Practical No.:7

Title: Generate fractal patterns using
a. Bezier b. Koch Curve

Name: Samruddhi Sangram Yadav.

Roll No.:S564

A. Bezier Curve

Code:

```
#include <bits/stdc++.h>
#include <GL/glut.h>
#include <cmath>
#include <iostream>
// This is a point class, used to store the coordinates of the point
class Point
{
public:
    int x, y;
    void setxy(int _x, int _y)
    {
        x = _x;
        y = _y;
    }
};
// Number of points
static int POINTSNUM = 0;
// Used to store a collection of points, because the Bezier curves with 4 points are drawn, so the array
size is 4
static Point points[4];
// Initialization function
void init(void)
{
    glClearColor(0.0, 0.0, 0.0, 0); // Set the background to black
    glColor3f(1.0, 1.0, 1.0); // The drawing color is white
    glPointSize(4.0); // The size of the set point is 2*2 pixels
    glMatrixMode(GL_PROJECTION); // Set the appropriate matrix
    glLoadIdentity(); // Is a non-parameter valueless function, its function is to replace the current
matrix with a 4×4 identity matrix
    gluOrtho2D(0.0, 600.0, 0.0, 480.0); // Parallel projection, the four parameters are x, y range
}
// Draw points
void setPoint(Point p)
{
    glBegin(GL_POINTS);
    glVertex2f(p.x, p.y);
    glEnd();
    glFlush();
}
// Draw a straight line
void setline(Point p1, Point p2)
{
    glBegin(GL_LINES);
```

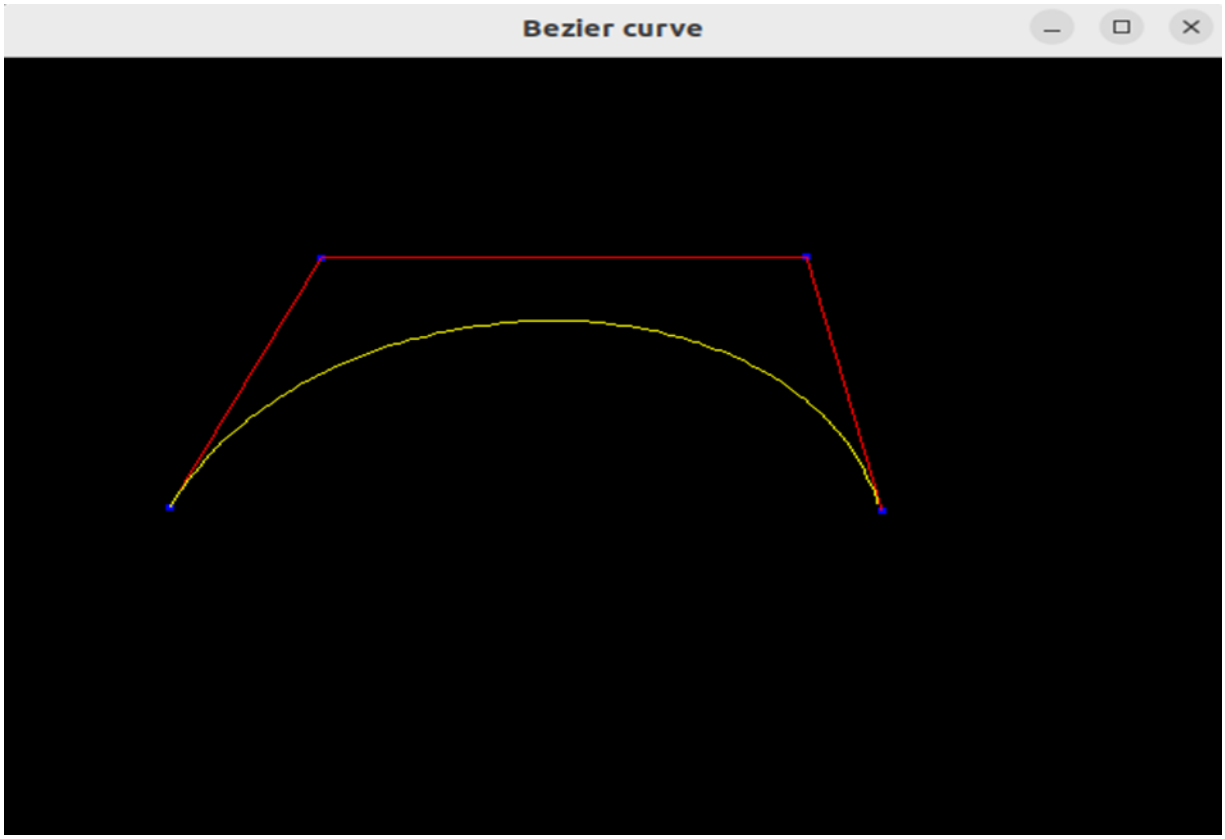
```
    glVertex2f(p1.x, p1.y); // Set vertex coordinates
    glVertex2f(p2.x, p2.y);
    glEnd();
    glFlush(); // Empty the buffer
}
// Draw Bezier curve (with 'u' instead of 't')
Point setBezier(Point p1, Point p2, Point p3, Point p4, double u)
{
    Point p;
    double a1 = pow((1 - u), 3);          // (1 - u)^3
    double a2 = pow((1 - u), 2) * 3 * u;  // (1 - u)^2 * 3 * u
    double a3 = 3 * u * u * (1 - u);      // 3 * u^2 * (1 - u)
    double a4 = u * u * u;                // u^3
    p.x = a1 * p1.x + a2 * p2.x + a3 * p3.x + a4 * p4.x; // Compute the x-coordinate
    p.y = a1 * p1.y + a2 * p2.y + a3 * p3.y + a4 * p4.y; // Compute the y-coordinate
    return p;
}
// Mouse event
void mymouseFunction(int button, int state, int x, int y)
{

    if (state == GLUT_DOWN) // If the mouse is pressed, the left and right buttons are not
distinguished

    {
        points[POINTSNUM].setxy(x, 480 - y); // When looking for the coordinates of the mouse point
here
        glColor3f(0.0, 0.0, 1.0); // Set the color of the point, draw the point
        setPoint(points[POINTSNUM]);
        glColor3f(1.0, 0.0, 0.0); // Set the color of the line, draw the line
        if (POINTSNUM > 0)
            setline(points[POINTSNUM - 1], points[POINTSNUM]);
        // If 4 bezier curves are reached, the counter will be cleared afterward
        if (POINTSNUM == 3)
        {
            // Draw Bezier curve
            glColor3f(1.0, 1.0, 0.0); // Set the color of the Bezier curve
            Point p_current = points[0]; // Set as starting point
            for (double u = 0.0; u <= 1.0; u += 0.01) // Increase u increment to make smoother curve
            {
                Point P = setBezier(points[0], points[1], points[2], points[3], u); // Use 'u' here instead of 't'
                setline(p_current, P);
                p_current = P;
            }
            POINTSNUM = 0; // Reset after drawing the curve
        }
        else
        {

            POINTSNUM++; // Increment the point counter
        }
    }
```

```
}
int main(int argc, char *argv[])
{
    glutInit(&argc, argv); // Fixed format
    glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE);   // Cache mode
    glutInitWindowSize(600, 480);    // The size of the display box

    glutInitWindowPosition(100, 100); // Determine the position of the upper left corner of the display box

    glutCreateWindow("Bezier curve");

    init(); // Initialize
    glutMouseFunc(mymouseFunction); // Add mouse event

    glutMainLoop(); // Enter the GLUT event processing loop

    return 0;

}
```

Output:

dell@dell-Latitude-E5430-non-vPro:~$ g++ bezier.cpp -lGL -lGLU -lglut

dell@dell-Latitude-E5430-non-vPro:~$ ./a.out



B) Koch Curve

Code:

```
#include <GL/glut.h>
#include <math.h>
// Initial points of the line
float ax = -200, ay = 0;
float bx = 200, by = 0;
// Recursive depth
int depth = 2;
void drawKochCurve(float x1, float y1, float x2, float y2, int n)
{
   if (n == 0)
   {
      glVertex2f(x1, y1);
      glVertex2f(x2, y2);
   }
   else
   {
      float x3 = (2 * x1 + x2) / 3;
      float y3 = (2 * y1 + y2) / 3;
      float x4 = (x1 + 2 * x2) / 3;
      float y4 = (y1 + 2 * y2) / 3;
      // Calculate peak of the triangle
      float dx = x4 - x3;
      float dy = y4 - y3;
      float x = x3 + (dx * cos(M_PI / 3) - dy * sin(M_PI / 3));
      float y = y3 + (dx * sin(M_PI / 3) + dy * cos(M_PI / 3));
      drawKochCurve(x1, y1, x3, y3, n - 1);
      drawKochCurve(x3, y3, x, y, n - 1);
      drawKochCurve(x, y, x4, y4, n - 1);
      drawKochCurve(x4, y4, x2, y2, n - 1);
   }
}
void display()
{
   glClear(GL_COLOR_BUFFER_BIT);
   glColor3f(0.0, 1.0, 1.0);
   glBegin(GL_LINES);
   drawKochCurve(ax, ay, bx, by, depth);
   glEnd();
   glFlush();
}
void init()
{
   glClearColor(0.0, 0.0, 0.0, 1.0); // black background
   gluOrtho2D(-300, 300, -200, 200); // setting coordinate system
}
int main(int argc, char** argv)
{
   glutInit(&argc, argv);
   glutInitWindowSize(600, 400);
   glutInitWindowPosition(100, 100);
```

```
    glutCreateWindow("Koch Curve - OpenGL in C++");
    init();
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```

Output:

```
dell@dell-Latitude-E5430-non-vPro:~$ g++ koch.cpp -lGL -lGLU -lglut
dell@dell-Latitude-E5430-non-vPro:~$ ./a.out
```