CERTIFICATE

Class VII Year 4th

This is to certify that the work entered in this Journal is the work of

~~Shri~~ / Kumari DHARA · R · RACHH

of VII division C Roll No. -

Uni. Exam No. 1RV18CS404 has satisfactorily completed the required number of practical and worked for the 1st term / 2nd term / both the terms of the year 2020 2021 in the college laboratory as laid down by the university.

Head of the Department	External Examiner	Internal Examiner Subject teacher

Date : / / 20 Department of

INDEX

PARTICULARS OF THE EXPERIMENTS PERFORMED

Expt. No.	Name of Experiment	Page No.	Date of Experiment	Date of Submission	Remarks
1	Bresenham's line Drawing	1	12/01/2021		
2	Bresenham's Circle & ellipse Drawing	7	12/01/2021		
3	3D Sierpinski gasket	16	12/01/2021		
4	Scan-line area filling	19	12/01/2021		
5	House rotation	22	14/01/2021		
6	Cohen-Sutherland line Clipping	26	14/01/2021 14/01/2021		
7	Liang-Barsky line Clipping	31	14/01/2021	15/01/202	
8	Cohen-Hodgeman polygon Clipping	35	14/01/2021		
9	Moving Car	39	15/01/2021		
10	Color Cube Spin	42	15/01/2021		
11	Curve Menu	45	15/01/2021		
12	Bezier Curve	50	15/01/2021		

Q1 Write a program to generate a line using Bresenham's line drawing technique. Consider slopes greater than one and slopes less than one. User must be able to draw as many lines & specify inputs through keyboard/mouse.

→
```
#include <iostream>
#include <GL/glut.h>
#include <time.h>
using namespace std;
int flag = 0;
void draw_pixel(int x, int y)
{
    glColor3f(1,0,0);
    glBegin(GL_POINTS);
    glVertex2i(x,y);
    glEnd();
    glFlush();
}
void draw_line()
{
    int dx, dy, i, e;
    int incx, incy, inc1, inc2;
    int x, y;
    dx = x2 - x1;
    dy = y2 - y1;
```

```
if (dx<0) dx = -dx;
if (dy<0) dy = -dy;
incx = 1;
if (x2 < x1)
    incx = -1;
incy = 1;
if (y2 < y1)
    incy = -1;
x = x1;
y = y1;
if (dx > dy)
{
    draw_pixel(x, y);
    e = 2 * dy - dx;
    inc1 = 2 * (dy - dx);
    inc2 = 2 * dy;
    for (i=0; i<dx; i++)
    {
        if (e > 0)
        {
            y+ = incy;
            e+ = inc1;
        }
        else
            e+ = inc2;
```

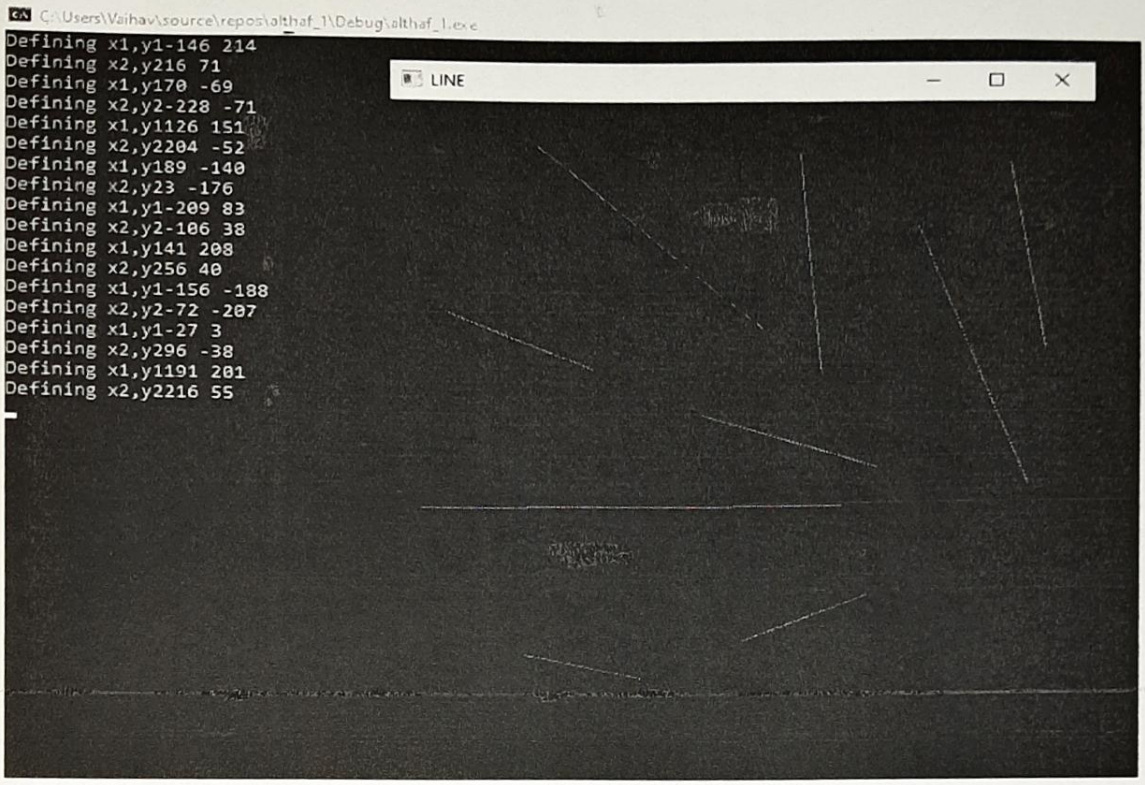
```
            x+=incx;
            draw_pixel(x,y);
        }
    }
    else
    {
        draw_pixel(x,y);
        e=2*dx-dy;
        inc1=2*(dx-dy);
        inc2=2*dx
        for(i=0;i<dy;i++)
        {
            if(e>0)
            {
                x+=incx;
                e+=inc1;
            }
            else
                e+=inc2;
            y+=incy;
            draw_pixel(x,y);
        }
    }
    glFlush();
}
```

```
void myinit()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glClearColor(1,1,1,1);
    gluOrtho2D(-250, 250, -250, 250);
}
void MyMouse(int button, int state, int x, int y)
{
  switch(button)
  {
  case GLUT_LEFT_BUTTON;
     if (state == GLUT_DOWN)
     {
       if (flag == 0)
       {
         printf("Defining x1, y1");
         x1 = x - 250
         yc1 = 250 - y;
         flag++;
         count << x1 << " " << yc1 << "\n";
       }
       else
       {
         printf("Defining x2, y2");
         x2 = x - 250;
```

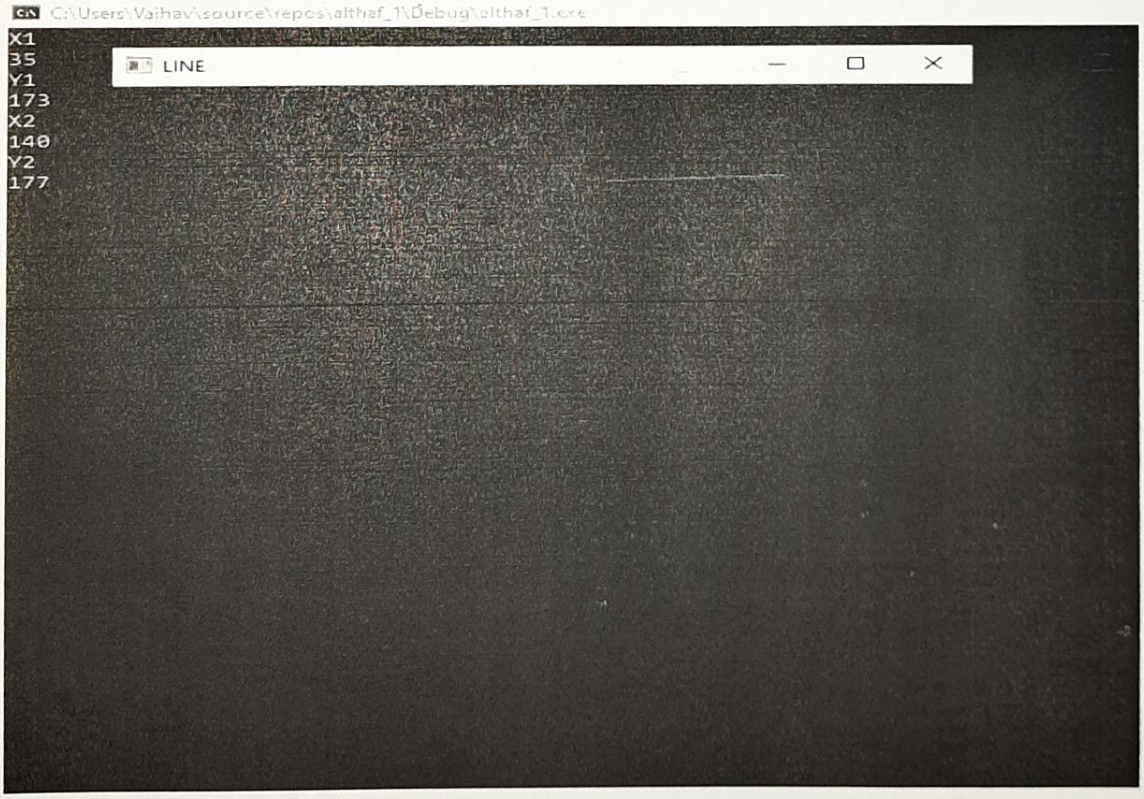
```
            y2 = 250-y;
            flag = 0;
            cout << x2 << " " << y2 << "\n";
            draw-line();
        }
    } break;
}
void display ()
{ }
int main(int ac, char* av[])
{ /* //For keyboard
    cout << "x1 \n";
    cin >> x1;
    cout << "y1 \n";
    cin >> y1;
    cout << "x2 \n";
    cin >> x2;
    cout << "y2 \n";
    cin >> y2;
    //End keyboard */
    glutInit(&ac, av);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500,500);
    glutInitWindowPosition(100,200);
    glutCreateWindow("LINE");
```

```
myinit();
glutMouseFunc(MyMouse);
//draw_line();
glutDisplayFunc(~~MyMouse~~, display);
glutMainLoop();
}
```

Input through mouse:



Input through keyboard



EXP.: No. 2

PAGE No. 7
DATE 12 01 2021

Q Write a program to generate a circle and ellipse using Bresenham's circle drawing and ellipse drawing techniques. Use two windows to draw circle in one window and ellipse in the other window. User can specify inputs through keyboard/mouse.

→
```
#include <gl/glut.h>
#include <stdio.h>
#include <math.h>
int xc, yc, r;
int rx, ry, xce, yce;
void draw-circle (int xc, int yc, int x, int y)
{
    glBegin (GL_POINTS);
    glVertex2i (xc + x, yc+y);
    glVertex2i (xc - x, yc+y);
    glVertex2i (xc+x, yc-y);
    glVertex2i (xc-x, yc-y);
    glVertex2i (xc+y, yc+x);
    glVertex2i (xc-y, yc+x);
    glVertex2i (xc+y, yc-x);
    glVertex2i (xc-y, yc-x);
    glEnd();
}
void circlebres()
{
```

```
glClear(GL_COLOR_BUFFER_BIT);
int x=0, y=r;
int d=3-2*r;
while (x<=y)
{
    draw_circle(xc,yc,x,y);
    x++;
    if (d<0)
        d=d+4*x+6;
    else
    {
        y--;
        d=d+4*(x-y)+10;
    }
    draw_circle(xc,yc,x,y);
}
glFlush();
}
int p1_x, p2_x, p1_y, p2_y;
int point1_done = 0;
void myMouseFuncCircle(int button, int state, int x, int y)
{
  if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN&&
     point1_done == 0)
  {
```

```
        p1-x = x - 250;
        p1-y = 250 - y;
        point1_done = 1;
    }
    else if(button==GLUT_LEFT_BUTTON && state=GLUT_DOWN)
    {
        p2-x = x - 250;
        p2-y = 250 - y;
        xc = p1-x;
        yc = p1-y;
        float exp =(p2-x - p1-x)*(p2-x - p1-x) +
                    (p2-y - p1-y)*(p2-y - p1-y);
        r = (int)(sqrt(exp));
        circlebres();
        point1_done = 0;
    }
}
////Ellipse////
void draw_ellipse(int xce, int yce, int x, int y)
{
    glBegin (GL_POINTS);
    glVertex2i(x + xce, y + yce);
    glVertex2i(-x + xce, y + yce);
    glVertex2i(x + xce, -y + yce);
    glVertex2i(-x + xce, -y + yce);
```

```
    glEnd();
}
void midptellipse()
{
    glClear(GL_COLOR_BUFFER_BIT);
    float dx, dy, d1, d2, x, y;
    x=0;
    y=ry;
    // initial decision parameter of region 1
    d1=(ry*ry)-(rx*rx*ry)+(0.25*rx*rx);
    dx=2*ry*ry*x;
    dy=2*rx*rx*y;
    // For region 1
    while(dx<dy)
    {
        draw_ellipse(xce, yce, x, y);
        if(d1<0)
        {
            x++;
            dx=dx+(2*ry*ry);
            d1=d1+dx+(ry*ry);
        }
        else
        {
            x++;
```

```
            y--;
            dx = dx + (2 * ry * ry);
            dy = dy - (2 * rx * rx);
            d1 = d1 + dx - dy + (ry * ry);
        }
    }
    d2 = ((ry * ry) * ((x+0.5) * (x+0.5))) +
      ((rx * rx) * ((y-1) * (y-1))) - (rx * rx * ry * ry);
    while (y >= 0)
    {
      draw_ellipse(xce, yce, x, y);
      if (d2 > 0)
      {
        y--;
        dy = dy - (2 * rx * rx);
        d2 = d2 + (rx * rx) - dy;
      }
      else
      {
        y--;
        x++;
        dx = dx + (2 * ry * ry);
        dy = dy - (2 * rx * rx);
        d2 = d2 + dx - dy + (rx * rx);
      }
    }
```

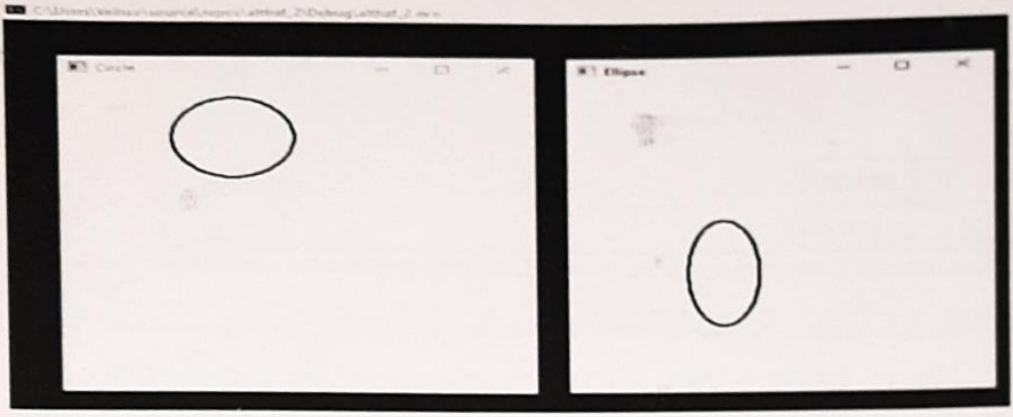
```
    glFlush();
}
int p1e_x, p2e_x, p1e_y, p2e_y, p3e_x, p3e_y;
int point1e_done = 0;
void myMouseFunc(int button, int state, int x, int y)
{
    if(button == GLUT_LEFT_BUTTON && state == GLUT_DOWN&&
    point1e_done == 0)
    {
        p1e_x = x - 250;
        p1e_y = 250 - y;
        xcl = p1e_x;
        ycl = p1e_y;
        point1e_done = 1;
    }
    else if (button == GLUT_LEFT_BUTTON && state ==
        GLUT_DOWN && point1e_done == 1)
    {
        p2e_x = x - 250;
        p2e_y = 250 - y;
        ~~point1e_done~~ float exp = (p2e_x - p1e_x)*
        (p2e_x - p1e_x) + (p2e_y - p1e_y) * (p2e_y - p1e_y);
        rx = (int)(sqrt(exp));
        point1e_done = 2;
    }
```

```
else if(button==GLUT_LEFT_BUTTON && state==GLUT_DOWN
&& point1e_done==2)
{
    p3e_x = x-250;
    p3e_y = 250-y;
    float exp=(p3e_x-p1e_x)*(p3e_x-p1e_x)+(p3e_y-
    p1e_y);
    ry=(int)(sqrt(exp));
    midptellipse();
    point1e_done=0;
  }
}
void myDrawing()
{}
void myDrawingc()
{}
void minit()
{
  glClearColor(1,1,1,1);
  glColor3f(1.0,0.0,0.0);
  glPointSize(3.0);
  gluOrtho2D(-250,250,-250,250);
}
void main(int argc, char* argv[])
{
```

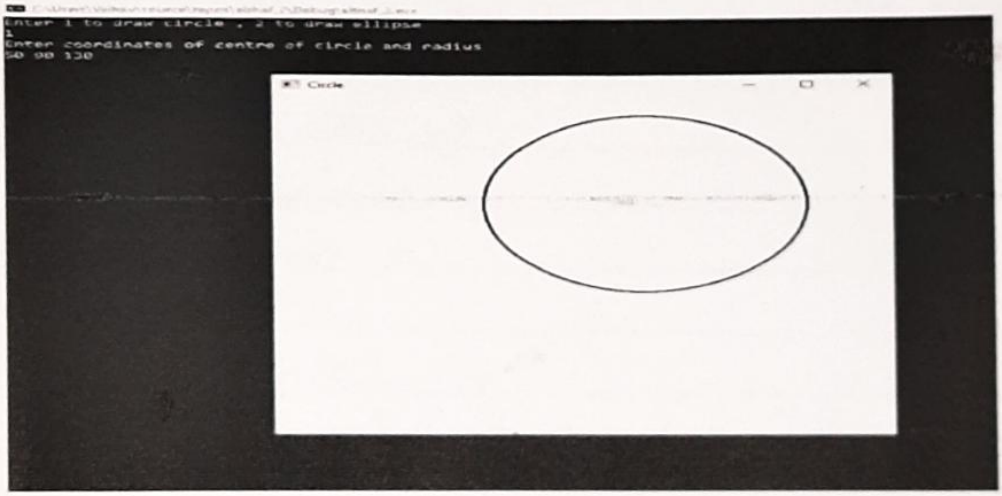
```
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
glutInitWindowSize(500, 500);
glutInitWindowPosition(0,0);
/* //For Mouse
int id1 = glutCreateWindow("circle");
glutSetWindow(id1);
glutMouseFunc(myMouseFuncCircle);
glutDisplayFunc(myDrawingC);
minit();
glutInitWindowSize(500, 500);
glutInitWindowPosition(600, 100);
int id2 = glutCreateWindow("Ellipse");
glutSetWindow(id2);
glutMouseFunc(myMouseFunc);
glutDisplayFunc(myDrawing);
//End Mouse */
//For Keyboard
printf("Enter 1 To draw circle, 2 To draw ellipse\n");
int ch;
scanf("%d", &ch);
switch(ch){
case 1;
printf("Enter coordinates of centre of circle & radius\n");
scanf("%d %d %d", &xc, &yc, &r);
```

```
glutCreateWindow("Circle");
glutDisplayFunc(circlebres);
break;
Case 2:
printf("Enter co-ordinates of centre of ellipse and major
 and minor radius \n");
scanf("%d %d %d %d",&xc, &yc, &rx, &ry);
glutCreateWindow("Ellipse");
glutDisplayFunc(midptellipse);
break;
}
// End Keyboard
minit();
glutMainLoop();
}
```

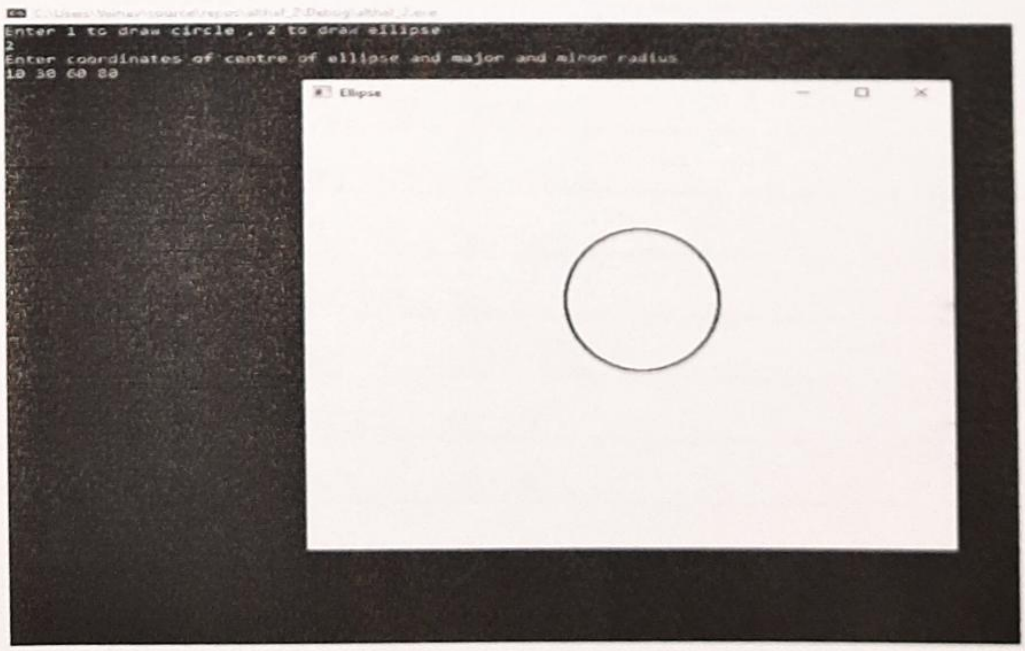
Input through mouse



Input through keyboard :Circle



Ellipse



Q Write a program to recursively subdivides a Tetrahedron to form 3D Sierpinski gasket. The no. of recursive steps is to be specified at execution time.

→
```
#include <gl/glut.h>
#include <stdio.h>
int m;
typedef float point[3];
point Tetra[4]={{0,100,-100},{0,0,100},{100,-100,-100},
        {-100,-100,-100}};
void Tetrahedron(void);
void myinit(void);
void divide_Triangle(point a, point b, point c, int m);
void draw_Triangle(point p1, point p2, point p3);
int main(int argc, char ** argv)
{ //int m;
  printf("Enter the number of iterations:");
  scanf_s("%d", &m);
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB|
                                GLUT_DEPTH);
  glutInitWindowPosition(100, 200);
  glutInitWindowSize(500, 500);
  glutCreateWindow("Serpinski Gasket");
  glutDisplayFunc(Tetrahedron);
  glEnable(GL_DEPTH_TEST);
```

```
    myinit();
    glutMainLoop();
}
void divide_triangle(point a, point b, point c, int m)
{
  point v1, v2, v3;
  int j;
  if(m>0){
     for (j=0; j<3; j++)
        v1[j] = (a[j]+b[j])/2;
     for (j=0; j<3; j++)
        v2[j] = (a[j]+c[j])/2;
     for (j=0; j<3; j++)
        v3[j] = (b[j]+c[j])/2;

  divide_triangle(a, v1, v2, m-1);
  divide_triangle(c, v2, v3, m-1);
  divide_triangle(b, v3, v1, m-1);
  }
  else
     draw_triangle(a,b,c);
}
void myinit()
{ glClearColor(1,1,1,1);
  glOrtho(-500.0, 500.0, -500.0, 500.0, -500.0, 500.0); }
```

```
void Tetrahedron(void)
{ //myinit();
  glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
  glColor3f(1.0,0.0,0.0);
  divide_Triangle(Tetra[0], Tetra[1], Tetra[2], m);
  glColor3f(0.0,1.0,0.0);
  divide_Triangle(Tetra[3], Tetra[2], Tetra[1], m);
  glColor3f(0.0,0.0,1.0);
  divide_Triangle(Tetra[0], Tetra[3], Tetra[1], m);
  glColor3f(0.0,0.0,0.0);
  divide_Triangle(Tetra[0], Tetra[2], Tetra[3], m);
  glFlush();
}
void draw_Triangle(point p1, point p2, point p3)
{
  glBegin(GL_TRIANGLES);
  glVertex3fv(p1);
  glVertex3fv(p2);
  glVertex3fv(p3);
  glEnd();
}
```

C:\Users\Vaihav\source\repos\althaf_3\Debug\althaf_3.exe

```
Enter the number of iterations: 4
```

Seirpinski Gasket

C:\Users\Vaihav\source\repos\althaf_3\Debug\althaf_3.exe

```
Enter the number of iterations: 1
```

Seirpinski Gasket

Q Write a program To fill any given polygon using Scan-line area filling algorithm

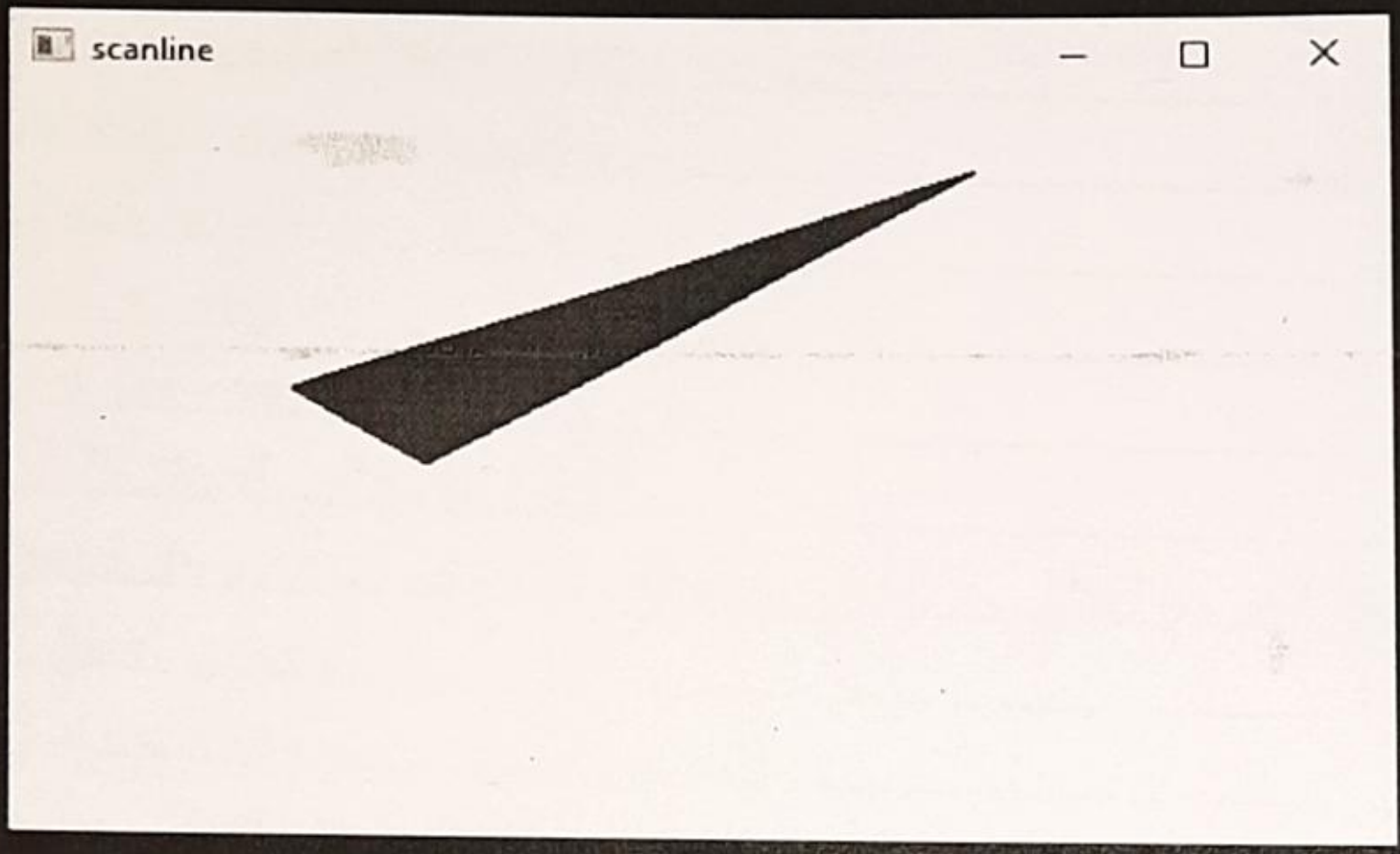
→
```
#include <stdlib.h>
#include <gl/glut.h>
#include <algorithm>
#include <iostream>
#include <windows.h>
using namespace std;
float x[100], y[100];
int n, m;
int wx=500, wy=500;
static float intx[10]={0};
void draw-line(float x1, float y1, float x2, float y2, int scanline)
{ float temp;
  if(y2<y1)
  {   Temp=x1; x1=x2; x2=Temp;
      temp=y1; y1=y2; y2=Temp;
  }
  if(scanline>y1 && scanline<y2)
      intx[m++]=x1+(scanline-y1)*(x2-x1)/(y2-y1);
}
void scanfill(float x[], float y[])
{ for(int s1=0; s1<=wy; s1++)
  {   m=0;
      for(int i=0; i<n; i++)
      {   edgeDetect(x[i], y[i], x[(i+1)%n], y[(i+1)%n], s1);
      }
      sort(int x, (int x+m));
```

```
        if(m >= 2)
            for (int i=0; i<m; i=i+2)
                draw-line(intx[i], s1, intx[i+1], s1);
}
void display_filled_polygon()
{
  glClear(GL_COLOR_BUFFER_BIT);
  glLineWidth(2);
  glBegin(GL_LINE_LOOP);
  for(int i=0; i<n; i++)
      glVertex2f(x[i], y[i]);
  glEnd();
  scanfill(x, y);
} //glFlush();
void myinit()
{ glClearColor(1,1,1,1);
  glColor3f(0,0,1);
  glPointSize(1);
  gluOrtho2D(0, wx, 0, wy);
}
void main(int ac, char* av[])
{ glutInit(&ac, av);
  printf("Enter no. of sides: \n");
  scanf("%d", &n);
  printf("Enter coordinates of endpoints: \n");
  for(int i=0; i<n; i++)
```

```
{
    printf("x-coord Y-coord : \n");
    scanf("%f %f", &x[i], &y[i]);
}
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
glutInitWindowSize(500, 500);
glutInitWindowPosition(0, 0);
glutCreateWindow("scanline");
glutDisplayFunc(display_filled_polygon);
myInit();
glutMainLoop();
}
```

Select C:\Users\Vaihav\source\repos\althaf_4\Debug\althaf_4.exe

```
Enter no. of sides:
3
Enter coordinates of endpoints:
X-coord Y-coord:
150 250
X-coord Y-coord:
100 300
X-coord Y-coord:
350 450
```



Q Write a program To create a house like figure & perform foll- functions (i) rotate it about a given fixed point using openGL Transformation func. (ii) Reflect it about an axis $y = mx + c$ using openGL transformation func.

→
```
#include<gl/glut.h>
#include<math.h>
#include<stdio.h>
float house[11][2]={{100,200},{200,250},{300,200},{100,200},
{100,100},{175,100},{175,150},{225,150},{225,100},{300,100},{300,200}};
int angle;
float m,c,Theta;
void display()
{ glClearColor(1,1,1,0);
  glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  glOrtho2D(-450,450,-450,450);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  glColor3f(1,0,0);
  glBegin(GL_LINE_LOOP);
  for(int i=0; i<11; i++)
     glVertex2fv(house[i]);
  glEnd(); glFlush();
  glPushMatrix();
```

```
glTranslatef(100, 100, 0);
glRotatef(angle, 0, 0, 1);
glTranslatef(-100, -100, 0);
glColor3f(1,1,0);
glBegin(GL_LINE_LOOP);
for (int i=0; i<11; i++)
    glVertex2fv(house[i]);
glEnd();
glPopMatrix();
glFlush();
}
void display2()
{ glClearColor(1,1,1,0);
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(-450, 450, -450, 450);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glColor3f(1,0,0);
glBegin(GL_LINE_LOOP);
for(int i=0; i<11; i++)
    glVertex2fv(house[i]);
glEnd(); glFlush();
float x1=0, x2=500;
```

```
float y1 = m * x1 + C;
float y2 = m * x2 + C;
glColor3f(1,1,0);
glBegin(GL_LINES);
glVertex2f(x1, y1);
glVertex2f(x2, y2);
glEnd(); glFlush();
glPushMatrix();
glTranslatef(0,c,0);
Theta = atan(m);
Theta = theta * 180/3.14;
glRotatef(Theta, 0,0,1);
glScalef(1,-1,1);
glRotatef(-Theta, 0,0,1)
glTranslate(0,-c,0);
glBegin(GL_LINE_LOOP);
for(int i=0; i<11; i++)
    glVertex2fv(house[i]);
glEnd(); glPopMatrix(); glFlush();
}
void myInit()
{
glClearColor(1.0, 1.0, 1.0, 1.0);
glColor3f(1.0, 0.0, 0.0);
glLineWidth(2.0);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(-450, 450, -450, 450);
}
```

```
void mouse(int btn, int state, int x, int y)
{
  if (btn==GLUT_LEFT_BUTTON && state==GLUT_DOWN)
  {
    display();
  else if (btn==GLUT_RIGHT_BUTTON && state==GLUT_DOWN)
  {
    display2();
}}
void main(int argc, char** argv)
{ printf("Enter rotation angle \n");
  scanf("%d", &angle);
  printf("Enter c & m value for line y=mx+c \n");
  scanf("%f %f", &c, &m);
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
  glutInitWindowSize(900, 900);
  glutInitWindowPosition(100, 100);
  glutCreateWindow("House Rotation");
  glutDisplayFunc(display);
  glutMouseFunc(mouse);
  myInit();
  glutMainLoop();
}
```

C:\Users\Vaihav\source\repos\althaf_5\Debug\althaf_5.exe

```
Enter the rotation angle
15
Enter c and m value for line y=mx+c
50 100
```

House Rotation

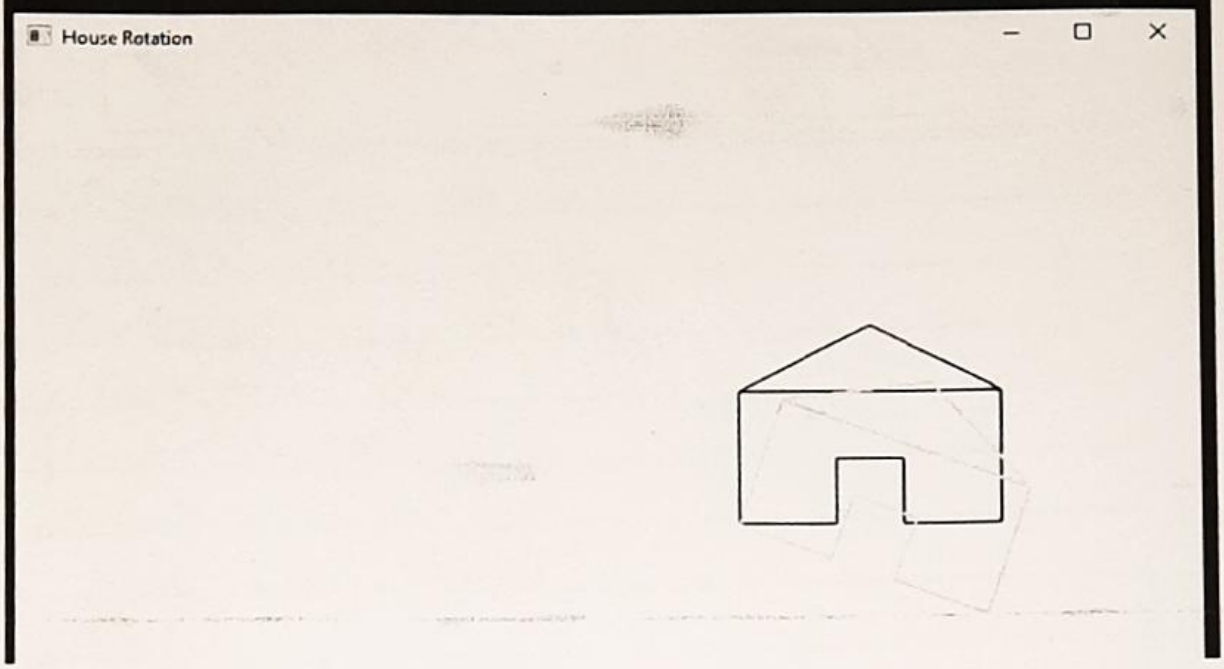
C:\Users\Vaihav\source\repos\althaf_5\Debug\althaf_5.exe

```
Enter the rotation angle
15
Enter c and m value for line y=mx+c
50 100
```

House Rotation

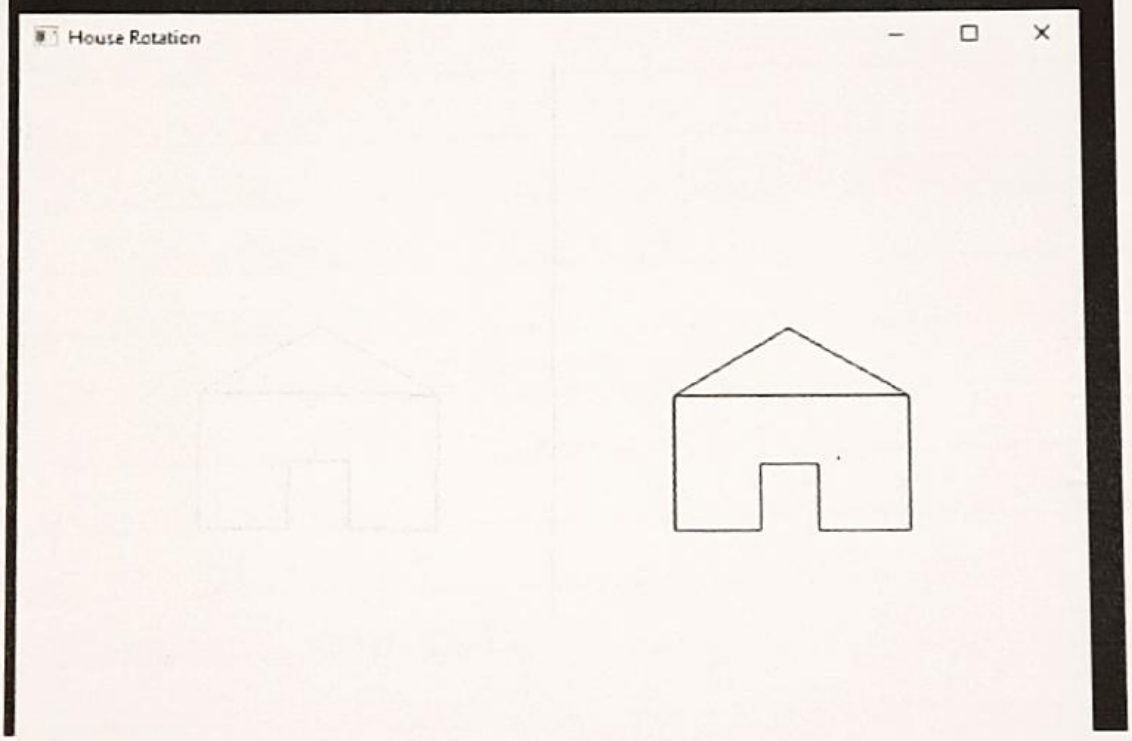
C:\Users\Vaibav\source\repos\althaf_5\Debug\althaf_5.exe

```
Enter the rotation angle
-20
Enter c and m value for line y=mx+c
40 120
```



C:\Users\Vaibav\source\repos\althaf_5\Debug\althaf_5.exe

```
Enter the rotation angle
-20
Enter c and m value for line y=mx+c
40 120
```



Q Write program to implement The Cohen-Sutherland line clipping algorithm. Make provision to specify the input for multiple lines, window for clipping & viewport for displaying the clipped image.

→
```
#include <stdio.h>
#include <stdlib.h>
#include <gl/glut.h>
#define outcode int
#define true 1
#define false 0
double xmin, ymin, xmax, ymax;
double xvmin, yvmin, xvmax, yvmax;
const int RIGHT = 4;
const int LEFT = 8;
const int TOP = 1;
const int BOTTOM = 2;
int n;
struct line_segment
{ int x1; int y1; int x2; int y2; };
struct line_segment ls[10];
outcode computeoutcode(double x, double y)
{ outcode code = 0;
  if(y > ymax)
     code |= TOP;
  else if(y < ymin)
```

```
        code1 = BOTTOM;
    if (x > xmax)
        code1 = RIGHT;
    else if (x < xmin)
        code1 = LEFT;
    return code;
}
void cohensuther(double x0, double y0, double x1, double y1)
{
    outcode outcode0, outcode1, outcodeout;
    bool accept = false, done = false;
    outcode0 = computeoutcode(x0, y0);
    outcode1 = computeoutcode(x1, y1);
    do
    {
        if(!(outcode0 | outcode1))
        {
            accept = True;
            done = True;
        }
        else if (outcode0 & outcode1)
            done = True;
        else
        {
            double x, y;
            outcodeout = outcode0 ? outcode0 : outcode1;
            if (outcodeout & TOP)
            {
                x = x0 + (x1 - x0) * (ymax - y0) / (y1 - y0);
                y = ymax;
            }
            else if (outcodeout & BOTTOM)
            {
                x = x0 + (x1 - x0) * (ymin - y0) / (y1 - y0);
                y = ymin;
```

```
else if (outcodeout & RIGHT)
{ y = y0 + (y1 - y0) * (xmax - x0) / (x1 - x0);
  x = xmax;
else
{
  y = y0 + (y1 - y0) * (xmin - x0) / (x1 - x0);
  x = xmin;
}
if (outcodeout == outcode0)
{
  x0 = x;
  y0 = y;
  outcode0 = computeoutcode(x0, y0);
}
else
{
  x1 = x;
  y1 = y;
  outcode1 = computeoutcode(x1, y1);
}
}while (!done);
if (accept)
{ double sx = (xvmax - xvmin) / (xmax - xmin);
  double sy = (yvmax - yvmin) / (ymax - ymin);
  double vx0 = xvmin + (x0 - xmin) * sx;
  double vy0 = yvmin + (y0 - ymin) * sy;
  double vx1 = xvmin + (x1 - xmin) * sx;
  double vy1 = yvmin + (y1 - ymin) * sy;
  glColor3f(0, 0, 1);
  glBegin(GL_LINES);
  glVertex2d(vx0, vy0);
```

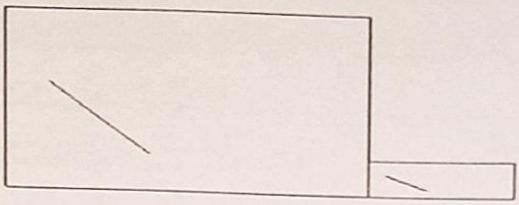
```
        glVertex2d(vx1, vy1);
        glEnd();
    }
}
void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(0,0,1);
    glBegin(GL_LINE_LOOP);
    glVertex2f(xmin, ymin);
    glVertex2f(xmax, ymin);
    glVertex2f(xmax, ymax);
    glVertex2f(xmin, ymax);
    glEnd();
    for (int i=0; i<n; i++)
    {
        glBegin(GL_LINES);
        glVertex2d(Is[i].x1, Is[i].y1);
        glVertex2d(Is[i].x2, Is[i].y2);
        glEnd();
    }
    for (int i=0; i<n; i++)
        cohensuther(Is[i].x1, Is[i].y1, Is[i].x2, Is[i].y2);
    glFlush();
}
void myinit()
{
    glClearColor(1,1,1,1);
    glColor3f(1,0,0);
    glPointSize(1.0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
```

```
}  gluOrtho2D(0,500,0,500);
void main(int argc, char** argv)
{
  printf("Enter window co-or(xmin ymin xmax ymax):\n
  scanf("%lf %lf %lf %lf", &xmin, &ymin, &xmax, &ymax);
  printf("Enter viewport co-or(xvmin yvmin xvmax yvmax):\n
  scanf("%lf %lf %lf %lf", &xvmin, &yvmin, &xvmax, &yvmax)
  printf("Enter no. of lines:\n");
  scanf("%d", &n);
  for(int i=0; i<n; i++)
  {  printf("Enter line endpoints(x1 y1 x2 y2):\n");
     scanf("%d %d %d %d", &ls[i].x1, &ls[i].y1, &ls[i].x2,
           &ls[i].y2);
  }
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
  glutInitWindowSize(500, 500);
  glutInitWindowPosition(0,0);
  glutCreateWindow("Clip");
  myinit();
  glutDisplayFunc(display);
  glutMainLoop();
}
```

C:\Users\Vaibhav\source\repos\althaf_6\Debug\althaf_6.exe

```
Enter window coordinates (xmin ymin xmax ymax):
50 50 300 300
Enter viewport coordinates (xvmin yvmin xvmax yvmax) :
300 50 400 100
Enter no. of lines:
1
Enter line endpoints (x1 y1 x2 y2):
150 100 80 200
```

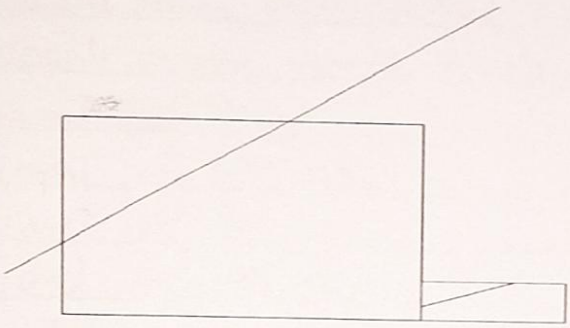
clip



C:\Users\Vaibhav\source\repos\althaf_6\Debug\althaf_6.exe

```
Enter window coordinates (xmin ymin xmax ymax):
50 50 300 300
Enter viewport coordinates (xvmin yvmin xvmax yvmax) :
300 50 400 100
Enter no. of lines:
1
Enter line endpoints (x1 y1 x2 y2):
10 100 350 450
```

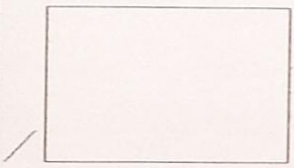
clip



C:\Users\Vaibhav\source\repos\althaf_6\Debug\althaf_6.exe

```
Enter window coordinates (xmin ymin xmax ymax):
50 50 300 300
Enter viewport coordinates (xvmin yvmin xvmax yvmax) :
300 50 400 100
Enter no. of lines:
1
Enter line endpoints (x1 y1 x2 y2):
10 50 40 100
```

clip



Q Write program to implement the Liang-Barsky line clipping algorithm. Make provision to specify the input for multiple lines, window for clipping & viewport for displaying the ~~dipp~~ clipped image.

→
```
#include<stdio.h>
#include<GL/glut.h>
double xmin, ymin, xmax, ymax;
double xvmin, yvmin, xvmax, yvmax;
int n;
struct line-segment{
 int x1; int y1; int x2 int y2; };
struct line-segment IS[10];
int cliptest(double p, double q, double* u1, double* u2)
{ double r;
  if (p) r = q/p;
  if (p<0.0)
  {
    if (r > *u1) *u1 = r;
    if (r > *u2) return(false);
  }
  else
    if (p>0.0)
    { if (q<0.0) return(false);
    }
  return(True);
}
void LianBarskyLineClipAndDraw(double x0, double y0,
   double x1, double y1)
{ double dx=x1-x0, dy=y1-y0, u1=0.0, u2=1.0;
```

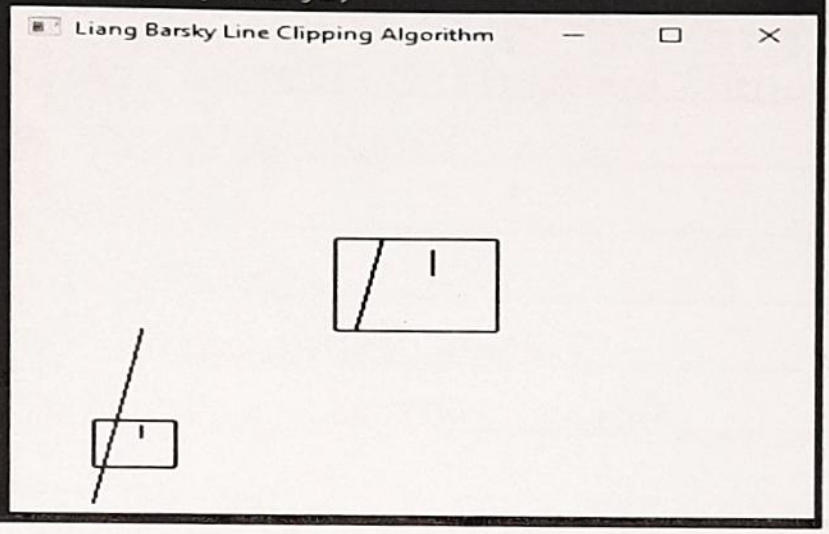
```
glColor3f(1.0, 0.0, 0.0);
glBegin(GL_LINE_LOOP);
glVertex2f(xvmin, yvmin);
glVertex2f(xvmax, yvmin);
glVertex2f(xvmax, yvmax);
glVertex2f(xvmin, yvmax);
glEnd();
if(cliptest(-dx, x0, -xmin, &u1, &u2))
  if(cliptest(dx, xmax, -x0, &u1, &u2))
    if(cliptest(-dy, y0-ymin, &u1, &u2))
      if(cliptest(dy, ymax-y0, &u1, &u2))
      { if(u2 < 1.0)
        {  x1 = x0 + u2 * dx;
           y1 = y0 + u2 * dy;
        }
        if(u1 > 0.0)
        {  x0 = x0 + u1 * dx;
           y0 = y0 + u1 * dy;
        }
        double sx = (xvmax - xvmin)/(xmax - xmin);
        double sy = (yvmax - yvmin)/(ymax - ymin);
        double vx0 = xvmin + (x0 - xmin) * sx;
        double vy0 = yvmin + (y0 - ymin) * sy;
        double vx1 = xvmin + (x1 - xmin) * sx;
        double vy1 = yvmin + (y1 - ymin) * sy;
        glColor3f(0.0, 0.0, 1.0);
        glBegin(GL_LINES);
```

```
        glVertex2d(vx0, vy0);
        glVertex2d(vx1, vy1);
        glEnd();
    }
}
void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 0.0, 0.0);
    for(int i=0; i<n; i++)
    {
        glBegin(GL_LINES);
        glVertex2d(Is[i].x1, Is[i].y1);
        glVertex2d(Is[i].x2, Is[i].y2);
    }  glEnd();
    glColor3f(0.0, 0.0, 1.0);
    glBegin(GL_LINE_LOOP);
    glVertex2f(xmin, ymin);
    glVertex2f(xmax, ymin);
    glVertex2f(xmax, ymax);
    glVertex2f(xmin, ymax);
    glEnd();
    for(int i=0; i<n; i++)
      LiangBarskyLineClipAndDraw(Is[i].x1, Is[i].y1, Is[i].x2, Is[i].y2);
}  glFlush();
void myinit()
{
    glClearColor(1.0, 1.0, 1.0, 1.0);
    glColor3F(1.0, 0.0, 0.0);
```

```
    glLineWidth(2.0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho2D(0.0, 499.0, 0.0, 499.0);
}
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(0, 0);
    printf("Window ~~co-ord~~: (xmin ymin xmax ymax) \n");
    scanf("%lf %lf %lf %lf", &xmin, &ymin, &xmax, &ymax);
    printf("viewport co-or: (xvmin, yvmin xvmax yvmax \n");
    scanf("%lf %lf %lf %lf", &xvmin, &yvmin, &xvmax, &yvmax);
    printf("Enter no. of lines: \n");
    scanf("%d", &n);
    for(int i=0; i<n; i++)
    {
        printf("Enter co-or: (x1 y1 x2 y2) \n");
        scanf("%d %d %d %d", &Is[i].x1, &Is[i].y1, &Is[i].x2, &Is[i].y2);
    }

    glutCreateWindow("Liang Barsky Line Clipping Algo");
    glutDisplayFunc(display);
    myinit();
    glutMainLoop();
}
```

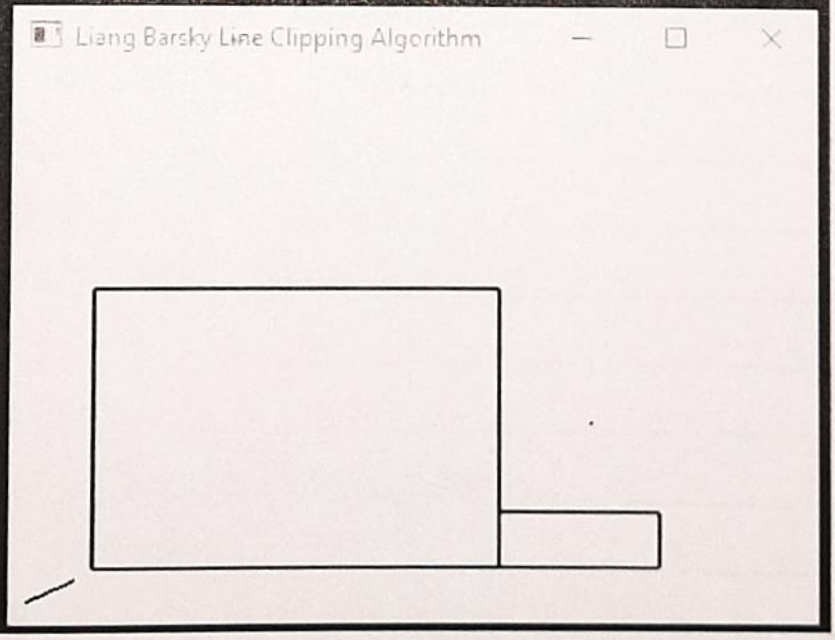
C:\Users\Vaihav\source\repos\althaf_7\Debug\althaf_7.exe.

```
Enter window coordinates: (xmin ymin xmax ymax)
50 50 100 100
Enter viewport coordinates: (xvmin yvmin xvmax yvmax)
200 200 300 300
Enter no. of lines:
2
Enter coordinates: (x1 y1 x2 y2)
50 10 80 200
Enter coordinates: (x1 y1 x2 y2)
80 80 80 95
```



Select C:\Users\Vaihav\source\repos\althaf_7\Debug\althaf_7.exe

```
50 50 300 300
Enter viewport coordinates: (xvmin yvmin xvmax yvmax)
300 50 400 100
Enter no. of lines:
1
Enter coordinates: (x1 y1 x2 y2)
10 20 40 40
```



Q Write program to implement Cohen-Sodgeman polygon clipping algo. Make provision to specify i/p polygon & window for clipping.

→
```
#include<iostream>
#include<GL/glut.h>
using namespace std;
int poly_size, poly_points[20][2], org_poly_size, org_poly_points[20][2]
    , clipper_size, clipper_points[20][2];
const int MAX_POINTS = 20;
void drawPoly(int p[][2], int n)
{
  glBegin(GL_POYGON);
  for(int i=0; i<n; i++)
      glVertex2f(p[i][0], p[i][1]);
  glEnd();
}
int x_intersect(int x1, int y1, int x2, int y2, int x3,
int y3, int x4, int y4)
{
    int num=(x1*y2 - y1*x2)*(x3-x4)-(x1-x2)*(x3*y4-y3*x4);
    int den=(x1-x2)*(y3-y4)-(y1-y2)*(x3-x4);
    return num/den;
}
int y_intersect(int x1, int y1, int x2, int y2, int x3,
int y3, int x4, int y4)
{
  int num=(x1*y2-y1*x2)*(y3-y4)-(y1-y2)*(x3*y4-y3*x4);
  int den=(x1-x2)*(y3-y4)-(y1-y2)*(x3-x4);
  return num/den;
}
void clip(int poly_points[][2], int &poly_size, int x1, int y1,
        int x2, int y2)
{
```

```
int new_points[MAX_POINTS][2], new_poly_size = 0;
for int (i=0; i<poly_size ; i++)
{
  int k=(i+1)% poly_size;
  int ix = poly_points[i][0], iy = poly_points[i][1];
  int kx = poly_points[k][0], ky = poly_points[k][1];
  int i_pos=(x2-x1)*(iy-y1)-(y2-y1)*(ix-x1);
  int k_pos=(x2-x1)*(ky-y1)-(y2-y1)*(kx-x1);
  if (i_pos >= 0 && k_pos >= 0)
  {
    new_points[new_poly_size][0]=kx;
    new_points[new_poly_size][1]=ky;
  } new_poly_size++;
  else if (i_pos<0 && k_pos >=0)
  {
    new_points[new_poly_size][0]=x_intersect(x1, y1,
    x2, y2, ix, iy, kx, ky);
    new_points[new_poly_size][1]=y_intersect(x1, y1, x2, y2, ix, iy, kx, ky);
  } new_poly_size++;
  else
  {
}  }
poly_size = new_poly_size;
for (int i=0; i<poly_size ;i++)
{
  poly_points[i][0]=new_points[i][0];
} poly_points[i][1]=new_points[i][1];
}
void init(){
  glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
  glMatrixMode(GL_PROJECTION);
```

```
    glLoadIdentity();
    glOrtho(0.0, 500.0, 0.0, 500.0, 0.0, 500.0);
}   glClear(GL_COLOR_BUFFER_BIT);
void display()
{   init();
    glColor3f(1.0f, 0.0f, 0.0f);
    drawPoly(clipper-points, clipper-size);
    glColor3f(0.0f, 1.0f, 0.0f);
    drawPoly(org-poly-points, org-poly-size);
    for(int i=0; i<clipper-size; i++)
    {   int k=(i+1)% clipper-size;
        clip(poly-points, poly-size, clipper-points[i][0],
    }   clipper-points[i][1], clipper-points[k][0], clipper points[k][1]);
    glColor3f(0.0f, 0.0f, 1.0f);
    drawPoly(poly-points, poly-size);
    glFlush();
}
int main(int argc, char* argv[])
{   printf("Enter no. of vertices: \n");
    scanf-s("%d", &poly-size);
    org-poly-size = poly-size;
    for(int i=0; i<poly-size; i++)
    {   printf("Polygon Vertex: \n");
        scanf-s("%d%d", &poly-points[i][0], &poly-points[i][1]);
        org-poly-points[i][0] = poly-points[i][0];
        org-poly-points[i][1] = poly-points[i][1];
    }
```

```
printf("Enter no. of vertices of clipping window: ");
scanf-s("%d", &clipper-size);
for (int i=0; i<clipper-size ; i++)
{ printf("Clip Vertex: \n");
  scanf-s("%d%d", &clipper-points[i][0], &clipper-points[i][1]);
}
  org-poly-points[i][0]=poly-points[i][0];
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
glutInitWindowSize(400, 400);
glutInitWindowPosition(100, 100);
glutCreateWindow("Polygon Clipping");
glutDisplayFunc(display);
glutMainLoop();
return 0;
}
```

C:\Users\Vaibhav\source\repos\althaf_8\Debug\althaf_8.exe

```
Enter no. of vertices:
5
Polygon Vertex:
200 200
Polygon Vertex:
300 200
Polygon Vertex:
350 280
Polygon Vertex:
250 360
Polygon Vertex:
150 280
Enter no. of vertices of clipping window:4
Clip Vertex:
175 200
Clip Vertex:
325 200
Clip Vertex:
325 360
Clip Vertex:
175 360
```

Polygon Clipping!

C:\Users\Vaibhav\source\repos\althaf_8\Debug\althaf_8.exe

```
Enter no. of vertices:
4
Polygon Vertex:
80 150
Polygon Vertex:
130 90
Polygon Vertex:
190 250
Polygon Vertex:
130 300
Enter no. of vertices of clipping window:4
Clip Vertex:
100 90
Clip Vertex:
180 90
Clip Vertex:
180 180
Clip Vertex:
100 180
```

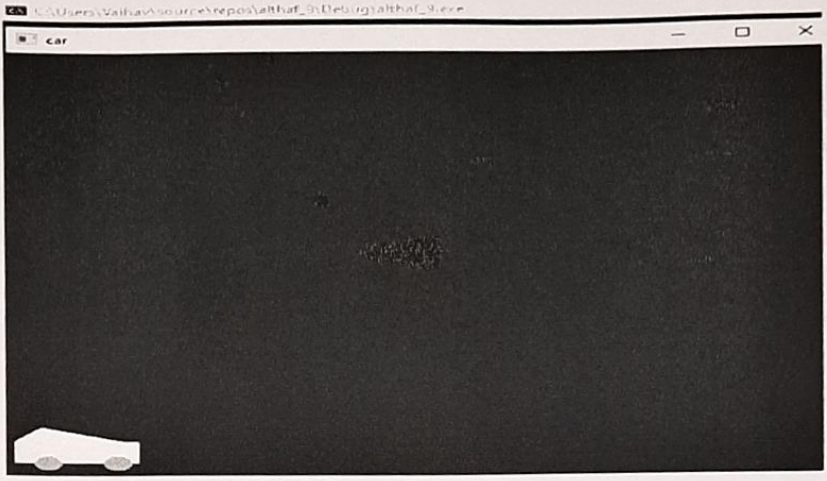
Polygon Clipping!

Q Write program to model a car like figure using display lists and move a car from one end of screen to other. User is able to control the speed with mouse.

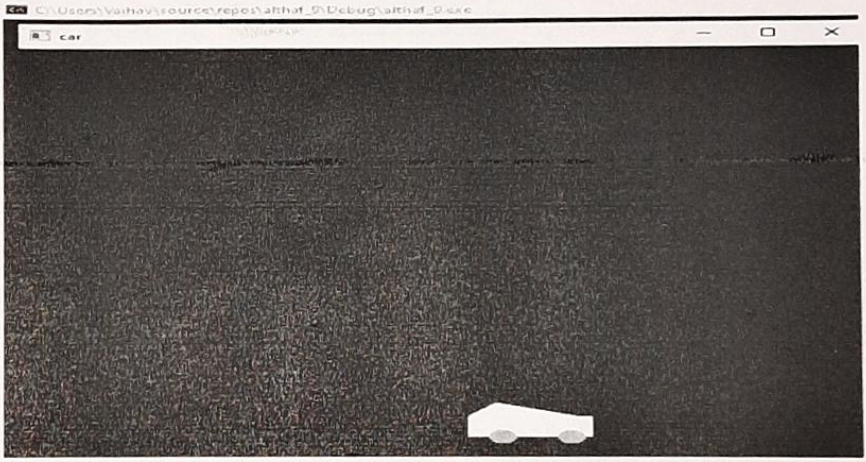
→
```
#include <GL/glut.h>
#include <math.h>
#include <stdio.h>
#define CAR 1
#define WHEEL 2
float s=1;
void carlist() {
    glNewList(CAR, GL_COMPILE);
    glColor3f(1,1,1);
    glBegin(GL_POLYGON);
    glVertex3f(0,25,0);
    glVertex3f(90,25,0);
    glVertex3f(90,55,0);
    glVertex3f(80,55,0);
    glVertex3f(20,75,0);
    glVertex3f(0,55,0);
    glEnd(); glEndList();
}
void wheellist(){
    glNewList(WHEEL, GL_COMPILE_AND_EXECUTE);
    glColor3f(0,1,1);
    glutSolidSphere(10,25,25);
    glEndList();
}
```

```
void mykeyboard(unsigned char key, int x, int y) {
    switch (key) {
    case 't': glutPostRedisplay();
            break;
    case 'q' : exit(0);
    default: break;
    }
}
void myInit() {
    glClearColor(0, 0, 0, 0);
    glOrtho(0, 600, 0, 600, 0, 600);
}
void draw-wheel() {
    glColor3f(0, 1, 1);
    glutSolidSphere(10, 25, 25);
}
void moveCar(float s) {
    glTranslatef(s, 0.0, 0.0);
    glCallList(CAR);
    glPushMatrix();
    glTranslatef(25, 25, 0.0);
    glCallList(WHEEL);
    glPopMatrix();
    glPushMatrix();
    glTranslatef(75, 25, 0.0);
    glCallList(WHEEL);
    glFlush();
}
void myDisp() {
    glClear(GL_COLOR_BUFFER_BIT);
```

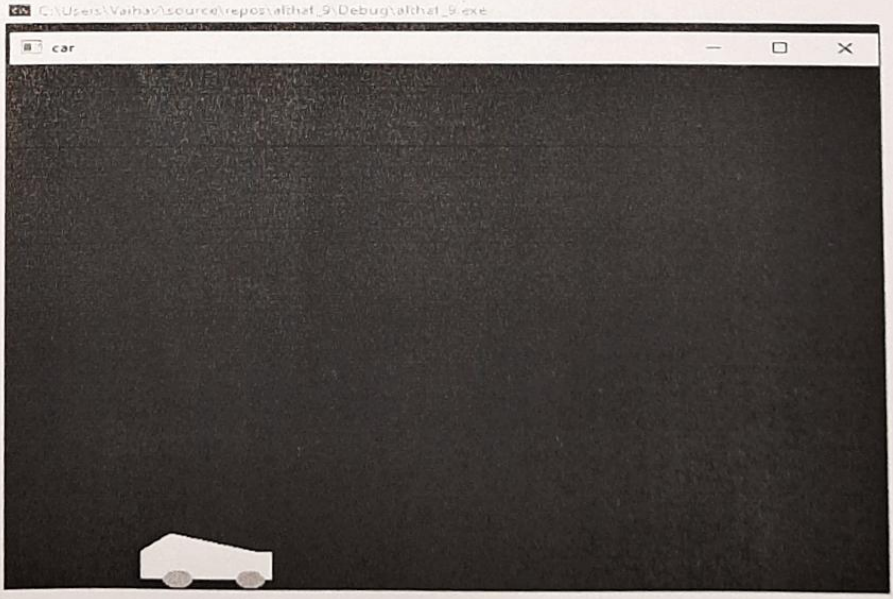
```
} carlist(); moveCar(S); wheellist();
void mouse(int btn, int state, int x, int y){
  if(btn == GLUT_LEFT_BUTTON && state == GLUT_DOWN){
    S+=5 ; myDisp();
  }
  else if(btn == GLUT_RIGHT_BUTTON && state == GLUT_DOWN){
    S+=2 ; myDisp();
  }
}

int main(int argc, char* argv[])
{
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
  glutInitWindowSize(600, 500);
  glutInitWindowPosition(100, 100);
  glutCreateWindow("car");
  myInit();
  glutDisplayFunc(myDisp);
  glutMouseFunc(mouse);
  glutKeyboardFunc(mykeyboard);
  glutMainLoop();
}
```



Click left mouse button to move the car



We can Press t also to move car and q to quit



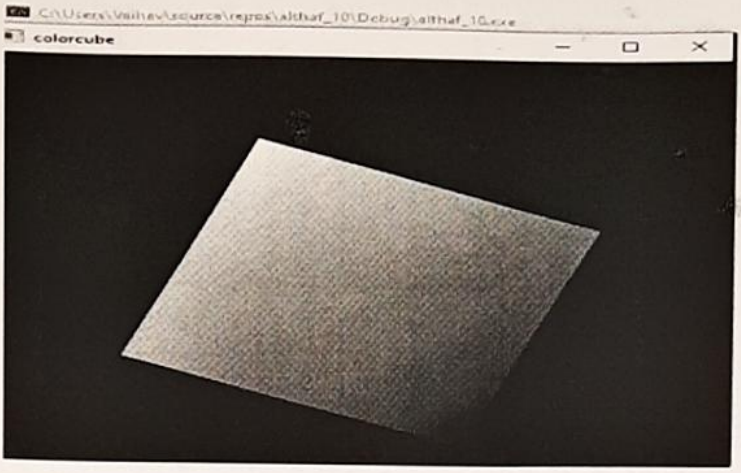
Q Write program to create a color cube and spin it using Open GL Transformations

→
```
#include <stdlib.h>
#include <GL/glut.h>
#include <gl\GL.h>
#include <gl\GLU.h>
#include <time.h>
GLFloat vertices[]={-1.0,-1.0,-1.0,1.0,-1.0,-1.0,1.0,1.0,-1.0,-1.0,1.0,-1.0,
-1.0,-1.0,1.0,1.0,-1.0,1.0,1.0,1.0,1.0,-1.0,1.0,1.0};
GLFloat normals[]={-1.0,-1.0,-1.0,1.0,-1.0,-1.0,1.0,1.0,-1.0,-1.0,
1.0,-1.0,-1.0,-1.0,1.0,1.0,-1.0,1.0,1.0,1.0,1.0,-1.0,1.0,1.0};
GLFloat colors[]={0.0,0.0,0.0,1.0,0.0,0.0,0.0,1.0,1.0,0.0,0.0,1.0,
0.0,0.0,0.0,1.0,1.0,0.0,1.0,1.0,1.0,1.0,0.0,1.0,1.0};
GLubyte cubeIndices[]={0,3,2,1,2,3,7,6,0,4,7,3,1,2,6,5,4,5,6,7,0,1,5,4};
Static GLFloat Theta[]={0.0,0.0,0.0};
Static GLFloat beta[]={0.0,0.0,0.0};
Static GLint axis=2;
void delay(float secs)
{ float end = clock()/CLOCKS_PER_SEC + secs;
  while((clock()/CLOCKS_PER_SEC)<end);
}
void displaySingle(void)
{ glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glLoadIdentity();
  glRotatef(Theta[0], 1.0, 0.0, 0.0);
  glRotatef(Theta[1], 0.0, 1.0, 0.0);
  glRotatef(Theta[2], 0.0, 0.0, 1.0);
```

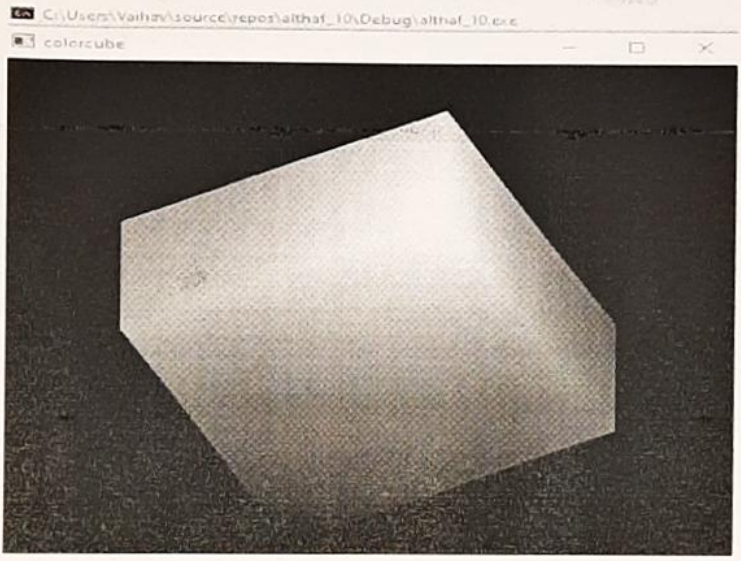
```
    glDrawElements(GL_QUADS, 24, GL_UNSIGNED_BYTE, cubeIndices);
    glBegin(GL_LINES);
    glVertex3f(0.0, 0.0, 0.0);
    glVertex3f(1.0, 1.0, 1.0);
    glEnd();
    glFlush();
}
void spinCube()
{
    delay(0.01);
    theta[axis] += 2.0;
    if(theta[axis] > 360.0) theta[axis] -= 360.0;
    glutPostRedisplay();
}
void mouse(int btn, int state, int x, int y)
{
    if(btn == GLUT_LEFT_BUTTON && state == GLUT_DOWN) axis = 0;
    if(btn == GLUT_MIDDLE_BUTTON && state == GLUT_DOWN) axis = 1;
    if(btn == GLUT_RIGHT_BUTTON && state == GLUT_DOWN) axis = 2;
}
void myReshape(int w, int h)
{
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    if(w <= h)
      glOrtho(-2.0, 2.0, -2.0*(GLFloat)h/(GLFloat)w, 2.0*(GLFloat)h/
        (GLFloat)w, -10.0, 10.0);
    else
      glOrtho(-2.0*(GLFloat)w/(GLFloat)h, 2.0*(GLFloat)w/(GLFloat)h,
        -2.0, 2.0, -10.0, 10.0);
    glMatrixMode(GL_MODELVIEW);
}
```

```
void main (int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowPosition(100, 100);
    glutInitWindowSize(500, 500);
    glutCreateWindow("colorcube");
    glutReshapeFunc(myReshape);
    glutDisplayFunc(displaySingle);
    glutIdleFunc(spinCube);
    glutMouseFunc(mouse);
    glEnable(GL_DEPTH_TEST);
    glEnableClientState(GL_COLOR_ARRAY);
    glEnableClientState(GL_NORMAL_ARRAY),
    glEnableClientState(GL_VERTEX_ARRAY);
    glVertexPointer(3, GL_FLOAT, 0, vertices);
    glVerColorPointer(3, GL_FLOAT, 0, colors),
    glNormalPointer(GL_FLOAT, 0, normals),
    glColor3f(1.0, 1.0, 1.0);
    glutMainLoop();
}
```

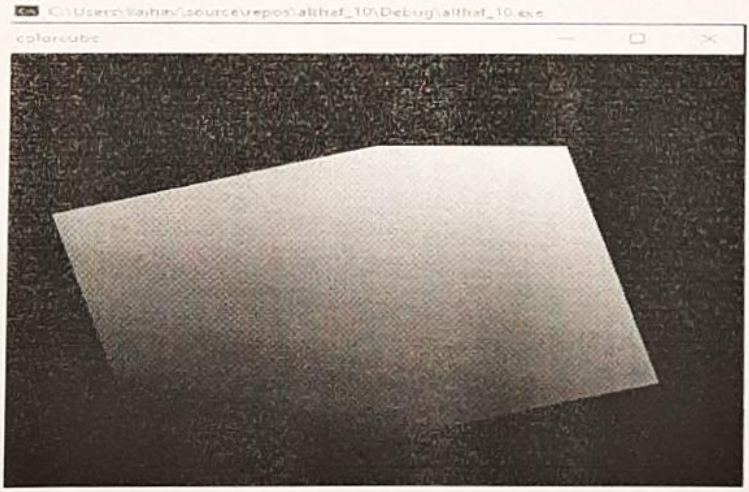
Press left mouse button to rotate about x-axis



Press right mouse button to rotate about y-axis



Press middle mouse button to rotate about z-axis



Q Create a menu with 3 entries named curves, colors & quit. The entry curves has a sub menu which has 4 entries namely Limacon, Cardiod, Three-Leaf, & Spiral. The color menu has sub menu with all 8 colors of RGB color model. Write pgm to create the above hierarchical menu & attach appropriate services to each menu entries with mouse buttons

→
```
#include<gl/glut.h>
#include<math.h>
#include<stdio.h>
struct screenPt {
    int x; int y; };
Typedef enum{ limacon=1, cardioid=2, threeleaf=3,
 spiral=4, } curveName;
int w=600, h=500;
int curve=1;
int red=0, green=0, blue=0;
void myinit (void){
    glClearColor(1.0, 1.0, 1.0, 1.0);
    glMatrixMode(GL_PROJECTION);
    glOrtho2D(0.0, 200.0, 0.0, 150.0);
}
void lineSegment (screenPt p1, screenPt p2)
{   glBegin(GL_LINES);
    glVertex2i(p1.x, p1.y);
    glVertex2i(p2.x, p2.y);
    glEnd(); glFlush();
}
```

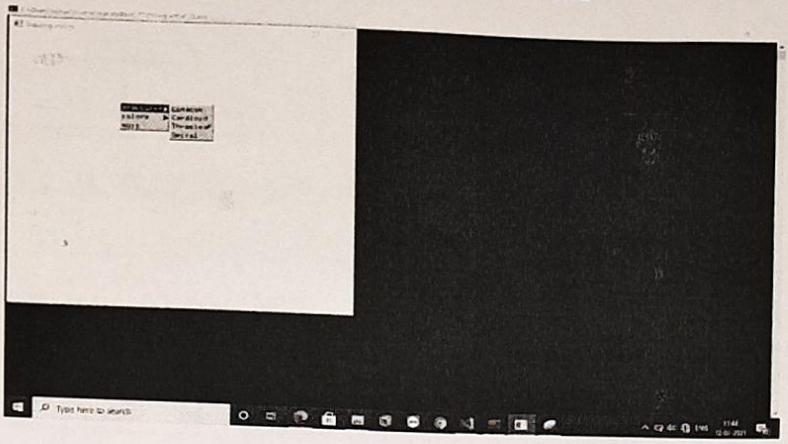
```
void drawCurve(int curveNum){
    const double TwoPi = 6.283185;
    const int a=175, b=60;
    float r, theta, dtheta=1.0/float(a);
    int x0=200, y0=250;
    ScreenPt curvePt[2];
    curve=curveNum;
    glColor3F(red, green, blue);
    curvePt[0].x = x0;
    curvePt[0].y = y0;
    glClear(GL_COLOR_BUFFER_BIT);
    switch(curveNum){
     case limacon: curvePt[0].x+=a+b; break;
     case cardioid: curvePt[0].x+=a+a; break;
     case threeLeaf: curvePt[0].x+=a; break;
     case spiral: break; default: break;
    }
    Theta=dtheta;
    while (Theta < twoPi){
        switch(curveNum){
         case limacon: r=a*cos(Theta)+b; break;
         case cardioid: r=a*(1+cos(theta)); break;
         case threeleaf: r=a*cos(3*Theta); break;
         case spiral: r=(a/4.0)*Theta; break;
         default: break;
        }
        curvePt[1].x = x0 + r*cos(Theta);
        curvePt[1].y = y0 + r*sin(Theta);
```

```
        lineSegment(curvePt[0], curvePt[1]);
        curvePt[0].x = curvePt[1].x;
        curvePt[0].y = curvePt[1].y;
        theta += dtheta;
    }
}
void colorMenu(int id){
    switch(id){
        case 0: break;
        case 1: red=0;
                green=0;
                blue=1; break;
        case 2: red=0;
                green=1
                blue=0; break;
        case 4: red=1;
                green=0;
                blue=0; break;
        case 3: red=0;
                green=1;
                blue=1; break;
        case 5: red=1;
                green=0;
                blue=1; break;
        case 6: red=1; green=1;
                blue=0; break;
```

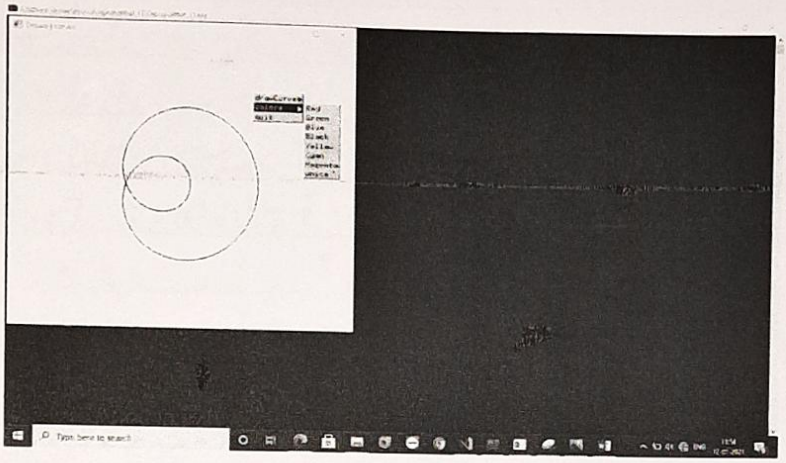
```
            case 7: red=1; green=1;
                    blue=1; break;
            default: break;
        }
        draw curve(curve);
    }
    void main_menu(int id){
    switch (id){
        case 3: exit(0);
        default: break;
    }
}
void mydisplay(){
  glClear(GL_COLOR_BUFFER_BIT);
/*printf("1-limacon\n 2-cardioid \n 3-Threeleaf\n
        4-spiral \n");
  scanf-s("%d", &curveNum); */
}
void myreshape(int nw, int nh){
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluOrtho2D(0.0,(double) nw, 0.0,(double) nh);
  glClear(GL_COLOR_BUFFER_BIT);
}
void main(int argc, char** argv)
{ glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
  glutInitWindowSize(w, h);
  glutInitWindowPosition(100, 100);
```

```
glutCreateWindow("Drawing curves");
int curveId = glutCreateMenu(drawCurves);
glutAddMenuEntry("Limacon", 1);
glutAddMenuEntry("Cardioid", 2);
glutAddMenuEntry("Threeleaf", 3);
glutAddMenuEntry("Spiral", 4);
glutAttachMenu(GLUT_LEFT_BUTTON);
int colorId = glutCreateMenu(colorMenu);
glutAddMenuEntry("Red", 4);
glutAddMenuEntry("Green", 2);
glutAddMenuEntry("Blue", 1);
glutAddMenuEntry("Black", 0);
glutAddMenuEntry("Yellow", 6);
glutAddMenuEntry("Cyan", 3);
glutAddMenuEntry("Magenta", 5);
glutAddMenuEntry("white", 7);
glutAttachMenu(GLUT_LEFT_BUTTON);
glutCreateMenu(main_menu);
glutAddSubMenu("drawCurves", curveId);
glutAddSubMenu("colors", colorId);
glutAddMenuEntry("quit", 3);
glutAttachMenu(GLUT_LEFT_BUTTON);
myinit(); glutDisplayFunc(mydisplay);
glutReshapeFunc(myreshape);
glutMainLoop();
}
```

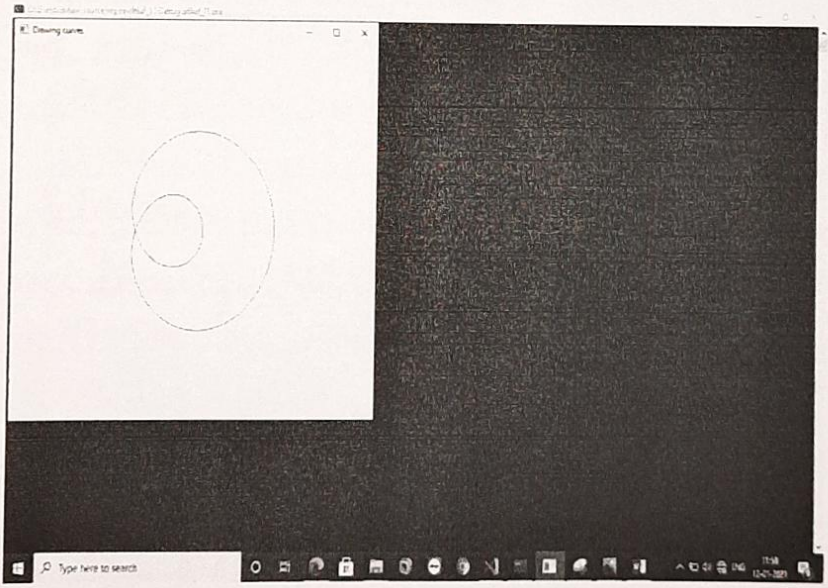
Click left mouse button to show and select the menu



We can choose colour or quit



Choose colour



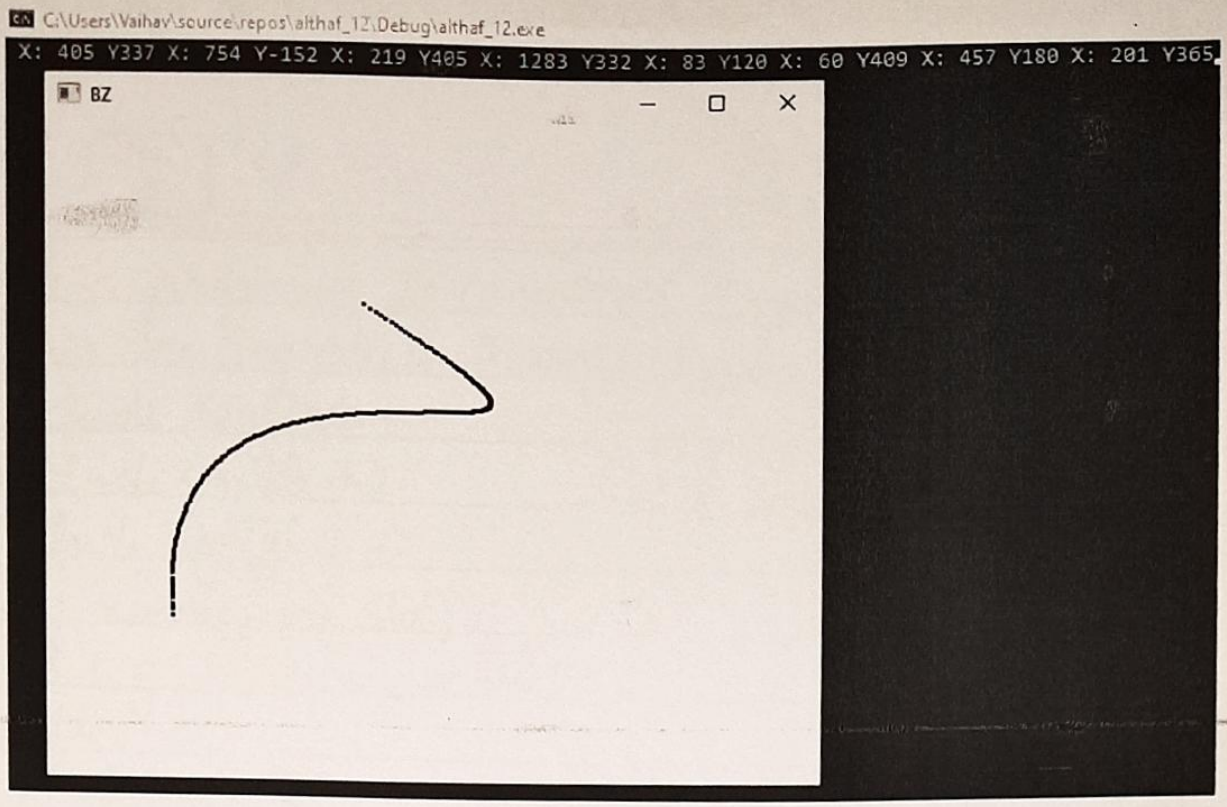
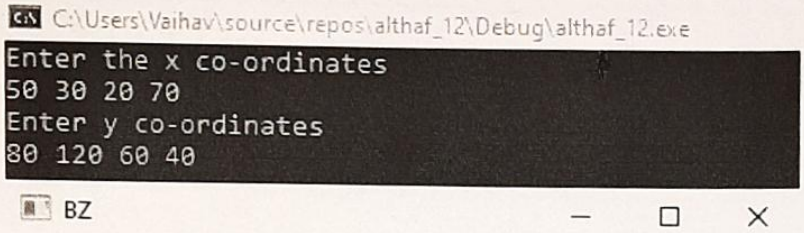
Q Write a program to construct Bezier curve control points are supplied through keyboard/mouse

→
```
#include <iostream>
#include <math.h>
#include <gl/glut.h>
using namespace std;
float f, g, h, xc[4], yc[4];
int flag = 0;
void myInit(){
    glClearColor(1,1,1,1);
    glColor3f(1,1,1);
    glPointSize(5);
    gluOrtho2D(0,500,0,500);
}
void drawPixel(Float x, float y)
{
    glBegin(GL_POINTS);
    glVertex2f(x, y);
    glEnd();
}
void display(){
    glClear(GL_COLOR_BUFFER_BIT);
    int i;  double t;
    glColor3f(0,0,0);
    glBegin(GL_POINTS);
```

```
for (t=0; t<1; t=t+0.005) {
    double xt = pow(1-t,3)* x1[0]+3*t* pow(1-t,2)*
    x1[1]+3*pow(t,2)*(1-t)* x1[2]+ pow(t,3)* x1[3];
    double yt = pow(1-t,3)* yc[0]+3*t* pow(1-t,2)*
    yc[1]+3*pow(t,2)*(1-t)* yc[2]+pow(t,3)*yc[3];
    glVertex2f(xt, yt);
  }
  glColor3f(1,1,0);
  for(i=0;i<4;i++)
  {
    glVertex2f(x1[i], yc[i]);
  } glEnd(); glFlush();
}
void mymouse(int btn, int state, int x, int y)
{ if(btn==GLUT_LEFT_BUTTON && state==GLUT_DOWN && flag<4)
  {
    x1[flag]=x;
    yc[flag]=500-x;
    cout << "x:" << x << "y" << 500-y;
    glPointSize(3);
    glColor3f(1,1,0);
    glBegin(GL_POINTS);
    glVertex2i(x, 500-y);
    glEnd();
    glFlush();
    iflag++;
  }
```

```
        if (flag>=4 && btn==GLUT_LEFT_BUTTON)
        { glColor3f(0,0,1);
          display();
          flag=0;
        }
}

int main(int argc, char* argv[])
{
  glutInit(&argc, argv);
/* cout << "Enter the x co-ordinates";
   cin >> x1[0] >> x1[1] >> x1[2] >> x1[3];
   cout << "Enter y co-ordinates";
   cin >> yc[0] >> yc[1] >> yc[2] >> yc[3]; */
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
  glutInitWindowSize(500, 500);
  glutInitWindowPosition(0,0);
  glutCreateWindow("BZ");
  glutDisplayFunc(display);
  glutMouseFunc(mymouse);
  myInit();
  glutMainLoop();
}
```

Draw BZ curve through mouse



Enter co-ordinates via keyboard