VISVESVARAYA TECHNOLOGICAL UNIVERSITY
"Jnana Sangama", Belgavi-590 014

Laboratory Programs of
"Computer Graphics and Visualization with Mini Project"
18CSL67

Prepared by
Mr. Anser Pasha C.A., B.E., M.Tech
Asst. Prof, Dept of CS&E,
A.I.T., CHIKMAGALUR

Department of Computer Science and Engineering
ADHICHUNCHANAGIRI INSTITUTE OF TECHNOLOGY
CHIKMAGALUR -577102

2021-22

Syllabus

Subject: Computer Graphics Lab **Subject Code:18CSL68**

Part-A
1. Implement Brenham's line drawing algorithm for all types of slope.
2. Create and rotate a triangle about the origin and a fixed point.
3. Draw a colour cube and spin it using OpenGL transformation matrices.
4. Draw a color cube and allow the user to move the camera suitably to experiment with perspective viewing.
5. Clip a line using Cohen-Sutherland algorithm.
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.
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.
8. Develop a menu driven program to animate a flag using Bezier Curve algorithm.
9. Develop a menu driven program to fill the polygon using scan line algorithm.
Part-B
Student should develop mini project on the topics mentioned below or similar applications using Open GL API. Consider all types of attributes like color, thickness, styles, font, background, speed etc., while doing mini project.

Program No. 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 bresenhams_line_draw(int x1, int y1, int x2, int y2)
{
        int dx,dy,m,x,y, decision_parameter;
        dx = x2 - x1;
        dy = y2 - y1;
        m = dy/dx;

        if(m < 1)
        {
      decision_parameter = 2*dy - dx;
      x = x1;                                   // initial x
      y = y1;                                   // initial y
      if(dx < 0)                          //  decide the first point and second point
       {
         x = x2;
         y = y2;
         x2 = x1;
      }
      draw_pixel(x, y);                         // plot a point
```

```
        while(x < x2)                                   // from 1st point to 2nd point
        {
            if(decision_parameter >= 0)
            {
                x = x+1;
                y = y+1;
                decision_parameter=decision_parameter + 2*dy - 2*dx ;
            }


else
            {
                x = x+1;
                y = y;
                decision_parameter = decision_parameter + 2*dy;
            }
            draw_pixel(x, y);
        }
    }
    else if(m > 1)
    {
        int decision_parameter = 2*dx - dy;
        int x = x1; // initial x
        int y = y1; // initial y
        if(dy < 0)
        {
            x = x2;
            y = y2;
            y2 = y1;
        }
        draw_pixel(x, y);
        while(y < y2)
        {
```

```
            if(decision_parameter >= 0)
            {
                x = x+1;
                y = y+1;
                decision_parameter=decision_parameter + 2*dx - 2*dy;
            }
            else
            {
                y = y+1;
                x = x;
                decision_parameter = decision_parameter + 2*dx;
            }
            draw_pixel(x, y);
        }
    }
    else if (m == 1)
    {
        int x = x1;
        int y = y1;
        draw_pixel(x, y);
        while(x < x2)
        {
            x = x+1;
            y = y+1;
            draw_pixel(x, y);
        }
    }
}

void init()
{
    glClearColor(1,1,1,1);
    gluOrtho2D(0.0, 500.0, 0.0, 500.0);       // left ->0, right ->500, bottom ->0, top ->500
}
```

```
void display()
{
   glClear(GL_COLOR_BUFFER_BIT);
   bresenhams_line_draw(x1, y1, x2, y2);
   glFlush();
}

int main(int argc, char **argv)
{
   printf( "Enter Start Points (x1,y1)\n");
   scanf("%d%d", &x1, &y1);                          // 1st point from user

   printf( "Enter End Points (x2,y2)\n");
   scanf("%d%d", &x2, &y2);                          // 2nd point from user

   glutInit(&argc, argv);                            // initialize graphics system
   glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);      // single buffered mode with RGB colour variants
   glutInitWindowSize(500, 500);                     // 500 by 500 window size
   glutInitWindowPosition(220, 200);                 // where do you wanna see your window
   glutCreateWindow("Bresenham's Line Drawing");     // the title of your window

   init();                                           // initialize the canvas

   glutDisplayFunc(display);                         // call display function

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

Program No. 2:

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

```
#include<stdio.h>
#include<gl/glut.h>
float h, k, theta;
void triangle()
{
        glBegin(GL_TRIANGLES);
        glVertex2f(h, k);
        glVertex2f(h+400, k);
        glVertex2f(h+200, k+400);
        glEnd();
}
void init()
{
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D( - 600, 600, - 600, 600);
        glMatrixMode(GL_MODELVIEW);
}
void display()
{
        glClear(GL_COLOR_BUFFER_BIT);
        triangle();
        glLoadIdentity();
        glRotatef(theta, 0, 0, 1);
        glTranslatef(-h, -k, 0);
        glColor3f(0, 0, 1);
        triangle();
        glFlush();
}
```

```
int main(int argc, char **argv)
{
        glutInit(&argc, argv);
        printf("Enter Fixed Points (x,y) for Rotation: \n");
        scanf("%f%f", &h, &k);
        printf("Enter the rotation angle\n");
        scanf("%f", &theta);
        glutInitDisplayMode(GLUT_SINGLE);
        glutInitWindowSize(600, 600);
        glutCreateWindow("Triangle Rotation");
        glutDisplayFunc(display);
        init();
        glutMainLoop();
}
```

-

Program No. 3:

Draw a colour cube and spin it using the OpenGL transformation matrices

```
#include<gl/glut.h>
GLfloat ver[8][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 col[8][3]={{0,0,1}, {0,1,1}, {1,1,1},{1,0,1}, {0,0,0},{0,1,0},{1,1,0},{1,0,0}};
float theta[ ]={0,0,0};
int axis=0;

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

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

```
void display()
{
        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);
        color_cube();
        glFlush();
        glutSwapBuffers();
}

void spincube()
{
        theta[axis]+=0.1;
        if(theta[axis]>=360)
        theta[axis]=0;
        display();
}

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

int main(int argc, char **argv)
{
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_DOUBLE | GLUT_DEPTH);
        glutInitWindowSize(500,500);
        glutCreateWindow("Rotating Color Cube");
        glutReshapeFunc(myReshape);
        glutDisplayFunc(display);
        glutIdleFunc(spincube);
        glutMouseFunc(mouse);
        glEnable(GL_DEPTH_TEST);
        glutMainLoop();
}
```

Program No. 4:

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

```
#include<stdio.h>
#include<gl/glut.h>
GLfloat ver[8][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 col[8][3]={{0,0,1}, {0,1,1}, {1,1,1},{1,0,1}, {0,0,0},{0,1,0},{1,1,0},{1,0,0}};
float v1[3]={0,0,5};

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

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

```
void display()
{
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glLoadIdentity();
gluLookAt(v1[0],v1[1],v1[2],0,0,0,0,1,0);
colorcube();
glFlush();
glutSwapBuffers();
}

void key(unsigned char f, int x, int y)
{
   if(f=='x')  v1[0]  -= 0.10;
   if(f=='X')  v1[0]  += 0.10;
   if(f=='y')  v1[1]  -= 0.10;
   if(f=='Y')  v1[1] += 0.10;
   if(f=='z')  v1[2]  -= 0.10;
   if(f=='Z')  v1[2]  += 0.10;
   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*h/w,2.0*h/w,2.0,20);
   else
   glFrustum(-2.0*w/h,2.0*w/h,-2.0,2.0,2.0,20);
   glMatrixMode(GL_MODELVIEW);
}
```

```
int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_DEPTH);
    glutInitWindowSize(500,500);
    glutCreateWindow("Moving Camera around the Cube");
    glutInitWindowPosition(10,10);
    glutDisplayFunc(display);
    glutKeyboardFunc(key);
    glEnable(GL_DEPTH_TEST);
    glutReshapeFunc(myReshape);
    glutMainLoop();
}
```

Program No. 5:

Clip a line using Cohen-Sutherland algorithm

```
#include<stdio.h>
#include<gl/glut.h>
float xmin=50,ymin=50,xmax=100,ymax=100;
float xvmin=200,yvmin=200,xvmax=400,yvmax=400;
int left=1,right=2,bottom=4,top=8;
float sx,sy,vx1,vy1,vx2,vy2;
float x1,y1,x2,y2;
int compute(float x,float y)
{
        int code=0;
        if(x<xmin)
                code=left;
        if(x>xmax)
                code=right;
        if(y<ymin)
                code=bottom;
        if(y>ymax)
                code=top;
        return code;
}
void cohen(float x1,float y1,float x2,float y2)
{
        float x,y;
        int accept=0,done=0,codep,codeq,code;
        codep=compute(x1,y1);
        codeq=compute(x2,y2);
        do
        {
        if(!(codep | codeq))
        {
        accept=1;
        done=1;
        }
```

```
else if(codep & codeq)
        done=1;
else
{
        code=codep?codep:codeq;
        if(code & left)
        {
                x=xmin;
        y=y1+(y2-y1)*(xmin-x1)/(x2-x1);
        }
        else if(code & right)
        {
        x=xmax;
        y=y1+(y2-y1)*(xmax-x1)/(x2-x1);
        }
        else if(code & bottom)
        {
        y=ymin;
        x=x1+(x2-x1)*(ymin-y1)/(y2-y1);
        }
        else
        {
        y=ymax;
        x=x1+(x2-x1)*(ymax-y1)/(y2-y1);
        }
        if(code= =codep)
        {
        x1=x;
        y1=y;
        codep=compute(x1,y1);
        }
        else
        {
        x2=x;
        y2=y;
        codeq=compute(x2,y2);
```

```
		}
		}
		}
		while(!done);
		if(accept)
		{
			sx=(xvmax-xvmin)/(xmax-xmin);
			sy=(yvmax-yvmin)/(ymax-ymin);
			vx1=xvmin+(x1-xmin)*sx;
			vy1=yvmin+(y1-ymin)*sy;
			vx2=xvmax+(x2-xmax)*sx;
			vy2=yvmax+(y2-ymax)*sy;
		}
}
void display()
{
		glClear(GL_COLOR_BUFFER_BIT);
		glLineWidth(2);
		//input line segment
		glBegin(GL_LINES);
		glColor3f(1,1,1);
		glVertex2f(x1,y1);
		glVertex2f(x2,y2);
		glEnd();
		//clipping window
		glBegin(GL_LINE_LOOP);
		glVertex2f(xmin,ymin);
		glVertex2f(xmax,ymin);
		glVertex2f(xmax,ymax);
		glVertex2f(xmin,ymax);
		glEnd();
		cohen(x1,y1,x2,y2);
		glBegin(GL_LINE_LOOP);
		glVertex2f(xvmin,yvmin);
		glVertex2f(xvmax,yvmin);
		glVertex2f(xvmax,yvmax);
```

```
        glVertex2f(xvmin,yvmax);
        glEnd();
        //to clip the line segment
        glBegin(GL_LINES);
        glVertex2f(vx1,vy1);
        glVertex2f(vx2,vy2);
        glEnd();
        glFlush();
}
void init()
{
        glClearColor(0,0,0,1);
        gluOrtho2D(0,500,0,500);
}
int main(int argc,char ** argv)
{
        glutInit(&argc,argv);
        printf("Enter the values of x1,y1,x2 and y2\n");
        scanf("%f%f%f%f",&x1,&y1,&x2,&y2);
        glutInitDisplayMode(GLUT_SINGLE);
        glutInitWindowSize(500,500);
        glutCreateWindow("Clipping window");
        init();
        glutDisplayFunc(display);
        glutMainLoop();
}
```

Lab Experiment: 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.

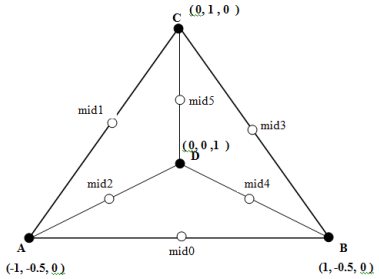
```
#include<stdio.h>
#include<gl/glut.h>
void init()
{
	glMatrixMode(GL_PROJECTION); glLoadIdentity();
	glOrtho(-4,4,-4,4,-10,10);
	glMatrixMode(GL_MODELVIEW);
}
void wall()
{
	glPushMatrix();
	glScalef(2.0,0.05,2);
	 glutSolidCube(2);
	glPopMatrix();
	glPushMatrix();
	glTranslatef(-2,2,0);
	glRotatef(-90,0,0,1);
	glScalef(2,0.05,2);
	glutSolidCube(2);
	glPopMatrix();
	glPushMatrix();
	glTranslatef(0,2,-2);
	glRotatef(90,1,0,0);
	glScalef(2,0.05,2);
	glutSolidCube(2);
	glPopMatrix();
}
```

```
void table()
{
        glPushMatrix();
        glTranslatef(0,0.5,0);
        glScalef(1,0.05,1);
        glutSolidCube(2);
        glPopMatrix();

        glPushMatrix();
        glTranslatef(-0.8,0.2,0.8);
        glScalef(0.1,0.05,0.1);
        glutSolidCube(2);
        glPopMatrix();

        glPushMatrix();
        glTranslatef(0.8,0.2,0.8);
        glScalef(0.1,0.05,0.1);
         glutSolidCube(2);
        glPopMatrix();

        glPushMatrix();
        glTranslatef(0.8,0.2,-0.8);
        glScalef(0.1,0.05,0.1);
         glutSolidCube(2);
        glPopMatrix();

        glPushMatrix();
        glTranslatef(-0.8,0.2,-0.8);
        glScalef(0.1,0.05,0.1);
        glutSolidCube(2);
        glPopMatrix();
}
```

```
void teapot()
{
        glPushMatrix();
        glTranslatef(0,1,0);
        glRotatef(45,0,1,0);
        glutSolidTeapot(0.5);
        glPopMatrix();
}
void display()
{
        float amb[ ]={1.9,0.7,1};
        float pos[ ]={2,4,1};
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glLoadIdentity();
        glMaterialfv(GL_FRONT,GL_AMBIENT,amb);
        glLightfv(GL_LIGHT0,GL_POSITION,pos);
        gluLookAt(2.5,1,2,0,0.5,0,0,1,0);
        wall();
        table();
        teapot();
        glFlush();
}
void main()
{
        glutInitDisplayMode(GLUT_SINGLE | GLUT_DEPTH);
        glutInitWindowSize(600,600);
        glutCreateWindow("teapot");
        init();
        glutDisplayFunc(display);
        glEnable(GL_DEPTH_TEST);
        glShadeModel(GL_SMOOTH);
        glEnable(GL_LIGHTING);
        glEnable(GL_LIGHT0);
        glEnable(GL_NORMALIZE);
        glutMainLoop();
}
```

Lab Experiment: 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.




```
#include<stdio.h>
#include<gl/glut.h>
float v[4][3]={{-1,-0.5,0},{1,-0.5,0},{0,1,0},{0,0,1}};
int n;
void triangle(float *a,float *b,float *c)
{
glVertex3fv(a); glVertex3fv(b); glVertex3fv(c);
}

void tetra(float *a,float *b,float *c,float *d)
{
glColor3f(1,0,0); triangle(a,b,c); glColor3f(0,1,0); triangle(a,c,d); glColor3f(0,0,1);
        triangle(a,b,d); glColor3f(1,1,0); triangle(b,c,d);
}
void divide(float *a,float *b,float *c,float *d,int k)
{
float m[6][3]; int j;
if(k>0)
{
for(j=0;j<3;j++)
{
m[0][j]=(a[j]+b[j])/2;
m[1][j]=(a[j]+c[j])/2;
m[2][j]=(a[j]+d[j])/2;
```

```
m[3][j]=(b[j]+c[j])/2;
m[4][j]=(b[j]+d[j])/2;
m[5][j]=(c[j]+d[j])/2;
}
divide(a,m[0],m[1],m[2],k-1);
divide(m[0],b,m[3],m[4],k-1);
divide(m[1],m[3],c,m[5],k-1);
divide(m[2],m[4],m[5],d,k-1);
}
                    else
tetra(a,b,c,d);
   }
void display()
{
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glEnable(GL_DEPTH_TEST); glBegin(GL_TRIANGLES);
        divide(v[0],v[1],v[2],v[3],n);
glEnd();
glFlush();
  }
void init()
{
glOrtho(-2,2,-2,2,-10,10);
}
void main()
{
printf("Enter the number of divisions\n");
scanf("%d",&n);
glutInitDisplayMode(GLUT_SINGLE | GLUT_DEPTH);
        glutInitWindowSize(700,700); glutCreateWindow("Sierpeniski Gasket");
init();
glutDisplayFunc(display);
glutMainLoop();
}
```

Lab Experiment: 08

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

```
#include<windows.h>
#include <GL/glu.h>
#include <GL/glut.h>
#include<math.h>
int AnFlag = 0;
int yFlag = 1,xFlag = 1;
float yC=-50,xC=-10;
float x[4],Y1[4],y2[4],y3[4];
void Menu(int Id)
{
        switch(Id)
        {
        case 1 : AnFlag = 1; break;
        case 2 : AnFlag = 0; break;
        case 3 : exit(0);
        }
}
void Idle()
{
        if(AnFlag == 1)
        {
        if(yC<50 && yFlag == 1)
        yC = yC + 0.2;
        if(yC>=50 && yFlag == 1)
        yFlag = 0;
        if(yC>-50 && yFlag == 0)
        yC = yC - 0.2;
        if(yC<=-50 && yFlag == 0)
        yFlag = 1;
        if(xC<20 && xFlag == 1)
        xC = xC + 0.2;
```

```
            if(xC>=20 && xFlag == 1)
            xFlag = 0;
            if(xC>-20 && xFlag == 0)
            xC = xC - 0.2;
            if(xC<=-20 && xFlag == 0)
            xFlag = 1;
            }
            glutPostRedisplay();
        }
        void Draw()
        {
            int i;
            double t,xt[200],y1t[200],y2t[200],y3t[200];
            glClear(GL_COLOR_BUFFER_BIT);
            x[0] = 300-xC; x[1] = 200; x[2] = 200; x[3] = 100;
            Y1[0] = 450; Y1[1] = 450+yC; Y1[2] = 450-yC; Y1[3] = 450;
            y2[0] = 400; y2[1] = 400+yC; y2[2] = 400-yC; y2[3] = 400;
            y3[0] = 350; y3[1] = 350+yC; y3[2] = 350-yC; y3[3] = 350;
            i=0;
            for (t = 0.0; t < 1.0; t += 0.005)
            {
xt[i] = pow(1-t,3)*x[0]+3*t*pow(1-t,2)*x[1]+3*pow(t,2)*(1-t)*x[2]+pow(t,3)*x[3];
y1t[i] = pow(1-t,3)*Y1[0]+3*t*pow(1-t,2)*Y1[1]+3*pow(t,2)*(1-)*Y1[2]+pow(t,3)*Y1[3];
y2t[i] = pow(1-t,3)*y2[0]+3*t*pow(1-t,2)*y2[1]+3*pow(t,2)*(1-t)*y2[2]+pow(t,3)*y2[3];
y3t[i] = pow(1-t,3)*y3[0]+3*t*pow(1-t,2)*y3[1]+3*pow(t,2)*(1-t)*y3[2]+pow(t,3)*y3[3];
            i++;
            }
            glColor3f(1,1,0);
            glBegin(GL_QUAD_STRIP);
            for (i=0;i<200;i++)
            {
            glVertex2d(xt[i],y1t[i]);
            glVertex2d(xt[i],y2t[i]);
            }
            glEnd();
            glColor3f(1,0,0);
```

```
        glBegin(GL_QUAD_STRIP);
        for (i=0;i<200;i++)
        {
        glVertex2d(xt[i],y2t[i]);
        glVertex2d(xt[i],y3t[i]);
        }
        glEnd();
        glColor3f(0.5,0.5,0.5);
        glRectf(85,460,100,0);
        glFlush();
}
void MyInit()
{
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(0,500,0,500);
        glMatrixMode(GL_MODELVIEW);
        glutCreateMenu(Menu);
        glutAddMenuEntry("Play Animation",1);
        glutAddMenuEntry("Stop Animation",2);
        glutAddMenuEntry("Exit",3);
        glutAttachMenu(GLUT_RIGHT_BUTTON);
}
int main(int argC,char *argV[])
{
        glutInit(&argC,argV);
        glutInitDisplayMode(GLUT_RGB|GLUT_SINGLE);
        glutInitWindowPosition(0,0);
        glutInitWindowSize(500,500);
        glutCreateWindow("Flag");
        MyInit();
        glutDisplayFunc(Draw);
        glutIdleFunc(Idle);
        glutMainLoop();
        return 0;
}
```

Lab Experiment: 09

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

```
#include <GL/glut.h>
int LE[500], RE[500];
int Fill_Flag = 0,Line_Flag = 0;
void Intersection(GLfloat x1,GLfloat y1,GLfloat x2,GLfloat y2)
{
        GLfloat temp,M,x;
        int i;
        if(y1>y2)
        {
                temp = y1;
                y1 = y2;
                y2 = temp;
                temp = x1;
                x1 = x2;
                x2 = temp;
        }
        if(y2 - y1 == 0)
                M = x2 - x1;
        else
                M = ( x2 - x1 ) / (y2 - y1);
        x = x1;
        for(i=y1;i<=y2;i++)
        {
                if(x<LE[i])
                {
                        LE[i] = x;
                }
                if(x>RE[i])
                {
                        RE[i] = x;
                }
                x = x + M;
        }  }
```

```
void Display()
{
        GLfloat x1=250,y1=150;
        GLfloat x2=400,y2=250;
        GLfloat x3=250,y3=350;
        GLfloat x4=100,y4=250;
        int i;
        GLint x,y;
        glClear(GL_COLOR_BUFFER_BIT);
        for(i=0;i<500;i++)
        {
                LE[i] = 500;
                RE[i] = 0;
        }
        if(Line_Flag == 1)
        {
                glColor3f(0,1,0);
                glBegin(GL_LINE_LOOP);
                glVertex2f(x1,y1);
                glVertex2f(x2,y2);
                glVertex2f(x3,y3);
                glVertex2f(x4,y4);
                glEnd();
        }
        glColor3f(1,0,0);
        Intersection(x1,y1,x2,y2);
        Intersection(x2,y2,x3,y3);
        Intersection(x3,y3,x4,y4);
        Intersection(x4,y4,x1,y1);
        if(Fill_Flag == 1)
        {
                for(y=0;y<500;y++)
                {
                        if(LE[y]<=RE[y])
                        {
```

```
                                for(x=LE[y];x<=RE[y];x++)
                                {
                                glBegin(GL_POINTS);
                                glVertex2f(x,y);
                                glEnd();
                                glFlush();
                                }
                        }
                }
        }
}


void Line_Menu(int Id)
{
        if(Id == 1)
                Line_Flag = 1;
        if(Id == 2)
                Line_Flag = 2;
        glutPostRedisplay();
}


void Main_Menu(int Id)
{
        if(Id == 1)
                Fill_Flag = 1;
        if(Id == 2)
                exit(0);
        glutPostRedisplay();
}
```

```
int main(int argC, char *argV[])
{
        int Id;
        glutInit(&argC,argV);
        glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE);
        glutInitWindowSize(500,500);
        glutInitWindowPosition(50,50);
        glutCreateWindow("Lab9 : Scan-Line Fill Algorithm");
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(0,500,0,500);
        glMatrixMode(GL_MODELVIEW);
        Id = glutCreateMenu(Line_Menu);
        glutAddMenuEntry("Yes",1);
        glutAddMenuEntry("No",2);
        glutCreateMenu(Main_Menu);
        glutAddSubMenu("Out Line",Id);
        glutAddMenuEntry("Start Fill",1);
        glutAddMenuEntry("Exit",2);
        glutAttachMenu(GLUT_RIGHT_BUTTON);
        glutDisplayFunc(Display);
        glutMainLoop();
        return 0;
}
```