| | | | | | |
|---|---|---|---|---|---|
| **Name** | **:** | **V Vikram** | **Class** | **:** CSE ‘C’ |
| **Reg. No.** | **:** | **18 5001 194** | **Date** | **:** 10/10/2021 |
| **Subject** | **:** | **UCS1712---Graphics and Multimedia Lab** | | |

## QUESTION :

# Lab Exercise 8
# 3-Dimensional Transformations in C++ using OpenGL

- Perform the following basic 3D Transformations on any 3D Object.
    1) Translation
    2) Rotation
    3) Scaling
- Use only homogeneous coordinate representation and matrix multiplication to perform transformations.
- Set the camera to any position on the 3D space.
- Have (0,0,0) at the center of the screen. Draw X , Y and Z axis.

## CODE :-

## Source.cpp :

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

using namespace std;
constexpr auto PI = 3.14;

//Output Window constraints
const int WINDOW_WIDTH = 800;
const int WINDOW_HEIGHT = 800;
const int X_MIN = -675;
const int X_MAX = 675;
const int Y_MIN = -675;
const int Y_MAX = 675;
const int Z_MIN = -675;
const int Z_MAX = 675;
int option;
double tx, ty, tz;      //translation factors
double ang, angRad;     //angle and radian conversion
double sx, sy, sz;      // scaling factors
double xf, yf, zf;      // fixed points

//to store the TRANSFORMATION matrix
vector<vector<double>> Transformation(4, vector<double>(4, 0));
//to store the 3D object's vertices
vector<vector<double>> vertices(24, vector<double>(3, 0));

vector<vector<double>> scale();
vector<vector<double>> set_Vertices();
vector<vector<double>> Transform_Object();
vector<vector<double>> translate();
void draw_3D_Object(vector<vector<double>> temp, int count);
void menu_driven();
void disp();

#include"Header.h"

void init()
{
      glClearColor(1.0, 1.0, 1.0, 1.0);// Set background color to black
and opaque
      glOrtho(X_MIN, X_MAX, Y_MIN, Y_MAX, Z_MIN, Z_MAX);
      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
```

```
	glEnable(GL_DEPTH_TEST);
	glEnable(GL_BLEND);
	glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

}

void myDisplay()
{
	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);// Clear color
and depth buffers

	//draw X_Y_Z planes
	glBegin(GL_LINES);
	glColor3f(0.0, 0.0, 0.0);
	glVertex3d(-1100, -800, 0);
	glVertex3d(0, 0, 0);
	glVertex3d(0,0,0);
	glVertex3d(1100,0,0);
	glVertex3d(0, 0, 0);
	glVertex3d(0,1100,1100);
	glEnd();

	disp(); //to rotate ORIGINAL object
	draw_3D_Object(vertices,0);
	menu_driven();
	glFlush();
}

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

	glutInit(&argc, argv);
	glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
	glutInitWindowSize(WINDOW_WIDTH, WINDOW_HEIGHT);
	glutInitWindowPosition(0, 0);
	glutCreateWindow("Ex-8: 3-Dimensional Transformations");

	vertices = set_Vertices();
	cout << "\n\t\t-----------------------------------";
	cout << "\n\t\tEx-8: 3-Dimensional Transformations";
	cout << "\n\t\t-----------------------------------";
	cout << "\n\n\tOptions :-";
	cout << "\n\t\t1) Translation";
	cout << "\n\t\t2) Rotation";
	cout << "\n\t\t3) Scaling ";
	cout << "\n\n\t\tSelect option -> ";
	cin >> option;


	init();
	glutDisplayFunc(myDisplay);
	glutMainLoop();
	return 0;
}
```

## Header.h:

```
#pragma once

vector<vector<double>> translate()
{
	vector<vector<double>> temp(4, vector<double>(4, 0));

	temp[0][0] = 1;
	temp[1][1] = 1;
	temp[2][2] = 1;
	temp[3][3] = 1;
	temp[0][3] = tx;
	temp[1][3] = ty;
	temp[2][3] = tz;

	return temp;
}

vector<vector<double>> scale()
{
	vector<vector<double>> temp(4, vector<double>(4, 0));

	temp[0][0] = sx;
	temp[1][1] = sy;
	temp[2][2] = sz;
	temp[3][3] = 1;
	temp[0][3] = (1 - sx)*xf;
	temp[1][3] = (1 - sy)*yf;
	temp[2][3] = (1 - sz)*zf;

	return temp;
}

vector<vector<double>> rotate_X()
{
	vector<vector<double>> temp(4, vector<double>(4, 0));

	temp[0][0] = 1;
	temp[1][1] = cos(angRad);
	temp[2][2] = cos(angRad);
	temp[3][3] = 1;
	temp[2][1] = sin(angRad);
	temp[1][2] = -1 * sin(angRad);

	return temp;
}

vector<vector<double>> rotate_Y()
{
	vector<vector<double>> temp(4, vector<double>(4, 0));
```

```
	temp[0][0] = cos(angRad);
	temp[1][1] = 1;
	temp[2][2] = cos(angRad);
	temp[3][3] = 1;
	temp[0][2] = sin(angRad);
	temp[2][0] = -1 * sin(angRad);

	return temp;
}

vector<vector<double>> rotate_Z()
{
	vector<vector<double>> temp(4, vector<double>(4, 0));

	temp[0][0] = cos(angRad);
	temp[1][1] = cos(angRad);
	temp[2][2] = 1;
	temp[3][3] = 1;
	temp[1][0] = sin(angRad);
	temp[0][1] = -1 * sin(angRad);

	return temp;
}

/* TRANSLATE the rotated object
(to prevent original and transformed
object from juxtaposing) */

vector<vector<double>> move_rotate()
{
	Transformation[0][3]  += 400 ;
	Transformation[1][3]  += 0;
	Transformation[2][3]  += 300;

	return Transformation;
}

vector<vector<double>> Transform_Object()
{
	vector<vector<double>> newVertices(24, vector<double>(3, 0));

	for (int i = 0; i < 24; i++) {
		vector<double> curpoint(4, 0), matProduct(4, 0);
		curpoint[0] = vertices[i][0];
		curpoint[1] = vertices[i][1];
		curpoint[2] = vertices[i][2];
		curpoint[3] = 1;

		for (int j = 0; j < 4; j++) {
			for (int k = 0; k < 4; k++) {
				matProduct[j] += Transformation[j][k] *
curpoint[k];
```

```
                }
            }
            newVertices[i][0] = round(matProduct[0]);
            newVertices[i][1] = round(matProduct[1]);
            newVertices[i][2] = round(matProduct[2]);
      }
      return newVertices;
}

void menu_driven()
{
      vector<vector<double>> new_vertices(24, vector<double>(3, 0));
      char sub_opt;
      switch (option) {
            case 1: {
                  cout << "\n\n\t\tTranslation factors ( tx ty tz ) : ";
                  cin >> tx >> ty >> tz;
                  Transformation = translate();
                  new_vertices = Transform_Object();
                  draw_3D_Object(new_vertices,1);
                  break;
            }
            case 2: {
                  cout << "\n\n\t\tAngle of rotation : ";
                  cin >> ang;
                  angRad = ang * PI / 180;

                  cout << "\n\n\t\tRotate about : ";
                  cout << "\n\n\t\t\ta) X-axis ";
                  cout << "\n\n\t\t\tb) Y-axis ";
                  cout << "\n\n\t\t\tc) Z-axis -> ";
                  cin >> sub_opt;
                  if (sub_opt == 'a')    Transformation = rotate_X();
                  else if (sub_opt == 'b')     Transformation =
rotate_Y();
                  else if (sub_opt == 'c')     Transformation =
rotate_Z();
                  else cout << "\n\n\t\t\tINVALID INPUT!!";
                  Transformation = move_rotate();
                  new_vertices = Transform_Object();
                  draw_3D_Object(new_vertices,1);
                  break;
            }
            case 3: {
                  cout << "\n\n\t\tScaling factors ( sx sy sz ) : ";
                  cin >> sx >> sy >> sz;
                  cout << "\n\t\tScale about : ";
                  cin >> xf >> yf >>zf;
                  Transformation = scale();
                  new_vertices = Transform_Object();
                  draw_3D_Object(new_vertices,1);
                  break;
            }
```

```
            default: cout << "\n\n\t\t\tINVALID INPUT!!";
      }
}

//Set the vertices of the 3D object
vector<vector<double>> set_Vertices() {

      vector<vector<double>> vertices(24, vector<double>(3, 0));

      vertices[0][0] = -100.0;
      vertices[0][1] = 100.0;
      vertices[0][2] = 100.0;

      vertices[1][0] = 100.0;
      vertices[1][1] = 100.0;
      vertices[1][2] = 100.0;

      vertices[2][0] = 100.0;
      vertices[2][1] = -100.0;
      vertices[2][2] = 100.0;

      vertices[3][0] = -100.0;
      vertices[3][1] = -100.0;
      vertices[3][2] = 100.0;
      //----------------------

      vertices[4][0] = -100.0;
      vertices[4][1] = 100.0;
      vertices[4][2] = -100.0;

      vertices[5][0] = 100.0;
      vertices[5][1] = 100.0;
      vertices[5][2] = -100.0;

      vertices[6][0] = 100.0;
      vertices[6][1] = -100.0;
      vertices[6][2] = -100.0;

      vertices[7][0] = -100.0;
      vertices[7][1] = -100.0;
      vertices[7][2] = -100.0;
      //----------------------

      vertices[8][0] = -100.0;
      vertices[8][1] = 100.0;
      vertices[8][2] = -100.0;

      vertices[9][0] = -100.0;
      vertices[9][1] = 100.0;
      vertices[9][2] = 100.0;

      vertices[10][0] = -100.0;
      vertices[10][1] = -100.0;
```

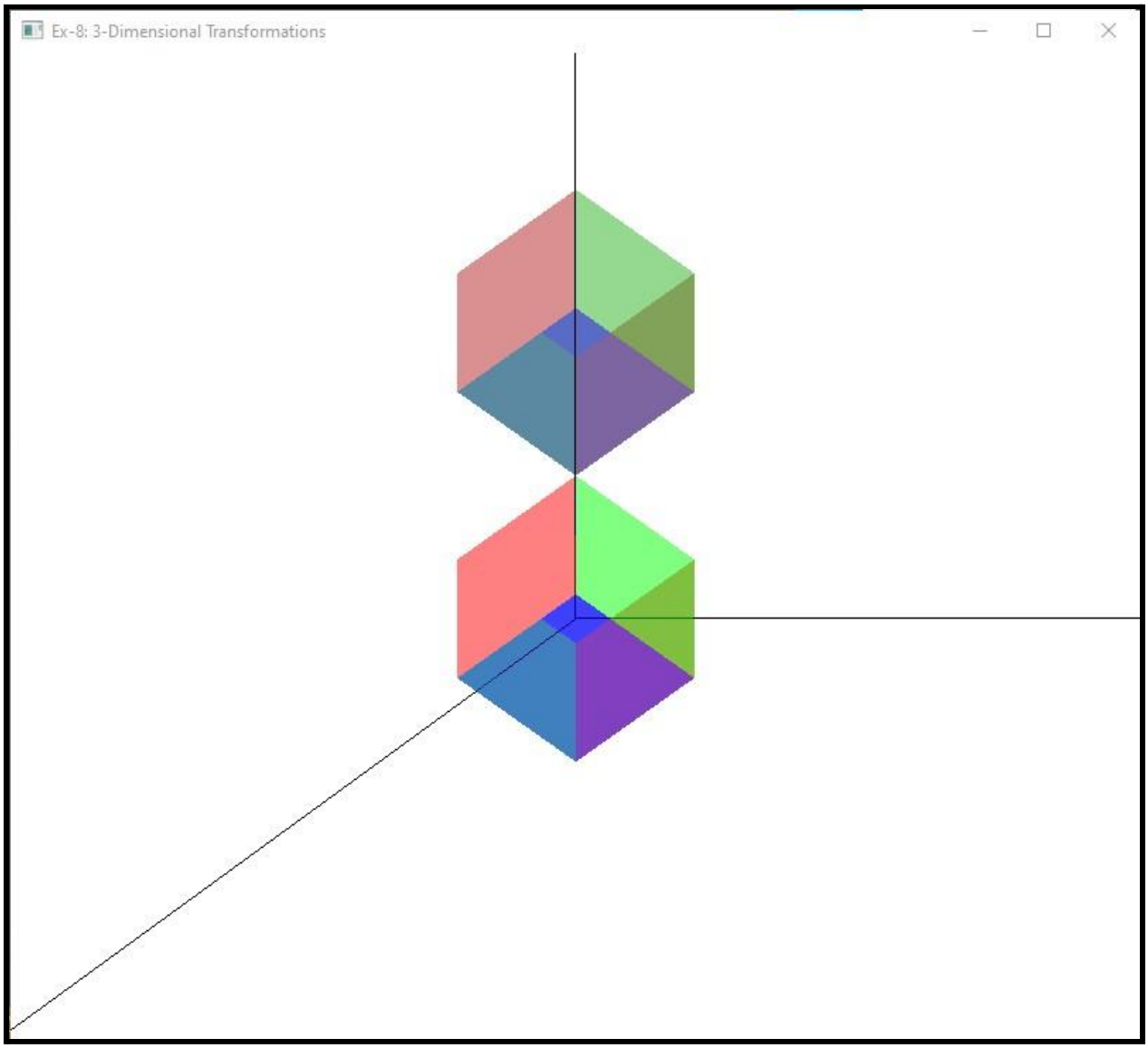
```
    vertices[10][2] = 100.0;

    vertices[11][0] = -100.0;
    vertices[11][1] = -100.0;
    vertices[11][2] = -100.0;
    //----------------------

    vertices[12][0] = 100.0;
    vertices[12][1] = 100.0;
    vertices[12][2] = -100.0;

    vertices[13][0] = 100.0;
    vertices[13][1] = 100.0;
    vertices[13][2] = 100.0;

    vertices[14][0] = 100.0;
    vertices[14][1] = -100.0;
    vertices[14][2] = 100.0;

    vertices[15][0] = 100.0;
    vertices[15][1] = -100.0;
    vertices[15][2] = -100.0;
    //----------------------

    vertices[16][0] = -100.0;
    vertices[16][1] = 100.0;
    vertices[16][2] = -100.0;

    vertices[17][0] = 100.0;
    vertices[17][1] = 100.0;
    vertices[17][2] = -100.0;

    vertices[18][0] = 100.0;
    vertices[18][1] = 100.0;
    vertices[18][2] = 100.0;

    vertices[19][0] = -100.0;
    vertices[19][1] = 100.0;
    vertices[19][2] = 100.0;
    //----------------------

    vertices[20][0] = -100.0;
    vertices[20][1] = -100.0;
    vertices[20][2] = -100.0;

    vertices[21][0] = 100.0;
    vertices[21][1] = -100.0;
    vertices[21][2] = -100.0;

    vertices[22][0] = 100.0;
    vertices[22][1] = -100.0;
    vertices[22][2] = 100.0;
```

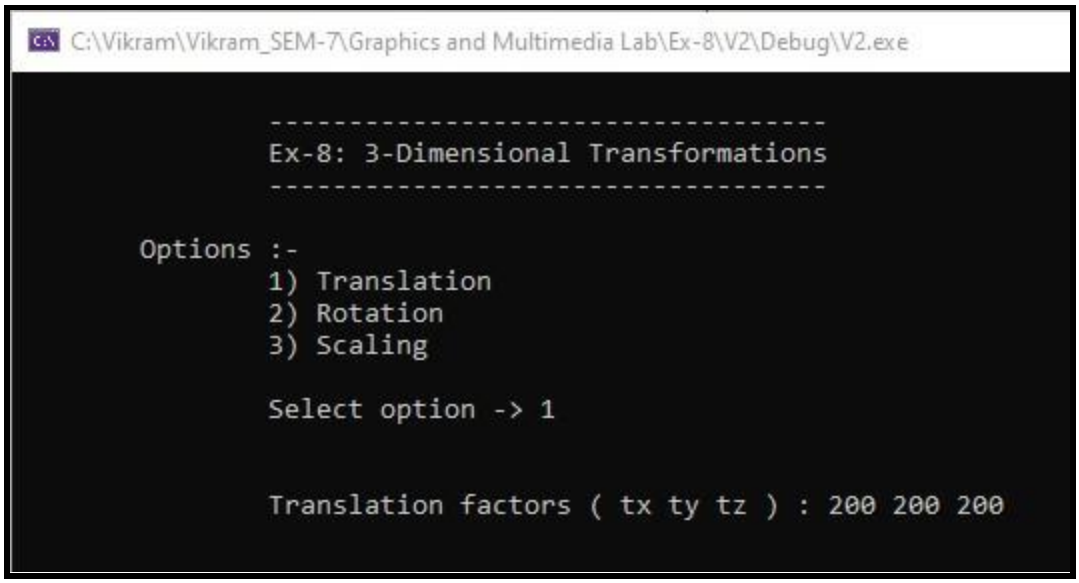
```
	vertices[23][0] = -100.0;
	vertices[23][1] = -100.0;
	vertices[23][2] = 100.0;

	return vertices;
}

void draw_3D_Object(vector<vector<double>> temp,int count) {
	vector<vector<double>> color(3, vector<double>(3, 0));

	//color for ORIGINAL object
	if (count == 0) {
		color[0][0] = 1.0f;

		color[1][1] = 1.0f;

		color[2][2] = 1.0;
	}
	//color for TRANSFORMED object
	if (count == 1) {
		color[0][0] = 179.0 / 255.0f;
		color[0][1] = 33.0 / 255.0f;
		color[0][2] = 33.0 / 255.0f;

		color[1][0] = 43.0 / 255.0f;
		color[1][1] = 179.0 / 255.0f;
		color[1][2] = 33.0 / 255.0f;

		color[2][0] = 33.0 / 255.0f;
		color[2][1] = 57.0 / 255.0f;
		color[2][2] = 179.0 / 255.0f;
	}

	glBegin(GL_QUADS);

	//Color for FRONT and BACK sides of the 3D object
	glColor4f(color[0][0], color[0][1], color[0][2],0.5);

	for (int i = 0; i < 24; i++) {
		//Color for LEFT and RIGHT sides of the 3D object
		if (i == 8 )
			glColor4f(color[1][0], color[1][1], color[1][2], 0.5);

		//Color for TOP and BOTTOM sides of the 3D object
		if (i == 16 )
			glColor4f(color[2][0], color[2][1], color[2][2], 0.5);

		glVertex3f(temp[i][0], temp[i][1], temp[i][2]);

		if ( (i+1) % 4 == 0) {
			glEnd();	//END if 4 end-points are plotted
			if(i!=23)	glBegin(GL_QUADS);	//BEGIN if not the
end of ARRAY
```

```
            }
      }
}

//Transformations - Built-in functions- only for reference

void disp()
{
      glRotatef(45, 1, 0, 0);
      glRotatef(45, 0, 1, 0);
      glRotatef(0, 0, 0, 1);
}
```

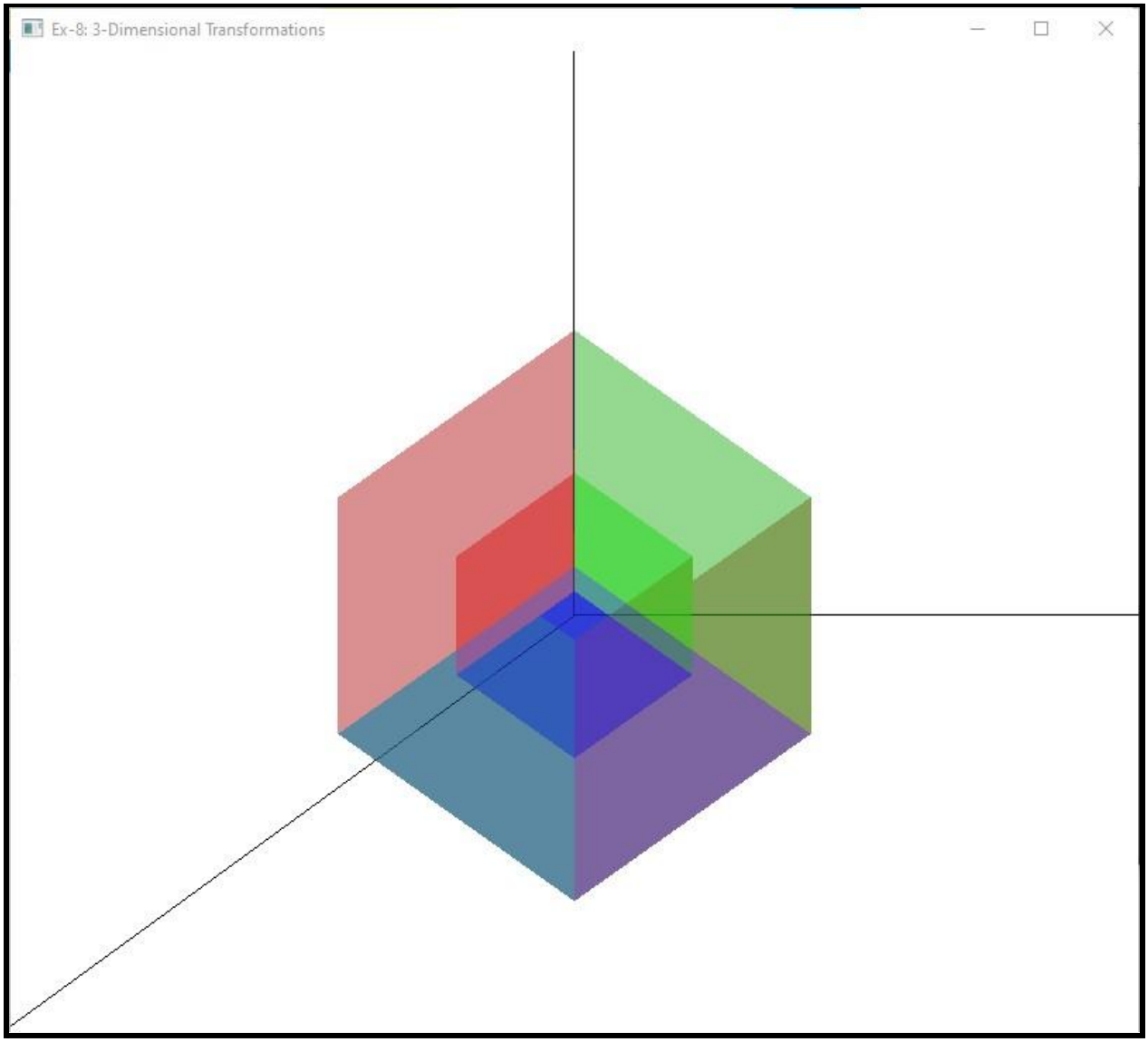
## OUTPUT SNAPSHOTS :

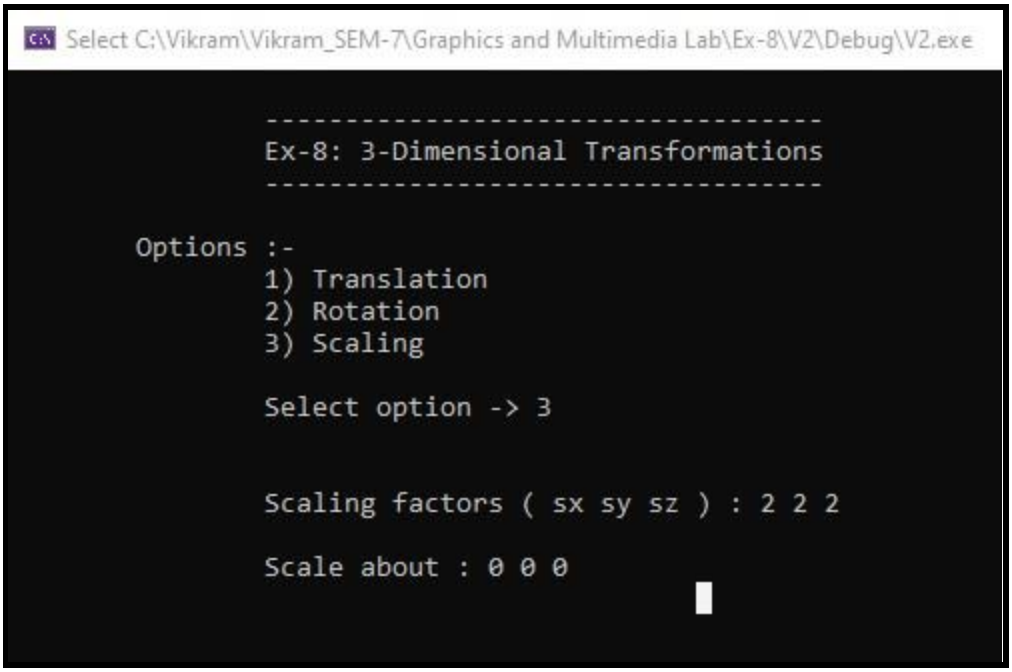
## 1.TRANSLATION :



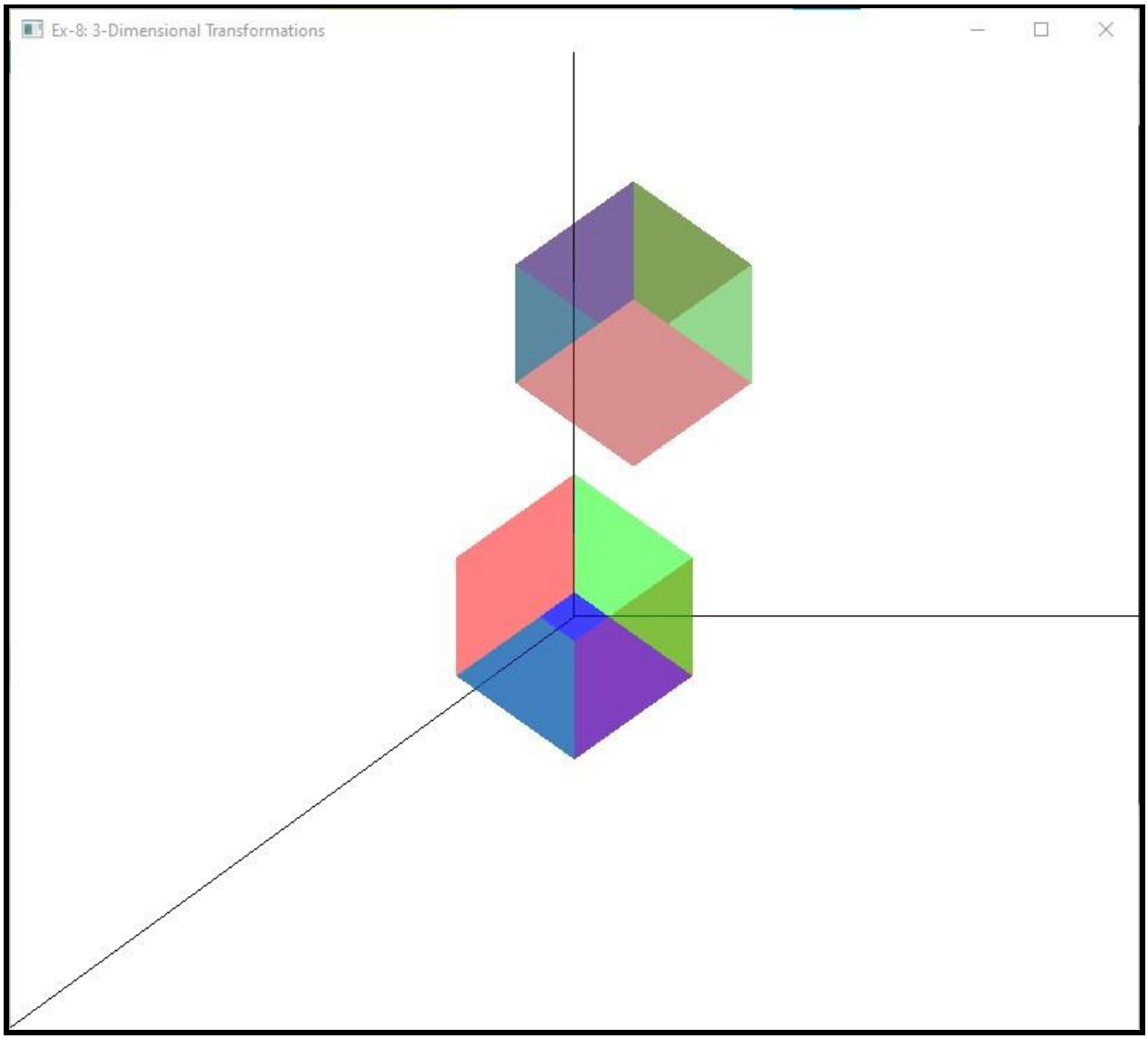

C:\Vikram\Vikram_SEM-7\Graphics and Multimedia Lab\Ex-8\V2\Debug\V2.exe
Ex-8: 3-Dimensional Transformations
Options :-
1) Translation
2) Rotation
3) Scaling
Select option -> 1
Translation factors ( tx ty tz ) : 200 200 200

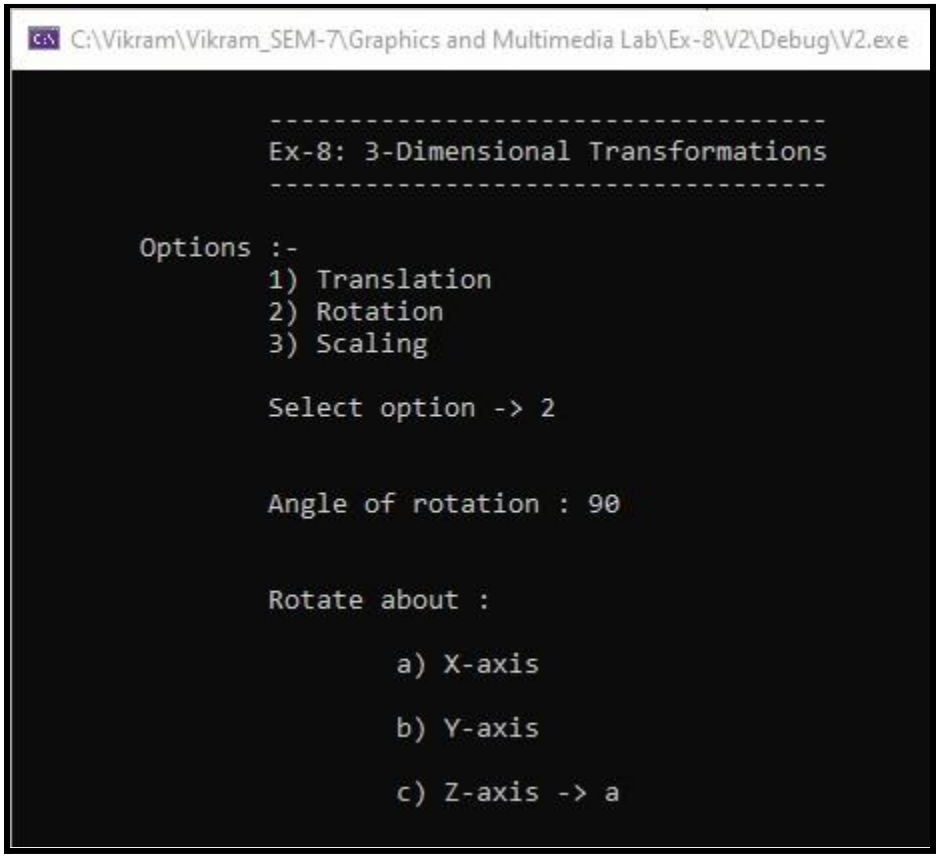
## 2.SCALING :



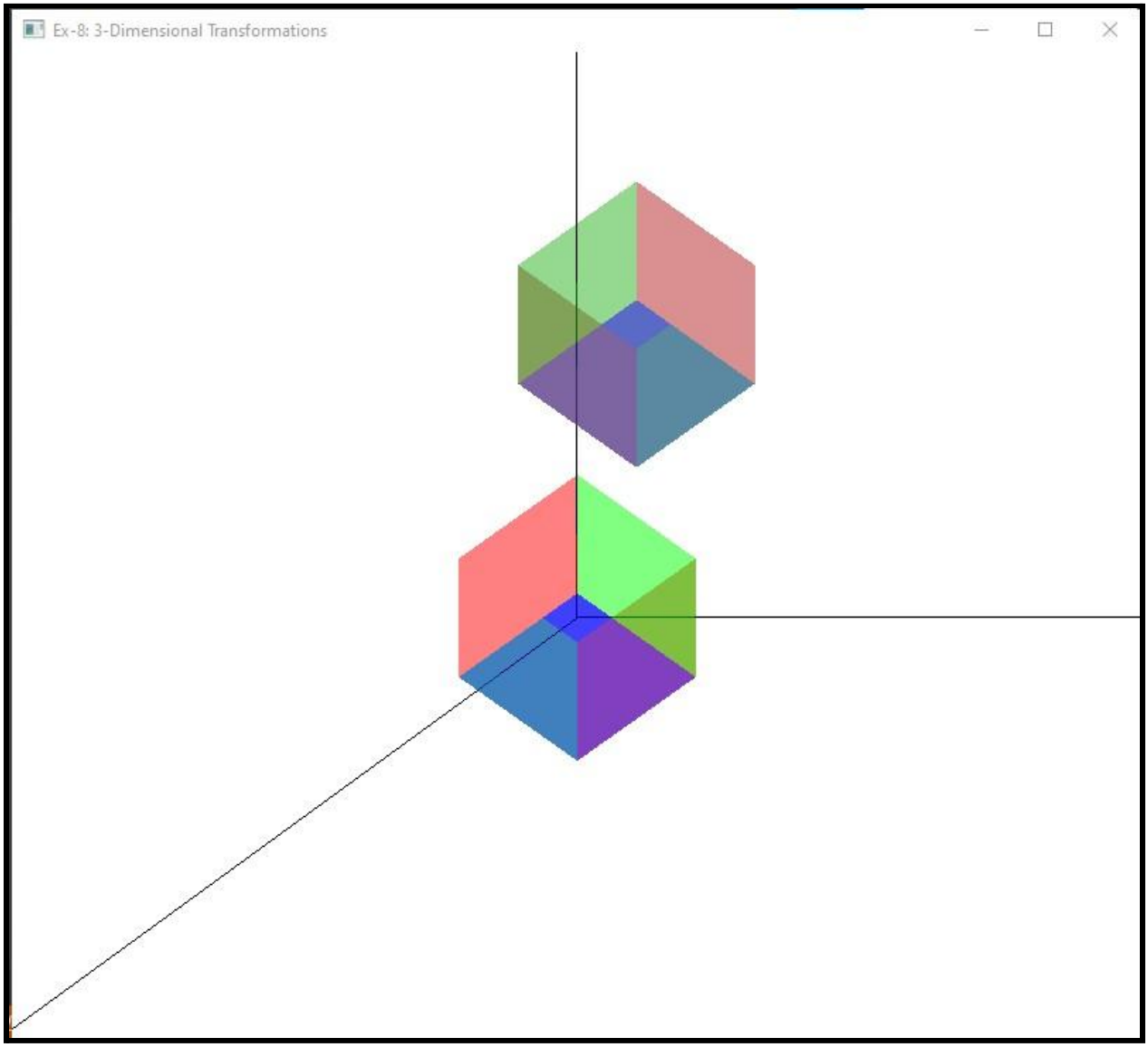

Select C:\Vikram\Vikram_SEM-7\Graphics and Multimedia Lab\Ex-8\V2\Debug\V2.exe
Ex-8: 3-Dimensional Transformations
Options :-
1) Translation
2) Rotation
3) Scaling
Select option -> 3
Scaling factors ( sx sy sz ) : 2 2 2
Scale about : 0 0 0

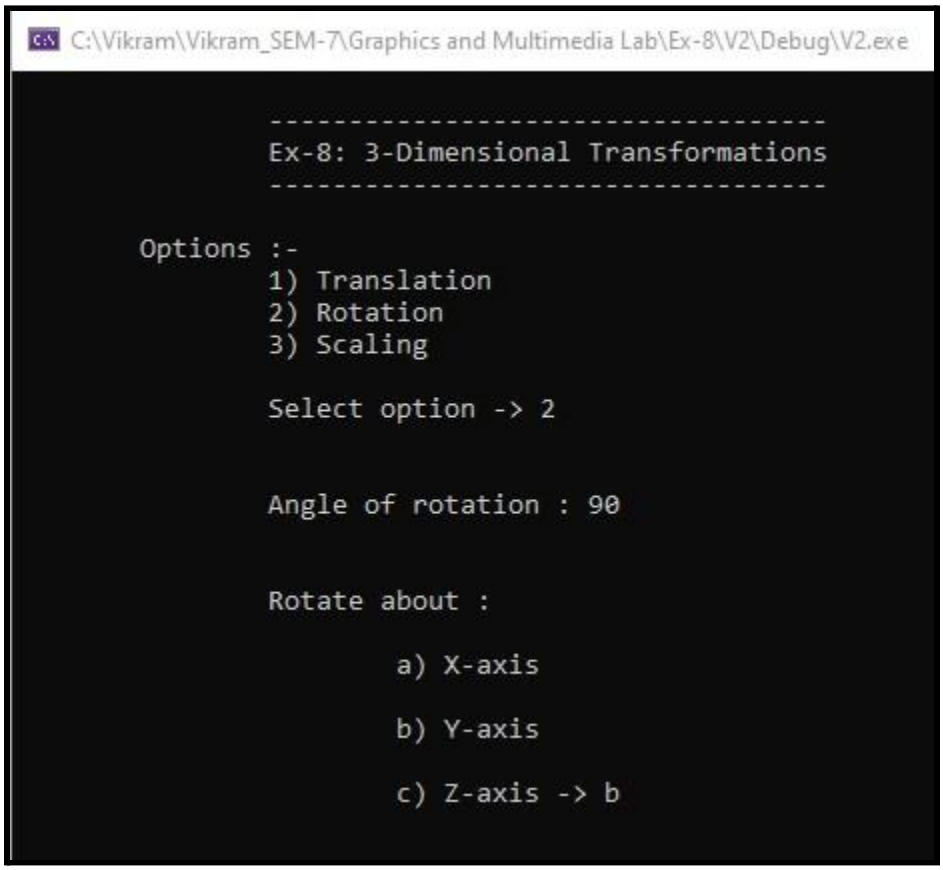
**3 (a) . ROTATION - X : (transformed object has been translated to differentiate from the original object)**



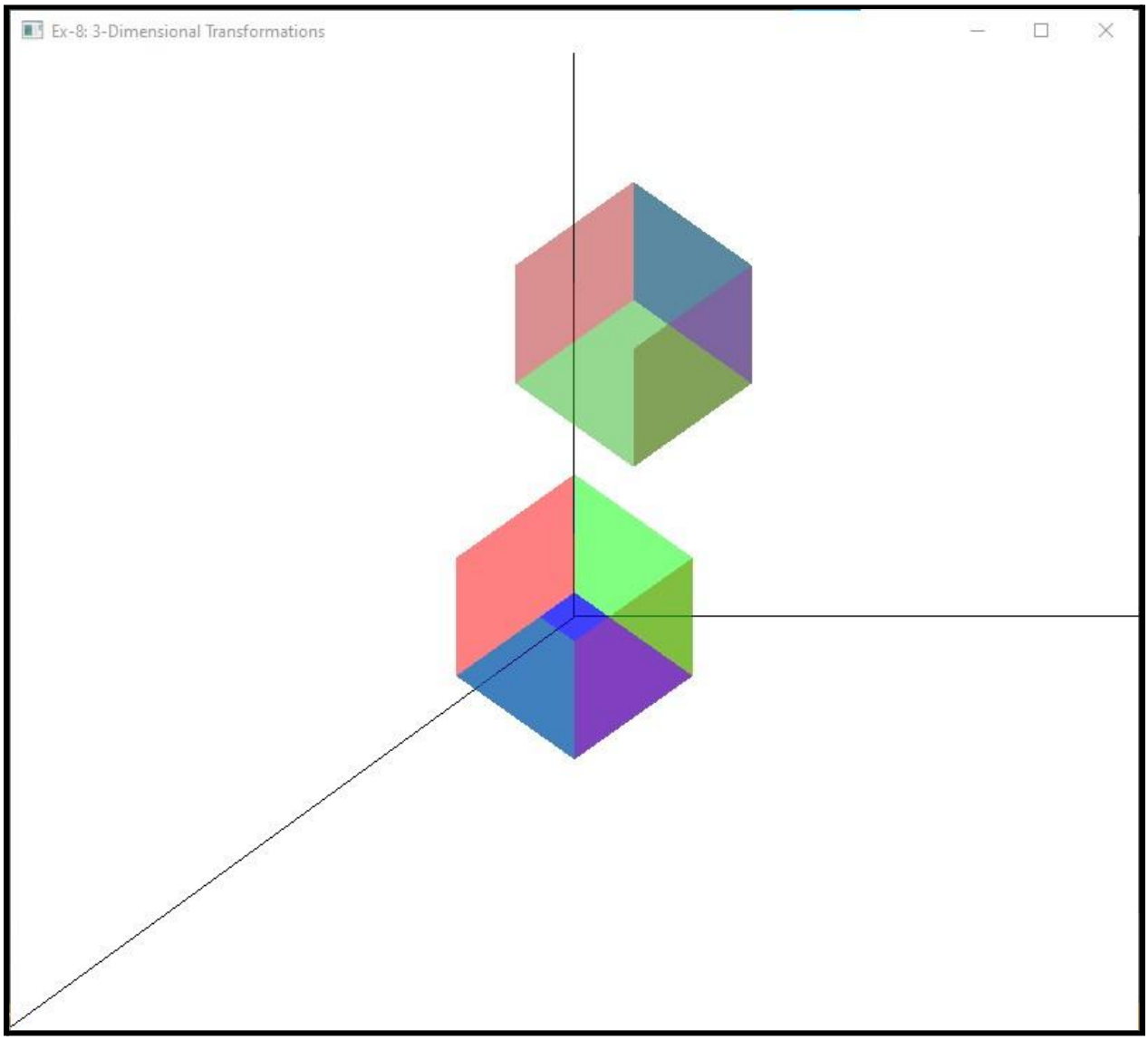

C:\Vikram\Vikram_SEM-7\Graphics and Multimedia Lab\Ex-8\V2\Debug\V2.exe
Ex-8: 3-Dimensional Transformations
Options :-
1) Translation
2) Rotation
3) Scaling
Select option -> 2
Angle of rotation : 90
Rotate about :
a) X-axis
b) Y-axis
c) Z-axis -> a

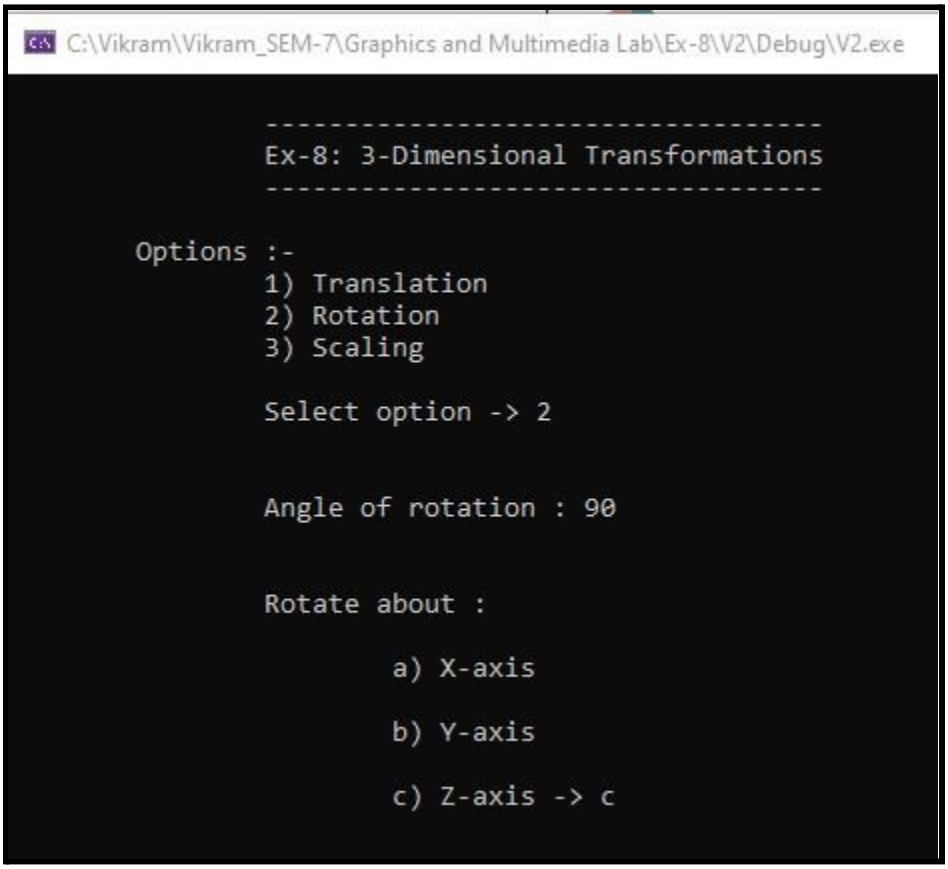
## 3 (b) . ROTATION - Y :




C:\Vikram\Vikram_SEM-7\Graphics and Multimedia Lab\Ex-8\V2\Debug\V2.exe
Ex-8: 3-Dimensional Transformations
Options :-
1) Translation
2) Rotation
3) Scaling
Select option -> 2
Angle of rotation : 90
Rotate about :
a) X-axis
b) Y-axis
c) Z-axis -> b

## 3 (c) . ROTATION - Z :






## CONCLUSION :

Thus the basic 3D Transformations such as Translation , Rotation, Scaling were performed on the 3D Object.