**Graphics Report**

*Luke Whiting*

**Installation instructions**

Using the system requires either a precompiled .exe file or access to the source code and a compiler (easiest to use is exists within Visual Studio).

Once you have acquired one of the prerequisites above you can run the files either by double-clicking the .exe or by pressing f5 when in visual studio with the project open.

Controls:

Ship: Debug:

Forward – W Enable only SpotLighting – F3

Backward – S Enable only DirectionalLighting – F4

Turn Left – A Enable only PointLighting – F5

Turn Right – D Enable all Lighting– F6

Camera: Enable wireFrame – F1

Forward – W Disable wireFrame – F2

Backward – S

Strafe Left – A

Strafe Right – D

General:

Switch to camera1 – 1

Switch to dynamic camera – 2

Switch to Debug Camera – 9

Set dynamic Camera position and rotation – 8

**Description of graphics features and architecture of the application**

In the application, the architecture is very object-oriented. Most core functions of the application are abstracted to help with readability and modularity. The main application functions as a modular game or application sitting atop a graphics engine, with other code abstracted behind it.

The current application can handle rendering, lighting and manipulating mesh’s easily. There are many systems that allow custom mesh’s to be loaded into the application and lighting and manipulation are handled automatically.

Starting from the start, the application class currently handles all the DirectX, direct3D and device (winAPI) initialization. Then it moves onto setting up the default camera object with its hard-coded values this is then passed onto the cameraManager which handles switching the cameras if there is more than one in the scene. The lighting objects are next and get set to hard coded values as well, these are also passed to a manager to help with toggling their activity. The program then moves onto loading the objects using the mesh class and puts them into a vector to be drawn later. After this it loops out of the application.cpp and checks the winAPI to see if any inputs have been given and if there are any, they get processed and it goes back into the draw and update loop that repeats until the program closes. Inside the update we currently have a timer and update the individual mesh’s generated by the mesh class to how we’d like them to be displayed in the world. In the draw we setup the shaders with all the buffers and resources that where defined in the DirectX initialisation and set the lighting then we draw the mesh’s to the screen using a for loop through the vector mentioned before.

Moving onto the mesh class, this class is one of the main classes in the application and handles everything mesh and object related. The mesh class consists of a super-class that has generic virtual functions inside that define drawing and updating. This is then inherited in all the objects sub-classes that then change some of the update or loading code to fit their specific object. Objects generate their mesh’s in their constructor using the OBJLoader provided and allow for a texture path to be passed in to allow the DDSTextureLoader to load its texture into the local TextureResource used later in draw. Each class has its own local variables that hold its MeshData, its worldMatrix and the TextureResourceView that is used for drawing specific textures on different objects.

Next is the Light Class which is much smaller currently than the mesh class due to it being incomplete. The light class functions similarly to the mesh class, whereas it handles everything to do with lights and lighting, it has a super-class structure exactly like the mesh class, where the sub-classes inherit from the super-class and change a few functions to fit their needs. The current only light implemented in this class is the directional Light, this light works to light our scene in a very basic way showing all the attributes of the different lighting types (ambient, diffuse and specular). This light takes some arguments in its constructor to change the light direction and specular intensity however little is changed in this as later the material values will most likely be abstracted to another struct or class.

Finally, there is the Camera Class, which is the newest class in the project, to avoid sounding like a stuck record this class has the same structure as the previous ones, however with a key difference. This class has a lot of helper functions that are only found within the super-class declaration since these helper functions are not going to be updated by each camera and will help in keeping each sub-class a little cleaner. The current sub-classes of Camera are; LookToCam and LookAtCam. Both these cams function very similarly but have key differences, one being that these cams either look to a location or centre and look at a location, these cams can do different things, the LookToCam is used for the DebugCamera since it can then use the LookTo function to look to a direction and that direction be influenced by the mouse position. The LookAtCamera currently isn’t used but will be used for more 3D modelling type cameras where the camera will orbit around a central point most likely an object to help with debugging or general viewing. These sub-classes break down into the debugCamera, StaticCamera and firstPersonCamera.

There are also some classes that help manage other classes. These include, the cameraManager and the lightManager. These do what they say on the tin, they manage all of the assets in the scene allowing them to be toggled, switched between and manipulated easily without duplicating code everywhere and also allows a easy way to check the current state if different areas of the engine. For instance, you can check which camera is currently selected using the cameraManager and you can see what types of lights are being enabled in the lightManager. You can also change these easily within the class using functions so switching them on the fly is very easy.

**Class Diagram**

**Test Documentation**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Description** | **Expected Result** | **Observed Result** | **Actions** | **Date** |
| Application Window Opens | The window opens as usual with no graphical errors | The window opens with no problems | N/A | **7/12/19** |
| DirectX and Direct3D initialize | DirectX and Direct3D initialize properly and do not cause fatal errors | All DirectX functions work properly and do not cause the application to close | N/A | **7/12/19** |
| Rendering (Back-face culling) | Back face culling will work properly, culling every face that is behind another face. | Back face culling works as normal | N/A | **8/12/19** |
| Toggling (Back-face culling) | Back face culling can be toggled on and off when using wireframe mode. | Back face culling can be turned off but will not come back on. | Made sure to re-enable back-face culling once back in normal viewing mode. | **8/12/19** |
| Rendering (Lighting) | Lighting illuminates’ proper areas and doesn’t show any graphical glitches | Lighting works properly, having specular reflections that is in line with its direction etc. | N/A | **9/12/19** |
| Rendering (Multi-Lights) | Multiple-Lights render properly and do not show any graphical glitches | Multiple-Lights work properly, however some graphics glitches can happen when spotlights are positioned perpendicular to a face. | Moved spotlights to not be fully facing down or perpendicular at a face. | **9/12/19** |
| Movement (speed) | The movement of the boat is constant and goes in a set direction | The movement of the boat constantly speed-up until seemingly light speed. | The movement speed is capped to deny light speed entry. | **9/12/19** |
| Movement (Rotation) | The movement of the boat should be in relation to where it is pointing. | The movement of the boat does not move correctly along its facing axis. | Using trig, got the forward axis and allowed for it to move along a rotated axis. | **10/12/19** |
| Mesh (Loading) | Mesh’s can be loaded correctly to be displayed in the scene. | All the objects in the scene are loaded correctly, can also load custom models. | N/A | **11/12/19** |
| Mesh (Movement) | Each mesh can be moved and rotation individually | Each mesh can be moved and rotation individually with different rotations etc. | N/A | **11/12/19** |

**References:**

Waves used in program: <https://catlikecoding.com/unity/tutorials/flow/waves/>