NSGI's

NAVASHYADIR GROUP OF INSTITUTE FACULTY OF ENGINEERING, PUNE

DEPARTMENT OF AIML ENGINEERING

LAB MANUAL

CLASS: - S.E.AIML.

SUBJECT: - Computer Graphics Lab Manual

SEMESTER:- IInd

by

Dr. S.N.Gujar

SAVITRIBAI PHULE PUNE UNIVERSITY

Experiment No	1
Title	Install and explore the OpenGL.
Roll No.	
College	NSGIFOE , Pune
Class	S.E. (AIML)
Date	
Subject	Computer Graphics

Computer Graphics(Basic and how to install OpenGL/Glut)

INTRODUCTION

What is Computer Graphics?
Branch of Computer Science.
Computer + Graphs + Pics = Computer Graphics.
Drawing line, Chart, Graphs etc. on the screen using Programming language is computer Graphics.
Book Definition
Computer graphics is the branch of computer science that deals with generating images with the aid of computers. It displays the information in the from of graphics objects such as picture, charts, graphs and diagrams instead of simple text. We can say computer graphics makes it possible to express data in pictorial form.
OpenGL
It is cross-platform, cross-language API for rendering 2D and 3D Graphics(Vector Graphics).
Refer https://www.khronos.org/opengl/
GLUT

http://freeglut.sourceforge.net/docs/api.php#Introduction

https://www.opengl.org/resources/libraries/glut/spec3/node10.html

https://www.opengl.org/resources/libraries/glut/spec3/node11.html

https://www.opengl.org/resources/libraries/glut/spec3/node12.html

https://www.opengl.org/resources/libraries/glut/spec3/node13.html

https://www.opengl.org/resources/libraries/glut/spec3/node15.html

How to Setup Glut on Windows

https://medium.com/swlh/setting-opengl-for-windows-d0b45062caf

How to install Ubuntu in Windows Virtual Box

https://youtu.be/x5MhydijWmc

How to install OpenGL / Glut (Ubuntu)

Step 1: sudo apt-get update
To update your basic packages
Step 2: sudo apt-get install build-essential
For installing essential packages.
Step 3: sudo apt-get install freeglut3 freeglut3-dev
Step 4: sudo apt-get install binutils-gold
Step 5: sudo apt-get install g++ cmake
Step 6: sudo apt-get install mesa-common-dev mesa-utils
Step 7: sudo apt-get install libglew-dev libglew1.5-dev libglm-dev
Step 8: glxinfo | grep OpenGL

Create cpp file and write your code

Run on terminal:

```
g++ MyProg.cpp -lGL -lGLU -lglut (for C++ program)
./a.out
```

Basic Code # Triangle

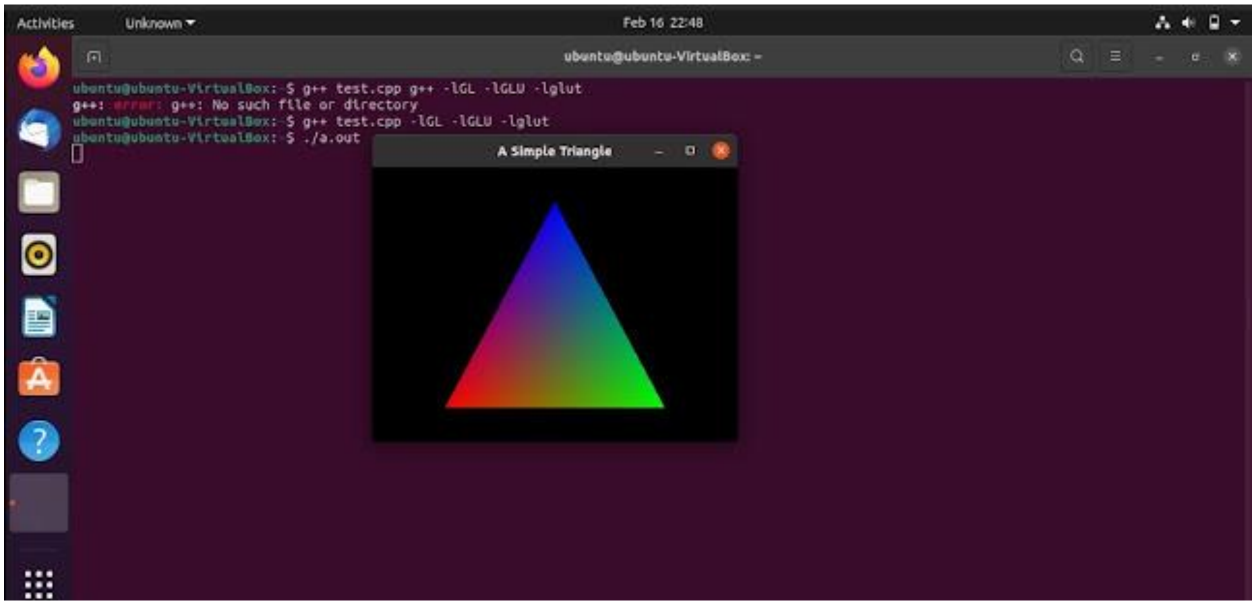
In C++

```
// A simple introductory program; its main window contains a static picture
// of a triangle, whose three vertices are red, green and blue.  The program

#include <GL/glut.h>


// Clears the current window and draws a triangle.
void display() {

  // Set every pixel in the frame buffer to the current clear color.
  glClear(GL_COLOR_BUFFER_BIT);

  // Drawing is done by specifying a sequence of vertices.  The way these
  // vertices are connected (or not connected) depends on the argument to
  // glBegin.  GL_POLYGON constructs a filled polygon.
  glBegin(GL_POLYGON);
    glColor3f(1, 0, 0); glVertex3f(-0.6, -0.75, 0.5);
    glColor3f(0, 1, 0); glVertex3f(0.6, -0.75, 0);
    glColor3f(0, 0, 1); glVertex3f(0, 0.75, 0);
  glEnd();

  // Flush drawing command buffer to make drawing happen as soon as possible.
  glFlush();
}

// Initializes GLUT, the display mode, and main window; registers callbacks;
// enters the main event loop.
int main(int argc, char** argv) {

  // Use a single buffered window in RGB mode (as opposed to a double-buffered
  // window or color-index mode).
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);

  // Position window at (80,80)-(480,380) and give it a title.
  glutInitWindowPosition(80, 80);
  glutInitWindowSize(400, 300);
  glutCreateWindow("A Simple Triangle");

  // Tell GLUT that whenever the main window needs to be repainted that it
  // should call the function display().
  glutDisplayFunc(display);
  // Tell GLUT to start reading and processing events.  This function
  // never returns; the program only exits when the user closes the main
  // window or kills the process.
  glutMainLoop();
}
```

How to run glut/ OpenGL in g++ (Ubuntu)

```
g++ MyProg.cpp -lGL -lGLU -lglut
```

OUTPUT



```
ubuntu@ubuntu-VirtualBox:~$ g++ test.cpp g++ -lGL -lGLU -lglut
g++: error: g++: No such file or directory
ubuntu@ubuntu-VirtualBox:~$ g++ test.cpp -lGL -lGLU -lglut
ubuntu@ubuntu-VirtualBox:~$ ./a.out
```

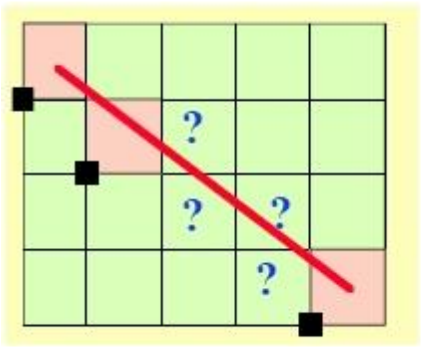
Experiment No	2
Title	Implement DDA and Bresenham line drawing algorithm to draw: i) Simple Line ii) Dotted Lin iii) Dashed Line iv) Solid line; using mouse interface Divide the screen in four quadrants wit center as (0, 0). The line should work for all the slopes positive as well as negative.
Roll No.	
College	NSGIFOE , Pune
Class	S.E. (AIML)
Date	
Subject	Computer Graphics

Problem Statement:

Write a program to implement following algorithms :

 DDA Line Drawing Algorithm

 Bresenham Line Drawing Algorithm



Objective:

To understand the different algorithms for line & circle generation and their implementation.

Problem definition:

Rasterization : Process of determining which pixels provide the best approximation to a desired

line on the screen.

Scan Conversion: Combination of rasterization and generating the picture in scan line order.

General requirements

 Straight lines must appear as straight lines.

 They must start and end accurately.

 Lines should have constant brightness along their length

 Lines should drawn rapidly.

Line drawing algorithm

 Programmer specifies (x,y) values of end pixels

 Need algorithm to figure out which intermediate pixels are on line path

 Pixel (x,y) values constrained to integer values

 Actual computed intermediate line values may be floats

 Rounding may be required. E.g. computed point (10.48, 20.51) rounded to (10, 21)

 Rounded pixel value is off actual line path.

 Sloped lines end up having jaggies

 Vertical, horizontal lines, no jaggies

```
#include<iostream>
#include<GL/glut.h>
using namespace std;
int Algo,type;
void Init()
{
   glClearColor(0,0,0,0);
   glColor3f(0,1,0);
   gluOrtho2D(0,640,0,480);
   glClear(GL_COLOR_BUFFER_BIT);
}
int sign(float a){
   if(a==0){
      return 0;
   }
   if(a>0){
      return 1;
   }
   return -1;
}
void B_Line(int x_1,int y_1,int x_2,int y_2,int t){

   float dy, dx, m , P;
   dy = y_2 - y_1;
   dx = x_2 - x_1;

   m = dy/dx;

   P = 2*dy - dx;
   int x = x_1, y = y_1;
   cout<<"\n x1 = "<<x<<" y1 = "<<y;
   if(m<1){
      int cnt=1;
      for(int i=0; i<=dx;i++){

         if(t == 1){
            glBegin(GL_POINTS);
               glVertex2i(x,y);
            glEnd();
            }
            if(t == 2){

                if(i%2==0){
                    glBegin(GL_POINTS);
                        glVertex2i(x,y);
                    glEnd();
                }
            }
            if(t == 3){

                if(cnt <= 10){
                    glBegin(GL_POINTS);
                        glVertex2i(x,y);
                    glEnd();

                }
                cnt++;
                if(cnt == 15){
                    cnt =1;
                }
            }
            if(P<0){
                x = x +1;
                y =y;
                P = P + 2*dy;
            }
            else{
                x= x+1;
                y = y+1;
                P = P + 2*dy - 2*dx;
            }
        }
    }
    else{
        int cnt = 1;
        for(int i=0;i<=dy;i++){
            if(t == 1){
                glBegin(GL_POINTS);
                    glVertex2i(x,y);
                glEnd();
            }
            if(t == 2){

                if(i%2==0){
                    glBegin(GL_POINTS);
                        glVertex2i(x,y);
                    glEnd();
                }
            }
            if(t == 3){

                if(cnt <= 10){
```

```
                glBegin(GL_POINTS);
                    glVertex2i(x,y);
                glEnd();

            }
            cnt++;
            if(cnt == 15){
                cnt =1;
            }

        }
        if(P<0){

            x = x;
            y =y+1;
            P = P + 2*dx;
        }
        else{

            x= x+1;
            y = y+1;
            P = P + 2*dx - 2*dy;

        }


    }
  }
  cout<<"\n xlast = "<<x<<" ylast = "<<y;
  glFlush();
}

void DDA_LINE(int x_1,int y_1,int x_2,int y_2, int t){
  float dx,dy,length;
  dx = x_2-x_1;
  dy = y_2-y_1;
  if(abs(dx) >= abs(dy)){
      length = abs(dx);
  }
  else{
      length = abs(dy);
  }
  float xin, yin;
  xin = dx/length;
  yin = dy/length;
  float x,y;
  x = x_1 + 0.5 * sign(xin);
  y = y_1 + 0.5 * sign(yin);
  int i=0;
  int cnt =1;
  while(i<=length){
      if(t == 1){
          glBegin(GL_POINTS);
```

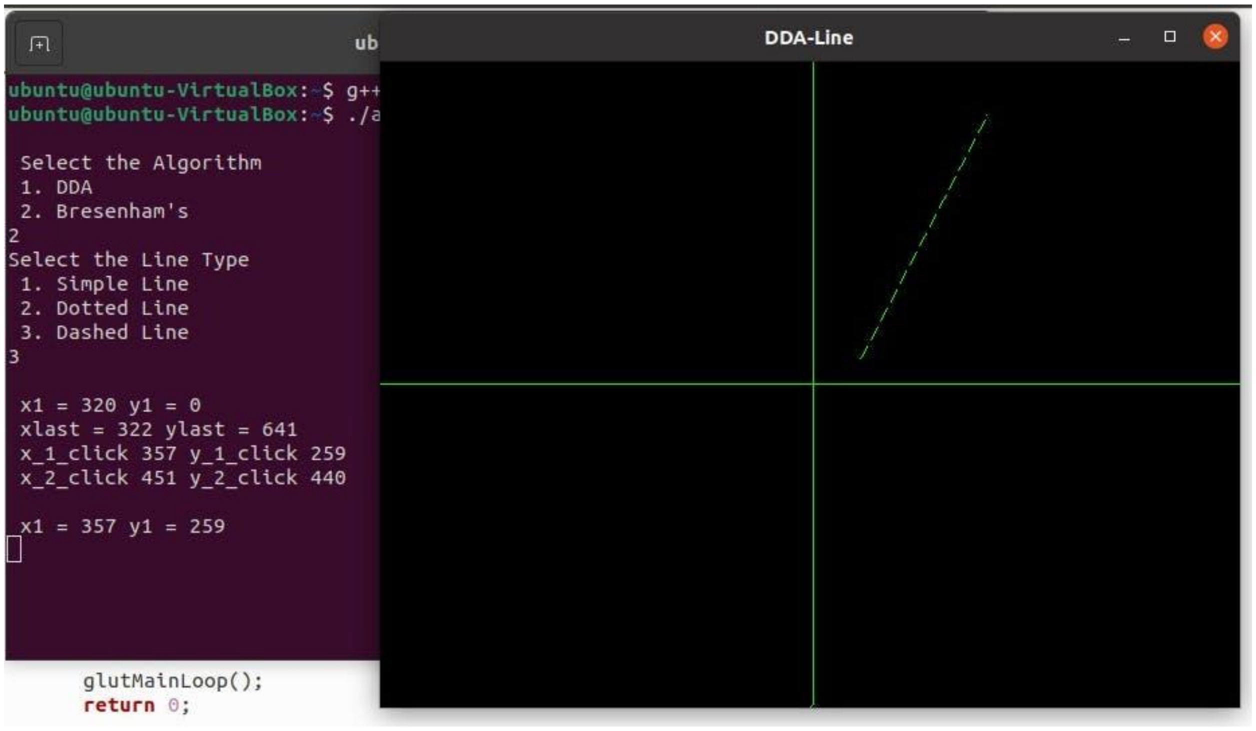
```
            glVertex2i(x,y);
          glEnd();
        }
        if(t == 2){

          if(i%2==0){
            glBegin(GL_POINTS);
              glVertex2i(x,y);
            glEnd();
          }
        }
        if(t == 3){

          if(cnt <= 10){
            glBegin(GL_POINTS);
              glVertex2i(x,y);
            glEnd();
          }
          cnt++;
          if(cnt == 15){
            cnt =1;
          }     }

        x = x + xin;
        y = y + yin;
        i++ ;

    }

    glFlush();

}

void display()
{
    DDA_LINE(0,240,640,240,1);
    B_Line(320,0,320,640,1);


    glFlush();
}
void mymouse(int b,int s, int x, int y)
{
    static int x_s,y_s,x_e,y_e,pt=0;
    if(b==GLUT_LEFT_BUTTON && s==GLUT_DOWN)
    {
      if(pt==0)
      {
        x_s =x;
        y_s =480 - y;
        pt++;

        glBegin(GL_POINTS);
```

```
            glVertex2i(x_s,y_s);
         glEnd();
      }
      else
      {
         x_e=x;
         y_e=480-y;
         cout<<"\n x_1_click "<<x_s<<" y_1_click "<<y_s;
         cout<<"\n x_2_click "<<x_e<<" y_2_click "<<y_e<<"\n";
         glBegin(GL_POINTS);
            glVertex2i(x_e,y_e);
         glEnd();
         if(Algo == 1){
            DDA_LINE(x_s,y_s,x_e,y_e,type);
         }
         if(Algo == 2){
            B_Line(x_s,y_s,x_e,y_e,type);
         }

      }


   }
   else if(b==GLUT_RIGHT_BUTTON && s==GLUT_DOWN)
      {
         pt=0;
      }
   glFlush();
}




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

   cout<<"\n Select the Algorithm \n 1. DDA \n 2. Bresenham's \n";
   cin>>Algo;
   cout<<"Select the Line Type \n 1. Simple Line \n 2. Dotted Line\n 3. Dashed Line \n";
   cin>>type;

   if((Algo == 1 || Algo == 2 )&&(type==1 || type==2 || type==3)){

   }
   else{
      cout<<"\n Option enter are wrong \n";
      return 0;
   }

   glutInit(&argc,argv);
   glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
   glutInitWindowPosition(100,100);
```

```
    glutInitWindowSize(640,480);
    glutCreateWindow("DDA-Line");
    Init();
    glutDisplayFunc(display);
    glutMouseFunc(mymouse);

    glutMainLoop();
    return 0;

}
```

OUTPUT

```
g++ filename.cpp -lGL -lGLU -lglut

./a.out
```

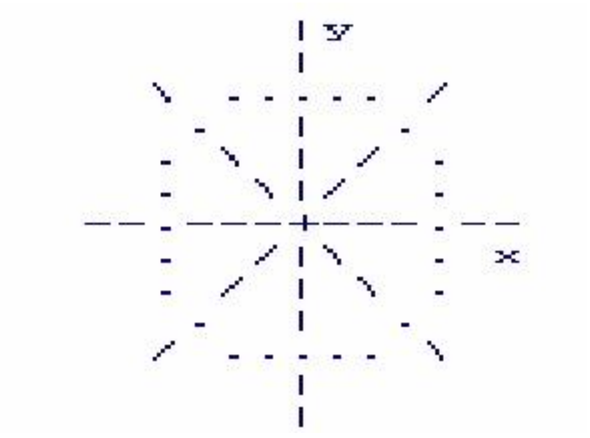


```
ubuntu@ubuntu-VirtualBox:~$ g++
ubuntu@ubuntu-VirtualBox:~$ ./a

 Select the Algorithm
 1. DDA
 2. Bresenham's
2
Select the Line Type
 1. Simple Line
 2. Dotted Line
 3. Dashed Line
3

 x1 = 320 y1 = 0
 xlast = 322 ylast = 641
 x_1_click 357 y_1_click 259
 x_2_click 451 y_2_click 440

 x1 = 357 y1 = 259
```

```
      glutMainLoop();
      return 0;
```

Experiment No	3
Title	Implement Bresenham circle drawing algorithm draw any object. The object should be displayed in all the quadrants with respect to center and radius
Roll No.	
College	NSGIFOE , Pune
Class	S.E. (AIML)
Date	
Subject	Computer Graphics

Bresenham's Circle Algorithm
Circles have the property of being highly symmetrical, which is handy when it comes to drawing them on a display screen.



We know that there are 360 degrees in a circle. First we see that a circle is symetrical about the x axis, so only the first 180 degrees need to be calculated. Next we see that it's also symetrical about the y axis, so now we only need to calculate the first 90 degrees. Finally we see that the circle is also symetrical about the 45 degree diagonal axis, so we only need to calculate the first 45 degrees. Bresenham's circle algorithm calculates the locations of the pixels in the first 45 degrees. It assumes that the circle is centered on the origin. So for every pixel (x,y) it calculates we draw a pixel in each of the 8 octants of the circle :

```
PutPixel(CenterX + X, Center Y + Y)
PutPixel(CenterX + X, Center Y - Y)
PutPixel(CenterX - X, Center Y + Y)
PutPixel(CenterX - X, Center Y - Y)
PutPixel(CenterX + Y, Center Y + X)
PutPixel(CenterX + Y, Center Y - X)
PutPixel(CenterX - Y, Center Y + X)
PutPixel(CenterX - Y, Center Y - X)
```

Algorithm :
Given a radius for the circle we perform this initialisation:

```
d := 3 - (2 * RADIUS)
x := 0
y := RADIUS
```

Now for each pixel we do the following operations until x = y :

```
Draw the 8 circle pixels
if d < 0 then
d := d + (4 * x) + 6
else
begin
d := d + 4 * (x - y) + 10
y := y - 1;
end;
```

Bresenham's Circle Generation Algorithm

Input to the Algorithm
(xc, yc) Integer Centre Co-ordinates and „r" radius.

Output of the Algorithm
Turning ON the pixels on circular boundary.

Data Variables

(xc,yc)	: Integer, representing center co-ordinates.
r	: Integer, radius of the circle
d	: Integer, decision variable.
x,y	: Integer, points used for turning ON the pixel.

```
Algorithm BresenhamCircleGen (xc,yc,r: Integer)
Step 1.  Start
Step 2.  x=0
Step 3.  y = r; { Initial point on the circle}
Step 4.  d = 3 - 2 * r; { Initialise decision variable}
Step 5.  if (x > y ) go to step no 17
Step 6.  Plot(xc + x, yc +y);
Step 7.  Plot(xc - x, yc + y);
Step 8.  Plot(xc + x, yc - y);
Step 9.  Plot(xc - x, yc - y);
Step 10. Plot(xc + y, yc + x);
Step 11. Plot(xc - Y,yc + x);
Step 12. Plot(xc +y, yc - x);
Step 13. Plot(xc - Y,yc - x); { Plot eight symmetric points}
Step 14. If (d > = 0) Then
{
d = d + 4 * (x - y) + 10;
y = y-1;
}
else
{
d = d + 4 * x + 6;
}
Step 15. x=x+1
Step 16. go to step no 5
Step 17. stop.
```

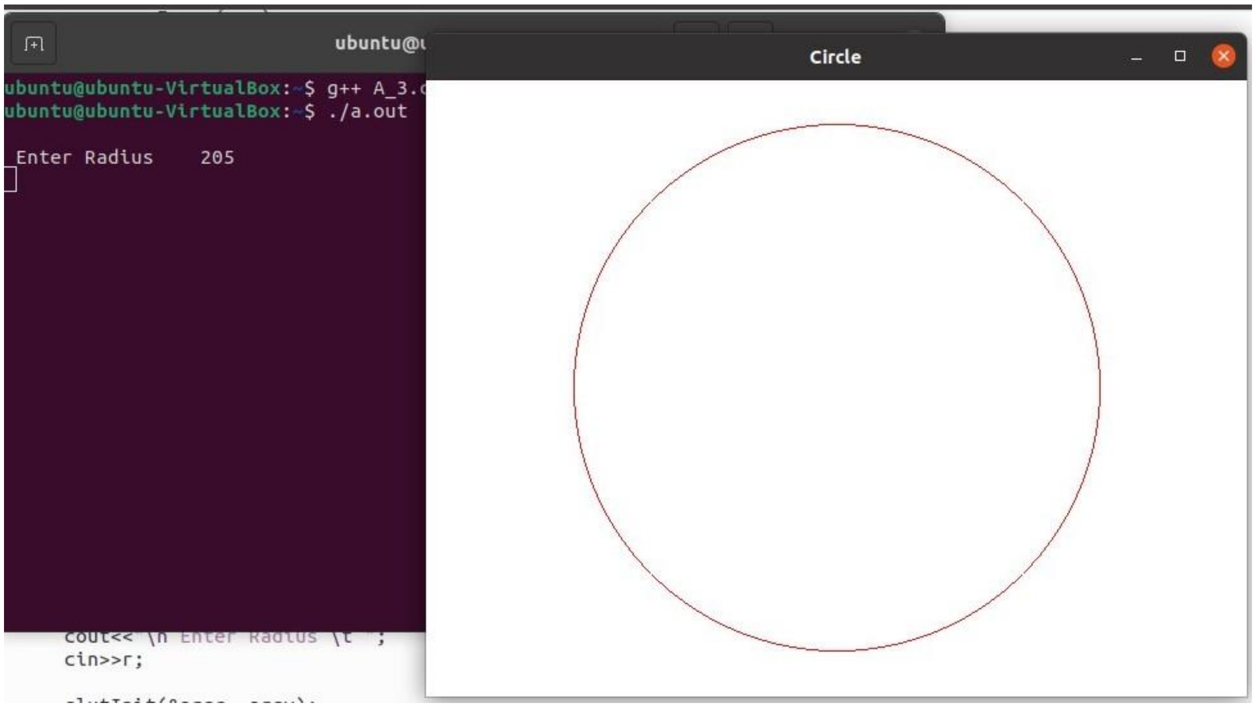
Code

```
#include<GL/glut.h>
#include<iostream>
using namespace std;

int r;

void E_way(int x, int y){

    glBegin(GL_POINTS);
      glVertex2i(x+320,y+240);
```

```
            glVertex2i(y+320,x+240);
            glVertex2i(y+320, -x+240);
            glVertex2i(x+320, -y+240);
            glVertex2i(-x+320,-y+240);
            glVertex2i(-y+320,-x+240);
            glVertex2i(-y+320,x+240);
            glVertex2i(-x+320,y+240);
        glEnd();
        glFlush();

}

void B_circle(){

    float d;
    d = 3 - 2*r;

    int x,y;
    x = 0 ;
    y = r ;

    do{
        E_way(x,y);

        if(d<0){

            d=d+4*x+6;
        }
        else{
            d= d+4*(x-y)+10;
            y=y-1;
        }
        x=x+1;

    }while(x<y);


}

void init(){

    glClearColor(1,1,1,0);
    glColor3f(1,0,0);
    gluOrtho2D(0,640,0,480);
    glClear(GL_COLOR_BUFFER_BIT);
}

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

    cout<<"\n Enter Radius \t ";
    cin>>r;

    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);

    glutInitWindowPosition(100,100);
    glutInitWindowSize(640,480);
    glutCreateWindow("Circle");
    init();
    glutDisplayFunc(B_circle);

    glutMainLoop();

    return 0;
}
```

OUTPUT

```
g++ filename.cpp -lGL -lGLU -lglut
./a.out
```



```
ubuntu@ubuntu-VirtualBox:~$ g++ A_3.c
ubuntu@ubuntu-VirtualBox:~$ ./a.out

 Enter Radius    205
```

```
    cout<<"\n Enter Radius \t";
    cin>>r;
```

Experiment No	4
Title	Implement the following polygon filling methods : i) Flood fill / Seed fill ii) Boundary fill ; using mouse click, keyboard interface and menu driven programming
Roll No.	
College	NSGIFOE , Pune
Class	S.E. (AIML)
Date	
Subject	Computer Graphics

Polygon Filling Approaches:

1. Scan Line fill approaches
2. Seed fill approaches

There are two types of seed fill algorithms.

1. Boundary Fill Algorithm

2. Flood Fill Algorithm.

1 Boundary Fill

Boundary Fill is algorithm used for the purpose of coloring figures in computer graphics. Boundary fill fills chosen area with a color until the given colored boundary is found.

Algorithm :

Step 1 : The boundary fill procedure accepts the input as coordinates of an interior point (x, y), a fill color, and a boundary color.

Step 2 : Starting from (x, y)which is seed pixel, the procedure tests the neighboring positions to determine whether they are boundary color.

Step 3 : If not, they are painted with the fill color, and the neighbors are tested.

Step 4 : This process continues until all pixels up to the boundary color for the area have been tested.

There are two methods for filling the pixel and find the neighbor pixel :

(i) 4-connected.

(ii) 8-connected.

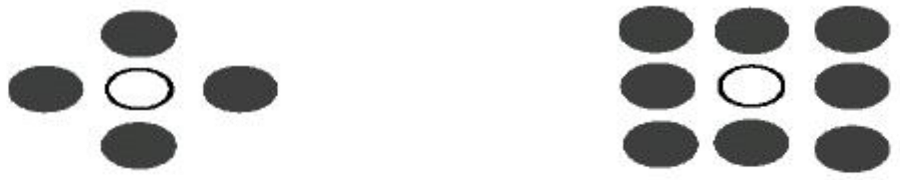


Fig.: Four connected method and Eight connected method

(i) 4-Connected Method :

Four_Fill (x, y, fill_col, bound_color)

if (curr_pixel_color != bound_color) and (curr_pixel_color != fill_col) then

set_pixel(x, y, fill_col)

Four_Fill (x+1, y, fill_col, bound_col);

```
Four_Fill (x-1, y, fill_col, bound_col);

Four_Fill (x, y+1, fill_col, bound_col);

Four_Fill( x, y-1, fill_col, bound_col);

end;
```

(ii) 8-Connected Method

The fill operation can proceed above, below, right and left side as well as through diagonal pixels of the current

pixels. This process will continue until we find a boundary with different color.

2 Flood Fill

The purpose of Flood Fill is to color an entire area of connected pixels with the same color.

Similarly boundary fill algorithm, we start with seed pixel, seed pixel is examined for specified interior color instead of boundary color

Pseudo Code for Flood fill Algorithm

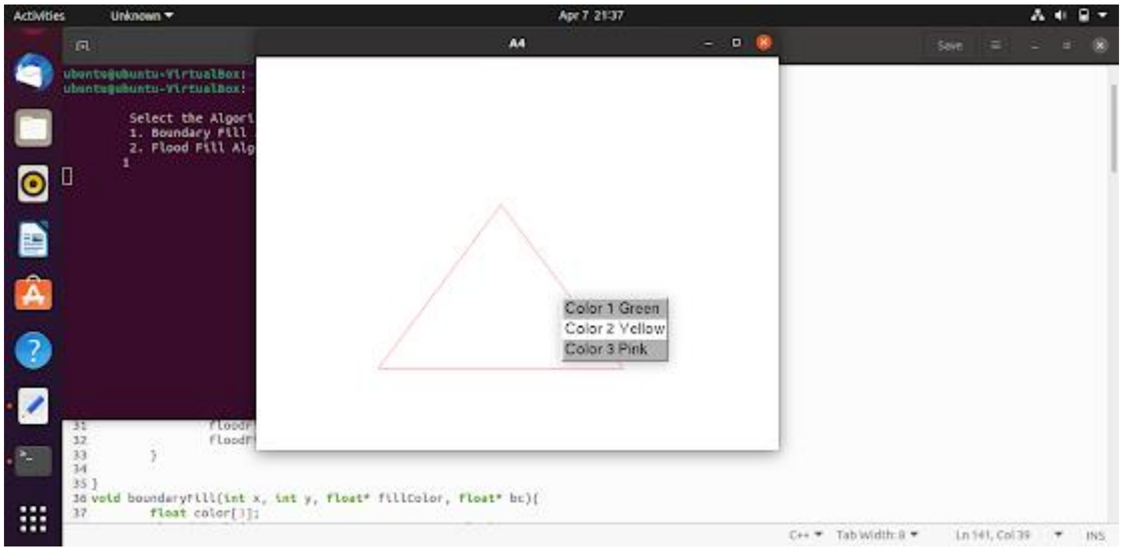
```
Flood-fill (node, old-color, replacement-color):
If the color of node is not equal to old-color, return.
Set the color of node to replacement-color.
Perform Flood-fill (one step to the left of node, old-color, replacement-color).
Perform Flood-fill (one step to the right of node, old-color, replacement-color).
Perform Flood-fill (one step to the top of node, old-color, replacement-color).
Perform Flood-fill (one step to the bottom of node, old-color, replacement-color)
```

Code And Output

```
#include <iostream>
#include <math.h>
#include <GL/glut.h>

using namespace std;

float R=0,G=0,B=0;
int Algo;

void init(){
    glClearColor(1.0,1.0,1.0,0.0);
    glMatrixMode(GL_PROJECTION);
    gluOrtho2D(0,640,0,480);
}


void floodFill(int x, int y, float *newCol, float *oldcol){
    float pixel[3];
    glReadPixels(x,y,1,1,GL_RGB,GL_FLOAT,pixel);
    if(oldcol[0]==pixel[0] && oldcol[1]==pixel[1] && oldcol[2]==pixel[2]){

        glBegin(GL_POINTS);
            glColor3f(newCol[0],newCol[1],newCol[2]);
            glVertex2i(x,y);
        glEnd();
        glFlush();

        floodFill(x,y+1,newCol,oldcol);
        floodFill(x+1,y,newCol,oldcol);
        floodFill(x,y-1,newCol,oldcol);
        floodFill(x-1,y,newCol,oldcol);
    }

}
void boundaryFill(int x, int y, float* fillColor, float* bc){
    float color[3];
    glReadPixels(x,y,1.0,1.0,GL_RGB,GL_FLOAT,color);

        if((color[0]!=bc[0] || color[1]!=bc[1] || color[2]!=bc[2]) && (fillColor[0]!=color[0] || fillColor[1]!=color[1]
|| fillColor[2]!=color[2])){

            glColor3f(fillColor[0],fillColor[1],fillColor[2]);
            glBegin(GL_POINTS);
                glVertex2i(x,y);
            glEnd();
            glFlush();
            boundaryFill(x+1,y,fillColor,bc);
            boundaryFill(x-1,y,fillColor,bc);
            boundaryFill(x,y+1,fillColor,bc);
            boundaryFill(x,y-1,fillColor,bc);


        }

     return;
}

void mouse(int btn, int state, int x, int y){

    y = 480-y;
      if(btn == GLUT_LEFT_BUTTON && state == GLUT_DOWN){

          float bcol[] = {1,0,0};
          float oldcol[] = {1,1,1};
          float newCol[] = {R,G,B};

          if(Algo==1){
              boundaryFill(x,y,newCol,bcol);
          }
          if(Algo==2){
              floodFill(x,y,newCol,oldcol);
        }
      }
}

void B_Draw(){

  glClear(GL_COLOR_BUFFER_BIT);
  glColor3f(1,0,0);
  glBegin(GL_LINE_LOOP);
    glVertex2i(150,100);
    glVertex2i(300,300);
    glVertex2i(450,100);
  glEnd();
  glFlush();

}

void F_Draw(){

  glClear(GL_COLOR_BUFFER_BIT);

  glBegin(GL_LINES);
    glColor3f(1,0,0);glVertex2i(150,100);glVertex2i(300,300);
  glEnd();
  glBegin(GL_LINE_LOOP);
    glColor3f(0,0,1);glVertex2i(300,300);glVertex2i(450,100);
  glEnd();
  glBegin(GL_LINE_LOOP);
    glColor3f(0,0,0);glVertex2i(450,100);glVertex2i(150,100);
  glEnd();
  glFlush();

}

void goMenu(int value){

  switch(value){

    case 1:
      R = 0, G = 1, B=0;
      break;
    case 2:
      R = 1, G = 1, B=0;
      break;
    case 3:
      R = 1, G = 0, B=1;
      break;

  }
  glutPostRedisplay();
}

int main(int argc, char** argv){
  cout<<"\n \t Select the Algorithm ";
  cout<<"\n \t 1. Boundary Fill Algorithm ";
  cout<<"\n \t 2. Flood Fill Algorithm \n \t";
  cin>>Algo;

  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
  glutInitWindowSize(640,480);
  glutInitWindowPosition(200,200);
  glutCreateWindow("A4");
  init();
  glutCreateMenu(goMenu);

   glutAddMenuEntry("Color 1 Green",1);
   glutAddMenuEntry("Color 2 Yellow",2);
   glutAddMenuEntry("Color 3 Pink",3);
   glutAttachMenu(GLUT_RIGHT_BUTTON);

  if(Algo==1){
    glutDisplayFunc(B_Draw);
  }
  if(Algo==2){
    glutDisplayFunc(F_Draw);
  }
  glutMouseFunc(mouse);
  glutMainLoop();
  return 0;
}
```

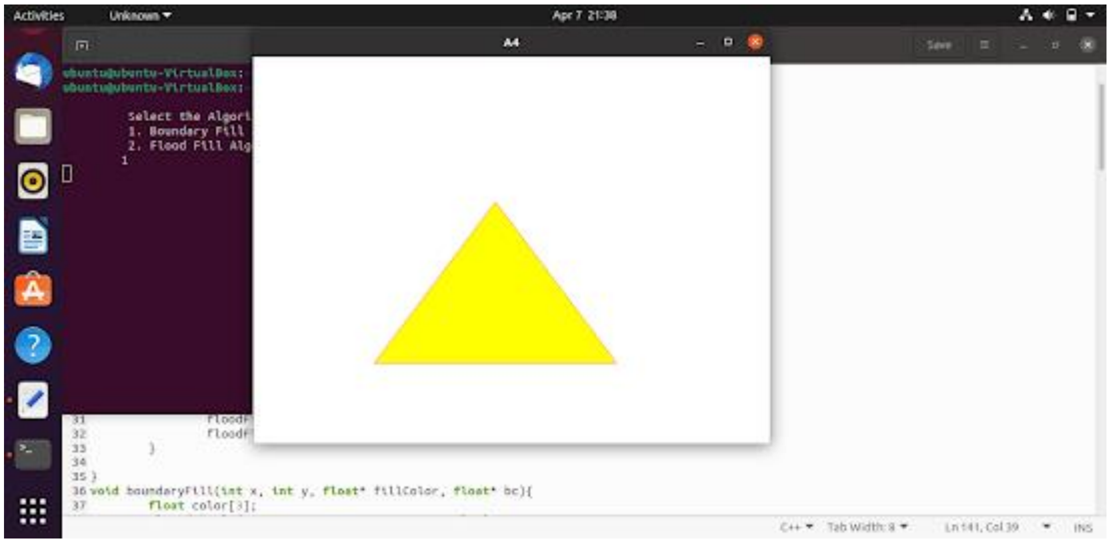
Output

```
g++ filename.cpp -lGL -lGLU -lglut
./a.out
```



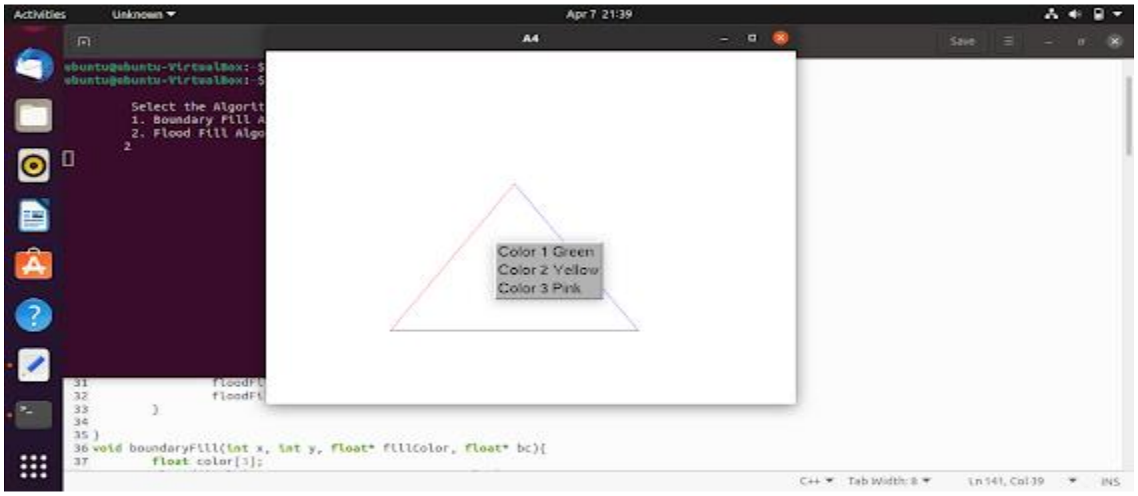
```
ubuntu@ubuntu-VirtualBox:
ubuntu@ubuntu-VirtualBox:

        Select the Algori
        1. Boundary Fill
        2. Flood Fill Alg
        1
```

```
                FloodF
                FloodF
        }

}
void boundaryFill(int x, int y, float* fillColor, float* bc){
        float color[3];
```



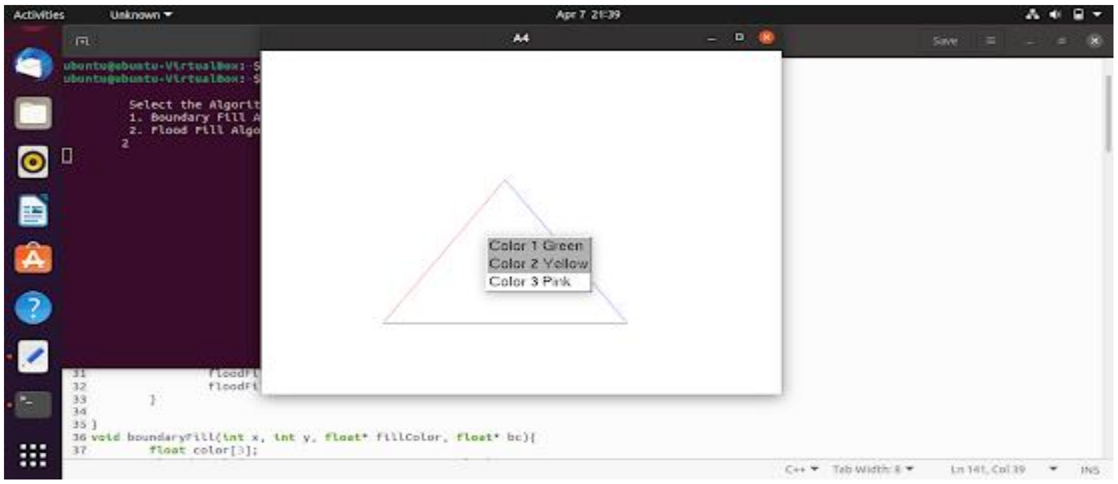
```
ubuntu@ubuntu-VirtualBox:
ubuntu@ubuntu-VirtualBox:

        Select the Algori
        1. Boundary Fill
        2. Flood Fill Alg
       1
```

```
                floodF
                floodF
        }

}
void boundaryFill(int x, int y, float* fillColor, float* bc){
        float color[3];
```



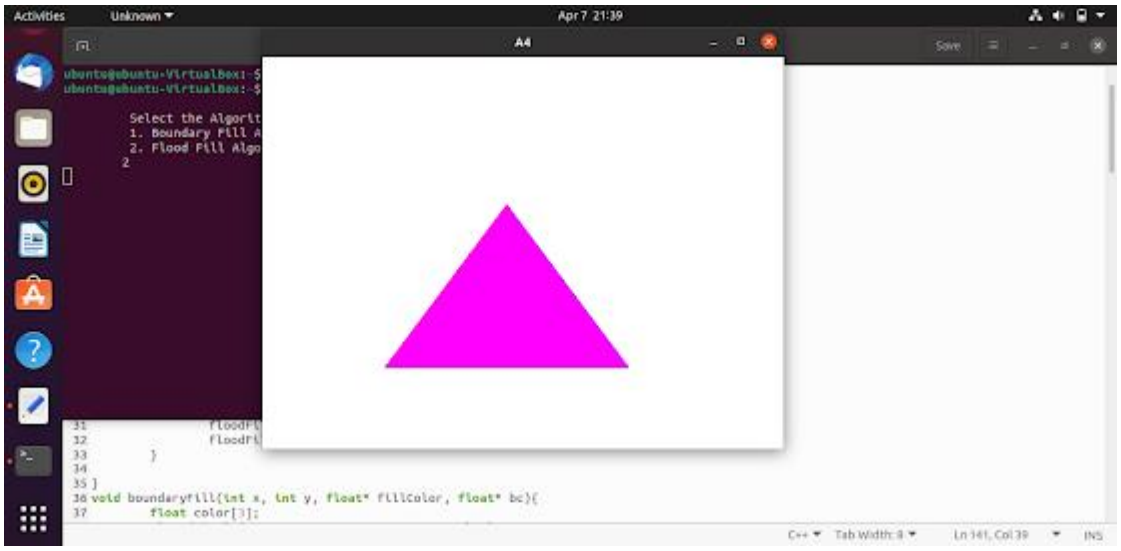
```
ubuntu@ubuntu-VirtualBox:~$
ubuntu@ubuntu-VirtualBox:~$

        Select the Algorit
        1. Boundary Fill A
        2. Flood Fill Algo
       2
```

```
                floodFi
                floodFi
        }

}
void boundaryFill(int x, int y, float* fillColor, float* bc){
        float color[3];
```



```
ubuntu@ubuntu-VirtualBox:~$
ubuntu@ubuntu-VirtualBox:~$

        Select the Algorit
        1. Boundary Fill A
        2. Flood Fill Algo
       2
```

```
                floodFi
                floodFi
        }

}
void boundaryFill(int x, int y, float* fillColor, float* bc){
        float color[3];
```



```
ubuntu@ubuntu-VirtualBox:~$
ubuntu@ubuntu-VirtualBox:~$

        Select the Algorit
        1. Boundary Fill A
        2. Flood Fill Algo
       2
```

```
                        floodFi
                        floodFi
                }

}
void boundaryFill(int x, int y, float* fillColor, float* bc){
        float color[3];
```

Experiment No	5
Title	Implement Cohen Sutherland polygon clipping metho to clip the polygon with respect the viewport and window. Use mouse click, keyboard interface
Roll No.	
College	NSGIFOE , Pune
Class	S.E. (AIML)
Date	
Subject	Computer Graphics

Line Clipping – Cohen Sutherland

In computer graphics, '***line clipping'*** is the process of removing lines or portions of lines outside of an area of interest. Typically, any line or part thereof which is outside of the viewing area is removed.

The Cohen–Sutherland algorithm is a computer graphics algorithm used for line clipping. The algorithm divides a two-dimensional space into 9 regions (or a three-dimensional space into 27 regions), and then efficiently determines the lines and portions of lines that are visible in the center region of interest (the viewport).

The algorithm was developed in 1967 during flight simulator work by Danny Cohen and Ivan Sutherland

The design stage includes, excludes or partially includes the line based on where:

- Both endpoints are in the viewport region (bitwise OR of endpoints == 0): trivial accept.
- Both endpoints share at least one non-visible region which implies that the line does not cross the visible region. (bitwise AND of endpoints != 0): trivial reject.
- Both endpoints are in different regions: In case of this nontrivial situation the algorithm finds one of the two points that is outside the viewport region (there will be at least one point outside). The intersection of the outpoint and extended viewport border is then calculated (i.e. with the parametric equation for the line) and this new point replaces the outpoint. The algorithm repeats until a trivial accept or reject occurs.

The numbers in the figure below are called outcodes. The outcode is computed for each of the two points in the line. The outcode will have four bits for two-dimensional clipping, or six bits in the three-dimensional case. The first bit is set to 1 if the point is above the viewport. The bits in the 2D outcode represent: Top, Bottom, Right, Left. For example the outcode 1010 represents a point that is top-right of the viewport. Note that the outcodes for endpoints **must** be recalculated on each iteration after the clipping occurs.

Sutherland Hodgman Polygon Clipping

It is used for clipping polygons. It works by extending each line of the convex *clip polygon* in turn and selecting only vertices from the *subject polygon* those are on the visible side.

An algorithm that clips a polygon must deal with many different cases. The case is particularly note worthy in that the concave polygon is clipped into two separate polygons. All in all, the task of clipping seems rather complex. Each edge of the polygon must be tested against each edge of the clip rectangle; new edges must be added, and existing edges must be discarded, retained, or divided. The algorithm begins with an input list of all vertices in the subject polygon. Next, one side of the clip polygon is extended infinitely in both directions, and the path of the subject polygon is traversed. Vertices from the input list are inserted into an output list if they lie on the visible side of the extended clip polygon line, and new vertices are added to the output list where the subject polygon path crosses the extended clip polygon line.

This process is repeated iteratively for each clip polygon side, using the output list from one stage as the input list for the next. Once all sides of the clip polygon have been processed, the final generated list of vertices defines a new single polygon that is entirely visible. Note that if the subject polygon was concave at vertices outside the clipping polygon, the new polygon may have coincident (i.e. overlapping) edges – this is acceptable for rendering, but not for other applications such as computing shadows.

The following example illustrate a simple case of polygon clipping

Sutherland and Hodgman's polygon-clipping algorithm uses a divide-and-conquer strategy: It solves a series of simple and identical problems that, when combined, solve the overall problem. The simple problem is to clip a polygon against a single infinite clip edge. Four clip edges, each defining one boundary of the clip rectangle, successively clip a polygon against a clip rectangle.

Note the difference between this strategy for a polygon and the Cohen-Sutherland algorithm for clipping a line: The polygon clipper clips against four edges in succession, whereas the line clipper tests the outcode to see which edge is crossed, and clips only when necessary.

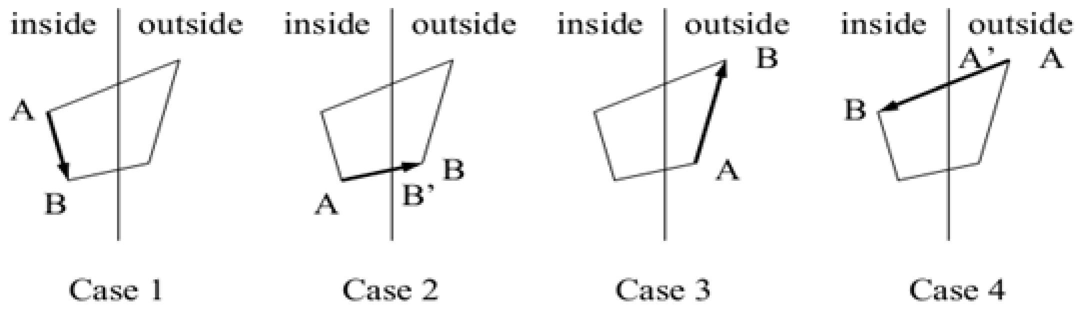
Steps of Sutherland-Hodgman's polygon-clipping algorithm

- Polygons can be clipped against each edge of the window one at a time. Windows/edge intersections, if any, are easy to find since the X or Y coordinates are already known.
- Vertices which are kept after clipping against one window edge are saved for clipping against the remaining edges.
- Note that the number of vertices usually changes and will often increases.
- We are using the Divide and Conquer approach

- After clipped by the right and bottom clip boundaries.

The original polygon and the clip rectangle.

Clipping polygons would seem to be quite complex. A single polygon can actually be split into multiple polygons .The Sutherland-Hodgman algorithm clips a polygon against all edges of the clipping region in turn. The algorithm steps from vertex to vertex, adding 0, 1, or 2 vertices to the output list at each step.



The Sutherland-Hodgeman Polygon-Clipping Algorithms clips a given polygon successively against the edges of the given clip-rectangle. These clip edges are denoted with e1, e2, e3, and e4, here. The closed polygon is represented by a list of its verteces (v1 to vn; Since we got 15 verteces in the example shown above, vn = v15.).

Clipping is computed successively for each edge. The output list of the previous clipping run is used as the inputlist for the next clipping run. 1st run: Clip edge: e1; inputlist = {v1, v2, ..., v14, v15}, the given polygon

```
#include <iostream>
#include <math.h>
#include <time.h>
#include <GL/glut.h>

using namespace std;

int wxmin = 200,wxmax=500,wymax=350, wymin=100;
int points[10][2];
int edge;

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 Draw(){
   glClearColor(1.0,1.0,1.0,0.0);
   glClear(GL_COLOR_BUFFER_BIT);
   glColor3f(0.2,0.2,1);
   glBegin(GL_POLYGON);
      for(int i=0; i<edge; i++)
      {
         glVertex2i(points[i][0],points[i][1]);
      }
   glEnd();
   glFlush();

   glColor3f(0,1,0);
   glBegin(GL_LINE_LOOP);
      glVertex2i(200,100);
      glVertex2i(500,100);
      glVertex2i(500,350);
      glVertex2i(200,350);
   glEnd();
   glFlush();
```

```
}
int BottomCliping(int e){

float m=0;
int x=0,k=0;
int t[10][2];

   for(int i=0; i<e; i++){
      if(points[i][1] < wymin){

         if(points[i+1][1]  < wymin){

         }
         else if(points[i+1][1] > wymin){
            float x1,x2;
            float y1,y2;
            x1 = points[i][0];
            y1 = points[i][1];
            x2 = points[i+1][0];
            y2 = points[i+1][1];
            x = ((1/((y2-y1)/(x2-x1))) * (wymin - y1) )+ x1;
            t[k][0] = x;
            t[k][1] = wymin;
            k++;

         }

      }
      else if(points[i][1]>wymin){

         if(points[i+1][1] > wymin){
            t[k][0] = points[i][0];
            t[k][1] = points[i][1];
            k++;
         }
         else if(points[i+1][1] < wymin){
            float x1,x2;
            float y1,y2;
            x1 = points[i][0];
            y1 = points[i][1];
            x2 = points[i+1][0];
            y2 = points[i+1][1];

            x = ((1/((y2-y1)/(x2-x1))) * (wymin - y1) )+ x1;

            t[k][0] = x1;
```

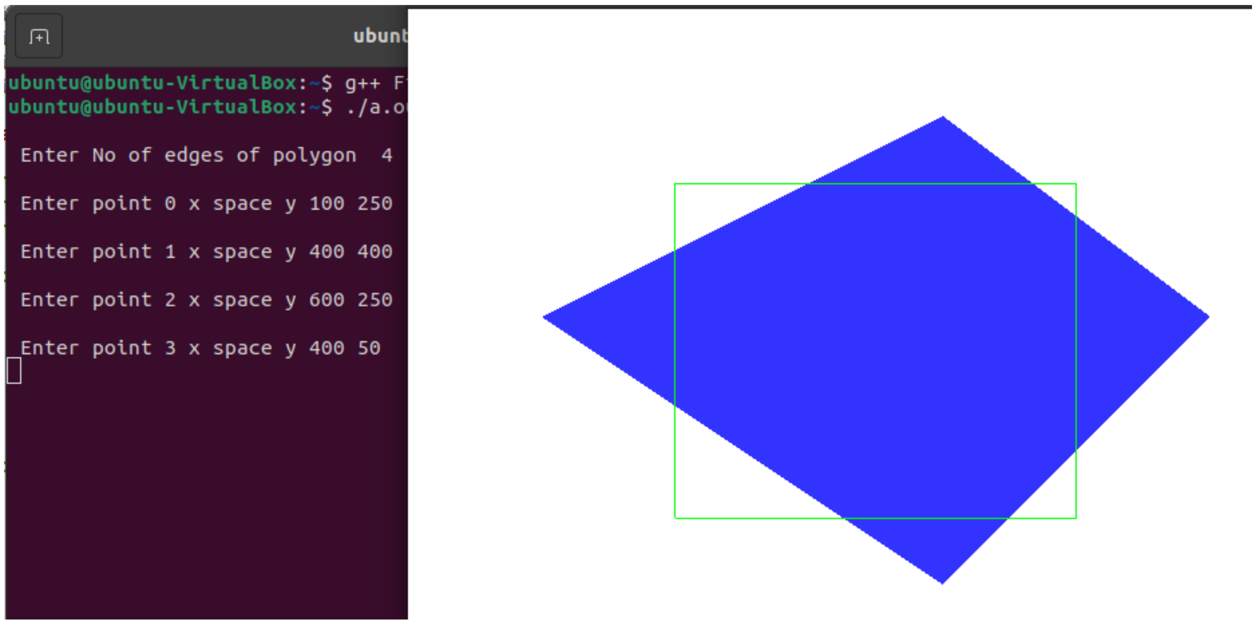
```
                t[k][1] = y1;
                k++;
                t[k][0] = x;
                t[k][1] = wymin;
                k++;

            }

        }

    }
    cout<<"k = "<<k;
    for(int i=0; i<10;i++)
    {
        points[i][0] = 0;
        points[i][1] = 0;

    }

    for(int i=0; i<k;i++)
    {
        cout<<"\n"<<t[i][0]<<" "<<t[i][1];
        points[i][0] = t[i][0];
        points[i][1] = t[i][1];

    }
    points[k][0] = points[0][0];
    points[k][1] = points[0][1];
    return k;

}




int TopCliping(int e){

float m=0;
int x=0,k=0;
int t[10][2];

    for(int i=0; i<e; i++){
        if(points[i][1] > wymax){

            if(points[i+1][1]  > wymax){
            }
            else if(points[i+1][1] < wymax){
                float x1,x2;
                float y1,y2;
                x1 = points[i][0];
                y1 = points[i][1];
                x2 = points[i+1][0];
                y2 = points[i+1][1];
                x = ((1/((y2-y1)/(x2-x1))) * (wymax - y1) )+ x1;
                t[k][0] = x;
                t[k][1] = wymax;
                k++;

            }

        }
        else if(points[i][1]<wymax){

            if(points[i+1][1] < wymax){
                t[k][0] = points[i][0];
                t[k][1] = points[i][1];
                k++;
            }
            else if(points[i+1][1] > wymax){
                float x1,x2;
                float y1,y2;
                x1 = points[i][0];
                y1 = points[i][1];
                x2 = points[i+1][0];
                y2 = points[i+1][1];

                x = ((1/((y2-y1)/(x2-x1))) * (wymax - y1) )+ x1;

                t[k][0] = x1;
                t[k][1] = y1;
                k++;
                t[k][0] = x;
                t[k][1] = wymax;
                k++;


            }

        }

    }
    cout<<"k = "<<k;
```

```
    for(int i=0; i<10;i++)
    {
        points[i][0] = 0;
        points[i][1] = 0;

    }

    for(int i=0; i<k;i++)
    {
        cout<<"\n"<<t[i][0]<<" "<<t[i][1];
        points[i][0] = t[i][0];
        points[i][1] = t[i][1];

    }
    points[k][0] = points[0][0];
    points[k][1] = points[0][1];
    return k;

}

int leftCliping(int e){

float m=0;
int y=0, k = 0;
int t[10][2];
    for(int i=0;i<e;i++)
    {

        if(points[i][0] < wxmin){

            if(points[i+1][0] < wxmin){
                cout<<"\n Test 1";

            }
            else if (points[i+1][0] > wxmin){
                cout<<"\n Test 2";
                float x1,x2;
                float y1,y2;
                x1 = points[i][0];
                y1 = points[i][1];
                x2 = points[i+1][0];
                y2 = points[i+1][1];
                y = (((y2-y1)/(x2-x1)) * (wxmin - x1) )+ y1;
                t[k][0] = wxmin;
                t[k][1] = y;
                k++;
            }
        }
        else if(points[i][0] > wxmin){

            if(points[i+1][0] > wxmin){

                t[k][0] = points[i][0];
                t[k][1] = points[i][1];
                k++;
            }
            else if(points[i+1][0] < wxmin){


                float x1,x2;
                float y1,y2;
                x1 = points[i][0];
                y1 = points[i][1];
                x2 = points[i+1][0];
                y2 = points[i+1][1];

                y = ((y2-y1)/(x2-x1)*(wxmin - x1)) + y1;

                t[k][0] = x1;
                t[k][1] = y1;
                k++;
                t[k][0] = wxmin;
                t[k][1] = y;
                k++;
            }


        }
    }
    cout<<"k = "<<k;
    for(int i=0; i<10;i++)
    {
        points[i][0] = 0;
        points[i][1] = 0;

    }

    for(int i=0; i<k;i++)
    {
        cout<<"\n"<<t[i][0]<<" "<<t[i][1];
        points[i][0] = t[i][0];
        points[i][1] = t[i][1];

    }
```

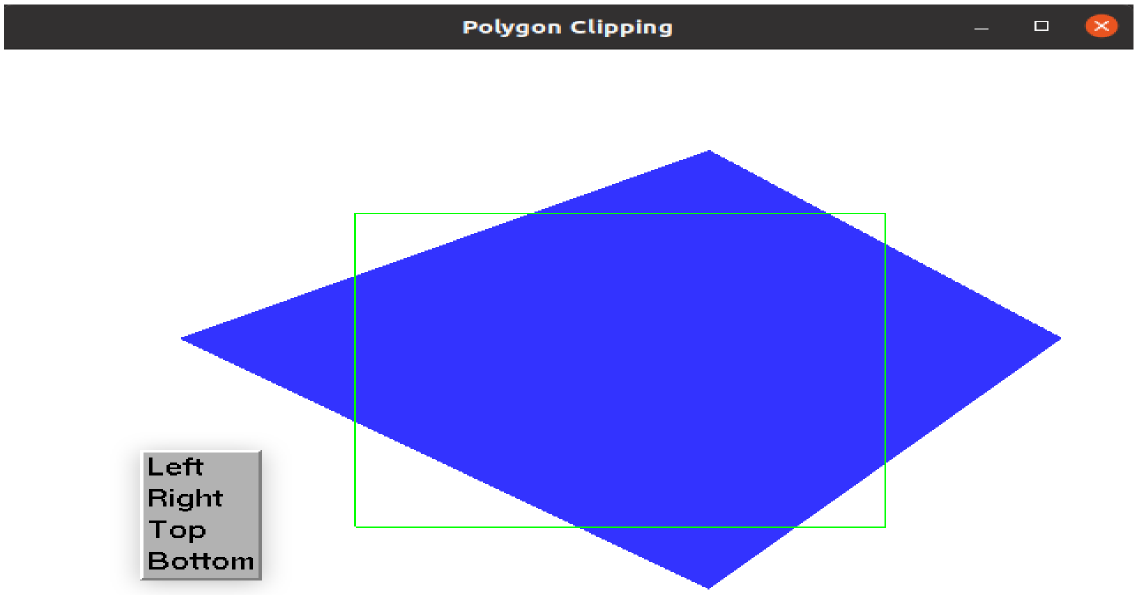
```
    points[k][0] = points[0][0];
    points[k][1] = points[0][1];
    return k;
}

int RightCliping(int e){

float m=0;
int y=0, k = 0;
int t[10][2];
    for(int i=0;i<e;i++)
    {

        if(points[i][0] > wxmax){

            if(points[i+1][0] > wxmax){


            }
            else if(points[i+1][0] < wxmax){

                float x1,x2;
                float y1,y2;
                x1 = points[i][0];
                y1 = points[i][1];
                x2 = points[i+1][0];
                y2 = points[i+1][1];
                y = (((y2-y1)/(x2-x1)) * (wxmax - x1) )+ y1;
                t[k][0] = wxmax;
                t[k][1] = y;
                k++;
            }

        }
        else if(points[i][0] < wxmax){

            if(points[i+1][0] < wxmax){

                t[k][0] = points[i][0];
                t[k][1] = points[i][1];
                k++;
            }
            else if(points[i+1][0] > wxmax){

                float x1,x2;
                float y1,y2;
                x1 = points[i][0];
                y1 = points[i][1];
                x2 = points[i+1][0];
                y2 = points[i+1][1];

                y = ((y2-y1)/(x2-x1)*(wxmax - x1)) + y1;
                t[k][0] = x1;
                t[k][1] = y1;
                k++;
                t[k][0] = wxmax;
                t[k][1] = y;
                k++;
            }
        }
    }
    cout<<"k = "<<k;
    for(int i=0; i<10;i++)
    {
        points[i][0] = 0;
        points[i][1] = 0;

    }
    for(int i=0; i<k;i++)
    {
        cout<<"\n"<<t[i][0]<<" "<<t[i][1];
        points[i][0] = t[i][0];
        points[i][1] = t[i][1];

    }
    points[k][0] = points[0][0];
    points[k][1] = points[0][1];
    return k;
}

void P_C(){

    Draw();
}


void goMenu(int value){

    switch(value){

        case 1:
            edge = leftCliping(edge);
            Draw();
            break;
        case 2:
            edge = RightCliping(edge);
            Draw();
            break;
        case 3:
            edge = TopCliping(edge);
            Draw();
            break;
        case 4:
            edge = BottomCliping(edge);
            Draw();
            break;

    }
    glutPostRedisplay();
}

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


    cout<<"\n Enter No of edges of polygon  ";
    cin>>edge;

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

        cout<<"\n Enter point "<<i<<" x space y ";
        cin>>points[i][0]>>points[i][1];

    }
    points[edge][0] = points[0][0];
    points[edge][1] = points[0][1];
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutInitWindowSize(640,480);
    glutInitWindowPosition(200,200);
    glutCreateWindow("Polygon Clipping");
    init();

    glutCreateMenu(goMenu);

         glutAddMenuEntry("Left",1);
         glutAddMenuEntry("Right",2);
         glutAddMenuEntry("Top",3);
         glutAddMenuEntry("Bottom",4);
         glutAttachMenu(GLUT_RIGHT_BUTTON);

        glutDisplayFunc(P_C);
    glutMainLoop();
    return 0;
}
```

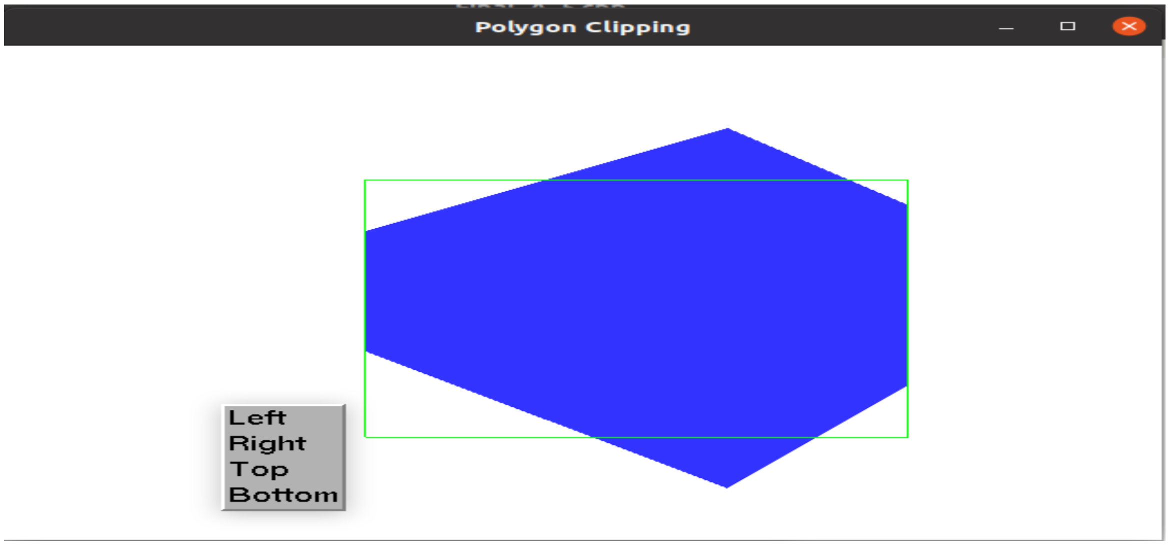
Out put

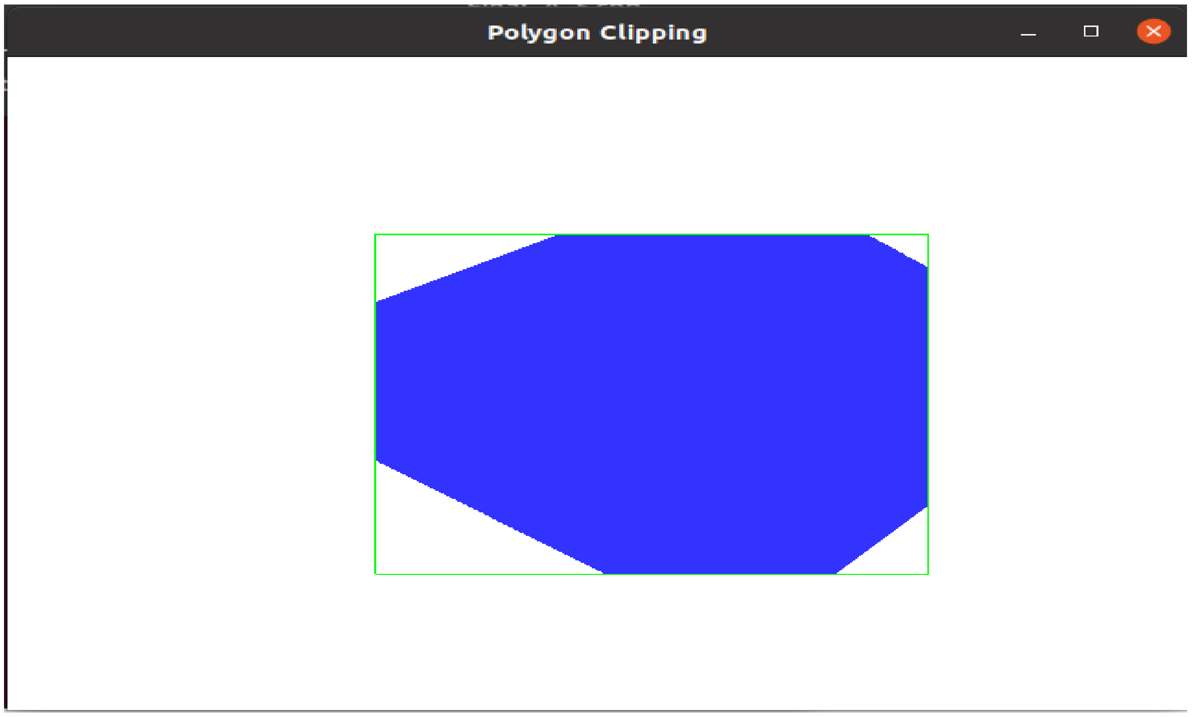
```
g++ filename.cpp -lGL -lGLU -lglut
./a.out
```



```
ubuntu@ubuntu-VirtualBox:~$ g++ F
ubuntu@ubuntu-VirtualBox:~$ ./a.o

 Enter No of edges of polygon  4

 Enter point 0 x space y 100 250

 Enter point 1 x space y 400 400

 Enter point 2 x space y 600 250

 Enter point 3 x space y 400 50
```



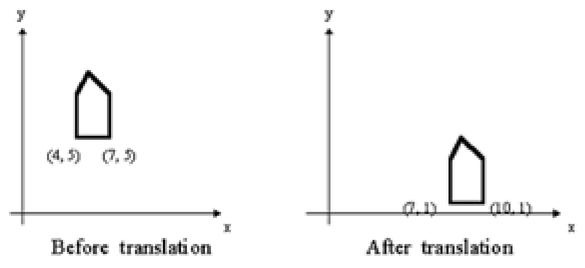

Polygon Clipping
Left
Right
Top
Bottom


Polygon Clipping

Experiment No	6
Title	Implement following 2D transformations on the obje with respect to axis : i) Scaling ii) Rotation about arbitrary point iii) Reflecti
Roll No.	
College	NSGIFOE , Pune
Class	S.E. (AIML)
Date	
Subject	Computer Graphics

Translation: Translation is defined as moving the object from one position to another *position along*

straight line path.



We can move the objects based on translation distances along x and y axis. tx denotes translation distance along x-axis and ty denotes translation distance along y axis.

Translation Distance: It is nothing but by how much units we should shift the object from one location to another along x, y-axis.

Consider (x,y) are old coordinates of a point. Then the new coordinates of that same point (x',y') can be obtained as follows:

X'=x+tx

Y'=y+ty

Scaling: scaling refers to changing the size of the object either by increasing or decreasing. We will increase or decrease the size of the object based on scaling factors along x and y -axis.

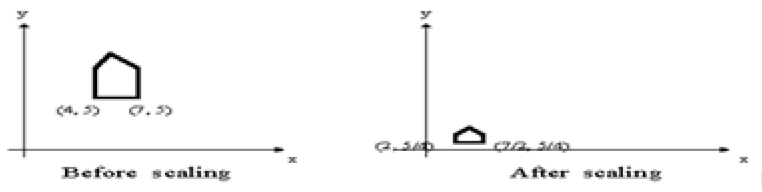
If (x, y) are old coordinates of object, then new coordinates of object after applying scaling

transformation are obtained as:

x'=x*sx

y'=y*sy.

sx and sy are scaling factors along x-axis and y-axis. we express the above equations in matrix form as:



Rotation : A rotation repositions all points in an object along a circular path in the plane centered at the

pivot point. We rotate an object by an angle theta

New coordinates after rotation depend on both x and y

$x' = x\cos\theta - y\sin\theta$

$y' = x\sin\theta + y\cos\theta$

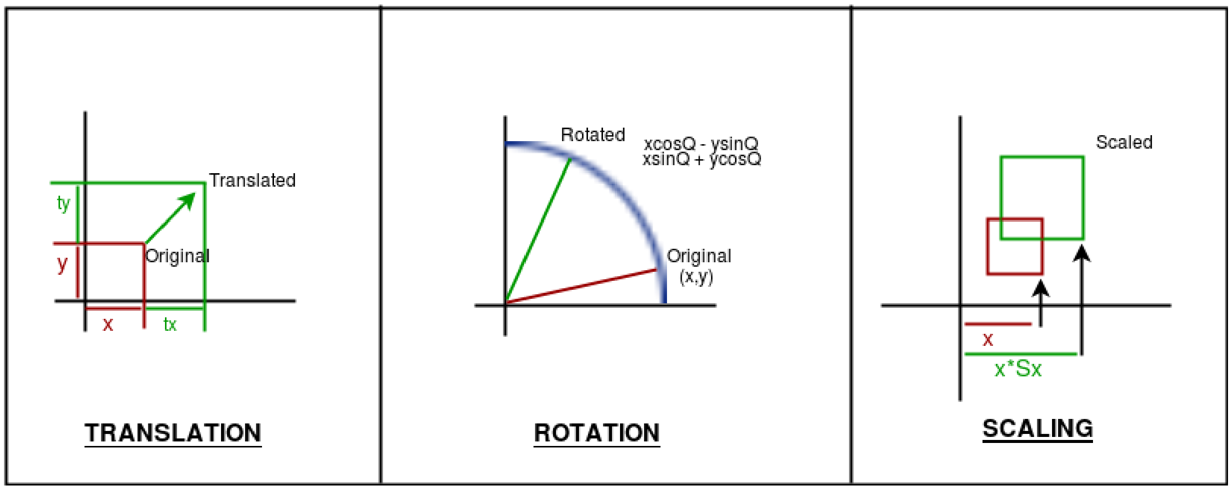
or in matrix form:

P' = R • P,

R-rotation matrix.

$$R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

```
Formula:  X = xcosA - ysinA
          Y = xsinA + ycosA,
   A is the angle of rotation.
The above formula will rotate the point around the origin.
To rotate around a different point, the formula:
          X = cx + (x-cx)*cosA - (y-cy)*sinA,
          Y = cx + (x-cx)*sinA + (y-cy)*cosA,
                 cx, cy is centre coordinates,
                 A is the angle of rotation.

The OpenGL function is glRotatef (A, x, y, z).
```



Reflection: It is a transformation which produces a mirror image of an object. The mirror image can be either about x-axis or y-axis. The object is rotated by180°.

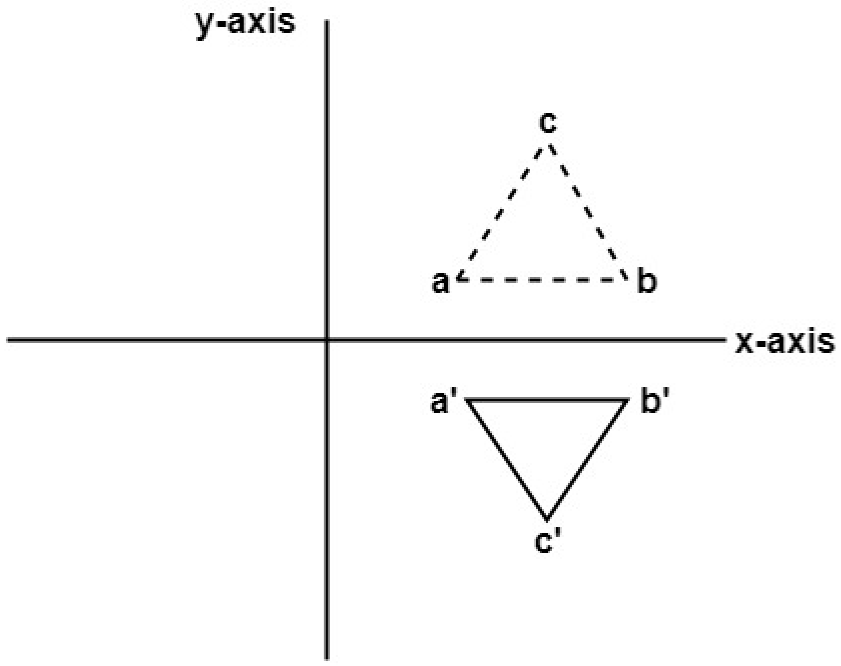
Types of Reflection:

1. Reflection about the x-axis
2. Reflection about the y-axis
3. Reflection about an axis perpendicular to xy plane and passing through the origin
4. Reflection about line y=x

1. Reflection about x-axis: The object can be reflected about x-axis with the help of the following matrix

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

In this transformation value of x will remain same whereas the value of y will become negative. Following figures shows the reflection of the object axis. The object will lie another side of the x-axis.

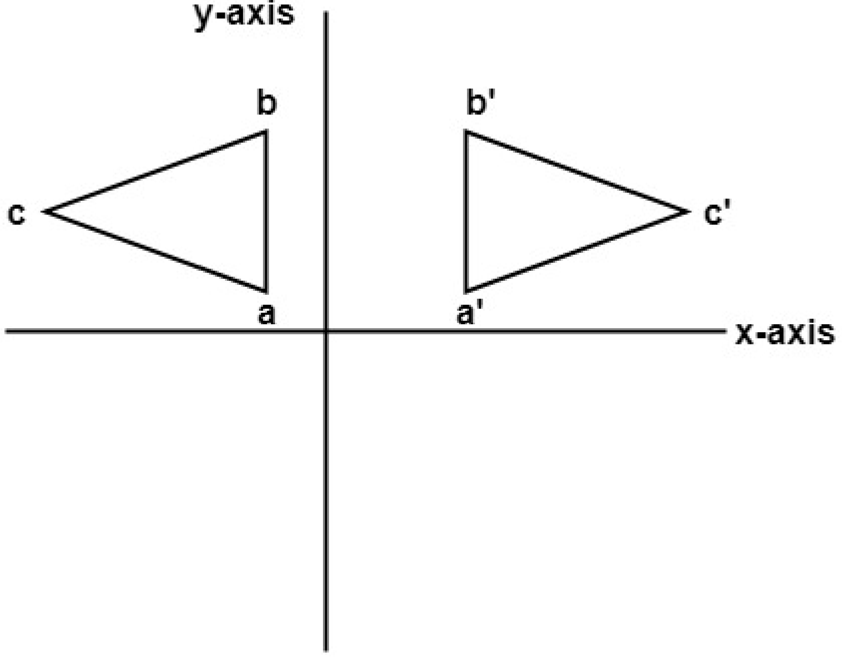


2. Reflection about y-axis: The object can be reflected about y-axis with the help of following transformation matrix

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

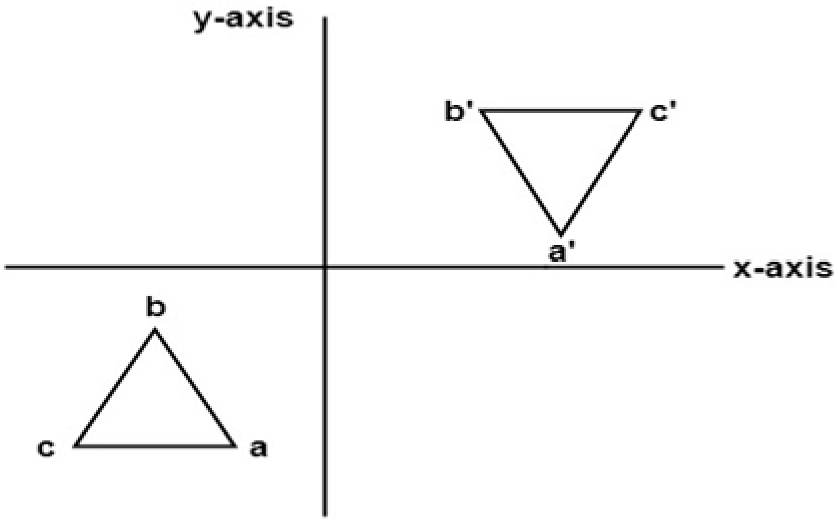
Here the values of x will be reversed, whereas the value of y will remain the same. The object will lie another side of the y-axis.

The following figure shows the reflection about the y-axis



3. Reflection about an axis perpendicular to xy plane and passing through origin:
In the matrix of this transformation is given below

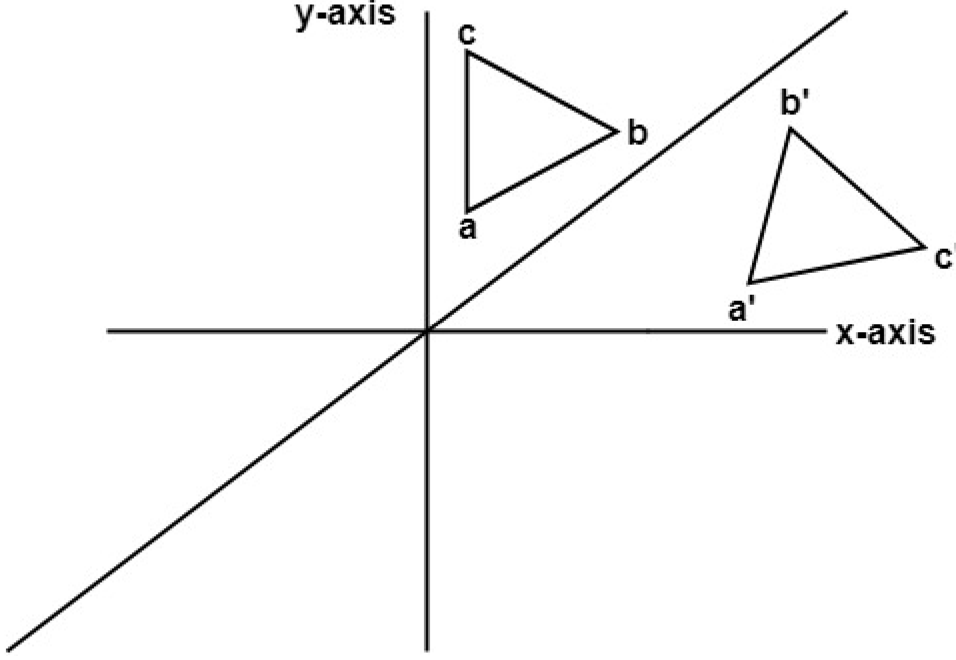
$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



In this value of x and y both will be reversed. This is also called as half revolution about the origin.

4. Reflection about line y=x: The object may be reflected about line y = x with the help of following transformation matrix

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



First of all, the object is rotated at 45°. The direction of rotation is clockwise. After it reflection is done concerning x-axis. The last step is the rotation of y=x back to its original position that is counterclockwise at 45°.

Example: A triangle ABC is given. The coordinates of A, B, C are given as

A (3 4)
B (6 4)
C (4 8)

Find reflected position of triangle i.e., to the x-axis

CODE

```
 #include <iostream>
#include <math.h>
#include <time.h>
#include <GL/glut.h>
#include <vector>

using namespace std;

int edge;
```

```
vector<int> xpoint;
vector<int> ypoint;

int ch;

double round(double d){

   return floor(d + 0.5);
}

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 translation(){
   int tx, ty;
   cout<<"\t Enter Tx, Ty \n";
   cin>> tx>> ty;

   //Translate the point
   for(int i=0;i<edge;i++){

      xpoint[i] = xpoint[i] + tx;
      ypoint[i] = ypoint[i] + ty;

   }


   glBegin(GL_POLYGON);
      glColor3f(0,0,1);
      for(int i=0;i<edge;i++){
         glVertex2i(xpoint[i],ypoint[i]);
      }
   glEnd();
   glFlush();
}

void rotaion(){
   int cx, cy;
   cout<<"\n Enter Ar point x , y ";
   cin >> cx >> cy;

   cx = cx+320;
   cy = cy+240;
   glColor3f(0.0, 1.0, 0.0);
   glBegin(GL_POINTS);
```

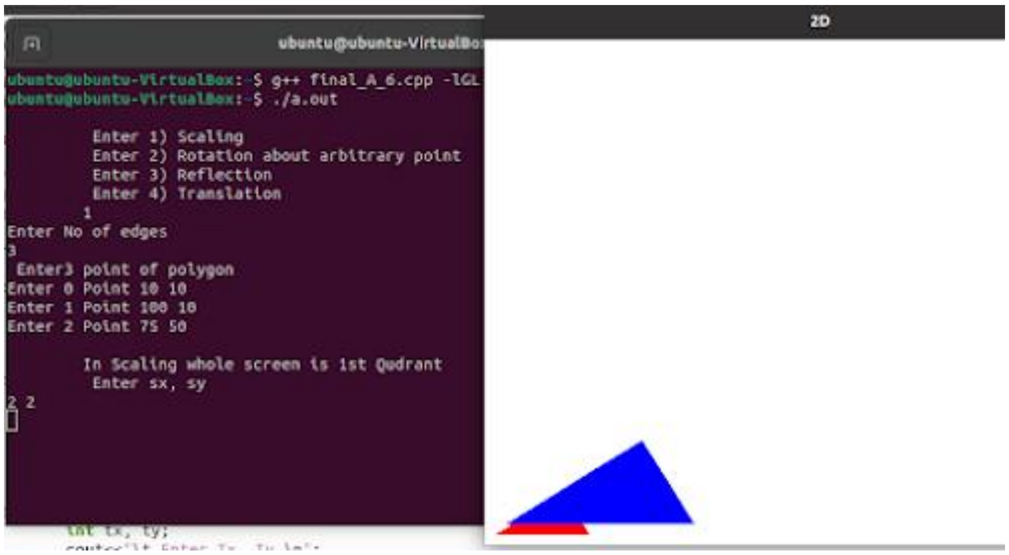
```
        glVertex2i(cx,cy);
    glEnd();
    glFlush();

    double the;
    cout<<"\n Enter thetha ";
    cin>>the;
    the = the * 3.14/180;

    glColor3f(0,0,1.0);
    glBegin(GL_POLYGON);
        for(int i=0;i<edge;i++){
            glVertex2i(round(((xpoint[i] - cx)*cos(the) - ((ypoint[i]-cy)*sin(the))) + cx),
                   round(((xpoint[i] - cx)*sin(the) + ((ypoint[i]-cy)*cos(the))) + cy));
        }
    glEnd();
    glFlush();
}

void scale(){

    glColor3f(1.0,0,0);
    glBegin(GL_POLYGON);
        for(int i=0;i<edge;i++){
            glVertex2i(xpoint[i]-320,ypoint[i]-240);
        }
    glEnd();
    glFlush();
    cout<<"\n\tIn Scaling whole screen is 1st Qudrant \n";
    int sx, sy;
    cout<<"\t Enter sx, sy \n";
    cin>> sx>> sy;

    //scale the point
    for(int i=0;i<edge;i++){

        xpoint[i] = (xpoint[i]-320) * sx;
        ypoint[i] = (ypoint[i]-240) * sy;
    }

    glColor3f(0,0,1.0);
    glBegin(GL_POLYGON);
        for(int i=0;i<edge;i++){
            glVertex2i(xpoint[i],ypoint[i]);
        }
    glEnd();
    glFlush();
}
```

```
void reflection(){
    char reflection;
    cout<<"Enter Reflection Axis \n";
    cin>> reflection;

    if(reflection == 'x' || reflection == 'X'){

        glColor3f(0.0,0.0,1.0);
        glBegin(GL_POLYGON);
            for(int i=0;i<edge;i++){
                glVertex2i(xpoint[i], (ypoint[i] * -1)+480);
            }
        glEnd();
        glFlush();

    }
    else if(reflection == 'y' || reflection == 'Y'){
        glColor3f(0.0,0.0,1.0);
        glBegin(GL_POLYGON);
            for(int i=0;i<edge;i++){
                glVertex2i((xpoint[i] * -1)+640,(ypoint[i]));
            }
        glEnd();
        glFlush();
    }
}

void Draw(){

    if(ch==2 || ch==3 || ch==4){
        glColor3f(1.0,0,0);
        glBegin(GL_LINES);
            glVertex2i(0,240);
            glVertex2i(640,240);
        glEnd();
        glColor3f(1.0,0,0);
        glBegin(GL_LINES);
            glVertex2i(320,0);
            glVertex2i(320,480);
        glEnd();
        glFlush();

        glColor3f(1.0,0,0);
        glBegin(GL_POLYGON);
            for(int i=0;i<edge;i++){
                glVertex2i(xpoint[i],ypoint[i]);
            }
        glEnd();
        glFlush();
    }
    if(ch==1){
       scale();
    }
    else if(ch == 2){
       rotaion();
    }
    else if( ch == 3){
       reflection();
    }
    else if (ch == 4){
       translation();
    }
}

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

    cout<<"\n \t Enter 1) Scaling ";
    cout<<"\n \t Enter 2) Rotation about arbitrary point";
    cout<<"\n \t Enter 3) Reflection";
    cout<<"\n \t Enter 4) Translation  \n \t";

    cin>>ch;

    if(ch==1 || ch==2 || ch==3 || ch==4){

       cout<<"Enter No of edges \n";
       cin>> edge;

       int xpointnew, ypointnew;
       cout<<" Enter"<< edge <<" point of polygon \n";
       for(int i=0;i<edge;i++){

          cout<<"Enter "<< i << " Point ";
          cin>>xpointnew>>ypointnew;

          xpoint.push_back(xpointnew+320);
          ypoint.push_back(ypointnew+240);

       }
          glutInit(&argc, argv);
          glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
          glutInitWindowSize(640,480);
          glutInitWindowPosition(200,200);
          glutCreateWindow("2D");
          init();
          glutDisplayFunc(Draw);

       glutMainLoop();
```

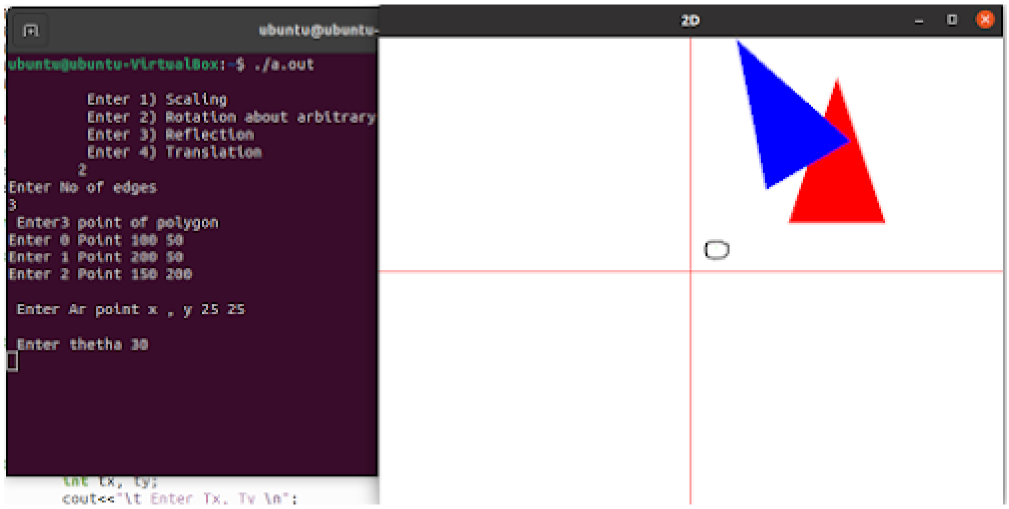
```
            return 0;
        }
        else{
            cout<<"\n \t Check Input run again";
        return 0;
    }
}
```

OUTPUT

```
g++ filename.cpp -lGL -lGLU -lglut
./a.out
```

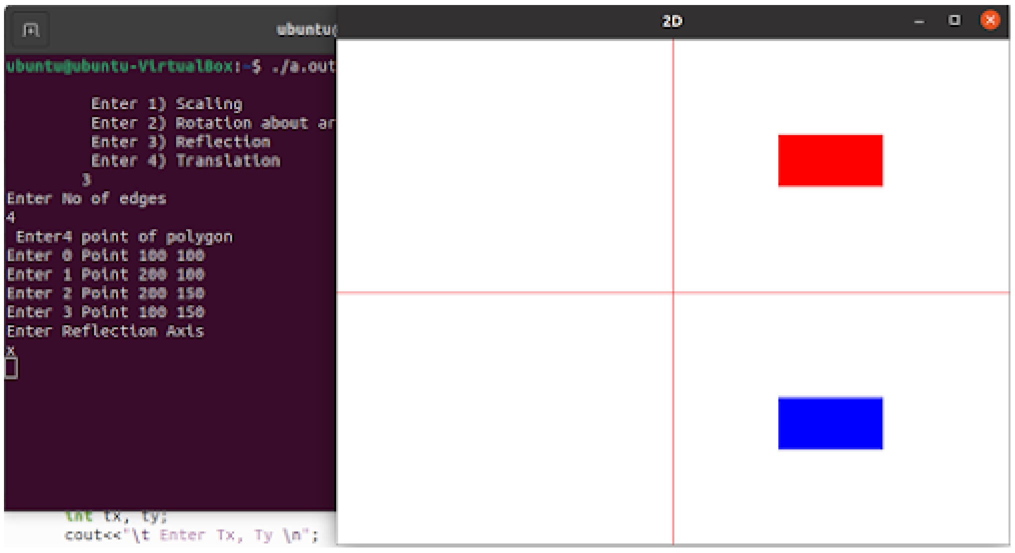


```
ubuntu@ubuntu-VirtualBox:~$ g++ final_A_6.cpp -lGL
ubuntu@ubuntu-VirtualBox:~$ ./a.out

         Enter 1) Scaling
         Enter 2) Rotation about arbitrary point
         Enter 3) Reflection
         Enter 4) Translation
        1
Enter No of edges
3
 Enter3 point of polygon
Enter 0 Point 10 10
Enter 1 Point 100 10
Enter 2 Point 75 50

        In Scaling whole screen is 1st Qudrant
         Enter sx, sy
2 2
```

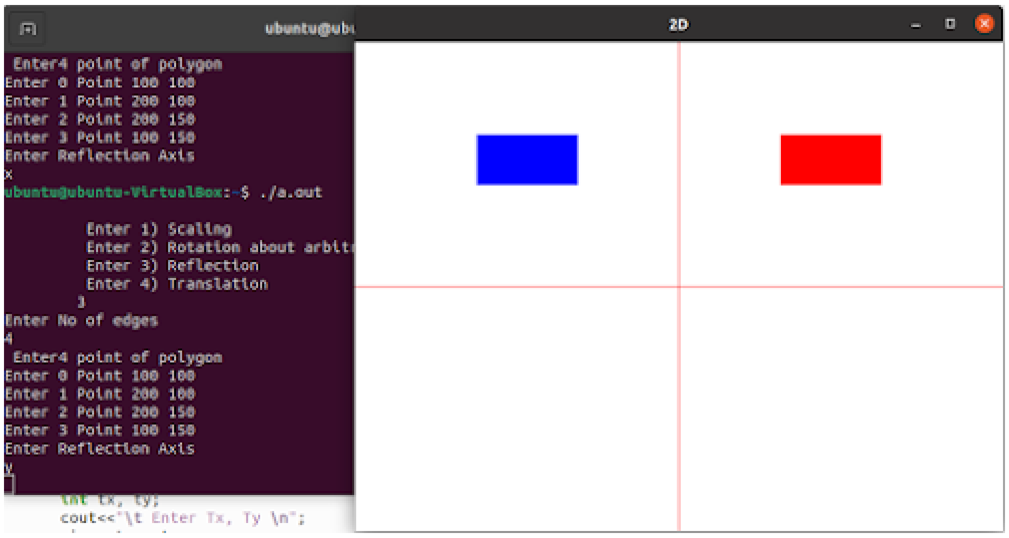


```
ubuntu@ubuntu-VirtualBox:~$ ./a.out

        Enter 1) Scaling
        Enter 2) Rotation about arbitrary
        Enter 3) Reflection
        Enter 4) Translation
       2
Enter No of edges
3
 Enter3 point of polygon
Enter 0 Point 100 50
Enter 1 Point 200 50
Enter 2 Point 150 200

 Enter Ar point x , y 25 25

 Enter thetha 30
```

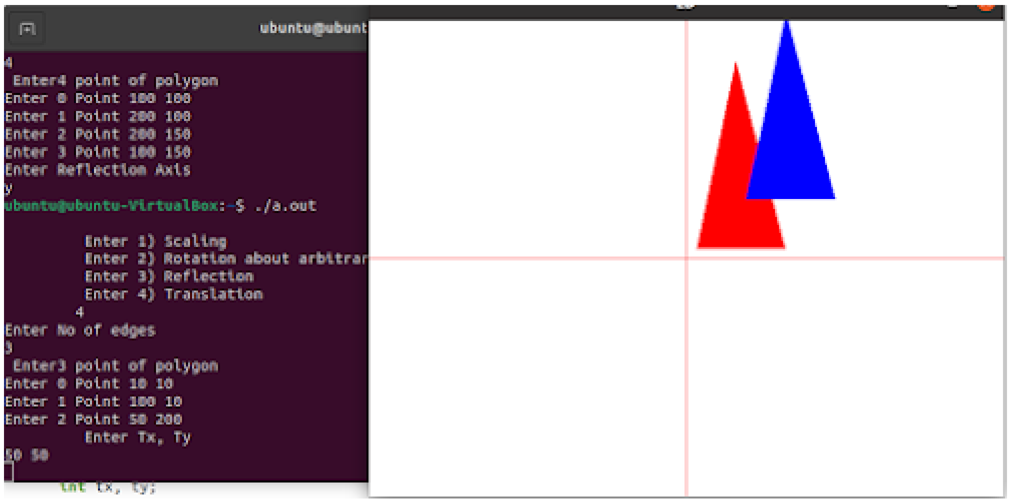
```
int tx, ty;
cout<<"\t Enter Tx, Ty \n";
```



```
ubuntu@ubuntu-VirtualBox:~$ ./a.out

         Enter 1) Scaling
         Enter 2) Rotation about ar
         Enter 3) Reflection
         Enter 4) Translation
        3
Enter No of edges
4
 Enter4 point of polygon
Enter 0 Point 100 100
Enter 1 Point 200 100
Enter 2 Point 200 150
Enter 3 Point 100 150
Enter Reflection Axis
x

      int tx, ty;
      cout<<"\t Enter Tx, Ty \n";
```



```
 Enter4 point of polygon
Enter 0 Point 100 100
Enter 1 Point 200 100
Enter 2 Point 200 150
Enter 3 Point 100 150
Enter Reflection Axis
x
ubuntu@ubuntu-VirtualBox:~$ ./a.out

         Enter 1) Scaling
         Enter 2) Rotation about arbit
         Enter 3) Reflection
         Enter 4) Translation
        3
Enter No of edges
4
 Enter4 point of polygon
Enter 0 Point 100 100
Enter 1 Point 200 100
Enter 2 Point 200 150
Enter 3 Point 100 150
Enter Reflection Axis
y

      int tx, ty;
      cout<<"\t Enter Tx, Ty \n";
```



```
4
 Enter4 point of polygon
Enter 0 Point 100 100
Enter 1 Point 200 100
Enter 2 Point 200 150
Enter 3 Point 100 150
Enter Reflection Axis
y
ubuntu@ubuntu-VirtualBox:~$ ./a.out

         Enter 1) Scaling
         Enter 2) Rotation about arbitrar
         Enter 3) Reflection
         Enter 4) Translation
        4
Enter No of edges
3
 Enter3 point of polygon
Enter 0 Point 10 10
Enter 1 Point 100 10
Enter 2 Point 50 200
         Enter Tx, Ty
50 50

      int tx, ty;
```

Experiment No	7
Title	Generate fractal patterns using i) Bezier ii) Koch Curve
Roll No.	
College	NSGIFOE , Pune
Class	S.E. (AIML)
Date	
Subject	Computer Graphics

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 −

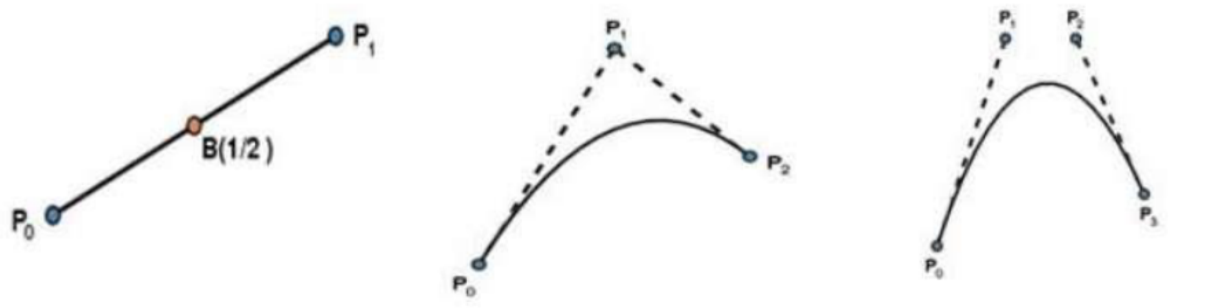
∑k=0nPiBni(t)∑k=0nPiBin(t)

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

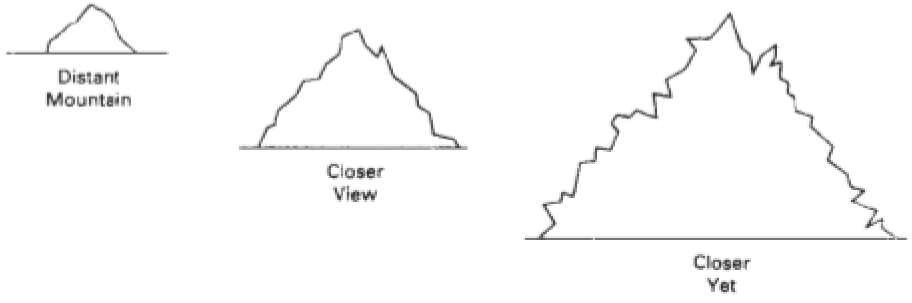
Fractals are geometric objects. Many real-world objects like ferns are shaped like fractals. Fractals are formed by iterations. Fractals are self-similar.

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.

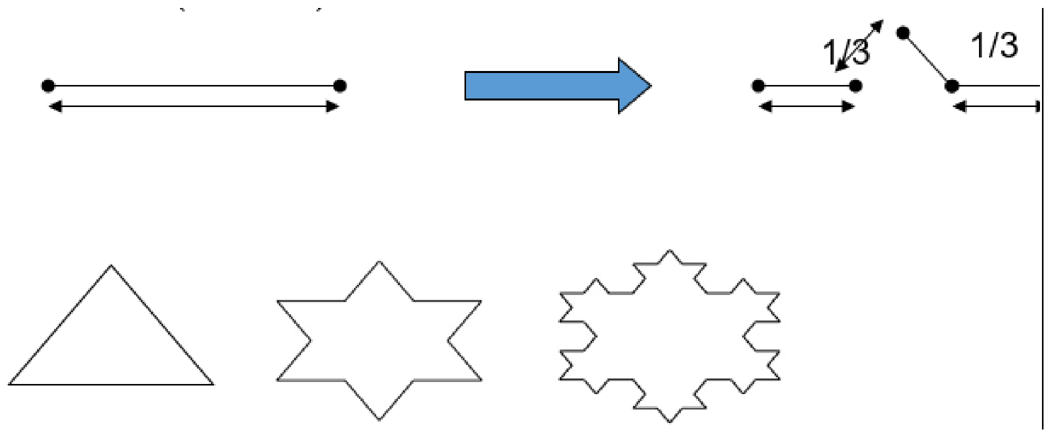
In computer graphics, we use fractal functions to create complex objects



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:

$y_{mid} = (y(a)+y(b))/2 + r$, where

r is the random offset

- This is how to generate the random offset r:

$r = sr_g|b-a|$, where

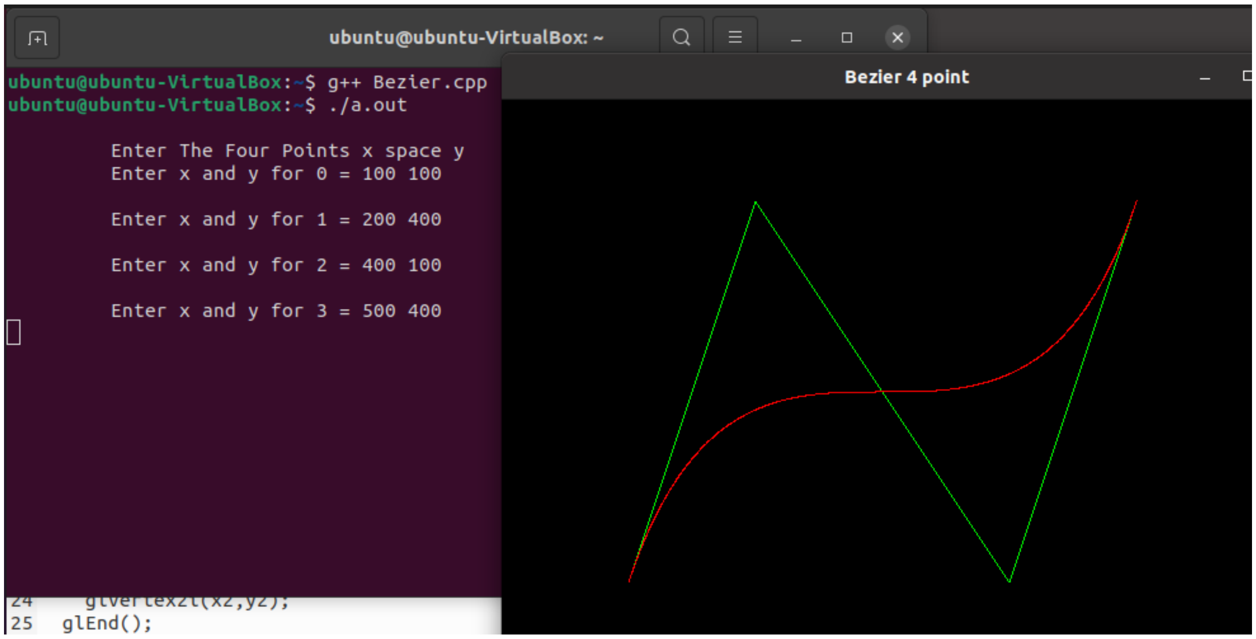
s is a user-selected "roughness" factor, and

r_g 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;
}
```

OUTPUT



```
ubuntu@ubuntu-VirtualBox:~$ g++ Bezier.cpp
ubuntu@ubuntu-VirtualBox:~$ ./a.out

         Enter The Four Points x space y
         Enter x and y for 0 = 100 100

         Enter x and y for 1 = 200 400

         Enter x and y for 2 = 400 100

         Enter x and y for 3 = 500 400
```

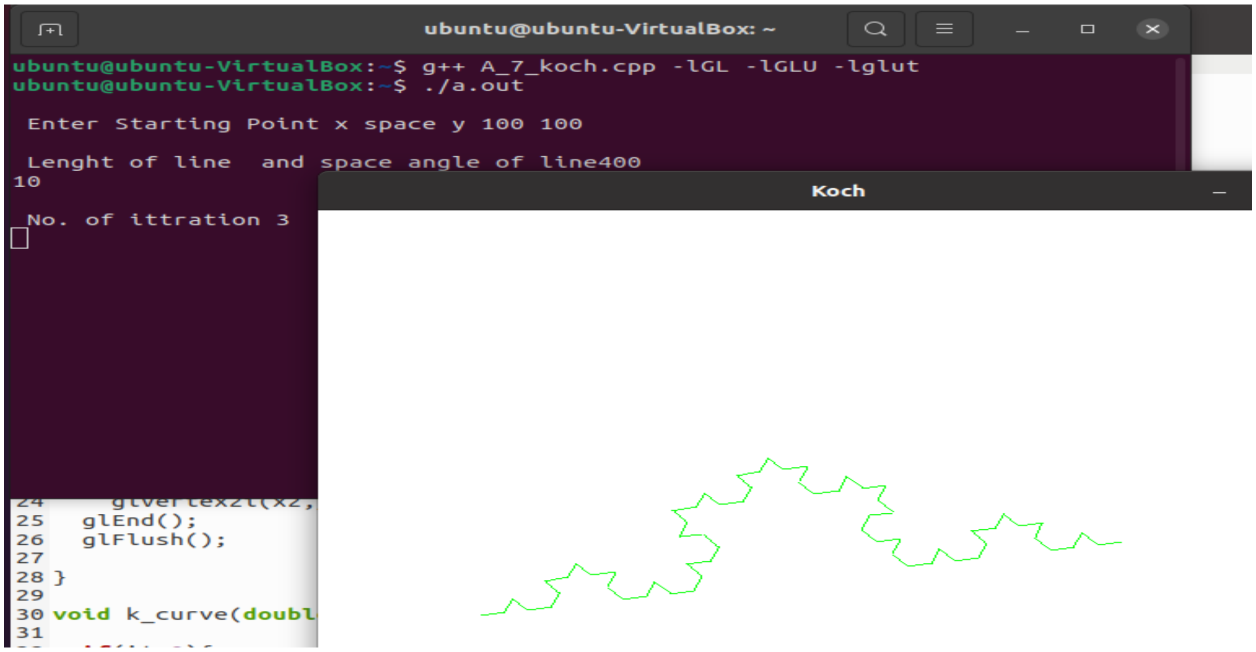
```
    glVertex2i(x2,y2);
  glEnd();
```

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:~$ 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
```

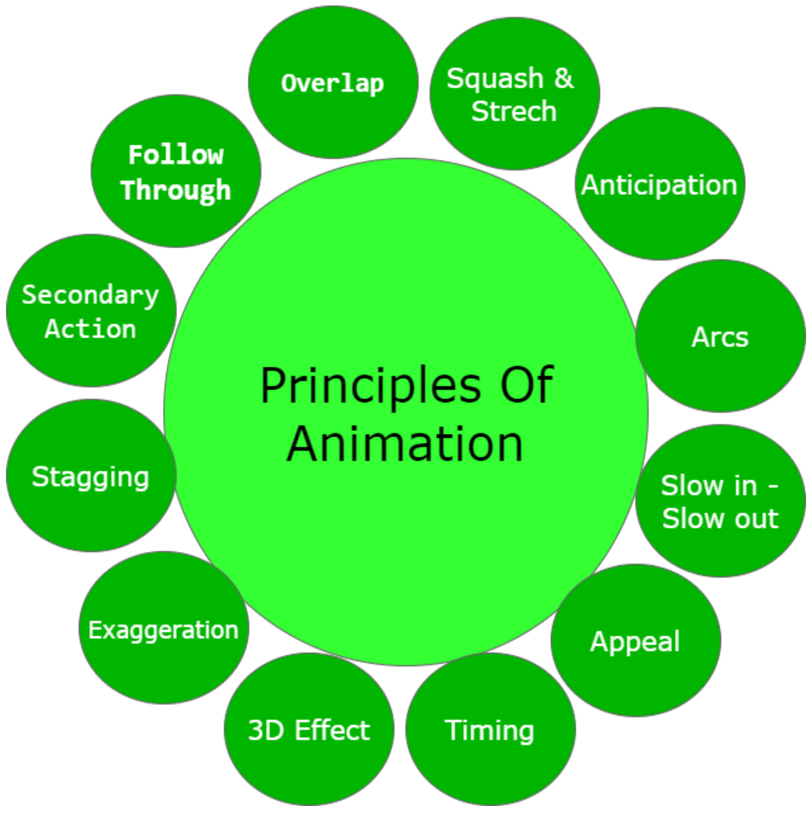
```
  glEnd();
  glFlush();
}

void k_curve(doubl
```

Experiment No	8
Title	Implement animation principles for any object
Roll No.	
College	NSGIFOE , Pune
Class	S.E. (AIML)
Date	
Subject	Computer Graphics

Theory:

Animation is defined as a series of images rapidly changing to create an illusion of movement. We replace the previous image with a new image which is a little bit shifted. Animation Industry is having a huge market nowadays. To make an efficacious animation there are some principles to be followed.

Principle of Animation:



There are 12 major principles for an effective and easy to communicate animation.

1. **Squash and Strech:**
   This principle works over the physical properties that are expected to change in any process. Ensuring proper squash and stretch makes our animation more convincing.

   For Example: When we drop a ball from height, there is a change in its physical property. When the ball touches the surface, it bends slightly which should be depicted in animation properly.

2. **Anticipation:**
   Anticipation works on action.Animation has broadly divided into 3 phases:
3. **1.** Preparation phase
4. **2.** Movement phase
   3. Finish

In Anticipation, we make our audience prepare for action. It helps to make our animation look more realistic.

For Example: Before hitting the ball through the bat, the actions of batsman comes under anticipation. This are those actions in which the batsman prepares for hitting the ball.

5. **Arcs:**
   In Reality, humans and animals move in arcs. Introducing the concept of arcs will increase the realism. This principle of animation helps us to implement the realism through projectile motion also.

   For Example, The movement of the hand of bowler comes under projectile motion while doing bowling.

6. **Slow in-Slow out:**
   While performing animation, one should always keep in mind that in reality object takes time to accelerate and slow down. To make our animation look realistic, we should always focus on its slow in and slow out proportion.

   For Example, It takes time for a vehicle to accelerate when it is started and similarly when it stops it takes time.

7. **Appeal:**
   Animation should be appealing to the audience and must be easy to understand. The syntax or font style used should be easily understood and appealing to the audience. Lack of symmetry and complicated design of character should be avoided.
8. **Timing:**

   Velocity with which object is moving effects animation a lot. The speed should be handled with care in case of animation.

   For Example, An fast-moving object can show an energetic person while a slow-moving object can symbolize a lethargic person. The number of frames used in a slowly moving object is less as compared to the fast-moving object.

9. **3D Effect:**
   By giving 3D effects we can make our animation more convincing and effective. In 3D Effect, we convert our object in a 3-dimensional plane i.e., X-Y-Z plane which improves the realism of the object.

   For Example, a square can give a 2D effect but cube can give a 3D effect which appears more realistic.

10. **8. Exaggeration:**
    Exaggeration deals with the physical features and emotions. In Animation, we represent emotions and feeling in exaggerated form to make it more realistic. If

there is more than one element in a scene then it is necessary to make a balance between various exaggerated elements to avoid conflicts.

11. **Stagging:**
    Stagging is defined as the presentation of the primary idea, mood or action. It should always be in presentable and easy to manner. The purpose of defining principle is to avoid unnecessary details and focus on important features only. The primary idea should always be clear and unambiguous.
12. **Secondary Action:**
    Secondary actions are more important than primary action as they represent the animation as a whole. Secondary actions support the primary or main idea.

    For Example, A person drinking a hot tea, then his facial expressions, movement of hands, etc comes under the secondary actions.

13. **Follow Through:**
    It refers to the action which continues to move even after the completion of action. This type of action helps in the generation of more idealistic animations.

    For Example: Even after throwing a ball, the movement of hands continues.

14. **Overlap:**
    It deals with the nature in which before ending the first action, the second action starts.

    For Example: Consider a situation when we are drinking Tea from the right hand and holding a sandwich in the left hand. While drinking a tea, our left-hand start showing movement towards the mouth which shows the interference of the second action before the end of the first action.

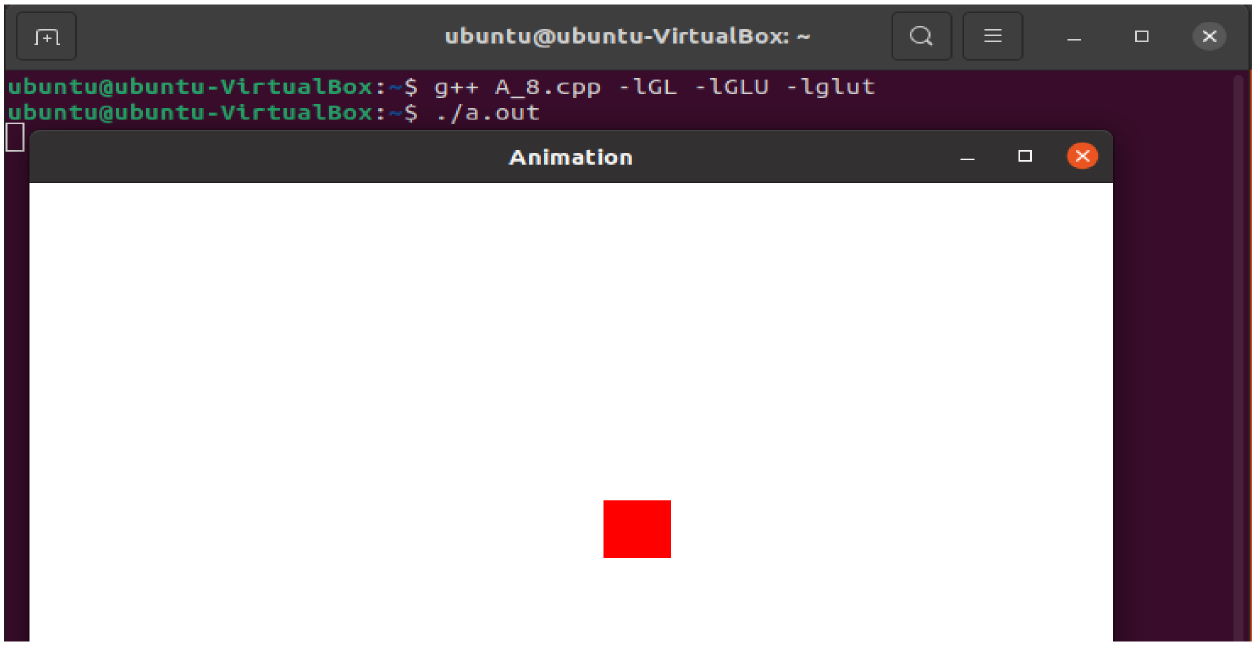
    Reference of Theory: https://www.geeksforgeeks.org/principles-of-animation/

CODE

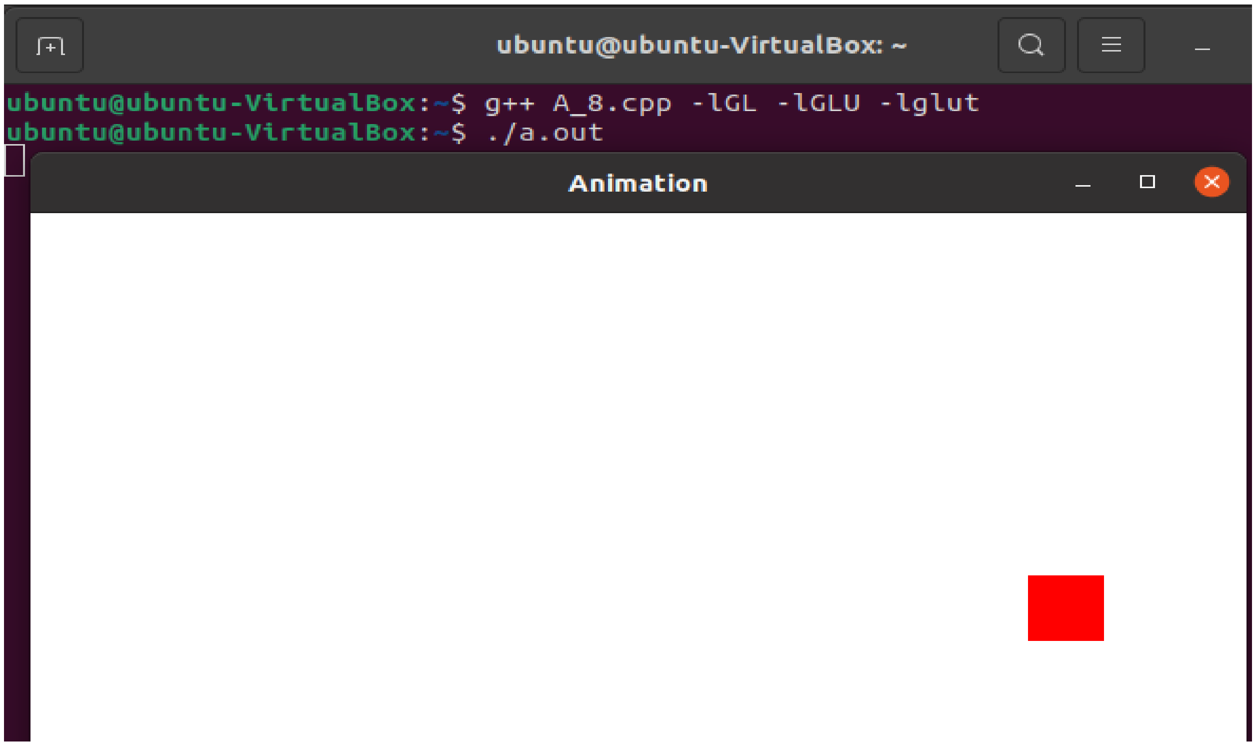
```
#include <iostream>
#include <math.h>
#include <time.h>
#include <GL/glut.h>

using namespace std;


int x=0;
int flag=0;
void init(){
   glClearColor(1.0,1.0,1.0,0.0);
   glMatrixMode(GL_PROJECTION);
```

```
    gluOrtho2D(0,640,0,480);
}

void object1(){

    glClear(GL_COLOR_BUFFER_BIT);

    glColor3f(1,0,0);
    glBegin(GL_POLYGON);
        glVertex2i(x,220);
        glVertex2i(x+40,220);
        glVertex2i(x+40,260);
        glVertex2i(x,260);
    glEnd();

    glutSwapBuffers();

}

void timer(int){

    glutPostRedisplay();
    glutTimerFunc(1000/60,timer,0);

    if(flag == 0){
        x = x+3;
    }
    if(flag == 1){
        x = x-3;
    }
    if(x==600){
        flag = 1;
    }
    if(x == 0){
        flag = 0;
    }

}
int main(int argc, char** argv){
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize(640,480);
    glutInitWindowPosition(200,200);
    glutCreateWindow("Animation");
    init();
    glutDisplayFunc(object1);
    glutTimerFunc(1000,timer,0);
    glutMainLoop();
    return 0;
}
```

OUTPUT



```
ubuntu@ubuntu-VirtualBox:~$ g++ A_8.cpp -lGL -lGLU -lglut
ubuntu@ubuntu-VirtualBox:~$ ./a.out
```



```
ubuntu@ubuntu-VirtualBox:~$ g++ A_8.cpp -lGL -lGLU -lglut
ubuntu@ubuntu-VirtualBox:~$ ./a.out
```