**EXERCISE-1**

**AIM:** Towrite a program to draw points on a plane in OpenGL.

**Description:** OpenGL (Open Graphics Library) is a cross-platform, hardware-accelerated, language-independent, industrial standard API for producing 3D (including 2D) graphics.

* glutInit: initializes GLUT, must be called before other GL/GLUT functions. It takes the same arguments as the main().
* glutInitDisplayMode: To specify the type of the display mode. It sets the initial display mode.

A geometric primitive (point, line segment etc) is enclosed within a pair glBegin and glEnd. glBegin specifies the type of geometric object, such as GL\_POINTS, GL\_LINES and GL\_POLYGON. To draw points GL\_POINTS is used. The vertices are usually specified in float precision. The function **glVertex2f** specifies the x and y coordinates of the vertex, and the z coordinate is set to zero. There is also a function **glVertex3f** that specifies all three coordinates. The "2" or "3" in the name tells how many parameters are passed to the function.

**Program:**

#include<GL/glut.h>

using namespace std;

void display()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(0.0,0.0,1.0);

glBegin(GL\_POINTS);

glVertex2f(15.0,15.0);

glVertex2f(66.0,66.0);

glVertex2f(117.0,118.0);

glVertex2f(168.0,168.0);

glVertex2f(219.0,220.0);

glVertex2f(270.0,168.0);

glVertex2f(322.0,118.0);

glVertex2f(373.0,66.0);

glVertex2f(425.0,15.0);

glEnd();

glFlush();

}

void myInit()

{

glClearColor(1.0,1.0,1.0,1.0);

glColor3f(1.0,0.0,0.0);

glPointSize(5.0);

gluOrtho2D(0.0,640.0,0.0,480.0);

}

int main(int argc,char \*\*argv)

{

glutInit(&argc,argv);

glutInitDisplayMode(GLUT\_SINGLE |GLUT\_RGB);

glutInitWindowSize(800,800);

glutInitWindowPosition(0,0);

glutCreateWindow("points");

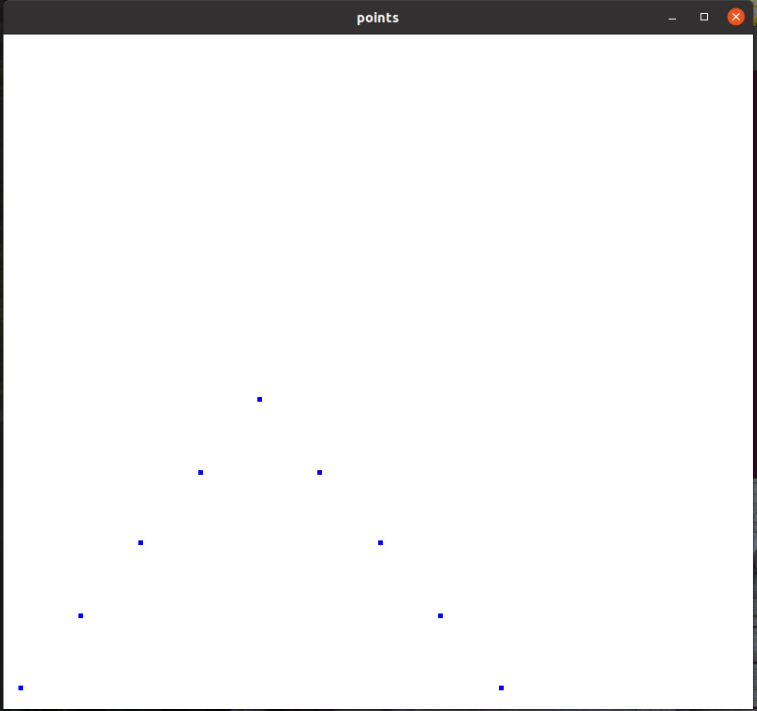
glutDisplayFunc(display);

myInit();

glutMainLoop();

}

**Output:**



**EXERCISE-2**

**AIM:** To write a program to draw a line on plane in OpenGL.

**Description:**

* glutInitWindowSize: specifies the initial window width and height, in pixels.
* glutInitWindowPosition: positions the top-left corner of the initial window at (x, y). The coordinates (x, y), in term of pixels, is measured in window coordinates, i.e., origin (0, 0) is at the top-left corner of the screen; x-axis pointing right and y-axis pointing down.
* glutCreateWindow: creates a window with the given title.

The sum of a point and a vector (or the subtraction of two points) leads to the notion of a line in an affine space. Thus, a line is infinitely long in both directions, a line segment is a finite piece of a line between two points, and a ray is infinitely long in one direction. To draw lines GL\_LINES is used.

**Program:**

#include<GL/glut.h>

using namespace std;

void Init(void)

{

glClearColor(1.0,1.0,1.0,1.0);

gluOrtho2D(0.0,300.0,0.0,200.0);

}

void display()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(0.0,1.0,0.0);

glBegin(GL\_LINES);

glVertex2i(50,50);

glVertex2i(100,150);

glVertex2i(100,150);

glVertex2i(150,50);

glVertex2i(75,100);

glVertex2i(125,100);

glEnd();

glColor3f(0.0,0.0,1.0);

glRasterPos2i(160,160);

glutBitmapCharacter(GLUT\_BITMAP\_HELVETICA\_12,'A');

glFlush();

}

int main(int argc,char \*\*argv)

{

glutInit(&argc,argv);

glutInitDisplayMode(GLUT\_SINGLE |GLUT\_RGB);

glutInitWindowSize(800,800);

glutInitWindowPosition(30,30);

glutCreateWindow("lines");

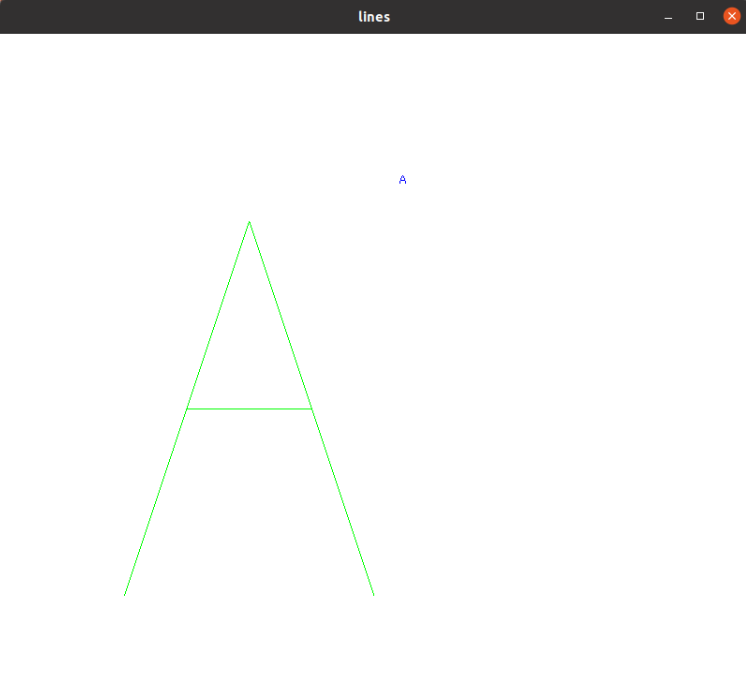
Init();

glutDisplayFunc(display);

glutMainLoop();

}

**Output:**



**EXERCISE-3**

**AIM:** To write a program to draw circle on plane in OpenGL.

**Description:**

* glutDisplayFunc: registers the callback function (or event handler) for handling window-paint event. The OpenGL graphic system calls back this handler when it receives a window re-paint request. In the example, we register the function display() as the handler.
* glutMainLoop: enters the infinite event-processing loop, i.e, put the OpenGL graphics system to wait for events (such as re-paint), and trigger respective event handlers (such as display()).
* glClearColor:specifies the red, green, blue, and alpha values used by glClear to clear the color buffers. Values specified by glClearColor are clamped to the range [0,1].
* glColor3f can be used to give each vertex it’s own color.

Circle is an eight-way symmetric figure. The shape of circle is the same in all quadrants. In each quadrant, there are two octants. If the calculation of the point of one octant is done, then the other seven points can be calculated easily by using the concept of eight-way symmetry.

**Program:**

#include<GL/glut.h>

#include<math.h>

#include<stdio.h>

using namespace std;

void draw\_pixel(GLint cx,GLint cy)

{

glColor3f(1.0,1.0,1.0);

glBegin(GL\_POINTS);

glVertex2i(cx,cy);

glEnd();

}

void plotpixels(GLint h,GLint k,GLint x,GLint y)

{

draw\_pixel(x+h,y+k);

draw\_pixel(-x+h,y+k);

draw\_pixel(x+h,-y+k);

draw\_pixel(-x+h,-y+k);

draw\_pixel(y+h,x+k);

draw\_pixel(y+h,-x+k);

draw\_pixel(-y+h,x+k);

draw\_pixel(-y+h,-x+k);

}

void circle\_draw(GLint h,GLint k,GLint r)

{

GLint d=1-r,x=0,y=r;

while(y>x)

{

plotpixels(h,k,x,y);

if(d<0)

d+=2\*x+3;

else

{

d+=2\*(x-y)+5;

y--;

}

x++;

}

plotpixels(h,k,x,y);

}

void Init()

{

glClearColor(0.0,0.0,0.0,0.0);

gluOrtho2D(0.0,400.0,0.0,300.0);

}

void display()

{

GLint xc=150,yc=150,r=50;

GLint x1c=100,y1c=100;

GLint x2c=150,y2c=50;

GLint x3c=200,y3c=100;

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(0.0,0.0,1.0);

glPointSize(2.0);

circle\_draw(xc,yc,r);

circle\_draw(x1c,y1c,r);

circle\_draw(x2c,y2c,r);

circle\_draw(x3c,y3c,r);

glFlush();

}

int main(int argc,char \*\*argv)

{

glutInit(&argc,argv);

glutInitDisplayMode(GLUT\_SINGLE |GLUT\_RGB);

glutInitWindowPosition(50,50);

glutInitWindowSize(400,300);

glutCreateWindow("Circle creation");

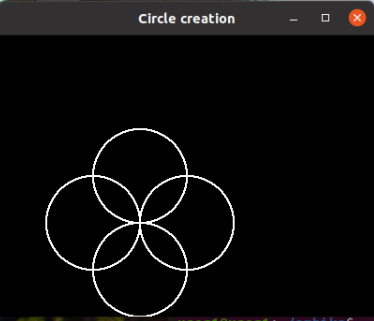
Init();

glutDisplayFunc(display);

glutMainLoop();

}

**Output:**

****

**EXERCISE-4**

**AIM:** Towrite a program draw a rectangle on a black background in OpenGL.

**Description:**

* **glMatrixMode** sets the current matrix mode. Mode can be GL\_MODELVIEW, GL\_PROJECTION, GL\_TEXTURE & GL\_COLOR.
* **glLoadIdentity** resets the current matrix to identity matrix.

The rectangle() is used to draw a rectangle. Coordinates of left top and right bottom corner are required to draw the rectangle. left specifies the X-coordinate of top left corner, top specifies the Y-coordinate of top left corner, right specifies the X-coordinate of right bottom corner, bottom specifies the Y-coordinate of right bottom corner.

**Program:**

#include<GL/glut.h>

void display()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glBegin(GL\_POLYGON);

glColor3f(0.0,1.0,0.0);

glVertex2f(-0.5,0.5);

glVertex2f(-0.75,0.5);

glVertex2f(-0.75,-0.5);

glVertex2f(-0.5,-0.5);

glEnd();

glBegin(GL\_POLYGON);

glColor3f(0.0,0.0,1.0);

glVertex2f(-0.5,-0.5);

glVertex2f(-0.5,0.5);

glVertex2f(0.5,0.5);

glVertex2f(0.5,-0.5);

glEnd();

glBegin(GL\_POLYGON);

glColor3f(0.0,1.0,0.0);

glVertex2f(0.5,0.5);

glVertex2f(0.75,0.5);

glVertex2f(0.75,-0.5);

glVertex2f(0.5,-0.5);

glEnd();

glFlush();

}

void init()

{

glClearColor(0.5,0.5,0.5,0.0);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

glOrtho(-1.0,1.0,-1.0,1.0,-1.0,1.0);

}

int main(int argc,char \*\*argv)

{

glutInit(&argc,argv);

glutInitDisplayMode(GLUT\_SINGLE |GLUT\_RGB);

glutInitWindowPosition(0,0);

glutInitWindowSize(800,500);

glutCreateWindow("rectangle");

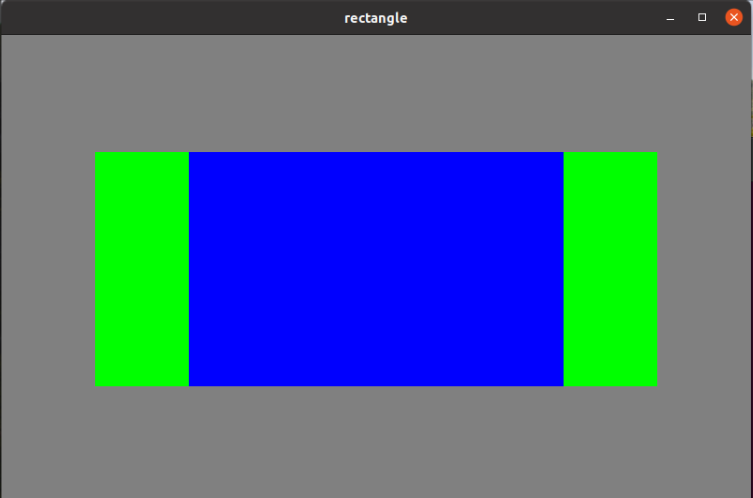
init();

glutDisplayFunc(display);

glutMainLoop();

}

**Output:**

****

**EXERCISE-5**

**AIM:** To Write a program to draw a square when we click on the mouse button in OpenGL.

**Description:** glutMouseFunc sets the mouse callback for the current window. When a user presses and releases mouse buttons in the window, each press and release generates mouse Callback the button parameter is one of GLUT\_RIGHT\_BUTTON, GLUT\_LEFT\_BUTTON, GLUT\_MIDDLE\_BUTTON. For systems with only two mouse buttons, it may not be possible to generate GLUT\_MIDDLE\_BUTTON callback. The state parameter is either GLUT\_UP or GLUT\_DOWN indicating whether the callback was due to a release or press respectively. The x and y callback parameters indicate the window relative coordinates when the mouse button state changed. If a GLUT\_DOWN callback for a specific button is triggered, the program can assume a GLUT\_UP callback for the same button will be generated when the mouse button is released even if the mouse has moved outside the window.

**Program:**

#include <GL/glut.h>

GLint wh = 500, ww = 500;

void myInit()

{

gluOrtho2D(0.0, ww, 0.0, wh);

glClearColor(1.0, 1.0, 1.0, 1.0);

glColor3f(1.0, 0.0, 0.0);

glPointSize(1.0);

}

void mouse(int btn, int state, int x, int y)

{

if (btn == GLUT\_LEFT\_BUTTON && state == GLUT\_DOWN)

{

exit(0);

}

if (btn == GLUT\_RIGHT\_BUTTON && state == GLUT\_DOWN)

{

y = wh - y;

glColor3f(0.0, 0.0, 1.0);

glBegin(GL\_POLYGON);

glVertex2f(x - 10, y - 10);

glVertex2f(x + 10, y - 10);

glVertex2f(x + 10, y + 10);

glVertex2f(x - 10, y + 10);

glEnd();

glFlush();

}

}

void display()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glFlush();

}

int main(int argc, char \*\*argv)

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowPosition(0, 0);

glutInitWindowSize(ww, wh);

glutCreateWindow("points");

myInit();

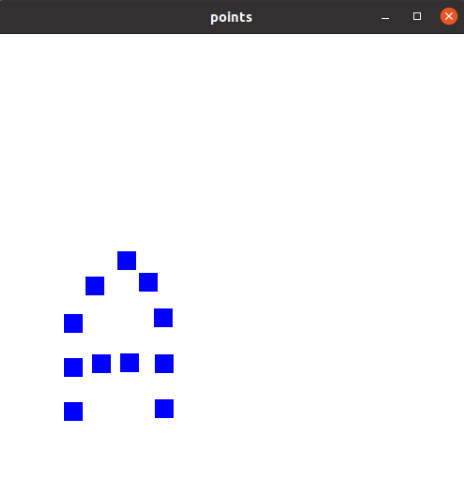
glutDisplayFunc(display);

glutMouseFunc(mouse);

glutMainLoop();

}

**Output:**

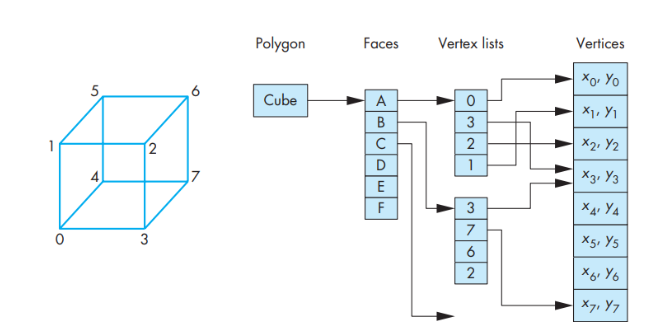


**EXERCISE-6**

**AIM:** To write a program to draw a color cube and spin it using open GL transformation matrices in OpenGL.

**Description:**

We could describe our cube through a set of vertex specifications.



The data specifying the location of the vertices contain the geometry and can be stored as a simple list or array, such as in vertices[8]—the vertex list. The top- level entity is a cube; we regard it as being composed of six faces. Each face consists of four ordered vertices. Each vertex can be specified indirectly through its index. This data structure is shown in above figure.

Render the cube using the array of vertices. Obtain the co-ordinates of the cube with the origin at the center of the cube. Draw the cube using three points. Increment the angle w.r.t current axis by specified value. Display the cube on the screen and rotate by the current value of angle around current axis. Repeat the process until mouse event occurs.

**Program:**

#include<GL/glut.h>

GLfloat vertices[][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[][3] ={{0.0,0.0,1.0},{1.0,1.0,1.0},{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}};

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(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)

{

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();

glFlush();

glutSwapBuffers();

}

void spincube()

{

theta[axis]+=10.0;

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 myInit()

{

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

glOrtho(-2.0,2.0,-2.0,2.0,-10.0,10.0);

glMatrixMode(GL\_MODELVIEW);

}

int main(int argc,char \*\*argv)

{

glutInit(&argc,argv);

glutInitDisplayMode(GLUT\_DEPTH | GLUT\_RGB | GLUT\_DOUBLE);

glutInitWindowSize(500,500);

glutCreateWindow("rotating a color cube");

myInit();

glutDisplayFunc(display);

glutIdleFunc(spincube);

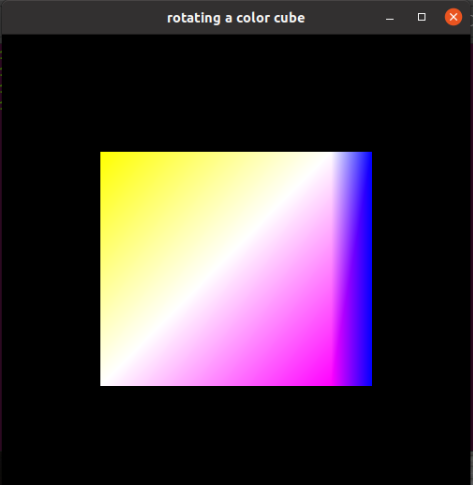
glutMouseFunc(mouse);

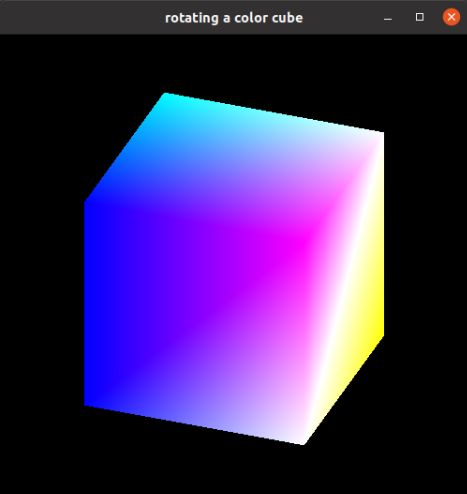
glEnable(GL\_DEPTH\_TEST);

glutMainLoop();

}

**Output:**





**EXERCISE-7**

**AIM:** To Write a program to create a house like figure and rotate it about a given fixed point using OpenGL functions in OpenGL.

**Description:**

* **Translation** is an operation that displaces points by a fixed distance in a given direction. To specify a translation, we need only to specify a displacement vector d, because the transformed points are given by P**`**=P+d for all points P on the object. OpenGL function is glTranslate(tx,ty,tz).
* **Rotation** is expressed as rotation through angle θ about an axis direction (x,y,z). OpenGL function is glRotatef(θ, x, y, z).
* **Scaling** is an affine non–rigid-body transformation by which we can make an object bigger or smaller. OpenGL function is glScale(sx,sy,sz).

This program is to create a house like figure and rotate it about a given fixed point using mouse callbacks in OpenGL.

**Program:**

#include<GL/glut.h>

using namespace std;

void house()

{

glColor3f(0.0,1.0,0.0);

glBegin(GL\_POLYGON);

glVertex2f(25,10);

glVertex2f(25,20);

glVertex2f(35,20);

glVertex2f(35,10);

glEnd();

glColor3f(1.0,1.0,0.0);

glBegin(GL\_POLYGON);

glVertex2f(10,10);

glVertex2f(50,10);

glVertex2f(50,50);

glVertex2f(10,50);

glEnd();

glColor3f(0.0,1.0,0.0);

glBegin(GL\_POLYGON);

glVertex2f(10,50);

glVertex2f(50,50);

glVertex2f(30,70);

glEnd();

}

void display(void)

{

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

house();

glFlush();

glutSwapBuffers();

}

void mouse(int btn,int state,int x,int y)

{

if(btn==GLUT\_LEFT\_BUTTON && state==GLUT\_DOWN)

{

glutDisplayFunc(display);

glTranslatef(35,35,0);

glRotatef(-45,0.0,0.0,1.0);

glTranslatef(-35,-35,0);

glutPostRedisplay();

}

}

void myinit()

{

glClearColor(1.0,1.0,1.0,1.0);

gluOrtho2D(-100,100,-100,100);

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

}

int main(int argc,char \*\*argv)

{

glutInit(&argc,argv);

glutInitDisplayMode(GLUT\_DEPTH | GLUT\_RGB | GLUT\_DOUBLE);

glutInitWindowSize(500,500);

glutCreateWindow("house");

myinit();

glutMouseFunc(mouse);

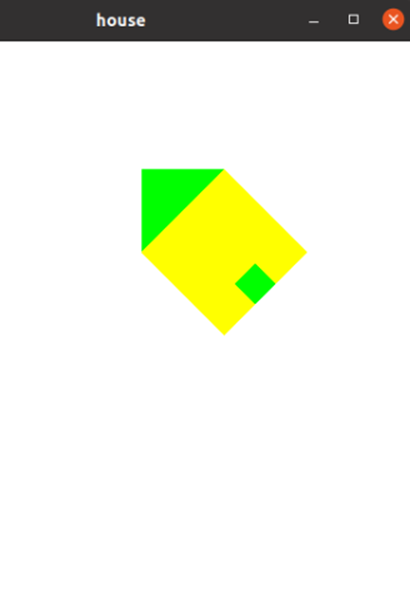
glutDisplayFunc(display);

glEnable(GL\_DEPTH\_TEST);

glutMainLoop();

}

**Output:**



**EXERCISE-8**

**AIM:** To Write a program to implement the Cohen-Sutherland line clipping algorithm. Make provision to specify the input line, window for clipping and viewport for displaying the clipped image in OpenGL.

**Description:** The Cohen-Sutherland algorithm works best when there are many line segments but few are actually displayed. In this case, most of the line segments lie fully outside one or two of the extended sides of the clipping rectangle and can thus be eliminated on the basis of their out codes. The other advantage is that this algorithm can be extended to three dimensions. The main disadvantage of the algorithm is that it must be used recursively.

**Program:**

#include<stdio.h>

#include<GL/glut.h>

#define outcode int

double xmin=50,ymin=50,xmax=100,ymax=100;// Windows boundaries

double xvmin=200,yvmin =200, xvmax=300,yvmax=300; // Viewport boundaries

const int RIGHT= 8; // bit codes for the right

const int LEFT =2; //bit codes for the left

const int TOP=4; // bit codes for the top

const int BOTTOM=1; //bit codes for the bottom

outcode ComputeOutCode(double x,double y); // used to compute bit codes of a point

// Cohen -Sutherland clipping algorithm clips a line from

// p0=(x0,y0) to p1 =(x1,y1) against a rectangle with.

// diagonal from (xmin,ymin)to (xmax,ymax)

void CohenSutherlandLineClipAnddraw(double x0,double y0,double x1,double y1)

{

//OutCodes for P0 ,P! and Whatever point lines outside the clip rectangle

outcode outcode0,outcode1,outcodeOut;

int accept =0,done =0;

//compute outcodes

outcode0= ComputeOutCode(x0,y0);

outcode1= ComputeOutCode(x1,y1);

do

{

if(!(outcode0|outcode1)) // logical or is 0 trivially accept and exit

{ accept=1;

done=1;

}

else

if(outcode0 & outcode1) // logical and is 0 trivially reject and exit

done=1;

else

{

//failed both tests , so calculate the line segment clip;

// from an outside point to an intersection with clip edge

double x,y;

// at least one endpoint is outside the clip rectangle ; pick it.

outcodeOut= outcode0?outcode0:outcode1;

//now find the intersection point ; slope m= (y1-y0)/(x1-x0)

// use formula y=y0+slope\*(x-x0),x=x0+(1/slope)\*(y-y0)

if(outcodeOut & TOP) //point is above the clip rectangle

{

x= x0+(x1-x0)\*(ymax-y0)/(y1-y0);

y=ymax;

}

else

if(outcodeOut & BOTTOM) //point is below the clip rectangle

{

x= x0+(x1-x0)\*(ymin-y0)/(y1-y0);

y=ymin;

}

else

if(outcodeOut & RIGHT) //point is to the right of clip rectangle

{

y= y0+(y1-y0)\*(xmax-x0)/(x1-x0);

x=xmax;

}

else //point is to the left of the clip rectangle

{

y= y0+(y1-y0)\*(xmin-x0)/(x1-x0);

x=xmin;

}

// now we move outside point to intersection point to clip

// and get ready for next pass.

if(outcodeOut == outcode0) // If the outside point was p0 update x0,y0 to x,y

{

x0=x;

y0=y;

outcode0 = ComputeOutCode(x0,y0);

}

else // If the outside point was p1 update x1,y1 to x,y

{

x1=x;

y1=y;

outcode1 = ComputeOutCode(x1,y1);

}

}

}while(!done);

if(accept)

{

// window to viewport mapping

double sx=(xvmax-xvmin)/(xmax-xmin);// scale parameter in x direction

double sy=(yvmax-yvmin)/(ymax-ymin);// scale parameter in y direction

double vx0 = xvmin+(x0-xmin)\*sx;

double vy0 = yvmin+(y0-ymin)\*sy;

double vx1 = xvmin+(x1-xmin)\*sx;

double vy1 = yvmin+(y1-ymin)\*sy;

//draw a red color viewport

glColor3f(1.0,0.0,0.0);

glBegin(GL\_LINE\_LOOP);

glVertex2f(xvmin,yvmin);

glVertex2f(xvmax,yvmin);

glVertex2f(xvmax,yvmax);

glVertex2f(xvmin,yvmax);

glEnd();

glColor3f(0.0,0.0,1.0);

glBegin(GL\_LINES);

glVertex2d(vx0,vy0);

glVertex2d(vx1,vy1);

glEnd();

}

}

// compute the bit code for a point (x,y) using the clip rectangle

// bounded diagonally by (xmin,ymin) and (xmax,ymax)

outcode ComputeOutCode(double x,double y)

{

outcode code =0;

if(y>ymax) //above the clip window

code |=TOP;

if(y<ymin) //below the clip window

code |=BOTTOM;

if(x>xmax) //to the right of the clip window

code |=RIGHT;

if(x<xmin) //to the left of the clip window

code |=TOP;

return code;

}

void display()

{

double x0=120,y0=10,x1=40,y1=130;

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(1.0,0.0,0.0); // draw red color lines

glBegin(GL\_LINES);

glVertex2d(x0,y0);

glVertex2d(x1,y1);

glVertex2d(60,20);

glVertex2d(80,120);

glEnd();

glColor3f(0.0,0.0,1.0); // draw a blue colored window

glBegin(GL\_LINE\_LOOP);

glVertex2f(xmin,ymin);

glVertex2f(xmax,ymin);

glVertex2f(xmax,ymax);

glVertex2f(xmin,ymax);

glEnd();

CohenSutherlandLineClipAnddraw(x0,y0,x1,y1);

CohenSutherlandLineClipAnddraw(60,20,80,120);

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(0.0,499.0,0.0,499.0);

}

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");

glutDisplayFunc(display);

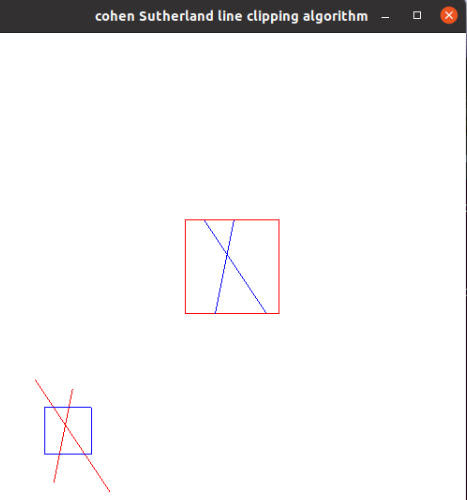
myinit();

glutMainLoop();

return 0;

}

**Output:**



**EXERCISE-9**

**AIM:** To Write a program to fill any given polygon using scan line area filling algorithm in OpenGL.

**Description:** Scanline filling is basically filling up of polygons using horizontal lines or scanlines. The purpose of the SLPF algorithm is to fill (color) the interior pixels of a polygon given only the vertices of the figure. To understand Scanline, think of the image being drawn by a single pen starting from bottom left, continuing to the right, plotting only points where there is a point present in the image, and when the line is complete, start from the next line and continue. This algorithm works by intersecting scanline with polygon edges and fills the polygon between pairs of intersections.

**Program:**

#include <stdlib.h>

#include <stdio.h>

#include<GL/glut.h>

GLvoid scanfill(float x1,float y1,float x2,float y2,float x3,float y3,float x4,float y4);

GLint k=0,m,n;

typedef struct

{

GLfloat x,y;

}point;

point p[250000];

void edgedetect(float x1,float y1,float x2,float y2)

{

GLfloat mx,my,temp,a,b;

GLint 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;

my=y2-y1;

printf("the change in delta x is %f and delta y are %f \n",mx,my);

for(i=0;i<=my;i++)

{

a=x1+i\*mx;

b=y1+i;

p[k].x=(x1+i\*mx);

p[k].y=y1+i;

k++;

}

}

void scanfill()

{

GLfloat x1=200,y1=10,x2=100,y2=250,x4=200,y4=150,x3=300,y3=400;

bool flag;

GLfloat sx[500][500] ;

GLint count[500]={0};

GLint c;

printf("k value is %d",k);

edgedetect(x1,y1,x2,y2);

edgedetect(x2,y2,x3,y3);

edgedetect(x3,y3,x4,y4);

edgedetect(x4,y4,x1,y1);

for(int m=0;m<500;m++)

{

c=0;

printf("\nforscanline %d x intercepts are::",m);

for(int n=0;n<k;n++)

{

if(p[n].y==m)

{

sx[c][m]=p[n].x;

count[m]=c;

printf("%f\t",sx[c][m]);

c++;

}

}

// printf("scan line is %d",m);

}

for(int m=0;m<500;m++)

{

if(count[m]>0)

{

for(int j=0;j<=count[m];j=j+2)

{

for(int z=sx[j][m];z<=sx[j+1][m];z++)

{

printf("(\t%d,%d \n)",z,m);;

glBegin(GL\_POINTS); glVertex2f(z,m); glEnd();

}

}

}

printf("scanline %d has finished\n",m);

}

}

void display(void)

{

glClear (GL\_COLOR\_BUFFER\_BIT);

glColor3f (1.0, 0.0, 0.0);

scanfill();

glFlush();

}

void myinit()

{

glClearColor(1.0,1.0,1.0,0.0);

glColor3f(1.0,0.0,0.0);

glPointSize(1.0);

gluOrtho2D(0.0,500,0.0,500.0);

}

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");

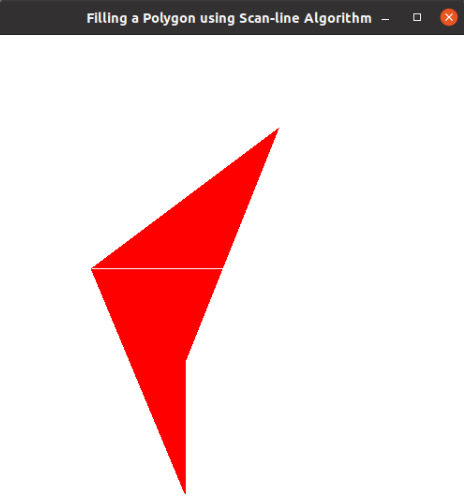
glutDisplayFunc(display);

myinit();

glutMainLoop();

}

**Output:**

****

**EXERCISE-10**

**AIM:** To implement 2D-Gasket and 3D-Gasket in OpenGL.

**Description:** Drawing of the gasket-an interesting shape that has a long history and is of interest in areas such as fractal geometry. The gasket is an object that can be defined recursively and randomly; in the limit, however, it has properties that are not at all random. We start with a two-dimension version, but as we will later, the three-dimensional version is almost identical. Suppose that we start with three points in space. As long as the points are Not collinear, they are the vertices of a unique triangle and also define a unique plane. We assume that this plane is the plane z = 0 and that these points, as specified in some convenient coordinate system,1 are (x1,y1,0), (x2,y2,0 ), and (x3,y3,0) the construction proceeds as follows:

Gasket algorithm:

1. Pick an initial point (x, y, z) at random inside the triangle.

2. Select one of the three vertices at random.

3. Find the location halfway between the initial point and the randomly selected vertex.

4. Display this new point by putting some sort of marker, such as a small circle, at the corresponding location on the display.

5. Replace the point at (x, y, z) with this new point.

6. Return to step 2

**Program:**

2D-Gasket

#include<GL/glut.h>

void display(void)

{

glClear ( GL\_COLOR\_BUFFER\_BIT );

glColor3f (0.0, 1.0, 0.0);

glBegin(GL\_LINES);

glVertex3f (0.0, 0.5,0.0);

glVertex3f (0.5, 0.0,0.0);

glEnd();

glColor3f (1.0, 1.0, 0.0);

glBegin(GL\_LINES);

glVertex3f (0.0, 0.5,0.0);

glVertex3f (-0.5, 0.0,0.0);

glEnd();

glColor3f (1.0, 0.0, 1.0);

glBegin(GL\_LINES);

glVertex3f (-0.5, 0.0,0.0);

glVertex3f (0.5, 0.0,0.0);

glEnd();

typedef GLfloat point2[3];

point2 vertices[3] = {{0.0,0.5,0.0},{-0.5, 0.0,0.0},{0.5,0.0,0.0}};

int i,j, k;

point2 p = {0.0, 0.25,0.0};

for(k = 0; k <= 4000;k++)

{

j = rand()%3;

if(j==0)

glColor3f (0.0, 1.0, 0.0);

else if(j==1)

glColor3f (1.0, 1.0, 0.0);

else

glColor3f (1.0, 0.0, 1.0);

p[0] = (p[0] + vertices[j][0])/2.0;

p[1] = (p[1] + vertices[j][1])/2.0;

p[2]=0.0;

glBegin(GL\_POINTS);

glVertex3fv(p);

glEnd();

}

glFlush();

}

int main(int argc, char \*\*argv)

{

glutInit(&argc,argv);

glutInitDisplayMode (GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(500, 500);

glutInitWindowPosition(0, 0);

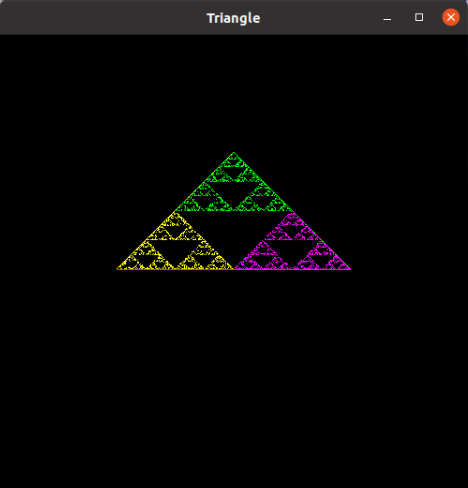
glutCreateWindow("Triangle");

glutDisplayFunc(display);

glutMainLoop();

}

**Output:**



3D-Gasket

#include <GL/glut.h>

#include <math.h>

void display(void)

{

typedef GLfloat point3[3];

point3 vertices[4] = {{25.0,25.0,25.0},{25.0, 75.0, 25.0},{75.0,25, 25},{75,75,75}};

int i,j, k;

point3 p = {35, 35, 35};

glClear (GL\_COLOR\_BUFFER\_BIT);

/\* Compute and display 1000 new points \*/

for(k = 0; k <= 10000;k++)

{

j = rand()%4; /\* pick vertex at random \*/

/\* Compute halfway point between vertex and p \*/

p[0] = (p[0] + vertices[j][0])/2.0;

p[1] = (p[1] + vertices[j][1])/2.0;

p[2] = (p[2] + vertices[j][2])/2.0;

/\* plot the point \*/

glBegin(GL\_POINTS);

glColor3f(p[0]/100,p[1]/100,p[2]/100);

glVertex3fv(p);

glEnd();

}

/\* don't wait!

\* start processing buffered OpenGL routines

\*/

glFlush ();

}

void init (void)

{

/\* select clearing color \*/

glClearColor (0.0, 0.0, 1.0, 0.0);

/\* initialize viewing values \*/

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

glOrtho(0.0, 100.0, 0.0, 100.0, -100.0, 100.0);

glMatrixMode(GL\_MODELVIEW);

}

/\*

\* Declare initial window size, position, and display mode

\* (single buffer and RGBA). Open window with "Sierpinski Gasket"

\* in its title bar. Call initialization routines.

\* Register callback function to display graphics.

\* Enter main loop and process events.

\*/

int main(int argc, char\*\* argv)

{

glutInit(&argc, argv);

glutInitDisplayMode (GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize (600, 600);

glutInitWindowPosition (100, 100);

glutCreateWindow ("3D Gasket");

init ();

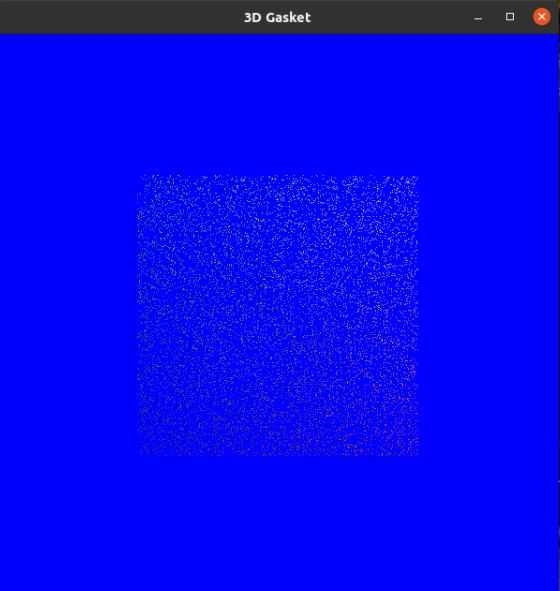
glutDisplayFunc(display);

glutMainLoop();

return 0; /\* ANSI C requires main to return int. \*/

}

**Output:**



**EXERCISE-11**

**AIM:** To create and execute a Display List in OpenGL.

**Description:** Display list is created in the init() routine. This display list contains OpenGL commands to draw a red triangle. Then in the display() routine, the display list is executed ten times. In addition, a line is drawn in immediate mode. Note that the display list allocates memory to store the commands and the values of any necessary variables.

**Program:**

#include <GL/gl.h>

#include <GL/glu.h>

#include <GL/glut.h>

#include <stdlib.h>

GLuint listName;

static void init (void)

{

listName = glGenLists (1);

glNewList (listName, GL\_COMPILE);

glColor3f (0.0, 1.0, 1.0); /\* current color red \*/

glBegin (GL\_TRIANGLES);

glVertex2f (0.0, 0.0);

glVertex2f (1.0, 0.0);

glVertex2f (0.0, 1.0);

glEnd ();

glTranslatef (1.5, 0.0, 0.0); /\* move position \*/

glEndList ();

glShadeModel (GL\_FLAT);

}

static void drawLine (void)

{

glBegin (GL\_LINES);

glVertex2f (0.0, 0.5);

glVertex2f (15.0, 0.5);

glEnd ();

}

void display(void)

{

GLuint i;

glClear (GL\_COLOR\_BUFFER\_BIT);

glColor3f (0.0, 1.0, 0.0); /\* current color green \*/

for (i = 0; i < 10; i++) /\* draw 10 triangles \*/

glCallList (listName);

drawLine (); /\* is this line green? NO! \*/

/\* where is the line drawn? \*/

glFlush ();

}

void reshape(int w, int h)

{

glViewport(0, 0, w, h);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

if (w <= h)

gluOrtho2D (0.0, 2.0, -0.5 \* (GLfloat)h/(GLfloat)w, 1.5 \* (GLfloat)h/(GLfloat)w);

else

gluOrtho2D (0.0, 2.0\*(GLfloat)w/(GLfloat)h, -0.5, 1.5);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

}

void keyboard(unsigned char key, int x, int y)

{

switch (key) {

case 27:

exit(0);

}

}

int main(int argc, char\*\* argv)

{

glutInit(&argc, argv);

glutInitDisplayMode (GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(650, 50);

glutCreateWindow(argv[0]);

init ();

glutReshapeFunc (reshape);

glutKeyboardFunc (keyboard);

glutDisplayFunc (display);

glutMainLoop();

return 0;

}

**Output:**



**EXERCISE-12**

**AIM:** To implement Picking and Selection in OpenGL.

**Description:**

**Picking** operation is really just a special case of selection. A special picking matrix is used in conjunction with the projection matrix to restrict drawing to a small region of the viewport, typically near the cursor. Some form of input, such as clicking a mouse button, initiates pick mode. With selection mode set, together with the special picking matrix, objects drawn near the cursor cause selection hits. Thus, during picking you are typically determining which objects are drawn near the cursor.

**Selection** means determining which graphical objects are within some specific volume of space. In selection, information about what is drawn is returned to the application rather than being placed in the framebuffer. In selection, the scene is first drawn into the framebuffer, then selection mode is entered and the scene is redrawn. While in selection mode, the contents of the framebuffer remain unchanged. When selection mode exits, a list of the objects that intersect the viewing volume (typically a different volume) is returned. The list returned the hit record is an array of integer identifiers corresponding to entries in the name stack.

**Program:**

PICKING

#include <GL/gl.h>

#include <GL/glu.h>

#include <GL/glut.h>

#include <stdlib.h>

#include <stdio.h>

void init(void)

{

glClearColor(0.0, 0.0, 0.0, 0.0);

glEnable(GL\_DEPTH\_TEST);

glShadeModel(GL\_FLAT);

glDepthRange(0.0, 1.0); /\* The default z mapping \*/

}

void drawRects(GLenum mode)

{

if (mode == GL\_SELECT)

glLoadName(1);

glBegin(GL\_QUADS);

glColor3f(1.0, 1.0, 0.0);

glVertex3i(2, 0, 0);

glVertex3i(2, 6, 0);

glVertex3i(6, 6, 0);

glVertex3i(6, 0, 0);

glEnd();

if (mode == GL\_SELECT)

glLoadName(2);

glBegin(GL\_QUADS);

glColor3f(0.0, 1.0, 1.0);

glVertex3i(3, 2, -1);

glVertex3i(3, 8, -1);

glVertex3i(8, 8, -1);

glVertex3i(8, 2, -1);

glEnd();

if (mode == GL\_SELECT)

glLoadName(3);

glBegin(GL\_QUADS);

glColor3f(1.0, 0.0, 1.0);

glVertex3i(0, 2, -2);

glVertex3i(0, 7, -2);

glVertex3i(5, 7, -2);

glVertex3i(5, 2, -2);

glEnd();

}

void processHits(GLint hits, GLuint buffer[])

{

unsigned int i, j;

GLuint names, \*ptr;

printf("hits = %d\n", hits);

ptr = (GLuint \*) buffer;

for (i = 0; i < hits; i++) { /\* for each hit \*/

names = \*ptr;

printf(" number of names for hit = %d\n", names); ptr++;

printf(" z1 is %g;", (float) \*ptr/0x7fffffff); ptr++;

printf(" z2 is %g\n", (float) \*ptr/0x7fffffff); ptr++;

printf(" the name is ");

for (j = 0; j < names; j++) { /\* for each name \*/

printf("%d ", \*ptr); ptr++;

}

printf("\n");

}

}

#define BUFSIZE 512

void pickRects(int button, int state, int x, int y)

{

GLuint selectBuf[BUFSIZE];

GLint hits;

GLint viewport[4];

if (button != GLUT\_LEFT\_BUTTON || state != GLUT\_DOWN)

return;

glGetIntegerv(GL\_VIEWPORT, viewport);

glSelectBuffer(BUFSIZE, selectBuf);

(void) glRenderMode(GL\_SELECT);

glInitNames();

glPushName(0);

glMatrixMode(GL\_PROJECTION);

glPushMatrix();

glLoadIdentity();

/\* create 5x5 pixel picking region near cursor location \*/

gluPickMatrix((GLdouble) x, (GLdouble) (viewport[3] - y),

5.0, 5.0, viewport);

glOrtho(0.0, 8.0, 0.0, 8.0, -0.5, 2.5);

drawRects(GL\_SELECT);

glPopMatrix();

glFlush();

hits = glRenderMode(GL\_RENDER);

processHits(hits, selectBuf);

}

void display(void)

{

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

drawRects(GL\_RENDER);

glFlush();

}

void reshape(int w, int h)

{

glViewport(0, 0, (GLsizei) w, (GLsizei) h);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

glOrtho(0.0, 8.0, 0.0, 8.0, -0.5, 2.5);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

}

int main(int argc, char \*\*argv)

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB | GLUT\_DEPTH);

glutInitWindowSize (200, 200);

glutInitWindowPosition (100, 100);

glutCreateWindow(argv[0]);

init();

glutMouseFunc(pickRects);

glutReshapeFunc(reshape);

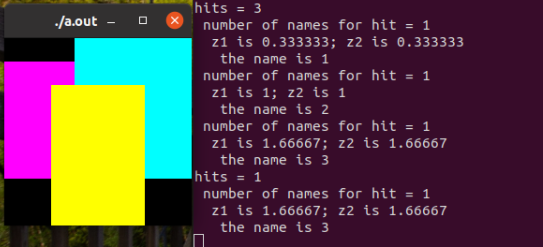
glutDisplayFunc(display);

glutMainLoop();

return 0;

}

**Output:**



SELECTION

#include <GL/gl.h>

#include <GL/glu.h>

#include <GL/glut.h>

#include <stdlib.h>

#include <stdio.h>

void drawTriangle (GLfloat x1, GLfloat y1, GLfloat x2,

GLfloat y2, GLfloat x3, GLfloat y3, GLfloat z)

{

glBegin (GL\_TRIANGLES);

glVertex3f (x1, y1, z);

glVertex3f (x2, y2, z);

glVertex3f (x3, y3, z);

glEnd ();

}

void drawViewVolume (GLfloat x1, GLfloat x2, GLfloat y1,

GLfloat y2, GLfloat z1, GLfloat z2)

{

glColor3f (1.0, 1.0, 1.0);

glBegin (GL\_LINE\_LOOP);

glVertex3f (x1, y1, -z1);

glVertex3f (x2, y1, -z1);

glVertex3f (x2, y2, -z1);

glVertex3f (x1, y2, -z1);

glEnd ();

glBegin (GL\_LINE\_LOOP);

glVertex3f (x1, y1, -z2);

glVertex3f (x2, y1, -z2);

glVertex3f (x2, y2, -z2);

glVertex3f (x1, y2, -z2);

glEnd ();

glBegin (GL\_LINES); /\* 4 lines \*/

glVertex3f (x1, y1, -z1);

glVertex3f (x1, y1, -z2);

glVertex3f (x1, y2, -z1);

glVertex3f (x1, y2, -z2);

glVertex3f (x2, y1, -z1);

glVertex3f (x2, y1, -z2);

glVertex3f (x2, y2, -z1);

glVertex3f (x2, y2, -z2);

glEnd ();

}

void drawScene (void)

{

glMatrixMode (GL\_PROJECTION);

glLoadIdentity ();

gluPerspective (40.0, 4.0/3.0, 1.0, 100.0);

glMatrixMode (GL\_MODELVIEW);

glLoadIdentity ();

gluLookAt (7.5, 7.5, 12.5, 2.5, 2.5, -5.0, 0.0, 1.0, 0.0);

glColor3f (0.0, 1.0, 0.0); /\* green triangle \*/

drawTriangle (2.0, 2.0, 3.0, 2.0, 2.5, 3.0, -5.0);

glColor3f (1.0, 0.0, 0.0); /\* red triangle \*/

drawTriangle (2.0, 7.0, 3.0, 7.0, 2.5, 8.0, -5.0);

glColor3f (1.0, 1.0, 0.0); /\* yellow triangles \*/

drawTriangle (2.0, 2.0, 3.0, 2.0, 2.5, 3.0, 0.0);

drawTriangle (2.0, 2.0, 3.0, 2.0, 2.5, 3.0, -10.0);

drawViewVolume (0.0, 5.0, 0.0, 5.0, 0.0, 10.0);

}

void processHits (GLint hits, GLuint buffer[])

{

unsigned int i, j;

GLuint names, \*ptr;

printf ("hits = %d\n", hits);

ptr = (GLuint \*) buffer;

for (i = 0; i < hits; i++) { /\* for each hit \*/

names = \*ptr;

printf (" number of names for hit = %d\n", names); ptr++;

printf(" z1 is %g;", (float) \*ptr/0x7fffffff); ptr++;

printf(" z2 is %g\n", (float) \*ptr/0x7fffffff); ptr++;

printf (" the name is ");

for (j = 0; j < names; j++) { /\* for each name \*/

printf ("%d ", \*ptr); ptr++;

}

printf ("\n");

}

}

#define BUFSIZE 512

void selectObjects(void)

{

GLuint selectBuf[BUFSIZE];

GLint hits;

glSelectBuffer (BUFSIZE, selectBuf);

(void) glRenderMode (GL\_SELECT);

glInitNames();

glPushName(0);

glPushMatrix ();

glMatrixMode (GL\_PROJECTION);

glLoadIdentity ();

glOrtho (0.0, 5.0, 0.0, 5.0, 0.0, 10.0);

glMatrixMode (GL\_MODELVIEW);

glLoadIdentity ();

glLoadName(1);

drawTriangle (2.0, 2.0, 3.0, 2.0, 2.5, 3.0, -5.0);

glLoadName(2);

drawTriangle (2.0, 7.0, 3.0, 7.0, 2.5, 8.0, -5.0);

glLoadName(3);

drawTriangle (2.0, 2.0, 3.0, 2.0, 2.5, 3.0, 0.0);

drawTriangle (2.0, 2.0, 3.0, 2.0, 2.5, 3.0, -10.0);

glPopMatrix ();

glFlush ();

hits = glRenderMode (GL\_RENDER);

processHits (hits, selectBuf);

}

void init (void)

{

glEnable(GL\_DEPTH\_TEST);

glShadeModel(GL\_FLAT);

}

void display(void)

{

glClearColor (0.0, 0.0, 0.0, 0.0);

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

drawScene ();

selectObjects ();

glFlush();

}

int main(int argc, char\*\* argv)

{

glutInit(&argc, argv);

glutInitDisplayMode (GLUT\_SINGLE | GLUT\_RGB | GLUT\_DEPTH);

glutInitWindowSize (200, 200);

glutInitWindowPosition (100, 100);

glutCreateWindow (argv[0]);

init();

glutDisplayFunc(display);

glutMainLoop();

return 0;

}

**Output:**

