Kathmandu University

Department of Computer Science and Engineering

Dhulikhel, Kavre

Lab Report #2

Computer Graphics

[Course Code: COMP 342]

[For the partial fulfillment of 3rd year/2nd Semester in Computer Engineering]

Submitted by:

Sabin Thapa

Roll no. 54

Submitted to:

Mr. Dhiraj Shrestha

Assistant Professor

Department of Computer Science and Engineering

Submission date: April 21, 2022

1. Midpoint Ellipse Drawing Algorithm

In Computer Graphics, the midpoint ellipse drawing algorithm is used to draw an ellipse. It plots points on an ellipse on the first quadrant by dividing the quadrant into two regions. Each pont (x, y) is then projected onto the remaining three quadrants using the four-point symmetry of an ellipse: (-x, y), (x, -y) and (-x,-y).

Algorithm

1. Input r_x, r_y and the center of the ellipse (x_c, y_c) and obtain the first point an an ellipse centered on the origin as:

$$(x_0, y_0) = (0, r_y)$$

2. Calculate the initial value of the decision parameter in region 1 as:

$$p1_0 = r^2_y - r^2_x r_y + \frac{1}{4} r^2_x$$

3. At each x_k position in region 1, starting at k - 0, perform the following test: if $p1_k < 0$, the next point along the ellipse centered in (0,0) is (x_{k+1}, y_k) and

$$p1_{k=1} = p1_k + 2r^2_y x_{k+1} + r^2_y$$

Otherwise, the next point along the circle is $(x_k + 1, y_k - 1)$ and

$$p1_{k=1} = p1_k + 2r^2_y x_{k+1} + r^2_y - 2r^2_x y_{k+1}$$

With

$$2r^2_y x_{k+1} = 2r^2_y x_k + 2r^2_y, \ 2r^2_x y_{k+1} = 2r^2_x y_k - 2r^2_x$$

And continue until $2r^2_y x >= 2r^2_x y$.

4. Calculate the initial value of the decision parameter in region 2 using the last point (x_o, y_0) calculated in the region 1 as:

$$p2_0 = r^2_y(x_0 + \frac{1}{2})^2 + r^2_x(y_0 - 1)^2 - r^2_x r^2_y$$

5. At each y_k position in region 2, starting at k=0, perform the following test: if $p2_k > 0$, the next point along the ellipse centered on (0,0) is $(x_k, y_k - 1)$ and

$$p2_{k+1} = p2_k - 2r^2_x y_{k+1} + r^2_x$$

Otherwise, the next point along the circle is $(x_k + 1, y_k - 1)$ and

$$p2_{k+1} = p2_k - 2r^2_x y_{k+1} + r^2_x + 2r^2_y x_{k+1}$$

Using the same incremental calculations for x and y as in region 1.

6. Determine symmetry points in the other three quadrants.

7. Move each calculated pixel position (x,y) onto the elliptical path centered on (x_c, y_c) and plot the coordinate values:

$$x = x + x_c,\ y = y + y_c$$

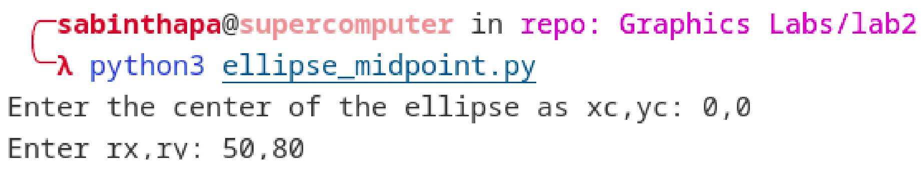
8. Repeat the steps for region 1 until $2r^2_y x >= 2r^2_x y$.

Source Code

```
from OpenGL.GL import *
from OpenGL.GLUT import *
from OpenGL.GLU import *

def initialize():
   glutInit(sys.argv)
   glutInitDisplayMode(GLUT_RGB)
   glutInitWindowSize(600,600)
   glutInitWindowPosition(0,0)
   glutCreateWindow("Mid-point Ellipse Drawing Algorithm")

   glClearColor(1.0,1.0,1.0,0.0)
   gluOrtho2D(-200,200,-200,200)

   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
   # axes
   glColor3f(0.0,0.0,1.0)
   glBegin(GL_LINES)
   glVertex2f(-200,0)
   glVertex2f(200,0)
   glVertex2f(0,200)
   glVertex2f(0,-200)
   glEnd()

'''Display each point '''
def display_point(x,y):
   glBegin(GL_POINTS)
   glVertex2f(x,y)
   glEnd()

'''Stopping Criteria'''
def stopping_criteria(x,y,rx,ry):
   return (ry*ry*x > rx*rx*y)

def translate_point(x,y,xc,yc):
   display_point(x+ xc, y+yc)
```

```python
def calc_symmetric_points(x,y,xc,yc):
   translate_point(x,y,xc,yc)
   translate_point(-x,y,xc,yc)
   translate_point(x,-y,xc,yc)
   translate_point(-x,-y,xc,yc)

def ellipse_algo(xc, yc, rx, ry):
   x =0
   y = ry
   pk= ry**2 - rx**2 *ry + (1/4)* rx**2 #decision param
   glColor3f(0.0,1.0,1.0)
   glPointSize(5.0)
   calc_symmetric_points(x,y,xc,yc)

   while( not stopping_criteria(x,y,rx,ry)):
       x+=1
       if(pk<=0):
           calc_symmetric_points(x,y,xc,yc)
           pk=pk + 2* ry**2 *x + ry**2
       else:
           y-=1
           calc_symmetric_points(x,y,xc,yc)
           pk=pk + 2* ry**2 * x + ry**2 - 2* rx**2*y
   x = 0
   y = rx
   while(not stopping_criteria(x,y,ry,rx)):
       x +=1
       if(pk<0):
           calc_symmetric_points(y,x,xc,yc)
           pk=pk + 2*rx**2*x +rx**2
       else:
           y-=1
           calc_symmetric_points(y,x,xc,yc)
           pk=pk + 2*rx**2*x + rx**2- 2*ry**2*y
   glFlush()

if __name__ == '__main__':
   center = input("Enter the center of the ellipse as xc,yc: ").split(',')
   xc, yc = int(center[0]), int(center[1])
   radius= input("Enter rx,ry: ").split(',')
   rx, ry = int(radius[0]), int(radius[1])

   initialize()
   glutDisplayFunc(lambda: ellipse_algo(xc,yc,rx,ry))
   glutIdleFunc(lambda: ellipse_algo(xc,yc,rx,ry))

   glutMainLoop()
```

Outputs



```
╭─sabinthapa@supercomputer in repo: Graphics Labs/lab2
╰─λ python3 ellipse_midpoint.py
Enter the center of the ellipse as xc,yc: 0,0
Enter rx,ry: 50,80
```

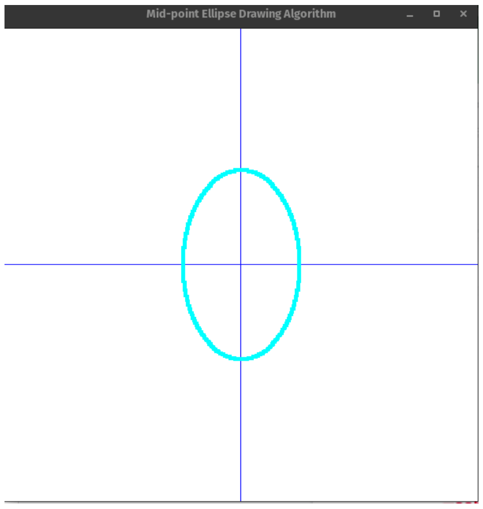
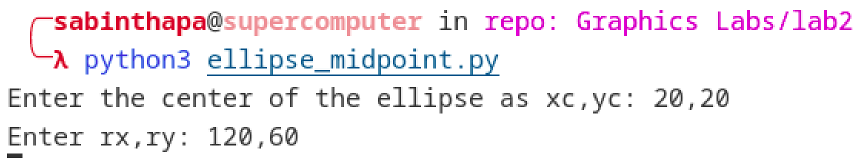


Fig: An ellipse centered at origin



```
╭─sabinthapa@supercomputer in repo: Graphics Labs/lab2
╰─λ python3 ellipse_midpoint.py
Enter the center of the ellipse as xc,yc: 20,20
Enter rx,ry: 120,60
```

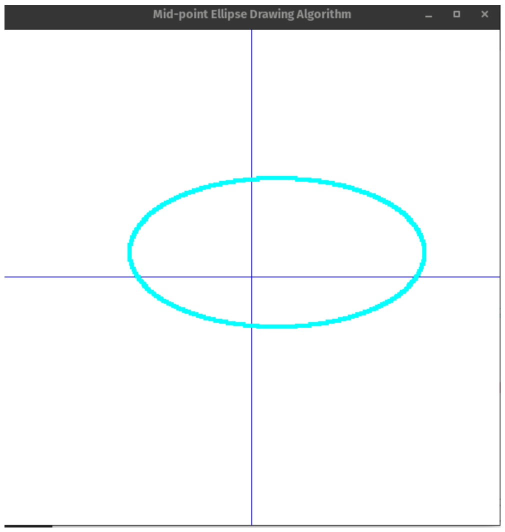


Fig: Ellipse centered at (20,20)

2. Transformation using homogeneous coordinates.

Inorder to transform any 2D object, we take its coordinate matrix and multiply it with the transformation matrix to obtain the resultant coordinate matrix. The resultant coordinate matrix is then plotted. The various transformations and their homogeneous transformation equations are discussed below:

1. **Translation**

$$\begin{bmatrix} X_{new} \\ Y_{new} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X_{old} \\ Y_{old} \\ 1 \end{bmatrix}$$

Translation Matrix

(Homogeneous Coordinates Representation)

2. **Rotation**

$$\begin{bmatrix} X_{new} \\ Y_{new} \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X_{old} \\ Y_{old} \\ 1 \end{bmatrix}$$

Rotation Matrix

(Homogeneous Coordinates Representation)

3. **Scaling**

$$\begin{bmatrix} X_{new} \\ Y_{new} \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X_{old} \\ Y_{old} \\ 1 \end{bmatrix}$$

Scaling Matrix

(Homogeneous Coordinates Representation)

4. **Reflection**

$$\begin{bmatrix} X_{new} \\ Y_{new} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X_{old} \\ Y_{old} \\ 1 \end{bmatrix}$$

Reflection Matrix

(Reflection Along X Axis)

(Homogeneous Coordinates Representation)

5. **Shearing**

$$\begin{bmatrix} X_{new} \\ Y_{new} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ Sh_x & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X_{old} \\ Y_{old} \\ 1 \end{bmatrix}$$

Shearing Matrix

(In X axis)

(Homogeneous Coordinates Representation)

Source Code

```
from OpenGL.GL import *

from OpenGL.GLUT import *

from OpenGL.GLU import *

import random

import math
```

```
def initialization():
   glutInit()
   glutInitDisplayMode(GLUT_RGBA)
   glutInitWindowSize(600,600)
   glutInitWindowPosition(300,300)
   glutCreateWindow("Transformations - 2D Square")
   glClearColor(1.0,1.0,1.0,0.0)
   gluOrtho2D(-300,300,-300,300)
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
   # axes
   glColor3f(0.0,0.0,1.0)
   glBegin(GL_LINES)
   glVertex2f(-300,0)
   glVertex2f(300,0)
   glVertex2f(0,300)
   glVertex2f(0,-300)
   glEnd()
vertices = [[10,10,110, 110],
            [10,110,110,10],
            [1,1,1,1]]

image = vertices

class Transformation:
   def __init__(self, vertices, image):
       self.vertices = vertices
```

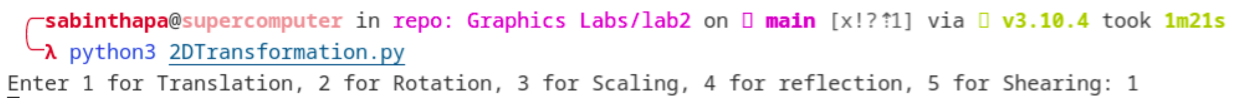
```
        self.image = image

    ''' Draw square'''

    def draw_shape(self):

        glBegin(GL_POLYGON)

        for i in range(len(self.image[0])):

            glVertex2f(self.image[0][i], self.image[1][i])

        glEnd()

    ''' Matrix multiplication '''

    def matrix_multiply(self, transformer):

            mult  =  [[sum(a*b  for  a,b  in  zip(X_row,Y_col))  for  Y_col  in
zip(*self.vertices)] for X_row in transformer]

        self.image = mult

        return

    '''Rotation'''

    def Rotate(self,angle):

        theta = angle*math.pi/180

        transformer= [[math.cos(theta), -math.sin(theta), 0],

                      [math.sin(theta), math.cos(theta), 0],

                      [0,0,1]]

        self.matrix_multiply(transformer)

        self.draw_shape()

    '''Translation'''

    def Translate(self,tx,ty):

        transformer= [[1,0,tx],

                      [0,1,ty],

                      [0,0,1]]
        self.matrix_multiply(transformer)

        self.draw_shape()

    '''Scaling'''

    def Scale(self,sx,sy):

        transformer= [[sx,0,0],

                      [0,sy,0],

                      [0,0,1]]

        self.matrix_multiply(transformer)

        self.draw_shape()

    '''Reflection X-axis'''

    def ReflectX(self):

        transformer= [[1,0,0],

                      [0,-1,0],

                      [0,0,1]]

        self.matrix_multiply(transformer)

        self.draw_shape()



    '''Reflection Y-axis'''

    def ReflectY(self):

        transformer= [[-1,0,0],

                      [0,1,0],

                      [0,0,1]]

        self.matrix_multiply(transformer)

        self.draw_shape()

    '''Shearing'''
```

```
    def Shear(self,shx, shy):

        transformer= [[1,shx,0],

                      [shy,1,0],

                      [0,0,1]]

        self.matrix_multiply(transformer)

        self.draw_shape()

def Transformations():

    glColor3f(0, 0, 1)

    t.draw_shape() #draws original square


    glColor3f(0.5, 0.5 , 1)


    if userInput == 1:

        #Translation by -100 units on x-axis

        t.Translate(-100, 0)

      elif userInput == 2:

        #Rotation by 30 degrees anticlockwise

        t.Rotate(30)


    elif userInput == 3:

        #Scaling by 2 units

        t.Scale(2,2)


    elif userInput == 4:

        t.ReflectY()
    elif userInput == 5:

        # x axis shear by 0,7

        t.Shear(0.7,0)


    glFlush()

    glutSwapBuffers();


if __name__ == "__main__":

    t = Transformation(vertices, image)

    global userInput

     userInput = int(input("Enter 1 for Translation, 2 for Rotation, 3 for
Scaling, 4 for reflection, 5 for Shearing: "))

    initialization()

    glutDisplayFunc(Transformations)

    glutMainLoop()
```

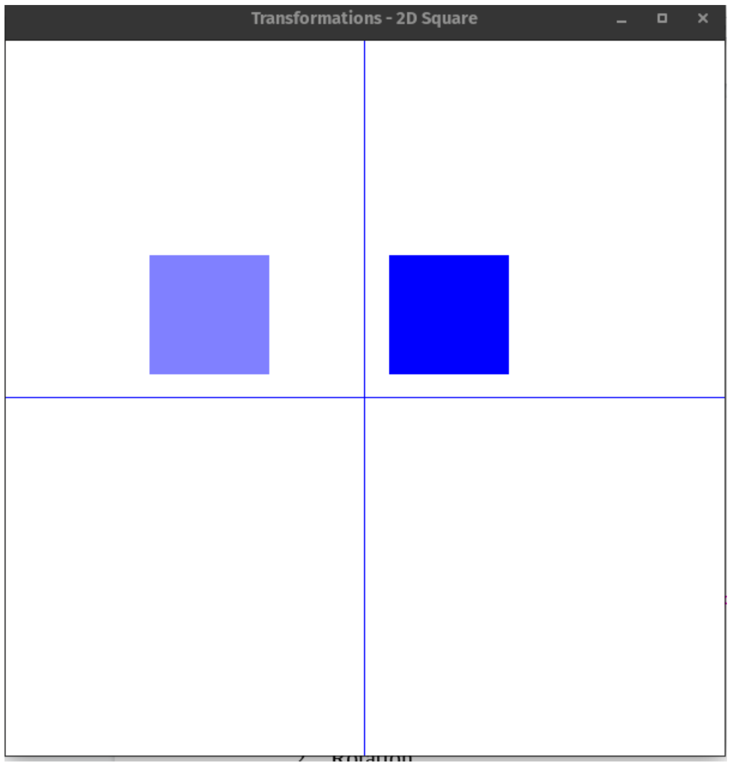
Outputs

Here, the dark colored square is the original image and the lighter one is the result of transformation in all the cases.

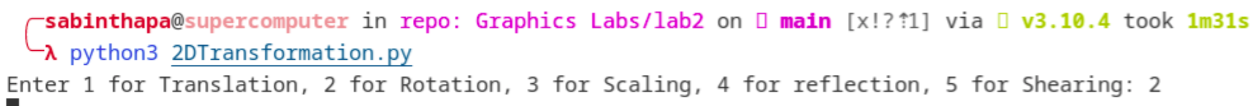
1. Translation



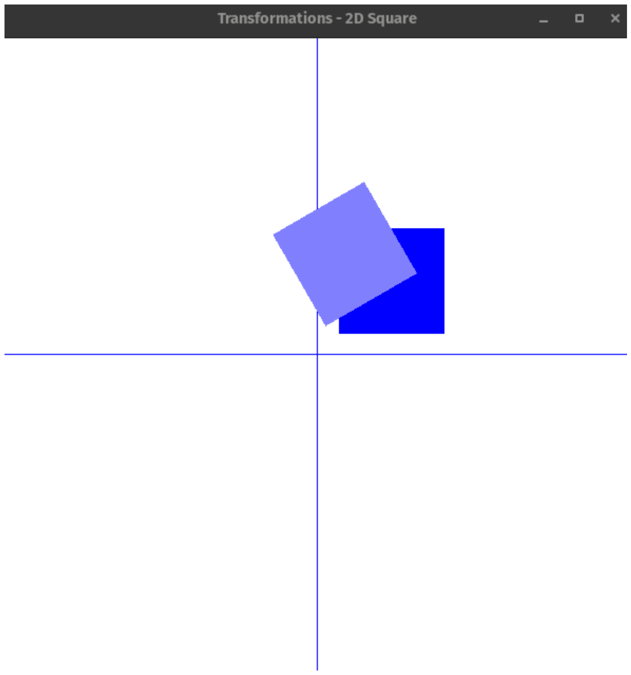
```
╭─sabinthapa@supercomputer in repo: Graphics Labs/lab2 on  main [x!?⇡1] via  v3.10.4 took 1m21s
╰─λ python3 2DTransformation.py
Enter 1 for Translation, 2 for Rotation, 3 for Scaling, 4 for reflection, 5 for Shearing: 1
```



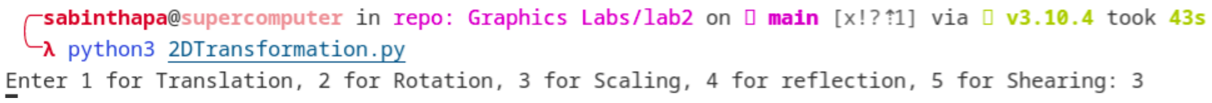
2. Rotation



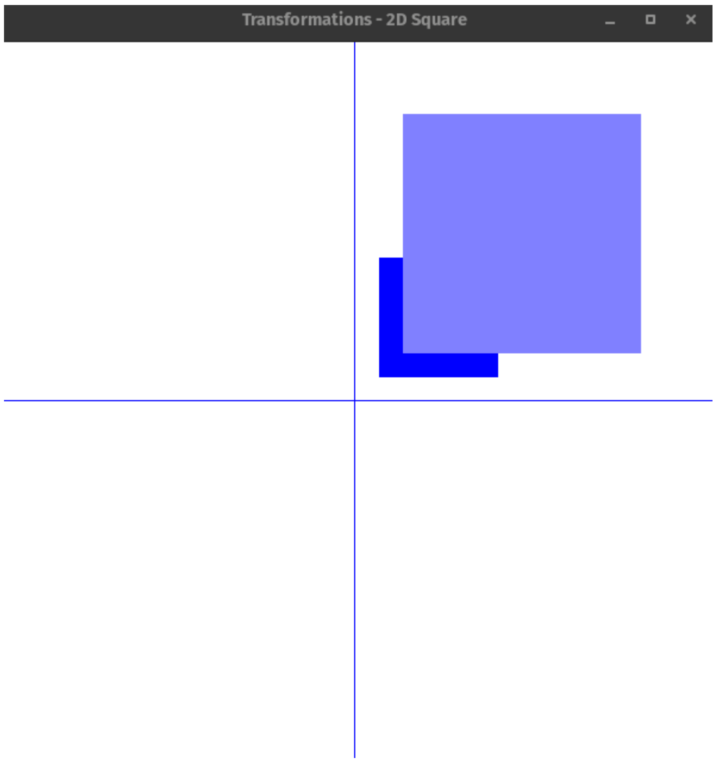
```
╭─sabinthapa@supercomputer in repo: Graphics Labs/lab2 on  main [x!?⇡1] via  v3.10.4 took 1m31s
╰─λ python3 2DTransformation.py
Enter 1 for Translation, 2 for Rotation, 3 for Scaling, 4 for reflection, 5 for Shearing: 2
▬
```



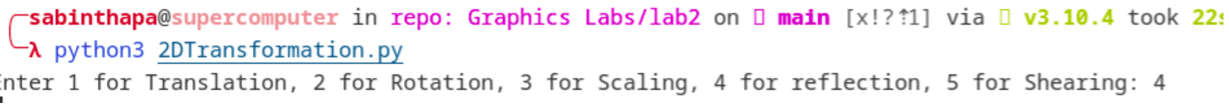
3. Scaling



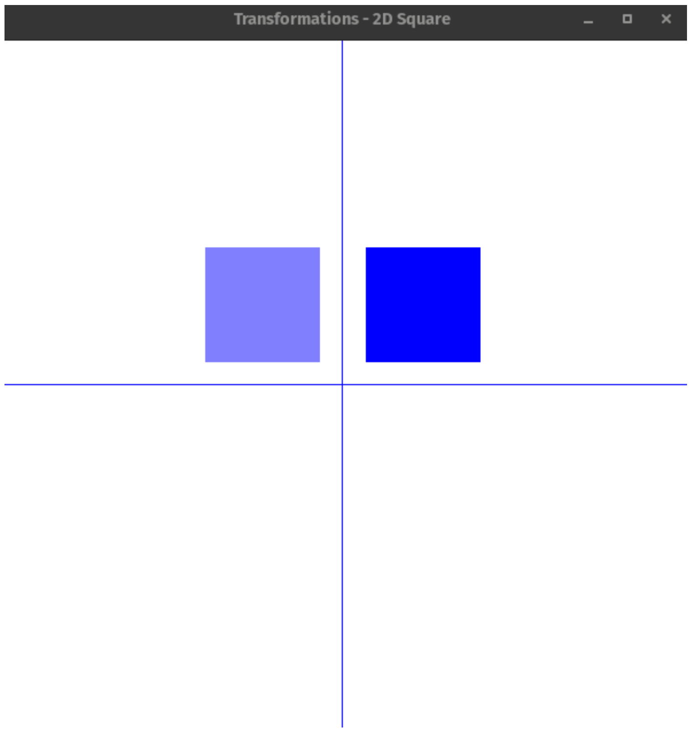
```
╭─sabinthapa@supercomputer in repo: Graphics Labs/lab2 on  main [x!?⇡1] via  v3.10.4 took 43s
╰─λ python3 2DTransformation.py
Enter 1 for Translation, 2 for Rotation, 3 for Scaling, 4 for reflection, 5 for Shearing: 3
```



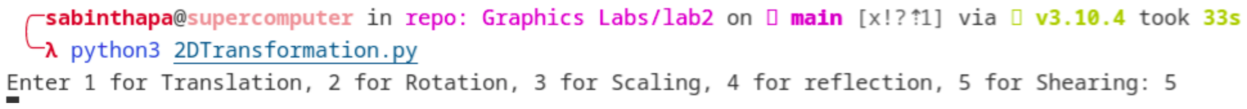
4. Reflection



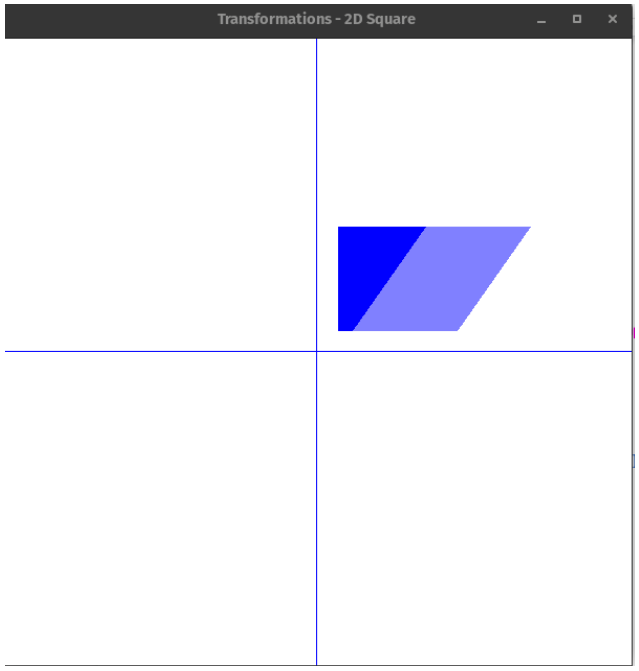
```
╭─sabinthapa@supercomputer in repo: Graphics Labs/lab2 on  main [x!?⇡1] via  v3.10.4 took 22
╰─λ python3 2DTransformation.py
Enter 1 for Translation, 2 for Rotation, 3 for Scaling, 4 for reflection, 5 for Shearing: 4
```



5. Shearing

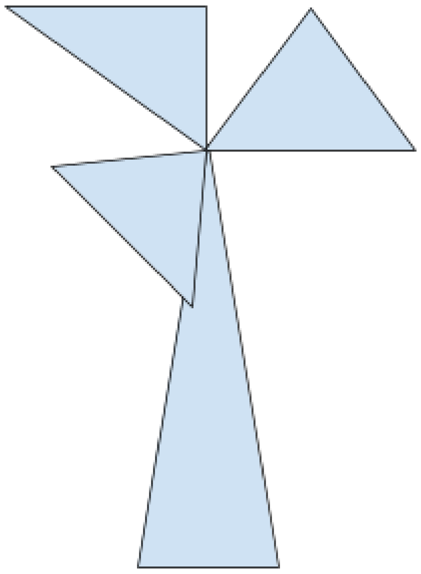


```
╭─sabinthapa@supercomputer in repo: Graphics Labs/lab2 on  main [x!?⇡1] via  v3.10.4 took 33s
╰─λ python3 2DTransformation.py
Enter 1 for Translation, 2 for Rotation, 3 for Scaling, 4 for reflection, 5 for Shearing: 5
```



3. Windmill Simulation using rotation transformation.

A windmill is a building or a structure with large blades on the outside, that when turned by the force of the wind, generate power. The structure of a typical windmill is shown below:



To achieve this simulation, we'll use the **Transformation** class defined above in 2. From that class, the **Rotation** method will be used to rotate the wings of the windmill. Also, some keyboard inputs will be added to control the speed of rotation: 'f' keypress will increase the speed while 's' keypress will decrease the speed of rotation.

Source Code

```
from _2DTransformation import Transformation

from OpenGL.GL import *

from OpenGL.GLUT import *

from OpenGL.GLU import *


#fan vertices

vertices = [[0,250,250],

               [0,0, 130],

               [1,1,1]]

image = vertices

rotation = 0

speed = 2


def initialization():

   glutInit()

   glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE)

   glutInitWindowSize(1000,1000)

   glutInitWindowPosition(0,0)

   glutCreateWindow("Wind Mill Simulation")


   glClearColor(1.0,1.0,1.0,0.0)

   gluOrtho2D(-500,500,-600,400)

   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
```

```
def timer(x):
   global rotation
   rotation += speed
   glutPostRedisplay()
   glutTimerFunc(round(1000/60), timer, 0)

#keyboard inputs to control rotation
def key_input(char, y, z):
   global speed
   if char == b'f':
       speed += 1
   elif char == b's':
       speed -= 1

def windmill_simulation():
   global rotation
   glClear(GL_COLOR_BUFFER_BIT);
   glColor3f(0.5, 0.6, 0.5)

   #base
   glBegin(GL_POLYGON)
   glVertex2f(0,5);
   glVertex2f(10,-450);


   glVertex2f(-120, -450);
```

```
    glEnd()


    windmill.Rotate(0+rotation)

    windmill.Rotate(120+rotation)

    windmill.Rotate(240+rotation)


    glFlush()

    glutSwapBuffers();


if __name__ == "__main__":

    initialization()

    windmill = Transformation(vertices, image)

    glutDisplayFunc(windmill_simulation)

    glutTimerFunc(0,timer,0)

    glutKeyboardFunc(key_input)

    glutMainLoop()
```

Output

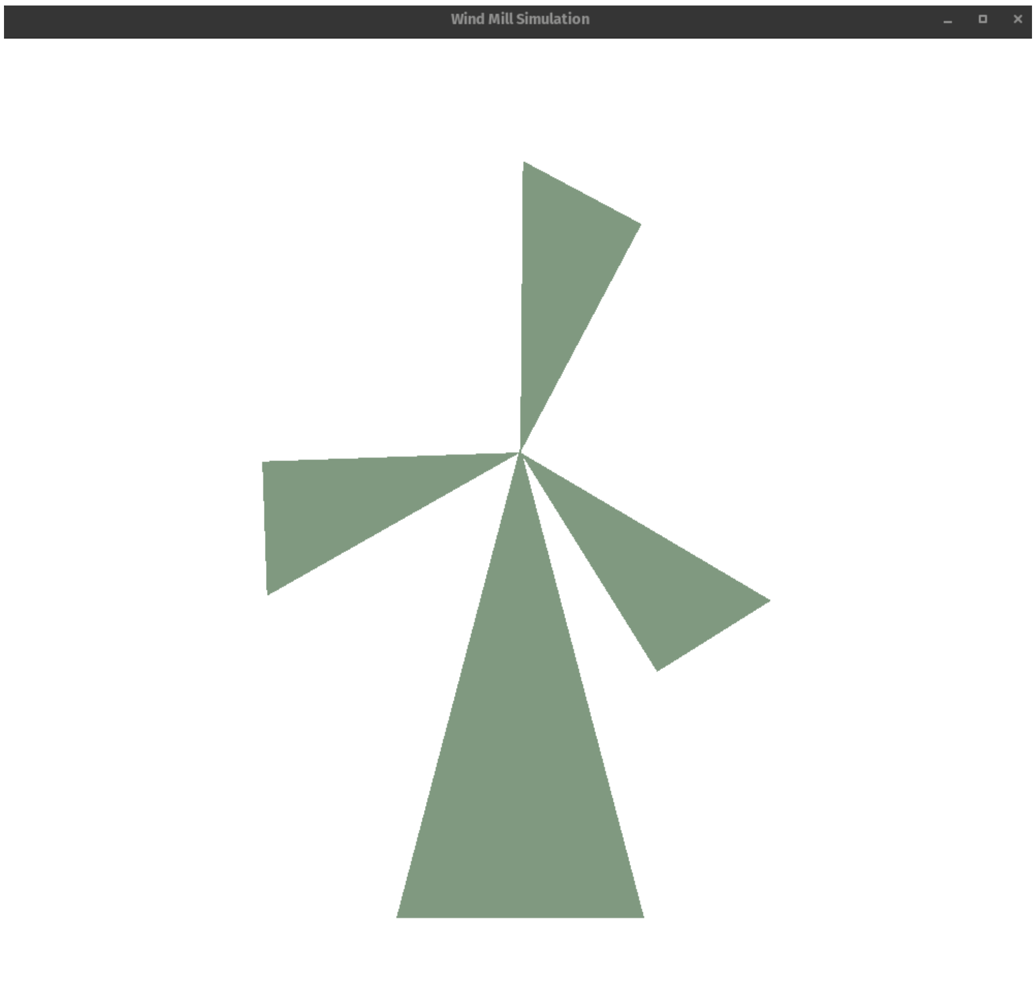


Fig. Windmill rotation

Conclusion

In this way, in the second lab, the midpoint ellipse drawing algorithm, 2D transformations and windmill simulation were achieved in Python using OpenGL. These implementations have helped us expand our knowledge on Computer Graphics and the way images are drawn pixel by pixel in computers. Various OpenGL functions like glutTimerFunc, glutKeyboardFunc and glutPostRedisplay were also implemented while drawing the windmill in this lab.