Bezier Curve

```
#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;
}
const Point & operator=(const Point &rPoint) //operator overloading for ‘=’ sign
{
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(0.0,0.0,0.0,0.0);
glColor3f(1.0,1.0,1.0);
glPointSize(3);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0.0,500.0,0.0,500.0);
}
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();
}
Point drawBezier(Point PT[], double t)//Calculate the bezier point
{
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;
}
Point drawBezierGeneralized(Point PT[], double t) //Calculate the bezier point
[generalized]
{
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 myMouse(int button, int state, int x, int y) // If left button was clicked
{
if(button == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
{
abc[points].setxy((float)x,(float)(SCREEN_HEIGHT-y));
// Store where mouse was clicked, Y is backwards.
points++;
drawDot(x,SCREEN_HEIGHT-y); // Draw the red dot.
if(points == clicks) // If (click-amout) points are drawn do the curve.
{
glColor3f(1.0,0.0,0.0);
for(int k=0;k<clicks-1;k++) // Drawing the control lines
drawLine(abc[k], abc[k+1]);
Point p1 = abc[0];
```

```
glColor3f(1.0,1.0,1.0);
// 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);
drawLine(p1, p2);
p1 = p2;
}
glColor3f(0.0,0.0,0.0);
points = 0;
}
}
}
void myDisplay()
{
glClear(GL_COLOR_BUFFER_BIT);
glFlush();
}
int main(int argc, char *argv[])
{
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutInitWindowSize(500,500);
glutInitWindowPosition(100,150);
glutCreateWindow("Bezier Curve");
glutMouseFunc(myMouse);
glutDisplayFunc(myDisplay);
myInit();
glutMainLoop();
return 0;
}
```

Koch Curve

```
#include<iostream>
#include<stdio.h>
#include<GL/glut.h>
#include<stdlib.h>
using namespace std;
#define SIN 0.86602540 // oin(60 dogreen)
int n;
int x1 = 0, x2 = 550, y1=0, y2 = 0;
void koch(int x1, int y1, int x2, int y2, int m)
{
int xx, yy, x[5], y[5], lx, ly, offx=50, offy = 300;
lx = (x2-x1)/3;
ly = (y2-y1)/3;
x[0] =x1; // Store point po
y[0] = y1;
```

```
x[4] = x2; // Store point p4
y[4] = y2;
x[1] =x[0] + lx; // Store point p1
y[1] = y[0] + ly;
x[3] = x[0] + 2*lx; // Store point p3
y[3] = y[0] + 2*ly;
xx = x[3] -x[1]; // Translate point p2 to origin
yy = y[3]- y[1];
x[2] = xx*(0.5) + yy*(SIN); // Perform rotation for point p3
y[2] = -xx*(SIN) + yy*(0.5);
x[2] = x[2] + x[1]; // Perform inverse translation
y[2] = y[2] + y[1];
if(m>0)
{
koch(x[0], y[0], x[1], y[1], m-1); // Recursive call to Draw part1
koch(x[1], y[1], x[2], y[2], m-1); // Recursive call to Draw part2
koch(x[2], y[2], x[3], y[3], m-1); // Recursive call to Draw part3
koch(x[3], y[3], x[4], y[4], m-1); // Recursive call to Draw part4
}
else
{
glBegin(GL_LINES);
glVertex2d(offx + x[0],650-(offy + y[0]));
glVertex2d(offx + x[1],650-(offy + y[1]));
glEnd();
glBegin(GL_LINES);
glVertex2d(offx + x[1],650-(offy + y[1]));
glVertex2d(offx + x[2],650-(offy + y[2]));
glEnd();
glBegin(GL_LINES);
glVertex2d(offx + x[2],650-(offy + y[2]));
glVertex2d(offx + x[3],650-(offy + y[3]));
glEnd();
glBegin(GL_LINES);
glVertex2d(offx + x[3],650-(offy + y[3]));
glVertex2d(offx + x[4],660-(offy + y[4]));
glEnd();
}
}
void display(void)
{
glClear(GL_COLOR_BUFFER_BIT);
glColor3f(1.0, 1.0, 1.0);
koch(x1, y1, x2, y2, n);
glFlush(); // send all output to display
}
void myinit() {
glClearColor(0.0, 0.0, 0.0, 1.0); // set background as black
glColor3f(1.0, 1.0, 0.0); // Draw in Yellow
glMatrixMode(GL_PROJECTION); // Establish the coordinate system
```

```
glLoadIdentity();
gluOrtho2D(0.0, 650.0, 0.0, 650.0);
}
int main(int argc, char **argv)
{
/* Initialise graphics mode
-----------------------------------*/
cout<<"\n Enter the level of curve generation : ";
cin>>n;
glutInit(&argc, argv); // Initialize the toolkit
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB); // Set display mode
glutInitWindowSize(650, 650); // Set window size
glutInitWindowPosition(0,0); // Set window position on the screen
// Open the screen window
glutCreateWindow("Koch Curve");
glutDisplayFunc(display); // Register redraw function
myinit();
glutMainLoop();
return 0;
}
```