COMPUTER GRAPHICS AND IMAGE PROCESSING LABORATORY(21CSL66)

LAB PROGRAM 1:

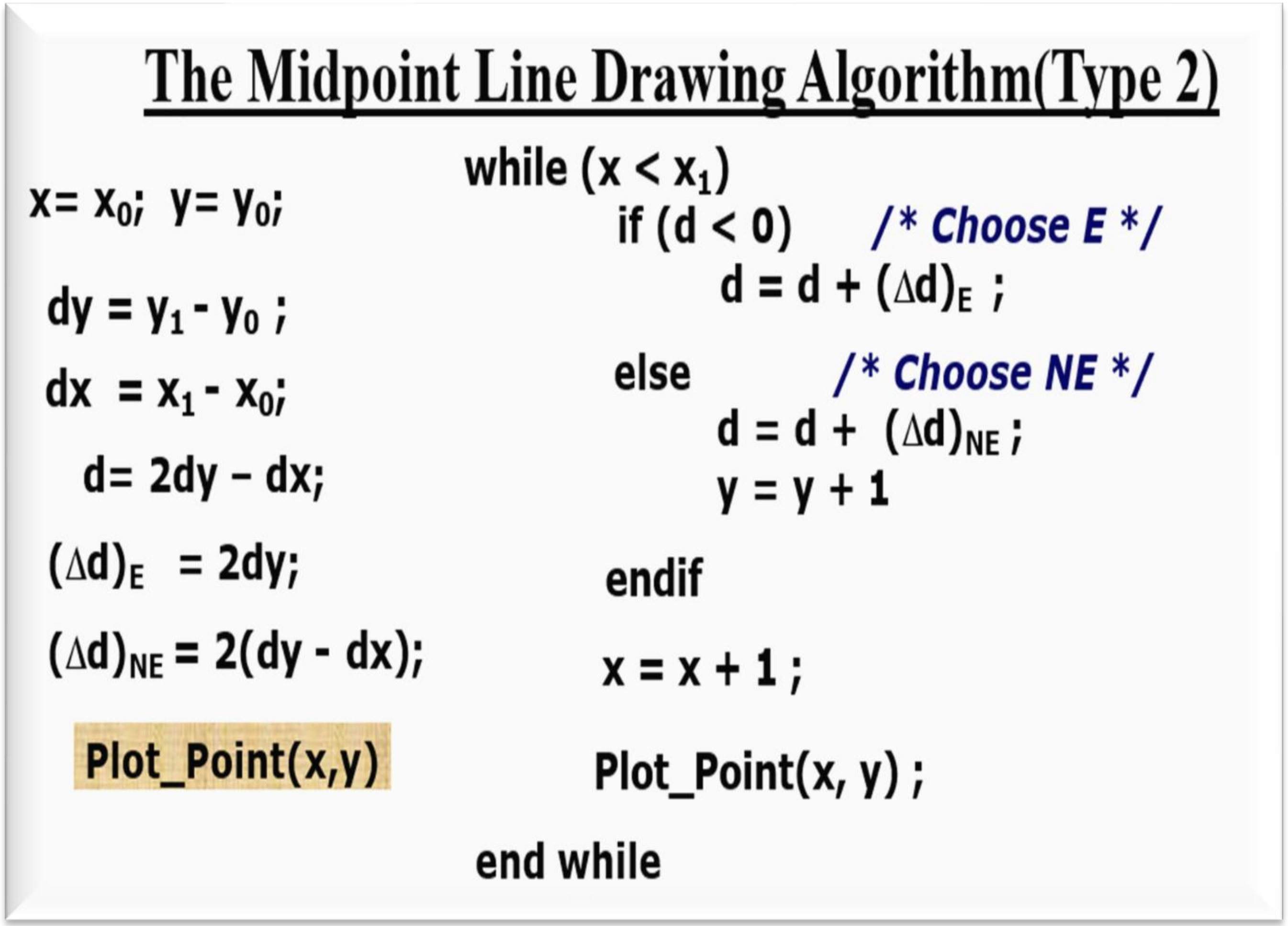
Develop a program to draw a line using Bresenham's line drawing algorithm for all types of slope.

Implement Bresenham’s line drawing algorithm for all types of slope

```
#include <stdio.h>
#include <gl/glut.h>

int x1, y1, x2, y2;

void myInit()
{
    glClearColor(0.0, 0.0, 0.0, 1.0);
    glClear(GL_COLOR_BUFFER_BIT);
    glMatrixMode(GL_MODELVIEW);
    gluOrtho2D(0, 500, 0, 500);
}
```

```
void draw_pixel(int x, int y)
{
    glBegin(GL_POINTS);
      glVertex2i(x, y);
    glEnd();
}
   void draw_line(int x1, int x2, int y1, int y2)
{
    int dx, dy, i, pk;
    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);
pk = 2 * dy-dx;
inc1 = 2*(dy-dx);
inc2 = 2*dy;
for (i=0; i<dx; i++)
 {
     if (pk >= 0)
{
     y += incy;
   pk += inc1;
}
    else
  pk += inc2;
    x += incx;
  draw_pixel(x, y);
} }
```



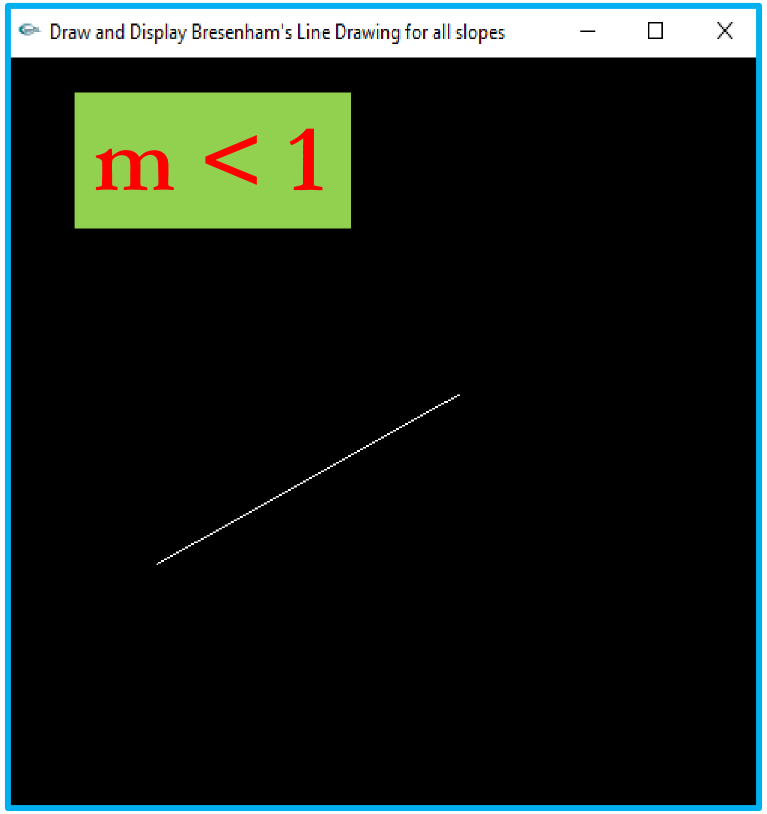
The Midpoint Line Drawing Algorithm(Type 2)

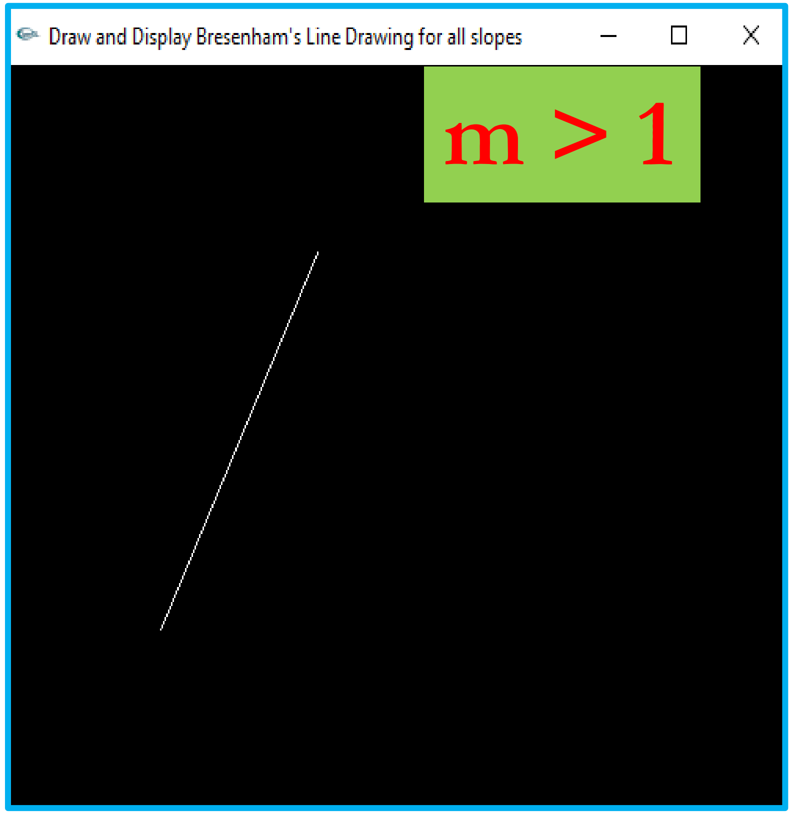
```
x= $x_0$;  y= $y_0$;

 dy = $y_1$ - $y_0$ ;
 dx  = $x_1$ - $x_0$;
   d= 2dy – dx;
 $(\Delta d)_E$  = 2dy;
 $(\Delta d)_{NE}$ = 2(dy - dx);
   Plot_Point(x,y)

while (x < $x_1$)
     if (d < 0)        /* Choose E */
          d = d + $(\Delta d)_E$ ;
     else              /* Choose NE */
          d = d +  $(\Delta d)_{NE}$ ;
          y = y + 1
     endif
     x = x + 1 ;
     Plot_Point(x, y) ;
end while
```

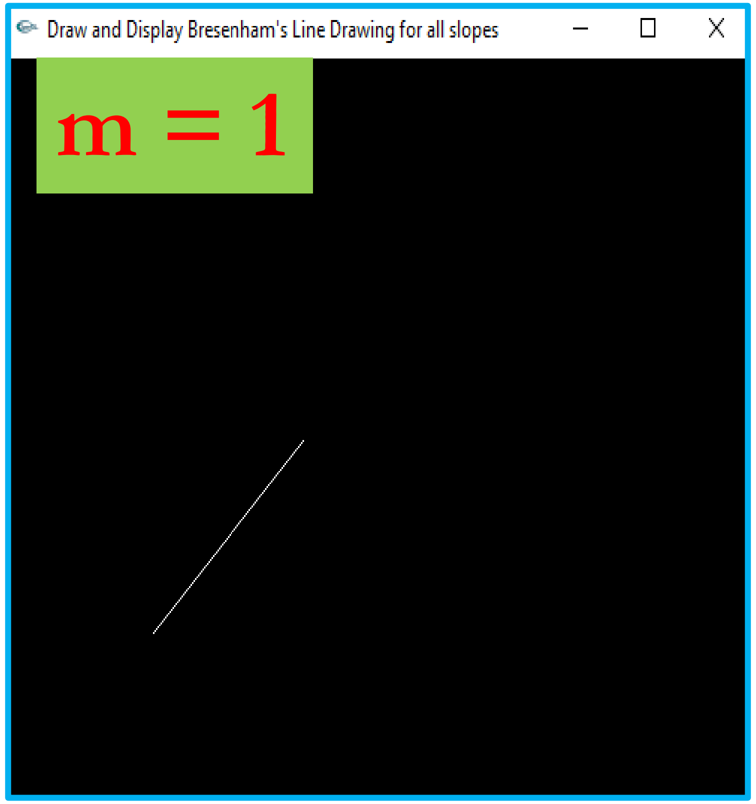
```
else
{
      draw_pixel(x, y);
      pk = 2*dx-dy;
      inc1 = 2*(dx-dy);
      inc2 = 2*dx;
     for (i=0; i<dy; i++)
{
       if (pk >= 0)
{
          x += incx;
        pk += inc1;
}
else
    pk += inc2;
     y += incy;
   draw_pixel(x, y);
   }
 }
}
void myDisplay()
 {
      draw_line(x1, x2,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

```
void main(int argc, char **argv)
{
printf( "Enter the Start point of the Line (x1,y1)\n");
scanf("%d %d", &x1,&y1);
printf( "Enter the End point of the Line (x2, y2)\n");
scanf("%d %d", &x2,&y2);
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutInitWindowSize(500, 500);
glutInitWindowPosition(0, 0);
glutCreateWindow(“Draw and Display Bresenham's Line Drawing for all slopes”);
myInit();
glutDisplayFunc(myDisplay);
glutMainLoop();
}
```

Implement Bresenham’s line drawing algorithm for all types of slope - Output

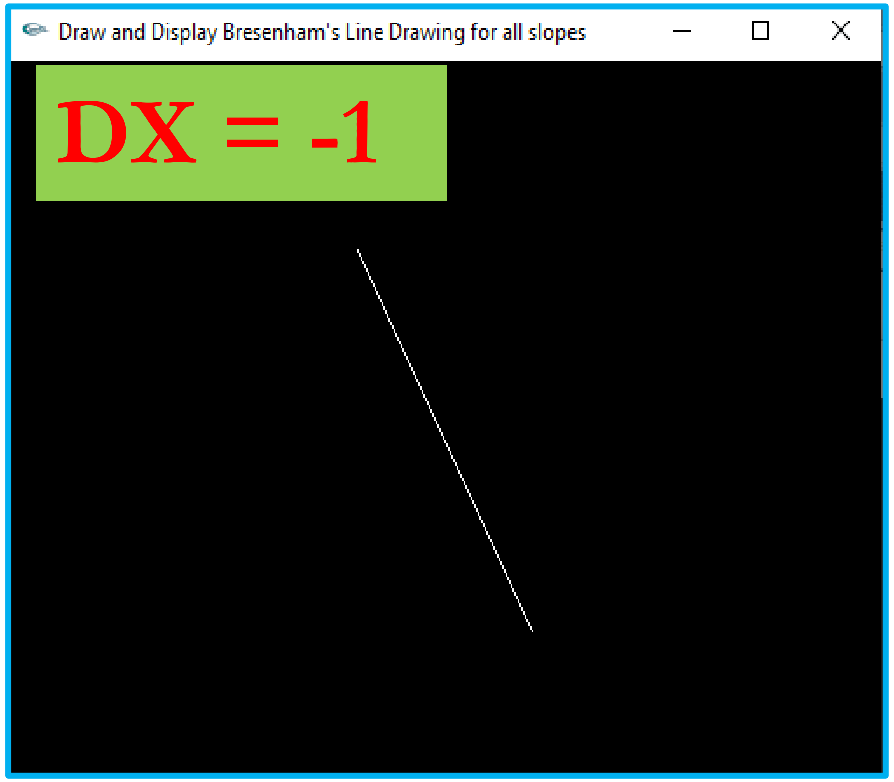


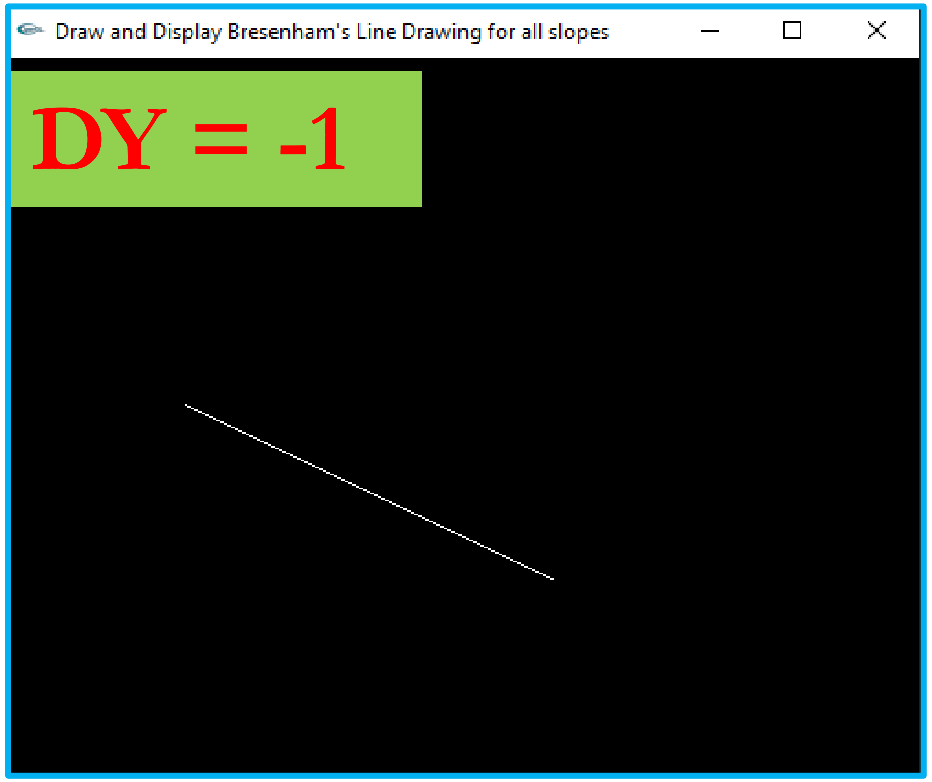




Implement Bresenham's line drawing algorithm for all types of slope - Output

Negative slope





Lab program : 2

Develop a program to demonstrate basic geometric operations on the 2D object

Recursive subdivision of triangle to form Sierpinski gasket – 2D

Sierpinski Gasket Algorithm:

A fractal object defined recursively and randomly

1. Take three points in a plane to form a triangle.

2. Randomly select any point inside the triangle and consider that your current position.

3. Randomly select any one of the three vertex points.

4. Move half the distance from your current position to the selected vertex.

5. Plot the current position.

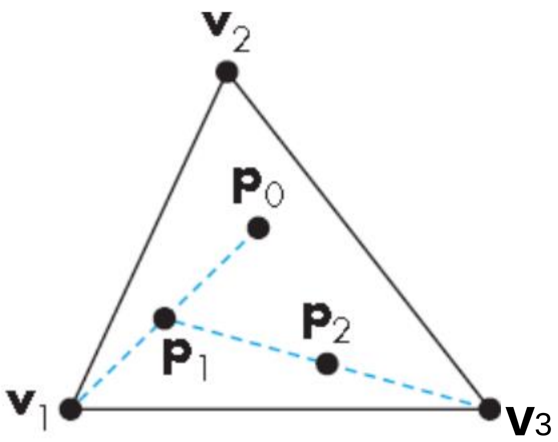
6. Repeat from step 3.

Sierpinski Gasket Algorithm:

A fractal object defined recursively and randomly

1. Pick an initial point p0 randomly inside the triangle
2. Select one of the vertices randomly
3. Find a point p1 at the middle of the line segment

   v1p0
4. Replace p0 with p1
5. Go back to step 2.



```
/* Recursive subdivision of triangle to form Sierpinski gasket – 2D */

#include <stdio.h>
#include <GL/glut.h>

typedef float point2[2];

/* initial triangle */

point2 v[ ]={{-1.0, -0.58}, {1.0, -0.58}, {0.0, 1.15}};

int n;
```

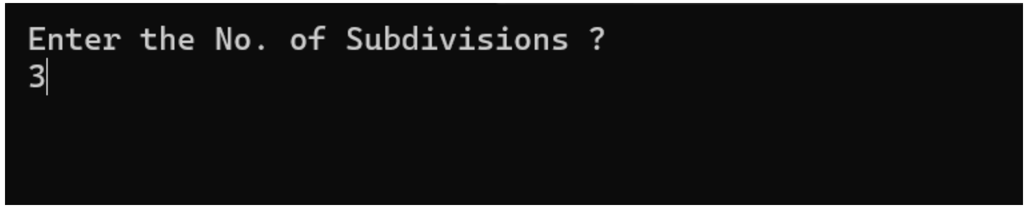
```
void triangle( point2 a, point2 b, point2 c)
/* display one triangle  */
{
    glBegin(GL_TRIANGLES);
       glVertex2fv(a);
       glVertex2fv(b);
       glVertex2fv(c);
    glEnd();
}
```

```
void divide_triangle(point2 a, point2 b, point2 c, int m)
{

    point2 v0, v1, v2;
    int j;
    if(m>0)
    {
        for(j=0; j<2; j++) v0[j]=(a[j]+b[j])/2;
        for(j=0; j<2; j++) v1[j]=(a[j]+c[j])/2;
        for(j=0; j<2; j++) v2[j]=(b[j]+c[j])/2;
        divide_triangle(a, v0, v1, m-1);
        divide_triangle(c, v1, v2, m-1);
        divide_triangle(b, v2, v0, m-1);
    }
    else(triangle(a,b,c)); /* draw triangle at end of recursion */
}
```

```
void display(void)
{

    glClear(GL_COLOR_BUFFER_BIT);
          divide_triangle(v[0], v[1], v[2], n);
    glFlush();
}
void myinit()
{
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluOrtho2D(-2.0, 2.0, -2.0, 2.0);
    glClearColor (1.0, 1.0, 1.0, 1.0);
    glColor3f(0.0,0.0,0.0);
}
```

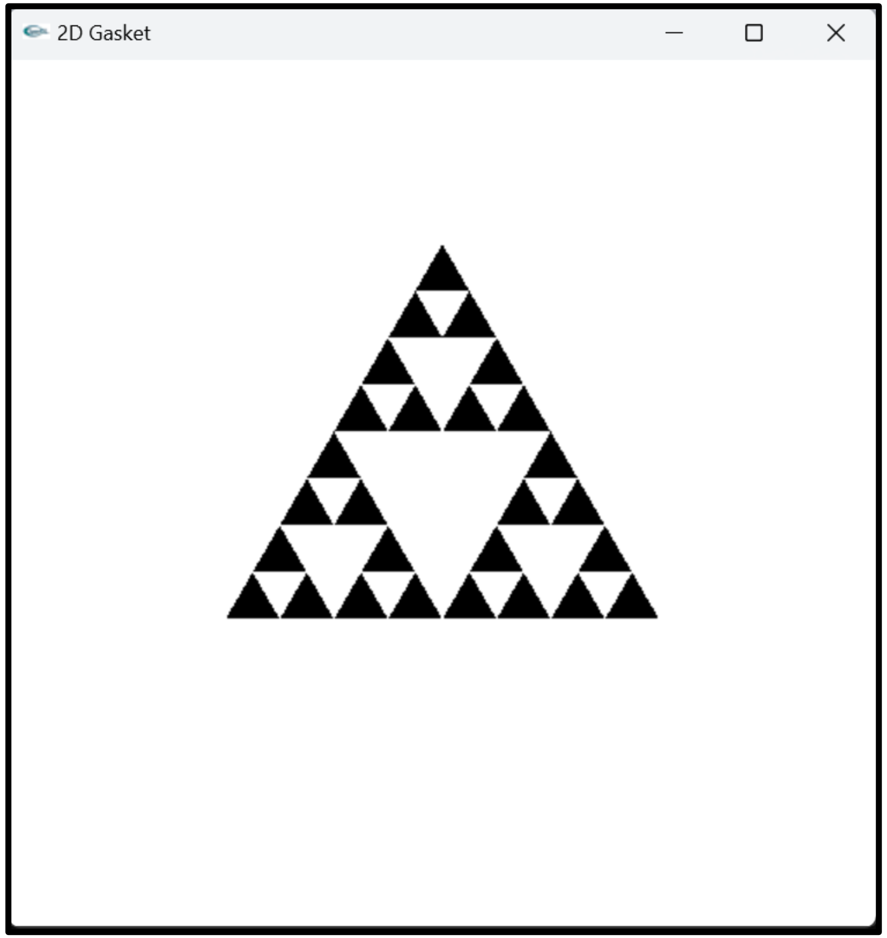
```
void main(int argc, char **argv)
{
   printf(" Enter the No. of Subdivisions ? \n");
   scanf("%d",&n);
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB );
    glutInitWindowSize(500, 500);
    glutCreateWindow(“2D Gasket”);
    glutDisplayFunc(display);
          myinit();
    glutMainLoop();
}
```

Recursive subdivision of triangle to form Sierpinski gasket – 2D - Output



```
Enter the No. of Subdivisions ?
3
```

n=3



Lab program:3

Develop a program to demonstrate basic geometric operations on the 3D object

Design, develop and implement recursively subdivide a tetrahedron to form 3D Sierpinski gasket.

```
/* Recursive subdivision of tetrahedron to form 3D Sierpinski gasket */

#include <stdlib.h>
#include <stdio.h>
#include <GL/glut.h>

typedef float point[3];

/* initial tetrahedron */

point v[]={{0.0, 0.0, 1.0}, {0.0, 0.942809, -0.33333},
         {-0.816497, -0.471405, -0.333333}, {0.816497, -0.471405, -0.333333}};

int n;
```

```
/* display one triangle using a line loop for wire frame, a single
normal for constant shading, or three normals for interpolative shading */
{
    glBegin(GL_POLYGON);
        glNormal3fv(a);
        glVertex3fv(a);
        glVertex3fv(b);
        glVertex3fv(c);
    glEnd();
}
```

```
void divide_triangle(point a, point b, point c, int m)
{
/* triangle subdivision using vertex numbers right hand rule applied to create
outward pointing faces */
    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(triangle(a,b,c)); /* draw triangle at end of recursion */
}
```

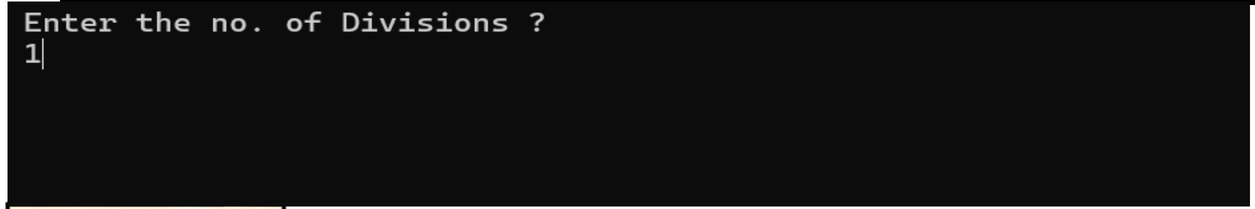
```
void tetrahedron( int m)
{

/* Apply triangle subdivision to faces of tetrahedron */

        glColor3f(1.0,0.0,0.0);
    divide_triangle(v[0], v[1], v[2], m);
        glColor3f(0.0,1.0,0.0);
    divide_triangle(v[3], v[2], v[1], m);
        glColor3f(0.0,0.0,1.0);
    divide_triangle(v[0], v[3], v[1], m);
        glColor3f(0.0,0.0,0.0);
    divide_triangle(v[0], v[2], v[3], m);
}
```

```
void display(void)
{

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
              glLoadIdentity();
    tetrahedron(n);
    glFlush();
}
```

```
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);
    glutPostRedisplay();
}
```

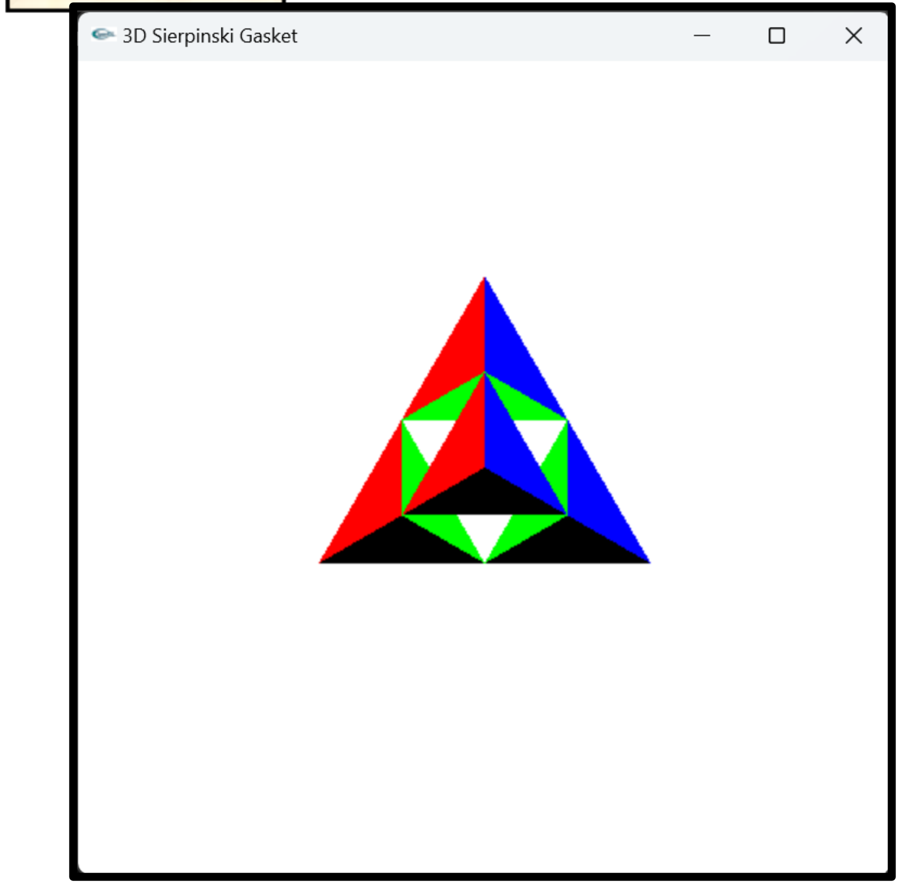
```
void main(int argc, char **argv)
{
        printf(" Enter the no. of Divisions ? \n");
        scanf("%d",&n);
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(500, 500);
    glutCreateWindow("3D Gasket");
    glutReshapeFunc(myReshape);
    glutDisplayFunc(display);
        glEnable(GL_DEPTH_TEST);
    glClearColor (1.0, 1.0, 1.0, 1.0);
    glutMainLoop();
}
```

Recursive subdivision of triangle to form Sierpinski gasket – 3D - Output

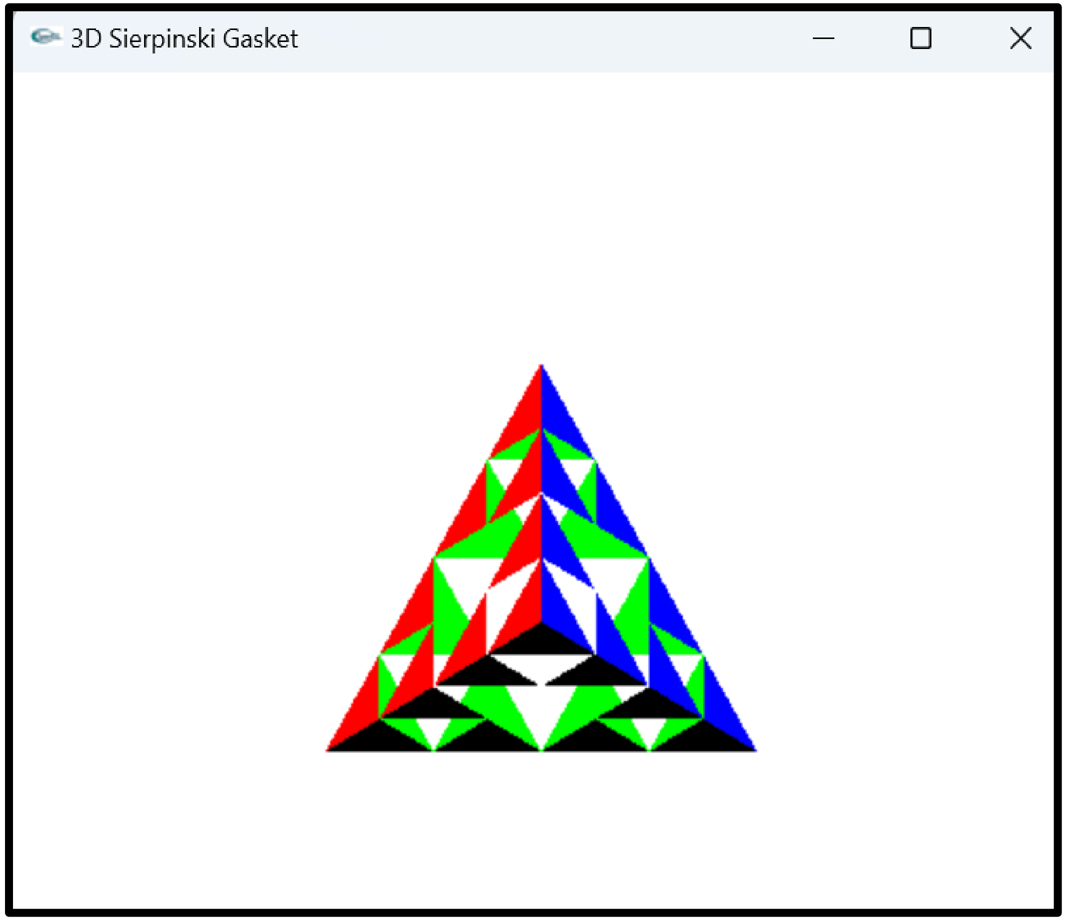


```
Enter the no. of Divisions ?
1
```

n=1



n=2


3D Sierpinski Gasket