CSE 168 Project

Mohammad (Sina) Nabizadeh - A59000982

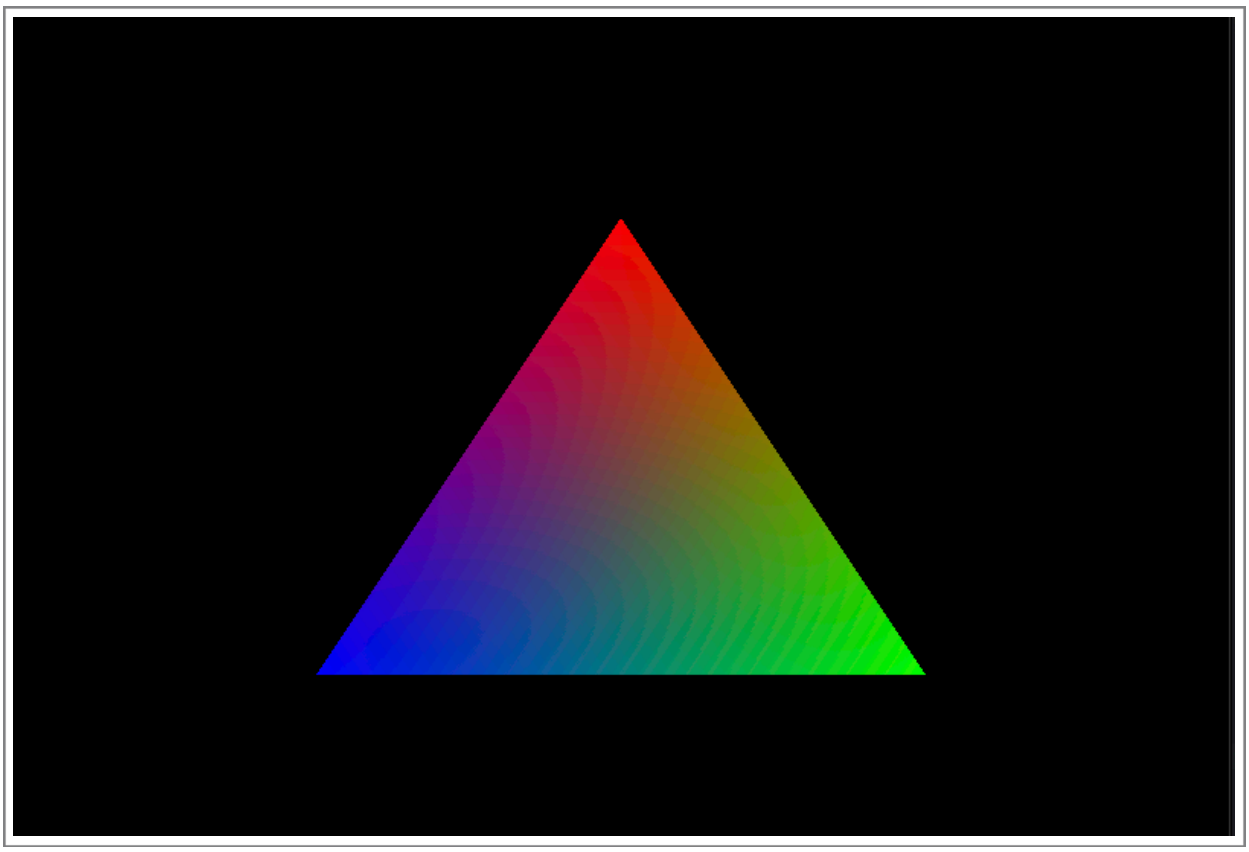


Figure 1. “Hello Triangle” using Vulkan API and GLSL

Vulkan Real-Time Effects

Spring 2021

Project Report

Project goal

Per my proposal, I followed the **Basic High Quality Rendering** project proposed in the final project course documentations with a twist. I hoped to use Vulkan API instead of OpenGL/ DirectX-11. This was a learning experience for me as I did 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 was confident that I can managed to do it in Vulkan given the time, but it turned out to be more difficult than I originally anticipated. Despite this, I managed to complete a large portion of my original plan.

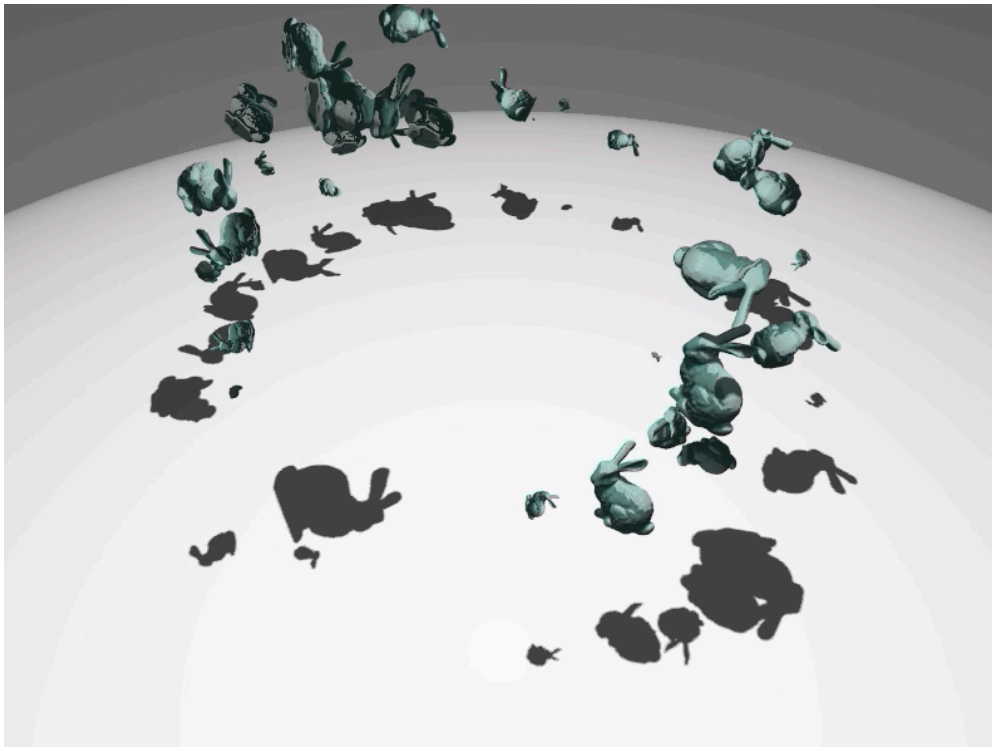
Vulkan program:

Shadow Mapping + Environment Mapping + Deferred Shading + half-implemented SSAO

I, first, displayed a simple triangle on the screen using the Vulkan API (Figure 1). This resulted in ~1000 lines of code. For this, I followed 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 was solely meant for the initial setup and did not include any advantage regarding the core features I plan to implement. Next, as I discussed in my proposal, I refactored my code using "vulkan.hpp" which made the code much cleaner than before.

I then added support for instancing so that multiple of the same objects could be rendered. Next, I added basic Phong lighting (with diffuse, specular and ambient components).

Afterwards, I added shadow mapping support with percentage closer filter (PCF) added, 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.



Following this, I added environment mapping with first using spherical reflection mapping which I learned is not as good as cube mapping so I implemented that. The reason is that spherical reflection mapping is view-dependent and thus one needs to generate a new sphere-map for every different view direction. On the other hand, cube maps are view-independent and hence I went with that for my final implementation.

For my 2 add-on ideas I am planning to add two post-effects to my scene: **Deferred Shading** and **SSAO (Screen Space Ambient Occlusion)**. For deferred shading, I would have to prepare multiple Geometry buffers (G-Buffers) to hold data such as normals, position, etc. to then use them in a subsequent pass for lighting. Once, this is done one can do post-effects such as SSAO much easier as now we have all the information available for any full-screen effects.