Stage 1: Start the Renderer

Using vkguide.dev, start the rendering engine itself. This guide uses the following development tools:

- Vulkan SDK
- CMake
- SDL2
- GLM
- Dear IMGUI
- STB Image
- TinyOBJ Loader
- VkBootstrap
- VMA

We will follow this guide to give us loose structure for our Vulkan engine. To start, we will use all of the above libraries *without* Animal3D. This way, we can be absolutely sure that our rendering engine can load textures, meshes, and shaders, then output fully textured & shaded scenes before we try to integrate it with Animal.

Stage 2: Prepare Animal3D to allow Plugins to choose their rendering API

If we want to allow plugins to choose an API, we have to account for the fact that A3D initializes the render context for OpenGL automatically upon window creation. We will need to move render context/instance initialization to the plugin's DemoState loading sequence as the very first thing to load. This functionality will read from a plugin config file to determine the API to use; if the API isn't specified, we can fall back to OpenGL.

Another consideration is that there are many existing plugins for A3D, all of which use OpenGL by default, and rely on A3D to create the OpenGL context for them. To account for this, when the player is instructed to load a plugin, we should check to see if it has a graphics API configuration file in a specific directory. If it does not, create a default OpenGL context, then load the plugin. If the config file does exist, then we create the context specified (or OpenGL if the config does not specify an API).

This way, we should be able to maintain backwards compatibility with existing plugins, without needing to change the individual plugins themselves.

We can build and test this functionality without integrating our Vulkan engine, just using existing plugins to test if we're able to define, link, create, and use a default OpenGL rendering context.
Here are the files we will likely need to change for this to happen:

animal3D-DemoPlayerApp:

- main_win.c
    - Contains #define for default render context which loads OpenGL library and dependencies.
    - Contains creation of default render context which will move
        - Also creates main window, but specifying a render context is optional
- a3_app_renderer_config.h
    - Contains preprocessor logic for loading renderer library dependencies

animal3D-DemoPlayerApp-Platform:

- a3_app_load.c
    - Loads information pertinent to the demo plugins
        - Number of plugins
        - Callback information
- a3_app_window.c
    - /animal3D-data/animal3D-demoinfo-debug.txt is loaded in, contains the demo information.
        - Won't if there is no rendering context specified
    - Loads selected demo when chosen from menu
        - This is where we will be able to create our render context for the window before we load the demo. We will also have to change the WM_CREATE logic so that it still loads demo menu information.

animal3D-DemoPlugin:

- Resources
    - We will have to add a new resource file to configure the rendering API
- This plugin will use A3VK during DemoState rendering and DemoMode rendering
- It will also need to use A3VK when loading geometry and shaders
    - We will need to create our own geometry descriptor loading structures
    - Shaders need to be loaded and compiled differently
- We can still use the same A3D nomenclature when it comes to rendering functions and structures, as long as we're able to handle linking and compiling with preprocessors.

Stage 3: Integrate Vulkan Renderer with Animal3D

By this point, we will have a functional Vulkan renderer, and Animal will be able to load existing GLSL plugins, just in a different way that allows us to control the render context more finely. It will still be able to render using OpenGL.

Now, our main task is going to be using Animal's standard functions and structures, but we'll have to reformat them in the Vulkan library so that they use our Vulkan wrapper. This way, end users of the framework can use the same function names and structures across any plugin, using any rendering library.