* **Open the text editor using command and type following code:**

**gedit circle.cpp (Here circle.cpp is your file name)**

* **Here’s a standard command to compile a basic OpenGL program using GL, GLU, and GLUT:**

**g++ your\_program.cpp -o c -lGL -lGLU -lglut**

* **Command for running code :**

**./c**

**2. Implement DDA and Bresenham line drawing algorithm to draw: i) Simple Line ii) Dotted Line iii) Dashed Line iv) Solid line; using mouse interface Divide the screen in four quadrants with center as (0, 0). The line should work for all the slopes positive as well as negative.**

#include<GL/glut.h>

#include<GL/glu.h>

#include<iostream>

#include<math.h>

#define h 700

#define w 700

using namespace std;

GLint xi,xii,yi,yii;

void setpixel(GLint x, GLint y)

{

glColor3f(0.0,0.0,1.0);

glBegin(GL\_POINTS);

glVertex2f(x,y);

glEnd();

glFlush();

}

void initialize()

{

glClearColor(0.6,0.6,0.6,0.0);

glClear(GL\_COLOR\_BUFFER\_BIT);

//glColor3f(1.0f,0.0f,0.0f);

//glPointSize(4.0);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(-w/2,w/2,-h/2,h/2);

}

void choice()

{

int i;

glPointSize(2.0);

for(i=-w; i<w; i++)

{

setpixel(0,i);

setpixel(i,0);

}

}

class line

{

public:

void dda(int item)

{

GLfloat dx,dy,step,x,y;

GLfloat xinc,yinc;

int i;

dx=xii-xi;

dy=yii-yi;

if(abs(dx)>=abs(dy))

step=abs(dx);

else

step=abs(dy);

xinc=(float)dx/step;

yinc=(float)dy/step;

x=xi;

y=yi;

setpixel(x,y);

for(i=1; i<=step; i++)

{

x=x+xinc;

y=y+yinc;

xi=x+0.5;

yi=y+0.5;

if(item==1)

{

setpixel(xi,yi);

}

if(item==2)

{

if(i%10<5)

{

setpixel(xi,yi);

}

}

if(item==3)

{

if(i%9>=2 && i%9!=7)

{

setpixel(xi,yi);

}

}

if(item==4)

{

glPointSize(4.0);

setpixel(xi,yi);

}

}

}

void bresenham(int item)

{

int dx,dy,P,tmp;

int i=1;

if(xii<xi && yii<yi)

{

tmp=xi;

xi=xii;

xii=tmp;

tmp=yi;

yi=yii;

yii=tmp;

}

dx=(xii-xi);

dy=(yii-yi);

if(dy<=dx&&dy>0)

{

dx=abs(dx);

dy=abs(dy);

P=(2\*dy)-dx;

setpixel(xi,yi);

int x=xi;

int y=yi;

while(x<=xii)

{

x++;

if(P<0)

{

P=P+(2\*dy);

}

else

{

y++;

P=P+(2\*dy)-(2\*dx);

}

if(item==1)

{

setpixel(x,y);

}

if(item==2 && i%10<5)

{

setpixel(x,y);

}

if(item==3 && (i%9>=2 && i%9!=7))

{

setpixel(x,y);

}

if(item==4)

{

glPointSize(4.0);

setpixel(x,y);

}

i++;

}

}

else if(dy>dx&&dy>0)

{

dx=abs(dx);

dy=abs(dy);

P=(2\*dx)-dy;

setpixel(xi,yi);

int x=xi;

int y=yi;

while(y<=yii)

{

y++;

if(P<0)

{

P=P+(2\*dx);

}

else

{

x++;

P=P+(2\*dx)-(2\*dy);

}

if(item==1)

{

setpixel(x,y);

}

if(item==2 && i%10<5)

{

setpixel(x,y);

}

if(item==3 && (i%9>=2 && i%9!=7))

{

setpixel(x,y);

}

if(item==4)

{

glPointSize(4.0);

setpixel(x,y);

}

i++;

}

}

else if(dy>=-dx)

{

dx=abs(dx);

dy=abs(dy);

P=(2\*dy)-dx;

setpixel(xi,yi);

int x=xi;

int y=yi;

while(x<=xii)

{

x++;

if(P<0)

{

P=P+(2\*dy);

}

else

{

y--;

P=P+(2\*dy)-(2\*dx);

}

if(item==1)

{

setpixel(x,y);

}

if(item==2 && i%10<5)

{

setpixel(x,y);

}

if(item==3 && (i%9>=2 && i%9!=7))

{

setpixel(x,y);

}

if(item==4)

{

glPointSize(4.0);

setpixel(x,y);

}

i++;

}

}

else if(dy<-dx)

{

dx=abs(dx);

dy=abs(dy);

P=(2\*dy)-dx;

setpixel(xi,yi);

int x=xi;

int y=yi;

while(y>=yii)

{

y--;

if(P<0)

{

P=P+(2\*dx);

}

else

{

x++;

P=P+(2\*dx)-(2\*dy);

}

if(item==1)

{

setpixel(x,y);

}

if(item==2 && i%10<5)

{

setpixel(x,y);

}

if(item==3 && (i%9>=2 && i%9!=7))

{

setpixel(x,y);

}

if(item==4)

{

glPointSize(4.0);

setpixel(x,y);

}

}

}

glFlush();

}

};

line l;

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

{

if(key==27)

exit(0);

else

cout<<"You entered the"<<key;

}

void menu(int item)

{

if(item==1)

{

l.dda(1);

}

if(item==2)

{

l.dda(2);

}

if(item==3)

{

l.dda(3);

}

if(item==4)

{

l.dda(4);

}

if(item==5)

{

l.bresenham(1);

}

if(item==6)

{

l.bresenham(2);

}

if(item==7)

{

l.bresenham(3);

}

if(item==8)

{

l.bresenham(4);

}

if(item==9)

{

exit(0);

}

}

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

{

if( state == GLUT\_DOWN )

{

if( button == GLUT\_LEFT\_BUTTON)

{

xi=x-350;

yi=350-y;

cout<<xi<<"\t";

cout<<yi<<"\n";

glPointSize(2.0);

glBegin(GL\_POINTS);

glVertex2i(xi,yi);

glEnd();

glFlush();

}

if( button == GLUT\_RIGHT\_BUTTON)

{

xii=x-350;

yii=350-y;

cout<<xii<<"\t";

cout<<yii<<"\n";

glPointSize(2.0);

glBegin(GL\_POINTS);

glVertex2i(xii,yii);

glEnd();

glFlush();

}

}

}

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

{

glutInit(&argc,argv);

glutInitWindowSize(h,w);

glutInitWindowPosition(100,0);

glutCreateWindow("Line DDA and Bresenham Here!!");

initialize();

glutDisplayFunc(choice);

glutMouseFunc(mouse);

glutKeyboardFunc(keyboard);

glutCreateMenu(menu);

glutAddMenuEntry("DDA\_SIMPLE",1);

glutAddMenuEntry("DDA\_DASH",2);

glutAddMenuEntry("DDA\_DASH DOT",3);

glutAddMenuEntry("DDA\_THICK",4);

glutAddMenuEntry("BRE\_SIMPLE",5);

glutAddMenuEntry("BRE\_DASH",6);

glutAddMenuEntry("BRE\_DASH DOT",7);

glutAddMenuEntry("BRE\_THICK",8);

glutAddMenuEntry("EXIT",9);

glutAttachMenu(GLUT\_MIDDLE\_BUTTON);

glutMainLoop();

return 0;

}

**3.** **Implement Bresenham circle drawing algorithm to draw any object. The object should be displayed in all the quadrants with respect to center and radius.**

#include <GL/glut.h>

#include <cmath>

#include <iostream>

// Global variables for center, radius, and selected quadrant

int centerX = 0, centerY = 0;

int radius = 0;

int selectedQuadrant = 0; // 1: Q1, 2: Q2, 3: Q3, 4: Q4

bool isCenterSet = false; // Flag to check if center is set

bool isRadiusSet = false; // Flag to check if radius is set

// Function to plot points at the given (x, y)

void plotPoint(int x, int y) {

glBegin(GL\_POINTS);

glVertex2i(x, y);

glEnd();

}

// Bresenham's Circle Drawing Algorithm

void bresenhamCircle(int radius) {

int x = 0;

int y = radius;

int p = 3 - 2 \* radius;

// Draw the circle in the selected quadrant

while (x <= y) {

if (selectedQuadrant == 1) { // Quadrant 1

plotPoint(centerX + x, centerY + y);

plotPoint(centerX - x, centerY + y);

plotPoint(centerX + x, centerY - y);

plotPoint(centerX - x, centerY - y);

plotPoint(centerX + y, centerY + x);

plotPoint(centerX - y, centerY + x);

plotPoint(centerX + y, centerY - x);

plotPoint(centerX - y, centerY - x);

} else if (selectedQuadrant == 2) { // Quadrant 2

plotPoint(centerX - x, centerY + y);

plotPoint(centerX + x, centerY + y);

plotPoint(centerX - x, centerY - y);

plotPoint(centerX + x, centerY - y);

plotPoint(centerX - y, centerY + x);

plotPoint(centerX + y, centerY + x);

plotPoint(centerX - y, centerY - x);

plotPoint(centerX + y, centerY - x);

} else if (selectedQuadrant == 3) { // Quadrant 3

plotPoint(centerX - x, centerY - y);

plotPoint(centerX + x, centerY - y);

plotPoint(centerX - x, centerY + y);

plotPoint(centerX + x, centerY + y);

plotPoint(centerX - y, centerY - x);

plotPoint(centerX + y, centerY - x);

plotPoint(centerX - y, centerY + x);

plotPoint(centerX + y, centerY + x);

} else if (selectedQuadrant == 4) { // Quadrant 4

plotPoint(centerX + x, centerY - y);

plotPoint(centerX - x, centerY - y);

plotPoint(centerX + x, centerY + y);

plotPoint(centerX - x, centerY + y);

plotPoint(centerX + y, centerY - x);

plotPoint(centerX - y, centerY - x);

plotPoint(centerX + y, centerY + x);

plotPoint(centerX - y, centerY + x);

}

if (p < 0) {

p = p + 4 \* x + 6;

} else {

p = p + 4 \* (x - y) + 10;

y--;

}

x++;

}

}

// Function to draw the division lines (quadrants)

void drawQuadrants() {

glColor3f(1.0f, 1.0f, 1.0f); // Set color to white

// Draw the vertical and horizontal lines to divide the window into 4 quadrants

glBegin(GL\_LINES);

glVertex2i(0, 300); glVertex2i(0, -300); // Vertical line

glVertex2i(-300, 0); glVertex2i(300, 0); // Horizontal line

glEnd();

}

// Function to handle mouse clicks and capture center, radius, and quadrant selection

void mouseFunc(int button, int state, int x, int y) {

if (button == GLUT\_LEFT\_BUTTON && state == GLUT\_DOWN) {

int window\_width = 600;

int window\_height = 600;

int screenCenterX = window\_width / 2;

int screenCenterY = window\_height / 2;

// Convert mouse click position to OpenGL coordinates

int openglY = screenCenterY - y; // Adjust for OpenGL coordinate system

if (!isCenterSet) {

// First click to set the center of the circle

centerX = x - screenCenterX; // Translate click to OpenGL coordinates

centerY = openglY;

isCenterSet = true; // Center has been set

std::cout << "Center Set at: (" << centerX << ", " << centerY << ")\n";

} else if (!isRadiusSet) {

// Second click to set the radius of the circle

int dx = x - screenCenterX - centerX; // Distance from center in x

int dy = openglY - centerY; // Distance from center in y

radius = static\_cast<int>(sqrt(dx \* dx + dy \* dy)); // Calculate the distance (radius)

isRadiusSet = true; // Radius has been set

std::cout << "Radius Set: " << radius << "\n";

} else {

// Third click to select the quadrant (divide the window into 4 quadrants)

if (x < screenCenterX && y > screenCenterY) {

selectedQuadrant = 2; // Quadrant 2 (top-left)

} else if (x > screenCenterX && y > screenCenterY) {

selectedQuadrant = 1; // Quadrant 1 (top-right)

} else if (x < screenCenterX && y < screenCenterY) {

selectedQuadrant = 3; // Quadrant 3 (bottom-left)

} else if (x > screenCenterX && y < screenCenterY) {

selectedQuadrant = 4; // Quadrant 4 (bottom-right)

}

// Redraw the window after setting the quadrant

glutPostRedisplay();

}

}

}

// Function to handle keyboard input

void keyboardFunc(unsigned char key, int x, int y) {

if (key == 'r' || key == 'R') {

// Reset center, radius, and selected quadrant on 'r' key press

isCenterSet = false;

isRadiusSet = false;

selectedQuadrant = 0;

radius = 0;

centerX = 0;

centerY = 0;

std::cout << "Resetting Circle\n"; // Inform user of reset

glutPostRedisplay(); // Redraw the window

}

}

// Display function to set up the OpenGL window

void display() {

glClear(GL\_COLOR\_BUFFER\_BIT); // Clear the screen

// Draw the quadrants first

drawQuadrants();

// Draw the circle if center and radius are set

glColor3f(1.0f, 1.0f, 1.0f); // Set color to white

if (isCenterSet && isRadiusSet && selectedQuadrant != 0) {

bresenhamCircle(radius); // Draw the circle in the selected quadrant

}

glFlush(); // Render the scene

}

// Initialize OpenGL settings

void initOpenGL() {

glClearColor(0.0f, 0.0f, 0.0f, 1.0f); // Set background color to black

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

glOrtho(-300, 300, -300, 300, -1.0, 1.0); // Set orthographic view with origin at center

glPointSize(2.0); // Set point size

}

// Main function

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

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(600, 600);

glutCreateWindow("Bresenham Circle Drawing with Mouse & Quadrant Selection");

initOpenGL(); // Initialize OpenGL settings

glutDisplayFunc(display); // Set display callback function

glutMouseFunc(mouseFunc); // Set mouse click callback function

glutKeyboardFunc(keyboardFunc); // Set keyboard input callback function

glutMainLoop(); // Enter the main loop

return 0;

}

**4.** **Implement the following polygon filling methods : i) Flood fill / Seed fill**

**ii) Boundary fill ; using mouse click, keyboard interface and menu driven programming**-

#include<GL/glut.h>

#include<iostream>

#include<math.h>

using namespace std;

void myInit(void)

{

glClearColor(0.0,0.0,0.0,0.0);

glClear(GL\_COLOR\_BUFFER\_BIT);

glPointSize(1.0);

glLineWidth(2.0);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0.0,500,0,500);

}

void display(void)

{

}

struct COLOR

{

float r;

float b;

float g;

};

class polygon\_fill

{

int compare\_color(COLOR a,COLOR b)

{

if(a.r==b.r&&a.b==b.b&&a.g==b.g)

return 1;

else

return 0;

}

void draw(int x,int y,COLOR c)

{

glColor3f(c.r,c.b,c.g);

glBegin(GL\_POINTS);

glVertex2f(x,y);

glEnd();

glFlush();

}

COLOR getpixelcolor(int x,int y)

{

COLOR c;

glReadPixels(x,y,1,1,GL\_RGB,GL\_FLOAT,&c);

return c;

}

public:

void bound\_fill(int x,int y,COLOR fill,COLOR border\_col)

{

COLOR color;

color=getpixelcolor(x,y);

if(compare\_color(color,border\_col)==0&&compare\_color(color,fill)==0)

{

draw(x,y,fill);

bound\_fill(x+1,y,fill,border\_col);

bound\_fill(x,y+1,fill,border\_col);

bound\_fill(x-1,y,fill,border\_col);

bound\_fill(x,y-1,fill,border\_col);

bound\_fill(x-1,y-1,fill,border\_col);

bound\_fill(x-1,y+1,fill,border\_col);

bound\_fill(x+1,y-1,fill,border\_col);

bound\_fill(x+1,y+1,fill,border\_col);

}

}

void flood\_fill(int x,int y,COLOR fill,COLOR bg\_color)

{

COLOR color;

color=getpixelcolor(x,y);

if(compare\_color(color,bg\_color)==1)

{

draw(x,y,fill);

flood\_fill(x+1,y,fill,bg\_color);

flood\_fill(x,y+1,fill,bg\_color);

flood\_fill(x-1,y,fill,bg\_color);

flood\_fill(x,y-1,fill,bg\_color);

}

}

};

int xc,yc,i=0;

int xarr[100];

int yarr[100];

void menu(int value)

{

polygon\_fill p;

COLOR fill,bcol,back\_col,c;

fill.r=0.0;

fill.b=0.0;

fill.g=1.0;

bcol.r=1.0;

bcol.b=0.0;

bcol.g=0.0;

back\_col.r=0.0;

back\_col.b=0.0;

back\_col.g=0.0;

switch(value)

{

case 1:

p.flood\_fill(xc,500-yc,fill,back\_col);

cout<<"COMPLETED FLOOD FILL"<<"\n";

break;

case 2:

p.bound\_fill(xc,500-yc,fill,bcol);

cout<<"COMPLETED BOUNDARY FILL"<<"\n";

break;

case 3:

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(0.0,0.0,0.0);

glBegin(GL\_POINTS);

glVertex2i(0,0);

glEnd();

glFlush();

i=0;

cout<<"SCREEN CLEARED"<<"\n";

break;

}

}

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

{

if(key==27)

exit(0);

else

cout<<"you pressed "<<"\n";

}

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

{

if(state==GLUT\_DOWN)

{

if(button==GLUT\_LEFT\_BUTTON)

{

xarr[i]=x;

yarr[i]=y;

i++;

if(i>1)

{

glColor3f(1.0,0.0,0.0);

glBegin(GL\_LINE\_STRIP);

glVertex2i(xarr[i-2],500-yarr[i-2]);

glVertex2i(xarr[i-1],500-yarr[i-1]);

glEnd();

glFlush();

}

}

if(button==GLUT\_RIGHT\_BUTTON)

{

xc=x;

yc=y;

}

}

return;

}

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

{

glutInit(&argc,argv);

glutInitWindowSize(500,500);

glutInitWindowPosition(200,100);

glutCreateWindow("POLYGON FILL");

glutDisplayFunc(display);

glutMouseFunc(mouse);

glutKeyboardFunc(keyboard);

glutCreateMenu(menu);

glutAddMenuEntry("FLOOD FILL",1);

glutAddMenuEntry("BOUNDARY FILL",2);

glutAddMenuEntry("CLEAR SCREEN",3);

glutAttachMenu(GLUT\_MIDDLE\_BUTTON);

cout<<"PLEASE FOLLOW THESE STEPS:"<<endl;

cout<<"1. MAKE PLOYGON by USING LEFT CLICK"<<endl;

cout<<"2. SELECT SEED PIXEL BY RIGHT CLICK"<<endl;

cout<<"FOR MENU, use the middle button of the mouse"<<endl;

myInit();

glutMainLoop();

return 0;

}

**5.** **Implement Cohen Sutherland polygon clipping method to clip the polygon with respect the viewport and window. Use mouse click, keyboard interface.**

#include <stdio.h>

#include <iostream>

#include <GL/glut.h>

#include <math.h>

using namespace std;

int result;

int xmin, ymin, xmax, ymax, pt[30][2], w[30][2], n = 0, flg = 0;

int leftClip(int limit, int xm) {

int i, j = 0, x1, y1, x2, y2;

float m;

for (i = 0; i < limit; i++) {

x1 = pt[i][0];

y1 = pt[i][1];

x2 = pt[(i + 1) % limit][0];

y2 = pt[(i + 1) % limit][1];

if (x2 - x1) m = (y2 - y1) \* 1.0 / (x2 - x1);

if (x1 < xm && x2 < xm) continue;

if (x1 > xm && x2 > xm) {

w[j][0] = x2; w[j++][1] = y2;

continue;

}

if (x1 > xm && x2 < xm) {

w[j][0] = xm;

w[j++][1] = y1 + m \* (xm - x1);

continue;

}

if (x1 < xm && x2 > xm) {

w[j][0] = xm;

w[j++][1] = y1 + m \* (xm - x1);

w[j][0] = x2;

w[j++][1] = y2;

}

}

for (i = 0; i < j; i++) {

pt[i][0] = w[i][0];

pt[i][1] = w[i][1];

w[i][0] = w[i][1] = 0;

}

if (j < limit)

for (; i < limit; i++)

pt[i][0] = pt[i][1] = 0;

return j;

}

// Repeat for other clips...

// topClip, rightClip, bottomClip (same as original)

int topClip(int limit, int ym) {

int i, j = 0, x1, y1, x2, y2;

float m;

for (i = 0; i < limit; i++) {

x1 = pt[i][0]; y1 = pt[i][1];

x2 = pt[(i + 1) % limit][0]; y2 = pt[(i + 1) % limit][1];

if (x2 - x1) m = (y2 - y1) \* 1.0 / (x2 - x1);

if (y1 < ym && y2 < ym) continue;

if (y1 > ym && y2 > ym) {

w[j][0] = x2; w[j++][1] = y2;

continue;

}

if (y1 > ym && y2 < ym) {

w[j][0] = x1 + (ym - y1) / m; w[j++][1] = ym;

continue;

}

if (y1 < ym && y2 > ym) {

w[j][0] = x1 + (ym - y1) / m; w[j++][1] = ym;

w[j][0] = x2; w[j++][1] = y2;

}

}

for (i = 0; i < j; i++) {

pt[i][0] = w[i][0];

pt[i][1] = w[i][1];

w[i][0] = w[i][1] = 0;

}

if (j < limit)

for (; i < limit; i++)

pt[i][0] = pt[i][1] = 0;

return j;

}

// Repeat rightClip and bottomClip...

int rightClip(int limit, int xm) {

int i, j = 0, x1, y1, x2, y2;

float m;

for (i = 0; i < limit; i++) {

x1 = pt[i][0]; y1 = pt[i][1];

x2 = pt[(i + 1) % limit][0]; y2 = pt[(i + 1) % limit][1];

if (x2 - x1) m = (y2 - y1) \* 1.0 / (x2 - x1);

if (x1 > xm && x2 > xm) continue;

if (x1 < xm && x2 < xm) {

w[j][0] = x2; w[j++][1] = y2;

continue;

}

if (x1 < xm && x2 > xm) {

w[j][0] = xm; w[j++][1] = y1 + m \* (xm - x1);

continue;

}

if (x1 > xm && x2 < xm) {

w[j][0] = xm; w[j++][1] = y1 + m \* (xm - x1);

w[j][0] = x2; w[j++][1] = y2;

}

}

for (i = 0; i < j; i++) {

pt[i][0] = w[i][0];

pt[i][1] = w[i][1];

w[i][0] = w[i][1] = 0;

}

if (j < limit)

for (; i < limit; i++)

pt[i][0] = pt[i][1] = 0;

return j;

}

int bottomClip(int limit, int ym) {

int i, j = 0, x1, y1, x2, y2;

float m;

for (i = 0; i < limit; i++) {

x1 = pt[i][0]; y1 = pt[i][1];

x2 = pt[(i + 1) % limit][0]; y2 = pt[(i + 1) % limit][1];

if (x2 - x1) m = (y2 - y1) \* 1.0 / (x2 - x1);

if (y1 > ym && y2 > ym) continue;

if (y1 < ym && y2 < ym) {

w[j][0] = x2; w[j++][1] = y2;

continue;

}

if (y1 < ym && y2 > ym) {

w[j][0] = x1 + (ym - y1) / m; w[j++][1] = ym;

continue;

}

if (y1 > ym && y2 < ym) {

w[j][0] = x1 + (ym - y1) / m; w[j++][1] = ym;

w[j][0] = x2; w[j++][1] = y2;

}

}

for (i = 0; i < j; i++) {

pt[i][0] = w[i][0];

pt[i][1] = w[i][1];

w[i][0] = w[i][1] = 0;

}

if (j < limit)

for (; i < limit; i++)

pt[i][0] = pt[i][1] = 0;

return j;

}

void display() {}

void init() {

glClearColor(0.0, 0.0, 0.0, 0.0);

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(1.0, 0.0, 0.0);

glPointSize(2.0);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0, 700, 0, 700);

}

void menu(int c) {

if (c == 1) {

result = leftClip(n, xmin);

result = topClip(result, ymin);

result = rightClip(result, xmax);

result = bottomClip(result, ymax);

}

if (c == 2) {

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(1.0, 1.0, 0.0);

glBegin(GL\_LINE\_LOOP);

glVertex2i(xmin, ymin); glVertex2i(xmax, ymin);

glVertex2i(xmax, ymax); glVertex2i(xmin, ymax);

glEnd();

glFlush();

for (int i = 0; i < result; i++) {

glColor3f(0.0, 0.0, 1.0);

glBegin(GL\_LINE\_STRIP);

glVertex2i(pt[i][0], pt[i][1]);

glVertex2i(pt[(i + 1) % result][0], pt[(i + 1) % result][1]);

glEnd();

glFlush();

}

}

}

void mouse(int button, int state, int cx, int cy) {

if (state == GLUT\_DOWN) {

if (button == GLUT\_LEFT\_BUTTON) {

pt[n][0] = cx;

pt[n][1] = 700 - cy;

n++;

if (n > 1) {

glColor3f(1.0, 0.0, 0.0);

glBegin(GL\_LINE\_STRIP);

glVertex2i(pt[n - 2][0], pt[n - 2][1]);

glVertex2i(pt[n - 1][0], pt[n - 1][1]);

glEnd();

glFlush();

}

}

if (button == GLUT\_RIGHT\_BUTTON) {

if (flg == 0) {

xmin = cx;

ymin = 700 - cy;

flg++;

} else {

xmax = cx;

ymax = 700 - cy;

glColor3f(1.0, 1.0, 0.0);

glBegin(GL\_LINE\_LOOP);

glVertex2i(xmin, ymin);

glVertex2i(xmax, ymin);

glVertex2i(xmax, ymax);

glVertex2i(xmin, ymax);

glEnd();

glFlush();

}

}

}

}

int main(int argc, char\* argv[]) {

glutInit(&argc, argv);

glutInitWindowSize(700, 700);

glutInitWindowPosition(500, 50);

glutCreateWindow("Polygon Clipping - Sutherland-Hodgman");

cout << "Left click to define polygon vertices.\n";

cout << "Right click twice to define clipping window.\n";

cout << "Middle click menu: EXECUTE to clip, SHOW CLIPPED to display.\n";

init();

glutMouseFunc(mouse);

glutDisplayFunc(display);

glutCreateMenu(menu);

glutAddMenuEntry("EXECUTE", 1);

glutAddMenuEntry("SHOW CLIPPED", 2);

glutAttachMenu(GLUT\_MIDDLE\_BUTTON);

glutMainLoop();

return 0;

}

**6.** **Implement following 2D transformations on the object with respect to axis : – i) Scaling ii) Rotation about arbitrary point iii) Reflection**

#include<iostream>

#include<GL/glut.h>

#include<math.h>

#include<bits/stdc++.h>

using namespace std ;

int m[20][3], n = 0 ;

void setpixel(GLint x, GLint y)

{

glColor3f(0.0,0.0,1.0);

glBegin(GL\_POINTS);

glVertex2f(x,y);

glEnd();

glFlush();

}

void choice()

{

int i;

glPointSize(2.0);

for(i=-700; i<700; i++)

{

setpixel(0,i);

setpixel(i,0);

}

}

void setpcolor(double r1, double b1, double g1 )

{

glColor3f(r1,b1,g1);

}

void conect(int x, int y, int px, int py)

{

glPointSize(2);

glBegin(GL\_LINE\_STRIP);

glVertex2i(x,y);

glVertex2i(px,py);

glEnd();

glFlush();

}

void translation(int tx,int ty)

{

int tm[3][3] = {{1,0,tx},{0,1,ty},{0,0,1} },ne[3]= {} ;

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

{

ne [0] = tm[0][0]\*m[i][0] + tm[0][1]\*m[i][1] + tm[0][2]\*m[i][2] ;

ne [1] = tm[1][0]\*m[i][0] + tm[1][1]\*m[i][1] + tm[1][2]\*m[i][2] ;

ne [2] = tm[2][0]\*m[i][0] + tm[2][1]\*m[i][1] + tm[2][2]\*m[i][2] ;

m[i][0] = ne[0] ;

m[i][1] = ne[1] ;

m[i][2] = ne[2] ;

}

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

{

int ni = (i+1)%n;

setpcolor(1,1,0) ;

conect(m[i][0],m[i][1],m[ni][0],m[ni][1]) ;

}

}

void rotation(double rot, int xm, int ym )

{

double pi = 3.14159265 ;

double rad = (pi/180.00) ;

rad \*= rot ;

double rm[3][3] = {{cos(rad),sin(rad),0},{-sin(rad),cos(rad),0},{-xm\*cos(rad)+ym\*sin(rad)+xm,-xm\*sin(rad)-ym\*cos(rad)+ym,1} } ;

int ne[3]= {};

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

{

ne [0] = rm[0][0]\*m[i][0] + rm[0][1]\*m[i][1] + rm[0][2]\*m[i][2] ;

ne [1] = rm[1][0]\*m[i][0] + rm[1][1]\*m[i][1] + rm[1][2]\*m[i][2] ;

ne [2] = rm[2][0]\*m[i][0] + rm[2][1]\*m[i][1] + rm[2][2]\*m[i][2] ;

m[i][0] = ne[0] ;

m[i][1] = ne[1] ;

m[i][2] = ne[2] ;

}

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

{

int ni = (i+1)%n;

setpcolor(1,1,0);

conect(m[i][0],m[i][1],m[ni][0],m[ni][1]) ;

}

}

void scale(int sx, int sy )

{

int sm[3][3] = {{sx,0,0},{0,sy,0},{0,0,1} } ;

int ne[3]= {} ;

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

{

ne [0] = sm[0][0]\*m[i][0] + sm[0][1]\*m[i][1] + sm[0][2]\*m[i][2] ;

ne [1] = sm[1][0]\*m[i][0] + sm[1][1]\*m[i][1] + sm[1][2]\*m[i][2] ;

ne [2] = sm[2][0]\*m[i][0] + sm[2][1]\*m[i][1] + sm[2][2]\*m[i][2] ;

m[i][0] = ne[0] ;

m[i][1] = ne[1] ;

m[i][2] = ne[2] ;

}

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

{

int ni = (i+1)%n;

setpcolor(1,1,0) ;

conect(m[i][0],m[i][1],m[ni][0],m[ni][1]) ;

}

}

void reflect(char c)

{

int sm[3][3] = {{1,0,0},{0,1,0},{0,0,1} } ;

if(c=='x'||c=='X')

{

sm[1][1]=-1 ;

}

else

{

sm[0][0] = - 1 ;

}

int ne[3]= {} ;

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

{

ne [0] = sm[0][0]\*m[i][0] + sm[0][1]\*m[i][1] + sm[0][2]\*m[i][2] ;

ne [1] = sm[1][0]\*m[i][0] + sm[1][1]\*m[i][1] + sm[1][2]\*m[i][2] ;

ne [2] = sm[2][0]\*m[i][0] + sm[2][1]\*m[i][1] + sm[2][2]\*m[i][2] ;

m[i][0] = ne[0] ;

m[i][1] = ne[1] ;

m[i][2] = ne[2] ;

}

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

{

int ni = (i+1)%n;

setpcolor(1,1,0) ;

conect(m[i][0],m[i][1],m[ni][0],m[ni][1]) ;

}

}

void init()

{

glClearColor(0.0,0.0,0.0,0.0);

glClear (GL\_COLOR\_BUFFER\_BIT);

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(1.0,0.0,0.0);

glPointSize(2.0);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(-350,350,-350,350) ;

}

void menu(int c)

{

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

{

int ni = (i+1)%n;

setpcolor(1,1,1) ;

conect(m[i][0],m[i][1],m[ni][0],m[ni][1]) ;

}

glFlush() ;

if (c==1)

{

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

{

cout<<m[i][0]<<" "<<m[i][1]<<endl;

}

}

else if(c==2)

{

int tx = 0, ty = 0 ;

cout <<"Enter x-translation factor : " ;

cin >> tx ;

cout <<"Enter y-translation factor : " ;

cin >> ty ;

translation(tx,ty) ;

}

else if(c==3)

{

double rot ;

int flg = 1,ym,xm ;

cout <<"Enter the arbitrary point x :" ;

cin >> xm ;

cout <<"Enter the arbitrary point y :" ;

cin >> ym ;

cout <<"Enter 1 for clockwise else enter 0 for anti-clock wise : " ;

cin >> flg ;

cout <<"Enter by how much degree the object is to be rotated : " ;

cin >> rot ;

if(flg)

{

rot = -rot ;

}

rotation(rot,xm,ym) ;

}

else if(c==4)

{

int sx = 1, sy = 1 ;

cout <<"Enter the horizontal scaling factor : " ;

cin >> sx ;

cout <<"Enter the vertical scaling factor : " ;

cin >> sy ;

scale(sx,sy) ;

}

else if(c==5)

{

char c ;

cout <<"Enter the axis of reflection : (X | Y |)" ;

cin >> c ;

reflect(c) ;

}

}

void mouse(int button, int state, int cx, int cy )

{

cx -= 350 ;

cy -= 350 ;

cy = - cy ;

if(state==GLUT\_DOWN)

{

if(button==GLUT\_LEFT\_BUTTON)

{

m[n][0] = cx ;

m[n][1] = cy ;

m[n][2] = 1 ;

n++;

if(n>1)

{

glColor3f(1.0,0.0,0.0);

glBegin(GL\_LINE\_STRIP);

glVertex2i(m[n-2][0],m[n-2][1]);

glVertex2i(m[n-1][0],m[n-1][1]);

glEnd();

glFlush();

}

}

}

}

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

{

glutInit(&argc,argv);

glutInitWindowSize(700,700);

glutInitWindowPosition(500,50);

glutCreateWindow(" 2D TRANSFORMATION ");

cout<<"PLEASE FOLLOW THESE STEPS:"<<endl;

cout<<"MAKE POLYGON by USING LEFT BUTTON CLICK"<<endl;

cout<<"FOR MENU, use the RIGHT button of the mouse"<<endl;

init();

glutDisplayFunc(choice);

glutMouseFunc(mouse);

glutCreateMenu(menu);

glutAddMenuEntry("DISPLAY AXES OF POLYGON",1);

glutAddMenuEntry("TRANSLATION",2);

glutAddMenuEntry("ROTATION",3);

glutAddMenuEntry("SCALING",4);

glutAddMenuEntry("REFLECTION",5);

glutAttachMenu(GLUT\_RIGHT\_BUTTON);

glutMainLoop();

return 0;

}

**7.** **Generate fractal patterns using i) Bezier ii) Koch Curve**

**1)Bezier curve**

#include <iostream>

#include <GL/glut.h>

#include <cmath>

using namespace std;

// This is a point class, used to store the coordinates of the point

class Point {

public:

int x, y;

void setxy(int \_x, int \_y) {

x = \_x;

y = \_y;

}

};

// Number of points

static int POINTSNUM = 0;

// Used to store a collection of points

static Point points[4];

// Initialization function

void init(void) {

glClearColor(0.0, 0.0, 0.0, 0); // Background to black

glColor3f(1.0, 1.0, 1.0); // Drawing color white

glPointSize(4.0); // Set point size

glMatrixMode(GL\_PROJECTION); // Set projection matrix

glLoadIdentity(); // Load identity matrix

gluOrtho2D(0.0, 600.0, 0.0, 480.0); // Set orthographic view

}

// Draw a point

void setPoint(Point p) {

glBegin(GL\_POINTS);

glVertex2f(p.x, p.y);

glEnd();

glFlush();

}

// Draw a line

void setline(Point p1, Point p2) {

glBegin(GL\_LINES);

glVertex2f(p1.x, p1.y);

glVertex2f(p2.x, p2.y);

glEnd();

glFlush();

}

// Compute a point on the Bezier curve

Point setBezier(Point p1, Point p2, Point p3, Point p4, double t) {

Point p;

double a1 = pow((1 - t), 3);

double a2 = pow((1 - t), 2) \* 3 \* t;

double a3 = 3 \* t \* t \* (1 - t);

double a4 = t \* t \* t;

p.x = a1 \* p1.x + a2 \* p2.x + a3 \* p3.x + a4 \* p4.x;

p.y = a1 \* p1.y + a2 \* p2.y + a3 \* p3.y + a4 \* p4.y;

return p;

}

// Display function (not doing anything here)

void display() {

// Optional clear screen logic can go here

}

// Mouse callback function

void mymouseFunction(int button, int state, int x, int y) {

if (state == GLUT\_DOWN) {

points[POINTSNUM].setxy(x, 480 - y);

glColor3f(1.0, 0.0, 0.0);

setPoint(points[POINTSNUM]);

glColor3f(1.0, 0.0, 0.0);

if (POINTSNUM > 0)

setline(points[POINTSNUM - 1], points[POINTSNUM]);

if (POINTSNUM == 3) {

glColor3f(1.0, 1.0, 0.0);

Point p\_current = points[0];

for (double t = 0.0; t <= 1.0; t += 0.05) {

Point P = setBezier(points[0], points[1], points[2], points[3], t);

setline(p\_current, P);

p\_current = P;

}

POINTSNUM = 0;

} else {

POINTSNUM++;

}

}

}

int main(int argc, char \*argv[]) {

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_RGB | GLUT\_SINGLE);

glutInitWindowSize(600, 480);

glutInitWindowPosition(100, 100);

glutCreateWindow("Bezier Curve");

init();

glutMouseFunc(mymouseFunction);

glutDisplayFunc(display);

glutMainLoop();

return 0;

}

**2)Koach curve**

#include <GL/glut.h>

#include <iostream>

#include <cstdlib>

#include <cmath>

using namespace std;

GLfloat oldx=500, oldy=800;

void koch(GLdouble dir, GLfloat len, GLint itr)

{

GLfloat newx;

GLfloat newy;

GLdouble rdir = 3.14159265358979323846/180.0 \* dir;

newx = oldx + len \* cos(rdir);

newy = oldy + len \* sin(rdir);

if(itr==0)

{

glVertex2f(oldx, oldy);

glVertex2f(newx, newy);

oldx = newx;

oldy = newy;

}

else

{

itr--;

koch(dir, len, itr);

dir+=60;

koch(dir, len, itr);

dir-=120;

koch(dir, len, itr);

dir+=60;

koch(dir, len, itr);

}

}

GLfloat i=3;

void display()

{

glColor3f(1,0,0);

glBegin(GL\_LINES);

koch(0,2,i);

koch(-120,2,i);

koch(120,2,i);

glEnd();

glFlush();

}

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

{

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

i++;

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

{

oldx=x;

oldy=fabs(720 - y);

display();

}

if(button == GLUT\_MIDDLE\_BUTTON && state == GLUT\_DOWN)

{

i=3;

glClear(GL\_COLOR\_BUFFER\_BIT);

glFlush();

}

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

{

glutInit(&argc,argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGBA);

glutInitWindowSize(1280, 720);

glutCreateWindow("Fractal");

gluOrtho2D(0,1280,0,720);

glClearColor(1,1,1,0);

glClearDepth(1);

glClear(GL\_COLOR\_BUFFER\_BIT);

cout<<"Left mouse button to draw Koch's Snowflake.\nRight Mouse button to increase the iterations.\nMiddle mouse button to clear the screen.\n";

glutDisplayFunc(display);

glutMouseFunc(mouse);

glutMainLoop();

return 0;

}

}

**8.** **Implement animation principles for any object.**

#include <iostream>

#include <stdlib.h>

#ifdef \_\_APPLE\_\_

#include <openGL/openGL.h>

#include <GLUT/glut.h>

#else

#include <GL/glut.h>

#endif

using namespace std;

float ballX = -0.8f;

float ballY = -0.3f;

float ballZ = -1.2f;

float colR = 3.0;

float colG = 1.5;

float colB = 1.0;

float bgColR = 0.0;

float bgColG = 0.0;

float bgColB = 0.0;

static int flag = 1;

void drawBall(void)

{

glColor3f(colR, colG, colB); // set ball colour

glTranslatef(ballX, ballY, ballZ); // moving it toward the screen a bit on creation

glutSolidSphere(0.05, 30, 30); // create ball.

}

void drawAv(void)

{

glBegin(GL\_POLYGON);

glColor3f(1.0, 1.0, 1.0);

glVertex3f(-0.9, -0.7, -1.0);

glVertex3f(-0.5, -0.1, -1.0);

glVertex3f(-0.2, -1.0, -1.0);

glVertex3f(0.5, 0.0, -1.0);

glVertex3f(0.6, -0.2, -1.0);

glVertex3f(0.9, -0.7, -1.0);

glEnd();

}

void drawClouds() {}

void keyPress(int key, int x, int y)

{

if (key == GLUT\_KEY\_RIGHT)

ballX -= 0.05f;

if (key == GLUT\_KEY\_LEFT)

ballX += 0.05f;

glutPostRedisplay();

}

void initRendering()

{

glEnable(GL\_DEPTH\_TEST);

glEnable(GL\_COLOR\_MATERIAL);

glEnable(GL\_LIGHTING); // Enable lighting

glEnable(GL\_LIGHT0); // Enable light #0

glEnable(GL\_LIGHT1); // Enable light #1

glEnable(GL\_NORMALIZE); // Automatically normalize normals

// glShadeModel(GL\_SMOOTH); //Enable smooth shading

}

// Called when the window is resized

void handleResize(int w, int h)

{

// Tell OpenGL how to convert from coordinates to pixel values

glViewport(0, 0, w, h);

glMatrixMode(GL\_PROJECTION); // Switch to setting the camera perspective

// Set the camera perspective

glLoadIdentity(); // Reset the camera

gluPerspective(45.0, // The camera angle

(double)w / (double)h, // The width-to-height ratio

1.0, // The near z clipping coordinate

200.0); // The far z clipping coordinate

}

void drawScene()

{

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

glClearColor(bgColR, bgColG, bgColB, 0.0);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

// Add ambient light

GLfloat ambientColor[] = {0.2f, 0.2f, 0.2f, 1.0f}; // Color (0.2, 0.2, 0.2)

glLightModelfv(GL\_LIGHT\_MODEL\_AMBIENT, ambientColor);

// Add positioned light

GLfloat lightColor0[] = {0.5f, 0.5f, 0.5f, 1.0f}; // Color (0.5, 0.5, 0.5)

GLfloat lightPos0[] = {4.0f, 0.0f, 8.0f, 1.0f}; // Positioned at (4, 0, 8)

glLightfv(GL\_LIGHT0, GL\_DIFFUSE, lightColor0);

glLightfv(GL\_LIGHT0, GL\_POSITION, lightPos0);

// Add directed light

GLfloat lightColor1[] = {0.5f, 0.2f, 0.2f, 1.0f}; // Color (0.5, 0.2, 0.2)

// Coming from the direction (-1, 0.5, 0.5)

GLfloat lightPos1[] = {-1.0f, 0.5f, 0.5f, 0.0f};

glLightfv(GL\_LIGHT1, GL\_DIFFUSE, lightColor1);

glLightfv(GL\_LIGHT1, GL\_POSITION, lightPos1);

// drawing the SUN

glPushMatrix();

drawBall();

glPopMatrix();

// drawing the Mount Avarest

glPushMatrix();

drawAv();

glPopMatrix();

// drawing the Clouds

glPushMatrix();

drawClouds();

glPopMatrix();

glutSwapBuffers();

}

// float \_angle = 30.0f;

void update(int value)

{

if (ballX > 0.9f)

{

ballX = -0.8f;

ballY = -0.3f;

flag = 1;

colR = 2.0;

colG = 1.50;

colB = 1.0;

bgColB = 0.0;

}

if (flag)

{

ballX += 0.001f;

ballY += 0.0007f;

colR -= 0.001;

// colG+=0.002;

colB += 0.005;

bgColB += 0.001;

if (ballX > 0.01)

{

flag = 0;

} }

if (!flag)

{

ballX += 0.001f;

ballY -= 0.0007f;

colR += 0.001;

colB -= 0.01;

bgColB -= 0.001;

if (ballX < -0.3)

{

flag = 1;

} }

glutPostRedisplay(); // Tell GLUT that the display has changed

// Tell GLUT to call update again in 25 milliseconds

glutTimerFunc(25, update, 0);

}

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

{

glutInit(&argc, argv);

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

glutInitWindowSize(400, 400);

glutCreateWindow("Sun");

initRendering();

glutDisplayFunc(drawScene);

glutFullScreen();

glutSpecialFunc(keyPress);

glutReshapeFunc(handleResize);

glutTimerFunc(25, update, 0);

glutMainLoop();

return (0);

}

**Animation another program**

#include <iostream>

#include <stdlib.h>

#ifdef \_\_APPLE\_\_

#include <openGL/openGL.h>

#include <GLUT/glut.h>

#else

#include <GL/glut.h>

#endif

// Global variables to control boat's position

float boatX = -0.8f; // Initial boat position

float boatY = -0.6f; // Boat's Y-position (on water)

float boatSpeed = 0.005f; // Boat's speed

void drawMountains() {

// Draw mountains in the background (simple triangles)

glColor3f(0.5, 0.5, 0.5); // Gray color for mountains

glBegin(GL\_TRIANGLES);

glVertex3f(-1.0f, -0.3f, 0.0f);

glVertex3f(0.0f, 0.5f, 0.0f);

glVertex3f(1.0f, -0.3f, 0.0f);

glVertex3f(-1.0f, -0.3f, 0.0f);

glVertex3f(-0.5f, 0.3f, 0.0f);

glVertex3f(0.0f, -0.3f, 0.0f);

glVertex3f(0.0f, -0.3f, 0.0f);

glVertex3f(0.5f, 0.3f, 0.0f);

glVertex3f(1.0f, -0.3f, 0.0f);

glEnd();

}

void drawWater() {

// Draw water (simple blue rectangle)

glColor3f(0.0f, 0.0f, 1.0f); // Blue color for water

glBegin(GL\_QUADS);

glVertex3f(-1.0f, -0.6f, 0.0f);

glVertex3f(1.0f, -0.6f, 0.0f);

glVertex3f(1.0f, -1.0f, 0.0f);

glVertex3f(-1.0f, -1.0f, 0.0f);

glEnd();

}

void drawBoat() {

// Draw the boat (simple rectangle with a triangle as the sail)

glColor3f(0.6f, 0.3f, 0.0f); // Brown color for boat body

glBegin(GL\_QUADS);

glVertex3f(boatX, boatY, 0.0f);

glVertex3f(boatX + 0.2f, boatY, 0.0f);

glVertex3f(boatX + 0.2f, boatY + 0.05f, 0.0f);

glVertex3f(boatX, boatY + 0.05f, 0.0f);

glEnd();

// Boat sail (simple triangle)

glColor3f(1.0f, 1.0f, 1.0f); // White color for the sail

glBegin(GL\_TRIANGLES);

glVertex3f(boatX + 0.05f, boatY + 0.05f, 0.0f);

glVertex3f(boatX + 0.15f, boatY + 0.15f, 0.0f);

glVertex3f(boatX + 0.2f, boatY + 0.05f, 0.0f);

glEnd();

}

void update(int value) {

// Move the boat from left to right

boatX += boatSpeed;

// If the boat moves off the screen, reset its position to the left

if (boatX > 1.0f) {

boatX = -1.0f;

}

// Redraw the scene

glutPostRedisplay();

// Call update function every 16 milliseconds (60 frames per second)

glutTimerFunc(16, update, 0);

}

void drawScene() {

// Clear the screen

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

// Set the background color

glClearColor(0.7f, 0.9f, 1.0f, 1.0f); // Light blue for the sky

// Draw mountains, water, and boat

drawMountains();

drawWater();

drawBoat();

// Swap the buffers to display the scene

glutSwapBuffers();

}

// Called when the window is resized

void handleResize(int w, int h) {

glViewport(0, 0, w, h);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluPerspective(45.0, (double)w / (double)h, 1.0, 200.0);

}

void initRendering() {

glEnable(GL\_DEPTH\_TEST); // Enable depth testing

glEnable(GL\_COLOR\_MATERIAL);

glEnable(GL\_LIGHTING);

glEnable(GL\_LIGHT0);

}

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

glutInit(&argc, argv);

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

glutInitWindowSize(800, 600); // Window size

glutCreateWindow("Boat Animation");

initRendering();

glutDisplayFunc(drawScene);

glutReshapeFunc(handleResize);

// Start the animation loop

glutTimerFunc(25, update, 0);

// Enter the GLUT event processing loop

glutMainLoop();

return 0;

}