GPU Programming
Lab Sheet 1

Pedro Hermosilla Casajus

1 Introduction

The objective of this exercise is to get familiar with the *GLSL* programming language and the development environment that we will use in the rest of lab sheets. The applications used to develop our shaders will be based on *Python* and *PyOpenGL*. However, for most of the sheets, the student will only need to write code in *GLSL*.

2 Configuration of the environment

The first thing we have to do is install *Python 3.7*. You can download it from: https://www.python.org/

Once we have installed Python we need to install several packages. First, we will need to install *numpy*. We should open the command line and execute the following command:

```
pip install numpy
```

The next package to install is *PyOpnGL*. We follow the same procedure as with the numpy package. In the command line we type:

```
pip install PyOpenGL
```

Lastly, we install *pygame*.

```
pip install pygame
```

Once everything is installed properly, we can execute our application. We open the command line and go to the folder where we uncompressed the code of the exercise. Then we execute the following command:

```
python Application.py --in3DModel Bunny.obj
```

We should obtain an output similar to:

```
pygame 1.9.4
Hello from the pygame community. https://www.pygame.org/contribute.html
Loaded model Bunny.obj
Vertexs: 14900
faces: 4968
Traceback (most recent call last):
  File "Application.py", line 125, in <module>
    MyGL = GLScene(512, 512, coordMin, coordMax, rendVert, rendFaces)
  File "Application.py", line 50, in __init__
    [GL.GL_VERTEX_SHADER, GL.GL_FRAGMENT_SHADER])
  File "...\OpenGLUtils.py", line 55, in load_shader
    self._link_(currProgram)
  File "...\OpenGLUtils.py", line 21, in _link_
    raise RuntimeError("Linking failue: "+log)
RuntimeError: Linking failue: ERROR: Definition for "void main()" not found.
```

Our program was able to run. However, since the shader files are empty the program was not able to show the 3D model.

3 Creating your first shaders (5 pts)

The first part of the exercise consists of creating a vertex and a fragment shader. The files *vertexShader.glsl* and *pixelShader.glsl* are currently empty and we should write the code to render a triangle mesh. Copy the following code and paste it in the file *vertexShader.glsl*:

```
#version 450

uniform mat4 worldViewProjMatrix;

in vec4 sPos;
in vec3 sNormal;

out vec3 normal;
```

```
void main()
{
    normal = sNormal;
    //TODO - Add the vertex transformation
}
```

As you can see there is a *TODO* comment. Here you have to add the code to transform the input vertex by the *worldViewProjMatrix*.

Then, copy the following code and paste it in the file *pixelShader.glsl*:

```
#version 450

uniform vec3 lightDirection;
uniform vec4 inColor;

in vec3 normal;

out vec4 outColor;

void main()
{
    outColor = dot(lightDirection, normal)*inColor;
}
```

Now if we run the application again we will see a green Bunny in the middle of the screen.

4 Shader description (5 pts)

In this part of the exercise, you will need to describe what the shader does and what are the differences compared to the exampled presented in the first lecture. Moreover, you will have to describe if you see some problem with the proposed implementation. Will this implementation work if we have multiple instances of the same model? In which coordinate system the light is defined?