Essay

## Physically-based rendering

"Physically based shading means we approximate what light actually does as opposed to approximating what we intuitively think it should do" - Unreal 4 documentation.

## Engine architecture

The engine follows the philosophy that systems should specialize in doing one thing and doing it well. In this case, the engine should only handle rendering. That means loading files, handling the scene graph and providing the engine with a window are all things that the user is responsible for. The engine has its own model and material format in order to efficiently interpret and render the data, but the user is in charge of loading these files. The contents of these files are then sent to the engine as an array of bytes that are then appropriately interpreted by the engine.

The Mesh Tool is a command line tool that converts common model formats into the model format the engine uses. First, the model is loaded using the Assimp library. The tool then checks which vertex attributes the model is made up of. For example, some models only have vertex positions and normals, while other meshes may contain one or more sets of texture coordinates. Finally, the tool outputs a file in the custom model format. This format lets the engine know which vertex attributes are present in the model and how they are laid out in memory.

The Material Tool is also a command line tool. This tool converts an HLSL file into a pre-compiled custom material format to be read by the engine. First, the tool loads the HLSL file. The shader describes both the vertex and the pixel shader. Both of these shaders are compiled using the Direct3D interface, and then extra information is collected by using the Direct3D shader reflection interface. This interface for easy extraction of the data flow information of a shader. The Material Tool uses this for two purposes. The first thing is finding out the vertex attribute layout, which is written in the same way as in the Mesh Tool. The second thing is scanning the constant buffers the shader uses. These constant buffers are then written to a class in a C++ header file, using the appropriate data types for the members and including padding where necessary. This allows the user to include the header and use this class to intuitively edit members of a constant buffer. The binary compiled shaders and vertex attribute meta data are all output into the same file that the engine can read.

These tools are used at asset build time, which improves run-time performance and reduces dependencies. For example, the engine itself does not need to know how to read model files and how to scan their vertex attributes, because the Mesh Tool has already done this at asset build time and has saved this as meta data. Because the Material Tool has also put the vertex attribute meta data inside the custom material format, the engine can quickly compare a model and a material to see if they are compatible. This is all very efficient because most of the work has already been done before the program is even compiled.