# Computer Graphics
# Hand-in-3

xFrednet

October 5, 2020

# 1 TGRA Hand-in 3

## 1.1 Lighting calculations (10%) (2014 No. 6)

A single light is in the light model in an OpenGL program. It has the ambient values (0.0, 0.0, 0.0), diffuse values (0.5, 0.3, 0.7), specular values (0.3, 0.8, 0.7) and position (5.0, 8.0, -1.0). There is also a global ambient factor of (0.3, 0.2, 0.4) in the light model. A camera is positioned in (4.0, 6.0, 5.0) and looks towards P.

P has the color values: ambient (0.4, 0.2, 0.3), diffuse (0.4, 0.7, 0.2) and specular (0.6, 0.6, 0.6). It has a shininess value of 13. It has the position (4.0, 4.0, 3.0) and a normal (0.0, 1.0, 0.0).

### 1.1.1 What will be the blue color value for P on the screen ?

I'm going to calculate the RGB value for the given point. This is not required but probably a good practice :)

$$n = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$$

$$S = light_{pos} - P = \begin{pmatrix} 1 \\ 4 \\ -4 \end{pmatrix}$$

**diffused lighting**

$$lambert = max\left(\frac{n \circ S}{|n| * |S|}, 0\right) = max\left(\frac{4}{\sqrt{33}}, 0\right) \approx 0.6963$$

**specular lighting**

$$v = camera_{pos} - P = \begin{pmatrix} 0 \\ 2 \\ 2 \end{pmatrix}$$

$$h = v + S = \begin{pmatrix} 1 \\ 6 \\ -2 \end{pmatrix}$$

$$phong = max\left(\frac{n \circ h}{|n| * |h|}\right) = max\left(\frac{6}{\sqrt{41}}, 0\right) \approx 0.9370$$

$$f = 13(shininess)$$

**light**

$$I = (I_d * material_d * lambert) + \left(I_s * material_s * phong^f\right) + (I_a * material_a) + (I_{gaf} * material_a)$$

$$I \approx \left(\begin{pmatrix} 0.5 \\ 0.3 \\ 0.7 \end{pmatrix} * \begin{pmatrix} 0.4 \\ 0.7 \\ 0.2 \end{pmatrix} * 0.6963\right) + \left(\begin{pmatrix} 0.3 \\ 0.8 \\ 0.7 \end{pmatrix} * \begin{pmatrix} 0.6 \\ 0.6 \\ 0.6 \end{pmatrix} * 0.9370^{13}\right) + \left(\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} * \begin{pmatrix} 0.4 \\ 0.2 \\ 0.3 \end{pmatrix}\right) + \left(\begin{pmatrix} 0.3 \\ 0.2 \\ 0.4 \end{pmatrix} * \begin{pmatrix} 0.4 \\ 0.2 \\ 0.3 \end{pmatrix}\right)$$

$$I \approx \left(\begin{pmatrix} 0.20 \\ 0.21 \\ 0.14 \end{pmatrix} * 0.6963\right) + \left(\begin{pmatrix} 0.18 \\ 0.48 \\ 0.42 \end{pmatrix} * 0.9370^{13}\right) + \begin{pmatrix} 0.12 \\ 0.04 \\ 0.12 \end{pmatrix}$$

$$I \approx \begin{pmatrix} 0.3365 \\ 0.3922 \\ 0.3977 \end{pmatrix}$$

The blue value of the object in point P is $\approx$ 0.3977

## 1.2 (10%) Cohen-Sutherland Clipping (2015 No. 2)

A clipping window has the following geometry:

- Window(left, right, bottom, top) = (200, 600, 100, 400)

A line with the following end points is drawn in the world:

- P1: (240, 480)
- P2: (140, 300)

### 1.2.1 Show how the Cohen-Sutherland clipping algorithm will clip these lines and what their final endpoints, if any, are. Show the coordinate values of P1 and P2 after each pass of the algorithm.

| | **left** | **center** | **right** |
|---|---|---|---|
| **top** | 1001 | 1000 | 1010 |
| **center** | 0001 | 0000 | 0010 |
| **bottom** | 0101 | 0100 | 0110 |

**Iteration 1**

```
P1_code = 1000
P2_code = 0001

// => Clip P1 to the top

P1.x += (top - P1.y) * (P2_x - P1_x) / (P2_y - P1_y) // -44.4444
P1.y = top

// P1: (195.5, 400)
// P2: (140  , 300)
```

**Iteration 2**

```
P1_code = 0001
P2_code = 0001

// Both points are outside on the same side
// => The entire line gets clipped. The points from the last
//    Iteration still stand
//      * P1: (195.5, 400)
//      * P2: (140  , 300)
```

---

Okay, this is the end of this hand-in. I have to say that I've gotten used to writing LaTeX by now. It's still slower than writing by hand or in the case of No.2 just using a monospace environment. But this is a good exercise and I had some fun writing these 300 lines ^^

# 2 TGRA Handin 2

## 2.1 (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:

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.

### 2.1.1 (5%) Write the code that would draw the scene described. Imagine you're already inside the display function and lights, colors and cameras have already been set.

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:
    self.model_matrix.load_identity()
    self.model_matrix.translate(e.get_position())
    self.model_matrix.scale(e.get_scale())

    self.shader.set_model_matrix(self.model_matrix.matrix)
    e.drawSolidCube()

    pygame.display.flip()
```

### 2.1.2 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 :)**

## 2.2 5. Camera Transformations (30%)

### 2.2.1 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.

$$origin = \begin{pmatrix} 8 \\ 40 \end{pmatrix}$$

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:

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

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

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

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

`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.

### 2.2.3 (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.