Design Doc (DirectX Application)

14/11/19 Luke Whiting

**Overview**

This design outlines what the current state and the future vision for the graphics application built in directX11. This document will outline in a technical light was is currently developed for the application at the time of writing and what features will need to be implemented in the future to achieve the envisioned future application.

**Context**

This project is being done to outline skill and prowess using graphics api’s, namely DirectX and to show how a graphical engine could be constructed from a framework using minimal external libraries and featuring adequate features for viewing (lighting, textures, materials, etc). This project will improve my skill as a programmer and help widen my knowledge of graphics api’s and the techniques and nuances that come with working with them.

**Goals**

Goals:

* Having a solid graphics frontend (Lighting, texturing, materials)
* Having a good interaction layer (Mouse input and picking to allow for user interaction)
* Having an efficient program (No unnecessary lag on low overhead code and making high overhead code as efficient as possible(shader etc))

Non-Goals:

* Making GTA 6 is directX11
* Making a full game engine out of this framework
* Making a perfect application
* Making anything that has real-time raytracing
* Making unreal engine 5

**Existing Solution**

The existing solution consists of a lot of classes, I am currently in the process of splitting the supplementary classes from the DirectX framework and making them interchangeable within the code and implementing new features which will allow for better representation of 3D objects in the space displayed (e.g. point lights and advanced texturing).

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. The lighting objects are next and get set to hard coded values as well it 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 a object to help with debugging or general viewing.

To conclude, there are helper Structure classes and headers to help with include and general struct management these hold: SimpleVertex (used for vertices), DirectionalLight, ConstantBuffer and Vector3 structs for later use. There is also some include helpers like the lightingIncludes and MeshIncludes which help organise the includes for these inherited classes.

**Future Solution**

The future application would be building on the current iteration by adding more advanced lighting and texturing techniques to the project.

These lighting and texturing techniques include, adding new point and spot lights to the Light class allowing there to be different types of illumination in a scene. Texture filtering to allow for texture to look better up close and at sharp view angles (e.g. Anisotropic Filtering). Building on this implementing new texture techniques to aid lighting and other calculations for instance normal mapping and bump mapping giving textures 3D properties and allowing the lighting to be calculated for the texture as well as the model underneath giving more depth.

On-top of this there should be more functionality to the engine part of the application. Adding the ability to properly debug with full classes dedicated to it (for printing to screen and picking objects on the screen). Also, making smaller but still as good Quality of life (QOL) changes to how the window is created and handled, adding the ability to change the resolution on the fly and going Fullscreen without the pixels stretching. Locking the mouse to the screen to help with movement whilst in debug camera aiding other areas too.

Furthermore, adding more camera functionality would advantageous as having a functioning first person camera as well as having an object focused model camera to inspect models with would help debugging and overall feel once the engine is used for a demo or game.

Finally, I would start to stop work on the mainly engine functionality and start to build out the demo slowly. Adding the game objects that would be used in this demo and positioning them in the scene to how id like, adding performance improvements and better UI to help with debugging the demo phase and in general help with giving more interaction in the demo.