Graphics Assignment No: 4

Shikhar Jaiswal
1601CS44

February 15, 2019

1 Assignment Description

Implement various 2D transformations and apply it to the given polygon.

2 Procedure

Installation

Install the following packages from the Ubuntu repository:
• freeglut3-dev
• mesa-common-dev
• libglm-dev

```
sudo apt-get install freeglut3 freeglut3-dev mesa-common-dev
libglm-dev
```

Check your */usr/include/GL* folder to verify the installation of the OpenGL headers that you intend to use.

Compiling and Linking

We will have to use the $-lglut$ linker option with $gcc/g++$ to compile a program with *glut* library.

For example, to compile the program, use the following to get the binary executable code:

```
g++ modelling_transformation.cpp -lGL -lGLU -lglut -o
modelling_transformation
```

3 Discussion

The primary objective of the assignment is to implement the various ways to transform a polygonal shape in 2D space. For this purpose, the various 2D transformations are implemented. For a detailed explanation, please refer the code.

Polygon Coordinates

```
64 96
224 32
416 192
424 352
224 224
64 320
64 96
```

OpenGL Code

```
#include <iostream>
#include <fstream>
#include <vector>
#include <GL/glut.h>
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include <glm/gtx/matrix_transform_2d.hpp>

using namespace std;

// Global variables for window size, choice of transformation
// and transformation matrices.
int choice;
int max_height = 1200, max_width = 800;
vector<glm::vec4> storage;
glm::mat4 scaler = glm::scale(glm::mat4(1.0f), glm::vec3(0.5f));
glm::mat4 shearerX = glm::shearX(glm::mat3(1.0f), -0.2f);
glm::mat4 shearerY = glm::shearY(glm::mat3(1.0f), -0.2f);
glm::mat4 reflectorX = glm::scale(glm::mat4(1.0f),
        glm::vec3(1.0f, -1.0f, 1.0f));
glm::mat4 reflectorY = glm::scale(glm::mat4(1.0f),
        glm::vec3(-1.0f, 1.0f, 1.0f));
glm::mat4 rotator = glm::transpose(glm::rotate(glm::mat3(1.0f),
        3.14f));
glm::mat4 translator = glm::transpose(glm::translate(
        glm::mat4(1.0f), glm::vec3(-200.0f, -100.0f, 0.0f)));
```

```
// Function to read the file with polygon coordinates
// and store them.
void store_polygon()
{
    ifstream polygon;
    polygon.open("polygon.txt");

    if (not polygon.is_open())
    {
        cerr << "The File Cannot Be Read!" << endl;
        return ;
    }

    float x, y;

    while (polygon >> x >> y)
    {
        storage.push_back(glm::vec4(x, y, 0.0f, 1.0f));
    }

    polygon.close();
}

// Function to print the processed polygon.
void print_processed(vector<glm::vec4> &v)
{
    // Colour fill.
    glColor3ub(255, 255, 255);
    // Set point sizes.
    glPointSize(2.0);

    float x1, y1, x2, y2;

    for (auto it = v.begin(); it != v.end(); it++)
    {
        glm::vec4 coordinate = *it;

        if (it == v.begin())
        {
            x1 = coordinate[0];
            y1 = coordinate[1];
        } else
        {
            x2 = coordinate[0];
            y2 = coordinate[1];
```

```
            glBegin(GL_LINES);
                glVertex2f(x1, y1);
                glVertex2f(x2, y2);
            glEnd();

            glFlush();
            x1 = x2;
            y1 = y2;
        }
    }
}

// Function to print the original polygon.
void original()
{
    print_processed(storage);
}

// Function to scale the original polygon.
void scaling()
{
    vector<glm::vec4> processed;

    for (auto i : storage)
    {
        i = i * scaler;
        processed.push_back(i);
    }

    print_processed(processed);
}

// Function to reflect the original polygon along X axis.
void reflectionX()
{
    vector<glm::vec4> processed;

    for (auto i : storage)
    {
        i = i * reflectorX;
        processed.push_back(i);
    }

    print_processed(processed);
}
```

```
// Function to reflect the original polygon along Y axis.
void reflectionY()
{
    vector<glm::vec4> processed;

    for (auto i : storage)
    {
        i = i * reflectorY;
        processed.push_back(i);
    }

    print_processed(processed);
}

// Function to shear the original polygon along X axis.
void shearX()
{
    vector<glm::vec4> processed;

    for (auto i : storage)
    {
        i = i * shearerX;
        processed.push_back(i);
    }

    print_processed(processed);
}

// Function to shear the original polygon along Y axis.
void shearY()
{
    vector<glm::vec4> processed;

    for (auto i : storage)
    {
        i = i * shearerY;
        processed.push_back(i);
    }

    print_processed(processed);
}

// Function to rotate the original polygon counter-clockwise.
void rotation()
{
```

```
    vector<glm::vec4> processed;

    for (auto i : storage)
    {
        i = i * rotator;
        processed.push_back(i);
    }

    print_processed(processed);
}

// Function to translate the original polygon.
void translation()
{
    vector<glm::vec4> processed;

    for (auto i : storage)
    {
        i = i * translator;
        processed.push_back(i);
    }

    print_processed(processed);
}

// Function to display the drawn and transformed polygon.
void display()
{
    if (choice == 1)
    {
        original();
    } else if (choice == 2)
    {
        scaling();
    } else if (choice == 3)
    {
        reflectionX();
    } else if (choice == 4)
    {
        reflectionY();
    } else if (choice == 5)
    {
        shearX();
    } else if (choice == 6)
    {
        shearY();
```

```
    } else if (choice == 7)
    {
        rotation();
    } else
    {
        translation();
    }
}

int main(int argc, char **argv)
{
    store_polygon();

    cout << "Welcome To 2D Transformer" << endl;
    cout << "1) Original" << endl;
    cout << "2) Scaling" << endl;
    cout << "3) Reflection along X Axis" << endl;
    cout << "4) Reflection along Y Axis" << endl;
    cout << "5) Shear along X Axis" << endl;
    cout << "6) Shear along Y Axis" << endl;
    cout << "7) Rotation" << endl;
    cout << "8) Translation" << endl;
    cout << "Enter Choice: ";
    cin >> choice;

    if (choice < 1 or choice > 8){
        cout << "Wrong Input Entered! Aborting!!" << endl;
        return 0;
    }

    // Initialize to the command-line arguments.
    glutInit(&argc, argv);
    // Setup the colour depth of the window buffers.
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);

    // Assign the position, size and name to the window.
    glutInitWindowPosition(100, 150);
    glutInitWindowSize(max_height, max_width);
    glutCreateWindow("2D Modelling Transformation");

    // Setup a black background.
    glClearColor(0.0, 0.0, 0.0, 0.0);
    // Setup viewing projection.
    glMatrixMode(GL_PROJECTION);
    // Initialize identity matrix.
    glLoadIdentity();
```

```
    // Setup a viewport.
    gluOrtho2D(-600.0, 600.0, -400.0, 400.0);
    // Set the display area colour set using glClearColor().
    glClear(GL_COLOR_BUFFER_BIT);

    // Pass the display function to generate the display.
    glutDisplayFunc(display);
    // Hand over the execution to the glut library.
    glutMainLoop();

    return 0;
}
```

Python Code

```
# Using turtle graphics library.
import turtle

# Set the window sizes.
max_height = 1200
max_width = 800
storage = []

# Function to print the processed polygon.
def print_processed(processed):
    x1 = processed[0][0]
    y1 = processed[0][1]
    for i in processed[1:]:
        x2 = i[0]
        y2 = i[1]
        turtle.penup()
        turtle.pencolor("white")
        turtle.goto(x1, y1)
        turtle.pendown()
        turtle.goto(x2, y2)
        turtle.penup()
        x1 = x2
        y1 = y2

# Function to process the polygon coordinates.
def process_matrix(matrix):
    processed = []
    for i in storage:
        j = [0.0, 0.0, 0.0, 0.0]
        for a in range(len(matrix)):
```

```
            for c in range(4):
                j[a] += matrix[a][c] * i[c]
        processed.append(j)
    print_processed(processed)

# Function to print the original polygon.
def original():
    print_processed(storage)

# Function to scale the original polygon.
def scaling():
    scaler = [[0.5, 0.0, 0.0, 0.0],
              [0.0, 0.5, 0.0, 0.0],
              [0.0, 0.0, 0.5, 0.0],
              [0.0, 0.0, 0.0, 1.0]]
    process_matrix(scaler)

# Function to reflect the original polygon along X axis.
def reflectionX():
    reflectorX = [[1.0, 0.0, 0.0, 0.0],
                  [0.0, -1.0, 0.0, 0.0],
                  [0.0, 0.0, 1.0, 0.0],
                  [0.0, 0.0, 0.0, 1.0]]
    process_matrix(reflectorX)

# Function to reflect the original polygon along Y axis.
def reflectionY():
    reflectorY = [[-1.0, 0.0, 0.0, 0.0],
                  [0.0, 1.0, 0.0, 0.0],
                  [0.0, 0.0, 1.0, 0.0],
                  [0.0, 0.0, 0.0, 1.0]]
    process_matrix(reflectorY)

# Function to shear the original polygon along X axis.
def shearX():
    shearerX = [[1.0, -0.2, 0.0, 0.0],
                [0.0, 1.0, 0.0, 0.0],
                [0.0, 0.0, 1.0, 0.0],
                [0.0, 0.0, 0.0, 1.0]]
    process_matrix(shearerX)

# Function to shear the original polygon along Y axis.
def shearY():
    shearerY = [[1.0, 0.0, 0.0, 0.0],
                [-0.2, 1.0, 0.0, 0.0],
                [0.0, 0.0, 1.0, 0.0],
```

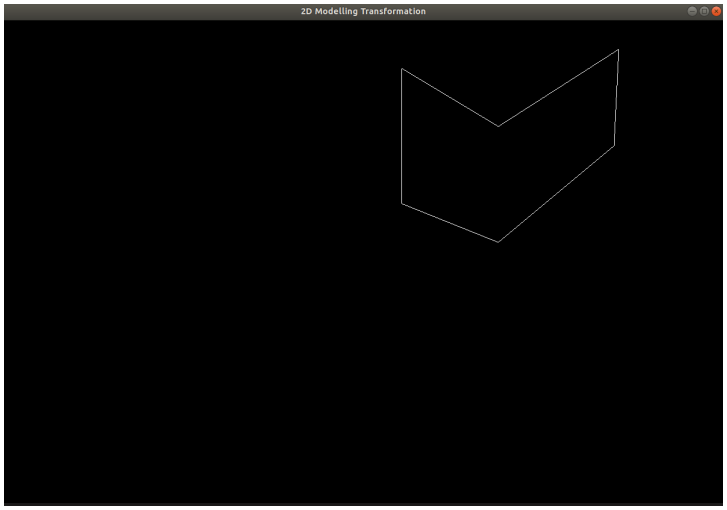
```
                [0.0, 0.0, 0.0, 1.0]]
    process_matrix(shearerY)

# Function to rotate the original polygon counter-clockwise.
def rotation():
    rotator = [[-1.0, 0.0, 0.0, 0.0],
               [0.0, -1.0, 0.0, 0.0],
               [0.0, 0.0, 1.0, 0.0],
               [0.0, 0.0, 0.0, 1.0]]
    process_matrix(rotator)

# Function to translate the original polygon.
def translation():
    translator = [[1.0, 0.0, 0.0, -200.0],
                  [0.0, 1.0, 0.0, -100.0],
                  [0.0, 0.0, 1.0, 0.0],
                  [0.0, 0.0, 0.0, 1.0]]
    process_matrix(translator)

# Function to read the file with polygon coordinates
# and storing them.
def draw_polygon(choice):
    with open('polygon.txt') as f:
        for line in f:
            x1, y1 = [float(x) for x in line.split()]
            storage.append([x1, y1, 0.0, 1.0])

    if choice == 1:
        original()
    elif choice == 2:
        scaling()
    elif choice == 3:
        reflectionX()
    elif choice == 4:
        reflectionY()
    elif choice == 5:
        shearX()
    elif choice == 6:
        shearY()
    elif choice == 7:
        rotation()
    else:
        translation()

# Function to display the drawn and filled polygon.
def display():
```

```
    print("1) Original")
    print("2) Scaling")
    print("3) Reflection along X Axis")
    print("4) Reflection along Y Axis")
    print("5) Shear along X Axis")
    print("6) Shear along Y Axis")
    print("7) Rotation")
    print("8) Translation")
    choice = int(input("Enter Choice: "))
    if (choice < 1 or choice > 8):
        print("Incorrect Input! Aborting!!")
        return
    draw_polygon(choice)

# Initial input.
print("2D Modelling Transformation\n")

# Initialization and background colour.
turtle.setup(max_height, max_width)
turtle.bgcolor("black")

# Set the fill colour to black.
turtle.fillcolor("black")
# Initiate the algorithm.
display()

# Exit on click.
turtle.exitonclick()
```

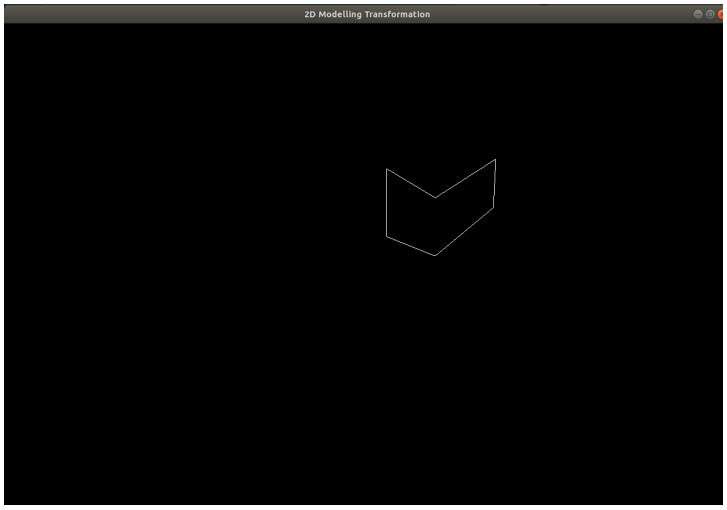
4 Result

OpenGL

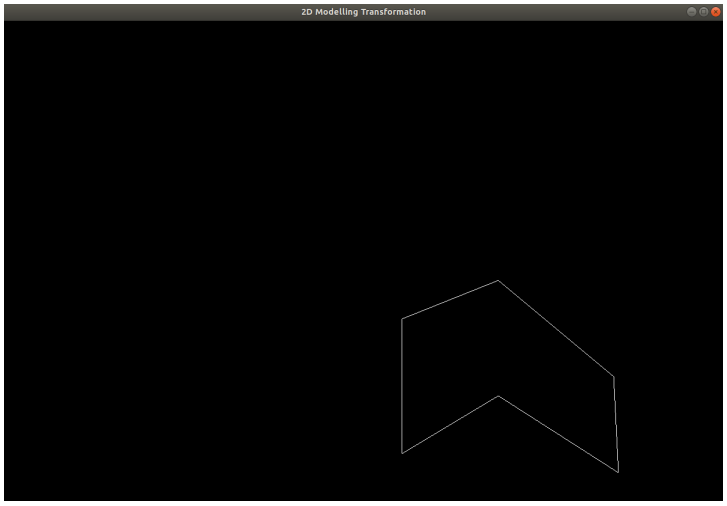
Original



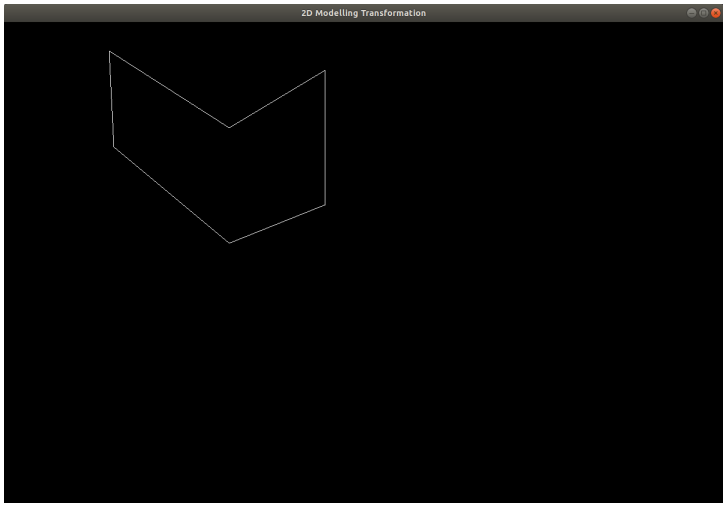
Scaling



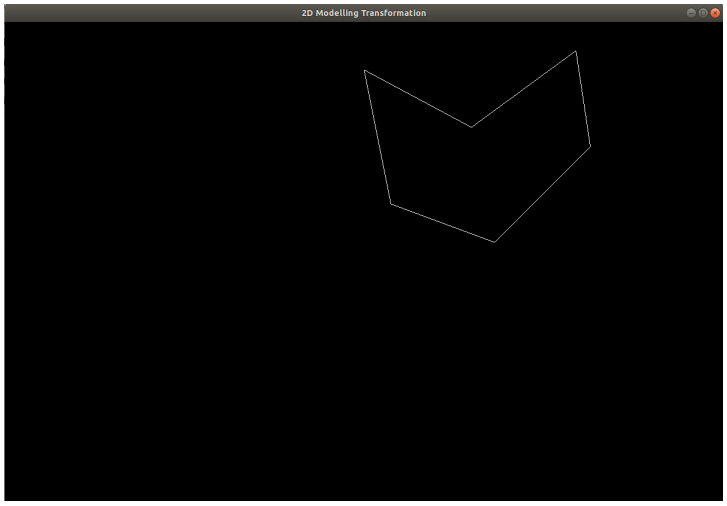
Reflection Around X Axis



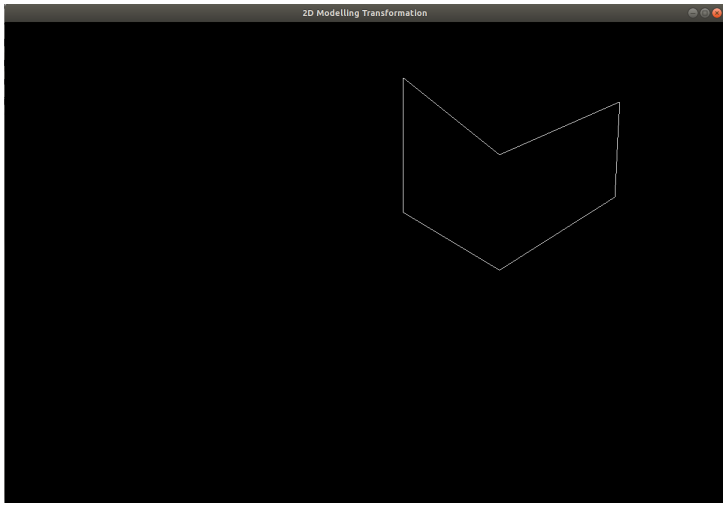
Reflection Around Y Axis



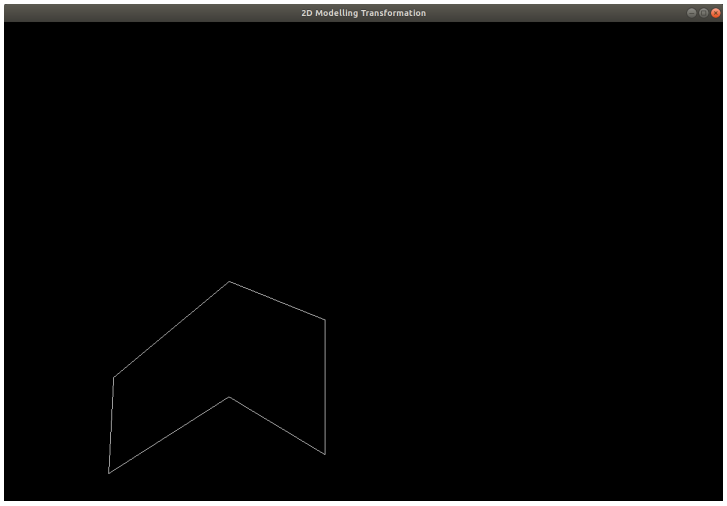
Shear Around X Axis



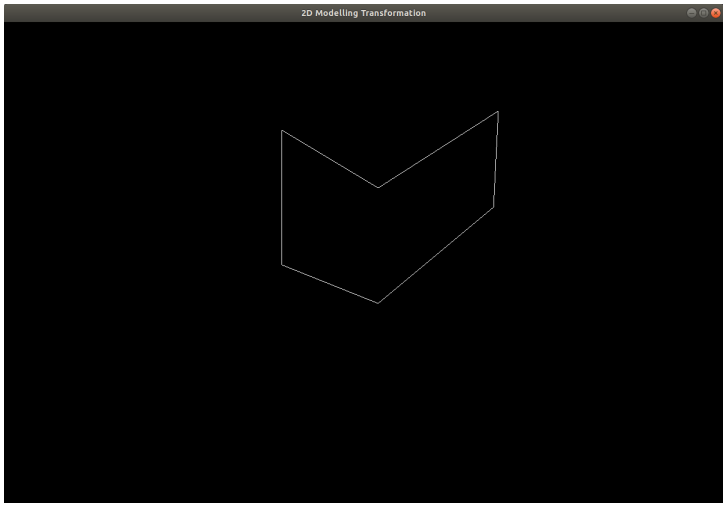
Shear Around Y Axis



Rotation

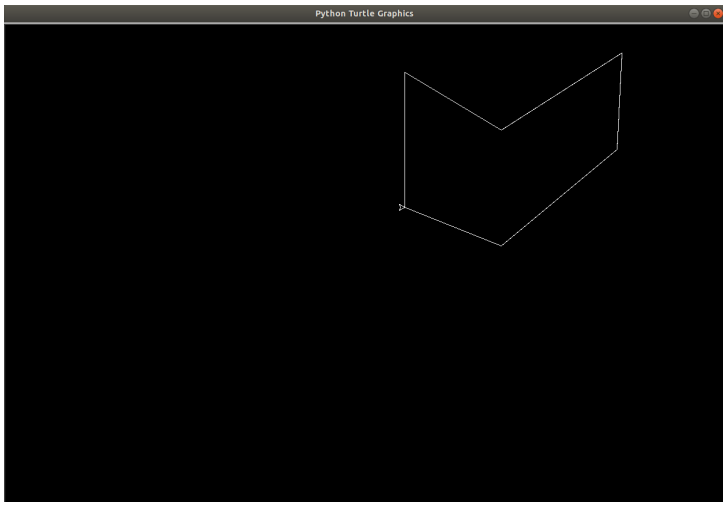


Translation

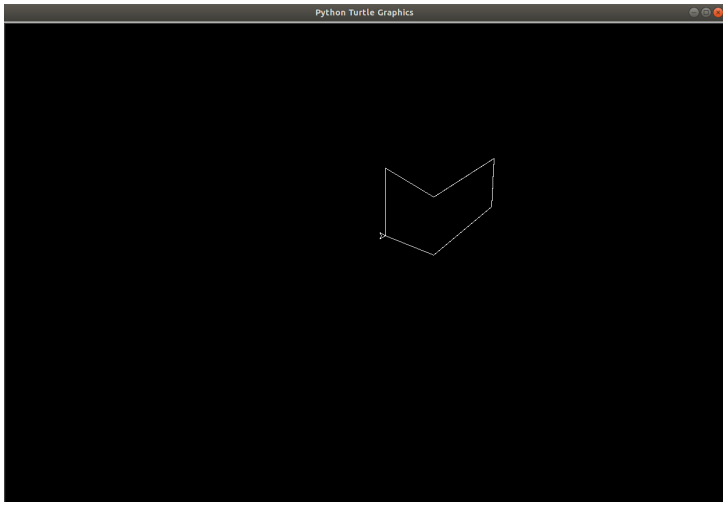


Python

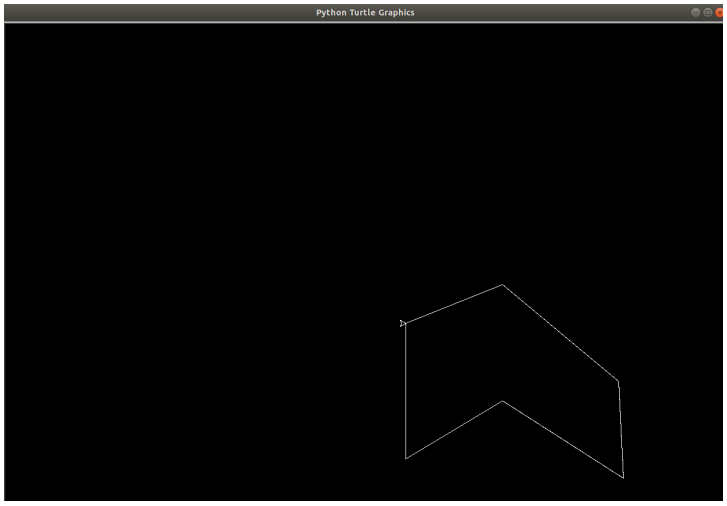
Original



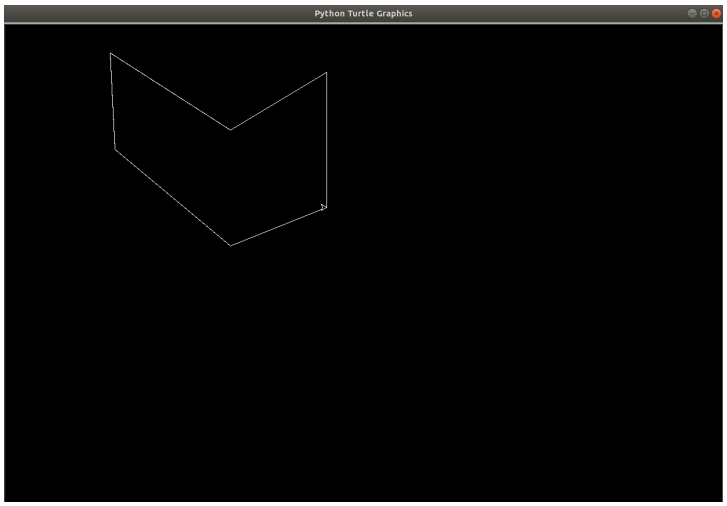
Scaling



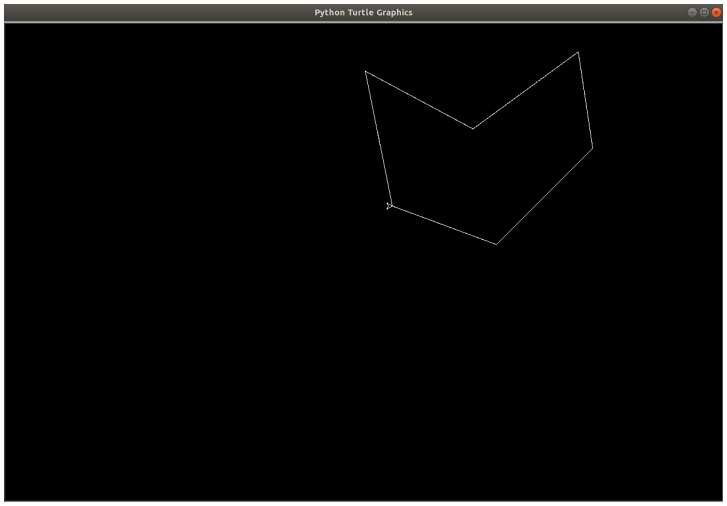
Reflection Around X Axis



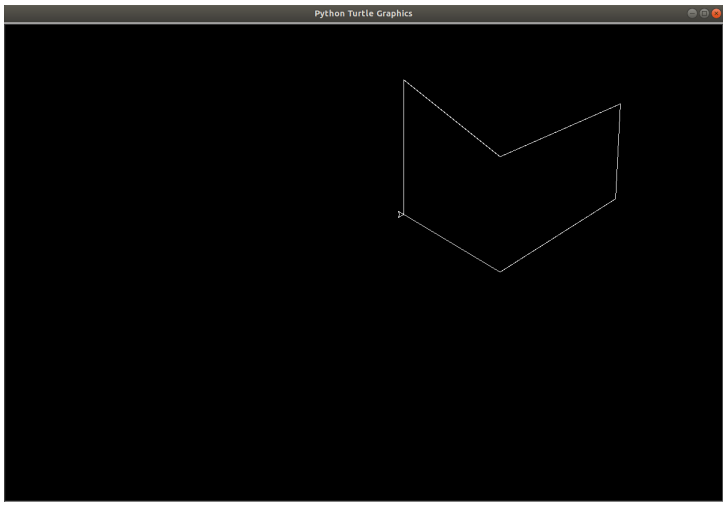
Reflection Around Y Axis



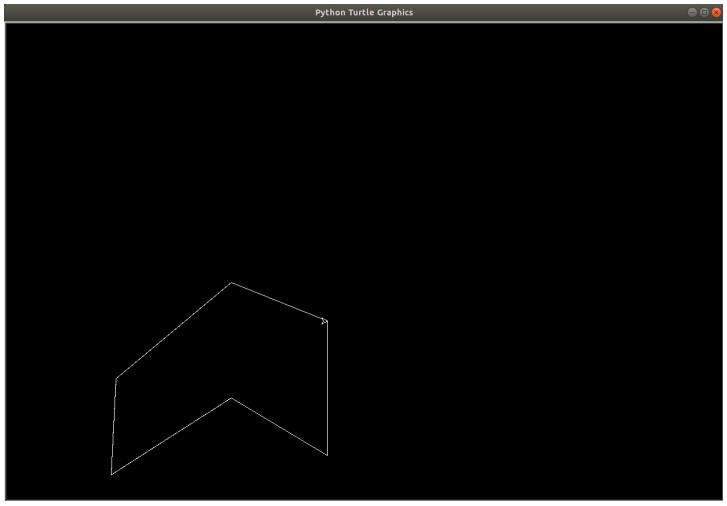
Shear Around X Axis



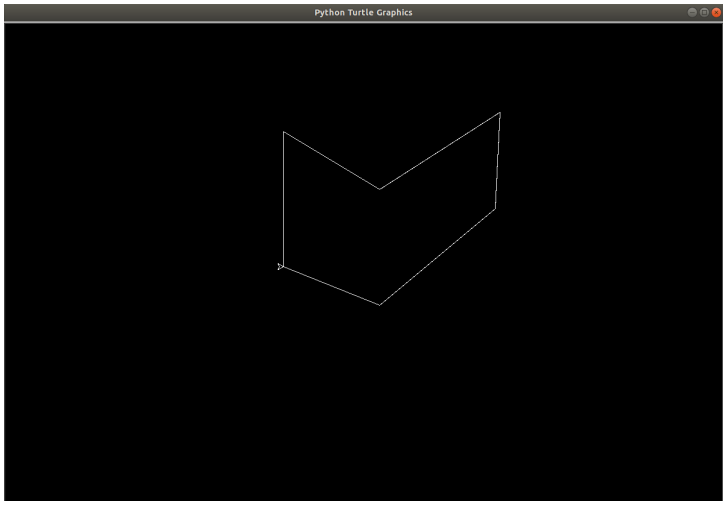
Shear Around Y Axis



Rotation



Translation



5 References

[1] How to install OpenGL/GLUT libraries

[2] An Introduction to OpenGL Programming

[3] Turtle Graphics - The Python Standard Library

[4] OpenGL Mathematics