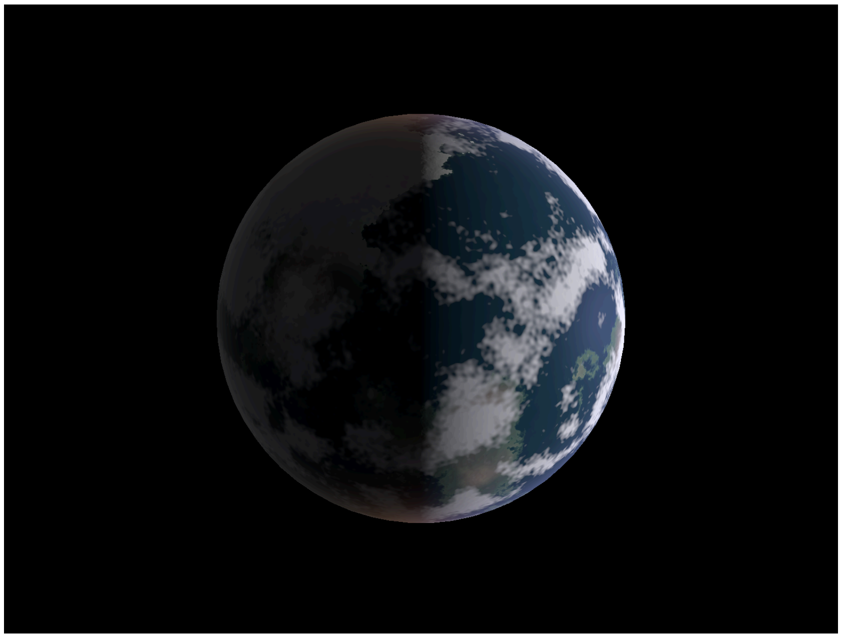
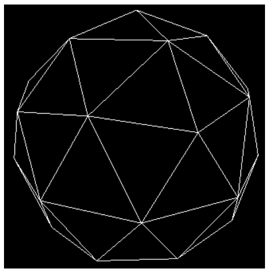
James Youngblood, CS6610, Final Project Report



What I have implemented

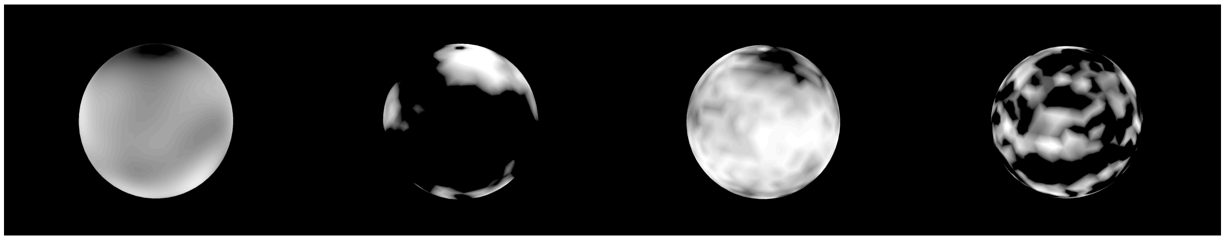
I have implemented a window, event loop, and render pipeline in Rust. The library "winit" handles the window and event system, and the library "wgpu" handles the render pipeline.

One of the benefits of using "wgpu" is support for WebGL and WebGPU. I have set up a GitHub page at https://soundeffects.github.io/planets_on_wgpu/ hosting the renderer.



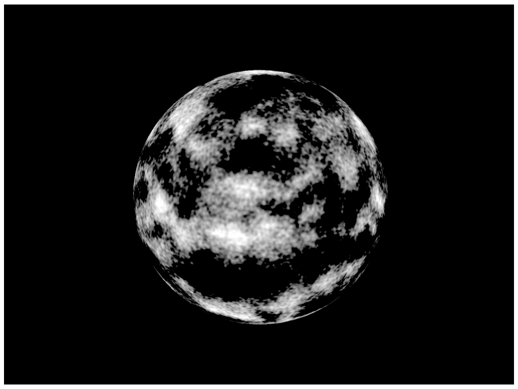
I use the "hexasphere" rust library to generate an ico-sphere mesh at runtime, with a fundamental topology similar to the image you see on the left. The sphere can be subdivided to a desired level of vertices.

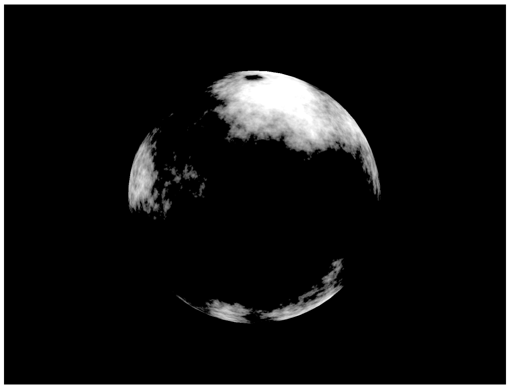
I used several layers of noise to determine the color at a given pixel of the planet. These are generated per-vertex in the vertex shader, at a random 3D coordinate offset by the planet's position, using a 3D simplex noise algorithm. The cloud is also offset by a changing value so that you see moving clouds.

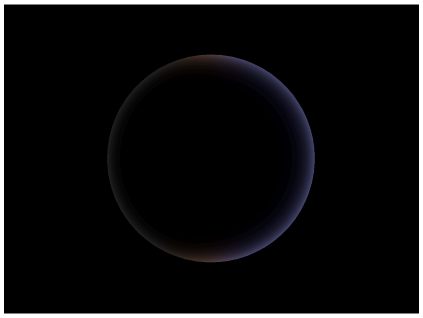


Heat map Height map Precipitation map Cloud map

In the fragment stage, I make a few more samples of noise to add detail to those layers of noise generated in the vertex stage.


Detailed clouds


Detailed height



From the maps, I blend a few colors together to get the terrain color for the given fragment. The final color compositing is done with an “atmosphere” effect around the edges of the sphere, as you see on the left.

Besides the GPU pipeline, I built a random tilt and rotation to the planet, creating a day/night cycle. I also build a camera controller which you can control by dragging the mouse or using the arrow keys, to orbit around the planet. You can toggle between a “surface mode”, zoomed in on the surface with the planet below, or an “orbit mode”, looking at the planet from the side. The screenshots have all been taken in orbit mode. Scrolling or pressing spacebar toggles the camera. You can toggle clouds with the “C” key.

What I could not implement

Noise sampling performed every frame. Initially I had intended to write the planet color to a texture after the first render, but I found it difficult to devise a texture coordinate mapping for my sphere mesh, and I noticed that the render still performed well (even on my phone in a web context), so I decided to drop texture rendering altogether.

I had planned to implement a skybox with a view of the stars, but I had a few difficulties with loading the image in a web context, as well as difficulties figuring out how to get cube maps working with wgpu. I decided to drop this feature for lack of time.

I’m not entirely happy with the way the clouds look. I had tried to stretch the horizontal axes (x-z plane) of noise sampling to create more cloud-like shapes, but I would liked to have experimented more with ways to create more cloud-looking patterns.

I had hoped to implement an “atmospheric scattering” effect around the sphere using ray-marching through an implicit sphere, but I didn’t have the time. I tried to implement the “next best thing” by adding atmospheric color to the edges of the sphere.

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. This should work across all operating systems.