

COMPUTER GRAPHICS LAB

**Submitted by: Submitted to:**

**NAME : Anmol yadav Dr.Pankaj Badoni**

**SAP: 500083814**

**ROLL NO.: R214220178**

**BATCH: 1 DevOps**

**Experiment 1**

**Lab-1(Draw Basic Shapes)**

Code:

#include<GL/gl.h>

#include<GL/glut.h>

#include<cstdio>

#include<cstring>

#include<cmath>

#include<iostream>

void myinit(void)

{

// glClearColor(0,0.75,1,0.0);

glClearColor(0,1,1,1);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

//glOrtho(0,0,0,0,0,-1);

}

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

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

if(state==GLUT\_DOWN){

glClearColor(0,0,0,0);

}

}

}

void printName(){

//for printing name

std::string s="NAME:Anmol SAP ID:500083814";

glRasterPos2f(-1.98,0.8);

int n=s.size();

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

glutBitmapCharacter(GLUT\_BITMAP\_TIMES\_ROMAN\_24,s[i]);

glEnd();

glFlush();

glutSwapBuffers();

}

void display(){

glClear(GL\_COLOR\_BUFFER\_BIT);

const int point1=0.3;

//triangle

glColor3f(0.5,0.4,0.07);

glBegin(GL\_TRIANGLES);

glVertex2f(0.3, 0);

glVertex2f(-0.3, 0);

glVertex2f(0.0,0.75);

glEnd();

//lower square

glColor3f(1, 0.8, 0);

glBegin(GL\_POLYGON);

glVertex3f (-0.3, 0, 0.0);

glVertex3f (0.3, 0, 0.0);

glVertex3f (0.3, -0.75, 0.0);

glVertex3f (-0.3, -0.75, 0.0);

glEnd();

//door

glBegin(GL\_POLYGON);

glColor3f(0,0,0);

glVertex3f (-0.1, -0.25, 0.0);

glColor3f(0,0,0);

glVertex3f (0.1, -0.25, 0.0);

glColor3f(0,1,0.2);

glVertex3f (0.1, -0.75, 0.0);

glColor3f(0.6, 0.8, 0.2);

glVertex3f (-0.1, -0.75, 0.0);

glColor3f(0, 1, 0.2);

glEnd();

glColor3f(0, 0, 0);

glBegin(GL\_LINES);

glVertex3f (0.1, -0.75, 0.0);

glVertex3f (-0.1, -0.75, 0.0);

glEnd();

// upper parallelogram

glColor3f(0.5,0.3,0.05);

glBegin(GL\_POLYGON);

glVertex3f (0.0, 0.75, 0.0);

glVertex3f (0.5, 0.75, 0.0);

glVertex3f (0.8, 0, 0.0);

glVertex3f (0.3, 0, 0.0);

glEnd();

// rightside square

glColor3f(1,0.5,0);

glBegin(GL\_POLYGON);

glVertex3f (0.8, 0, 0.0);

glVertex3f (0.8, -0.75, 0.0);

glVertex3f (0.3, -0.75, 0.0);

glVertex3f (0.3, 0, 0.0);

glEnd();

// window

glColor3f(0,0,0);

glBegin(GL\_POLYGON);

glVertex3f (0.7, -0.25, 0.0);

glColor3f(0,0,0);

glVertex3f (0.7, -0.6, 0.0);

glColor3f(0,0.3,0.6);

glVertex3f (0.45, -0.6, 0.0);

glColor3f(0,0.3,0.6);

glVertex3f (0.45, -0.25, 0.0);

glColor3f(0,0,0);

glEnd();

// lines of window

glColor3f(0, 0,0);

glBegin(GL\_LINES);

glVertex3f ((0.7+0.45)/2, -0.25, 0.0);

glVertex3f ((0.7+0.45)/2, -0.6, 0.0);

glEnd();

glBegin(GL\_LINES);

glVertex3f (0.45, -0.425, 0.0);

glVertex3f (0.7, -0.425, 0.0);

glEnd();

glFlush();

//for grass

glColor3f(0.2, 0.7, 0.3);

glBegin(GL\_POLYGON);

glVertex3f (-1, -0.75, 0.0);

glVertex3f (1, -0.75, 0.0);

glVertex3f (1, -1, 0.0);

glVertex3f (-1, -1, 0.0);

glEnd();

//for name

printName();

}

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

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(800,450);

glutInitWindowPosition(500,400);

glutCreateWindow("Hut");

myinit();

glutDisplayFunc(display);

glutMouseFunc(mouseClick);

glutMainLoop();

return EXIT\_SUCCESS;

}



# Lab 3-DDA

# Code:

# #include<GL/gl.h>

# #include<GL/glut.h>

# #include<cstdio>

# #include<cstring>

# #include<cmath>

# #include<iostream>

# float x\_1,x\_2,y\_1,y\_2; //line points

# int check=0; //flag variable

# int winwd=500; //window width

# int winht=500; //window height

# void myinit(void)

# {

# glClearColor(1,1,1,0.0);

# //glMatrixMode(GL\_PROJECTION);

# glLoadIdentity();

# glColor3f(1.0,0.0,0.0);

# gluOrtho2D(0,winwd,0,winht);

# glClear(GL\_COLOR\_BUFFER\_BIT);

# }

# void display(){

# //glClear(GL\_COLOR\_BUFFER\_BIT);

# float x,y,dx,dy,k,m,step,xinc,yinc;

# dx=x\_2-x\_1;

# dy=y\_2-y\_1;

# m=dy/dx; //slope

# if(abs(dx)>=abs(dy)) step = abs(dx);

# else step = abs(dy);

# x=x\_1; //intial positions

# y=y\_1;

# xinc=dx/step; //increment factors

# yinc=dy/step;

# //display slope

# std::string s2=std::to\_string(m);

# glRasterPos2f(x,y);

# int n2=s2.size();

# for(int i=0;i<n2;i++)

# glutBitmapCharacter(GLUT\_BITMAP\_TIMES\_ROMAN\_10,s2[i]);

# if(m<1 && m>0) glColor3f(1, 1, 0); //yellow if m<1

# else if(m>=1) glColor3f(1, 0, 0); //red if m>=1

# else if(m<-1) glColor3f(0, 1, 0); //green if m<-1

# else glColor3f(0, 0, 1); //blue if m>-1

# glBegin(GL\_POINTS);

# for(k=1;k<=step;k++){ //loop to generate points in order

# glVertex2f(x, y);

# x+=xinc;

# y+=yinc;

# }

# glEnd();

# glFlush();

# //printname

# glColor3f(0.5,0.3,0.05);

# }

# void mouseClick(int button,int state,int x, int y){

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

# if(check==0){ //starting point

# x\_1 = x;

# y\_1 = winht-y;

# check = 1;

# }

# else if(check==1) //end point

# {

# x\_2 = x;

# y\_2 = winht-y;

# check = 0;

# glutPostRedisplay();

# }

# }

# }

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

# glutInit(&argc, argv);

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

# glutInitWindowSize(winwd,winht);

# glutInitWindowPosition(0,0);

# glutCreateWindow("Lab3");

# myinit();

# glutMouseFunc(mouseClick);

# glutDisplayFunc(display);

# glutMainLoop();

# return EXIT\_SUCCESS;

# }

# Output:

# 

# Lab-4(Bresenham's Algorithm)

# #include<GL/gl.h>

# #include<GL/glut.h>

# #include<cstdio>

# #include<cstring>

# #include<cmath>

# #include<iostream>

# float x\_1,x\_2,y\_1,y\_2; //line point

# int check=0; //flag variable

# int winwd=500; //window width

# int winht=500; //window height

# void myinit(void)

# {

# glClearColor(1,1,1,0.0);

# //glMatrixMode(GL\_PROJECTION);

# glLoadIdentity();

# glColor3f(1.0,0.0,0.0);

# gluOrtho2D(0,winwd,0,winht);

# glClear(GL\_COLOR\_BUFFER\_BIT);

# }

# void printName(){

# //for printing name

# glEnd();

# glFlush();

# glutSwapBuffers();

# }

# void draw\_pixel(int x, int y) { //function to draw points

# glBegin(GL\_POINTS);

# glVertex2i(x, y);

# glEnd();

# }

# void draw\_line(float x\_1, float x\_2, float y\_1, float y\_2) {

# float dx, dy, k, p;

# float xinc, yinc, inc1, inc2; //for incrementing x, y and p

# float x,y;

# float m;

# 

# dx = x\_2-x\_1;

# dy = y\_2-y\_1;

# m=dy/dx;

# //display slope

# std::string s2=std::to\_string(m);

# glRasterPos2f(x\_1,y\_1);

# int n2=s2.size();

# for(int i=0;i<n2;i++)

# glutBitmapCharacter(GLUT\_BITMAP\_TIMES\_ROMAN\_10,s2[i]);

# if(m<1 && m>0) glColor3f(1, 1, 0); //yellow if m<1

# else if(m>=1) glColor3f(1, 0, 0); //red if m>=1

# else if(m<-1) glColor3f(0, 1, 0); //green if m<-1

# else glColor3f(0, 0, 1); //blue if m>-1

# //set intial points

# x = x\_1;

# y = y\_1;

# //modifying signs so that the values fit in the algo

# if (dx < 0) dx = -dx;

# if (dy < 0) dy = -dy;

# xinc = 1;

# if (x\_2 < x\_1) xinc = -1;

# yinc = 1;

# if (y\_2 < y\_1) yinc = -1;

# 

# if (dx > dy) {

# draw\_pixel(x, y);

# p = 2 \* dy-dx;

# inc1 = 2\*(dy-dx);

# inc2 = 2\*dy;

# for (k=0; k<dx; k++) {

# if (p >= 0) {

# y += yinc;

# p += inc1;

# }

# else{

# p += inc2;

# }

# x += xinc;

# draw\_pixel(x, y);

# }

# }

# else {

# draw\_pixel(x, y);

# p = 2\*dx-dy;

# inc1 = 2\*(dx-dy);

# inc2 = 2\*dx;

# for (k=0; k<dy; k++) {

# if (p >= 0) {

# x += xinc;

# p += inc1;

# }

# else{

# p += inc2;

# }

# y += yinc;

# draw\_pixel(x, y);

# }

# }

# }

# void display() {

# draw\_line(x\_1, x\_2, y\_1, y\_2);

# glColor3f(0.5,0.3,0.05);

# printName();

# glFlush();

# }

# void mouseClick(int button,int state,int x, int y){

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

# if(check==0){ //starting point

# x\_1 = x;

# y\_1 = winht-y;

# check = 1;

# draw\_pixel(x\_1, y\_1);

# }

# else if(check==1) //end point

# {

# x\_2 = x;

# y\_2 = winht-y;

# check = 0;

# draw\_pixel(x\_2, y\_2);

# draw\_line(x\_1, x\_2, y\_1, y\_2);

# glutPostRedisplay();

# }

# }

# }

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

# glutInit(&argc, argv);

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

# glutInitWindowSize(winwd,winht);

# glutInitWindowPosition(0,0);

# glutCreateWindow("Lab4");

# myinit();

# glutMouseFunc(mouseClick);

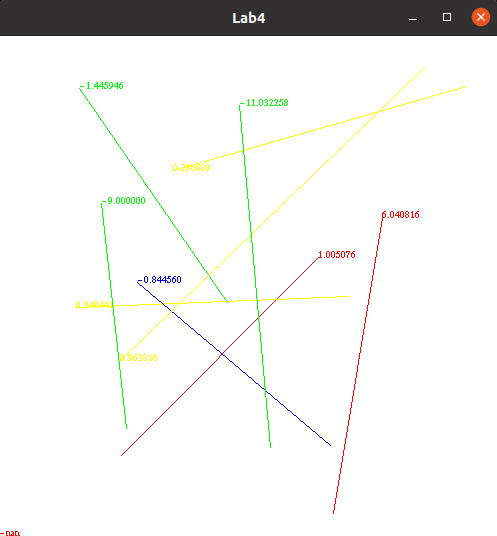
# glutDisplayFunc(display);

# glutMainLoop();

# return EXIT\_SUCCESS;

# }

**Output:**

****

# Lab-5(Seed Fill)

# Code:

# #include <math.h>

# #include <GL/glut.h>

# struct Point {

# GLint x;

# GLint y;

# };

# struct Color {

# GLfloat r;

# GLfloat g;

# GLfloat b;

# };

# void init() {

# glClearColor(1.0, 1.0, 1.0, 0.0);

# glColor3f(0.0, 0.0, 0.0);

# glPointSize(1.0);

# glMatrixMode(GL\_PROJECTION);

# glLoadIdentity();

# gluOrtho2D(0, 640, 0, 480);

# }

# Color getPixelColor(GLint x, GLint y) {

# Color color;

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

# return color;

# }

# void setPixelColor(GLint x, GLint y, Color color) {

# glColor3f(color.r, color.g, color.b);

# glBegin(GL\_POINTS);

# glVertex2i(x, y);

# glEnd();

# glFlush();

# }

# void floodFill(GLint x, GLint y, Color oldColor, Color newColor) {

# Color color;

# color = getPixelColor(x, y);

# if(color.r == oldColor.r && color.g == oldColor.g && color.b == oldColor.b)

# {

# setPixelColor(x, y, newColor);

# floodFill(x+1, y, oldColor, newColor);

# floodFill(x, y+1, oldColor, newColor);

# floodFill(x-1, y, oldColor, newColor);

# floodFill(x, y-1, oldColor, newColor);

# }

# return;

# }

# void onMouseClick(int button, int state, int x, int y)

# {

# Color newColor = {1.0f, 0.0f, 0.0f};

# Color oldColor = {1.0f, 1.0f, 1.0f};

# floodFill(320, 240, oldColor, newColor);

# }

# void draw\_circle(Point pC, GLfloat radius) {

# GLfloat step = 1/radius;

# GLfloat x, y;

# for(GLfloat theta = 0; theta <= 360; theta += step) {

# x = pC.x + (radius \* cos(theta));

# y = pC.y + (radius \* sin(theta));

# glVertex2i(x, y);

# }

# }

# void display(void) {

# Point pt = {320, 240};

# GLfloat radius = 50;

# glClear(GL\_COLOR\_BUFFER\_BIT);

# glBegin(GL\_POINTS);

# draw\_circle(pt, radius);

# glEnd();

# glFlush();

# }

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

# {

# glutInit(&argc, argv);

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

# glutInitWindowSize(640, 480);

# glutInitWindowPosition(200, 200);

# glutCreateWindow("Open GL");

# init();

# glutDisplayFunc(display);

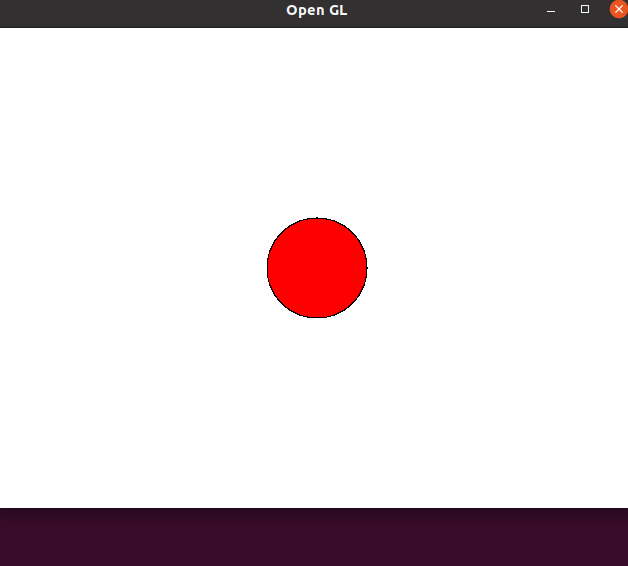
# glutMouseFunc(onMouseClick);

# glutMainLoop();

# return 0;

# }

**Output:**

****

# Lab-6(Line Clipping)

# Code:

#include <stdio.h>

#include <GL/glut.h>

#define outcode int

double xmin=50,ymin=50, xmax=100,ymax=100;

const int TOP = 1;

const int BOTTOM = 2;

const int RIGHT = 4;

const int LEFT = 8;

outcode ComputeOutCode (double x, double y);

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

{

outcode outcode0, outcode1, outcodeOut;

bool accept = false, done = false;

outcode0 = ComputeOutCode (x0, y0);

outcode1 = ComputeOutCode (x1, y1);

do

{

if (!(outcode0 | outcode1))

{

accept = true;

done = true;

}

else if (outcode0 & outcode1)

done = true;

else

{

double x, y;

outcodeOut = outcode0? outcode0: outcode1;

if (outcodeOut & TOP)

{

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

y = ymax;

}

else if (outcodeOut & BOTTOM)

{

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

y = ymin;

}

else if (outcodeOut & RIGHT)

{

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

x = xmax;

}

else

{

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

x = xmin;

}

if (outcodeOut == outcode0)

{

x0 = x;

y0 = y;

outcode0 = ComputeOutCode (x0, y0);

}

else

{

x1 = x;

y1 = y;

outcode1 = ComputeOutCode (x1, y1);

}

}

} while (!done);

if (accept)

{

glColor3f(1.0, 0.0, 0.0);

glBegin(GL\_LINE\_LOOP);

glVertex2f(xmin+200, ymin+200);

glVertex2f(xmax+200, ymin+200);

glVertex2f(xmax+200, ymax+200);

glVertex2f(xmin+200, ymax+200);

glEnd();

glColor3f(0.0,0.0,1.0); // draw blue colored clipped line

glBegin(GL\_LINES);

glVertex2d (x0+200, y0+200);

glVertex2d (x1+200, y1+200);

glEnd();

}

}

outcode ComputeOutCode (double x, double y)

{

outcode code = 0;

if (y > ymax)

code |= TOP;

else if (y < ymin)

code |= BOTTOM;

if (x > xmax)

code |= RIGHT;

else if (x < xmin)

code |= LEFT;

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);

glBegin(GL\_LINES);

glVertex2d (x0, y0);

glVertex2d (x1, y1);

glVertex2d (10,70);

glVertex2d (250,90);

glEnd();

glColor3f(0.0, 0.0, 1.0);

glBegin(GL\_LINE\_LOOP);

glVertex2d(xmin, ymin);

glVertex2d(xmax, ymin);

glVertex2d(xmax, ymax);

glVertex2d(xmin, ymax);

glEnd();

CohenSutherlandLineClipAndDraw(x0,y0,x1,y1);

CohenSutherlandLineClipAndDraw(10,70,250,90);

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);

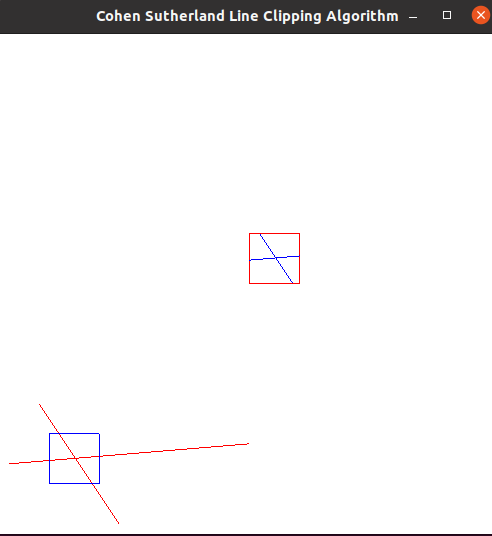
myInit();

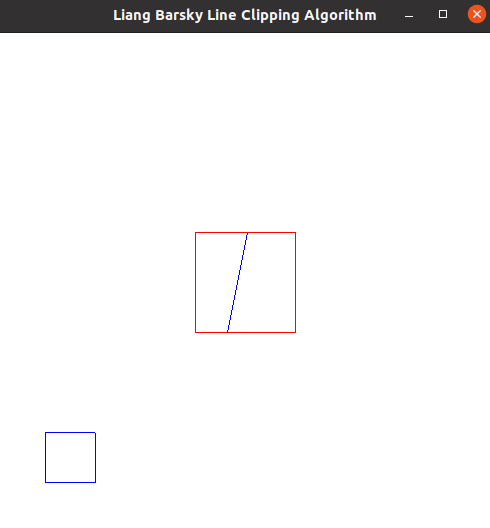
glutMainLoop();

return 0;

}

**Output**

****

****

# Lab 7(2D Transformation)

# Code:

#include<GL/glut.h>

#include<string.h>

#include<cmath>

#include<stdio.h>

#include<iostream>

#define PI 3.14

using namespace std;

float OriginalTriangle [3][3]={{150,250,350},{200,400,200},{1,1,1}};

float ModifiedTriangle [3][3]={};

static float tx=0, ty=0, theta=0, sx=0, sy=0;

void drawTriangle(float Tri[3][3])

{

  glClear(GL\_COLOR\_BUFFER\_BIT);

  glClearColor(0.0f, 0.0f, 0.0f, 0.8f);

  glBegin(GL\_POLYGON);

  glColor3f(1.0, 0.0, 0.0);

  glVertex2f(Tri[0][0],Tri[1][0]);

  glVertex2f(Tri[0][1],Tri[1][1]);

  glVertex2f(Tri[0][2],Tri[1][2]);

  glEnd();

  glFinish();

  glFlush();

}

void orignal()

{

  sx=0;sy=0;tx=0;ty=0;

  drawTriangle(OriginalTriangle);

}

void modify(float a[3][3], float b[3][3], float c[3][3])

{

  for(int i=0;i<3;i++)

  {

    for(int j=0;j<3;j++)

    {

      c[i][j]=0;

    }

  }

  for (int i=0;i<3;i++)

  {

    for(int j=0;j<3;j++)

    {

      float sum=0;

      for (int k=0;k<3;k++)

      {

        sum=sum+(b[i][k]\*a[k][j]);

      }

      c[i][j]=sum;

    }

  }

  for(int i=0;i<3;i++)

  {

    for(int j=0;j<3;j++)

    {

      OriginalTriangle[i][j]=c[i][j];

    }

  }

  drawTriangle(OriginalTriangle);

}

void showOption()

{

    cout<<"Choose the  type of 2d transformation:";

    cout<<"\t\n.Translation: a,w,s,d";

    cout<<"\t\n.Rotation: SPACEBAR";

    cout<<"\t\n.Scaling: z";

    cout<<"\t\n.Reflection: x, y";

    cout<<"\t\n.Orignal: o";

}

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

{

  if (key==97)

  {

    float b[3][3]={{1,0,-1},{0,1,0},{0,0,1}};

    modify(OriginalTriangle, b, ModifiedTriangle);

  }

  else

  if (key==115)

  {

    float b[3][3]={{1,0,0},{0,1,1},{0,0,1}};

    modify(OriginalTriangle, b, ModifiedTriangle);

  }

  else

  if (key==100)

  {

    float b[3][3]={{1,0,1},{0,1,0},{0,0,1}};

    modify(OriginalTriangle, b, ModifiedTriangle);

  }

  else

  if (key==119)

  {

    float b[3][3]={{1,0,0},{0,1,-1},{0,0,1}};

    modify(OriginalTriangle, b, ModifiedTriangle);

  }

  else

  if(key==111)

  orignal();

  if(key==122)

  {

    float b[3][3]={{2,0,0},{0,2,0},{0,0,1}};

    modify(OriginalTriangle, b, ModifiedTriangle);

  }

  else

  if(key==113)

  {

    float b[3][3]={{0.5,0,0},{0,0.5,0},{0,0,1}};

    modify(OriginalTriangle, b, ModifiedTriangle);

  }

  else

  if(key==32)

  {

    float b[3][3]={{cos(3.14/2), sin(3.14/2), 0},{-sin(3.14/2), cos(3.14/2),0},{0,0,1}};

    modify(OriginalTriangle, b, ModifiedTriangle);

  }

  else if(key==120)

  {

    float b[3][3]={{1,0,0},{0,-1,0},{0,0,1}};

    modify(OriginalTriangle, b, ModifiedTriangle);

  }

  else if(key==121)

  {

    float b[3][3]={{-1,0,0},{0,1,0},{0,0,1}};

    modify(OriginalTriangle, b, ModifiedTriangle);

  }

}

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

{

  showOption();

  glutInit(&argc, argv);

  glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

  glutInitWindowPosition(100, 100);

  glutInitWindowSize(800, 800);

  glutCreateWindow("2d Transformation");

  glMatrixMode(GL\_PROJECTION);

  glLoadIdentity();

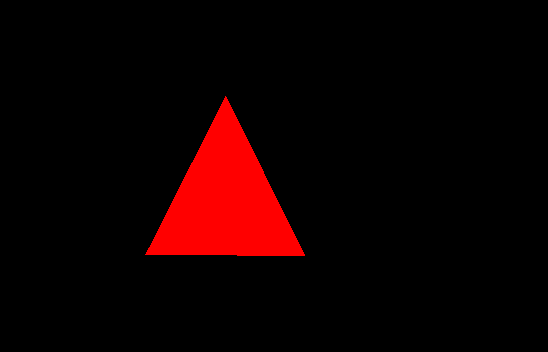
  gluOrtho2D(-500,500,500,-500);

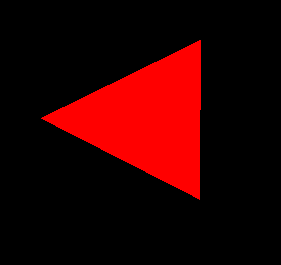
  glutKeyboardFunc(Keyboard);

  glutMainLoop();

}

**Output:**

****

****

# Lab 8(3D Transformation)

# Code:

# #include <stdio.h>

# #include <math.h>

# #include <GL/glut.h>

# float theta = 0.0f;

# float speed = 0.0f;

# void display();

# void specialKeys();

# double rotate\_y = 0;

# double rotate\_x = 0;

# double x\_axis = 0;

# double y\_axis = 0;

# double z\_axis = 0;

# double angle = 0;

# int i,p;

# float sudut = 0;

# void lighting();

# void house() {

# glBegin(GL\_QUADS);

# glColor3f (0.627, 0.322, 0.176);

# glVertex3f(0.20f, 0.05f, 0.0f);

# glVertex3f(0.20f, 0.0f, 0.0f);

# glVertex3f(0.30f, 0.0f, 0.0f);

# glVertex3f(0.30f, 0.05f, 0.0f);

# glEnd();

# glBegin(GL\_QUADS);

# glColor3f (0.627, 0.322, 0.176);

# glVertex3f(0.20f, 0.05f, 0.05f);

# glVertex3f(0.20f, 0.0f, 0.05f);

# glVertex3f(0.30f, 0.0f, 0.05f);

# glVertex3f(0.30f, 0.05f, 0.05f);

# glEnd();

# glBegin(GL\_QUADS);

# glColor3f (0.5f, 0.5f, 0.5f);

# glVertex3f(0.20f, 0.05f, 0.0f);

# glVertex3f(0.20f, 0.0f, 0.0f);

# glVertex3f(0.20f, 0.0f, 0.05f);

# glVertex3f(0.20f, 0.05f, 0.05f);

# glEnd();

# glBegin(GL\_QUADS);

# glColor3f (0.5f, 0.5f, 0.5f);

# glVertex3f(0.30f, 0.05f, 0.05f);

# glVertex3f(0.30f, 0.0f, 0.05f);

# glVertex3f(0.30f, 0.0f, 0.0f);

# glVertex3f(0.30f, 0.05f, 0.0f);

# glEnd();

# glBegin(GL\_QUADS);

# glColor3f (0.5f, 0.5f, 0.0f);

# glVertex3f(0.193f, 0.045f, -0.005f);

# glVertex3f(0.193f, 0.075f, 0.025f);

# glVertex3f(0.307f, 0.075f, 0.025f);

# glVertex3f(0.307f, 0.045f, -0.005f);

# glEnd();

# glBegin(GL\_QUADS);

# glColor3f (0.5f, 0.5f, 0.5f);

# glVertex3f(0.193f, 0.075f, 0.025f);

# glVertex3f(0.193f, 0.045f, 0.055f);

# glVertex3f(0.307f, 0.045f, 0.055f);

# glVertex3f(0.307f, 0.075f, 0.025f);

# glEnd();

# glBegin(GL\_TRIANGLES);

# glColor3f (.5f, 0.5f, 1.5f);

# glVertex3f(0.20f, 0.05f, 0.0f);

# glVertex3f(0.20f, 0.075f, 0.025f);

# glVertex3f(0.20f, 0.05f, 0.05f);

# glEnd();

# glBegin(GL\_TRIANGLES);

# glColor3f (.5f, 0.5f, 1.5f);

# glVertex3f(0.30f, 0.05f, 0.0f);

# glVertex3f(0.30f, 0.075f, 0.025f);

# glVertex3f(0.30f, 0.05f, 0.05f);

# glEnd();

# glBegin(GL\_QUADS);

# glColor3f (1.f, 1.0f, 1.5f);

# glVertex3f(0.199f, 0.035f, 0.015f);

# glVertex3f(0.199f, 0.0f, 0.015f);

# glVertex3f(0.199f, 0.0f, 0.03f);

# glVertex3f(0.199f, 0.035f, 0.03f);

# glEnd();

# glBegin(GL\_QUADS);

# glColor3f (1.f, 1.0f, 1.5f);

# glVertex3f(0.215f, 0.035f, -0.0001f);

# glVertex3f(0.215f, 0.013f, -0.0001f);

# glVertex3f(0.233f, 0.013f, -0.0001f);

# glVertex3f(0.233f, 0.035f, -0.0001f);

# glEnd();

# glBegin(GL\_QUADS);

# glColor3f (1.f, 1.0f, 1.5f);

# glVertex3f(0.265f, 0.035f, -0.0001f);

# glVertex3f(0.265f, 0.013f, -0.0001f);

# glVertex3f(0.283f, 0.013f, -0.0001f);

# glVertex3f(0.283f, 0.035f, -0.0001f);

# glEnd();

# glBegin(GL\_QUADS);

# glColor3f (1.f, 1.0f, 1.5f);

# glVertex3f(0.215f, 0.035f, 0.0501f);

# glVertex3f(0.215f, 0.013f, 0.0501f);

# glVertex3f(0.233f, 0.013f, 0.0501f);

# glVertex3f(0.233f, 0.035f, 0.0501f);

# glEnd();

# glBegin(GL\_QUADS);

# glColor3f (1.f, 1.0f, 1.5f);

# glVertex3f(0.265f, 0.035f, 0.0501f);

# glVertex3f(0.265f, 0.013f, 0.0501f);

# glVertex3f(0.283f, 0.013f, 0.0501f);

# glVertex3f(0.283f, 0.035f, 0.0501f);

# glEnd();

# }

# void field() {

# 

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

# glLoadIdentity();

# glRotatef( rotate\_x, 1.0, 0.0, 0.0);

# glRotatef( rotate\_y, 0.0, 1.0, 0.0);

# glPushMatrix();

# glScaled(2.5f ,2.5f ,2.5f);

# house();

# glPopMatrix();

# glFlush();

# glutSwapBuffers();

# theta = theta + 3.f + speed ;

# sudut = sudut + .5f;

# glutPostRedisplay();

# }

# double distance = 0.01;

# void speedkey(unsigned char key, int x , int y) {

# switch(key)

# {

# case 'q' :

# {

# z\_axis += distance ;

# speed += 1.5;

# printf("speed : %lf \n",speed );

# break;

# }

# case 'e' :

# {

# z\_axis -= distance ;

# speed -= 1.5;

# break;

# }

# }

# if (key == 97) {

# x\_axis -= distance;

# }

# else if (key == 100) {

# x\_axis += distance; }

# else if (key == 119) {

# y\_axis += distance; }

# else if (key == 115)

# y\_axis -= distance;

# glutPostRedisplay();

# if (key == 27)

# {

# glutLeaveGameMode() ;

# exit(0) ;

# }

# }

# void specialKeys( int key, int x, int y) {

# if (key == GLUT\_KEY\_RIGHT){

# rotate\_y += 5;

# }

# else if (key == GLUT\_KEY\_LEFT){

# rotate\_y -= 5;

# }

# else if (key == GLUT\_KEY\_UP){

# rotate\_x += 5;

# }

# else if (key == GLUT\_KEY\_DOWN){

# rotate\_x -= 5;

# }

# glutPostRedisplay();

# }

# void pindah( unsigned char key, int x, int y) {

# if (key == 97) {

# x\_axis -= distance; }

# else if (key == 100) {

# x\_axis += distance; }

# else if (key == 119) {

# y\_axis += distance; }

# else if (key == 115)

# y\_axis -= distance;

# glutPostRedisplay();

# if (key == 27)

# {

# glutLeaveGameMode() ;

# exit(0) ;

# }

# }

# void init(void)

# {

# glEnable (GL\_DEPTH\_TEST);

# glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_FILL);

# }

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

# {

# glutInit(&argc,argv);

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

# //glutInitWindowPosition(80,0);

# //glutInitWindowSize(1200,800);

# //glutCreateWindow("UTS")

# glEnable(GL\_DEPTH\_TEST);

# glutEnterGameMode();

# glutDisplayFunc(field);

# //lighting();

# //init();

# glutSpecialFunc(specialKeys);

# glutKeyboardFunc(speedkey);

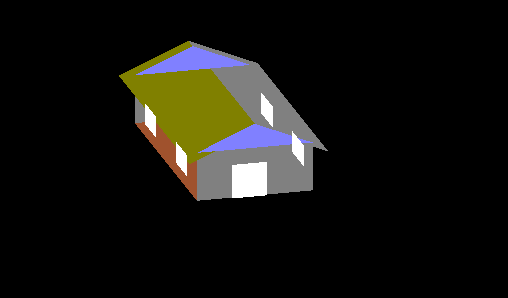
# //glutKeyboardFunc(pindah);

# glutMainLoop();

# return 0;

# }

# Output:

****

**VIVA 2:**

**BEIZER CURVE:**

**Code:**

#include<GL/glut.h>

#include<math.h>

#include<stdio.h>

GLfloat ya = 20;

int yb = 1, Animation=1;

void Animate()

{

if(Animation == 1)

{

if(ya>-20 && yb== 1)

ya= ya - 0.2;

if(ya<=-20 && yb == 1)

yb = 0;

if(ya<50 && yb == 0)

ya = ya + 0.2;

if(ya>=50 && yb == 0)

yb= 1;

}

glutPostRedisplay();

}

void Draw()

{

GLfloat x[4],y1[4],x1[4],x2[4],x4[4],x5[4],x6[4],x7[4],y2[4],y3[4],y4[4],y5[4],y6[4],y7[4];

GLdouble xt[200],y1t[200],x1t[200],x2t[200],x4t[100],x5t[100],x6t[200],x7t[200],y2t[200],y3t[200],y4t[100],y5t[100],y6t[200],y7t[200],t;

int i,c;

glClear(GL\_COLOR\_BUFFER\_BIT);

x[0] = 100; x[1] = 170; x[2] = 200; x[3] = 130;

y1[0] = 260; y1[1] = 220-ya; y1[2] = 220-ya; y1[3] = 300;

x1[0] = 280; x1[1] = 280; x1[2] = 280; x1[3] = 280;

y2[0] = 125; y2[1] = 165+ya; y2[2] = 165+ya; y2[3] = 85;

x2[0] = 100; x2[1] = 100; x2[2] = 100; x2[3] = 100;

y3[0] = 260; y3[1] = 260-ya; y3[2] = 210-ya; y3[3] = 125;

x4[0] = 130; x4[1] = 200; x4[2] = 200; x4[3] = 250;

y4[0] = 300; y4[1] = 300; y4[2] = 300; y4[3] = 300;

x5[0] = 250; x5[1] = 200; x5[2] = 225; x5[3] = 280;

y5[0] = 300; y5[1] = 220-ya; y5[2] = 220-ya; y5[3] = 260;

y6[0] = 85; y6[1] = 85; y6[2] = 85; y6[3] = 85;

y7[0] = 85; y7[1] = 165+ya; y7[2] = 165+ya; y7[3] = 125;

for(i=0,t=0,c=0;t<1;i++,t=t+0.01)

{

xt[i] = pow(1-t,3)\*x[0]+3\*t\*pow(1-t,2)\*x[1]+3\*pow(t,2)\*(1-t)\*x[2]+pow(t,3)\*x[3];

y1t[i] = pow(1-t,3)\*y1[0]+3\*t\*pow(1-t,2)\*y1[1]+3\*pow(t,2)\*(1-t)\*y1[2]+pow(t,3)\*y1[3];

x1t[i] = pow(1-t,3)\*x1[0]+3\*t\*pow(1-t,2)\*x1[1]+3\*pow(t,2)\*(1-t)\*x1[2]+pow(t,3)\*x1[3];

y2t[i] = pow(1-t,3)\*y2[0]+3\*t\*pow(1-t,2)\*y2[1]+3\*pow(t,2)\*(1-t)\*y2[2]+pow(t,3)\*y2[3];

x2t[i] = pow(1-t,3)\*x2[0]+3\*t\*pow(1-t,2)\*x2[1]+3\*pow(t,2)\*(1-t)\*x2[2]+pow(t,3)\*x2[3];

y3t[i] = pow(1-t,3)\*y3[0]+3\*t\*pow(1-t,2)\*y3[1]+3\*pow(t,2)\*(1-t)\*y3[2]+pow(t,3)\*y3[3];

x4t[i] = pow(1-t,3)\*x4[0]+3\*t\*pow(1-t,2)\*x4[1]+3\*pow(t,2)\*(1-t)\*x4[2]+pow(t,3)\*x4[3];

y4t[i] = pow(1-t,3)\*y4[0]+3\*t\*pow(1-t,2)\*y4[1]+3\*pow(t,2)\*(1-t)\*y4[2]+pow(t,3)\*y4[3];

x5t[i] = pow(1-t,3)\*x5[0]+3\*t\*pow(1-t,2)\*x5[1]+3\*pow(t,2)\*(1-t)\*x5[2]+pow(t,3)\*x5[3];

y5t[i] = pow(1-t,3)\*y5[0]+3\*t\*pow(1-t,2)\*y5[1]+3\*pow(t,2)\*(1-t)\*y5[2]+pow(t,3)\*y5[3];

y6t[i] = pow(1-t,3)\*y6[0]+3\*t\*pow(1-t,2)\*y6[1]+3\*pow(t,2)\*(1-t)\*y6[2]+pow(t,3)\*y6[3];

y7t[i] = pow(1-t,3)\*y7[0]+3\*t\*pow(1-t,2)\*y7[1]+3\*pow(t,2)\*(1-t)\*y7[2]+pow(t,3)\*y7[3];

c++;

}

glPointSize(3);

glColor3f(1,0,0);

//glRotatef( 15.0,0.0,0.0,1.0);

glScalef(0.4,0.4,0);

//glTranslatef(0.2,0.1,0);

glBegin(GL\_POINTS);

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

{

glVertex2d(xt[i],y1t[i]);

glVertex2d(x1t[i],y3t[i]);

glVertex2d(x2t[i],y3t[i]);

glVertex2d(xt[i],y2t[i]);

glVertex2d(x4t[i],y4t[i]);

glVertex2d(x5t[i],y5t[i]);

glVertex2d(x4t[i],y6t[i]);

glVertex2d(x5t[i],y7t[i]);

}

glEnd();

glFlush();

}

void Menu(int n)

{

if(n == 1)

Animation = 1;

else if(n == 2)

Animation= 0;

if(n == 3)

exit(0);

glutPostRedisplay();

}

void MyInit()

{

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0,500,0,500);

glMatrixMode(GL\_MODELVIEW);

glutCreateMenu(Menu);

glutAddMenuEntry("Start the Animation",1);

glutAddMenuEntry("Stop the Animation",2);

glutAddMenuEntry("Exit",3);

glutAttachMenu(GLUT\_RIGHT\_BUTTON);

}

int main(int argC,char \*argV[])

{

glutInit(&argC,argV);

glutInitWindowSize(1000,700);

glutInitWindowPosition(100,100);

glutInitDisplayMode(GLUT\_RGB | GLUT\_SINGLE);

glutCreateWindow("THRIPLE H");

MyInit();

glutDisplayFunc(Draw);

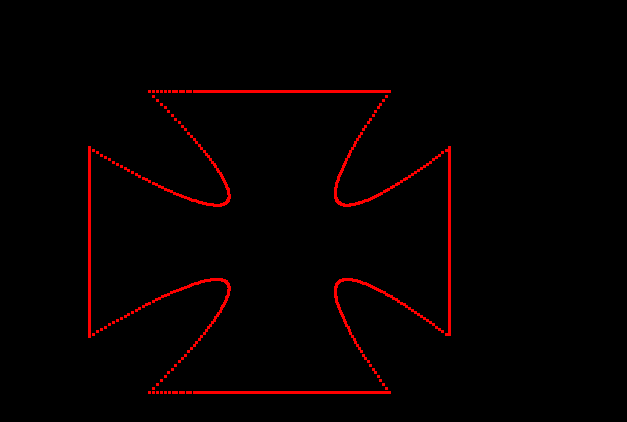
// glutIdleFunc(Animate);

glutMainLoop();

return 0;

}

**Output:**



**VIVA 3:**

**FRACTALS**

**Code:**

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

#include <stdlib.h>

#include <stdio.h>

#include <math.h>

#define dimension 800 // window size

int levelmax = 9; // maximum level of recursion

int nleaf = 2; // number of leafs per branch

float openness = 4.5; // controls how open the leafs are

float factor = 1.5; // controls how fast branch sizes decrease

float zoom = 1; // zoom of visualization

float offset = 0; // controls how crooked branches are

float windowx = 0.5, windowy = 0.5; // position of center of window

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

{

y = dimension - y;

if(button == 0 && state == 1)

{

windowx += (x-dimension/2.)/dimension/zoom;

windowy += (y-dimension/2.)/dimension/zoom;

}

glutPostRedisplay();

}

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

{

switch(key)

{

case 'a':

openness \*= 1/1.01;

glutPostRedisplay();

break;

case 'd':

openness \*= 1.01;

glutPostRedisplay();

break;

case 'z':

openness \*= 1/1.1;

glutPostRedisplay();

break;

case 'c':

openness \*= 1.1;

glutPostRedisplay();

break;

case 'r':

levelmax = 9; // maximum level of recursion

nleaf = 2; // number of leafs per branch

openness = 4.5; // controls how open the leafs are

factor = 1.5; // controls how fast branch sizes decrease

zoom = 1; // zoom of visualization

offset = 0; // controls how crooked branches are

windowx = 0.5, windowy = 0.5; // position of center of window

glutPostRedisplay();

break;

case '3':

nleaf += 1;

glutPostRedisplay();

break;

case '1':

nleaf -= 1;

glutPostRedisplay();

break;

case '.':

levelmax += 1;

glutPostRedisplay();

break;

case ',':

if(levelmax > 2)

{

levelmax -= 1;

glutPostRedisplay();

}

break;

case 'l':

offset -= M\_PI/100;

glutPostRedisplay();

break;

case 'k':

offset += M\_PI/100;

glutPostRedisplay();

break;

case 'e':

factor \*= 1/1.1;

glutPostRedisplay();

break;

case 'q':

factor \*= 1.1;

glutPostRedisplay();

break;

case 'o':

zoom \*= 1/1.1;

glutPostRedisplay();

break;

case 'p':

zoom \*= 1.1;

glutPostRedisplay();

break;

}

}

void init (void)

{

glClearColor(0.0, 0.0, 0.0, 0.0);

glShadeModel(GL\_FLAT);

glPixelStorei(GL\_UNPACK\_ALIGNMENT, 1);

}

void draw\_fractal(double x0, double y0, double angle0, int level)

{

if(level > levelmax) return;

float r = 0.15/pow(factor,level-1);

float angle;

float x, y, coss, sinn;

int i;

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

{

angle = offset + angle0 + i\*2\*M\_PI/nleaf/openness - (nleaf-1)\*2\*M\_PI/nleaf/openness/2;

coss = cos(angle);

sinn = sin(angle);

x = x0 + r\*coss;

y = y0 + r\*sinn;

glBegin(GL\_LINES);

glVertex2f(x0, y0);

glVertex2f(x, y);

glEnd();

draw\_fractal(x, y, atan2((y-y0), (x-x0)), level+1);

}

}

void image(void)

{

float tmp = 0.5/zoom;

glClear(GL\_COLOR\_BUFFER\_BIT);

glLoadIdentity();

gluOrtho2D(windowx-tmp, windowx+tmp, windowy-tmp, windowy+tmp);

draw\_fractal(0.5, 0.5, M\_PI/2, 1);

glFlush();

}

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

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(dimension, dimension);

glutInitWindowPosition(0, 0);

glutCreateWindow(argv[0]);

init();

glutDisplayFunc(image);

glutKeyboardFunc(keyboard);

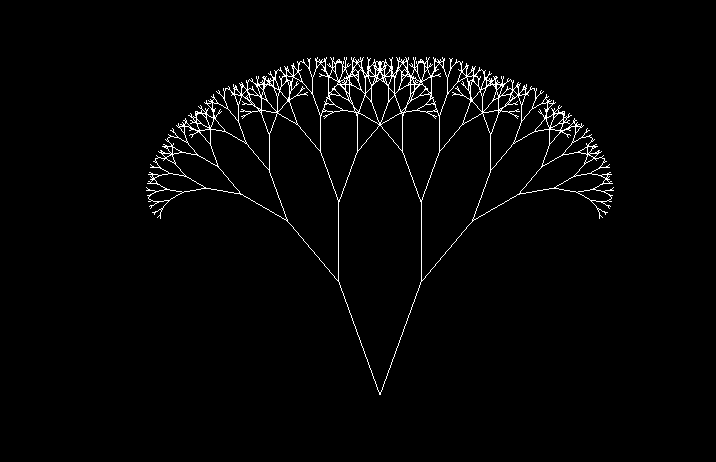
glutMouseFunc(mouse);

glutMainLoop();

return 0;

}

**Output:**

****