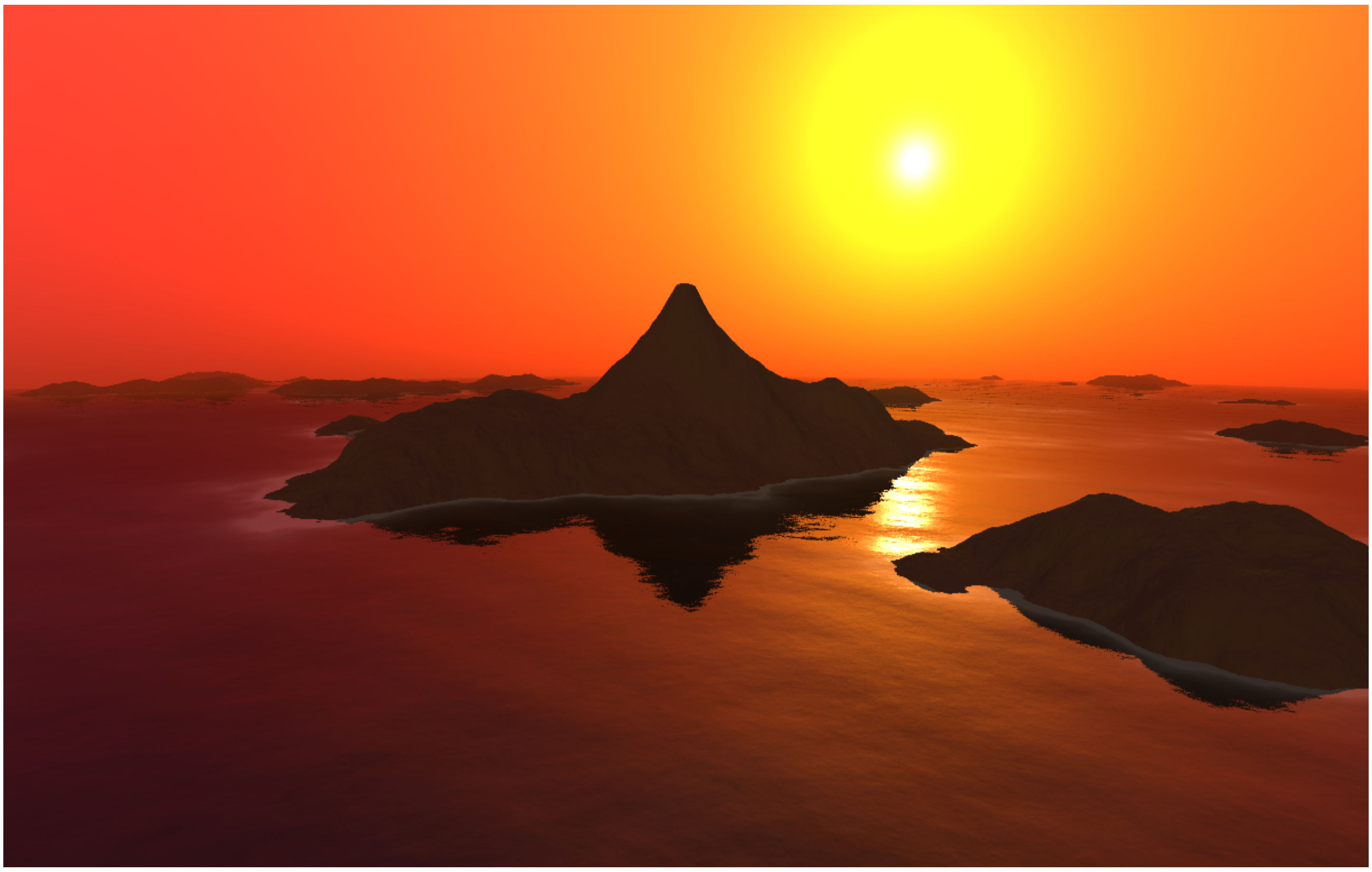
La Sapienza university of Rome

Raymarching vulcanic archipelagos



Try it live! https://www.shadertoy.com/view/4dcyzM

Luca di Bartolomeo

Computer graphics course final project
Teaching professor: Fabio Pellacini

February 9, 2018

1. Terrain generation

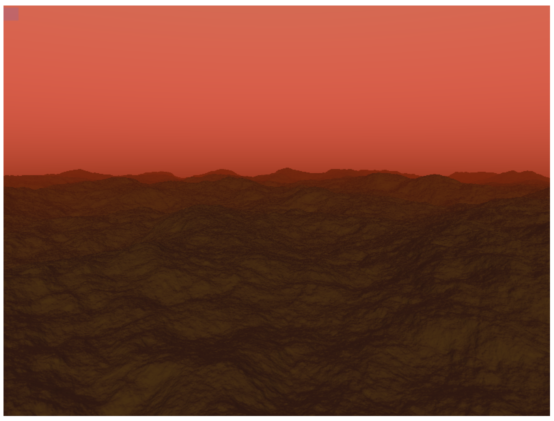


Figure 1: Normal fbm terrain

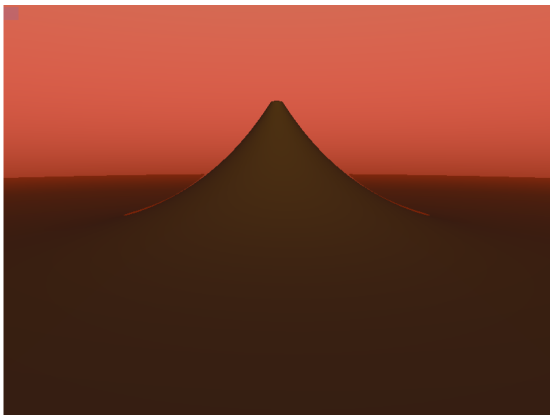


Figure 2: $e^{-\sqrt{x^2+z^2}}$

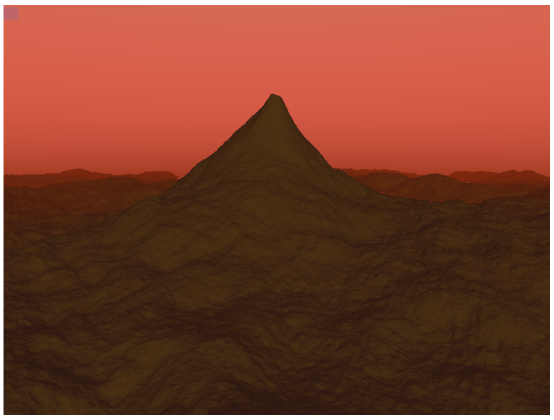


Figure 3: fbm + $e^{-\sqrt{x^2+z^2}}$

All the terrain is procedurally generated and represented as a heightmap. It is a combination of a noise function and an elevation function.
Initially, the height of the terrain in a particular point is generated through a fractal brownian motion (fbm) noise, which is defined as:

$$fbm(x,z) = \sum_{i=1}^{n} \left(\frac{1}{2}\right)^i \cdot noise(2^i x, 2^i z)$$

where n is the number of *octaves*. In fact, the fbm function is a sum of the same noise, each time with half the amplitude but twice the frequency. The *noise* function is a normal value noise generator (gradient noise like Perlin can be used too).
Assuming we know the analytical derivatives of the *noise*, the derivatives of fbm are the weighted sum of the derivatives of the *noise* for each octave. Thanks to this, the normal of the terrain can be calculated using a closed formula, without occurring in the time and precision penalties of numerical derivation.
The fbm heightmap, with the right parameters, produces a landscape like in Figure 1; to realize the volcano the heightmap was summed to this elevation function:

$$h \cdot e^{-k\sqrt{(x-x_0)^2+(z-z_0)^2}}$$

where (x_0, z_0) are the desired coordinates of the volcano (which for simplicity are $(0,0)$), h its height, and k its width. The effect of this function can be seen in Figure 2.
The crater is simply obtained by subtracting:

$$\max(0, w - \sqrt{x^2+z^2})$$

where w is the width of the crater.
By summing those three, the result is Figure 3: a natural-looking volcano, where the further the land is from the volcano, the less affected it is by the elevation function, thus forming a smooth rocky, hilly landscape.

To avoid a big contrast with the horizon, far away land slowly fades away by the following formula:

$$col = terrain\_col(1-\alpha) + sky\_col(\alpha) \quad \text{where} \quad \alpha = \left(\frac{dist}{maxdistance}\right)^3$$

2. Water

The water is an important element because it breaks the boring rocky landscape into sparse small islands. It's not a perfect blue plane: the height of the water is modified by a sum of various semi-random sine waves, in which their argument is shifted proportionally with time, to simulate the flowing of the water. To the sine waves we can also add some fbm noise to give water a natural-looking roughness on the surface:

$$water\_level(x, z) = y_0 + sin(x + z + time) + sin(\ldots) + \ldots + fbm(x, z)$$

To render water reflections, the derivatives are obtained numerically, as precision was not as important. Calling the derivatives r_x, r_z, the normal of the water surface is calculated as:

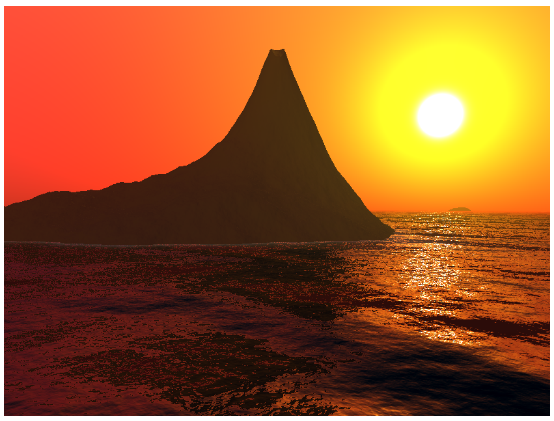


Figure 1: Very rough water reflections

$$n = \Sigma_x \times \Sigma_z = (1, r_x, 0) \times (0, r_z, 1) = (-r_x, 1, -r_z)$$

Having the normal, calculating water reflections is very easy: just raymarch the reflected ray an adequate number of steps; if it doesn't hit anything, render the color of the sky in that direction; otherwise, return the darker color of the terrain.
Another visually appealing effect, which is rendering the water brighter near the coasts, is achieved by:

$$water\_color+ = \frac{1}{water\_level - terrain\_level(x, z)}$$

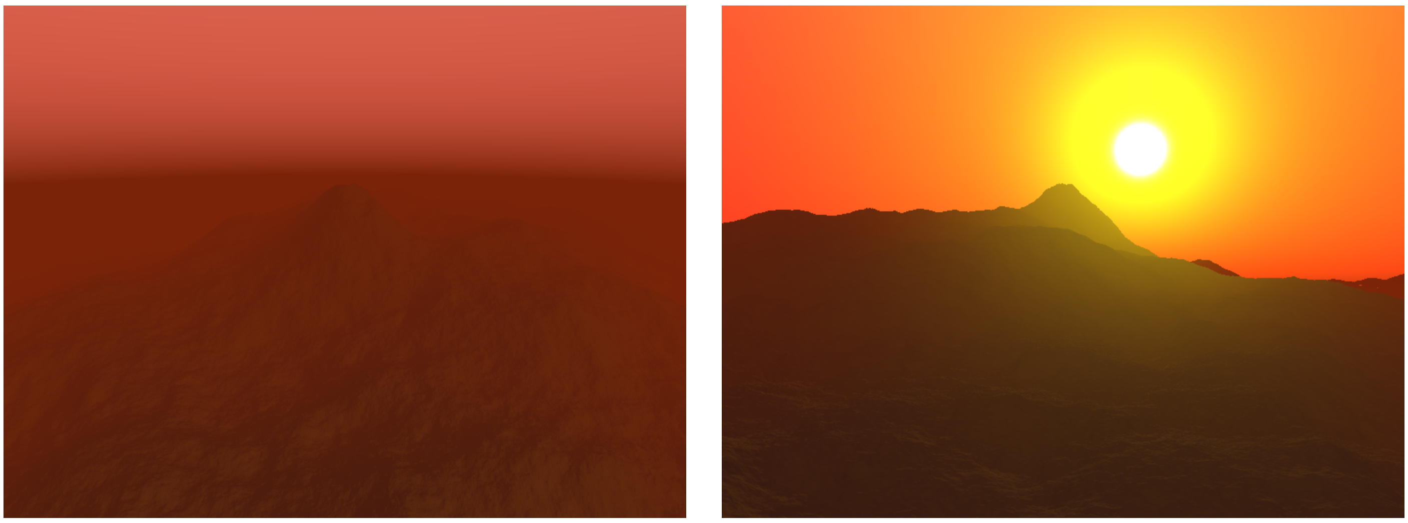
3. Fog, Sun glare

The amount of fog in a certain point at a distance d from the camera is calculated with:

$$fog\_amount = 1 - e^{-(d^2/maxdist)}$$

The color of the terrain is then blended with the fog color with fog_amount as the alpha amount.
To achieve a semi realistic sun glare effect, we can modify the mix the original fog color to the color of the sun (orange/yellowish) according to how much the camera is looking at the sun, and we calculate that with:

$$sun\_amount = dot(ray\_dir, sun\_dir)$$


(a) Normal fog color

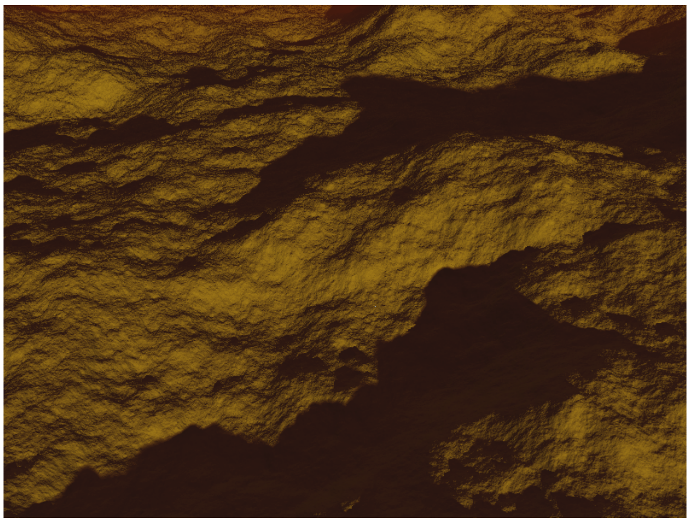
(b) Fog color shifts to yellow near the sun

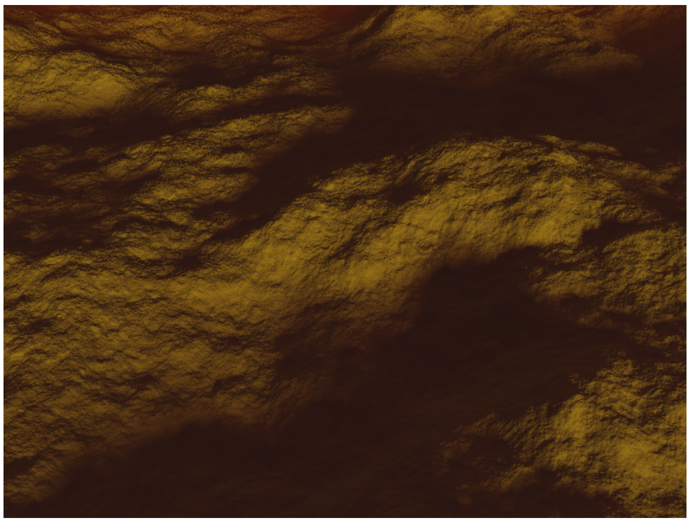
4. Soft shadows

Soft shadows make lighting a lot less artificial; although usually it is very expensive to compute soft shadows with raytracing, they are very easy to do (and very fast) while raymarching.
While calculating the shadow ray (the ray that goes from an object to a light source to see if it is shadowed), at each step we can get the penumbra (initialized at 1) this way:

$$penumbra = \min(penumbra, \frac{terrain(p.x, p.z) - p.y}{dist}), \quad \text{for each step where } p \text{ is the raymarching position}$$

The *penumbra* its initialized at 1, and the lower its value is, the closer the shadow ray has passed to an obstacle between the object and the light.


(a) Hard shadows


(b) Soft shadows

5. Special Thanks

Inigo Quielez, for his inspiring work and his wonderful tutorials.
Particularly useful were his fog tutorial, and his demos Elevated and Rainforest.
Frankenburg's water shader
nimitz's lens flare shader
Sara Di Bartolomeo, for her amazing artistic counseling.