Soft 323
Assignment 2: Gary the Cyborg

0.0 Introduction and sales pitch

“Gary the Cyborg” is an escape from a stranded island game. Where the player must destroy all of the zombies and ensure survival by collecting all of the resource crates. This product would be intended for developers who are interested in learning the concepts behind some of the areas that have been explored (terrain rendering, bullet creation and collisions between objects), or individuals who like to go around shooting zombies and collecting crates as a cyborg called Gary.

It is collectively meant to be a submission that explores terrain rendering and collision detection, but fun and quirky user goals have been added to make things livelier. The goals mainly focus on collecting items and shooting enemies.

The work comprising the second assignment submission is a continuation from the first; meaning that I have added to the work provided for the “flying thing and object loader” submission and the additions have been listed in section 1.2. In the earliest form, the version of Basic Thing 3D that is used is “Follow Cam” – from the beginning of the first term.

A video of the application has been placed online purely for reference, with everything running as intended; to view it please click here: https://www.youtube.com/watch?v=PUdr_hD2qsE – the video is unlisted, so it can only be viewed by those that are sent this link.

Important: When the application runs, if it turns out to be too fast, then to quickly solve the problem, refer to the OnFrameMove method of Basic Thing Starts Here.cpp and adjust the speedIncrease variable at the start of the method accordingly; this shouldn’t be a massive problem. The video provided above shows the application working as it was intended, so in the worst case scenario I have added it to provide a means of clarity into how everything operated during the development process.

1.0 What I started with; where I got it; what work is my own

1.1 What I started with and where I got it:

Referring back to the introduction, this work is an expansion on the first assignments submission for this module. Tutorials from braynzarSoft.net have been used to aid with specific parts of the object loader and the height map (braynarSoft, 2015). Frank Lunas book has been helpful in understanding the concepts behind terrain rendering and frustum culling (Luna, 2012). The features that revolve around the creation of bullets are unaided.

The culling that has been used is from the Basic Thing 3D tutorial that is dedicated to demonstrating how to not render objects that are outside of the direction that the main player is pointing.

The models that are used in this submission are all from free websites that provide free models. All links to the models and tutorials used can be found in the references/bibliography section.

1.2 Independent learning (what work is my own):

As a short summary, the parts of the assignment that have required **independent learning** are:

- Height mapping has been implemented, and uses the greyscale values of a bitmap image to determine the surface height of terrain. Colouring for the terrain has been conducted by taking another bitmap image and extracting the red, green and blue values and then using a pixel shader to map the colour correctly. Around five days was spent implementing this.
- The object loader from the first assignment has been expanded. Another five days was spent getting my head around how all of this worked:
    - Objects that require different types of faces can be loaded (for example a position and texture, but not a normal).
    - Material subsets and textures – extraction and use of variables from within an mtl file.
- The player can move around the terrain and along the contours.
- The collection of resources (boxes that are placed on the map) using collision detection between two points.
- Bullets can be fired using the “ctrl” button.
- Zombies follow the main player around, however I have left the “if it touches the player s/he dies part out; as the frame rate and speed tends to differ between machines; on my laptop it is 130fps and on my desktop it is 800fps – and the zombies speed up considerably in the second case. It would be hard to test the application if the zombies are extremely fast.
- When a bullet is fired and touches a zombie, the zombie is destroyed and the counter is decreased.
- User goals:
    - Collect all of the crates
    - Destroy all of the zombies

2.0 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).

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

How to work it:
Open the application by double clicking the solution file and pressing the run button (in either release or debug mode(s)).

Controls:

Ctrl: Shoot a bullet at a zombie. If it collides then the zombie will be destroyed.

X button: zoom's the camera outwards.

Z button: zoom's the camera inwards.

4.0 From a programmer's point of view

For each of the features that required independent learning**, the flow will start from "Basic Thing3D Starts Here.cpp"**; so for each of the sub-sections, it should be assumed that the description starts from within that class.

Each object type (Height Map, Object3D, Thing3D) derives positional/matrix values from the parent class "Thing3DAbstract".

4.1 Height mapping:

- This is created during the applications launch.
- Refer to the "NewFloor" method in "Basic Thing Starts Here.cpp".
- The flow of the method through the HeightMap.cpp class is as follows:

- The height map is loaded, using a bitmap file. This translates to extracting the positional information from the bitmap, by reading the greyscale values and calculating the information accordingly. The X and Z values are defined by the size of the inner and outer loops during this process, whereas the Y values are collected from the specific file channel and are scaled accordingly.

  After this, the vertex normals and texture coordinates are calculated and are stored, along with the positional information.

  Finally, the index and vertex buffers are created and use the information collected from the greyscale bitmap to create the triangles that come together to form the terrain.

- The colour map is loaded next (HeightMap.ccp -> LoadColourMap). This takes the red, green and blue values from a non-greyscale bitmap image and maps them accordingly. This uses a specific pixel and vertex shader and requires that an additional element is added to the input layout, which it has been (The input layout is in Thing3DVS_PS_Pair.h).
- Unfortunately, inverted mapping is required in order to make the main player go up and down the terrain in the correct way (in my case). The terrain is invisible unless flipped over by 180 degrees. Rotating the HeightMap by 180 degrees results in the vertex and index buffer information being rotated, which fixes this issue. However, the issue that arises from this is that the structure positional variables (found in HeightMap.h) do not rotate alongside this, so when applying the maths that determines the Y position of the player, it is also opposite, unless the information from an inverted bitmap is used instead for this purpose.

4.2 Object Loader:

- In “Basic Thing 3D Starts Here”, the methods that create multiple vectors/instances of Object3Ds (those that use .obj files) are: NewCyborg, NewCyborgGun, NewMonsters, NewBuildings, NewResources and NewBullets.
- Each method creates one or more instances of Object3D.
- The Object Loader programmatic flow is found in “ObjectLoader.cpp”. The object file is read and the triangle vertex information is stored, along with each subsets start and end index point and subsequent texture.

4.3 Moving around the terrain:

- In “Basic Thing3D Starts Here”, in the onD3D11FrameRender method, the height of the player relative to the height map is stored in the playerRelativeToHM variable. This is calculated by passing the X and Z position of the main player to the getHeight method of the HeightMap.ccp class.
- Where the player is relative to the terrain is fairly easy. The player starts at positions 0, 0, 0 (x, y and z), and the Height Map also does. The X and Z variables can be

used to determine the index once they have been rounded down using floorf, because the player's position can be used to determine the correct index.

- It is also worth mentioning that the zombies also use this method to determine the correct terrain position.

4.4 Collisions between objects

- The terrain has multiple objects on it. When the player interacts with a box, or shoots a bullet at a zombie, the collision happens when the X and Z positions get to a point where a difference can be calculated. A reaction occurs if the calculation returns that the two objects are in the same position.

4.5 Bullets

- It is worth mentioning that a bullet is an Object3D (as it is created by loading an .obj file).
- Bullets can be fired using the "ctrl" button. They are created in the NewBullets method and initially twenty are created during load, and not whilst the application is "running".
- The class responsible for managing the state of the bullets is "Bullet3D", which comprises of a set of methods that can be used to initialise bullets, or retrieve information, and a structure to contain variables. For each bullet (which is an Object3D), there is a starting x and z position that accompany Booleans which determine whether the bullet is visible or not.
- A bullet is made active when it is fired (pressing the ctrl button) and only one bullet can be made at a time (meaning that twenty bullets are not fired at once). Please refer to OnFrameMove to see the following:
    - The bullets are cycled through when the "ctrl" button is pressed. If a bullet is found that is not active, then it is set to active and the for loop no longer continues. If all bullets are active, it means that all bullets have been used up.
    - Where one or move bullets are active, they move forward instead of being set.
    - If a bullets starting X and Z position has a difference of twenty or more, then they are declared "invisible" and are subsequently not rendered anymore.
    - Only bullets that are declared "active" and "visible" are rendered.

5.0 Software engineering issues

Making the project very object-orientated has allowed for the development to be much faster than it would have been if it was procedural. Optimisations were made wherever possible to prevent computations that were not necessary. In this context, an example would be to not render bullets unless they are visible alongside only creating them before the application runs (during launch).

C++ is a very hard language to become good at. Wherever possible I have tried to pass by reference, but this can sometimes be the cause of additional memory issues. There are probably a number of changes that I can make to specific classes to make them more optimal.

The mind-set that I had during the implementation of the features that were shown during the demonstration has been to - wherever possible - create all OO-objects whilst the application is loading (before it is in its playable state). This ensures that there are no freezes/halts caused during runtime that are a consequence of creating new objects.

Culling has been implemented to increase performance and results in not rendering objects that are outside of the directional view of the player. This has been adopted from one of the Basic Thing 3D examples.

6.0 Evaluation

This has probably been one of the hardest modules that I have undertaken during this degree. There have been many times where ten or twelve hours in a specific day had passed and I would still be working hard to fix a problem. Through stubbornness and hard-work I had managed to implement height maps, instances of objects in an .obj format, bullets, enemies and a resource collection feature. I am happy with the submission because of the amount of effort that I put into it and also what I had achieved and learnt during the entire process.

In the summer or future I am going to look into animating object files by applying weights to certain parts of the body in order to understand how animations work, in addition to understanding how collisions can only remove one specific part of an object, instead of the whole thing. I want to look into understanding how to load in animated files too, and get my head completely around frustum culling.

7.0 References

Luna, F., (2012). *Introduction to 3D Game Programming with Directx 11*.Boston: Mercury Learning and Information.

tf3dm.com (2015). *Free 3d Models.* [online]. Available at: http://tf3dm.com/. Accessed: 17th March 2015.

TurboSquid (2015). *TurboSquid*. [online]. Available at: http://www.turbosquid.com/Search/3D-Models/free/obj. Accessed: 17th March 2015.

8.0 Bibliography

3dregenerator (2015). *Barrack*. [online]. Available at: http://tf3dm.com/3d-model/barrack-48931.html. Accessed: [17th March 2015].

3dregenerator (2015). *Suitcase_Box*. [online]. Available at: http://tf3dm.com/3d-model/suitcase-box-49962.html. Accessed: [17th March 2015].

Acebedo, E (2011). *30 Free High Resolution Grass Textures*. Available at: h ttp://naldzgraphics.net/freebies/30-free-high-resolution-grass-textures/. Accessed: [10th January 2015].

BraynzarSoft. (2015). *Lesson 21: Direct3D 11 Loading Static Models (.obj format)*. [online]. Available at: http://braynzarsoft.net/index.php?p=D3D11OBJMODEL. Accessed: [18th March 2015].

BraynzarSoft. (2015). *Lesson 29: Direct3D 11 Heightmap(Terrain).* [online]. Available at: http://braynzarsoft.net/index.php?p=D3D11HM. Accessed: [18th March 2015].

Creative Commons (2014). *Attribution 3.0 unported*. Available at: https://creativecommons.org/licenses/by/3.0/. Accessed: [12th January 2015].

Meilleur, G., (2014). *45 Free Water Photoshop Patterns Textures.* [online]. Available at: http://www.seodesign.us/photoshop-patterns/45-free-water-photoshop-patterns-textures/. Accessed: [10th January 2015].

mrlegoboy (2015). *Bullet*. [online]. Available at: http://www.turbosquid.com/FullPreview/Index.cfm/ID/772463. Accessed: [17th March 2015].

Oosten, J. (2014). *Introduction to DirectX11*. [online]. Available at: http://3dgep.com/introduction-to-directx-11/. Accessed: [12th January 2015].

Perpetuum (2011). *Behind the Bots: Generating the perpetuum maps.* [online]. Available at: http://blog.perpetuum-online.com/posts/2011-01-25-behind-the-bots-generating-the-perpetuum-maps/. Accessed: [17th March 2015].

Rastertek. (unknown). *Tutorial 2: Height Maps.* [online]. Available at: http://www.rastertek.com/tertut02.html. Accessed: [18th March 2015].

Ysup12 (2015). *Dead Space 3 Slasher Model*. [online]. Available at: http://tf3dm.com/3d-model/dead-space-3-slasher-61944.html. Accessed: [17th March 2015].

.