RAJIV GANDHI INSTITUTE OF TECHNOLOGY,BENGALURE-32
Department of Computer Science & Engineering

COMPUTER GRAPHICS LABORATORY
(17CSL68)

Prepared by:
Ms. Rajani kodagali
Assistant Professor Dept.
of CSE

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-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.

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 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

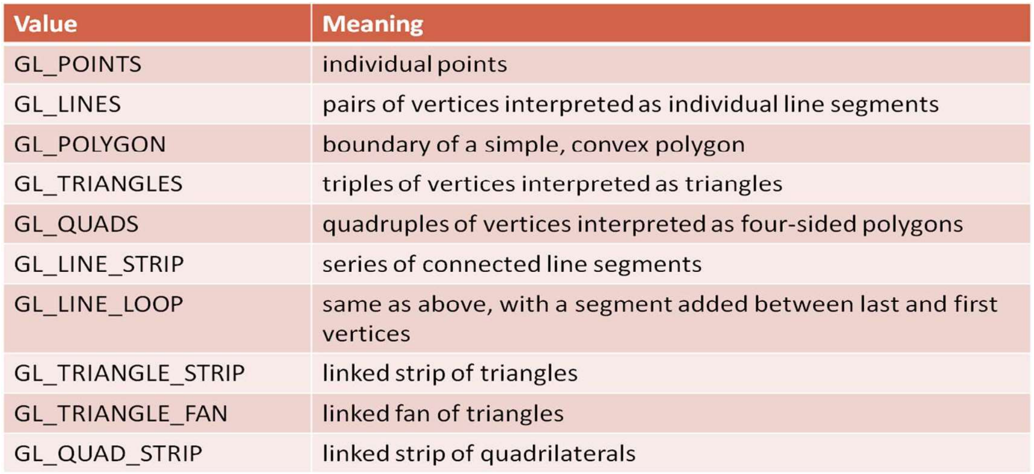
Event Loop

- 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

- **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.

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 execution steps

Step 1: Create a Visual Studio 2005 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, and check the Empty Project box. 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)

Sample programs

1. Program to create a simple primitive in OpenGL

```
#include<GL/glut.h>
void mydisplay()
{
        glClear(GL_COLOR_BUFFER_BIT);
        glBegin(GL_POINTS);
                glVertex2f(-0.5,-0.5);
                glVertex2f(0.5,0.5);
                glVertex2f(-0.5,0.5);
        glEnd();
        glFlush();
}
int main(int argc, char **argv)
{
        glutInit(&argc, argv);
        glutCreateWindow("Simple");
        glutDisplayFunc(mydisplay);
        glutMainLoop();
}
```

2. Program to create keyboard interface & window sizing

```
#include<GL/glut.h>
void display()
{ /* Called when OpenGL needs to update the display*/
        glClearColor(1.0,1.0,1.0,0.0);
        glClear(GL_COLOR_BUFFER_BIT);
        glFlush();
}
void keyboard(unsigned char key, int x, int y)
{ /* called when a key is pressed */
        if(key==27) exit(0);          /* 27 is the escape */
}
int main(int argc, char **argv)
{
        glutInit(&argc, argv);    /* Initialize OpenGL*/
        glutInitWindowSize(500,500); /*Set Window size*/
        glutInitWindowPosition(10,10); /*Set Window Position*/
        glutCreateWindow("Hai"); /* Create the window*/
        glutDisplayFunc(display);
        glutKeyboardFunc(keyboard);
        glutMainLoop();
}
```

3. Program to rotate a cube

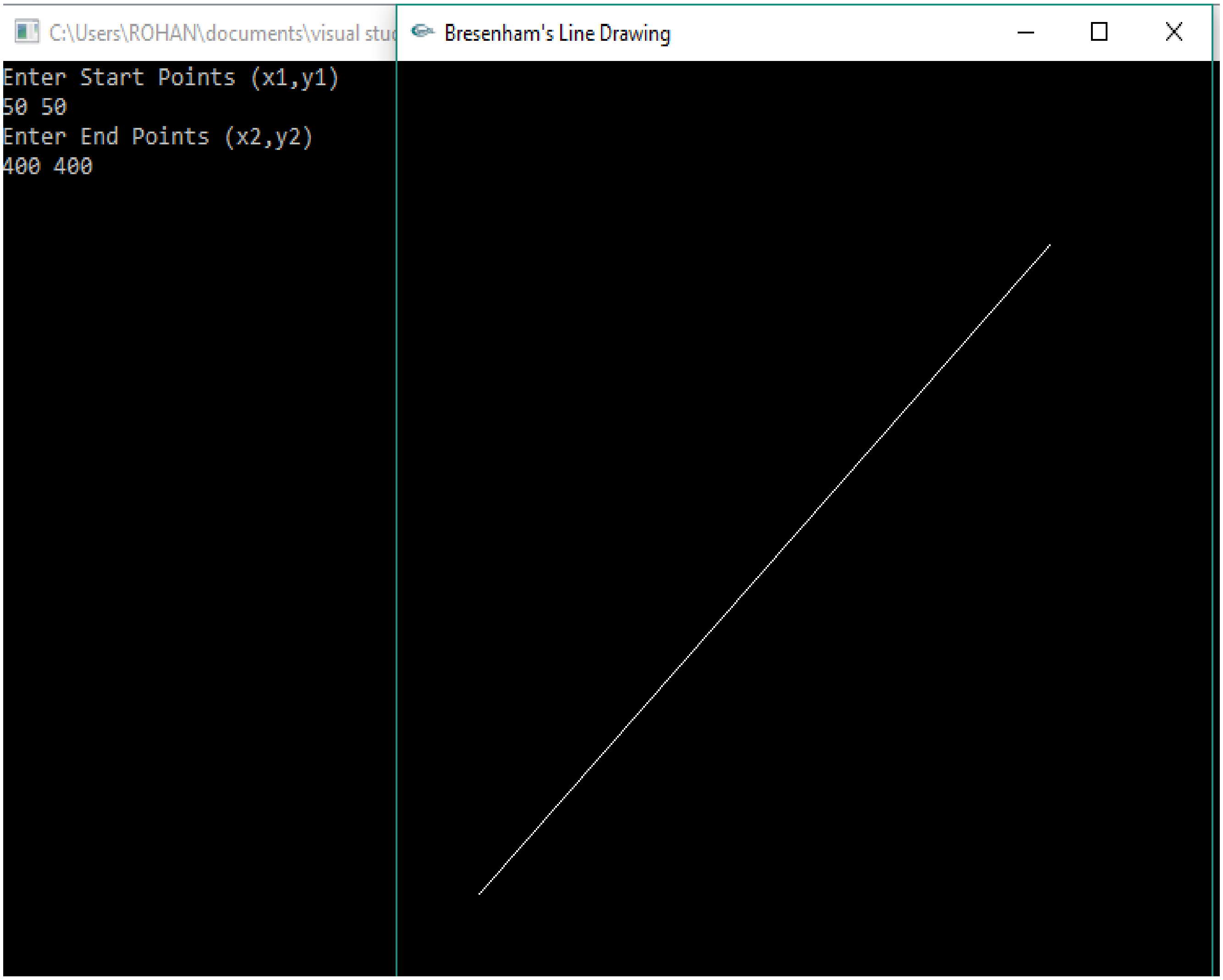
```
#include<GL/glut.h>
GLfloat angle=0.0;
void spin(void)
{
        Angle+=1.0;
        glutPostRedisplay()
}

void display(void)
{
        glClear(GL_COLOR_BUFFER_BIT);
        glLoadIdentity();
        gluLookAt(0.0,0.0,0.5,0.0,0.0,0.0,0.0,1.0,0.0);
        glRotatef(angle,1,0,0);
        glRotatef(angle,0,1,0);
        glRotatef(angle,0,0,1);
        glutWireCube(2.0);
}
void reshape(int width, int height)
{
        glViewport(0,0,(GLsizei)width, (GLsizei)height);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluPerspective(60, (GLfloat)width / (GLfloat)height,1.0,100);
        glMatrixMode(GL_MODELVIEW);
}
int main(int argc,char **argv)
{
        glutInit(&argc,argv);
        glutInitWindowSize(500,500);
        glutInitWindowPosition(100,100);
        glutCreateWindow("Rotating Cube");
        glutDisplayFunc(display);
        glutReshape(reshape);
        gltuIdleFunc(spin);
        glutMainLoop();
}
```

1. Implement Brenham’s line drawing algorithm for all types of slope.

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

int x1, y1, x2, y2;

void draw_pixel(int x, int y)
{
      glColor3f(1.0,0.0,0.0);
      glBegin(GL_POINTS);
      glVertex2i(x, y);
      glEnd();
}

void brenhams_line_draw(int x1, int y1, int x2, int y2)
{
      int dx=x2-x1,dy=y2-y1;
      int p=2*dy*dx;
      int twoDy=2*dy;
      int twoDyMinusDx=2*(dy-dx);         // paranthesis are required
      int x=x1,y=y1;
      if(dx<0)
      {
            x=x2;
            y=y2;
            x2=x1;
      }
      draw_pixel(x, y);
      while(x<x2)
      {
            x++;
            if(p<0)
                  p+=twoDy;
            else
            {
                  y++;
                  p+=twoDyMinusDx;
            }
            draw_pixel(x, y);
      }
}
void myInit()
{
      glClearColor(0.0,0.0,0.0,1.0);
      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      gluOrtho2D(0.0, 500.0, 0.0, 500.0);
      glMatrixMode(GL_MODELVIEW);
}
void display()
{
      glClear(GL_COLOR_BUFFER_BIT);
      brenhams_line_draw(x1, y1, x2, y2);
      glFlush();
}
```

```
void main(int argc, char **argv)
{
	printf( "Enter Start Points (x1,y1)\n");
	scanf("%d %d", &x1, &y1);
	printf( "Enter End Points (x2,y2)\n");
	scanf("%d %d", &x2, &y2);

	glutInit(&argc, argv);
	glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
	glutInitWindowSize(500, 500);
	glutInitWindowPosition(0, 0);
	glutCreateWindow("Bresenham's Line Drawing");
	myInit();
	glutDisplayFunc(display);
	glutMainLoop();
}
```

OUTPUT :



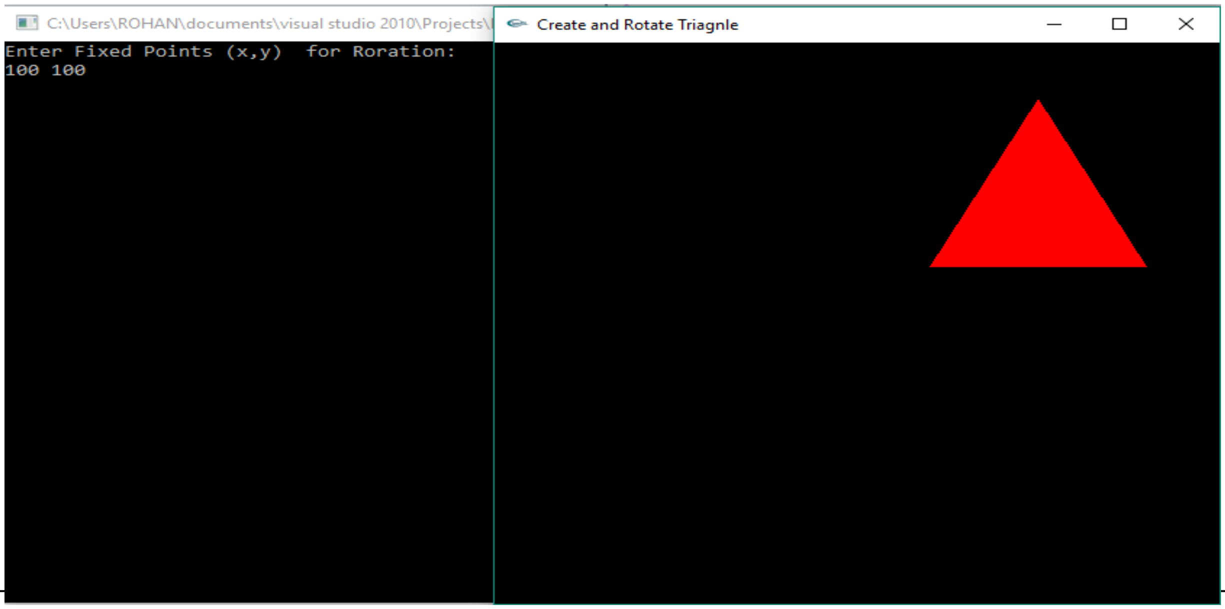
```
Enter Start Points (x1,y1)
50 50
Enter End Points (x2,y2)
400 400
```

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

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

int x,y;
int rFlag=0;

void draw_pixel(float x1,float y1)
{
      glColor3f(0.0,0.0,1.0);
      glPointSize(5.0);
      glBegin(GL_POINTS);
      glVertex2f(x1,y1);
      glEnd();
}

void triangle()
{
      glColor3f(1.0,0.0,0.0);
      glBegin(GL_POLYGON);
         glVertex2f(100,100);
         glVertex2f(250,400);
         glVertex2f(400,100);
      glEnd();
}

float th=0.0;
float trX=0.0,trY=0.0;
void display()
{
      glClear(GL_COLOR_BUFFER_BIT);
      glLoadIdentity();

      if(rFlag==1) //Rotate Around origin
      {
            trX=0.0;
            trY=0.0;
            th+=0.1;
            draw_pixel(0.0,0.0);
      }
      if(rFlag==2) //Rotate Around Fixed Point
      {
            trX=x;
            trY=y;
            th+=0.1;
            draw_pixel(x,y);
      }
      glTranslatef(trX,trY,0.0);
      glRotatef(th,0.0,0.0,1.0);
      glTranslatef(-trX,-trY,0.0);
        triangle();
      glutPostRedisplay();
      glutSwapBuffers();
}
```

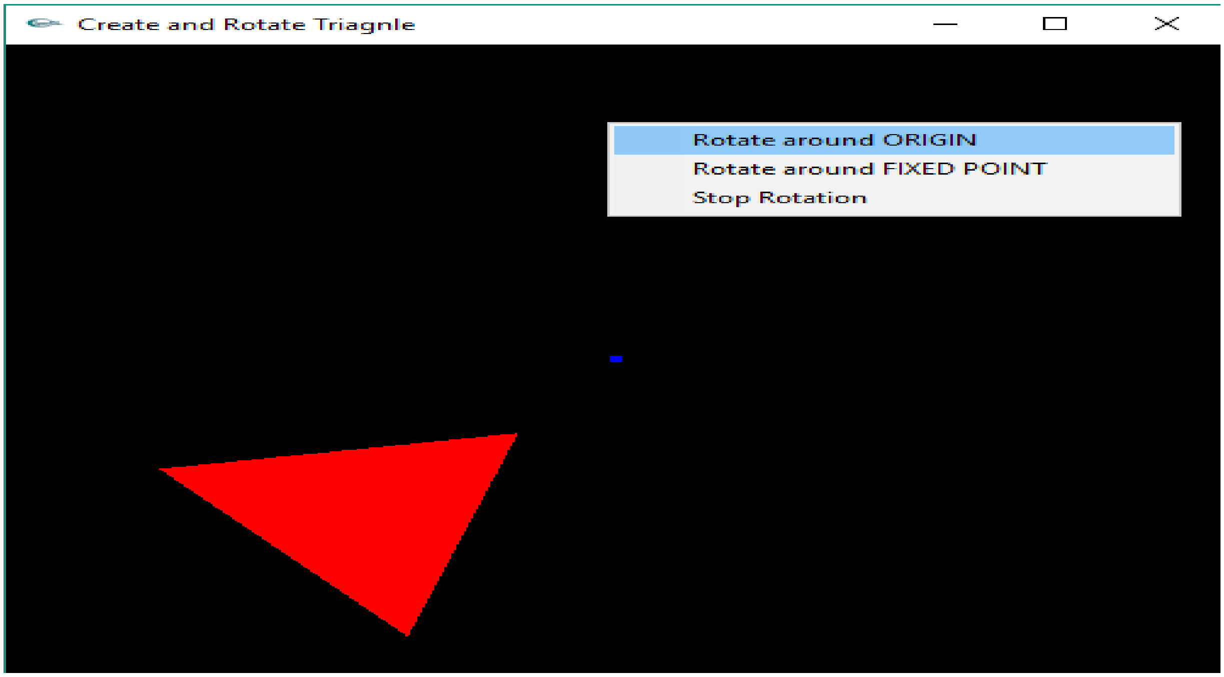
```
void myInit()
{
      glClearColor(0.0,0.0,0.0,1.0);
      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      gluOrtho2D(-500.0, 500.0, -500.0, 500.0);
      glMatrixMode(GL_MODELVIEW);
}


void rotateMenu (int option)
{
      if(option==1)
            rFlag=1;
      if(option==2)
            rFlag=2;
      if(option==3)
            rFlag=3;
}
void main(int argc, char **argv)
{
      printf( "Enter Fixed Points (x,y) for Roration: \n");
      scanf("%d %d", &x, &y);
      glutInit(&argc, argv);
      glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB);
      glutInitWindowSize(500, 500);
      glutInitWindowPosition(0, 0);
      glutCreateWindow("Create and Rotate Triangle");
      myInit();
      glutDisplayFunc(display);
      glutCreateMenu(rotateMenu);
      glutAddMenuEntry("Rotate around ORIGIN",1);
      glutAddMenuEntry("Rotate around FIXED POINT",2);
      glutAddMenuEntry("Stop Rotation",3);
      glutAttachMenu(GLUT_RIGHT_BUTTON);
      glutMainLoop();
}
```

OUTPUT :

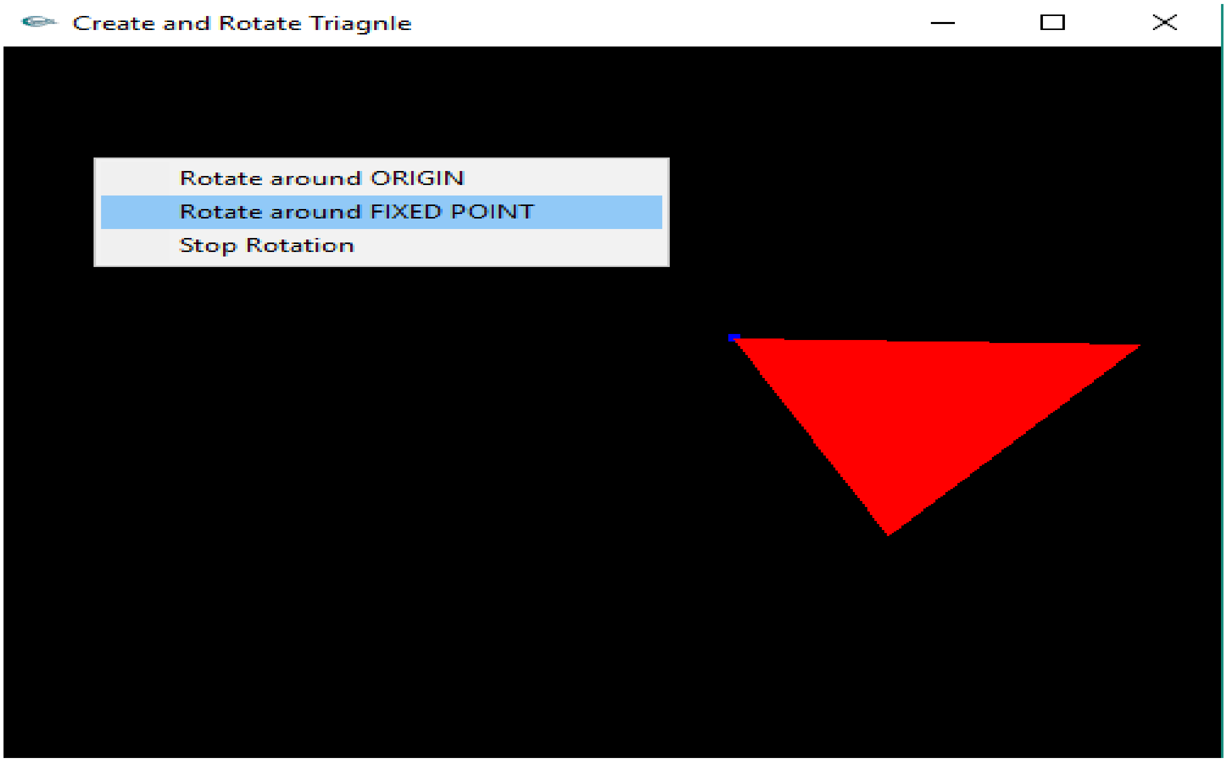

C:\Users\ROHAN\documents\visual studio 2010\Projects\
Create and Rotate Triagnle

```
Enter Fixed Points (x,y)  for Roration:
100 100
```

Rotation Around the ORIGIN



Rotation Around the FIXED POINT

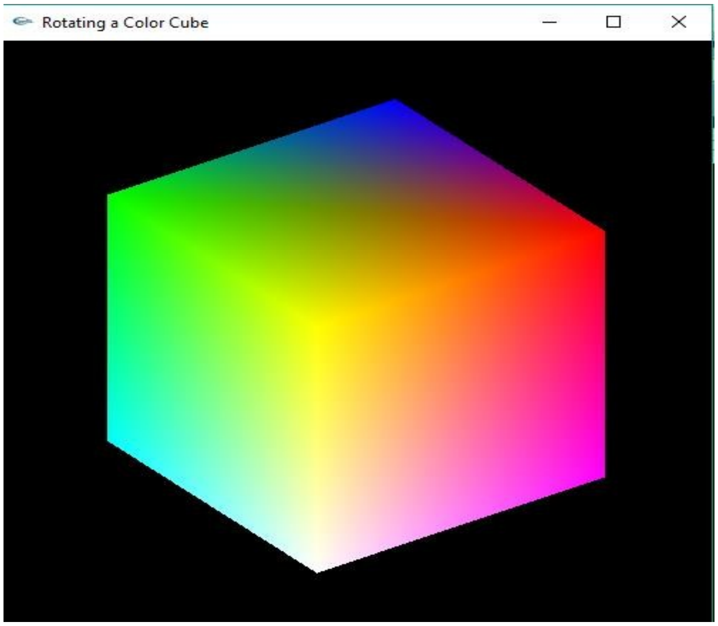


3. Program to draw a color cube and spin it using OpenGL transformation matrices.

```
#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}};


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);
}

GLfloat theta[] = {0.0,0.0,0.0};
GLint axis = 2;

void display(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);
      colorcube();
      glutSwapBuffers();
}

void spinCube()
{
      theta[axis] += 1.0;
      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_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(500, 500);
    glutCreateWindow("Rotating a Color Cube");
    glutReshapeFunc(myReshape);
    glutDisplayFunc(display);
    glutIdleFunc(spinCube);
    glutMouseFunc(mouse);
    glEnable(GL_DEPTH_TEST); /* Enable hidden--surface--removal */
    glutMainLoop();
}
```

OUTPUT :

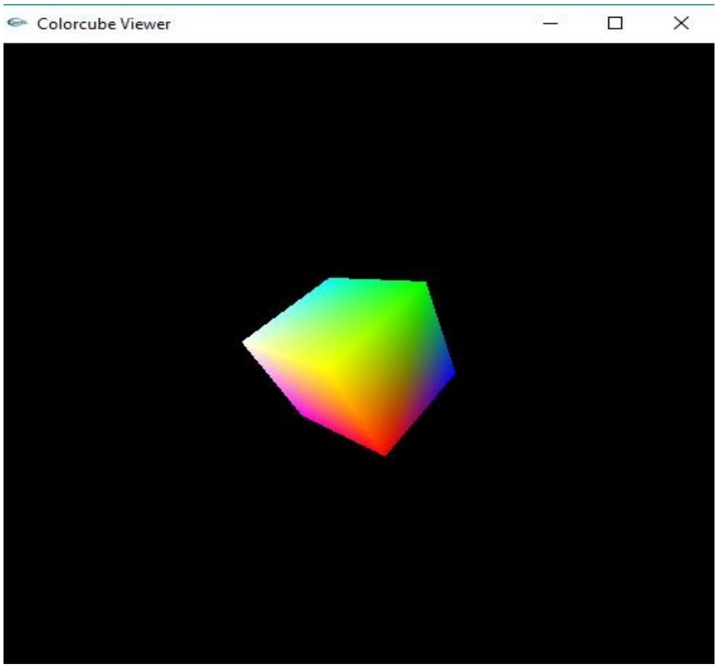


4. Program to draw a color cube and allow the user to move the camera suitably to experiment with perspective viewing.

```
#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}};


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);
}

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

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);
}

void  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();
}
```

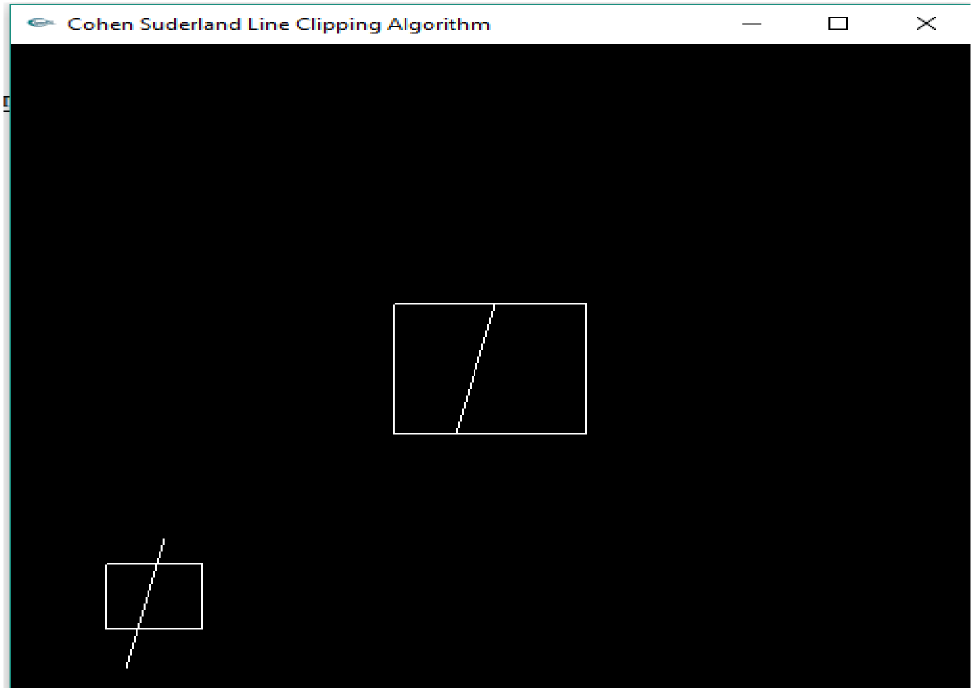
OUTPUT :



5. Program to clip a lines using Cohen-Sutherland line-clipping algorithm.

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

double xmin=50,ymin=50, xmax=100,ymax=100;
double xvmin=200,yvmin=200,xvmax=300,yvmax=300;
const int RIGHT = 8;
const int LEFT = 2;
const int TOP = 4;
const int BOTTOM = 1;

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);
      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(1.0, 1.0, 1.0);
            glBegin(GL_LINE_LOOP);
                  glVertex2f(xvmin, yvmin);
                  glVertex2f(xvmax, yvmin);
                  glVertex2f(xvmax, yvmax);
                  glVertex2f(xvmin, yvmax);
            glEnd();

            glColor3f(1.0,1.0,1.0);
            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(1.0,1.0,1.0);
      glBegin(GL_LINES);
        glVertex2d (x0, y0);
        glVertex2d (x1, y1);
      glEnd();
      glColor3f(1.0, 1.0, 1.0);
      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(0.0,0.0,0.0,1.0);
      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      gluOrtho2D(0.0,500.0,0.0,500.0);
      glMatrixMode(GL_MODELVIEW);

}
void main(int argc, char **argv)
{
      glutInit(&argc,argv);
      glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
      glutInitWindowSize(500,500);
      glutInitWindowPosition(0,0);
      glutCreateWindow("Cohen Suderland Line Clipping Algorithm");
      myinit();
      glutDisplayFunc(display);
      glutMainLoop();
}
```

OUTPUT :

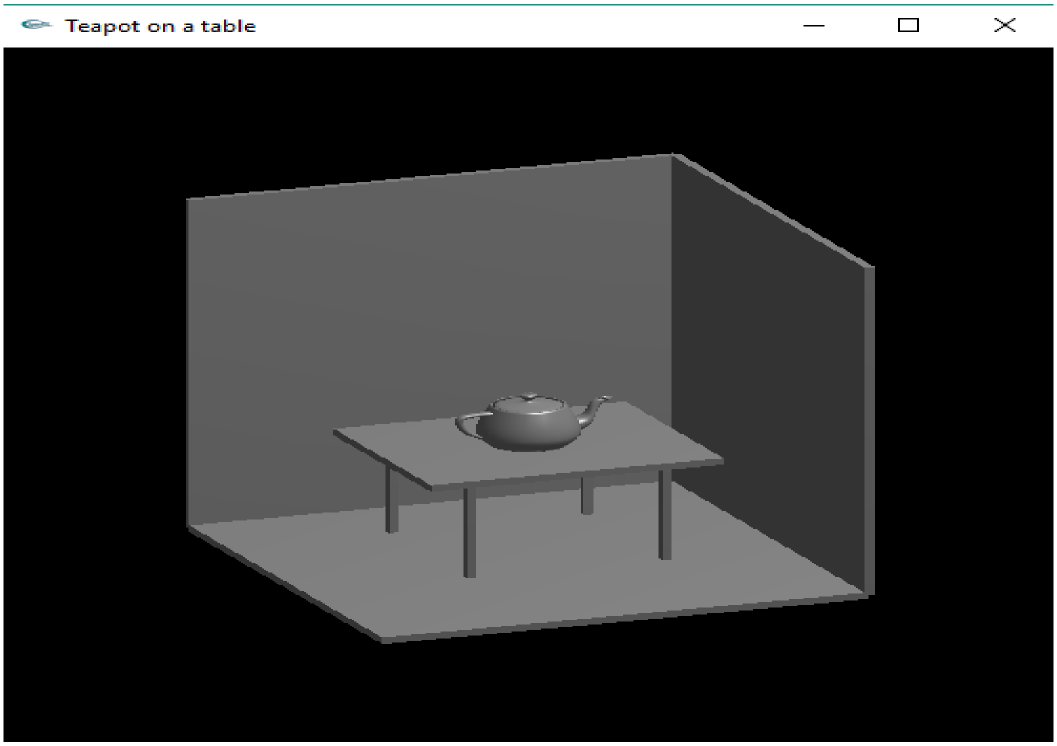


6. Program to draw a simple shaded scene consisting of a tea pot on a table. Define suitably the position and properties of the light source along with the properties of the properties of the surfaces of the solid object used in the scene.

```
#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(1.0);
      glPopMatrix();
}

void tableLeg(GLfloat x,GLfloat y,GLfloat z)
{
      glPushMatrix();
      glTranslatef(x,y,z);
      glScalef(0.02,0.3,0.02);
      glutSolidCube(1.0);
      glPopMatrix();
}

void wall(GLfloat x,GLfloat y,GLfloat z)
{
      glPushMatrix();
      glTranslatef(x,y,z);
      glScalef(1.0,1.0,0.02);
      glutSolidCube(1.0);
      glPopMatrix();
}

void light()
{
      GLfloat mat_ambient[]={1.0,1.0,1.0,1.0};
      GLfloat mat_diffuse[]={0.5,0.5,0.5,1.0};
      GLfloat mat_specular[]={1.0,1.0,1.0,1.0};
      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[]={2.0,6.0,3.0,1.0};
      GLfloat lightIntensity[]={0.7,0.7,0.7,1.0};
      glLightfv(GL_LIGHT0,GL_POSITION,light_position);
      glLightfv(GL_LIGHT0,GL_DIFFUSE,lightIntensity);
}
```

```
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(-2.0,2.0,5.0,0.0,0.0,0.0,0.0,1.0,0.0);

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

      teapot(0.0,teapotP,0.0);                              //Create teapot

      tableTop(0.0,tabletopP,0.0);                    //Create table’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(0.0,0.0,-wallP);                           //Create 1st wall
      glRotatef(90.0,1.0,0.0,0.0);
      wall(0.0,0.0,wallP);                            //Create 2nd wall
      glRotatef(90.0,0.0,1.0,0.0);
      wall(0.0,0.0,wallP);                            //Create 3rd wall
      glFlush();
}

void myinit()
{
      glClearColor(0.0,0.0,0.0,1.0);
      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      glOrtho(-1.0,1.0,-1.0,1.0,-1.0,10.0);
      glMatrixMode(GL_MODELVIEW);
}

void 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();
}
```

OUTPUT :

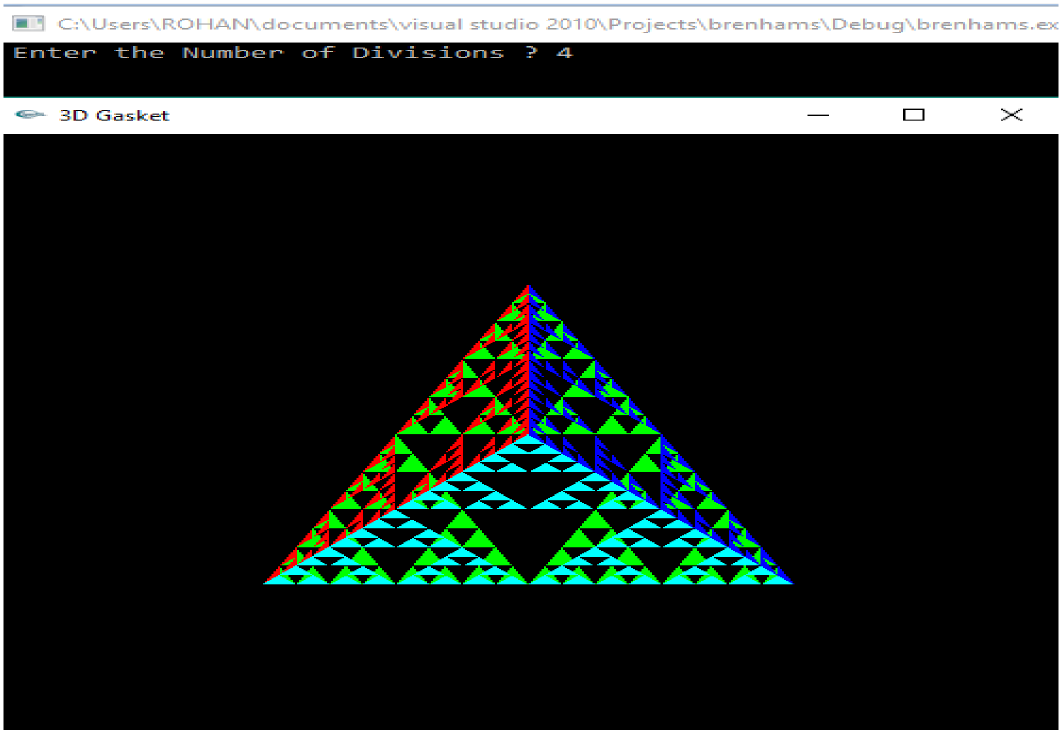


7. Program to recursively subdivide a tetrahedron to from 3D Sierpinski gasket. The number of recursive steps is to be specified by the user

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

float point v[][3]={{0.0, 0.0, 0.0}, {0.0, 1.0, -1.0},{-1.0, -1.0, -1.0},
                    {1.0, -1.0, -1.0}};
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 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(triangle(a,b,c)); /* draw triangle at end of recursion */
}


void tetrahedron( int m)
{
      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);
}

void main(int argc, char **argv)
{
      printf(" Enter the Number of Divisions ? ");
      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();
}
```

OUTPUT :


C:\Users\ROHAN\documents\visual studio 2010\Projects\brenhams\Debug\brenhams.ex
3D Gasket

```
Enter the Number of Divisions ? 4
```

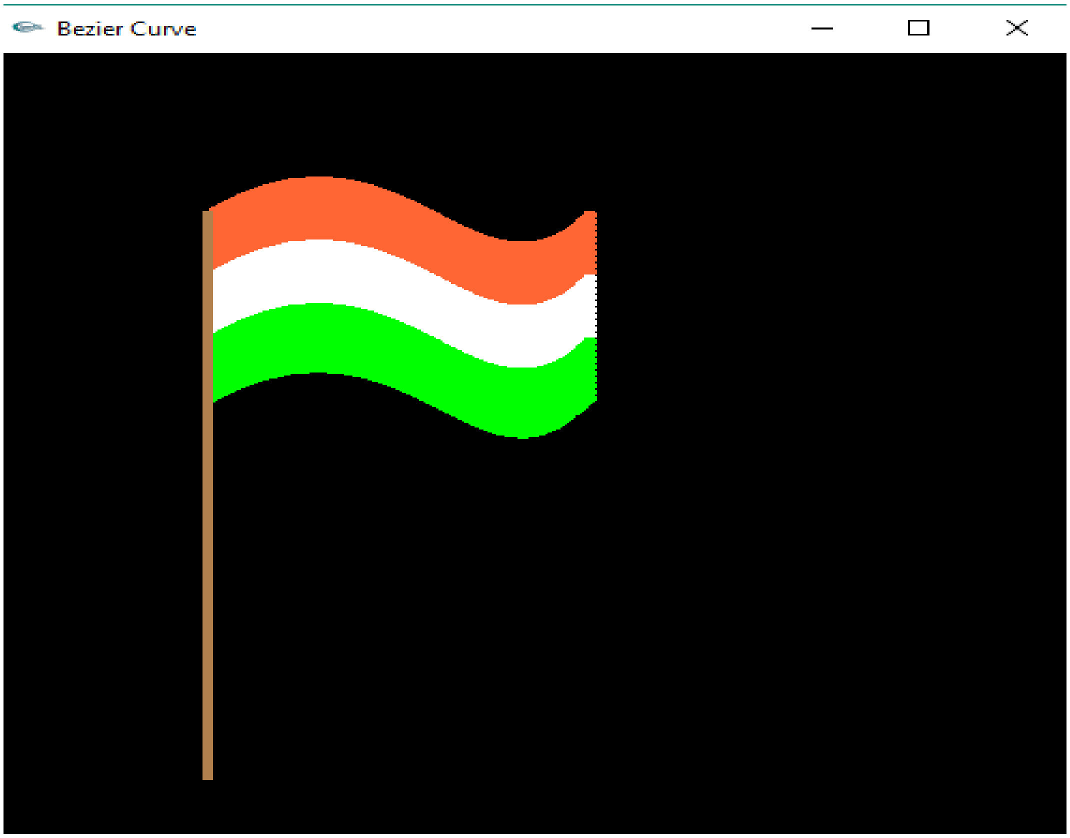
8. Develop a menu driven program to animate a flag using Bezier curve algorithm.

```
#include<GL/glut.h>
#include<stdio.h>
#include<math.h>
#define PI 3.1416
typedef struct point
{
            GLfloat x, y, z;
};
void bino(int n, int *C)
{
      int 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(float u, point *pt1, int cPt, point *pt2, int *C)
{
            int k, n=cPt-1;
            float bFcn;
            pt1 ->x =pt1 ->y = pt1->z=0.0;
            for(k=0; k< cPt; k++)
            {
                        bFcn = C[k] * pow(u, k) * pow( 1-u, n-k);
                        pt1 ->x += pt2[k].x * bFcn;
                        pt1 ->y += pt2[k].y * bFcn;
                        pt1 ->z += pt2[k].z * bFcn;
            }
}
void bezier(point *pt1, int cPt, int bPt)
{
            point bcPt;
            float u;
            int *C, k;
            C= new int[cPt];
            bino(cPt-1, C);
            glBegin(GL_LINE_STRIP);
            for(k=0; k<=bPt; k++)
            {
                        u=float(k)/float(bPt);
                        computeBezPt(u, &bcPt, cPt, pt1, C);
                        glVertex2f(bcPt.x, bcPt.y);
            }
            glEnd();
            delete[]C;
}
```

```
float theta = 0;
void display()
{
            glClear(GL_COLOR_BUFFER_BIT);
            int nCtrlPts = 4, nBCPts =20;
            point ctrlPts[4] = {{100, 400, 0},  {150, 450, 0},  {250, 350, 0},
                                {300, 400, 0}};
            ctrlPts[1].x +=50*sin(theta * PI/180.0);
            ctrlPts[1].y +=25*sin(theta * PI/180.0);
            ctrlPts[2].x -= 50*sin((theta+30)  * PI/180.0);
            ctrlPts[2].y -= 50*sin((theta+30)  * PI/180.0);
            ctrlPts[3].x -= 25*sin((theta)  * PI/180.0);
            ctrlPts[3].y += sin((theta-30)  * PI/180.0);
            theta+=0.2;
            glClear(GL_COLOR_BUFFER_BIT);
            glColor3f(1.0, 1.0, 1.0);
            glPointSize(5);
            glPushMatrix();
            glLineWidth(5);
            glColor3f(1, 0.4, 0.2);  //Indian flag: Orange color code
            for(int i=0;i<50;i++)
            {
                        glTranslatef(0, -0.8, 0);
                        bezier(ctrlPts, nCtrlPts, nBCPts);
            }
            glColor3f(1, 1, 1);     //Indian flag: white color code
            for(int i=0;i<50;i++)
            {
                        glTranslatef(0, -0.8, 0);
                        bezier(ctrlPts, nCtrlPts, nBCPts);
            }
            glColor3f(0, 1, 0);     //Indian flag: green color code
            for(int i=0;i<50;i++)
            {
                        glTranslatef(0, -0.8, 0);
                        bezier(ctrlPts, nCtrlPts, nBCPts);
            }
            glPopMatrix();
            glColor3f(0.7, 0.5,0.3);
            glLineWidth(5);
            glBegin(GL_LINES);
                        glVertex2f(100,400);
                        glVertex2f(100,40);
            glEnd();

            glutPostRedisplay();
            glutSwapBuffers();
}
void init()
{
            glMatrixMode(GL_PROJECTION);
            glLoadIdentity();
            gluOrtho2D(0,500,0,500);
}
```

```
void main(int argc, char **argv)
{
            glutInit(&argc, argv);
            glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
            glutInitWindowPosition(0, 0);
            glutInitWindowSize(500,500);
            glutCreateWindow("Bezier Curve");
            init();
            glutDisplayFunc(display);
            glutMainLoop();
}
```

OUTPUT :

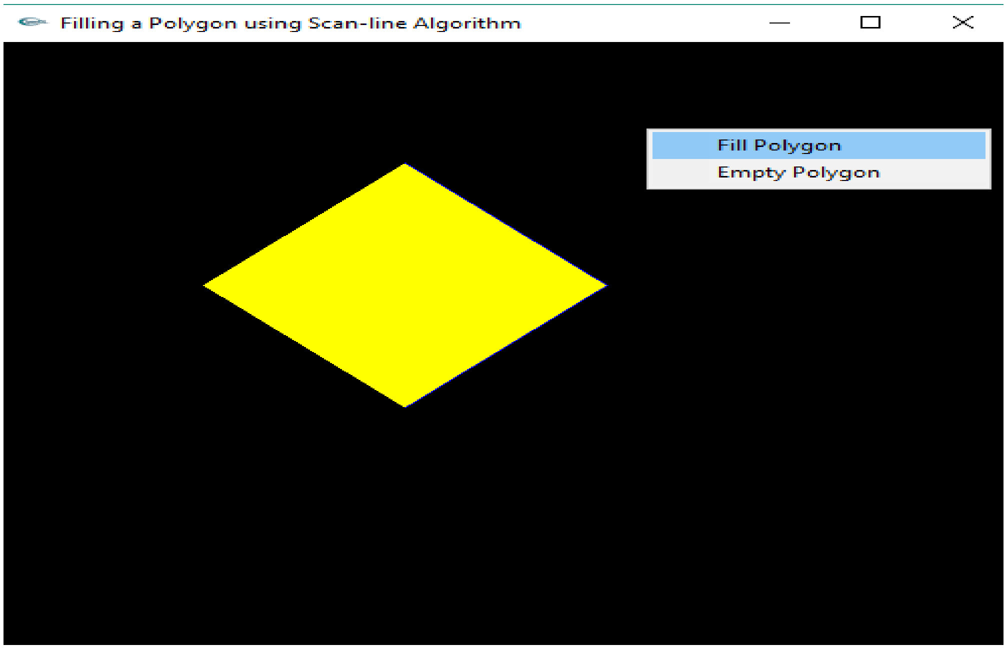


9. Develop a menu driven program to fill any given polygon using scan-line area filling algorithm.

```
#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();
}
```

OUTPUT :



WEB LINK :

https://drive.google.com/open?id=1nS2plGPXyv4n3I0bBYnWSMBae3gTzl6J

You can visit above link to get the following

1. **Softcopy of CG LAB(15CSL68) Manual.**
2. **Output daigrams of each experiments.**