**Name:** **SID:** **LAB Section:**

**Lab 7 - Part 1: Translation, rotation and scale**

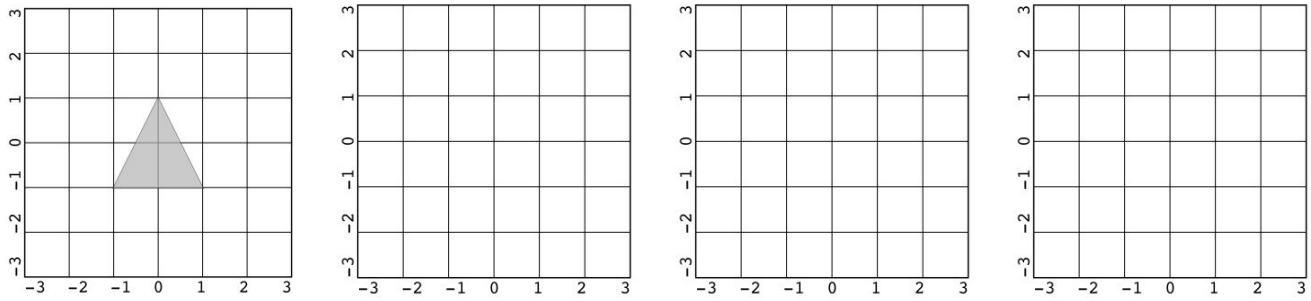
**1.** Given a 2D vector p = (x, y), represent the operation we need to translate, scale and rotate p using matrix multiplication and vector addition.

Translation by (a, b) Rotation by Θ Scale by (s, t)

$p_{tran}$ = $p_{rot}$ = $p_{sca}$ =

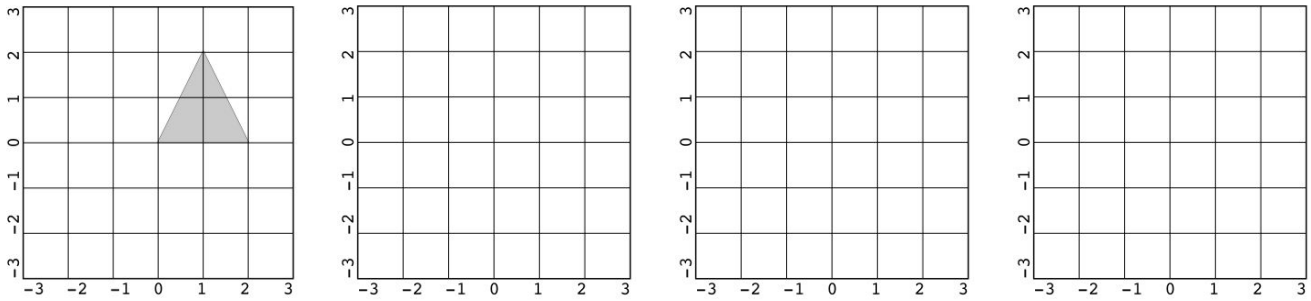
**2.** For the triangle centered at the origin depicted in Fig. 1, draw the resulting triangle after each of the following transformations.

**Fig. 1** Translation (1, 1) Rotation by 90° Scale by (2, 2)



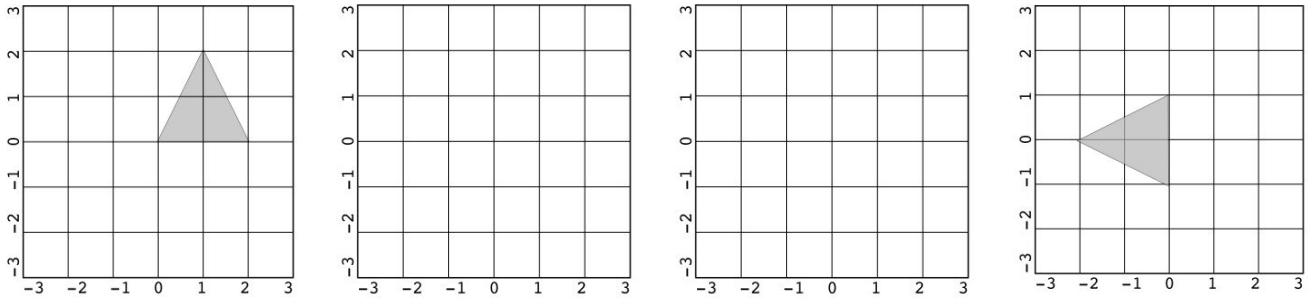

**3.** Suppose we translate the triangle in Fig. 1 by (1, 1). What is the result if we apply the same transformations? Why is the result different?

**Fig. 2** Translation (1, 1) Rotation by 90° Scale by (2, 2)



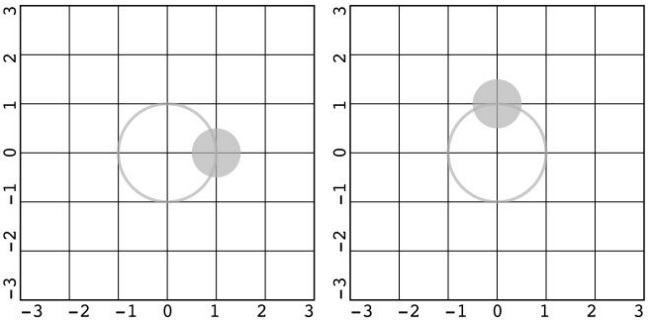
**4.** Write two operations that transform the triangle from Fig. 3 to Fig. 4.

**Fig. 3** _____ _____ **Fig. 4**



**5.** Suppose we want to animate the circle in Fig. 5 around the origin. At time t = 0, the circle is at (1, 0), at time t = 1, the circle is at (0, 1), and so on. Write a formula to animate the circle in terms of t.

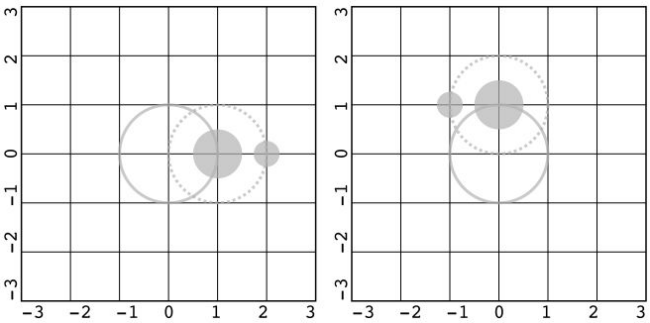
**Fig. 5**



t = 0 t = 1

**6.** Suppose we want to animate a circle around the circle as in Fig. 6. Write a formula to animate the small circle in terms of t.

**Fig. 5**



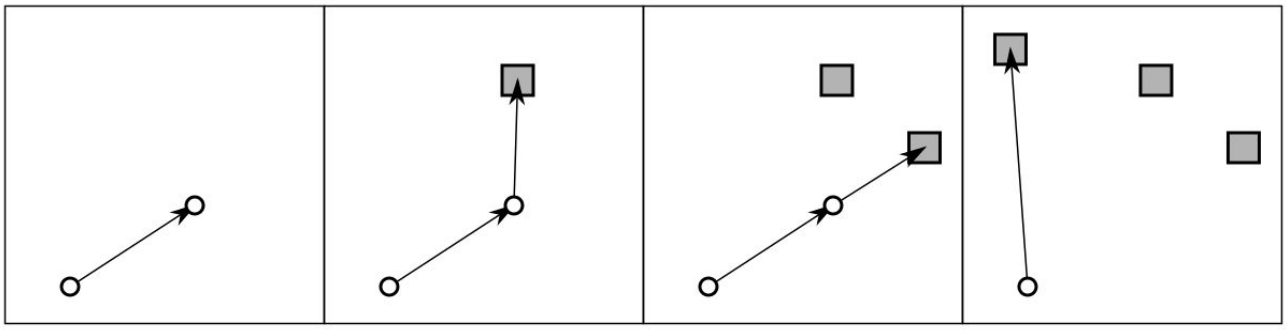
t = 0 t = 1

**Name:** **SID:** **LAB Section:**

**Lab 7 - Part 2: OpenGL transformations and code**

You probably noticed we need some transformations of the big circle in Fig. 5 to determine the position of the small circle. You can imagine the transformations being applied to the small circle relatively to the big circle. In order to do this, OpenGL employs a stack of transformation matrices: at the top, we have the transformations of the small circle; at second to the top, we have the transformations of the big circle. If we wanted to add new circles circulating the big circle, we can just drop all transformations applied to the small circle by removing the top of the stack and add new transformations for the new circle.

**1.** Suppose we want to draw three squares at the three positions depicted in Fig. 6. For the sequence of translations and *draw squares*, write whether we should store the current position using **push (S)** or discard the recent translations using **pop (P).**



_____; Translate; _____; Translate; Draw; _____; Translate; Draw; _____; Translate; Draw;

**2.** We will need the following six OpenGL functions in this lab. Write a brief description for each one of the functions.

glPushMatrix:

glPopMatrix:

glTranslatef(x, y, z):

glRotated(angle, x, y, z):

glScalef(x, y, z):

**2.** Suppose the top of the ModelView stack is the identity. What is on top of the stack after the following OpenGL operations? Write the 4x4 matrix form.

```
glPushMatrix();
glTranslatef(1, 1, 1);
glRotated(90, 0, 0, 1);
glPushMatrix();
glScalef(2, 2, 2);
glPopMatrix();
```

**Top of the stack:**

You can use the following rotation matrix equation taken from Wikipedia.

**Rotation matrix from axis and angle** [ edit ]

For some applications, it is helpful to be able to make a rotation with a given axis. Given a unit vector $\mathbf{u} = (u_x, u_y, u_z)$, where $u_x^2 + u_y^2 + u_z^2 = 1$, the matrix for a rotation by an angle of $\theta$ about an axis in the direction of $\mathbf{u}$ is[4]

$$R = \begin{bmatrix} \cos\theta + u_x^2(1-\cos\theta) & u_x u_y(1-\cos\theta) - u_z\sin\theta & u_x u_z(1-\cos\theta) + u_y\sin\theta \\ u_y u_x(1-\cos\theta) + u_z\sin\theta & \cos\theta + u_y^2(1-\cos\theta) & u_y u_z(1-\cos\theta) - u_x\sin\theta \\ u_z u_x(1-\cos\theta) - u_y\sin\theta & u_z u_y(1-\cos\theta) + u_x\sin\theta & \cos\theta + u_z^2(1-\cos\theta) \end{bmatrix}.$$

**3.** Download the skeleton code on iLearn and implement the function **draw_event** in the **application.cpp** file such that we have the following animation of a simple solar system (try to implement in this order):

- ❑ The star is at the **origin** and has **radius 4.**
- ❑ A planet[1] (**radius 3**) goes around the sun at a **distance of 20.** The planet completes one year **every 10 seconds.**
- ❑ A moon (**radius 2**) goes around the planet at a **distance of 5.** The moon completes one circle around the planet **every 3 seconds.**
- ❑ A satellite orbits the moon (**radius 1**) at a **distance of 2.** The satellite orbits the moon **every 1 second.**

- You can use the glutSolidSphere(**R,** 20, 20) to draw a sphere of radius R.
- You can use t.elapsed() inside draw_event to get the number of seconds elapsed since the program started drawing.

**To think:**

1. Can you make the entire solar system move linearly along an axis?
*You can achieve this by a single line of code.*

2. Are the number of pushes and pops in an OpenGL code always the same?

[1] The planet is round the same way the Earth is round (not flat!).