.0f, 1.0f, 1.0f);
        glBegin(GL_QUADS);
        glVertex2f(trainPos.x - normal.x * 0.05f - trainTangent.x * 0.04f, trainPos.y - normal.y *
0.05f - trainTangent.y * 0.04f);
        glVertex2f(trainPos.x + normal.x * 0.05f - trainTangent.x * 0.04f, trainPos.y + normal.y *
0.05f - trainTangent.y * 0.04f);
        glVertex2f(trainPos.x + normal.x * 0.05f + trainTangent.x * 0.04f, trainPos.y + normal.y *
0.05f + trainTangent.y * 0.04f);
        glVertex2f(trainPos.x - normal.x * 0.05f + trainTangent.x * 0.04f, trainPos.y - normal.y *
0.05f + trainTangent.y * 0.04f);
        glEnd();

        // Update the train position
    }

void updateTrain(int value) {
    if (isTrainMoving) {
        trainMovePos += 0.005f;
        if (trainMovePos > 1.0f) trainMovePos = 0.0f;
        glutPostRedisplay();
        glutTimerFunc(100, updateTrain, value);
    }
}

void mouse(int button, int state, int x, int y) {
    // Change the coordinate system from pixel to normalized range of -1 to 1
    // Don't hardcode the window size. Use window width and height.
    float mx = (float)x / (glutGet(GLUT_WINDOW_WIDTH) / 2) - 1;
    float my = -(float)y / (glutGet(GLUT_WINDOW_HEIGHT) / 2) + 1;

    if (button == GLUT_RIGHT_BUTTON && state == GLUT_DOWN) {
        controlPoints.push_back({ mx, my });
        selectedPoint = controlPoints.size() - 1;
    }
    else if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN) {
        for (int i = 0; i < controlPoints.size(); i++) {
            float dx = controlPoints[i].x - mx;
            float dy = controlPoints[i].y - my;
            if (fabs(dx) < 0.05f && fabs(dy) < 0.05f) { // This is not pixel unit. This
is in the normalized range of -1 to 1.
                // Debug the dx and dy
                printf("dx: %f, dy: %f\n", dx, dy);
                selectedPoint = i;
                break;
            }
        }
    }
    else if (button == GLUT_LEFT_BUTTON && state == GLUT_UP) {
        selectedPoint = -1;
    }

    glutPostRedisplay();
}

void motion(int x, int y) {
    if (selectedPoint != -1) {
        // Change the coordinate system from pixel to normalized range of -1 to 1
        // Don't hardcode the window size. Use window width and height.
        controlPoints[selectedPoint].x = (float)x / (glutGet(GLUT_WINDOW_WIDTH) / 2) - 1;
        controlPoints[selectedPoint].y = -(float)y / (glutGet(GLUT_WINDOW_HEIGHT) / 2) + 1;
        glutPostRedisplay();
    }
}

void display() {
    glClear(GL_COLOR_BUFFER_BIT); // Clear the screen
    glColor3f(1.0f, 0.0f, 0.0f); // Set color to red

```

```

        drawBezierCurve();
        drawRails();
        drawControlPoints();
        drawTrain();

        glFlush(); // Render now
    }

void initGL() {
    glClearColor(0.0f, 0.0f, 0.0f, 1.0f); // Set background color to black and opaque
    glMatrixMode(GL_PROJECTION); // To operate on the Projection matrix
    glLoadIdentity(); // Reset the projection matrix
    gluOrtho2D(-1.0, 1.0, -1.0, 1.0); // Set the viewport
}

int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(600, 600);
    glutCreateWindow("Simple Train Simulation");
    glewInit();

    initGL();

    glutDisplayFunc(display);
    glutMouseFunc(mouse);
    glutMotionFunc(motion);

    updateTrain(0);

    glutMainLoop();
    return 0;
}

// [Todo]
/*
1. Train: add more carriages, smoke, light, sound, speed control.
2. Track: track type, station.
3. Environment: fractal tree, cloud, rain, rain ripples.
4. UI: buttons (speed, weather).
*/

```