# Computer Graphics Hand-in-2

xFrednet

September 28, 2020

# 1 TGRA Handin 2

## 1.1 (10%) Transformations and matrices

You have access to the same matrix and graphics classes as we built together this semester, that is a BoxGraphic class with a drawSolidCube that sends vertices into the pipeline for a cube of the size 1x1x1, centered in (0,0,0), and a ModelMatrix class with the basic transformation operations, matrix stack operations and shader manipulation operations. These classes have been created and initialized elsewhere. You don't need to remember the exact names of functions, just the idea behind each one you need to call, and what parameters they take.

What you want to draw is the following scene:

### 1.1.1 a) A box of size 2x2x2, centered in (9,5,-2) and under it a big flat floor which is 10x10 in the x-z plane but only 0.8 thick. Its corner should be in (0,0,0) and its top edge level with the base x-z plane.

I assume that the BoxGraphic instances have been setup with the following scale and position class members: I ignore the rotation because it wasn't mentioned :)

$$Scale_{Box} = \begin{pmatrix} 2.0 \\ 2.0 \\ 2.0 \end{pmatrix} \; Position_{Box} = \begin{pmatrix} 9.0 \\ 5.0 \\ -2.0 \end{pmatrix}$$

$$Scale_{Plane} = \begin{pmatrix} 10.0 \\ 0.8 \\ 10.0 \end{pmatrix} \; Position_{Plane} = \begin{pmatrix} 5.0 \\ -0.4 \\ 5.0 \end{pmatrix}$$

I've created two solutions for the code. The first one is just the code I created without looking at my code or any material. The seconds solution is a revamped and detailed solution. I would like to get feedback for both if possible.

**First solution:**

```
vertex_count = 6 * 2 * 3
for e in entities:
    matrix = (ModelMatrix.identity()
        .translate(e.position)
        .scale(e.scale))
    gl.glLoadUniform(self.transformation_matrix_loc, matrix.c_ptr())
    e.drawSolidCube()
```

**Second solution:**

```
for e in entities:
    matrix = (ModelMatrix.identity()
            .translate(e.position)
            .scale(e.scale))
    # Bind buffers
    gl.glEnableVertexAttribArray(StandardShaderProgram.POSITION_ATTR)
    gl.glEnableVertexAttribArray(StandardShaderProgram.COLOR_ATTR)

    # Draw the beautiful
    gl.glUniformMatrix4fv(transformation_matrix_loc, 1, gl.GL_FALSE, matrix.c_ptr())
    e.drawSolidCube()

    # Unbind the thingies
    gl.glDisableVertexAttribArray(StandardShaderProgram.POSITION_ATTR)
    gl.glDisableVertexAttribArray(StandardShaderProgram.COLOR_ATTR)
```

### 1.1.2 b) (5%) Show the values that are in the shader's model matrix when the first box is drawn.

$$model\_matrix = \begin{pmatrix} 2 & 0 & 0 & 9 \\ 0 & 2 & 0 & 5 \\ 0 & 0 & 2 & -2 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

**Check with python for validation:**

```
>>> import glm
>>> mat = glm.mat4()
>>> mat = glm.translate(mat, e.position)
>>> mat = glm.scale(mat, e.scale)
>>> print(mat)

[            2 |             0 |             0 |             0 ]
[            0 |             2 |             0 |             0 ]
[            0 |             0 |             2 |             0 ]
[            9 |             5 |            -2 |             1 ]
```

---

**LaTeX is so much fun :)**

## 1.2 5. Camera Transformations (30%)

### 1.2.1 a) (15%) A camera is set up to be positioned in (0,8,4) looking at the point (0,3,-1). It has an up vector (0,0,-1). Find the point of origin and vectors for the camera's coordinate frame.

$$origin = \begin{pmatrix} 0 \\ -8 \\ -4 \end{pmatrix}$$

I'm not a 100% sure which vectors I should calculate for the coordinate frame but I expect that these are $\vec{v}, \vec{n}, \vec{u}$ for the view matrix. This would also kind of make sense. So here we go:

$$\vec{n} = normalize(\vec{origin} - \vec{looking\_at}) = normalize \begin{pmatrix} 0-0 \\ 8-3 \\ 4--1 \end{pmatrix} = normalize \begin{pmatrix} 0 \\ 5 \\ 5 \end{pmatrix} \approx \begin{pmatrix} 0 \\ 0.707 \\ 0.707 \end{pmatrix}$$

$$\vec{u} = normalize\left(\vec{up} \times \vec{n}\right) \approx normalize \begin{pmatrix} 0.707 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

$$\vec{v} = normalize\left((\vec{n} \times \vec{u})\right) \approx \begin{pmatrix} 0 \\ 0.707 \\ -0.707 \end{pmatrix}$$

### 1.2.2 (5%) Set up the values in a matrix that represents this position and orientation of a camera. Which matrix in your shader should be set to these values?

From formula sheet: $\begin{pmatrix} u_x & u_y & u_z & -(eye \circ u) \\ v_x & v_y & v_z & -(eye \circ v) \\ n_x & n_y & n_z & -(eye \circ n) \\ 0 & 0 & 0 & 1 \end{pmatrix}$

$\texttt{ViewMatrix} \approx \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0.707 & -0.707 & -2.828 \\ 0 & 0.707 & 0.707 & -8.485 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

The view matrix should be set to these values.

### 1.2.3 (10%) The camera should have a field of view of 75 grad, an aspect ratio of 16:9, a near plane at 3 and a far plane at 25. Find the exact values for a matrix that calculates this camera. Which matrix in your shader should be set to these values?

From formula sheet: $\begin{pmatrix} \frac{2N}{right-left} & 0 & \frac{right+left}{right-left} & 0 \\ 0 & \frac{2N}{top-bottom} & \frac{top+bottom}{top-bottom} & 0 \\ 0 & 0 & \frac{-(F+N)}{F-N} & \frac{-2*F*N}{F-N} \\ 0 & 0 & -1 & 0 \end{pmatrix}$

N = 3

F = 25

fov = 75

aspect = $\frac{16}{9}$

top = $N * tan\left(\frac{fov}{2}\right) \approx 3 * 0.767 \approx 2.301$

bottom = $-top \approx -2.301$

right = $top * aspect \approx 4.091$

left $= -right \approx -4.091$

$$\texttt{ProjectionMatrix} \approx \begin{pmatrix} 0.733 & 0 & 0 & 0 \\ 0 & 1.304 & 0 & 0 \\ 0 & 0 & -1.273 & 6.812 \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

The projection matrix should be set to these values.