Project 1 Report – Bezier Curve

D11315807

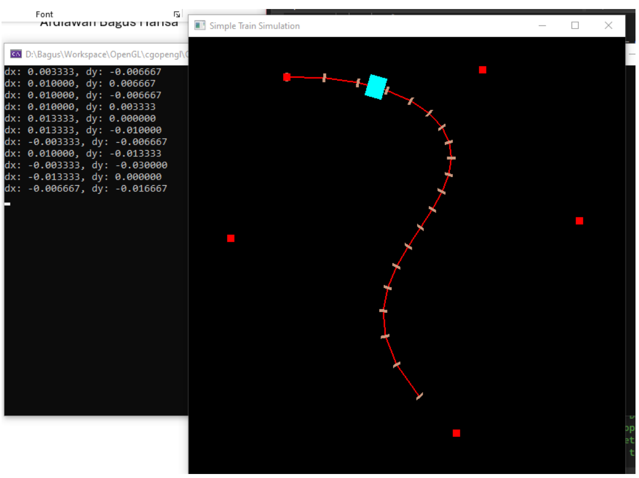
Ardiawan Bagus Harisa

Department of CSIE

Tasks:

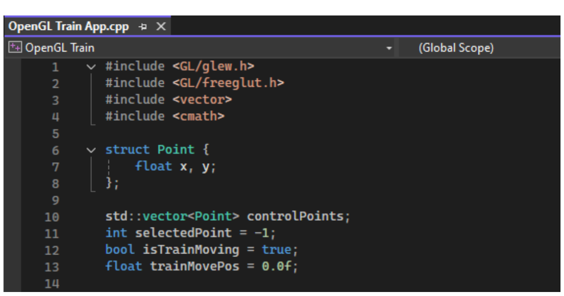
1. Display four arbitrary control points.
2. Render Bezier curves.
3. Interactive control point adjustments.
4. Recursive Implementation.

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.




```cpp
#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;
```

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*.

```cpp
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;
    }
```

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.

```cpp
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);
    }
}
```

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

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.

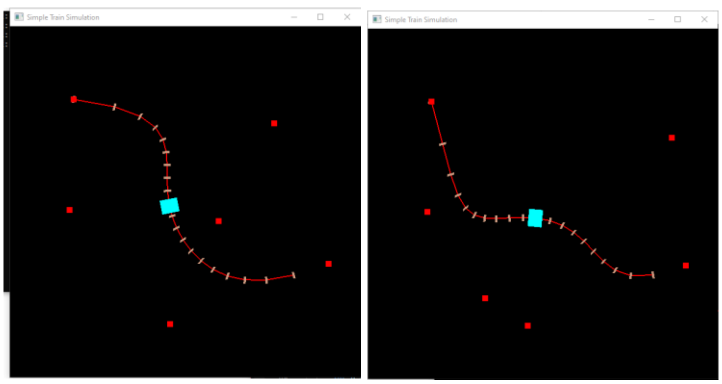
```cpp

// 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);
}
```

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*.

```
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){ ... }
    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();
}
```

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

```
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_WIDTH) / 2) + 1;
        glutPostRedisplay();
    }
}
```

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.

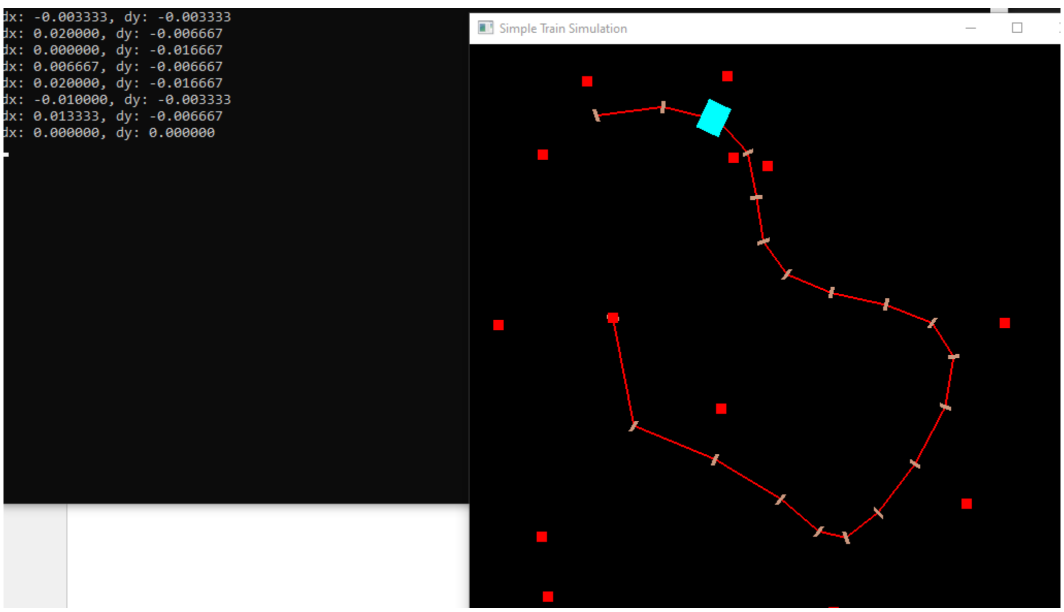
```
void drawTrain() {
    if (controlPoints.size() < 2) {
        return;
    }

    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
}
```

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

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

```
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();
}
```

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.



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_WIDTH) / 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).
*/
```