SRI SIVASUBRAMANIYA NADAR COLLEGE OF ENGINEERING

(AN AUTONOMOUS INSTITUTION, AFFILIATED TO ANNA UNIVERSITY)

Rajiv Gandhi Salai (OMR), Kalavakkam - 603 110.

LABORATORY RECORD

NAME	: Sabarivasan Velayutham
Reg. No.	: 205001085
Dept.	: CSE Sem. : VII Sec. : B

SSN

SRI SIVASUBRAMANIYA NADAR
COLLEGE OF ENGINEERING, CHENNAI

(AN AUTONOMOUS INSTITUTION, AFFILIATED TO ANNA UNIVERSITY)

BONAFIDE CERTIFICATE

Certified that this is the bonafide record of the practical work done in the

UCS1712 Graphics and Multimedia Laboratory by

Name ....Sabarivasan Velayutham....

Register Number ....205001085....

Semester ....VII....

Branch ....CSE....

Sri Sivasubramaniya Nadar College of Engineering, Kalavakkam.

During the Academic year 2023-2024

[illegible] 23/11/23

Faculty

[illegible]

Head of the Department

Submitted for the....End Semester....Practical Examination held at SSNCE on....28/11/2023

Internal Examiner

External Examiner

INDEX

Name : Sabarivasan Velayutham Reg. No. 205001085

Sem : VII Sec : B

Ex. No.	Date of Expt.	Title of the Experiment	Page No.	Signature of the Faculty	Remarks
1	16/8/23	Study of Basic Output Primitives	1		
2	30/8/23	DDA Line Drawing Algorithm	14	[illegible]	
3	4/9/23	Bresenham's Line Drawing Algorithm	24		
4	12/9/23	Midpoint Circle Drawing Algorithm	35		
5	24/9/23	2D Transformations	39	[illegible]	
6	29/9/23	2D Composite Transformation	51		
7	4/10/23	Cohen Southerland Line Clipping	57		
8	12/10/23	3D Transformations	65	[illegible]	
9	19/10/23	3D Projections	76		
10	25/10/23	Creating a 3D scene	82		
11	04/10/23	Image Editing and Manipulation	89		
12	14/10/23	Creating 2D Animations	92	[illegible]	
13	18/11/23	Mini project	93		

GRAPHICS LAB
EX 1

205001085
SABARIVASAN V

AIM:

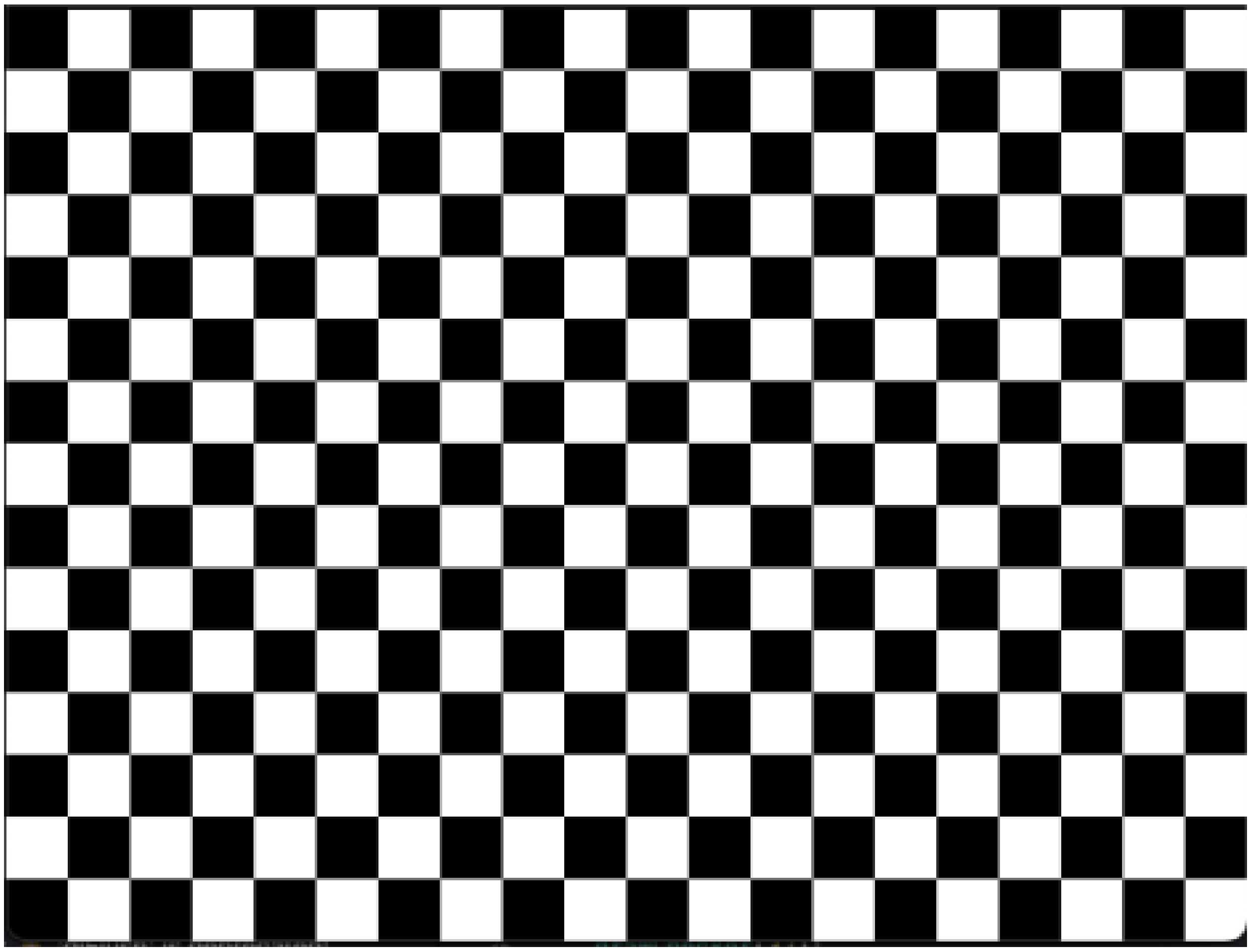
Study of basic output primitives in c++ using openGL.

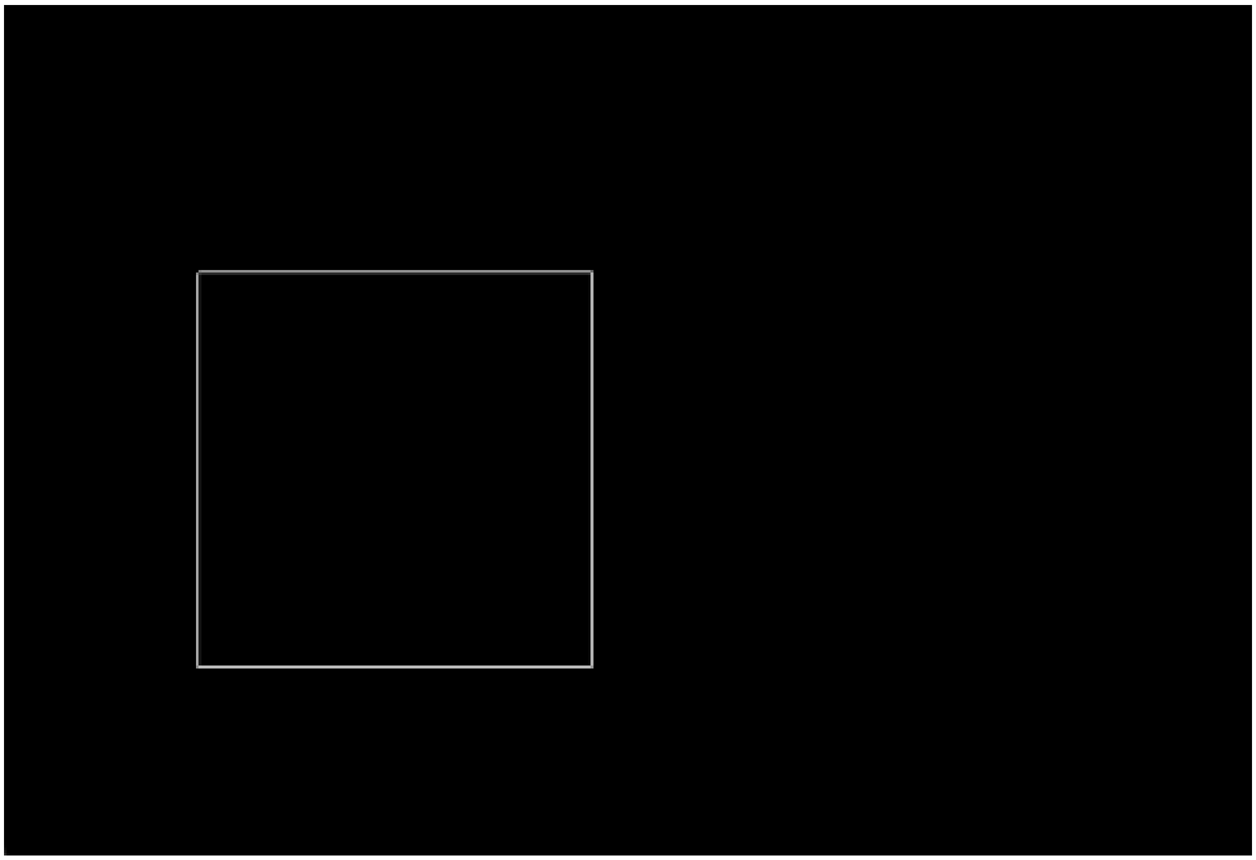
CODE:

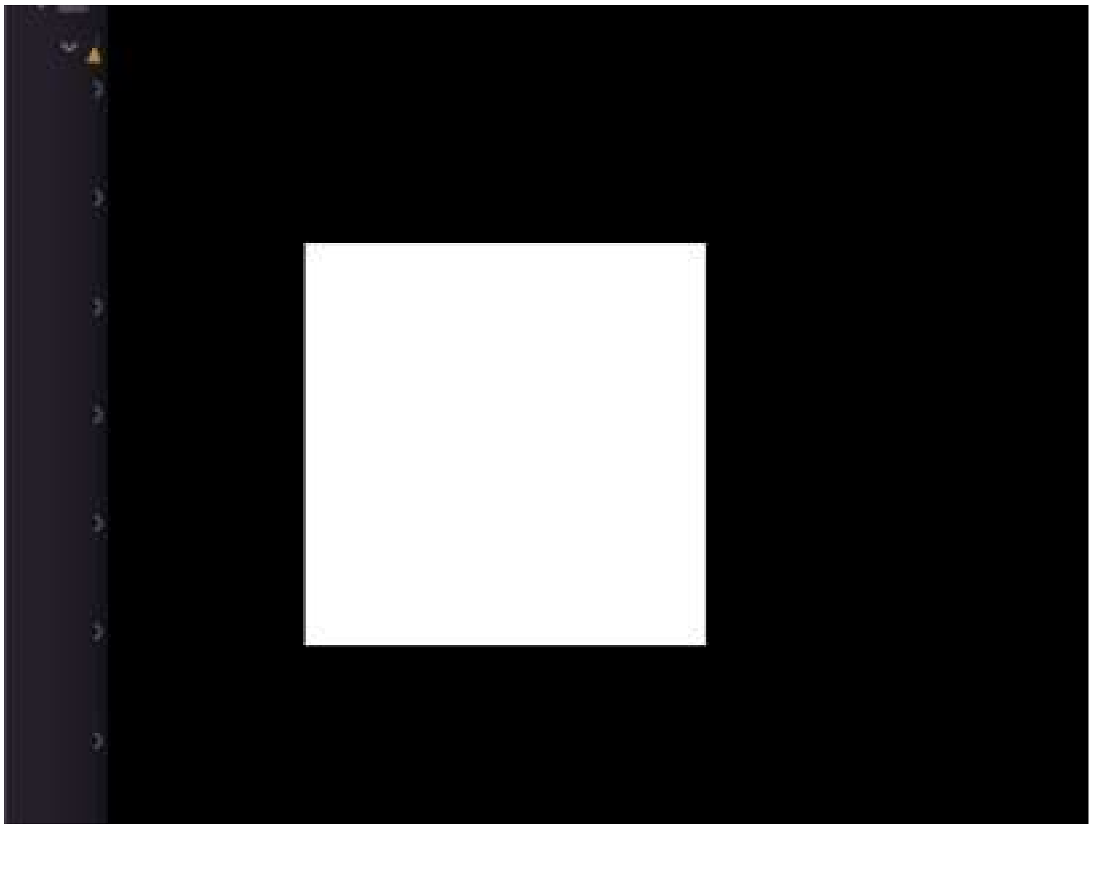
```
#include<GLUT/glut.h>
void myInit() {
glClearColor(0.0,0.0,0.0,0.0);
glColor3f(1.0f,1.0f,1.0f);
glPointSize(8);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0.0,640.0,0.0,480.0);
// glClearColor(1.0,1.0,1.0,0.0); // set white background color
// glMatrixMode(GL_PROJECTION);
// glLoadIdentity();
// gluOrtho2D(0.0, 640.0, 0.0, 480.0);
}
void drawChecker(int size)
{
int i=0;
int j=0;
for (i = 0; i < 100 ; ++i) {
for (j = 0; j < 100; ++j) {
if((i + j)%2 == 0) // if i + j is even
glColor3f( 0.0, 0.0, 0.0);
else
glColor3f( 1.0, 1.0, 1.0);
glRecti(i*size, j*size, (i+1)*size, (j+1)*size); // draw the rectangle
}
}
glFlush();
}
void checkerboard(void) {
glClear(GL_COLOR_BUFFER_BIT); // clear the screen
drawChecker(32);
}
void myDisplay() {
glClear(GL_COLOR_BUFFER_BIT);
//glBegin(GL_LINE_STRIP);
glBegin(GL_LINE_LOOP);
//glBegin(GL_QUADS);
//glBegin(GL_QUAD_STRIP);
//glBegin(GL_POLYGON);
//glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, 'A'); glRasterPos2f(300, 100);
glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, 'A'); glVertex2d(300,100);
glVertex2d(100,100);
glVertex2d(100,300);
glVertex2d(300,300);
glEnd();
glBegin(GL_LINE_LOOP);
glVertex2d(100,300);
glVertex2d(300,300);
glVertex2d(200, 420);
glEnd();
// glBegin(GL_QUAD_STRIP);
// glVertex2d(170,100);
// glVertex2d(230,100);
// glVertex2d(170,170);
// glVertex2d(230,170);
// glEnd();
glBegin(GL_LINES);
glVertex2d(200,420);
glVertex2d(360,420);
glEnd();
glBegin(GL_LINES);
glVertex2d(300,300);
glVertex2d(460,300);
glEnd();
glBegin(GL_LINES);
glVertex2d(360,420);
glVertex2d(460,300);
glEnd();
glBegin(GL_LINES);
glVertex2d(460,300);
glVertex2d(460,100);
glEnd();
glBegin(GL_LINES);
glVertex2d(460,100);
glVertex2d(300,100);
glEnd();
glBegin(GL_LINES);
glVertex2d(170,100);
glVertex2d(170,170);
glEnd();
glBegin(GL_LINES);
glVertex2d(230,100);
glVertex2d(230,170);
glEnd();
glBegin(GL_LINES);
glVertex2d(170,170);
glVertex2d(230,170);
glEnd();
// glRasterPos2f(100, 150);
// glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, 'A'); glFlush();
glBegin;
glEnd();
}
```

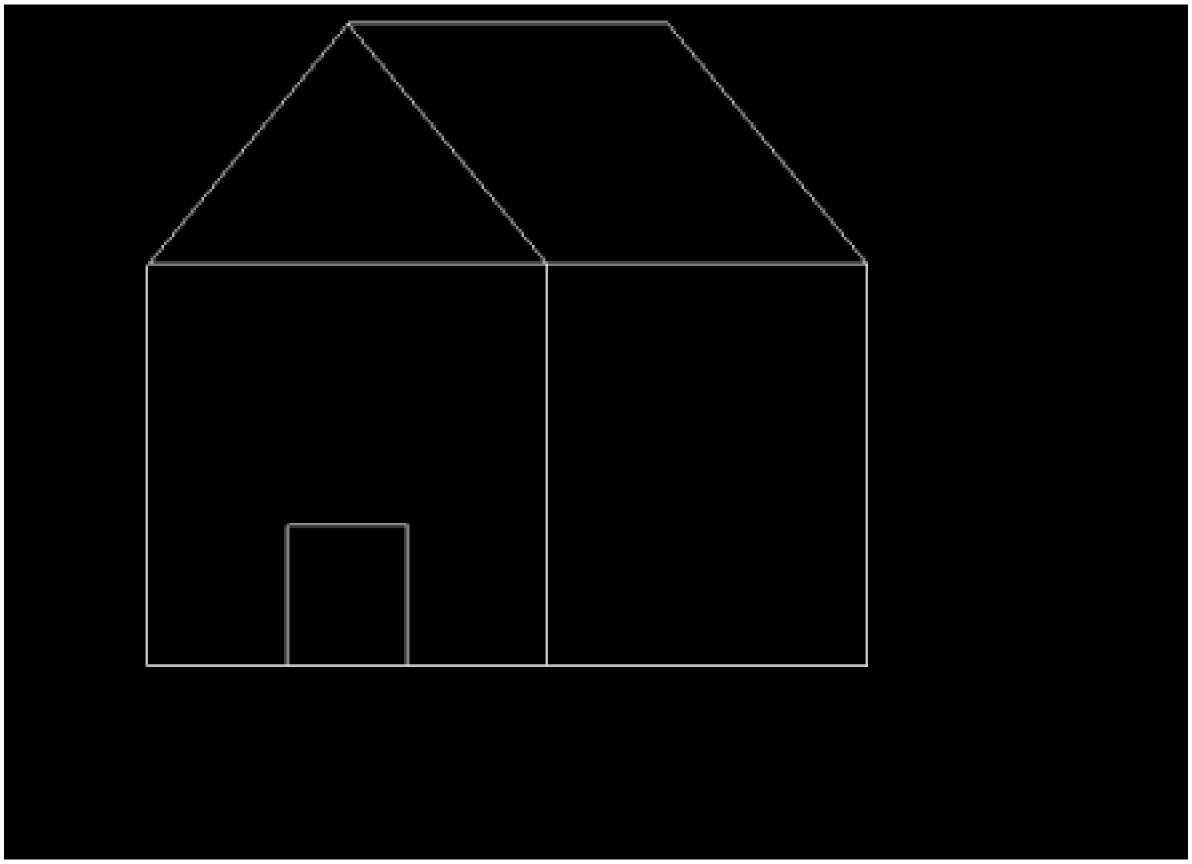
```
int main(int argc,char* argv[]) {
// glutInit(&argc,argv);
// glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
// glutInitWindowSize(640,480);
// glutCreateWindow("First Exercise");
// glutDisplayFunc(checkerboard);
// myInit();
// glutMainLoop();
// return 1;
glutInit(&argc, argv); // initialize the toolkit
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB); // set display mode
glutInitWindowSize(640,480); // set window size
//glutInitWindowPosition(100, 150); // set window position on screen glutCreateWindow("null"); //
open the screen window
glutDisplayFunc(myDisplay); // register redraw function myInit();
glutMainLoop();
                }
```

OUTPUT:









LEARNING OUTCOMES:

I learned how to use openGL to construct lines and other shapes in C++.

EX 2

205001085
SABARIVASAN V

AIM:

To plot points that make up the line with endpoints (x0,y0) and (xn,yn) using the DDA line drawing algorithm. Case 1: +ve slope Left to Right line Case 2: +ve slope Right to Left line
Case 3: -ve slope Left to Right line

Case 4: -ve slope Right to Left line Each case has two subdivisions

(i) |m|<= 1 (ii) |m|>1
Note that all four cases of line drawing must be given as test cases.

Note that all four cases of line drawing must be given as test cases.

ALGORITHM:

○ Inputlineendpoints,(x1,y1)and(x2,y2) setpixelatposition(x1,y1)
calculateslopem=(y2-y1)/(x2-x1)
For +ve slope (left to right)
○ Case |m|≤1: Sample at unit x intervals and compute each successive y. Repeat the following steps until (x2, y2) is reached:
○ **yk+1 = yk + m** where(m=y/ x)
xk+1 =xk +1
set pixel at position **(xk+1,Round(yk+1))**
○ **Case |m|>1**: Sample at unit y intervals and compute each

successive x.

- ○ Repeat the following steps until (x2, y2) is reached: **xk+1 = xk+ 1/m**
- ○ **yk+1 =yk +1**
  set pixel at position **(Round(xk+1), yk+1)**

For +ve slope(right end point to left end point)
Case |m|≤1: Sample at unit x intervals and compute each successive y.
Repeat the following steps until (x2, y2) is reached:

yk+1 = yk - m where(m=y/ x)
xk+1 =xk -1
set pixel at position (xk-1,Round(yk-1))

Case |m|>1: Sample at unit y intervals and compute each successive x.
Repeat the following steps until (x2, y2) is reached:

xk+1 = xk -1/m
yk+1 =yk -1
set pixel at position (Round(xk-1), yk-1)

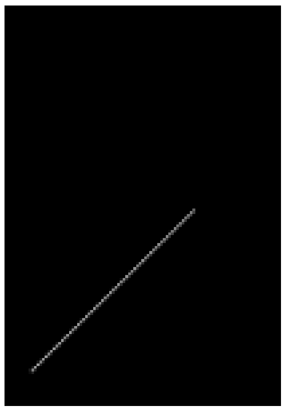
CODE :

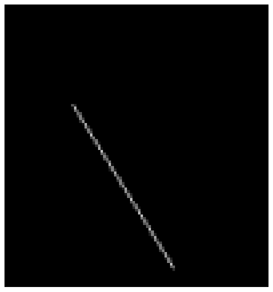
```
#include<GLUT/glut.h>
#include<iostream>
#include<cmath>
using namespace std;
void myInit() {
glClearColor(1.0,1.0,1.0,0.0);
glColor3f(0.0f,0.0f,0.0f);
glPointSize(10);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(-150.0,640.0,-150.0,480.0);
}
void myDisplay() {
glClear(GL_COLOR_BUFFER_BIT);
float x0,y0,xn,yn,x,y,m;
cin>>x0>>y0>>xn>>yn;
m = (yn-y0)/(xn-x0);
x=x0;
y=y0;
string p=to_string((int)x0);
int i=0,j=0;
while(i<p.length()){
glRasterPos2f(x0+j, y0-20);
glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_10, p[i]); i++;
j+=5;
}
glRasterPos2f(x0+j, y0-20);
glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_10, ',');
p=to_string((int)y0);
i=0;
while(i<p.length()){
glRasterPos2f(x0+j, y0-20);
```

```
glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_10, p[i]); i++;
j+=5;
}
p=to_string((int)xn);
i=0,j=0;
while(i<p.length()){
glRasterPos2f(xn+j, yn-20);
glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_10, p[i]); i++;
j+=5;
}
glRasterPos2f(xn+j, yn-20);
glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_10, ',');
p=to_string((int)yn);
i=0;
while(i<p.length()){
glRasterPos2f(xn+j, yn-20);
glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_10, p[i]); i++;
j+=5;
}
glBegin(GL_POINTS);
if(x0<xn){
if(m>0){
if(abs(m)<1){
while(x!=xn){
glVertex2d((int)round(x), (int)round(y));
x+=1;
y+=abs(m);
}
}
else{
while(y!=yn){
glVertex2d((int)round(x), (int)round(y));
x+=1/abs(m);
y+=1;
}
}
}
else{
if(abs(m)<1){
while(x!=xn){
glVertex2d((int)round(x), (int)round(y)); x+=1;
y-=abs(m);
}
}
else{
while(y!=yn){
glVertex2d((int)round(x), (int)round(y)); x+=1/abs(m);
y-=1;
}
}
}
}
else{
if(m>0){
if(abs(m)<1){
while(x!=xn){
glVertex2d((int)round(x), (int)round(y)); x-=1;
y-=abs(m);
}
}
else{
while(y!=yn){
glVertex2d((int)round(x), (int)round(y)); x-=1/abs(m);
y-=1;
}
}
}
else{
```

```
if(abs(m)<1){
while(x!=xn){
glVertex2d((int)round(x), (int)round(y)); x-=1;
y+=abs(m);
}
}
else{
while(y!=yn){
glVertex2d((int)round(x), (int)round(y)); x-=1/abs(m);
y+=1;
}
}
}
}
glEnd();
glFlush();
}
int main(int argc,char* argv[]) {
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutInitWindowSize(640,480);
glutInitWindowPosition((glutGet(GLUT_SCREEN_WIDTH)-640)/2,(glutGet(
GLUT_SCREEN_HEIGHT)-480)/2); glutCreateWindow("Second Exercise");
glutDisplayFunc(myDisplay);
myInit();
glutMainLoop();
return 1;
}
```

SAMPLE I/O:





LEARNING OUTCOME:
I learnt how to use the DDA line drawing algorithm in c++ using the openGL library to draw a line.

EX 3

205001085
SABARIVASAN V

AIM:

To plot points that make up the line with endpoints (x0,y0) and (xn,yn) using Bresenham's line drawing algorithm.

Case 1: +ve slope Left to Right line

Case 2: +ve slope Right to Left line

Case 3: -ve slope Left to Right line

Case 4: -ve slope Right to Left line Each case has two subdivisions

(i) $|m| <= 1$ (ii) $|m| > 1$
Note that all four cases of line drawing must be given as test cases.

ALGORITHM:

Input Line Endpoints,(x0,y0)and(xn,yn)
Load (x0,y0) into the frame buffer that is first point Calculate The Constants X,y,2y and 2y-2x calculate parameter p0 = 2 y - x Set pixel At Position(x0,y0) repeat the following steps until(xn,yn)is reached:

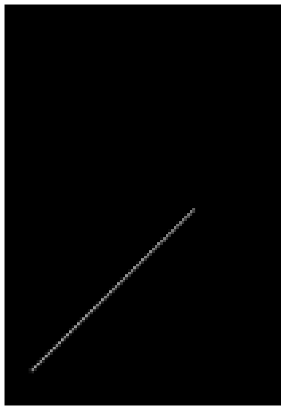
if pk < 0
set the next pixel at position (xk +1, yk) calculate new pk+1 = pk + 2 y

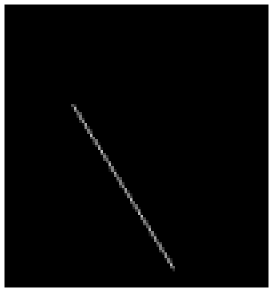
if pk ≥ 0
set the next pixel at position (xk +1, yk + 1) calculate new pk+1 = pk + 2(y - x)
Repeat last step x times.

CODE :

```
#include <GLUT/glut.h>
#include <stdio.h>
#include<math.h>
int xstart, ystart, xend, yend;
void myInit() {
glClear(GL_COLOR_BUFFER_BIT);
glClearColor(0.0, 0.0, 0.0, 1.0);
glMatrixMode(GL_PROJECTION);
gluOrtho2D(0, 500, 0, 500);
}
void draw_pixel(int x, int y) {
glBegin(GL_POINTS);
glVertex2i(x, y);
glEnd();
}
void draw_line(int xstart, int xend, int ystart, int yend) { int dx, dy, i, e;
int incx, incy, inc1, inc2;
int x,y;
dx = abs(xend-xstart);
dy = abs(yend-ystart);
incx = 1;
if (xend < xstart) incx = -1;
incy = 1;
if (yend < ystart) incy = -1;
x = xstart;
y = ystart;
if (dx > dy) {
draw_pixel(x, y);
e = 2 * dy-dx;
inc1 = 2*(dy-dx);
inc2 = 2*dy;
for (i=0; i<dx; i++) {
if (e >= 0) {
y += incy;
e += inc1;
}
else
e += inc2;
x += incx;
draw_pixel(x, y);
}
} else {
draw_pixel(x, y);
e = 2*dx-dy;
inc1 = 2*(dx-dy);
inc2 = 2*dx;
for (i=0; i<dy; i++) {
if (e >= 0) {
x += incx;
e += inc1;
}
else
e += inc2;
y += incy;
draw_pixel(x, y);
}
}
}
void myDisplay() {
draw_line(xstart, xend, ystart, yend);
glFlush();
}
int main(int argc, char **argv) {
printf( "Enter (xstart, ystart, xend, yend)\n"); scanf("%d %d %d %d", &xstart,
&ystart, &xend, &yend); glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB); glutInitWindowSize(500,
500);
glutInitWindowPosition(0, 0);
glutCreateWindow("Excercise 3");
myInit();
glutDisplayFunc(myDisplay);
glutMainLoop();
}
```

SAMPLE I/O:





LEARNING OUTCOME:

I learnt how to use bresenham’s line drawing algorithm in c++ using the openGL library to draw a line.

EX 4

205001085
SABARIVASAN V

AIM:

a) To plot points that make up the circle with center (xc,yc) and radius r using the Midpoint circle drawing algorithm. Give atleast 2 test cases.

Case 1: With center (0,0)
Case 2: With center (xc,yc)
b) To draw any object using line and circle drawing algorithms.

ALGORITHM:

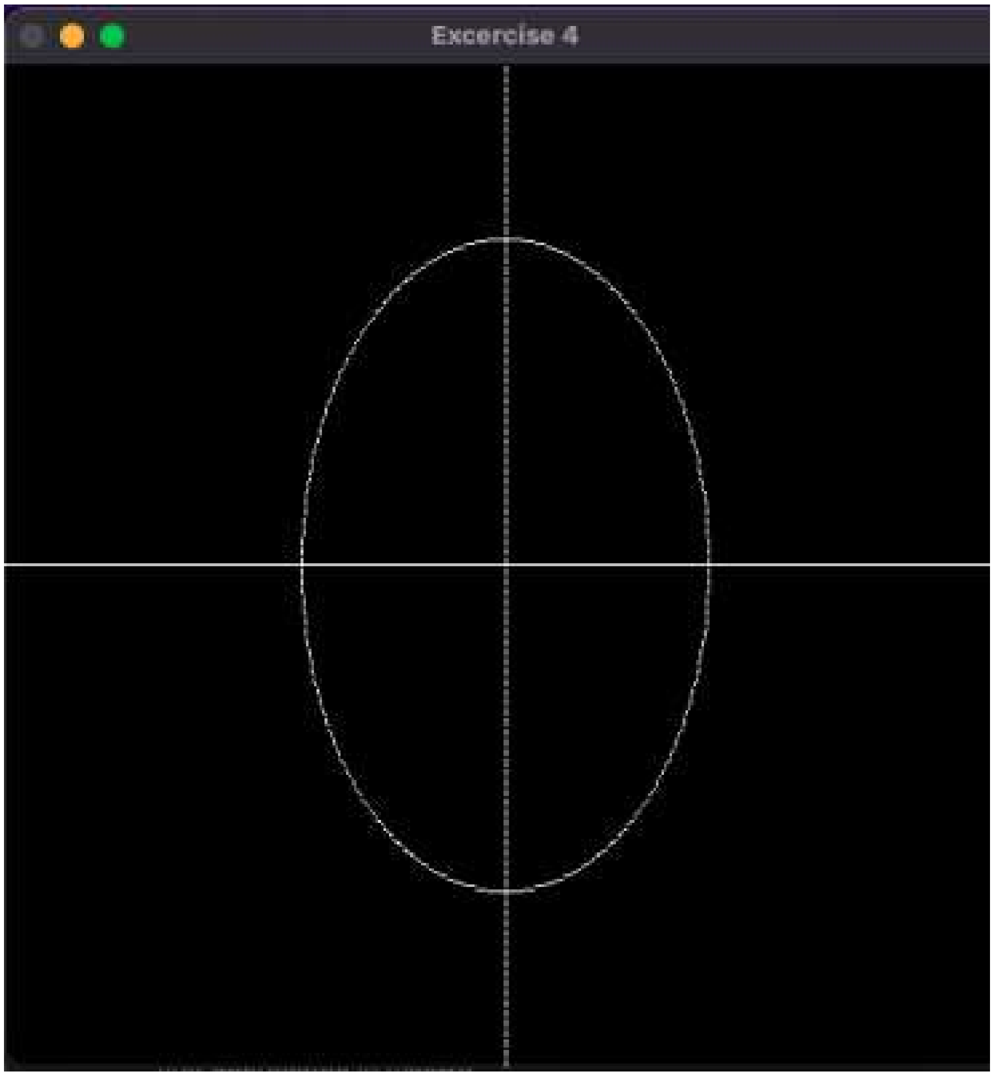
1. Input radius r and circle center (xc, yc). set the first point (x0 , y0) = (0, r).
2. Calculate the initial value of the decision parameter as p0 = 1 – r. 3. At each xk position, starting at k = 0, perform the following test: Ifpk <0,
plot(xk +1,yk)andpk+1 =pk +2xk+1 +1,
Else,
where2xk+1 =2xk +2and2yk+1 =2yk –2.
plot (xk+1,yk –1)andpk+1 =pk +2xk+1 +1–2yk+1,
4. Determine symmetry points on the other seven octants. 5. Move each calculated pixel position (x, y) onto the circular path centered on (xc, yc) and plot the coordinate values: x = x +xc ,y=y+yc
6. Repeat steps 3 through 5 until x y.
7. For all points, add the center point (xc, yc)

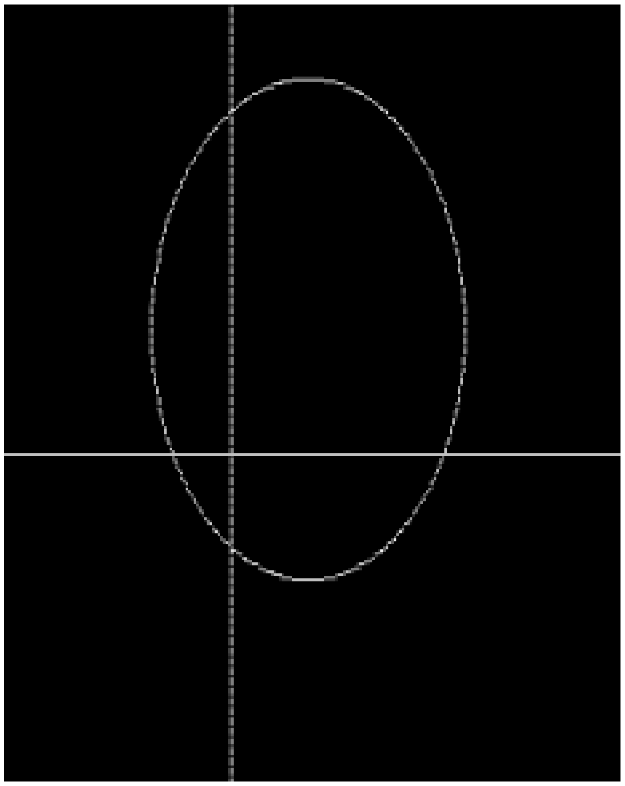
CODE:

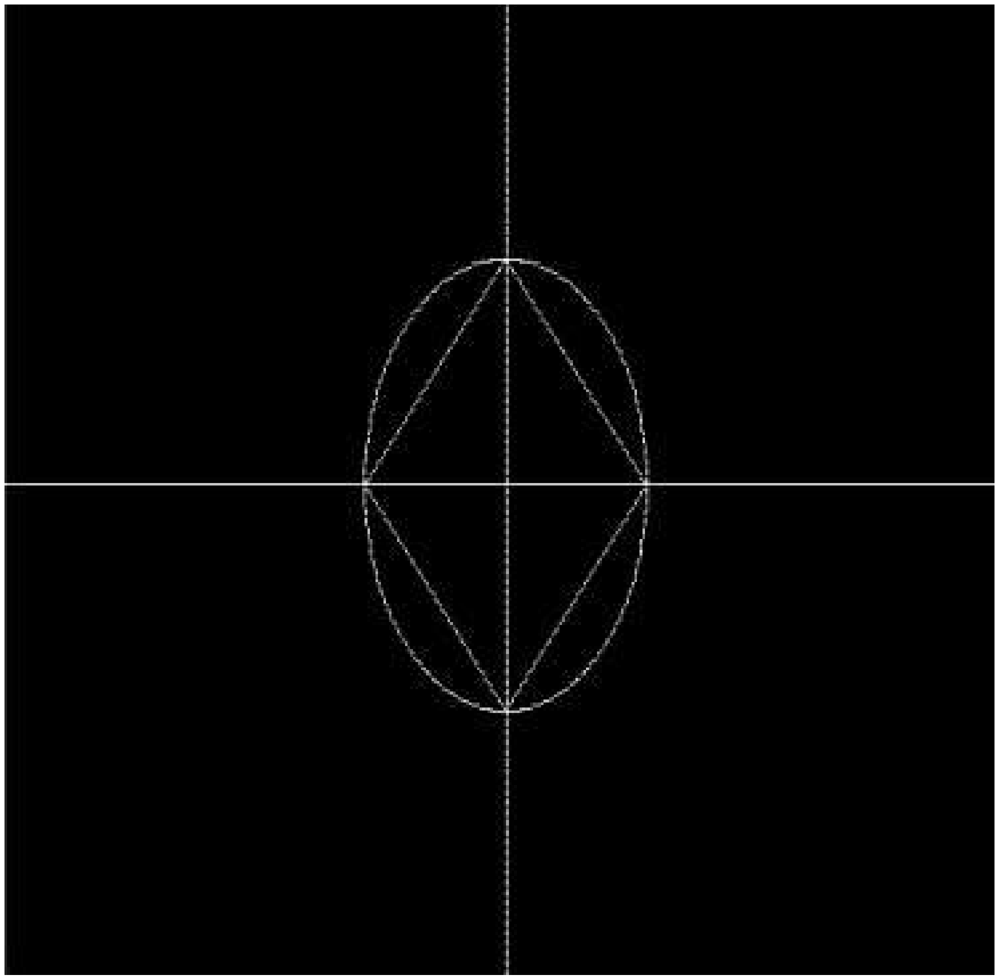
```
#include <GLUT/glut.h>
#include <stdio.h>
#include<math.h>
int xstart, ystart, xend, yend, xc, yc, radius;
void myInit() {
glClear(GL_COLOR_BUFFER_BIT);
glClearColor(0.0, 0.0, 0.0, 1.0);
glMatrixMode(GL_PROJECTION);
gluOrtho2D(-320, 320, -200, 200);
}
void draw_pixel(int x, int y) {
glBegin(GL_POINTS);
glVertex2i(x, y);
glEnd();
}
void draw_line(int xstart, int xend, int ystart, int yend) {
int dx, dy, i, e;
int incx, incy, inc1, inc2;
int x,y;
dx = abs(xend-xstart);
dy = abs(yend-ystart);
incx = 1;
if (xend < xstart) incx = -1;
incy = 1;
if (yend < ystart) incy = -1;
x = xstart;
y = ystart;
if (dx > dy) {
draw_pixel(x, y);
e = 2 * dy-dx;
inc1 = 2*(dy-dx);
inc2 = 2*dy;
for (i=0; i<dx; i++) {
if (e >= 0) {
y += incy;
e += inc1;
}
else
e += inc2;
x += incx;
draw_pixel(x, y);
}
} else {
draw_pixel(x, y);
e = 2*dx-dy;
inc1 = 2*(dx-dy);
inc2 = 2*dx;
for (i=0; i<dy; i++) {
if (e >= 0) {
x += incx;
e += inc1;
}
else
e += inc2;
y += incy;
draw_pixel(x, y);
}
}
}
void comb(int x, int y, int xc, int yc){ // x += xc;
// y += yc;
draw_pixel(x+xc, y+yc);
draw_pixel(y+xc, x+yc);
draw_pixel(-x+xc, y+yc);
draw_pixel(-y+xc, x+yc);
draw_pixel(-x+xc, -y+yc);
draw_pixel(-y+xc, -x+yc);
draw_pixel(x+xc, -y+yc);
draw_pixel(y+xc, -x+yc);
}
void draw_circle(int xc, int yc, int radius){ int p0 = 1 - radius;
int xstart = 0;
int ystart = radius;
comb(xstart, ystart, xc, yc);
while(xstart < ystart){
xstart += 1;
int pnew = p0 + 2*xstart + 1;
if(p0 > 0){
ystart -= 1;
pnew -= 2*ystart;
}
comb(xstart, ystart, xc, yc);
p0 = pnew;
}
}
void custom_shape(){
}
void myDisplay() {
//draw_line(xstart, xend, ystart, yend); draw_line(-1000, 1000,0, 0);
draw_line(0, 0,-1000, 1000);
draw_circle(xc, yc, radius);
draw_line(radius, 0, 0, radius);
draw_line(-radius, 0, 0, radius);
draw_line(radius, 0, 0, -radius);
draw_line(-radius, 0, 0, -radius);
glFlush();
}
int main(int argc, char **argv) {
// printf( "Enter (xstart, ystart, xend, yend)\n"); // scanf("%d %d %d %d", &xstart, &ystart, &xend, &yend);
printf("Enter xc, yc and radius\n");
```

```
scanf("%d %d %d", &xc, &yc, &radius);
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB); glutInitWindowSize(500, 500);
glutInitWindowPosition(0, 0);
glutCreateWindow("Excercise 4");
myInit();
glutDisplayFunc(myDisplay);
glutMainLoop();
               }
```

SAMPLE I/O:







LEARNING OUTCOME:

I learned how to use the midpoint circle drawing algorithm in c++ using the openGL library to draw circles.

205001085
SABARIVASAN V

Lab Exercise 5: 2D Transformations in C++ using OpenGL

Aim:

To apply the following 2D transformations on objects and to render the final output along with the original object.

1) Translation 2) Rotation

a) about origin

b) with respect to a fixed point (xr,yr) 3) Scaling with respect to

a) origin - Uniform Vs Differential Scaling

b) fixed point (xf,yf) 4) Reflection with respect to

a) x-axis
b) y-axis
c) origin
d) the line x=y

5) Shearing
a) x-direction shear

b) y-direction shear

Algorithm:

Application of a sequence of transformations to a point:

P' = M2.M1.P

= M.P

Composite transformations are formed by calculating the matrix product of the individual transformations and forming products of the transformation matrix.

Code:

```
#include <stdio.h>

#include <iostream>

#include<GLUT/glut.h>

#include <cmath>

using namespace std;

void plot(int x1,int x2,int y1, int y2)

{

glBegin(GL_LINES);

glVertex2i(x1,y1);

glVertex2i(x2, y2);

glEnd();

}

void myInit (void)

{

glClearColor(1.0, 1.0, 1.0, 0.0);

glColor3f(0.0f, 0.0f, 0.0f);

glPointSize(4.0);

glMatrixMode(GL_PROJECTION);

glLoadIdentity();

gluOrtho2D(-320.0,320.0,-240.0,240.0); }

void matmul(float M[3][3],int P[3][2],float res[3][2]){ res[0][0]=0;res[1][0]=0;res[2][0]=0;

res[0][1]=0;res[1][1]=0;res[2][1]=0;

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

```
for(int j=0;j<2;j++){

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

res[i][j]+=M[i][k]*P[k][j];

}

}

}

}

void matmul2(float A[][3],float B[][3],float M[][3]){ for(int i=0;i<3;i++){

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

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

M[i][j]+=A[i][k]*B[k][j];

}

}

}

}

void matmul3(float M[][3],int P[3][4],float res[3][4]){ res[0][0]=0;res[1][0]=0;res[2][0]=0;

res[0][1]=0;res[1][1]=0;res[2][1]=0;

res[0][2]=0;res[1][2]=0;res[2][2]=0;

res[0][3]=0;res[1][3]=0;res[2][3]=0;

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

for(int j=0;j<4;j++){

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

res[i][j]+=M[i][k]*P[k][j];

}

}
```

```
}

}

void translate(int x1,int y1,int x2,int y2,int tx,int ty){

plot(x1,x2,y1,y2);

int hom[3][2];

hom[0][0]=x1;hom[1][0]=y1;hom[2][0]=1;

hom[0][1]=x2;hom[1][1]=y2;hom[2][1]=1;

float T[3][3];

T[0][0]=1;

T[0][1]=0;

T[0][2]=tx;

T[1][0]=0;

T[1][1]=1;

T[1][2]=ty;

T[2][0]=0;

T[2][1]=0;

T[2][2]=1;

float res[3][2];

matmul(T,hom,res);

plot((int)round(res[0][0]),(int)round(res[0][1]),(int)round(res[1][0]),(int)round(res[1][1])); }

void rotation(int x1,int y1,int x2,int y2,int tx,int ty){

plot(x1,x2,y1,y2);

int hom[3][2];

hom[0][0]=x1;hom[1][0]=y1;hom[2][0]=1;

hom[0][1]=x2;hom[1][1]=y2;hom[2][1]=1;
```

```
float T[3][3]={0.0};

T[0][0]=1;

T[0][1]=0;

T[0][2]=tx;

T[1][0]=0;

T[1][1]=1;

T[1][2]=ty;

T[2][0]=0;

T[2][1]=0;

T[2][2]=1;

float rad;

cout<<"Enter angle in radians:";cin>>rad;

float T2[3][3];

T2[0][0]=cos(rad);

T2[0][1]=-sin(rad);

T2[0][2]=0;

T2[1][0]=sin(rad);

T2[1][1]=cos(rad);

T2[1][2]=0;

T2[2][0]=0;

T2[2][1]=0;

T2[2][2]=1;

float res[3][2]={0.0},M[3][3]={0.0},M2[3][3]={0.0};

matmul2(T,T2,M);

T[0][0]=1;
```

```
T[0][1]=0;

T[0][2]=-tx;

T[1][0]=0;

T[1][1]=1;

T[1][2]=-ty;

T[2][0]=0;

T[2][1]=0;

T[2][2]=1;

matmul2(M, T,M2);

matmul(M2, hom, res);

plot((int)round(res[0][0]),(int)round(res[0][1]),(int)round(res[1][0]),(int)round(res[1][1])); }

void scaling(int x1,int y1,int x2,int y2,int tx,int ty){

plot(x1,x2,y1,y2);

int hom[3][2];

hom[0][0]=x1;hom[1][0]=y1;hom[2][0]=1;

hom[0][1]=x2;hom[1][1]=y2;hom[2][1]=1;

float T[3][3]={0.0};

T[0][0]=1;

T[0][1]=0;

T[0][2]=tx;

T[1][0]=0;

T[1][1]=1;

T[1][2]=ty;

T[2][0]=0;

T[2][1]=0;
```

```
T[2][2]=1;

float sx,sy;

cout<<"Enter sx,sy:";cin>>sx>>sy;

float T2[3][3];

T2[0][0]=sx;

T2[0][1]=0;

T2[0][2]=0;

T2[1][0]=0;

T2[1][1]=sy;

T2[1][2]=0;

T2[2][0]=0;

T2[2][1]=0;

T2[2][2]=1;

float res[3][2]={0.0},M[3][3]={0.0},M2[3][3]={0.0};

matmul2(T,T2,M);

T[0][0]=1;

T[0][1]=0;

T[0][2]=-tx;

T[1][0]=0;

T[1][1]=1;

T[1][2]=-ty;

T[2][0]=0;

T[2][1]=0;

T[2][2]=1;

matmul2(M, T,M2);
```

```
matmul(M2, hom, res);

plot((int)round(res[0][0]),(int)round(res[0][1]),(int)round(res[1][0]),(int)round(res[1][1])); }

void reflection(int x1,int y1,int x2,int y2){

plot(x1,x2,y1,y2);

int hom[3][2];

hom[0][0]=x1;hom[1][0]=y1;hom[2][0]=1;

hom[0][1]=x2;hom[1][1]=y2;hom[2][1]=1;

float T[3][3]={0.0};

T[0][0]=1;

T[0][1]=0;

T[0][2]=0;

T[1][0]=0;

T[1][1]=-1;

T[1][2]=0;

T[2][0]=0;

T[2][1]=0;

T[2][2]=1;

float res[3][2]={0.0};

matmul(T, hom, res);

plot((int)round(res[0][0]),(int)round(res[0][1]),(int)round(res[1][0]),(int)round(res[1][1]));

T[0][0]=-1;

T[0][1]=0;

T[0][2]=0;

T[1][0]=0;

T[1][1]=1;
```

```
T[1][2]=0;

T[2][0]=0;

T[2][1]=0;

T[2][2]=1;

matmul(T, hom, res);

plot((int)round(res[0][0]),(int)round(res[0][1]),(int)round(res[1][0]),(int)round(res[1][1]));

T[0][0]=-1;

T[0][1]=0;

T[0][2]=0;

T[1][0]=0;

T[1][1]=-1;

T[1][2]=0;

T[2][0]=0;

T[2][1]=0;

T[2][2]=1;

matmul(T, hom, res);

plot((int)round(res[0][0]),(int)round(res[0][1]),(int)round(res[1][0]),(int)round(res[1][1]));

T[0][0]=0;

T[0][1]=1;

T[0][2]=0;

T[1][0]=1;

T[1][1]=0;

T[1][2]=0;

T[2][0]=0;

T[2][1]=0;
```

```
T[2][2]=1;

matmul(T, hom, res);

plot((int)round(res[0][0]),(int)round(res[0][1]),(int)round(res[1][0]),(int)round(res[1][1])); }

void shearing(int x1,int y1,int x2,int y2,int x3,int y3,int x4,int y4){ glBegin(GL_QUADS);

glVertex2d(x1, y1);

glVertex2d(x2, y2);

glVertex2d(x3, y3);

glVertex2d(x4, y4);

glEnd();

int hom[3][4];

hom[0][0]=x1;hom[1][0]=y1;hom[2][0]=1;

hom[0][1]=x2;hom[1][1]=y2;hom[2][1]=1;

hom[0][2]=x3;hom[1][2]=y3;hom[2][2]=1;

hom[0][3]=x4;hom[1][3]=y4;hom[2][3]=1;

float shx,shy;

cout<<"Enter shx,shy:";

cin>>shx>>shy;

float T[3][3]={0.0};

T[0][0]=1;

T[0][1]=shx;

T[0][2]=0;

T[1][0]=0;

T[1][1]=1;

T[1][2]=0;

T[2][0]=0;
```

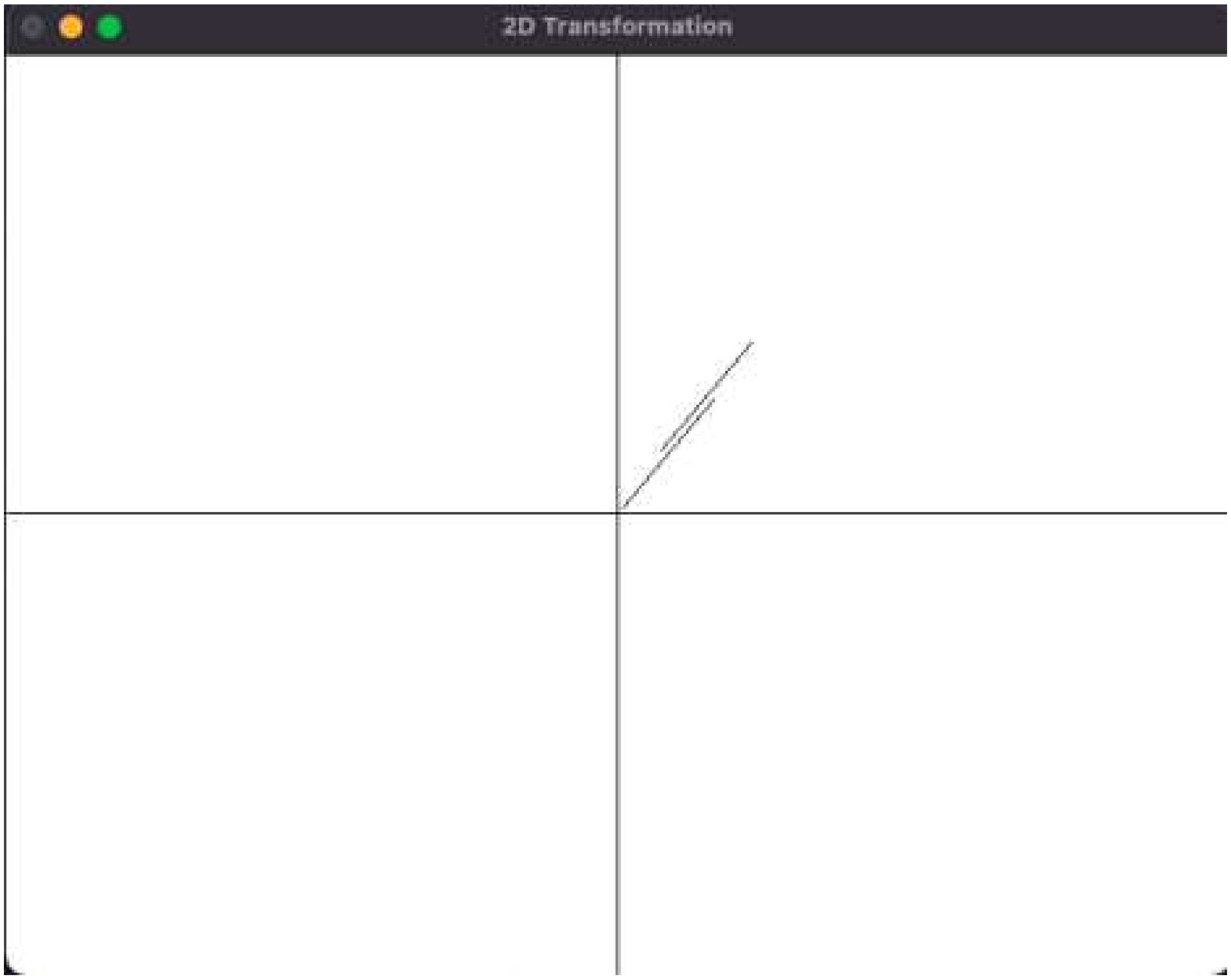
```
T[2][1]=0;

T[2][2]=1;

float res[3][4]={0.0};

matmul3(T, hom, res);

glBegin(GL_QUADS);

glVertex2d((int)round(res[0][0]), (int)round(res[1][0]));

glVertex2d((int)round(res[0][1]), (int)round(res[1][1]));

glVertex2d((int)round(res[0][2]), (int)round(res[1][2]));

glVertex2d((int)round(res[0][3]), (int)round(res[1][3]));

glEnd();

T[0][0]=1;

T[0][1]=0;

T[0][2]=0;

T[1][0]=shy;

T[1][1]=1;

T[1][2]=0;

T[2][0]=0;

T[2][1]=0;

T[2][2]=1;

matmul3(T, hom, res);

glBegin(GL_QUADS);

glVertex2d((int)round(res[0][0]), (int)round(res[1][0])); glVertex2d((int)round(res[0][1]),
(int)round(res[1][1])); glVertex2d((int)round(res[0][2]), (int)round(res[1][2]));
glVertex2d((int)round(res[0][3]), (int)round(res[1][3])); glEnd();

}

void myDisplay(void)
```

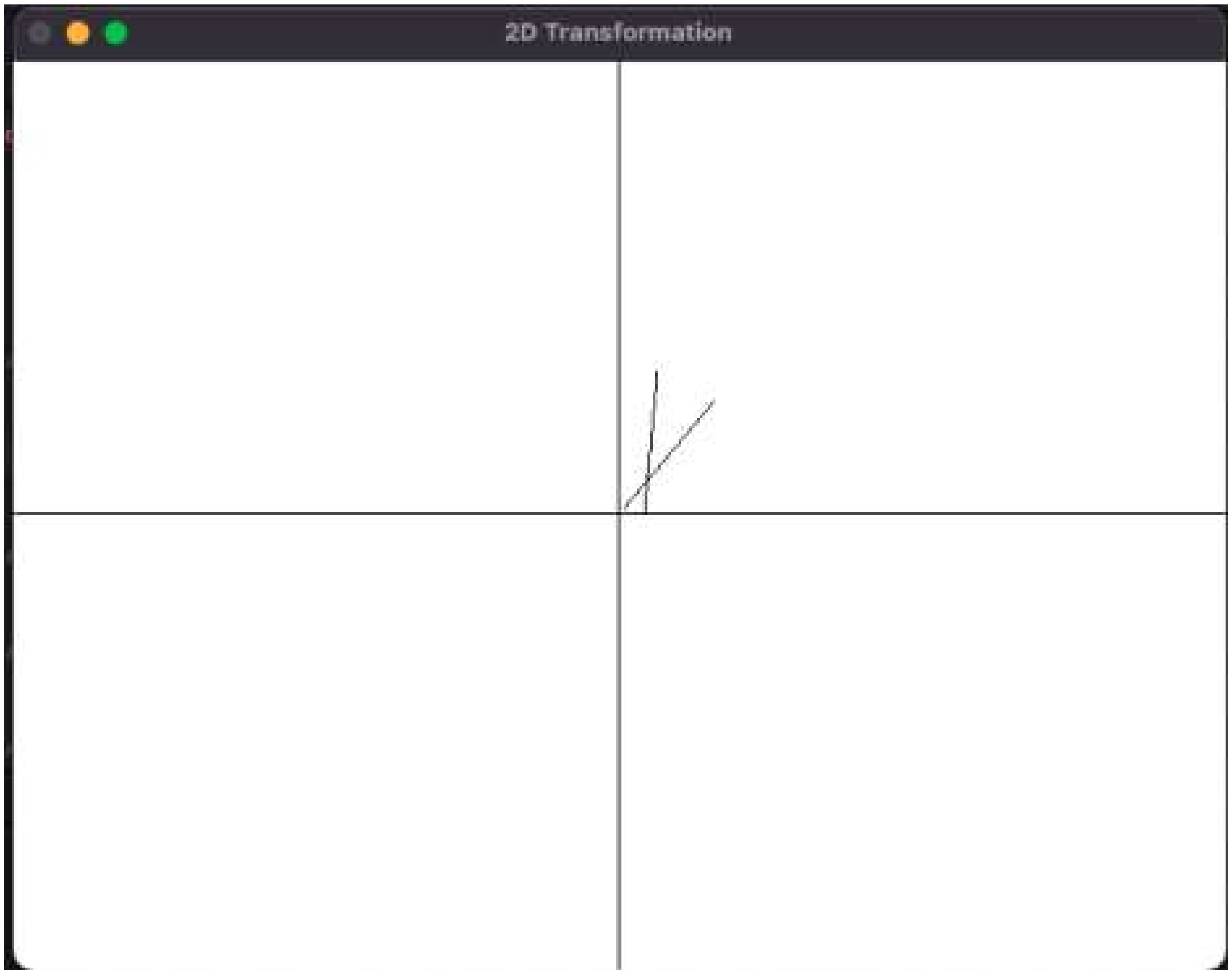
```
{

glClear (GL_COLOR_BUFFER_BIT);

glColor3f (0.0, 0.0, 0.0);

glPointSize(1.0);

glBegin(GL_LINES);

glVertex2i(-320, 0);

glVertex2i(320, 0);

glEnd();

glBegin(GL_LINES);

glVertex2i(0, -240);

glVertex2i(0,240 );

glEnd();

int x1,x2,y1,y2;

cout << "X-coordinate 1 : "; cin >> x1;

cout << "\nY-coordinate 1 : "; cin >> y1;

cout << "X-coordinate 2 : "; cin >> x2;

cout << "\nY-coordinate 2 : "; cin >> y2;

int tx,ty;

// cout<<"Enter tx,ty:";cin>>tx;cin>>ty;

// translate(x1,y1,x2,y2,tx,ty);

//rotation

// rotation(x1,y1,x2,y2,tx,ty);

//scaling

// scaling(x1,y1,x2,y2,tx,ty);

//reflection
```

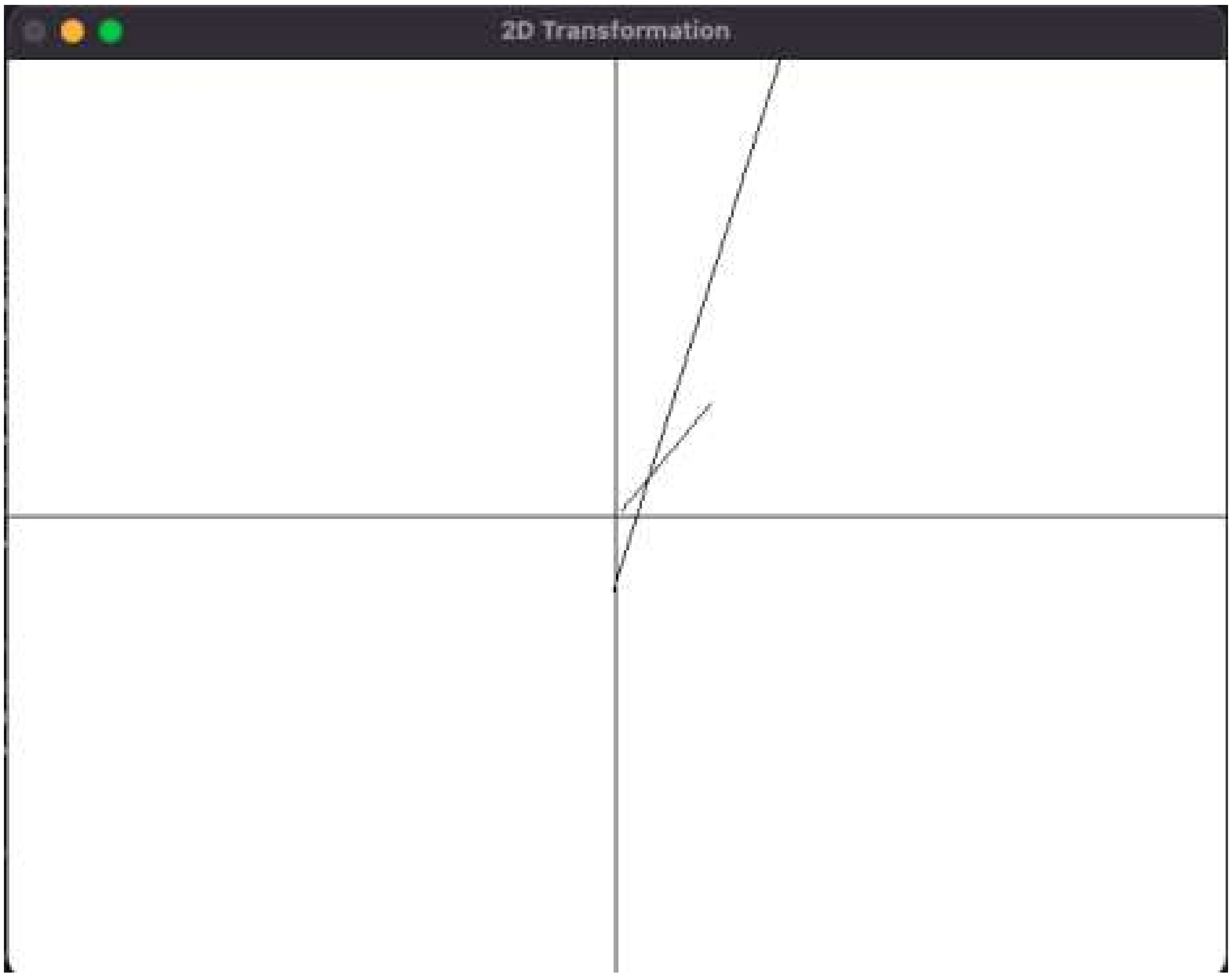
```
// reflection(x1,y1,x2,y2);

//shearing

int x3,y3,x4,y4;

cout<<"X-coordinate 3";

cin>>x3;

cout<<"Y-coordinate 3";

cin>>y3;

cout<<"X-coordinate 4";

cin>>x4;

cout<<"Y-coordinate 4";

cin>>y4;

shearing(x1,y1,x2,y2,x3,y3,x4,y4);

glFlush ();

}

int main(int argc, char** argv)

{

glutInit(&argc, argv);

glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);

glutInitWindowSize (640, 480);

// glutInitWindowPosition (100, 150);

glutCreateWindow ("2D Transformation");

glutDisplayFunc(myDisplay);

myInit ();

glutMainLoop();

return 0;
```

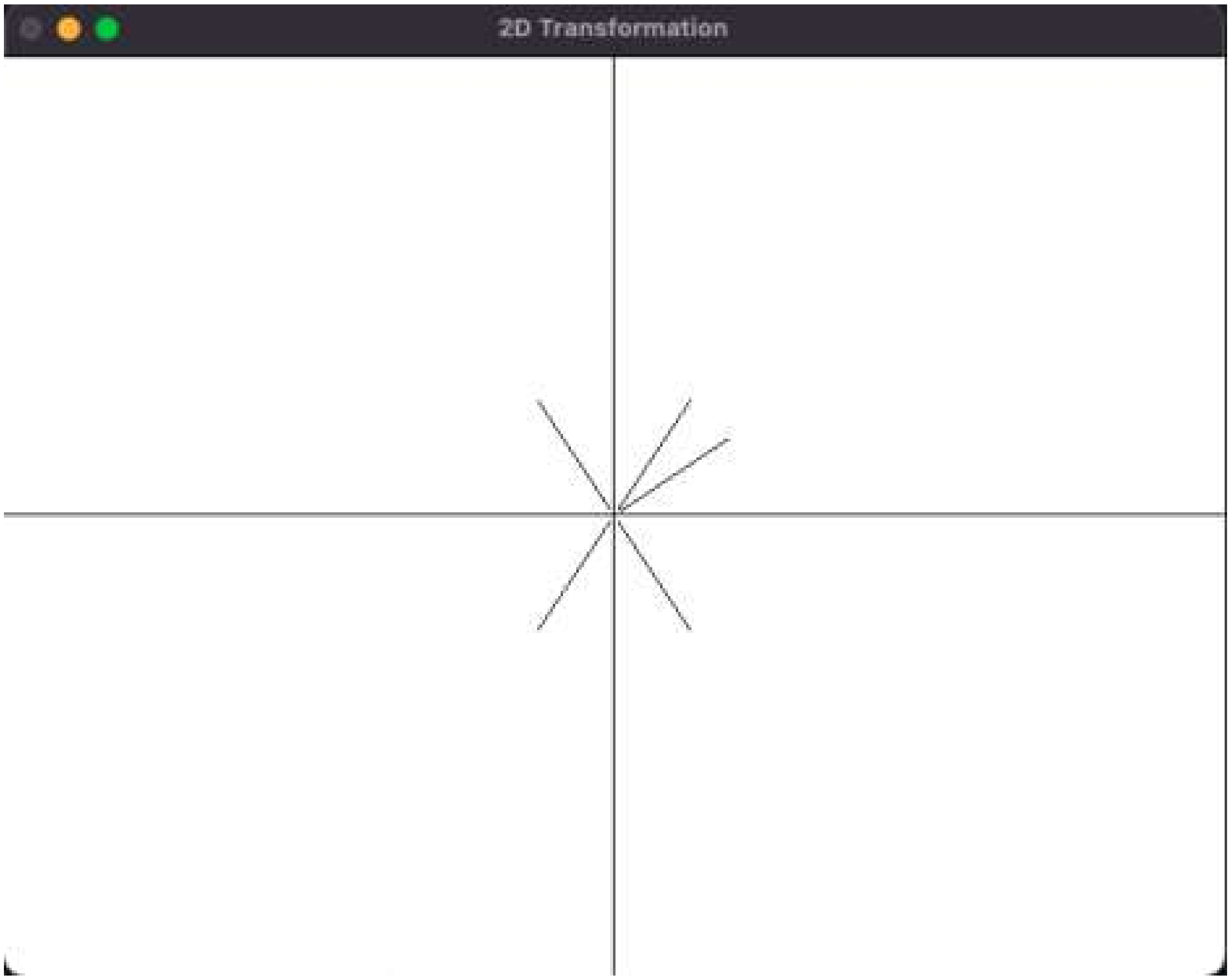
```
}
```

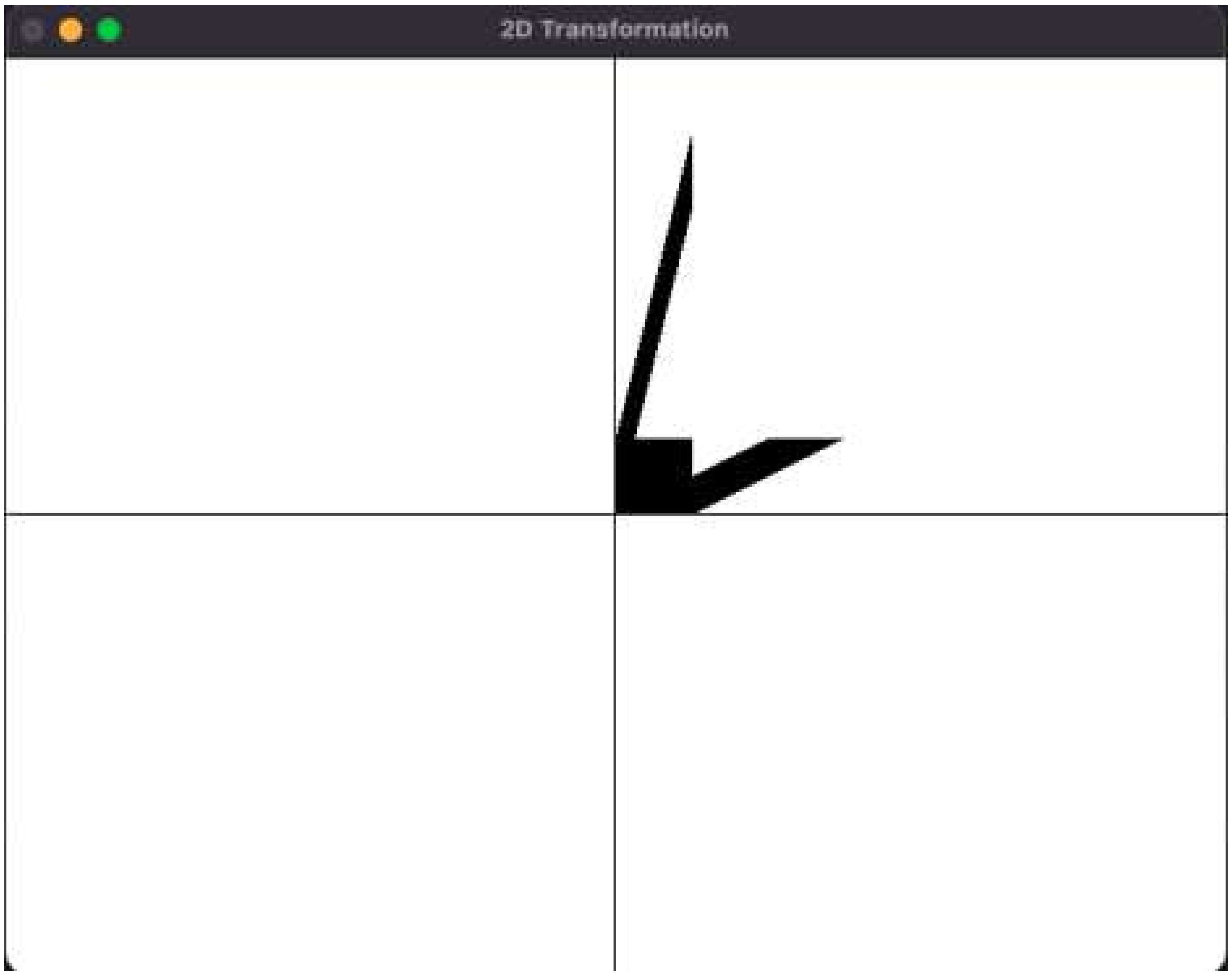
Output:

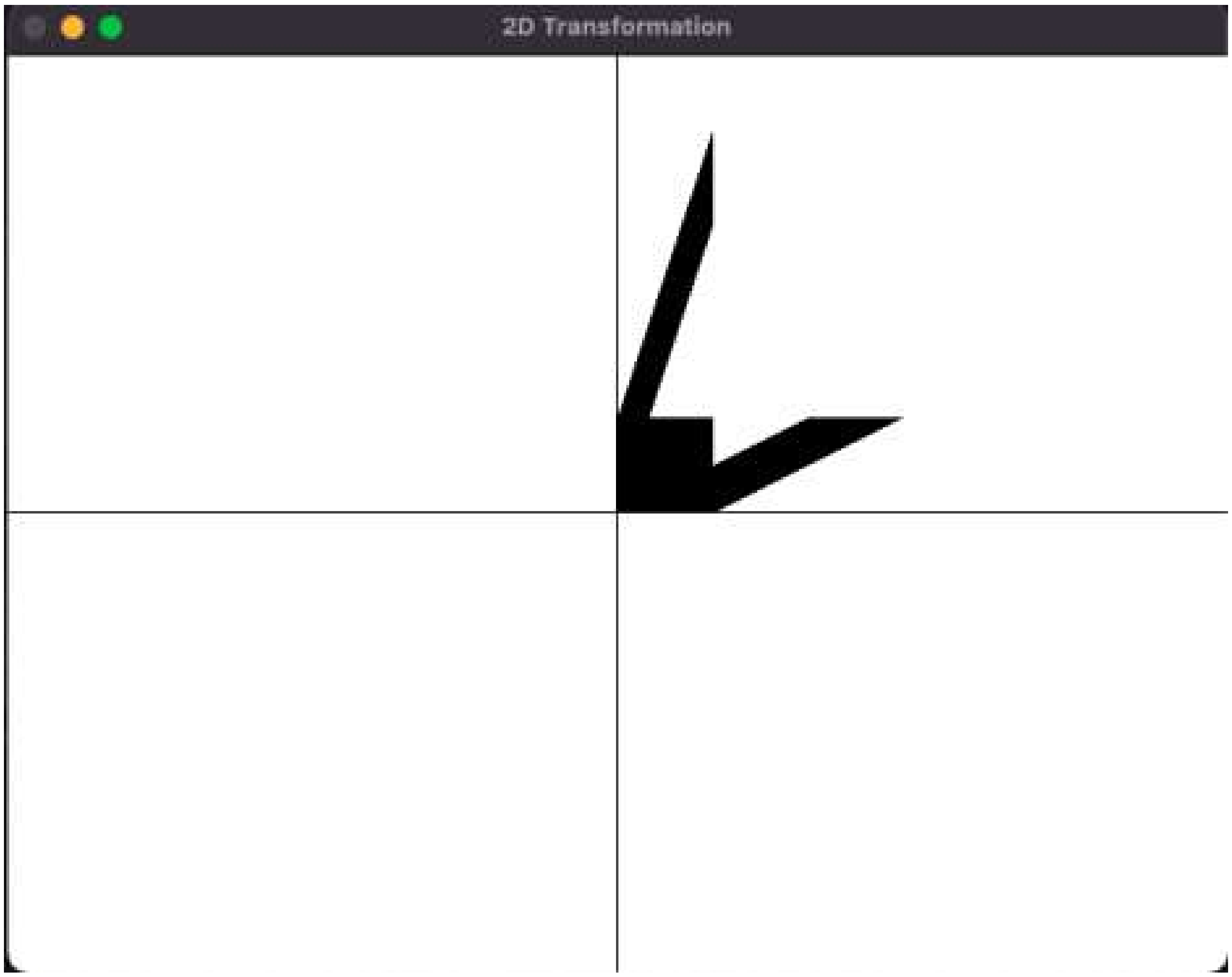



2D Transformation


2D Transformation


2D Transformation


2D Transformation



Learning Outcome:

Learnt to do composite transformations.

Learnt to do translation, reflection, shearing, rotation and scaling.

SSN COLLEGE OF ENGINEERING, KALAVAKKAM DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

UCS1712 - GRAPHICS AND MULTIMEDIA LAB

Assignment- 6 - 2D Composite Transformations and Windowing in C++ using OpenGL

Name: Sabarivasan V **Reg.No: 205001085**

Aim:

To compute the composite transformation matrix for any 2 transformations input by the user and apply it on the object.

1) Translation
2) Rotation
3) Scaling
4) Reflection
5) Shearing

Display the original and the transformed object.

Algorithm:

1. Get points of the object as input.
2. Draw the object.
3. Transform each vertex of the object.
4. Draw the object with the transformed vertices.

Code:

```
#define GL_SILENCE_DEPRECATION
#include<GLUT/glut.h>
#include<stdio.h>
#include<iostream>
#include<math.h> using
namespace std; float
toRad(float xDeg) { return
xDeg * 3.14159 / 180; }
void myInit() {
glClearColor(1, 1, 1, 1);
//
```

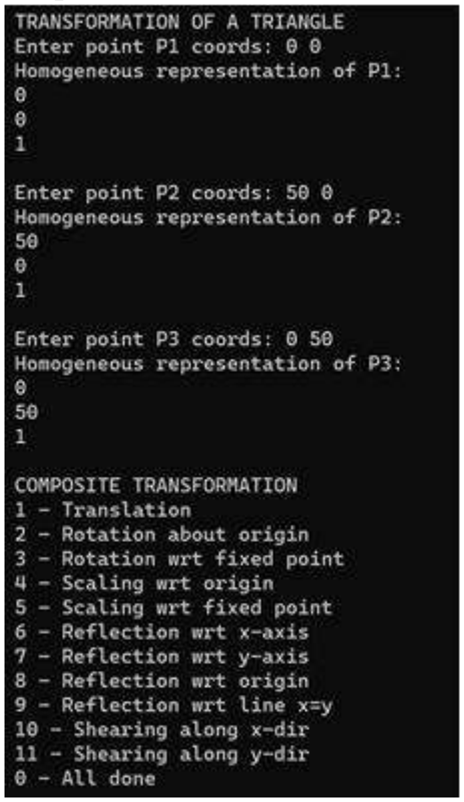
```
violet glColor3f(0.0f, 0.0f, 0.5f); //dark blue
//glPointSize(10);
glMatrixMode(GL_PROJECTION);
glLineWidth(2);


glLoadIdentity(); gluOrtho2D(0.0,
640.0, 0.0, 480.0);
} void displayPoint(float x, float y) {
glBegin(GL_POINTS); glVertex2d(x + 320, y
+ 240); glEnd(); } void
displayHomogeneousPoint(float* h) { float x
= *(h + 0); float y = *(h + 1); glColor4f(0, 1,
0.4, 1); //green displayPoint(x, y); } void
displayLine(int x1, int y1, int x2, int y2) {
glBegin(GL_LINES); glVertex2d(x1 + 320, y1
+ 240); glVertex2d(x2 + 320, y2 +
240); glEnd(); } void displayTriangle(int x1, int y1, int x2, int
y2, int x3, int y3) { glBegin(GL_TRIANGLES); glVertex2d(x1
+ 320, y1 + 240); glVertex2d(x2 + 320, y2 + 240);
glVertex2d(x3 + 320, y3 +
240); glEnd(); } void displayTransformedTriangle(float* p1,
float* p2, float* p3) { float x1 = *(p1 + 0); float y1 = *(p1 + 1);
float x2 = *(p2 + 0);
float y2 = *(p2 + 1);
float x3 =
*(p3 + 0); float y3 =
*(p3 + 1);
glColor4f(0, 1, 0.4, 1); //green
displayTriangle(x1, y1, x2, y2, x3, y3);
} void drawPlane() {
glClear(GL_COLOR_BUFFER_BIT);
glColor4f(0, 0, 0, 1); //yellow
displayLine(-320, 0, 320, 0); //x-axis
displayLine(0, -240, 0, 240); //y-axis
glFlush();


} void printMenu() { cout << "1 - Translation"
<< endl; cout << "2 - Rotation about origin"
<< endl; cout << "3 - Rotation wrt fixed point"
<< endl; cout << "4 - Scaling wrt origin" <<
```

```
endl; cout << "5 - Scaling wrt fixed point" <<
endl; cout << "6 - Reflection wrt x-axis" <<
endl; cout << "7 - Reflection wrt y-axis" <<
endl; cout << "8 - Reflection wrt origin" <<
endl; cout << "9 - Reflection wrt line x=y" <<
endl; cout << "10 - Shearing along x-dir" <<
endl; cout << "11 - Shearing along y-dir" <<
endl; cout << "0 - All done" << endl;
} void printMatrix(float* arr, int m, int
n)
{
int i, j;
for (i = 0; i < m; i++) { for (j =
0; j < n; j++) cout << *((arr +
i*n) + j) << " "; cout << endl; }
}
float* mulMatrix(float* a, int m1, int n1, float* b, int m2, int n2) {
if (n1 != m2) { cout << "Multiplication Input Error" << endl;
return NULL; } float* res = new float[m1 * n2]; for (int i = 0; i <
m1; i++) { for (int j = 0; j < n2; j++) { *((res + i*n2) + j) = 0; for
(int k = 0; k < n1; k++) {
*((res + i*n2) + j) += *((a + i*n1) + k) * *((b + k*n2) + j);
}
} } return res; } void
printPoint(float* P) {
printMatrix(P, 3, 1);
}


void printMatrix3(float* M) {
printMatrix(M, 3, 3);
} float* transformPoint(float* m, float* p)
{ return mulMatrix(m, 3, 3, p, 3, 1);
} float* mulTransforms(float* m1, float*
m2)
{ return mulMatrix(m1, 3, 3, m2, 3, 3);
} float* getTransformationMatrix()
{
cout << "COMPOSITE TRANSFORMATION" << endl;
float* compositeMatrix = new float[3 * 3]; for (int i = 0;
i < 3; i++) { for (int j = 0; j < 3; j++) {
compositeMatrix[i*3 + j] = (i == j) ? 1 :
0;
} }
printMenu();
int ch;
do {
cout << "\nChoose required transformation: ";
cin >> ch; switch (ch) { case 1: { cout <<
"TRANSLATION" << endl;
float tx, ty; cout << "Enter
translation values: "; cin >> tx >>
ty; float T[3][3] = { {1, 0, tx},
{0, 1, ty},
{0, 0, 1}
}; float* temp = mulTransforms((float*)T,
compositeMatrix); delete[] compositeMatrix;
compositeMatrix = temp; break; } case 2: { cout
<< "ROTATION ABOUT ORIGIN" << endl;
float angle; cout << "Enter
rotation angle: "; cin >> angle;


float theta = toRad(angle);
float c = cos(theta); float s
= sin(theta); float R[3][3]
= { {c, -s, 0},
{s, c, 0},
{0, 0, 1}
}; float* temp = mulTransforms((float*)R,
compositeMatrix); delete[] compositeMatrix;
compositeMatrix = temp; break;
} case 3: { cout << "ROTATION WRT FIXED
POINT" << endl; float angle; cout << "Enter rotation
angle: "; cin >> angle; float theta = toRad(angle);
float c = cos(theta); float s = sin(theta); float xr, yr;
cout << "Enter fixed point coords: "; cin >> xr >> yr;
float R[3][3] = { {c, -s, (xr * (1-c)) + (yr * s)},
{s, c, (yr * (1-c)) - (xr * s)},
{0, 0, 1}
}; float* temp = mulTransforms((float*)R,
compositeMatrix); delete[] compositeMatrix;
compositeMatrix = temp; break; } case 4: {
cout << "SCALING WRT ORIGIN" << endl;
float sx, sy; cout << "Enter scaling
factor values: "; cin >> sx >> sy; float
S[3][3] = { {sx, 0, 0},
{0, sy, 0},
{0, 0, 1}


}; float* temp = mulTransforms((float*)S,
compositeMatrix); delete[] compositeMatrix;
compositeMatrix = temp; break; } case 5: { cout <<
"SCALING WRT FIXED POINT" << endl;
float sx, sy; cout << "Enter scaling
factor values: "; cin >> sx >> sy;
float xf, yf; cout << "Enter fixed
point coords: "; cin >> xf >> yf;
float S[3][3] = { {sx, 0, xf * (1-sx)},
{0, sy, yf * (1-sy)},
{0, 0, 1}
}; float* temp = mulTransforms((float*)S,
compositeMatrix); delete[] compositeMatrix;
compositeMatrix = temp; break; } case 6: { cout <<
"REFLECTION WRT X-AXIS" << endl; float RF[3][3] =
{ {1, 0, 0},
{0, -1, 0},
{0, 0, 1}
}; float* temp = mulTransforms((float*)RF,
compositeMatrix); delete[] compositeMatrix;
compositeMatrix = temp; break; } case 7: { cout
<< "REFLECTION WRT Y-AXIS" << endl;
float RF[3][3] = { {-1, 0, 0},
{0, 1, 0},
{0, 0, 1}
};
```

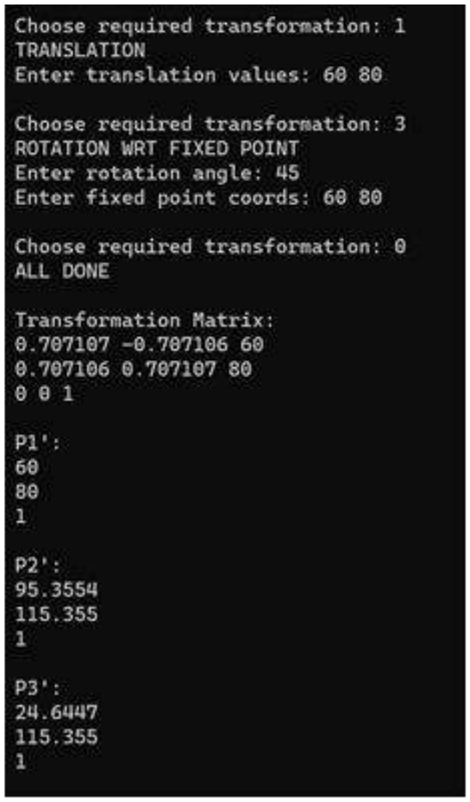
```
float* temp = mulTransforms((float*)RF,
compositeMatrix); delete[] compositeMatrix;
compositeMatrix = temp; break; }
case 8: { cout << "REFLECTION WRT
ORIGIN" << endl; float RF[3][3] = { {-1, 0, 0},
{0, -1, 0},
{0, 0, 1}
}; float* temp = mulTransforms((float*)RF,
compositeMatrix); delete[] compositeMatrix;
compositeMatrix = temp; break; } case 9: { cout <<
"REFLECTION WRT LINE X=Y" << endl; float
RF[3][3] = { {0, 1, 0},
{1, 0, 0},
{0, 0, 1}
}; float* temp = mulTransforms((float*)RF,
compositeMatrix); delete[] compositeMatrix;
compositeMatrix = temp; break;
} case 10: { cout << "SHEARING ALONG X-
DIR" << endl;
float shx, yref = 0; cout <<
"Enter shear value: "; cin >>
shx; cout << "Enter yref
value: "; cin >> yref; float
SH[3][3] = { {1, shx,-shx *
yref},
{0, 1, 0},
{0, 0, 1}
}; float* temp = mulTransforms((float*)SH,
compositeMatrix); delete[] compositeMatrix;


compositeMatrix = temp;
break; } case 11: { cout << "SHEARING
ALONG Y-DIR" << endl;
float shy, xref = 0; cout <<
"Enter shear value: "; cin >>
shy; cout << "Enter yref
value: "; cin >> xref; float
SH[3][3] = { {1, 0, 0},
{shy, 1, -shy * xref},
{0, 0, 1}
}; float* temp = mulTransforms((float*)SH,
compositeMatrix); delete[] compositeMatrix;
compositeMatrix = temp; break; } case 0: {
cout << "ALL DONE" << endl;
}
default:
break;
}
} while (ch != 0); return
compositeMatrix;
} void
plotTransform()
{
cout << "TRANSFORMATION OF A TRIANGLE" << endl;
//Point P1 float x1, y1; cout << "Enter point P1 coords:
"; cin >> x1 >> y1; float* P1 = new float[3]{ {x1},
{y1}, {1} }; cout << "Homogeneous representation of
P1: " << endl; printPoint(P1); cout << endl; //Point P2
float x2, y2; cout << "Enter point P2 coords: ";
cin >> x2 >> y2;


float* P2 = new float[3] { {x2}, { y2 }, { 1 } }; cout <<
"Homogeneous representation of P2: " << endl;
printPoint(P2); cout << endl; //Point P3 float x3, y3;
cout << "Enter point P3 coords: "; cin >> x3 >> y3;
float* P3 = new float[3] { {x3}, { y3 }, { 1 } }; cout <<
"Homogeneous representation of P3: " << endl;
printPoint(P3); cout << endl;
//plot triangle displayTriangle(x1, y1,
x2, y2, x3, y3); float* M =
getTransformationMatrix(); if (M !=
NULL) {
cout << "\nTransformation Matrix: " <<
endl; printMatrix3(M); cout << "\nP1': "
<< endl; float* Q1 = transformPoint(M,
P1); printPoint(Q1); cout << "\nP2': " <<
endl; float* Q2 = transformPoint(M, P2);
printPoint(Q2); cout << "\nP3': " << endl;
float* Q3 = transformPoint(M, P3);
printPoint(Q3);
```

```
displayTransformedTriangle(Q1, Q2, Q3);
delete[] Q1; delete[] Q2; delete[] Q3; }
delete[] M; delete[] P1; delete[] P2;
delete[] P3; } void plotChart() {
glClear(GL_COLOR_BUFFER_BIT);
drawPlane(); plotTransform(); glFlush(); }


int main(int argc, char* argv[]) {
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA);
glutInitWindowSize(640, 480);
glutCreateWindow("Transformations");
glutDisplayFunc(plotChart); myInit(); glutMainLoop();
return 1; }
```

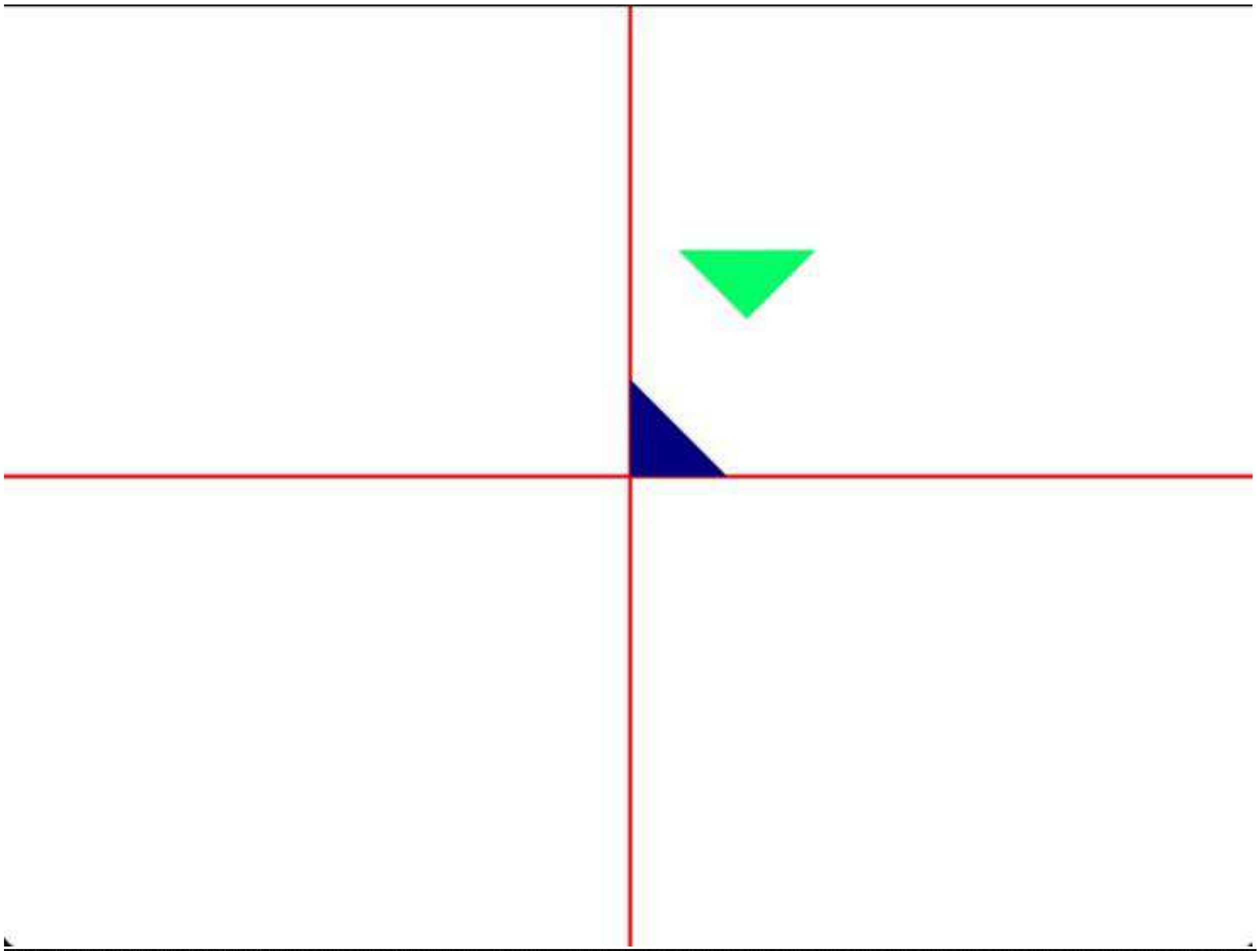
Output:

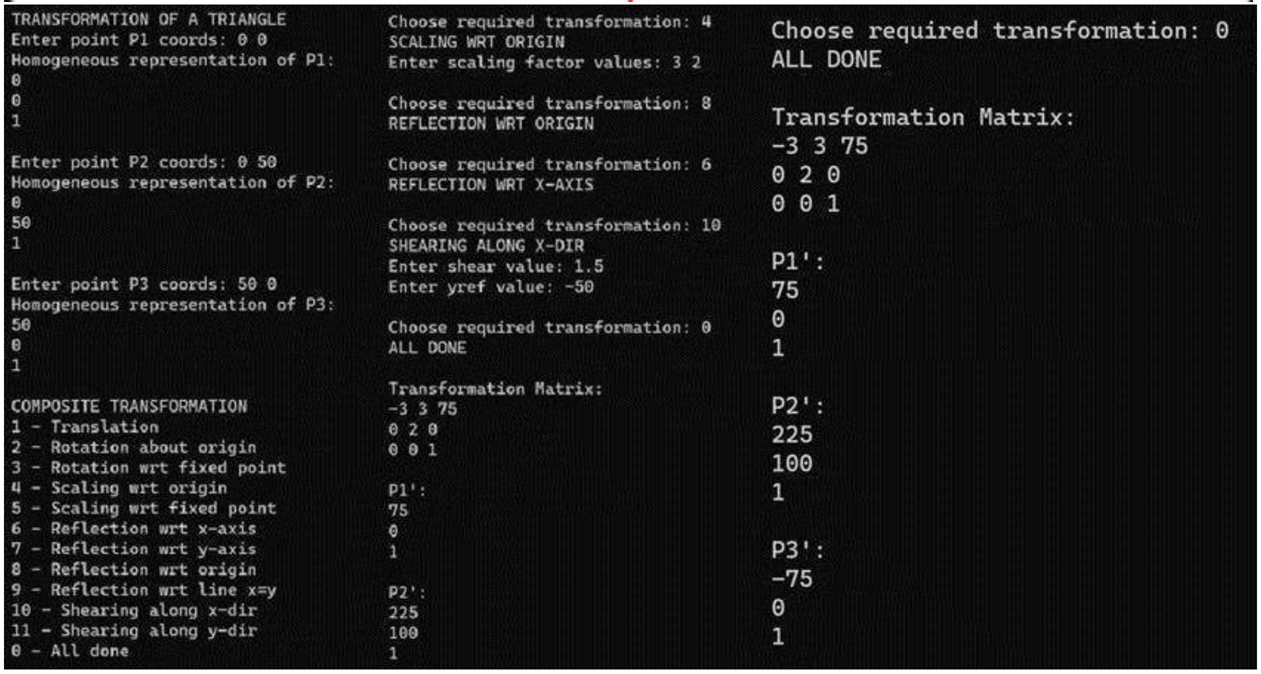


```
TRANSFORMATION OF A TRIANGLE
Enter point P1 coords: 0 0
Homogeneous representation of P1:
0
0
1

Enter point P2 coords: 50 0
Homogeneous representation of P2:
50
0
1

Enter point P3 coords: 0 50
Homogeneous representation of P3:
0
50
1

COMPOSITE TRANSFORMATION
1 - Translation
2 - Rotation about origin
3 - Rotation wrt fixed point
4 - Scaling wrt origin
5 - Scaling wrt fixed point
6 - Reflection wrt x-axis
7 - Reflection wrt y-axis
8 - Reflection wrt origin
9 - Reflection wrt line x=y
10 - Shearing along x-dir
11 - Shearing along y-dir
0 - All done
```



```
Choose required transformation: 1
TRANSLATION
Enter translation values: 60 80

Choose required transformation: 3
ROTATION WRT FIXED POINT
Enter rotation angle: 45
Enter fixed point coords: 60 80

Choose required transformation: 0
ALL DONE

Transformation Matrix:
0.707107 -0.707106 60
0.707106 0.707107 80
0 0 1

P1':
60
80
1

P2':
95.3554
115.355
1

P3':
24.6447
115.355
1
```

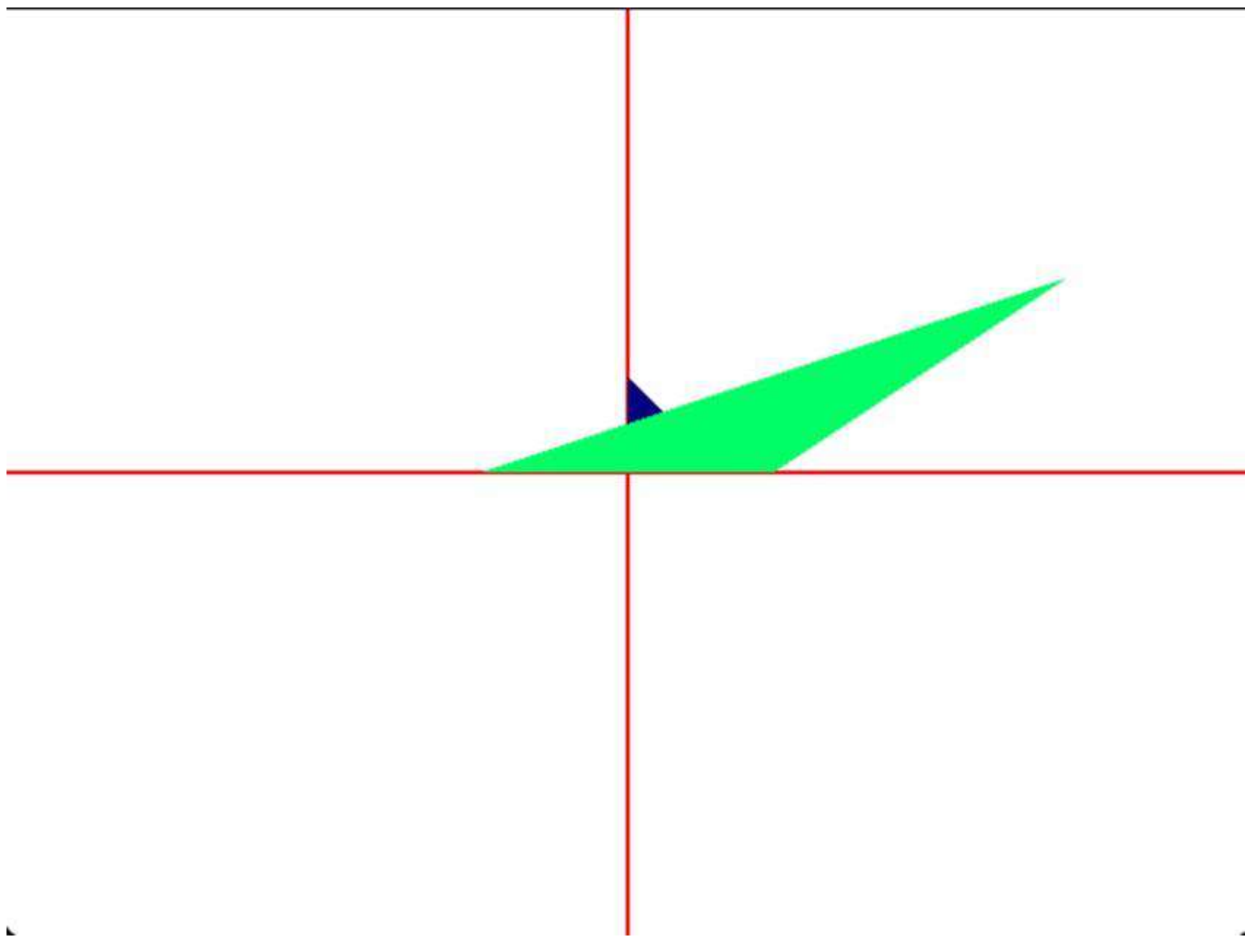




```
TRANSFORMATION OF A TRIANGLE
Enter point P1 coords: 0 0
Homogeneous representation of P1:
0
0
1

Enter point P2 coords: 0 50
Homogeneous representation of P2:
0
50
1

Enter point P3 coords: 50 0
Homogeneous representation of P3:
50
0
1

COMPOSITE TRANSFORMATION
1 - Translation
2 - Rotation about origin
3 - Rotation wrt fixed point
4 - Scaling wrt origin
5 - Scaling wrt fixed point
6 - Reflection wrt x-axis
7 - Reflection wrt y-axis
8 - Reflection wrt origin
9 - Reflection wrt line x=y
10 - Shearing along x-dir
11 - Shearing along y-dir
0 - All done
```

```
Choose required transformation: 4
SCALING WRT ORIGIN
Enter scaling factor values: 3 2

Choose required transformation: 8
REFLECTION WRT ORIGIN

Choose required transformation: 6
REFLECTION WRT X-AXIS

Choose required transformation: 10
SHEARING ALONG X-DIR
Enter shear value: 1.5
Enter yref value: -50

Choose required transformation: 0
ALL DONE

Transformation Matrix:
-3 3 75
0 2 0
0 0 1

P1':
75
0
1

P2':
225
100
1
```

```
Choose required transformation: 0
ALL DONE

Transformation Matrix:
-3 3 75
0 2 0
0 0 1

P1':
75
0
1

P2':
225
100
1

P3':
-75
0
1
```



Aim:

Create a window with any 2D object and a different sized viewport. Apply window to viewport transformation on the object. Display both window and viewport.

Algorithm:

1. Store the window dimensions and the viewport dimensions.
2. Get points of the object as input and draw it on the window.
3. Apply window to viewport transformation on the object as:
    a. Sx = (xvmax - xvmin) / (xwmax - xwmin)
    b. xv = xvmin + (xw - xwmin) * Sx
    c. Similarly, for the y-coordinates.
4. Draw the object on the viewport.

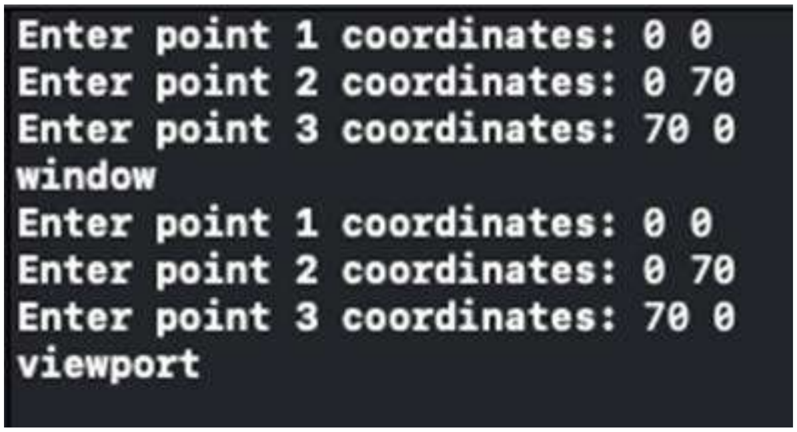
Code:

```
#define GL_SILENCE_DEPRECATION
#include<GLUT/glut.h>
#include<stdio.h>
#include<iostream> #include<math.h> using namespace std;
float xvmax = 640, yvmax = 480, xwmax = 1280, ywmax = 960;
void myInit_window() { glClearColor(1,1,1,1.0);
glColor3f(0.0f,0.0f,0.0f); glPointSize(3); glLineWidth(3);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0.0,1280.0,0.0,960.0);
} void myInit_viewport() {
glClearColor(1,1,1,1.0);
glColor3f(0.0f,0.0f,0.0f);
glPointSize(3); glLineWidth(3);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0.0,640.0,0.0,480.0);
}
void displayaxes_window(){
glBegin(GL_LINES);
glColor4f(0,0.5,0,1); //y - axis
glVertex2d(640,0);
glVertex2d(640,960); //x - axis
glVertex2d(0,480);
glVertex2d(1280,480); glEnd();
```

```
} void
displayaxes_viewport(){
glBegin(GL_LINES);
glColor4f(0,0.5,0,1); //y -
axis glVertex2d(320,0);
glVertex2d(320,480); //x -
axis


glVertex2d(0,240);
glVertex2d(640,240)
; glEnd(); } void drawObject(int
window){ float x1, y1; cout <<
"Enter point 1 coordinates: "; cin >>
x1 >> y1;

float x2, y2; cout << "Enter point 2
coordinates: "; cin >> x2 >> y2;

float x3, y3; cout << "Enter point 3
coordinates: "; cin >> x3 >> y3;

if (window) { cout <<
"window\n";
glBegin(GL_TRIANGLES);
glColor4f(0.4,0,0.8,1); glVertex2d(x1 +
(xwmax/2), y1 + (ywmax/2)); glVertex2d(x2 +
(xwmax/2), y2 + (ywmax/2)); glVertex2d(x3 +
(xwmax/2), y3 + (ywmax/2)); glEnd();
glFlush(); } else { cout << "viewport\n"; float
sx = xvmax/xwmax, sy = yvmax/ywmax; float
S[3][3] = {{sx, 0, 0}, {0, sy, 0}, {0, 0, 1}};
float T[3][3] = {{x1, y1, 1}, {x2, y2, 1}, {x3,
y3, 1}}; float R[3][3] = {{0, 0, 0}, {0, 0, 0},
{0, 0, 0}};

for (int i = 0 ; i < 3 ; i++)
{ for (int j = 0; j < 3;
j++)
{ for (int k = 0; k < 3;
k++)
{
R[i][j] += S[i][k] * T[k][j];
}
} }
glBegin(GL_TRIANGLES);
glColor4f(0,0,0.8,1);



glVertex2d(R[0][0] + (xvmax/2), R[0][1] + (yvmax/2));
glVertex2d(R[1][0] + (xvmax/2), R[1][1] + (yvmax/2));
glVertex2d(R[2][0] + (xvmax/2), R[2][1] + (yvmax/2));
glEnd(); glFlush();
}

} void plotWindow_window() {
myInit_window();
glClear(GL_COLOR_BUFFER_BIT);
displayaxes_window();
drawObject(1);
glFlush();
glutSwapBuffers()
;
} void plotWindow_viewport() {
myInit_viewport();
glClear(GL_COLOR_BUFFER_BIT);
displayaxes_viewport();
drawObject(0);
glFlush();
glutSwapBuffers()
;
} int main(int argc, char* argv[])
{ glutInit(&argc, argv);
glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA);

glutInitWindowSize(xwmax, ywmax); int
window = glutCreateWindow("Window");

glutInitWindowSize(xvmax, yvmax); int
viewport = glutCreateWindow("Viewport");
```

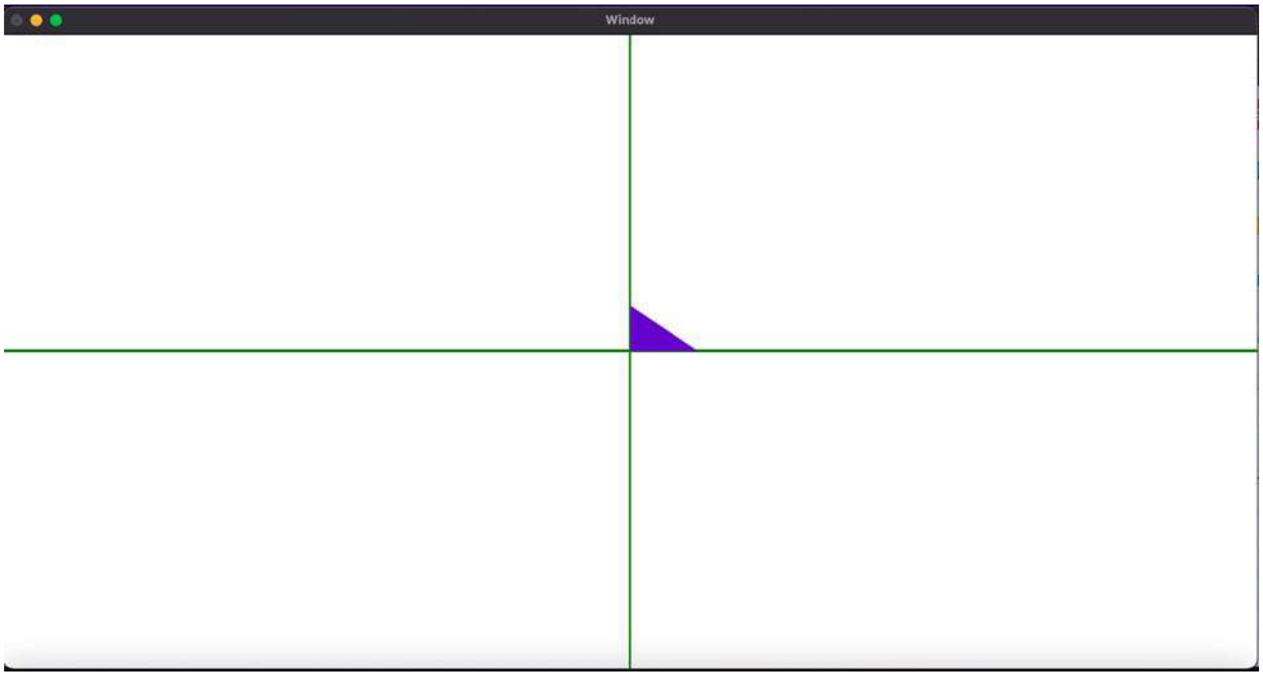
```
glutSetWindow(window);
glutDisplayFunc(plotWindow_window);

glutSetWindow(viewport);
glutDisplayFunc(plotWindow_viewport);


glutMainLoop();
return 1; }
```

Output:

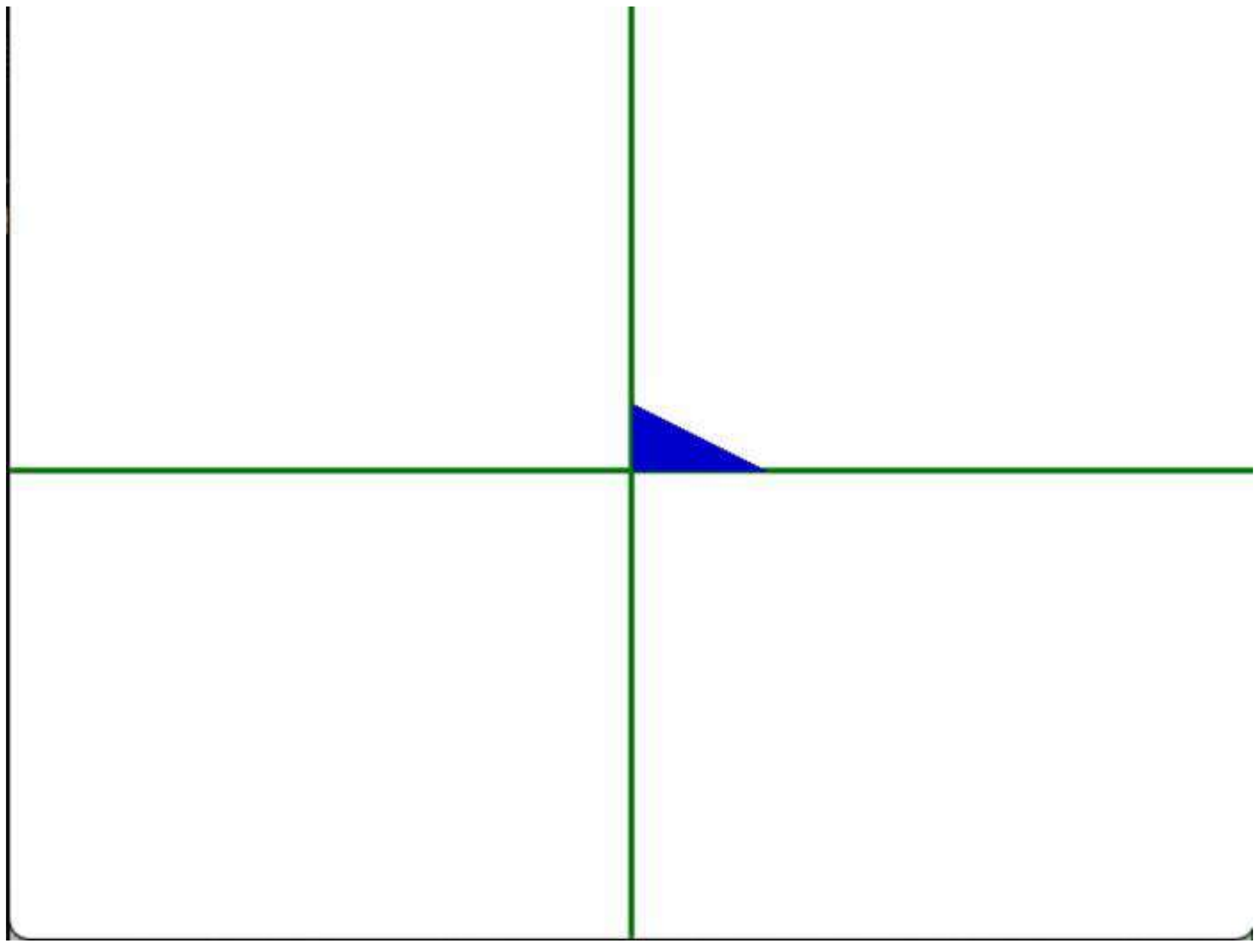


```
Enter point 1 coordinates: 0 0
Enter point 2 coordinates: 0 70
Enter point 3 coordinates: 70 0
window
Enter point 1 coordinates: 0 0
Enter point 2 coordinates: 0 70
Enter point 3 coordinates: 70 0
viewport
```

Window:



Viewport:



SSN COLLEGE OF ENGINEERING, KALAVAKKAM DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
UCS1712 - GRAPHICS AND MULTIMEDIA LAB

Assignment- 7 - Cohen Sutherland Line Clipping Algorithm

Name: Sabarivasan V **Reg.No: 205001085**

Aim:

Apply Cohen Sutherland line clipping on a line (x1,y1) (x2,y2) with respect to a clipping window (XWmin,YWmin) (XWmax,YWmax).

After clipping with an edge, display the line segment with the calculated intermediate intersection points.

ALGORITHM:

1) Assign the region codes to both endpoints. 2)
Perform OR operation on both of these
endpoints. 3) if OR = 0000, then it is completely visible
(inside the window).
Else
Perform AND operation on both these endpoints.
i) if AND ≠ 0000,
then the line is invisible and not inside the window. Also, it can't be considered for clipping. ii)
else
AND = 0000, the line is partially inside the window and considered for
Clipping.

4) After confirming that the line is partially inside the window, then we find the intersection with the boundary of the window. By using the following formula:-

Slope:- m= (y2-y1)/(x2-x1)

a) If the line passes through top or the line intersects with the top boundary of the window.
x = x + (y_wmax – y)/m y = y_wmax

b) If the line passes through the bottom or the line intersects with the bottom boundary of the window. x = x +
(y_wmin – y)/m y =
y_wmin

c) If the line passes through the left region or the line intersects with the left boundary of the window.

y = y+ (x_wmin – x)*m
x = x_wmin

d) If the line passes through the right region or the line intersects with the right boundary of the window.

y = y + (x_wmax
-x)*m x = x_wmax
5) Now, overwrite the endpoints with a new one and update it.
6) Repeat the 4th step till your line doesn't get completely clipped

CODE:

```
#include <iostream>
#include<GLUT/glut.h>
using namespace std;
void myInit() {
glClearColor(1,1,1,0.0);
glColor3f(1.0f,1.0f,1.0f);
glPointSize(3);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0.0,640.0,0.0,480.0);
}
// Defining region codes
const int INSIDE = 0; //
0000 const int LEFT = 1; //
0001 const int RIGHT = 2; //
0010 const int BOTTOM = 4; //
0100 const int TOP = 8; // 1000
// Defining x_max, y_max and x_min, y_min for
// clipping rectangle. Since diagonal points are
// enough to define a rectangle
const int x_max = 400;
const int y_max
= 400; const int x_min
= 200; const int y_min
= 200;
// Function to compute region code for a point(x, y)
int computeCode(double x, double y)
{
// initialized as being inside int code =
INSIDE; if (x < x_min) // to the left of
rectangle code |= LEFT; else if (x > x_max)
// to the right of rectangle code |= RIGHT;
if (y < y_min) // below the rectangle
code |= BOTTOM;
```

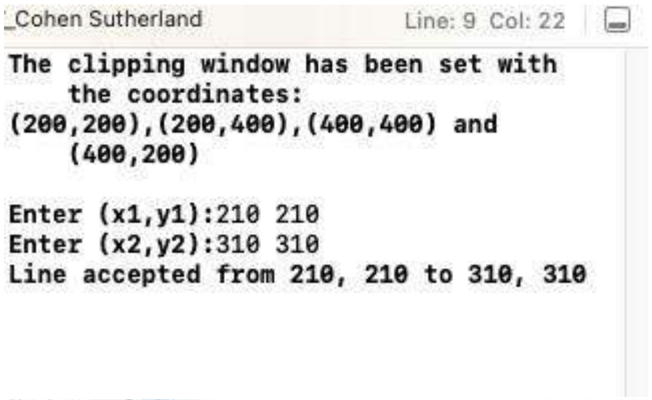
```
else if (y > y_max) // above the rectangle
code |= TOP; return code;
}
// Implementing Cohen-Sutherland algorithm //
Clipping a line from P1 = (x2, y2) to P2 = (x2, y2)
void lineclip(double x1, double y1, double x2,
double y2)
{
// Compute region codes for P1, P2
int code1 = computeCode(x1, y1);
int code2 = computeCode(x2, y2);
// Initialize line as outside the rectangular window
bool accept = false; while (true) { if ((code1 ==
0) && (code2 == 0)) { // If both endpoints lie
within rectangle accept = true; break; } else if
(code1 & code2) {
// If both endpoints are outside rectangle,
// in same region
break; } else {
// Some segment of line lies within the
// rectangle int
code_out;
double x=0, y=0;
// At least one endpoint is outside the
// rectangle, pick it. if (code1 != 0)
code_out = code1; else
code_out = code2;
// Find intersection point;
// using formulas y = y1 + slope * (x - x1),
// x = x1 + (1 / slope) * (y - y1) if
(code_out & TOP) {
// point is above the clip rectangle x = x1 +
(x2 - x1) * (y_max - y1) / (y2 - y1); y =
y_max; } else if (code_out & BOTTOM) {
// point is below the rectangle x = x1 + (x2 -
x1) * (y_min - y1) / (y2 - y1); y = y_min; }
else if (code_out & RIGHT) { // point is to
the right of rectangle y = y1 + (y2 - y1) *
(x_max - x1) / (x2 - x1); x = x_max; } else
if (code_out & LEFT) { // point is to the left
of rectangle y = y1 + (y2 - y1) * (x_min -
x1) / (x2 - x1); x = x_min;
}
// Now intersection point x, y is found
// We replace point outside rectangle
// by intersection point if (code_out
== code1) { x1 = x; y1 = y; code1 =
computeCode(x1, y1);
} else { x2 = x; y2 = y; code2
= computeCode(x2, y2);
}
} } if (accept) { cout << "Line accepted from
" << x1 << ", " << y1 << " to " << x2 << ", "
<< y2 << endl; glBegin(GL_LINES);
glColor3f(0.5f,0.1f,0.5f);
glVertex2d(x1,y1);
glVertex2d(x2,y2);
glEnd(); glFlush(); }
else cout << "Line rejected" <<
endl; } // Driver code void
myDisplay() { cout<<"The
clipping window has been set
with the coordinates:\n";
cout<<"(200,200),(200,400),(40
0,400) and (400,200)\n\n";
// First Line segment
int x1,x2,y1,y2;
cout<<"Enter (x1,y1):";
cin>>x1>>y1;
cout<<"Enter (x2,y2):";
cin>>x2>>y2;
glFlush();
glClear(GL_COLOR_BUFFER_BIT);
glBegin(GL_LINES);
glColor3f(0.5f,0.1f,0.1f);
glVertex2d(200,200);
glVertex2d(200,400);
glEnd(); glFlush();
glBegin(GL_LINES);
glColor3f(0.5f,0.1f,0.1f);
glVertex2d(200,400);
glVertex2d(400,400);
glEnd(); glFlush();
glBegin(GL_LINES);
glColor3f(0.5f,0.1f,0.1f);
glVertex2d(400,400);
```

```
glVertex2d(400,200);
glEnd(); glFlush();
glBegin(GL_LINES);
glColor3f(0.5f,0.1f,0.1f);
glVertex2d(400,200);
glVertex2d(200,200);
glEnd(); glFlush();
// P11 = (5, 5), P12 = (7, 7)
glBegin(GL_POINTS);
glVertex2d(x1,y1);
glEnd(); glFlush();
glBegin(GL_POINTS);
glVertex2d(x2,y2);
glEnd(); glFlush();
lineclip(x1,y1,x2,y2); //
Second Line segment
// P21 = (7, 9), P22 = (11, 4)
//lineclip(7,9, 11, 4);;
// Third Line segment
// P31 = (1, 5), P32 = (4, 1)
//lineclip(1, 5, 4, 1); } int
main(int argc,char* argv[]) {
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutInitWindowSize(640,480);
glutCreateWindow("Line Clipping");
glutDisplayFunc(myDisplay); myInit();
glutMainLoop()
; return 1;
}
```

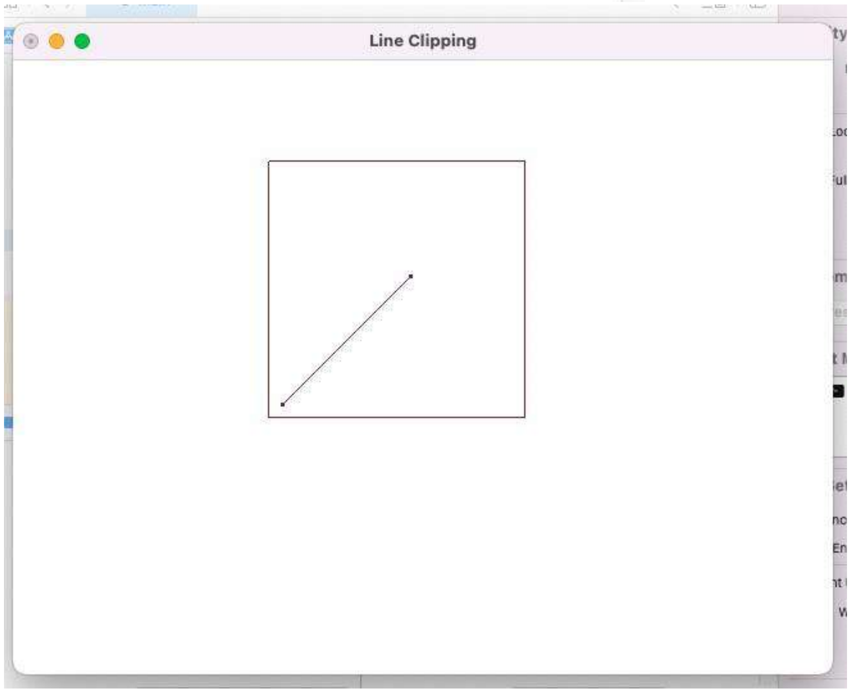
OUTPUT:

The black square is the clipping window

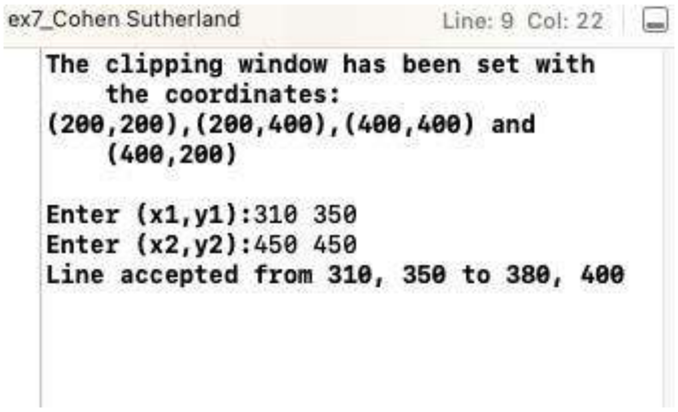
Case 1: Line lies within the clipping window



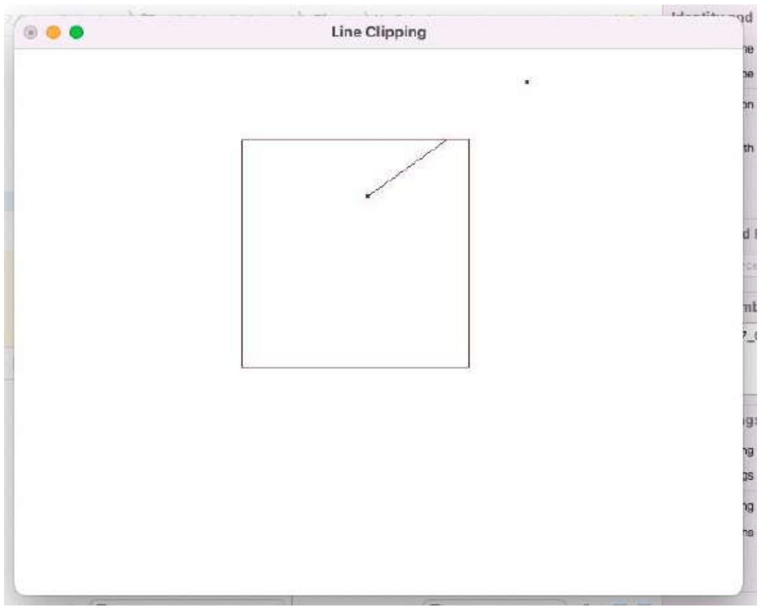
```
_Cohen Sutherland                    Line: 9  Col: 22
The clipping window has been set with
    the coordinates:
(200,200),(200,400),(400,400) and
    (400,200)

Enter (x1,y1):210 210
Enter (x2,y2):310 310
Line accepted from 210, 210 to 310, 310
```



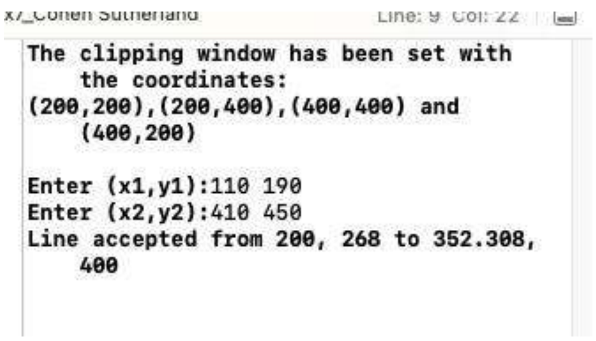
Case 2: One point lies inside the clipping window



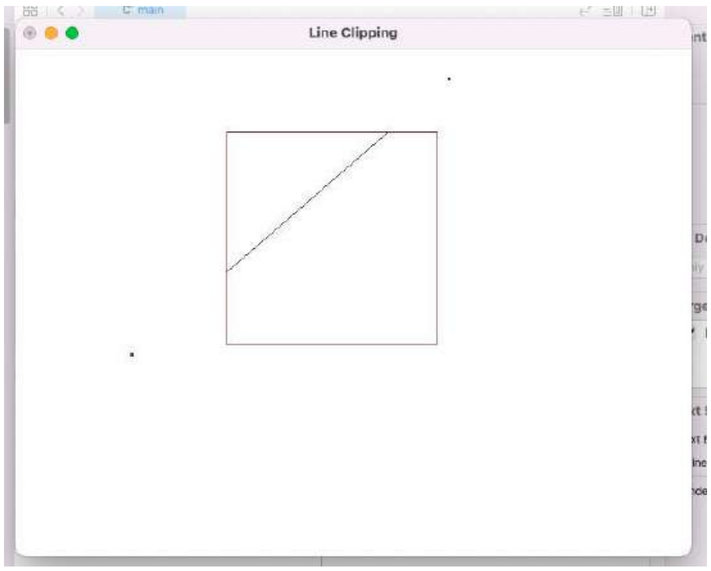
```
ex7_Cohen Sutherland                 Line: 9  Col: 22
The clipping window has been set with
    the coordinates:
(200,200),(200,400),(400,400) and
    (400,200)

Enter (x1,y1):310 350
Enter (x2,y2):450 450
Line accepted from 310, 350 to 380, 400
```



Case 3: Both points lies outside the Clipping window, but lines crosses through the clipping window



```
The clipping window has been set with
    the coordinates:
(200,200),(200,400),(400,400) and
    (400,200)

Enter (x1,y1):110 190
Enter (x2,y2):410 450
Line accepted from 200, 268 to 352.308,
    400
```



SSN COLLEGE OF ENGINEERING, KALAVAKKAM DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
UCS1712 - GRAPHICS AND MULTIMEDIA LAB

Assignment- 8 - 3-Dimensional Transformations in C++ using OpenGL

Name: Sabarivasan V **Reg.No: 205001085**

Ex.No: 08

Aim:

To perform the following basic 3D Transformations on any 3D Object:

1) Translation
2) Rotation
3) Scaling

Use only homogeneous coordinate representation and matrix multiplication to perform transformations. Set the camera to any position on the 3D space. Have (0,0,0) at the center of the screen. Draw X , Y and Z axes.

ALGORITHM:

1. Start the program.
2. Display the cube.
3. Input the translation vector tx,ty,tz.
4. Using the function line, display the object before and after translation.
5. Input the scaling factor and reference point.
6. Using the function line, display the object before and after scaling.
7. Input the rotation angle.
8. Using the function line,display the object before and after rotation.
9. Stop the Program.

CODE:

```
#include<stdio.h>
#include<GL/glut.h> //Change to <GLUT/glut.h> in Mac
#include<math.h>
#include<string.h>
#include<iostream>
using namespace std;
#define pi 3.142857
```

```
typedef float Matrix4[4][4];
Matrix4 theMatrix; static
GLfloat input[8][3] =
{{40,40,-50},{90,40,-50},{90,90,-50},{40,90,-50},{30,30,0},{80,30,0},{80,80,0},{30,80,0}};
float output[8][3]; float tx=100,
ty=100, tz=100; float sx=-2,
sy=2, sz=2; float angle=60; int
choice, choiceRot; void
setIdentityM(Matrix4 m) { for
(int i = 0; i < 4; i++) for (int j =
0; j < 4; j++) m[i][j] = (i == j);
}

//PUT SOME FUNCTION HERE

void translate(int tx, int ty, int tz) {
  for (int i = 0; i < 8; i++) {


              output[i][0] = input[i][0] + tx;
              output[i][1] = input[i][1] + ty;
              output[i][2] = input[i][2] + tz;
  } } void scale(int sx, int sy, int
sz) {
  theMatrix[0][0] = sx;
theMatrix[1][1] = sy;
theMatrix[2][2] = sz; } void
RotateX(float angle) { angle =
angle * 3.142 / 180;
theMatrix[1][1] = cos(angle);
theMatrix[1][2] = -sin(angle);
theMatrix[2][1] = sin(angle);
theMatrix[2][2] = cos(angle); }
void RotateY(float angle) {
angle = angle * 3.14 / 180;
theMatrix[0][0] = cos(angle);
theMatrix[0][2] = -sin(angle);
theMatrix[2][0] = sin(angle);
theMatrix[2][2] = cos(angle); }
void RotateZ(float angle) {
angle = angle * 3.14 / 180;
theMatrix[0][0] = cos(angle);
theMatrix[0][1] = sin(angle);
theMatrix[1][0] = -sin(angle);
theMatrix[1][1] = cos(angle);
```

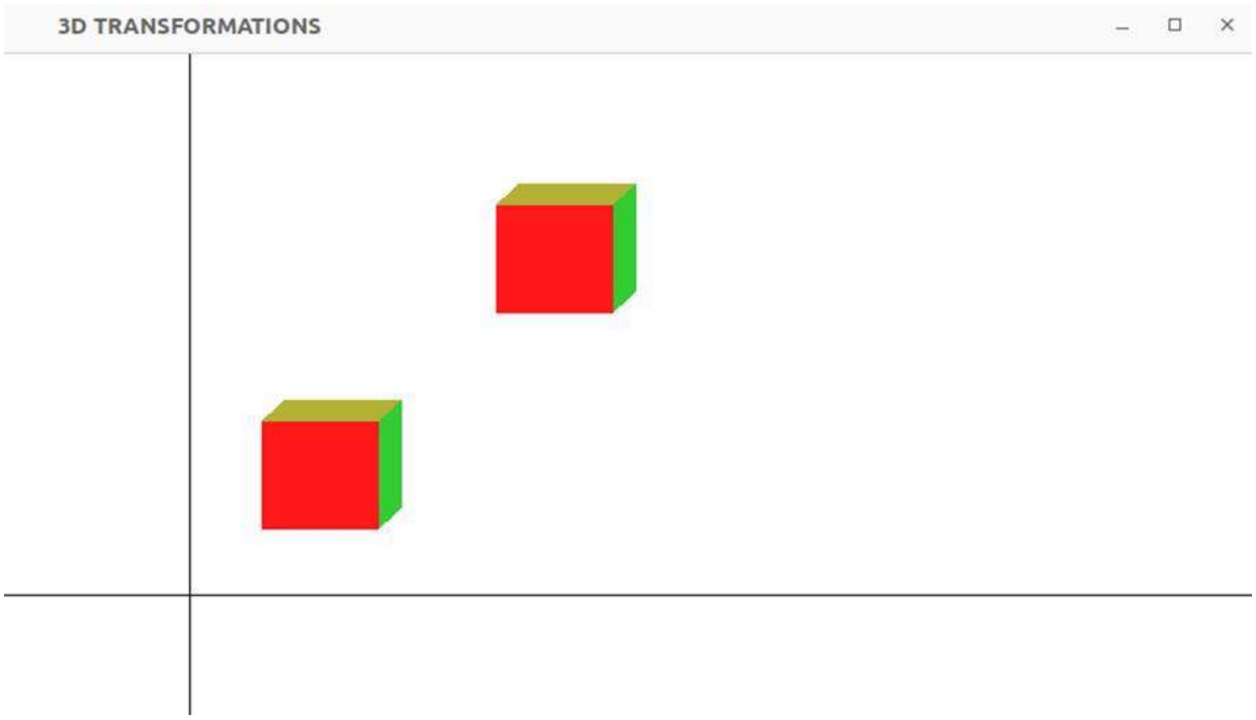
```
} void multiplyM()
{
  for (int i = 0; i < 8; i++) { for
            (int j = 0; j < 3; j++) {
            output[i][j] = 0;
            for (int k = 0; k < 3; k++) {
            output[i][j] = output[i][j] + input[i][k] * theMatrix[k][j];
            }
            }
  }
}

//To draw the solid void
draw(float a[8][3]) {
  glBegin(GL_QUADS);
  glColor3f(0.7, 0.4, 0.5); //behind
  glVertex3fv(a[0]);
  glVertex3fv(a[1]);
  glVertex3fv(a[2]);
  glVertex3fv(a[3]);
  glColor3f(0.8, 0.2, 0.4);
  //bottom glVertex3fv(a[0]);
  glVertex3fv(a[1]);
  glVertex3fv(a[5]);
  glVertex3fv(a[4]);
  glColor3f(0.3, 0.6, 0.7); //left
  glVertex3fv(a[0]);
  glVertex3fv(a[4]);
  glVertex3fv(a[7]);
  glVertex3fv(a[3]);
  glColor3f(0.2, 0.8, 0.2); //right
  glVertex3fv(a[1]);
  glVertex3fv(a[2]);
  glVertex3fv(a[6]);
  glVertex3fv(a[5]);
  glColor3f(0.7, 0.7, 0.2); //up
  glVertex3fv(a[2]);
  glVertex3fv(a[3]);
  glVertex3fv(a[7]);
  glVertex3fv(a[6]);
  glColor3f(1.0, 0.1, 0.1);
  glVertex3fv(a[4]);
  glVertex3fv(a[5]);
  glVertex3fv(a[6]);
  glVertex3fv(a[7]);
  glEnd();
```

```
}


/* This is just to call the functions also draw X and Y axis
 here and use output to label stuff) */
void display (void){

            glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
            //black glColor3f(0.0,
            0.0, 0.0);

  glBegin(GL_LINES); // Plotting X-Axis
  glVertex3d(-1000, 0, 0);
  glVertex3d(1000, 0, 0); glEnd();
  glBegin(GL_LINES); // Plotting Y-Axis
  glVertex3d(0, -1000, 0); glVertex3d(0,
  1000, 0); glEnd();
  glBegin(GL_LINES); // Plotting Z-Axis
  glVertex3d(0, 0, -1000); glVertex3d(0,
  0, 1000); glEnd();


            //Call function
   draw(input);
  setIdentityM(theMatrix);
  switch (choice) { case 1:
            translate(tx, ty, tz);
            break;
  case 2:
            scale(sx, sy, sz);
            multiplyM();
            break;
  case 3: switch (choiceRot)
            { case 1:
            RotateX(angle);
            break; case 2:
            RotateY(angle);
            break; case 3:
            RotateZ(angle);
  break; default: break;
  } multiplyM(); break;
  } draw(output);
  glFlush();

            glFlush();
}
```

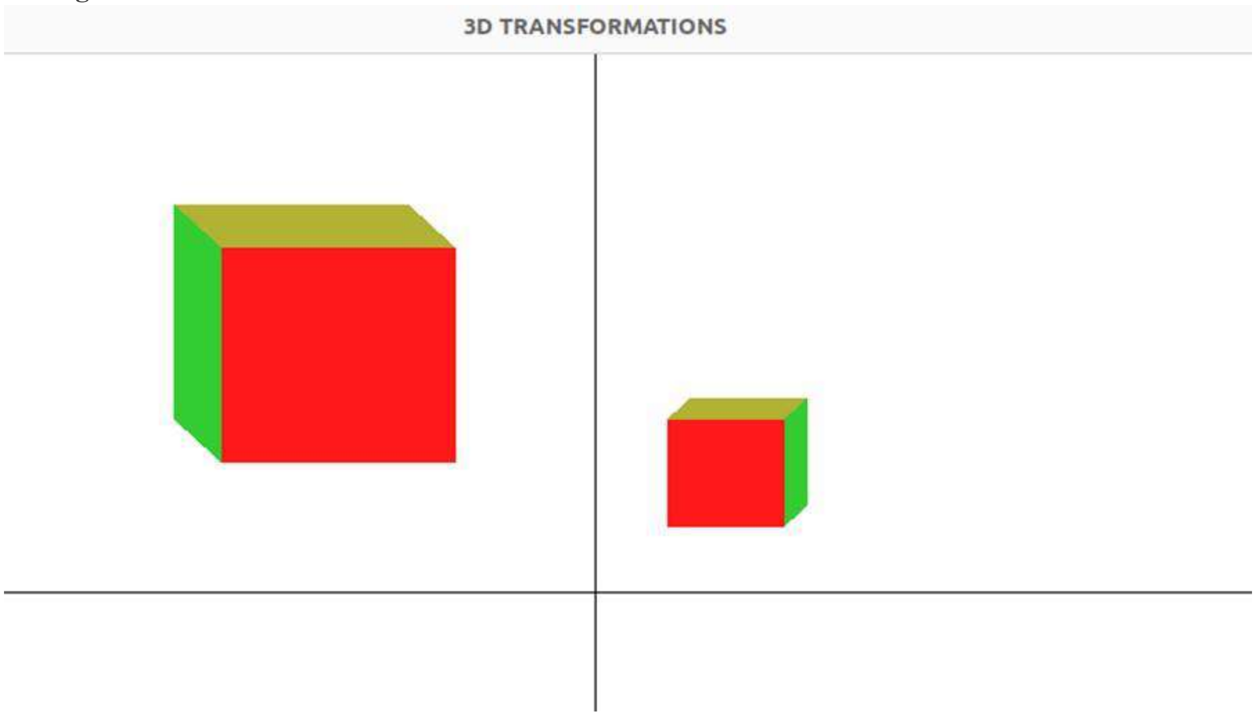
```
int main (int argc, char** argv){ /*--------WINDOW INITS-------*/
            glutInit(&argc, argv); //Mandatory. Initializes the GLUT library.
            glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
            glutInitWindowSize(1380, 700); //Set the size of output window (kinda optional)
            glutInitWindowPosition(200, 200); //position of output window on screen (optional)
            glutCreateWindow("3D TRANSFORMATIONS");// Giving name to window

            /*-------OTHER INITS-------*/ glClearColor(1.0, 1.0, 1.0, 1.0); //Clear the buffer values
            for color AND set these values
            /*can set initial color here also*/ glMatrixMode(GL_PROJECTION); //Uses something
            called "projection matrix" to represent glLoadIdentity(); //load the above matrix to fill with
            identity values glOrtho(-454.0, 454.0, -250.0, 250.0, -250.0, 250.0); gluPerspective(100, 100,
            100, 100); glEnable(GL_DEPTH_TEST); cout << "Enter your choice
            number:\n1.Translation\n2.Scaling\n3.Rotation\n=>";
  cin >> choice;
  switch (choice)
  { case 1: break;
  case 2:
            break;
  case 3:
            cout << "Enter your choice for Rotation about axis:\n1.parallel to X-axis." <<
            "(y& z)\n2.parallel to Y-axis.(x& z)\n3.parallel to Z-axis." <<
            "(x& y)\n =>";
            cin >>
            choiceRot; break;
  default:
            break;
  }
   glutDisplayFunc(display); //sets the display callback for the current window
   glutMainLoop(); //Enters event processing loop. Compulsory return 0;
}
```

OUTPUT:

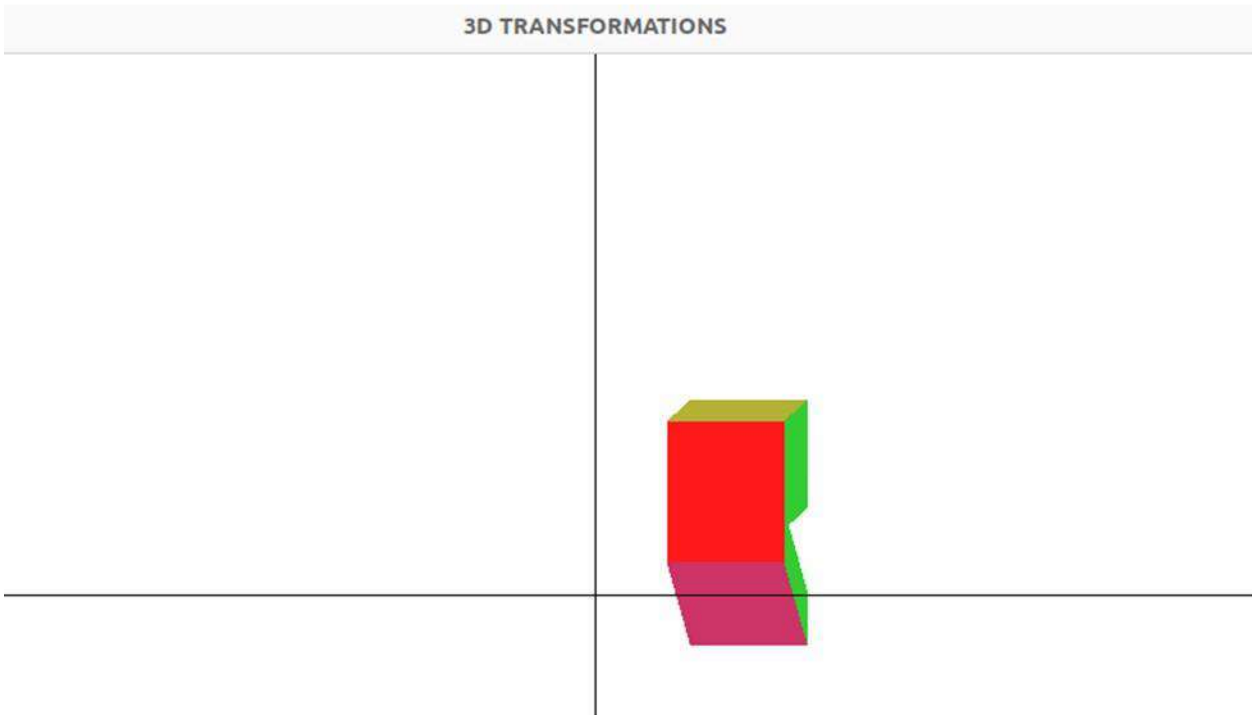
Translate along X and Y and Z



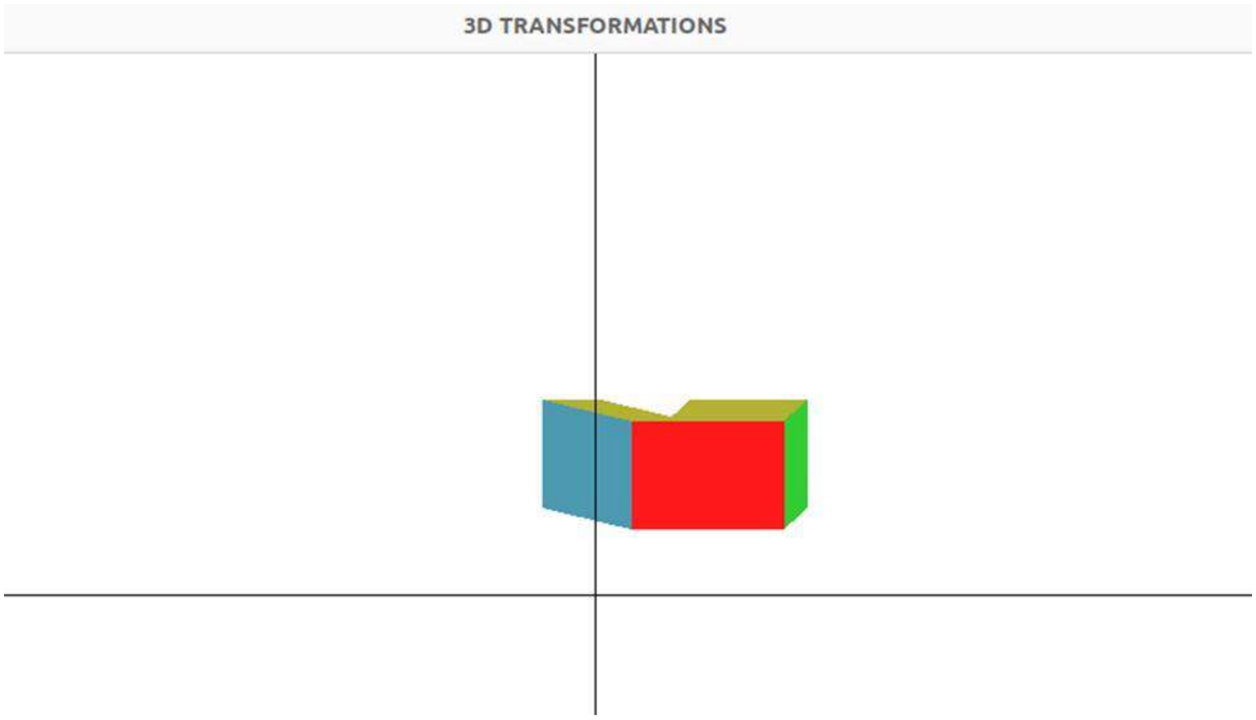
Scaling at X Y and Z



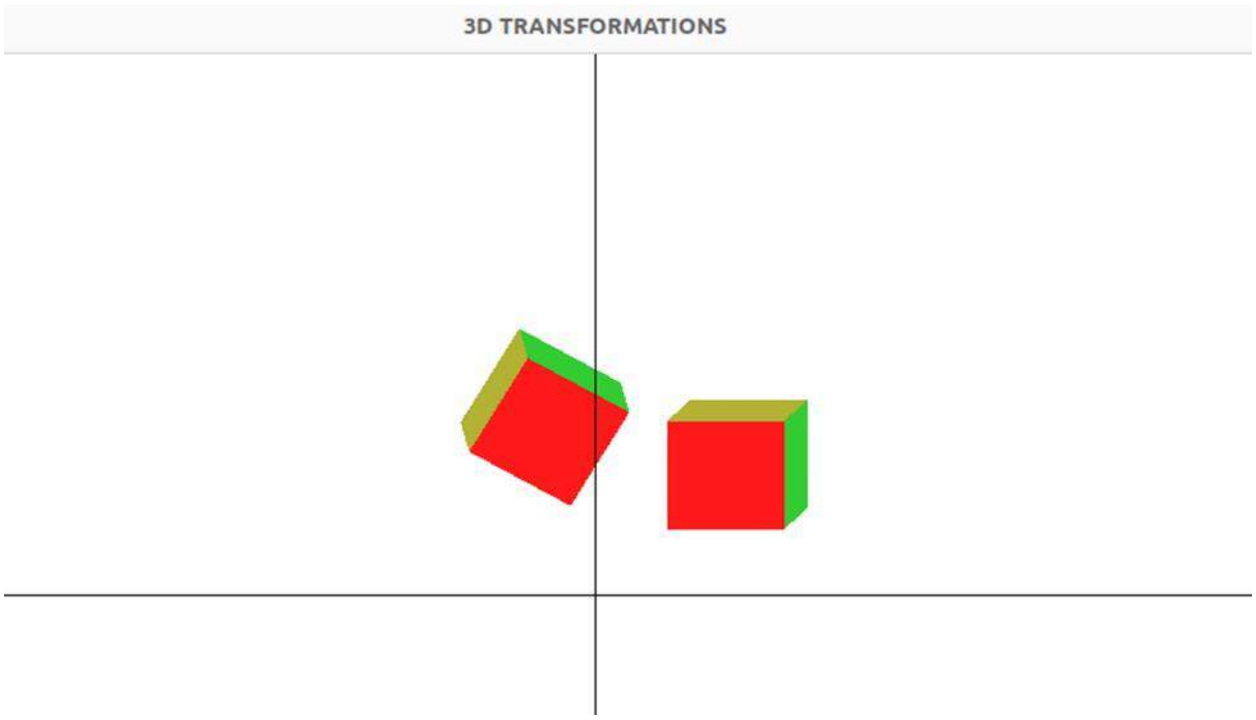
Rotation wrt X



Rotate wrt Y

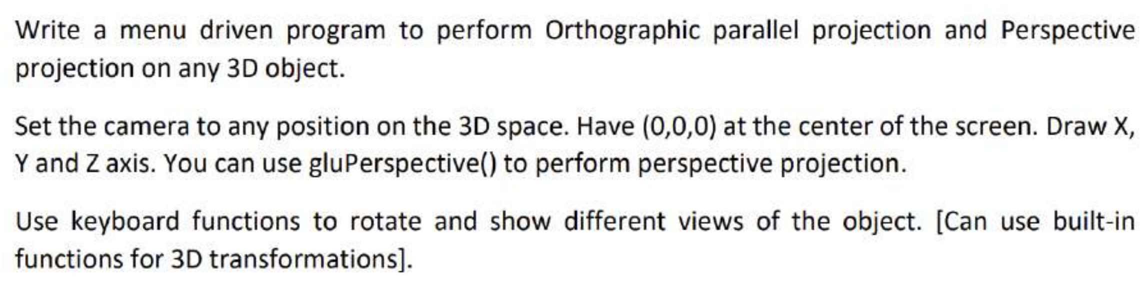


Rotate wrt Z


3D TRANSFORMATIONS

GML Ex 9
Sabarivasan V
CSE-B
205001085

Question:



Write a menu driven program to perform Orthographic parallel projection and Perspective projection on any 3D object.

Set the camera to any position on the 3D space. Have (0,0,0) at the center of the screen. Draw X, Y and Z axis. You can use gluPerspective() to perform perspective projection.

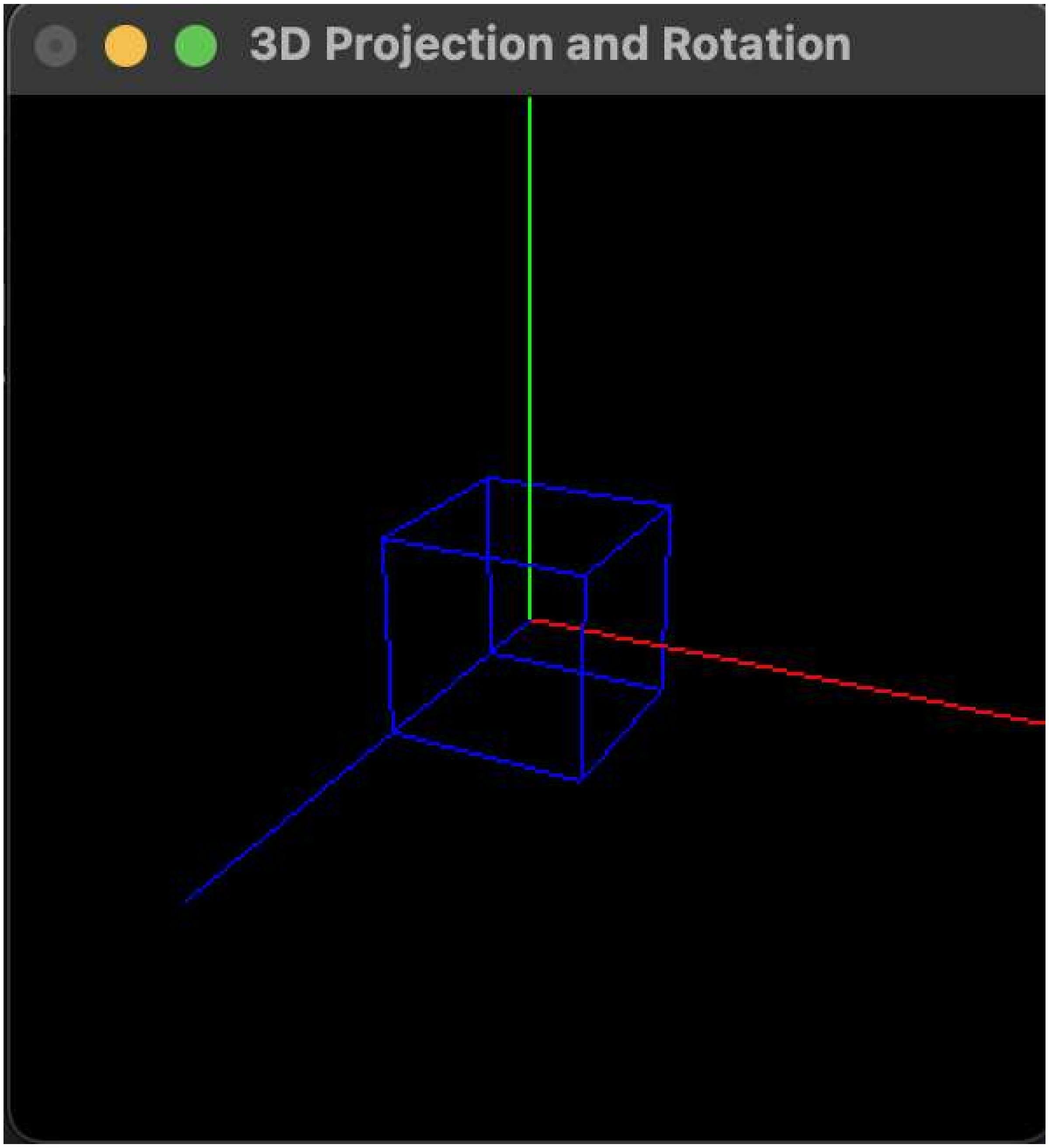
Use keyboard functions to rotate and show different views of the object. [Can use built-in functions for 3D transformations].

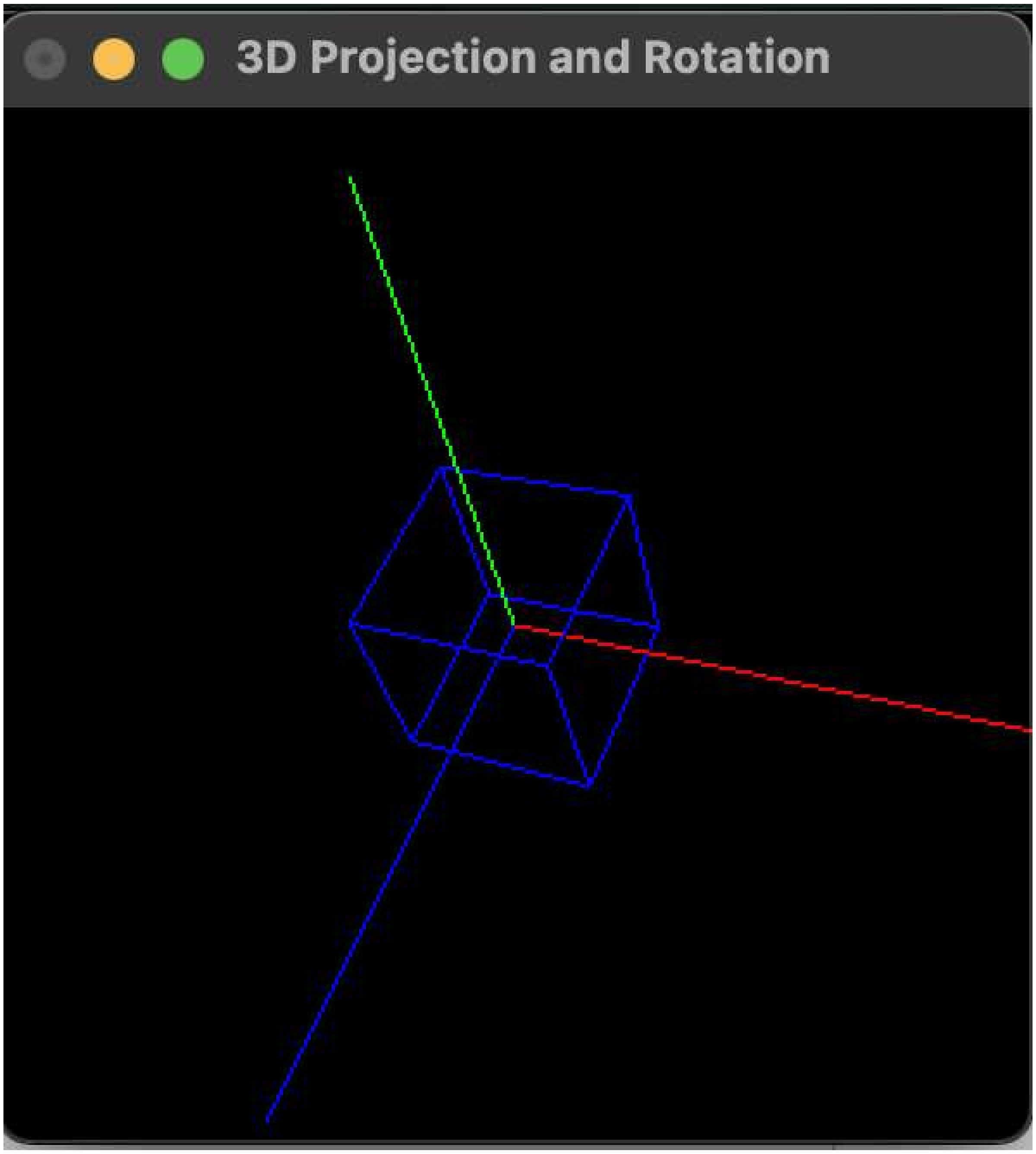
Code:

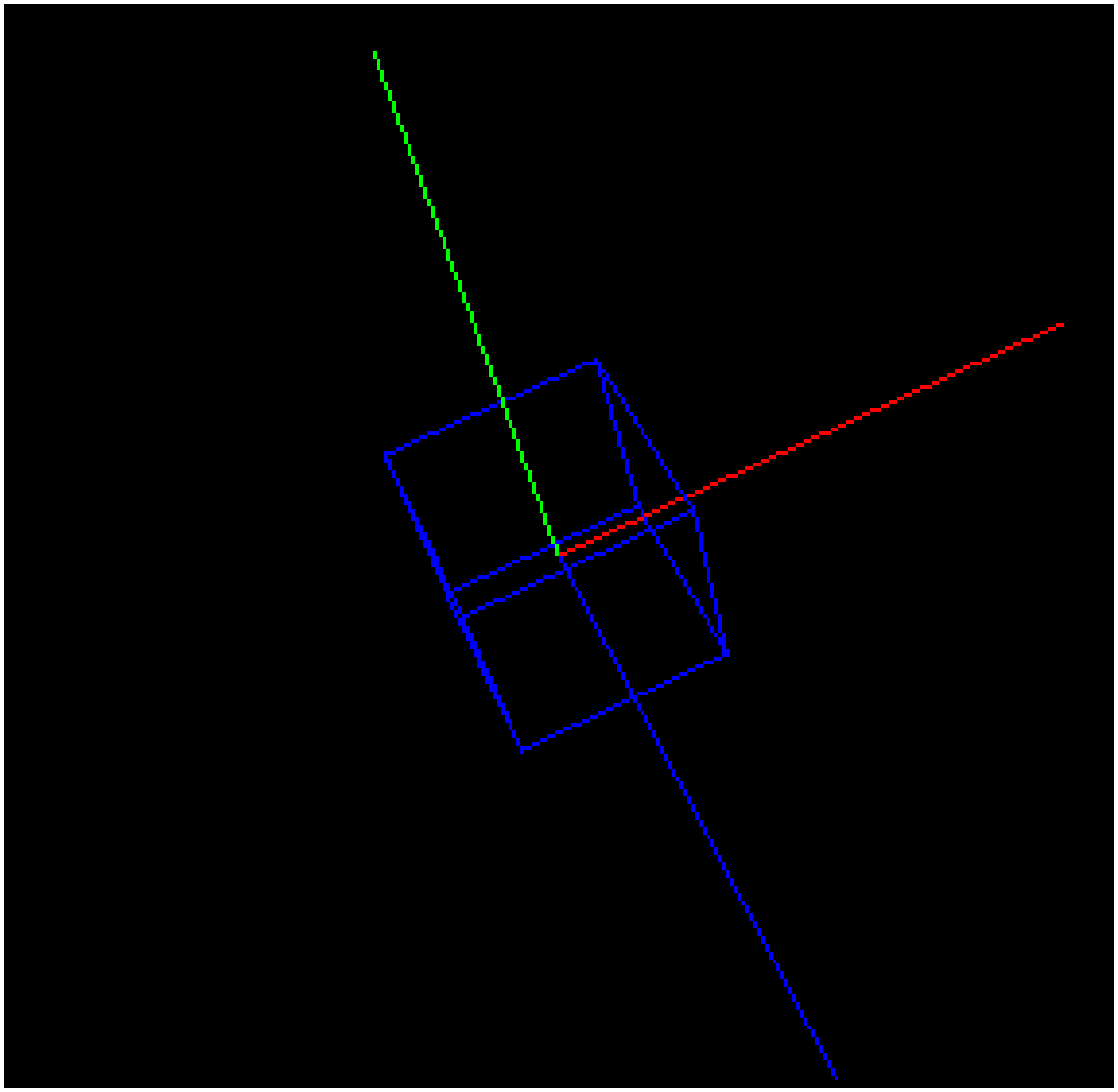
```
#include <GLUT/glut.h>
#include <iostream>
float angleX = 0.0, angleY = 0.0, angleZ = 0.0;
float cameraPosition[] = {5.0, 5.0, 10.0};
void drawAxes() {
    glColor3f(1.0, 0.0, 0.0); // Red X-axis
    glBegin(GL_LINES);
    glVertex3f(0.0, 0.0, 0.0);
    glVertex3f(5.0, 0.0, 0.0);
    glEnd();
    glColor3f(0.0, 1.0, 0.0); // Green Y-axis
    glBegin(GL_LINES);
    glVertex3f(0.0, 0.0, 0.0);
    glVertex3f(0.0, 5.0, 0.0);
    glEnd();
    glColor3f(0.0, 0.0, 1.0); // Blue Z-axis
    glBegin(GL_LINES);
    glVertex3f(0.0, 0.0, 0.0);
    glVertex3f(0.0, 0.0, 5.0);
    glEnd();
```

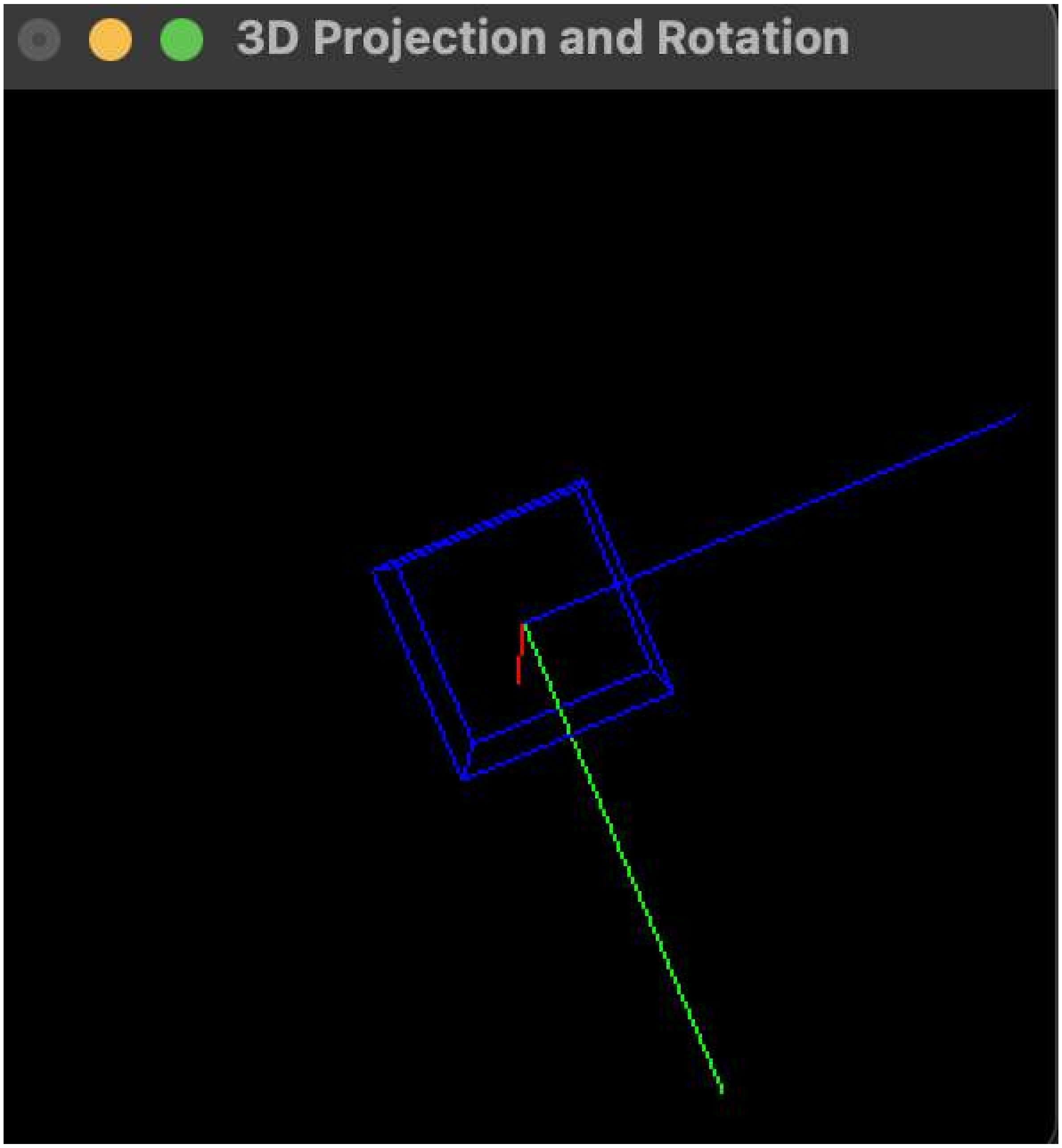
```
}
void drawCube() {
    glutWireCube(2.0); // You can replace this with your 3D
object drawing code
}
void display() {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    gluLookAt(cameraPosition[0], cameraPosition[1],
cameraPosition[2],
              0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
    glRotatef(angleX, 1.0, 0.0, 0.0);
    glRotatef(angleY, 0.0, 1.0, 0.0);
    glRotatef(angleZ, 0.0, 0.0, 1.0);
    drawAxes();
    drawCube();
    glutSwapBuffers();
}
void reshape(int width, int height) {
    glViewport(0, 0, width, height);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(45.0, (float)width / (float)height, 1.0,
100.0);
    glMatrixMode(GL_MODELVIEW);
}
void keyboard(unsigned char key, int x, int y) {
    switch (key) {
        case 'x':
            angleX += 5.0;
            break;
        case 'X':
            angleX -= 5.0;
            break;
        case 'y':
            angleY += 5.0;
            break;
        case 'Y':
            angleY -= 5.0;
            break;
        case 'z':
            angleZ += 5.0;
            break;
        case 'Z':
            angleZ -= 5.0;
            break;
    }
    glutPostRedisplay();
}
int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutCreateWindow("3D Projection and Rotation");
    glEnable(GL_DEPTH_TEST);
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutKeyboardFunc(keyboard);
    glutMainLoop();
    return 0;
}
```

Output:


3D Projection and Rotation


3D Projection and Rotation




3D Projection and Rotation

Name : Sabarivasan V

Class : CSE – B

RegNo : 205001085

EXERCISE - 10: CREATING A 3D SCENE IN C++ USING OPENGL

AIM:

Write a C++ program using Opengl to draw atleast four 3D objects. Apply lighting and texture and render the scene. Apply transformations to create a simple 3D animation. [Use built-in transformation functions]

ALGORITHM:

1. Set ammbience, diffuse, specular and shininess values of material and light source
2. Enable lighting so that the renderer can see light, turn LIGHT0 on, enable depth test for rendering depth and enable texture
3. Initialise frame buffer using glutInit()
4. Set display mode as single buffer for 2D graphics with RGB colour using glutInitDisplayMode()
5. Set output window size as 640, 640 pixels using glutWindowSize()
6. Create the output window using glutCreateWindow()
7. Call function to draw using glutDisplayFunc()
8. Set visibility of faces using depth test parameter using glEnable()
9. In the display function
   - 9.1. Set background colour (RGB, opacity) as white using glClear()
   - 9.2. Clear frame buffer using glClear()
   - 9.3. Set matrix mode to manipulate matrix values using glMatrixMode()
   - 9.4. Load identity matrix using glLoadIdentity()
   - 9.5. Set camera position
   - 9.6. Set background color and matrixmode
   - 9.7. Push matrix
   - 9.8. Enable and disable GL_TEXTURE_2D

    9.9. Add objects to the animation by pushing matrix, setting color and shininess using glMaterialfv()
    9.10. Set transformations
    9.11. Create the object
    9.12. Pop the matrix
    9.13. Repeat the same for the other
    9.14. End using glEnd()
    9.15. Flush frame buffer using glFlush()
10. Refresh the screen repeatedly while calling the function to draw using glutMainLoop()

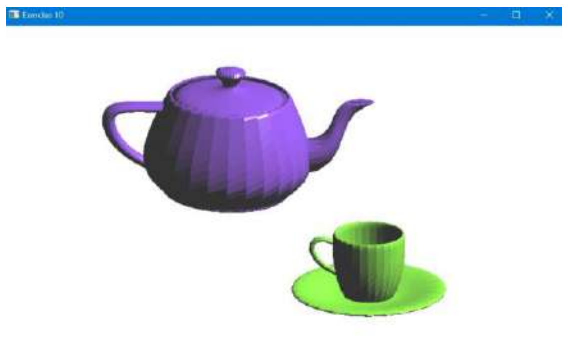
CODE:
```
#include <gl/freeglut.h>
#include <Windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <iostream>

GLfloat black[] = { 0.0, 0.0, 0.0, 1.0 };
GLfloat white[] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat direction[] = { 1.0, 1.0, 1.0, 0.0 };


float teapot_rotate = 0.2, teapot_rotate_direction = 1, teapot_posx =
-0.5, teapot_posy = 1.0, teapot_xplace = 0, teapot_yplace = 0; float
teaspoon_posx = 0.75, teaspoon_posy = 2.5, teaspoon_yplace = 0;
float sugar1_posx = 0.65, sugar1_posy = 2.5, sugar2_posx = 0.8,
sugar2_posy = 2.75, sugar1_yplace = 0, sugar2_yplace = 0; float
teacup_rotate = 0; #define red {0xff, 0x00, 0x00}
#define magenta {0xff, 0, 0xff}
GLubyte texture[][3] = {
    red, magenta,
    magenta, red,
};

void display() { glClear(GL_COLOR_BUFFER_BIT |

    GL_DEPTH_BUFFER_BIT); glMatrixMode(GL_MODELVIEW);

    glPushMatrix();
```

```
glEnable(GL_TEXTURE_2D);
glDisable(GL_TEXTURE_2D);

// Add a teacup to the scene.
glPushMatrix();
GLfloat teacup_color[] = { 0.482, 1, 0.161, 0.0 };
GLfloat teacup_mat_shininess[] = { 100 };
glMaterialfv(GL_FRONT, GL_DIFFUSE, teacup_color);
glMaterialfv(GL_FRONT, GL_SHININESS,
teacup_mat_shininess); glTranslatef(0.75, -0.25, 0.0);
glRotatef(teacup_rotate, 0, 1, 0); glutSolidTeacup(1.0);
glPopMatrix();

// Add a teapot to the scene.
glPushMatrix();
GLfloat teapot_color[] = { 0.486, 0.212, 0.871, 0.0 };
GLfloat teapot_mat_shininess[] = { 100 };
glMaterialfv(GL_FRONT, GL_DIFFUSE, teapot_color);
glMaterialfv(GL_FRONT, GL_SHININESS,
teapot_mat_shininess); glTranslatef(teapot_posx,
teapot_posy, 0.0); glRotatef(teapot_rotate, 0, 0, 1);
glutSolidTeapot(0.75); glPopMatrix();

// Add a sugar cubes to the scene.
GLfloat sugar_color[] = { 1, 1, 1, 0.0 };
GLfloat sugar_mat_shininess[] = { 50 };

glPushMatrix(); glMaterialfv(GL_FRONT, GL_DIFFUSE,
sugar_color); glMaterialfv(GL_FRONT, GL_SHININESS,
sugar_mat_shininess); glTranslatef(sugar1_posx,
sugar1_posy, 0.0); glRotatef(-45.0, 0, 0, 1);
glutSolidCube(0.1); glPopMatrix();

glPushMatrix(); glMaterialfv(GL_FRONT, GL_DIFFUSE,
sugar_color); glMaterialfv(GL_FRONT, GL_SHININESS,
sugar_mat_shininess); glTranslatef(sugar2_posx,
sugar2_posy, 0.0); glRotatef(45.0, 0, 0, 1);
glutSolidCube(0.1); glPopMatrix();

// Add a teaspoon to the scene.
```

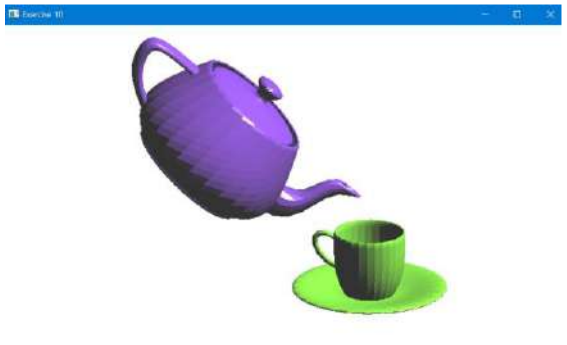
```
    glPushMatrix();
    GLfloat teaspoon_color[] = { 0.2, 0.2, 0.2, 0.0 }; GLfloat
    teaspoon_mat_shininess[] = { 100 }; glMaterialfv(GL_FRONT,
    GL_DIFFUSE, teaspoon_color); glMaterialfv(GL_FRONT,
    GL_SHININESS, teaspoon_mat_shininess);
    glTranslatef(teaspoon_posx, teaspoon_posy, 0.0);
    glRotatef(135, 0, 1, 0); glRotatef(-60, 1, 0, 0);
    glutSolidTeaspoon(1.25); glPopMatrix();


    if (teapot_rotate_direction == 1 && teapot_rotate > -45.0)
teapot_rotate -= 0.5; if (teapot_rotate_direction == 1 &&
    teapot_rotate <= -45.0)
teapot_rotate_direction = -1; if (teapot_rotate_direction
    == -1 && teapot_rotate < 0)
teapot_rotate += 0.5; if (teapot_rotate_direction == -1 &&
    teapot_rotate >= 0)
teapot_rotate_direction = 0;

    teacup_rotate -= 0.2;

    if (teapot_rotate_direction == 0) { if (teapot_posx > -1.25 &&
        teapot_xplace == 0) teapot_posx -=
0.05; if (teapot_posx <= -1.25) teapot_xplace = 1;
        if (teapot_posy > 0 && teapot_yplace == 0) teapot_posy -=
0.05; if (teapot_posy <= -1) teapot_yplace = 1;
    }

    if (teapot_rotate_direction == 0) { if (sugar1_posy > -0.5 &&
        sugar1_yplace == 0) sugar1_posy -=
0.05; if (sugar1_posy <= -0.5) sugar1_yplace = 1; if (sugar2_posy >
        -0.5 && sugar2_yplace == 0) sugar2_posy -=
0.05; if (sugar2_posy <= -0.5) sugar2_yplace = 1; }
    if (sugar1_yplace == 1 && sugar2_yplace == 1) { if
        (teaspoon_posy > -0.25 && teaspoon_yplace == 0)
teaspoon_posy -= 0.05; //-0.5 if (teaspoon_posy <= -
        0.5) teaspoon_yplace = 1;
    }
    glutSwapBuffers();
}

void reshape(GLint w, GLint h) {
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    GLfloat aspect = GLfloat(w) / GLfloat(h); glLoadIdentity();
    glOrtho(-2.5, 2.5, -2.5 / aspect, 2.5 / aspect, -10.0,
    10.0);
}

void init() { glClearColor(1,
    1, 1, 1);

    glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, white);
    glMaterialfv(GL_FRONT, GL_SPECULAR, white);
    glMaterialf(GL_FRONT, GL_SHININESS, 30);

    glLightfv(GL_LIGHT0, GL_AMBIENT, black);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, white);
    glLightfv(GL_LIGHT0, GL_SPECULAR, white);
    glLightfv(GL_LIGHT0, GL_POSITION, direction);
    glEnable(GL_LIGHTING);                // so the renderer considers
light
    glEnable(GL_LIGHT0);                  // turn LIGHT0 on
    glEnable(GL_DEPTH_TEST);              // so the renderer considers
depth
    glShadeModel(GL_FLAT);

    glEnable(GL_TEXTURE_2D);
    glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
    glTexImage2D(GL_TEXTURE_2D,
        0,                    // level 0
        3,                    // use only R, G, and B components
        2, 2,                 // texture has 2x2 texels
        0,                    // no border
        GL_RGB,               // texels are in RGB format
        GL_UNSIGNED_BYTE,  // color components are unsigned bytes
    texture); glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
    GL_NEAREST); glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
    GL_NEAREST); glRotatef(20.0, 1.0, 0.0, 0.0);
}

void sceneDemo(int v)
{ glutPostRedisplay(); glutTimerFunc(1000 /
    24, sceneDemo, 0);
}

// The usual application statup code. int main(int argc,
char** argv) { glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
glutInitWindowPosition(80, 80); glutInitWindowSize(800,
600); glutCreateWindow("Exercise 10");
glutReshapeFunc(reshape); glutDisplayFunc(display);
glutTimerFunc(1000, sceneDemo, 0); init(); glutMainLoop();
}
```

OUTPUT:

Frame 1

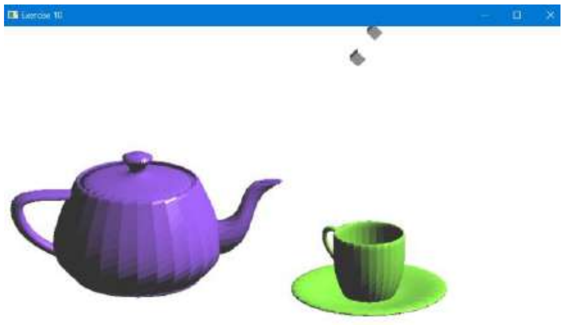


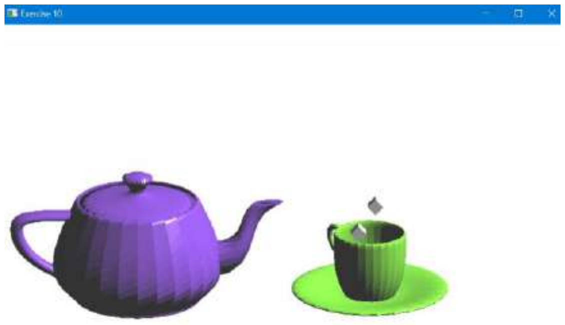
Frame 2



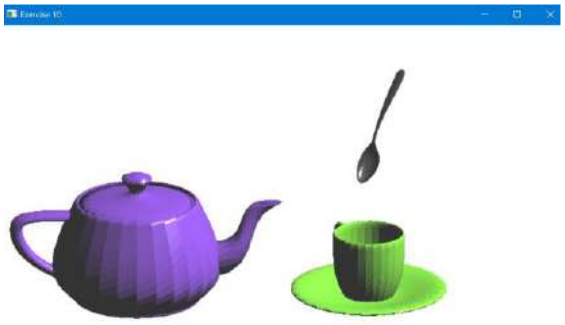
Frame 3

Frame 4

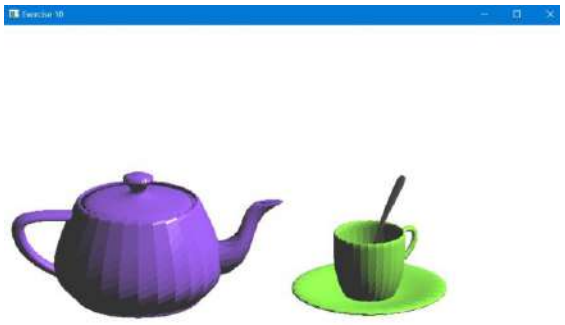




Frame 5



Frame 6



LEARNING OUTCOMES:

1. Learnt to adjust the parameters of the frames
2. Learnt to plot points and mark coordinates
3. Learnt to plot points in 3-Dimensions
4. Learnt to animate object in 3D
5. Learnt to add interactions between object

Name : Sabarivasan V - 205001085
Class : CSE – B

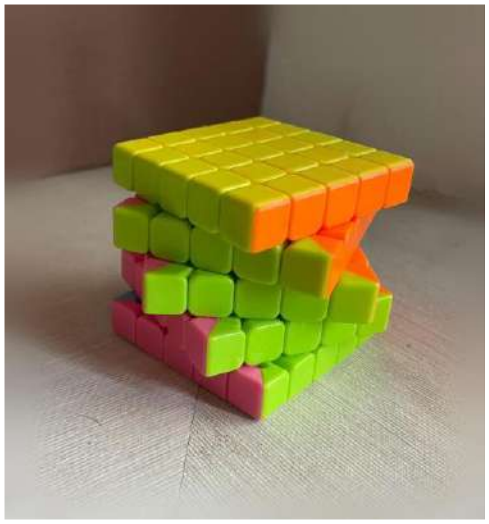
EXERCISE - 11: IMAGE EDITING AND ANIMATION

AIM:

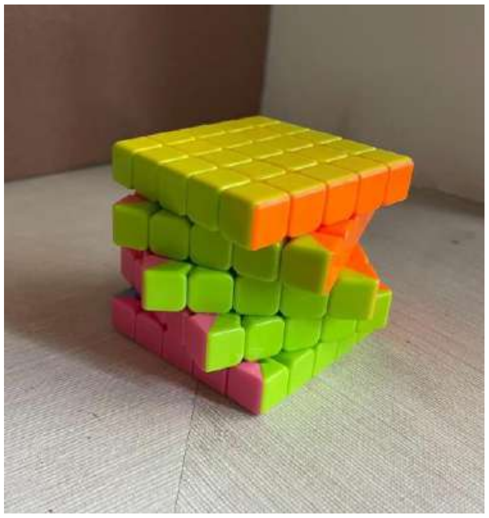
a) Using GIMP, include an image and apply filters, noise and masks.

b) Using GIMP, create a GIF animated image.

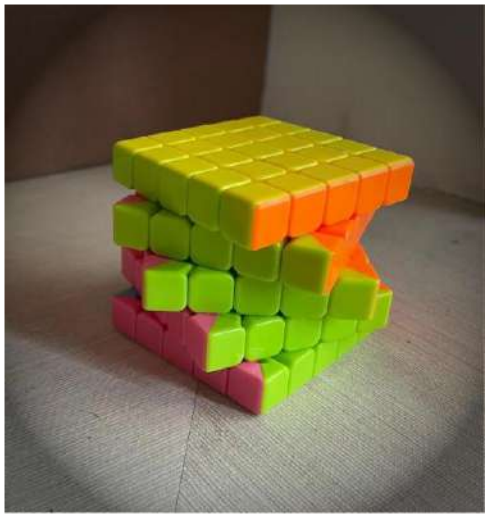
a) INPUT IMAGE: GAUSSIAN BLUR:



ENHANCED CONTRAST:

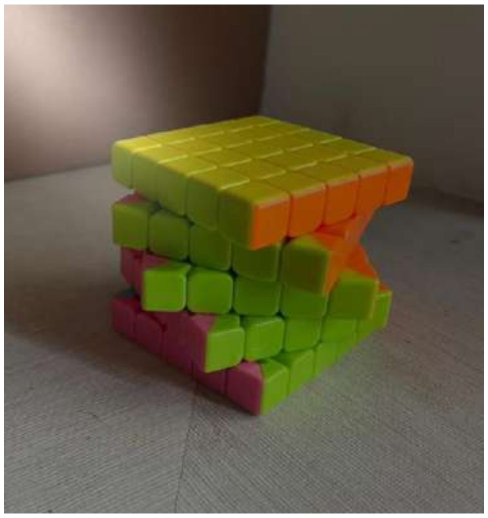


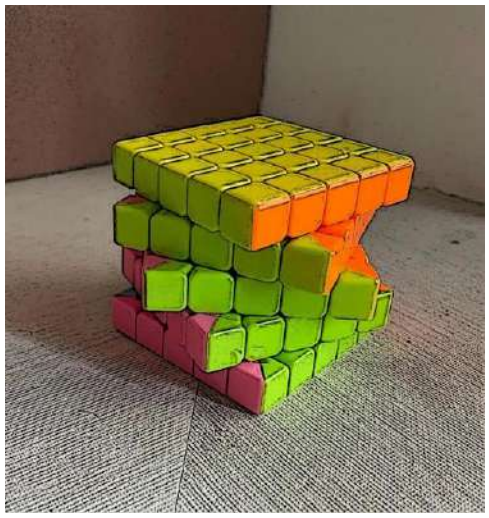
VIGNETTE:



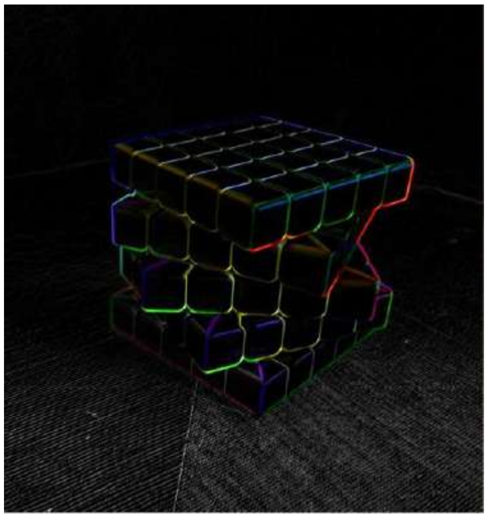
LIGHTING:

CARTOON:

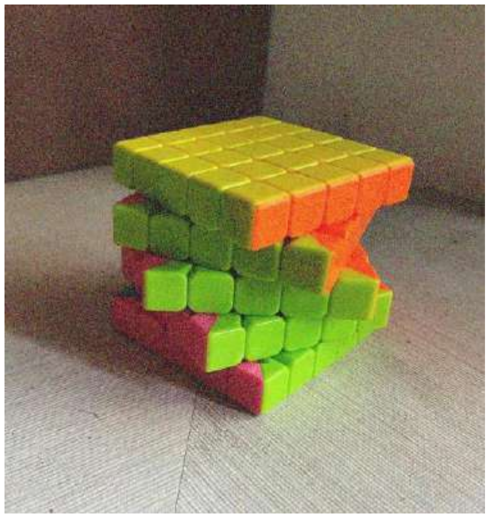




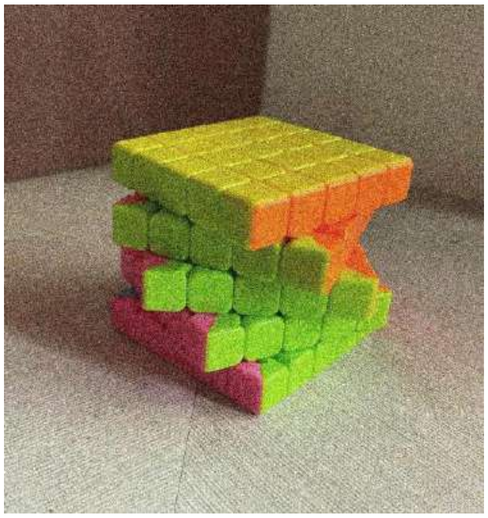
NEON EDGE DETECTION:



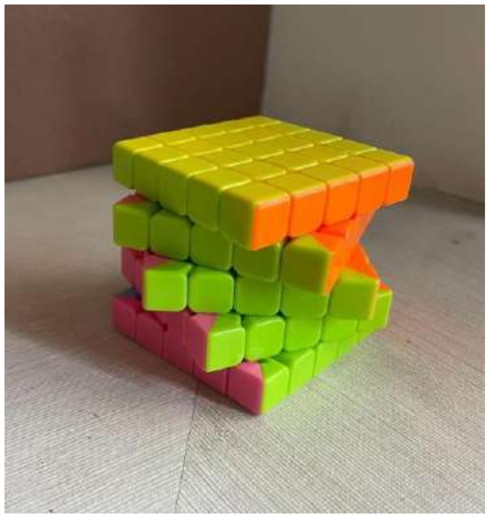
RGB NOISE:



CIE ICH NOISE:



WHITE MASK:

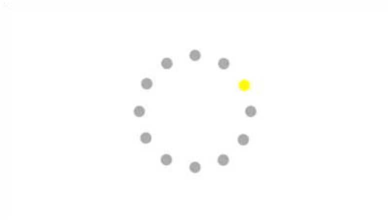


b) ANIMATION - FRAMES

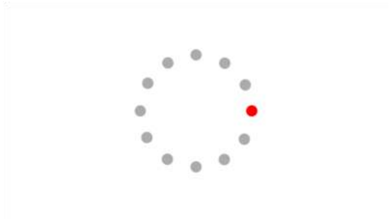
FRAME 1

FRAME 2

FRAME 3



FRAME 4



FRAME 5

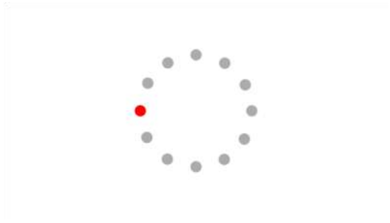
FRAME 6

FRAME 7

FRAME 8

FRAME 9

FRAME 10



FRAME 11

FRAME 12

LEARNING OUTCOMES:

1. Learnt the various tools in GIMP
2. Learnt to apply various filters
3. Learnt to add noise to images
4. Learnt to add masks on images
5. Learnt to create a GIF animated image

Name : Sabarivasan V – 205001085

Class : CSE – B

EXERCISE - 12: CREATING 2D ANIMATION

AIM:

Using GIMP, include layers and create a simple animation of your choice.

BALL BOUNCING ANIMATION - FRAMES:

FRAME 1

FRAME 2

FRAME 3

FRAME 4

FRAME 5

FRAME 6

FRAME 7

FRAME 8

FRAME 9

FRAME 10

FRAME 11

FRAME 12

LEARNING OUTCOMES:

1. Learnt the various tools in GIMP
2. Learnt to apply various filters
3. Learnt to add noise to images
4. Learnt to add masks on images
5. Learnt to create a GIF animated image

Graphics and Multimedia Lab Mini Project

Sabarivasan V - 205001085

Shajith Hameed - 205001097

Blender:

Blender is a powerful and versatile open-source 3D computer graphics software that is used for creating animated films, visual effects, 3D games, and more. Developed by the Blender Foundation, Blender boasts a comprehensive suite of tools for modeling, sculpting, texturing, lighting, rendering, rigging, animating, and even video editing. It supports a wide range of file formats and is compatible with various operating systems, including Windows, macOS, and Linux.

Here's a brief overview of some key features and aspects of Blender:

1. **User Interface (UI):** Blender has a unique and customizable interface. The UI includes panels, editors, and various workspace layouts that cater to different aspects of 3D content creation.

2. **Modeling:** Blender provides a range of modeling tools for creating 3D objects, whether through traditional mesh-based modeling or more organic sculpting techniques.

3. **Texturing and Materials:** Artists can apply textures and materials to objects, giving them realistic surfaces. Blender supports a variety of texture types, including images, procedural textures, and more.

4. **Animation:** Blender excels in animation, allowing users to create complex character animations, simulate physics, and even handle particles and fluids. The timeline and graph editor are essential components for animators.

5. **Rigging:** Rigging involves creating a skeleton structure for 3D models, enabling them to move realistically. Blender's armature system makes rigging relatively straightforward.

6. **Lighting and Rendering:** Blender has a powerful rendering engine called Cycles, capable of producing high-quality images. It also includes Eevee, a real-time rendering engine suitable for interactive previews and faster rendering.

7. Compositing and Video Editing: Blender features a node-based compositor for post-processing effects and adjustments. Additionally, it includes a video editor for assembling and editing video clips.

8. Add-ons and Python Scripting: Blender's functionality can be extended through add-ons, and it supports Python scripting for users who want to automate tasks or create custom tools.

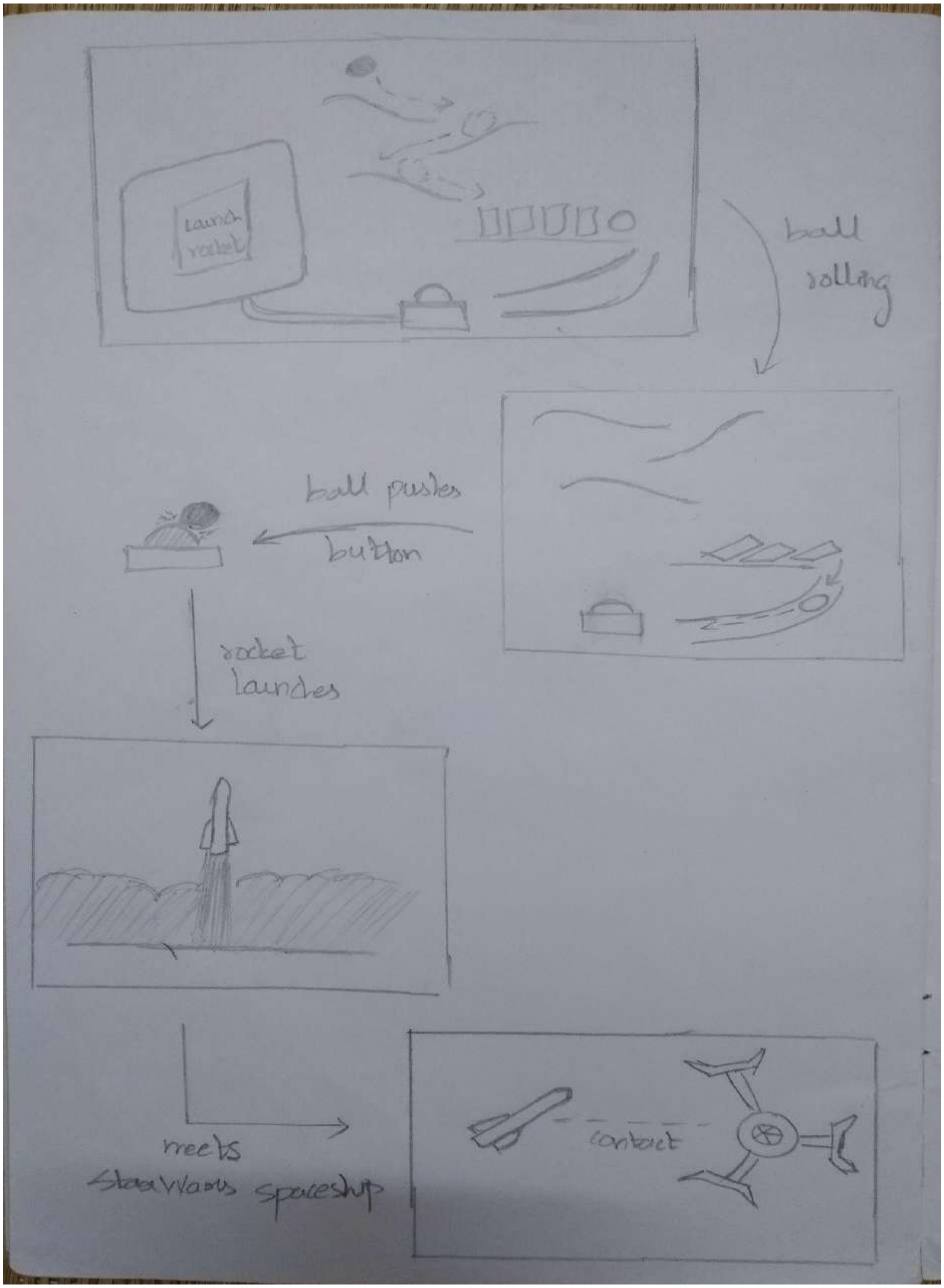
9. Community and Support: Blender has a vibrant and active community. Users can find tutorials, documentation, and support through forums, websites, and social media.

Blender's open-source nature encourages collaboration and continual improvement. It has gained popularity not only for its robust feature set but also because it's free and open to everyone. Whether you're a beginner or an experienced 3D artist, Blender provides a powerful platform for bringing your creative visions to life.

Description:

It's about a Mom(Alice) and her son going to a rocket launch as Alice is the chief engineer responsible for the launch . Alice’s bored son starts building something found in his backpack and her mom’s desk. When Alice’s son was showing how his new creation worked it accidentally triggers the launch button which leads to an early launch.
The launch goes smoothly fortunately , the shuttle detaches its extra parts to venture into deep space.
Three hours pass , the astronaut spots something outside his shuttle and to his surprise it’s an alien spaceship , the astronaut looks in shock , The end.

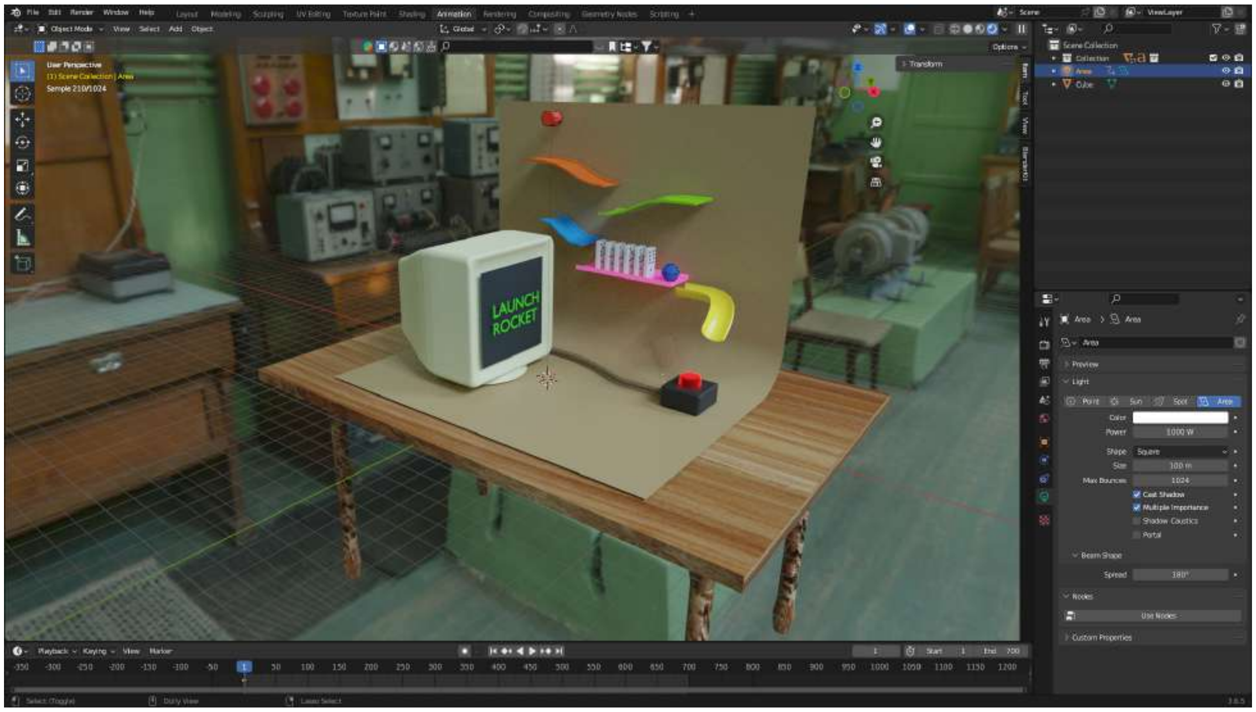
Storyboard Layout :



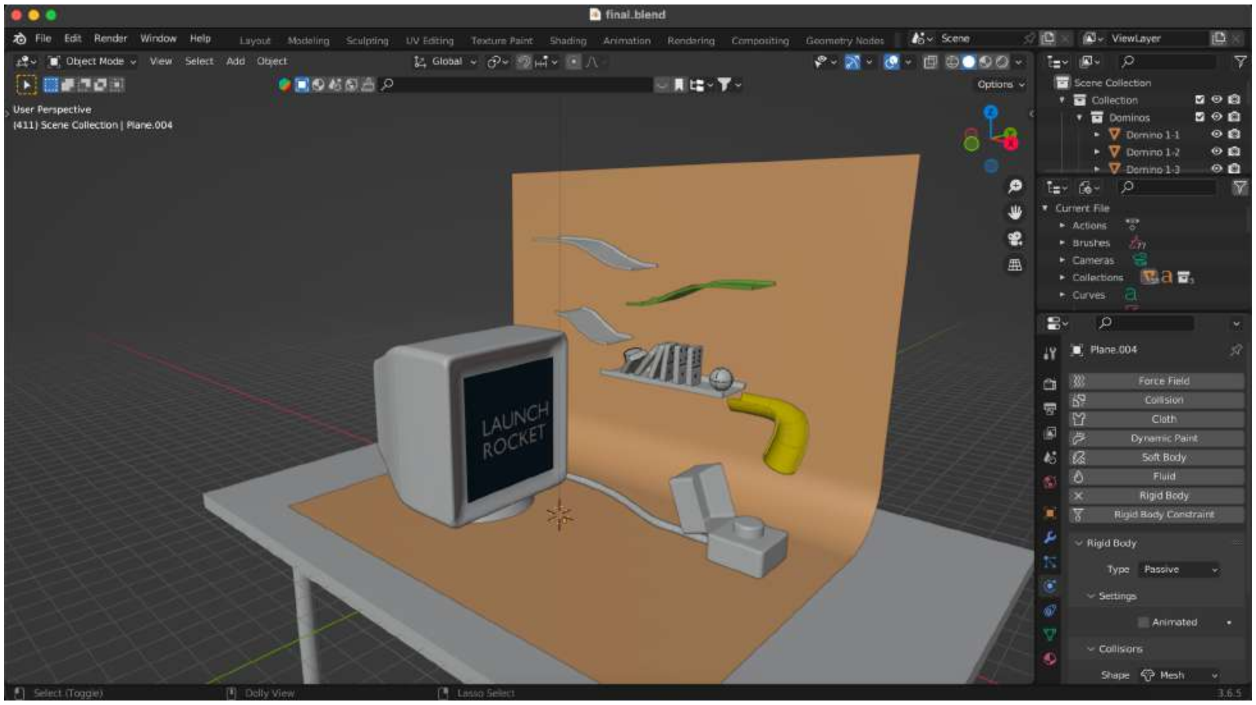
Object Description:

1.Cylinder - a red cylinder
2.Ramps - ramps where the cylinder and the ball rolls
3.Dominos - Cuboid shaped Dominos
4.Ball - Spherical shaped ball
5.Computer - A reformed cube shape in the shape of a old computer
6.Button - A combination of a red cylinder and a cuboid
7.Rocket - A combination of cylinders , cones with textures
8.Spaceship - A combination of spheres,planes and circles

Screenshots :



Color
Power 1000 W
Shape Square
Size 100 m
Max Bounces 1024
[✓] Cast Shadow
[✓] Multiple Importance
[ ] Shadow Caustics
[ ] Portal
Spread 180°

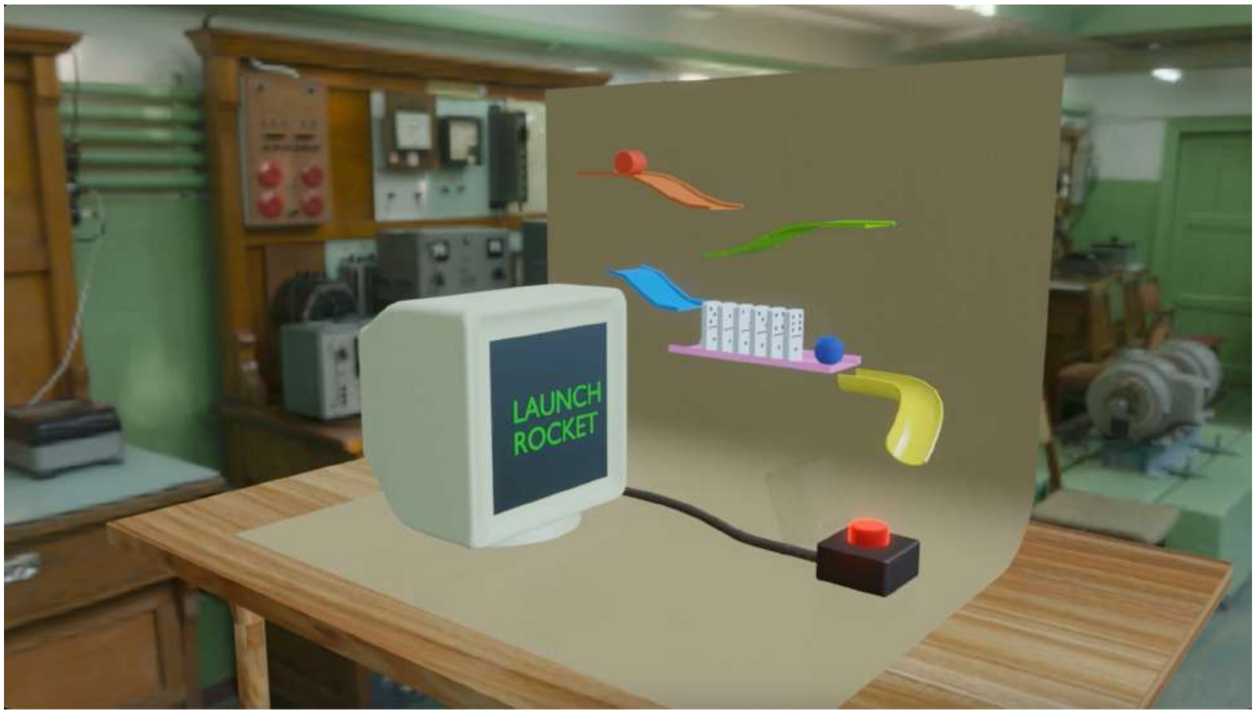


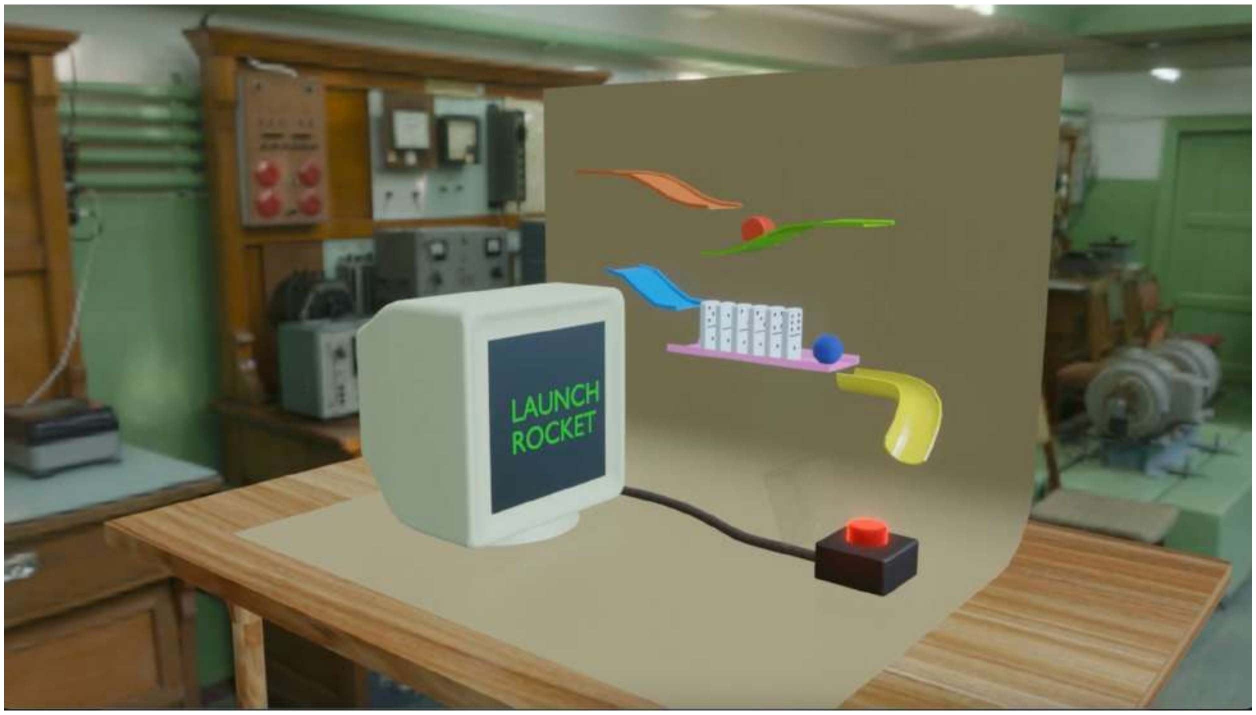
Rigid Body

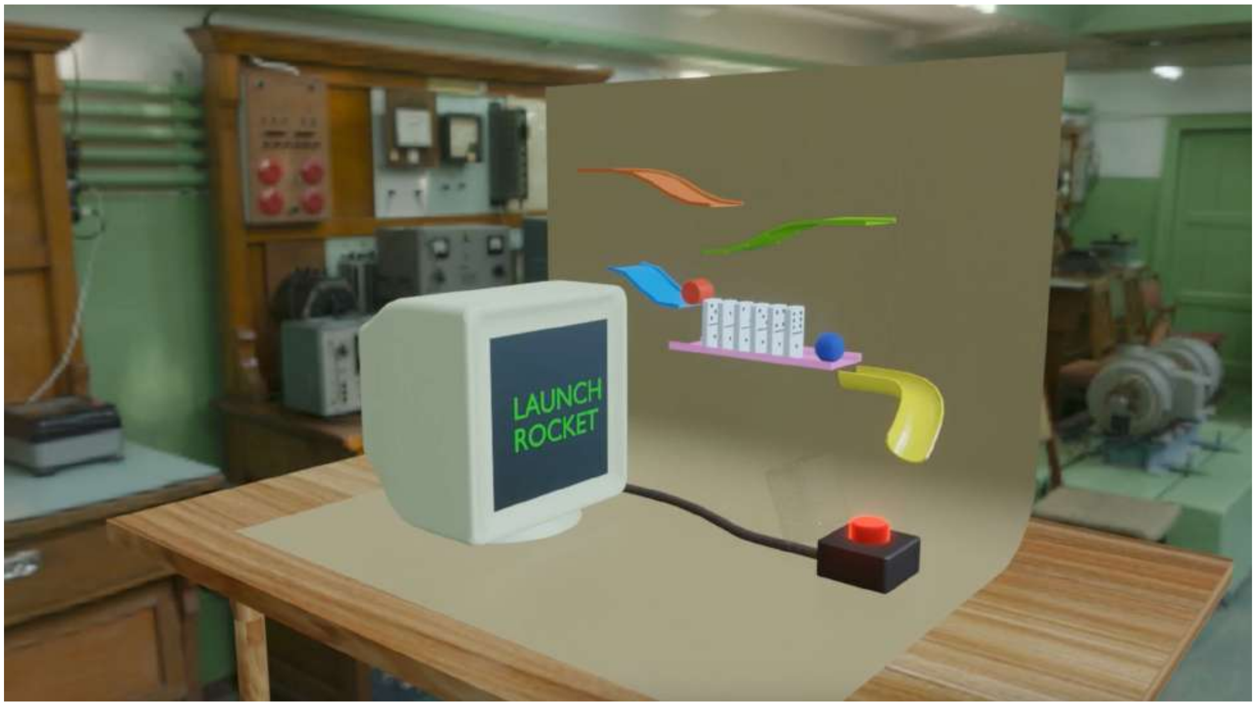
Type: Passive

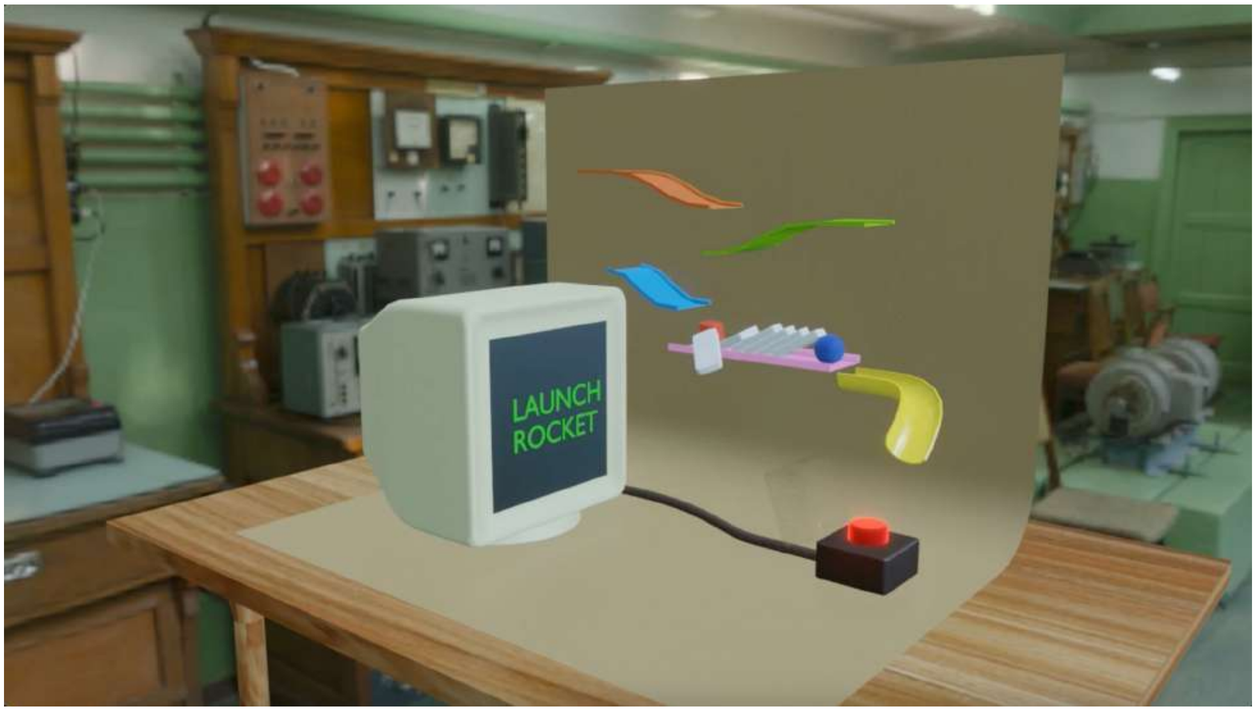
Animated

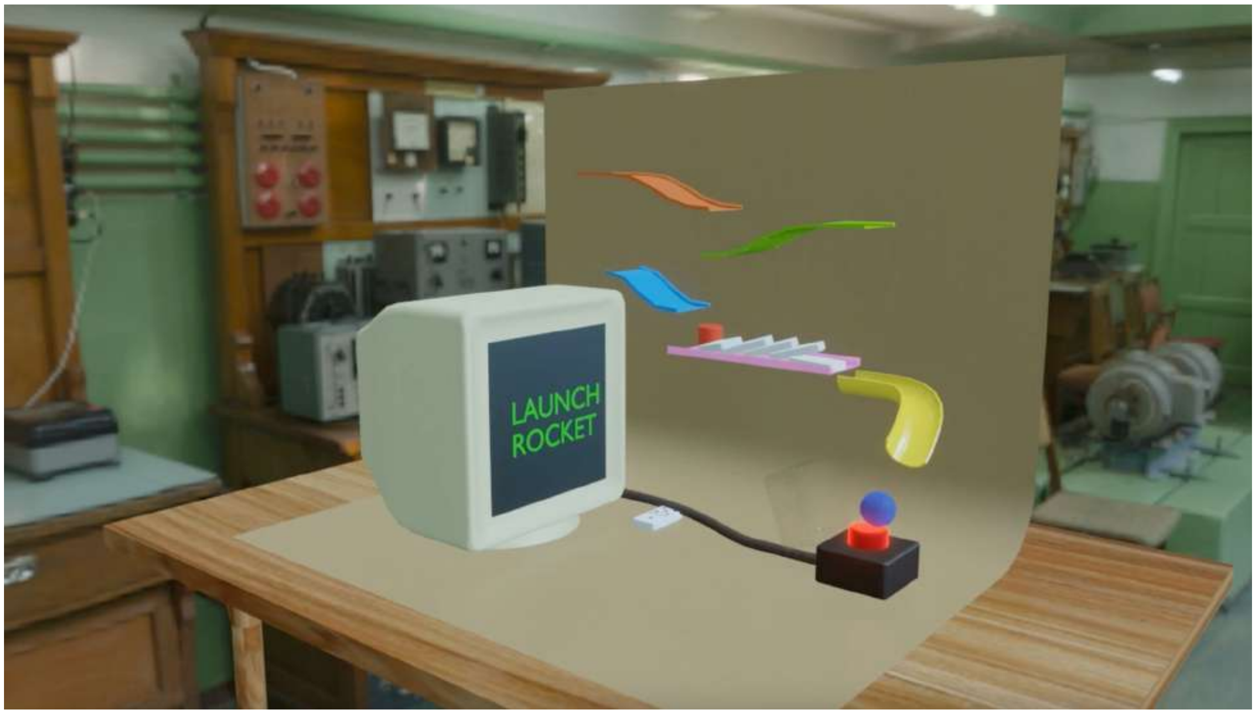
Shape: Mesh

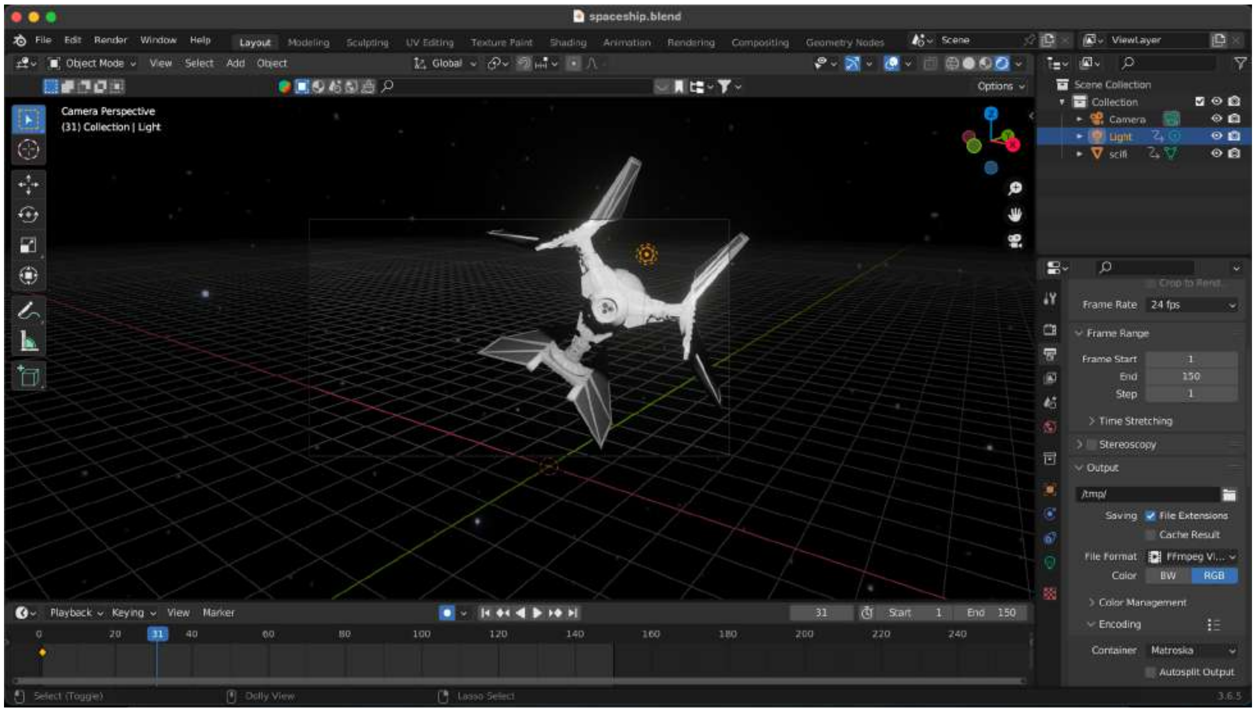



LAUNCH
ROCKET


LAUNCH
ROCKET


LAUNCH
ROCKET


LAUNCH
ROCKET


spaceship.blend
File Edit Render Window Help
Layout Modeling Sculpting UV Editing Texture Paint Shading Animation Rendering Compositing Geometry Nodes
Scene
ViewLayer
Object Mode View Select Add Object
Global
Options
Camera Perspective
(31) Collection | Light
Scene Collection
Collection
Camera
Light
scifi
Playback Keying View Marker
Select (Toggle)
Dolly View
Lasso Select
3.6.5

Frame Rate 24 fps

Frame Range
Frame Start 1
End 150
Step 1

Time Stretching
Stereoscopy

Output
/tmp/
Saving ✓ File Extensions
Cache Result
File Format Ffmpeg Vi...
Color BW [✓ RGB]
Color Management
Encoding
Container Matroska
Autosplit Output

31 Start 1 End 150

Type of Interpolation:

Linear , Bezier Curve are the interpolation methods used in this project

Learning Outcomes:

1. Learned to Create a Story with a Storyboard.
2. Learned to use Blender and video editing tools like Davinci resolve 18.
3. Learned to render 3D animations with textures , lighting and physics.