CSE 168 Project

Mohammad (Sina) Nabizadeh - A59000982

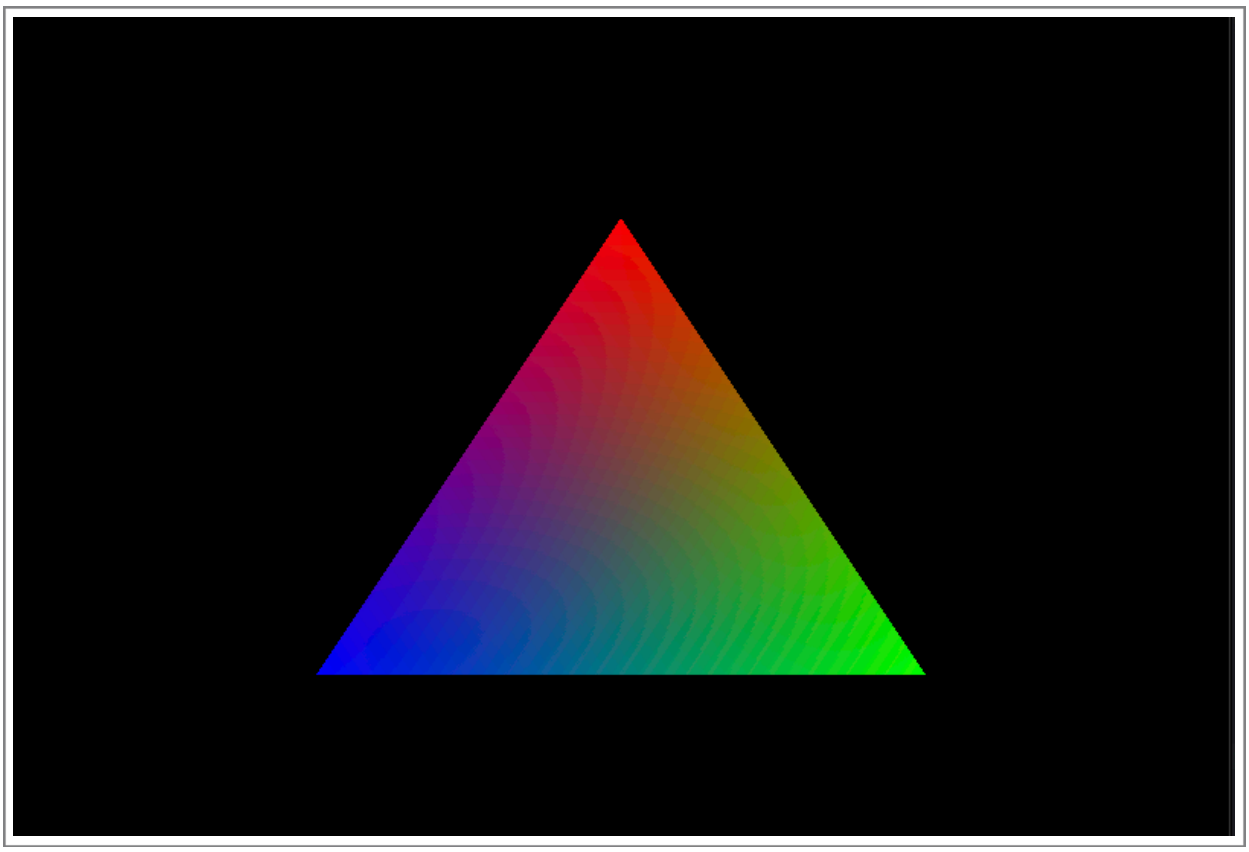

Figure 1. “Hello Triangle” using Vulkan API and GLSL

Vulkan Real-Time Effects

Sprint 2021

Project Proposal

Project goal

My plan is to follow the **Basic High Quality Rendering** project proposed in the final project course documentations with a twist. I hope to use Vulkan API instead of OpenGL/DirectX-11. This would be learning experience for me as I have not previously implement graphics application using modern low-level APIs such as Vulkan/DirectX-12/Metal. Since Vulkan also uses GLSL, the shader coding side would be very similar to an OpenGL implementation. I am confident that I can managed to do it in Vulkan given the time, but if it turns out to be extremely difficult, I will switch back to using OpenGL.

Setting up a simple Vulkan program

So far, I have managed to display a simple triangle on the screen using the Vulkan API (Figure 1). This has resulted in ~1000 lines of code, but I am confident that with some refactoring and using "vulkan.hpp" I can refactor the code to a smaller size. From what I have read online, it might even get to a point that it would be comparable, in implementation size, to a regular DirectX-11/ Modern OpenGL implementation.

Next, I plan to follow the Vulkan API tutorial from vulkan-tutorial.com which would teach me the simple setup and usage of basic elements such as vertex buffers, uniforms, and loading models. It is important to note that this tutorial is solely meant for the initial setup and will not include any advantage regarding the core features I plan to implement.

Once I have the basic program setup, the first goal is to setup a simple scene consisting of an object (e.g. Stanford bunny) where I can write up lighting code (e.g. Phong lighting for diffuse and specular) and move the camera around. Afterwards, I can add a plane underneath the object so that shadows can be seen. I will then proceed to implement **shadow mapping** which consists of rendering a depth only texture from the light point of view and using it a subsequent pass to query for shadow points. Next, I will have to read on how to setup a cube map texture and look it up in the fragment shader for specular reflection to implement **environment mapping**.

For my 2 add-on ideas I am planning to add two post-effects to my scene: **bloom** and **SSAO (Screen Space Ambient Occlusion)**. For bloom, all I have to do is to render a blurred version of my scene and blend it with my original image. As for SSAO, I believe I can reconfigure my setup so I have a couple of G-buffers such as normals, positions, and some sort of random sampling texture for hemisphere sampling. I feel SSAO might turn out to be too difficult and I might switch to doing a **Depth of field** post-process effect instead.