Jimmy Griffiths
Josh Mahler
Dan Buckstein & Peter Sikachev
Vulkan Application README

We worked mainly on the same computer for the project, swapping off the keyboard every once in a while. However, Jimmy mainly wrote a lot of the initial stuff with GLFW, and some other initial Vk stuff with Instances, Debug managing, physical devices, and some stuff with rendering and synchronization. Josh wrote logical devices, swap chains, image views, and the simple fragment and vertex shaders. Image views, render passes, and the final creation of the pipeline were done by both of us.

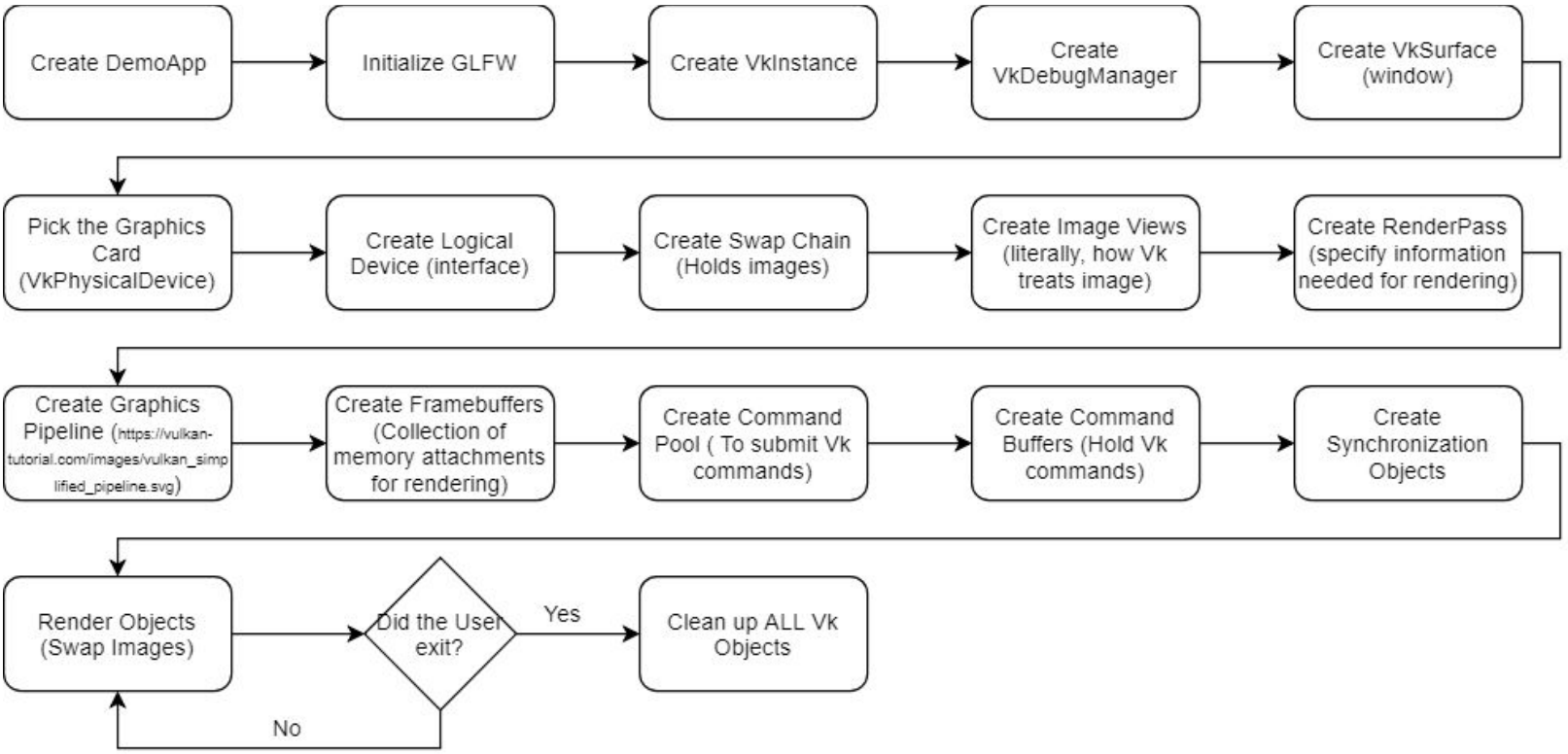
The link to our repository is here:
https://bitbucket.org/ChamplainJamesG/griffiths_mahler_egp-300-101-vulkan/src/Develop/

The final commit is commit 24 on the branch Develop.

The purpose of this project was to create a simple Vulkan pipeline. With our knowledge of the pipeline that we learned from this class, we would be able to write the pipeline from scratch, using Vk. The outcome of this, is that we will learn about different graphics APIs, and get to experience writing graphics code from scratch.

Our project fits into the category of “Prototype for the final project”. Our very first idea was to make it so that Animal3D could swap out using OpenGL with Vulkan. However, we quickly realized that would be an immense amount of work, even for the final. Therefore, we wrote our own simple demo application, that renders a 2D triangle using Vulkan & GLFW. Eventually for the final, we were hoping to render a more complex shader, such as Phong Shading, or something like that, rather than a 2D triangle on the FSQ. However, a simple shape is a good prototype that shows that we know how to use Vulkan, and can expand on it.

The pertinent code is in DemoApp.h and DemoApp.cpp. The application is nearly all Vulkan, with a little bit of GLFW. The functions in DemoApp should be organized in such a way that it follows the order of what needs to be done to create a Vk pipeline. From creating every necessary object and instance, to actually creating and rendering using the pipeline.


Create DemoApp
Initialize GLFW
Create VkInstance
Create VkDebugManager
Create VkSurface (window)
Pick the Graphics Card (VkPhysicalDevice)
Create Logical Device (interface)
Create Swap Chain (Holds images)
Create Image Views (literally, how Vk treats image)
Create RenderPass (specify information needed for rendering)
Create Graphics Pipeline (https://vulkan-tutorial.com/images/vulkan_simplified_pipeline.svg)
Create Framebuffers (Collection of memory attachments for rendering)
Create Command Pool (To submit Vk commands)
Create Command Buffers (Hold Vk commands)
Create Synchronization Objects
Render Objects (Swap Images)
Did the User exit?
Yes
No
Clean up ALL Vk Objects