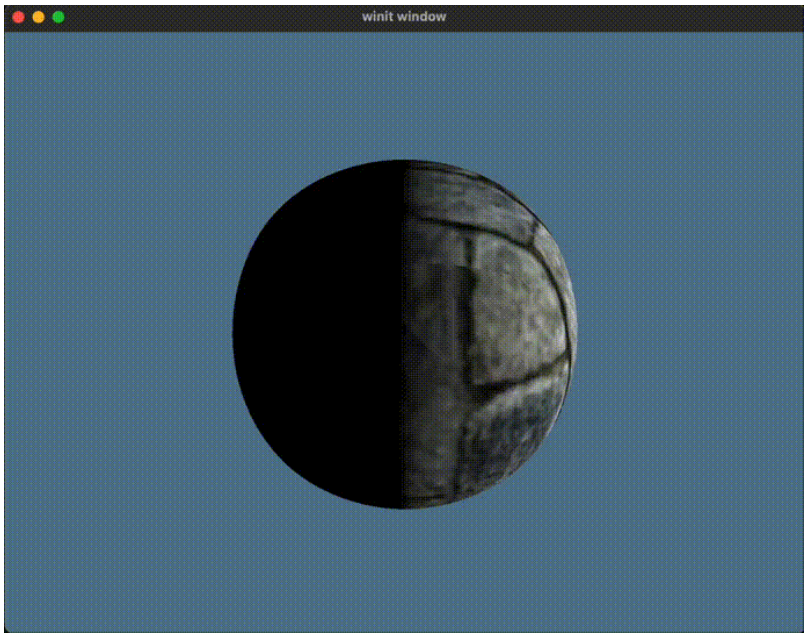
James Youngblood, CS6610, Final Project Progress Report



What I have implemented

I have implemented a window, event loop, and render pipeline in Rust. The library “winit” handles the window and event system, and the library “wgpu” handles the render pipeline.

One of the benefits of using “wgpu” is support for WebGL and WebGPU. I have set up a build target for web assembly and was able to get a couple of builds working in a web browser, although I am still fixing some issues with fetching assets in the web build.

Setting up the render pipeline a verbose process, more than I was used to with OpenGL. I have a little over a thousand lines of code just to render what you see in the screen capture above. This took more work than I expected, and I was not able to show off as much as I would have liked.

I have created a State struct to store general parameters of the render, and a Model struct to store the materials/textures and vertex data of the planet model. Right now, I have a test material consisting of a stone brick texture. The fragment shader simply samples the texture and performs diffuse shading using a simple directional light.

The camera has some basic controls to orbit around the planet and to zoom in and out, using the arrow keys.

What I plan to implement
I plan to implement a skybox with a view of the stars, instead of the blue background that is currently rendered. I have a texture for this included in the assets folder of the project already.

I plan to randomly generate the diffuse and normal texture for the planet, and to tune these random textures to look like continents and oceans.

I plan to render a semi-transparent sphere over the current sphere, with a cloud texture. This cloud texture will be randomly generated as well.

If I have time, I plan to implement an “atmospheric scattering” effect around the sphere, using ray-marching, which will probably be performed in the fragment shader for the most part. Cloud rendering may also be converted to ray-marching if I go down this route.

I plan to give the viewer some more controls, including setting a rotation speed for the planet (day/night cycle) and a camera target control, so that there are more options than looking at the planet’s center.

I plan to render a “sun” object in the direction of the sun, which would simply be a white circle.

How to compile and run the project
The dependencies are listed in the “Cargo.toml” file at the root directory of the project. Installing the Rust package manager, “cargo”, and running the command “cargo run” at the root directory of the project, will manage all dependencies and build/run the code.