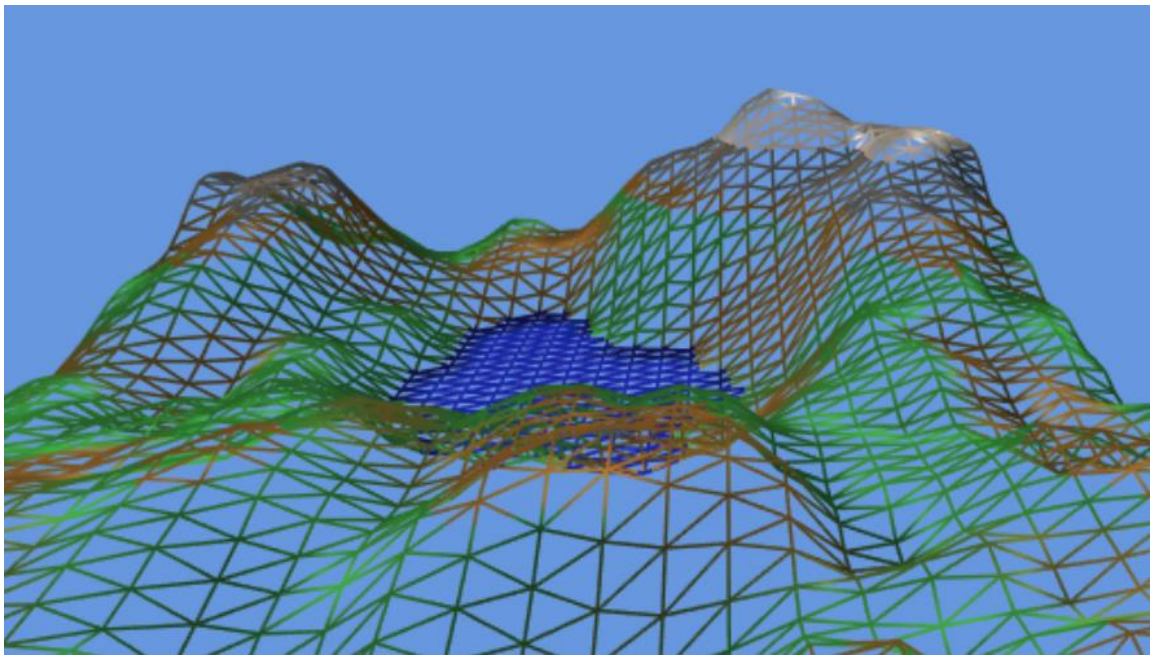
Noise generated landscape

Final project for DTU course 02561



TLDR:
This report aim to generate a landscape from pseudo-random noise functions in WebGL.
A 2D Simplex noise function is used as a elevation map. The color the landscape is derived by classifying the elevation into: water, ground and mountain, and the ground are varying between grass and sand by sampling another noise function.
The final animation gives acceptable impression, but the image seems rather soft, and more high frequent noise should be added.

Table of Contents

1 Context and usage

In some situations it can be difficult and time consuming to generate textures and landscapes for 3D scenes.

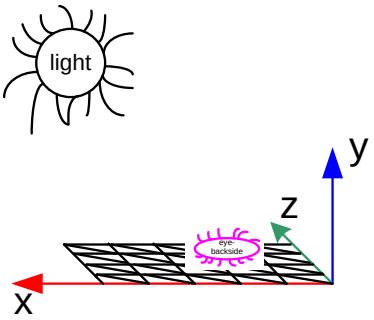
Procedural functions can e.g. be used to mimic the texture of a marble vase or a height map to use as terrain for a computer game. The same result could be obtained with photos or manual drawing. However with a few adjustments you can generate new variations of such texture, without the need new photos. Furthermore the procedural methods require very little space compared to a texture (even compressed).

The size aspect also make the procedural methods to be used in size-restricted animation (the 4k & 64k intros of the demoscene[1]), although this admittedly is a sub-genre.

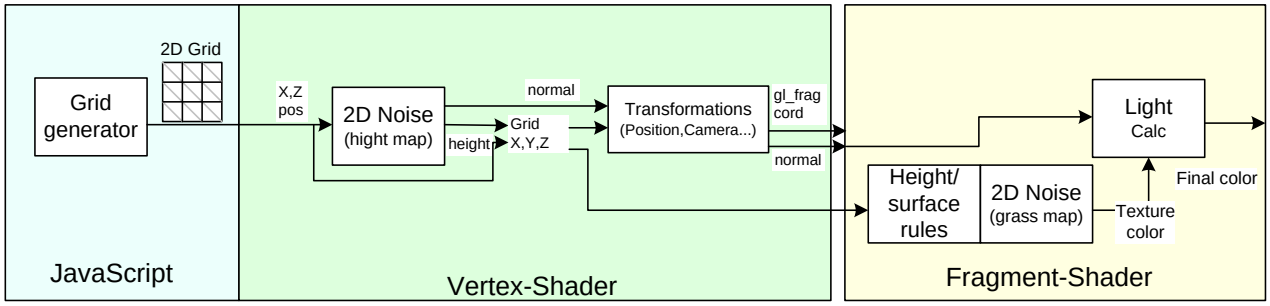
It is worth mentioning the methods might require more CPU/GPU power to compute than a texture (but on the other hand the transfer between CPU and GPU is often considered the biggest bottleneck).

2 Overview

Before digging into details this section will provide an overview. The idea is to sample 2D noise and use this as basis for a landscape.



To match the rasterization pipeline used in the exercises, we must map the 2D noise to triangles. The basis of the setup will hereby be a 2D plane of triangles, which hereby acts as a sampling function of the elevation. To render this 3D setup back to a 2D image we will use camera and perspective transformation, plus we include a light source to give a sense of depth.



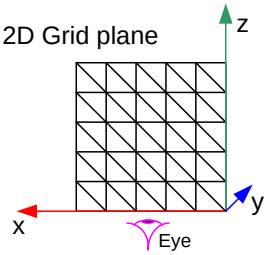
On the JavaScript side: 2D coordinates of the grid are generated and fed as attributes to the shader code.
In the vertex shader: The 2D coordinate is used as input to the noise function which returns an elevation, hereby a yielding a 3D point. Furthermore we obtain a normal vector (needed for light model) by sampling nearby points. Finally the 3D points is carried through the typical world, camera and perspective transforms. To allow for further surface characterizations the elevation and coordinate is carried on the fragment shader.
The fragment shader: Based upon elevation; threshold it determines if the point classifies as mountain (grey), water (blue) or ground. In the case of ground we sample another 2D noise to determine if there is grass.

[1] See wikipedia and pouet.net for productions

3 Implementation details

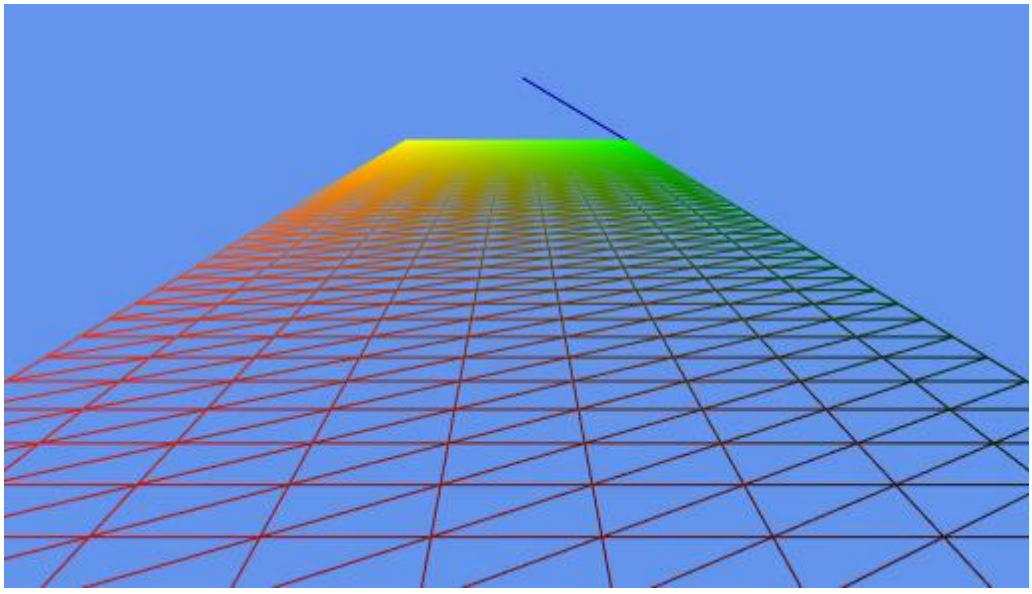
After the overview this chapter will cover more into detail with different aspect of the implementation. The implementation has been carried out as different "parts" in line with the format of the exercises. For most of the section there are one or more demos in the folder '*final_report*' with the WebGL-code.

3.1 The grid and scene basis



The grid is fixed to a size of 1.0 x 1.0 and split up into the requested amount of points in x and y direction. After defining the point we create indexes for the triangles. The result is a checker-board like pattern, where each field is divided in half.

Since we wish the landscape to extend in the depth we scale it in the z-direction with a TRS-matrix[2] and setup a camera looking straight down the z-axis.

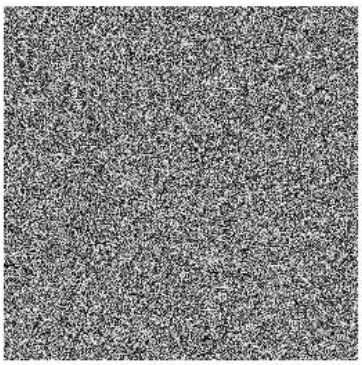


The result looks as illustrated above. The blue line on the right is z-axis of coordinate system used to verify the camera setup.

[2] TRS is a 4D-matrix. TRS is acronym for Translation + Scaling + Rotation... although applied in the opposite order.

3.2 Noise

To generate some kind of believable terrain we can not just completely white noise, changes are too rapid.



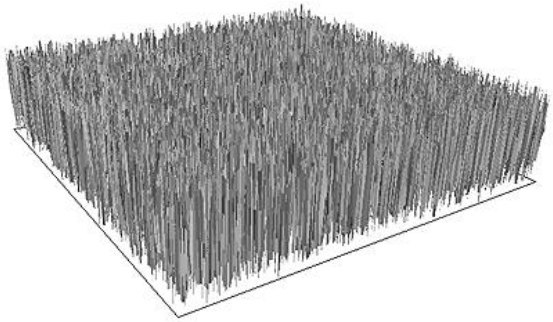


Image from https://mathematica.stackexchange.com/questions/159261/generate-white-noise-in-2d

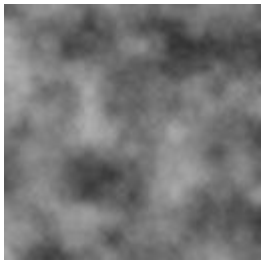
We need a more smooth (i.e. low-frequent) waves + perhaps a bit of random details with less amplitude. This would suggest the use of pink or brown noise, which by definition decreases the power-spectral-density with respectively 3 dB and 6dB/decade.

A common way to a similar effect is to add a low amplitude noise function to one or more downsampled noise functions with less amplitude.

3.2.1 Noise function design for computer graphics

There exist noise algoritms which developed/tweaked/engineered specifically to be used as noise for graphics, among the most famous are Perlin noise and Simplex Noise, both developed by Ken Perlin[3].

Perlin noise



Multiple frequencies of Perlin noise mix together

These algorithms have several steps[4] and it is outside the scope of this report to explain these in further detail (there is great risk of telling something wrong which could be spotted by those familiar with the algorithm[5]).

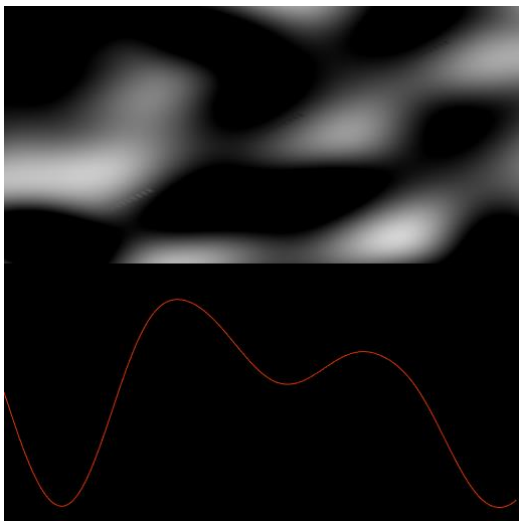
3.2.2 Noise for implementation

For this report it was chosen to find an implementation of Simplex noise on ShaderToy[6] with a MIT-license, and mostly threat it as a black box (and I cannot determine if it is authentic simplex noise).
These noise functions are not truly random, but instead use a hash function to produce pseudo random values based on an input value. The seeming random values can e.g. be obtained by multiplying the input coordinates with a floating point number and taking the fraction. This pseudo-random nature also has the advantage of being reproducible with same input.

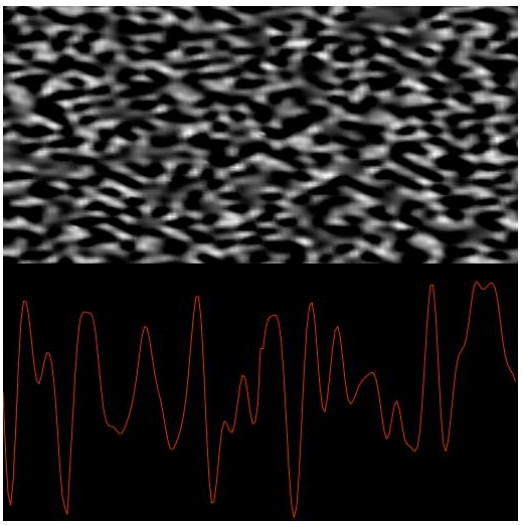
[3] Ken Perlins homepage with source https://mrl.nyu.edu/~perlin/doc/oscar.html
[4] There is a good explanation here: http://flafla2.github.io/2014/08/09/perlinnoise.html
[5] Noise explorer, Jeppe Frisvad: http://people.compute.dtu.dk/jerf/code/noise/
[6] 2D Simplex Noise, Inigo Quilez: https://www.shadertoy.com/view/Msf3WH, contains links to several noise functions.

3.2.3 Inspecting the noise

There is created a small test rig where the noise function can be evaluated a 1x frequency

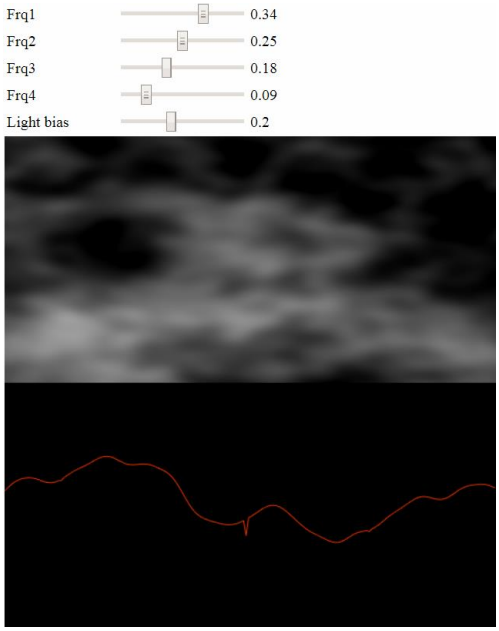


X and Y going from 0 to 2, slow waves

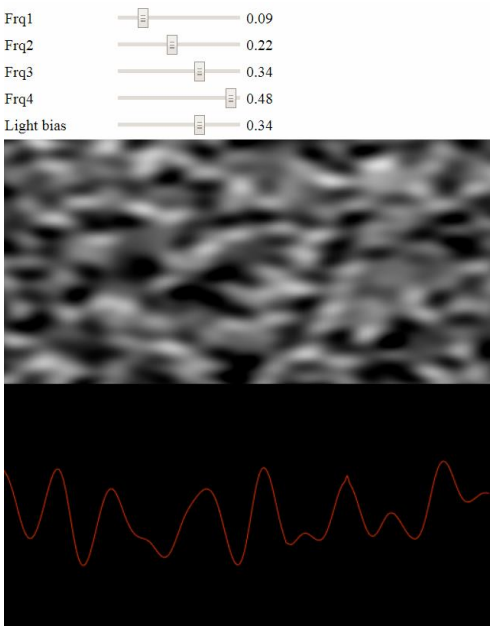


X and Y going from 0 to 16, more rapid waves

However we need to mix multiple frequencies together to get something useful. The second test rig allow for noise mixture of octave spaced frequencies, as well allow for enabling scrolling.

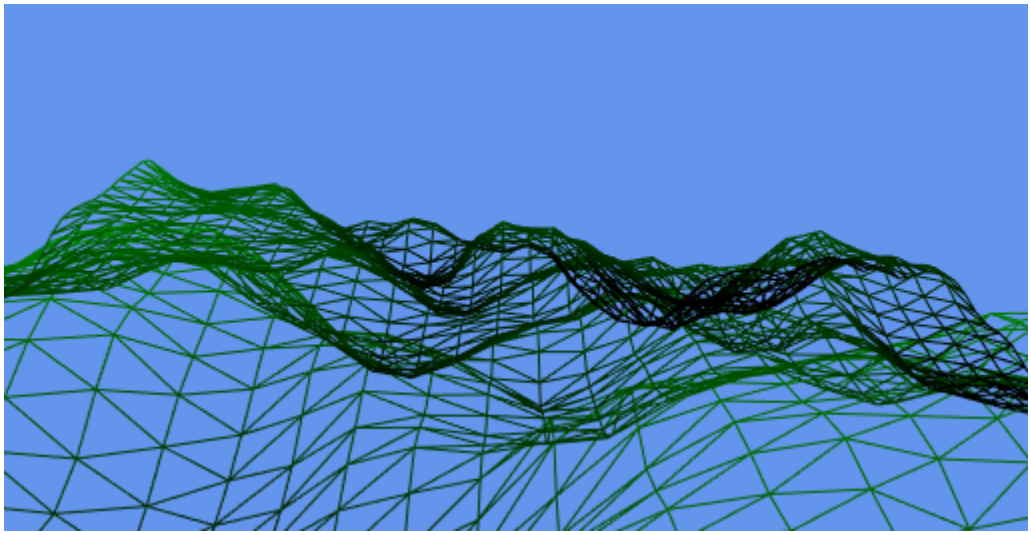


Emphasis on low frequencies



Emphasis on higher frequencies

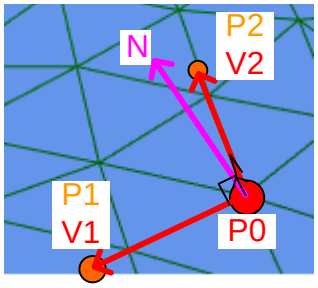
As the final noise there was used a mixture of 4x octaves, for each octave the amplitude is halved.



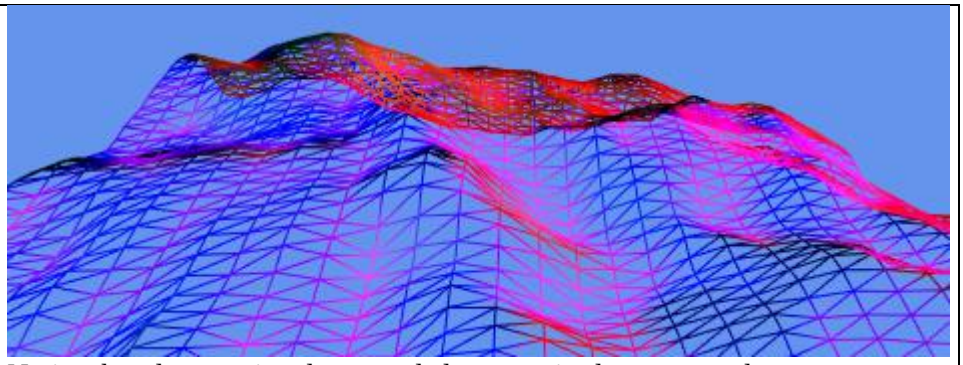
If we apply this to our grid it actually starts to look usable for landscape.

3.3 Getting surface normal

Next step is to determine the normal of the surface, which is required for calculating illumination with a light model. A problem with our setup is that the noise function does not directly provide gradient or normal as part of the result. The first thought is just to calculate is based upon the other points sampled, however since the calculations are performed in parallel in the vertex shaders, neighbor noise-values can not be exchanged[7]. The solution is just to determine the normal vector based upon 3x points: The intended sample point P0 and a small step in x and y-directions (P1, P2) make up two vectors (V1,V2) which are basis for calculate the normal with cross-product.



Normal = V1 x V2



Notice that slopes going downward along x-axis always are red.

To validate the result there is a small code example, which set the value of normal vector as surface color. Hereby the color of the landscape tilting left should be consistent across surface.

From the image this looks ok, however during the validation of the light it was noticed that the lightning gave an inverted impression of the light source. Using negative normal-vector solved this but it should not be needed.

[7] It might be possible to perform a 2-pass approach similar to shadow-mapping.

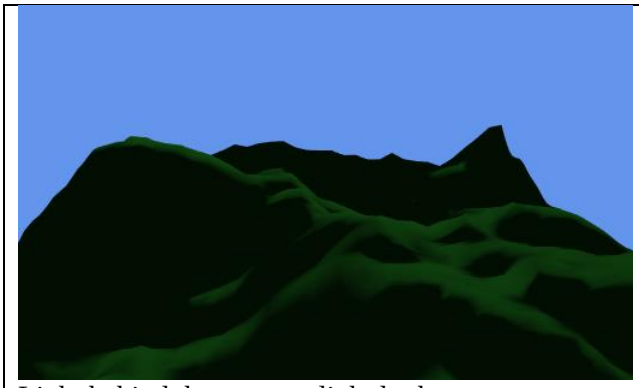
3.4 Illumination of the landscape

After obtaining a normal vector we can apply light which make the contours of the landscape more believable. It was chosen to use the modified Phong model (the one with half-way vector rather than reflection). This was simply due to the fact that the same model was available in the finished exercises.

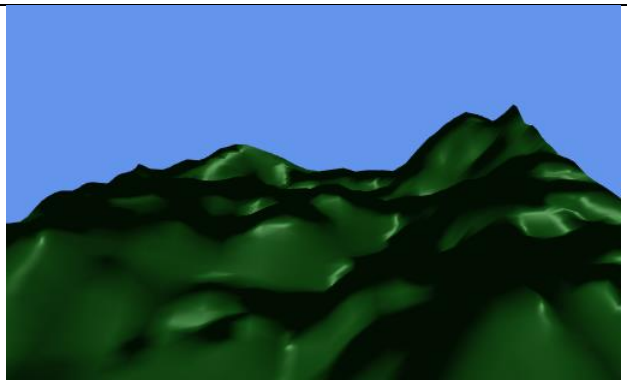
After seeing the result it is clear that the landscape to not really need any specular product and I turned the part of the light down (highlight on a field on grass looks odd). For the landscape use-case it would be fine to suffice with Gouraud shading.

3.4.1 Bad indication of light location.

During the implementation it was noticed that the model gave a false indication of where light was located. Having the light source located a bit behind the camera should make the visible surfaces appear illuminated, however the opposite was shown. By changing the sign of the normal vector this was mitigated, however this should not be needed. The normal vectors looked correct in previous images and all signs of the light calculations matched the equations of the Phong model[8].



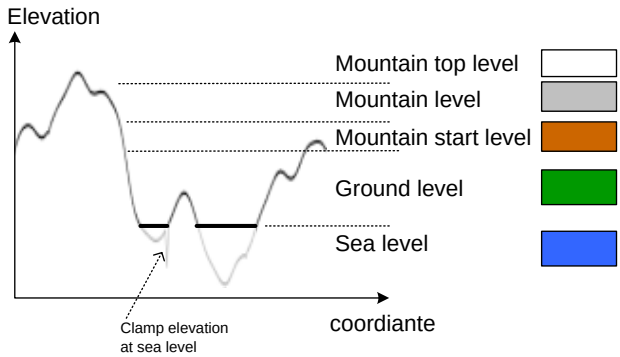
Light behind the camera, light looks wrong



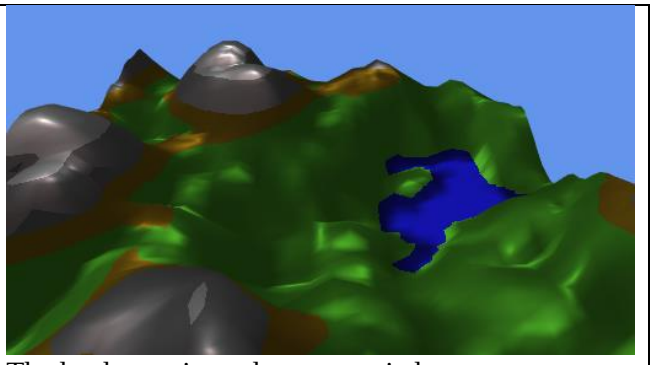
Light behind camera. Sign of normal vector is flipped. This make light look correct.

3.5 Elevation to color mapping

To make the landscape more interesting the elevation is used for varying the terrain, simply by selecting a suitable color[9]. The highest areas are designated as mountain, and the lowest areas are clamped as sea level. The sea-level clamp is performed in the vertex shader, and colors are determined in fragment shader.



Simple thresholds to determine color



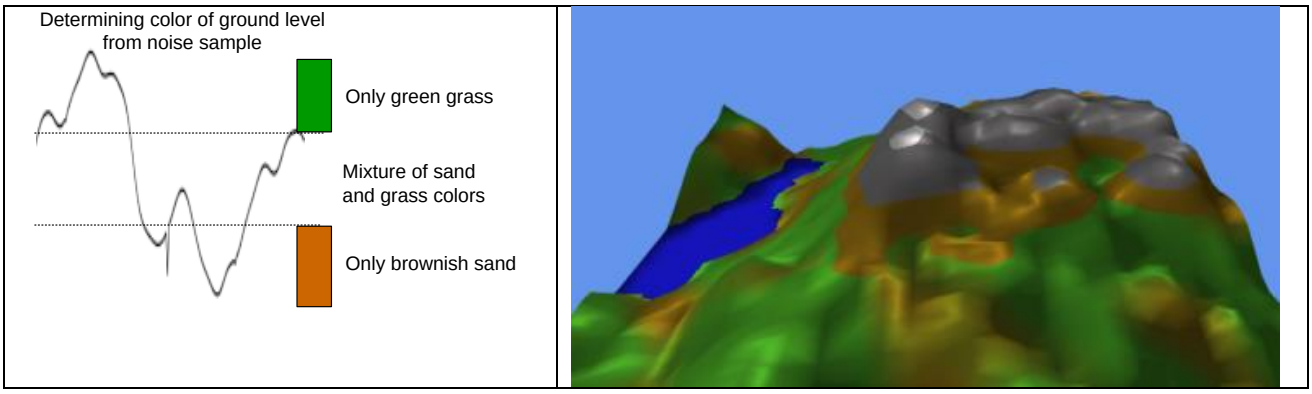
The landscape is made more varied.

[8] After completing i have found tutorials making a better explanation of the light model + provide clear GL shader code *Blinn-Phong reflection model in GLSL*: http://www.sunandblackcat.com/tipFullView.php?l=eng&topicid=30
[9] Heavy inspired by '*Making maps with noise functions*' https://www.redblobgames.com/maps/terrain-from-noise/

3.6 Random grass locations

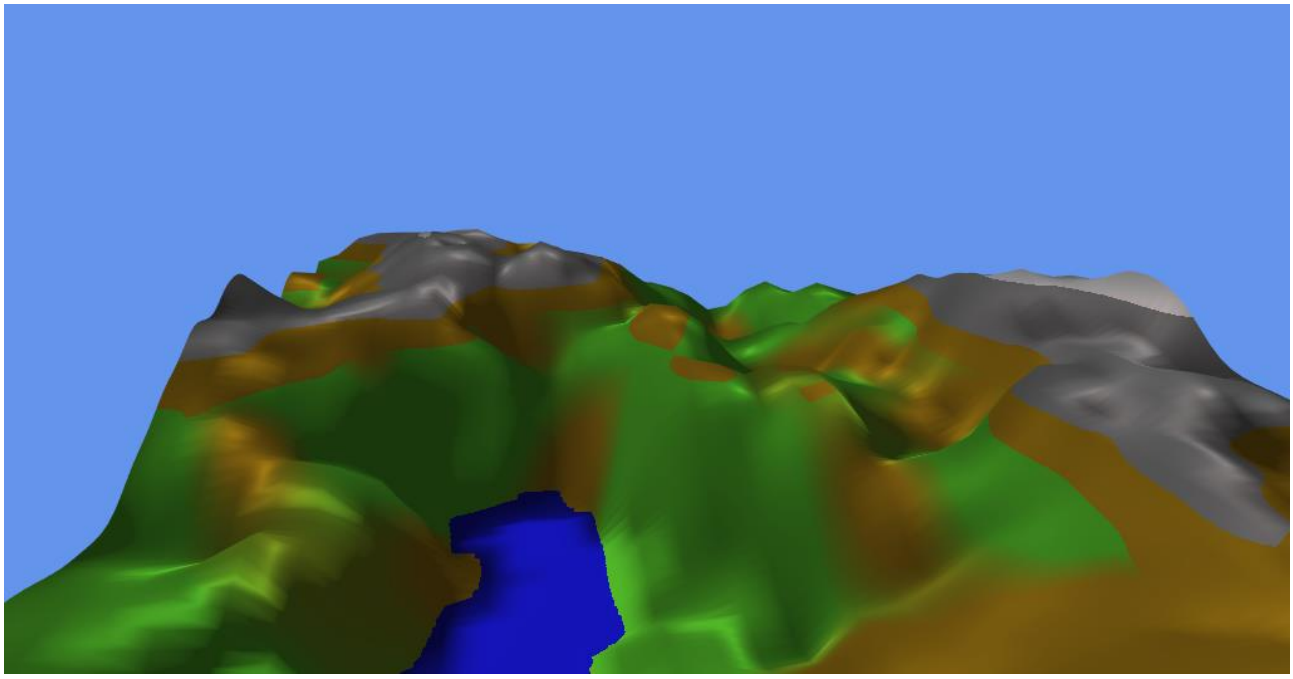
To make the elevation rules less obvious and the landscape more varied it was decided to make the ground level vary between grass and sand (if the elevation fall within the ground threshold in the first place).

This is implemented by yet again sampling a noise function, actually the same Simplex noise is copied to fragment shader, but the hash function is modified a bit. The noise value is clamped against an upper and lower threshold, resulting in grass and sand respectively. If the noise sample is between the threshold the two colors are mixed according to this.



3.7 Moving camera and wireframe mode

The final tweaks are to make the camera move and allow for wireframe rendering.
Moving camera is obtained by varying the eye position and re-calculating the camera matrix in each frame. The camera matrix uniform must be updated in each render call. It gives a nice floating (or seasick) effect.
Wireframe mode is possible since the grid both have indexes needed for triangles and lines.



4 Evaluation

- Overall
  All in all the implementation does create a scene that immediate a landscape pretty well.
- Too soft image
  The main complain about the landscape is that it is simply too soft and shiny[10].
- Unstable peaks
  If you look carefully on the animation you will notice that to top of the mountains sometimes wobble. The cause is not known, but some kind of numerical stability is expected the root cause.

5 Further work

- Improve sharpness of textures.
  To improve upon the too soft image it might help to experiment with adding high frequency noise. Another option would be to apply bump mapping to the surfaces might improve this.
- Fractal trees
  To add even more variation to the landscape trees could be added. Fractal trees are really simple to draw, however the current setup makes it a bit difficult.
  Fractal tree can be drawn by sending lines into the shader pipeline, however it is difficult for the JavaScript program to determine the suitable positions without knowing the elevation.
  Furthermore the trees should scroll along with the noise function sampling.

[10] The look somewhat resemble the look of many games for the Nintendo 64 console, which also console known for soft image and restrictions on space for textures.