Assignment 6: Perlin Noise

We have tried many nice-looking textures from images in our last homework. Now it is time to make our own by writing some simple math code. We will practice one of the most classical noise functions in the history of computer graphics — the Perlin noise function!

1 Reading

Before diving into the shader code, you may read a few classical articles about Perlin noise from our supplementary reading materials and our textbook. Here is the reading list:

- Course Slides of Noise
- Ken Perlin, Making Noise
- The book of shaders, noise
- The Perlin noise math FAQ

2 Starter Code

The starter code for this assignment, found under `assignments/a6/`, includes `main.cpp`, `a6_vert.vert`, `a6_frag.frag`, and `perlin.glsl`. Upon successful compilation and execution, you should see an image with a grey color, as depicted in Fig. 1.

We highly recommend you read the three shaders that were provided. The shader structure for this assignment is slightly different from the structure in the previous ones, because your implementation will be mainly focused in `perlin.glsl`, which implements several functions that will be called in `a6_vert.vert` and `a6_frag.frag`.

3 Requirements

For this assignment, you are expected to implement two main steps: implementing Perlin noise and generating a mountain terrain using the implemented Perlin noise.

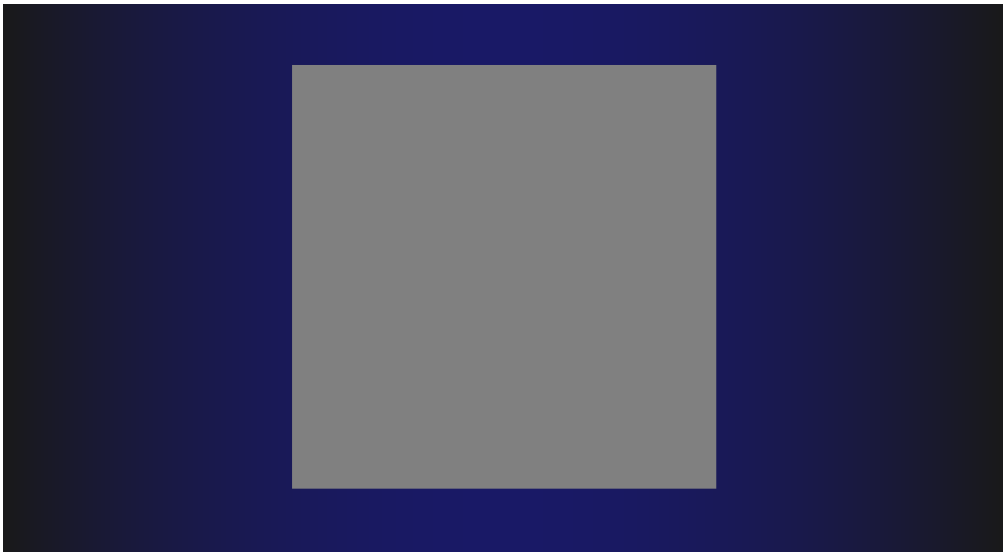

Figure 1: The default display of our scene setup with a grey plane.

4 Implementation Tasks

4.1 Step I: Perlin Noise

Part 1: Hash Function The first implementation task involves creating a hash function `hash2()` that accepts a 2D coordinate and returns a 'random' 2D vector. This function is foundational for generating noise patterns used in terrain rendering. We provided a default implementation for your reference, and you are encouraged to devise your own version by integrating different GLSL built-in functions to achieve different randomness.

Part 2: Perlin Noise Function The second task requires implementing the Perlin noise function for a single octave, using a given 2D position `p`. This involves calculating the grid cell index and fractional part from `p`, then leveraging these to compute the noise value. Your implementation task is to complete the function `perlin_noise(v)`: Given a 2D vector v, return the Perlin noise value at v. Refer to the course slides for mathematical details of constructing a Perlin noise function in 2D. You may use calls of `mix` and `dot` products, or you may implement the interpolation directly from its mathematical formula.

Part 3: Octave Synthesis This task involves synthesizing noise octaves by invoking the previously implemented `perlin_noise` function. Given a point `p` and an octave number `num`, the function must accumulate contributions from each frequency level to compute the Perlin noise octave. This process involves updating the amplitude and frequency at each level to add varying degrees of detail to the terrain.

If everything is implemented correctly, the result of Step 1 should **resemble** the image result shown in Figure 4. The result may differ when using a different hash function.

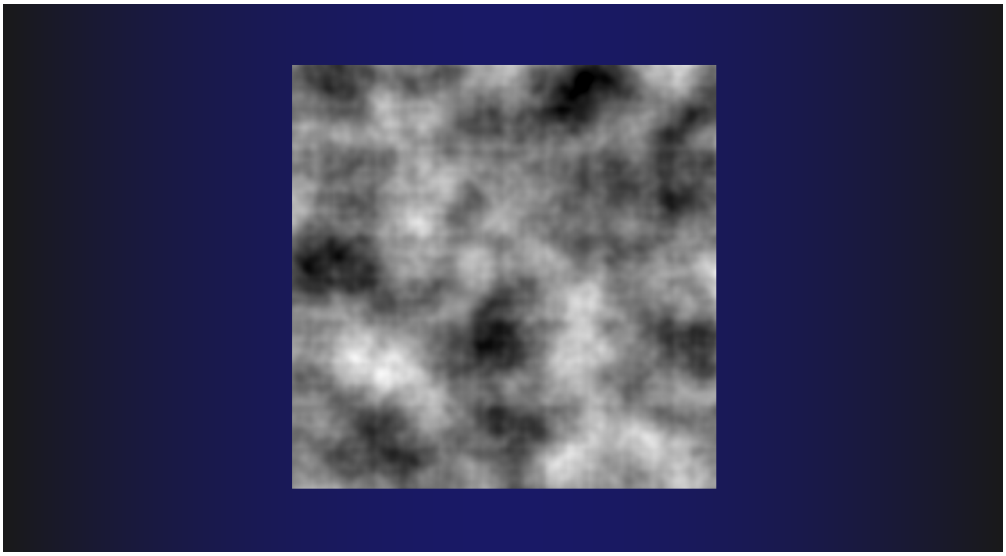

Figure 2: The Perlin noise result you will get after implementing the first step. Your result does not need to match the provided result.

4.2 Step II: Terrain Generation

Uncomment **#define Terrain 1** in the first line of `perlin.glsl` to switch the rendering mode from noise to terrain. The macro of Terrain is used in `a6_frag.frag`. Check the main function there to see how it works. After uncommenting this line, you will see a plane mesh with grey color and zero height on each mesh vertex (see Figure 3).

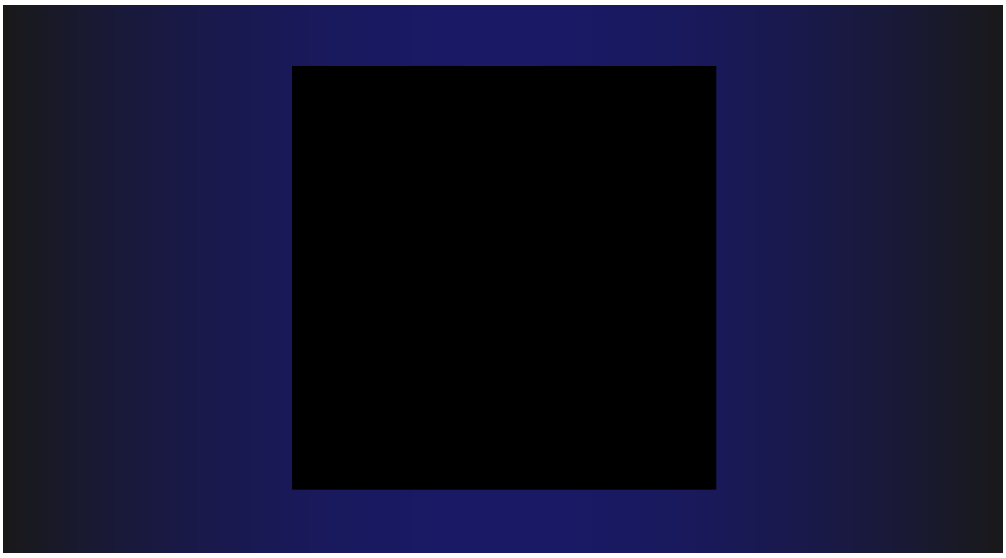

Figure 3: The default display of our scene setup for Step II with a black plane.

Part 1: Vertex Height Calculation In this task, you will create a function that determines the height of a point on the input terrain mesh using the `noise_octave` function you have implemented

previously. We provide a default implementation in the commented code for this function, and your task is to customize the height function for your own terrain model. Specifically, you need to calculate the height value by calling `noise_octave`.

Part 2: Normal Calculation The next task involves computing the normal for a given 2D point, which is essential for correct lighting and shading of the terrain. This requires determining the normal vector at a point `p` with its height generated by the noise function by calculating the heights of its adjacent points and utilizing their cross-product. Your task is to complete the function `compute_normal(v, d)`. In this function, `v` is a 2D point, and `d` is a float specifying the offset of a neighbor point in the x or y-axis. To compute the normal at `v`, you may calculate the position of the 4 neighboring points as follows:

- `v1 = vec3(v.x + d, v.y, height(vec2(v.x + d, v.y)))`
- `v2 = vec3(v.x - d, v.y, height(vec2(v.x - d, v.y)))`
- `v3 = vec3(v.x, v.y + d, height(vec2(v.x, v.y + d)))`
- `v4 = vec3(v.x, v.y - d, height(vec2(v.x, v.y - d)))`

After calculating the positions of `v1`, `v2`, `v3`, and `v4`, you will calculate a normal vector based on the cross product between vectors `v1v2` and `v3v4` that is specified by the four positions (you need to figure out the mathematical formula of this cross product by using `v1`, `v2`, `v3`, and `v4`). Don't forget to normalize the vector before returning the result.

Part 3: Phong Shading Next, we will implement a standard Phong shading model in `shading_phong()` function. You may reuse your implementation in the previous assignments for this task.

Part 4: Color Calculation In the last step, you will implement the calculation of emissive color on the terrain surface. This emissive color will be used to multiply with the Phong shading color to get the fragment color of the terrain. This emissive color represents the color transition according to the height of mountains or rivers. We provide a default implementation in the commented code for your reference, and you need to customize your own way of color calculation using your height function. You implementation will be in `shading_terrain()`.

The result of Step II will vary according to your own implementation. We provide an example using `height = 0.75 * noise_octave(v,10)` and multiplying values less than 0 by 1/2. The color scheme is a simple linear interpolation between green and brown based on height (see Figure 4).

5 Creative Expression

The creative expression theme for this assignment is **"Mountains and Rivers"** (You don't have to create both of them. But in computer graphics, they usually show up in a scene together) There are many ways to express creativity in this assignment. You can choose different Hash functions to generate terrain, call them with different octaves to produce height, and specify different color

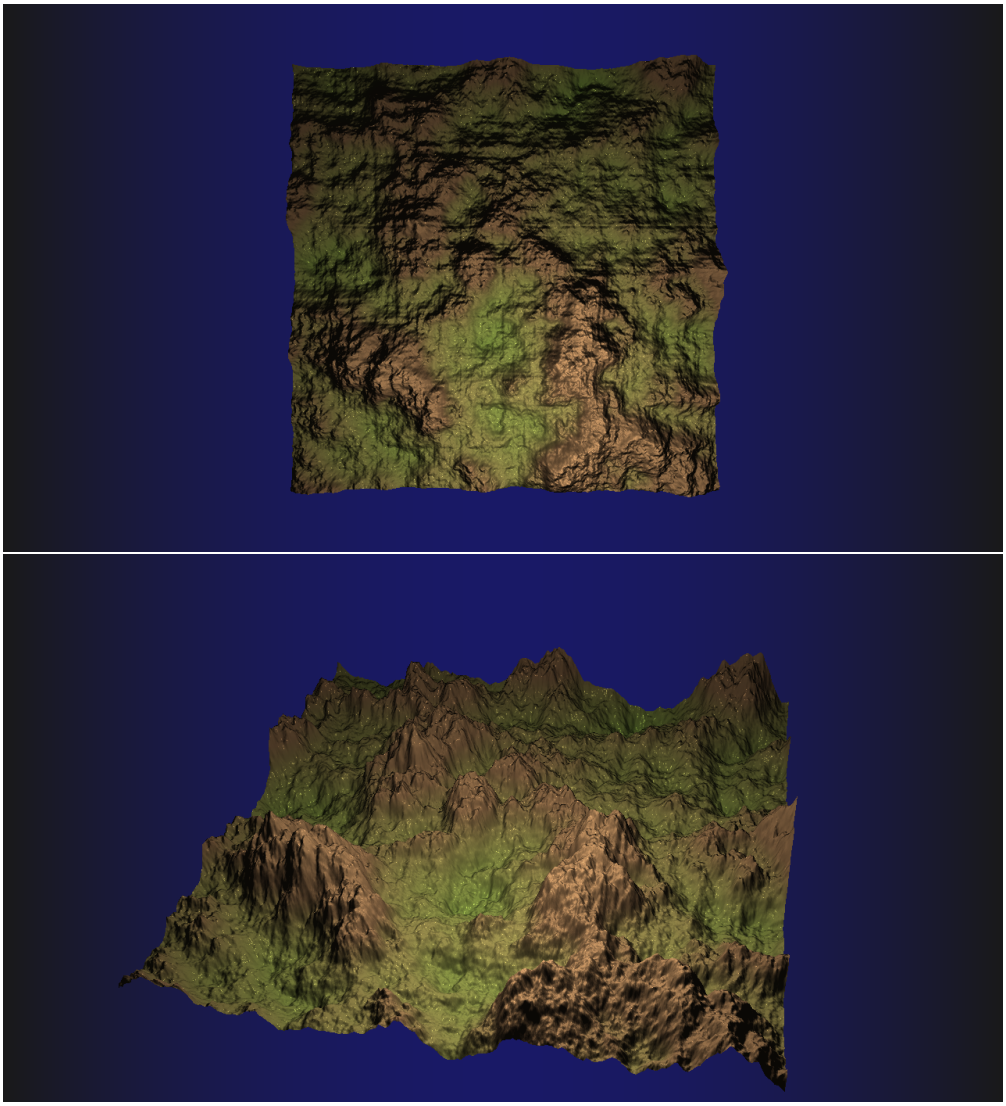

Figure 4: The mountain terrain with Phong shading and emissive color you will get after implementing all steps for Step II. The two images show the terrain from different camera views. You are asked to create your customized mountain and river terrain.

functions to express the color theme you desire. You can make snow, moon-like terrain, or even something that doesn't exist.

To motivate your creation, we list a few potential ways of generating different types of terrains with different noise functions:

Figure 5: An example of the terrain mesh obtained using a noise function with power function: $h = pow(noise, 0.1) * ....$

Figure 6: An example of the terrain mesh using an exponential function: $h = e^{\text{noise}} * ....$

Figure 7: An example of the terrain using an abs function: $h = (1 - abs(\text{noise})) * ...$, stretched along the y axis. (clouds use the same function but different color)

6 Submission

Submit the following components for evaluation:

- Your source code `perlin.glsl`;
- Screenshots of your Perlin noise image;
- A video or a few screenshots from different camera angles demonstrating your rendering of mountains and rivers;
- A concise paragraph that provides a technical explanation of your implementation for the customized scene.

7 Grading

This assignment is worth a total of 8 points, with the grading criteria outlined as follows:

1. Technical contribution (6 points): The core of the grading is based on the correct implementation of noise functions. The distribution of points is as follows:
   - Step I (Part 1-3): 3 points
   - Step II (Part 1-4): 4 points (1 point for each part)
2. Creative expression (1 point): This aspect focuses on your ability to create new terrains by devising noise functions and incorporating them with different lighting and shading effects.

8 Sharing your Work

You are encouraged to share your graphical work with the class. If you want to do so, please upload your image to the Ed post **A6 Gallery: Mountains and Rivers**. This is an excellent opportunity to engage with your peers and gain recognition for your work. Share with us the majestic world in your heart!