SSN COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UCS1712 – GRAPHICS AND MULTIMEDIA LAB

EX NO: 1 – Study of Basic output primitives in OpenGL

Name : **Lakshmi Priya B**
Register Number : **185001083**
Date : **22-07-2021**

1.

Aim

To create a window using OPENGL and to draw the following basic output primitives:

- POINTS
- LINES
- LINE_STRIP
- LINE_LOOP
- TRIANGLES
- TRIANGLE STRIP
- TRIANGLE FAN
- QUADS
- QUAD_STRIP
- POLYGON

Algorithm

1. Initialize using the glutInit() function with &argc and argv as parameters
2. Set the display mode as GLUT_SINGLE and specify the color scheme as GLUT_RGB
3. Specify the window size as 840, 680
4. Create a new window and set the title as “Output Primitives”
5. Define myInit() function:
    a. Set the background color as yellow
    b. Set the color of objects
    c. Set the size of point
    d. Set glMatrixMode(GL_PROJECTION)
    e. Specify the display area
6. Define the myDisplay() function:
    a. Clear the screen buffer
    b. Use glColor3f(), glBegin(), glVertex2d(), glEnd() to construct the shapes
    c. Draw two points using GL_POINTS
    d. Draw two lines using GL_LINES
    e. Draw a line strip using GL_LINE_STRIP
    f. Draw a line loop using GL_LINE_LOOP
    g. Draw triangle using GL_TRIANGLES
    h. Draw a triangle strip using GL_TRIANGLE_STRIP

i. Draw a triangle fan using GL_TRIANGLE_FAN
j. Draw quadrilateral using GL_QUADS
k. Draw a quadrilateral strip using GL_QUAD_STRIP
l. Draw a polygon using GL_POLYGON
m. Send all the output to display using glFlush()

7. Call glutDisplayFunc(myDisplay)
8. Call myInit() for additional initializations
9. Go into a loop using glutMainLoop() until event occurs
10. Run the application and view the output

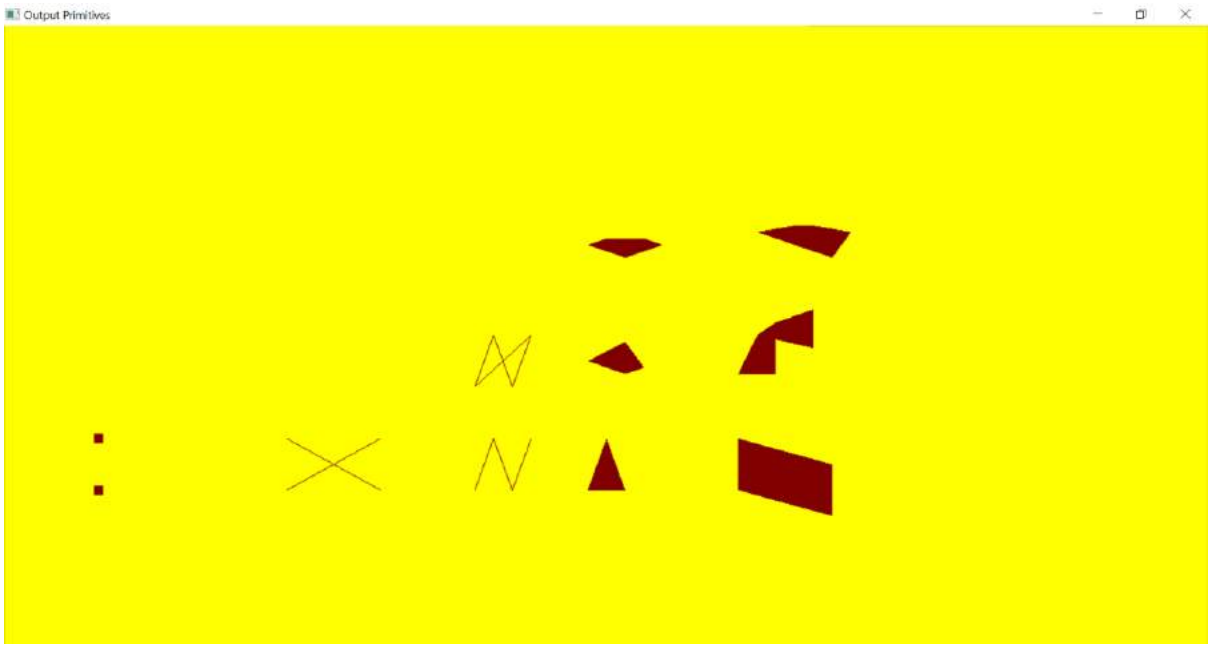
Code

Source.cpp:

```
#include<GL/glut.h>
void myInit()
{
	//background color red + blue = yellow
	glClearColor(1.0, 1.0, 0, 0.2);
	//setting color of objects = dark red
	glColor3f(0.5f, 0.0f, 0.0f);
	//size of point
	glPointSize(10);
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	gluOrtho2D(0.0, 640.0, 0.0, 480.0);
}
void myDisplay()
{
	glClear(GL_COLOR_BUFFER_BIT);

	glBegin(GL_POINTS);
	glVertex2d(50, 120);
	glVertex2d(50, 160);
	glEnd();

	glBegin(GL_LINES);
	glVertex2d(150, 120);
	glVertex2d(200, 160);

	glVertex2d(200, 120);
	glVertex2d(150, 160);
	glEnd();

	glBegin(GL_LINE_STRIP);
	glVertex2d(250, 120);
	glVertex2d(260, 160);
	glVertex2d(270, 120);
	glVertex2d(280, 160);
	glEnd();

	glBegin(GL_LINE_LOOP);
	glVertex2d(250, 200);
	glVertex2d(260, 240);
	glVertex2d(270, 200);
	glVertex2d(280, 240);
	glEnd();
```

```
        glBegin(GL_TRIANGLES);
        glVertex2d(310, 120);
        glVertex2d(330, 120);
        glVertex2d(320, 160);
        glEnd();

        glBegin(GL_TRIANGLE_STRIP);
        glVertex2f(330, 210);
        glVertex2f(340, 215);
        glVertex2f(310, 220);
        glVertex2f(330, 235);
        glEnd();

        glBegin(GL_TRIANGLE_FAN);
        glVertex2d(330, 300);
        glVertex2d(310, 310);
        glVertex2d(320, 315);
        glVertex2d(340, 315);
        glVertex2d(350, 310);
        glEnd();

        glBegin(GL_QUADS);
        glVertex2d(390, 120);
        glVertex2d(390, 160);
        glVertex2d(440, 140);
        glVertex2d(440, 100);
        glEnd();

        glBegin(GL_QUAD_STRIP);
        glVertex2d(390, 210);
        glVertex2d(410, 210);
        glVertex2d(400, 240);
        glVertex2d(410, 250);
        glVertex2d(430, 230);
        glVertex2d(430, 260);
        glEnd();

        glBegin(GL_POLYGON);
        glVertex2d(440, 300);
        glVertex2d(440, 310);
        glVertex2d(450, 320);
        glVertex2d(430, 325);
        glVertex2d(420, 325);
        glVertex2d(400, 320);
        glEnd();

        glFlush();
}

int main(int argc, char* argv[])
{
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
        glutInitWindowSize(840, 680);
        glutCreateWindow("Output Primitives");
        glutDisplayFunc(myDisplay);
        myInit();
        glutMainLoop();
        return 1;
}
```

Output Screenshot



Result

Thus, a window has been created using OPENGL and to draw the basic output primitives like POINTS, LINES, LINE_STRIP, LINE_LOOP, TRIANGLES, TRIANGLE STRIP, TRIANGLE FAN, QUADS, QUAD_STRIP, POLYGON.

2.

Aim

To create a window and draw a simple House using OpenGL.

Algorithm

1. Initialize using the glutInit() function with &argc and argv as parameters
2. Set the display mode as GLUT_SINGLE and specify the color scheme as GLUT_RGB
3. Specify the window size as 1200, 740
4. Set the starting position for the window as 0, 0
5. Create a new window and set the title as “House”
6. Define myInit() function:
    a. Clear all the screen color
    b. Set glMatrixMode(GL_PROJECTION)
    c. Specify the display area

7. Define the myDisplay() function:
    a. Clear the screen buffer
    b. Use glColor3f(), glBegin(), glVertex2i(), glEnd() to construct the shapes
    c. Draw green lawn and rectangular part of the house using GL_QUADS
    d. Draw rectangular ouline of house using GL_LINES after setting glLineWidth as 10
    e. Draw door using GL_POLYGON and door outline using GL_LINE_STRIP
    f. Draw left and right windows using GL_POLYGON
    g. Draw left and right windows outline using GL_LINE_LOOP
    h. Draw left and right window cross bars using GL_LINES
    i. Draw door knob using GL_POINTS after increasing point size to 15 using glPointSize()
    j. Draw foot steps using GL_POLYGON
    k. Draw the house's roof using GL_TRIANGLES
    l. Draw a center decoration to the roof using GL_TRIANGLE_FAN
    m. Draw smoke outlet from chimney using GL_TRIANGLE STRIP
    n. Draw chimney's outlet and body using GL_QUADS
    o. Draw chimney's outlet outline using GL_LINE_LOOP
    p. Dreaw chimney's body ouline using GL_LINES
    q. Draw top triangle outline using GL_LINE_LOOP
    r. Draw foot steps oultine using GL_LINE_STRIP and GL_LINES
    s. Draw a green pathway using GL_QUAD_STRIP
    t. Send all the output to display using glFlush()
8. Call glutDisplayFunc(myDisplay)
9. Call myInit() for additional initializations
10. Go into a loop using glutMainLoop() until event occurs
11. Run the application and view the output

Code

Source.cpp

```
#include <GL\glut.h>

// Function to initialize the drivers
void myInit(void)
{
	// Clear all the screen color
	glClearColor(0.5, 0.8, 0.9, 1.0);

	glMatrixMode(GL_PROJECTION);

	glLoadIdentity();

	// Specify the display area
	gluOrtho2D(0.0, 400.0, 0.0, 400.0);
}

// Function to display the House

void myDisplay(void)
{
	// Clear the screen buffer
	glClear(GL_COLOR_BUFFER_BIT);
```

```
// Lawn
glColor3f(0.36f, 0.83f, 0.28f);
glBegin(GL_QUADS);
glVertex2i(0, 0);
glVertex2i(0, 100);
glVertex2i(400, 100);
glVertex2i(400, 0);
glEnd();

// Rectangular part of house
glColor3f(0.60f, 0.42f, 0.16f);
glBegin(GL_QUADS);
glVertex2i(125, 250);
glVertex2i(125, 100);
glVertex2i(275, 100);
glVertex2i(275, 250);
glEnd();

// Rectangular inner part of house
glColor3f(0.96f, 0.74f, 0.44f);
glBegin(GL_QUADS);
glVertex2i(130, 242);
glVertex2i(130, 107);
glVertex2i(270, 107);
glVertex2i(270, 242);
glEnd();

// Rectangular outline of house
glColor3f(0.0f, 0.0f, 0.0f);
glLineWidth(10);
glBegin(GL_LINES);
glVertex2i(125, 250);
glVertex2i(125, 95);

glVertex2i(125, 98);
glVertex2i(275, 98);

glVertex2i(275, 95);
glVertex2i(275, 250);
glEnd();

// Door
glColor3f(0.396f, 0.24f, 0.016f);
glBegin(GL_POLYGON);
glVertex2i(180, 120);
glVertex2i(180, 200);
//glVertex2i(195, 210);
glVertex2i(220, 200);
glVertex2i(220, 120);
glEnd();

// Door outline
glColor3f(0.0f, 0.0f, 0.0f);
glLineWidth(5);
glBegin(GL_LINE_STRIP);
glVertex2i(180, 120);
glVertex2i(180, 200);
glVertex2i(220, 200);
glVertex2i(220, 120);
glEnd();
```

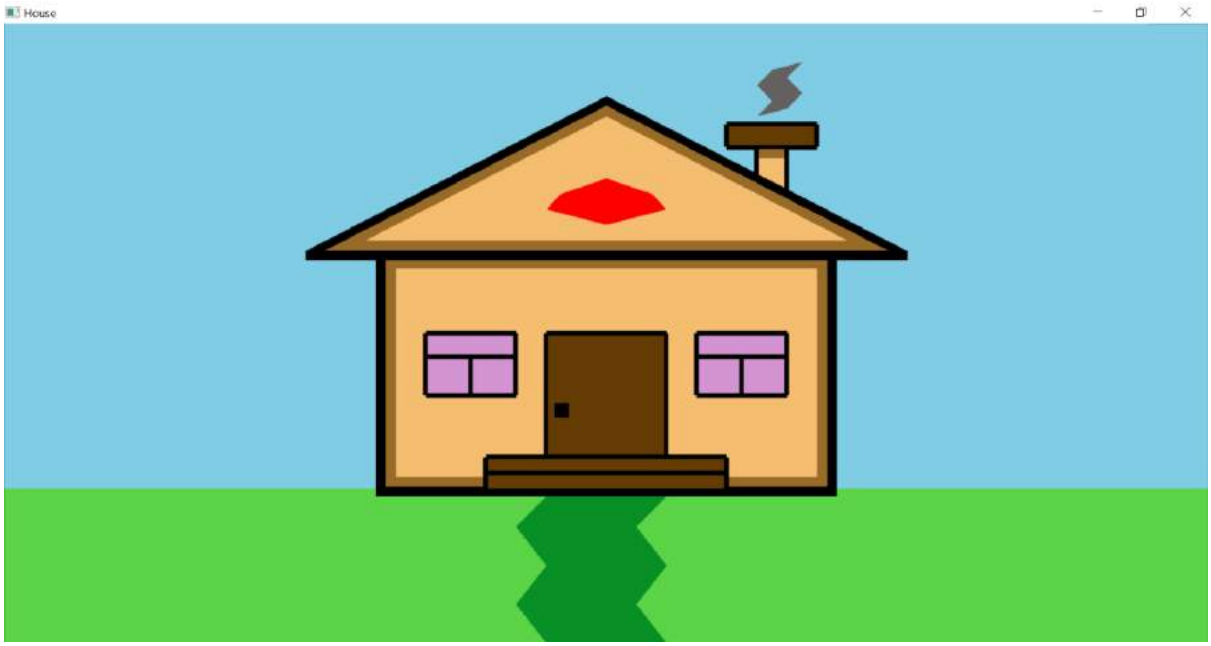
```
// Left Window
glColor3f(0.83f, 0.58f, 0.82f);
glBegin(GL_POLYGON);
glVertex2i(140, 160);
glVertex2i(140, 200);
glVertex2i(170, 200);
glVertex2i(170, 160);
glEnd();

// Left Window outline
glColor3f(0.0f, 0.0f, 0.0f);
glLineWidth(5);
glBegin(GL_LINE_LOOP);
glVertex2i(140, 160);
glVertex2i(140, 200);
glVertex2i(170, 200);
glVertex2i(170, 160);
glEnd();

// Left Window inner outline
glBegin(GL_LINES);
glVertex2i(140, 185);
glVertex2i(170, 185);

glVertex2i(155, 185);
glVertex2i(155, 160);
glEnd();

// Right Window
glColor3f(0.83f, 0.58f, 0.82f);
glBegin(GL_POLYGON);
glVertex2i(230, 160);
glVertex2i(230, 200);
glVertex2i(260, 200);
glVertex2i(260, 160);
glEnd();

// Right Window outline
glColor3f(0.0f, 0.0f, 0.0f);
glLineWidth(5);
glBegin(GL_LINE_LOOP);
glVertex2i(230, 160);
glVertex2i(230, 200);
glVertex2i(260, 200);
glVertex2i(260, 160);
glEnd();

// Right Window inner outline
glBegin(GL_LINES);
glVertex2i(230, 185);
glVertex2i(260, 185);

glVertex2i(245, 185);
glVertex2i(245, 160);
glEnd();

// Door knob
glColor3f(0.0f, 0.0f, 0.0f);
//size of point
glPointSize(15);
glBegin(GL_POINTS);
glVertex2i(185, 150);
```

```
glEnd();

// Create foot steps
glColor3f(0.396f, 0.24f, 0.016f);
glBegin(GL_POLYGON);
glVertex2i(160, 100);
glVertex2i(160, 120);
glVertex2i(240, 120);
glVertex2i(240, 100);
glEnd();

// House's top triangle
glColor3f(0.60f, 0.42f, 0.16f);
glBegin(GL_TRIANGLES);
glVertex2i(100, 250);
glVertex2i(300, 250);
glVertex2i(200, 350);
glEnd();

// Top triangle inner triangle
glColor3f(0.96f, 0.74f, 0.44f);
glLineWidth(10);
glBegin(GL_TRIANGLES);
glVertex2f(120, 260);
glVertex2f(280, 260);
glVertex2f(200, 340);
glEnd();

// House's top triangle part
glColor3f(1.0f, 0.0f, 0.0f);
glBegin(GL_TRIANGLE_FAN);
glVertex2i(200, 270);
glVertex2i(180, 280);
glVertex2i(185, 290);
glVertex2i(200, 300);
glVertex2i(215, 290);
glVertex2i(220, 280);
glEnd();

// chimney outlet smoke
glColor3f(0.4f, 0.38f, 0.38f);
glBegin(GL_TRIANGLE_STRIP);
glVertex2f(250, 340);
glVertex2f(260, 345);
glVertex2f(255, 350);
glVertex2f(265, 355);
glVertex2f(250, 360);
glVertex2f(260, 365);
glVertex2f(255, 370);
glVertex2f(265, 375);
glEnd();

// chimney outlet
glColor3f(0.396f, 0.24f, 0.016f);
glBegin(GL_QUADS);
glVertex2f(240, 320);
glVertex2f(240, 335);
glVertex2f(270, 335);
glVertex2f(270, 320);
glEnd();
```

```
// chimney body
glColor3f(0.96f, 0.74f, 0.44f);
glLineWidth(5);
glBegin(GL_QUADS);
glVertex2f(250, 300);
glVertex2f(250, 320);
glVertex2f(260, 320);
glVertex2f(260, 290);
glEnd();

// chimney outlet outline
glColor3f(0.0f, 0.0f, 0.0f);
glLineWidth(5);
glBegin(GL_LINE_LOOP);
glVertex2f(240, 320);
glVertex2f(240, 335);
glVertex2f(270, 335);
glVertex2f(270, 320);
glEnd();

glColor3f(0.60f, 0.42f, 0.16f);
glLineWidth(10);
glBegin(GL_LINE_LOOP);
glVertex2f(250, 316);
glVertex2f(260, 316);
glEnd();

// chimney body outline
glColor3f(0.0f, 0.0f, 0.0f);
glLineWidth(5);
glBegin(GL_LINES);
glVertex2f(250, 300);
glVertex2f(250, 320);

glVertex2f(260, 320);
glVertex2f(260, 290);
glEnd();

// Top triangle outline
glColor3f(0.0f, 0.0f, 0.0f);
glLineWidth(10);
glBegin(GL_LINE_LOOP);
glVertex2f(100, 250);
glVertex2f(300, 250);
glVertex2f(200, 350);
glEnd();

// Steps outline
glLineWidth(5);
glBegin(GL_LINE_STRIP);
glVertex2i(160, 100);
glVertex2i(160, 120);
glVertex2i(240, 120);
glVertex2i(240, 100);
glEnd();

glBegin(GL_LINES);
glVertex2i(160, 110);
glVertex2i(240, 110);
glEnd();
```

```
        // Pathway
        glColor3f(0.03f, 0.56f, 0.15f);
        glLineWidth(10);
        glBegin(GL_QUAD_STRIP);
        glVertex2f(180, 95);
        glVertex2f(220, 95);
        glVertex2f(170, 75);
        glVertex2f(210, 75);
        glVertex2f(180, 50);
        glVertex2f(220, 50);
        glVertex2f(170, 25);
        glVertex2f(210, 25);
        glVertex2f(180, 0);
        glVertex2f(220, 0);
        glEnd();

        // Sends all output to display
        glFlush();
}

// Driver Code
int main(int argc, char** argv)
{
        // Initialize using the init function
        glutInit(&argc, argv);

        // Sets the display mode and specify the colour scheme
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);

        // Specify the window size
        glutInitWindowSize(1200, 740);

        // Sets the starting position for the window
        glutInitWindowPosition(0, 0);

        // Creates the window and sets the title
        glutCreateWindow("House");

        glutDisplayFunc(myDisplay);

        // Additional initializations
        myInit();

        // Go into a loop until event occurs
        glutMainLoop();

        return 0;
}
```

Output Screenshot



Result

Thus, a simple House has been drawn using OpenGL.

SSN COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UCS1712 – GRAPHICS AND MULTIMEDIA LAB

EX NO: 2 – Drawing 2D Primitives – Line – DDA Algorithm

Name : **Lakshmi Priya B**
Register Number : **185001083**

Date : **29-07-2021**

1.

Aim

To plot points that make up the line with endpoints (x_0,y_0) and (x_n,y_n) using 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
with each case having two subdivisions (i) |m|<= 1 (ii) |m|>1.

Algorithm

1. Initialize using the glutInit() function with &argc and argv as parameters
2. Set the display mode as GLUT_SINGLE and specify the color scheme as GLUT_RGB
3. Specify the window position as 50, 100
4. Specify the window size as 400, 300
5. Create a new window and set the title as “DDA Line Drawing”
6. Define init() function:
    a. Set the background color as white
    b. Set glMatrixMode(GL_PROJECTION)
    c. Specify the display area
7. Call init() function
8. Define LineDDA() function:
    a. Case 1: slope is Positive and less than 1
        i. Sample at unit x interval (Δx=1) and compute each successive y value as:
        $y_{k+1} = y_k + m$
        ii. k takes integer values starting from 1, at the first point, and increasing by 1 on each step until reaching the final endpoint
        iii. The calculated y must be rounded to the nearest integer
    b. Case 2: slope is Positive and greater than 1
        i. Sample at unit y interval (Δy=1) and compute each successive y value as:
        ii. $x_{k+1} = x_k + (1/m)$
        iii. k takes integer values starting from 1, at the first point, and increasing by 1 on each step until reaching the final endpoint
        iv. The calculated x must be rounded to the nearest integer

c. Case 3: slope is negative and its absolute value is less than 1
    i. Follow the same way in Case 1
d. Case 4: slope is negative and its absolute value is greater than 1
    i. Follow the same way in Case 2
e. In the previous 4 cases, we start from the left to the right. If the state is reversed then:
    i. If the absolute slop is less than 1, set $\Delta x = -1$ and $y_{k+1} = y_k - m$
    ii. If the absolute slop is greater than 1, set $\Delta y = -1$ and $x_{k+1} = x_k - (1/m)$

9. Define the lineSegment() function:
    a. Display the window as the color of the current buffer using GL_COLOR_BUFFER_BIT
    b. Set the color of the display object
    c. Set the point size
    d. While there are more lines to be drawn:
        i. Get (x_0,y_0) and (x_n,y_n) as input from the user
        ii. Call LineDDA() with x_0, y_0, x_n, y_n as parameters
        iii. Send all the output to display using glFlush()
10. Call glutDisplayFunc(lineSegment)
11. Go into a loop using glutMainLoop() until event occurs
12. Run the application and view the output

Code

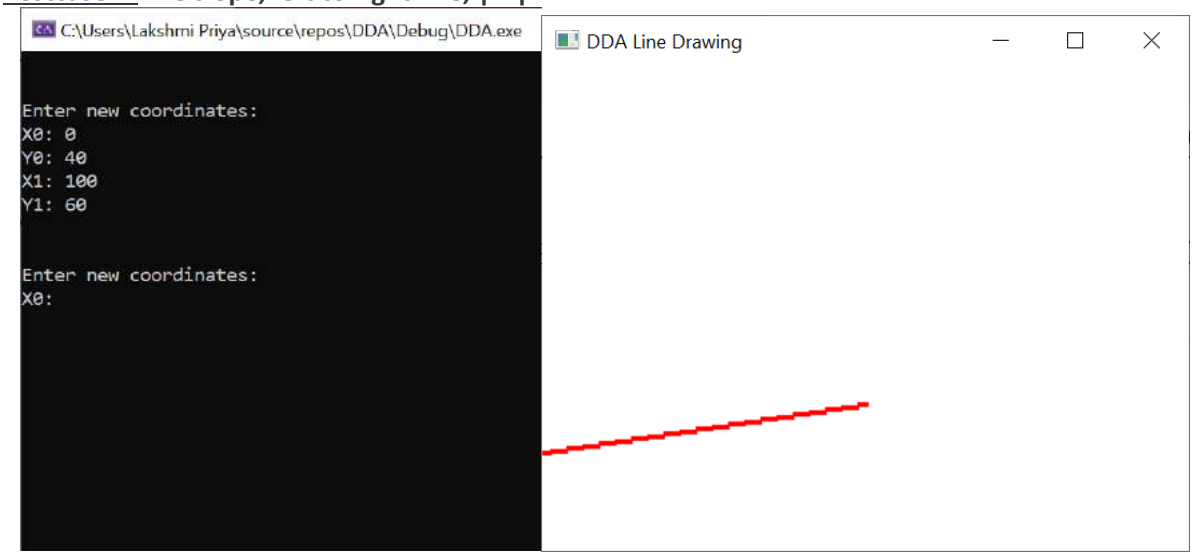
Source.cpp:

```
#include<GL/glut.h> //Handle window management operations
#include<iostream>
#include<math.h> //Mathematical functions
using namespace std;

//Initialization operation
void init()
{
	glClearColor(1.0, 1.0, 1.0, 0.0);
	glMatrixMode(GL_PROJECTION);
	gluOrtho2D(0.0, 200.0, 0.0, 200.0);
}

void LineDDA(float xs, float ys, float xe, float ye)
{
	//DDA drawing point and line algorithm
	float dx = (xe - xs);
	float dy = (ye - ys);

	int steps = abs(dx) > abs(dy) ? abs(dx) : abs(dy);

	float Xinc = dx / (float)steps;
	float Yinc = dy / (float)steps;

	float X = xs;
	float Y = ys;

	for (int i = 0; i <= steps; i++) {
		glVertex2i(round(X), round(Y));
		X += Xinc;
		Y += Yinc;
	}
}
```

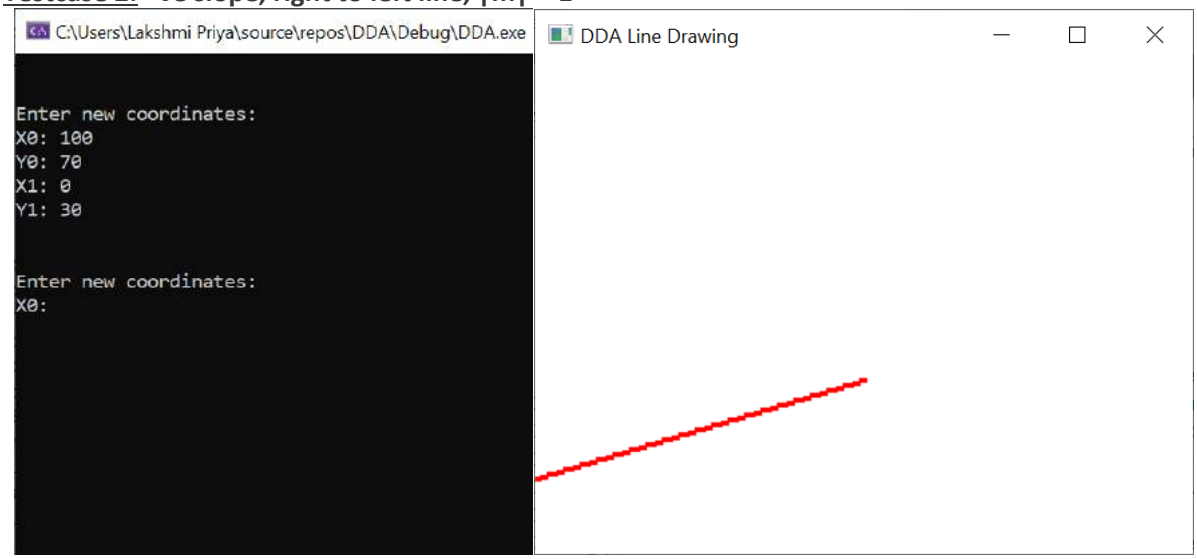
```
//Display straight line
void lineSegment(void)
{
	glClear(GL_COLOR_BUFFER_BIT); //Display the window as the color of the current
buffer
	glColor3f(0.0, 0.0, 0.0);   //Set the color of the display object, which is black
at this time.
	glPointSize(3.0f);       //Set the width of the point, that is, the line width

	float xs, ys, xe, ye;
	while (true) {
		glColor3f(1, 0.0, 0.0);
		glBegin(GL_POINTS);

		cout << "\n\nEnter new coordinates: \n";
		cout << "X0: ";
		cin >> xs;
		cout << "Y0: ";
		cin >> ys;
		cout << "X1: ";
		cin >> xe;
		cout << "Y1: ";
		cin >> ye;

		LineDDA(xs, ys, xe, ye);      //DDA algorithm draw line
		glEnd();

		glFlush();
	}
}

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

	//Initialize GLUT
	glutInit(&argc, argv);

	//Set the buffer and color mode of the window, the default single buffer and RGB
color
	glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);

	//Set the position of the window on the screen
	glutInitWindowPosition(50, 100);

	//Set window size
	glutInitWindowSize(400, 300);

	//Create a window and display the window title
	glutCreateWindow("DDA Line Drawing");

	init();

	glutDisplayFunc(lineSegment);

	glutMainLoop();
}
```

Output Screenshot

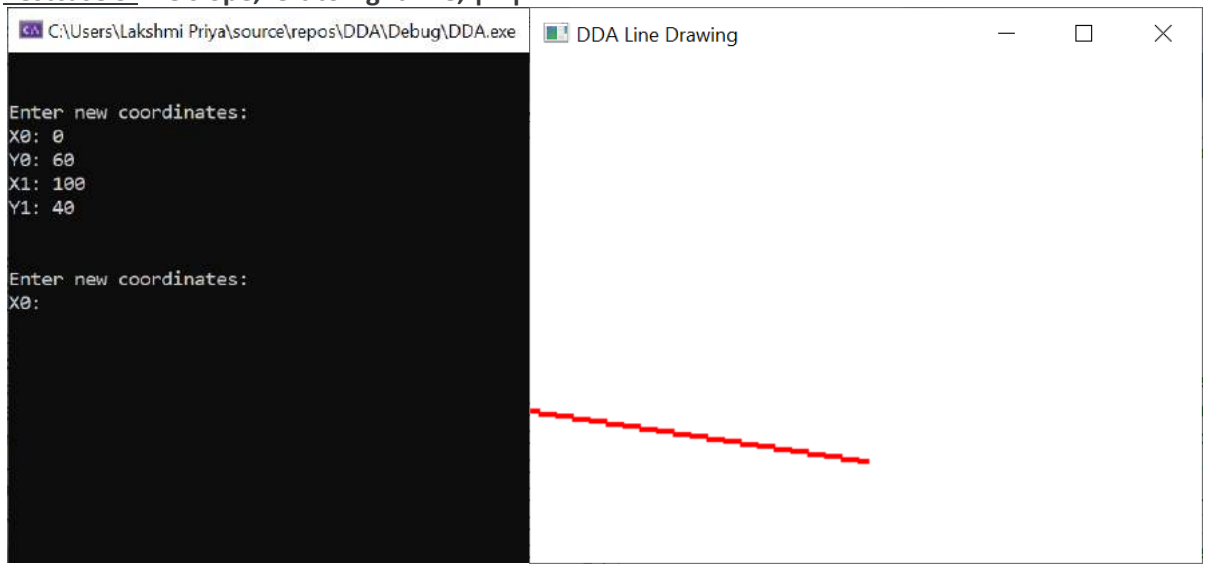
Testcase 1: +ve slope, left to right line, |m|<=1



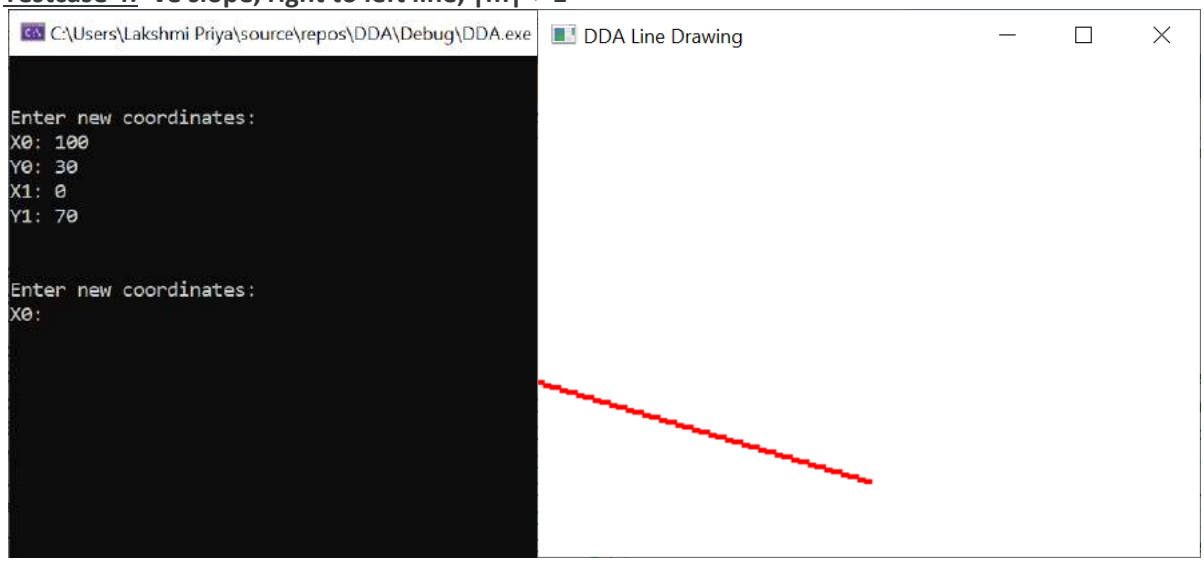
Testcase 2: +ve slope, right to left line, |m|<=1



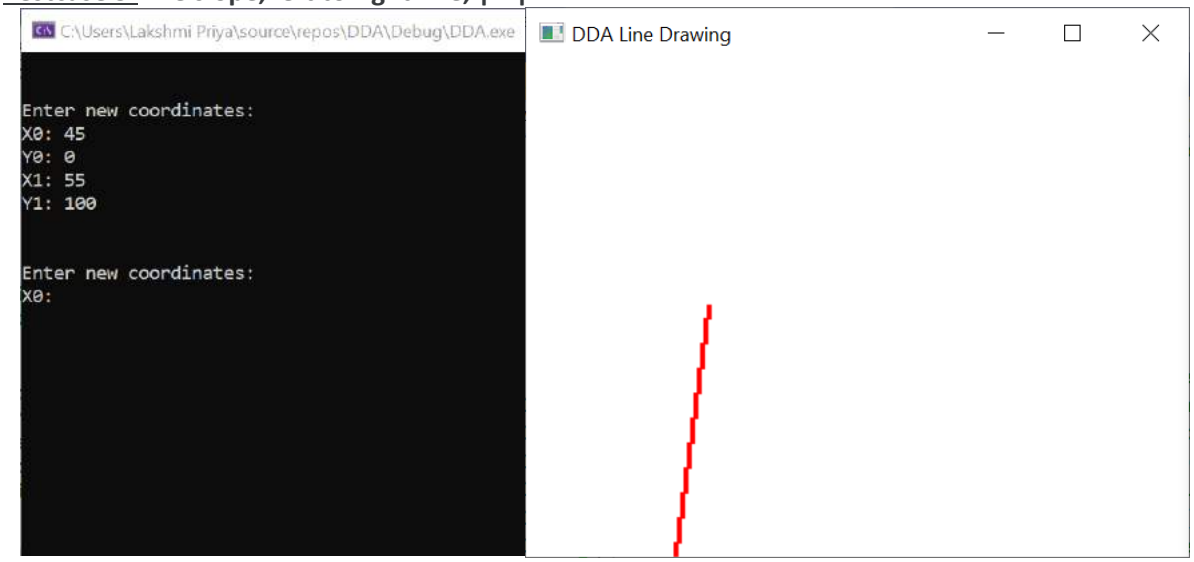
Testcase 3: -ve slope, left to right line, |m|<=1



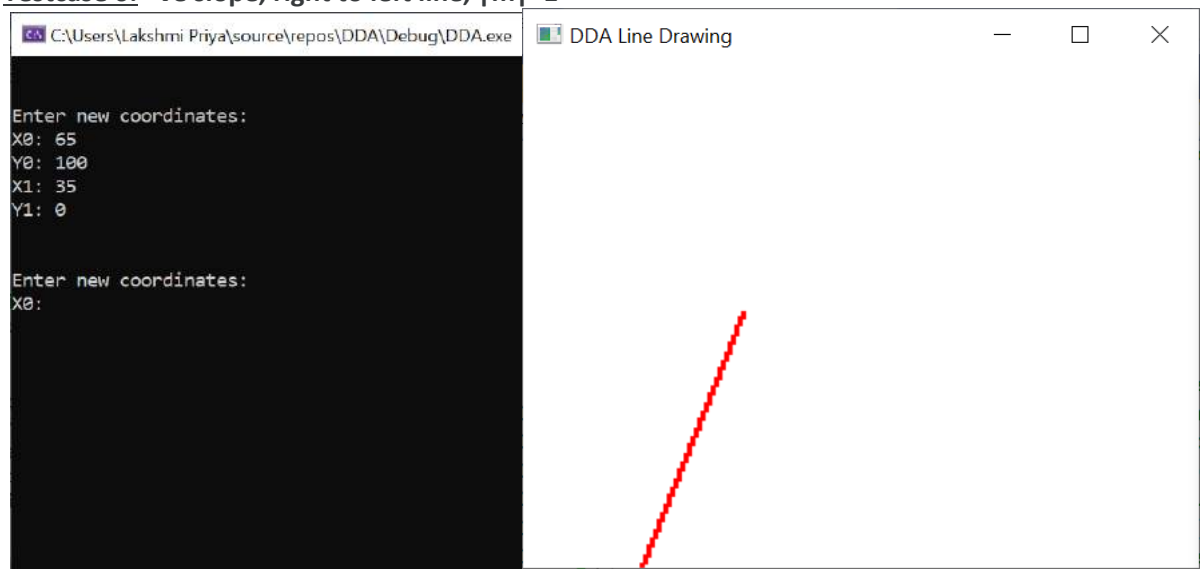
Testcase 4: -ve slope, right to left line, |m|<=1



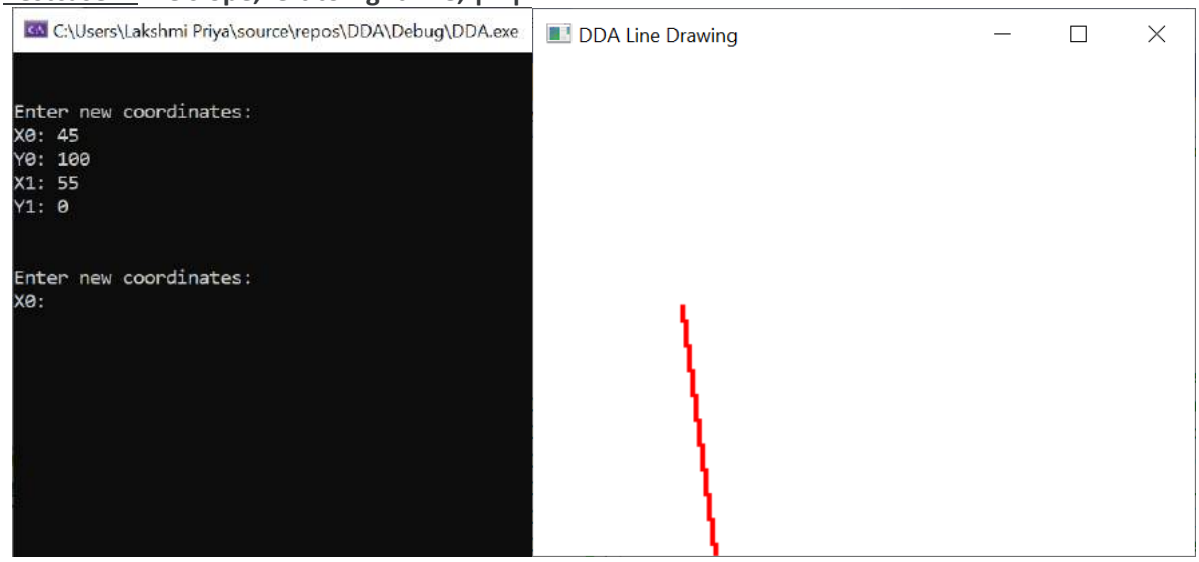
Testcase 5: +ve slope, left to right line, |m|>1



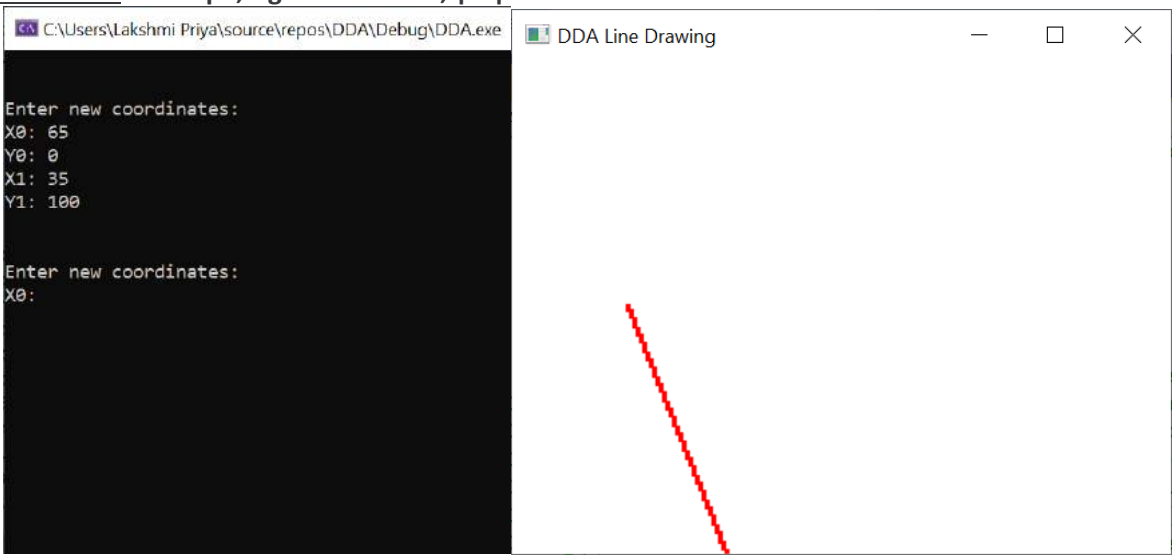
Testcase 6: +ve slope, right to left line, |m|>1



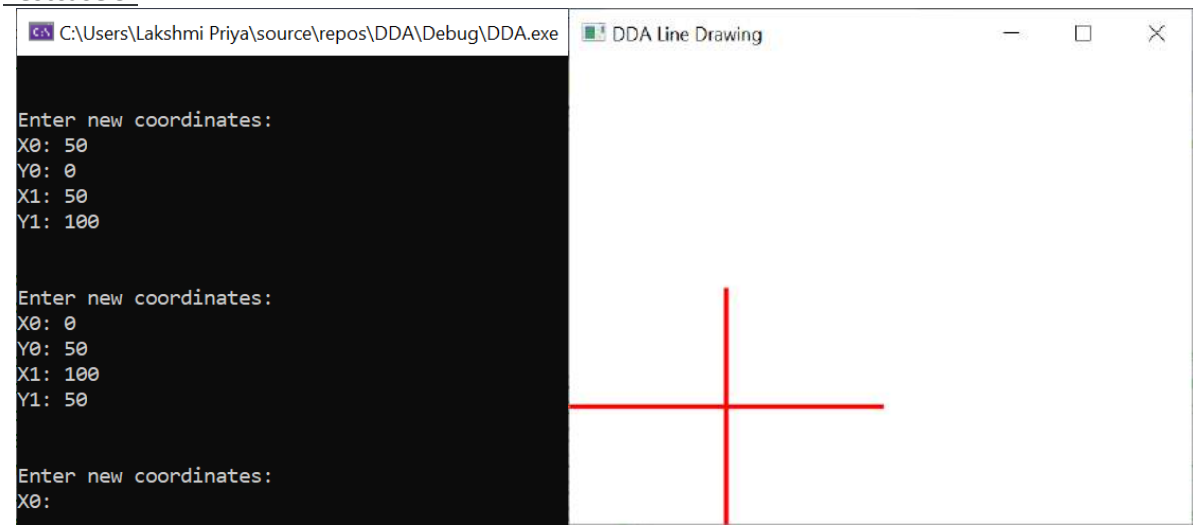
Testcase 7: -ve slope, left to right line, |m|>1



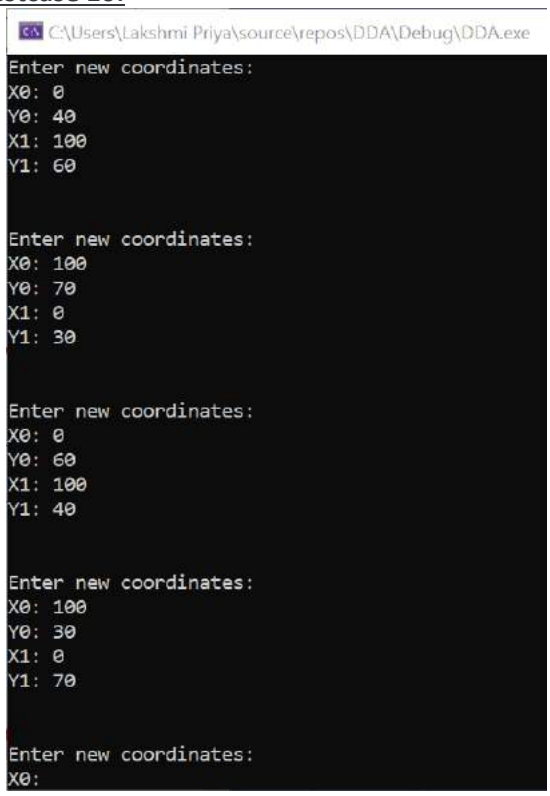
Testcase 8: -ve slope, right to left line, |m|>1

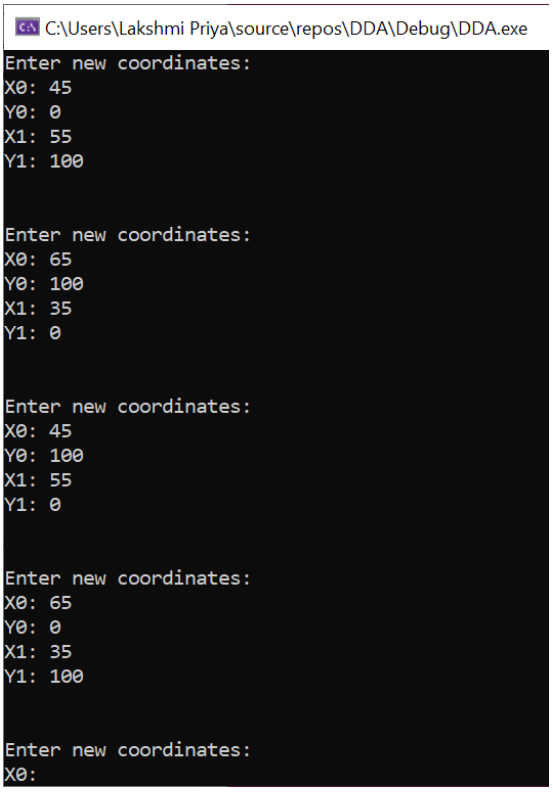


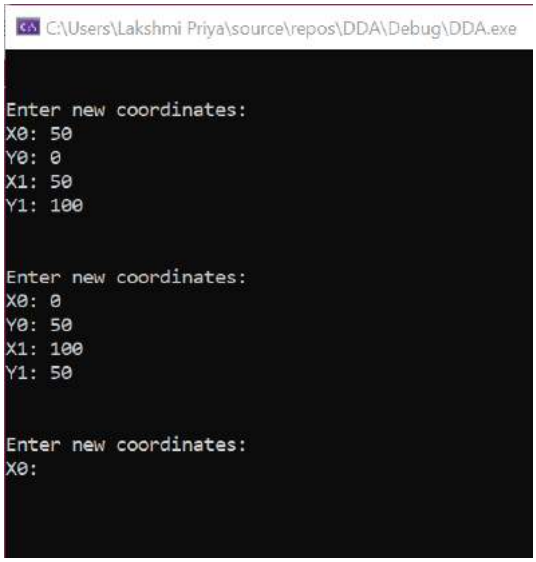
Testcase 9:

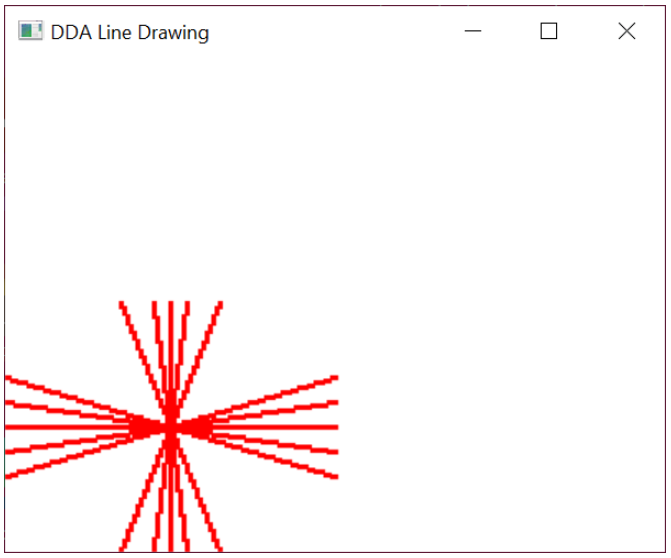


Testcase 10:









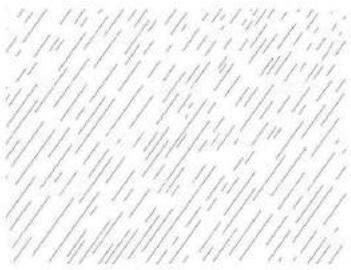
Result

Thus, lines are drawn using DDA line drawing algorithm and all the test cases have been covered.

2.

Aim

To replicate the following pattern using DDA algorithm:



Algorithm

1. Initialize using the glutInit() function with &argc and argv as parameters
2. Set the display mode as GLUT_SINGLE and specify the color scheme as GLUT_RGB
3. Specify the window position as 50, 100
4. Specify the window size as 600, 400
5. Create a new window and set the title as “DDA Line Drawing”
6. Define init() function:
    a. Set the background color as white
    b. Set glMatrixMode(GL_PROJECTION)
    c. Specify the display area
7. Call init() function
8. Define LineDDA() function:
    a. Case 1: slope is Positive and less than 1
        i. Sample at unit x interval ($\Delta x=1$) and compute each successive y value as: $y_{k+1} = y_k + m$
        ii. k takes integer values starting from 1, at the first point, and increasing by 1 on each step until reaching the final endpoint
        iii. The calculated y must be rounded to the nearest integer
    b. Case 2: slope is Positive and greater than 1
        i. Sample at unit y interval ($\Delta y=1$) and compute each successive y value as:
        ii. $x_{k+1} = x_k + (1/m)$
        iii. k takes integer values starting from 1, at the first point, and increasing by 1 on each step until reaching the final endpoint
        iv. The calculated x must be rounded to the nearest integer
    c. Case 3: slope is negative and its absolute value is less than 1
        i. Follow the same way in Case 1
    d. Case 4: slope is negative and its absolute value is greater than 1
        i. Follow the same way in Case 2
    e. In the previous 4 cases, we start from the left to the right. If the state is reversed then:
        i. If the absolute slop is less than 1, set $\Delta x = -1$ and $y_{k+1} = y_k - m$
        ii. If the absolute slop is greater than 1, set $\Delta y = -1$ and $x_{k+1} = x_k - (1/m)$
9. Define the lineSegment() function:
    a. Display the window as the color of the current buffer using GL_COLOR_BUFFER_BIT
    b. Set the color of the display object
    c. Set the point size
    d. Set starting and ending point of x, y (xs, ys, xe, ye respectively) as 0

e. Set max as 20 and min as 2
f. While xe < 200:
    i. Set xs = xs + random number between min and max
    ii. Set xe = xe + random number between min and max
    iii. Set ys = 0
    iv. Set ye = xe – xs (set so that slope of all lines is same)
    v. Set c = 0
    vi. while ye < 200:
        1. c = c + random number between min and max
        2. ys = ys + c
        3. ye = ye + c
        4. call DrawLineDDA() with xs, ys, xe, ye as parameters
        5. Send all the output to display using glFlush()
10. Call glutDisplayFunc(lineSegment)
11. Go into a loop using glutMainLoop() until event occurs
12. Run the application and view the output

Code

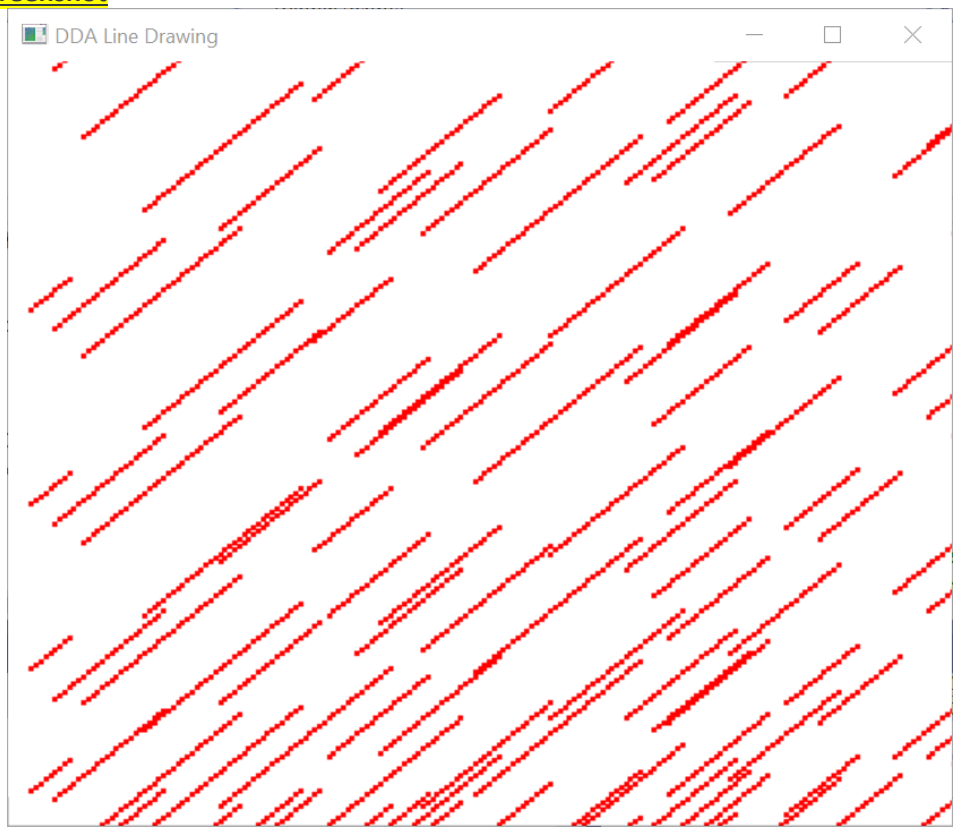
Source.cpp

```
#include<GL/glut.h> //Handle window management operations
#include <iostream>
#include<math.h> //Mathematical functions
using namespace std;

//Initialization operation
void init()
{
	//Clear the window background color to white (red, green, blue, alpha transparency)
	glClearColor(1.0, 1.0, 1.0, 0.0);

	//Set the projection matrix, indicating that a series of operations are to be performed on the projection
	glMatrixMode(GL_PROJECTION);

	gluOrtho2D(0.0, 200.0, 0.0, 200.0);
}

void DrawLineDDA(float xs, float ys, float xe, float ye)
{
	//DDA drawing point and line algorithm

	float dx = (xe - xs);
	float dy = (ye - ys);

	int steps = abs(dx) > abs(dy) ? abs(dx) : abs(dy);

	float Xinc = dx / (float)steps;
	float Yinc = dy / (float)steps;

	float X = xs;
	float Y = ys;
```

```
	for (int i = 0; i <= steps; i++) {
		glVertex2i(round(X), round(Y));
		X += Xinc;
		Y += Yinc;
	}
}

//Display straight line
void lineSegment(void)
{
	glClear(GL_COLOR_BUFFER_BIT);

	//Display the window as the color of the current buffer (the background color is not displayed)
	glColor3f(0.0, 0.0, 0.0);

	//Set the width of the point, that is, the line width
	glPointSize(2.5f);



	int xs = 0, ys = 0, xe = 0, ye = 0;
	glColor3f(1, 0.0, 0.0);
	int max = 20, min = 2;


	while (xe < 200) {
		xs += int(rand() % (max - min + 1) + min);
		xe += int(rand() % (max - min + 1) + min);

		ys = 0;
		ye = xe - xs;
		int c = 0;
		while(ye < 200) {
			c += int(rand() % (max - min + 1) + min);
			ys += c;
			ye += c;

			glBegin(GL_POINTS);
			DrawLineDDA(xs, ys, xe, ye);	//DDA algorithm draw line
			glEnd();

			glFlush();
		}
	}
}

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

	//Initialize GLUT
	glutInit(&argc, argv);

	//Set the buffer and color mode of the window, the default single buffer and RGB color
	glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);

	//Set the position of the window on the screen
	glutInitWindowPosition(50, 100);

	//Set window size
	glutInitWindowSize(600, 400);
```

```
        //Create a window and display the window title
        glutCreateWindow("DDA Line Drawing");

        init();

        glutDisplayFunc(lineSegment);

        glutMainLoop();
}
```

Output Screenshot



Result

Thus, the pattern has been replicated using DDA algorithm.

SSN COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UCS1712 – GRAPHICS AND MULTIMEDIA LAB

EX NO: 3 – Drawing 2D Primitives – Line – Bresenham's Algorithm

Name : **Lakshmi Priya B**
Register Number : **185001083**
Date : **05-08-2021**

1.

Aim

To plot points that make up the line with endpoints (x_0,y_0) and (x_n,y_n) using Bresenham's line drawing algorithm for the following case:

(i) $|m| < 1$
(ii) $|m| >= 1$

Algorithm

1. Initialize using the glutInit() function with &argc and argv as parameters
2. Set the display mode as GLUT_SINGLE and specify the color scheme as GLUT_RGB
3. Specify the window position as 50, 100
4. Specify the window size as 400, 300
5. Create a new window and set the title as "BLDA Line Drawing"
6. Define init() function:
    a. Set the background color as white
    b. Set glMatrixMode(GL_PROJECTION)
    c. Specify the display area
7. Call init() function
8. Define LineBLDA() function:
    a. Get xs, ys, xe, ye which are x, y coordinates of starting and ending points respectively
    b. Calculate dx = | xe – xs |
    c. Calculate dy = | ye – ys |
    d. Set x = xs and y = ys
    e. Set Xinc as -1 if xs > xe, else set as 1
    f. Set Yinc as -1 if ys > ye, else set as 1
    g. Set m = dy / dx
    h. Plot the first point (x, y)
    i. Case 1: slope is less than 1
        i. Set p = 2 * dy – dx
        ii. While i = 0 to dx in steps of 1
            1. If p < 0 then
                a. Plot (x + Xinc, y)
                b. p = p + 2 * dy
            2. Else

a. Plot (x + Xinc, y + Yinc)
b. p = p + 2 * dy – 2 * dx
c. y = y + Yinc

3. x = x + Xinc

j. Case 2: slope is greater than or equal to 1
   i. Set p = 2 * dx – dy
   ii. While i = 0 to dy in steps of 1
      1. If p < 0 then
         a. Plot (x, y + Yinc)
         b. p = p + 2 * dx
      2. Else
         a. Plot (x + Xinc, y + Yinc)
         b. p = p + 2 * dx – 2 * dy
         c. x = x + Xinc
      3. y = y + Yinc

9. Define the lineSegment() function:
   a. Display the window as the color of the current buffer using GL_COLOR_BUFFER_BIT
   b. Set the color of the display object
   c. Set the point size
   d. While there are more lines to be drawn:
      i. Get (x_0,y_0) and (x_n,y_n) as input from the user
      ii. Call LineBLDA() with x_0, y_0, x_n, y_n as parameters
      iii. Send all the output to display using glFlush()
10. Call glutDisplayFunc(lineSegment)
11. Go into a loop using glutMainLoop() until event occurs
12. Run the application and view the output

Code

Source.cpp:

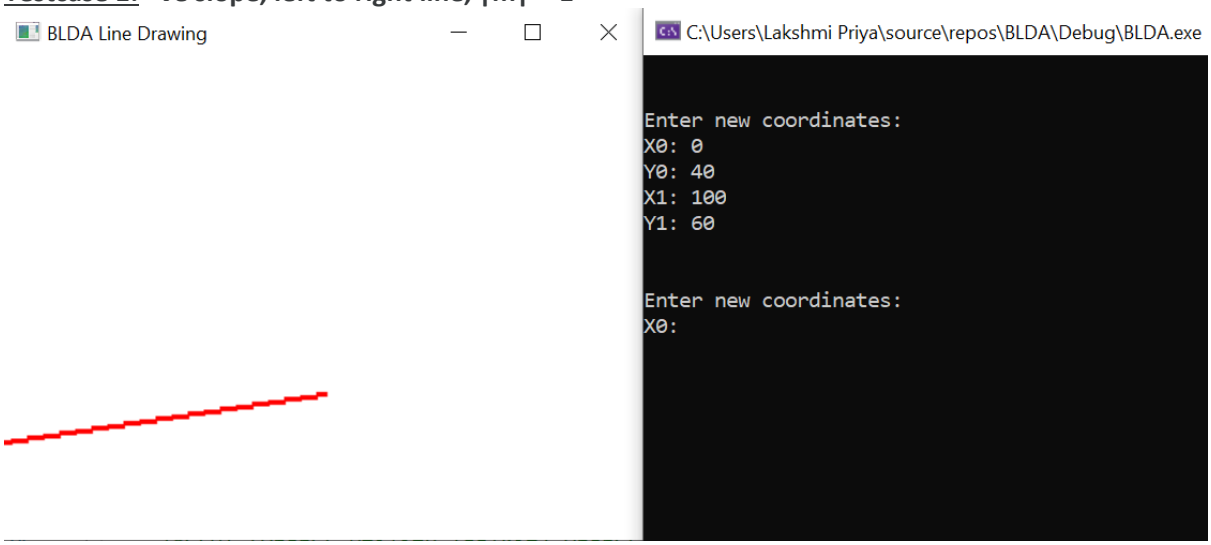
```
#include<GL/glut.h> //Handle window management operations
#include<iostream>
#include<math.h> //Mathematical functions
using namespace std;

//Initialization operation
void init()
{
	glClearColor(1.0, 1.0, 1.0, 0.0);
	glMatrixMode(GL_PROJECTION);
	gluOrtho2D(0.0, 200.0, 0.0, 200.0);
}

void LineBLDA(float xs, float ys, float xe, float ye)
{
	//BLDA drawing point and line algorithm
	float dx = abs(xe - xs);
	float dy = abs(ye - ys);
	int x = xs;
	int y = ys;
	int Xinc = xs > xe ? -1 : 1;
	int Yinc = ys > ye ? -1 : 1;
```

```
	float m = dy / dx;
	float p;

	glVertex2i(x, y);

	if (m < 1) {
		p = 2 * dy - dx;

		for (int i = 0; i <= abs(dx); i++) {
			if (p < 0) {
				glVertex2i(x + Xinc, y);
				p = p + 2 * dy;
			}
			else {
				glVertex2i(x + Xinc, y + Yinc);
				p = p + 2 * dy - 2 * dx;
				y = y + Yinc;
			}
			x = x + Xinc;
		}
	}
	else {
		p = 2 * dx - dy;
		for (int i = 0; i <= abs(dy); i++) {
			if (p < 0) {
				glVertex2i(x, y + Yinc);
				p = p + 2 * dx;
			}
			else {
				glVertex2i(x + Xinc, y + Yinc);
				p = p + 2 * dx - 2 * dy;
				x = x + Xinc;
			}
			y = y + Yinc;
		}
	}
}

//Display straight line
void lineSegment(void)
{
	glClear(GL_COLOR_BUFFER_BIT); //Display the window as the color of the current
buffer
	glColor3f(0.0, 0.0, 0.0);   //Set the color of the display object, which is black
at this time.
	glPointSize(3.0f);       //Set the width of the point, that is, the line width

	float xs, ys, xe, ye;
	while (true) {
		glColor3f(1, 0.0, 0.0);
		glBegin(GL_POINTS);

		cout << "\n\nEnter new coordinates: \n";
		cout << "X0: ";
		cin >> xs;
		cout << "Y0: ";
		cin >> ys;
		cout << "X1: ";
		cin >> xe;
		cout << "Y1: ";
		cin >> ye;
```

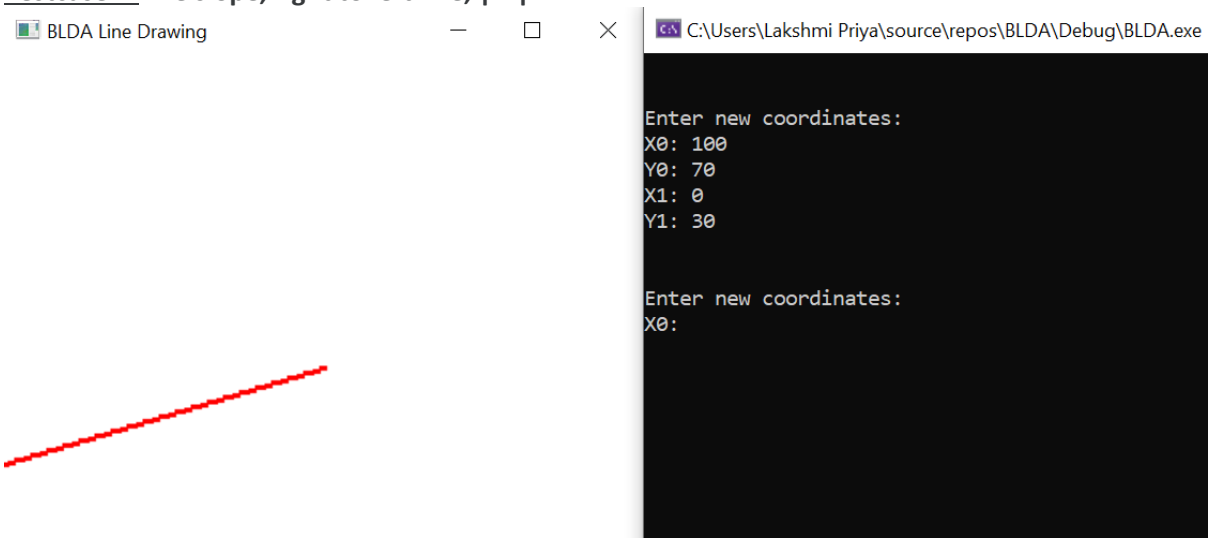
```
                LineBLDA(xs, ys, xe, ye);       //BLDA algorithm draw line
                glEnd();

                glFlush();
        }
}

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

        //Initialize GLUT
        glutInit(&argc, argv);

        //Set the buffer and color mode of the window, the default single buffer and RGB
color
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);

        //Set the position of the window on the screen
        glutInitWindowPosition(50, 100);

        //Set window size
        glutInitWindowSize(400, 300);

        //Create a window and display the window title
        glutCreateWindow("BLDA Line Drawing");

        init();

        glutDisplayFunc(lineSegment);

        glutMainLoop();
}
```

Output Screenshot

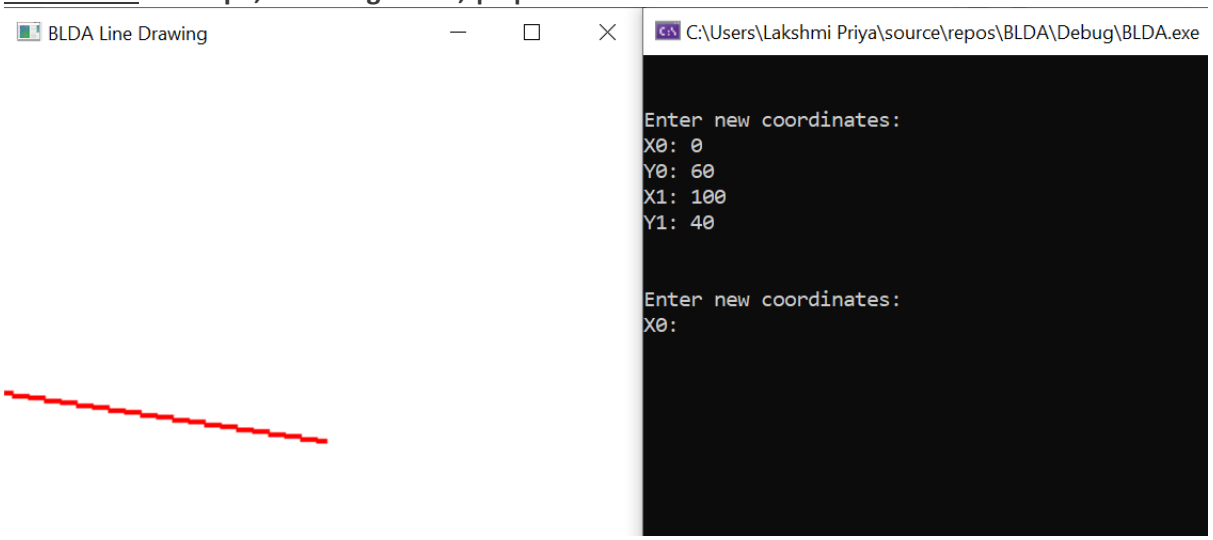
Testcase 1: +ve slope, left to right line, |m|<=1



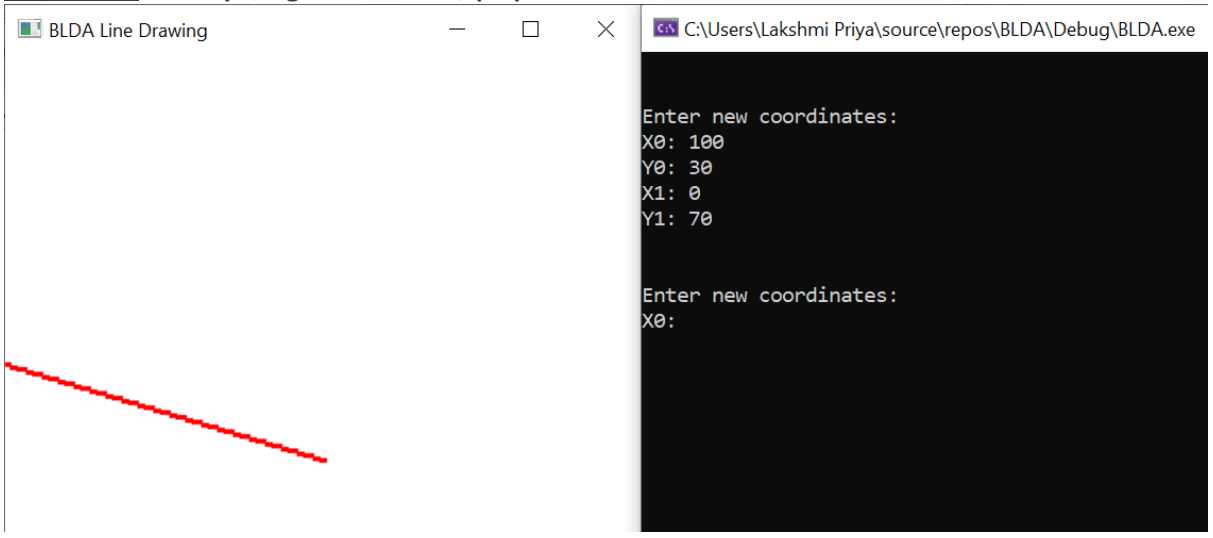
Testcase 2: +ve slope, right to left line, |m|<=1



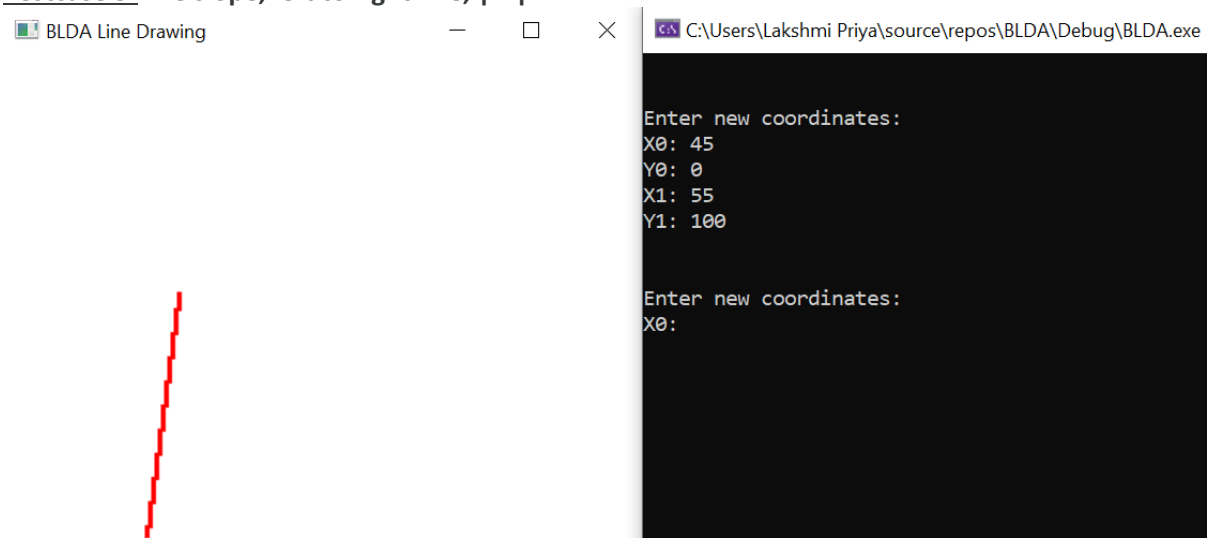
Testcase 3: -ve slope, left to right line, |m|<=1



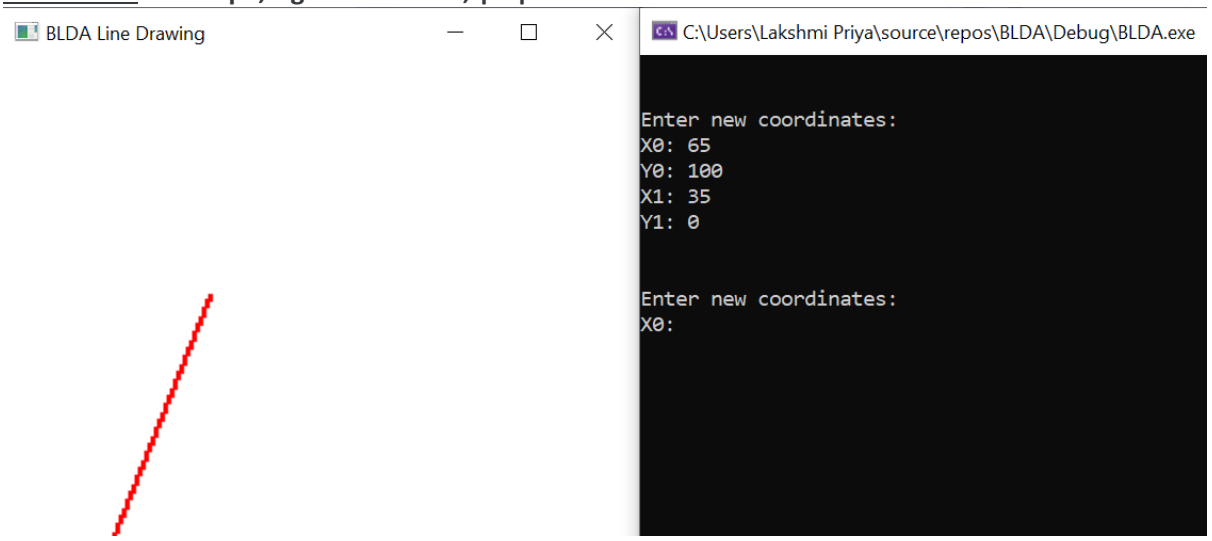
Testcase 4: -ve slope, right to left line, |m|<=1



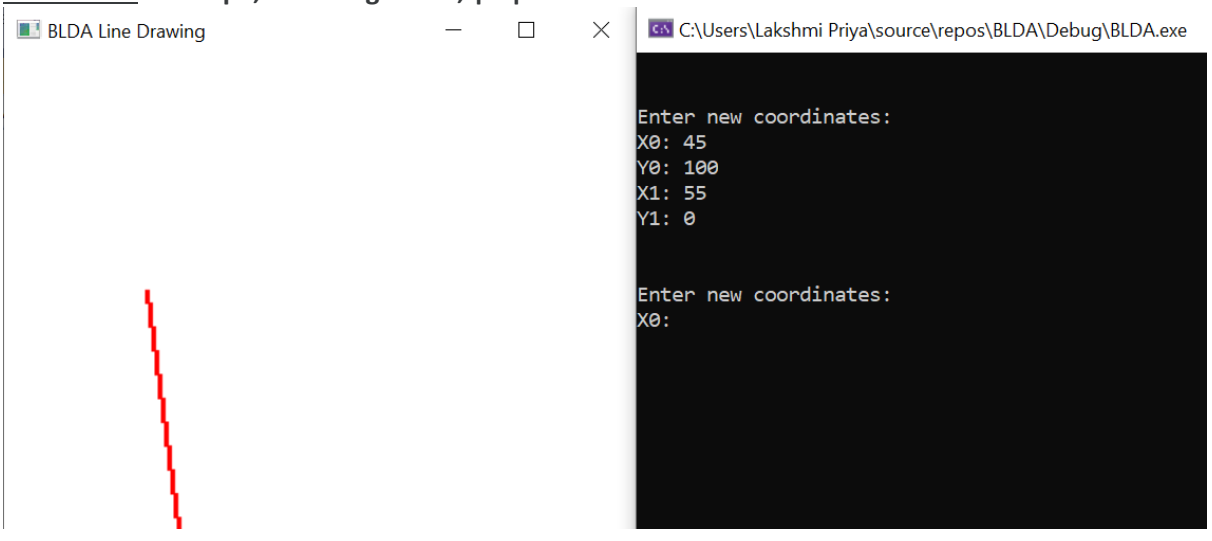
Testcase 5: +ve slope, left to right line, |m|>1



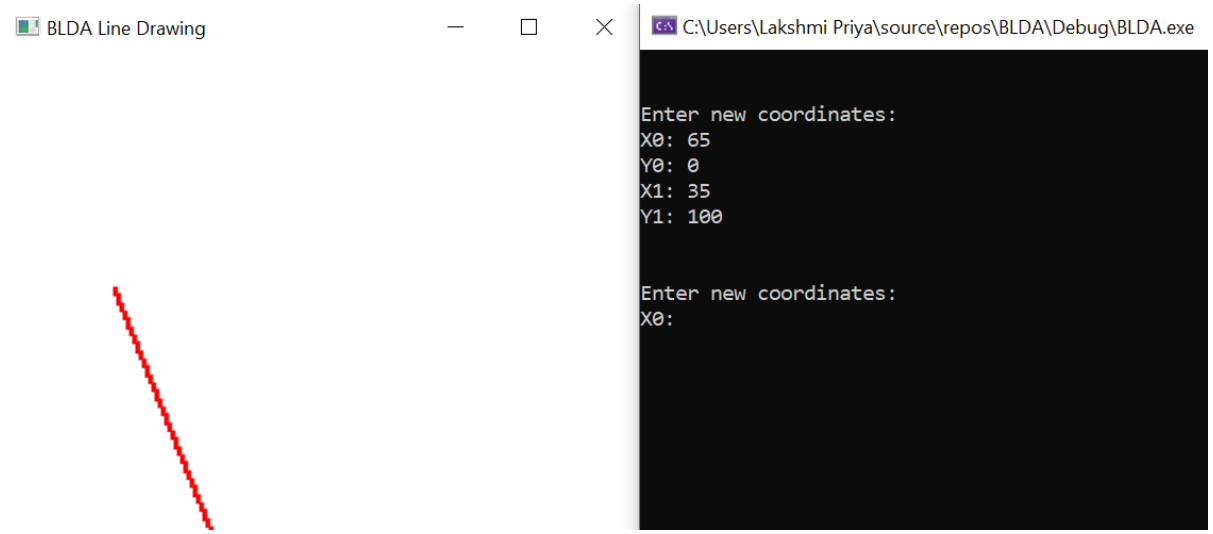
Testcase 6: +ve slope, right to left line, |m|>1



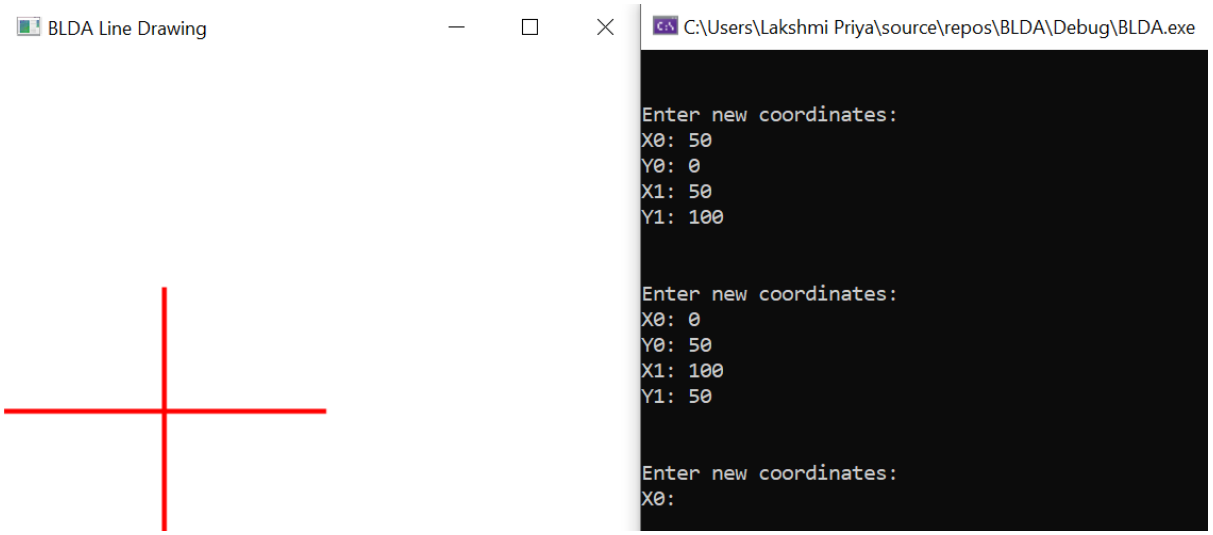
Testcase 7: -ve slope, left to right line, |m|>1



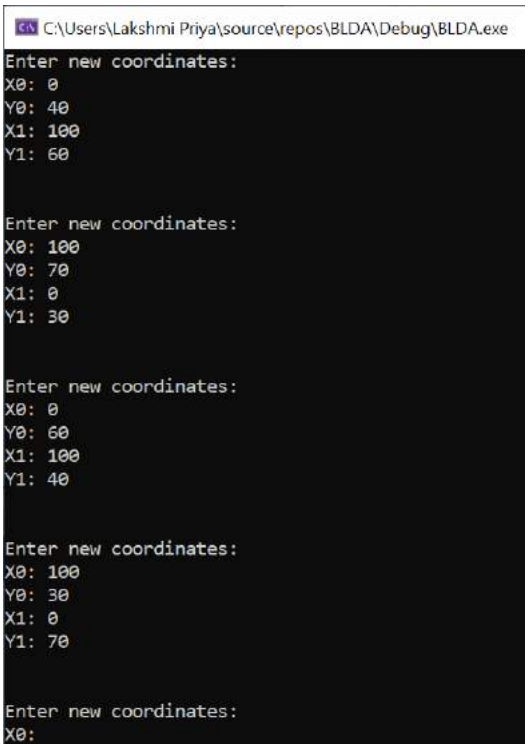
Testcase 8: **-ve slope, right to left line, |m|>1**

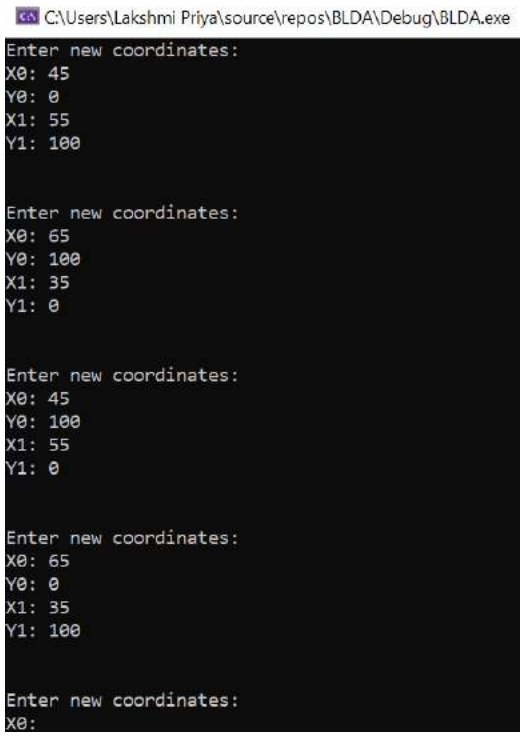


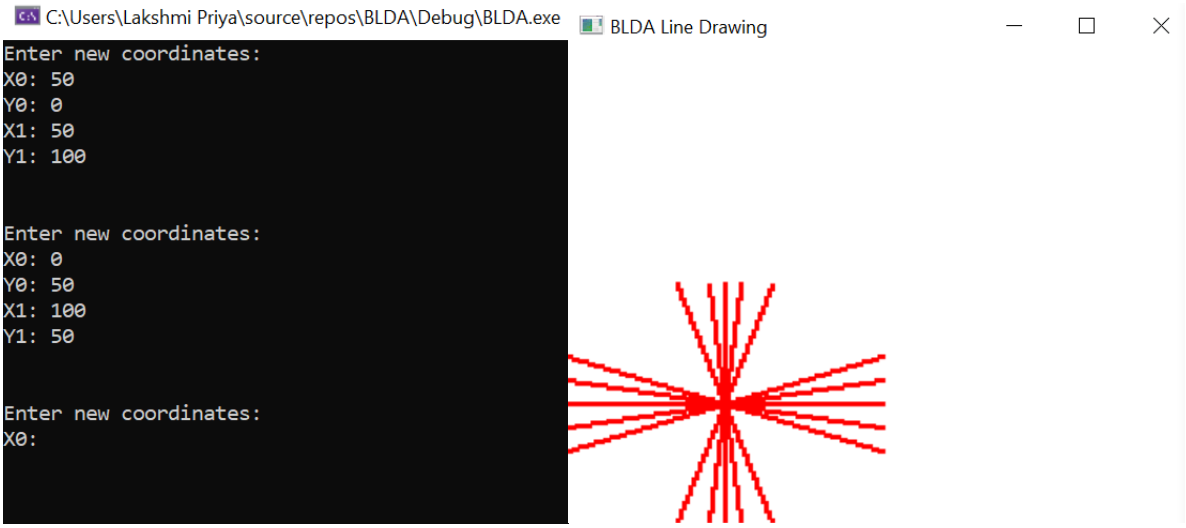
Testcase 9:



Testcase 10:







Result

Thus, lines are drawn using Bresenham's line drawing algorithm and all the test cases have been covered.

2.

Aim

To write a C++ program using OPENGL to write any Alphabet (using sleeping, slanting, standing lines) with the help of Bresenham's line drawing algorithm.

Algorithm

1. Initialize using the glutInit() function with &argc and argv as parameters
2. Set the display mode as GLUT_SINGLE and specify the color scheme as GLUT_RGB
3. Specify the window position as 50, 100
4. Specify the window size as 400, 300
5. Create a new window and set the title as "Alphabet using BLDA"
6. Define init() function:
    a. Set the background color as white
    b. Set glMatrixMode(GL_PROJECTION)
    c. Specify the display area
7. Call init() function
8. Define LineBLDA() function:
    a. Get xs, ys, xe, ye which are x, y coordinates of starting and ending points respectively
    b. Calculate dx = | xe – xs |
    c. Calculate dy = | ye – ys |
    d. Set x = xs and y = ys
    e. Set Xinc as -1 if xs > xe, else set as 1
    f. Set Yinc as -1 if ys > ye, else set as 1
    g. Set m = dy / dx
    h. Plot the first point (x, y)
    i. Case 1: slope is less than 1
        i. Set p = 2 * dy – dx
        ii. While i = 0 to dx in steps of 1
            1. If p < 0 then
                a. Plot (x + Xinc, y)
                b. p = p + 2 * dy
            2. Else
                a. Plot (x + Xinc, y + Yinc)
                b. p = p + 2 * dy – 2 * dx
                c. y = y + Yinc
            3. x = x + Xinc
    j. Case 2: slope is greater than or equal to 1
        i. Set p = 2 * dx – dy
        ii. While i = 0 to dy in steps of 1
            1. If p < 0 then
                a. Plot (x, y + Yinc)
                b. p = p + 2 * dx
            2. Else
                a. Plot (x + Xinc, y + Yinc)
                b. p = p + 2 * dx – 2 * dy
                c. x = x + Xinc
            3. y = y + Yinc

9. Define the lineSegment() function:
    a. Display the window as the color of the current buffer using GL_COLOR_BUFFER_BIT
    b. Set the color of the display object
    c. Set the point size
    d. Use LineBLDA() with x_0, y_0, x_n, y_n as parameters to draw the strokes of the alphabet
    e. Execute LineBLDA(60, 70, 90, 70) to draw the horizontal stroke line of letter L
    f. Execute LineBLDA(60, 130, 60, 70) to draw the vertical stroke line of letter L
    g. Execute LineBLDA(100, 70, 100, 130) to draw the vertical stroke line of letter P
    h. Execute LineBLDA(100, 130, 110, 127) to draw the inner upper slanting stroke line of letter P
    i. Execute LineBLDA(110, 127, 120, 115) to draw the outer upper slanting stroke line of letter P
    j. Execute LineBLDA(120, 115, 120, 105) to draw the vertical stroke line of letter P
    k. Execute LineBLDA(120, 105, 110, 93) to draw the outer lower slanting stroke line of letter P
    l. Execute LineBLDA(110, 93, 100, 90) to draw the inner lower slanting stroke line of letter P
    m. Send all the output to display using glFlush()
10. Call glutDisplayFunc(lineSegment)
11. Go into a loop using glutMainLoop() until event occurs
12. Run the application and view the output

Code

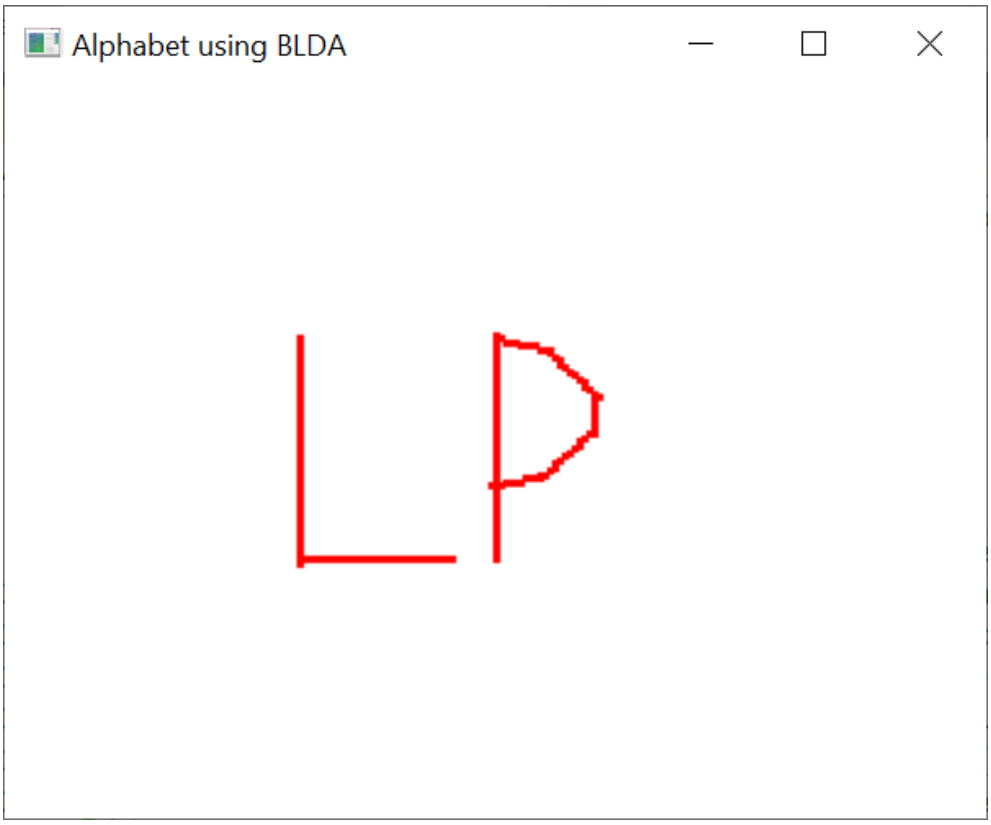
Source.cpp

```
#include<GL/glut.h> //Handle window management operations
#include<iostream>
#include<math.h> //Mathematical functions
using namespace std;

//Initialization operation
void init()
{
	//Clear the window background color to white (red, green, blue, alpha transparency)
	glClearColor(1.0, 1.0, 1.0, 0.0);

	//Set the projection matrix, indicating that a series of operations are to be performed on the projection
	glMatrixMode(GL_PROJECTION);
	gluOrtho2D(0.0, 200.0, 0.0, 200.0);
}

void LineBLDA(float xs, float ys, float xe, float ye)
{  //BLDA drawing point and line algorithm

	float dx = abs(xe - xs);
	float dy = abs(ye - ys);
	int x = xs;
	int y = ys;
	int Xinc = xs > xe ? -1 : 1;
	int Yinc = ys > ye ? -1 : 1;

	float m = dy / dx;
	float p;
```

```
	glVertex2i(x, y);

	if (m < 1) {
		p = 2 * dy - dx;

		for (int i = 0; i <= abs(dx); i++) {
			if (p < 0) {
				glVertex2i(x + Xinc, y);
				p = p + 2 * dy;
			}
			else {
				glVertex2i(x + Xinc, y + Yinc);
				p = p + 2 * dy - 2 * dx;
				y = y + Yinc;
			}
			x = x + Xinc;
			//cout << x << "-" << y << "@" << p << endl;
		}
	}
	else {
		p = 2 * dx - dy;
		for (int i = 0; i <= abs(dy); i++) {
			if (p < 0) {
				glVertex2i(x, y + Yinc);
				p = p + 2 * dx;
			}
			else {
				glVertex2i(x + Xinc, y + Yinc);
				p = p + 2 * dx - 2 * dy;
				x = x + Xinc;
			}
			y = y + Yinc;
			//cout << x << "-" << y << "@" << p << endl;
		}
	}

}


//Display straight line
void lineSegment(void)
{

	//Display the window as the color of the current buffer
	glClear(GL_COLOR_BUFFER_BIT);

	//Set the color of the display object, which is black at this time.
glColor3f(R,G,B)
	glColor3f(0.0, 0.0, 0.0);

	//Set the width of the point, that is, the line width
	glPointSize(3.0f);

	glColor3f(1, 0.0, 0.0);
	glBegin(GL_POINTS);
                                        */

	//BLDA algorithm draw line

	//horizontal stroke of L
	LineBLDA(60, 70, 90, 70);
```

```
	//vertical stroke of L
	LineBLDA(60, 130, 60, 70);

	//vertical stroke of P
	LineBLDA(100, 70, 100, 130);

	//inner upper slating stroke of P
	LineBLDA(100, 130, 110, 127);

	//outer upper slating stroke of P
	LineBLDA(110, 127, 120, 115);

	//vertical stroke of P
	LineBLDA(120, 115, 120, 105);

	//outer lower slating stroke of P
	LineBLDA(120, 105, 110, 93);

	//inner lower slating stroke of P
	LineBLDA(110, 93, 100, 90);

	glEnd();

	glFlush();

}


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

	//Initialize GLUT
	glutInit(&argc, argv);

	//Set the buffer and color mode of the window, the default single buffer and
RGB color
	glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);

	//Set the position of the window on the screen
	glutInitWindowPosition(50, 100);

	//Set window size
	glutInitWindowSize(400, 300);

	//Create a window and display the window title
	glutCreateWindow("Alphabet using BLDA");

	init();

	glutDisplayFunc(lineSegment);

	glutMainLoop();
}
```

Output Screenshot



Result

Thus, C++ program using OPENGL has been developed to write any Alphabet using sleeping, slanting, standing lines with the help of Bresenham's line drawing algorithm.

SSN COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UCS1712 – GRAPHICS AND MULTIMEDIA LAB

EX NO: 4 – Midpoint Circle Drawing Algorithm

Name : **Lakshmi Priya B**
Register Number : **185001083**
Date : **12-08-2021**

1.

Aim

To write a C++ program using OpenGL to implement Midpoint Circle drawing algorithm with radius and a centre given as user input.

Algorithm

1. Get the coordinates of the centre of the circle, xc and xy, as input from the user.
2. Get radius as input from the user.
3. Initialize using the glutInit() function with &argc and argv as parameters
4. Set the display mode as GLUT_SINGLE and specify the color scheme as GLUT_RGB
5. Specify the window position as 100, 150
6. Specify the window size as 400, 400
7. Create a new window and set the title as “Midpoint Circle Drawing Algorithm”
8. Define init() function:
    a. Set the background color as white
    b. Set glMatrixMode(GL_PROJECTION)
    c. Specify the display area
    d. Set the point size as 4
9. Define the plot(x, y) function:
    a. Set the point color
    b. Plot the point (x+xc, y+yc) using glVertex2i()
10. Define midPointCircleAlgorithm() function:
    a. Set x = 0 and y = r
    b. Set decision = 1 – r
    c. Call plot(x, y)
    d. While y > x:
        i. If decision < 0
            1. Increment x by 1
            2. Increment decision by 2 * x + 1
        ii. Else
            1. Decrement y by 1
            2. Increment x by 1
            3. Increment decision by 2 * (x - y) + 1
        iii. plot(x, y)

iv. plot(x, -y)
v. plot(-x, y)
vi. plot(-x, -y)
vii. plot(y, x)
viii. plot(-y, x)
ix. plot(y, -x)
x. plot(-y, -x)

11. Define the myDisplay() function:
    a. Display the window as the color of the current buffer using GL_COLOR_BUFFER_BIT
    b. Set the color of the display object
    c. Set the point size as 1
    d. Draw the x-axis and y-axis for reference using GL_LINES by specifing the coordinates
    e. Invoke the midPointCircleAlgorithm()
    f. Send all the output to display using glFlush()
12. Call glutDisplayFunc(myDisplay)
13. Call myinit() function
14. Go into a loop using glutMainLoop() until event occurs
15. Run the application and view the output after providing appropriate input.

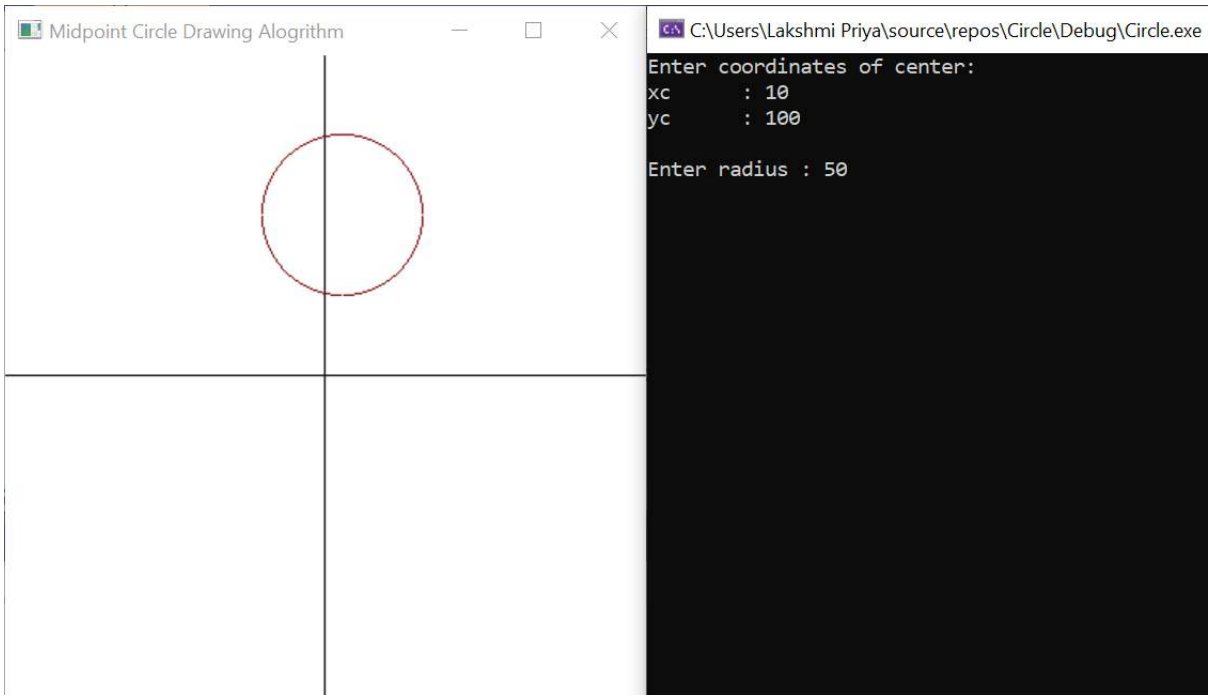
Code

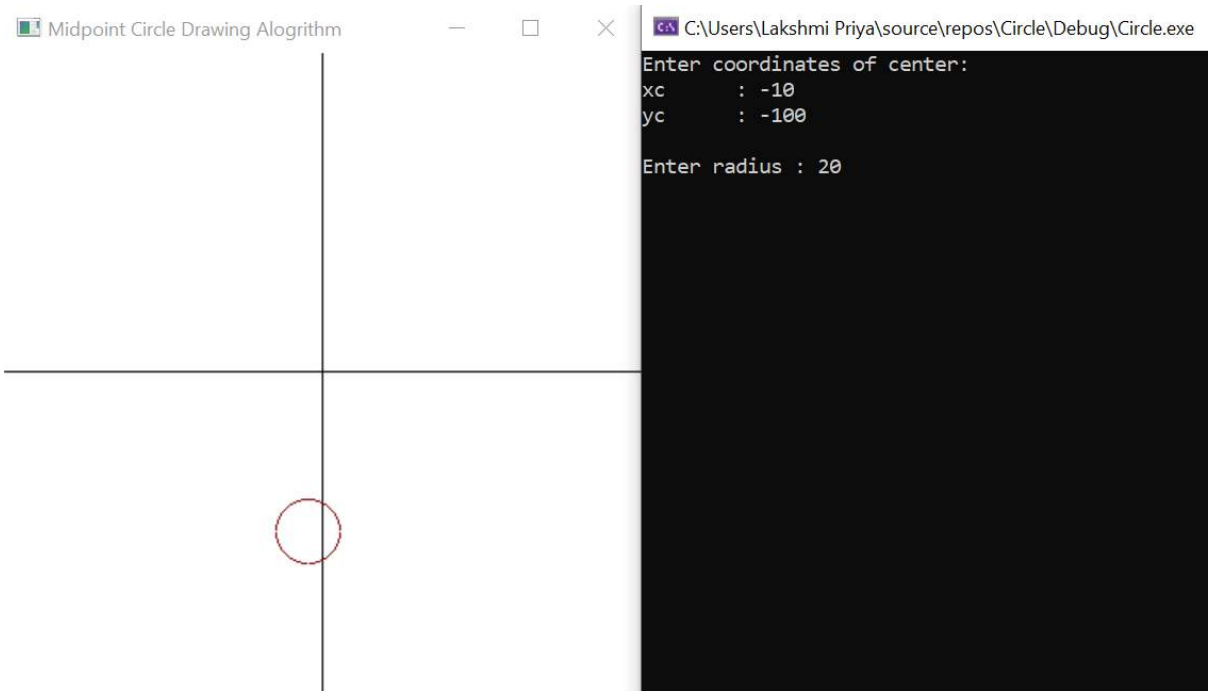
Source.cpp:

```
#include<iostream>
#include <GL/glut.h>
using namespace std;

int xc, yc, r;

void plot(int x, int y)
{
	glColor3f(0.5f, 0.0f, 0.0f);
	glBegin(GL_POINTS);
	glVertex2i(x + xc, y + yc);
	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(-200.0, 200.0, -200.0, 200.0);
}

void midPointCircleAlgorithm()
{
	int x = 0;
	int y = r;
	float decision = 1 - r;
	plot(x, y);
```

```
	while (y > x)
	{
		if (decision < 0)
		{
			x++;
			decision += 2 * x + 1;
		}
		else
		{
			y--;
			x++;
			decision += 2 * (x - y) + 1;
		}
		plot(x, y);
		plot(x, -y);
		plot(-x, y);
		plot(-x, -y);
		plot(y, x);
		plot(-y, x);
		plot(y, -x);
		plot(-y, -x);
	}
}

void myDisplay(void)
{
	glClear(GL_COLOR_BUFFER_BIT);
	glColor3f(0.0, 0.0, 0.0);
	glPointSize(1.0);

	glBegin(GL_LINES);
	glVertex2d(0, -200);
	glVertex2d(0, 200);
	glVertex2d(-200, 0);
	glVertex2d(200, 0);
	glEnd();

	midPointCircleAlgorithm();

	glFlush();
}

void main(int argc, char** argv)
{
	cout << "Enter coordinates of center:" << endl;
	cout << "xc   : ";
	cin >> xc;
	cout << "yc   : ";
	cin >> yc;
	cout << "\nEnter radius : ";
	cin >> r;

	glutInit(&argc, argv);
	glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
	glutInitWindowSize(400, 400);
	glutInitWindowPosition(100, 150);
	glutCreateWindow("Midpoint Circle Drawing Alogrithm");
	glutDisplayFunc(myDisplay);
	myInit();
	glutMainLoop();

}
```

Output Screenshot





Result

Thus, circle has been drawn using Midpoint Circle drawing algorithm with radius and a centre given as user input.

2.

Aim

To write a C++ program using OPENGL to replicate any circular object with the help of Midpoint Circle algorithm by using the necessary colours and elements to show details.

Algorithm

1. Initialize using the glutInit() function with &argc and argv as parameters
2. Set the display mode as GLUT_SINGLE and specify the color scheme as GLUT_RGB
3. Specify the window position as 100, 150
4. Specify the window size as 400, 400
5. Create a new window and set the title as “Target”
6. Set coordinates of centre (xc, yc) as 0, 0
7. Define init() function:
    a. Set the background color as white
    b. Set glMatrixMode(GL_PROJECTION)
    c. Specify the display area
    d. Set the point size as 10
8. Define the plot(x, y) function:
    a. Plot the point (x+xc, y+yc) using glVertex2i()
9. Define midPointCircleAlgorithm(r) function:
    a. Set x = 0 and y = r
    b. Set decision = 1 – r
    c. Call plot(x, y)
    d. While y > x:
        i. If decision < 0
            1. Increment x by 1
            2. Increment decision by 2 * x + 1
        ii. Else
            1. Decrement y by 1
            2. Increment x by 1
            3. Increment decision by 2 * (x - y) + 1
        iii. plot(x, y)
        iv. plot(x, -y)
        v. plot(-x, y)
        vi. plot(-x, -y)
        vii. plot(y, x)
        viii. plot(-y, x)
        ix. plot(y, -x)
        x. plot(-y, -x)
10. Define the myDisplay() function:
    a. Display the window as the color of the current buffer using GL_COLOR_BUFFER_BIT
    b. Set the point size as 2
    c. Set the color of the inner circle as yellow
    d. for i from 1 to 20:
        i. Invoke midPointCircleAlgorithm(i)

e. Draw two black outlines after setting black color using glColor3f() and by invoking midPointCircleAlgorithm(10) for inner outline and midPointCircleAlgorithm(20) for outer outline of yellow region
f. Set the color of the next concentric outer circle as red
g. for i from 21 to 40:
    i. Invoke midPointCircleAlgorithm(i)
h. Draw two black outlines after setting black color using glColor3f() and by invoking midPointCircleAlgorithm(30) for inner outline and midPointCircleAlgorithm(40) for outer outline of red region
i. Set the color of the next concentric outer circle as blue
j. for i from 41 to 60:
    i. Invoke midPointCircleAlgorithm(i)
k. Draw two black outlines after setting black color using glColor3f() and by invoking midPointCircleAlgorithm(50) for inner outline and midPointCircleAlgorithm(60) for outer outline of blue region
l. Set the color of the next concentric outer circle as black
m. for i from 61 to 80:
    i. Invoke midPointCircleAlgorithm(i)
n. Draw one white outline after setting white color using glColor3f() and by invoking midPointCircleAlgorithm(70) for inner outline of black region
o. Draw three black outlines after setting black color using glColor3f() and by invoking midPointCircleAlgorithm(80), midPointCircleAlgorithm(90), midPointCircleAlgorithm (100) for outline of white region
p. Send all the output to display using glFlush()

11. Call glutDisplayFunc(myDisplay)
12. Call myinit() function
13. Go into a loop using glutMainLoop() until event occurs
14. Run the application and view the output.

Code

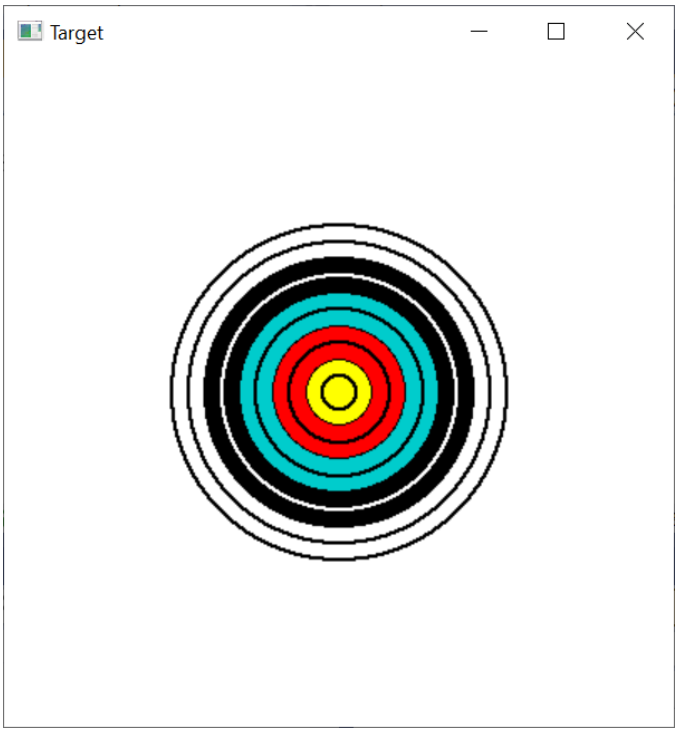
Source.cpp

```
#include<iostream>
#include <GL/glut.h>
using namespace std;

int xc = 0, yc = 0;

void plot(int x, int y)
{
	glBegin(GL_POINTS);
	glVertex2i(x + xc, y + yc);
	glEnd();
}

void myInit(void)
{
	glClearColor(1.0, 1.0, 1.0, 0.0);
	glColor3f(0.0f, 0.0f, 0.0f);
	glPointSize(10.0);
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	gluOrtho2D(-200.0, 200.0, -200.0, 200.0);
}
```

```
void midPointCircleAlgorithm(float r)
{
	int x = 0;
	int y = r;
	float decision = 1 - r;
	plot(x, y);

	while (y > x)
	{
		if (decision < 0)
		{
			x++;
			decision += 2 * x + 1;
		}
		else
		{
			y--;
			x++;
			decision += 2 * (x - y) + 1;
		}
		plot(x, y);
		plot(x, -y);
		plot(-x, y);
		plot(-x, -y);
		plot(y, x);
		plot(-y, x);
		plot(y, -x);
		plot(-y, -x);
	}

}

void myDisplay(void)
{
	glClear(GL_COLOR_BUFFER_BIT);
	glPointSize(2);

	//yellow inner circle
	glColor3f(1.0, 1.0, 0.0);
	for (int i = 1; i < 20; i++) {
		midPointCircleAlgorithm(i);
	}
	//black outline
	glColor3f(0.0, 0.0, 0.0);
	midPointCircleAlgorithm(10);
	midPointCircleAlgorithm(20);

	//red inner circle
	glColor3f(1.0, 0.0, 0.0);
	for (int i = 21; i < 40; i++) {
		midPointCircleAlgorithm(i);
	}
	//black outline
	glColor3f(0.0, 0.0, 0.0);
	midPointCircleAlgorithm(30);
	midPointCircleAlgorithm(40);

	//blue inner circle
	glColor3f(0.0, 0.8, 0.8);
	for (int i = 41; i < 60; i++) {
		midPointCircleAlgorithm(i);
	}
```

```
	//black outline
	glColor3f(0.0, 0.0, 0.0);
	midPointCircleAlgorithm(50);
	midPointCircleAlgorithm(60);

	//black inner circle
	glColor3f(0.0, 0.0, 0.0);
	for (int i = 61; i < 80; i++) {
		midPointCircleAlgorithm(i);
	}
	//white outline
	glColor3f(1.0, 1.0, 1.0);
	midPointCircleAlgorithm(70);

	//black outline
	glColor3f(0.0, 0.0, 0.0);
	midPointCircleAlgorithm(80);
	midPointCircleAlgorithm(90);
	midPointCircleAlgorithm(100);

	glFlush();
}

void main(int argc, char** argv)
{
	glutInit(&argc, argv);
	glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
	glutInitWindowSize(400, 400);
	glutInitWindowPosition(100, 150);
	glutCreateWindow("Target");
	glutDisplayFunc(myDisplay);
	myInit();
	glutMainLoop();
}
```

Output Screenshot



Result

Thus, C++ program using OPENGL to replicate the target board with the help of Midpoint Circle algorithm by using the necessary colours and elements to show details.

SSN COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UCS1712 – GRAPHICS AND MULTIMEDIA LAB

EX NO: 5a – 2D Transformations – Translation, Rotation and Scaling

Name : **Lakshmi Priya B**
Register Number : **185001083**

Date : **19-08-2021**

1.

Aim

To write a C++ menu-driven program using OPENGL to perform 2D transformations – translation, rotation, scaling for line and polygon.

Algorithm

1. Initialize using the glutInit() function with &argc and argv as parameters
2. Set the display mode as GLUT_SINGLE and specify the color scheme as GLUT_RGB
3. Specify the window position as 50, 50
4. Specify the window size as 600, 600
5. Create a new window and set the title as “2D Transformation”
6. Create a class point2D which has x, y as its attributes
7. Define init() function:
    a. Set the background color as white
8. Define winReshapeFcn(newWidth, newHeight)
    a. Set glMatrixMode(GL_PROJECTION)
    b. Specify the display area
    c. Display the window as the color of the current buffer using GL_COLOR_BUFFER_BIT
9. Define the matrix3x3SetIdentity(matIdet3x3) function:
    a. Set the diagonal elements of 3x3 matrix as 1
    b. Set the off-diagonal elements of 3x3 matrix as 0
10. Define matrix3x3PreMultiple(m1, m2) function:
    a. Multiply the two matrices m1 and m2
    b. Store the resultant matrix in m2
11. Define translate2D(tx, ty) function:
    a. Initialize matTransl as identity matrix using matrix3x3SetIdentity() function
    b. Set matTransl[0][2] = tx
    c. Set matTransl[1][2] = ty
    d. Call matrix3x3PreMultiply(matTransl, matComposite)
12. Define rotate2D(pivotPt, theta) function:
    a. Initialize matRot as identity matrix using matrix3x3SetIdentity() function
    b. Set matRot[0][0] = cos(theta)
    c. Set matRot[0][1] = -sin(theta)
    d. Set matRot[0][2] = pivotPt.x * (1 - cos(theta)) + pivotPt.y * sin(theta)

e. Set matRot[1][0] = sin(theta)
f. Set matRot[1][1] = cos(theta)
g. Set matRot[1][2] = pivotPt.y * (1 - cos(theta)) - pivotPt.x * sin(theta)
h. Call matrix3x3PreMultiply(matRot, matComposite)

13. Define scale2D(sx, sy, fixedPt) function:
    a. Initialize matScale as identity matrix using matrix3x3SetIdentity() function
    b. Set matScale[0][0] = sx
    c. Set matScale[0][2] = (1 - sx) * fixedPt.x
    d. Set matScale[1][1] = sy
    e. Set matScale[1][2] = (1 - sy) * fixedPt.y;
    f. Call matrix3x3PreMultiply(matScale, matComposite)
14. Define transformVerts2D(nVerts, verts, plotverts) function:
    a. Multiply the original object vertices and multiplication factors present in matComposite as per homogeneous coordinate representations
    b. Obtain the new transformed vertices of the object
    c. Store the new vertices in plotverts
15. Define drawObject(nVerts, verts) function:
    a. Draw the polygon using GL_POLYGON
        i. For k: 0 to nVerts-1
            1. Plot (verts[k].x, verts[k].y)
    b. Set color as black for the object outline
    c. Draw the outline using GL_LINE_LOOP
        i. For k: 0 to nVerts-1
            1. Plot (verts[k].x, verts[k].y)
16. Define the displayFcn() function:
    a. Declare nVerts, verts, plotverts
    b. Loop 1:
        i. Get number of vertices as input from user
        ii. Get coordinates of the vertices as input from the user
        iii. Display the window as the color of the current buffer using GL_COLOR_BUFFER_BIT
        iv. Draw the x-axis and y-axis for reference using GL_LINES by specifing the coordinates
        v. Draw the original object using drawObject(nVerts, verts)
        vi. Send all the output to display using glFlush()
        vii. Loop 2:
            1. Compute centroid
            2. Set pivot point and fixed point as centroid
            3. Call matrix3x3SetIdentity(matrixComposite)
            4. Display the menu of transformations available
            5. Get the option from the user
            6. If option is 1:
                a. Get translation along x as input
                b. Get translation along y as input
                c. Execute translate2D(tx, ty)
            7. Else if option is 2:
                a. Get rotation angle (theta) in degrees as input
                b. Execute rotate2D(pivPt, theta * pi / 180)

8. Else if option is 3:
    a. Get scaling factor along x as input
    b. Get scaling factor along y as input
    c. Execute scale2D(sx, sy, fixedPt)
9. Else if option is 4:
    a. Continue with Loop 1
10. Execute transformVerts2D(nVerts, verts, plotverts)
11. Execute drawObject(nVerts, plotVerts)
12. Send all the output to display using glFlush()

17. Define main() function:
    a. Call init() function
    b. Call glutDisplayFunc(displayFcn)
    c. Call glutReshapeFunc(winReshapeFcn)
    d. Go into a loop using glutMainLoop() until event occurs
18. Run the application.
19. Enter number of vertices in the object as input.
20. Either select the appropriate transformation to be applied like translation, rotation or scaling by choosing the appropriate menu or choose to input coordinates of a new object.
21. View the original object and the transformed object after providing appropriate input for transformation operation.

Code

Source.cpp:

```
#include <GL/glut.h>
#include <math.h>
#include <iostream>
using namespace std;

GLsizei winWidth = 600, winHeight = 600;
GLfloat xwcMin = -200.0, xwcMax = 200.0;
GLfloat ywcMin = -200.0, ywcMax = 200.0;

class point2d
{
	public: GLfloat x, y;
};

typedef GLfloat Matrix3x3[3][3];
Matrix3x3 matComposite;
const GLdouble pi = 3.14159;

void init(void)
{
	glClearColor(1.0, 1.0, 1.0, 0.0);
}

void matrix3x3SetIdentity(Matrix3x3 matIdent3x3)
{
	GLint row, col;
	for (row = 0; row < 3; row++)
		for (col = 0; col < 3; col++)
			matIdent3x3[row][col] = (row == col);
}
```

```
void matrix3x3PreMultiply(Matrix3x3 m1, Matrix3x3 m2)
{
	GLint row, col;
	Matrix3x3 matTemp;
	for (row = 0; row < 3; row++)
		for (col = 0; col < 3; col++)
			matTemp[row][col] = m1[row][0] * m2[0][col] + m1[row][1] *
m2[1][col] + m1[row][2] * m2[2][col];
	for (row = 0; row < 3; row++)
		for (col = 0; col < 3; col++)
			m2[row][col] = matTemp[row][col];
}

void translate2D(GLfloat tx, GLfloat ty)
{
	Matrix3x3 matTransl;
	matrix3x3SetIdentity(matTransl);
	matTransl[0][2] = tx;
	matTransl[1][2] = ty;
	matrix3x3PreMultiply(matTransl, matComposite);
}

void rotate2D(point2d pivotPt, GLfloat theta)
{
	Matrix3x3 matRot;
	matrix3x3SetIdentity(matRot);
	matRot[0][0] = cos(theta);
	matRot[0][1] = -sin(theta);
	matRot[0][2] = pivotPt.x * (1 - cos(theta)) + pivotPt.y * sin(theta);
	matRot[1][0] = sin(theta);
	matRot[1][1] = cos(theta);
	matRot[1][2] = pivotPt.y * (1 - cos(theta)) - pivotPt.x * sin(theta);
	matrix3x3PreMultiply(matRot, matComposite);
}

void scale2D(GLfloat sx, GLfloat sy, point2d fixedPt)
{
	Matrix3x3 matScale;
	matrix3x3SetIdentity(matScale);
	matScale[0][0] = sx;
	matScale[0][2] = (1 - sx) * fixedPt.x;
	matScale[1][1] = sy;
	matScale[1][2] = (1 - sy) * fixedPt.y;
	matrix3x3PreMultiply(matScale, matComposite);
}

void transformVerts2D(GLint nVerts, point2d* verts, point2d* plotverts)
{
	GLint k;
	GLfloat temp;
	for (k = 0; k < nVerts; k++)
	{
		temp = matComposite[0][0] * verts[k].x + matComposite[0][1] * verts[k].y
+ matComposite[0][2];
		plotverts[k].y = matComposite[1][0] * verts[k].x + matComposite[1][1] *
verts[k].y + matComposite[1][2];
		plotverts[k].x = temp;
	}
}
```

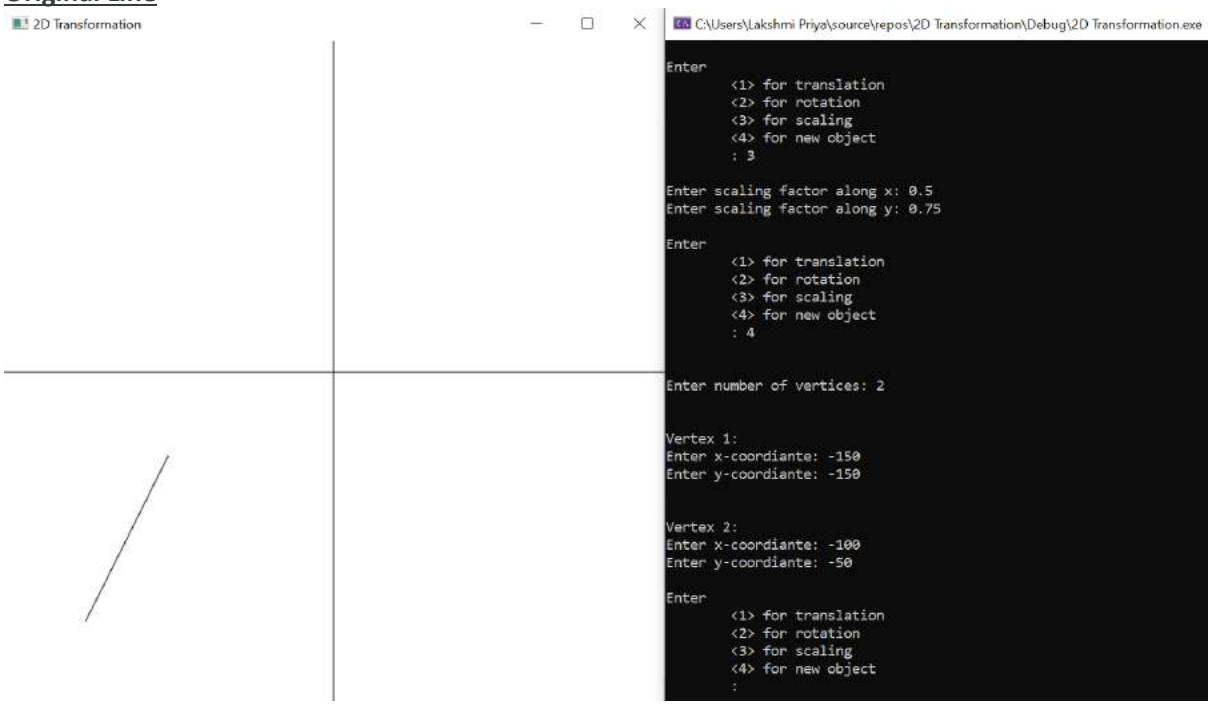
```
void drawObject(GLint nVerts, point2d* verts)
{
	GLint k;
	glBegin(GL_POLYGON);
	for (k = 0; k < nVerts; k++)
		glVertex2f(verts[k].x, verts[k].y);
	glEnd();

	glColor3f(0.0, 0.0, 0.0);
	glBegin(GL_LINE_LOOP);
	for (k = 0; k < nVerts; k++)
		glVertex2f(verts[k].x, verts[k].y);
	glEnd();

}

void displayFcn(void)
{
	GLint nVerts;
	point2d verts[10]; // = { {50, 50}, {25, 100}, {50, 150} ,{75, 100} };
	point2d plotverts[10];

	while (true) {

		cout << "\n\nEnter number of vertices: ";
		cin >> nVerts;

		for (int i = 0; i < nVerts; i++) {
			cout << "\n\nVertex " << i + 1 << ": " << endl;
			cout << "Enter x-coordiante: ";
			cin >> verts[i].x;
			cout << "Enter y-coordiante: ";
			cin >> verts[i].y;
		}

		GLfloat tx, ty;
		GLfloat sx, sy;
		GLdouble theta;
		glClear(GL_COLOR_BUFFER_BIT);

		glColor3f(0.0, 0.0, 0.0);
		glBegin(GL_LINES);
		glVertex2d(0, -200);
		glVertex2d(0, 200);
		glVertex2d(-200, 0);
		glVertex2d(200, 0);
		glEnd();

		glColor3f(1.0, 0.0, 0.0);
		drawObject(nVerts, verts);
		glFlush();

		int opt;
		while (true) {

			point2d centroidPt;
			GLint k, xSum = 0, ySum = 0;
			for (k = 0; k < nVerts; k++)
			{
				xSum += verts[k].x;
				ySum += verts[k].y;
			}
```

```
                        centroidPt.x = GLfloat(xSum) / GLfloat(nVerts);
                        centroidPt.y = GLfloat(ySum) / GLfloat(nVerts);
                        point2d pivPt, fixedPt;
                        pivPt = centroidPt;
                        fixedPt = centroidPt;

                        matrix3x3SetIdentity(matComposite);

                        cout << "\nEnter\n\t<1> for translation" <<
                                "\n\t<2> for rotation" <<
                                "\n\t<3> for scaling"<<
                                "\n\t<4> for new object\n\t: ";
                        cin >> opt;

                        switch (opt) {
                        case 1:
                                cout << "\nEnter translation along x: ";
                                cin >> tx;
                                cout << "Enter translation along y: ";
                                cin >> ty;
                                translate2D(tx, ty);
                                glColor3f(0.0, 1.0, 0.0);
                                break;
                        case 2:
                                cout << "\nEnter rotation angle (in degrees): ";
                                cin >> theta;
                                rotate2D(pivPt, theta * pi / 180);
                                glColor3f(0.0, 0.0, 1.0);
                                break;
                        case 3:
                                cout << "\nEnter scaling factor along x: ";
                                cin >> sx;
                                cout << "Enter scaling factor along y: ";
                                cin >> sy;
                                scale2D(sx, sy, fixedPt);
                                glColor3f(1.0, 1.0, 0.0);
                                break;
                        case 4:
                                break;
                        default:
                                cout << "Enter valid option!!" << endl;
                                continue;
                        }

                        if (opt == 4)
                                break;

                        transformVerts2D(nVerts, verts, plotverts);
                        drawObject(nVerts, plotverts);
                        glFlush();
                }
        }
}

void winReshapeFcn(GLint newWidth, GLint newHeight)
{
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(xwcMin, xwcMax, ywcMin, ywcMax);
        glClear(GL_COLOR_BUFFER_BIT);
}
```

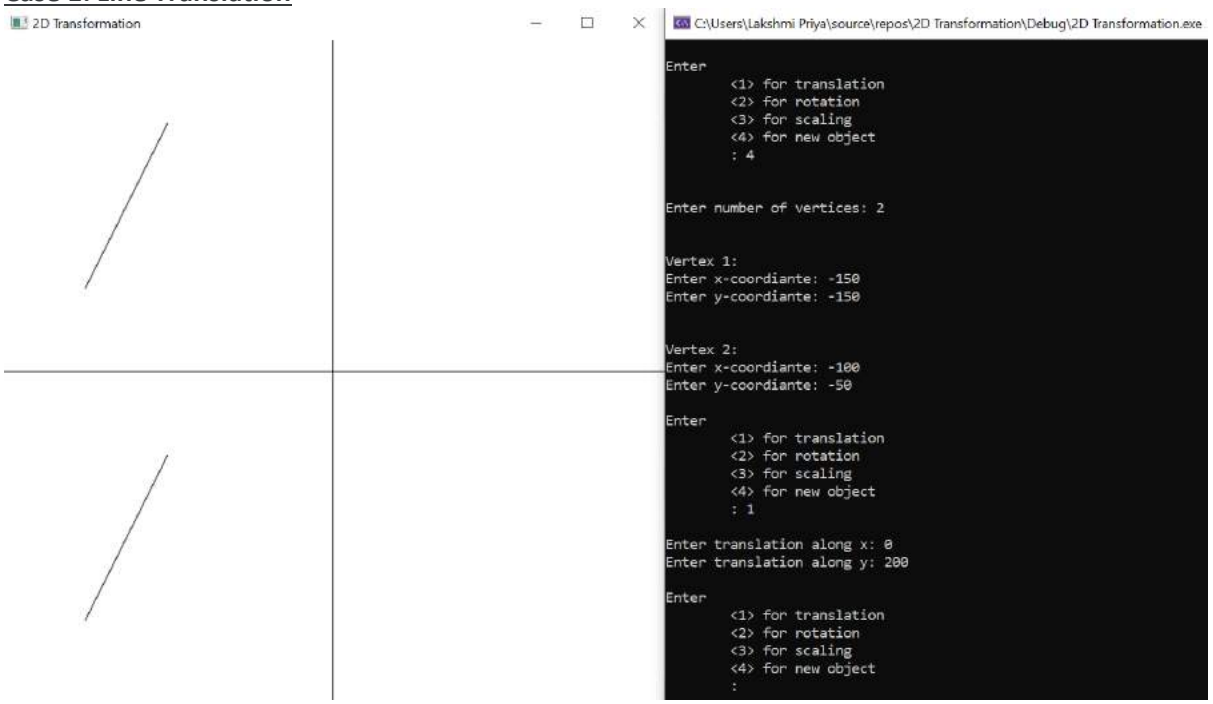
```
int main(int argc, char** argv)
{
	glutInit(&argc, argv);
	glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
	glutInitWindowPosition(50, 50);
	glutInitWindowSize(winWidth, winHeight);
	glutCreateWindow("2D Transformation");
	init();
	glutDisplayFunc(displayFcn);
	glutReshapeFunc(winReshapeFcn);
	glutMainLoop();
	return 0;
}
```

Output Screenshot

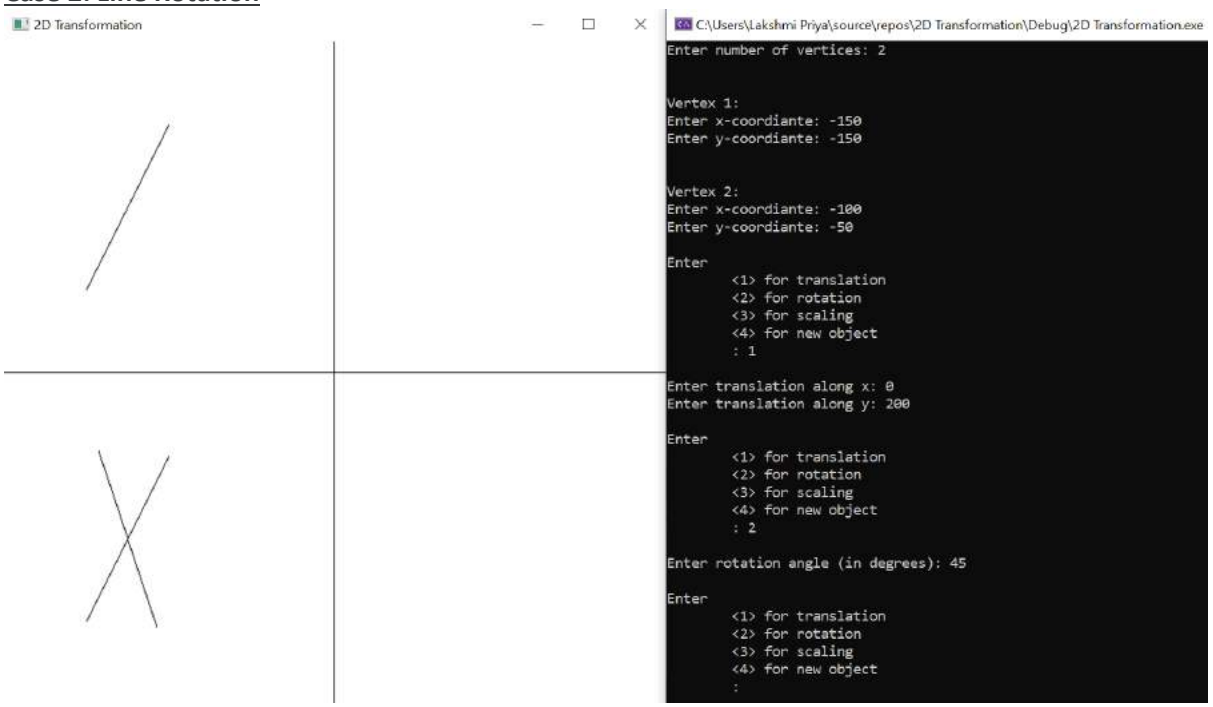
Original Line



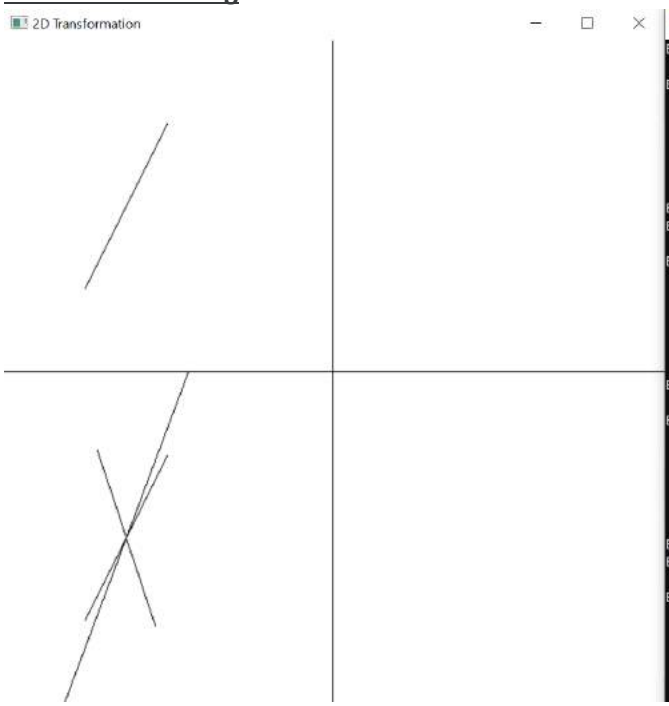
Case 1: Line Translation



Case 2: Line Rotation

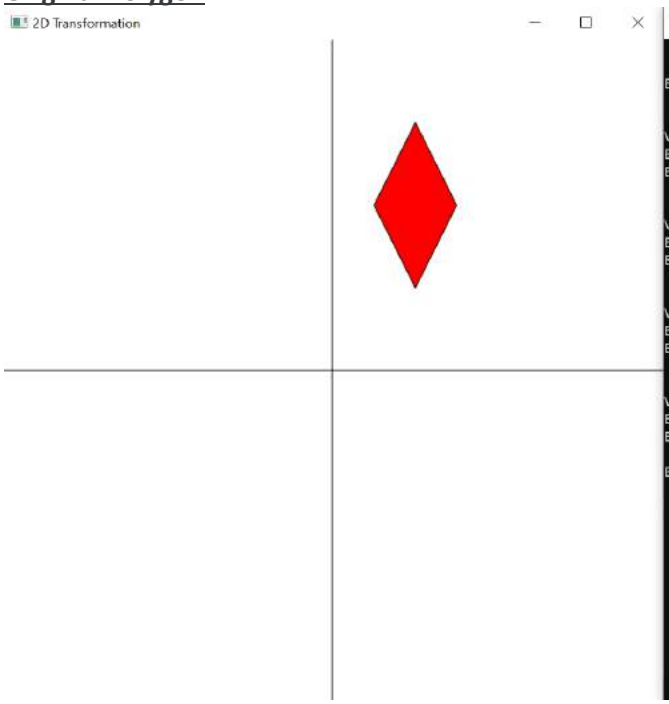


Case 3: Line Scaling



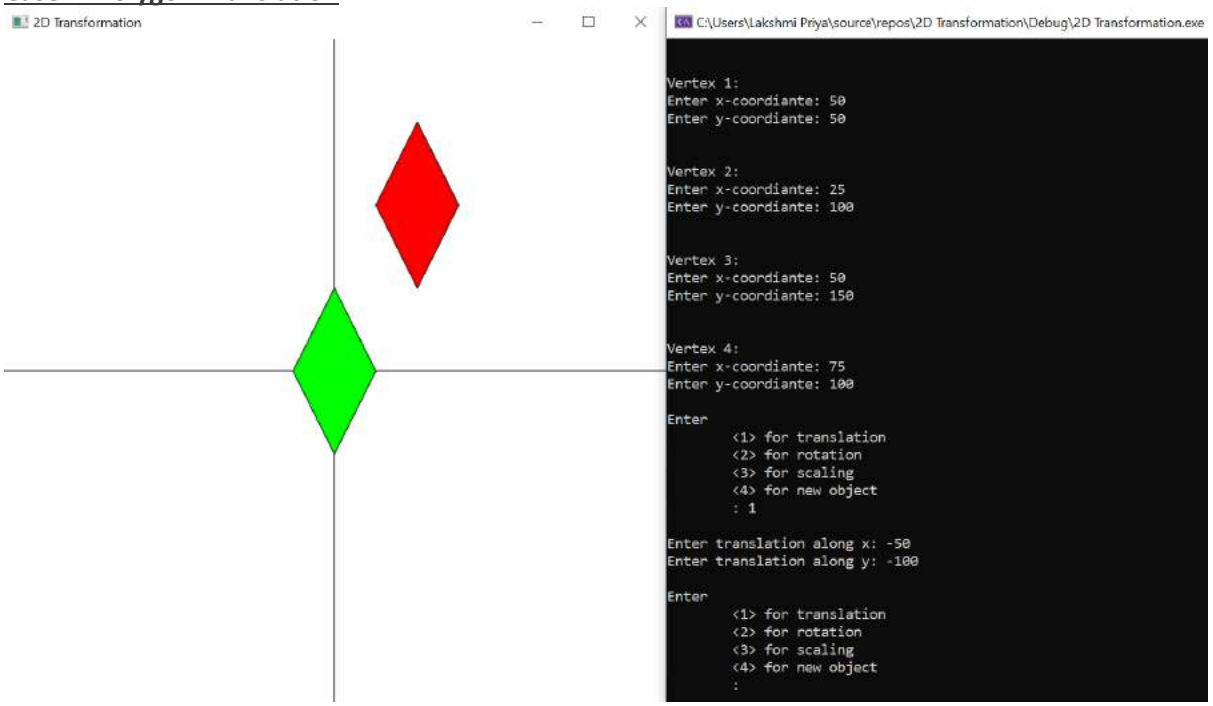

```
C:\Users\Lakshmi Priya\source\repos\2D Transformation\Debug\2D Transformation.exe
Enter y-coordiante: -50

Enter
        <1> for translation
        <2> for rotation
        <3> for scaling
        <4> for new object
        : 1

Enter translation along x: 0
Enter translation along y: 200

Enter
        <1> for translation
        <2> for rotation
        <3> for scaling
        <4> for new object
        : 2

Enter rotation angle (in degrees): 45

Enter
        <1> for translation
        <2> for rotation
        <3> for scaling
        <4> for new object
        : 3

Enter scaling factor along x: 1.5
Enter scaling factor along y: 2

Enter
        <1> for translation
        <2> for rotation
        <3> for scaling
        <4> for new object
        :
```

Original Polygon

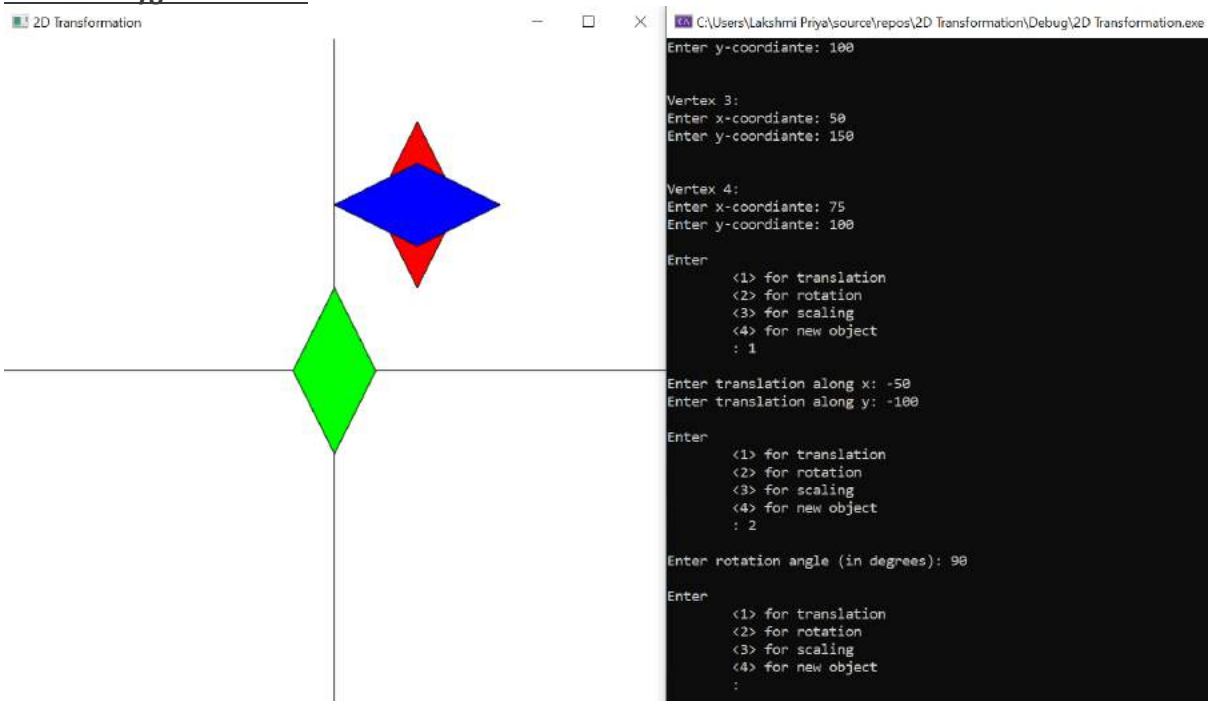



```
C:\Users\Lakshmi Priya\source\repos\2D Transformation\Debug\2D Transformation.exe

Enter number of vertices: 4


Vertex 1:
Enter x-coordiante: 50
Enter y-coordiante: 50


Vertex 2:
Enter x-coordiante: 25
Enter y-coordiante: 100


Vertex 3:
Enter x-coordiante: 50
Enter y-coordiante: 150


Vertex 4:
Enter x-coordiante: 75
Enter y-coordiante: 100

Enter
        <1> for translation
        <2> for rotation
        <3> for scaling
        <4> for new object
        :
```

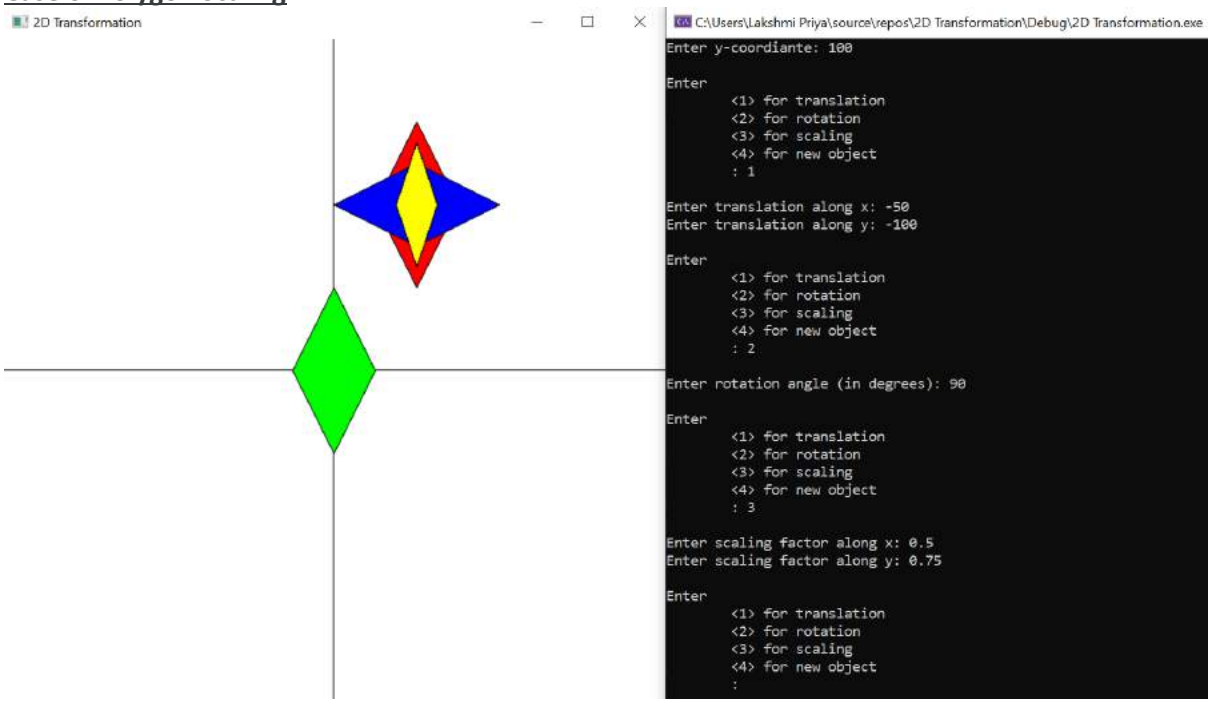
Case 4: Polygon Translation



Case 5: Polygon Rotation



Case 6: Polygon Scaling



Result

Thus, a C++ menu-driven program has been written using OPENGL to perform 2D transformations – translation, rotation, scaling for line and polygon.

SSN COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UCS1712 – GRAPHICS AND MULTIMEDIA LAB

EX NO: 5b – 2D Transformations – Reflection and Shearing

Name : **Lakshmi Priya B**
Register Number : **185001083**

Date : **26-08-2021**

Aim

To write a C++ menu-driven program using OPENGL to perform 2D transformations – reflection and shearing for polygons.

Algorithm

1. Initialize using the glutInit() function with &argc and argv as parameters
2. Set the display mode as GLUT_SINGLE and specify the color scheme as GLUT_RGB
3. Specify the window position as 50, 50
4. Specify the window size as 600, 600
5. Create a new window and set the title as “2D Transformation”
6. Create a class point2D which has x, y as its attributes
7. Define init() function:
    a. Set the background color as white
8. Define winReshapeFcn(newWidth, newHeight)
    a. Set glMatrixMode(GL_PROJECTION)
    b. Specify the display area
    c. Display the window as the color of the current buffer using GL_COLOR_BUFFER_BIT
9. Define the matrix3x3SetIdentity(matIdet3x3) function:
    a. Set the diagonal elements of 3x3 matrix as 1
    b. Set the off-diagonal elements of 3x3 matrix as 0
10. Define matrix3x3PreMultiple(m1, m2) function:
    a. Multiply the two matrices m1 and m2
    b. Store the resultant matrix in m2
11. Define reflection_x() function:
    a. Initialize matRef as identity matrix using matrix3x3SetIdentity() function
    b. Set matRef[1][1] = -1
    c. Call matrix3x3PreMultiply(matRef, matComposite)
12. Define reflection_y() function:
    a. Initialize matRef as identity matrix using matrix3x3SetIdentity() function
    b. Set matRef[0][0] = -1
    c. Call matrix3x3PreMultiply(matRef, matComposite)
13. Define reflection_origin() function:
    a. Initialize matRef as identity matrix using matrix3x3SetIdentity() function

    b. Setre matRef[0][0] = -1
    c. Set matRef[1][1] = -1
    d. Call matrix3x3PreMultiply(matRef, matComposite)
14. Define reflection_xy() function:
    a. Initialize matRef as identity matrix using matrix3x3SetIdentity() function
    b. Set matRef[0][0] = 0
    c. Set matRef[0][1] = 1
    d. Set matRef[1][0] = 1
    e. Set matRef[1][1] = 0
    f. Call matrix3x3PreMultiply(matRef, matComposite)
15. Define shearing_x(sf) function:
    a. Initialize matShear as identity matrix using matrix3x3SetIdentity() function
    b. Set matShear[0][1] = sf
    c. Call matrix3x3PreMultiply(matShear, matComposite)
16. Define shearing_y(sf) function:
    a. Initialize matShear as identity matrix using matrix3x3SetIdentity() function
    b. Set matShear[1][0] = sf
    c. Call matrix3x3PreMultiply(matShear, matComposite)
17. Define transformVerts2D(nVerts, verts, plotverts) function:
    a. Multiply the original object vertices and multiplication factors present in matComposite as per homogeneous coordinate representations
    b. Obtain the new transformed vertices of the object
    c. Store the new vertices in plotverts
18. Define drawObject(nVerts, verts) function:
    a. Draw the polygon using GL_POLYGON
        i. For k: 0 to nVerts-1
            1. Plot (verts[k].x, verts[k].y)
    b. Set color as black for the object outline
    c. Draw the outline using GL_LINE_LOOP
        i. For k: 0 to nVerts-1
            1. Plot (verts[k].x, verts[k].y)
19. Define the displayFcn() function:
    a. Declare nVerts, verts, plotverts
    b. Loop 1:
        i. Get number of vertices as input from user
        ii. Get coordinates of the vertices as input from the user
        iii. Display the window as the color of the current buffer using GL_COLOR_BUFFER_BIT
        iv. Draw the x-axis and y-axis for reference using GL_LINES by specifing the coordinates
        v. Draw the original object using drawObject(nVerts, verts)
        vi. Send all the output to display using glFlush()
        vii. Loop 2:
            1. Compute centroid
            2. Set pivot point and fixed point as centroid
            3. Call matrix3x3SetIdentity(matrixComposite)
            4. Display the menu of transformations available
            5. Get the option from the user

6. If option is 1:
    a. Execute reflection_x()
7. Else if option is 2:
    a. Execute reflection_y()
8. Else if option is 3:
    a. Execute reflection_origin()
9. Else if option is 4:
    a. Execute reflection_xy()
10. Else if option is 5:
    a. Get shearing factor along x as input
    b. Execute shearing_x(sf)
11. Else if option is 6:
    a. Get shearing factor along y as input
    b. Execute shearing_y(sf)
12. Else if option is 7:
    a. Continue with Loop 1
13. Execute transformVerts2D(nVerts, verts, plotverts)
14. Execute drawObject(nVerts, plotVerts)
15. Send all the output to display using glFlush()

20. Define main() function:
    a. Call init() function
    b. Call glutDisplayFunc(displayFcn)
    c. Call glutReshapeFunc(winReshapeFcn)
    d. Go into a loop using glutMainLoop() until event occurs
21. Run the application.
22. Enter number of vertices in the object as input.
23. Either select the appropriate transformation to be applied like reflection along x-axis, reflection along y-axis, reflection about origin, reflection about line x=y, shearing along x-axis or shearing along y-axis by choosing the appropriate menu or choose to input coordinates of a new object.
24. View the original object and the transformed object after providing appropriate input for transformation operation.

Code

Source.cpp:

```
#include <GL/glut.h>
#include <math.h>
#include <iostream>
using namespace std;

GLsizei winWidth = 600, winHeight = 600;
GLfloat xwcMin = -200.0, xwcMax = 200.0;
GLfloat ywcMin = -200.0, ywcMax = 200.0;

class point2d {
public: GLfloat x, y;
};

typedef GLfloat Matrix3x3[3][3];
Matrix3x3 matComposite;
const GLdouble pi = 3.14159;
```

```
void init(void)
{
	glClearColor(1.0, 1.0, 1.0, 0.0);
}

void matrix3x3SetIdentity(Matrix3x3 matIdent3x3)
{
	GLint row, col;
	for (row = 0; row < 3; row++)
		for (col = 0; col < 3; col++)
			matIdent3x3[row][col] = (row == col);
}

void matrix3x3PreMultiply(Matrix3x3 m1, Matrix3x3 m2)
{
	GLint row, col;
	Matrix3x3 matTemp;
	for (row = 0; row < 3; row++)
		for (col = 0; col < 3; col++)
			matTemp[row][col] = m1[row][0] * m2[0][col] + m1[row][1] *
m2[1][col] + m1[row][2] * m2[2][col];
	for (row = 0; row < 3; row++)
		for (col = 0; col < 3; col++)
			m2[row][col] = matTemp[row][col];
}

void reflection_x()
{
	Matrix3x3 matRef;
	matrix3x3SetIdentity(matRef);
	matRef[1][1] = -1;
	matrix3x3PreMultiply(matRef, matComposite);
}

void reflection_y()
{
	Matrix3x3 matRef;
	matrix3x3SetIdentity(matRef);
	matRef[0][0] = -1;
	matrix3x3PreMultiply(matRef, matComposite);
}

void reflection_origin()
{
	Matrix3x3 matRef;
	matrix3x3SetIdentity(matRef);
	matRef[0][0] = -1;
	matRef[1][1] = -1;
	matrix3x3PreMultiply(matRef, matComposite);
}

void reflection_xy()
{
	Matrix3x3 matRef;
	matrix3x3SetIdentity(matRef);
	matRef[0][0] = 0;
	matRef[0][1] = 1;
	matRef[1][0] = 1;
	matRef[1][1] = 0;
	matrix3x3PreMultiply(matRef, matComposite);
}
```

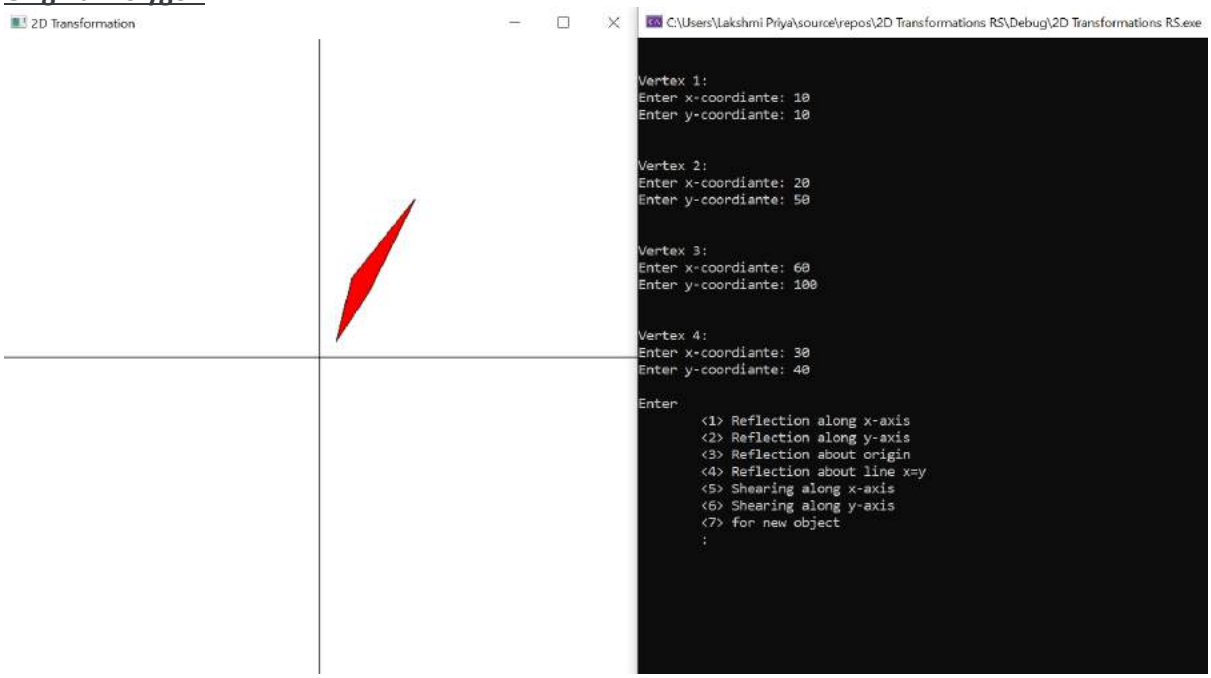
```
void shearing_x(float sf)
{
	Matrix3x3 matShear;
	matrix3x3SetIdentity(matShear);
	matShear[0][1] = sf;
	matrix3x3PreMultiply(matShear, matComposite);
}

void shearing_y(float sf)
{
	Matrix3x3 matShear;
	matrix3x3SetIdentity(matShear);
	matShear[1][0] = sf;
	matrix3x3PreMultiply(matShear, matComposite);
}

void transformVerts2D(GLint nVerts, point2d* verts, point2d* plotverts)
{
	GLint k;
	GLfloat temp;
	for (k = 0; k < nVerts; k++)
	{
		temp = matComposite[0][0] * verts[k].x + matComposite[0][1] * verts[k].y
+ matComposite[0][2];
		plotverts[k].y = matComposite[1][0] * verts[k].x + matComposite[1][1] *
verts[k].y + matComposite[1][2];
		plotverts[k].x = temp;
	}
}

void drawObject(GLint nVerts, point2d* verts)
{
	GLint k;
	glBegin(GL_POLYGON);
	for (k = 0; k < nVerts; k++)
		glVertex2f(verts[k].x, verts[k].y);
	glEnd();

	glColor3f(0.0, 0.0, 0.0);
	glBegin(GL_LINE_LOOP);
	for (k = 0; k < nVerts; k++)
		glVertex2f(verts[k].x, verts[k].y);
	glEnd();

}

void displayFcn(void)
{
	GLint nVerts;// = 3;
	point2d verts[10];// = { {10, 10}, {20, 50}, {30, 40} };//{ {50, 50}, {25,
100}, {50, 150} ,{100, 125}, {100, 75} };
	point2d plotverts[10];

	while (true) {

		cout << "\n\nEnter number of vertices: ";
		cin >> nVerts;

		for (int i = 0; i < nVerts; i++) {
			cout << "\n\nVertex " << i + 1 << ": " << endl;
			cout << "Enter x-coordiante: ";
			cin >> verts[i].x;
```

```
        cout << "Enter y-coordiante: ";
        cin >> verts[i].y;
}

glClear(GL_COLOR_BUFFER_BIT);

glColor3f(0.0, 0.0, 0.0);
glBegin(GL_LINES);
glVertex2d(0, -200);
glVertex2d(0, 200);
glVertex2d(-200, 0);
glVertex2d(200, 0);
glEnd();

glColor3f(1.0, 0.0, 0.0);
drawObject(nVerts, verts);
glFlush();

int opt;
float sf;

while (true) {

        matrix3x3SetIdentity(matComposite);

        cout << "\nEnter" <<
                "\n\t<1> Reflection along x-axis" <<
                "\n\t<2> Reflection along y-axis" <<
                "\n\t<3> Reflection about origin" <<
                "\n\t<4> Reflection about line x=y" <<
                "\n\t<5> Shearing along x-axis" <<
                "\n\t<6> Shearing along y-axis" <<
                "\n\t<7> for new object" <<
                "\n\t: ";
        cin >> opt;

        switch (opt) {
        case 1:
                reflection_x();
                glColor3f(0.0, 1.0, 0.0);
                break;

        case 2:
                reflection_y();
                glColor3f(0.0, 0.0, 1.0);
                break;

        case 3:
                reflection_origin();
                glColor3f(1.0, 1.0, 0.0);
                break;

        case 4:
                reflection_xy();
                glColor3f(0.0, 1.0, 1.0);
                break;

        case 5:
                cout << "Enter shearing factor: ";
                cin >> sf;

                shearing_x(sf);
```

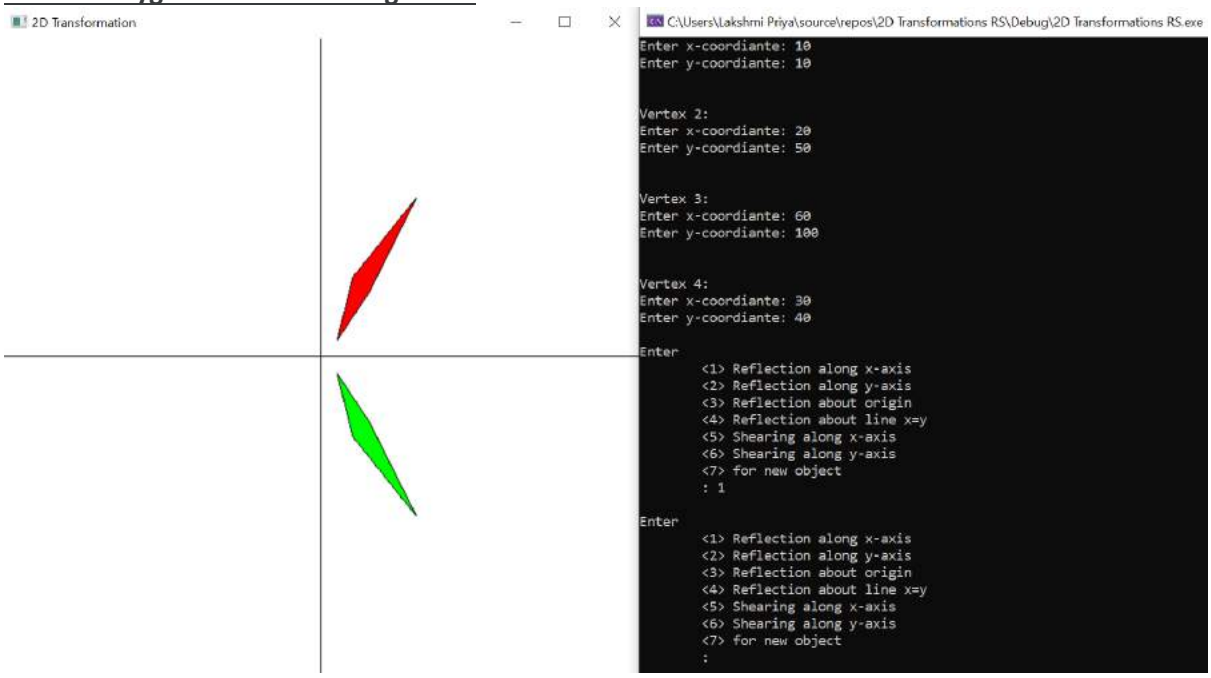
```
                        glColor3f(1.0, 0.0, 1.0);
                        break;

                case 6:
                        cout << "Enter shearing factor: ";
                        cin >> sf;

                        shearing_y(sf);
                        glColor3f(1.0, 0.0, 1.0);
                        break;

                case 7:
                        break;

                default:
                        cout << "Enter valid option!!" << endl;
                        continue;
                }

                if (opt == 7)
                        break;

                transformVerts2D(nVerts, verts, plotverts);
                drawObject(nVerts, plotverts);
                glFlush();
        }
    }
}

void winReshapeFcn(GLint newWidth, GLint newHeight)
{
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(xwcMin, xwcMax, ywcMin, ywcMax);
    glClear(GL_COLOR_BUFFER_BIT);
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowPosition(50, 50);
    glutInitWindowSize(winWidth, winHeight);
    glutCreateWindow("2D Transformation");
    init();
    glutDisplayFunc(displayFcn);
    glutReshapeFunc(winReshapeFcn);
    glutMainLoop();
    return 0;
}
```

Output Screenshot

Original Polygon

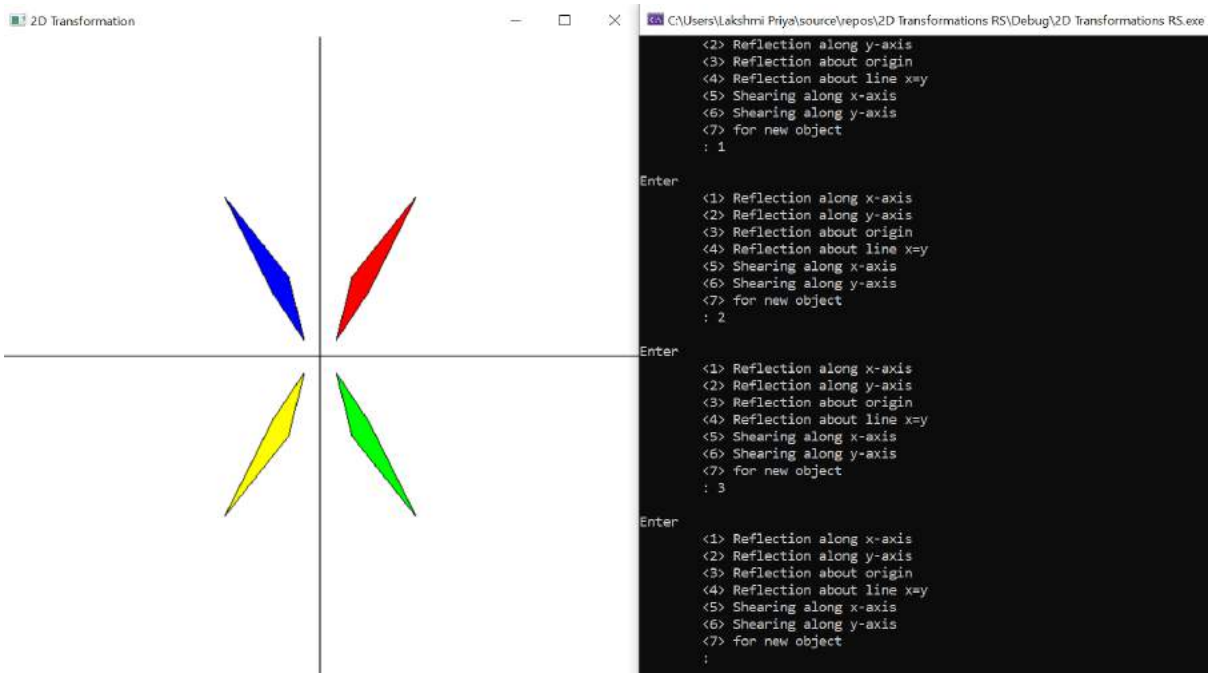


Case 1: Polygon Reflection along x-axis

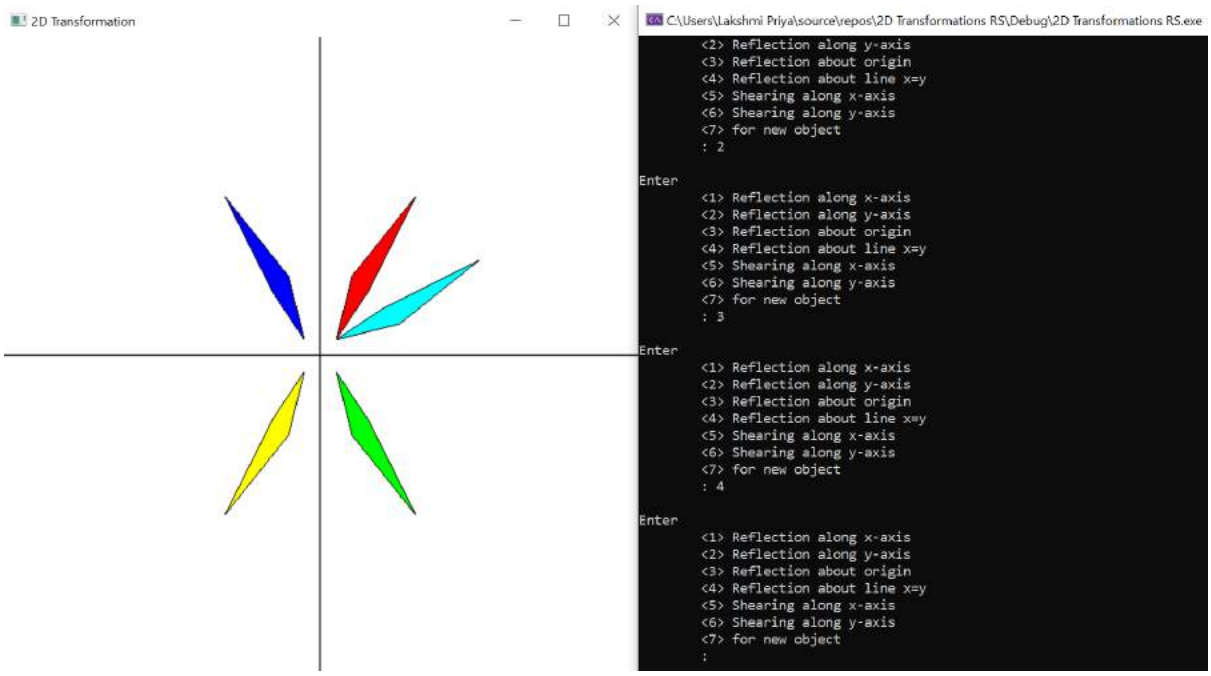


Case 2: Polygon Reflection along y-axis

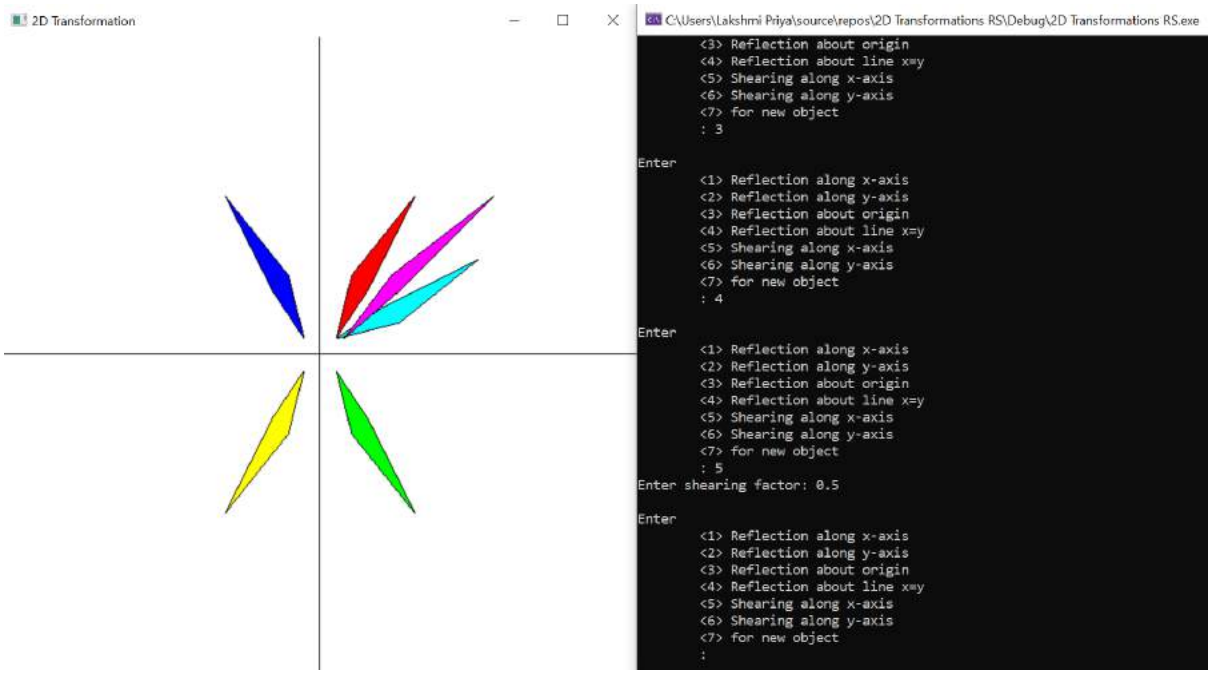
Case 3: Polygon Reflection about origin



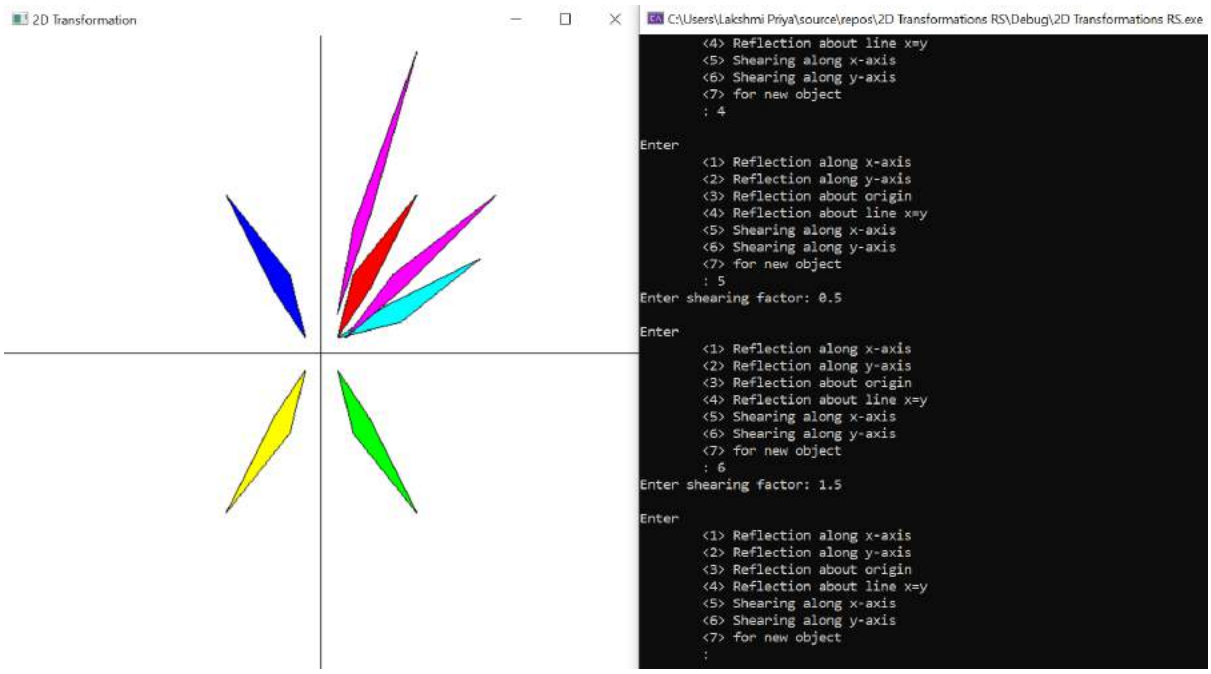
Case 4: Polygon Reflection about line x=y



Case 5: Polygon Shearing along x-axis



Case 6: Polygon Shearing along y-axis



Result

Thus, a C++ menu-driven program has been written using OPENGL to perform 2D transformations like reflection and shearing for polygons.

SSN COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UCS1712 – GRAPHICS AND MULTIMEDIA LAB

EX NO: 6a – 2D Transformations – Composite Transformation

Name : **Lakshmi Priya B**
Register Number : **185001083**
Date : **03-09-2021**

Aim

To write a C++ program using OPENGL to perform 2D composite transformations involving translation, rotation, scaling, reflection and shearing for input object.

Algorithm

1. Initialize using the glutInit() function with &argc and argv as parameters
2. Set the display mode as GLUT_SINGLE and specify the color scheme as GLUT_RGB
3. Specify the window position as 50, 50
4. Specify the window size as 600, 600
5. Create a new window and set the title as “2D Composite Transformation”
6. Create a class point2D which has x, y as its attributes
7. Define init() function:
    a. Set the background color as white
8. Define winReshapeFcn(newWidth, newHeight)
    a. Set glMatrixMode(GL_PROJECTION)
    b. Specify the display area
    c. Display the window as the color of the current buffer using GL_COLOR_BUFFER_BIT
9. Define the matrix3x3SetIdentity(matIdet3x3) function:
    a. Set the diagonal elements of 3x3 matrix as 1
    b. Set the off-diagonal elements of 3x3 matrix as 0
10. Define matrix3x3PreMultiple(m1, m2) function:
    a. Multiply the two matrices m1 and m2
    b. Store the resultant matrix in m2
11. Define translate2D(tx, ty) function:
    a. Initialize matTransl as identity matrix using matrix3x3SetIdentity() function
    b. Set matTransl[0][2] = tx
    c. Set matTransl[1][2] = ty
    d. Call matrix3x3PreMultiply(matTransl, matComposite)
12. Define rotate2D(pivotPt, theta) function:
    a. Initialize matRot as identity matrix using matrix3x3SetIdentity() function
    b. Set matRot[0][0] = cos(theta)
    c. Set matRot[0][1] = -sin(theta)
    d. Set matRot[0][2] = pivotPt.x * (1 - cos(theta)) + pivotPt.y * sin(theta)

e. Set matRot[1][0] = sin(theta)
f. Set matRot[1][1] = cos(theta)
g. Set matRot[1][2] = pivotPt.y * (1 - cos(theta)) - pivotPt.x * sin(theta)
h. Call matrix3x3PreMultiply(matRot, matComposite)

13. Define scale2D(sx, sy, fixedPt) function:
    a. Initialize matScale as identity matrix using matrix3x3SetIdentity() function
    b. Set matScale[0][0] = sx
    c. Set matScale[0][2] = (1 - sx) * fixedPt.x
    d. Set matScale[1][1] = sy
    e. Set matScale[1][2] = (1 - sy) * fixedPt.y;
    f. Call matrix3x3PreMultiply(matScale, matComposite)
14. Define reflection_x() function:
    a. Initialize matRef as identity matrix using matrix3x3SetIdentity() function
    b. Set matRef[1][1] = -1
    c. Call matrix3x3PreMultiply(matRef, matComposite)
15. Define reflection_y() function:
    a. Initialize matRef as identity matrix using matrix3x3SetIdentity() function
    b. Set matRef[0][0] = -1
    c. Call matrix3x3PreMultiply(matRef, matComposite)
16. Define reflection_origin() function:
    a. Initialize matRef as identity matrix using matrix3x3SetIdentity() function
    b. Setre matRef[0][0] = -1
    c. Set matRef[1][1] = -1
    d. Call matrix3x3PreMultiply(matRef, matComposite)
17. Define reflection_xy() function:
    a. Initialize matRef as identity matrix using matrix3x3SetIdentity() function
    b. Set matRef[0][0] = 0
    c. Set matRef[0][1] = 1
    d. Set matRef[1][0] = 1
    e. Set matRef[1][1] = 0
    f. Call matrix3x3PreMultiply(matRef, matComposite)
18. Define shearing_x(sf) function:
    a. Initialize matShear as identity matrix using matrix3x3SetIdentity() function
    b. Set matShear[0][1] = sf
    c. Call matrix3x3PreMultiply(matShear, matComposite)
19. Define shearing_y(sf) function:
    a. Initialize matShear as identity matrix using matrix3x3SetIdentity() function
    b. Set matShear[1][0] = sf
    c. Call matrix3x3PreMultiply(matShear, matComposite)
20. Define transformVerts2D(nVerts, verts) function:
    a. Multiply the original object vertices and multiplication factors present in matComposite as per homogeneous coordinate representations
    b. Obtain the new transformed vertices of the object
    c. Rewrite the vertices in verts to store the new vertices
21. Define drawObject(nVerts, verts) function:
    a. Draw the polygon using GL_POLYGON
        i. For k: 0 to nVerts-1
            1. Plot (verts[k].x, verts[k].y)

b. Set color as black for the object outline
c. Draw the outline using GL_LINE_LOOP
    i. For k: 0 to nVerts-1
        1. Plot (verts[k].x, verts[k].y)

22. Define the displayFcn() function:
    a. Declare nVerts, verts
    b. Loop 1:
        i. Get number of vertices as input from user
        ii. Get coordinates of the vertices as input from the user
        iii. Display the window as the color of the current buffer using GL_COLOR_BUFFER_BIT
        iv. Draw the x-axis and y-axis for reference using GL_LINES by specifing the coordinates
        v. Draw the original object using drawObject(nVerts, verts)
        vi. Send all the output to display using glFlush()
        vii. Loop 2:
            1. Compute centroid
            2. Set pivot point and fixed point as centroid
            3. Call matrix3x3SetIdentity(matrixComposite)
            4. Display the menu of transformations available
            5. Get the option from the user
            6. If option is 1:
                a. Get translation along x as input
                b. Get translation along y as input
                c. Execute translate2D(tx, ty)
            7. Else if option is 2:
                a. Get rotation angle (theta) in degrees as input
                b. Execute rotate2D(pivPt, theta * pi / 180)
            8. Else if option is 3:
                a. Get scaling factor along x as input
                b. Get scaling factor along y as input
                c. Execute scale2D(sx, sy, fixedPt)
            9. Else if option is 4:
                a. Execute reflection_x()
            10. Else if option is 5:
                a. Execute reflection_y()
            11. Else if option is 6:
                a. Execute reflection_origin()
            12. Else if option is 7:
                a. Execute reflection_xy()
            13. Else if option is 8:
                a. Get shearing factor along x as input
                b. Execute shearing_x(sf)
            14. Else if option is 9:
                a. Get shearing factor along y as input
                b. Execute shearing_y(sf)
            15. Else if option is 10:
                a. Continue with Loop 1

16. Execute transformVerts2D(nVerts, verts)
17. Execute drawObject(nVerts)
18. Send all the output to display using glFlush()

23. Define main() function:
    a. Call init() function
    b. Call glutDisplayFunc(displayFcn)
    c. Call glutReshapeFunc(winReshapeFcn)
    d. Go into a loop using glutMainLoop() until event occurs
24. Run the application.
25. Enter number of vertices in the object as input.
26. Either select the appropriate transformation to be applied like translation, rotation, scaling, reflection or shearing by choosing the appropriate menu or choose to input coordinates of a new object.
27. View the original object and the transformed object after providing appropriate input for transformation operation.
28. Repeat step 26 if composite transformation has to be applied on the transformed object successively.

Code

Source.cpp:

```
#include <GL/glut.h>
#include <math.h>
#include <iostream>
using namespace std;

GLsizei winWidth = 600, winHeight = 600;
GLfloat xwcMin = -200.0, xwcMax = 200.0;
GLfloat ywcMin = -200.0, ywcMax = 200.0;

class point2d
{
	public: GLfloat x, y;
};

typedef GLfloat Matrix3x3[3][3];
Matrix3x3 matComposite;
const GLdouble pi = 3.14159;


void init(void)
{
	glClearColor(1.0, 1.0, 1.0, 0.0);
}

void matrix3x3SetIdentity(Matrix3x3 matIdent3x3)
{
	GLint row, col;
	for (row = 0; row < 3; row++)
		for (col = 0; col < 3; col++)
			matIdent3x3[row][col] = (row == col);
}
```

```
void matrix3x3PreMultiply(Matrix3x3 m1, Matrix3x3 m2)
{
	GLint row, col;
	Matrix3x3 matTemp;
	for (row = 0; row < 3; row++)
		for (col = 0; col < 3; col++)
			matTemp[row][col] = m1[row][0] * m2[0][col] + m1[row][1] *
m2[1][col] + m1[row][2] * m2[2][col];
	for (row = 0; row < 3; row++)
		for (col = 0; col < 3; col++)
			m2[row][col] = matTemp[row][col];
}

void translate2D(GLfloat tx, GLfloat ty)
{
	Matrix3x3 matTransl;
	matrix3x3SetIdentity(matTransl);
	matTransl[0][2] = tx;
	matTransl[1][2] = ty;
	matrix3x3PreMultiply(matTransl, matComposite);
}

void rotate2D(point2d pivotPt, GLfloat theta)
{
	Matrix3x3 matRot;
	matrix3x3SetIdentity(matRot);
	matRot[0][0] = cos(theta);
	matRot[0][1] = -sin(theta);
	matRot[0][2] = pivotPt.x * (1 - cos(theta)) + pivotPt.y * sin(theta);
	matRot[1][0] = sin(theta);
	matRot[1][1] = cos(theta);
	matRot[1][2] = pivotPt.y * (1 - cos(theta)) - pivotPt.x * sin(theta);
	matrix3x3PreMultiply(matRot, matComposite);
}

void scale2D(GLfloat sx, GLfloat sy, point2d fixedPt)
{
	Matrix3x3 matScale;
	matrix3x3SetIdentity(matScale);
	matScale[0][0] = sx;
	matScale[0][2] = (1 - sx) * fixedPt.x;
	matScale[1][1] = sy;
	matScale[1][2] = (1 - sy) * fixedPt.y;
	matrix3x3PreMultiply(matScale, matComposite);
}

void reflection_x()
{
	Matrix3x3 matRef;
	matrix3x3SetIdentity(matRef);
	matRef[1][1] = -1;
	matrix3x3PreMultiply(matRef, matComposite);
}

void reflection_y()
{
	Matrix3x3 matRef;
	matrix3x3SetIdentity(matRef);
	matRef[0][0] = -1;
	matrix3x3PreMultiply(matRef, matComposite);
}
```

```
void reflection_origin()
{
	Matrix3x3 matRef;
	matrix3x3SetIdentity(matRef);
	matRef[0][0] = -1;
	matRef[1][1] = -1;
	matrix3x3PreMultiply(matRef, matComposite);
}

void reflection_xy()
{
	Matrix3x3 matRef;
	matrix3x3SetIdentity(matRef);
	matRef[0][0] = 0;
	matRef[0][1] = 1;
	matRef[1][0] = 1;
	matRef[1][1] = 0;
	matrix3x3PreMultiply(matRef, matComposite);
}

void shearing_x(float sf)
{
	Matrix3x3 matShear;
	matrix3x3SetIdentity(matShear);
	matShear[0][1] = sf;
	matrix3x3PreMultiply(matShear, matComposite);
}

void shearing_y(float sf)
{
	Matrix3x3 matShear;
	matrix3x3SetIdentity(matShear);
	matShear[1][0] = sf;
	matrix3x3PreMultiply(matShear, matComposite);
}

void transformVerts2D(GLint nVerts, point2d* verts)
{
	GLint k;
	GLfloat temp;
	GLfloat xmin = verts[0].x, xmax = verts[0].x, ymin = verts[0].y, ymax =
verts[0].y;
	for (k = 0; k < nVerts; k++)
	{
		temp = matComposite[0][0] * verts[k].x + matComposite[0][1] * verts[k].y
+ matComposite[0][2];
		verts[k].y = matComposite[1][0] * verts[k].x + matComposite[1][1] *
verts[k].y + matComposite[1][2];
		verts[k].x = temp;
	}
}

void drawObject(GLint nVerts, point2d* verts)
{
	GLint k;
	glBegin(GL_POLYGON);
	for (k = 0; k < nVerts; k++)
		glVertex2f(verts[k].x, verts[k].y);
	glEnd();

	glColor3f(0.0, 0.0, 0.0);
	glBegin(GL_LINE_LOOP);
```

```
	for (k = 0; k < nVerts; k++)
		glVertex2f(verts[k].x, verts[k].y);
	glEnd();

}

void displayFcn(void)
{
	GLint nVerts;
	point2d verts[10];// = { {50.0, 25.0}, {150.0, 25.0}, {100.0, 100.0} ,{50, 100}
};

	while (true) {

		cout << "\n\nEnter number of vertices: ";
		cin >> nVerts;

		for (int i = 0; i < nVerts; i++) {
			cout << "\n\nVertex " << i + 1 << ": " << endl;
			cout << "Enter x-coordiante: ";
			cin >> verts[i].x;
			cout << "Enter y-coordiante: ";
			cin >> verts[i].y;
		}

		GLfloat tx, ty;
		GLfloat sx, sy;
		GLdouble theta;
		glClear(GL_COLOR_BUFFER_BIT);

		glColor3f(0.0, 0.0, 0.0);
		glBegin(GL_LINES);
		glVertex2d(0, -200);
		glVertex2d(0, 200);
		glVertex2d(-200, 0);
		glVertex2d(200, 0);
		glEnd();

		glColor3f(1.0, 0.0, 0.0);
		drawObject(nVerts, verts);
		glFlush();

		int opt;
		float sf;

		while (true) {

			point2d pivPt, fixedPt;

			matrix3x3SetIdentity(matComposite);

			cout << "\nEnter\n\t<1> for translation" <<
				"\n\t<2> for rotation" <<
				"\n\t<3> for scaling" <<
				"\n\t<4> Reflection along x-axis" <<
				"\n\t<5> Reflection along y-axis" <<
				"\n\t<6> Reflection about origin" <<
				"\n\t<7> Reflection about line x=y" <<
				"\n\t<8> Shearing along x-axis" <<
				"\n\t<9> Shearing along y-axis" <<
				"\n\t<10> for new object\n\t: ";
			cin >> opt;
```

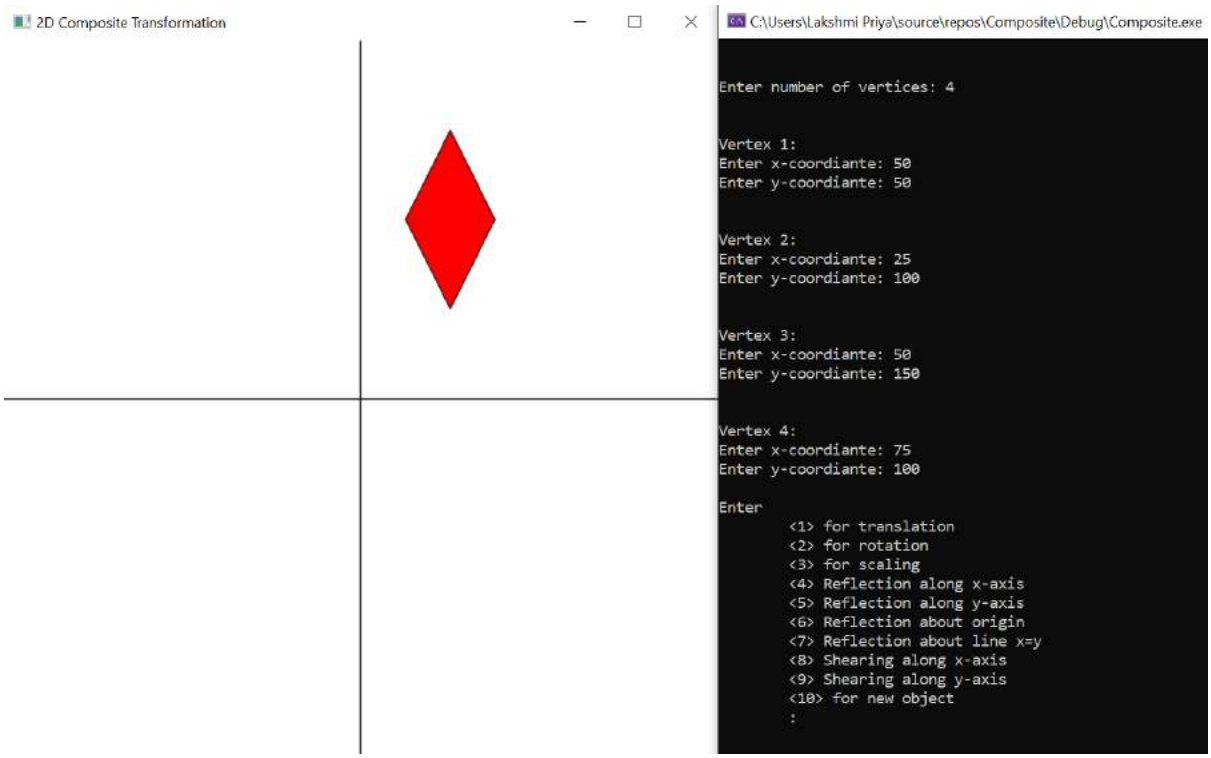
```
switch (opt) {
case 1:
       cout << "\nEnter translation along x: ";
       cin >> tx;
       cout << "Enter translation along y: ";
       cin >> ty;
       translate2D(tx, ty);
       glColor3f(0.0, 1.0, 0.0);
       break;

case 2:
       cout << "\nEnter rotation angle (in degrees): ";
       cin >> theta;
       cout << "\nEnter pivot point x-coordinate: ";
       cin >> pivPt.x;
       cout << "Enter pivot point y-coordinate: ";
       cin >> pivPt.y;
       rotate2D(pivPt, theta * pi / 180);
       glColor3f(0.0, 0.0, 1.0);
       break;

case 3:
       cout << "\nEnter scaling factor along x: ";
       cin >> sx;
       cout << "Enter scaling factor along y: ";
       cin >> sy;
       cout << "\nEnter fixed point x-coordinate: ";
       cin >> fixedPt.x;
       cout << "Enter fixed point y-coordinate: ";
       cin >> fixedPt.y;
       scale2D(sx, sy, fixedPt);
       glColor3f(1.0, 1.0, 0.0);
       break;

case 4:
       reflection_x();
       glColor3f(0.0, 1.0, 0.0);
       break;

case 5:
       reflection_y();
       glColor3f(0.0, 0.0, 1.0);
       break;

case 6:
       reflection_origin();
       glColor3f(1.0, 1.0, 0.0);
       break;

case 7:
       reflection_xy();
       glColor3f(0.0, 1.0, 1.0);
       break;

case 8:
       cout << "Enter shearing factor: ";
       cin >> sf;

       shearing_x(sf);
       glColor3f(1.0, 0.0, 1.0);
       break;
```

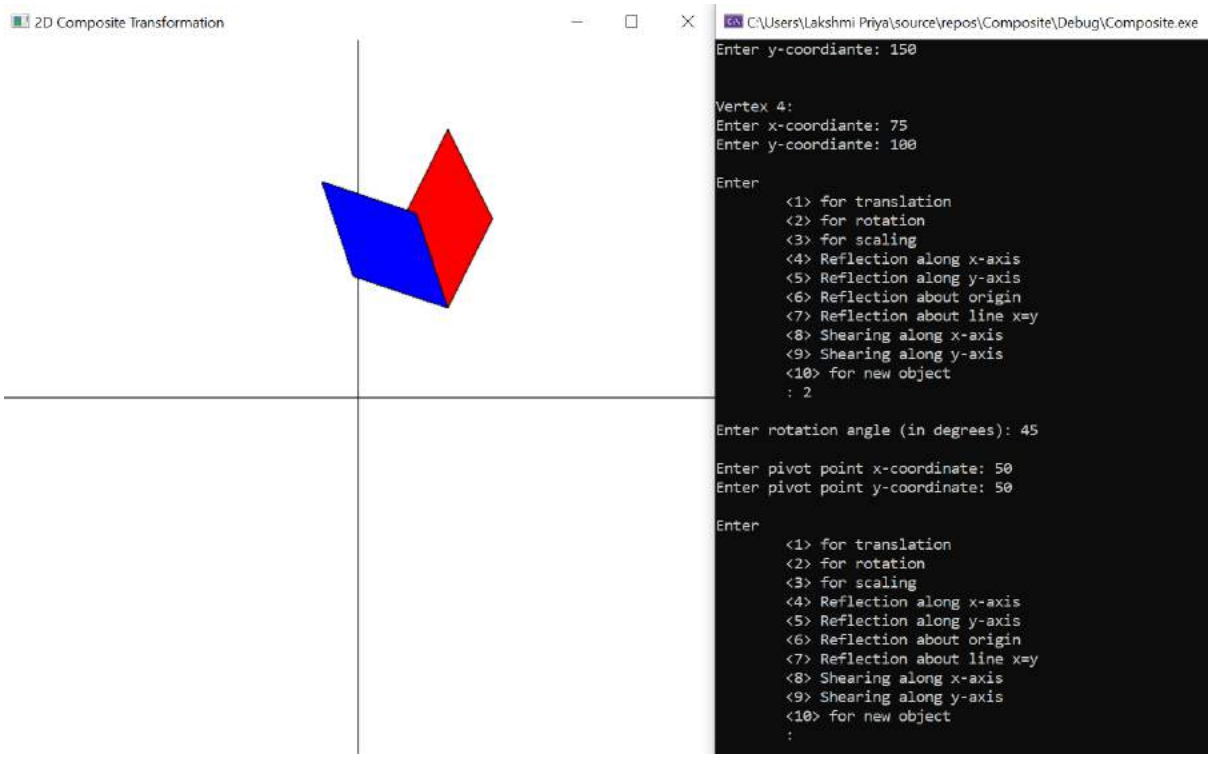
```
                        case 9:
                                cout << "Enter shearing factor: ";
                                cin >> sf;

                                shearing_y(sf);
                                glColor3f(1.0, 0.0, 1.0);
                                break;

                        case 10:
                                break;

                        default:
                                cout << "Enter valid option!!" << endl;
                                continue;
                        }

                        if (opt == 10)
                                break;
                        transformVerts2D(nVerts, verts);
                        drawObject(nVerts, verts);
                        glFlush();
                }
        }
}

void winReshapeFcn(GLint newWidth, GLint newHeight)
{
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(xwcMin, xwcMax, ywcMin, ywcMax);
        glClear(GL_COLOR_BUFFER_BIT);
}

int main(int argc, char** argv)
{
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
        glutInitWindowPosition(50, 50);
        glutInitWindowSize(winWidth, winHeight);
        glutCreateWindow("2D Composite Transformation");
        init();
        glutDisplayFunc(displayFcn);
        glutReshapeFunc(winReshapeFcn);
        glutMainLoop();
        return 0;
}
```

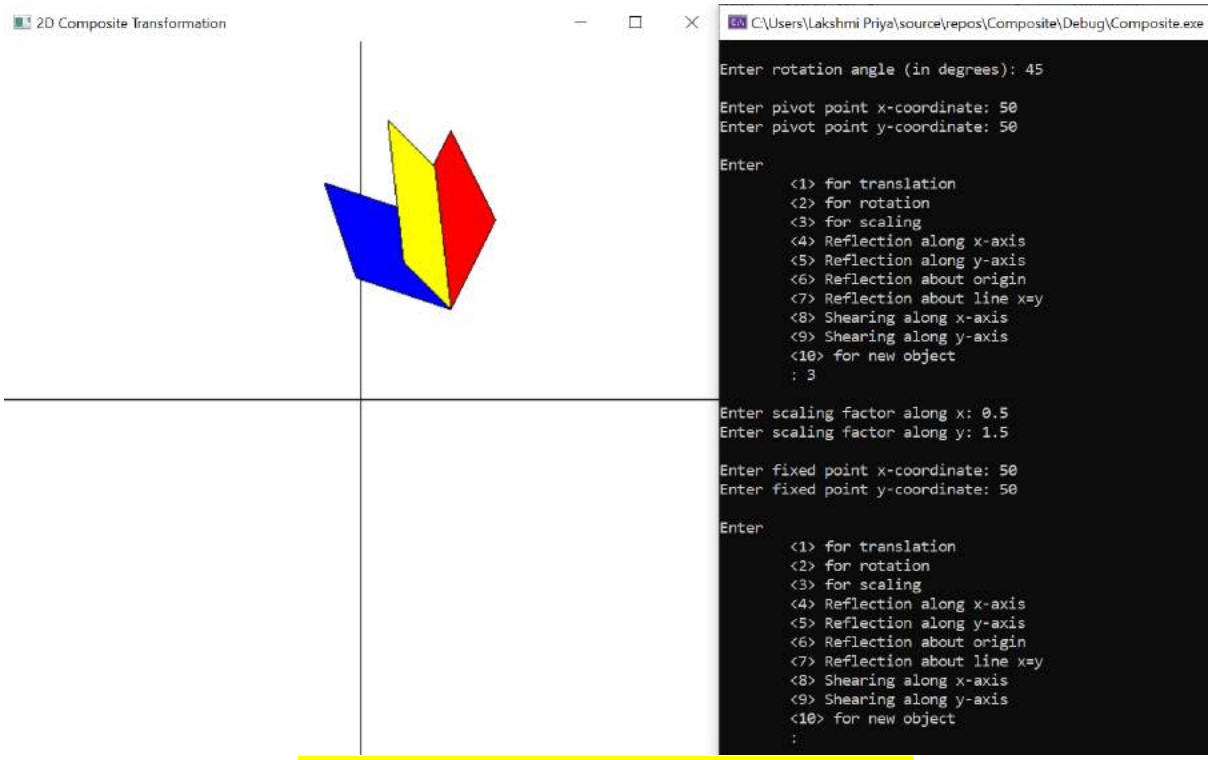
Output Screenshot

Case 1: Rotation about fixed point + Scaling

Test Case 1:

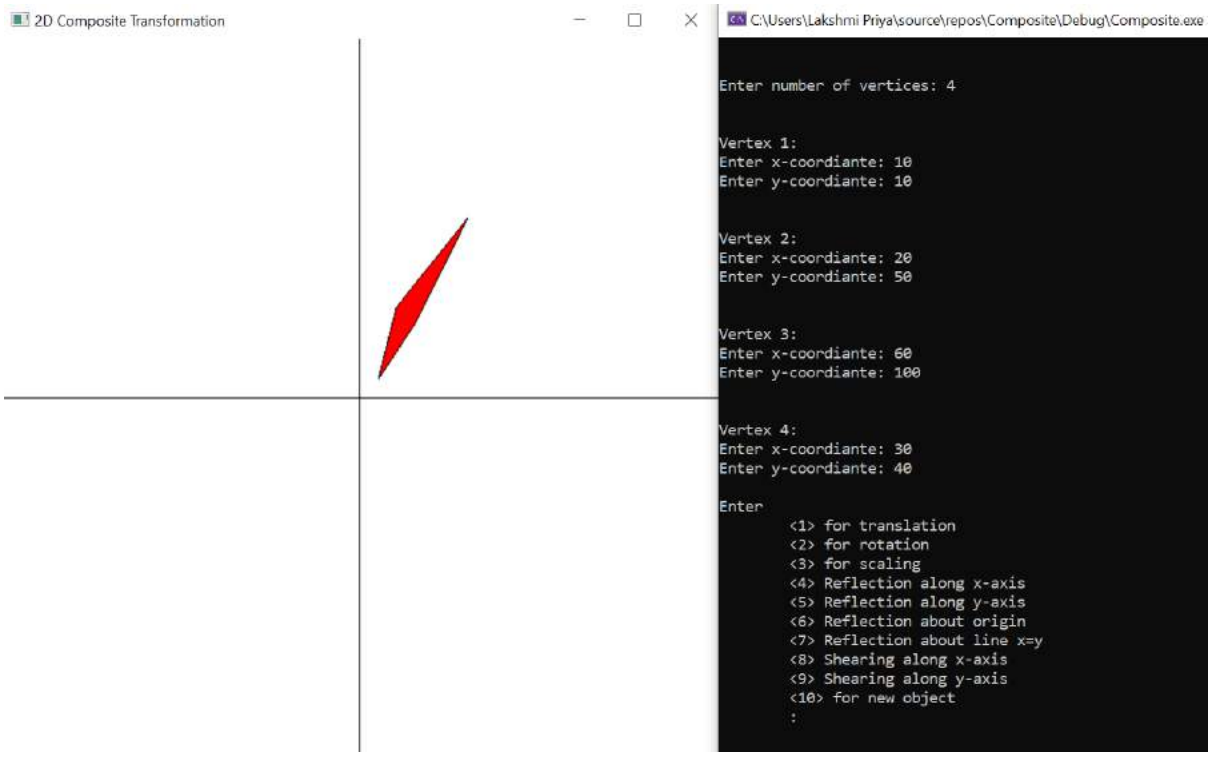


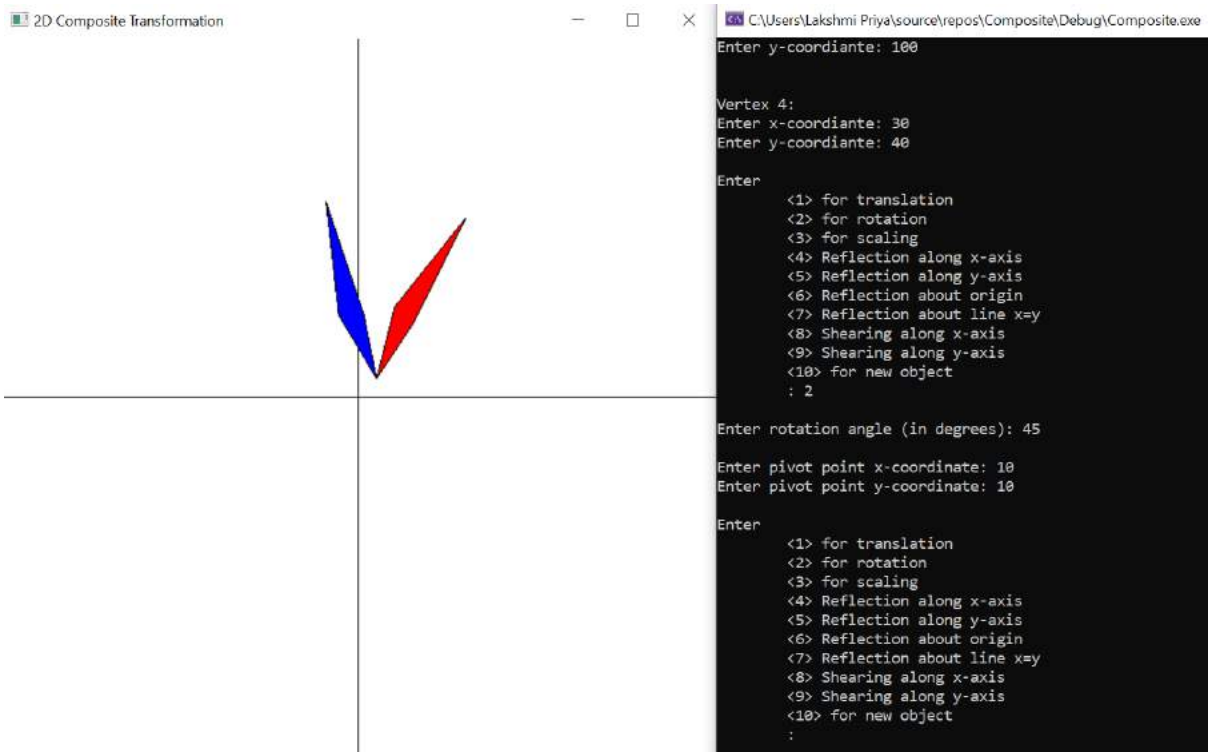


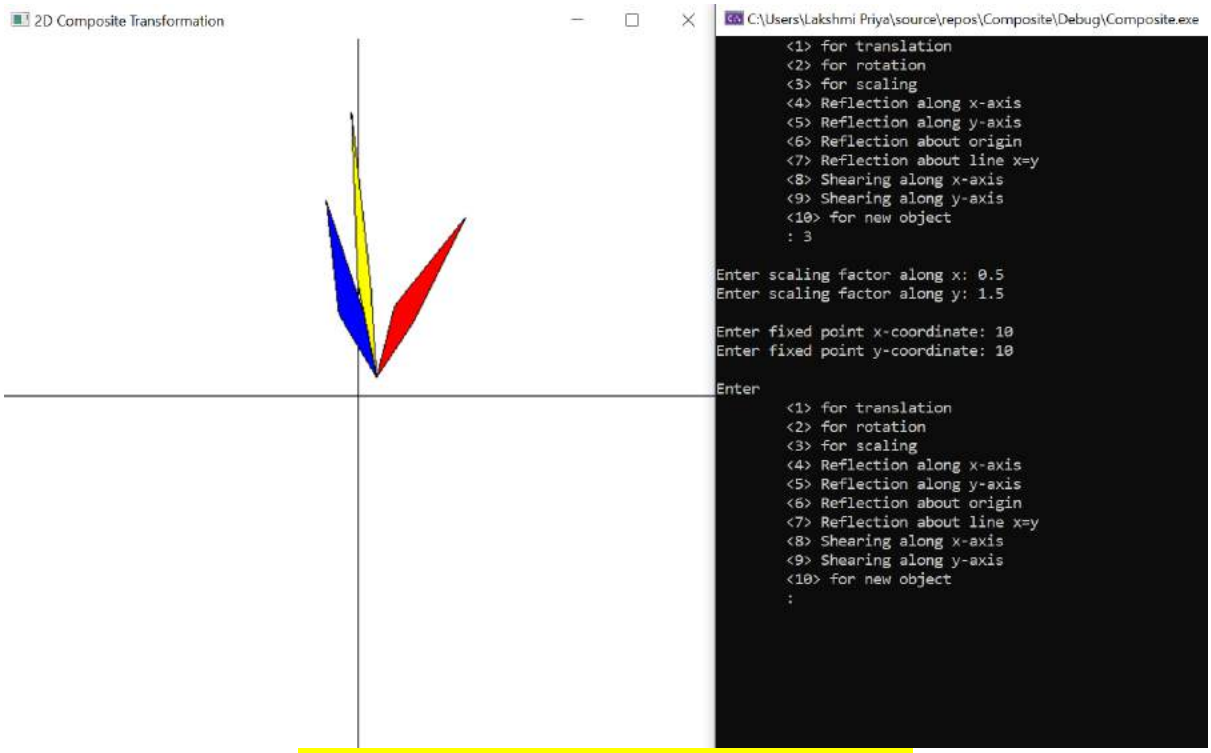


Transformation applied on the transformed object

Test Case 2:



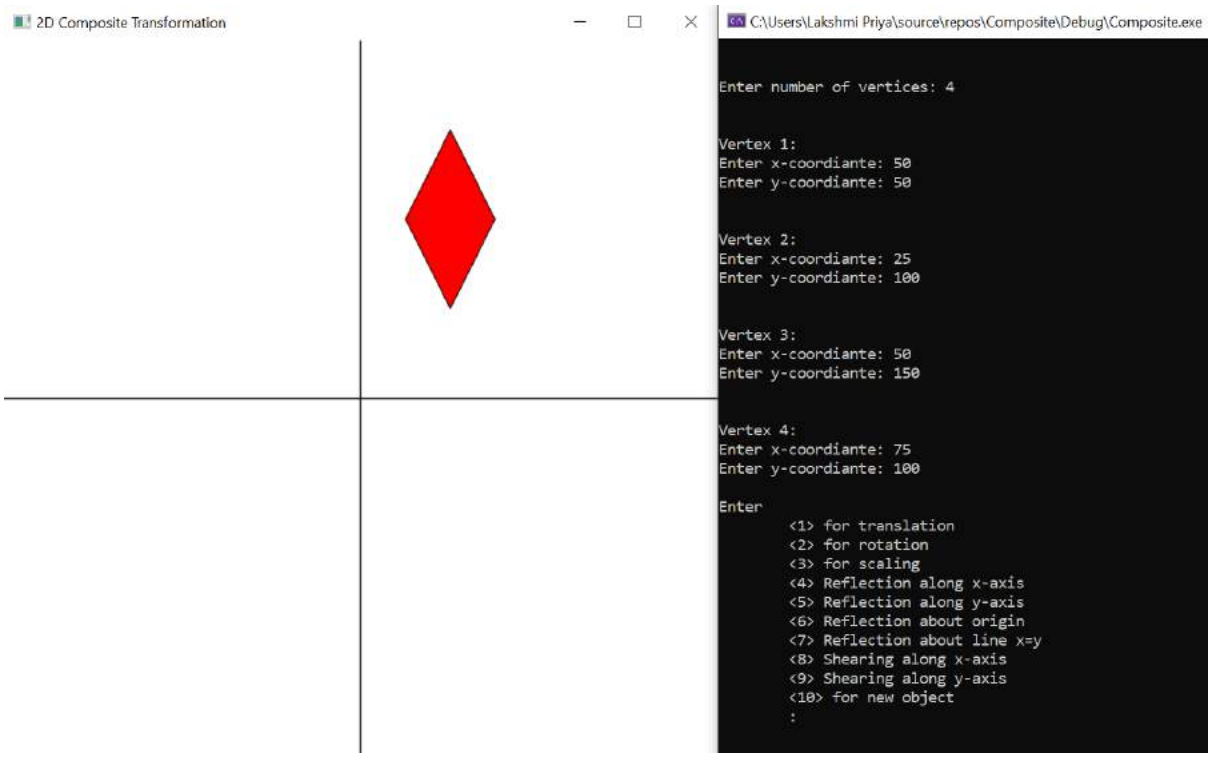


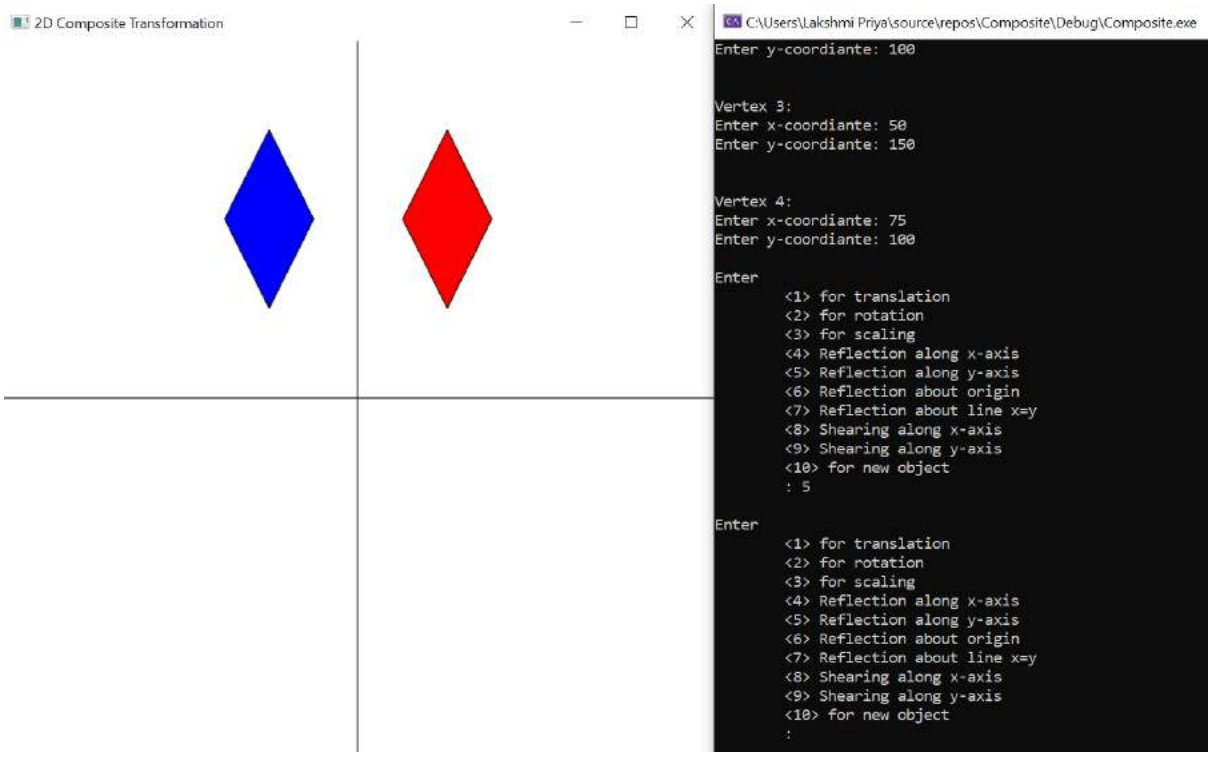


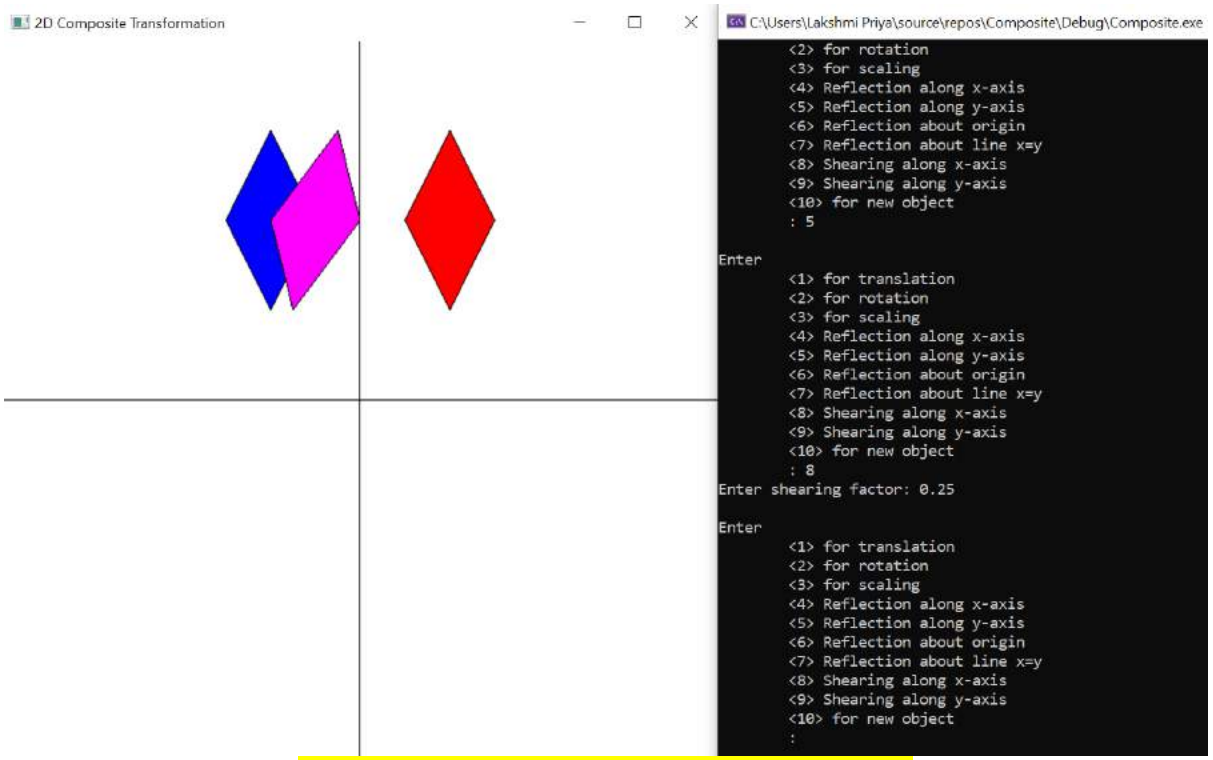
Transformation applied on the transformed object

Case 2: Reflection + Shearing

Test Case 1:

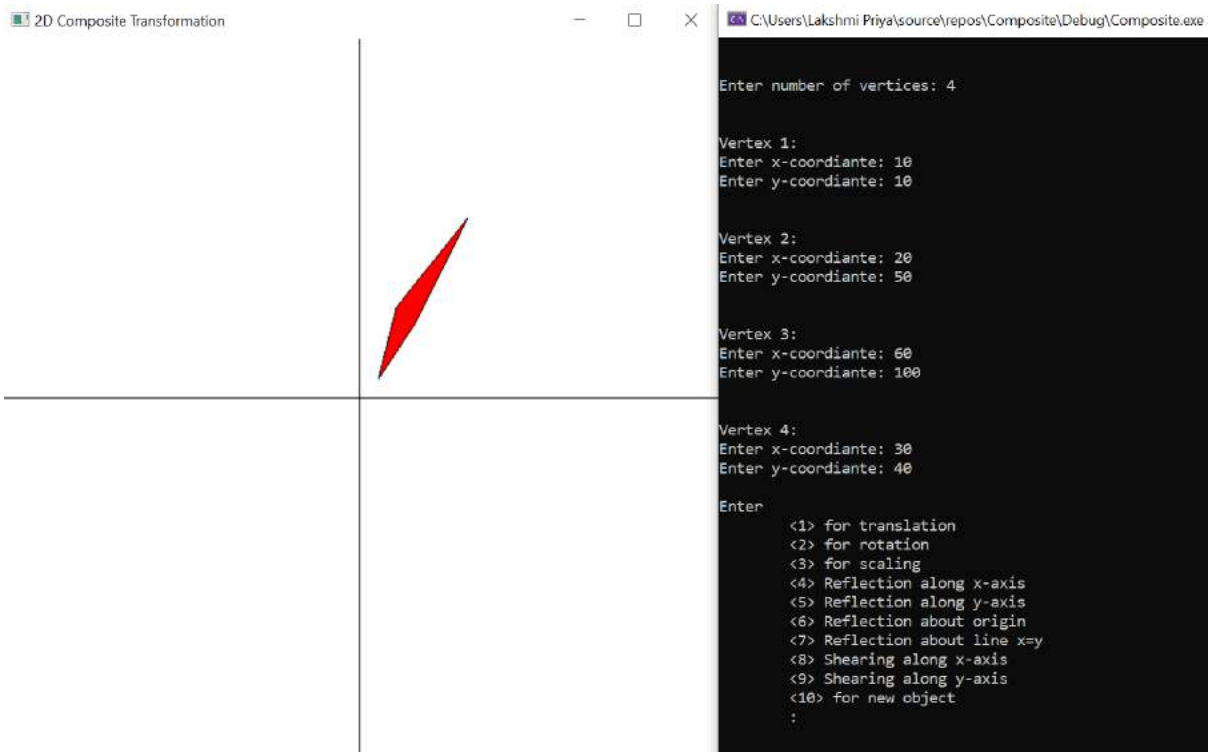


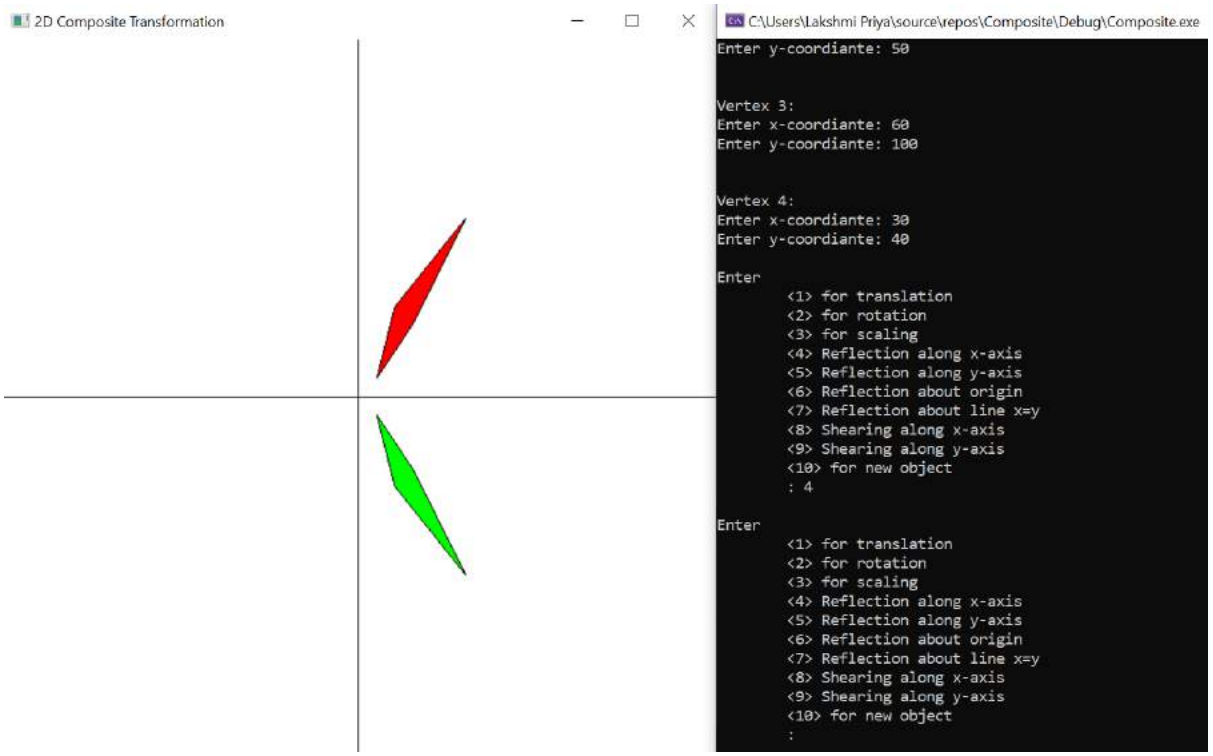


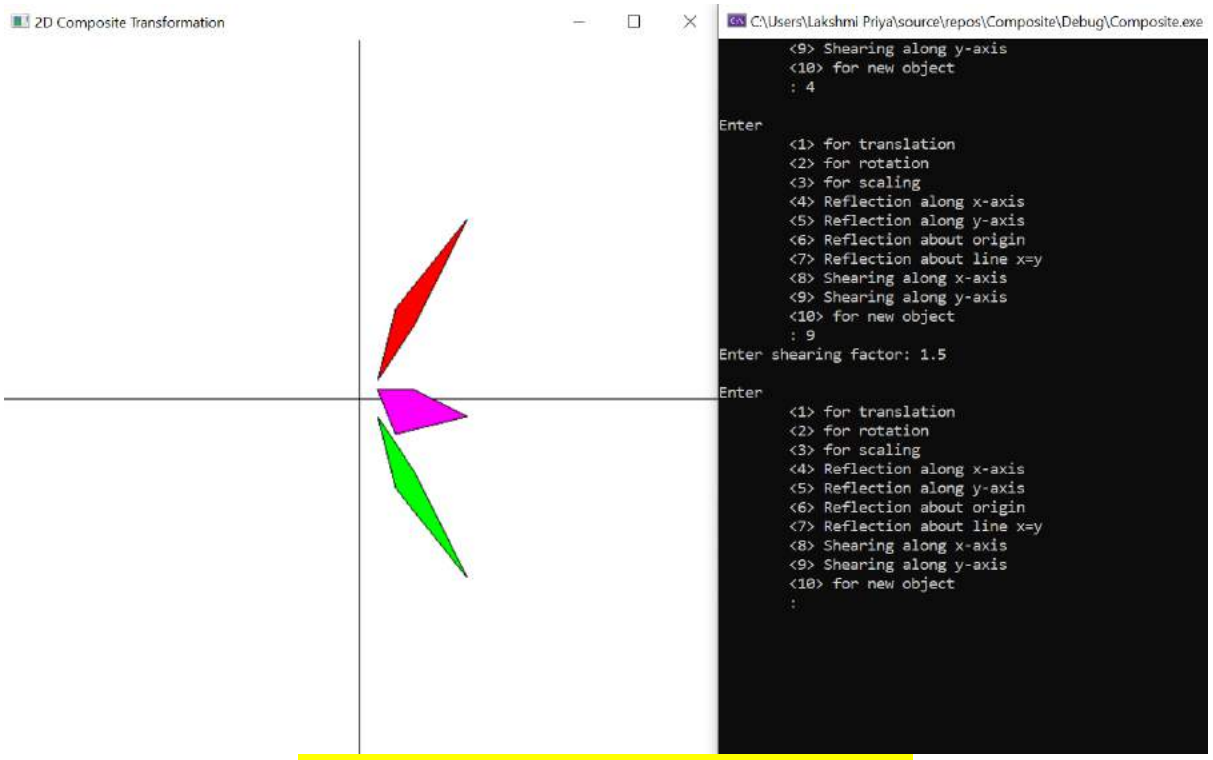


Transformation applied on the transformed object

Test Case 2:







Transformation applied on the transformed object

Result

Thus, a C++ program has been written using OPENGL to perform 2D composite transformations involving translation, rotation, scaling, reflection and shearing for input object.

SSN COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UCS1712 – GRAPHICS AND MULTIMEDIA LAB

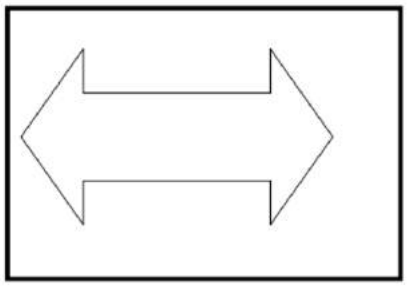
EX NO: 6b – Window to view port Mapping

Name : **Lakshmi Priya B**
Register Number : **185001083**
Date : **03-09-2021**

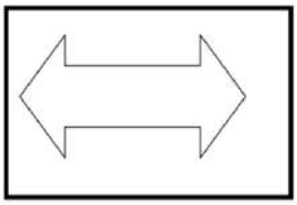
Aim

To create an object and window as given below in order to create a view port of size smaller than the window and apply window to viewport transformation of the object.

Original Object:



After Transformations



Algorithm

1. Set the boundary values for window in x_wmax, y_wmax, x_wmin, y_wmin
2. Set the boundary values for viewport in x_vmax, y_vmax, x_vmin, y_vmin
3. Set the points in the object to be drawn
4. Set npoints to contain the number of vertices in the object
5. Define WindowtoViewport(x_w, y_w, result) function:
    a. Define x_v, y_v
    b. Calculate sx as (x_vmax - x_vmin) / (x_wmax - x_wmin)
    c. Calculate sy as (y_vmax - y_vmin) / (y_wmax - y_wmin)
    d. Calculate x_v as x_vmin + ((x_w - x_wmin) * sx)
    e. Calculate y_v as y_vmin + ((y_w - y_wmin) * sy)

f. Print the coordinates of point on window and viewport
g. Store x_v in result[0]
h. Store y_v in result[1]

6. Define the display1() function:
    a. Display the window as the color of the current buffer using GL_COLOR_BUFFER_BIT
    b. Specify the display area of the window
    c. Set the color of the window
    d. Draw the window based on the boundary of window using x_wmin, x_wmax, y_wmin and y_wmax
    e. For each vertex of the object:
        i. Plot the vertex
    f. Send all the output to display using glFlush()
7. Define the display2() function:
    a. Display the viewport as the color of the current buffer using GL_COLOR_BUFFER_BIT
    b. Specify the display area of the viewport
    c. Set the color of the viewport
    d. Draw the viewport based the boundary of viewport using x_vmin, x_vmax, y_vmin and y_vmax
    e. For each vertex of the object:
        i. Get the mapped vertex by executing WindowtoViewport() function by passing the vertex as parameter
        ii. Plot the mapped vertex returned by the WindowtoViewport() function
    f. Send all the output to display using glFlush()
8. Define the main() function:
    a. Initialize using the glutInit() function with &argc and argv as parameters
    b. Specify the first window position as 50, 50
    c. Specify the first window size as 500, 500
    d. Create first window and set the title as “Window”
    e. Register display1 as callback function for “Window”
    f. Specify the second window position as 650, 50
    g. Specify the second window size as 500, 500
    h. Create second window and set the title as “Viewport”
    i. Register display2 as callback function for “Viewport”
    j. Go into a loop using glutMainLoop() until event occurs
9. Run the application to map from window to viewport and view the transformed object.

Code

Source.cpp:

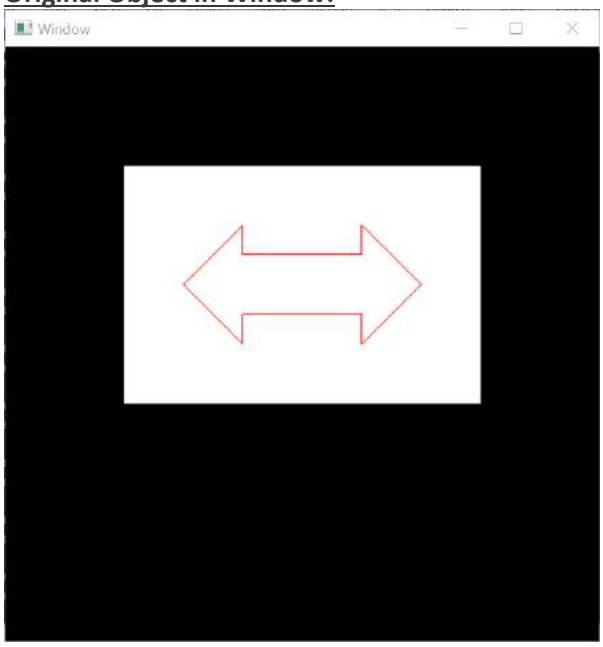
```
#include <iostream>
#include <GL/glut.h>
using namespace std;

// boundary values for window
int x_wmax = 80, y_wmax = 80, x_wmin = 20, y_wmin = 40;

// boundary values for viewport
int x_vmax = 60, y_vmax = 60, x_vmin = 30, y_vmin = 40;
```

```
// points in object
int points[][2] =    { {30, 60}, {40, 70}, {40, 65}, {60, 65}, {60, 70},
                       {70, 60}, {60, 50}, {60, 55}, {40, 55}, {40, 50}
                     };
int npoints = 10;


// Function for window to viewport transformation
void WindowtoViewport(int x_w, int y_w, int result[])
{
	// point on viewport
	int x_v, y_v;

	// scaling factors for x coordinate and y coordinate
	float sx, sy;

	// calculating sx and sy
	sx = (float)(x_vmax - x_vmin) / (x_wmax - x_wmin);
	sy = (float)(y_vmax - y_vmin) / (y_wmax - y_wmin);

	// calculating the point on viewport
	x_v = x_vmin + (float)((x_w - x_wmin) * sx);
	y_v = y_vmin + (float)((y_w - y_wmin) * sy);

	cout << "\nThe point on window: (" << x_w << "," << y_w << ")" << endl;
	cout << "The point on viewport: (" << x_v << "," << y_v << ")" << endl;

	result[0] = x_v;
	result[1] = y_v;
}


// Display1 callback
void display1()
{
	// clear the draw buffer .
	glClear(GL_COLOR_BUFFER_BIT);   // Erase everything
	// gluOrtho2D(xwcMin, xwcMax, ywcMin, ywcMax);
	gluOrtho2D(0, 100, 0, 100);

	// create a polygon ( define the vertexs)
	glBegin(GL_POLYGON);
	glColor3f(1, 1, 1);
	glVertex2f(x_wmin, y_wmin);
	glVertex2f(x_wmin, y_wmax);
	glVertex2f(x_wmax, y_wmax);
	glVertex2f(x_wmax, y_wmin);
	glEnd();

	glBegin(GL_LINE_LOOP);
	glColor3f(1, 0, 0);

	for (int i = 0; i < npoints; i++) {
		glVertex2f(points[i][0], points[i][1]);
	}

	glEnd();

	glFlush();
}
```

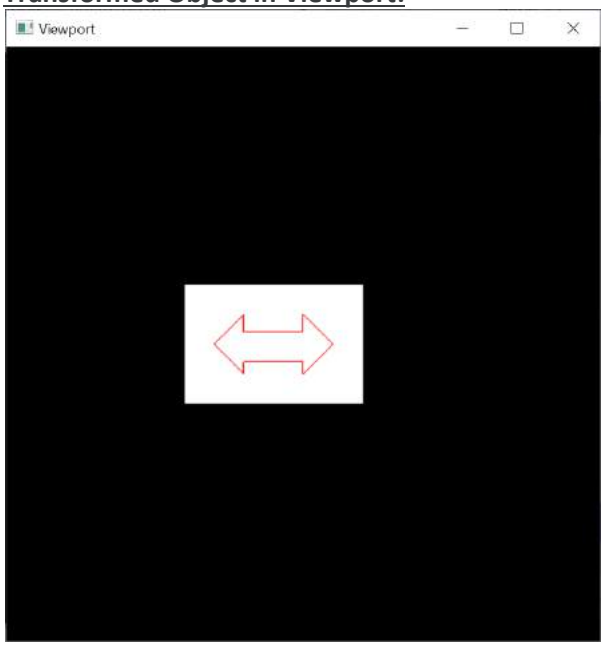
```
// Display2 callback
void display2()
{
	// clear the draw buffer .
	glClear(GL_COLOR_BUFFER_BIT);

	// gluOrtho2D(xwcMin, xwcMax, ywcMin, ywcMax);
	gluOrtho2D(0, 100, 0, 100);

	// create a polygon ( define the vertexs)
	glBegin(GL_POLYGON);
	glColor3f(1, 1, 1);
	glVertex2f(x_vmin, y_vmin);
	glVertex2f(x_vmin, y_vmax);
	glVertex2f(x_vmax, y_vmax);
	glVertex2f(x_vmax, y_vmin);
	glEnd();

	glBegin(GL_LINE_LOOP);
	glColor3f(1, 0, 0);

	int result[2];
	for (int i = 0; i < npoints; i++) {
		WindowtoViewport(points[i][0], points[i][1], result);
		glVertex2f(result[0], result[1]);
	}

	glEnd();

	glFlush();
}


// Main execution  function
int main(int argc, char* argv[])
{
	glutInit(&argc, argv);		// Initialize GLUT

	glutInitWindowPosition(50, 50);
	glutInitWindowSize(500, 500);

	glutCreateWindow("Window");	// Create a window 1
	glutDisplayFunc(display1);	// Register display1 callback

	glutInitWindowPosition(650, 50);
	glutInitWindowSize(500, 500);

	glutCreateWindow("Viewport");	// Create a window 2
	glutDisplayFunc(display2);	// Register display2 callback

	glutMainLoop();			// Enter main event loop
}
```

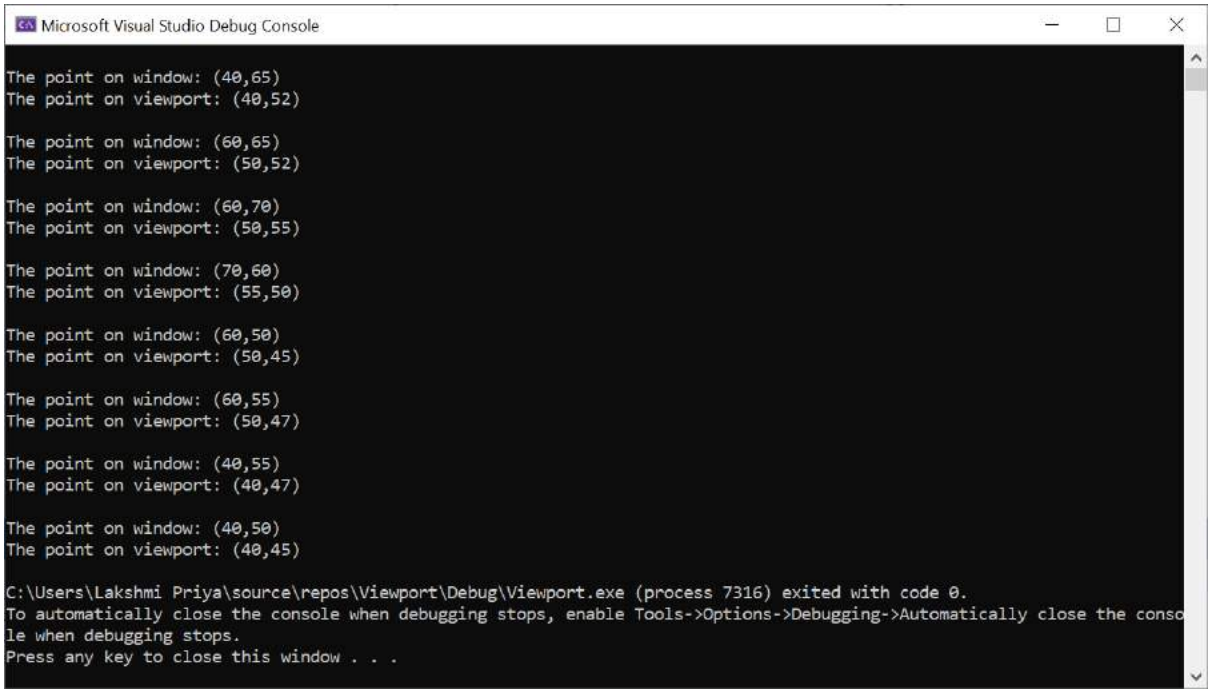
Output Screenshot

Original Object in Window:



Transformed Object in Viewport:





Result

Thus, view port of size smaller than the window has been created and window to viewport transformation of the object has been applied.

SSN COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UCS1712 – GRAPHICS AND MULTIMEDIA LAB

EX NO: 7 – Cohen-Sutherland Line Clipping Algorithm

Name : **Lakshmi Priya B**
Register Number : **185001083**

Date : **11-09-2021**

Aim

To write a C++ program using OPENGL to clip a line using Cohen-Sutherland line clipping algorithm.

Algorithm

1. Define init() function:
    a. Set the background color as white
    b. Set glMatrixMode(GL_PROJECTION)
    c. Specify the display area
2. Define code(x, y) function:
    a. Set c as 0
    b. If y > ymax:
        i. Set c as 8
    c. If y < ymin:
        i. Set c as 4
    d. If x > xmax:
        i. Set c as c | 2
    e. If x < xmin:
        i. Set c as c | 1
    f. Return c
3. Define CohenSutherlandLineClipping(x1, y1, x2, y2) function:
    a. Call drawLine() to display the line before clipping
    b. Send all the output to display using glFlush()
    c. To assign a region code for two endpoints of given line:
        i. Execute code(x1, y1) and store returned value in c1
        ii. Execute code(x2, y2) and store returned value in c2
    d. If both endpoints have a region code 0000:
        i. Report that the given line is completely inside
    e. Else, perform the logical AND operation for both region codes
        i. If the result is not 0000, then given line is completely outside
        ii. Else line is partially inside
            1. Choose an endpoint of the line that is outside the given rectangle
            2. Find the intersection point of the rectangular boundary (based on region code)

3. Replace endpoint with intersection point and update region code
4. Display the clipped line at this stage by calling drawLine() function
5. Repeat until we find a clipped line either trivially accepted or trivially rejected

4. Define display() function:
    a. Call CohenSutherlandLineClipping(xd1, yd1, xd2, yd2)
5. Define drawLine() function:
    a. Print the end points of line at current stage
    b. Display the window as the color of the current buffer using GL_COLOR_BUFFER_BIT
    c. Set the color of the display object
    d. Draw the x-axis and y-axis for reference
    e. Draw the window using GL_LINE_LOOP
    f. Draw the line at current stage using GL_LINES
    g. Send all the output to display using glFlush()
6. Define main() function:
    a. Get the window co-ordinates xmin, ymin, xmax and ymax as input from the user
    b. Get the start and end point co-ordinates of line xd1, yd1, xd2 and yd2
    c. Initialize using the glutInit() function with &argc and argv as parameters
    d. Set the display mode as GLUT_SINGLE and specify the color scheme as GLUT_RGB
    e. Specify the window position as 0, 0
    f. Specify the window size as 600, 600
    g. Create a new window and set the title as “Cohen-Sutherland Line Clipping”
    h. Call glutDisplayFunc(display)
    i. Call init() function
    j. Go into a loop using glutMainLoop() until event occurs
7. Run the application
8. View the output of line before clipping and clipped line in stages, one boundary at a time

Code

Source.cpp:

```
#include<GL/glut.h>
#include<iostream>

void display();
void drawLine();
using namespace std;

float xmin, ymin, xmax, ymax;
float xd1, yd1, xd2, yd2;
int c;
int flag = 1;


void init(void)
{
    glClearColor(0.0, 0, 0, 0);
    glMatrixMode(GL_PROJECTION);
    gluOrtho2D(-12, 12, -12, 12);
}
```

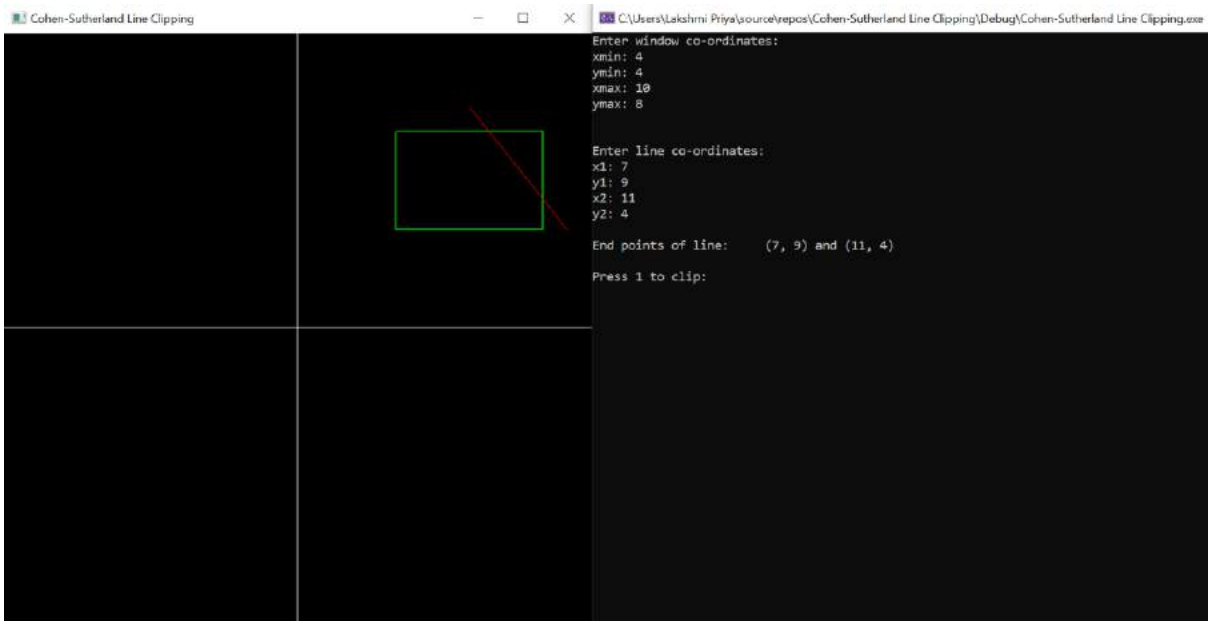
```
int code(float x, float y)
{
    int c = 0;
    if (y > ymax)c = 8;
    if (y < ymin)c = 4;
    if (x > xmax)c = c | 2;
    if (x < xmin)c = c | 1;
    return c;
}


void CohenSutherlandLineClipping(float x1, float y1, float x2, float y2)
{
    drawLine();
    glFlush();

    int c1 = code(x1, y1);
    int c2 = code(x2, y2);
    float m = (y2 - y1) / (x2 - x1);
    while ((c1 | c2) > 0)
    {
        if ((c1 & c2) > 0)
        {
            cout << "\n\nLine lies outsite boundary!";
            flag = 0;
            break;
        }

        float xi = x1; float yi = y1;
        int c = c1;
        if (c == 0)
        {
            c = c2;
            xi = x2;
            yi = y2;
        }
        float x, y;
        if ((c & 8) > 0)
        {
            y = ymax;
            x = xi + 1.0 / m * (ymax - yi);
        }
        else
            if ((c & 4) > 0)
            {
                y = ymin;
                x = xi + 1.0 / m * (ymin - yi);
            }
            else
                if ((c & 2) > 0)
                {
                    x = xmax;
                    y = yi + m * (xmax - xi);
                }
                else
                    if ((c & 1) > 0)
                    {
                        x = xmin;
                        y = yi + m * (xmin - xi);
                    }

        if (c == c1)
```

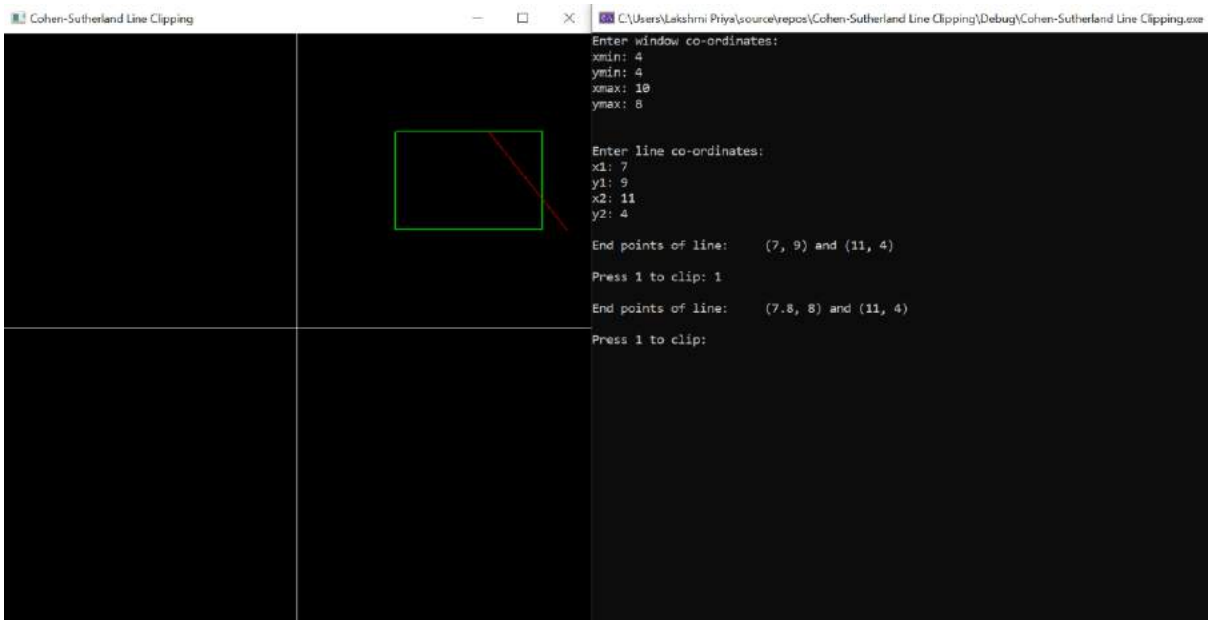
```
        {
            xd1 = x;
            yd1 = y;
            cout << "\nPress 1 to clip: ";
            cin >> c;
            drawLine();
            c1 = code(xd1, yd1);
        }

        if (c == c2)
        {
            xd2 = x;
            yd2 = y;
            cout << "\nPress 1 to clip: ";
            cin >> c;
            drawLine();
            c2 = code(xd2, yd2);
        }
    }
    if (flag == 1)
        cout << "\n\nLine lies within boundary!!";
}


void display()
{
    CohenSutherlandLineClipping(xd1, yd1, xd2, yd2);
}


void drawLine()
{
    cout << "\nEnd points of line:\t(" << xd1 << ", " << yd1 << ") and (" << xd2 << ",
" << yd2 << ")\n";

    glClear(GL_COLOR_BUFFER_BIT);

    glColor3f(1.0, 1.0, 1.0);
    glBegin(GL_LINES);
    glVertex2f(0, 12);
    glVertex2f(0, -12);
    glVertex2f(-12, 0);
    glVertex2f(12, 0);
    glEnd();

    glColor3f(0.0, 1.0, 0.0);
    glBegin(GL_LINE_LOOP);
    glVertex2f(xmin, ymin);
    glVertex2f(xmin, ymax);
    glVertex2f(xmax, ymax);
    glVertex2f(xmax, ymin);
    glEnd();

    glColor3f(1.0, 0.0, 0.0);
    glBegin(GL_LINES);
    glVertex2f(xd1, yd1);
    glVertex2f(xd2, yd2);
    glEnd();
    glFlush();

}
```

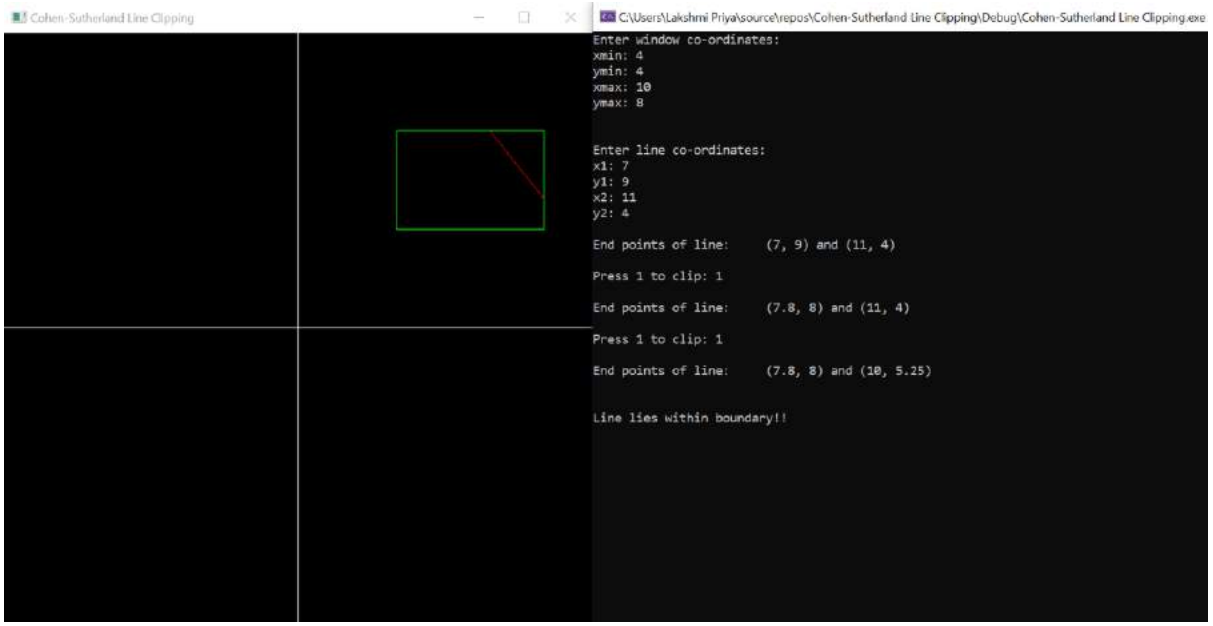
```
int main(int argc, char** argv)
{
    cout << "Enter window co-ordinates:\n";
    cout << "xmin: ";
    cin >> xmin;
    cout << "ymin: ";
    cin >> ymin;
    cout << "xmax: ";
    cin >> xmax;
    cout << "ymax: ";
    cin >> ymax;

    cout << "\n\nEnter line co-ordinates:\n";
    cout << "x1: ";
    cin >> xd1;
    cout << "y1: ";
    cin >> yd1;
    cout << "x2: ";
    cin >> xd2;
    cout << "y2: ";
    cin >> yd2;

    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(600, 600);
    glutInitWindowPosition(0, 0);
    glutCreateWindow("Cohen-Sutherland Line Clipping");
    glutDisplayFunc(display);
    init();
    glutMainLoop();
    return 0;
}
```

Output Screenshot

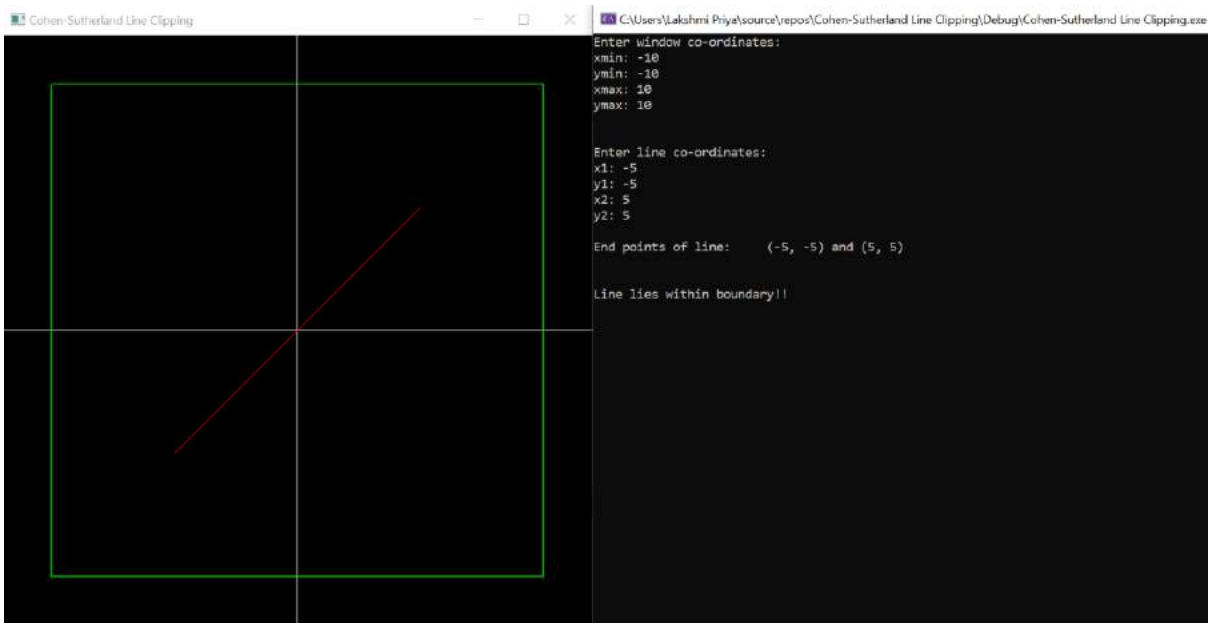
Case 1: Line Clipped



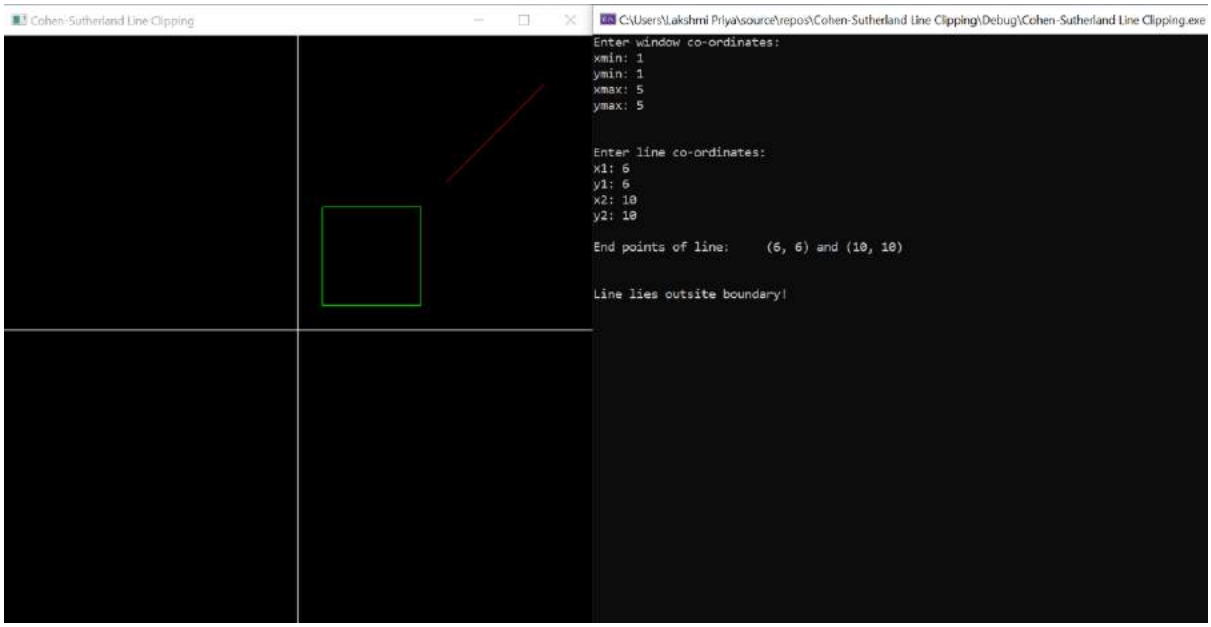

Cohen-Sutherland Line Clipping
C:\Users\Lakshmi Priya\source\repos\Cohen-Sutherland Line Clipping\Debug\Cohen-Sutherland Line Clipping.exe
Enter window co-ordinates:
xmin: 4
ymin: 4
xmax: 10
ymax: 8
Enter line co-ordinates:
x1: 7
y1: 9
x2: 11
y2: 4
End points of line: (7, 9) and (11, 4)
Press 1 to clip: 1
End points of line: (7.8, 8) and (11, 4)
Press 1 to clip:

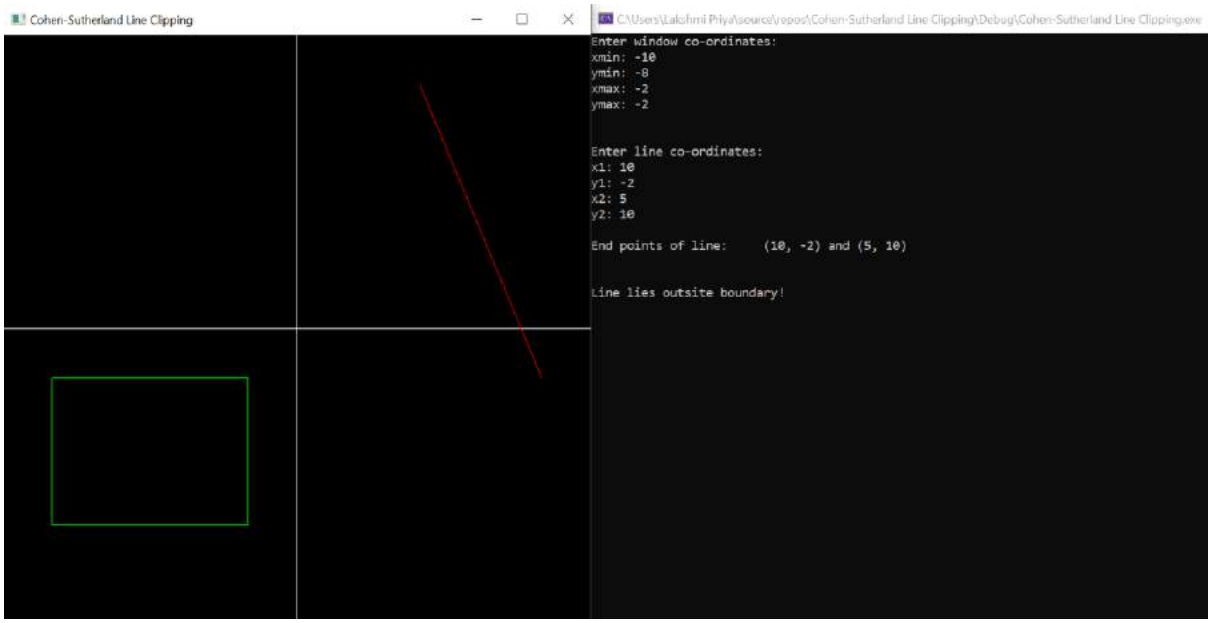

Cohen-Sutherland Line Clipping
C:\Users\Lakshmi Priya\source\repos\Cohen-Sutherland Line Clipping\Debug\Cohen-Sutherland Line Clipping.exe
Enter window co-ordinates:
xmin: 4
ymin: 4
xmax: 10
ymax: 8
Enter line co-ordinates:
x1: 7
y1: 9
x2: 11
y2: 4
End points of line: (7, 9) and (11, 4)
Press 1 to clip: 1
End points of line: (7.8, 8) and (11, 4)
Press 1 to clip: 1
End points of line: (7.8, 8) and (10, 5.25)
Line lies within boundary!!

Case 2: Line Accepted



Case 3: Line Rejected





Result

Thus, a C++ program has been written using OPENGL to clip a line using Cohen-Sutherland line clipping algorithm.

SSN COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UCS1712 – GRAPHICS AND MULTIMEDIA LAB

EX NO: 8 – 3D Transformations – Translation, Rotation and Scaling

Name : **Lakshmi Priya B**
Register Number : **185001083**
Date : **25-09-2021**

Aim

To write a C++ menu-driven program using OPENGL to perform 3D transformations – translation, scaling and rotation along all three axes.

Algorithm

1. Initialize using the glutInit() function with &argc and argv as parameters
2. Set the display mode as GLUT_SINGLE, specify the color scheme as GLUT_RGB and specify GLUT_DEPTH
3. Specify the window position as 50, 50
4. Specify the window size as 600, 600
5. Create a new window and set the title as “3D Transformation”
6. Create a class point3D which has x, y, z as its attributes
7. Define init() function:
    a. Set the background color as white
8. Define winReshapeFcn(newWidth, newHeight)
    a. Set glMatrixMode(GL_PROJECTION)
    b. Specify the display area
    c. Enable GL_DEPTH_TEST
    d. Display the window as GL_COLOR_BUFFER_BIT and GL_DEPTH_BUFFER_BIT
9. Define the matrix4x4SetIdentity(matIdet4x4) function:
    a. Set the diagonal elements of 4x4 matrix as 1
    b. Set the off-diagonal elements of 4x4 matrix as 0
10. Define matrix4x4PreMultiple(m1, m2) function:
    a. Multiply the two matrices m1 and m2
    b. Store the resultant matrix in m2
11. Define translate3D(tx, ty) function:
    a. Initialize matTransl as identity matrix using matrix4x4SetIdentity() function
    b. Set matTransl[0][3] = tx
    c. Set matTransl[1][3] = ty
    d. Set matTransl[2][3] = tz
    e. Call matrix4x4PreMultiply(matTransl, matComposite)
12. Define rotate3D_x_axis(theta) function:
    a. Initialize matRot as identity matrix using matrix4x4SetIdentity() function

b. Set matRot[1][1] = cos(theta)
c. Set matRot[1][2] = -sin(theta)
d. Set matRot[2][1] = sin(theta)
e. Set matRot[2][2] = cos(theta)
f. Call matrix4x4PreMultiply(matRot, matComposite)

13. Define rotate3D_y_axis(theta) function:
    a. Initialize matRot as identity matrix using matrix4x4SetIdentity() function
    b. Set matRot[0][0] = cos(theta)
    c. Set matRot[0][2] = sin(theta)
    d. Set matRot[2][0] = -sin(theta)
    e. Set matRot[2][2] = cos(theta)
    f. Call matrix4x4PreMultiply(matRot, matComposite)
14. Define rotate3D_z_axis(theta) function:
    a. Initialize matRot as identity matrix using matrix4x4SetIdentity() function
    b. Set matRot[0][0] = cos(theta)
    c. Set matRot[0][1] = -sin(theta)
    d. Set matRot[1][0] = sin(theta)
    e. Set matRot[1][1] = cos(theta)
    f. Call matrix4x4PreMultiply(matRot, matComposite)
15. Define scale3D(sx, sy, sz, fixedPt) function:
    a. Initialize matScale as identity matrix using matrix4x4SetIdentity() function
    b. Set matScale[0][0] = sx
    c. Set matScale[0][3] = (1 - sx) * fixedPt.x
    d. Set matScale[1][1] = sy
    e. Set matScale[1][3] = (1 - sy) * fixedPt.y;
    f. Set matScale[2][2] = sz
    g. Set matScale[2][3] = (1 - sz) * fixedPt.z;
    h. Call matrix4x4PreMultiply(matScale, matComposite)
16. Define transformVerts3D(nVerts, verts, plotverts) function:
    a. Multiply the original object vertices and multiplication factors present in matComposite as per homogeneous coordinate representations
    b. Obtain the new transformed vertices of the object
    c. Store the new vertices in plotverts
17. Define drawObject(nVerts, verts) function:
    a. Draw each face of the object using GL_QUADS
        i. For each face of the object:
            1. For each vertex in the face of the object:
                a. Plot (verts[k].x, verts[k].y, verts[k].z)
18. Define the displayFcn() function:
    a. Declare nVerts, verts, plotverts
    b. Loop 1:
        i. Get number of vertices as input from user
        ii. Get coordinates of the vertices as input from the user
        iii. Display the window as the color of the current buffer using GL_COLOR_BUFFER_BIT
        iv. Draw the x-axis and y-axis for reference using GL_LINES by specifing the coordinates
        v. Draw the original object using drawObject(nVerts, verts)

vi. Send all the output to display using glFlush()
vii. Loop 2:
    1. Compute centroid
    2. Set fixed point as centroid
    3. Call matrix4x4SetIdentity(matrixComposite)
    4. Display the menu of transformations available
    5. Get the option from the user
    6. If option is 1:
        a. Get translation along x as input
        b. Get translation along y as input
        c. Get translation along z as input
        d. Execute translate3D(tx, ty, tz)
    7. Else if option is 2:
        a. Get rotation angle (theta) in degrees as input
        b. Execute rotate3D_x_axis(theta * pi / 180)
    8. Else if option is 3:
        a. Get rotation angle (theta) in degrees as input
        b. Execute rotate3D_y_axis(theta * pi / 180)
    9. Else if option is 4:
        a. Get rotation angle (theta) in degrees as input
        b. Execute rotate3D_z_axis(theta * pi / 180)
    10. Else if option is 5:
        a. Get scaling factor along x as input
        b. Get scaling factor along y as input
        c. Get scaling factor along z as input
        d. Execute scale2D(sx, sy, sz, fixedPt)
    11. Else if option is 6:
        a. Continue with Loop 1
    12. Execute transformVerts3D(nVerts, verts, plotverts)
    13. Execute drawObject(nVerts, plotVerts)
    14. Send all the output to display using glFlush()

19. Define main() function:
    a. Call init() function
    b. Call glutDisplayFunc(displayFcn)
    c. Call glutReshapeFunc(winReshapeFcn)
    d. Go into a loop using glutMainLoop() until event occurs
20. Run the application.
21. Enter number of vertices in the object as input.
22. Either select the appropriate transformation to be applied like translation, rotation or scaling by choosing the appropriate menu or choose to input coordinates of a new object.
23. View the original object and the transformed object after providing appropriate input for transformation operation.

Code

Source.cpp:

```
#include <GL/glut.h>
#include <math.h>
#include <iostream>
using namespace std;

GLsizei winWidth = 600, winHeight = 600;
GLfloat xwcMin = -200.0, xwcMax = 200.0;
GLfloat ywcMin = -200.0, ywcMax = 200.0;
GLfloat zwcMin = -200.0, zwcMax = 200.0;

class point3d
{
public: GLfloat x, y, z;
};

typedef GLfloat Matrix4x4[4][4];
Matrix4x4 matComposite;
const GLdouble pi = 3.14159;

void init(void)
{
	glClearColor(1.0, 1.0, 1.0, 0.0);
}

void Matrix4x4SetIdentity(Matrix4x4 matIdent4x4)
{
	GLint row, col;
	for (row = 0; row < 4; row++)
		for (col = 0; col < 4; col++)
			matIdent4x4[row][col] = (row == col);
}

void Matrix4x4PreMultiply(Matrix4x4 m1, Matrix4x4 m2)
{
	GLint row, col;
	Matrix4x4 matTemp;
	for (row = 0; row < 4; row++)
		for (col = 0; col < 4; col++)
			matTemp[row][col] = m1[row][0] * m2[0][col] + m1[row][1] *
m2[1][col] + m1[row][2] * m2[2][col] + m1[row][3] * m2[3][col];
	for (row = 0; row < 4; row++)
		for (col = 0; col < 4; col++)
			m2[row][col] = matTemp[row][col];
}

void translate3D(GLfloat tx, GLfloat ty, GLfloat tz)
{
	Matrix4x4 matTransl;
	Matrix4x4SetIdentity(matTransl);
	matTransl[0][3] = tx;
	matTransl[1][3] = ty;
	matTransl[2][3] = tz;
	Matrix4x4PreMultiply(matTransl, matComposite);
}
```

```
void rotate3D_x_axis(GLfloat theta)
{
    //x-axis rotation
    Matrix4x4 matRot;
    Matrix4x4SetIdentity(matRot);
    matRot[1][1] = cos(theta);
    matRot[1][2] = -sin(theta);
    matRot[2][1] = sin(theta);
    matRot[2][2] = cos(theta);
    Matrix4x4PreMultiply(matRot, matComposite);
}

void rotate3D_y_axis(GLfloat theta)
{
    //y-axis rotation
    Matrix4x4 matRot;
    Matrix4x4SetIdentity(matRot);
    matRot[0][0] = cos(theta);
    matRot[0][2] = sin(theta);
    matRot[2][0] = -sin(theta);
    matRot[2][2] = cos(theta);
    Matrix4x4PreMultiply(matRot, matComposite);
}

void rotate3D_z_axis(GLfloat theta)
{
    //z-axis rotation
    Matrix4x4 matRot;
    Matrix4x4SetIdentity(matRot);
    matRot[0][0] = cos(theta);
    matRot[0][1] = -sin(theta);
    matRot[1][0] = sin(theta);
    matRot[1][1] = cos(theta);
    Matrix4x4PreMultiply(matRot, matComposite);
}

void scale3D(GLfloat sx, GLfloat sy, GLfloat sz, point3d fixedPt)
{
       Matrix4x4 matScale;
       Matrix4x4SetIdentity(matScale);
       matScale[0][0] = sx;
       matScale[0][3] = (1 - sx) * fixedPt.x;
       matScale[1][1] = sy;
       matScale[1][3] = (1 - sy) * fixedPt.y;
       matScale[2][2] = sz;
       matScale[2][3] = (1 - sz) * fixedPt.z;
       Matrix4x4PreMultiply(matScale, matComposite);
}

void transformVerts3D(GLint nVerts, point3d* verts, point3d* plotverts)
{
       GLint k;  GLfloat temp;
       for (k = 0; k < nVerts; k++)
       {
              plotverts[k].x = matComposite[0][0] * verts[k].x + matComposite[0][1] *
verts[k].y + matComposite[0][2] * verts[k].z + matComposite[0][3];
              plotverts[k].y = matComposite[1][0] * verts[k].x + matComposite[1][1] *
verts[k].y + matComposite[1][2] * verts[k].z + matComposite[1][3];
              plotverts[k].z = matComposite[2][0] * verts[k].x + matComposite[2][1] *
verts[k].y + matComposite[2][2] * verts[k].z + matComposite[2][3];
       }
}
```

```
void drawObject(point3d* verts)
{
    glBegin(GL_QUADS);
    glColor3f(1.0, 1.0, 0.0); //behind face
    glVertex3f(verts[0].x, verts[0].y, verts[0].z);
    glVertex3f(verts[1].x, verts[1].y, verts[1].z);
    glVertex3f(verts[2].x, verts[2].y, verts[2].z);
    glVertex3f(verts[3].x, verts[3].y, verts[3].z);
    glEnd();

    glBegin(GL_QUADS);
    glColor3f(0.0, 1.0, 1.0);  //bottom face
    glVertex3f(verts[0].x, verts[0].y, verts[0].z);
    glVertex3f(verts[1].x, verts[1].y, verts[1].z);
    glVertex3f(verts[5].x, verts[5].y, verts[5].z);
    glVertex3f(verts[4].x, verts[4].y, verts[4].z);
    glEnd();

    glBegin(GL_QUADS);
    glColor3f(1.0, 0.0, 1.0); //left face
    glVertex3f(verts[0].x, verts[0].y, verts[0].z);
    glVertex3f(verts[4].x, verts[4].y, verts[4].z);
    glVertex3f(verts[7].x, verts[7].y, verts[7].z);
    glVertex3f(verts[3].x, verts[3].y, verts[3].z);
    glEnd();

    glBegin(GL_QUADS);
    glColor3f(0.0, 0.0, 1.0);  //right face
    glVertex3f(verts[1].x, verts[1].y, verts[1].z);
    glVertex3f(verts[2].x, verts[2].y, verts[2].z);
    glVertex3f(verts[6].x, verts[6].y, verts[6].z);
    glVertex3f(verts[5].x, verts[5].y, verts[5].z);
    glEnd();

    glBegin(GL_QUADS);
    glColor3f(0.0, 1.0, 0.0); //up face
    glVertex3f(verts[2].x, verts[2].y, verts[2].z);
    glVertex3f(verts[3].x, verts[3].y, verts[3].z);
    glVertex3f(verts[7].x, verts[7].y, verts[7].z);
    glVertex3f(verts[6].x, verts[6].y, verts[6].z);
    glEnd();

    glBegin(GL_QUADS);
    glColor3f(1.0, 0.0, 0.0); //front face
    glVertex3f(verts[4].x, verts[4].y, verts[4].z);
    glVertex3f(verts[5].x, verts[5].y, verts[5].z);
    glVertex3f(verts[6].x, verts[6].y, verts[6].z);
    glVertex3f(verts[7].x, verts[7].y, verts[7].z);
    glEnd();

    glFlush();
}

void displayFcn(void)
{
       GLint nVerts;
       point3d verts[10]; // = { {50, 50}, {25, 100}, {50, 150} ,{75, 100} };
       point3d plotverts[10];

    drawObject(verts);
    glFlush();
```

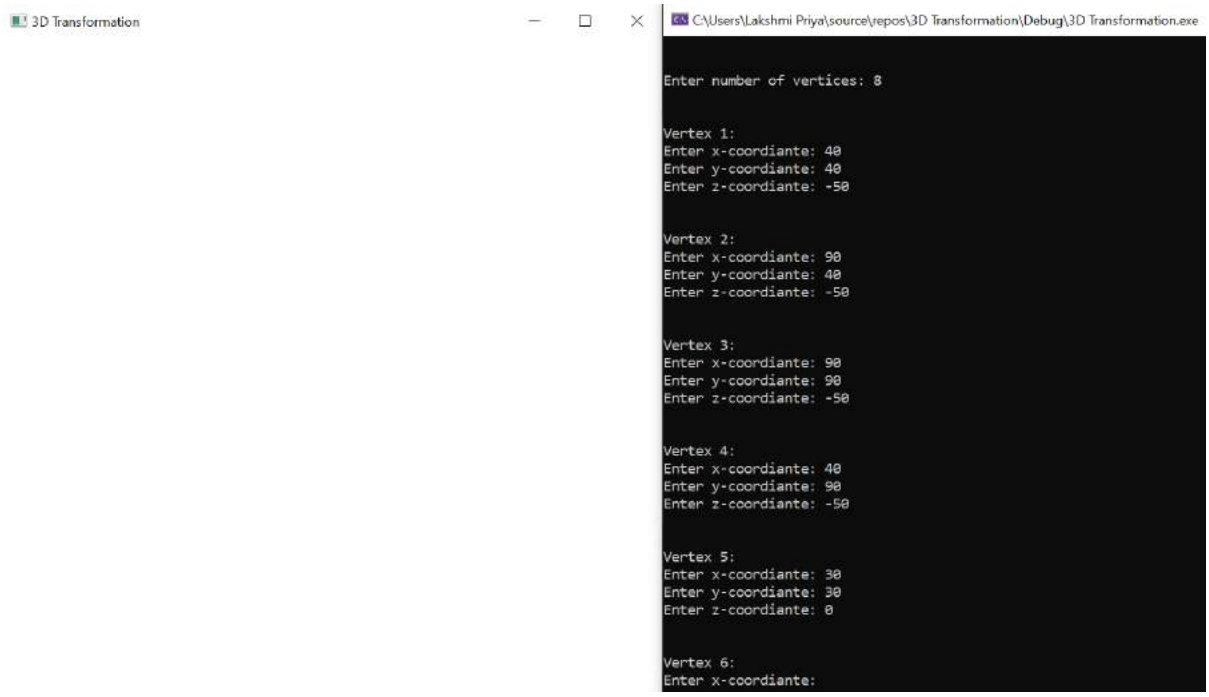
```
while (true) {
	cout << "\n\nEnter number of vertices: ";
	cin >> nVerts;

	for (int i = 0; i < nVerts; i++) {
		cout << "\n\nVertex " << i + 1 << ": " << endl;
		cout << "Enter x-coordiante: ";
		cin >> verts[i].x;
		cout << "Enter y-coordiante: ";
		cin >> verts[i].y;
		cout << "Enter z-coordiante: ";
		cin >> verts[i].z;
	}

	GLfloat tx, ty, tz;
	GLfloat sx, sy, sz;
	GLdouble theta;
	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

	glColor3f(0.0, 0.0, 0.0);
	glBegin(GL_LINES);

	glVertex3d(0, -200, 0);
	glVertex3d(0, 200, 0);

	glVertex3d(-200, 0, 0);
	glVertex3d(200, 0, 0);

	glVertex3d(0, 0, -200);
	glVertex3d(0, 0, 200);

	glEnd();

	glColor3f(1.0, 0.0, 0.0);
	drawObject(verts);
	glFlush();

	int opt;
	while (true) {
		point3d centroidPt;
		GLint k, xSum = 0, ySum = 0, zSum = 0;
		for (k = 0; k < nVerts; k++)
		{
			xSum += verts[k].x;
			ySum += verts[k].y;
			zSum += verts[k].z;

		}
		centroidPt.x = GLfloat(xSum) / GLfloat(nVerts);
		centroidPt.y = GLfloat(ySum) / GLfloat(nVerts);
		centroidPt.z = GLfloat(zSum) / GLfloat(nVerts);
		point3d fixedPt;
		fixedPt = centroidPt;

		Matrix4x4SetIdentity(matComposite);

		cout << "\nEnter\n\t<1> for translation" <<
			"\n\t<2> for rotation about x axis" <<
		"\n\t<3> for rotation about y axis" <<
		"\n\t<4> for rotation about z axis" <<
			"\n\t<5> for scaling" <<
			"\n\t<6> for new object\n\t: ";
```

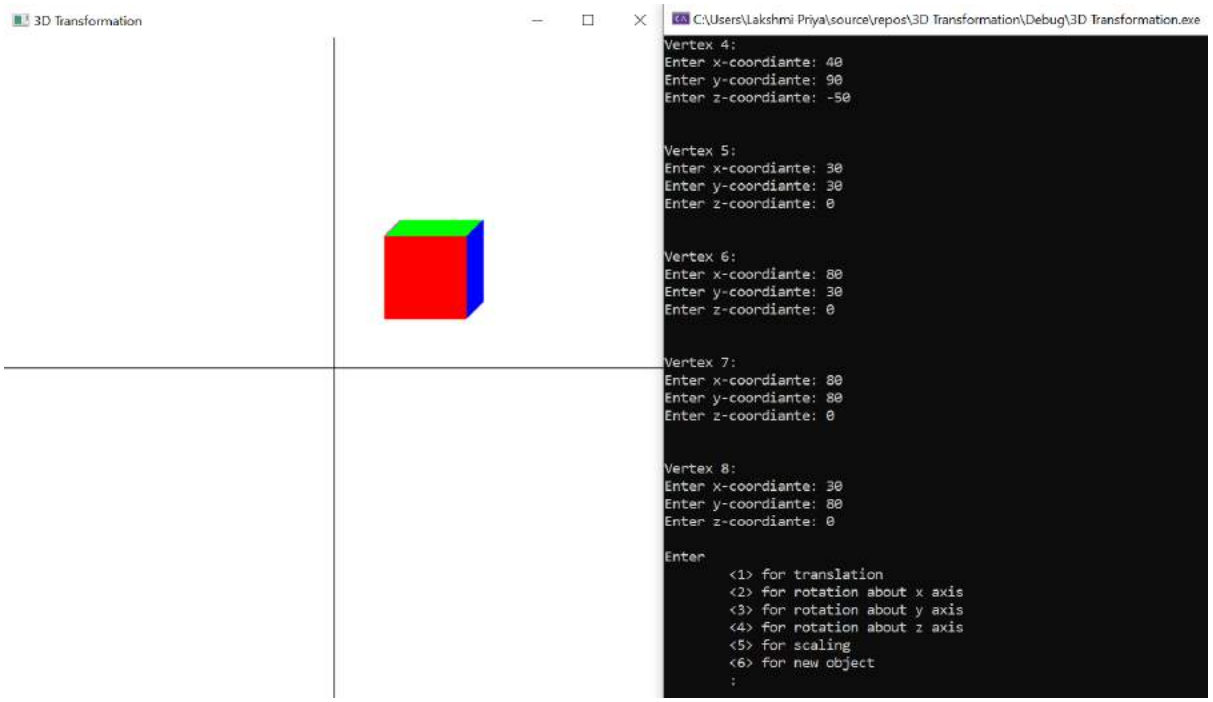
```
                        cin >> opt;

                        switch (opt) {
                        case 1:
                               cout << "\nEnter translation along x: ";
                               cin >> tx;
                               cout << "Enter translation along y: ";
                               cin >> ty;
                               cout << "Enter translation along z: ";
                               cin >> tz;
                               translate3D(tx, ty, tz);
                               glColor3f(0.0, 1.0, 0.0);
                               break;
                        case 2:
                               cout << "\nEnter rotation angle (in degrees): ";
                               cin >> theta;
                               rotate3D_x_axis(theta * pi / 180);
                               glColor3f(0.0, 0.0, 1.0);
                               break;
                case 3:
                    cout << "\nEnter rotation angle (in degrees): ";
                    cin >> theta;
                    rotate3D_y_axis(theta * pi / 180);
                    glColor3f(0.0, 0.0, 1.0);
                    break;
                case 4:
                    cout << "\nEnter rotation angle (in degrees): ";
                    cin >> theta;
                    rotate3D_z_axis(theta * pi / 180);
                    glColor3f(0.0, 0.0, 1.0);
                    break;
                        case 5:
                               cout << "\nEnter scaling factor along x: ";
                               cin >> sx;
                               cout << "Enter scaling factor along y: ";
                               cin >> sy;
                               cout << "Enter scaling factor along z: ";
                               cin >> sz;
                               scale3D(sx, sy, sz, fixedPt);
                               glColor3f(1.0, 1.0, 0.0);
                               break;
                        case 6:
                               break;
                        default:
                               cout << "Enter valid option!!" << endl;
                               continue;
                        }

                        if (opt == 6)
                               break;

                        transformVerts3D(nVerts, verts, plotverts);
                        drawObject(plotverts);
                        glFlush();
                  }
        }
}

void winReshapeFcn(GLint newWidth, GLint newHeight)
{
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
```

```
        glOrtho(xwcMin, xwcMax, ywcMin, ywcMax, zwcMin, zwcMax);
        // To Render the surfaces Properly according to their depths
        glEnable(GL_DEPTH_TEST);
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
}

int main(int argc, char** argv)
{
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
        glutInitWindowPosition(50, 50);
        glutInitWindowSize(winWidth, winHeight);
        glutCreateWindow("3D Transformation");
        init();
        glutDisplayFunc(displayFcn);
        glutReshapeFunc(winReshapeFcn);
        glutMainLoop();
        return 0;
}
```

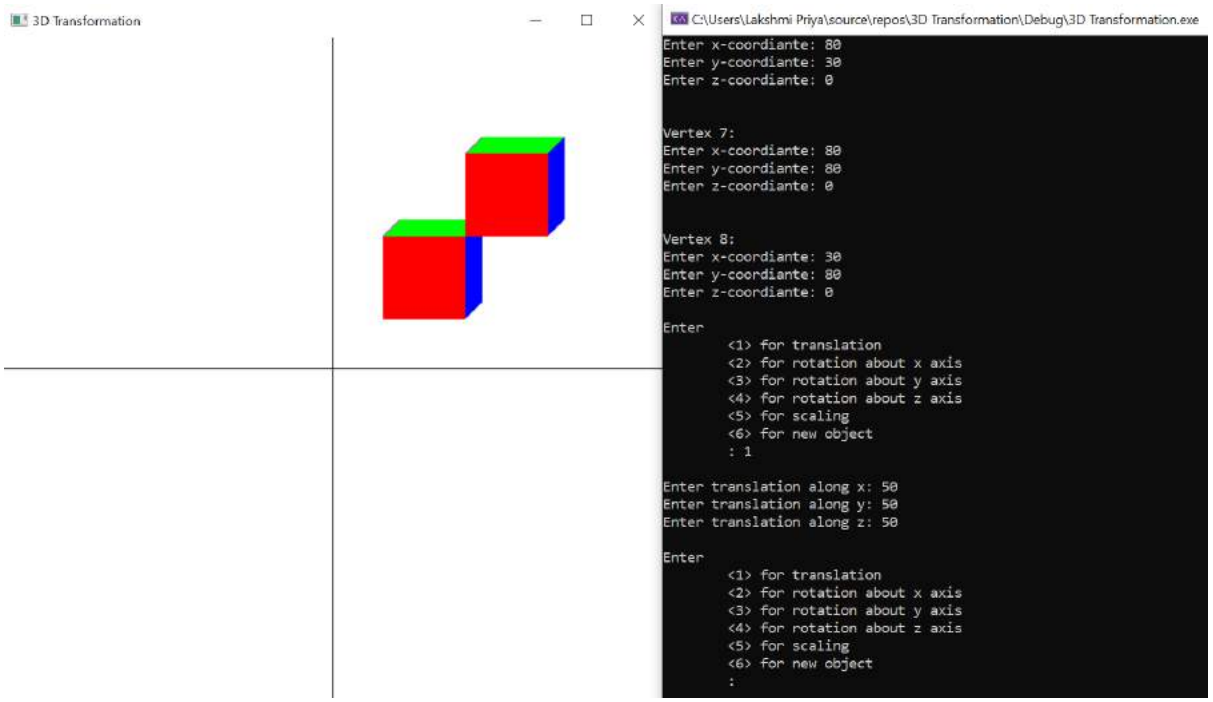
Output Screenshot

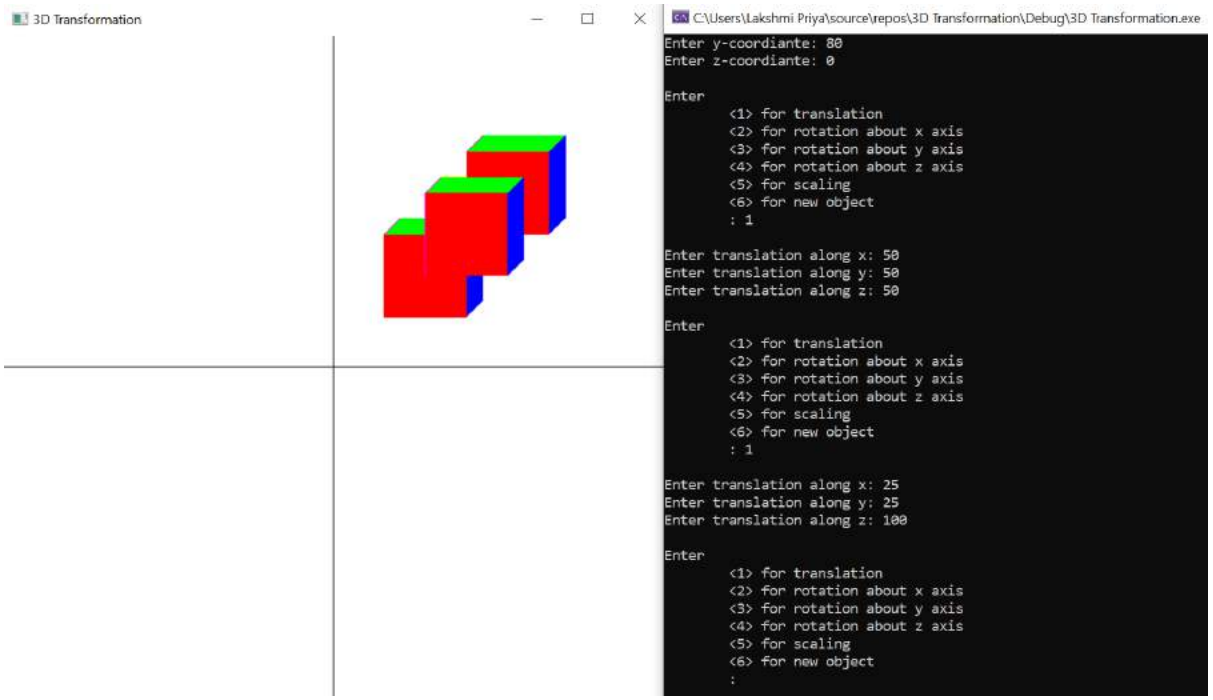
Original Object



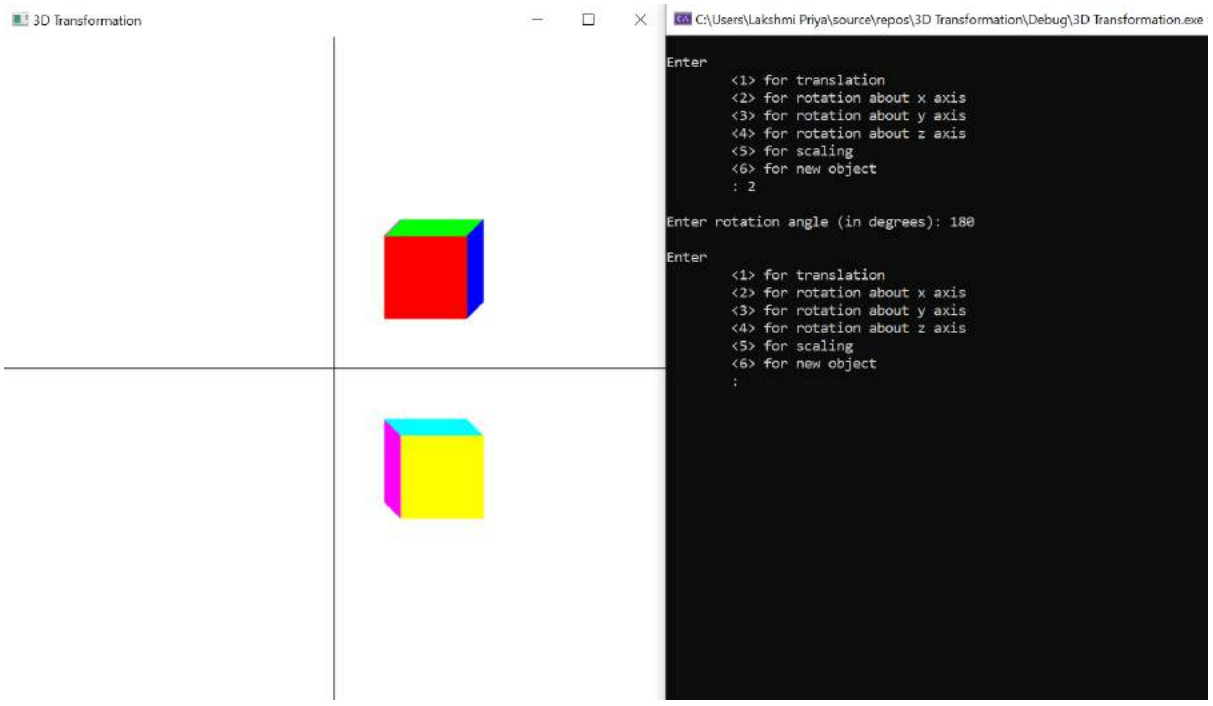


Case 1: Translation

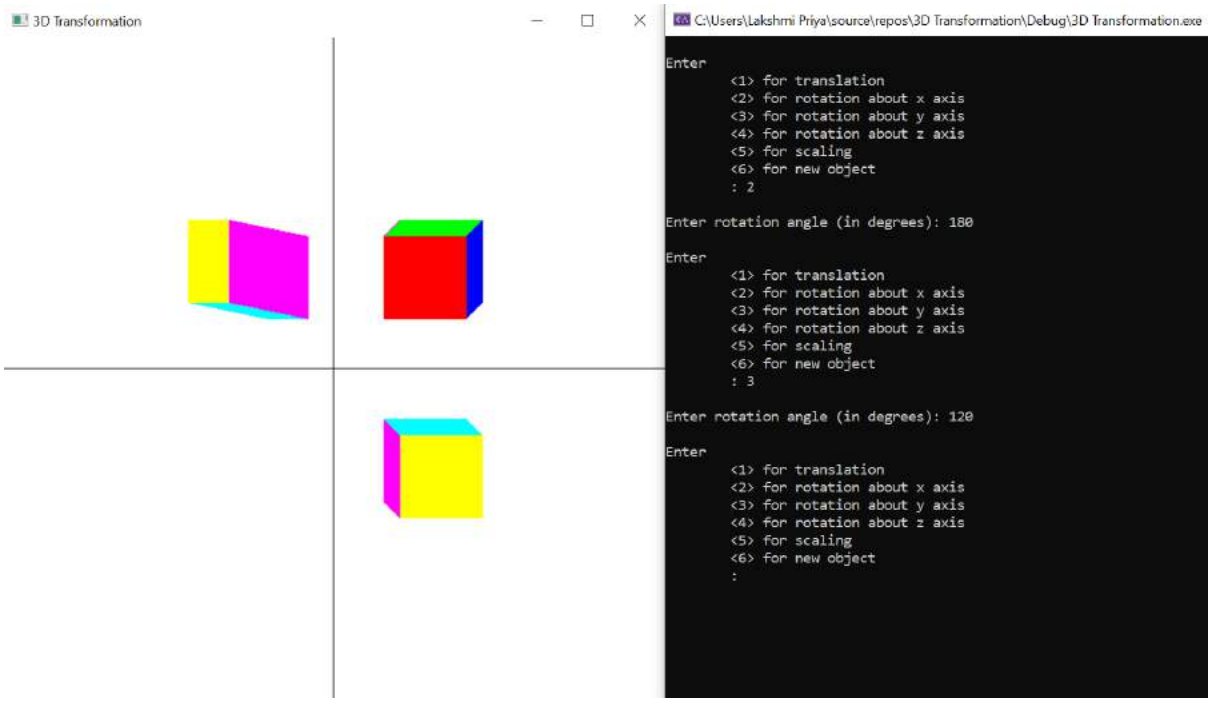




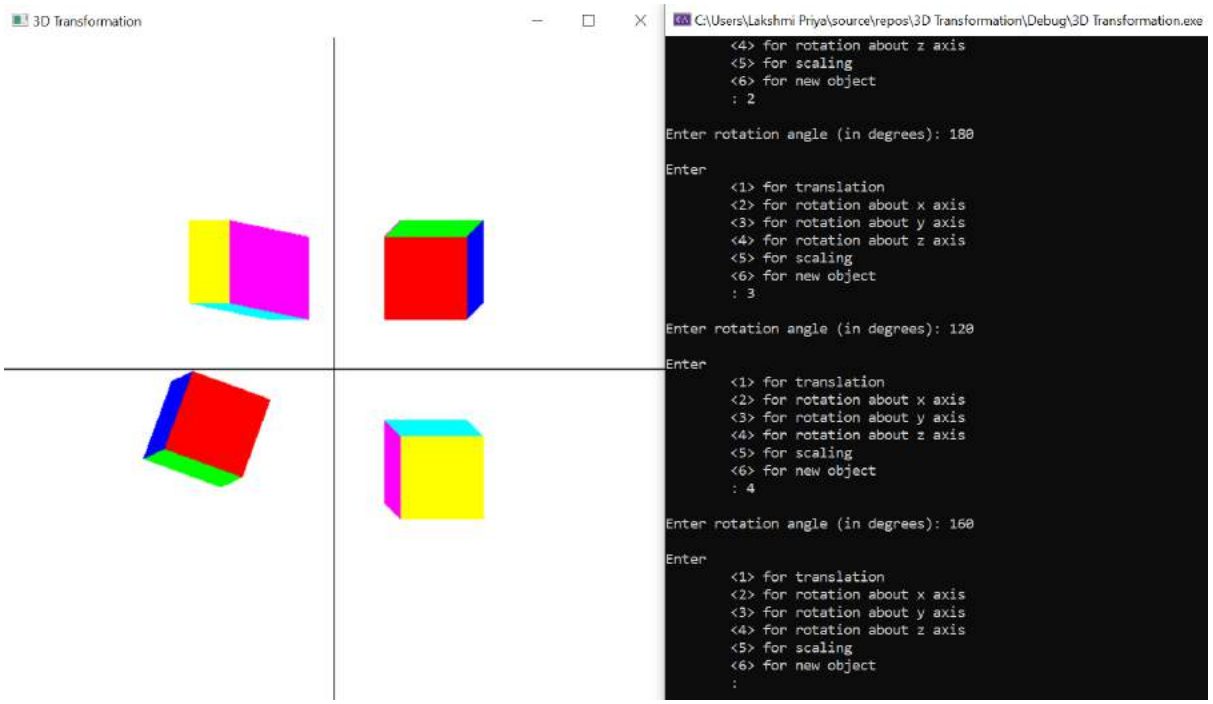
Case 2a: Rotation along x axis



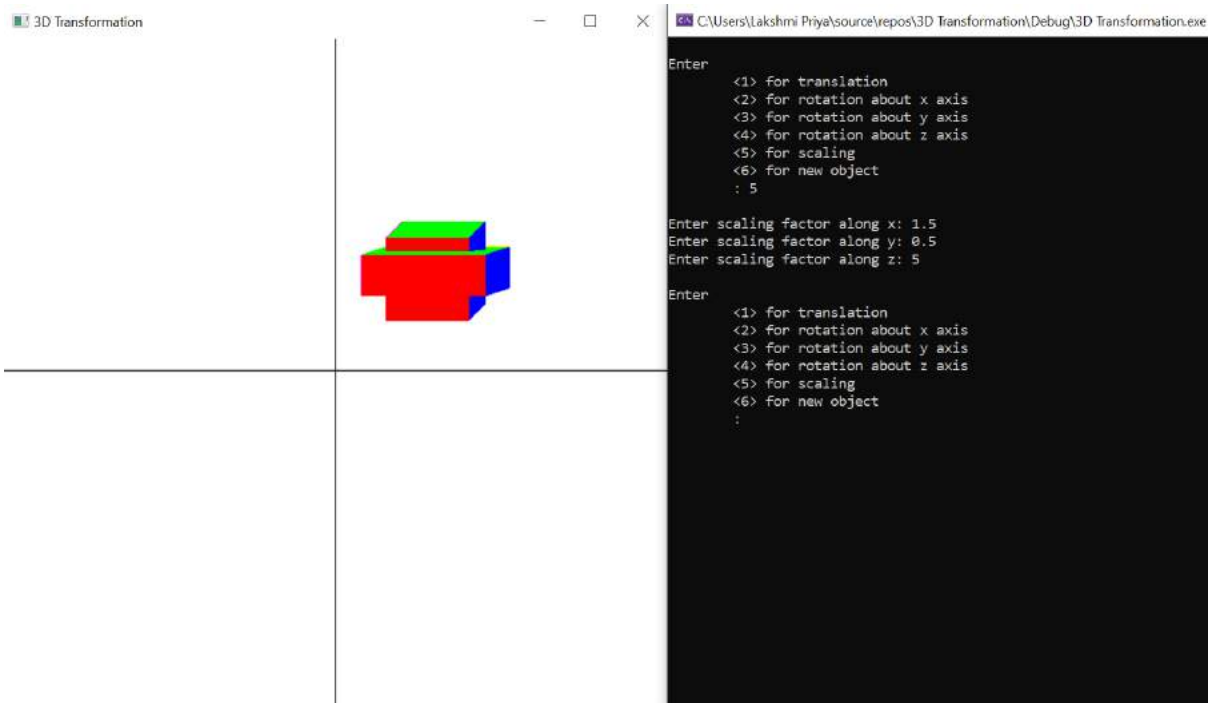
Case 2b: Rotation along y axis



Case 2c: Rotation along z axis



Case 3: Scaling



Result

Thus, a C++ menu-driven program has been written using OPENGL to perform 3D transformations – translation, scaling and rotation along all three axes.

SSN COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UCS1712 – GRAPHICS AND MULTIMEDIA LAB

EX NO: 9 – 3D Projections

Name : **Lakshmi Priya B**
Register Number : **185001083**
Date : **11-10-2021**

Aim

To write a C++ program using OPENGL to perform 3D Projections – Orthographic and Perspective.

Algorithm

1. Define the display1() function:
    a. Set the background color as white using glClearColor()
    b. Display the window as the color of the current buffer using GL_COLOR_BUFFER_BIT
    c. Set the color of the object to be drawn using glColor3f()
    d. Render a wireframe teapot of size 1 using glutWireTeapot(1)
    e. Send all the output to display using glFlush()
2. Define the display2() function:
    a. Set the background color as white using glClearColor()
    b. Display the window as the color of the current buffer using GL_COLOR_BUFFER_BIT
    c. Set the color of the object to be drawn using glColor3f()
    d. Rotate and translate the world around to look like the camera moved using gluLookAt()
    e. Render a wireframe teapot of size 1 using glutWireTeapot(1)
    f. Send all the output to display using glFlush()
3. Define the reshape1() function:
    a. Set the viewport using glViewport()
    b. Use glOrtho() for orthographic projection
4. Define the reshape2() function:
    a. Set the viewport using glViewport()
    b. Use glFrustum() for perspective projection
5. Define the main() function:
    a. Initialize using the glutInit() function with &argc and argv as parameters
    b. Specify the display mode as GLUT_SINGLE and GLUT_RGB
    c. Specify the first window position as 100, 100
    d. Specify the first window size as 500, 500
    e. Create first window and set the title as “3D Projections - Orthographic”
    f. Register display1 as display callback function for “3D Projections - Orthographic”
    g. Register reshape1 as reshape callback function for “3D Projections - Orthographic”

h. Specify the second window position as 700, 100
i. Create second window and set the title as “3D Projections - Perspective”
j. Register display2 as display callback function for “3D Projections - Perspective”
k. Register reshape2 as reshape callback function for “3D Projections - Perspective”
l. Go into a loop using glutMainLoop() until event occurs

6. Run the application to view the projected image of the object in both orthographic and perspective projections.

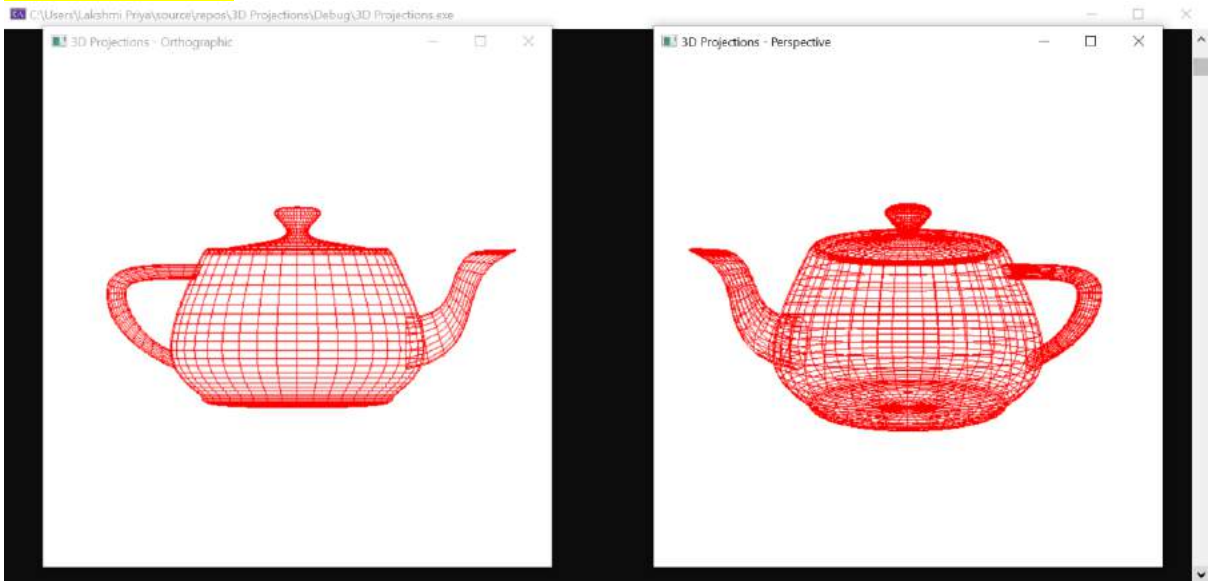
Code

Source.cpp:

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

void display1() {
    glClearColor(1, 1, 1, 0);
    glClear(GL_COLOR_BUFFER_BIT);
    glLoadIdentity();

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

    // render a wireframe teapot of size 1
    glutWireTeapot(1);
    glFlush();

}

void display2() {
    glClearColor(1, 1, 1, 0);
    glClear(GL_COLOR_BUFFER_BIT);
    glLoadIdentity();

    // Rotate and translate the world around to look like the camera moved.
    // gluLookAt(<#GLdouble eyeX#>, <#GLdouble eyeY#>, <#GLdouble eyeZ#>, <#GLdouble
centerX#>, <#GLdouble centerY#>, <#GLdouble centerZ#>, <#GLdouble upX#>, <#GLdouble
upY#>, <#GLdouble upZ#>)
    gluLookAt(0.0, 0.0, -3.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);


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

    // render a wireframe teapot of size 1
    glutWireTeapot(1);
    glFlush();

}

static void reshape1(int w, int h) {

    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-2.0, 2.0, -2.0, 2.0, -1.5, 1.5);

    glMatrixMode(GL_MODELVIEW);
    glFlush();
}
```

```
static void reshape2(int w, int h) {
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();

    glFrustum(-1.0, 1.0, -1.0, 1.0, 1.5, 20.0);

    glMatrixMode(GL_MODELVIEW);
    glFlush();
}

int main(int argc, char* argv[]) {
    glutInit(&argc, argv);

    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("3D Projections - Orthographic");
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glShadeModel(GL_FLAT);
    glutDisplayFunc(display1);
    glutReshapeFunc(reshape1);

    glutInitWindowPosition(700, 100);
    glutCreateWindow("3D Projections - Perspective");
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glShadeModel(GL_FLAT);
    glutDisplayFunc(display2);
    glutReshapeFunc(reshape2);

    glutMainLoop();
    return 1;
}
```

Output Screenshot



Result

Thus, 3D Orthographic and Perspective projections have been done using OPENGL.

SSN COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UCS1712 – GRAPHICS AND MULTIMEDIA LAB

EX NO: 10 – Image Editing and Manipulation

Name : **Lakshmi Priya B**
Register Number : **185001083**
Date : **16-10-2021**

Aim

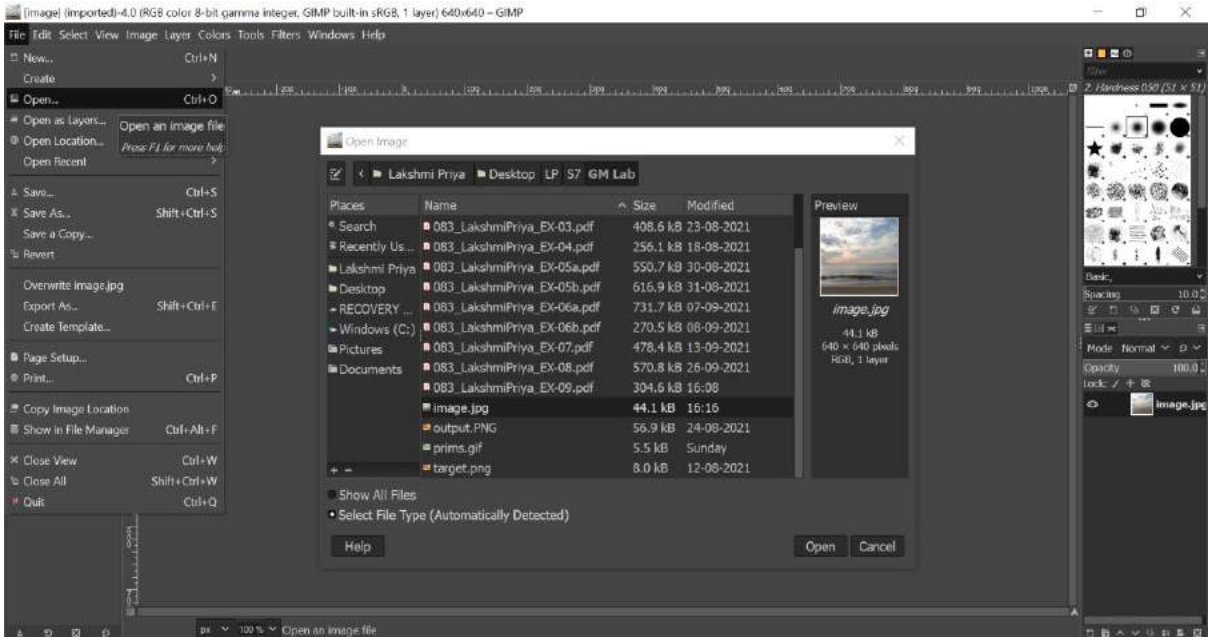
(i) To perform basic operations on an image like retouch image, add filters, masks, etc. using GIMP software to enhance it.
(ii) To create a simple gif animated image.

Algorithm

1. For basic image operations like image retouching, image filtering and image masks:
    a. Open the image to be edited or manipulated in GIMP by going to File → Open
    b. View the opened image in the image window
    c. Retouch the image by changing brightness and contrast, colorizing the image, etc.
    d. Apply blur, pixelize filters
    e. Light and shadow filters can also be applied
    f. Add layer mask if needed
2. For creating gif animated image:
    a. Choose any image which has to be made into an animated gif
    b. Open the image and view it in the image window
    c. Right click on the image in the Layers Dialog and select Duplicate Layer option to create a duplicate of the layer
    d. Create a depth map to animate the image by creating duplicate layers of the image and denoting which parts of the image are closer to the viewer by filling with lighter shades and which parts are farther to the viewer by filling with darker shades
    e. Finally merge all this duplicate layers to get a final depth map
    f. Blur the generated depth map using Gaussian Blurring technique to perform an averaging of neighbouring pixels with the normal distribution as weighting
    g. Create multiple animation frames of a single image by using Displace Mapping technique which uses the depth map generated earlier to perform the horizontal and vertical displacement
    h. To get preview of animation go to Filters → Animation → Playback
    i. To save the gif, go to File → Export As and save as animation with loop forever feature

Steps & Output Screenshot

Opening image:



Original image:

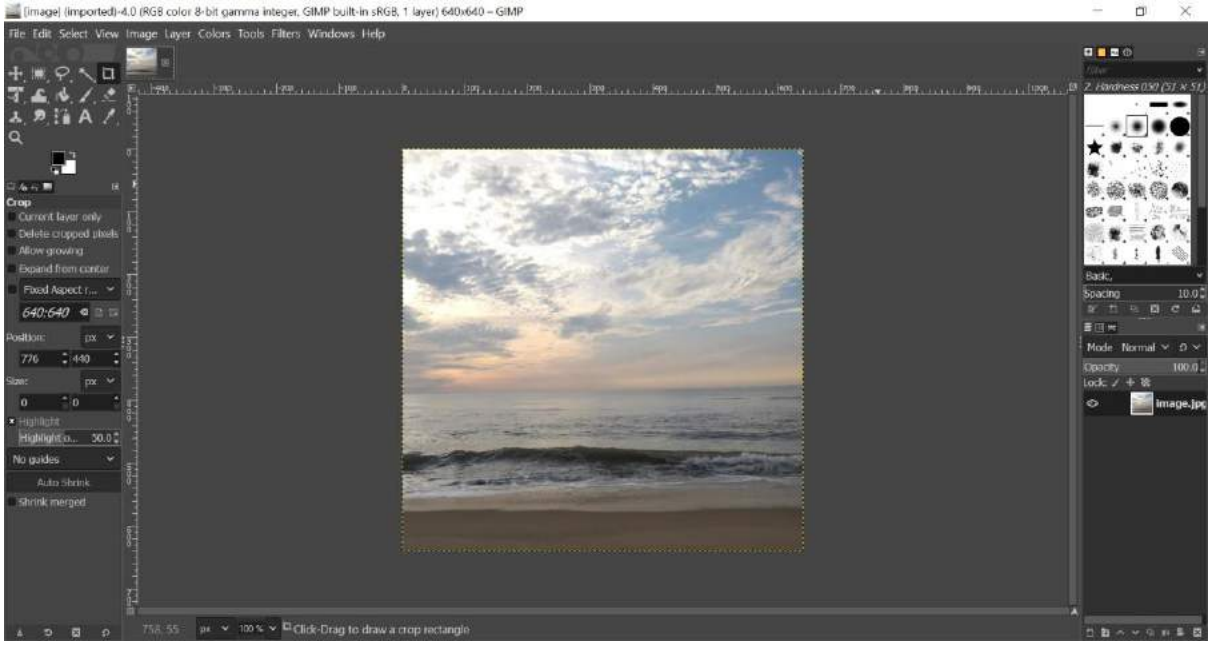
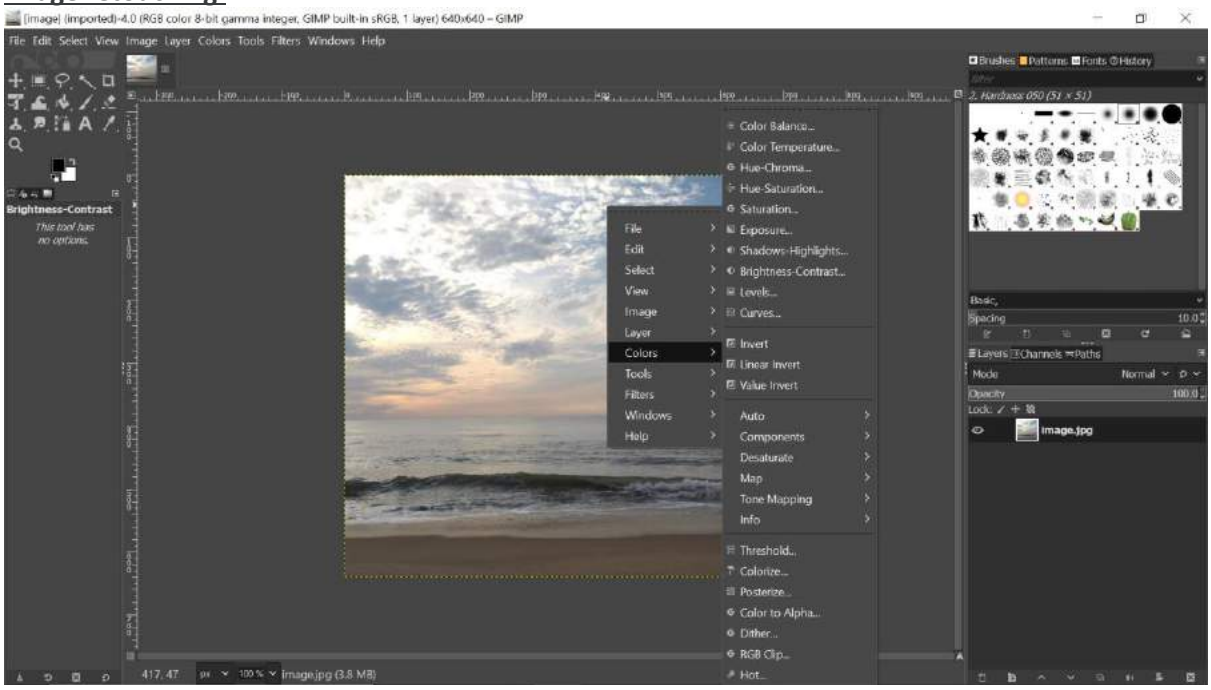
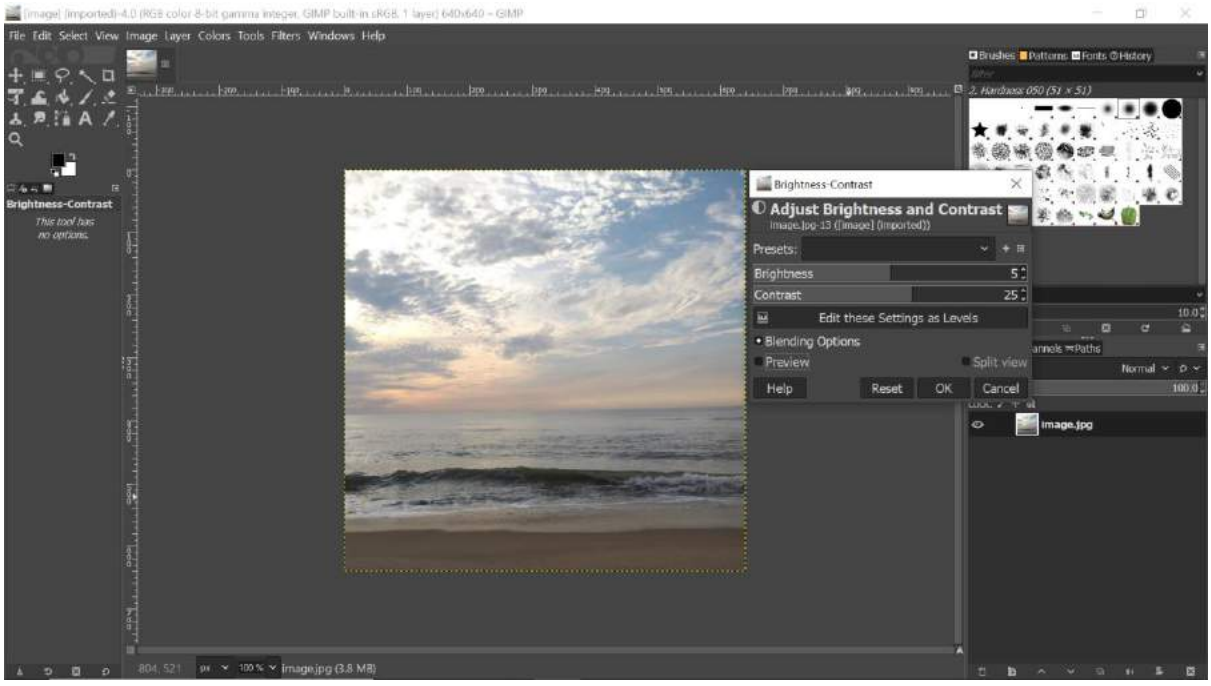
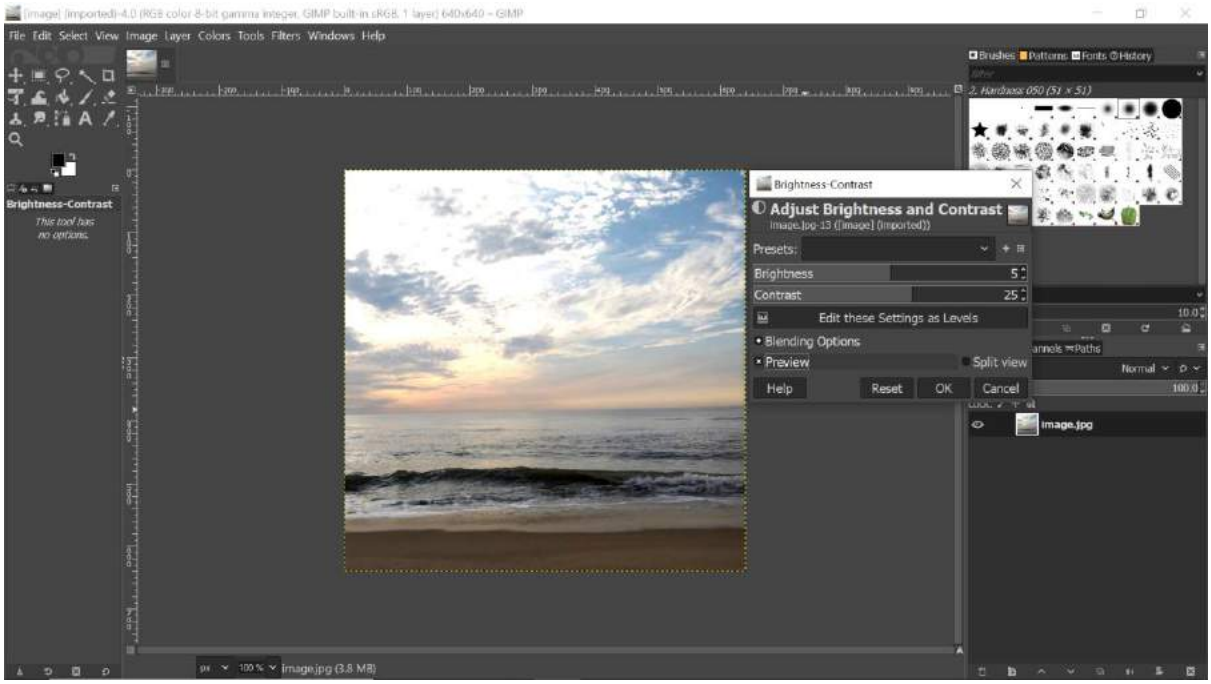
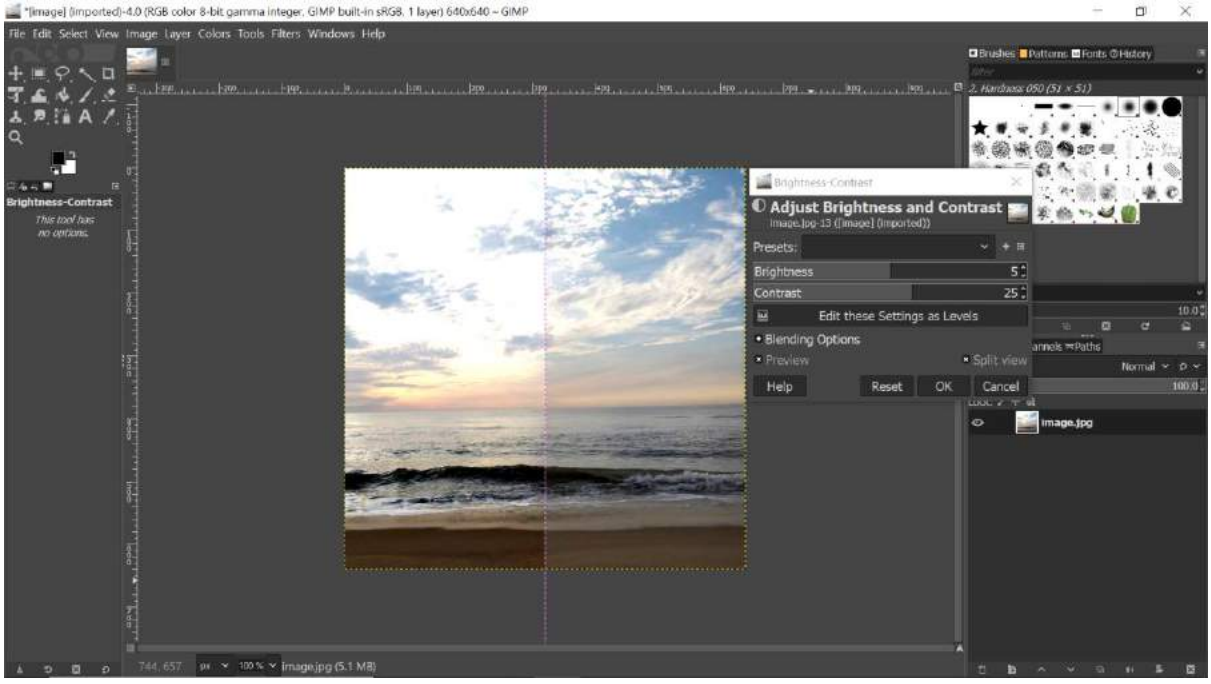


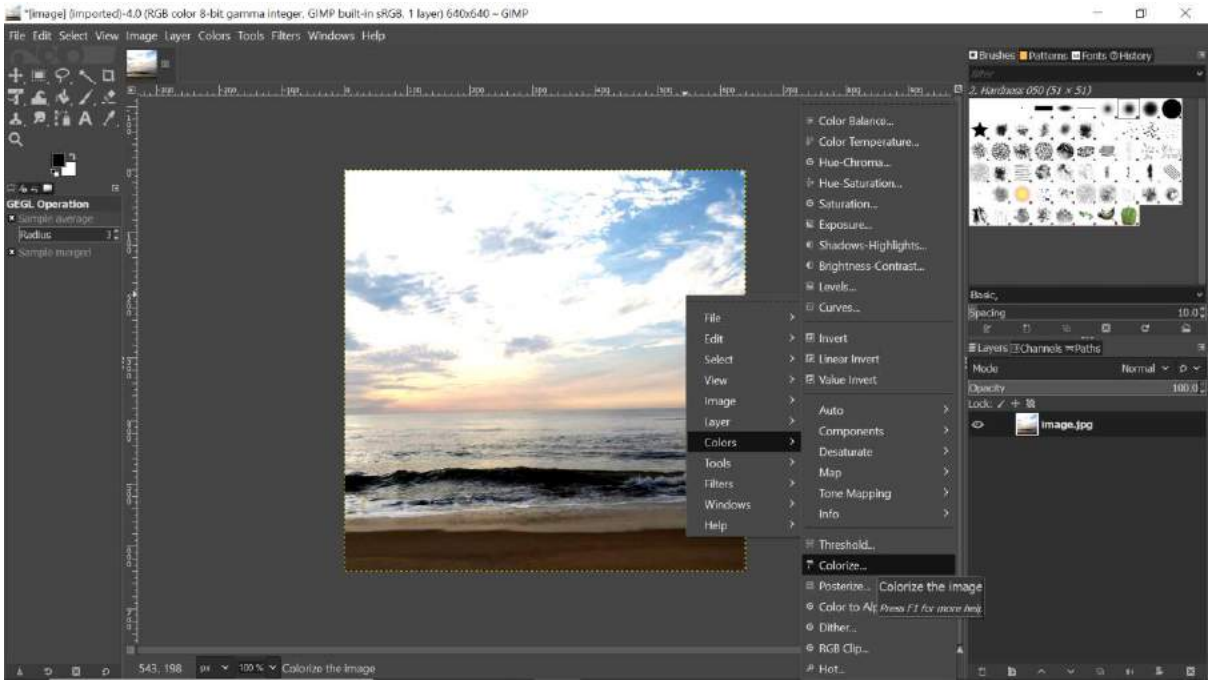
Image retouching:

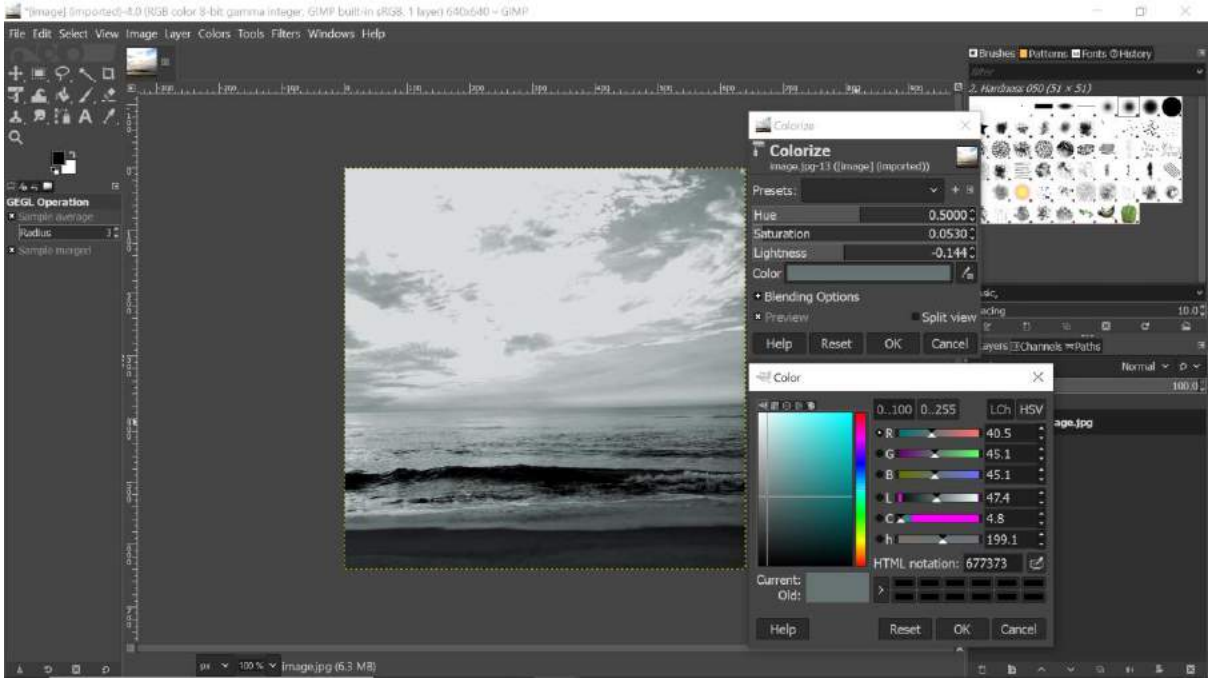





[Image] (Imported)-4.0 (RGB color 8-bit gamma integer, GIMP built-in sRGB, 1 layer) 640x640 – GIMP
File Edit Select View Image Layer Colors Tools Filters Windows Help
Brightness-Contrast
This tool has no options.
Brightness-Contrast
Adjust Brightness and Contrast
image.jpg-13 ([Image] (Imported))
Presets:
Brightness 5
Contrast 25
Edit these Settings as Levels
Blending Options
Preview
Split view
Help
Reset
OK
Cancel
Brushes Patterns Fonts History
2. Hardness 050 (51 × 51)
Normal
image.jpg
image.jpg (3.8 MB)


*[Image] (Imported)-4.0 (RGB color 8-bit gamma integer, GIMP built-in sRGB, 1 layer) 640x640 – GIMP
File Edit Select View Image Layer Colors Tools Filters Windows Help
Brightness-Contrast
This tool has no options.
Brightness-Contrast
Adjust Brightness and Contrast
image.jpg-13 ([Image] (Imported))
Presets:
Brightness 5
Contrast 25
Edit these Settings as Levels
Blending Options
Preview
Split view
Help
Reset
OK
Cancel
Brushes Patterns Fonts History
2. Hardness 050 (51 × 51)
Normal
image.jpg
744, 657
image.jpg (5.1 MB)


File Edit Select View Image Layer Colors Tools Filters Windows Help
Color Balance...
Color Temperature...
Hue-Chroma...
Hue-Saturation...
Saturation...
Exposure...
Shadows-Highlights...
Brightness-Contrast...
Levels...
Curves...
Invert
Linear Invert
Value Invert
Auto
Components
Desaturate
Map
Tone Mapping
Info
Threshold...
Colorize...
Posterize...
Dither...
RGB Clip...
Colorize the image
Press F1 for more help
GEGL Operation
Sample average
Radius
Sample merged
Layers Channels Paths
Mode
Normal
Opacity
Lock:
image.jpg
543, 198
Colorize the image


File Edit Select View Image Layer Colors Tools Filters Windows Help
Colorize
image.jpg-13 ([image] (imported))
Presets:
Hue 0.5000
Saturation 0.0530
Lightness -0.144
Color
Blending Options
Preview
Split view
Help
Reset
OK
Cancel
Color
0..100
0..255
LCh
HSV
R 40.5
G 45.1
B 45.1
L 47.4
C 4.8
h 199.1
HTML notation: 677373
Current:
Old:
GEGL Operation
Sample average
Radius
Sample merged
Layers Channels Paths
image.jpg (6.3 MB)

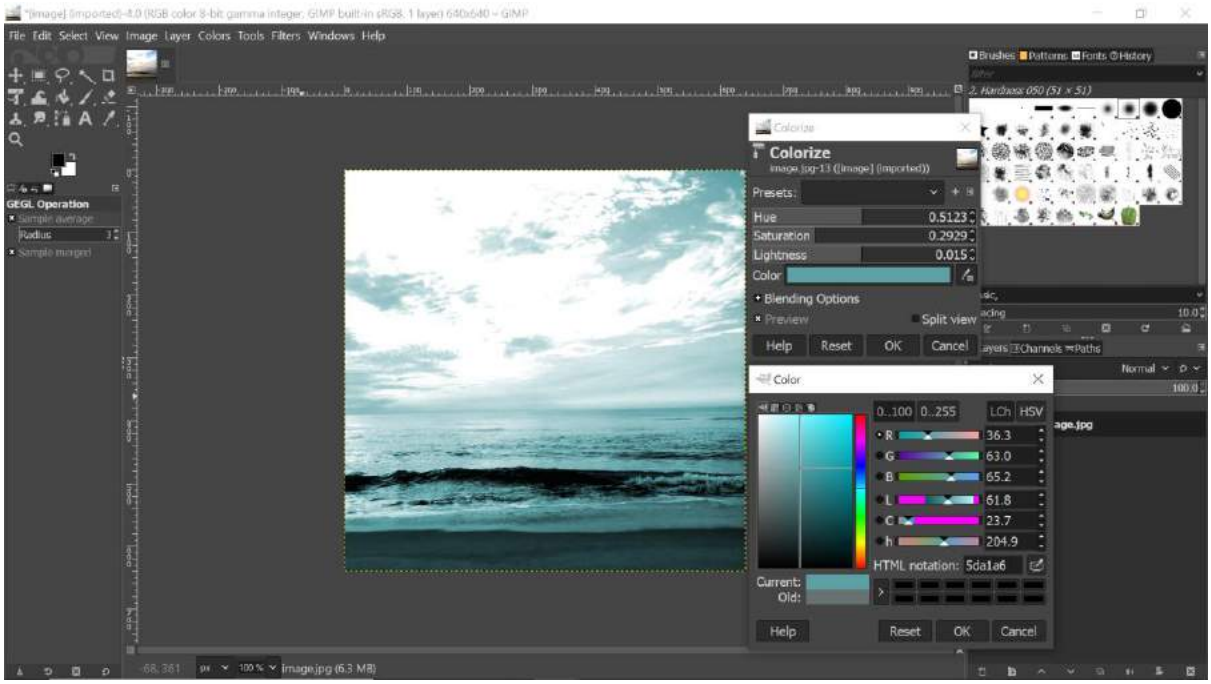
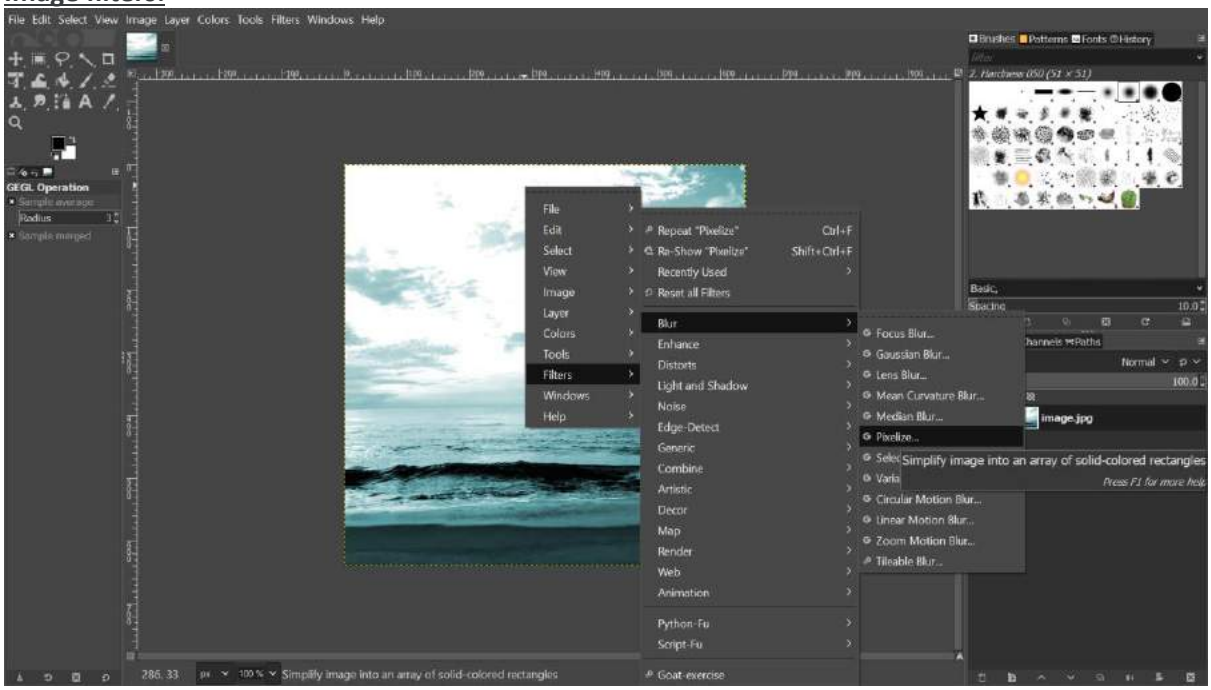
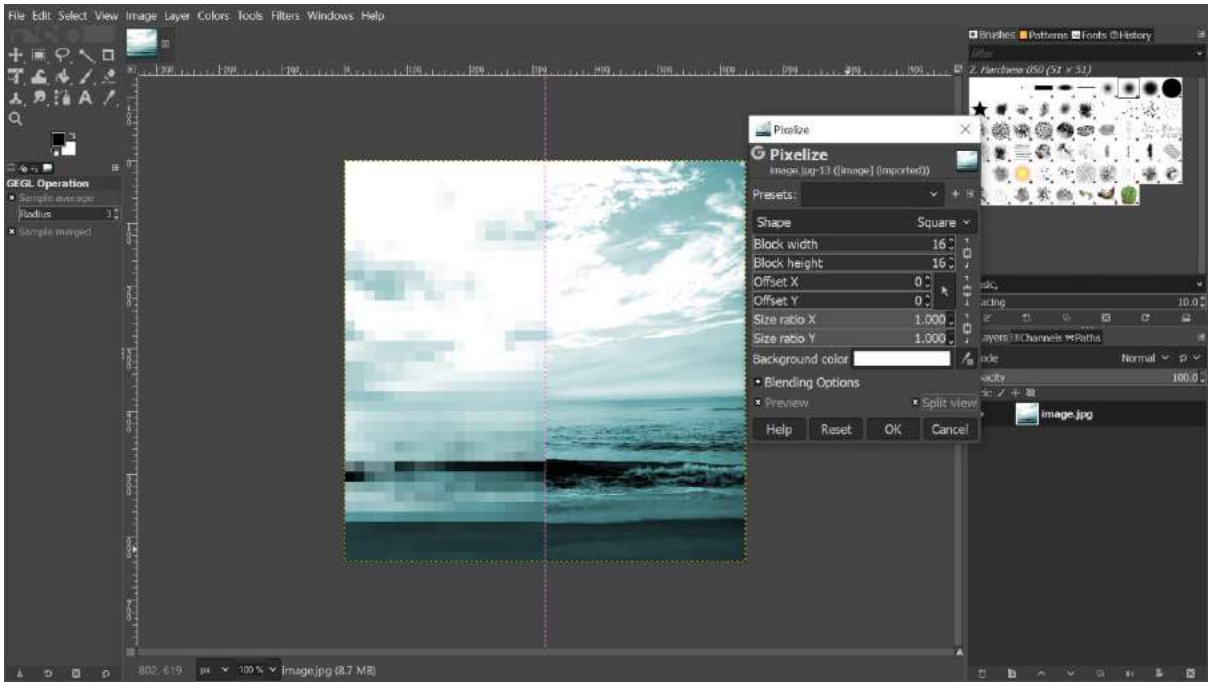
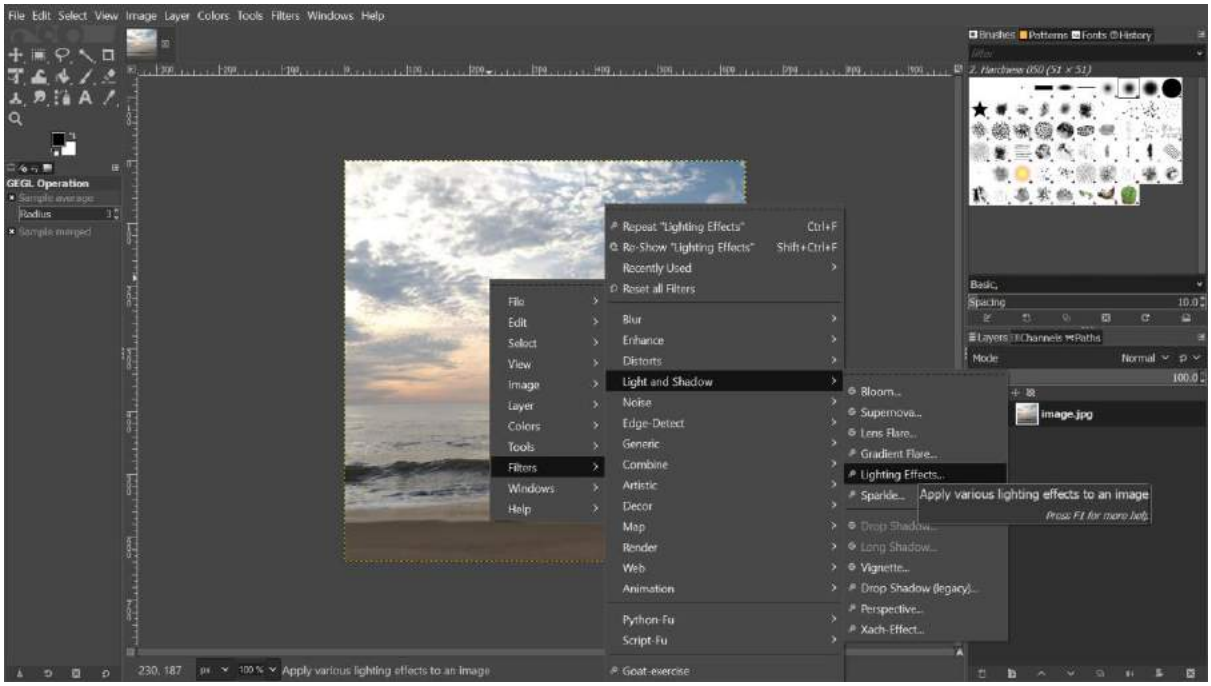
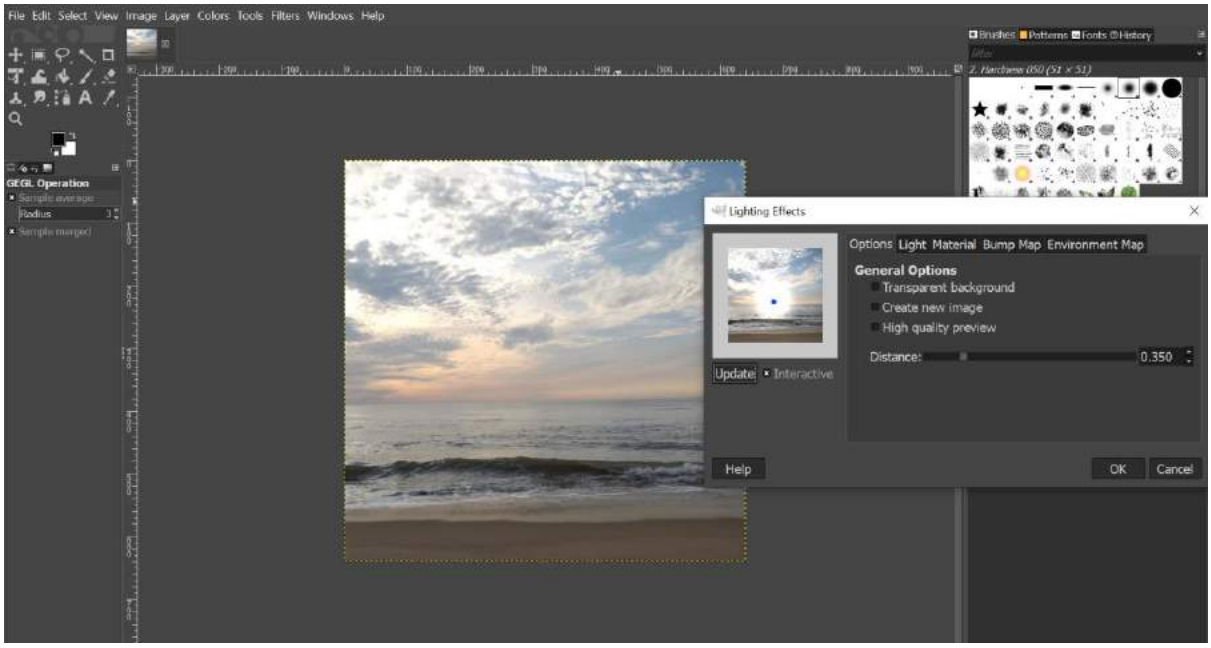


Image filters:




File Edit Select View Image Layer Colors Tools Filters Windows Help
Pixelize
Pixelize
image.jpg-13 ([image] (imported))
Presets:
Shape
Square
Block width
16
Block height
16
Offset X
0
Offset Y
0
Size ratio X
1.000
Size ratio Y
1.000
Background color
Blending Options
Preview
Split view
Help
Reset
OK
Cancel
GEGL Operation
Radius
Brushes
Patterns
Fonts
History
Layers
Channels
Paths
Normal
image.jpg
802, 619
image.jpg (8.7 MB)


File Edit Select View Image Layer Colors Tools Filters Windows Help
Repeat "Lighting Effects" Ctrl+F
Re-Show "Lighting Effects" Shift+Ctrl+F
Recently Used
Reset all Filters
File
Edit
Select
View
Image
Layer
Colors
Tools
Filters
Windows
Help
Blur
Enhance
Distorts
Light and Shadow
Noise
Edge-Detect
Generic
Combine
Artistic
Decor
Map
Render
Web
Animation
Python-Fu
Script-Fu
Goat-exercise
Bloom...
Supernova...
Lens Flare...
Gradient Flare...
Lighting Effects...
Sparkle...
Drop Shadow...
Long Shadow...
Vignette...
Drop Shadow (legacy)...
Perspective...
Xach-Effect...
Apply various lighting effects to an image
Press F1 for more help
GEGL Operation
Radius
Brushes
Patterns
Fonts
History
Layers
Channels
Paths
Normal
image.jpg
230, 187
Apply various lighting effects to an image


File Edit Select View Image Layer Colors Tools Filters Windows Help
Brushes Patterns Fonts History
2. Hardness 050 (51 × 51)
GEGL Operation
Sample average
Radius
Sample merged
Lighting Effects
Options Light Material Bump Map Environment Map
General Options
Transparent background
Create new image
High quality preview
Distance:
0.350
Update
Interactive
Help
OK
Cancel

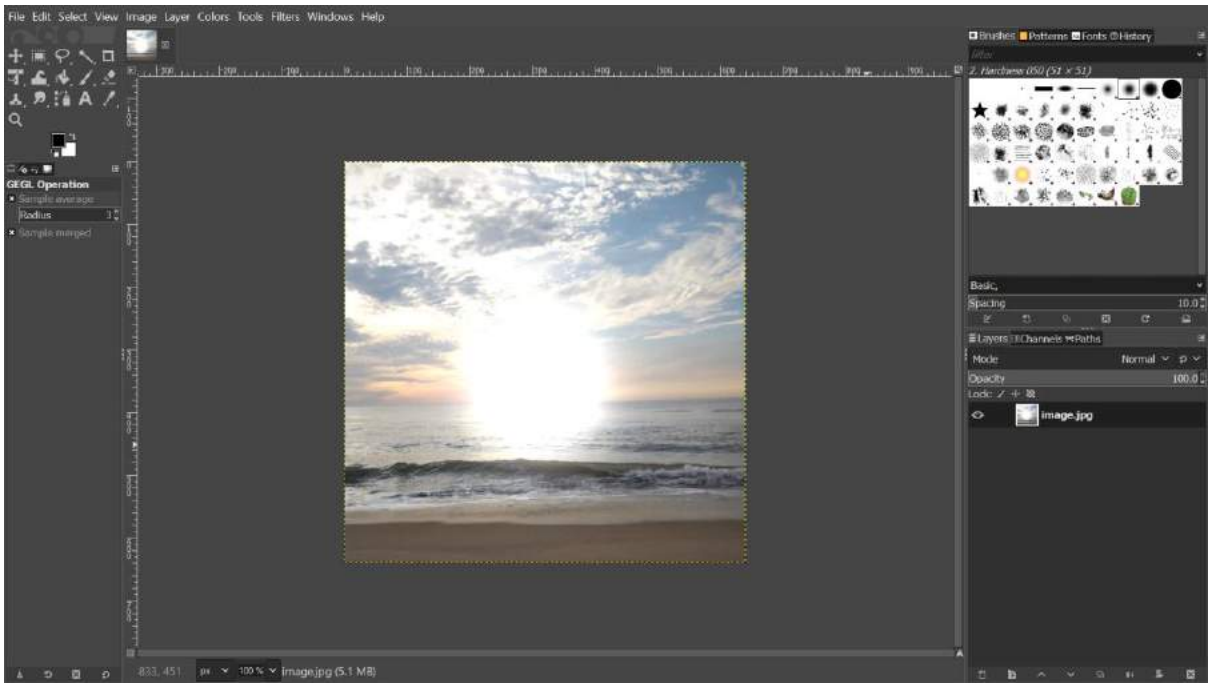
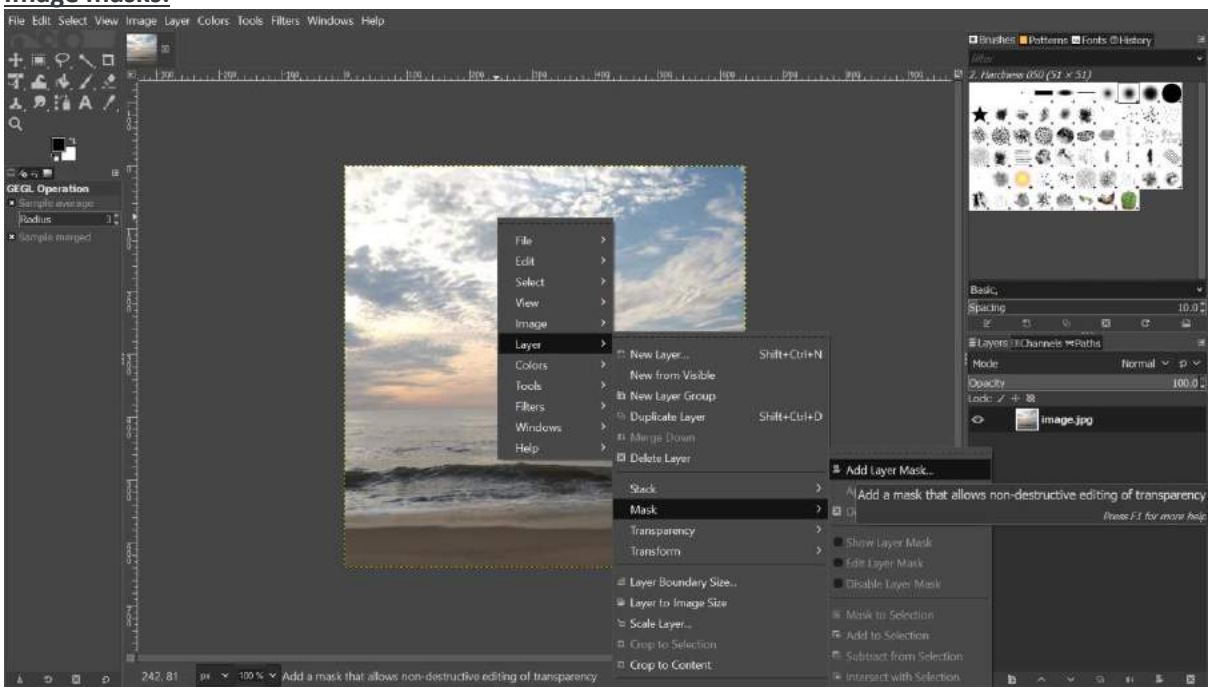
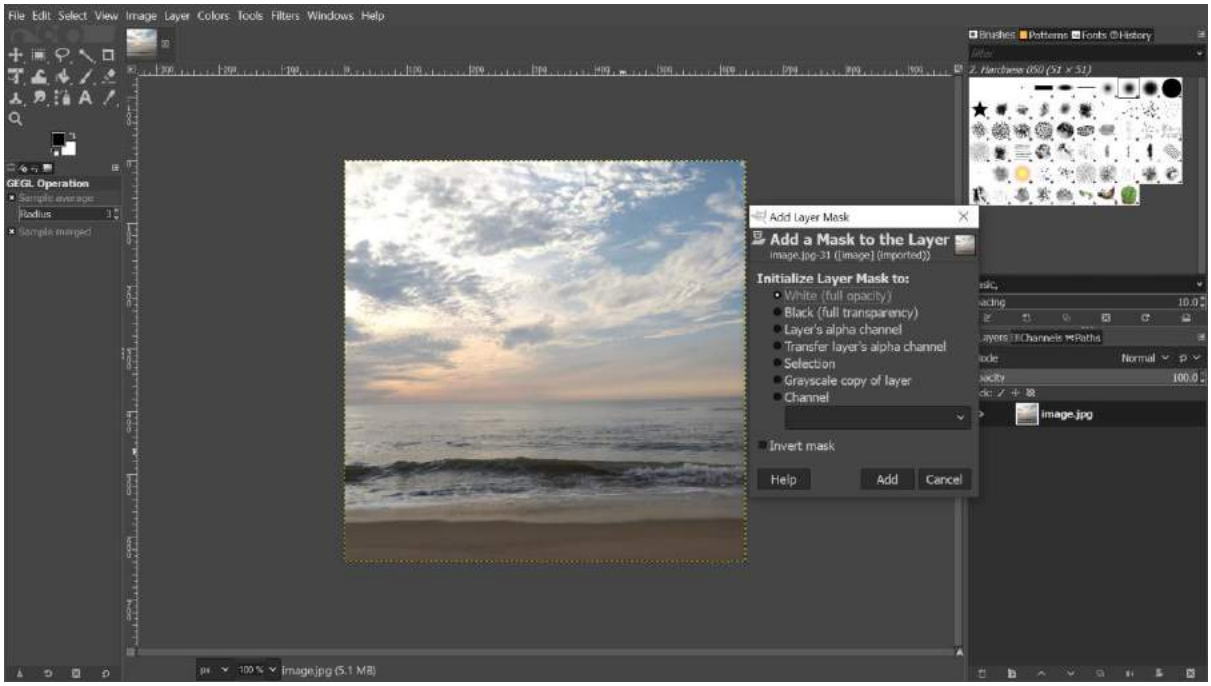
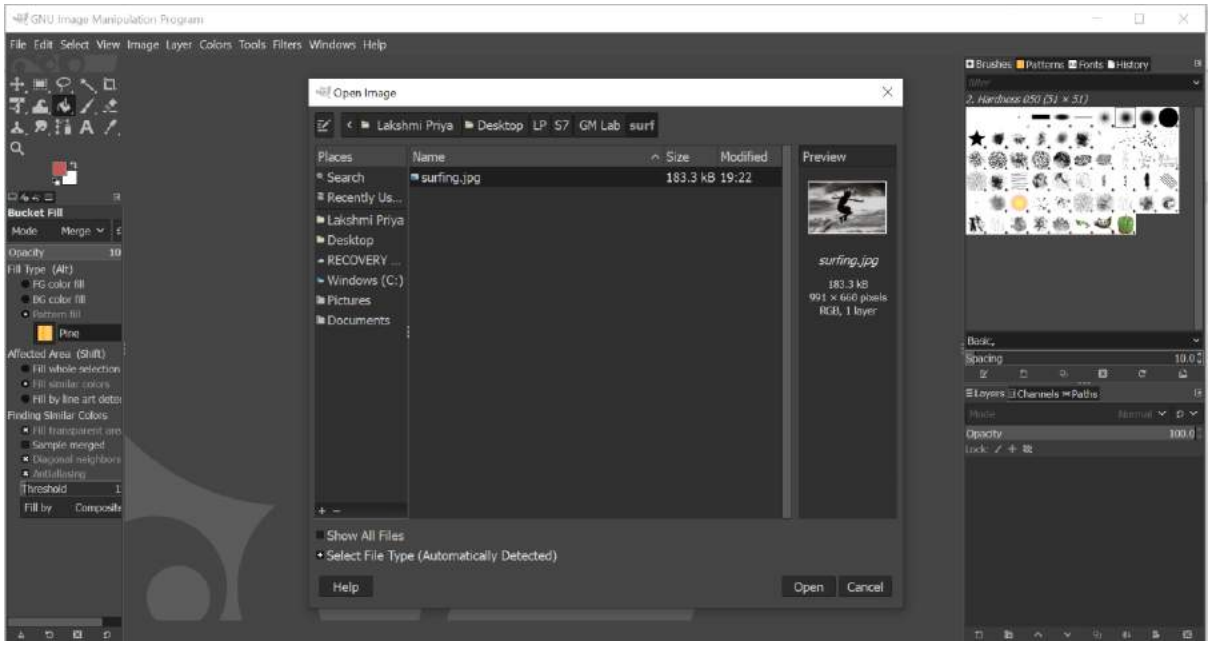

File Edit Select View Image Layer Colors Tools Filters Windows Help
Brushes Patterns Fonts History
2. Hardness 050 (51 × 51)
GEGL Operation
Sample average
Radius
Sample merged
Basic,
Spacing
10.0
Layers Channels Paths
Mode
Normal
Opacity
100.0
Lock:
image.jpg
833, 451
100 %
image.jpg (5.1 MB)

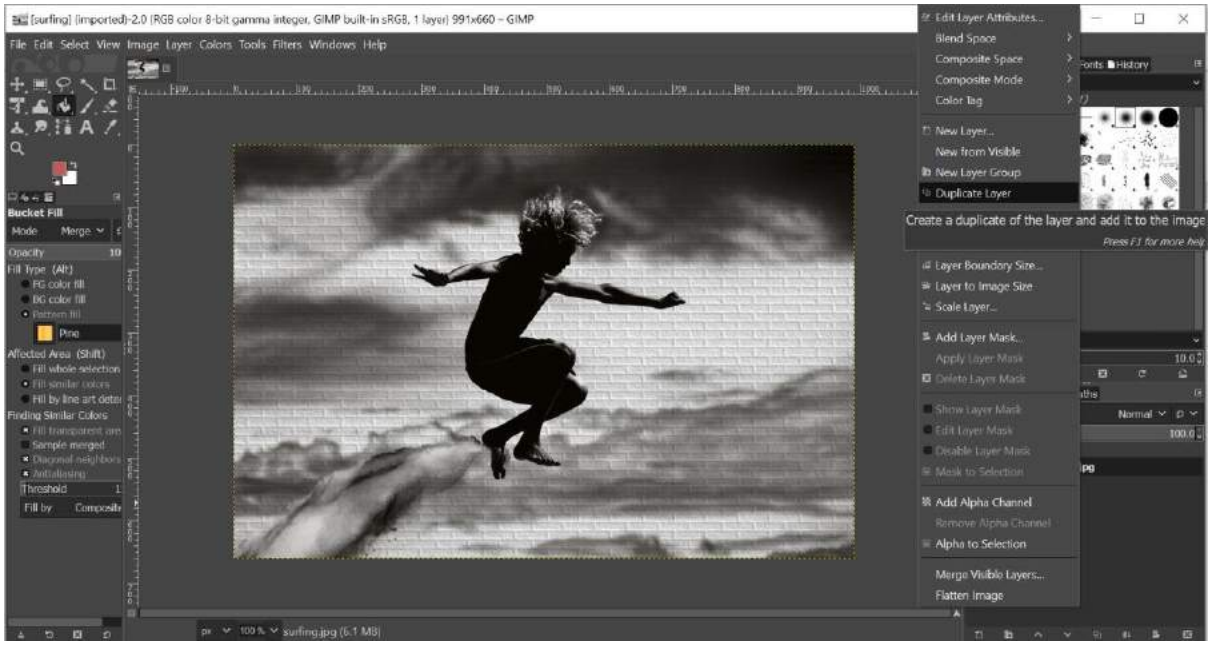
Image masks:


File Edit Select View Image Layer Colors Tools Filters Windows Help
File
Edit
Select
View
Image
Layer
Colors
Tools
Filters
Windows
Help
New Layer... Shift+Ctrl+N
New from Visible
New Layer Group
Duplicate Layer Shift+Ctrl+D
Merge Down
Delete Layer
Stack
Mask
Transparency
Transform
Layer Boundary Size...
Layer to Image Size
Scale Layer...
Crop to Selection
Crop to Content
Add Layer Mask...
Add a mask that allows non-destructive editing of transparency
Show Layer Mask
Edit Layer Mask
Disable Layer Mask
Mask to Selection
Add to Selection
Subtract from Selection
Intersect with Selection
image.jpg
Add a mask that allows non-destructive editing of transparency

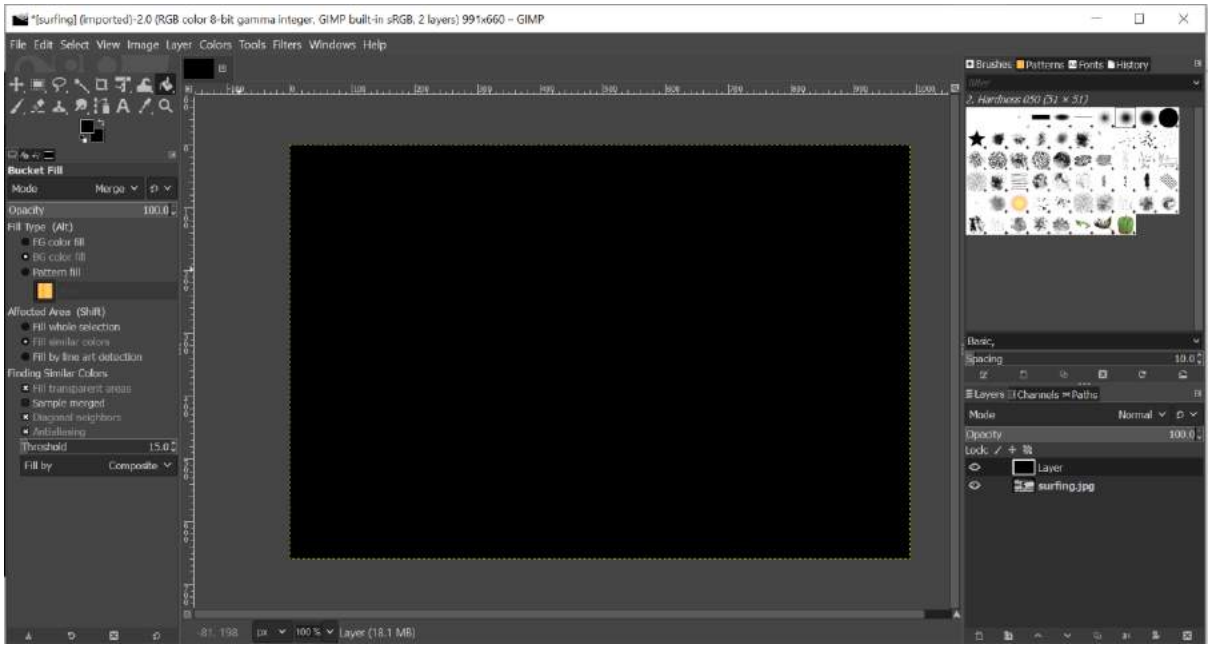

File Edit Select View Image Layer Colors Tools Filters Windows Help
Add Layer Mask
Add a Mask to the Layer
image.jpg-31 ([image] (imported))
Initialize Layer Mask to:
White (full opacity)
Black (full transparency)
Layer's alpha channel
Transfer layer's alpha channel
Selection
Grayscale copy of layer
Channel
Invert mask
Help
Add
Cancel
image.jpg
image.jpg (5.1 MB)

GIF ANIMATED IMAGE

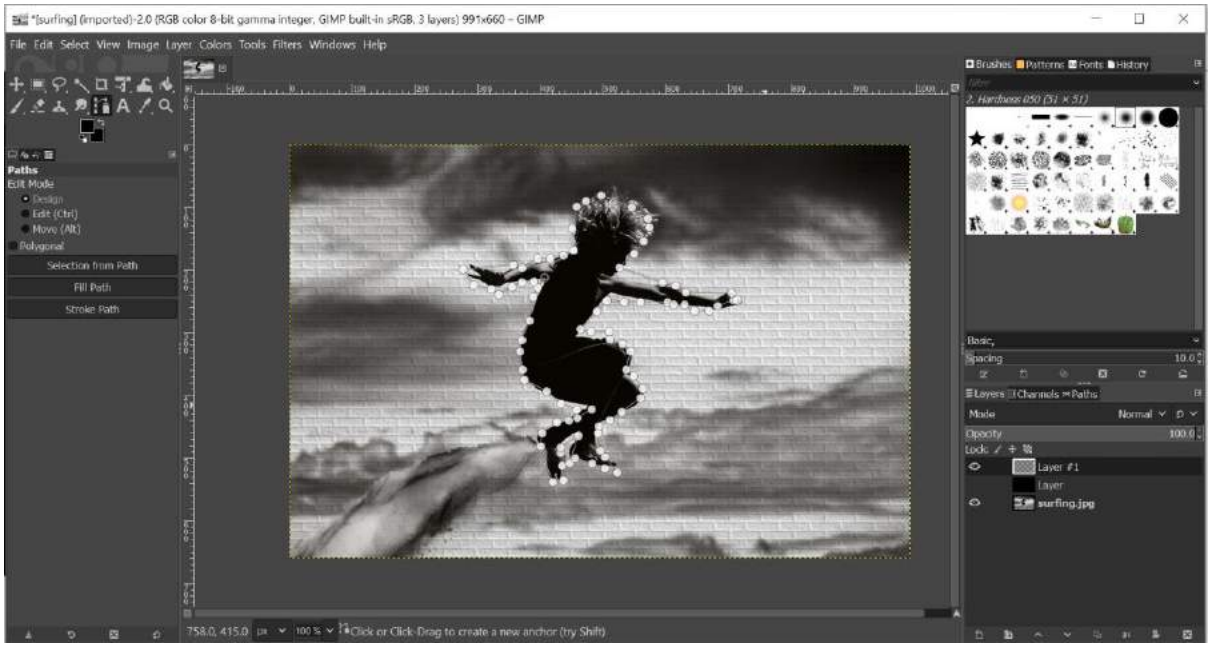


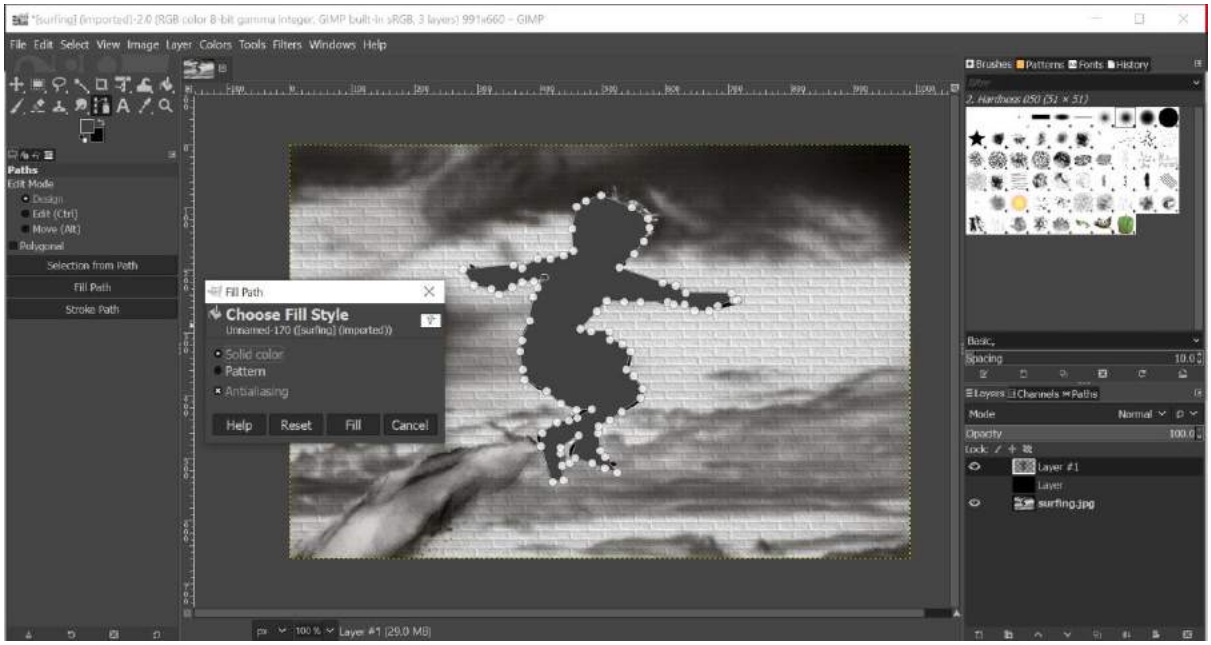


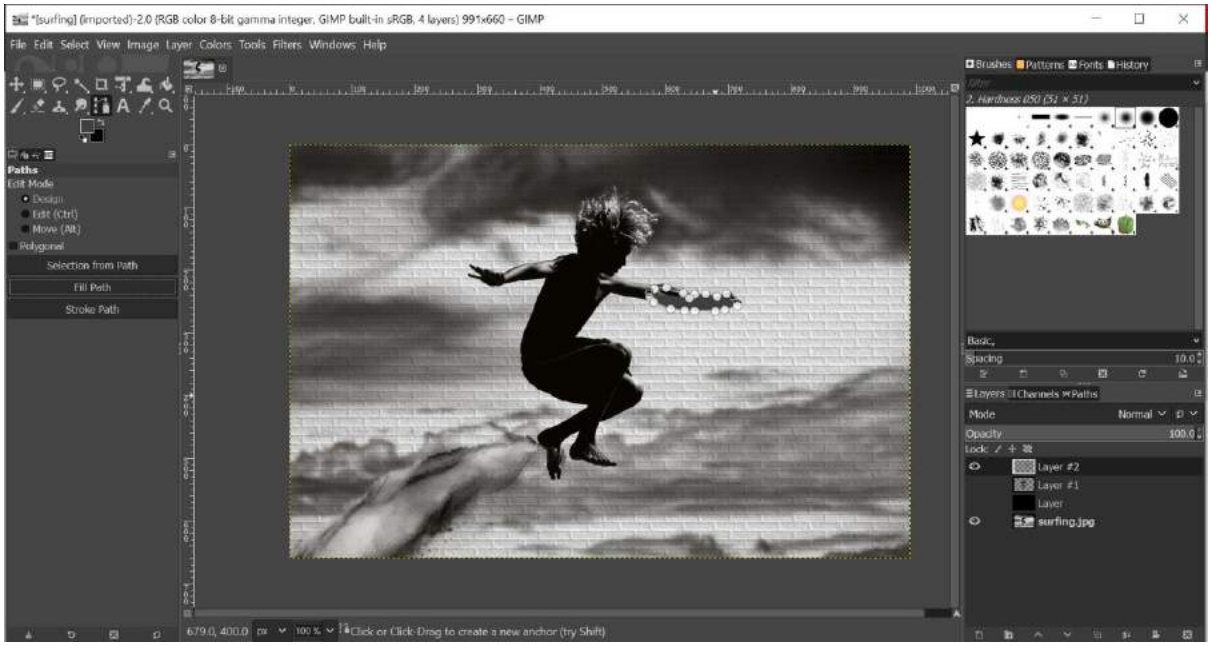
Bucket fill of bgcolor to generate Depth Map

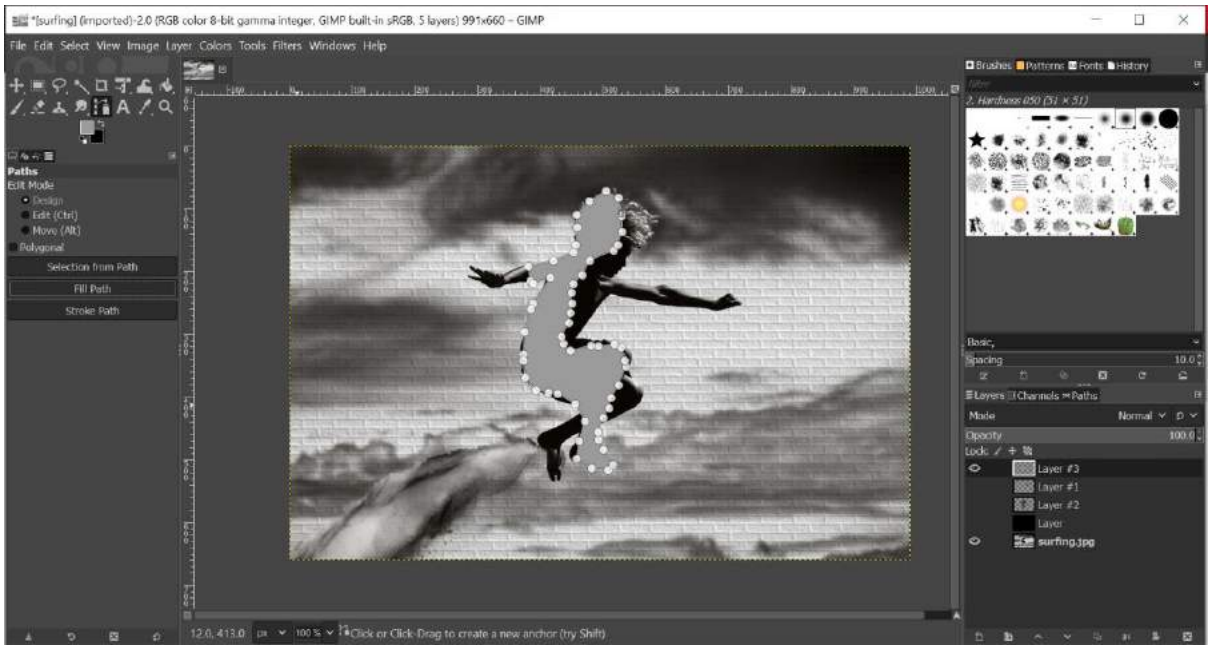


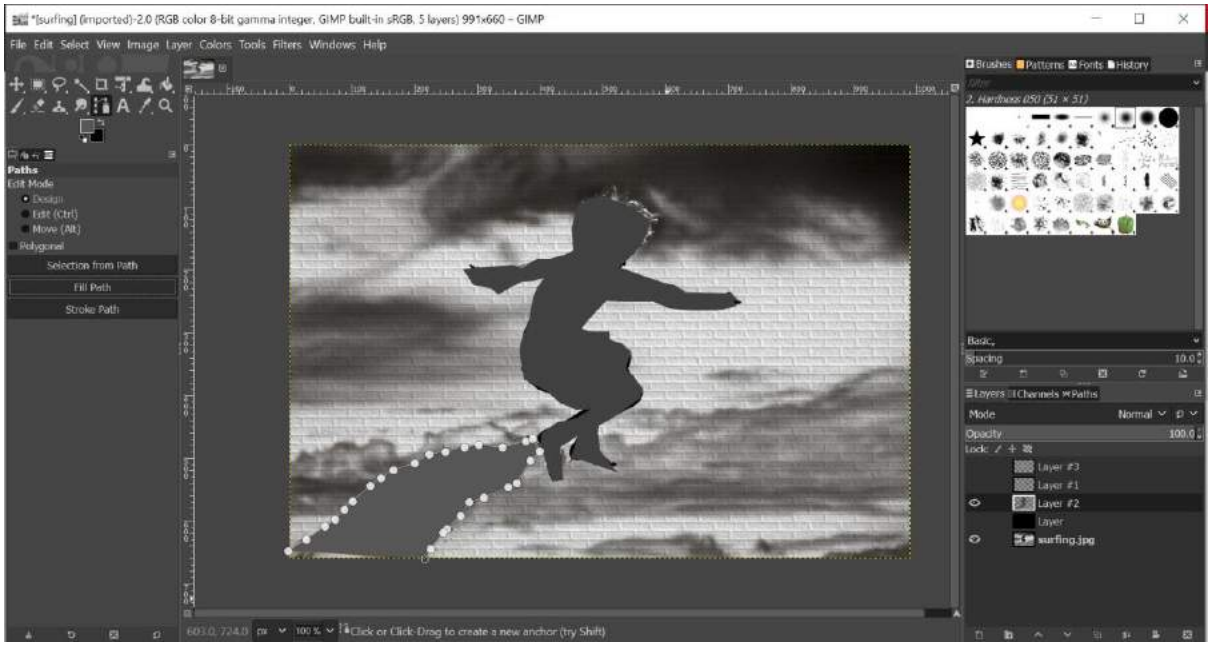
New layer: paths tool path around subject

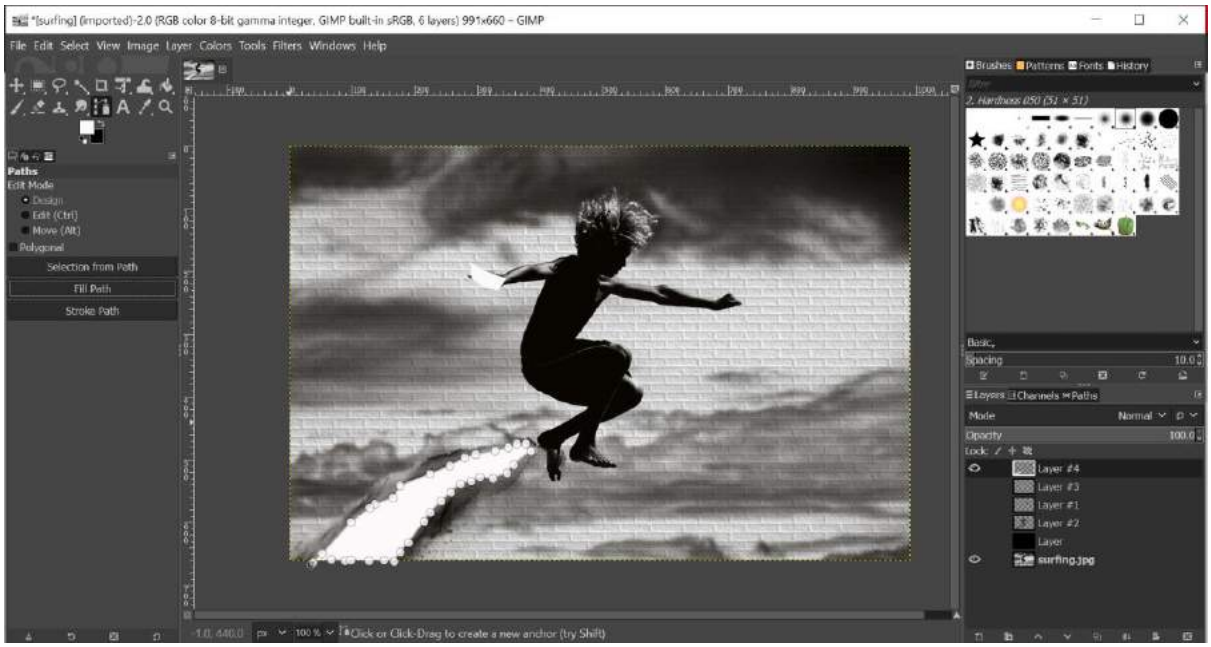

*[surfing] (imported)-2.0 (RGB color 8-bit gamma integer, GIMP built-in sRGB, 3 layers) 991x660 – GIMP
File Edit Select View Image Layer Colors Tools Filters Windows Help
Paths
Edit Mode
Design
Edit (Ctrl)
Move (Alt)
Polygonal
Selection from Path
Fill Path
Stroke Path
Brushes Patterns Fonts History
2. Hardness 050 (51 × 51)
Basic,
Spacing 10.0
Layers Channels Paths
Mode Normal
Opacity 100.0
Lock:
Layer #1
Layer
surfing.jpg
758.0, 415.0 px 100 % Click or Click-Drag to create a new anchor (try Shift)

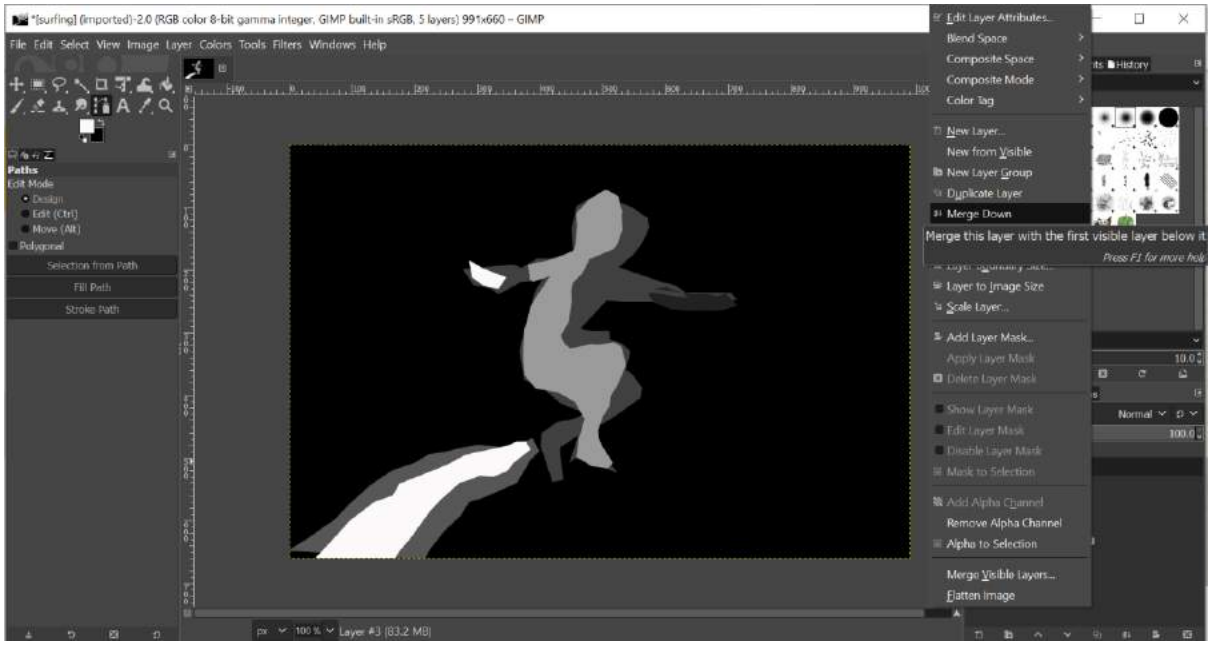

*[surfing] (imported)-2.0 (RGB color 8-bit gamma integer, GIMP built-in sRGB, 3 layers) 991x660 – GIMP
File Edit Select View Image Layer Colors Tools Filters Windows Help
Paths
Edit Mode
Design
Edit (Ctrl)
Move (Alt)
Polygonal
Selection from Path
Fill Path
Stroke Path
Fill Path
Choose Fill Style
Unnamed-170 ([surfing] (imported))
Solid color
Pattern
Antialiasing
Help Reset Fill Cancel
Brushes Patterns Fonts History
2. Hardness 050 (51 × 51)
Basic,
Spacing 10.0
Layers Channels Paths
Mode Normal
Opacity 100.0
Lock:
Layer #1
Layer
surfing.jpg
px 100 % Layer #1 (29.0 MB)

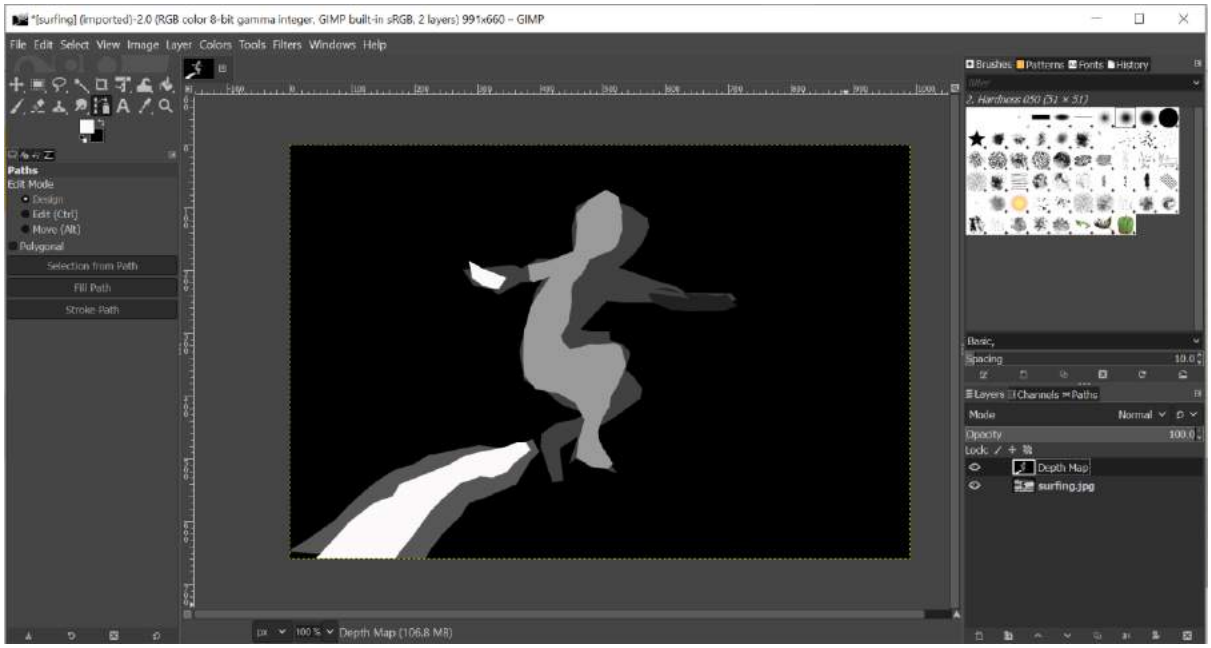

*[surfing] (imported)-2.0 (RGB color 8-bit gamma integer, GIMP built-in sRGB, 4 layers) 991×660 – GIMP
File Edit Select View Image Layer Colors Tools Filters Windows Help
Paths
Edit Mode
Design
Edit (Ctrl)
Move (Alt)
Polygonal
Selection from Path
Fill Path
Stroke Path
Brushes Patterns Fonts History
2. Hardness 050 (51 × 51)
Basic,
Spacing 10.0
Layers Channels Paths
Mode Normal
Opacity 100.0
Lock:
Layer #2
Layer #1
Layer
surfing.jpg
679.0, 400.0 px 100 % Click or Click-Drag to create a new anchor (try Shift)


*[surfing] (imported)-2.0 (RGB color 8-bit gamma integer, GIMP built-in sRGB, 5 layers) 991×660 – GIMP
File Edit Select View Image Layer Colors Tools Filters Windows Help
Paths
Edit Mode
Design
Edit (Ctrl)
Move (Alt)
Polygonal
Selection from Path
Fill Path
Stroke Path
Brushes Patterns Fonts History
2. Hardness 050 (51 × 51)
Basic,
Spacing 10.0
Layers Channels Paths
Mode Normal
Opacity 100.0
Lock:
Layer #3
Layer #1
Layer #2
Layer
surfing.jpg
12.0, 413.0 px 100 % Click or Click-Drag to create a new anchor (try Shift)

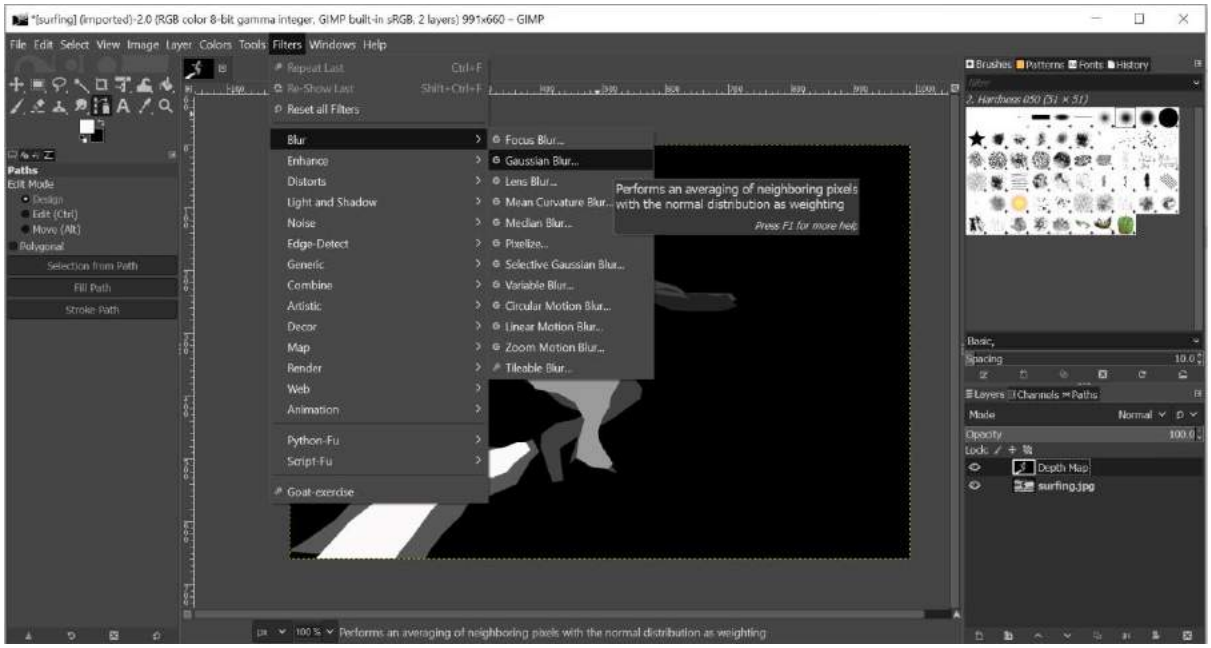

*[surfing] (imported)-2.0 (RGB color 8-bit gamma integer, GIMP built-in sRGB, 5 layers) 991×660 – GIMP
File Edit Select View Image Layer Colors Tools Filters Windows Help
Paths
Edit Mode
Design
Edit (Ctrl)
Move (Alt)
Polygonal
Selection from Path
Fill Path
Stroke Path
Brushes Patterns Fonts History
2. Hardness 050 (51 × 51)
Basic,
Spacing 10.0
Layers Channels Paths
Mode Normal
Opacity 100.0
Lock:
Layer #3
Layer #1
Layer #2
Layer
surfing.jpg
Click or Click-Drag to create a new anchor (try Shift)

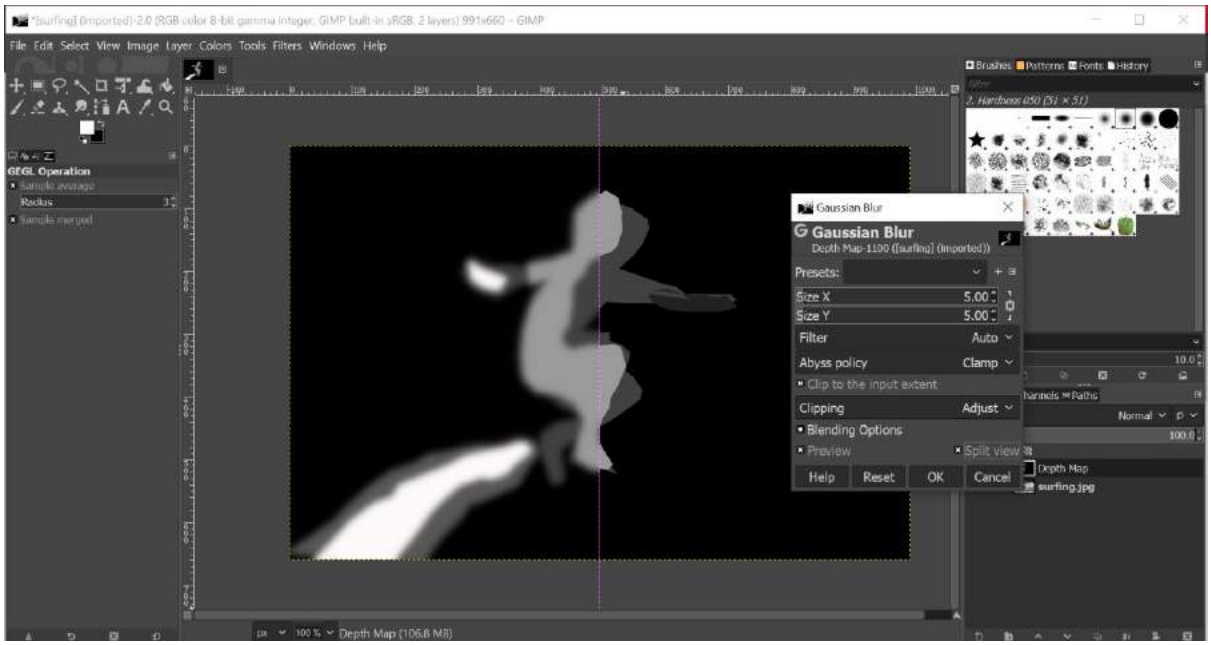

*[surfing] (imported)-2.0 (RGB color 8-bit gamma integer, GIMP built-in sRGB, 6 layers) 991×660 – GIMP
File Edit Select View Image Layer Colors Tools Filters Windows Help
Paths
Edit Mode
Design
Edit (Ctrl)
Move (Alt)
Polygonal
Selection from Path
Fill Path
Stroke Path
Brushes Patterns Fonts History
2. Hardness 050 (51 × 51)
Basic,
Spacing 10.0
Layers Channels Paths
Mode Normal
Opacity 100.0
Lock:
Layer #4
Layer #3
Layer #1
Layer #2
Layer
surfing.jpg
Click or Click-Drag to create a new anchor (try Shift)

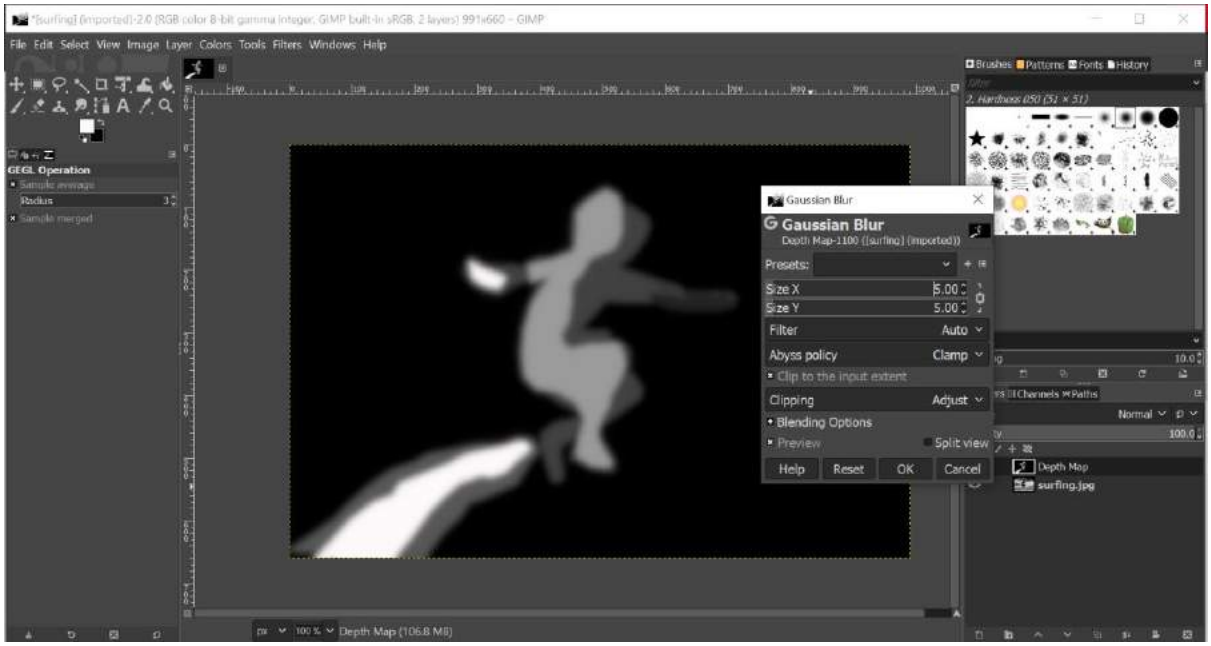

*[surfing] (imported)-2.0 (RGB color 8-bit gamma integer, GIMP built-in sRGB, 5 layers) 991×660 – GIMP
File Edit Select View Image Layer Colors Tools Filters Windows Help
Paths
Edit Mode
Design
Edit (Ctrl)
Move (Alt)
Polygonal
Selection from Path
Fill Path
Stroke Path
Edit Layer Attributes...
Blend Space
Composite Space
Composite Mode
Color Tag
New Layer...
New from Visible
New Layer Group
Duplicate Layer
Merge Down
Merge this layer with the first visible layer below it
Press F1 for more help
Layer to Image Size
Scale Layer...
Add Layer Mask...
Apply Layer Mask
Delete Layer Mask
Show Layer Mask
Edit Layer Mask
Disable Layer Mask
Mask to Selection
Add Alpha Channel
Remove Alpha Channel
Alpha to Selection
Merge Visible Layers...
Flatten Image
Layer #3 (83.2 MB)


*[surfing] (imported)-2.0 (RGB color 8-bit gamma integer, GIMP built-in sRGB, 2 layers) 991×660 – GIMP
File Edit Select View Image Layer Colors Tools Filters Windows Help
Paths
Edit Mode
Design
Edit (Ctrl)
Move (Alt)
Polygonal
Selection from Path
Fill Path
Stroke Path
Brushes
Patterns
Fonts
History
Basic,
Spacing
Layers
Channels
Paths
Mode
Normal
Opacity
Lock:
Depth Map
surfing.jpg
Depth Map (106.8 MB)

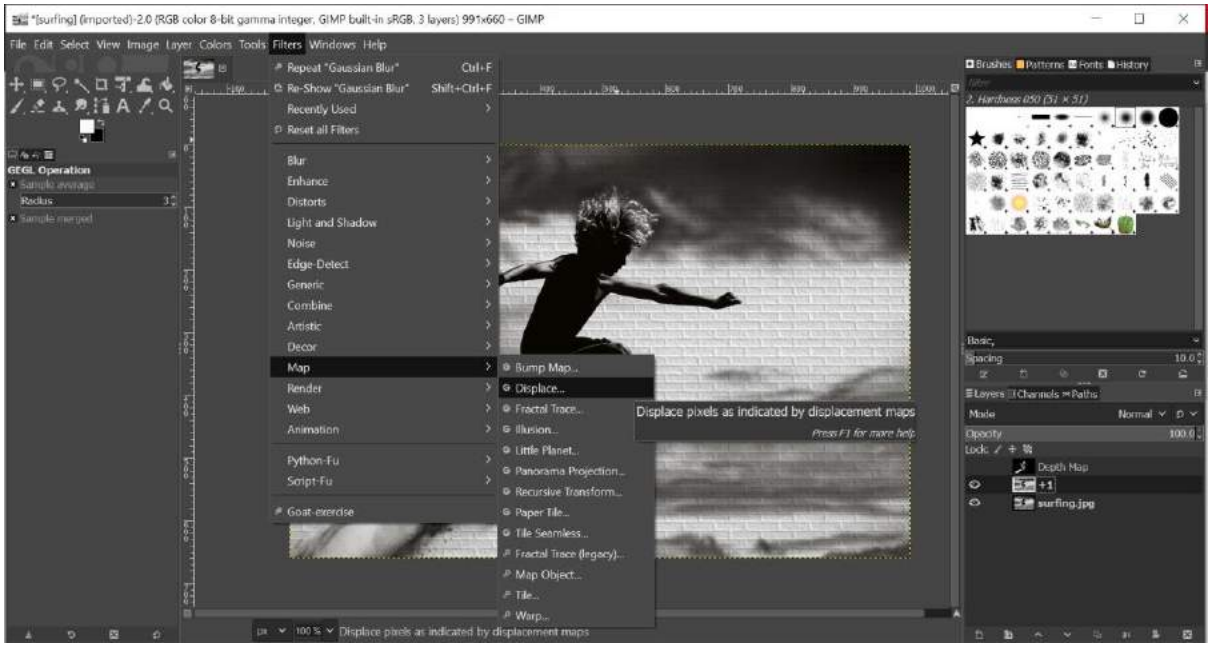
Blur

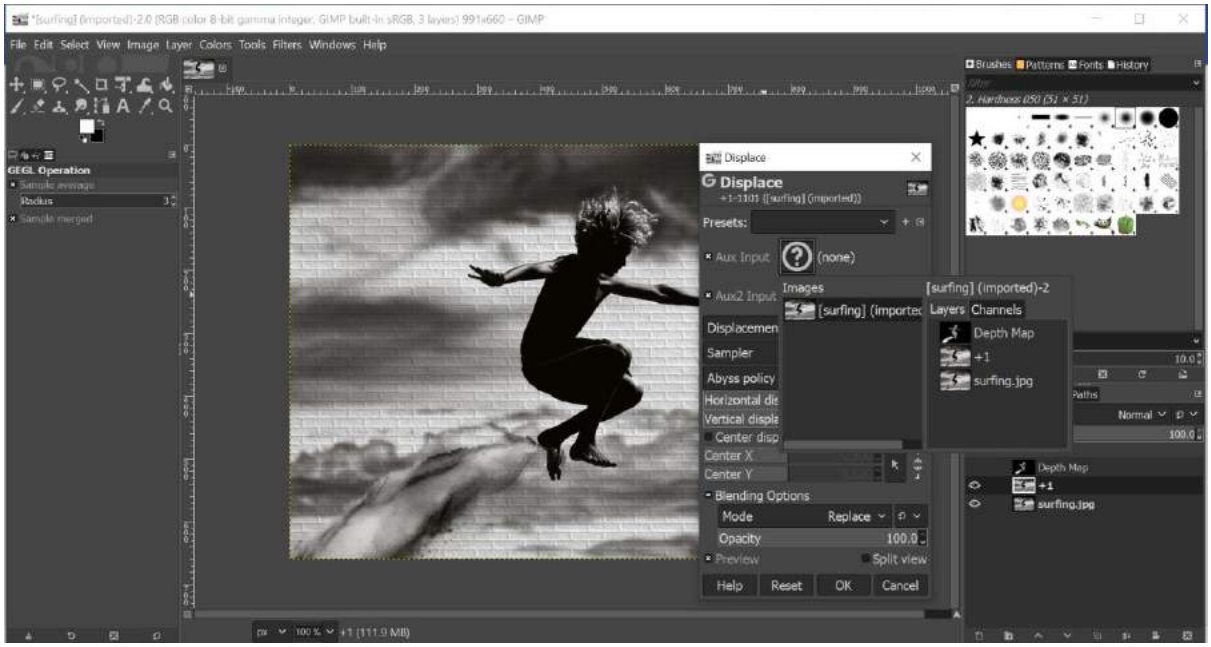

File Edit Select View Image Layer Colors Tools Filters Windows Help
Repeat Last Ctrl+F
Re-Show Last Shift+Ctrl+F
Reset all Filters
Blur
Enhance
Distorts
Light and Shadow
Noise
Edge-Detect
Generic
Combine
Artistic
Decor
Map
Render
Web
Animation
Python-Fu
Script-Fu
Goat-exercise
Focus Blur...
Gaussian Blur...
Lens Blur...
Mean Curvature Blur...
Median Blur...
Pixelize...
Selective Gaussian Blur...
Variable Blur...
Circular Motion Blur...
Linear Motion Blur...
Zoom Motion Blur...
Tileable Blur...
Performs an averaging of neighboring pixels with the normal distribution as weighting
Press F1 for more help
Paths
Edit Mode
Selection from Path
Fill Path
Stroke Path
Brushes Patterns Fonts History
Layers Channels Paths
Depth Map
surfing.jpg
Performs an averaging of neighboring pixels with the normal distribution as weighting


File Edit Select View Image Layer Colors Tools Filters Windows Help
GEGL Operation
Radius
Gaussian Blur
Gaussian Blur
Depth Map-1100 ([surfing] (imported))
Presets:
Size X 5.00
Size Y 5.00
Filter Auto
Abyss policy Clamp
Clip to the input extent
Clipping Adjust
Blending Options
Preview
Split view
Help Reset OK Cancel
Brushes Patterns Fonts History
Layers Channels Paths
Depth Map
surfing.jpg
Depth Map (106.8 MB)


File Edit Select View Image Layer Colors Tools Filters Windows Help
GEGL Operation
Radius
Gaussian Blur
Gaussian Blur
Depth Map-1100 ([surfing] (imported))
Presets:
Size X
Size Y
Filter
Auto
Abyss policy
Clamp
Clip to the input extent
Clipping
Adjust
Blending Options
Preview
Split view
Help
Reset
OK
Cancel
Depth Map
surfing.jpg
Depth Map (106.8 MB)


File Edit Select View Image Layer Colors Tools Filters Windows Help
GEGL Operation
Radius
Edit Layer Attributes...
Blend Space
Composite Space
Composite Mode
Color Tag
New Layer...
New from Visible
New Layer Group
Duplicate Layer
Create a duplicate of the layer and add it to the image
Layer Boundary Size...
Layer to Image Size
Scale Layer...
Add Layer Mask...
Apply Layer Mask
Delete Layer Mask
Show Layer Mask
Edit Layer Mask
Disable Layer Mask
Mask to Selection
Add Alpha Channel
Remove Alpha Channel
Alpha to Selection
Merge Visible Layers...
Flatten Image
surfing.jpg (109.9 MB)


*[surfing] (imported)-2.0 (RGB color 8-bit gamma integer, GIMP built-in sRGB, 3 layers) 991x660 – GIMP
File Edit Select View Image Layer Colors Tools Filters Windows Help
Repeat "Gaussian Blur" Ctrl+F
Re-Show "Gaussian Blur" Shift+Ctrl+F
Recently Used
Reset all Filters
Blur
Enhance
Distorts
Light and Shadow
Noise
Edge-Detect
Generic
Combine
Artistic
Decor
Map
Render
Web
Animation
Python-Fu
Script-Fu
Goat-exercise
Bump Map...
Displace...
Fractal Trace...
Illusion...
Little Planet...
Panorama Projection...
Recursive Transform...
Paper Tile...
Tile Seamless...
Fractal Trace (legacy)...
Map Object...
Tile...
Warp...
Displace pixels as indicated by displacement maps
Press F1 for more help
GEGL Operation
Radius
Brushes Patterns Fonts History
Layers Channels Paths
Mode Normal
Opacity 100.0
Depth Map
+1
surfing.jpg
Displace pixels as indicated by displacement maps

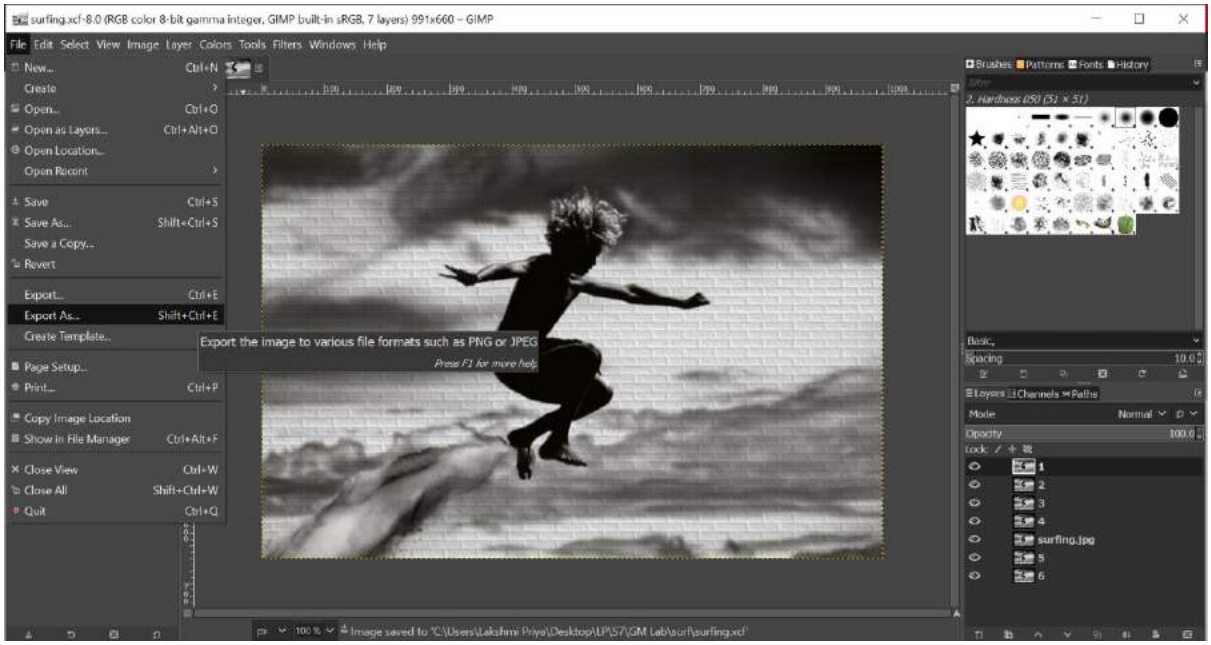

*[surfing] (imported)-2.0 (RGB color 8-bit gamma integer, GIMP built-in sRGB, 3 layers) 991x660 – GIMP
File Edit Select View Image Layer Colors Tools Filters Windows Help
GEGL Operation
Radius
Displace
Displace
Presets:
Aux Input
(none)
Aux2 Input
Images
[surfing] (imported)-2
Layers Channels
Depth Map
+1
surfing.jpg
Displacemen
Sampler
Abyss policy
Horizontal dis
Vertical displa
Center disp
Center X
Center Y
Blending Options
Mode Replace
Opacity 100.0
Preview
Split view
Help Reset OK Cancel
Brushes Patterns Fonts History
Paths
Normal
100.0
Depth Map
+1
surfing.jpg
+1 (111.9 MB)

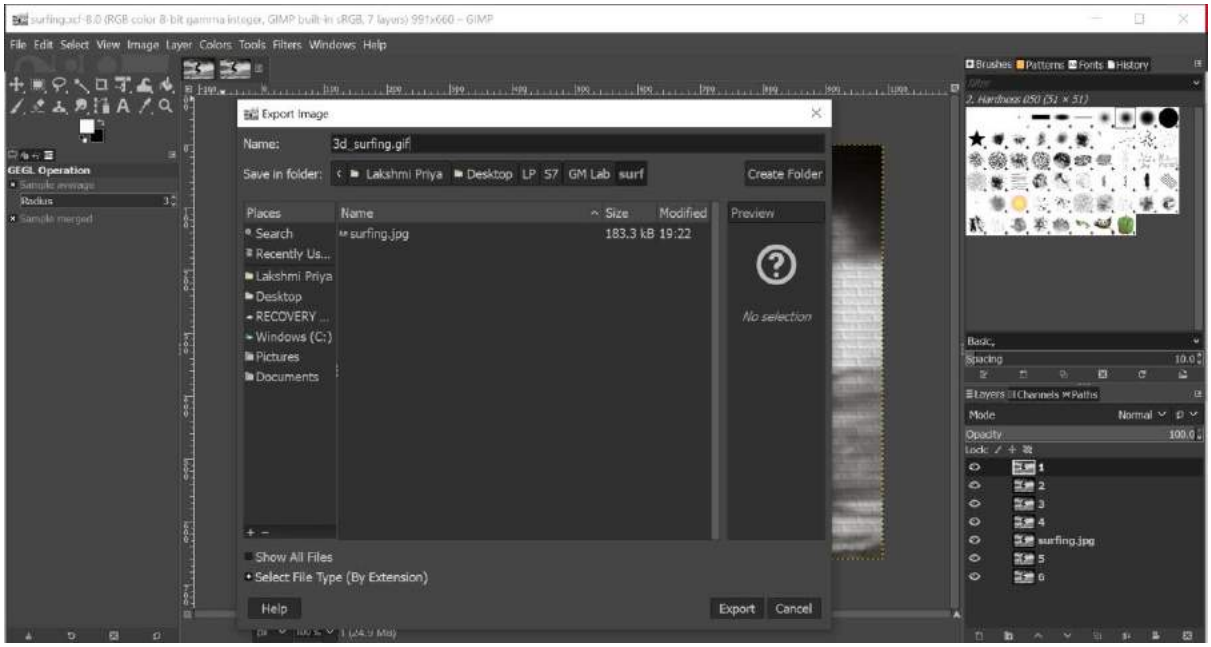

*[surfing] (imported)-2.0 (RGB color 8-bit gamma integer, GIMP built-in sRGB, 3 layers) 991×660 – GIMP
File Edit Select View Image Layer Colors Tools Filters Windows Help
GEGL Operation
Sample average
Radius
Sample merged
Displace
Displace
+1-1101 ([surfing] (imported))
Presets:
Aux Input
Depth Map
Aux2 Input
Depth Map
Displacement mode
Cartesian
Sampler
Cubic
Abyss policy
Clamp
Horizontal displacement
-10.00
Vertical displacement
-5.00
Center displacement
Center X
Center Y
Blending Options
Mode
Replace
Opacity
100.0
Preview
Split view
Help
Reset
OK
Cancel
Brushes
Patterns
Fonts
History
2. Hardboss 050 (51 × 51)
Layers
Channels
Paths
Normal
Depth Map
+1
surfing.jpg
+1 (111.9 MB)

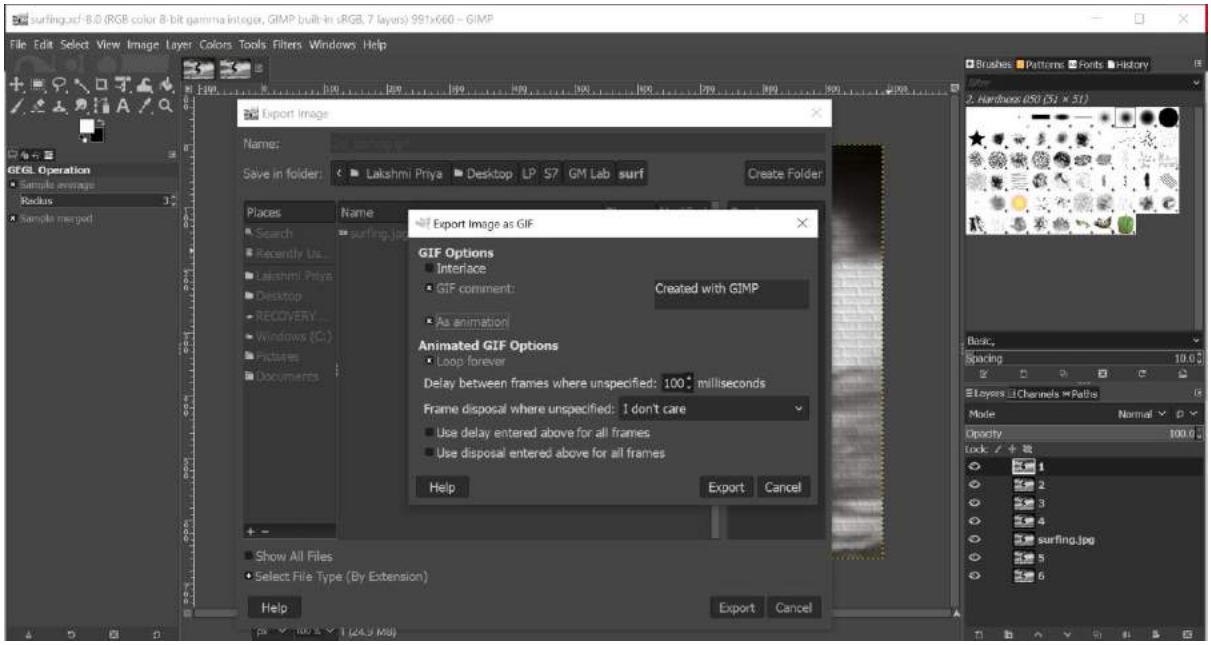

*[surfing] (imported)-2.0 (RGB color 8-bit gamma integer, GIMP built-in sRGB, 4 layers) 991×660 – GIMP
File Edit Select View Image Layer Colors Tools Filters Windows Help
GEGL Operation
Sample average
Radius
Sample merged
Displace
Displace
-1-1103 ([surfing] (imported))
Presets:
Aux Input
Depth Map
Aux2 Input
Depth Map
Displacement mode
Cartesian
Sampler
Cubic
Abyss policy
Clamp
Horizontal displacement
-6.00
Vertical displacement
-4.00
Center displacement
Center X
Center Y
Blending Options
Mode
Replace
Opacity
100.0
Preview
Split view
Help
Reset
OK
Cancel
Brushes
Patterns
Fonts
History
2. Hardboss 050 (51 × 51)
Basic,
Spacing
10.0
Layers
Channels
Paths
Mode
Normal
Opacity
100.0
Lock:
Depth Map
surfing.jpg
-3
-4
-3 (122.5 MB)


*[surfing] (imported)-2.0 (RGB color 8-bit gamma integer, GIMP built-in sRGB, 8 layers) 991×660 – GIMP
File Edit Select View Image Layer Colors Tools Filters Windows Help
Repeat "Displace" Ctrl+F
Re-Show "Displace" Shift+Ctrl+F
Recently Used
Reset all Filters
Blur
Enhance
Distorts
Light and Shadow
Noise
Edge-Detect
Generic
Combine
Artistic
Decor
Map
Render
Web
Animation
Python-Fu
Script-Fu
Goat-exercise
Blend...
Burn-in...
Rippling...
Spinning Globe...
Waves...
Optimize (Difference)
Optimize (for GIF)
Playback...
Unoptimize
GEGL Operation
Radius
Brushes Patterns Fonts History
Layers Channels Paths
Mode Normal
Opacity
Depth Map
surfing.jpg
Preview a GIMP layer-based animation
Press F1 for more help


Animation Playback: surfing.xcf
File Edit Select View Image Layer Colors Tools Filters Windows Help
Cumulative layers (combine)
1×
10 fps
100.0 %
Frame 1 of 7
GEGL Operation
Radius
Brushes Patterns Fonts History
Layers Channels Paths
Mode Normal
Opacity
surfing.jpg


surfing.xcf-8.0 (RGB color 8-bit gamma integer, GIMP built-in sRGB, 7 layers) 991x660 – GIMP
File Edit Select View Image Layer Colors Tools Filters Windows Help
New... Ctrl+N
Create
Open... Ctrl+O
Open as Layers... Ctrl+Alt+O
Open Location...
Open Recent
Save Ctrl+S
Save As... Shift+Ctrl+S
Save a Copy...
Revert
Export... Ctrl+E
Export As... Shift+Ctrl+E
Create Template...
Page Setup...
Print... Ctrl+P
Copy Image Location
Show in File Manager Ctrl+Alt+F
Close View Ctrl+W
Close All Shift+Ctrl+W
Quit Ctrl+Q
Export the image to various file formats such as PNG or JPEG
Press F1 for more help
Brushes Patterns Fonts History
Layers Channels Paths
surfing.jpg


surfing.xcf-8.0 (RGB color 8-bit gamma integer, GIMP built-in sRGB, 7 layers) 991x660 – GIMP
File Edit Select View Image Layer Colors Tools Filters Windows Help
Export Image
Name: 3d_surfing.gif
Save in folder: Lakshmi Priya Desktop LP S7 GM Lab surf
Create Folder
Places
Search
Recently Us...
Lakshmi Priya
Desktop
RECOVERY ...
Windows (C:)
Pictures
Documents
Name Size Modified
surfing.jpg 183.3 kB 19:22
Preview
No selection
Show All Files
Select File Type (By Extension)
Help
Export Cancel
GEGL Operation
Brushes Patterns Fonts History
Layers Channels Paths
surfing.jpg



Output Animation File

Attached as “surfing_animation.gif” as separate file

Result

Thus, various image editing and image manipulations have been done using GIMP.

SSN COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UCS1712 – GRAPHICS AND MULTIMEDIA LAB

EX NO: 11 – 2D Animation

Name : **Lakshmi Priya B**
Register Number : **185001083**
Date : **16-10-2021**

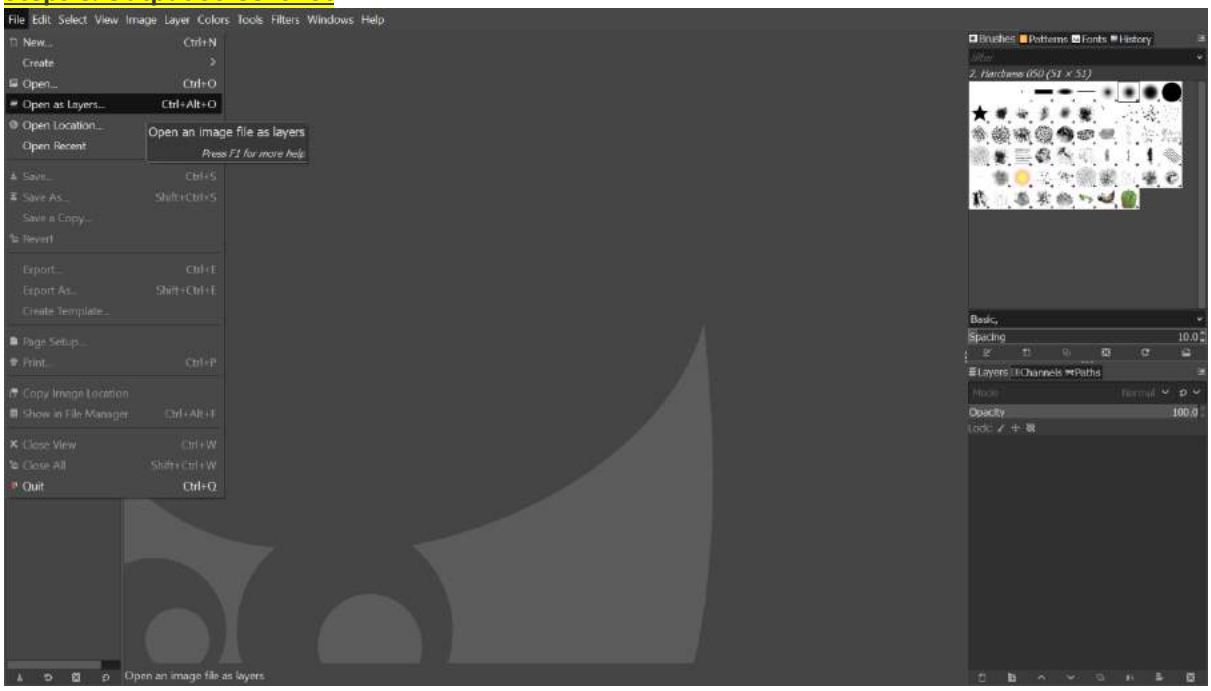
Aim

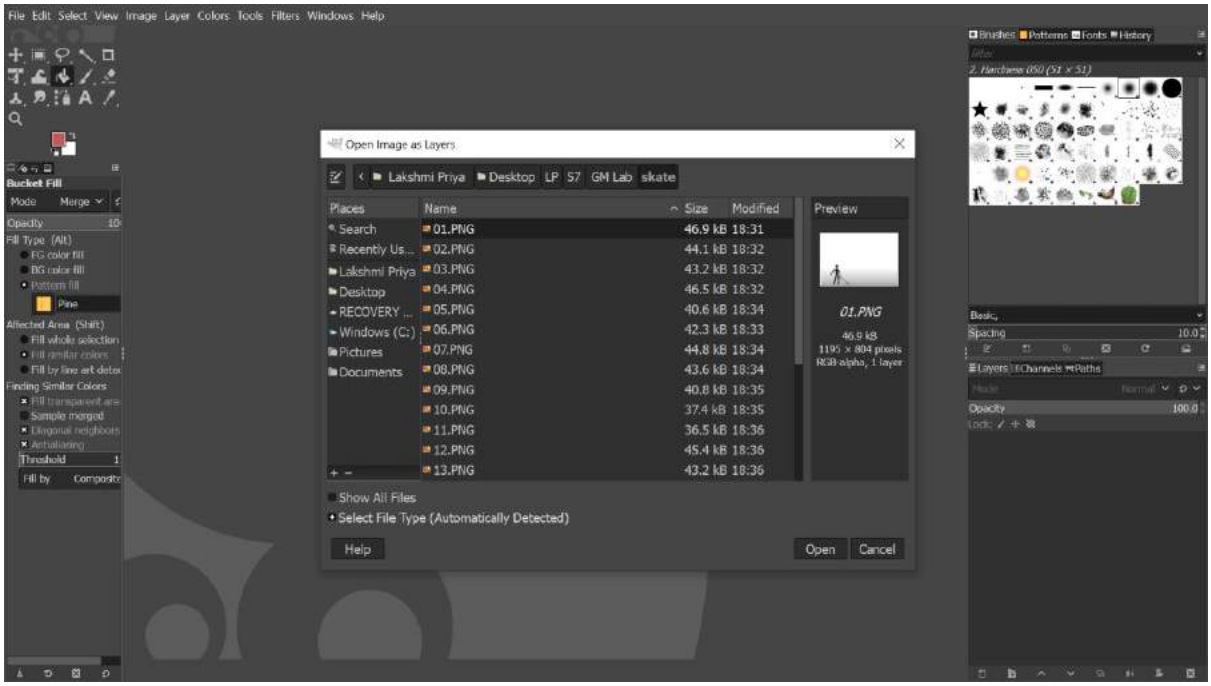
To create a simple 2D animation sequence video with minimum of two objects using GIMP Software.

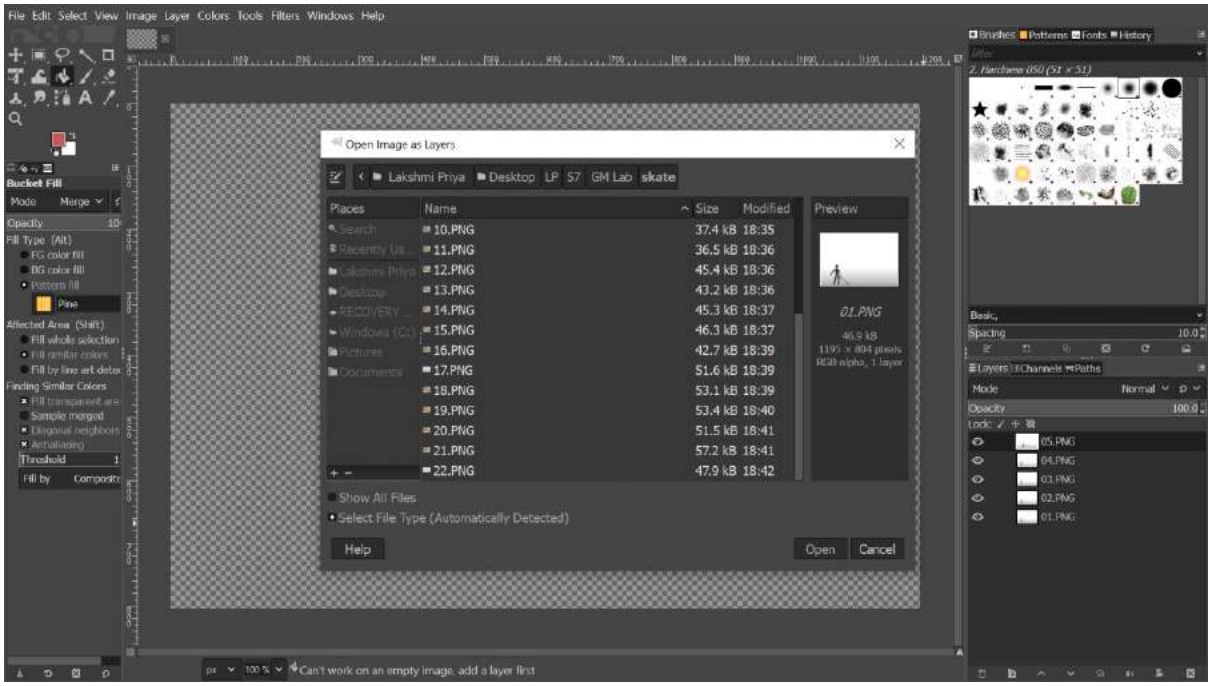
Algorithm

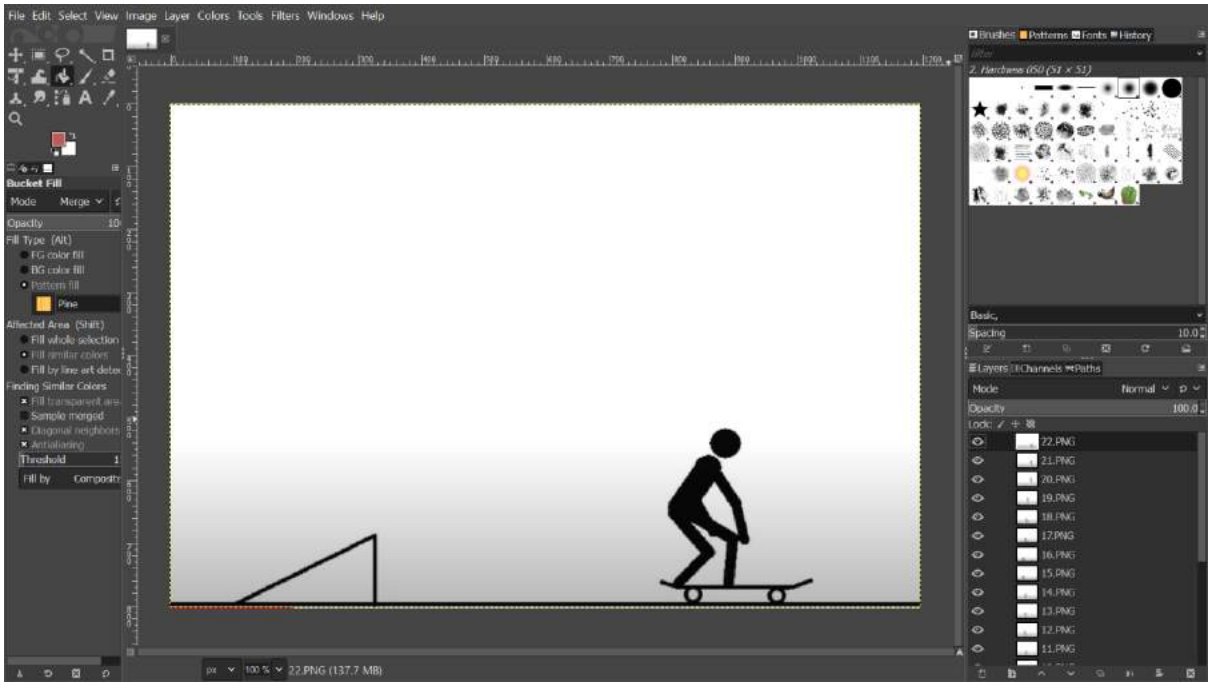
1. Collect a sequence of 2D animation frames with at least two objects
2. Go to File → Open as Layers to add all the images
3. View all the uploaded images in the Layers Dialog
4. Enable the loop forever features
5. Set appropriate delay between frames
6. To save the 2D animation, go to File → Export As and save as animation

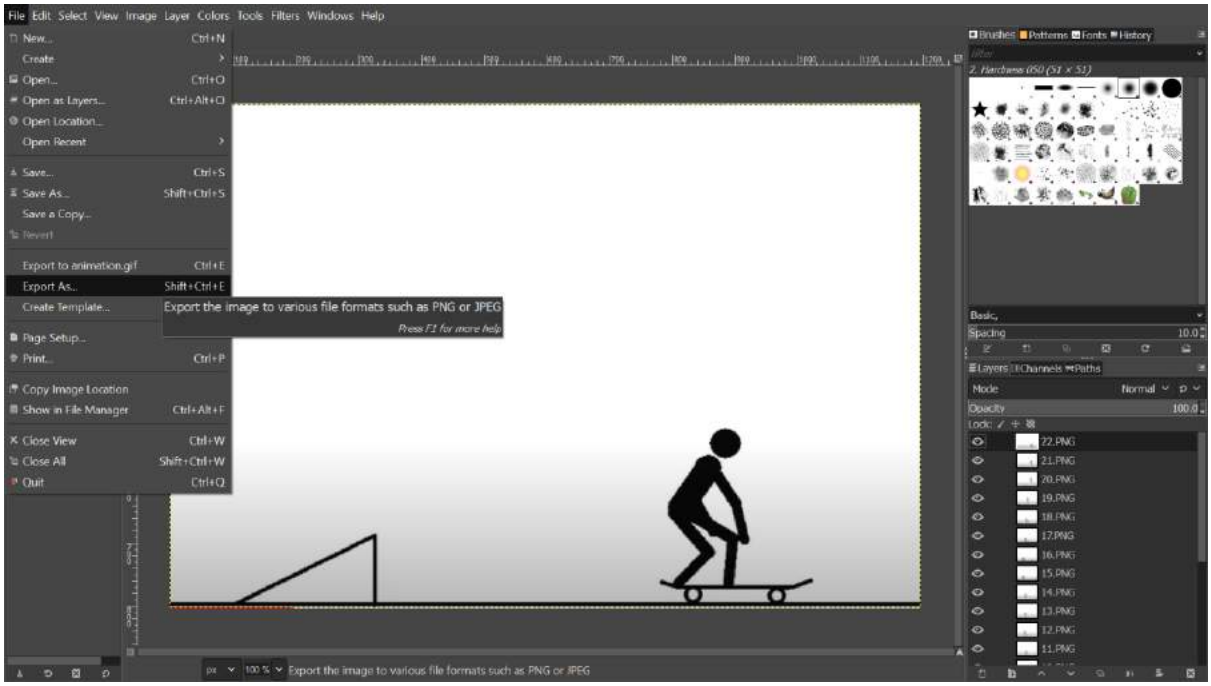
Steps & Output Screenshot

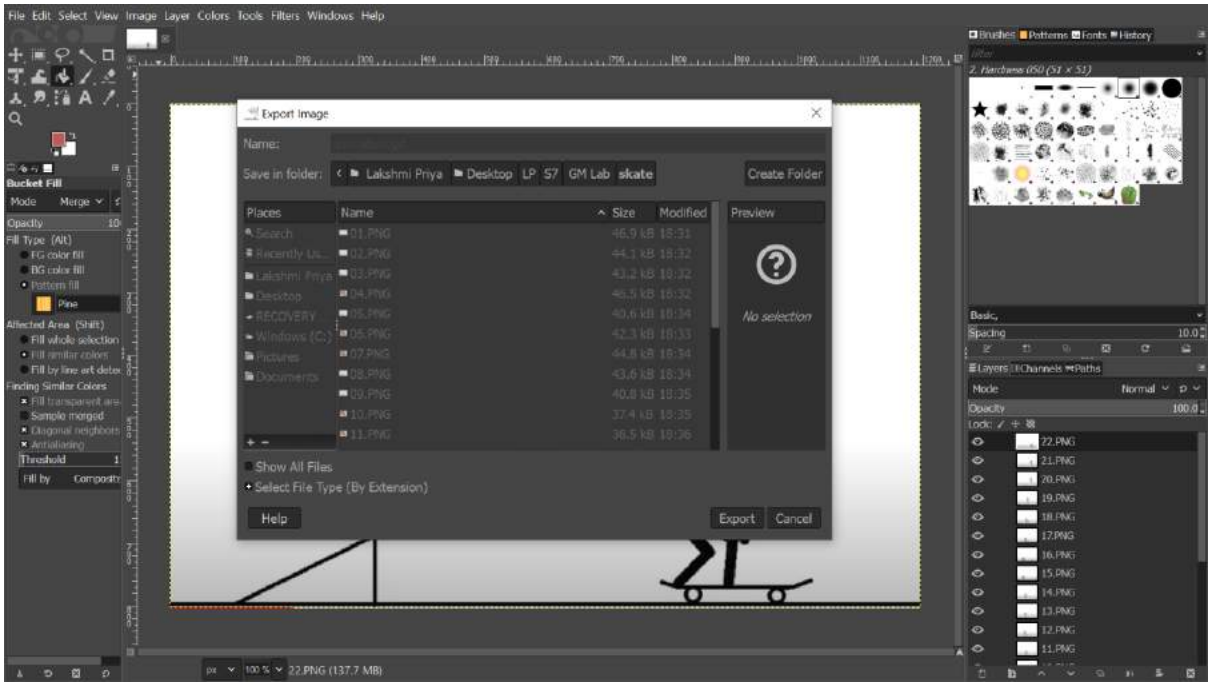


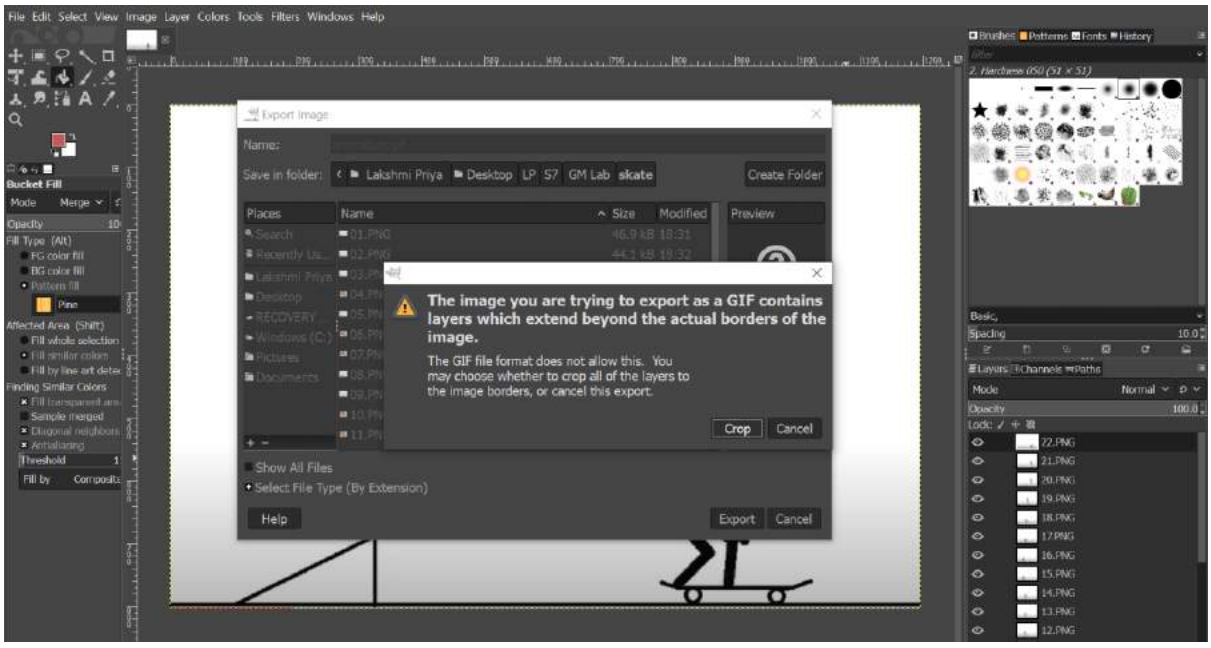

File Edit Select View Image Layer Colors Tools Filters Windows Help
Open Image as Layers
Lakshmi Priya Desktop LP S7 GM Lab skate
Places
Search
Recently Us...
Lakshmi Priya
Desktop
RECOVERY ...
Windows (C:)
Pictures
Documents
Name
Size
Modified
01.PNG 46.9 kB 18:31
02.PNG 44.1 kB 18:32
03.PNG 43.2 kB 18:32
04.PNG 46.5 kB 18:32
05.PNG 40.6 kB 18:34
06.PNG 42.3 kB 18:33
07.PNG 44.8 kB 18:34
08.PNG 43.6 kB 18:34
09.PNG 40.8 kB 18:35
10.PNG 37.4 kB 18:35
11.PNG 36.5 kB 18:36
12.PNG 45.4 kB 18:36
13.PNG 43.2 kB 18:36
Preview
01.PNG
46.9 kB
1195 × 804 pixels
RGB alpha, 1 layer
Show All Files
Select File Type (Automatically Detected)
Help
Open
Cancel
Bucket Fill
Brushes
Patterns
Fonts
History
Spacing
Layers
Channels
Paths
Opacity

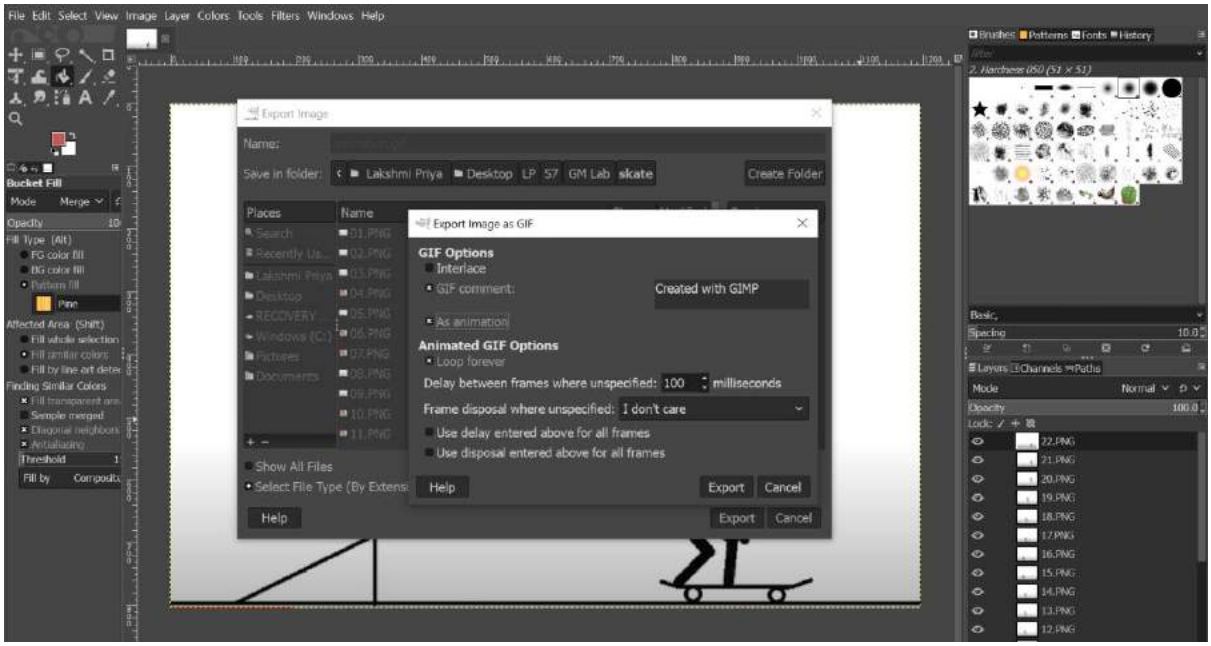

File Edit Select View Image Layer Colors Tools Filters Windows Help
Open Image as Layers
Lakshmi Priya Desktop LP S7 GM Lab skate
Places
Name
Size
Modified
10.PNG 37.4 kB 18:35
11.PNG 36.5 kB 18:36
12.PNG 45.4 kB 18:36
13.PNG 43.2 kB 18:36
14.PNG 45.3 kB 18:37
15.PNG 46.3 kB 18:37
16.PNG 42.7 kB 18:39
17.PNG 51.6 kB 18:39
18.PNG 53.1 kB 18:39
19.PNG 53.4 kB 18:40
20.PNG 51.5 kB 18:41
21.PNG 57.2 kB 18:41
22.PNG 47.9 kB 18:42
Preview
01.PNG
46.9 kB
1195 × 804 pixels
RGB alpha, 1 layer
Show All Files
Select File Type (Automatically Detected)
Help
Open
Cancel
Can't work on an empty image, add a layer first
Bucket Fill
Brushes
Patterns
Fonts
History
Spacing
Layers
Channels
Paths
Opacity
05.PNG
04.PNG
03.PNG
02.PNG
01.PNG


File Edit Select View Image Layer Colors Tools Filters Windows Help
Bucket Fill
Mode Merge
Opacity
Fill Type (Alt)
FG color fill
BG color fill
Pattern fill
Pine
Affected Area (Shift)
Fill whole selection
Fill similar colors
Fill by line art detec
Finding Similar Colors
Fill transparent are
Sample merged
Diagonal neighbors
Antialiasing
Threshold
Fill by Composite
Brushes Patterns Fonts History
2. Hardness 050 (51 × 51)
Basic,
Spacing 10.0
Layers Channels Paths
Mode Normal
Opacity 100.0
Lock:
22.PNG
21.PNG
20.PNG
19.PNG
18.PNG
17.PNG
16.PNG
15.PNG
14.PNG
13.PNG
12.PNG
11.PNG
px 100 % 22.PNG (137.7 MB)


File Edit Select View Image Layer Colors Tools Filters Windows Help
New... Ctrl+N
Create
Open... Ctrl+O
Open as Layers... Ctrl+Alt+O
Open Location...
Open Recent
Save... Ctrl+S
Save As... Shift+Ctrl+S
Save a Copy...
Revert
Export to animation.gif Ctrl+E
Export As... Shift+Ctrl+E
Create Template...
Export the image to various file formats such as PNG or JPEG
Press F1 for more help
Page Setup...
Print... Ctrl+P
Copy Image Location
Show in File Manager Ctrl+Alt+F
Close View Ctrl+W
Close All Shift+Ctrl+W
Quit Ctrl+Q
Brushes Patterns Fonts History
2. Hardness 050 (51 × 51)
Basic,
Spacing 10.0
Layers Channels Paths
Mode Normal
Opacity 100.0
Lock:
22.PNG
21.PNG
20.PNG
19.PNG
18.PNG
17.PNG
16.PNG
15.PNG
14.PNG
13.PNG
12.PNG
11.PNG
px 100 % Export the image to various file formats such as PNG or JPEG


File Edit Select View Image Layer Colors Tools Filters Windows Help
Export Image
Name:
Save in folder:
Lakshmi Priya
Desktop
LP
S7
GM Lab
skate
Create Folder
Places
Name
Size
Modified
Preview
No selection
Show All Files
Select File Type (By Extension)
Help
Export
Cancel
Brushes
Patterns
Fonts
History
Layers
Channels
Paths
Mode
Normal
Opacity
Lock:
22.PNG
21.PNG
20.PNG
19.PNG
18.PNG
17.PNG
16.PNG
15.PNG
14.PNG
13.PNG
12.PNG
11.PNG
22.PNG (137.7 MB)


File Edit Select View Image Layer Colors Tools Filters Windows Help
Export Image
Name:
Save in folder:
Lakshmi Priya
Desktop
LP
S7
GM Lab
skate
Create Folder
Places
Name
Size
Modified
Preview
The image you are trying to export as a GIF contains layers which extend beyond the actual borders of the image.
The GIF file format does not allow this. You may choose whether to crop all of the layers to the image borders, or cancel this export.
Crop
Cancel
Show All Files
Select File Type (By Extension)
Help
Export
Cancel
Brushes
Patterns
Fonts
History
Layers
Channels
Paths
Mode
Normal
Opacity
Lock:
22.PNG
21.PNG
20.PNG
19.PNG
18.PNG
17.PNG
16.PNG
15.PNG
14.PNG
13.PNG
12.PNG



Output Animation File

Attached as “skate.gif” as separate file

Result

Thus, a simple 2D animation sequence video with minimum of two objects has been created using GIMP Software.

SSN COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UCS1712 – GRAPHICS AND MULTIMEDIA LAB

EX NO: 12 – 3D Animation

Name : **Lakshmi Priya B**
Register Number : **185001083**

Date : **25-10-2021**

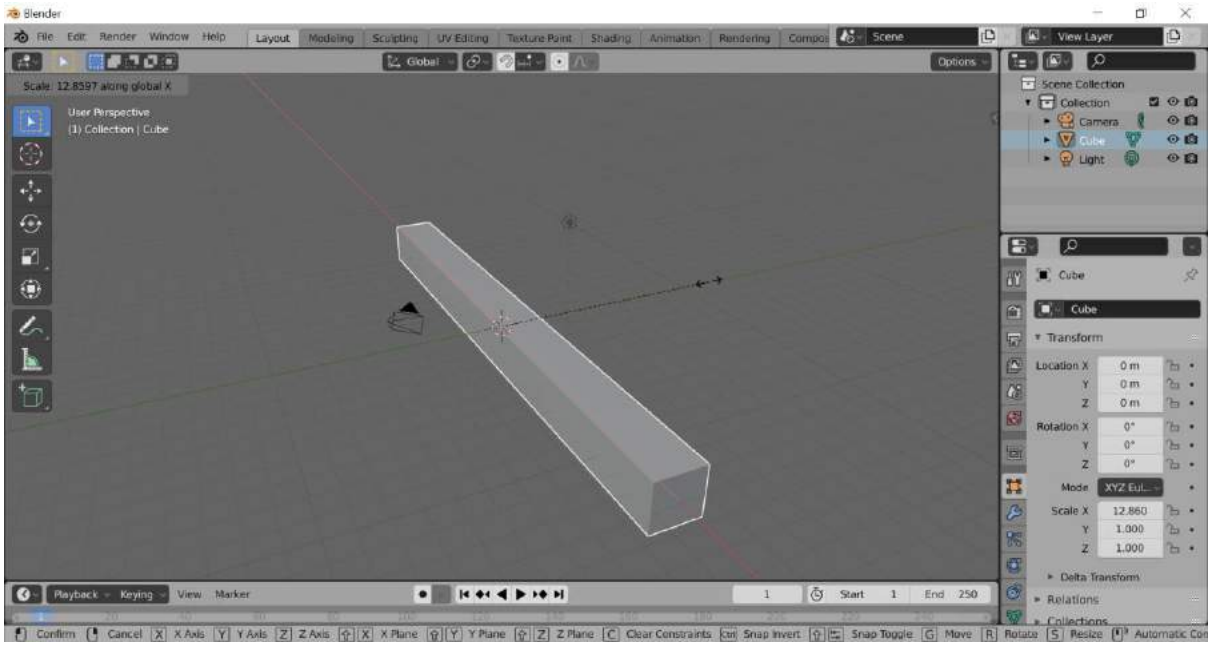
Aim

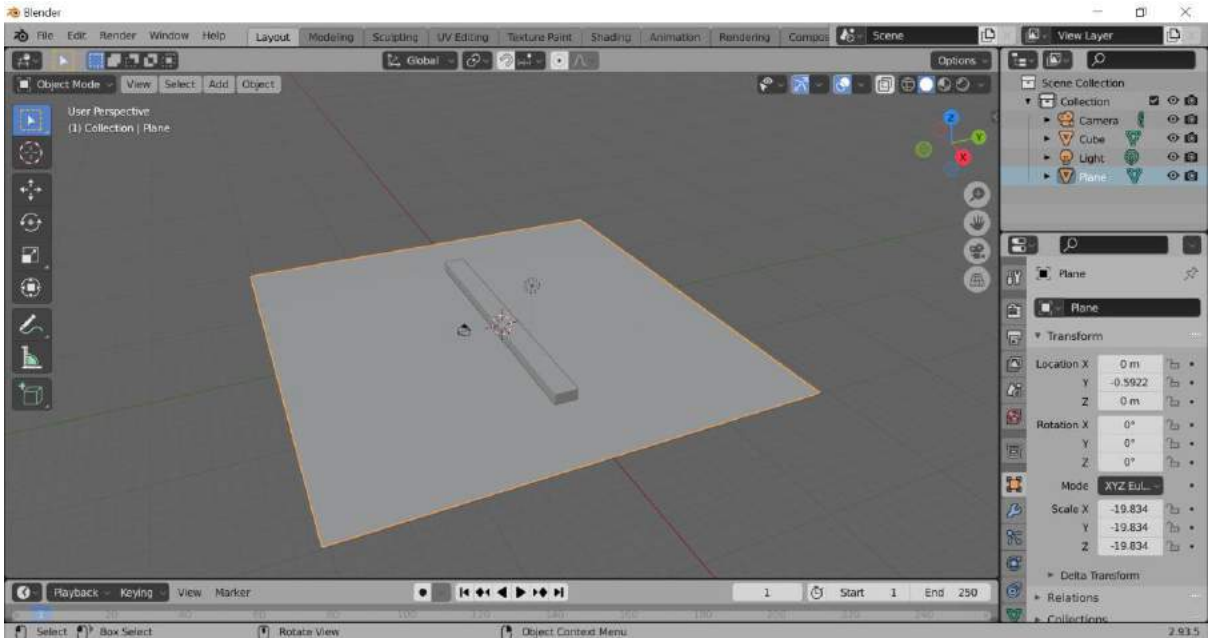
To create a simple 3D animation sequence with minimum of 2 objects using Blender Software.

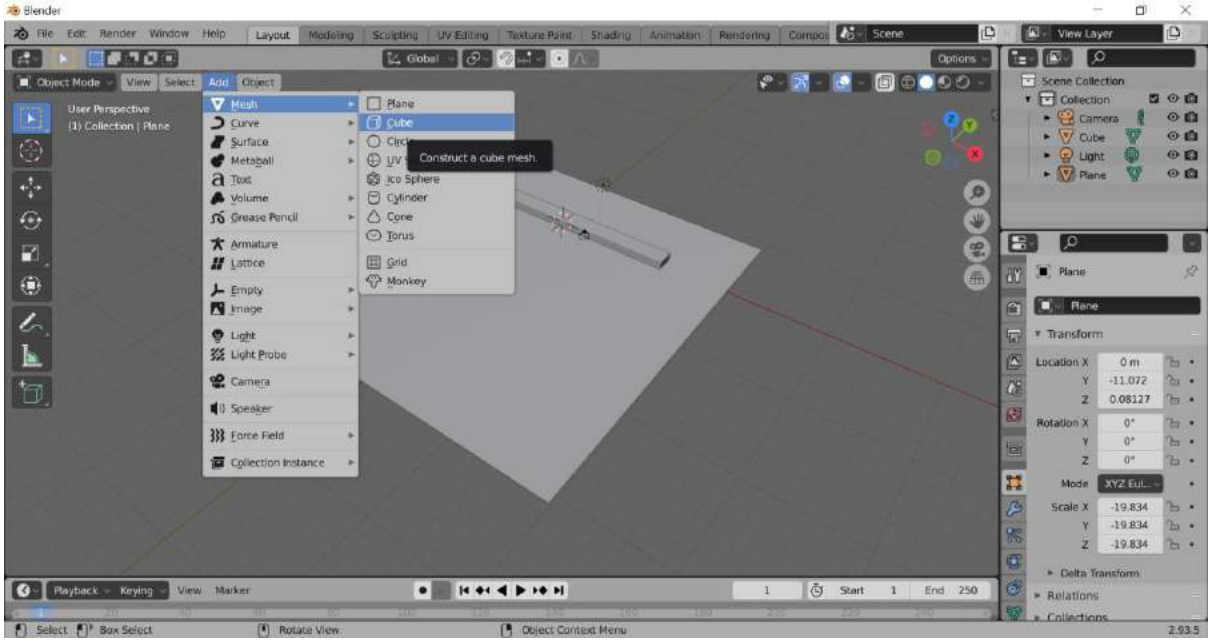
Algorithm

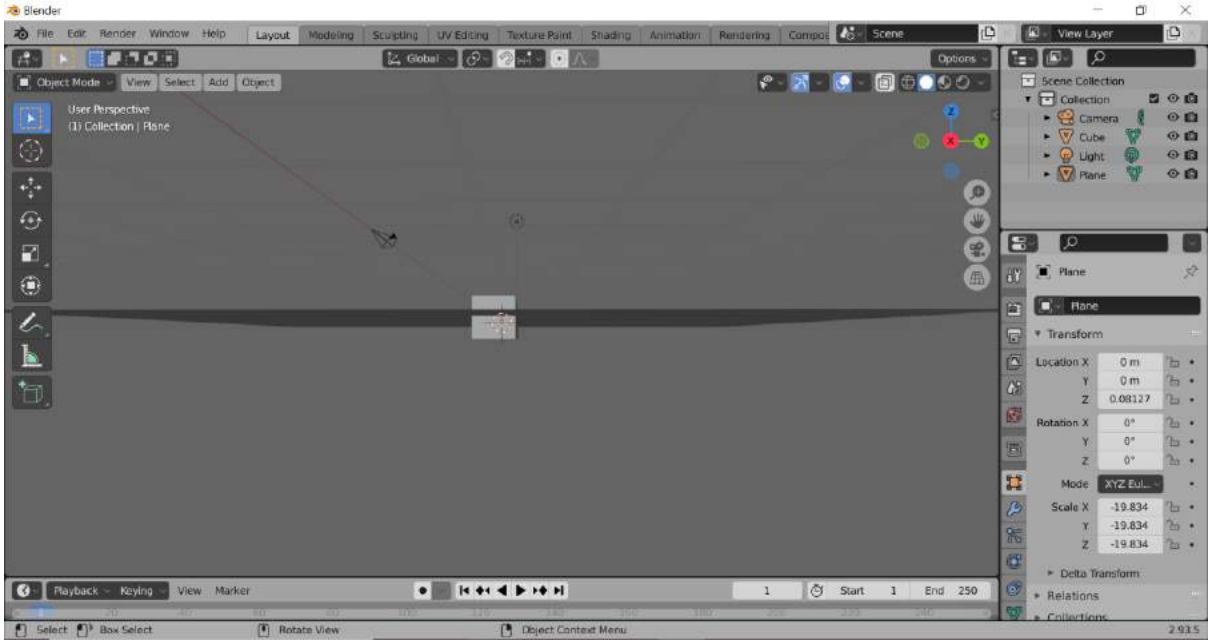
1. Open Blender Software
2. Scale the rendered cube on the x axis by clicking “s” and then “x” and dragging cursor to scale
3. Left click and confirm the scale
4. Goto Add -> Mesh -> Plane and scale plane
5. Click “g” and then “y” to grab plane on y axis and move
6. Click “g” and then “z” to grab plane on z axis and move
7. Goto Add -> Mesh -> Cube to add cube
8. Change rigid body settings of the cube
9. Set planes’s rigid body settings to passive
10. Select animate option for cuboid
11. To add animation of dropping cube, click “n” to see the location and “i” to keyframe at time 1
12. Translate cuboid and keyframe at time 150
13. Select cube and change its material surface to diffuse
14. To add more cubes, duplicate the current cube by using Shift+D
15. Select color of cubes at different times and click “i” by keeping cursor on color in material surface settings to keyframe the colours and animate change in color
16. Switch to rendered view by clicking on the top right round icon
17. Bake the animation to pre compute the animations
18. Set lighting conditions
19. Set the background colour
20. Set the color of the objects
21. Set Edit -> Preferences -> Input -> Emulate Numpad and click 0 to start camera
22. Lock camera to view and fix the camera position
23. Choose the output options and render animation by clicking “Render Animation” in Render option in top bar
24. Save the animation as AVI JPEG format
25. Play the animation saved in chosen location and verify the output

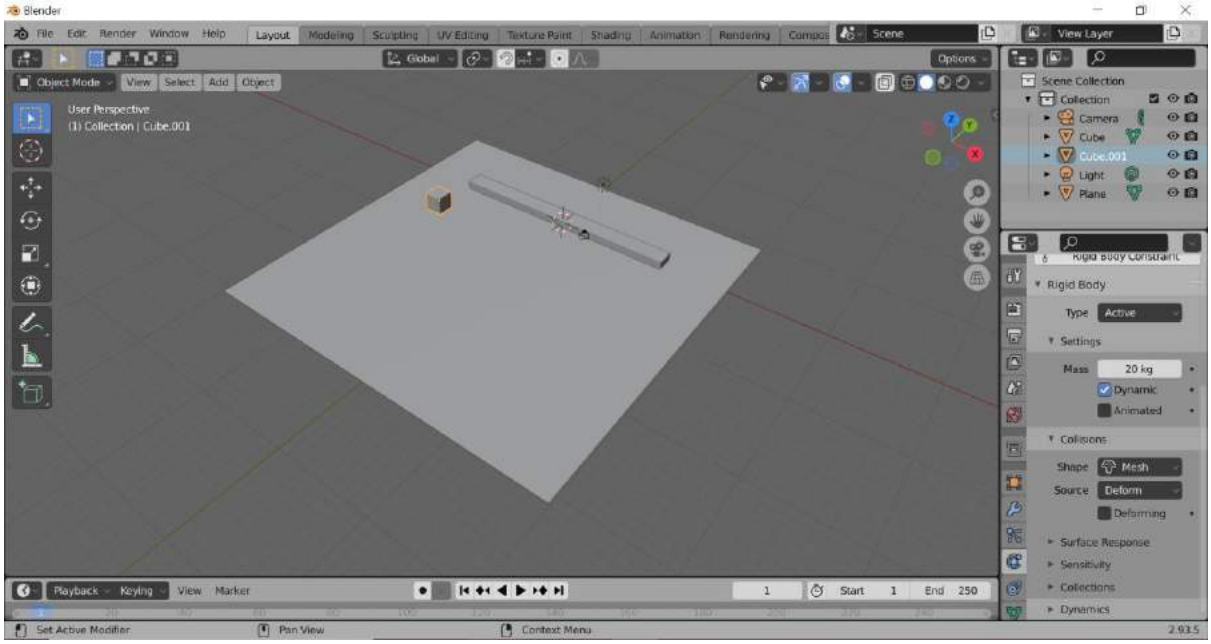
Steps & Output Screenshots

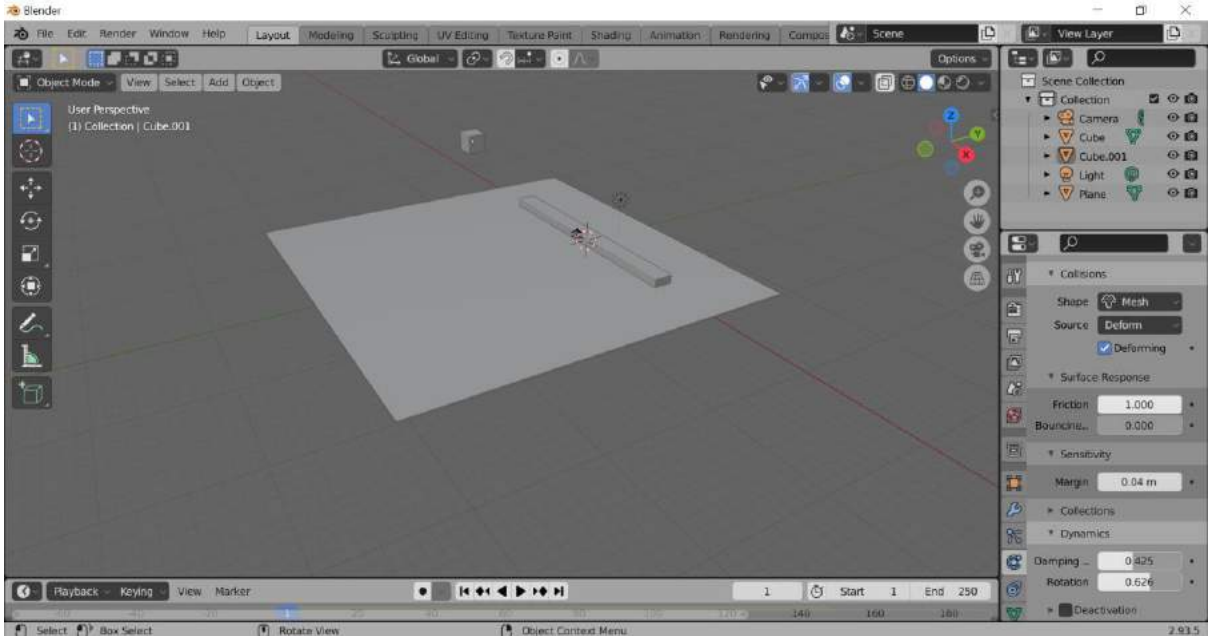


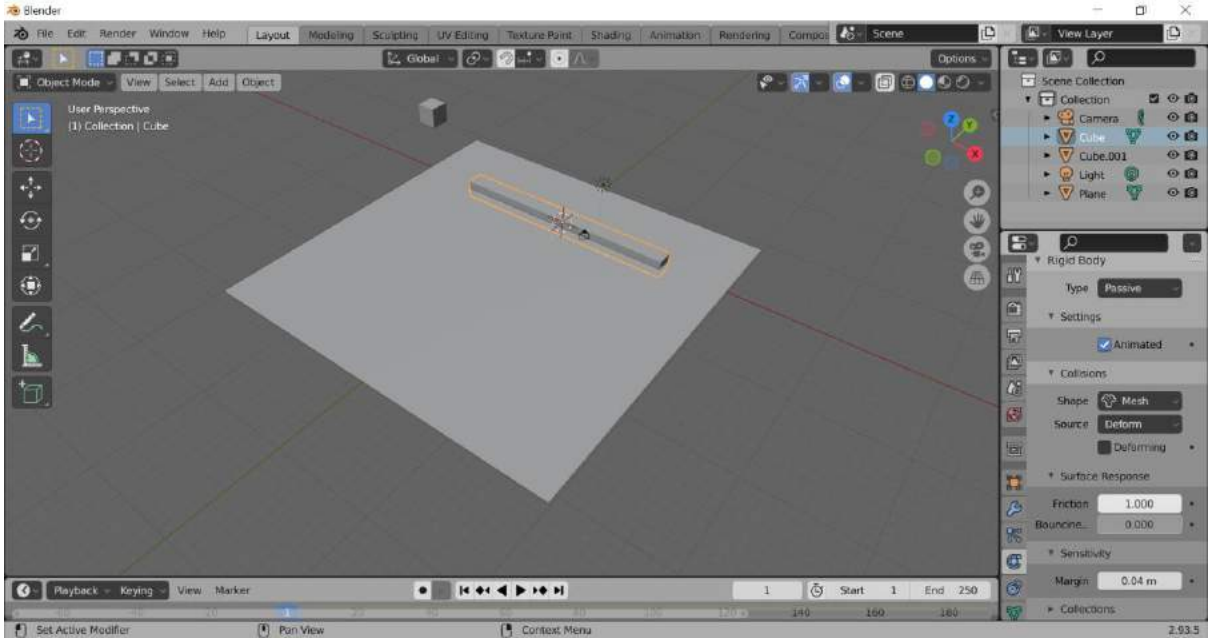


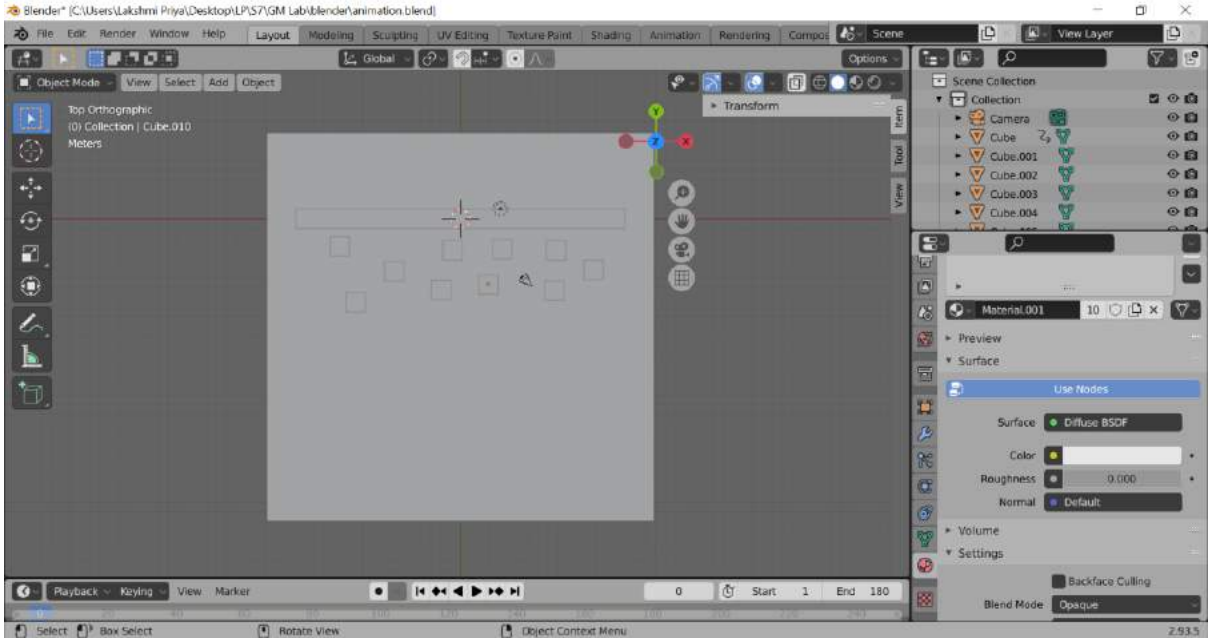

Blender
File Edit Render Window Help
Layout Modeling Sculpting UV Editing Texture Paint Shading Animation Rendering Compos
Scene
View Layer
Object Mode View Select Add Object
Global
Options
User Perspective
(1) Collection | Plane
Mesh
Curve
Surface
Metaball
Text
Volume
Grease Pencil
Armature
Lattice
Empty
Image
Light
Light Probe
Camera
Speaker
Force Field
Collection Instance
Plane
Cube
Circle
UV Sphere
Ico Sphere
Cylinder
Cone
Torus
Grid
Monkey
Construct a cube mesh.
Scene Collection
Collection
Camera
Cube
Light
Plane
Transform
Location X 0 m
Y -11.072
Z 0.08127
Rotation X 0°
Y 0°
Z 0°
Mode XYZ Eul...
Scale X -19.834
Y -19.834
Z -19.834
Delta Transform
Relations
Collections
Playback Keying View Marker
1
Start 1
End 250
Select
Box Select
Rotate View
Object Context Menu
2.93.5

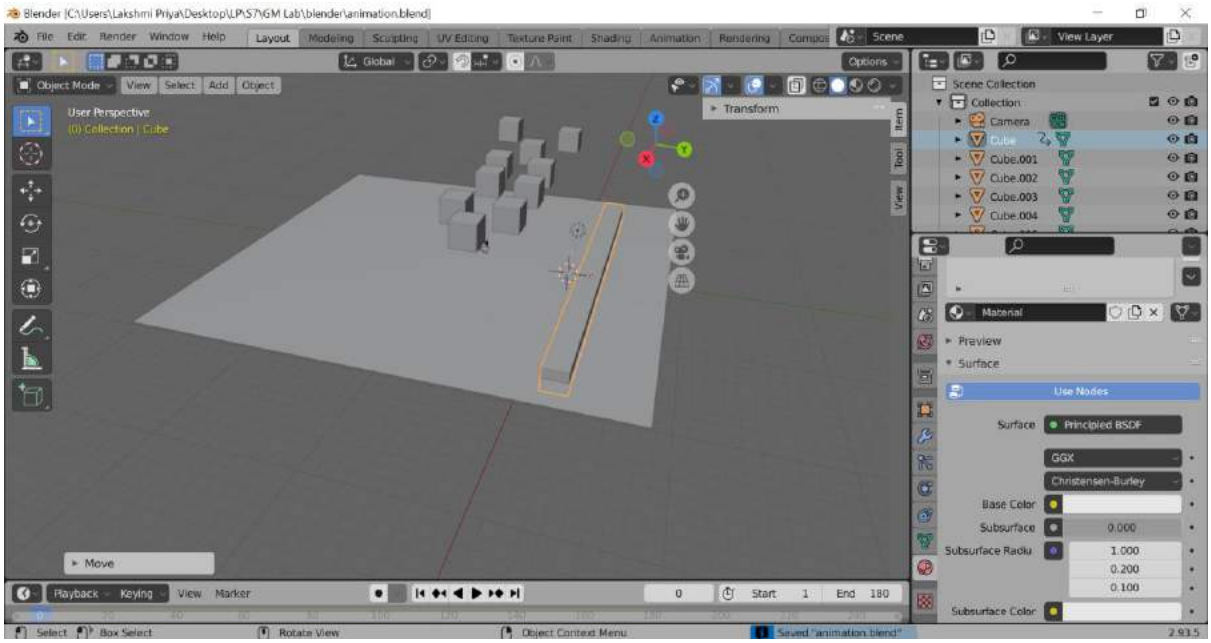

Blender
File Edit Render Window Help
Layout Modeling Sculpting UV Editing Texture Paint Shading Animation Rendering Compos
Scene
View Layer
Object Mode View Select Add Object
Global
Options
User Perspective
(1) Collection | Plane
Scene Collection
Collection
Camera
Cube
Light
Plane
Transform
Location X 0 m
Y 0 m
Z 0.08127
Rotation X 0°
Y 0°
Z 0°
Mode XYZ Eul...
Scale X -19.834
Y -19.834
Z -19.834
Delta Transform
Relations
Collections
Playback Keying View Marker
1
Start 1
End 250
Select
Box Select
Rotate View
Object Context Menu
2.93.5

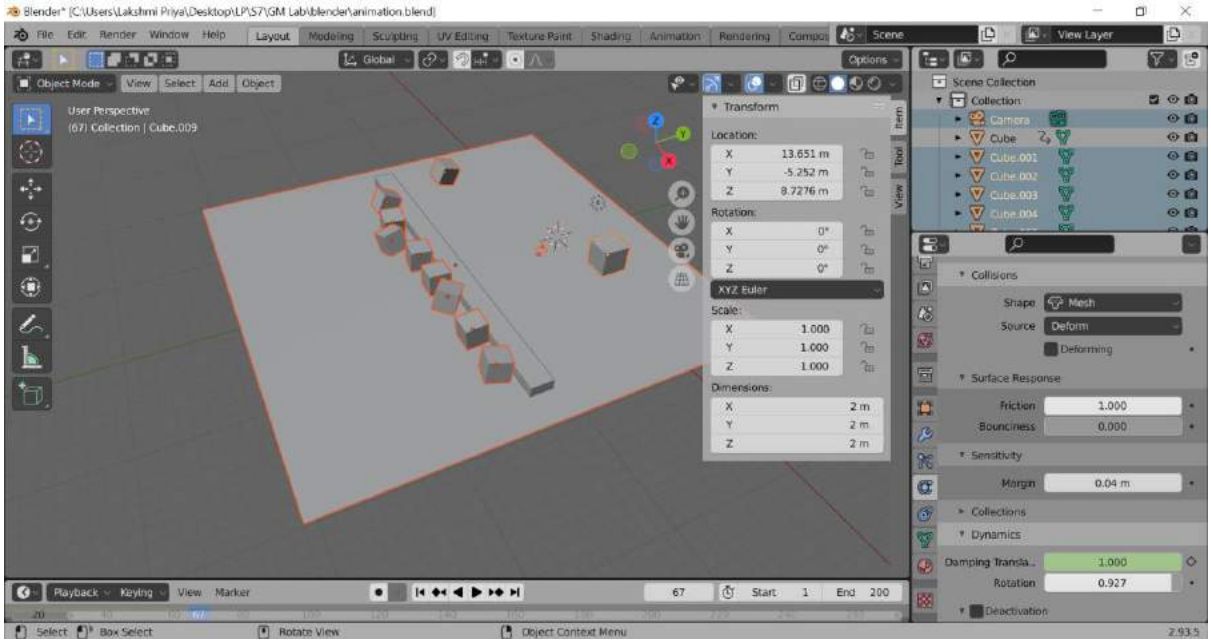

Blender
File Edit Render Window Help
Layout Modeling Sculpting UV Editing Texture Paint Shading Animation Rendering
Scene
View Layer
Object Mode View Select Add Object
Global
Options
User Perspective
(1) Collection | Cube.001
Scene Collection
Collection
Camera
Cube
Cube.001
Light
Plane
Rigid Body Constraint
Rigid Body
Type Active
Settings
Mass 20 kg
Dynamic
Animated
Collisions
Shape Mesh
Source Deform
Deforming
Surface Response
Sensitivity
Collections
Dynamics
Playback Keying View Marker
1
Start 1
End 250
Set Active Modifier
Pan View
Context Menu
2.93.5

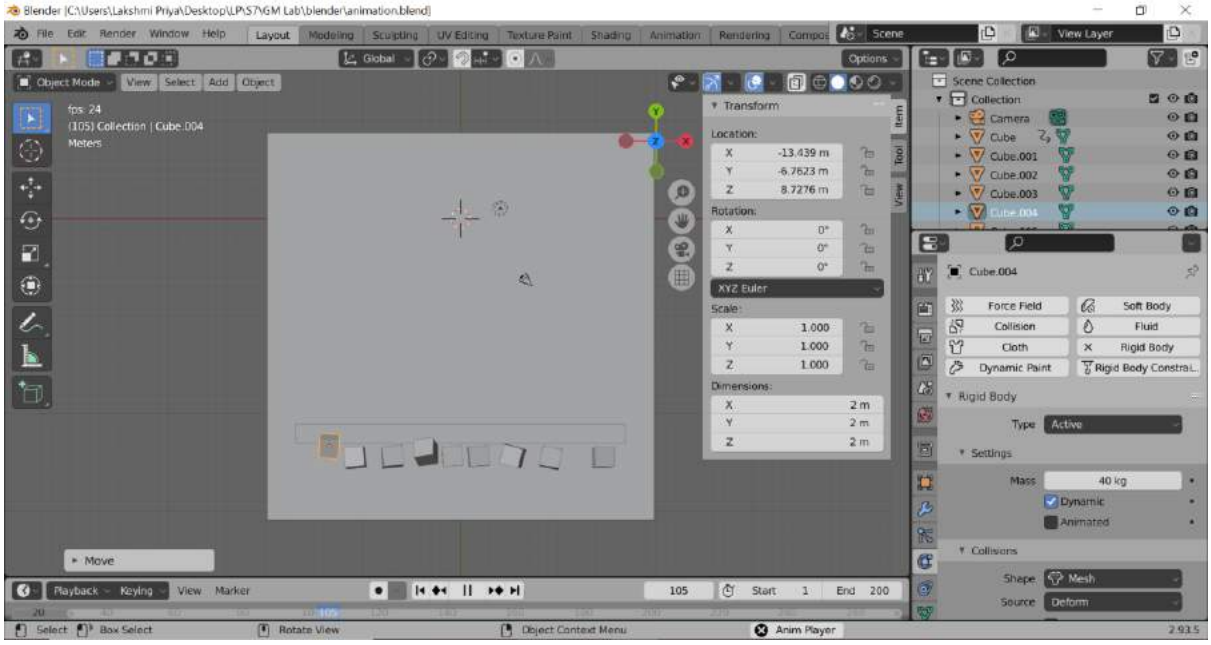

Blender
File Edit Render Window Help
Layout Modeling Sculpting UV Editing Texture Paint Shading Animation Rendering
Scene
View Layer
Object Mode View Select Add Object
Global
Options
User Perspective
(1) Collection | Cube.001
Scene Collection
Collection
Camera
Cube
Cube.001
Light
Plane
Collisions
Shape Mesh
Source Deform
Deforming
Surface Response
Friction 1.000
Bouncine... 0.000
Sensitivity
Margin 0.04 m
Collections
Dynamics
Damping ... 0.425
Rotation 0.626
Deactivation
Playback Keying View Marker
1
Start 1
End 250
Select
Box Select
Rotate View
Object Context Menu
2.93.5

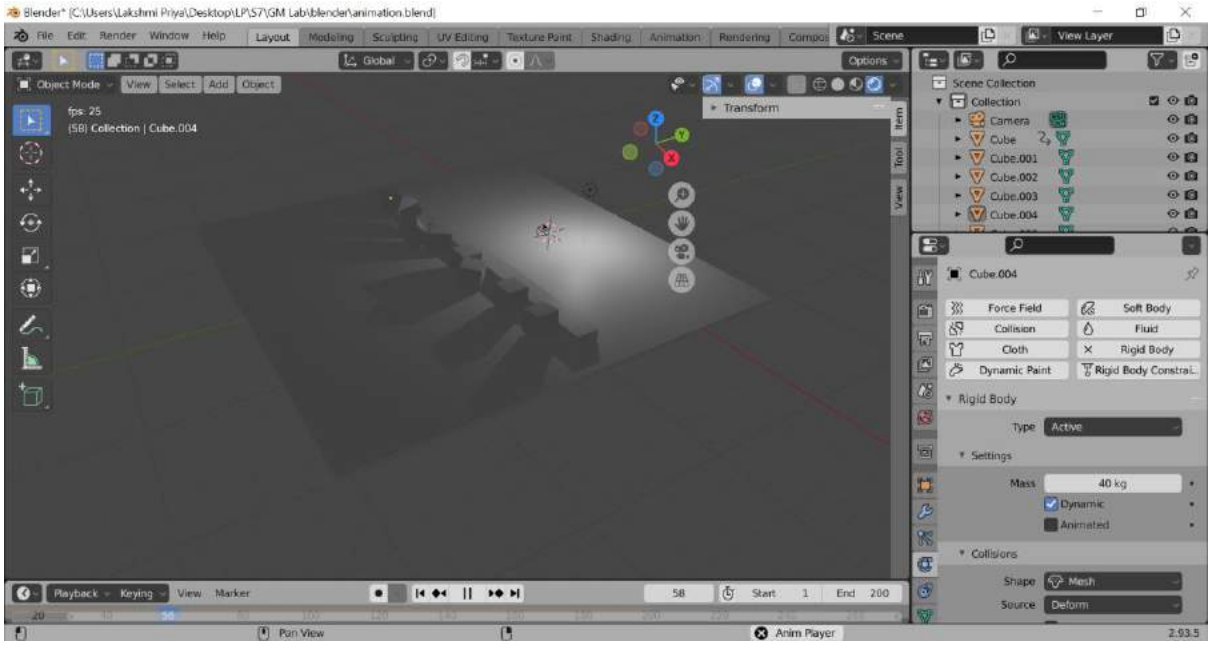

Blender
File Edit Render Window Help
Layout Modeling Sculpting UV Editing Texture Paint Shading Animation Rendering
Scene
View Layer
Object Mode View Select Add Object
Global
Options
User Perspective
(1) Collection | Cube
Scene Collection
Collection
Camera
Cube
Cube.001
Light
Plane
Rigid Body
Type Passive
Settings
Animated
Collisions
Shape Mesh
Source Deform
Deforming
Surface Response
Friction 1.000
Bouncine... 0.000
Sensitivity
Margin 0.04 m
Collections
Playback Keying View Marker
Start 1 End 250
Set Active Modifier
Pan View
Context Menu
2.93.5

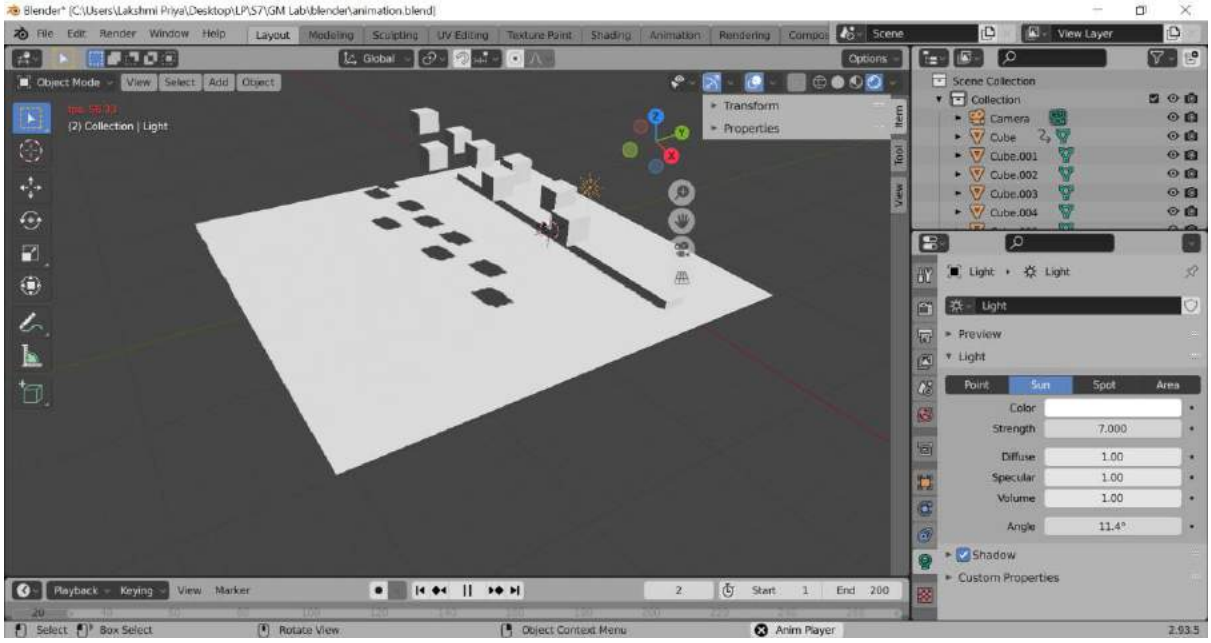

Blender* (C:\Users\Lakshmi Priya\Desktop\LP\S7\GM Lab\blender\animation.blend)
File Edit Render Window Help
Layout Modeling Sculpting UV Editing Texture Paint Shading Animation Rendering
Scene
View Layer
Object Mode View Select Add Object
Global
Options
Top Orthographic
(0) Collection | Cube.010
Meters
Transform
Item
Tool
View
Scene Collection
Collection
Camera
Cube
Cube.001
Cube.002
Cube.003
Cube.004
Material.001
Preview
Surface
Use Nodes
Surface Diffuse BSDF
Color
Roughness 0.000
Normal Default
Volume
Settings
Backface Culling
Blend Mode Opaque
Playback Keying View Marker
Start 1 End 180
Select
Box Select
Rotate View
Object Context Menu
2.93.5

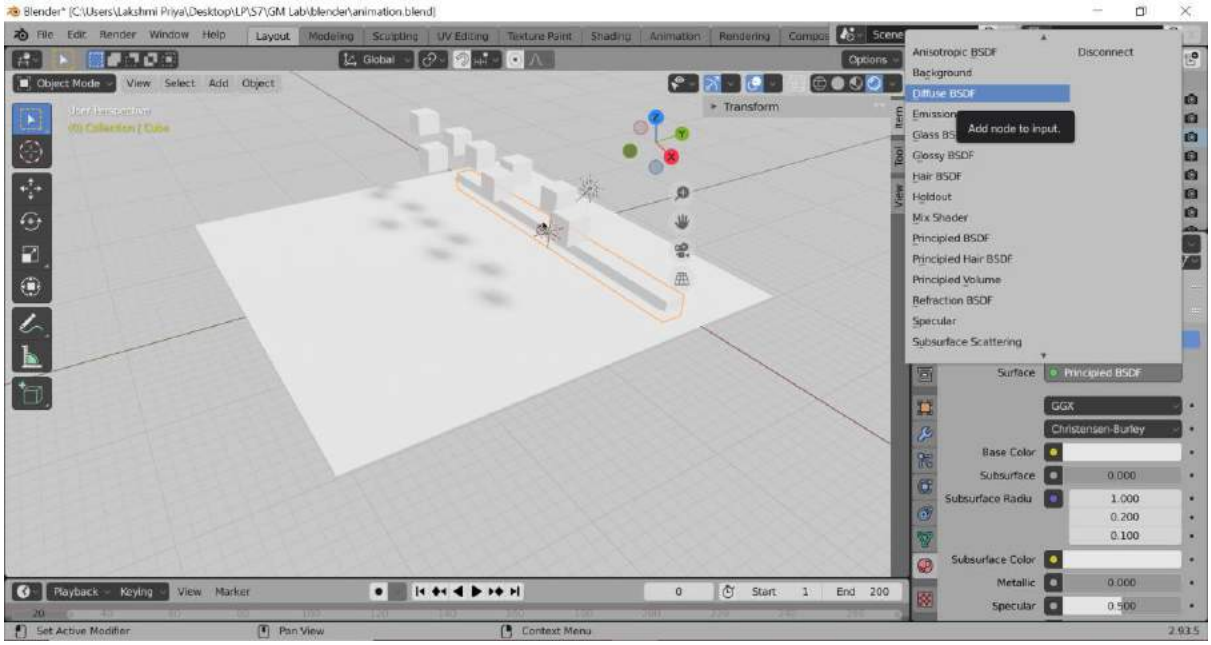

Blender [C:\Users\Lakshmi Priya\Desktop\LP\S7\GM Lab\blender\animation.blend]
File Edit Render Window Help
Layout Modeling Sculpting UV Editing Texture Paint Shading Animation Rendering Compos
Scene
View Layer
Object Mode View Select Add Object
Global
Options
User Perspective
(0) Collection | Cube
Transform
Item Tool View
Scene Collection
Collection
Camera
Cube
Cube.001
Cube.002
Cube.003
Cube.004
Material
Preview
Surface
Use Nodes
Surface Principled BSDF
GGX
Christensen-Burley
Base Color
Subsurface 0.000
Subsurface Radiu 1.000
0.200
0.100
Subsurface Color
Move
Playback Keying View Marker
0
Start 1
End 180
Select
Box Select
Rotate View
Object Context Menu
Saved "animation.blend"
2.93.5

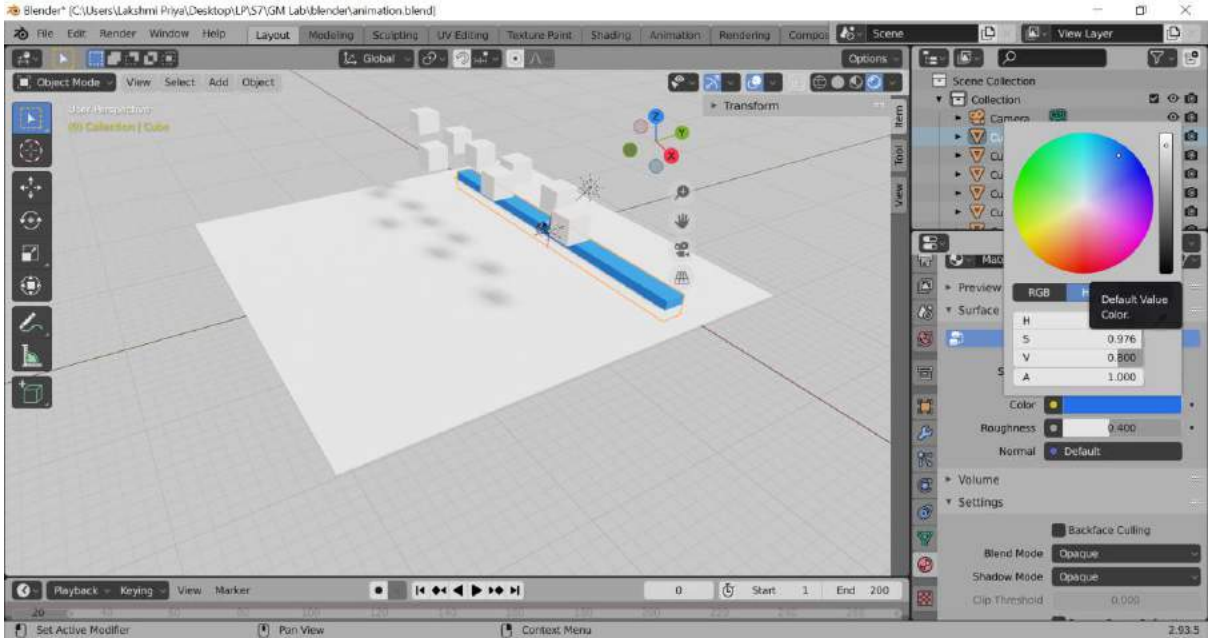

Blender* [C:\Users\Lakshmi Priya\Desktop\LP\S7\GM Lab\blender\animation.blend]
File Edit Render Window Help
Layout Modeling Sculpting UV Editing Texture Paint Shading Animation Rendering Compos
Scene
View Layer
Object Mode View Select Add Object
Global
Options
User Perspective
(67) Collection | Cube.009
Transform
Location:
X 13.651 m
Y -5.252 m
Z 8.7276 m
Rotation:
X 0°
Y 0°
Z 0°
XYZ Euler
Scale:
X 1.000
Y 1.000
Z 1.000
Dimensions:
X 2 m
Y 2 m
Z 2 m
Item Tool View
Scene Collection
Collection
Camera
Cube
Cube.001
Cube.002
Cube.003
Cube.004
Collisions
Shape Mesh
Source Deform
Deforming
Surface Response
Friction 1.000
Bounciness 0.000
Sensitivity
Margin 0.04 m
Collections
Dynamics
Damping Transla.. 1.000
Rotation 0.927
Deactivation
Playback Keying View Marker
67
Start 1
End 200
Select
Box Select
Rotate View
Object Context Menu
2.93.5

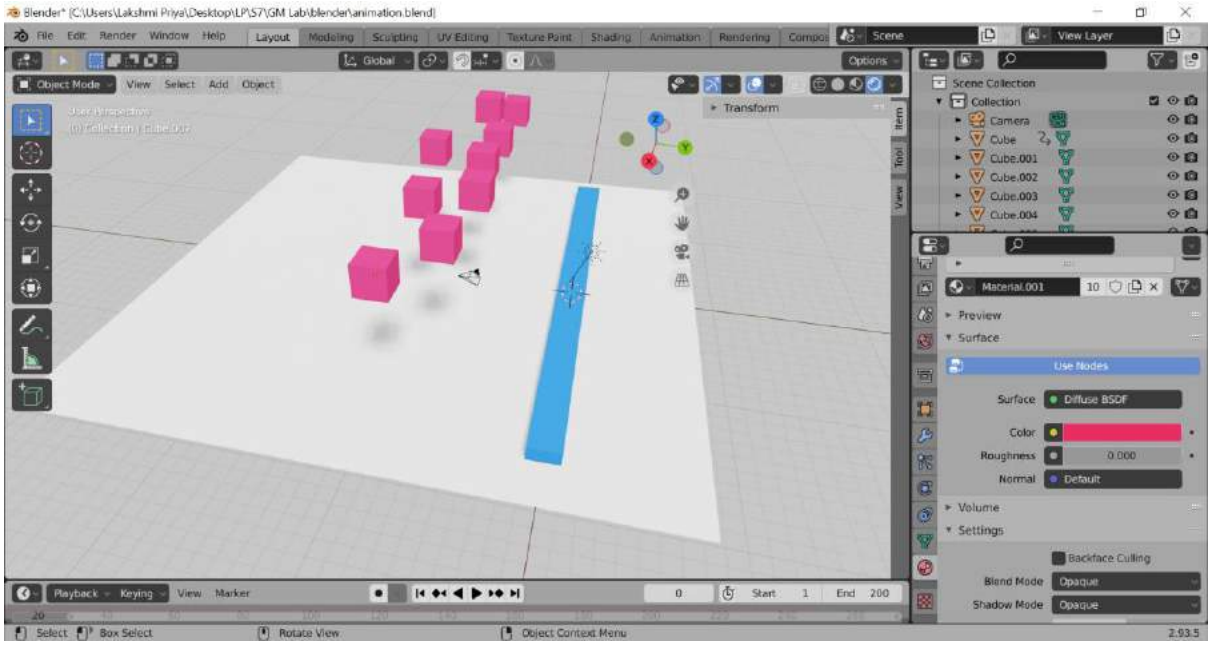

Blender [C:\Users\Lakshmi Priya\Desktop\LP\S7\GM Lab\blender\animation.blend]
File Edit Render Window Help
Layout Modeling Sculpting UV Editing Texture Paint Shading Animation Rendering
Scene
View Layer
Object Mode View Select Add Object
Global
Options
fps: 24
(105) Collection | Cube.004
Meters
Transform
Location:
X -13.439 m
Y -6.7623 m
Z 8.7276 m
Rotation:
X 0°
Y 0°
Z 0°
XYZ Euler
Scale:
X 1.000
Y 1.000
Z 1.000
Dimensions:
X 2 m
Y 2 m
Z 2 m
Scene Collection
Collection
Camera
Cube
Cube.001
Cube.002
Cube.003
Cube.004
Cube.004
Force Field
Soft Body
Collision
Fluid
Cloth
Rigid Body
Dynamic Paint
Rigid Body Constrai..
Rigid Body
Type Active
Settings
Mass 40 kg
Dynamic
Animated
Collisions
Shape Mesh
Source Deform
Move
Playback Keying View Marker
105
Start 1
End 200
Select
Box Select
Rotate View
Object Context Menu
Anim Player
2.93.5

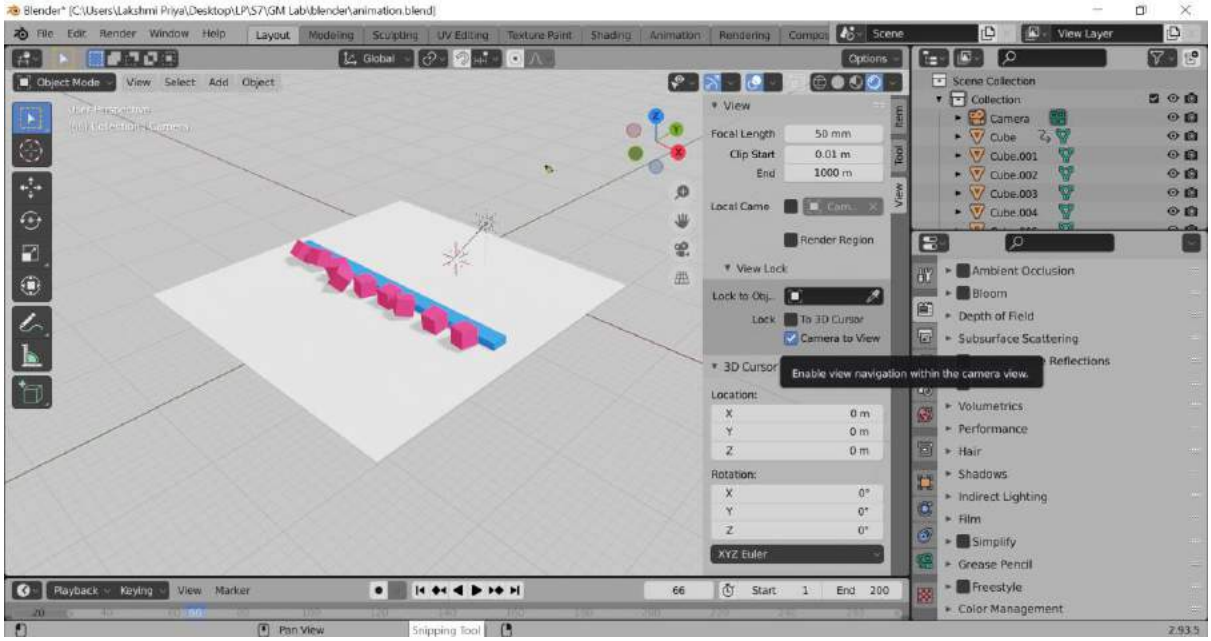

Blender* [C:\Users\Lakshmi Priya\Desktop\LP\S7\GM Lab\blender\animation.blend]
File Edit Render Window Help
Layout Modeling Sculpting UV Editing Texture Paint Shading Animation Rendering
Scene
View Layer
Object Mode View Select Add Object
Global
Options
fps: 25
(58) Collection | Cube.004
Transform
Scene Collection
Collection
Camera
Cube
Cube.001
Cube.002
Cube.003
Cube.004
Cube.004
Force Field
Soft Body
Collision
Fluid
Cloth
Rigid Body
Dynamic Paint
Rigid Body Constrai..
Rigid Body
Type Active
Settings
Mass 40 kg
Dynamic
Animated
Collisions
Shape Mesh
Source Deform
Playback Keying View Marker
58
Start 1
End 200
Pan View
Anim Player
2.93.5

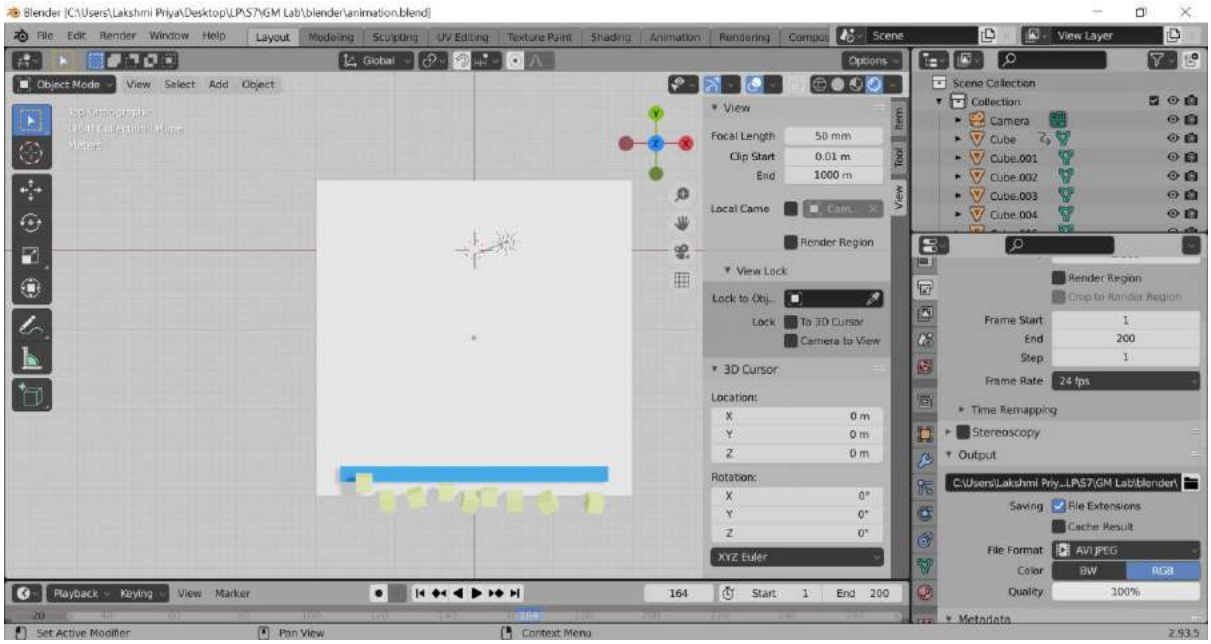

Blender* [C:\Users\Lakshmi Priya\Desktop\LP\S7\GM Lab\blender\animation.blend]
File Edit Render Window Help
Layout Modeling Sculpting UV Editing Texture Paint Shading Animation Rendering Compos
Scene
View Layer
Object Mode View Select Add Object
Global
Options
(2) Collection | Light
Transform
Properties
Scene Collection
Collection
Camera
Cube
Cube.001
Cube.002
Cube.003
Cube.004
Light
Preview
Point Sun Spot Area
Color
Strength 7.000
Diffuse 1.00
Specular 1.00
Volume 1.00
Angle 11.4°
Shadow
Custom Properties
Playback Keying View Marker
Start 1 End 200
Select Box Select
Rotate View
Object Context Menu
Anim Player
2.93.5

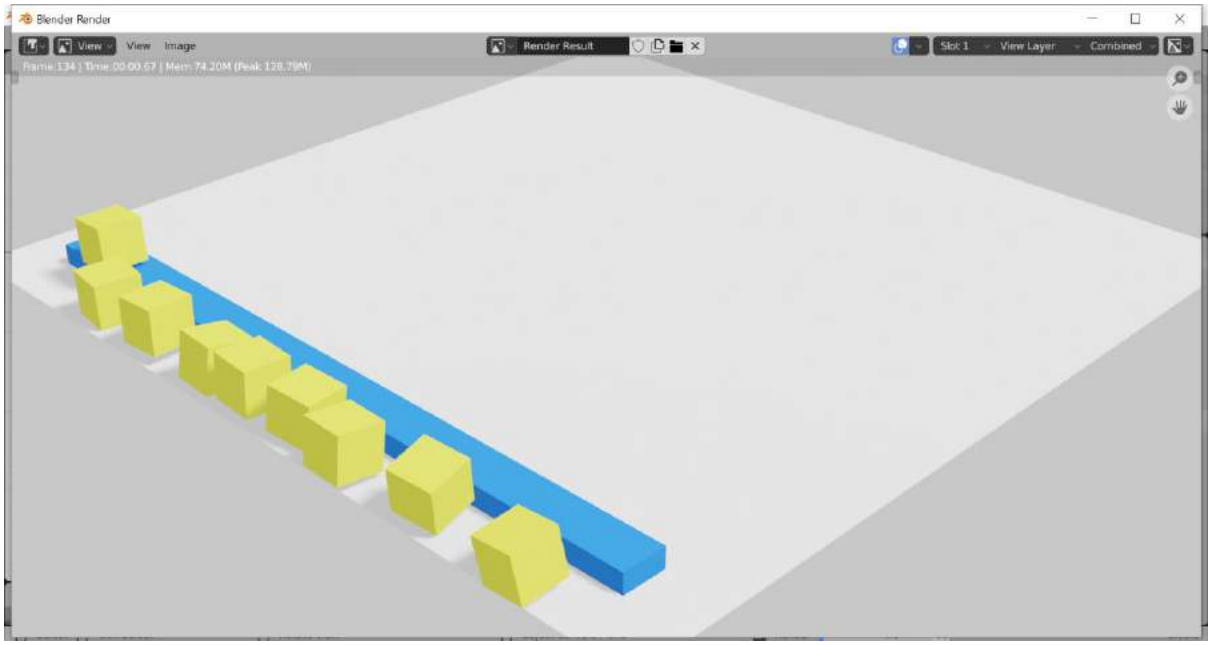

Blender* [C:\Users\Lakshmi Priya\Desktop\LP\S7\GM Lab\blender\animation.blend]
File Edit Render Window Help
Layout Modeling Sculpting UV Editing Texture Paint Shading Animation Rendering Compos
Object Mode View Select Add Object
Global
Options
Transform
Add node to input.
Anisotropic BSDF
Background
Diffuse BSDF
Emission
Glass BSDF
Glossy BSDF
Hair BSDF
Holdout
Mix Shader
Principled BSDF
Principled Hair BSDF
Principled Volume
Refraction BSDF
Specular
Subsurface Scattering
Disconnect
Surface
Principled BSDF
GGX
Christensen-Burley
Base Color
Subsurface 0.000
Subsurface Radiu 1.000 0.200 0.100
Subsurface Color
Metallic 0.000
Specular 0.500
Playback Keying View Marker
Start 1 End 200
Set Active Modifier
Pan View
Context Menu
2.93.5


Blender* [C:\Users\Lakshmi Priya\Desktop\LP\S7\GM Lab\blender\animation.blend]
File Edit Render Window Help
Layout Modeling Sculpting UV Editing Texture Paint Shading Animation Rendering
Scene
View Layer
Object Mode View Select Add Object
Global
Options
Transform
Scene Collection
Collection
Camera
Default Value
Color
RGB
H
S 0.976
V 0.800
A 1.000
Preview
Surface
Color
Roughness 0.400
Normal Default
Volume
Settings
Backface Culling
Blend Mode Opaque
Shadow Mode Opaque
Clip Threshold 0.000
Playback Keying View Marker
Start 1 End 200
Set Active Modifier
Pan View
Context Menu
2.93.5


Blender* [C:\Users\Lakshmi Priya\Desktop\LP\S7\GM Lab\blender\animation.blend]
File Edit Render Window Help
Layout Modeling Sculpting UV Editing Texture Paint Shading Animation Rendering
Scene
View Layer
Object Mode View Select Add Object
Global
Options
Transform
Scene Collection
Collection
Camera
Cube
Cube.001
Cube.002
Cube.003
Cube.004
Material.001
Preview
Surface
Use Nodes
Surface Diffuse BSDF
Color
Roughness 0.000
Normal Default
Volume
Settings
Backface Culling
Blend Mode Opaque
Shadow Mode Opaque
Playback Keying View Marker
Start 1 End 200
Select
Box Select
Rotate View
Object Context Menu
2.93.5


Blender* [C:\Users\Lakshmi Priya\Desktop\LP\S7\GM Lab\blender\animation.blend]
File Edit Render Window Help
Layout Modeling Sculpting UV Editing Texture Paint Shading Animation Rendering Compos
Scene
View Layer
Object Mode View Select Add Object
Global
Options
View
Focal Length 50 mm
Clip Start 0.01 m
End 1000 m
Local Came
Render Region
View Lock
Lock to Obj.
Lock To 3D Cursor
Camera to View
3D Cursor
Location:
X 0 m
Y 0 m
Z 0 m
Rotation:
X 0°
Y 0°
Z 0°
XYZ Euler
Enable view navigation within the camera view.
Scene Collection
Collection
Camera
Cube
Cube.001
Cube.002
Cube.003
Cube.004
Ambient Occlusion
Bloom
Depth of Field
Subsurface Scattering
Reflections
Volumetrics
Performance
Hair
Shadows
Indirect Lighting
Film
Simplify
Grease Pencil
Freestyle
Color Management
Playback Keying View Marker
66
Start 1
End 200
Pan View
Snipping Tool
2.93.5


Blender [C:\Users\Lakshmi Priya\Desktop\LP\S7\GM Lab\blender\animation.blend]
File Edit Render Window Help
Layout Modeling Sculpting UV Editing Texture Paint Shading Animation Rendering Compos
Scene
View Layer
Object Mode View Select Add Object
Global
Options
View
Focal Length 50 mm
Clip Start 0.01 m
End 1000 m
Local Came
Render Region
View Lock
Lock to Obj.
Lock To 3D Cursor
Camera to View
3D Cursor
Location:
X 0 m
Y 0 m
Z 0 m
Rotation:
X 0°
Y 0°
Z 0°
XYZ Euler
Scene Collection
Collection
Camera
Cube
Cube.001
Cube.002
Cube.003
Cube.004
Render Region
Crop to Render Region
Frame Start 1
End 200
Step 1
Frame Rate 24 fps
Time Remapping
Stereoscopy
Output
C:\Users\Lakshmi Priy...LP\S7\GM Lab\blender\
Saving File Extensions
Cache Result
File Format AVI JPEG
Color BW RGB
Quality 100%
Metadata
Playback Keying View Marker
164
Start 1
End 200
Set Active Modifier
Pan View
Context Menu
2.93.5



Output Animation File

Attached as “3D_animation.avi” as separate file

Result

Thus, a simple 3D animation sequence has been created with objects using Blender Software.

3D ANIMATION

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

void initialize(void)
{
	glClearColor(1.0, 1.0, 1.0, 0.0);
	//glShadeModel(GL_SMOOTH);
	//GLfloat light_diffuse[] = { 1.0, 1.0, 1.0, 1.0 };
	//GLfloat light_position[] = { 0, 0, 1, 0 };
	//glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
	//glLightfv(GL_LIGHT0, GL_POSITION, light_position);
	glEnable(GL_LIGHTING);
	glEnable(GL_LIGHT0);
	glEnable(GL_DEPTH_TEST);
}

int INC = 1;
void drawScene(int state)
{
	if (state == 0)
		INC = 1;
	else if (state == 10)
		INC = -1;

	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
	glLoadIdentity();
	gluLookAt(0.0, 1.0, 7.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
	glMatrixMode(GL_MODELVIEW);

	// Cube
	glPushMatrix();
	GLfloat cube_color[] = { 1, 0.5, 0.0, 0.0 };
	glMaterialfv(GL_FRONT, GL_DIFFUSE, cube_color);
	glScalef(4, 1.5, 1.0);
	glTranslatef(0.2, -1.0, 0.0);
	glutSolidCube(1.0);
	glPopMatrix();

	// Torus
	glPushMatrix();
	GLfloat torus_color[] = { 0.59, 0.1, 0.55, 1.0 };
	glMaterialfv(GL_FRONT, GL_DIFFUSE, torus_color);
	glTranslatef(-3, -1.5, 0.0);
	glutSolidTorus(0.3, 0.7, 10, 10);
	glPopMatrix();

	// Teapot
	glPushMatrix();
	glEnable(GL_TEXTURE_2D);
	GLfloat teapot_color[] = { 0.7, 0.7, 0.7, 0.0 };
	GLfloat mat_shininess[] = { 100 };
	glMaterialfv(GL_FRONT, GL_DIFFUSE, teapot_color);
	glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);
	glRotatef(45, 0, 0, 1);
	glTranslatef(-1.2, 0.8, 0.0);
	glutSolidTeapot(0.7);
	glDisable(GL_TEXTURE_2D);
	glPopMatrix();

	// Sphere
	glPushMatrix();
```

```
	GLfloat ball_color[] = { 0.0, 1, 1, 1.0 };
	glMaterialfv(GL_FRONT, GL_DIFFUSE, ball_color);
	glTranslatef(2, 2.1 - 0.25 * state, 0);
	glutSolidSphere(0.5, 10, 10);
	glPopMatrix();

	glutSwapBuffers();
	glutTimerFunc(1000 / 60, drawScene, state + INC);
}

void reshape(int w, int h)
{
	glViewport(0, 0, (GLsizei)w, (GLsizei)h);
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	gluPerspective(75, 1, 1, 20);
	glMatrixMode(GL_MODELVIEW);
}

void sceneDemo() {
	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
	glutTimerFunc(1000 / 60, drawScene, 0);
}

int main(int argc, char** argv)
{
	glutInit(&argc, argv);
	glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
	glutInitWindowSize(500, 500);
	glutCreateWindow("3D Scene");
	initialize();
	glutDisplayFunc(sceneDemo);
	glutReshapeFunc(reshape);
	glutMainLoop();
	return 0;
}
```

MENU

```
#include <windows.h>
#include <stdio.h>
#include <gl/glut.h>
#include <conio.h>
#include <math.h>

void myInit() {
	glClearColor(1.0, 1.0, 1.0, 0.0);
	glColor3f(1.0f, 1.0f, 1.0f);
	glPointSize(2);
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	gluOrtho2D(0.0, 640.0, 0.0, 480.0);
}
void myDisplay(int x, int y) {
	glColor3f(0.0f, 0.0f, 0.0f);
	glBegin(GL_POINTS);
	glVertex2f(x, y);
	glEnd();
}
void myShow() {
}
```

```
void Bresenham(int x1, int y1, int x2, int y2) {
	glClear(GL_COLOR_BUFFER_BIT);
	// int x0 = 100; int y0 = 80; int xn = 240; int yn = 140;
	int dx = x2 - x1;
	int dy = y2 - y1;
	int p, i;
	dx = abs(dx);
	dy = abs(dy);
	float m = (float)dy / (float)dx;
	int x = x1, y = y1;
	printf("Initial Point: (%d, %d)\n", x, y);
	printf("Slope is %f\n", m);
	int flag = 0;
	if (dx >= dy) { // m<=1, sampling along x-axis
		myDisplay(x, y);
		if (y2 > y) {
			flag = 1; // taking care of left to right(and right to left)
		}
		p = (2 * dy) - dx;
		if (x < x2) { // left to right
			while (x < x2) {
				x = x + 1;
				if (p >= 0) {
					p = p + ((2 * dy) - (2 * dx));
					if (flag == 1) {
						y = y + 1;
					}
					else {
						y = y - 1;
					}
				}
				else {
					p = p + (2 * dy);
				}
				myDisplay(x, y);
				//printf("x=%d y=%d p=%d\n",x,y,p);
			}
		}
		else {
			while (x > x2) { // right to left
				x = x - 1;
				if (p >= 0) {
					p = p + ((2 * dy) - (2 * dx));
					if (flag == 1) {
						y = y + 1;
					}
					else {
						y = y - 1;
					}
				}
				else {
					p = p + (2 * dy);
				}
				myDisplay(x, y);
				//printf("x=%d y=%d p=%d\n",x,y,p);
			}
		}
	}
	else { // m>1, sampling along y-axis
		myDisplay(x, y);
		if (x2 > x) {
			flag = 1; // taking care of left to right and right to left
```

```
		}
		p = (2 * dx) - dy;
		if (y < y2) {
			while (y < y2) { // left to right
				y = y + 1;
				if (p >= 0) {
					p = p + ((2 * dx) - (2 * dy));
					if (flag == 1) {
						x = x + 1;
					}
					else {
						x = x - 1;
					}
				}
				else {
					p = p + (2 * dx);
				}
				myDisplay(x, y);
				//printf("x=%d y=%d p=%d\n",x,y,p);
			}
		}
		else {
			while (y > y2) { // right to left
				y = y - 1;
				if (p >= 0) {
					p = p + ((2 * dx) - (2 * dy));
					if (flag == 1) {
						x = x + 1;
					}
					else {
						x = x - 1;
					}
				}
				else {
					p = p + (2 * dx);
				}
				myDisplay(x, y);
				//printf("x=%d y=%d p=%d\n",x,y,p);
			}
		}
	}
	printf("Final Point: (%d, %d)\n\n\n", x, y);
	glFlush();
}
void mainMenuHandler(int choice) {
	switch (choice) {
	case 1:
		printf("Left to Right +ve Slope m>1\n");
		Bresenham(100, 140, 200, 300);
		break;
	case 2:
		printf("Right to Left +ve Slope m>1\n");
		Bresenham(200, 200, 100, 40);
		break;
	case 3:
		printf("Left to Right -ve Slope m>1\n");
		Bresenham(40, 200, 180, 20);
		break;
	case 4:
		printf("Right to Left -ve Slope m>1\n");
		Bresenham(200, 200, 60, 360);
		break;
```

```
        case 5:
                printf("Left to Right +ve Slope m<=1\n");
                Bresenham(100, 80, 240, 140);
                break;
        case 6:
                printf("Right to Left +ve Slope m<=1\n");
                Bresenham(200, 200, 40, 140);
                break;
        case 7:
                printf("Left to Right -ve Slope m<=1\n");
                Bresenham(40, 200, 200, 40);
                break;
        case 8:
                printf("Right to Left -ve Slope m<=1\n");
                Bresenham(200, 200, 60, 260);
                break;
        default:
                break;
        }
}
int main(int argc, char* argv[]) {
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
        glutInitWindowSize(640, 480);
        glutCreateWindow("Bresenham's Line Drawing Algorithm");
        glutDisplayFunc(myShow);
        myInit();
        glutCreateMenu(mainMenuHandler);
        glutAddMenuEntry("Left to Right +ve Slope m<=1", 5);
        glutAddMenuEntry("Right to Left +ve Slope m<=1", 6);
        glutAddMenuEntry("Left to Right -ve Slope m<=1", 7);
        glutAddMenuEntry("Right to Left -ve Slope m<=1", 8);
        glutAddMenuEntry("Left to Right +ve Slope m>1", 1);
        glutAddMenuEntry("Right to Left +ve Slope m>1", 2);
        glutAddMenuEntry("Left to Right -ve Slope m>1", 3);
        glutAddMenuEntry("Right to Left -ve Slope m>1", 4);
        glutAttachMenu(GLUT_RIGHT_BUTTON);
        glutMainLoop();
        return 1;
}
```