James Moran

This report shows the development process for the <name-pending> DirectX application.

James Moran CGP600 AE2 – Individual Project Report

Contents

# Introduction

## Initial Development Actions

I started off by using an older Tutorial project (Tutorial 08 Exercise 01), as a basis for this assignment’s project. This is because I deemed this project the most advanced working version of the project (no discernible memory leaks), which is also stable. I went through the code, refactoring suitable functions/methods, as well as properties, into classes, with respects to which actions these methods/functions perform and which class would require the respective properties for such.

There is a global function (called in the scope of Main.cpp), that initialises the classes. When this function is called in the WinMain() function, assert() macro-calls are used to make sure the pointers to these classes, are valid. (cplusplus.com, 2000-2017)

For referring to pointers (not copying them), by certain classes, of which the pointer is that of another class,\*& is used for such (instead of \*\*), to call the functions of that pointer’s class. (Matthew Hoggan, 2017)

After refactoring all of the components of the project into their own classes, came the process of cleaning-up the project, by resolving as many warnings as possible.

First off, came the process of resolving macro-definition classes between winerror.h and dxgi.h. Although a process is listed on MSDN, to resolve this issue, I chose the answer that was not the accepted answer (with the second highest amount of up-votes), which simply involves adding $(WindowsSDK\_IncludePath) to the Include Directories section of VC++ Directories. This resolved the occurrence of these warning messages. (gradbot, 2012)

After resolving the issue noted in the above paragraph, I resolved an issue with DirectX Memory-Leaks, which I was made aware of, by D3D11 issuing warnings to me, via the output window. These appear after terminating the application, with messages similar to those listed in Appendix A: Fig. 1.

I resolved live objects not being cleaned-up (which is what these messages indicate), by following all of the steps of the tutorial, that is available from the same source as the quote that is noted, in Appendix A: Fig. 1. (Master Kenneth, 2014)

# Additions to Satisfy the Basic Requirements

After the initial development actions, came the process of adding to the project, in order to satisfy the basic requirements (listed on the assignment brief).

First off, came that of adding functionality to the project to load and draw .obj files (assets, exported from a 3D-Modeling package, such as Autodesk’s 3D Studio-Max).

After the above system had been implemented appropriately, came the product of a simple model to represent the Player’s ship, using 3DS Max 2017. Appendix A: Fig. 2 is the reference image that was used to create a static-mesh, to represent the Player’s ship. (Mike Celestino, 2015)

After going through the process to develop this static-mesh, then I was able to show it in the default scene of the game. Appendix A: Fig. 3 is an image of the ship static-mesh side-by-side with the reference image on a plane (render-image produced by 3DSMax). Appendix A: Fig. 4 shows the ship as one is able to see it in the default scene of the game.

## Requirement 2 Implementation

As the first requirement has already been satisfied (as this application is a 3D game, using Visual Studio as the IDE, with object-orientated C++ in combination with DirectX(11) for the project’s implementation), I have now considered this requirement, at this stage of the project’s development cycle.

The second requirement is: ‘The player should be able to move around an environment’. For this purpose, I would want the camera behind the Player, looking down at them for a 3rd person perspective, whilst following them. I would also have wanted to place a (simple) static obstacle in the scene, that can be used to verify that the Player is moving.

I started off the implementation required for this requirement, by setting up movement for the Player, in line with their Y-Rotation (Yaw), so they always move in the direction they are facing. After attempting various methods to set-up a third-person camera, that follows the Player (iedoc, 2015), I settled on a method for an ‘Arc-Camera’ that keeps distance with the Player and always faces them, no matter the direction the Player’s hover-tank is facing. (Allen Sherrod and Wendy Jones, 2012)

## Requirement 3 Implementation

With a third-person camera now moving with the Player as they move through the level, came that of fulfilling the third requirement. The third requirement is: ‘The environment should be quite large, and must have static and moving obstacles (e.g., blocks, statues, and rolling rocks). These must be models (though they can be simple if you like). Some of these objects should be able to be pushed by the player, and others should be collectable.’ I started off by going about the implementation of a StaticObstacle class, for static-obstacles in the game scene.

After this, came putting together the static-mesh for the static-obstacles in Scene0 (the first scene of the game). I used the reference image available under Appendix A: Fig. 5. to create this obstacle’s static-mesh. (© 1970-2017 Analytical Scientific, LTD.)

These can be seen in the default game-scene, with the Player’s ship, in Fig. 6. of Appendix A.

## Requirement 5 Implementation

With basic obstacles in the scene, that have lighting, came this requirement: ‘The player should collide with objects; either stopping for static objects, or pushing them if they are moveable, or pick them up if they are collectable.’. For this, I went about the implementation of a collision system for all GameObjects.

I implemented a basic bounding-sphere collision system, which would find the centre point of each object (for the centre point of the sphere), then find the furthest vertex from this centre point (which would become the radius of this sphere), that one could then use for collision checking using Pythagoras’s theorem (if the distance between the centre points of the spheres is less than that of the sum of each sphere’s radius, then a collision has occurred as per this system).

# Enhancements

## Scene-Manager

After setting-up a third-person camera to follow the Player as they move, I went about the implementation of a GameScene class, for the SceneManager to manage the active scenes of the game (not only certain GameObjects), as per the design laid out for the SceneManager, by John McGrath. (John McGrath and James Moran, 2017)

The GameScene would then manage any GameObjects that are part of that particular scene, but only when it is active.

# Appendix A: Figures

Figure 1: A quote from masterkenth.com, as an example of D3D11 Warning messages:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | D3D11 WARNING: Process is terminating. Using simple reporting. Please call ReportLiveObjects() at runtime for standard reporting. [ STATE\_CREATION WARNING #0: UNKNOWN]  D3D11 WARNING: Live Producer at 0x007464B0, Refcount: 4. [ STATE\_CREATION WARNING #0: UNKNOWN]  D3D11 WARNING:  Live Object at 0x007476F8, Refcount: 0. [ STATE\_CREATION WARNING #0: UNKNOWN]  D3D11 WARNING:  Live Object at 0x01A07580, Refcount: 0. [ STATE\_CREATION WARNING #0: UNKNOWN]  D3D11 WARNING:  Live Object at 0x01A07264, Refcount: 0. [ STATE\_CREATION WARNING #0: UNKNOWN]  D3D11 WARNING:  Live Object at 0x01A10ACC, Refcount: 0. [ STATE\_CREATION WARNING #0: UNKNOWN]  D3D11 WARNING:  Live Object at 0x01A11204, Refcount: 0. [ STATE\_CREATION WARNING #0: UNKNOWN]  D3D11 WARNING:  Live Object at 0x01A1295C, Refcount: 0. [ STATE\_CREATION WARNING #0: UNKNOWN]  D3D11 WARNING:  Live Object at 0x01A11944, Refcount: 0. [ STATE\_CREATION WARNING #0: UNKNOWN]  D3D11 WARNING:  Live Object at 0x01A11B60, Refcount: 0. [ STATE\_CREATION WARNING #0: UNKNOWN]  D3D11 WARNING:  Live Object at 0x01A120C4, Refcount: 0. [ STATE\_CREATION WARNING #0: UNKNOWN]  D3D11 WARNING:  Live Object at 0x01A133BC, Refcount: 0. [ STATE\_CREATION WARNING #0: UNKNOWN]  D3D11 WARNING:  Live Object at 0x01A1373C, Refcount: 0. [ STATE\_CREATION WARNING #0: UNKNOWN]  D3D11 WARNING:  Live Object at 0x01A14A2C, Refcount: 0. [ STATE\_CREATION WARNING #0: UNKNOWN]  D3D11 WARNING:  Live Object at 0x01A14CB4, Refcount: 0. [ STATE\_CREATION WARNING #0: UNKNOWN]  D3D11 WARNING:  Live Object at 0x01A153AC, Refcount: 0. [ STATE\_CREATION WARNING #0: UNKNOWN]  D3D11 WARNING:  Live Object at 0x01A16D7C, Refcount: 1. [ STATE\_CREATION WARNING #0: UNKNOWN]  D3D11 WARNING:  Live Object at 0x01A22B7C, Refcount: 1. [ STATE\_CREATION WARNING #0: UNKNOWN]  D3D11 WARNING:  Live Object at 0x0C17154C, Refcount: 1. [ STATE\_CREATION WARNING #0: UNKNOWN]  D3D11 WARNING:  Live Object at 0x01A26904, Refcount: 0. [ STATE\_CREATION WARNING #0: UNKNOWN]  D3D11 WARNING:  Live Object at 0x01A2A2CC, Refcount: 0. [ STATE\_CREATION WARNING #0: UNKNOWN]  D3D11 WARNING:  Live Object at 0x0C1687F4, Refcount: 0. [ STATE\_CREATION WARNING #0: UNKNOWN]  D3D11 WARNING:  Live Object at 0x0C16BE0C, Refcount: 0. [ STATE\_CREATION WARNING #0: UNKNOWN]  D3D11 WARNING:  Live Object at 0x01A1ADC4, Refcount: 0. [ STATE\_CREATION WARNING #0: UNKNOWN]  D3D11 WARNING: Live                         Object :     22 [ STATE\_CREATION WARNING #0: UNKNOWN]  DXGI WARNING: Live Producer at 0x006AE3A8, Refcount: 4. [ STATE\_CREATION WARNING #0: ]  DXGI WARNING:  Live Object at 0x006AEA38, Refcount: 2. [ STATE\_CREATION WARNING #0: ]  DXGI WARNING: Live                         Object :      1 [ STATE\_CREATION WARNING #0: ] |

(Master Kennth, 2014)

Figure 2: A top-down view of a Lego interpretation of the Millennium Falcon, used as a reference image to aid in the product of the static-mesh to represent the Player’s ship:



(Mike Celestino, 2015)

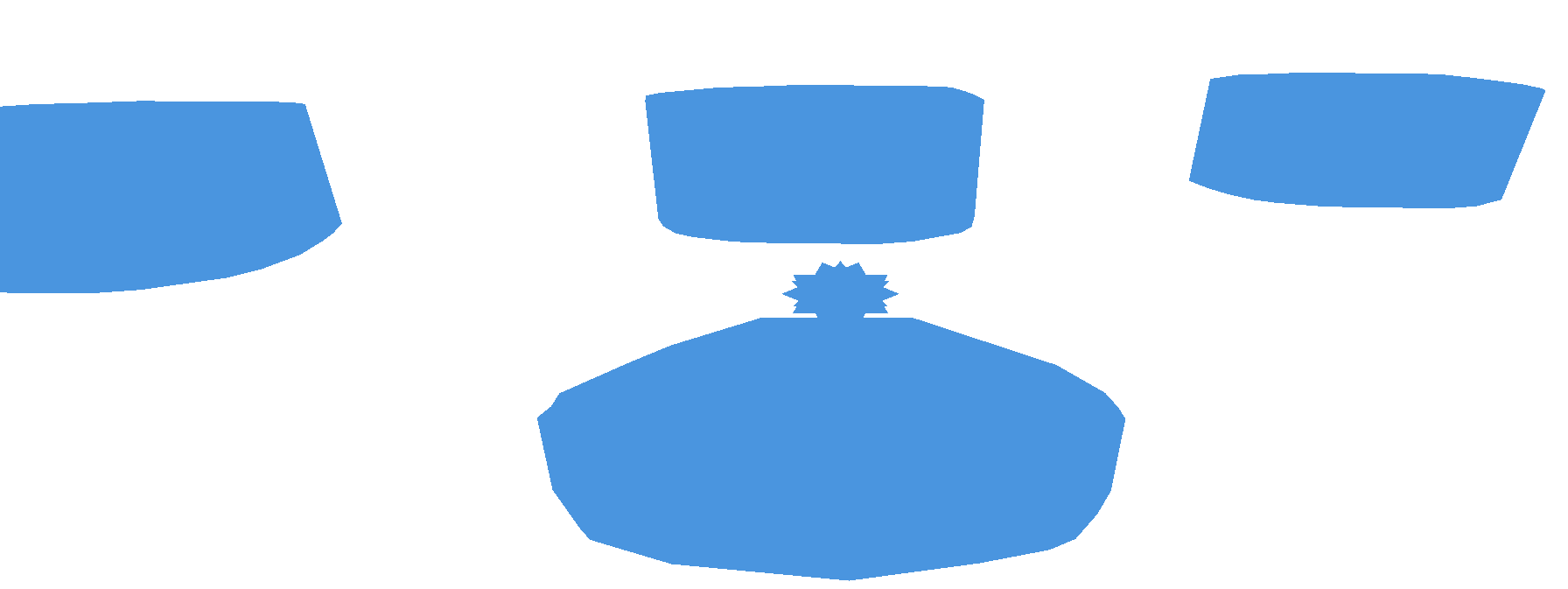
Figure 3: A top-down render-view from 3DSMax, of the ship reference image shown side-by-side with the produced ship’s static-mesh:

Figure 4: A perspective-view of the ship’s static-mesh in the default game scene:



Figure 5: Side-on View of an igneous rock, used a reference image to produce static-obstacles in the default game scene:

(© 1970-2017 Analytical Scientific, LTD.)

Figure 6: The Player’s hover-tank with a few rocks, in the default game-scene:

# Appendix B: Class Hierarchy and Class Diagrams

The project’s class hierarchy:



# References

1. © cplusplus.com, 2000-2017. *assert* [viewed on the 29/11/2017]. Available from: <http://www.cplusplus.com/reference/cassert/assert/>
2. Matthew Hoggan, 2012. *Reason to Pass a Pointer by Reference in C++?* [viewed on the 30/11/2017]. Available from: <https://stackoverflow.com/questions/10240161/reason-to-pass-a-pointer-by-reference-in-c>
3. gradbot, 2012, *Macro definition clash between directx headers and winerror.h* [viewed on the 07/12/2017]. Available from: https://stackoverflow.com/questions/12663091/macro-definition-clash-between-directx-headers-and-winerror-h
4. Master Kenneth, 2014, DirectX Leak Debugging [viewed on the 07/12/2017]. Available from: <http://masterkenth.com/directx-leak-debugging/>
5. Mike Celestino, 2015. *Lego Millennium Falcon Top-Down View* [digital image] [viewed on the 09/12/2017]. Available from: http://www.insidethemagic.net/2015/12/enormous-custom-lego-millennium-falcon-proves-star-wars-fans-dedication-attention-to-detail/
6. iedoc, 2015. 32. Simple 3rd Person Camera [Viewed on the 24/12/2017]. Available from: <https://www.braynzarsoft.net/viewtutorial/q16390-32-simple-3rd-person-camera>
7. Allen Sherrod and Wendy Jones, 2012. *Beginning DirectX® 11 Game Programming.* Boston, MA: Course Technology PTR
8. John McGrath and James Moran, 2017. *CGP600 Advanced Games Programming AE1 – Group Project.* Southampton: Southampton Solent University.
9. © 1970-2017 Analytical Scientific, LTD, 2017. Pumice Rock [Digital Image] [Viewed on the 26/12/2017]. Available from: <https://www.analyticalsci.com/store/p432/Pumice_Rock.html>