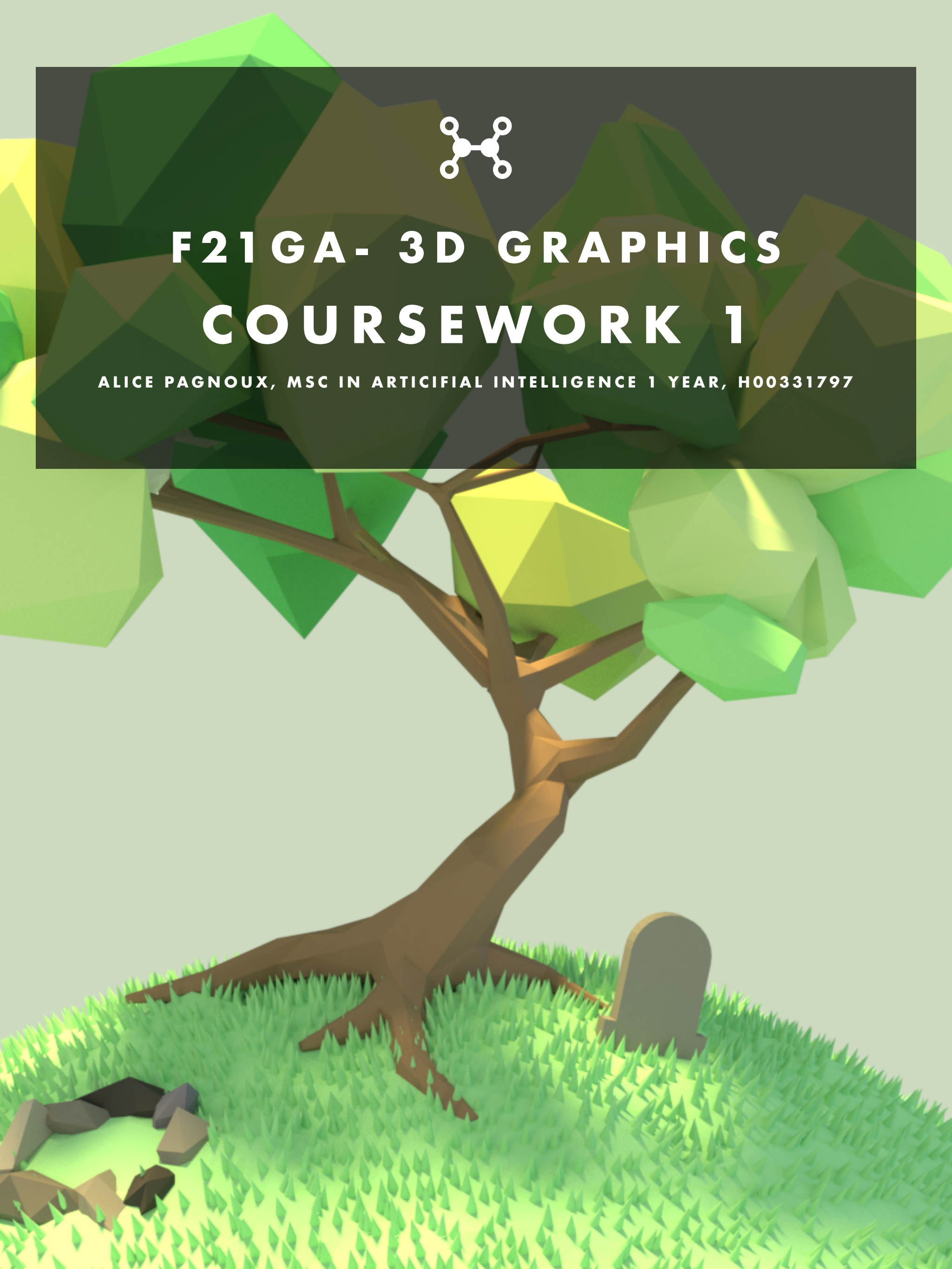


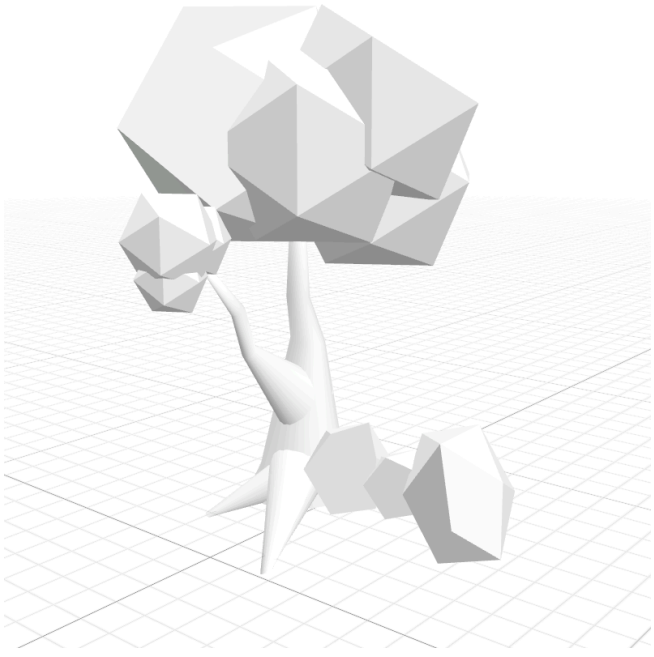
F21GA- 3D GRAPHICS

COURSEWORK 1

ALICE PAGNOUX, MSC IN ARTICIFIAL INTELLIGENCE 1 YEAR, H00331797

BEHIND THE SCENE

The main idea was to create a scene the user could interact with and observe, its many characteristics such as the contrast, lights and movements. This coursework is meant to peel up the different stages of the forward rendering mapping pipeline step by step. We will take the example of a simple tree, designed as a low poly.



1.MODELIZATION AND LOADING

First of all, we need to create our object, composed by different shapes. The software Maya Autodesk allows us to build up shapes such as cubes and cones, and group them together. The idea was to design a very simple scene so we could manipulate its characteristics easier. To handle all of them, I used Xcode for MacOS. The very first step was to implement many different libraries to handle our code to be. The model is at then end loaded to the code thanks to the Assip library and functions.

I. Library implementation

GLFW

GLFW s an Open Source, multi-platform library for OpenGL, OpenGL ES and Vulkan development on the desktop. It provides a simple API for creating windows, contexts and surfaces, receiving input and events. [1]

GLEW

The OpenGL Extension Wrangler Library (GLEW) is a cross-platform open-source C/C++ extension loading library. GLEW provides efficient run-time mechanisms for determining which OpenGL extensions are supported on the target platform. [2]

GLM

OpenGL Mathematics is a header only C++ mathematics library for graphics software based on the OpenGL Shading Language (GLSL) specifications. [3]

2. MATERIALS AND LIGHTING

1. VBO, VAO and EBO

The Vertex Buffer Object is an OpenGL feature that provides methods for uploading vertex data such as the position, the color, etc. It a memory buffer, an array of bytes of memory which is already in the

GPU. We want to define a bunch of data that represents our tree for example. We will put them in the GPU's RAM. We also tell the GPU how to read and interpret our datas, and how to put them on the screen. So in other words, a shader is just a program that runs on the GPU. « Vertex and fragment shaders are the most important shaders in the whole pipeline, because they expose the pure basic functionality of the GPU. » [4]. With vertex shaders, I computed the geometry of the object that I wanted to render. On an other hand, fragment shaders control how my geometry will look on screen : it manages the colors and the texture.

How to implement them ?
First of all, we need to declare variable that will contain the ids for each shader (vbo, vao and ebo).

```
GLuint vao, vbo, ebo;
```

For the VBO:
We need to use the function:

```
glGenBuffers(1, &vbo);
```

Then we have to bind the vbo, or load the data into our vertex buffer. Binding tells which vbo is going to be considered as active for the future operations on it.

```
glBindBuffer(GL_ARRAY_BUFFER, this->vbo);
```

Our datas here are our vertices, so we shall send the datas of the object (the tree) to the graphic card. So we define the size, the datas and the type of the VBO:

```
glBufferData(GL_ARRAY_BUFFER, this
->vertices.size() * sizeof(Vertex), &this-
->vertices[0], GL_STATIC_DRAW);
```

At the end, we enable our VBO:

```
glBindVertexArray(0);
```

It works quite the same for the VAO:
We create (generate) the VAO:

```
glGenVertexArrays(1, &this->vao);
```

Then we need to bind it, to tell our GPU which one we are going to use:

```
glBindVertexArray(this->vao);
```

At the end, we enable our VAO:

```
glEnableVertexAttribArray(0);
```

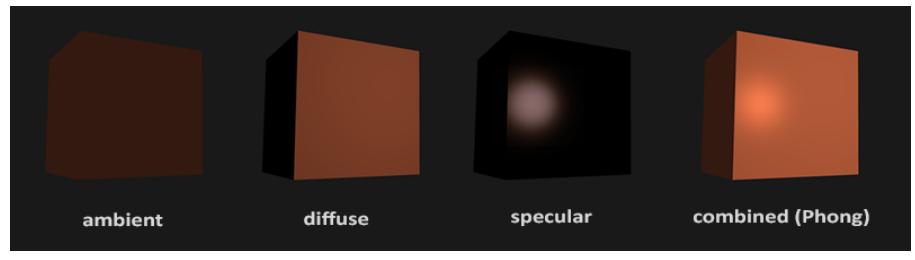
The Element Buffer Object is a type of buffer that lets us re-use vertices to create multiple primitives out of them. So, for example, we can create a rectangle out of 4 vertices only. I implemented it the same way as the two previous shaders.

2. Textures

For this part, I have been using SOIL, as recommended by [6]. It is used for uploading textures into OpenGL. [9]. Textures are used to add more details and realism to the objects. They are a 2D image (in the code, they're always referred as vec2 and not vec3). We need to tell each vertex of the object which part of the texture it corresponds to. Each vertex will then have a texture coordinate associated that specifies what part of the texture image to sample from. So in the first place, we have to set up the textures coordinates:

```
glm::vec2 TextureCoordinates;
```

3. Basic lighting



A. Diffuse

A diffuse light is used to define a surface's main color. It is like painting on the surface of an object. The diffuse component is the light that comes from one direction, so it's brighter if it comes squarely down on a surface than if it barely glances off the

surface. Once it hits a surface, however, it's scattered equally in all directions, so it appears equally bright, no matter where the eye is located. [10]

B. Ambiant

Ambient illumination is light that's been scattered so much by the environment that its direction is impossible to determine - it seems to come from all directions. Backlighting in a room has a large ambient component, since most of the light that reaches your eye has first bounced off many surfaces. A spotlight outdoors has a tiny ambient component; most of the light travels in the same direction, and since you're outdoors, very little of the light reaches your eye after bouncing off other objects. When ambient light strikes a surface, it's scattered equally in all directions. [10]

C. Specular

Specula textures are used to define a surface's shininess and highlight colors. The higher the value of a pixel, the shinier the surface will appear on the screen. specular light comes from a particular direction, and it tends to bounce off the surface in a preferred direction. A well-collimated laser beam bouncing off a high-quality mirror produces almost 100 percent specular reflection. Shiny metal or plastic has a high specular component, and chalk or carpet has almost none. In other words, specularity is like the shininess. [10]

3. CAMERA

1. Movements

In order to apply movements to our objects, we need to include the GLM library.

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

We are able, in the code, to change the position of the camera by translating it, and by changing the scale as well.

```
glm::mat4 model(1);
model = glm::translate(model, glm::vec3(0.0f, -1.75f, 0.0f));
model = glm::scale(model, glm::vec3(0.2f, 0.2f, 0.2f));
```

4. ANIMATIONS AND INTERACTIONS

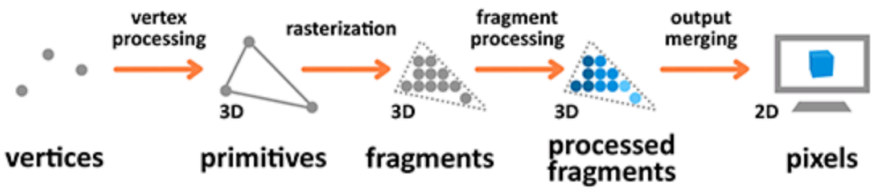
1. Keyboard interaction

The camera can be moved both by the mouse and with the keyboard (W, Q, S, D). The keyboard moves the position of the camera and the mouse allows to rotate it.

CONCLUSION

The goal behind this coursework was to create a simple scene, containing objects, and to show the different aspects we need to think about when we create a video game for example. So we had to go through all the steps required according to the pipeline that was demonstrated in class in

order to show all the characteristics that such a scene could have. For example, we implemented a 3D depth, added textures to the objects, and created a source of light so we could see the influence of the light and darkness on the scene.



REFERENCES

I want to give the credits of my code to 'Sonar Systems'[6] and 'The Cherno' [5]. I have been following their tutorial videos and learnt more about the graphic conception thanks to them. My code is largely inspired by their explanations and their own code that are at obtainable on Github.

[1] GLFW Library - https://www.glfw.org

[2] GLEW Library - https://www.opengl.org/sdk/libs/GLEW/

[3] GLM Library - https://glm.g-truc.net/0.9.9/index.html

[4] Types of shaders - https://subscription.packtpub.com/book/hardware_and_creative/9781849698009/1/ch01lvl1sec10/types-of-shaders

[5] The Cherno - OpenGL tutorials, https://www.youtube.com/watch?v=W3gAzLwfIP0&list=PLlrATfBNZ98foTJPJ_Ev03o2oq3-GGOS2

[6] Sonar Systems - Modern OpenGL 3.0 - [GETTING STARTED], [MODEL LOADING] and [LIGHTING] - Youtube : https://www.youtube.com/watch?v=Tz0dq2krCW8&list=PLRtjMdoYXLf6zUMDJVRZYV-6g6n62vet8&index=1. Github : https://github.com/SonarSystems/Modern-OpenGL-Tutorials

[7] OpenGL-tutorial - http://www.opengl-tutorial.org/beginners-tutorials/tutorial-2-the-first-triangle/

[8] Thebennybox - Youtube : « Update: "Proper" OBJ Loading and Modern OpenGL ». Github : https://github.com/BennyQBD/ModernOpenGLTutorial

[9] SOIL Library - https://bitbucket.org/SpartanJ/soil2

[10] Lights - https://www.glprogramming.com/red/chapter05.html