Feedback - Unity Application

Initial feedback given was focused on the camera not suiting the application (*at the time this was Unity’s Cinemachine free look camera*), sluggishness of the character’s movement and the lack of responsiveness when the jump button was pressed.

## Character Movement

The character’s sluggishness was quickly fixed by setting the animation speed to 1.2x. Root motion was used to move the character’s transform based on the animation’s translation, so no further changes were needed.

## Camera

Swapping from Cinemachine’s free-look camera to third-person virtual camera helped tremendously, although this removed the function of looking around the character using mouse input - a script was written to add this functionality and the overall feel of the camera was a significant improvement.

Evaluation of Tech - OpenGL App

## Dependencies

Assimp was a pain to get set up initially, converting the project generation from CMake to Premake with many options. The library itself is well documented and fairly easy to use; there are also many tutorials on the library.

GLFW is used as the library for handling application windows and processing input from the OS; many resources are available and the setup was very straightforward.

## Forward & Deferred Rendering

Once some abstraction layers were written for framebuffers and render passes, implementing deferred rendering was very simple. Issues that arose between the mesh and lighting passes were easily found thanks to the help of the [RenderDoc](https://renderdoc.org/) and [NVidia NSight](https://developer.nvidia.com/tools-overview) applications.

## Shadow Mapping

Basic shadow mapping was placed into the engine somewhat late in the project, towards the end of the graphics implementations so some refactoring was required.

Cascaded shadow mapping would work at times, but I failed to scale it up for many light sources. I came very close with an implementation using a 2D texture array for the shadow maps and a geometry shader to render many meshes per shadow pass, but with the limitation of uniform buffers another option was needed for passing per-light information to the GPU; storage buffer objects were the closest implementation for lighting data storage, but even with manual offsets and debugging with NVidia NSight to match the data offsets, the data did not want to map correctly between the CPU and GPU - I am unsure at this time whether this may have been due to differences in struct alignment, internal storage of glm’s data, or another factor.

# References

* LearnOpenGL – Joey De Vries - [Website](https://learnopengl.com/)
* Real Time Rendering Resources – [Website / Book](http://www.realtimerendering.com/)
* OGLDev – Etay Meiri - [Website](https://ogldev.org/)
* OpenGL Tutorial - [Website](http://www.opengl-tutorial.org/)