COMPUTER GRAPHICS & MULTIMEDIA LAB MANUAL

13CS601

1. Midpoint Line

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

int x1, y1, x2, y2;

void init()
{
    glClearColor(0, 0, 0, 0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(-500, 500, -500, 500);
}

void midPoint(int x1, int y1, int x2, int y2)
{
    int di,dx, dy, x, y, A, B;
    dx = x2 - x1;
    dy = y2 - y1;
    di = (2 * dy) - dx;
    A = 2 * (dy - dx);
    B = 2 * dy;
    y = y1; x = x1;
```

```
    glBegin(GL_POINTS);
    glVertex2f(x, y);
    while(x<x2){
        if(di<=0){
            di = di + B;
            x++;
        }
        else{
            di = di + A;
            x++;
            y++;
        }
        glVertex2f(x, y);
    }
    glEnd();
    glFlush();
}

void display()
{
    glClear(GL_COLOR_BUFFER_BIT);

    glColor3f(1.0, 0.0, 0.0);
    midPoint(x1, y1, x2, y2);

    glBegin(GL_LINES);
    glColor3f(1,1,1);
    glVertex2f(0,500);
    glVertex2f(0,-500);
    glVertex2f(-500,0);
    glVertex2f(500,0);
```

```
    glEnd();
    glFlush();
}

int main(int argc, char *argv[])
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowPosition(50, 50);
    glutInitWindowSize(800, 800);
    glutCreateWindow("Midpoint line algorithm");
    glClearColor(0,0,0,1);
    init();
    printf("Enter the vertices of the starting point and end point\n");
    scanf("%d%d%d%d", &x1, &y1, &x2, &y2);
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```

2. MIDPOINT CIRCLE

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

int r;

void init()
{
```

```
    glClearColor(0, 0, 0, 0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(-500, 500, -500, 500);
}

void midCircle(int r)
{
    int x=0,y=r;
    int d = 1.25-r;
    glBegin(GL_POINTS);
    while(y>x){
        if(d<0)
            d+= (2*x) + 3;
        else{
            d+= (2*(x-y)) + 5;
            y--;
        }
        x++;
        glVertex2f(x+250,y+250);
        glVertex2f(-x+250,y+250);
        glVertex2f(x+250,-y+250);
        glVertex2f(-x+250,-y+250);
        glVertex2f(y+250,x+250);
        glVertex2f(-y+250,x+250);
        glVertex2f(y+250,-x+250);
        glVertex2f(-y+250,-x+250);
    }
    glEnd();
    glFlush();
}
```

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

    glColor3f(1.0, 0.0, 0.0);
    midCircle(r);

    glBegin(GL_LINES);
    glColor3f(1,1,1);
    glVertex2f(0,500);
    glVertex2f(0,-500);
    glVertex2f(-500,0);
    glVertex2f(500,0);
    glEnd();
    glFlush();
}

int main(int argc, char *argv[])
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowPosition(50, 50);
    glutInitWindowSize(800, 800);
    glutCreateWindow("Midpoint Circle Algorithm");
    glClearColor(0,0,0,1);
    init();
    printf("Enter the radius of the circle\n");
    scanf("%d", &r);
    glutDisplayFunc(display);
    glutMainLoop();
```

```
    return 0;
}
```

3. COHEN SUTHERLAND

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

typedef int outcode;
outcode TOP=8,BOTTOM=4,RIGHT=2,LEFT=1,INSIDE=0;
int x0,y0,x1,y1,Xmin,Ymin,Xmax,Ymax;

void init()
{
    glClearColor(0, 0, 0, 0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0, 1000, 0, 1000);
}

outcode Computeoutcode(int x,int y)
{
        outcode code=INSIDE;
        if(y>Ymax)
                code|=TOP;
        if(y<Ymin)
                code|=BOTTOM;
        if(x>Xmax)
```

```
            code|=RIGHT;
        if(x<Xmin)
            code|=LEFT;
        return code;
}

void cohenSutherland(int x0,int y0,int x1,int y1)
{
        double x,y;
        outcode outcode0,outcode1,outcodeout;
        int accept=0,done=0;
        outcode0=Computeoutcode(x0,y0);
        outcode1=Computeoutcode(x1,y1);
        do
        {
            if(!(outcode0|outcode1))
            {
                accept=1;
                done=1;

            }
            else if(outcode0 & outcode1)
                done=1;
            else
            {
                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==0);
if(accept)
{
        glBegin(GL_LINES);
        glVertex2f(x0+450,y0+300);
```

```
                glVertex2f(x1+450,y1+300);
                glEnd();
        }
        glFlush();
}

void display()
{
    int a=x0,b=y0,c=x1,d=y1;
    glClear(GL_COLOR_BUFFER_BIT);

    glBegin(GL_LINE_LOOP);              //Before Clipping
    glColor3f(1,1,1);
    glVertex2f(Xmin,Ymax+300);
    glVertex2f(Xmax,Ymax+300);
    glVertex2f(Xmax,Ymin+300);
    glVertex2f(Xmin,Ymin+300);
    glEnd();

    glBegin(GL_LINES);
    glColor3f(1,0,0);
    glVertex2f(a,b+300);
    glVertex2f(c,d+300);
    glEnd();

    glColor3f(1, 0, 0);                 //After Clipping
    cohenSutherland(a,b,c,d);

    glBegin(GL_LINE_LOOP);
    glColor3f(1,1,1);
    glVertex2f(Xmin+450,Ymax+300);
```

```
    glVertex2f(Xmax+450,Ymax+300);
    glVertex2f(Xmax+450,Ymin+300);
    glVertex2f(Xmin+450,Ymin+300);
    glEnd();
    glFlush();
}

int main(int argc, char *argv[])
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowPosition(50, 50);
    glutInitWindowSize(800, 600);
    glutCreateWindow("Cohen Sutherland Algorithm");
    glClearColor(0,0,0,1);
    init();
    printf("Enter the size of the clipping area\n");
    scanf("%d%d%d%d", &Xmin, &Ymin, &Xmax, &Ymax);
    printf("Enter the vertices of the starting point and end point\n");
    scanf("%d%d%d%d", &x0, &y0, &x1, &y1);
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```

4. LIANG BARSKY

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

```
float xmin,ymin,xmax,ymax;
float x0,y0,x1,y1;

void init()
{
	glClearColor(0,0,0,1.0);
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	gluOrtho2D(0.0,1000,0.0,1000);
}

int diptest(float p,float q,float *t1,float *t2)
{
	float t=q/p;
	if(p<0.0)
	{
		if(t>*t1)
			*t1=t;
		if(t>*t2)
			return 0;
	}
	else if(p>0.0)
	{
		if(t<*t2)
			*t2=t;
		if(t<*t1)
			return 0;
	}
	else if(p==0.0)
	{
```

```
                if(q<0.0)
                        return 0;
        }
        return 1;
}

void lblcd(float x0,float y0,float x1,float y1)
{
        float dx=x1-x0,dy=y1-y0,t1=1.0,te=0.0;
        if(diptest(-dx,x0-xmin,&te,&t1))
                if(diptest(dx,xmax-x0,&te,&t1))
                        if(diptest(-dy,y0-ymin,&te,&t1))
                                if(diptest(dy,ymax-y0,&te,&t1))
                                {
                                        if(t1<1.0)
                                        {
                                                x1=x0+t1*dx;
                                                y1=y0+t1*dy;
                                        }
                                        if(te>0.0)
                                        {
                                                x0=x0+te*dx;
                                                y0=y0+te*dy;
                                        }
                                }
            if(!(x0<xmin && y0<ymin))
            {
                glColor3f(1.0,0.0,0.0);
                glBegin(GL_LINES);
                glVertex2f(x0+400,y0+300);
                glVertex2f(x1+400,y1+300);
```

```
            glEnd();
            glFlush();
        }
}

void display()
{
    glClear(GL_COLOR_BUFFER_BIT);

    glColor3f(1,1,1);
    glColor3f(1.0,0.0,0.0);                    //Before clipping
    glBegin(GL_LINES);
    glVertex2f(x0,y0+300);
    glVertex2f(x1,y1+300);
    glEnd();

    glColor3f(1.0,1.0,1.0);
    glBegin(GL_LINE_LOOP);
    glVertex2f(xmin,ymin+300);
    glVertex2f(xmax,ymin+300);
    glVertex2f(xmax,ymax+300);
    glVertex2f(xmin,ymax+300);
    glEnd();

    lblcd(x0,y0,x1,y1);                        //After clipping
    glColor3f(1.0,1.0,1.0);

    glBegin(GL_LINE_LOOP);
    glVertex2f(xmin+400,ymin+300);
    glVertex2f(xmax+400,ymin+300);
    glVertex2f(xmax+400,ymax+300);
```

```
        glVertex2f(xmin+400,ymax+300);
        glEnd();
        glFlush();
}

int main(int argc, char **argv)
{
        glutInit(&argc,argv);
        glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
        glutInitWindowSize(800,600);
        glutInitWindowPosition(50,50);
        glutCreateWindow("Liang Barsky Algorithm");
        init();
        printf("Enter the size of the clipping area\n");
   scanf("%f%f%f%f", &xmin, &ymin, &xmax, &ymax);
        printf("Enter the vertices of the starting point and end point\n");
        scanf("%f%f%f%f",&x0,&y0,&x1,&y1);
        glutDisplayFunc(display);
        glutMainLoop();
        return 0;
}
```

5. SIERPINSKY'S GASKET

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

void myInit()
{
```

```
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0, 10.0, 0.0, 10.0);
    glMatrixMode(GL_MODELVIEW);
    glClearColor(1.0, 1.0, 1.0, 1.0);
    glColor3f(0.0, 0.0, 1.0);
}

void triangle(GLfloat *a, GLfloat *b, GLfloat *c)
{
    glVertex2fv(a);
    glVertex2fv(b);
    glVertex2fv(c);
}

void draw_triangle(GLfloat *a, GLfloat *b, GLfloat *c, int k)
{
    GLfloat ab[2], bc[2], ac[2];
    int j;
    if(k>0)
    {
        for(j=0;j<2;j++)
                        ab[j] = (a[j]+b[j])/2.0;
                for(j=0;j<2;j++)
                        bc[j] = (b[j]+c[j])/2.0;
        for(j=0;j<2;j++)
                        ac[j] = (a[j]+c[j])/2.0;

        draw_triangle(a, ab, ac, k-1);
        draw_triangle(b, bc, ab, k-1);
        draw_triangle(c, ac, bc, k-1);
```

```
    }
    else
        triangle(a, b, c);
}

void display()
{
    GLfloat a[2] = {1.0, 1.0};
    GLfloat b[2] = {6.0, 1.0};
    GLfloat c[2] = {3.5, 5.0};

    glClear(GL_COLOR_BUFFER_BIT);

    glBegin(GL_TRIANGLES);
    draw_triangle(a, b, c, 4);
    glEnd();
    glFlush();
}

int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(0, 0);
    glutCreateWindow("Spski Gasket");
    glutDisplayFunc(display);
    myInit();
    glutMainLoop();
    return 0;
}
```

6. 3D GASKET

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

void myInit()
{
   glClearColor(1.0, 1.0, 1.0, 1.0);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   glOrtho(-50.0, 50.0, -50.0, 50.0, -50.0, 50.0);          // set viewing volume to 14 X 14 X 14
   glMatrixMode(GL_MODELVIEW);
}

void triangle(GLfloat *a, GLfloat *b, GLfloat *c)
{
   glVertex3fv(a);                                          // draw triangle using vertices a, b, c
   glVertex3fv(b);
   glVertex3fv(c);
}

void draw_triangle(GLfloat *a, GLfloat *b, GLfloat *c, GLfloat *d)
{
   glColor3f(1.0, 0.0, 0.0);                          // assign color for each of the side
   triangle(a, b, c);                                 // draw triangle between a, b, c
   glColor3f(0.0, 1.0, 0.0);
   triangle(a, b, d);
   glColor3f(0.0, 0.0, 1.0);
   triangle(a, d, c);
   glColor3f(0.0, 0.0, 0.0);
```

```
    triangle(b, c, d);
}

void draw_tetrahedra(GLfloat *a, GLfloat *b, GLfloat *c, GLfloat *d, int k)
{
    int j;
    GLfloat ab[3], bc[3], ac[3], ad[3], bd[3], cd[3];          // mid-points of tetrahedron
    if(k>0)
    {
        for(j=0;j<3;j++) ab[j] = (a[j] + b[j])/2.0;     // calculate mid-point between a and b
        for(j=0;j<3;j++) bc[j] = (b[j] + c[j])/2.0;             // calculate mid-point between b and c
        for(j=0;j<3;j++) ac[j] = (a[j] + c[j])/2.0;             // calculate mid-point between a and c
        for(j=0;j<3;j++) ad[j] = (a[j] + d[j])/2.0;             // calculate mid-point between a and d
        for(j=0;j<3;j++) bd[j] = (b[j] + d[j])/2.0;             // calculate mid-point between b and d
        for(j=0;j<3;j++) cd[j] = (c[j] + d[j])/2.0;             // calculate mid-point between c and d

        draw_tetrahedra(a, ab, ac, ad, k-1);          // draw tetrahedra between points a, ab, ac, ad
        draw_tetrahedra(ab, b, bc, bd, k-1);          // draw tetrahedra between points ab, b, bc, bd
        draw_tetrahedra(ac, bc, c, cd, k-1);          // draw tetrahedra between points ac, bc, c, cd
        draw_tetrahedra(ad, bd, cd, d, k-1);          // draw tetrahedra between points ad, bd, cd, d
    }
    else
                draw_triangle(a,b,c,d);                             // draw tetrahedra between points a, b, c, d
}

void display()
{
    GLfloat a[3] = {0.0, 16.0, 0.0},                        // co-ordinates in 3D geometry
          b[3] = {0.0, 0.0, -28.0},
          c[3] = {16.0, 28.0, 24.0},
          d[3] = {-24.0, 24.0, -16.0};
```

```
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);          // clear color buffer and depth buffer for HSR
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_TRIANGLES);
    draw_tetrahedra(a, b, c, d, 1);
    glEnd();
    glFlush();
}

int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);      // use GLUT_DEPTH if HSR is being used
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(0, 0);
    glutCreateWindow("3D Sierpinski Gasket");
    glutDisplayFunc(display);
    myInit();
    glEnable(GL_DEPTH_TEST);                                    // enable Hidden Surface Removal Algorithm
    glutMainLoop();
    return 0;
}
```

7. 3D CUBE (ROTATION)

```
/* Rotating cube with color interpolation */
```

```
/* Demonstration of use of homogeneous coordinate
transformations and simple data structure for representing
cube from Chapter 4 */

/*Both normals and colors are assigned to the vertices */
/*Cube is centered at origin so (unnormalized) normals
are the same as the vertex values */
#include <stdlib.h>
#include <GL/glut.h>

        GLfloat vertices[8][3] = {{-1.0,-1.0,-1.0},{1.0,-1.0,-1.0},
        {1.0,1.0,-1.0}, {-1.0,1.0,-1.0}, {-1.0,-1.0,1.0},
        {1.0,-1.0,1.0}, {1.0,1.0,1.0}, {-1.0,1.0,1.0}};

        /*GLfloat normals[8][3] = {{-1.0,-1.0,-1.0},{1.0,-1.0,-1.0},
        {1.0,1.0,-1.0}, {-1.0,1.0,-1.0}, {-1.0,-1.0,1.0},
        {1.0,-1.0,1.0}, {1.0,1.0,1.0}, {-1.0,1.0,1.0}}; */

        GLfloat colors[9][3] = {{0.0,0.0,0.0},{1.0,0.0,0.0},
        {1.0,1.0,0.0}, {0.0,1.0,0.0}, {0.0,0.0,1.0},
        {1.0,0.0,1.0}, {1.0,1.0,1.0}, {0.0,1.0,1.0},{1.0,0.0,1.0}};

void polygon(int a, int b, int c , int d)
{
        /* draw a polygon via list of vertices */
        glBegin(GL_POLYGON);
        glColor3fv(colors[a]);
        //glNormal3fv(normals[a]);
        glVertex3fv(vertices[a]);
        glColor3fv(colors[b]);
```

```
        //glNormal3fv(normals[b]);
        glVertex3fv(vertices[b]);
        glColor3fv(colors[c]);
        //glNormal3fv(normals[c]);
        glVertex3fv(vertices[c]);
        glColor3fv(colors[d]);
        //glNormal3fv(normals[d]);
        glVertex3fv(vertices[d]);
        glEnd();
}

void colorcube(void)
{
        /* map vertices to faces */
        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);

}

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

void display(void)
{
        /* display callback, clear frame buffer and z buffer,
   rotate cube and draw, swap buffers */
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

```
	glLoadIdentity();
	glRotatef(theta[0], 1.0, 0.0, 0.0);
	//glTranslatef(0.5,0.5,0);
	//glScalef(0.5,0.5,1);
	glRotatef(theta[1], 0.0, 1.0, 0.0);
	glRotatef(theta[2], 0.0, 0.0, 1.0);

	colorcube();

	glFlush();
	glutSwapBuffers();
}

void spinCube()
{
	/* Idle callback, spin cube 2 degrees about selected axis */
	theta[axis] += 0.1;
	if( theta[axis] > 360.0 )
		theta[axis] -= 360.0;
	/* display(); */
	glutPostRedisplay();
}

void keyboard(unsigned char key, int x, int y)
{
	/* mouse callback, selects an axis about which to rotate */
	if(key=='x' || key =='X')
		axis = 0;
	else if(key=='y' || key =='Y')
		axis = 1;
	else if(key=='z' || key =='Z')
```

```
                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);
}


main(int argc, char **argv)
{
   glutInit(&argc, argv);
   /* need both double buffering and z buffer */
   glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
   glutInitWindowSize(500, 500);
   glutCreateWindow("colorcube");
   glutReshapeFunc(myReshape);
   glutDisplayFunc(display);
   glutIdleFunc(spinCube);
   glutKeyboardFunc(keyboard);
   glEnable(GL_DEPTH_TEST); /* Enable hidden--surface--removal */
   glutMainLoop();
```

```
}
```

8. HOUSE ROTATION

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

GLfloat house[3][9]= {{150.0,150.0,225.0,300.0,300.0,200.0,200.0,250.0,250.0},
                      {150.0,350.0,450.0,350.0,150.0,150.0,200.0,200.0,150.0},
                      {1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0}};
GLfloat rot_mat[3][3]={{0},{0},{0}};
GLfloat result[3][9]={{0},{0},{0}};
GLfloat h=150.0;
GLfloat k=150.0;
GLfloat theta;

void multiply()
{
        int p,q,r;
        for(p=0;p<3;p++)
        for(q=0;q<9;q++)
        {
                result[p][q]=0;
                for(r=0;r<3;r++)
                        result[p][q]=result[p][q]+rot_mat[p][r]*house[r][q];
        }
}

void rotate()
```

```
{
        GLfloat m,n;
        m=-h*(cos(theta)-1)+k*(sin(theta));
        n=-k*(cos(theta)-1)-h*(sin(theta));
        rot_mat[0][0]=cos(theta);
        rot_mat[0][1]=-sin(theta);
        rot_mat[0][2]=m;
        rot_mat[1][0]=sin(theta);
        rot_mat[1][1]=cos(theta);
        rot_mat[1][2]=n;
        rot_mat[2][0]=0;
        rot_mat[2][1]=0;
        rot_mat[2][2]=1;
        multiply();
}

void drawhouse()
{
        glColor3f(0.0,0.0,1.0);
        glBegin(GL_LINE_LOOP);
        glVertex2f(house[0][0]+200,house[1][0]);
        glVertex2f(house[0][1]+200,house[1][1]);
        glVertex2f(house[0][3]+200,house[1][3]);
        glVertex2f(house[0][4]+200,house[1][4]);
        glEnd();
        glColor3f(0.0,0.0,1.0);
        glBegin(GL_LINE_LOOP);
        glVertex2f(house[0][5]+200,house[1][5]);
        glVertex2f(house[0][6]+200,house[1][6]);
        glVertex2f(house[0][7]+200,house[1][7]);
        glVertex2f(house[0][8]+200,house[1][8]);
```

```
        glEnd();
        glColor3f(0.0,0.0,1.0);
        glBegin(GL_LINE_LOOP);
        glVertex2f(house[0][1]+200,house[1][1]);
        glVertex2f(house[0][2]+200,house[1][2]);
        glVertex2f(house[0][3]+200,house[1][3]);
        glEnd();
}

void drawrotatedhouse()
{
        glColor3f(0.0,0.0,0.0);
        glBegin(GL_LINE_LOOP);
        glVertex2f(result[0][0]+200,result[1][0]);
        glVertex2f(result[0][1]+200,result[1][1]);
        glVertex2f(result[0][3]+200,result[1][3]);
        glVertex2f(result[0][4]+200,result[1][4]);
        glEnd();
        glColor3f(0.0,0.0,0.0);
        glBegin(GL_LINE_LOOP);
        glVertex2f(result[0][5]+200,result[1][5]);
        glVertex2f(result[0][6]+200,result[1][6]);
        glVertex2f(result[0][7]+200,result[1][7]);
        glVertex2f(result[0][8]+200,result[1][8]);
        glEnd();
        glColor3f(0.0,0.0,0.0);
        glBegin(GL_LINE_LOOP);
        glVertex2f(result[0][1]+200,result[1][1]);
        glVertex2f(result[0][2]+200,result[1][2]);
        glVertex2f(result[0][3]+200,result[1][3]);
        glEnd();
```

```
}

void display()
{
	glClear(GL_COLOR_BUFFER_BIT);
	drawhouse();
	rotate();
	drawrotatedhouse();
	glFlush();
}

void myinit()
{
	glClearColor(1.0,1.0,1.0,1.0);
	glColor3f(1.0,0.0,0.0);
	glPointSize(1.0);
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	gluOrtho2D(0.0,750.0,0.0,750.0);
}

int main(int argc,char **argv)
{
	printf("Enter the rotation angle\n");
	scanf("%f",&theta);
	theta = theta * (3.14/180.0);
	glutInit(&argc,argv);
	glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
	glutInitWindowSize(700,700);
	glutInitWindowPosition(0,0);
	glutCreateWindow("House Rotation");
```

```
	glutDisplayFunc(display);
	myinit();
	glutMainLoop();
	return 0;
}
```

9. MESH

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

int x,y;

void display()
{
	glClear(GL_COLOR_BUFFER_BIT);
	int i,j;
	for(i=100;i<=x;i+=25){
		glColor3f(0,0,0);
		for(j=100;j<=y;j+=25){
			glBegin(GL_LINE_LOOP);
			glVertex2f(i,j);
			glVertex2f(i,j+25);
			glVertex2f(i+25,j+25);
			glVertex2f(i+25,j);
			glEnd();
			glFlush();
		}
```

```
        }
        glFlush();
}

void myinit()
{
        glClearColor(1.0,1.0,1.0,1.0);
        gluOrtho2D(0.0,1000.0,0.0,1000.0);
}

int main(int argc,char **argv)
{
        printf("Enter the number of rows and columns\n");
        scanf("%d%d",&x,&y);
        glutInit(&argc,argv);
        glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
        glutInitWindowSize(700,700);
        glutInitWindowPosition(0,0);
        glutCreateWindow("House Rotation");
        glutDisplayFunc(display);
        myinit();
        glutMainLoop();
        return 0;
}
```

10. POLYGON FILLING

```
#define BLACK 0
#include<stdlib.h>
```

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

float x1,x2,x3,x4,y1,y2,y3,y4;
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(0.0,1.0,0.0);
	glPointSize(3.0);
	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=1;y<500;y++)
	{
		if(le[y]<=re[y])
			for(i=(int)le[y];i<(int)re[y];i++){
				draw_pixel(i,y);
				//sleep(1);
				glFlush();
			}
	}
```

```
}

void display()
{
	x1=100.0;y1=50.0;x2=50.0;y2=100.0;x3=100.0;y3=150.0;x4=150.0;y4=100.0;
	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();
	scanfill(x1,y1,x2,y2,x3,y3,x4,y4);
	glFlush();
}

void myinit()
{
	glClearColor(1.0,1.0,1.0,1.0);
	glColor3f(1.0,0.0,0.0);
	glPointSize(1.0);
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	gluOrtho2D(0.0,200.0,0.0,200.0);
}

int 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");
        glutDisplayFunc(display);
        myinit();
        glutMainLoop();
}
```

11. INTERACTION

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

void init()
{
   glClearColor(0, 0, 0, 0);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   gluOrtho2D(0, 500, 0, 500);
}

void circle()
{
   glClear(GL_COLOR_BUFFER_BIT);
   int r=200;
   int x=0,y=r;
   int d = 1.25-r;
   glColor3f(1,0,0);
   glBegin(GL_POLYGON);
   while(y>x){
```

```
        if(d<0)
            d+= (2*x) + 3;
        else{
            d+= (2*(x-y)) + 5;
            y--;
        }
        x++;
        glVertex2f(x+250,y+250);
        glVertex2f(-x+250,y+250);
        glVertex2f(x+250,-y+250);
        glVertex2f(-x+250,-y+250);
        glVertex2f(y+250,x+250);
        glVertex2f(-y+250,x+250);
        glVertex2f(y+250,-x+250);
        glVertex2f(-y+250,-x+250);
    }
    glEnd();
    glFlush();
}

void rectangle(){
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(0,1,0);
    glBegin(GL_POLYGON);
    glVertex2f(100,100);
    glVertex2f(100,300);
    glVertex2f(400,300);
    glVertex2f(400,100);
    glEnd();
    glFlush();
}
```

```
void triangle(){
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(0,0,1);
    glBegin(GL_POLYGON);
    glVertex2f(100,100);
    glVertex2f(250,400);
    glVertex2f(400,100);
    glEnd();
    glFlush();
}

void mymouse(int btn,int state,int x,int y)
{
    if(btn==GLUT_RIGHT_BUTTON && state==GLUT_DOWN)
        rectangle();
    else if(btn==GLUT_LEFT_BUTTON && state==GLUT_DOWN)
        triangle();
    else if(btn==GLUT_MIDDLE_BUTTON && state==GLUT_DOWN)
        circle();
}

void mykey(unsigned char k,int x,int y)
{
    if(k=='R' || k=='r')
        rectangle();
    else if(k=='t' || k=='T')
        triangle();
    else if(k=='c' || k=='C')
        circle();
}
```

```
void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glFlush();
}

int main(int argc, char**argv)
{
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500,500);
    glutInitWindowPosition(0,0);
    glutCreateWindow("Closed Objects");
    glutMouseFunc(mymouse);
    glutKeyboardFunc(mykey);
    glutDisplayFunc(display);
    init();
    glutMainLoop();
    return 0;
}
```