**Computer Graphics**

1. **Bresenham’s Line Drawing Algorithm:**

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

**void myinit()**

**{**

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

**glClearColor( 0, 0, 0, 1 );**

**gluOrtho2D(0,500,0,500);**

**}**

**void draw\_pixel(int x,int y)**

**{**

**glBegin(GL\_POINTS);**

**glVertex2d(x,y);**

**glEnd();**

**}**

**void bresenhams(int x1,int y1,int x2,int y2)**

**{**

**int dx,dy,x,y,p0,p,i,incx=1,incy=1;**

**dx=abs(x2-x1);**

**dy=abs(y2-y1);**

**if(x2<x1)incx=-1;**

**if(y2<y1)incy=-1;**

**x=x1;**

**y=y1;**

**if(dx>dy)**

**{**

**draw\_pixel(x,y);**

**p=2\*dy-dx;**

**for(i=0;i<dx;i++)**

**{**

**x=x+incx;**

**if(p>=0)**

**{**

**y=y+incy;**

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

**}**

**else**

**{**

**y=y;**

**p=p+2\*dy;**

**}**

**draw\_pixel(x,y);**

**}**

**}**

**else**

**{**

**draw\_pixel(x,y);**

**p=2\*dx-dy;**

**for(i=0;i<dy;i++)**

**{**

**y=y+incy;**

**if(p>=0)**

**{**

**x=x+incx;**

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

**}**

**else**

**{**

**x=x;**

**p=p+2\*dx;**

**}**

**draw\_pixel(x,y);**

**}**

**}**

**}**

**void display()**

**{**

**glColor3f( 1, 0, 0 );**

**bresenhams(20,20,300,50); //Slope <1**

**bresenhams(20,20,50,300); //slope >1**

**bresenhams(20,20,300,300); //slope=1**

**bresenhams(50,300,20,20); //Negative slope >1**

**bresenhams(300,50,20,20); // Negative slope <1**

**glFlush();**

**}**

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

**{**

**glutInit(&argc, argv);**

**glutInitDisplayMode(GLUT\_SINGLE);**

**glutInitWindowSize(500,500);**

**glutInitWindowPosition(100,100);**

**glutCreateWindow("Triangle");**

**glutDisplayFunc(display);**

**myinit();**

**glutMainLoop();**

**return 0;**

**}**

**Ex 2: 2 Basic Geometric Operations – 2D Object**

**Code:**

#include<GL/glut.h>

#include<stdio.h>

#include <math.h>

float x[3][3]={{0,100,50},{0,0,50},{1,1,1}};

float r[3][3];

void myinit()

{

glClearColor(1,1,1,0);

gluOrtho2D(-100,500,-100,500);

}

void triangle(float x[3][3])

{

glColor4s(1,1,1,0);

glBegin(GL\_TRIANGLES);

glVertex2f(x[0][0],x[1][0]);

glVertex2f(x[0][1],x[1][1]);

glVertex2f(x[0][2],x[1][2]);

glEnd();

}

void matrixmul(float mul[3][3]){

for (int i=0;i<3;i++)

for(int j=0;j<3;j++)

{

r[i][j]=0;

for (int k=0;k<3;k++)

r[i][j]=r[i][j]+mul[i][k]\*x[k][j];

}

}

void translation(){

float t[3][3]={{1,0,100},{0,1,0},{0,0,1}};

printf("enter the values of tx and ty");

scanf("%f %f",&t[0][2],&t[1][2]);

matrixmul(t);

triangle(r);

}

void scaling(){

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

printf("enter the values of sx and sy");

scanf("%f %f",&s[0][0],&s[1][1]);

matrixmul(s);

triangle(r);

}

void rotation()

{

float theta=0;

printf("enter the angle");

scanf("%f",&theta);

float angle=theta \*3.14/180;

float cosx=cos(angle);

float sinx=sin(angle);

float rr[3][3]={{cosx,-sinx,0},{sinx,cosx,0},{0,0,1}};

matrixmul(rr);

triangle(r);

}

void displayMe()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3d(1,0,0);

int ch;

printf("enter the choice \n0 for normal triangle \n1 for translation\n2 for scaling\n3 for rotation\n");

scanf("%d",&ch);

glColor3d(1,1,1);

switch(ch)

{

case 0:

triangle(x);

break;

case 1:

translation();

break;

case 2:

scaling();

break;

case 3:

rotation();

break;

default:

printf("enter a valid choice");

}

glColor3d(1,0,0);

triangle(x);

glFlush();

}

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

{

glutInit(&argc,argv);

glutInitDisplayMode(GLUT\_SINGLE);

glutInitWindowSize(500,500);

glutCreateWindow("Line Drawing Algorithm");

myinit();

glutDisplayFunc(displayMe);

glutMainLoop();

return 0;

}

## Conclusion:

The OpenGL program demonstrates rotation, transformation and Scaling the object implemented and executed successfully.

**Program-3**

**Develop a program to demonstrate basic geometric operations on the 3D object.**

**Code:**

#include <GL/glut.h>

#include <stdlib.h>

#include<stdio.h>

typedef float point[3];

point v[]={{0.0,0.0,1.0},

{0.0,1.0,0.0},

{-1.0,-1.0,0.0},

{1.0,-1.0,0.0}};

int n;

void triangle(point a,point b,point c)

{

glBegin(GL\_TRIANGLES);

glVertex3fv(a);

glVertex3fv(b);

glVertex3fv(c);

glEnd();

}

void divide\_tri(point a,point b,point c,int m)

{

point v1,v2,v3; int j;

if (m>0)

{

for(j=0;j<3;j++)

v1[j]=(a[j]+b[j])/2;

for(j=0;j<3;j++)

v2[j]=(a[j]+c[j])/2;

for(j=0;j<3;j++)

v3[j]=(b[j]+c[j])/2;

divide\_tri(a,v1,v2,m-1);

divide\_tri(c,v2,v3,m-1);

divide\_tri(b,v3,v1,m-1);

}

else

triangle(a,b,c);

}

void tetrahedron(int m)

{

glColor3f(1.0,0.0,0.0);

divide\_tri(v[0],v[1],v[2],m);

glColor3f(0.0,0.0,0.0);

divide\_tri(v[3],v[2],v[1],m);

glColor3f(0.0,1.0,.0);

divide\_tri(v[0],v[3],v[1],m);

glColor3f(0.0,0.0,1.0);

divide\_tri(v[0],v[2],v[3],m);

}

void display()

{

tetrahedron(n);

glFlush();

}

void myinit()

{

glClearColor(1.0,1.0,1.0,1.0);

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

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

}

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

{

printf("\nEnter the number of recursive steps you want");

scanf("%d", &n);

glutInit(&argc,argv);

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

glutInitWindowSize(500,500);

glutCreateWindow("Ex 8: 3d Sierpinski's Gasket");

glutDisplayFunc(display);

myinit();

glEnable(GL\_DEPTH\_TEST);

glutMainLoop();

return 0;

}

**Program -4 Develop a program to demonstrate 2D transformation on basic objects**

**Code**:

#include<GL/glut.h>

#include<stdio.h>

void myinit()

{

gluOrtho2D(-500,500,-500,500);

}

void drawtriangle()

{

glBegin(GL\_POLYGON);

glVertex2f(100,100);

glVertex2f(200,100);

glVertex2f(150,150);

glEnd();

}

void translate()

{

glPushMatrix();

glTranslated(100,0,0);

drawtriangle();

glPopMatrix();

}

void rotate\_triangle()

{

glPushMatrix();

glRotated(45,0,0,1);

drawtriangle();

glPopMatrix();

}

void pivot\_point\_rotate()

{ glColor3f(1,1,0); // yellow

glPushMatrix();

glTranslated(100,100,0); //translate back to the original position

glRotated(45,0,0,1); // Rotate degree 45

glTranslated(-100,-100,0); //translate to Origin

drawtriangle();

glPopMatrix();

}

void scale\_triangle()

{

glPushMatrix();

glScaled(2,2,1);

drawtriangle();

glPopMatrix();

}

void pivot\_point\_scale()

{ glColor3f(1,1,0); // yellow

glPushMatrix();

glTranslated(100,100,0);

glScaled(2,2,1);

glTranslated(-100,-100,0);

drawtriangle();

glPopMatrix();

}

void display()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glClearColor(1,1,1,0);

glColor3f(1,0,0); //Red

drawtriangle();

//glutPostRedisplay();

glFlush();

}

void menu\_rotate(int id)

{

switch(id)

{

case 1:

translate();

break;

case 2:

rotate\_triangle();

break;

case 3:

pivot\_point\_rotate();

break;

case 4:

scale\_triangle();

break;

case 5:

pivot\_point\_scale();

break;

default:

exit(0);

}

//glutPostRedisplay();

}

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

{

glutInit(&argc,argv);

glutInitDisplayMode(GLUT\_SINGLE|GLUT\_RGB);

glutInitWindowSize(500,500);

glutCreateWindow("Transformation");

myinit();

glutDisplayFunc(display);

glutCreateMenu(menu\_rotate);

glutAddMenuEntry("Translate",1);

glutAddMenuEntry("Rotation About origin",2);

glutAddMenuEntry("Rotation About Fixed Point",3);

glutAddMenuEntry("Scale About Origin",4);

glutAddMenuEntry("Scale About Fixed Point",5);

glutAddMenuEntry("EXIT",6);

glutAttachMenu(GLUT\_RIGHT\_BUTTON);

glutMainLoop();

return 0;

}

**Animation Effect on Simple Objects**

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

**float ambient[]={1,1,1,1}; float light\_pos[]={27,80,2,3};**

**void obj(double tx,double ty,double tz,double sx,double sy,double sz)**

**{**

**glRotated(50,0,1,0); glRotated(10,-1,0,0);**

**glRotated(11.7,0,0,-1);**

**glTranslated(tx,ty,tz); glScaled(sx,sy,sz); glutSolidCube(1); glLoadIdentity();**

**}**

**void display()**

**{**

**glViewport(0,0,700,700); glClear(GL\_COLOR\_BUFFER\_BIT|GL\_DEPTH\_BUFFER\_BIT);**

**obj(0,0,0.5,1,1,0.04); // three walls obj(0,-0.5,0,1,0.04,1);**

**obj(-0.5,0,0,0.04,1,1);**

**obj(0,-0.3,0,0.02,0.2,0.02); // four table legs**

**obj(0,-0.3,-0.4,0.02,0.2,0.02);**

**obj(0.4,-0.3,0,0.02,0.2,0.02);**

**obj(0.4,-0.3,-0.4,0.02,0.2,0.02);**

**obj(0.2,-0.18,-0.2,0.6,0.02,0.6); // table top glRotated(50,0,1,0);**

**glRotated(10,-1,0,0);**

**glRotated(11.7,0,0,-1);**

**glTranslated(0.3,-0.1,-0.3); glutSolidTeapot(0.09);// tea pot glFlush();**

**glLoadIdentity();**

**}**

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

**{**

**glutInit(&argc, argv); glutInitDisplayMode(GLUT\_SINGLE|GLUT\_RGB|GLUT\_DEPTH); glutInitWindowSize(700,700);**

**glutCreateWindow("Teapot");**

**glutDisplayFunc(display);**

**glEnable(GL\_LIGHTING);**

**glEnable(GL\_LIGHT0);**

**glMaterialfv(GL\_FRONT,GL\_AMBIENT,ambient);**

**glLightfv(GL\_LIGHT0,GL\_POSITION,light\_pos); glEnable(GL\_DEPTH\_TEST);**

**glutMainLoop();**

**}**

**3D Animation**

**Image Processing**

**Ex: 7**

**import cv2**

**import matplotlib.pyplot as plt**

**image = cv2.imread('index.jpeg')**

**image\_mat= cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)**

**ht,wd,c=image.shape**

**midy=ht//2**

**midx=wd//2**

**tl=image\_mat[:midy,:midx]**

**tr=image\_mat[:midy,midx:]**

**bl=image\_mat[midy:,:midx]**

**br=image\_mat[midy:,midx:]**

**fig,axs=plt.subplots(2,2)**

**l\_title=["Top Left","Top Right","Bottom Left","Bottom Right"]**

**l\_var=[tl,tr,bl,br]**

**k=0**

**for i in range(2):**

**for j in range(2):**

**axs[i,j].imshow(l\_var[k])**

**axs[i,j].set\_title(l\_title[k])**

**axs[i,j].axis("off")**

**k=k+1**

**plt.axis("off")**

**plt.show()**

**Ex: 9 - Edge Detection**

**import cv2**

**import numpy as np**

**from matplotlib import pyplot as plt**

**image = cv2.imread('index.jpeg')**

**image\_mat= cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)**

**gray\_image = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)**

**sobel\_x = np.array([[-1, 0, 1],**

**[-2, 0, 2],**

**[-1, 0, 1]])**

**sobel\_y = np.array([[-1, -2, -1],**

**[ 0, 0, 0],**

**[ 1, 2, 1]])**

**edges\_x = cv2.filter2D(gray\_image, -1, sobel\_x)**

**edges\_y = cv2.filter2D(gray\_image, -1, sobel\_y)**

**edges = cv2.addWeighted(edges\_x, 0.5, edges\_y, 0.5, 0)**

**edges\_rgb = cv2.cvtColor(edges, cv2.COLOR\_BGR2RGB)**

**sobelx = cv2.Sobel(gray\_image, cv2.CV\_64F, 1, 0, ksize=5)**

**sobely = cv2.Sobel(gray\_image, cv2.CV\_64F, 0, 1, ksize=5)**

**texture=sobelx+sobely**

**l\_title=["Original","Edge Detection","Texture Extraction"]**

**l\_var=[image\_mat,edges\_rgb,texture]**

**fig,axs=plt.subplots(1,3)**

**for i in range(3):**

**axs[i].imshow(l\_var[i])**

**axs[i].set\_title(l\_title[i])**

**plt.show()**

**—--------------------------------------------------------------------------------------------------**

**Exercise – 10 : Write a program to blur and smoothing an image**

import cv2

import numpy as np

import matplotlib.pyplot as plt

from skimage.metrics import structural\_similarity as ssim

image = cv2.imread('fruit.jpg')

image\_rgb = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

kernel\_size = 9

blur\_kernel = np.ones((kernel\_size, kernel\_size), dtype=np.float32) / (kernel\_size \* kernel\_size)

blurred = cv2.filter2D(image\_rgb, -1, blur\_kernel)

smooth\_kernel =

np.array([[2, 2, 2],

[2, 10, 2],

[2, 2, 2]], dtype=np.float32) / 13

smoothed = cv2.filter2D(image\_rgb, -1, smooth\_kernel)

gray\_original = cv2.cvtColor(image\_rgb, cv2.COLOR\_RGB2GRAY)

gray\_blurred = cv2.cvtColor(blurred, cv2.COLOR\_RGB2GRAY)

gray\_smoothed = cv2.cvtColor(smoothed, cv2.COLOR\_RGB2GRAY)

ssim\_original\_blurred, \_ = ssim(gray\_original, gray\_blurred, full=True)

ssim\_original\_smoothed, \_ = ssim(gray\_original, gray\_smoothed, full=True)

print(f'SSIM between original and blurred images: {ssim\_original\_blurred:.4f}')

print(f'SSIM between original and smoothed images: {ssim\_original\_smoothed:.4f}')

fig, axs = plt.subplots(1,3, figsize=(12, 10))

axs[0].imshow(image\_rgb)

axs[0].set\_title('Original Image')

axs[0].axis('off')

axs[1].imshow(blurred)

axs[1].set\_title('Blurred Image')

axs[1].axis('off')

axs[2].imshow(smoothed)

axs[2].set\_title('Smoothed Image')

axs[2].axis('off')

plt.tight\_layout()

plt.show()

**Output:**



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Exercise 11: Write a program to Contour an Image**

import cv2

import numpy as np

import matplotlib.pyplot as plt

image = cv2.imread('shape.jpg')

gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

# Perform Canny edge detectio

edged = cv2.Canny(gray, 30, 200)

# Finding Contours

contours, hierarchy = cv2.findContours(edged.copy(), cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

print("Number of Contours found = " + str(len(contours)))

# Draw all contours on the original image

cv2.drawContours(image, contours, -1, (0, 255, 0), 3)

# Create a list to store cropped images of each object

cropped\_images = []

# Iterate through contours

for i, contour in enumerate(contours):

# Create a mask image for each contour

mask = np.zeros\_like(gray)

cv2.drawContours(mask, [contour], 0, 255, -1)

# Extract the object using the mask

object\_extracted = np.zeros\_like(image)

object\_extracted[mask == 255] = image[mask == 255]

# Convert BGR to RGB for displaying with Matplotlib

object\_extracted\_rgb = cv2.cvtColor(object\_extracted, cv2.COLOR\_BGR2RGB)

# Append the extracted object to the list

cropped\_images.append(object\_extracted\_rgb)

fig, axs = plt.subplots(1, len(cropped\_images)+2, figsize=(12, 4))

# Plot the original image

axs[0].imshow(cv2.cvtColor(image, cv2.COLOR\_BGR2RGB))

axs[0].set\_title('Original Image')

axs[0].axis('off')

axs[1].imshow(edged, cmap='gray')

axs[1].set\_title('Canny Edges')

axs[1].axis('off')

for i in range(len(cropped\_images)):

axs[i+2].imshow(cropped\_images[i])

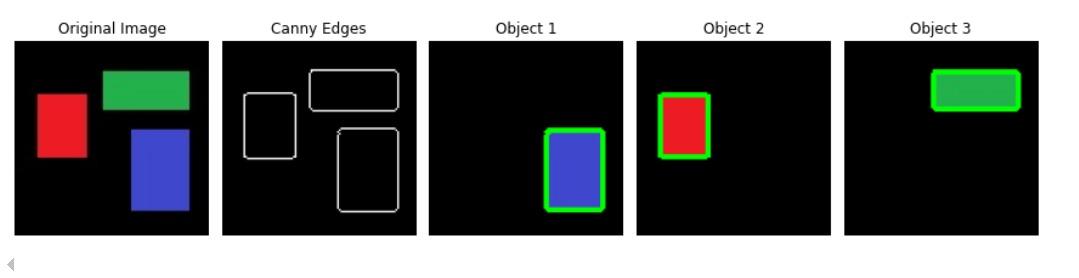
axs[i+2].set\_title(f'Object {i+1}')

axs[i+2].axis('off')

plt.tight\_layout()

plt.show()

**Output**



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Exercise – 12: Write a program to detect a face/s in an Image**

import cv2

import matplotlib.pyplot as plt

# Load the image

image\_path = 'cricket.jpg'

image = cv2.imread(image\_path)

face\_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade\_frontalface\_default.xml')

gray\_image = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

# Detect faces in the image

faces = face\_cascade.detectMultiScale(gray\_image, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))

# Initialize a list to store cropped faces

cropped\_faces = []

# Draw rectangles around the detected faces and crop them

for (x, y, w, h) in faces:

cv2.rectangle(image, (x, y), (x+w, y+h), (255, 0, 0), 2)

cropped\_faces.append(image[y:y+h, x:x+w])

# Display each cropped face separately

plt.figure(figsize=(12, 6))

for i, face in enumerate(cropped\_faces):

plt.subplot(1, len(cropped\_faces), i + 1)

plt.imshow(cv2.cvtColor(face, cv2.COLOR\_BGR2RGB))

plt.axis('off')

plt.title(f'Face {i + 1}')

plt.tight\_layout()

plt.show()



**Computer Graphics - Animation of 3D Objects**

#include <GL/glut.h>

float ambient[]={1,0,0,1};

float light\_pos[]={2,2,2,1};

static float theta[3] = {0,0,0};

int axis = 0;

int ch=1;

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

{

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

axis = 0;

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

axis = 1;

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

axis = 2;

}

void idle(){

theta[axis] += 2;

if(theta[axis] > 360)

theta[axis] = 0;

glutPostRedisplay();

}

void display()

{

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

glClearColor(1,1,1,1);

glLoadIdentity();

glRotatef(theta[0],1,0,0); // rotation about x

glRotatef(theta[1],0,1,0); // rotate about y

glRotatef(theta[2],0,0,1); // rotate about z

if(ch==1)

glutSolidCube(1);

if(ch==2)

glutSolidTeapot(0.5);

if(ch==3)

glutSolidCone(0.5,0.5,20,20);

glFlush();

glutSwapBuffers(); // use whenever you use double buffer

}

void menu(int id)

{

switch(id)

{

case 1:

ch=1;

break;

case 2:

ch=2;

break;

case 3:

ch=3;

break;

}

}

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

{

glutInit(&argc,argv);

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

glutInitWindowSize(500,500);

glutCreateWindow("Color Cube");

glutCreateMenu(menu);

glutAddMenuEntry("Cube",1);

glutAddMenuEntry("Teapot",2);

glutAddMenuEntry("Cone",3);

glutAttachMenu(GLUT\_RIGHT\_BUTTON);

glutDisplayFunc(display);

glEnable(GL\_LIGHTING);

glEnable(GL\_LIGHT0);

glMaterialfv(GL\_FRONT,GL\_AMBIENT,ambient);

glLightfv(GL\_LIGHT0,GL\_POSITION,light\_pos);

glutMouseFunc(mouse);

glutIdleFunc(idle);

glShadeModel(GL\_SMOOTH);

glEnable(GL\_DEPTH\_TEST);

glutMainLoop();

return 0;

}