LAB MANUAL
Computer Graphics Laboratory
With Mini Project
[18CSL67]

Prepared By,
Prof. Shryavani K
Assistant Professor
Department of CSE
BIET

Prof. Sumana C
Assistant Professor
Department of CSE
BIET

Department of Computer Science and Engineering
Bapuji Institute of Engineering and Technology

SYLLABUS

COMPUTER GRAPHICS LABORATORY WITH MINI PROJECT
[As per Choice Based Credit System (CBCS) scheme]
(Effective from the academic year 2018 -2019)

Course Objectives: This course will enable students to

- **Demonstrate simple algorithms using OpenGL Graphics Primitives and attributes.**
- **Implementation of line drawing and clipping algorithms using OpenGL functions.**
- **Design and implementation of algorithms Geometric transformations on both 2D and 3D objects.**

Course outcome: This course will enable students to

- **Implement the concepts of computer graphics primitives like Lines, Polygons (Traingle, Cube and 3D gasket).**
- **Applying different lighting and shading properties to the surface objects.**
- **Animate the real world problems using OpenGL API's.**
- **Design and Implement the computer graphics applications(Miniproject) using OpenGL**

Part A

SL NO	PROGRAMS
01	Implement Bresenham's line drawing algorithm for all types of slope
02	Create and rotate a triangle about the origin and a fixed point
03	Draw a color cube and spin it using OpenGL transformation matrices
04	Draw a color cube and allow the user to move the camera suitably to experiment with perspective viewing
05	Clip a lines using Cohen-Sutherland algorithm
06	To draw a simple shaded scene consisting of a tea pot on a table. Define suitably the position and properties of the light source along with the properties of the surfaces of the solid object used in the scene
07	Design, develop and implement recursively subdivide a tetrahedron to form 3D Sierpinski gasket. The number of recursive steps is to be specified by the user
08	Develop a menu driven program to animate a flag using Bezier Curve algorithm
09	Develop a menu driven program to fill the polygon using scan line algorithm

Introduction to Computer Graphics
Introduction to Open GL

OpenGL is a software interface to graphics hardware. This interface consists of about 150 distinct commands that you use to specify the objects and operations needed to produce interactive three-dimensional applications. OpenGL is designed as a streamlined, hardware-independent interface to be implemented on many different hardware platforms. With OpenGL, you can build up your desired model from a small set of *geometric primitives* - points, lines, and polygons. A sophisticated library that provides these features could certainly be built on top of OpenGL. The OpenGL Utility Library (GLU) provides many of the modeling features. GLU is a standard part of every OpenGL implementation.

OpenGL as a State Machine

OpenGL is a state machine. It is called a state machine because it can be put into various states until you change them. As you've already seen, the current color is a state variable. You can set the current color to white, red, or any other color, and thereafter every object is drawn with that color until you set the current color to something else.

The current color is only one of many state variables that OpenGL maintains. Others control such things as the current viewing and projection transformations; line and polygon stipple patterns, polygon drawing modes, pixel-packing conventions, positions and characteristics of lights, and material properties of the objects being drawn. Many state variables refer to modes that are enabled or disabled with the command **glEnable()** or **glDisable()**. Each state variable or mode has a default value, and at any point you can query the system for each variable's current value.

OpenGL-Related Libraries

OpenGL provides a powerful but primitive set of rendering commands, and all higher-level drawing must be done in terms of these commands. Also, OpenGL programs have to use the underlying mechanisms of the windowing system. A number of libraries exist to allow you to simplify your programming tasks, including the following:

- The OpenGL Utility Library (GLU) contains several routines that use lower-level OpenGL commands to perform such tasks as setting up matrices for specific viewing orientations and projections, performing polygon tessellation, and rendering surfaces. This library is provided as part of every OpenGL implementation. GLU routines use the prefix **glu**.
- The OpenGL Utility Toolkit (GLUT) is a window system-independent toolkit. It contains rendering commands but is designed to be independent of any window system or operating system. Consequently, it contains no commands for opening windows or reading events from the keyboard or mouse. Since OpenGL drawing commands are limited to those that generate simple geometric primitives (points, lines, and polygons), GLUT includes several routines that create more complicated three-dimensional objects such as a sphere, a torus, and a teapot. GLUT may not be satisfactory for full-featured OpenGL applications, but you may find it a useful starting point for learning OpenGL.

Include Files

For all OpenGL applications, you want to include the gl.h header file in every file. Almost all OpenGL applications use GLU, the aforementioned OpenGL Utility Library, which requires inclusion of the glu.h header file. So almost every OpenGL source file begins with

#include <GL/gl.h>

#include <GL/glu.h>

If you are using GLUT for managing your window manager tasks, you should include

#include <GL/glut.h>

Note that glut.h includes gl.h, glu.h automatically, so including all three files is redundant.

Associated utility libraries

Several libraries are built on top of or beside OpenGL to provide features not available in OpenGL itself. Libraries such as GLU can be found with most OpenGL implementations, and others such as GLUT and SDL have grown over time and provide rudimentary cross-platform windowing and mouse functionality, and if unavailable can easily be downloaded and added to a development environment. Simple graphical user interface functionality can be found in libraries like GLUI or FLTK. Still other libraries like GLAux (OpenGL Auxiliary Library) are deprecated and have been superseded by functionality commonly available in more popular libraries.

Other libraries have been created to provide OpenGL application developers a simple means of managing OpenGL extensions and versioning. Examples of these libraries include GLEW (the OpenGL Extension Wrangler Library) and GLEE (the OpenGL Easy Extension Library). In addition to the aforementioned simple libraries, other higher-level object-oriented scene graph retained mode libraries exist such as PLIB, OpenSG, OpenSceneGraph, and OpenGL Performer. These are available as cross-platform free/open source or proprietary programming interfaces written on top of OpenGL and systems libraries to enable the creation of real-time visual simulation applications.

Comprises several libraries with varying levels of abstraction: GL, GLU, and GLUT

- Software Interface to Graphics Hardware
- Consists of about 150 Distinct Commands
- Hardware-independent Interface
    - no command for windows or user input handling
    - does not include low-level I/O management
- Mid-level, device-independent, portable graphics subroutine package
- Developed primarily by SGI
- 2D/3D graphics, lower-level primitives (polygons)
- Basis for higher-level libraries/toolkits

OpenGL Hierarchy

- **Several levels of abstraction are provided**
- **GL**
  - Lowest level: vertex, matrix manipulation
  - glVertex3f(point.x, point.y, point.z)
- **GLU**
  - Helper functions for shapes, transformations
  - gluPerspective(fovy, aspect, near, far)
  - gluLookAt(0, 0, 10, 0, 0, 0, 0, 1, 0);
- **GLUT**
  - Highest level: Window and interface management
  - glutSwapBuffers()
  - glutInitWindowSize (500, 500);

OpenGL Implementations

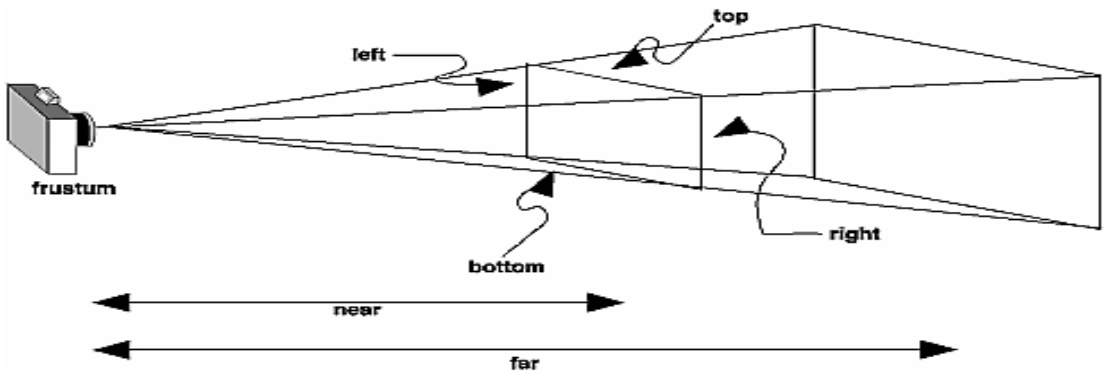
- OpenGL IS an API (think of as collection of .h files):
  - #include <GL/gl.h>
  - #include <GL/glu.h>
  - #include <GL/glut.h>
- Windows, Linux, UNIX, etc. all provide a platform specific implementation.
- Windows: opengl32.lib glu32.lib glut32.lib
- Linux: -l GL -l GLU –l GLUT

OpenGL API

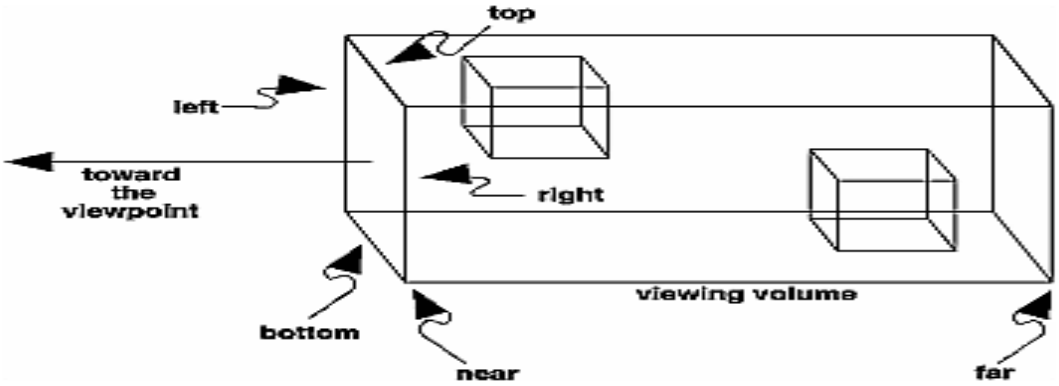
- As a programmer, you need to do the following things:
  - Specify the location/parameters of camera.
  - Specify the geometry (and appearance).
  - Specify the lights (optional).
- OpenGL will compute the resulting 2D image

OpenGL: Camera

- Two things to specify:
  - Physical location of camera in the scene (MODELVIEW matrix in OpenGL).
  - Projection properties of the camera (PROJECTION matrix in OpenGL):



void glFrustum(GLdouble left, GLdouble right, GLdouble bottom,GLdouble top, GLdouble near, GLdouble far);



void glOrtho(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far);

OpenGL Conventions

- Many functions have multiple forms:
  - glVertex2f, glVertex3i, glVertex4dv, etc.
- Number indicates number of arguments
- Letters indicate type
  - f: float, d: double, ub: unsigned byte, etc.
- "v" (if present) indicates a single pointer argument

Required files for Windows

- In the System Directory
  - glu32.dll
  - opengl32.dll
  - glut32.dll
- In the C++ Include Directory
  - gl\gl.h
  - l\glu.h
  - gl\glaux.h (probably won't need it)
  - gl\glut.h (includes both gl.h and glu.h)
- In the C++ Library Directory
  - gl\glu32.lib
  - l\opengl32.lib
  - gl\glaux.lib (probably won't need it)
  - gl\glut32.lib

Event Loop

- OpenGL programs often run in an event loop:
  - Start the program
  - Run some initialization code
  - Run an infinite loop and wait for events such as
    - Key press
    - Mouse move, click
    - Reshape window

- Expose event

OpenGL Command Syntax (1)

- OpenGL commands start with "gl"
- OpenGL constants start with "GL_"
- Some commands end in a number and one, two or three letters at the end (indicating number and type of arguments)
- A Number indicates number of arguments
- Characters indicate type of argument

OpenGL Command Syntax (2)

Suffix	Data Type	C-Type	OpenGL Type
b	8-bit integer	signed char	GLbyte
s	16-bit integer	short	GLshort
i	32-bit integer	long	GLint, GLsizei
f	32-bit float	float	GLfloat, GLclampf
d	64-bit float	double	GLdouble, GLclampd
ub	8-bit unsigned	unsigned char	GLubyte, GLboolean
us	16-bit unsigned	unsigned short	GLushort
ui	32-bit unsigned	unsigned int	GLint, GLenum, GLbitfield

OpenGL Command Syntax (3)

- **glClearColor()** – Specifies the background color
- **glClear()** – Erases the output with background color
- **glMatrixMode()** – Chooses projection/modelview matrix
- **glBegin()/glEnd()** – Model data pumped within this block
- **glVertex()** – Pumps vertex data into OpenGL
- **glViewport()** – Resizes the OpenGL viewport
- **glOrtho()** – Specifies orthogonal view volume
- **glPolygonMode()** – Specifies whether to draw filled polygons or wire-frame polygons

OpenGL Primitives

Value	Meaning
GL_POINTS	individual points
GL_LINES	pairs of vertices interpreted as individual line segments
GL_POLYGON	boundary of a simple, convex polygon
GL_TRIANGLES	triples of vertices interpreted as triangles
GL_QUADS	quadruples of vertices interpreted as four-sided polygons
GL_LINE_STRIP	series of connected line segments
GL_LINE_LOOP	same as above, with a segment added between last and first vertices
GL_TRIANGLE_STRIP	linked strip of triangles
GL_TRIANGLE_FAN	linked fan of triangles
GL_QUAD_STRIP	linked strip of quadrilaterals

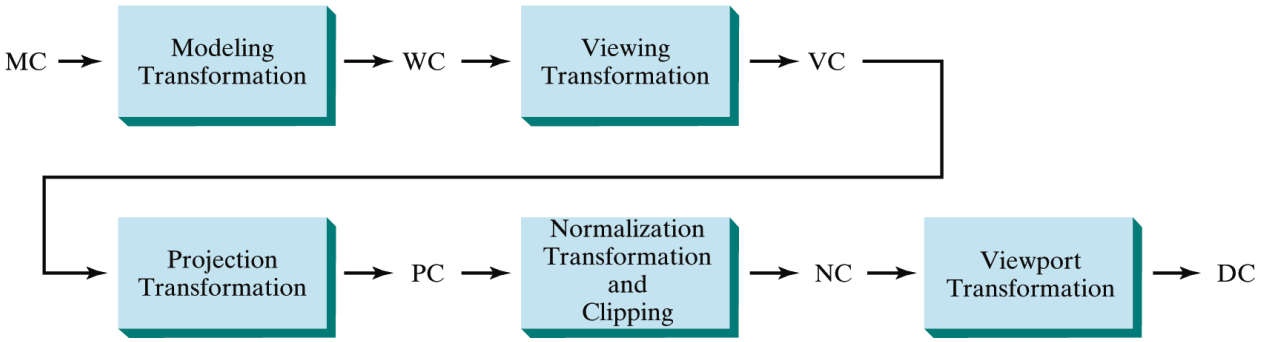
OpenGL Program Organization

- **main:**
  - find GL visual and create window
  - initialize GL states (e.g. viewing, color, lighting)
  - initialize display lists
  - loop
    - check for events (and process them)
    - if window event (window moved, exposed, etc.)
    - modify viewport, if needed
    - redraw
    - else if mouse or keyboard
    - do something, e.g., change states and redraw
- **redraw:**
  - clear screen (to background color)
  - change state(s), if needed
  - render some graphics
  - change more states
  - render some more graphics

glMatrixMode

- glMatrixMode
  - - specify which matrix is the current matrix
- C Specification
  - void glMatrixMode(GLenum *mode*)
- Parameters
  - *mode* Specifies which matrix stack is the target for subsequent matrix operations. Three values are accepted: GL_MODELVIEW, GL_PROJECTION, and GL_TEXTURE. The default value is GL_MODELVIEW.
- Description
  - glMatrixMode sets the current matrix mode. *mode* can assume one of three values: GL_MODELVIEW Applies subsequent matrix operations to the modelview matrix stack. GL_PROJECTION Applies subsequent matrix operations to the projection matrix stack.

General 3D Viewing Pipeline



- Modeling coordinates (MC)
- World coordinates (WC)
- Viewing coordinates (VC)
- Projection coordinates (PC)
- Normalized coordinates (NC)
- Device coordinates (DC)

OpenGL 3D Viewing Functions

- Viewing-transformation function
  - glMatrixMode(GL_MODELVIEW);
  - gluLookAt(x0,y0,z0,xref,yref,zref,vx,vy,vz);
  - Default: gluLookAt(0,0,0, 0,0,-1, 0,1,0);
  - OpenGL orthogonal-projection function
  - glMatrixMode(GL_PROJECTION);
  - gluOrtho(xwmin,xwmax, ywmin,ywmax, dnear,dfar);
  - Default: gluOrtho(-1,1, -1,1, -1,1);
  - Note that
    - dnear and dfar must be assigned positive values
    - znear=-dnear and zfar=-dfar
    - The near clipping plane is the view plane

Open GL program installation and execution steps

Open GL libraries path

File	Location
glut.h	C:\Program Files\Microsoft Visual Studio 11.0\VC\\include\GL
glut32.dll	• For windows 7 C:\WINDOWS\System32 • For windows 10 or above C:\WINDOWS\SystemWOW64
glut32.lib	C:\Program Files\Microsoft Visual Studio 11.0\VC\lib

Step 1: Create a Visual Studio 2017 Project

To create an empty console project in Visual Studio, do the following:

1. Create a new project (File ---> New ---> --->Project)
2. In the Project Types: pane, select Visual C++, Win32. Then select Win 32 Console Application in the Templates: pane. Name your project, select the location for the project and click OK.
3. Click the Application Settings tab on the left, check the Empty Project box and uncheck Security Development Lifecycle (SDL). Then click Finish button.

Step 2: Add Source Code

1. Select Project, Add New Item
2. In the Categories pane, select Visual C++, Code. Then select C++ File (.cpp) in the Templates: pane. Name your file, and then click Add.

Step 3: Compile and Run the project

a. Compile the Program

From the Visual Studio's menu Build option (Build ---> Build Solution)

b. Execute the program

From the Visual Studio's menu Debug option (Debug ---> Start Without Debugging)

Experiment 1:

Implement Brenham's line drawing algorithm for all types of slope

AIM:

To implement Bresenham's line drawing algorithm with different values of slopes

ALGORITHM:

Bresenham's Line-Drawing Algorithm.

Step 1 - Input the two end-points of line, storing the left end-point in (x0,y0).

Step 2 - Plot the point (x0,y0).

Step 3 - Calculate the constants dx, dy, 2dy, and (2dy – 2dx) and get the first value for the decision parameter as -

$p0=2dy-dx$

Step 4 - At each Xk along the line, starting at k = 0, perform the following test -

If $pk< 0$, the next point to plot is $(xk+1,yk)$ and $pk+1=pk+2dy$

Otherwise, $(xk,yk+1)$

$pk+1=pk+2dy-2dx$

Step 5 - Repeat step 4 (dx – 1) times.

For m > 1, find out whether you need to increment x while incrementing y each time.

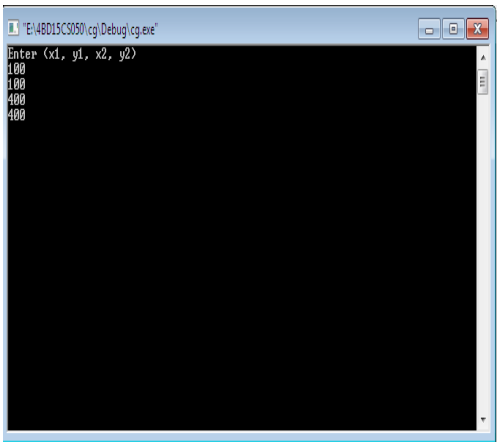
After solving, the equation for decision parameter Pk
will be very similar, just the x and y in the equation gets interchanged.

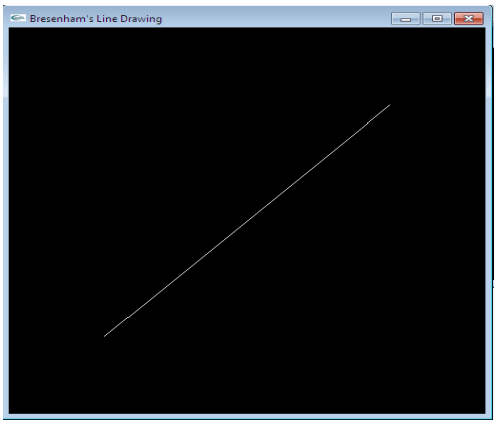
PROGRAM:

```
#include <GL/glut.h>
#include <stdio.h>
int x1, y1, x2, y2;
void myInit()
 {
glClear(GL_COLOR_BUFFER_BIT);
glClearColor(0.0, 0.0, 0.0, 1.0);
glMatrixMode(GL_PROJECTION);
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, 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);
}
}
}
void myDisplay()
 {
draw_line(x1, x2, y1, y2);
glFlush();
}
int main(int argc, char **argv) {
printf( "Enter (x1, y1, x2, y2)\n");
scanf("%d %d %d %d", &x1, &y1, &x2, &y2);
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutInitWindowSize(500, 500);
glutInitWindowPosition(0, 0);
glutCreateWindow("Bresenham's Line Drawing");
myInit();
glutDisplayFunc(myDisplay);
glutMainLoop();
return 0;
}
```

Sample Output:





Experiment 2: Create and rotate a triangle about the origin and a fixed point

AIM:

To Create and rotate a triangle about the origin and a fixed point.

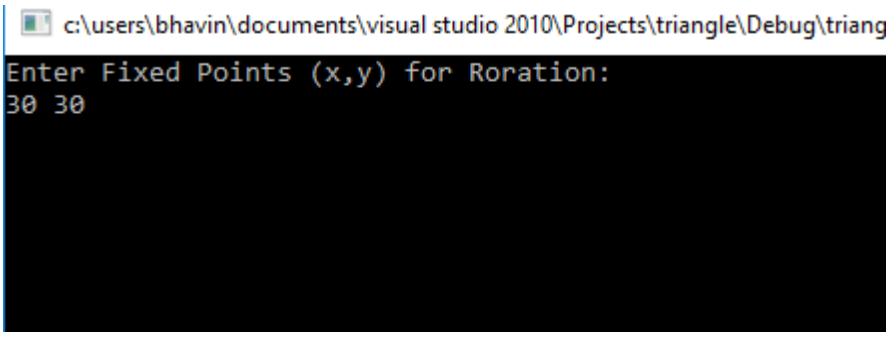
PROGRAM:

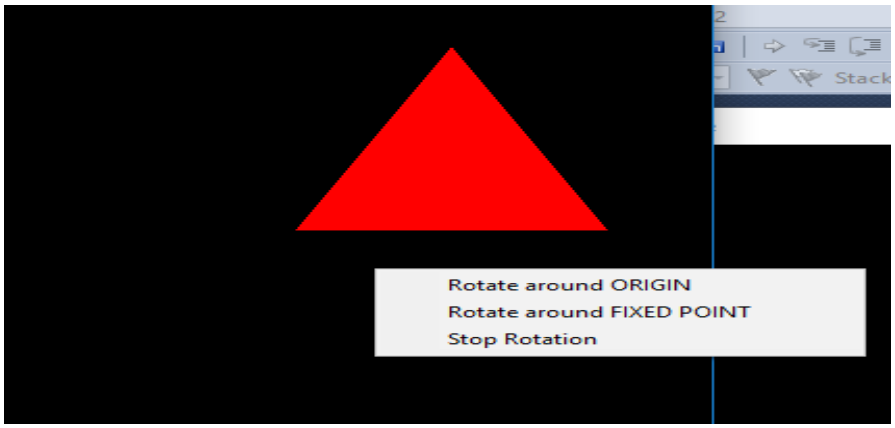
```
#include<GL/glut.h>
#include<stdio.h>
int x,y;
int where_to_rotate=0;     // don't rotate initially
float rotate_angle=0;      // initial angle
float translate_x=0,translate_y=0;  // initial translation

void draw_pixel(float x1, float y1)
{
   glPointSize(5);
   glBegin(GL_POINTS);
      glVertex2f(x1,y1);     // plot a single point
   glEnd();
}

void triangle(int x, int y)
{
   glColor3f(1,0,0);
   glBegin(GL_POLYGON);    // drawing a Triangle
      glVertex2f(x,y);
      glVertex2f(x+400,y+300);
      glVertex2f(x+300,y+0);
   glEnd();
}

void display()
{
   glClear(GL_COLOR_BUFFER_BIT);
   glLoadIdentity();

   glColor3f(1,1,1);            // mark origin point as white dot
   draw_pixel(0,0);            // plot origin - white colour

   if (where_to_rotate == 1) //Rotate Around origin
   {
      translate_x = 0;          // no translation for rotation around origin
      translate_y = 0;
      rotate_angle += 1;       // the amount of rotation angle
   }

   if (where_to_rotate == 2) //Rotate Around Fixed Point
   {
      translate_x = x;       // SET the translation to wherever the customer says
      translate_y = y;
      rotate_angle += 1;     // the amount of rotation angle
```

```
      glColor3f(0,0,1);      // mark the customer coordinate as blue dot
      draw_pixel(x,y);       // plot the customer coordinate - blue colour
   }

   glTranslatef(translate_x, translate_y, 0);    // ACTUAL translation +ve
   glRotatef(rotate_angle, 0, 0, 1);                  // rotate
   glTranslatef(-translate_x, -translate_y, 0);  // ACTUAL translation -ve

   triangle(translate_x,translate_y);               // what to rotate? - TRIANGLE

   glutPostRedisplay();              // call display function again and again
   glutSwapBuffers();               // show the output
}

void init()
{
   glClearColor(0,0,0,1); //setting to black
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   gluOrtho2D(-800, 800, -800, 800);
   glMatrixMode(GL_MODELVIEW);
}

void rotateMenu (int option)
{
   if(option==1)
      where_to_rotate=1;     // rotate around origin

   if(option==2)
      where_to_rotate=2;     // rotate around customer's coordinates

   if(option==3)
      where_to_rotate=3;     // stop rotation
}

int main(int argc, char **argv)
{
   printf( "Enter Fixed Points (x,y) for Rotation: \n");
   scanf("%d %d", &x, &y);              // getting the user's coordinates to rotate

   glutInit(&argc, argv);                  // initialize the graphics system
   glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB);  // SINGLE also works
   glutInitWindowSize(800, 800);    // 800 by 800 size..you can change it
   glutInitWindowPosition(0, 0);      // where do you wanna see your window
   glutCreateWindow("Create and Rotate Triangle");     // title

   init();                                 // initialize the canvas

   glutDisplayFunc(display);          // call display function

   glutCreateMenu(rotateMenu);     // menu items
   glutAddMenuEntry("Rotate around ORIGIN",1);
```

```
    glutAddMenuEntry("Rotate around FIXED POINT",2);
    glutAddMenuEntry("Stop Rotation",3);
    glutAttachMenu(GLUT_RIGHT_BUTTON);

    glutMainLoop();                    // run forever
}
```

Sample Output:





Experiment 3:

Draw a color cube and spin it using OpenGL transformation matrices

AIM:

To Draw a color cube and spin it using OpenGL transformation matrices

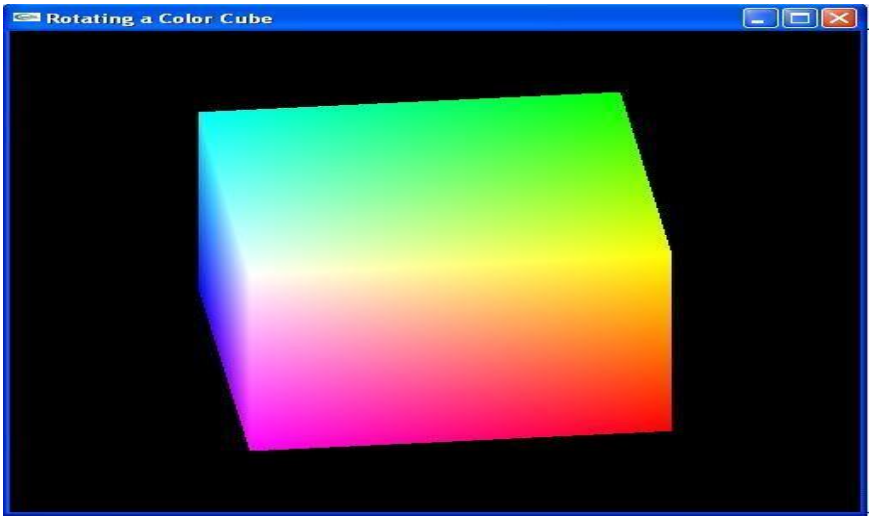
PROGRAM:

```
#include<stdlib.h>
#include<GL/glut.h>
GLfloat vertices[]= {-1, -1, -1,
                      1, -1, -1,
                      1, 1, -1,
                      -1, 1, -1,
                      -1, -1, 1,
                      1, -1, 1,
                      1, 1, 1,
                      -1, 1, 1
                    };

GLfloat colors[]= {0, 0, 0,
                   1, 0, 0,
                   1, 1, 0,
                   0, 1, 0,
                   0, 0, 1,
                   1, 0, 1,
                   1, 1, 1,
                   0, 1, 1
                  };

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};
static GLint axis=2;

void display(void)
{
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glLoadIdentity();

  glRotatef(theta[0], 1, 0, 0);
  glRotatef(theta[1], 0, 1, 0);
  glRotatef(theta[2], 0, 0, 1);

  glDrawElements(GL_QUADS,24,GL_UNSIGNED_BYTE,cubeIndices);
```

```
    glutSwapBuffers();
}

void spinCube()
{
    theta[axis] += 2;

    if(theta[axis] > 360)
        theta[axis] -= 360;

    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, 2, -2*(GLfloat)h/(GLfloat)w, 2*(GLfloat)h/(GLfloat)w, -10, 10);
    else
        glOrtho(-2*(GLfloat)w/(GLfloat)h, 2*(GLfloat)w/(GLfloat)h,  -2, 2, -10, 10);
    glMatrixMode(GL_MODELVIEW);
}

int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|GLUT_DEPTH);
    glutInitWindowSize(500, 500);
    glutCreateWindow("Spin a color cube");

    glutReshapeFunc(myReshape);
    glutDisplayFunc(display);
    glutIdleFunc(spinCube);
    glutMouseFunc(mouse);

    glEnable(GL_DEPTH_TEST);

    glEnableClientState(GL_COLOR_ARRAY);
    glEnableClientState(GL_VERTEX_ARRAY);
```

```
    glVertexPointer(3, GL_FLOAT, 0, vertices);
    glColorPointer(3, GL_FLOAT, 0, colors);

    glColor3f(1, 1, 1);

    glutMainLoop();
}
```

Sample Output:



Experiment 4:

Draw a color cube and allow the user to move the camera suitably to experiment with perspective viewing.

AIM:

To Draw a color cube and allow the user to move the camera suitably to experiment with perspective viewing.

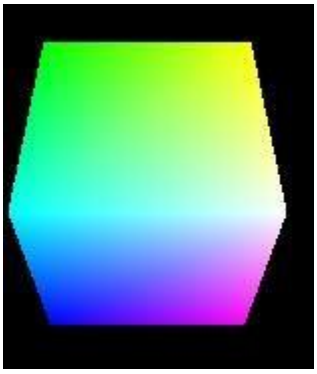
PROGRAM:

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

GLfloat vertices[][3] = {   {-1,-1,-1},
                            {1,-1,-1},
                            {1,1,-1},
                            {-1,1,-1},
                            {-1,-1,1},
                            {1,-1,1},
                            {1,1,1},
                            {-1,1,1}
                        };

GLfloat colors[][3] = {     {1,0,0},
                            {1,1,0},
                            {0,1,0},
                            {0,0,1},
                            {1,0,1},
                            {1,1,1},
                            {0,1,1},
                            {0.5,0.5,0.5}
                        };

GLfloat theta[] = {0.0,0.0,0.0};
GLint axis = 2;
GLdouble viewer[]= {0.0, 0.0, 5.0}; // initial viewer location //

void polygon(int a, int b, int c, int d)
{
   glBegin(GL_POLYGON);
      glColor3fv(colors[a]);
      glVertex3fv(vertices[a]);
      glColor3fv(colors[b]);
      glVertex3fv(vertices[b]);
      glColor3fv(colors[c]);
      glVertex3fv(vertices[c]);
      glColor3fv(colors[d]);
      glVertex3fv(vertices[d]);
   glEnd();
}

void colorcube(void)
```

```
{
   polygon(0,3,2,1);
   polygon(0,4,7,3);
   polygon(5,4,0,1);
   polygon(2,3,7,6);
   polygon(1,2,6,5);
   polygon(4,5,6,7);
}

void display(void)
{
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
   glLoadIdentity();
   gluLookAt(viewer[0],viewer[1],viewer[2], 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
   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);
   colorcube();
   glFlush();
   glutSwapBuffers();
}

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;
   theta[axis] += 2.0;
   if( theta[axis] > 360.0 ) theta[axis] -= 360.0;
   display();
}

void keys(unsigned char key, int x, int y)
{
   if(key == 'x') viewer[0]-= 1.0;
   if(key == 'X') viewer[0]+= 1.0;
   if(key == 'y') viewer[1]-= 1.0;
   if(key == 'Y') viewer[1]+= 1.0;
   if(key == 'z') viewer[2]-= 1.0;
   if(key == 'Z') viewer[2]+= 1.0;
   display();
}

void myReshape(int w, int h)
{
   glViewport(0, 0, w, h);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   if(w<=h)
      glFrustum(-2.0, 2.0, -2.0 * (GLfloat) h/ (GLfloat) w, 2.0* (GLfloat) h / (GLfloat) w,2.0, 20.0);
   else
      glFrustum(-2.0, 2.0, -2.0 * (GLfloat) w/ (GLfloat) h, 2.0* (GLfloat) w / (GLfloat) h, 2.0, 20.0);
```

```
    glMatrixMode(GL_MODELVIEW);
}

int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(500, 500);
    glutCreateWindow("Colorcube Viewer");

    glutReshapeFunc(myReshape);
    glutDisplayFunc(display);
    glutMouseFunc(mouse);
    glutKeyboardFunc(keys);

    glEnable(GL_DEPTH_TEST);
    glutMainLoop();
}
```

Sample Output:



Mouse Left and Right click to rotate

Experiment 5:

Clip a lines using Cohen-Sutherland algorithm.

AIM:

To Clip a lines using Cohen-Sutherland algorithm.

ALGORITHM:

Cohen-sutherland line clipping Algorithm

To perform the trivial acceptance and rejection tests, we extend the edges of the window to divide the plane of the window into the nine regions. Each end point of the line segment is then assigned the code of the region in which it lies.

1. Given a line segment with endpoint $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$
2. Compute the 4-bit codes for each endpoint.

   If both codes are **0000**,(bitwise OR of the codes yields 0000) line lies completely **inside** the window: pass the endpoints to the draw routine.

   If both codes have a 1 in the same bit position (bitwise AND of the codes is **not** 0000), the line lies **outside** the window. It can be trivially rejected.

3. If a line cannot be trivially accepted or rejected, at least one of the two endpoints must lie outside the window and the line segment crosses a window edge. This line must be **clipped** at the window edge before being passed to the drawing routine.
4. Examine one of the endpoints, say $P_1 = (x_1, y_1)$. Read P_1 's 4-bit code in order: **Left**-to-**Right**, **Bottom**-to-**Top**.
5. When a set bit (1) is found, compute the **intersection** **I** of the corresponding window edge with the line from P_1 to P_2. Replace P_1 with **I** and repeat the algorithm.

1001	1000	1010
0001	0000	0010
0101	0100	0110

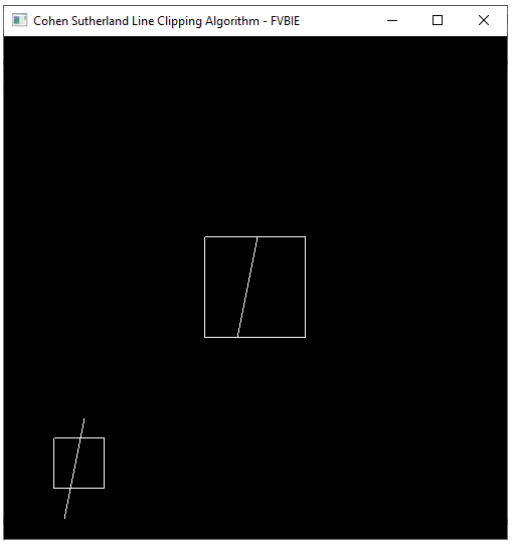
PROGRAM:

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

double xmin = 50, ymin = 50, xmax = 100, ymax = 100;          //window coordinates
double xvmin = 200, yvmin = 200, xvmax = 300, yvmax = 300;  //viewport coordinates

const int LEFT = 1;               // code words for LEFT, RIGHT, BOTTOM &TOP.
const int RIGHT = 2;
const int BOTTOM = 4;
const int TOP = 8;

int ComputeOutCode (double x, double y)
{
   int code = 0;
   if (y > ymax)                  //above the clip window
      code |= TOP;
   else if (y < ymin)             //below the clip window
      code |= BOTTOM;
   if (x > xmax)                  //to the right of clip window
      code |= RIGHT;
   else if (x < xmin)           //to the left of clip window
      code |= LEFT;
   return code;
}

void CohenSutherland(double x0, double y0,double x1, double y1)
{
   int outcode0, outcode1, outcodeOut;
   bool accept = false, done = false;
   outcode0 = ComputeOutCode (x0, y0);     //calculate the region of 1st point
   outcode1 = ComputeOutCode (x1, y1);      //calculate the region of 2nd point

   do
   {
      if (!(outcode0 | outcode1))
      {
         accept = true;
         done = true;
      }
      else if (outcode0 & outcode1)
         done = true;
      else
      {
         double x, y;
         double m = (y1 - y0)/(x1 - x0);
         outcodeOut = outcode0? outcode0: outcode1;

         if (outcodeOut & TOP)
         {
            x = x0 + (1/m) * (ymax - y0);
            y = ymax;
         }
         else if (outcodeOut & BOTTOM)
         {
            x = x0 + (1/m) * (ymin - y0);
            y = ymin;
         }
         else if (outcodeOut & RIGHT)
         {
```

```
                y = y0 +  m * (xmax - x0);
                x = xmax;
            }
            else
            {
                y = y0 + m * (xmin - x0);
                x = xmin;
            }
            /* Intersection calculations over */

            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;

        glBegin(GL_LINE_LOOP);
            glVertex2f(xvmin, yvmin);
            glVertex2f(xvmax, yvmin);
            glVertex2f(xvmax, yvmax);
            glVertex2f(xvmin, yvmax);
        glEnd();

        glBegin(GL_LINES);
            glVertex2d (vx0, vy0);
            glVertex2d (vx1, vy1);
        glEnd();
    }
}

void display()
{
    double x0 = 60, y0 = 20, x1 = 80, y1 = 120;
    glClear(GL_COLOR_BUFFER_BIT);

    glColor3f(1, 1, 1);//white

    glBegin(GL_LINES);
        glVertex2d (x0, y0);
        glVertex2d (x1, y1);
    glEnd();
```

```
    glBegin(GL_LINE_LOOP);
        glVertex2f(xmin, ymin);
        glVertex2f(xmax, ymin);
        glVertex2f(xmax, ymax);
        glVertex2f(xmin, ymax);
    glEnd();

    CohenSutherland(x0, y0, x1, y1);

    glFlush();
}

void myinit()
{
    glClearColor(0, 0, 0, 1);//black
    gluOrtho2D(0, 500, 0, 500);
}

int main(int argc, char **argv)
{
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(0, 0);
    glutCreateWindow("Cohen Sutherland Line Clipping Algorithm");

    myinit();

    glutDisplayFunc(display);

    glutMainLoop();
}
```

Sample Output:



Experiment 6:

To draw a simple shaded scene consisting of a tea pot on a table. Define suitably the position and properties of the light source along with the properties of the surfaces of the solid object used in the scene.

AIM:

To draw a simple shaded scene consisting of a tea pot on a table. Define suitably the position and properties of the light source along with the properties of the surfaces of the solid object used in the scene.

PROGRAM

```
#include<GL/glut.h>
void teapot(GLfloat x,GLfloat y,GLfloat z)
{
   glPushMatrix();
   glTranslatef(x, y, z);
   glutSolidTeapot(0.1);
   glPopMatrix();
}
void tableTop(GLfloat x, GLfloat y, GLfloat z)
{
   glPushMatrix();
   glTranslatef(x, y, z);
   glScalef(0.6, 0.02, 0.5);
   glutSolidCube(1);
   glPopMatrix();
}
void tableLeg(GLfloat x, GLfloat y, GLfloat z)
{
   glPushMatrix();
   glTranslatef(x, y, z);
   glScalef(0.02, 0.3, 0.02);
   glutSolidCube(1);
   glPopMatrix();
}
void wall(GLfloat x, GLfloat y, GLfloat z)
{
   glPushMatrix();
   glTranslatef(x, y, z);
   glScalef(1, 1, 0.02);
   glutSolidCube(1);
   glPopMatrix();
}
void light()
{
   GLfloat mat_ambient[] = {1, 1, 1, 1};
   GLfloat mat_diffuse[] = {0.5, 0.5, 0.5, 1};
   GLfloat mat_specular[] = {1, 1, 1, 1};
   GLfloat mat_shininess[] = {50.0f};

   glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
```

```
    glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
    glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
    glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);

    GLfloat light_position[]  =  {2, 6, 3, 1};
    GLfloat light_intensity[]  =  {0.7, 0.7, 0.7, 1};
    glLightfv(GL_LIGHT0, GL_POSITION, light_position);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, light_intensity);
}
void display()
{
    GLfloat teapotP = -0.07, tabletopP = -0.15, tablelegP = 0.2, wallP = 0.5;
    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();

    gluLookAt(-2, 2, 5, 0, 0, 0,  0, 1, 0);

    light(); //Adding light source to your project

    teapot(0, teapotP, 0); //Create teapot

    tableTop(0, tabletopP, 0); //Create tableâ€™s top

    tableLeg(tablelegP, -0.3, tablelegP); //Create 1st leg
    tableLeg(-tablelegP, -0.3, tablelegP); //Create 2nd leg
    tableLeg(-tablelegP, -0.3, -tablelegP); //Create 3rd leg
    tableLeg(tablelegP, -0.3, -tablelegP); //Create 4th leg

    wall(0, 0, -wallP); //Create 1st wall
    glRotatef(90, 1, 0, 0);

    wall(0, 0, wallP); //Create 2nd wall
    glRotatef(90, 0, 1, 0);

    wall(0, 0, wallP); //Create 3rd wall
    glFlush();
}
void myinit()
{
    glClearColor(0, 0, 0, 1);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-1, 1, -1, 1, -1, 10);
    glMatrixMode(GL_MODELVIEW);
}
int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB|GLUT_DEPTH);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(0, 0);
    glutCreateWindow("Teapot on a table");
```

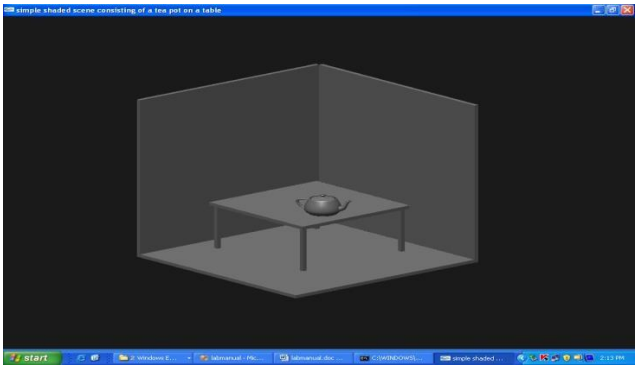
```
    myinit();

    glutDisplayFunc(display);

    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);

    glShadeModel(GL_SMOOTH);

    glEnable(GL_NORMALIZE);
    glEnable(GL_DEPTH_TEST);

    glutMainLoop();
 }
```

OR

```
#include<gl/glut.h>
void obj(double tx,double ty,double tz,double sx,double sy,double sz)
{
        glRotated(50,0,1,0);
        glRotated(10,-1,0,0);
        glRotated(11.7,0,0,-1);

        glTranslated(tx,ty,tz);
        glScaled(sx,sy,sz);
        glutSolidCube(1);
        glLoadIdentity();
}
void display()
{
        glViewport(0,0,700,700);
        glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);

        obj(0,0,0.5,1,1,0.04); // three walls
        obj(0,-0.5,0,1,0.04,1);
        obj(-0.5,0,0,0.04,1,1);

        obj(0,-0.3,0,0.02,0.2,0.02); // four table legs
        obj(0,-0.3,-0.4,0.02,0.2,0.02);
        obj(0.4,-0.3,0,0.02,0.2,0.02);
        obj(0.4,-0.3,-0.4,0.02,0.2,0.02);

        obj(0.2,-0.18,-0.2,0.6,0.02,0.6); // table top

        glRotated(50,0,1,0);
        glRotated(10,-1,0,0);
        glRotated(11.7,0,0,-1);

        glTranslated(0.3,-0.1,-0.3);
```

```
        glutSolidTeapot(0.09);
        glFlush();
        glLoadIdentity();
}
void main()
{
        float ambient[]={1,1,1,1};
        float light_pos[]={27,80,2,3};
        glutInitWindowSize(700,700);
        glutCreateWindow("scene");
        glutDisplayFunc(display);
        glEnable(GL_LIGHTING);
        glEnable(GL_LIGHT0);
        glMaterialfv(GL_FRONT,GL_AMBIENT,ambient);
        glLightfv(GL_LIGHT0,GL_POSITION,light_pos);
        glEnable(GL_DEPTH_TEST);
        glutMainLoop();
}
```

Output:



Experiment 7:

Design, develop and implement recursively subdivide a tetrahedron to form 3D sierpinski gasket. The number of recursive steps is to be specified by the user.

AIM:

To Design, develop and implement recursively subdivide a tetrahedron to form 3D sierpinski gasket. The number of recursive steps is to be specified by the user..

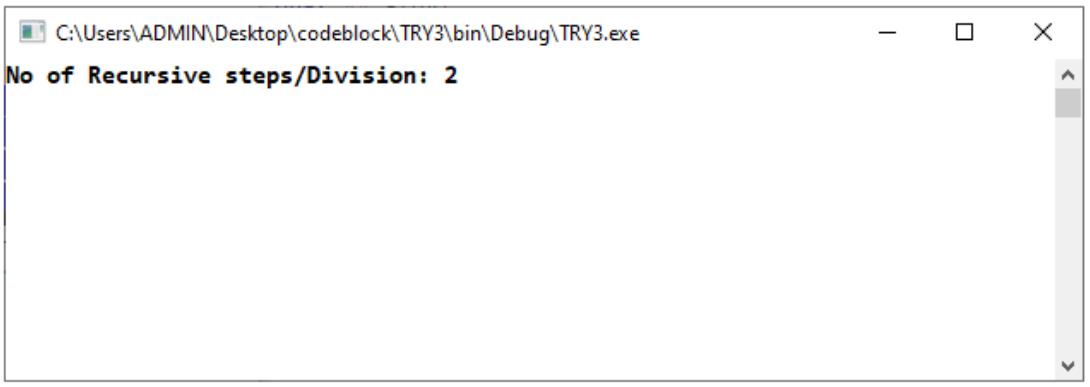
PROGRAM

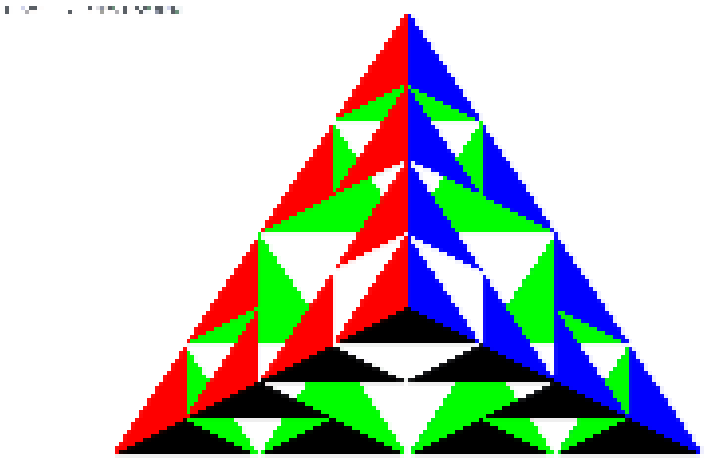
```
#include<stdlib.h>
#include<stdio.h>
#include<GL/glut.h>

typedef float point[3];
point v[]= {{0, 0, 1}, {0, 1, 0}, {-1, -0.5, 0}, {1, -0.5, 0}};
int n;

void triangle(point a,point b,point c)
{
   glBegin(GL_POLYGON);
   glVertex3fv(a);
   glVertex3fv(b);
   glVertex3fv(c);
   glEnd();
}

void divide_triangle(point a,point b,point c,int n)
{
   point v1,v2,v3;
   int j;
   if(n>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] = (c[j]+b[j])/2;

      divide_triangle(a,v1,v2,n-1);
      glFlush();
      divide_triangle(c,v2,v3,n-1);
      glFlush();
      divide_triangle(b,v3,v1,n-1);
      glFlush();
   }
   else(triangle(a,b,c));
}
```

```
void tetrahedron(int n)
{
   glColor3f(1, 0, 0);
   divide_triangle(v[0], v[1], v[2], n);

   glColor3f(0,  1,  0);
   divide_triangle(v[3], v[2], v[1], n);

   glColor3f(0,  0,  1);
   divide_triangle(v[0], v[3], v[1], n);

   glColor3f(0,  0,  0);
   divide_triangle(v[0], v[2], v[3], n);

}
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, 2, -2*(GLfloat)h/(GLfloat)w, 2*(GLfloat)h/(GLfloat)w, -10, 10);
   else
      glOrtho(-2*(GLfloat)w/(GLfloat)h, 2*(GLfloat)w/(GLfloat)h, -2, 2, -10, 10);

   glMatrixMode(GL_MODELVIEW);
   glutPostRedisplay();
}
int main(int argc,char ** argv)
{
   printf("No of Recursive steps/Division: ");
   scanf("%d",&n);
   glutInit(&argc,argv);
   glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB|GLUT_DEPTH);
   glutCreateWindow(" 3D Sierpinski gasket");

   glutReshapeFunc(myReshape);

   glutDisplayFunc(display);

   glEnable(GL_DEPTH_TEST);

   glClearColor(1, 1, 1, 0);
```

```
    glutMainLoop();

    return 0;
}
```

Sample Output:





Experiment 8:

Develop a menu driven program to animate a flag using Bezier Curve algorithm.

AIM:

To develop a menu driven program to animate a flag using Bezier Curve algorithm

PROGRAM

```
#include<GL/glut.h>
#include<stdio.h>
#include<math.h>
#define PI 3.1416

static int win,val=0,CMenu;

void CreateMenu(void);
void Menu(int value);

struct wcPt3D
{
GLfloat x, y, z;
};

GLsizei winWidth = 600, winHeight = 600;
GLfloat xwcMin = 0.0, xwcMax = 130.0;
GLfloat ywcMin = 0.0, ywcMax = 130.0;
void bino(GLint n, GLint *C)
{
GLint k, j;
for(k=0;k<=n;k++)
{
C[k]=1;
for(j=n;j>=k+1; j--)
C[k]*=j;
for(j=n-k;j>=2;j--)
C[k]/=j;
}
}
void computeBezPt(GLfloat u,struct wcPt3D *bezPt, GLint nCtrlPts,struct wcPt3D *ctrlPts,
GLint *C)
{
GLint k, n=nCtrlPts-1;
GLfloat bezBlendFcn;
bezPt ->x =bezPt ->y = bezPt->z=0.0;
for(k=0; k< nCtrlPts; k++)
{
bezBlendFcn = C[k] * pow(u, k) * pow( 1-u, n-k);
bezPt ->x += ctrlPts[k].x * bezBlendFcn;
bezPt ->y += ctrlPts[k].y * bezBlendFcn;
```

```
bezPt ->z += ctrlPts[k].z * bezBlendFcn;
}
}
void bezier(struct wcPt3D *ctrlPts, GLint nCtrlPts, GLint nBezCurvePts)
{
struct wcPt3D bezCurvePt;
GLfloat u;
GLint *C, k;
C= new GLint[nCtrlPts];
bino(nCtrlPts-1, C);
glBegin(GL_LINE_STRIP);
for(k=0; k<=nBezCurvePts; k++)
{
u=GLfloat(k)/GLfloat(nBezCurvePts);
computeBezPt(u, &bezCurvePt, nCtrlPts, ctrlPts, C);
glVertex2f(bezCurvePt.x, bezCurvePt.y);
}
glEnd();
delete[]C;
}
void displayFcn()
{
GLint nCtrlPts = 4, nBezCurvePts =20;
static float theta = 0;
struct wcPt3D ctrlPts[4] = {{20, 100, 0},{30, 110, 0},{50, 90, 0},{60, 100, 0}};
ctrlPts[1].x +=10*sin(theta * PI/180.0);
ctrlPts[1].y +=5*sin(theta * PI/180.0);
ctrlPts[2].x -= 10*sin((theta+30) * PI/180.0);
ctrlPts[2].y -= 10*sin((theta+30) * PI/180.0);
ctrlPts[3].x-= 4*sin((theta) * PI/180.0);
ctrlPts[3].y += sin((theta-30) * PI/180.0);
theta+=0.1;
glClear(GL_COLOR_BUFFER_BIT);
glColor3f(1.0, 1.0, 1.0);
glPointSize(5);
//Indian Flag
if(val==1){
glPushMatrix();
glLineWidth(5);
glColor3f(1.0,0.5,0); //Indian flag: Orange color code
for(int i=0;i<8;i++)
{
glTranslatef(0, -0.8, 0);
bezier(ctrlPts, nCtrlPts, nBezCurvePts);
}
glColor3f(1,1,1); //Indian flag: white color code
for(int i=0;i<8;i++)
{
glTranslatef(0, -0.8, 0);
bezier(ctrlPts, nCtrlPts, nBezCurvePts);
```

```
}
glColor3f(0,1.0,0); //Indian flag: green color code
for(int i=0;i<8;i++)
{
glTranslatef(0, -0.8, 0);
bezier(ctrlPts, nCtrlPts, nBezCurvePts);
}
glPopMatrix();
glColor3f(0.7, 0.5,0.3);
glLineWidth(5);
glBegin(GL_LINES);
glVertex2f(20,100);
glVertex2f(20,40);
glEnd();
glFlush();
}
//Karnataka Flag
if(val==2){
glPushMatrix();
glLineWidth(5);
glColor3f(1.0, 1.0, 0.0); //Karnataka flag: Yellow color code
for(int i=0;i<12;i++)
{
glTranslatef(0, -0.8, 0);
bezier(ctrlPts, nCtrlPts, nBezCurvePts);
}
glColor3f(1, 0.0, 0.0); //Karnataka flag: Red color code
for(int i=0;i<12;i++)
{
glTranslatef(0, -0.8, 0);
bezier(ctrlPts, nCtrlPts, nBezCurvePts);
}
glPopMatrix();
glColor3f(0.7, 0.5,0.3);
glLineWidth(5);
glBegin(GL_LINES);
glVertex2f(20,100);
glVertex2f(20,40);
glEnd();
glFlush();
}
glutPostRedisplay();
glutSwapBuffers();
}

void winReshapeFun(GLint newWidth, GLint newHeight)
{
glViewport(0, 0, newWidth, newHeight);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
```

```
gluOrtho2D(xwcMin, xwcMax, ywcMin, ywcMax);
glClear(GL_COLOR_BUFFER_BIT);
}

void CreateMenu(void)
{
	CMenu= glutCreateMenu(Menu);//Creaate Menu Option

	glutAddMenuEntry("Indian Flag",1);
	glutAddMenuEntry("Karnataka Flag",2);
	glutAddMenuEntry("Exit",0);

	glutAttachMenu(GLUT_RIGHT_BUTTON);

}

void Menu(int value)
{
	if(value==0)
	{
	glutDestroyWindow(win);
	exit(0);
	}
	else {
	val=value;
	}
}

int main(int argc, char **argv)
{
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
glutInitWindowPosition(50, 50);
glutInitWindowSize(winWidth, winHeight);
glutCreateWindow("Prg. 8 Bezier Curve");
CreateMenu();
glutDisplayFunc(displayFcn);
glutReshapeFunc(winReshapeFun);
glutMainLoop();
}
```

Sample Output:

Experiment 9:

Develop a menu driven program to fill the polygon using scan line algorithm.

AIM:

To Develop a menu driven program to fill the polygon using scan line algorithm.

ALGORITHM:

Step 1 − Find out the Ymin and Ymax from the given polygon.

Step 2 − ScanLine intersects with each edge of the polygon from Ymin to Ymax. Name each intersection point of the polygon. As per the figure shown above, they are named as p0, p1, p2, p3.

Step 3 − Sort the intersection point in the increasing order of X coordinate i.e. (p0, p1), (p1, p2), and (p2, p3).

Step 4 − Fill all those pair of coordinates that are inside polygons and ignore the alternate pairs.

PROGRAM

```
#include <stdlib.h>
#include  <stdio.h>
#include <glut.h>
float x1,x2,x3,x4,y1,y2,y3,y4; int fillFlag=0;
void edgedetect(float x1,float y1,float x2,float y2,int *le,int *re)
{
        float mx,x,temp; int i;
if((y2-y1)<0)
{
        temp=y1;y1=y2;y2=temp;
        temp=x1;x1=x2;x2=temp;  }
if((y2-y1)!=0)
        mx=(x2-x1)/(y2-y1);
else  mx=x2-x1;
x=x1;
for(i=y1;i<=y2;i++)
{
        if(x<(float)le[i])
                le[i]=(int)x;
if(x>(float)re[i])
        re[i]=(int)x;
x+=mx;
}
}
void draw_pixel(int x,int y)
```

```
{  glColor3f(1.0,1.0,0.0);
glBegin(GL_POINTS);
glVertex2i(x,y);
glEnd();
}
void scanfill(float x1,float y1,float x2,float y2,float x3,float y3,float x4,float y4)
{
        int le[500],re[500];
int i,y;
for(i=0;i<500;i++)
{
        le[i]=500;
re[i]=0;
}
edgedetect(x1,y1,x2,y2,le,re);
edgedetect(x2,y2,x3,y3,le,re);
edgedetect(x3,y3,x4,y4,le,re);
edgedetect(x4,y4,x1,y1,le,re);
for(y=0;y<500;y++)
{
        for(i=(int)le[y];i<(int)re[y];i++)
                draw_pixel(i,y);
}
}
void display()
{
        x1=200.0;y1=200.0;x2=100.0;y2=300.0;x3=200.0;y3=400.0;x4=300.0;y4=300.0;
        glClear(GL_COLOR_BUFFER_BIT);
        glColor3f(0.0, 0.0, 1.0);
        glBegin(GL_LINE_LOOP);
        glVertex2f(x1,y1);
        glVertex2f(x2,y2);
        glVertex2f(x3,y3);
        glVertex2f(x4,y4);
        glEnd();
        if(fillFlag==1)
                scanfill(x1,y1,x2,y2,x3,y3,x4,y4);
        glFlush();
}

void init()
{
        glClearColor(0.0,0.0,0.0,1.0);
glColor3f(1.0,0.0,0.0);
glPointSize(1.0);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0.0,499.0,0.0,499.0);
}
void fillMenu(int option)
```

```
{
        if(option==1)
fillFlag=1;
        if(option==2)
fillFlag=2;
        display();
}
void main(int argc, char* argv[])
{ glutInit(&argc,argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutInitWindowSize(500,500);
glutInitWindowPosition(0,0);
glutCreateWindow("Filling a Polygon using Scan-line Algorithm");
init();
glutDisplayFunc(display);
glutCreateMenu(fillMenu);
glutAddMenuEntry("Fill Polygon",1);
glutAddMenuEntry("Empty Polygon",2);
glutAttachMenu(GLUT_RIGHT_BUTTON);
glutMainLoop();
}
```

Sample Output: