Assignment 4: Transformation in 2D and 3D

CS Spring 2018

Due Date:

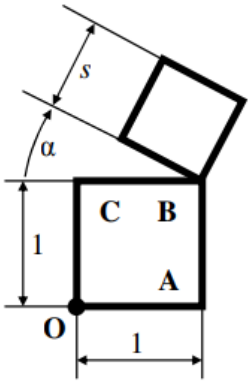
Follow the instructions carefully. If you encounter any problems in the setup, please do not hesitate to reach out to TA.

Introduction

This assignment reviews basic geometric transformations by some exercises, both 2D and 3D transformations are included. The "transformation" discussed in the assignment refers to geometric operation applied to all the points of an object. An object may be moved, rotated, scaled. Several transformations may be combined and may completely change the position, orientation, and shape of the object. In the end, you can try yourselves to transform an object in OpenGL with the help of GLM which was mentioned in last assignment.

Math problems

1. Given a square with width and height 1, with the lower left corner in the origin:



   a. Give a transformation matrix to transform this square to a square with width *s*, rotated over an angle α, such that the lower right corner coincides with the upper right corner of the original square (see figure).
   b. We consider a more generic version. Give a recipe for a transformation matrix for a rotated and scaled unit square **OABC**, such that a new corner **O'**, **A'**, **B'** or **C'** coincides with one of the original corners **O**, **A**, **B** of **C**.

2. We consider 3D rotations.
   a. Give 3×3 rotation matrices for a rotation $\mathbf{R}_x(\alpha)$ around the *x*-as and a rotation $\mathbf{R}_z(\alpha)$ around the *z*-axis.
   b. Calculate (explicitly) the composite rotation matrix for a rotation over 90 degrees ($\pi/2$ radians) around the *x*-axis followed by a rotation over 90 degrees over the (global) *z*-axis.
   c. What is the effect of these two rotations? Elucidate your answer.
   d. Do we get the same effect if we change the order of the rotations?

3. We consider a distorted cube. In the original state the cube is described by $0 \le x, y, z \le 1$. After distortion the following applies:
   - The lower face (in the XOY-plane) remains fixed;
   - The upper face is horizontal, but moves: the point (0,0,1) moves to (*a, b, c*) ;
   - all edges remain straight and all faces remain flat.

   a. Give a transformation matrix for this distortion.

4. Given a digital map. It is desired to show a part of this map in a viewport on the screen. In the center of the viewport the point **C** (in map coordinates) has to be shown, the width of the part to be displayed is *w* (again in map coordinates). Obviously, distortion is not allowed. The viewport is specified in pixels. For the pixel coordinates it holds that the origin (the point (0,0)) is located in the upper left corner and the point (Nx−1, Ny−1) in the lower right corner. The upper left corner of the viewport has coordinates ($\mathbf{X}_{mi}$, $\mathbf{Y}_{mi}$), the lower right corner ($\mathbf{X}_{ma}$, $\mathbf{Y}_{ma}$).
   a. Give a transformation to map a point *Q* in map coordinates to a point *Q'* in pixel coordinates.

   b. A user indicates a point **R'** (pixel coordinates). Give a transformation to determine the corresponding point *R* in map coordinates.

5. Given a 3D scene that is viewed with a virtual camera. The camera has position **P**, points in the direction **W**, and lines in the direction of a vector **V** are shown vertically on the screen. It may be assumed that **V** and **W** are unit vectors and that they are orthogonal. In the following questions the effect of a given camera movement on **P**, **W** and **V** is asked for.
   a. Give recipes to move the camera forward and backward, up and down, and to the left and to the right.
   b. Give recipes to rotate the camera: *pitch* (left/right), *yaw* (up/down), *roll* (rotation around center of viewing axis).
   c. Suppose that the camera is centered at a point **C**. Give again recipes for the rotation, such that the camera remains fixed on this point.

Practice

In last assignment, GLM has already been installed properly, it's time to see how we can actually use this knowledge to our advantage. Most of GLM's functionality that we need can be found in only 3 headers files that we'll include as follows:

```
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include <glm/gtc/type_ptr.hpp>
```

Let's see if we can put our transformation knowledge to good use by translating a vector of (1,0,0) by (1,1,0) (note that we define it as a glm::vec4 with its homogenous coordinate set to 1.0:

```
glm::vec4 vec(1.0f, 0.0f, 0.0f, 1.0f);
```

```
glm::mat4 trans;
trans = glm::translate(trans, glm::vec3(1.0f, 1.0f, 0.0f));
vec = trans * vec;
std::cout << vec.x << vec.y << vec.z << std::endl;
```

The next big question is: how do we get the transformation matrix to the shaders? We shortly mentioned before that GLSL also has a mat4 type. So we'll adapt the vertex shader to accept a mat4 uniform variable and multiply the position vector by the matrix uniform:

```
#version 330 core
layout (location = 0) in vec3 aPos;
layout (location = 1) in vec2 aTexCoord;

out vec2 TexCoord;

uniform mat4 transform;

void main()
{
    gl_Position = transform * vec4(aPos, 1.0f);
    TexCoord = vec2(aTexCoord.x, aTexCoord.y);
}
```

We added the uniform and multiplied the position vector with the transformation matrix before passing it to gl_Position. Our container should now be twice as small and rotated 90 degrees (tilted to the left). We still need to pass the transformation matrix to the shader though:

```
unsigned int transformLoc = glGetUniformLocation(ourShader.ID,
"transform");
glUniformMatrix4fv(transformLoc,          1,          GL_FALSE,
glm::value_ptr(trans));
```

Keep in mind that in the previous case we could declare the transformation matrix anywhere, but now we have to create it every iteration so we continuously update the rotation. This means we have to re-create the transformation matrix in each iteration of the game loop. Usually when rendering scenes we have several transformation matrices that are re-created with new values each render iteration.

Exercises

1. Finish *Math problems* section, hand in the homework in any forms.
2. Try to draw a self-rotating square, or triangle, and it can also freely move within the screen