J.Pritchard U1661665

U1661665@unimail.hud.ac.uk

Studio 2 – a game scene in direct x 11

Term 1 assignment.

# Contents.

* **Studio2 – Term 1 assignment –A Game scene in DirectX 11.**
* **Design history.**
* **Assignment overview.**
  + Philosophy.
    - What I was attempting to achieve.
    - Software and languages used.
  + Common questions.
    - What is the game scene?
    - Why create this game scene?
    - What do I control?
* **Feature set.**
  + General features.
* **The game world.**
  + Rendering system.
    - Overview
    - 2D/3D rendering.
  + Camera.
    - Overview
  + Game engine.
    - Overview.
    - Collision detection.
  + Lighting models.
    - Overview.
* **The world layout.**
  + Overview.
    - The inner walls.
    - The floors and roof.
    - The door.
    - The window frame.
    - The table
    - The outer walls.
    - Skybox.
* **Musical scores and sound effects.**
  + Overview
* **Sources used.**
  + Overview.

# Design History.

This is a brief history of this document.

As it stands, this is the first and last version of this document. This was created at the end of development for the assignment to document my experiences gained throughout.

This will include things such as:

Difficulties overcome,

Techniques used,

Features attempted,

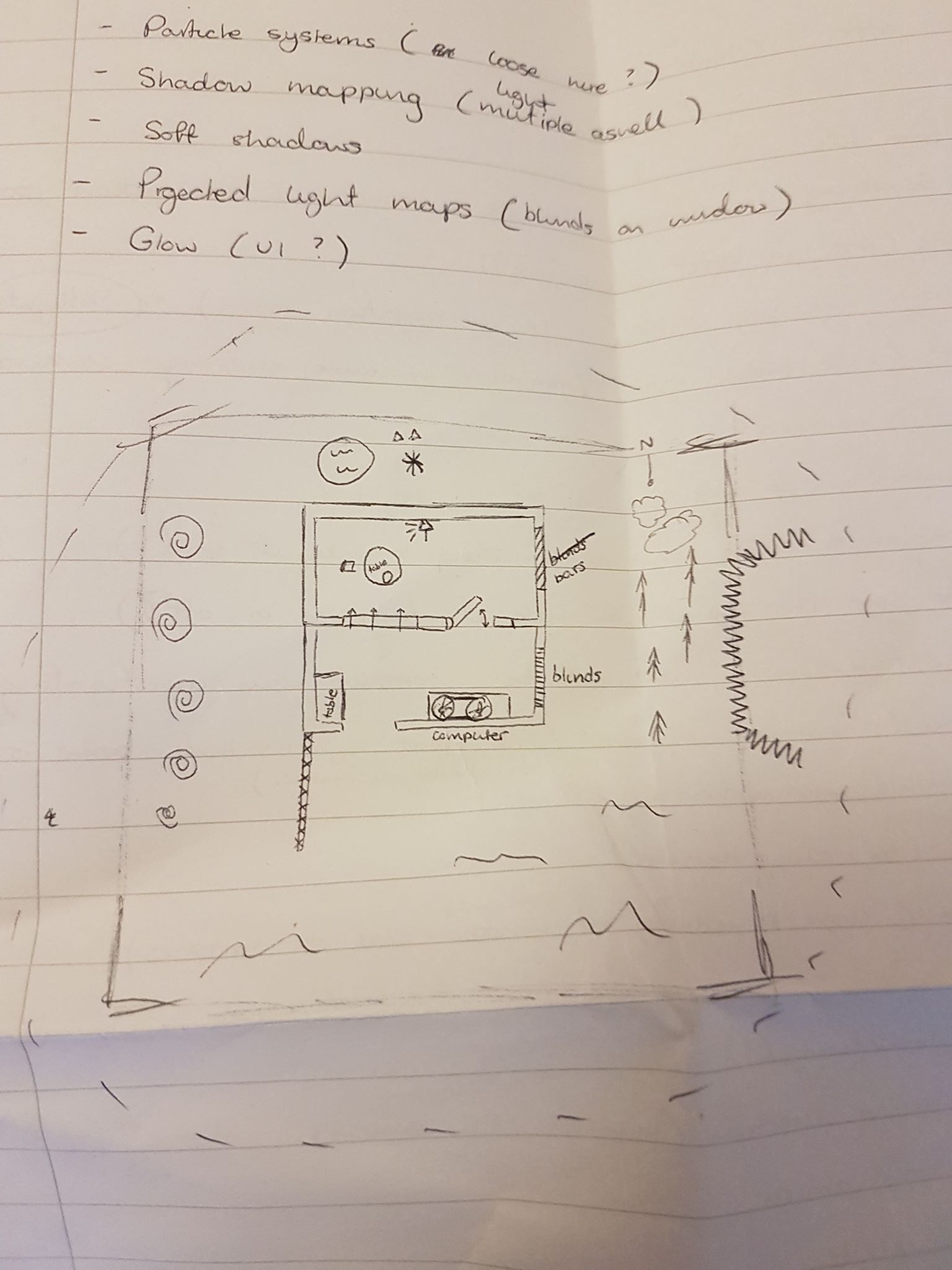
Lessons learnt,

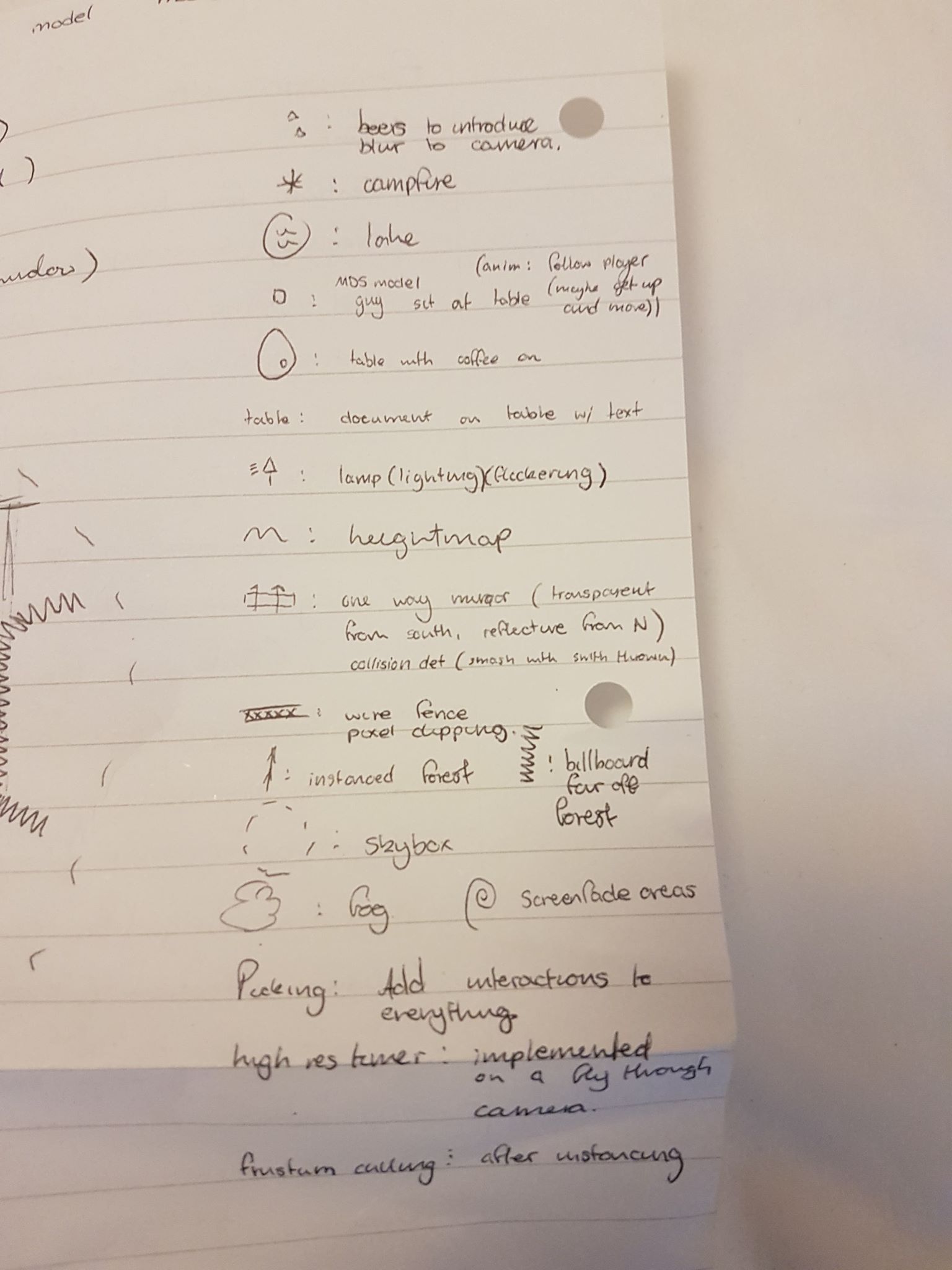
Sources of additional reading.

# Assignment overview.

## Philosophy.

### What I was attempting to achieve.

My game scene was intended to be an interrogation room in the middle of a forest. Not only would this allow me to show off multiple techniques of 3D rendering, as well as create an atmosphere that would make the game scene something greater than the sum of its parts. I looked at series of RasterTek and Brayzarsoft tutorials and for each technology discussed, came up with an idea of where this could fit into my game scene. I have attached a sketch of my potential scene below. 



Obviously, not everything planned here came to fruition, however I still thought it was important to have a rough idea of things I wanted to achieve.

### Software and languages used.

My assignment was written in DirectX 11 as per the specifications. A consequence of this is that it will only run on devices using the Windows operating system. Despite this, Understanding DirectX 11 is an important step to being able to develop game engines for Windows.

## Common questions.

### What is the game scene?

This game scene is an interrogation room. It contains a rendered interior consisting of 2 rooms separated by a wall holding a window and an opening and closing door. Inside the interrogation room side there is a table. The walls in the interrogation side are different to that on the observation side.

### Why create this game scene?

This game scene, as an assignment, is intended to be my introduction to programming in DirectX. I designed this game scene in particular because I believed I could incorporate most of the technologies covered in the RasterTek series into a believable scene.

### What do I control?

The player currently controls a partial FPS camera. There is no character associated with the camera, however if suspension of disbelief is your thing, then imagine that the camera is the viewpoint of an FBI interrogator with an extendable neck.

# Feature set.

## General features.

Controllable camera.

Ambient lighting.

Specular lighting.

Diffuse lighting.

Advanced object transformations.

Code efficiency.

Rendering efficiency.

Soundtrack.

# The game world.

## Rendering system.

### Overview.

My game scene uses a DirectX 11 rendering system. The general structure is based off the Assignment template given to us therefore the RasterTek tutorial series.

### 2D/3D rendering.

The code structure follows a basic DirectX 3D vertex processing pipeline, converting vertices in models to world space, the view space, projection, clipping/scaling and being rasterized. All of the conversions are done using matrix multiplications/manipulation. The matrices are never accessed directly, however are manipulated via provided DirectX library functions such as XMMatrixMultiply and XMMatrixRotationAxis.

All of these operations are done in the GraphicsClass, which is responsible for rendering objects through different shaders and processing movement input.

Instead of rendering every object in a single function, I decided to create functions for larger groups of objects within this GraphicsClass. Examples of these functions are ‘RenderDoor’ or ‘Render InnerWalls’. These allowed me greater speed of debugging as if something didn’t work correctly, I could isolate it to one of these functions and fix it there, without worrying if any other object I was rendering was causing the problems.

Where rending multiple objects one after the other are concerned, I tended to use for loops and use the iterator within the loop to manipulate the matrix translations using hard coded values. This allowed me to save huge amounts of code repetition by simply changing the values passed to these functions using mathematics.

Something I conceived towards the end of development on this assignment was the creation of a generic ‘PlaceModel’ or similarly named function that would take in values for transformation, rotation and scaling. These could be worked out ahead of time and passed into this function to further reduce code repletion. This became apparent because I was beginning to notice that every RenderXYZ function contained the same basic code, with different values being passed in. Minimizing this to a single line of code with only the parameters changing would have increased the speed at which I could place static models and debug them.

However, the above concept would have come with the disadvantage of having to work out the values ahead of time. This would have been difficult to do without the use of an in-engine editor which I did not have access to, instead relying on mostly trial and error to get the placement and scaling of objects correct. For some models which I created by myself in 3DS max I could use the values there to assist me with ballpark values. For the premade models I used from the internet however, this was not always possible.

## Camera.

### Overview

The camera is controlled via the WASD keys, left control and space bar, and the arrow keys.

|  |  |
| --- | --- |
| W | Move forwards |
| A | Strafe left. |
| S | Move backwards |
| D | Strafe right. |
| L Ctrl | Move vertically down |
| Space | Move vertically up. |
| Up | Look up. |
| Left | Look left. |
| Down | Look down. |
| Right | Look right. |

I was intending to model this after a basic FPS camera with the addition of moving vertically down and up to gain a better view of the scene.

There are three improvements I wanted to make to the system I developed. The first would have been using the mouse to control the looking around (rather than the arrow keys which feel rather unwieldy). The second would have been removing the ability to control height and instead use Space to jump or crouch. The third would have been allowing the shift button to make the camera move faster (a sprint function, if you will). These three improvements together would have created a camera controller very similar to most modern FPS games. I decided to focus on implementing other things however, as the current model works well for creating and editing the game scene I was tasked with creating.

## Game Engine.

### Overview.

For this assignment the game engine simply consists of the rendering component. There is no in-engine tracking of game objects or their components, and each object is treated completely individually.

I did however make an improvement to the code base which could be classed as an in-code game engine element. I created a modelHolder class to hold and initialize every Model used within the renderer. This was to avoid having this done within the GraphicsClass as I thought a separate class should be responsible for loading and providing references to modelClass objects. This had the advantage of speeding up adding models to the renderer as it was the same process and same way it had to be done each time.

Like what was experienced with the code repetition in the graphicsClass, I could have created a generic function for adding a model, and represented each of the models within the renderer within a class wide ModelClass array. This would have allowed me to add a new model to the modelHolder class slightly easier. The only disadvantage this method would have had would have been making it harder to read where each model was within the array. This could have been fixed using a public modelName enum or something of the like describing values to array positions.

### Collision detection.

I wanted to implement this into the camera system, where the player wouldn’t be able to go out of the bounds of the room. This would have allowed more realism to the scene however would have been done with hard coded values as developing a generic collision detection system for my renderer would have been far outside the scope of this assignment.

## Lighting model.

### Overview.

I decided to use ambient, specular and diffuse lighting for every single object in my scene. This allowed a degree of realism into my scene. The shine effect created on the door as it moves and on the floor is quite nice in my opinion. I wanted to include point lights and directional lights in my scene instead of a generic light as is currently used.

I would have lit the entire scene with lights and had some of them flickering. This would have created at atmospheric effect and showed off the technology of DirectX 11 quite well. Unfortunately, this never came to be and I had to settle for the aforementioned lighting.

# The world layout.

## Overview.

The game scene as a whole has already been discussed in this document.

### The inner walls.

The inner walls are a sci-fi wall model found online. They are modelled as a place and have a diffuse texture wrapped to them. This allows for more efficient rendering, as the backs and sides of the plane are never considered by the renderer, only the front facing vertices. This did make placing them a slightly more laborious task however, as small gaps between the planes had to be fixed accurately. To place these walls I used a series of for loops for each side of the room, with the wall containing the door and mirror using harder coded values to place the individual wall segments.

### The floors and roof.

The floors and roof are also planes and offer the same renderer efficiency discussed. The floors were easy to place and were done first. These were used as the basis for placing everything else in the world. The texture used also inadvertently allowed me a grid based pattern and assisted in placing the objects in their correct positions. Each square on the floor texture measured 25x25 units. The roofs were harder to implement and the rotation values made it difficult to get correct, however eventually these worked.

### The door.

This is comprised of two objects, the door frame and the door itself. These are textured separately, however due to the engine being used which does not support full texture unwrapping, the reverse side of the door has the texture reversed. Splitting this into 2 models allowed me to achieve the swinging door separate from the frame however. Each frame this was calculated by a series of steps.

1. Move the door away from the origin of the scene so that the edge of the door hits (0,0,0).
2. Rotate the door around the origin.
3. Calculate the difference in x and z position needed for this new rotation.
4. Move the door to its needed position, using the calculated difference in x and z.

This is a down to earth method for rotating around an arbitrary point of the model and works well enough to be believable. This took a considerable amount of time to work out.

### The window frame.

The window frame was originally one object, however I decided to split it into two objects to attempt to fix a texture stretching problem. I attempted to stretch textures in one direction and then wrap them to the other direction so that they would be shrunk to fit and look better. This concept didn’t quite work however, and would have been better achieved by repeating the texture in accordance with the ratio of width to height on the window frame objects. There was originally going to be a one way mirror contained within the window frame, using reflection technology on the interrogation side, and blending on the observation side. This never came to be unfortunately.

### The table.

The table is a simple object with a generic wood texture applied to it. If everything in my initial plan came to fruition then this table would have held lamps, a gun maybe or other smaller objects that would have added to the scene.

### The outer walls.

The outer walls are scaled cubes with the stone texture provided in the assignment template. I wanted to use the bumpmap texturing method for these objects however couldn’t get it to work. If I had managed this, it would have looked very nice with the ambient lighting used.

### Skybox.

I attempted to implement a skybox using cube mapping. The technique behind this is discussed in a tutorial on braynzar soft. I attempted for a week to implement this however this was before I realized the assignment template was more similar to the RasterTek framework. The braynzar soft technique would have used cube mapping on a texture cube to map a sphere object’s pixels to pixels on the texture cube. The skybox would have moved with the camera and had its z buffer disabled so that it was always drawn behind any other rendered object. The technology is all included within the source code, however implementing it never worked. I’ve left the code there so that perhaps a future revisit of this code once more familiar with DirectX 11 would allow me to see the errors I made in implementation.

# Musical scores and sound effects.

## Overview.

The game scene uses Direct sound to include the James bond theme song which plays as the viewer explores the scene. This would have been another method to add atmosphere to the scene. If I had a little bit more time with this mechanic I could have attempted 3D sound to have other sounds be present throughout the level and get louder the closer the user is to the object.

The audio format used is WAV and follows the red book standard of 2 channel 16 bit 41000Hz.

The SoundClass code is quite sophisticated and follows a series of checks to ensure that the music being played is of the correct standard and contains the correct headers.

I did encounter a problem with this code where an MP3 of the song converted to WAV included unwanted header data to be packaged, therefore invalidating it with the code. This was fixed by re-exporting the WAV with audacity and removing the unwanted header data however.

# Sources used.

## Overview.

RasterTek DirectX 11 tutorials:

<http://www.rastertek.com/>

BraynzarSoft DirectX 11 tutorials.

<https://www.braynzarsoft.net/viewtutorial/q16390-braynzar-soft-directx-11-tutorials>