Soft 323
Assignment 1: Flying Thing with Object Loader

0.0 Introduction

For this assignment, I originally started out by creating a working prototype that was comprised of a few files. The application in its original state was free from memory crashes, worked perfectly, but lacked an object orientated approach, which is why I decided to swap over and adopt and use the framework that you kindly provided as a starting point. The reason for this was purely centred on allowing me to learn all of the concepts required for the second assignment (I presumed that the assignment brief from last year would not change too much).

The additions that have been added will be pointed out in the report alongside random thought processes where appropriate. Unfortunately, memory de-allocation issues and the object loader are the primary concerns, but I have given both a good go, and a large amount of time was put into making everything appear as it currently is.

1.0 Version of Visual Studio and DirectX SDK

Version:

- Visual Studio Professional 2012

DirectX SDK:

- Microsoft DirectX SDK (June 2010)

Additional information:

The executable, include and library directories point towards the 32 bit files for the SDK. The platform toolset is Visual Studio 2012 (v110).

1.1 An end user's point of view – how to work it

Summary of application:

You may find this submission rather quirky. It features a tiger that flies around an island, There is a harbour, a big rock, zebras and lizards, (that will in time act as if they are eating grass), the sea and finally (to prove that the object loader sort of works) and a cup that has loaded textures.

Controls:

Spacebar: makes the tiger fly upwards. If the tiger is idle in the air, the speed and height decreases and he will lower to the ground.

X button: zoom's the camera outwards.

Z button: zoom's the camera inwards.

1.2 From a programmer's point of view

Although this application was created using the provided framework, to make this easier to mark and to quickly understand the flow of the application in terms of the changes made, the following list has been constructed:

Sound and initialisation:

- The DXUT methods for initialising the application, DirectX Input and the creation of meshes and objects are carried out from the **Basic Thing 3D STARTS HERE** file.

  Although the sound manager class could have been in an object oriented sense (passing the file name to a set method for example), the flying thing class was the only one that used sound. In assignment 2, I may change this so that I can make the zebras have noises and the planned change in terms of structure will reflect this.

  To be more precise however: The **sound manager** is used and contained in the **flying thing.cpp** file.

Flapping thing with wings (or components that move relative to other bits):

- All of the meshes and objects have been created in the **OnD3D11CreateDevice** method in the **Basic Thing 3D STARTS HERE** file. Each object (or the arrays of objects) is created from the methods that can be found there.

- The **FlyingThing3D.cpp** contains the creation of the wing meshes and the sky – since the main player is this object and it is only created once. Therefore look at the

CreateFlyingThing method to see the flow in terms of the creation of meshes, and of course the RenderForMyImplementation method for the changes during runtime.

- The **Thing3D.cpp** files in terms of the creation of meshes and the rendering has also changed and might be interesting – although these are only minor changes.

Extra directinput functionality and relationship to the object:

- The variables for speed and height are changed in the DoBehaviour methods for **FlyingTiger3D.cpp** and are referenced in the OnFrameMove method of **Basic Thing 3D Starts here.**

- The height variable has been created, this is used in the following ways:
  - When the tiger stops in mid-air, the height decreases.
  - Of course, when the spacebar is pressed the tiger raises upwards (along with the camera), and downloads when the spacebar is not pressed down.
  - For the camera setup, look at the camera sections for both the FlyingTiger.cpp RenderForMyImplementation method, and the **Basic Thing 3D STARTS HERE**, render method.

- Extra functionality: Spacebar.

- Extra functionality: X button to zoom out.

- Extra functionality: Z button to zoom in.

- To come: move the camera left and right.

- To come: I want to make some sort of shooting dynamic once I change the tiger to something more fitting (when the object loader works completely).

- Thoughts: If I had a main player that could both fly and walk, I could create both a jump button and a fly button – which might be something that may happen.

Memory de-allocation:

- The abstract classes destructor is called when an object is deleted, which required me to create a cleanup method for the shaders (since for the arrays of tiles and animals, they seem to share a global shader and therefore memory errors occur if more than one shader is deleted).

- For each object, an attempt to clean up resources has been made, so therefore look at each class's destructor. As a side note: you were definitely correct when you said that it would cause a headache trying to clean everything up.

- The sound manager class seems to mess with the programs ability to change vertical sync. This will have to be addressed in the future. This also hinders the programs ability to enter full screen mode too.

- Resizing seems to work correctly.

Other creative arty effects:

- The grass and sea use the same sdkmesh. The .bmp file however that is in the folder where the mesh is located (there are two seafloor.sdkmesh folders) are different, and the look is changed because of this. Refer to the NewGrass and NewLand methods in **BasicThing3D STARTS HERE.** The bmps used are from websites that provide free pictures, and the links can be found in the methods.

- Shadows: The tiger and island have shadow effects. The tiger in particular uses a different shader that was found from an online tutorial and the link can be found in the pixel shader file for the application (I was intending to find a way to create fur for the tiger, but it did not work out as such).

- Wagging tail: For the tiger, the tail wags up and down. This can be seen by zooming the camera down (if it can't be seen that clearly).

- To come: I will be adding a beach soon.

- To come: wagging heads for the zebras, in a non-uniform fashion (a random variable determines the starting position of the head, and it will look as if they are eating grass).

Object Loader:

- When the application is loaded, a cup can be seen that has the correct texture and shape. The initialisation of the positioning and creation can be found in the NewCup method of the **Basic Thing 3D STARTS HERE.cpp** file.

- Unlike the Things, the NewCup method sets the file that is used by the object that is to be loaded. This links to the Object3D.cpp file and the setObjMeshFromFile method within it.

  This method obtains the information from the .obj file, within a structure that is returned from the LoadMesh method in the **ObjectLoader.cpp** file. Once this is done, the creation of the shaders (Create method for Object3D.cpp) determines the vertex and index buffers as well as the triangles that are created.

- Vertices, indices and vertex normals are mapped correctly (or so I believe), however the vertex textures do not work correctly – the variable information is loaded correctly into the structure, but there are conflicts with other shader's in the program when it comes to mapping the correct colours/textures.

- Allen Sherrod's and Wendy Jones's Beginning DirectX11 Game Programming book was used to understand the concepts behind the creation of an object loader, alongside the sample code provided in the application "start of OBJ loader", and making the structures work with the buffers took a long amount of time to get right – although more work is still required.

1.3 Evaluation

I was hopeful about creating an application that was impressive both from a programmatic and creative viewpoint. In terms of the changes that I would have made to the prototype, I would have liked to have had a beach, houses and boats but unfortunately I did not have time to add these. Additionally, if I had made the zebras eat grass by bobbing their heads backwards and forwards, I believe it would have looked both quirky and impressive.

Completing the object loader would have been beneficial too, although I feel as if getting the texture of the cup correct was an achievement in itself, based on the amount of days it took to get it to that point – admittedly, most of the time was spent trying to make the cup actually appear on the screen.

I am happy with the progress that I have made so far. If I had kept the non-object orientated implementation that I had constructed and provided that as my submission, it may have been cleaner in terms of memory de-allocations, but I would have learnt less as a result – and knowledge is more important to me than having full screen or vertical sync working perfectly.