Rohan Nyati

500075940

R177219148

Batch-5 (Ai & Ml)

EXPERIMENT-9 & 10

```
#include<windows.h>
#include <stdlib.h>
#include <GL/glut.h>

/// the control points for the curve
float Points[4][3] = {
		{ 10,10,0 },
		{  5,10,2 },
		{ -5,0,0 },
		{-10,5,-2}
};

/// the level of detail of the curve
unsigned int LOD=20;

void OnReshape(int w, int h)
{
		if (h==0)
		    h=1;
```

```
	// set the drawable region of the window
	glViewport(0,0,w,h);

	// set up the projection matrix
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();

	// just use a perspective projection
	gluPerspective(45,(float)w/h,0.1,100);

	// go back to modelview matrix so we can move the objects about
	glMatrixMode(GL_MODELVIEW);
}

//------------------------------------------------------------	OnDraw()
//
void OnDraw() {

	// clear the screen & depth buffer
	glClear(GL_DEPTH_BUFFER_BIT|GL_COLOR_BUFFER_BIT);

	// clear the previous transform
	glLoadIdentity();

	// set the camera position
	gluLookAt(1,10,30,	//	eye pos
```

```
                    0,0,0,   //        aim point
                    0,1,0);  //        up direction

glColor3f(1,1,0);
glBegin(GL_LINE_STRIP);

// use the parametric time value 0 to 1
for(int i=0;i!=LOD;++i) {

    float t = (float)i/(LOD-1);

    // the t value inverted
    float it = 1.0f-t;

    // calculate blending functions
    float b0 = it*it*it/6.0f;
    float b1 = (3*t*t*t - 6*t*t +4)/6.0f;
    float b2 = (-3*t*t*t +3*t*t + 3*t + 1)/6.0f;
    float b3 =  t*t*t/6.0f;

    // sum the control points mulitplied by their respective blending functions
    float x = b0*Points[0][0] +
                    b1*Points[1][0] +
                    b2*Points[2][0] +
                    b3*Points[3][0] ;

    float y = b0*Points[0][1] +
```

```
                        b1*Points[1][1] +
                        b2*Points[2][1] +
                        b3*Points[3][1] ;

    float z = b0*Points[0][2] +
                        b1*Points[1][2] +
                        b2*Points[2][2] +
                        b3*Points[3][2] ;

    // specify the point
    glVertex3f( x,y,z );
}
glEnd();

// draw the control points
glColor3f(0,1,0);
glPointSize(3);
glBegin(GL_POINTS);
for(int i=0;i!=4;++i) {
    glVertex3fv( Points[i] );
}
glEnd();

// draw the hull of the curve
glColor3f(0,1,1);
glBegin(GL_LINE_STRIP);
for(int i=0;i!=4;++i) {
```

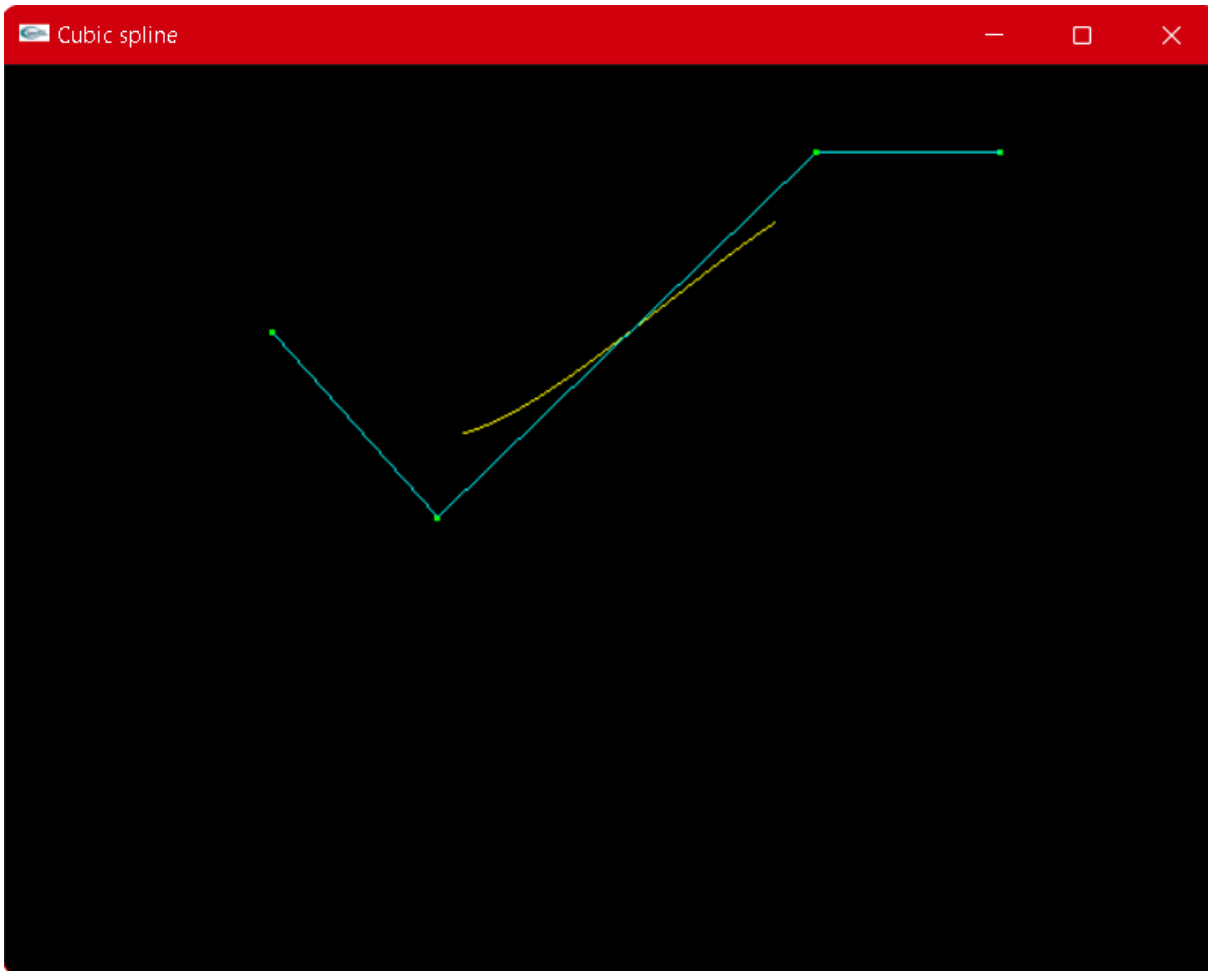
```
                glVertex3fv( Points[i] );
        }
        glEnd();

        // currently we've been drawing to the back buffer, we need
        // to swap the back buffer with the front one to make the image visible
        glutSwapBuffers();
}

//------------------------------------------------------------        OnInit()
//
void OnInit() {
        // enable depth testing
        glEnable(GL_DEPTH_TEST);
}

//------------------------------------------------------------        OnExit()
//
void OnExit() {
}

//------------------------------------------------------------        OnKeyPress()
//
void OnKeyPress(unsigned char key,int,int) {
        switch(key) {

        // increase the LOD
```

```
		case '+':
			++LOD;
			break;

		// decrease the LOD
		case '-':
			--LOD;

			// have a minimum LOD value
			if (LOD<3)
				LOD=3;
			break;
		default:
			break;
		}

		// ask glut to redraw the screen for us...
		glutPostRedisplay();
}

//------------------------------------------------------------		main()
//
int main(int argc,char** argv) {

		// initialise glut
		glutInit(&argc,argv);
```

```
// request a depth buffer, RGBA display mode, and we want double buffering
glutInitDisplayMode(GLUT_DEPTH|GLUT_RGBA|GLUT_DOUBLE);

// set the initial window size
glutInitWindowSize(640,480);

// create the window
glutCreateWindow("Cubic spline");

// set the function to use to draw our scene
glutDisplayFunc(OnDraw);

// set the function to handle changes in screen size
glutReshapeFunc(OnReshape);

// set the function for the key presses
glutKeyboardFunc(OnKeyPress);

// run our custom initialisation
OnInit();

// set the function to be called when we exit
atexit(OnExit);

// this function runs a while loop to keep the program running.
glutMainLoop();
return 0;
```

```
}
```

OUTPUT:




```
#include<windows.h>

#include <iostream>

#include <stdlib.h>

#include <GL/glut.h>

#include <math.h>

using namespace std;
```

```
class Point {  //Point class for taking the points
public:
float x, y;
void setxy(float x2, float y2) {
x = x2; y = y2;
}
//operator overloading for '=' sign
const Point& operator=(const Point& rPoint) {
x = rPoint.x;
y = rPoint.y;
return *this;
}

};

int factorial(int n) {
if (n <= 1)
return(1);
else
n = n * factorial(n - 1);
return n;
}

float binomial_coff(float n, float k) {
float ans;
ans = factorial(n) / (factorial(k) * factorial(n - k));
return ans;
```

```
}

Point abc[20];
int SCREEN_HEIGHT = 500;
int points = 0;
int clicks = 4;

void myInit() {
glClearColor(1.0, 1.0, 1.0, 0.0);
glColor3f(0.0, 0.0, 0.0);
glPointSize(3);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0.0, 640.0, 0.0, 500.0);

}

void drawDot(int x, int y) {
glBegin(GL_POINTS);
glVertex2i(x, y);
glEnd();
glFlush();
}

void drawLine(Point p1, Point p2) {
glBegin(GL_LINES);
glVertex2f(p1.x, p1.y);
```

```
glVertex2f(p2.x, p2.y);

glEnd();

glFlush();

}

//Calculate the bezier point

Point drawBezier(Point PT[], double t) {

Point P;

P.x = pow((1 - t), 3) * PT[0].x + 3 * t * pow((1 - t), 2) * PT[1].x + 3 * (1 - t) * pow(t, 2) * PT[2].x
+ pow(t, 3) * PT[3].x;

P.y = pow((1 - t), 3) * PT[0].y + 3 * t * pow((1 - t), 2) * PT[1].y + 3 * (1 - t) * pow(t, 2) * PT[2].y
+ pow(t, 3) * PT[3].y;


return P;

}

//Calculate the bezier point [generalized]

Point drawBezierGeneralized(Point PT[], double t) {

Point P;

P.x = 0; P.y = 0;

for (int i = 0; i < clicks; i++) {

P.x = P.x + binomial_coff((float)(clicks - 1), (float)i) * pow(t, (double)i) * pow((1 - t), (clicks - 1 -
i)) * PT[i].x;

P.y = P.y + binomial_coff((float)(clicks - 1), (float)i) * pow(t, (double)i) * pow((1 - t), (clicks - 1 -
i)) * PT[i].y;

}

return P;

}


void MouseClickFunc(int button, int state, int x, int y) {
```

```
// If left button was clicked
if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN) {
// Store where mouse was clicked, Y is backwards.
abc[points].setxy((float)x, (float)(SCREEN_HEIGHT - y));
points++;
// Draw the red  dot.
drawDot(x, SCREEN_HEIGHT - y);

// If (click-amout) points are drawn do the curve.
if (points == clicks){
glColor3f(0.2, 1.0, 0.0);
// Drawing the control lines
for (int k = 0; k < clicks - 1; k++)
drawLine(abc[k], abc[k + 1]);

Point p1 = abc[0];
/* Draw each segment of the curve.Make t increment in smaller amounts for a more detailed curve.*/
for (double t = 0.0; t <= 1.0; t += 0.02) {
Point p2 = drawBezierGeneralized(abc, t);
cout << p1.x << "  ,  " << p1.y << endl;
cout << p2.x << "  ,  " << p2.y << endl;
cout << endl;
drawLine(p1, p2);
p1 = p2;
}
glColor3f(0.0, 0.0, 0.0);
```

```
points = 0;
}
}
}


void DisplayFunc() {
glClear(GL_COLOR_BUFFER_BIT);
glFlush();
}

int main(int argc, char* argv[]) {
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
glutInitWindowSize(640, 500);
glutInitWindowPosition(100, 150);
glutCreateWindow("Bezier Curve");
glutMouseFunc(MouseClickFunc);
glutDisplayFunc(DisplayFunc);
myInit();
glutMainLoop();

return 0;
}
```

OUTPUT: