horizontal line

**Loghin Vlad-Andrei**

An 3, Grupa B6, UAIC FII

Game Design - Project Proposal

Game Engine implemented in Vulkan API & C++

**25th February 2021**

# OVERVIEW

The goal of the project is to implement a functional, cross-platform game engine using the Vulkan API for 3D graphics acceleration. The project has been started early 2020, going through several APIs in early development before establishing on the currently used API. In its current state, the engine is capable of rendering independent 3d objects, controlling them in space, applying textures on objects, and using pre-compiled shaders. Latest in work feature was an Entity-Component system for Game Objects, similar to what can be seen in Unity3D or Unreal Engines. In addition, in a previous version, on a different API, lighting and shadows have been implemented with the use of accelerated graphics shaders ( OpenGL Core API ). Porting said features should not prove a challenge.

# GOALS

### Achieved Goals in Current Revision ( VK API )

1. Establish Required Dependencies for API SDK
2. Implement Project Template
3. Implement Base Vulkan Instance
4. Link Instance with Window Created through API ( GLFW API )
5. Implement a GPU debugging interface ( VK Validating Layers )
6. Implement a GPU device acquisition interface ( VK Physical / Logical Device )
7. Implement Required GPU - CPU bus interface data type ( VK Queue )
8. Implement a Shader Precompiler Module
   1. Implement a configurator for Shader Precompiler
9. Implement Vulkan Pipeline for image buffering ( and Swap Chain )
   1. Implement Swap Chain Recreation ( on Window Resize )
10. Implement Command Buffering ( Precompiled/Compiled Shader + Input data ( Vertices )
11. MILESTONE 1 : Draw a polygon
12. Implement Command Buffering additional data acquisition ( Colors for Fragment Shaders )
13. Implement efficient, easy to use interfaces for Vertex, Color, Texture Data send to GPU ( Buffers, Vertex Buffers, Index Buffers )
14. Implement Uniform Buffer Object interfaces - data shared between CPU and GPU - used in model translation / rotation / scaling.
15. MILESTONE 2 : Rotate Polygon
16. Implement Textures for Shaders ( using stb\_image from dependencies )
17. Implement interfaces for GPU memory pooling, index optimisation.
18. Implement interface for Depth Testing.
19. MILESTONE 3 : Draw a textured cube
20. MILESTONE 4 : Independent Objects. One moveable, second moves by itself
21. Texture Mipmapping - Smaller textures for far-away objects
22. Anisotropic Filtering - Shading

### Achieved Goals in Previous Revision ( OpenGL Core ) to be ported to Vulkan API

1. Model loading using library in dependencies ( ASSIMP ) - will be ported or Proprietary Loading Library shall be used.
2. Advanced Shader Features : Ambient, Diffusion, Specular Lighting.
3. Advanced Shader Features 2 : Global Lighting
4. Advanced Shader Features 3 : Lighting from a light-casting object ( ex. A Lamp )
   1. Object can be non-static ( moves )
5. Advanced Shader Features 4 : Lighting from PC ( Spotlight - ex. A Flashlight )
6. Advanced Shader Features : Shadows
7. PC Interaction : Camera Interaction with Mouse
8. PC Interaction : Basic Movement Interface

### Future Goals

1. Mesh. Model Classes
2. Entity - Component System for attributes existing as children of others.
3. UI - by dependency / Proprietary Library.
4. Base, Simple Editor : Qt6. ( Can Link Surface to Vulkan Drawing Surface )
5. Object Scripts -> Pipeline Recreation. Just in time Compilation.
6. Scene Logic : Saving Scene, Loading Scene with assets.
7. Optional - 3rd person PC perspective

### Far Future Goals

1. Dynamic Actor System - NPC.
2. AI Modeling
3. Networking Capabilities
4. Shader Features : HDR, DLSS, Raytracing

# SPECIFICATIONS

Technologies Used :

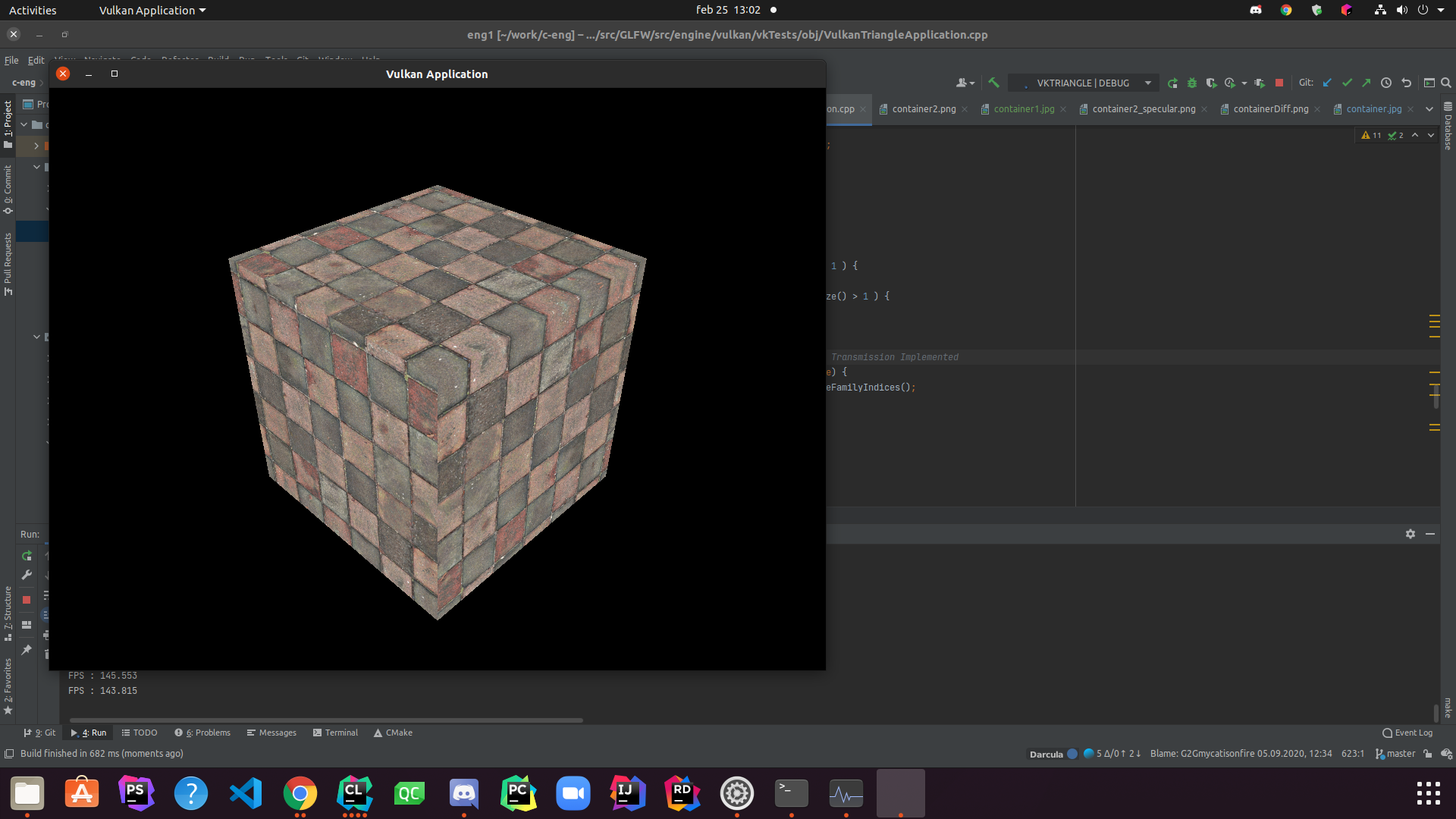
* C++ with partial c++2a support : gcc 9.3 or higher / clang 10.0 or higher / mingw 8.0 or higher / MVSC ( untested ) - Required
* CMake 3.10.2 or higher - Required
* Vulkan API 1.2 - Required
* GLFW for context acquisition - Required
* GLAD for GLFW OS Linkage Headers - Required
* GLM for fast graphical maths - Required
* Proprietary JSON library ( will be replaced by data structures library ) - Required by Precompiler
* Proprietary Shader Precompiler and Parser - Optional
* ASSIMP / Proprietary Object Loader - Optional

# WEEK 2 ACHIEVEMENTS

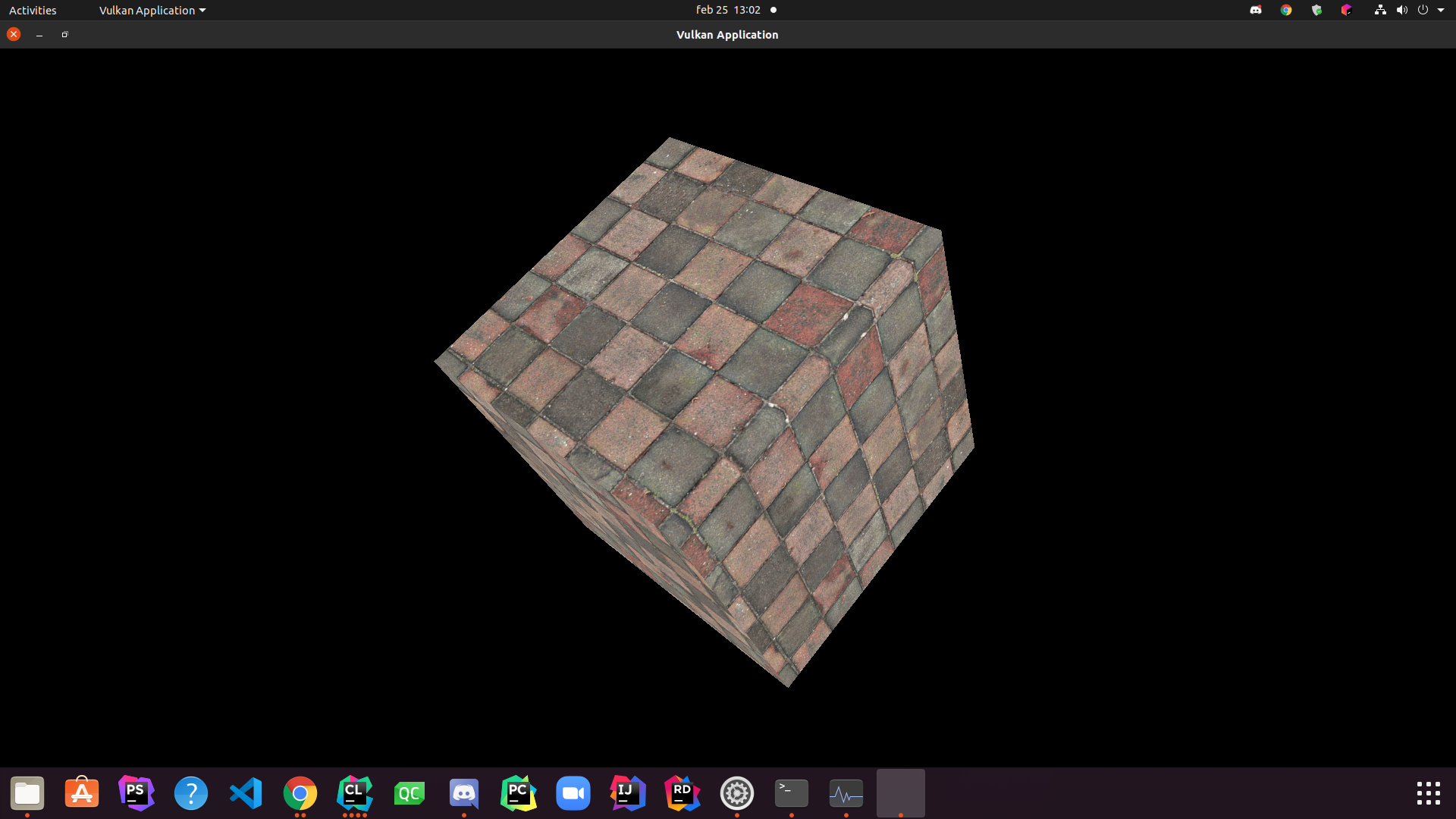
1. Started cleaning up proprietary dependencies for possible errors
2. Fixed Repository submodule linkage
3. Fixed Incompatibilities with recent API updates
4. Detected Error from Validation Layer from GPU Memory Pool Allocation caused by a Vulkan API Update
5. Re-Checked both Linux and Windows Compatibility
6. Cleaning Up Code and preparations for future commits on Entity-Component Implementation

Examples of current running state :

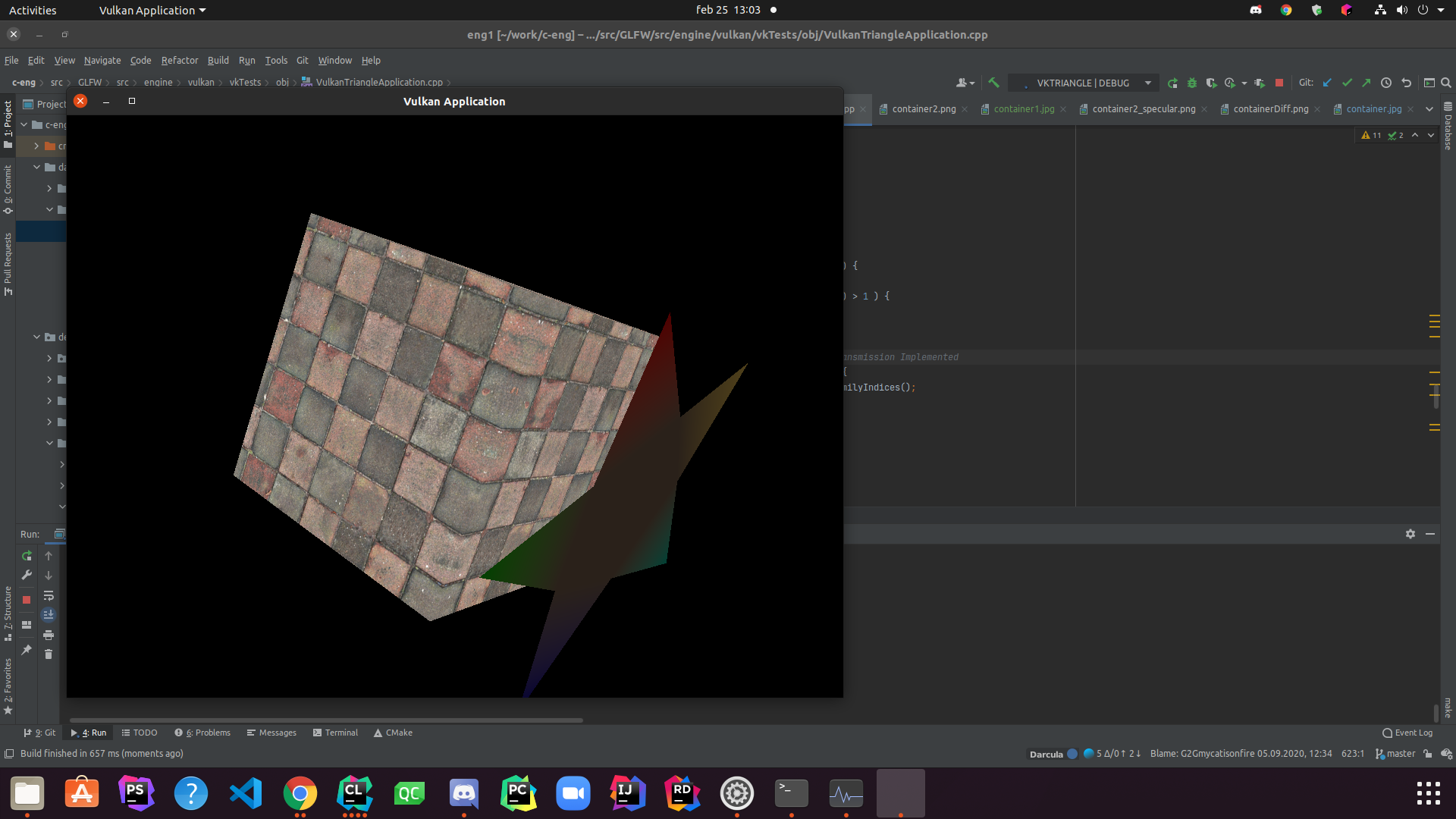
Loaded Model :



Rotated Model :



Independent Models : (2d Star for reference )



# REFERENCES

1. <https://github.com/LoghinVladDev/c-eng> - Engine Repository
2. <https://vulkan.lunarg.com/> - Vulkan LunarG page - developers of Vulkan. API Documentation requires account on site or on Khronos Group website
3. <https://www.glfw.org/> , <https://github.com/glfw/glfw> - GLFW home page and repository location
4. <https://github.com/Dav1dde/glad>, <https://glad.dav1d.de/> - GLAD repository location and web service for header generation
5. <https://github.com/g-truc/glm> - OpenGL Mathematics Library
6. <https://github.com/assimp/assimp> - Open Asset Import Library - Used in some cases, will be replaced with proprietary due to some compatibility issues
7. <https://github.com/nothings/stb/blob/master/stb_image.h> - Image Loading - May be replaced with Proprietary to avoid Licensing conflicts.