Experiment No 7

Title Generate fractal patterns using i) Bezier Koch Curve

Bezier curve is discovered by the French engineer **Pierre Bézier**. These curves can be generated under the control of other points. Approximate tangents by using control points are used to generate curve. The Bezier curve can be represented mathematically as −

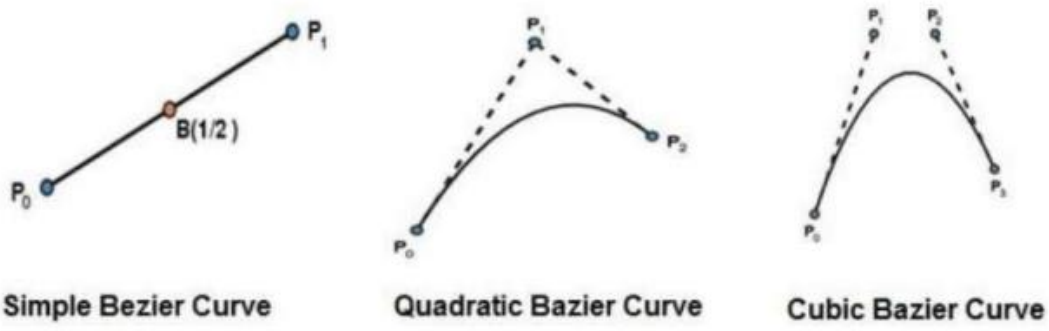
$$\sum k=0nPiBni(t)\sum k=0nPiBin(t)$$

Where pipi is the set of points and Bni(t)Bin(t) represents the Bernstein polynomials which are given by −

$$Bni(t)=(ni)(1-t)n-itiBin(t)=(ni)(1-t)n-iti$$

Where **n** is the polynomial degree, **i** is the index, and **t** is the variable.

The simplest Bézier curve is the straight line from the point P0P0 to P1P1. A quadratic Bezier curve is determined by three control points. A cubic Bezier curve is determined by four control points.



Simple Bezier Curve Quadratic Bazier Curve Cubic Bazier Curve

Properties of Bezier Curves

Bezier curves have the following properties

They generally follow the shape of the control polygon, which consists of the segments joining the control points.

They always pass through the first and last control points.

They are contained in the convex hull of their defining control points.

The degree of the polynomial defining the curve segment is one less that the number of defining polygon point. Therefore, for 4 control points, the degree of the polynomial is 3, i.e. cubic polynomial.

A Bezier curve generally follows the shape of the defining polygon.

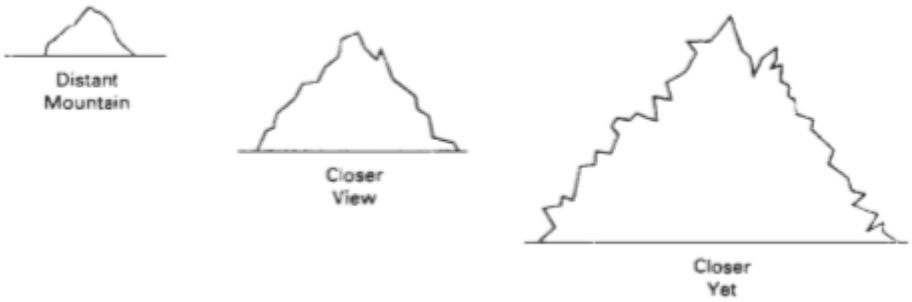
The direction of the tangent vector at the end points is same as that of the vector determined by first and last segments.

The convex hull property for a Bezier curve ensures that the polynomial smoothly follows the control points.
No straight line intersects a Bezier curve more times than it intersects its control polygon.

They are invariant under an affine transformation.

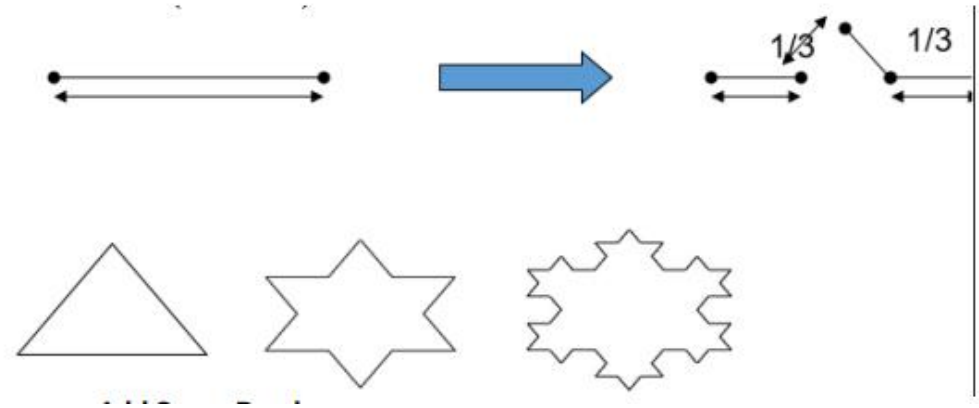
Bezier curves exhibit global control means moving a control point alters the shape of the whole curve.

A given Beziercurve can be subdivided at a point t=t0 into two Bezier segments which join together at the point corresponding to the parameter value t=t0.
Koch Curve
Fractals are geometric objects. Many real-world objects like ferns are shaped like fractals. Fractals are formed by iterations. Fractals are self-similar.
In computer graphics, we use fractal functions to create complex object
The object representations uses Euclidean-geometry methods; that is, object shapes were described with equations. These methods are adequate for describing manufactured objects: those that have smooth surfaces and regular shapes. But natural objects, such as mountains and clouds, have irregular or fragmented features, and Euclidean methods do not realistically model these objects. Natural objects can be realistically described with fractal-geometry methods, where procedures rather than equations are used to model objects.
In computer graphics, fractal methods are used to generate displays of natural objects and visualizations . The self-similarity properties of an object can take different forms, depending on the choice of fractal representation.
In computer graphics, we use fractal functions to create complex objects



A mountain outlined against the sky continues to have the same jagged shape as we view it from a closer and closer. We can describe the amount of variation in the object detail with a number called the fractal dimension. Examples: In graphics applications, fractal representations are used to model terrain, clouds, water, trees and other plants, feathers, fur, and various surface textures, and just to make pretty patterns. In other disciplines, fractal patterns have been found in the distribution of stars, river islands, and moon craters; in rain fields; in stock market variations; in music; in traffic flow; in urban property utilization; and in the boundaries of convergence regions for numerical-analysis techniques

Koch Fractals (Snowflakes)



Add Some Randomness:

• The fractals we've produced so far seem to be very regular and "artificial".

• To create some realism and variability, simply change the angles slightly sometimes based on a random number generator.

• For example, you can curve some of the ferns to one side.

• For example, you can also vary the lengths of the branches and the branching factor.

Terrain (Random Mid-point Displacement):

• Given the heights of two end-points, generate a height at the mid-point.

• Suppose that the two end-points are a and b. Suppose the height is in the y direction, such that the height at a is y(a), and the height at b is y(b).

• Then, the height at the mid-point will be:

ymid = (y(a)+y(b))/2 + r, where

r is the random offset

• This is how to generate the random offset r:

r = srg|b-a|, where

s is a user-selected “roughness” factor, and

rg is a Gaussian random variable with mean 0 and variance 1

CODE FOR Bezier Curves

```
#include <iostream>
#include <math.h>
#include <time.h>
#include <GL/glut.h>
using namespace std;
int x[4],y[4];
void init(){
 glClearColor(1.0,1.0,1.0,0.0);
 glMatrixMode(GL_PROJECTION);
 gluOrtho2D(0,640,0,480);
 glClear(GL_COLOR_BUFFER_BIT);
}
void putpixel(double xt,double yt )
{
 glColor3f(1,0,0);
 glBegin(GL_POINTS);
 glVertex2d(xt,yt);
 glEnd();
 glFlush();
}
void Algorithm(){
 glColor3f(0,1,0);
 glBegin(GL_LINES);
```

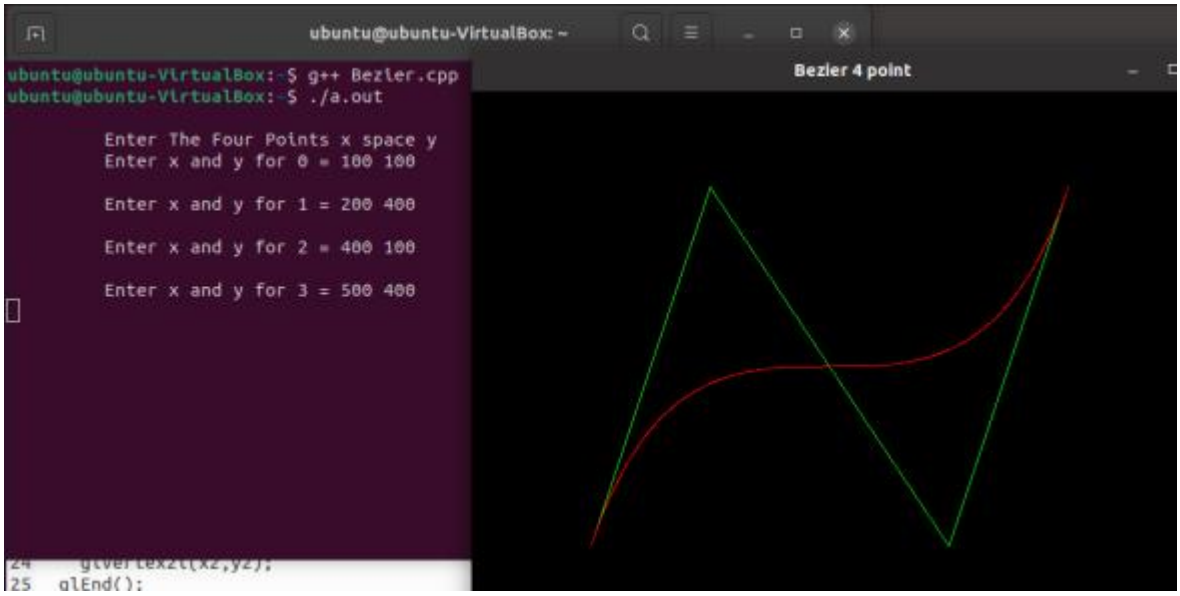
```
 glVertex2i(x[0],y[0]);
 glVertex2i(x[1],y[1]);
 glVertex2i(x[1],y[1]);
 glVertex2i(x[2],y[2]);
 glVertex2i(x[2],y[2]);
 glVertex2i(x[3],y[3]);
 glEnd();
 glFlush();
 double t;
 for (t = 0.0; t < 1.0; t += 0.0005)
 {
 double xt = pow(1-t, 3) * x[0] + 3 * t * pow(1-t, 2) * x[1] + 3 * pow(t, 2) * (1-t) * x[2] +
pow(t, 3) * x[3];
 double yt = pow(1-t, 3) * y[0] + 3 * t * pow(1-t, 2) * y[1] + 3 * pow(t, 2) * (1-t) * y[2] +
pow(t, 3) * y[3];
 putpixel(xt, yt);
 }
}
int main(int argc, char** argv){

 cout<<"\n \t Enter The Four Points x space y ";
 for(int i=0;i<4;i++){
 cout<<"\n \t Enter x and y for "<<i<<" = ";
 cin>>x[i]>>y[i];
 }

 glutInit(&argc, argv);
 glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
 glutInitWindowSize(640,480);
 glutInitWindowPosition(200,200);
 glutCreateWindow("Bezier 4 point");
```

```
 init();
 glutDisplayFunc(Algorithm);

 glutMainLoop();
 return 0;
}
```



CODE FOR Koch Curve

```
#include <iostream>
#include <math.h>
#include <time.h>
#include <GL/glut.h>
using namespace std;
double x,y,len,angle;
int it;
void init(){
 glClearColor(1.0,1.0,1.0,0.0);
 glMatrixMode(GL_PROJECTION);
 gluOrtho2D(0,640,0,480);
 glClear(GL_COLOR_BUFFER_BIT);
```

```
}
void line1(int x1, int y11, int x2,int y2){

 glColor3f(0,1,0);
 glBegin(GL_LINES);
 glVertex2i(x1,y11);
 glVertex2i(x2,y2);
 glEnd();
 glFlush();
}
void k_curve(double x, double y, double len, double angle, int it){
 if(it>0){

 len /=3;
 k_curve(x,y,len,angle,(it-1));
 x += (len * cosl(angle * (M_PI)/180));
 y += (len * sinl(angle * (M_PI)/180));
 k_curve(x,y, len, angle+60,(it-1));
 x += (len * cosl((angle + 60) * (M_PI)/180));
 y += (len * sinl((angle + 60) * (M_PI)/180));
 k_curve(x,y, len, angle-60,(it-1));
 x += (len * cosl((angle - 60) * (M_PI)/180));
 y += (len * sinl((angle - 60) * (M_PI)/180));
 k_curve(x,y,len,angle,(it-1));
 }
 else
 {
 line1(x,y,(int)(x + len * cosl(angle * (M_PI)/180) + 0.5),(int)(y + len * sinl(angle *
(M_PI)/180) + 0.5));
 }
}
```

```
void Algorithm(){
 k_curve(x,y,len,angle,it);

}
int main(int argc, char** argv){

 cout<<"\n Enter Starting Point x space y ";
 cin>>x>>y;
 cout <<"\n Lenght of line and space angle of line";
 cin>>len>>angle;
 cout<<"\n No. of ittration ";
 cin>>it;
 glutInit(&argc, argv);
 glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
 glutInitWindowSize(640,480);
 glutInitWindowPosition(200,200);
 glutCreateWindow("Koch");
 init();
 glutDisplayFunc(Algorithm);

 glutMainLoop();
 return 0;
}
```

OUTPUT


ubuntu@ubuntu-VirtualBox: ~
ubuntu@ubuntu-VirtualBox:~$ g++ A_7_koch.cpp -lGL -lGLU -lglut
ubuntu@ubuntu-VirtualBox:~$ ./a.out
Enter Starting Point x space y 100 100
Lenght of line and space angle of line400
10
No. of ittration 3
Koch
glEnd();
glFlush();
}
void k_curve(doubl