Procedural Terrain

Using WebGL & Three.js

Student:
Paula Mihalcea

Professor:
Prof. Stefano Berretti

September 9th, 2020

Table of contents

Procedural generation

- Procedural generation is a method of creating data algorithmically by using a random or pseudo-random process.

- A series of parameters, such as a random seed or deterministic variables, can be modified in order to obtain substantially different data.

- In computer graphics, procedural algorithms are used to generate textures and 3D models.

An example of procedural generation used to create realistic 3D tree models.

Pros & Cons

- Pros:
    - allows for more content that does not need to be stored;
    - creates pseudo-random or even unpredictable models;
    - reduces the time needed to manually create 3D models;
    - depending on the algorithm and implementation, allows for a great variety of different models.

- Cons:
    - unpredictable, might create difficulties;
    - depending on the algorithm, models might look very similar to each other;
    - generating models like terrain by using tiles introduces the problem of tile matching.
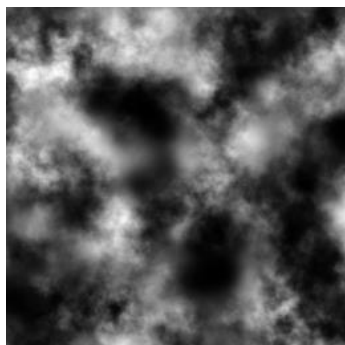
Heightmaps

- A heightmap is an image in which each pixel with coordinates (x, y) contains a certain value used to calculate its grayscale level and, by extension, the elevation along the z axis at coordinates (x, y).

- A data structure such as an array or matrix can be used in place of an actual image to store elevation values.
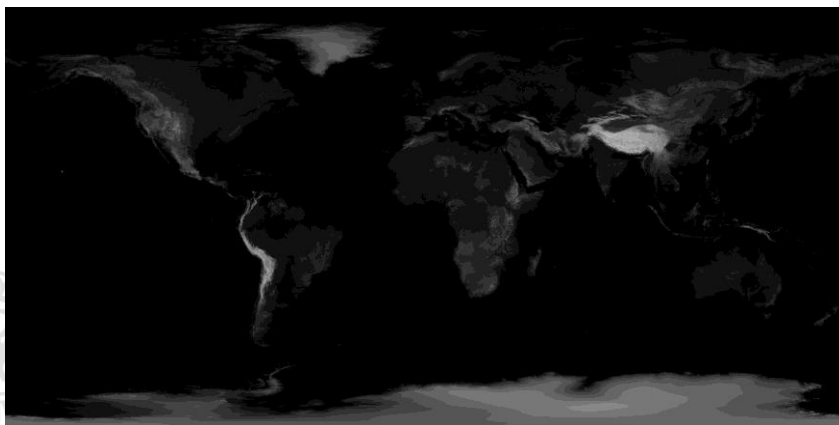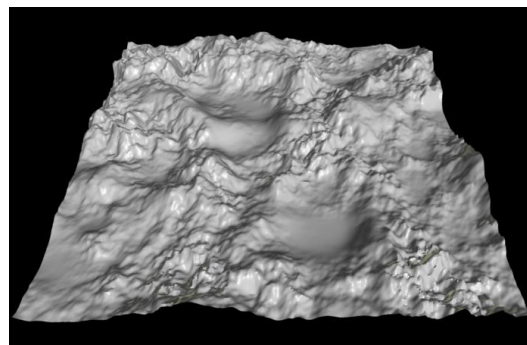

Heightmap of Earth's surface, normalized as 8-bit grayscale.

Procedural terrain

- The project creates 3D procedural terrain models by using heightmaps:

  1. First, a flat 2D plane is created;

  2. A heightmap is generated using a noise algorithm;

  3. Data contained in the heightmap is then applied to the flat plane to create elevation.

A heightmap.   The same heightmap converted to a 3D mesh.

Infinite terrain strategy

- It is not physically possible to create an infinite amount of terrain at once (GPU has limited memory).

- The illusion of infinite terrain is created by rendering only a finite number of square terrain tiles.

- Tiles are created and destroyed as the user moves the camera towards or away from them.

- In order to obtain adjacent terrain tiles that perfectly match each other along their edges, a pseudo-random algorithm is needed to generate their elevations.

Tile generation

- The tile generation process was possibly the most challenging aspect of the project.

- An algorithm had to be specifically created in order to ensure that new tiles matched the existing ones.

- In order to facilitate their indexing, terrain tiles are stored in a 3x3 2D array data structure, and are created one row/column at a time.

This screenshot from the project's early stages of development clearly shows the effects of not having perfectly matching tiles on the edges.

Tile generation algorithm

- The 2D space where the terrain lies consists of an imaginary, infinite matrix, in which each tile occupies a well-defined position given by the row and column number where it can be found.

- Given a pair of (i, j) coordinates corresponding to the tile's position in the terrain matrix, the tile's actual position in the scene (x, z), is given by the following formula (where l is the tile's side length):

$$x = l \cdot i - l \qquad z = l \cdot j - l$$

  - this formula puts in relation the tile's position in the terrain matrix with its position in the actual scene;
  - the (i, j) position of each tile in the terrain matrix is stored in a separate array for a simple and efficient handling;
  - an additional row and column index is stored in order to compute the correct imaginary matrix position.

Tile generation algorithm

Tile generation algorithm

Tile generation algorithm

Tile generation algorithm

Tile generation algorithm

Tile generation algorithm

Tile generation algorithm

Pseudo-random noise

- An effective method to get coherent elevations consists of using a pseudo-random algorithm, i.e. one that returns identical values for pixels (or, in general, points) having equal (x, y) coordinates (given the same seed every time).

- This way points on different tiles which are found at the same (x, y) coordinates (it happens along adjacent tiles borders) will have the same elevation, thus creating the illusion of a continuous piece of terrain.

- Among the many algorithms with these features we can find:
  - Perlin noise;
  - Simplex noise;
  - Diamond-Square noise.

Perlin noise

- Perlin noise creates a uniquely smooth landscape at medium parameters, but all peaks have an overall similar height.

- It has $O(N^2)$ complexity, where N is the number of dimensions (in our case $N=2$).
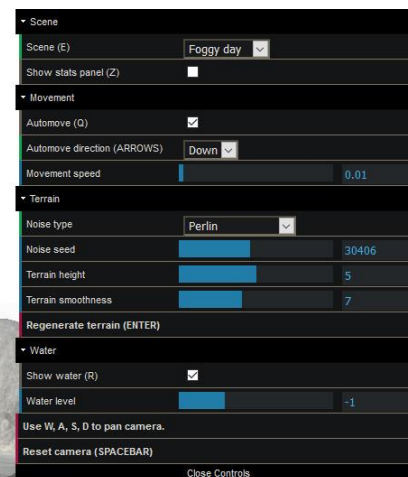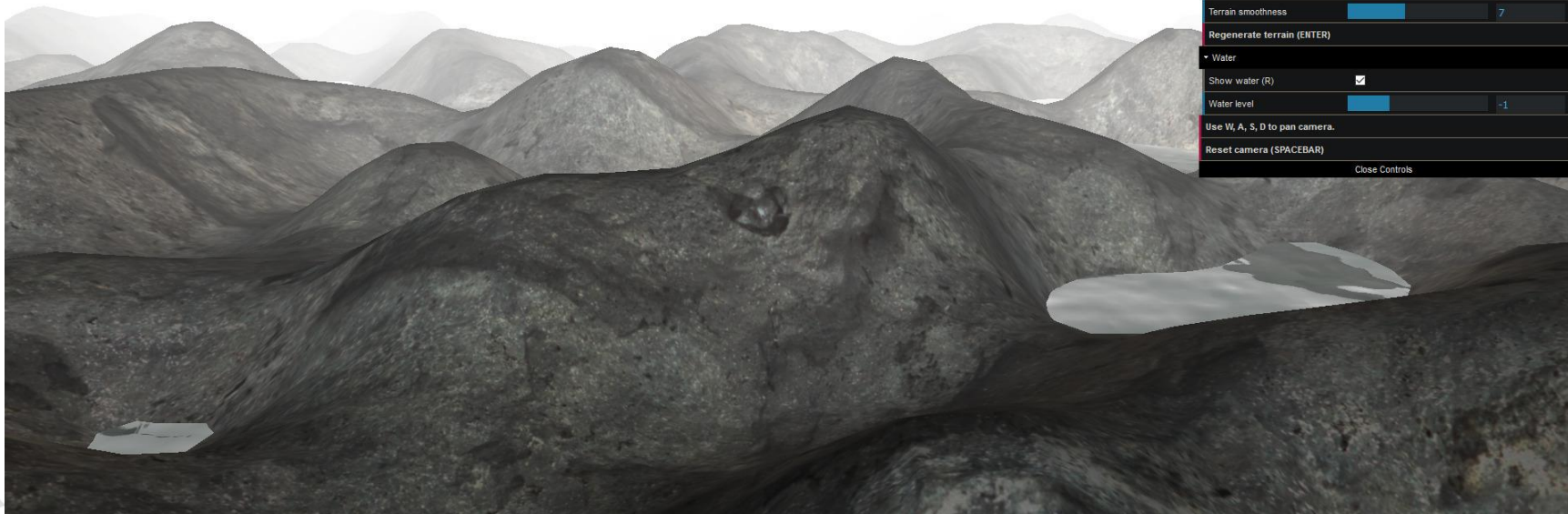
Simplex noise

- Simplex noise, a $O(N)$ variant of the Perlin algorithm designed to reduce directional artifacts, creates a similar landscape.

- Using the same seed, height and smoothness parameters as Perlin, it returns harsher and slightly higher peaks.
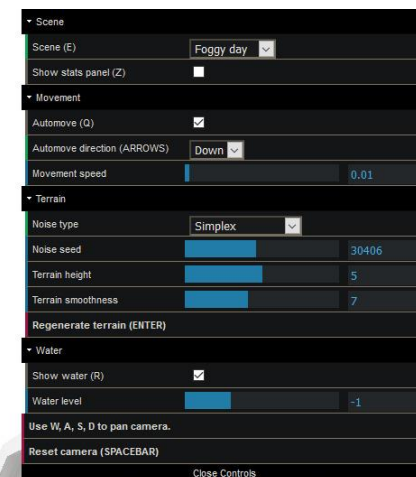
Diamond-Square noise

- A particular, tileable implementation of the Diamond-Square algorithm has been used in this project.

- The first thing that is usually observed about it is the fact that it's noticeably slower than the other two, especially for higher iterations.
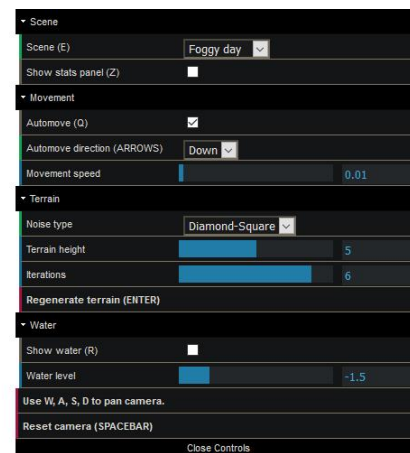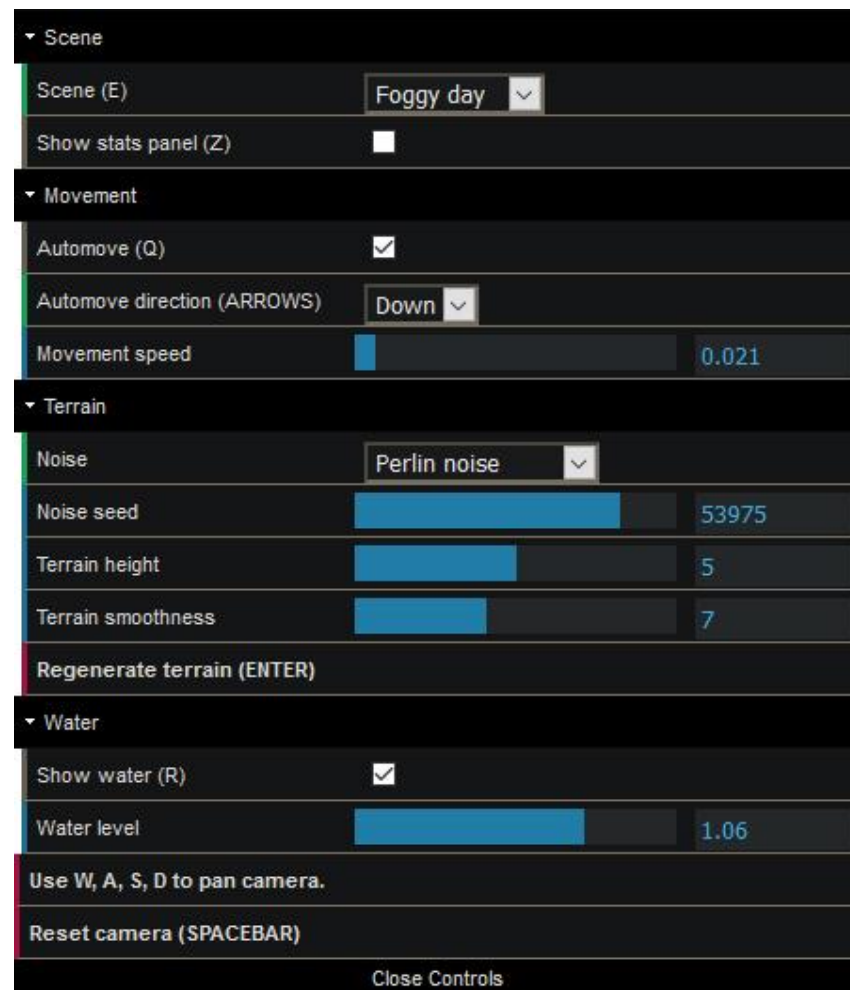
Diamond-Square noise

- However, unlike the previous two noise models, it creates a uniquely varied view with distinct plains and hills.

- It generates a repetitive landscape, but using an adequate texture and a large enough scale virtually eliminates this defect.
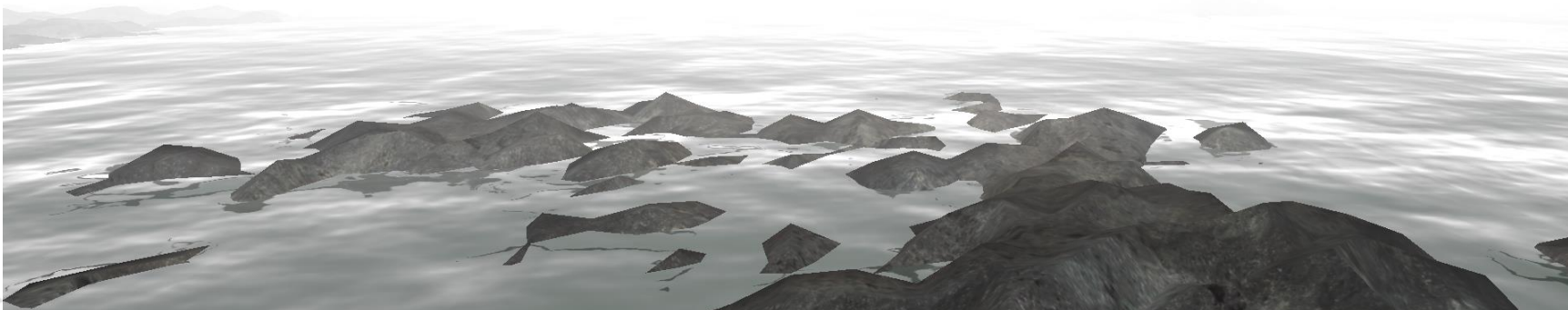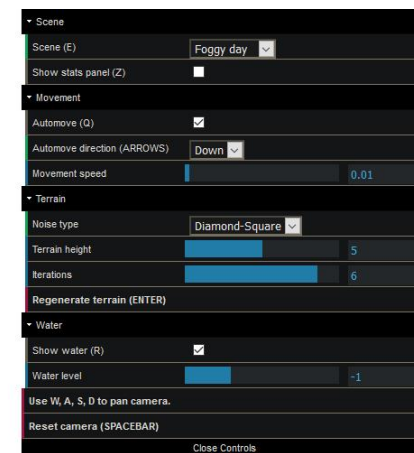
Controls

- A series of parameters can be set in the GUI available within the WebGL canvas.

- The GUI has been created using the Google Data Arts Team **dat.GUI** library, and is set to dynamically change its options based on the current selection (e.g. some sliders appear only when a certain noise type is selected).

- All controls have been tested and limited to reasonable minimum and maximum values, including the camera zoom and panning, in order to prevent the user from creating unrealistic scenes and/or finding the terrain's margins.

Controls

- A few buttons have also been implemented in order to:
  - regenerate the terrain with the new values set by the user (a manual refresh is needed in order to ensure that new tiles will match the existing ones);
  - reset the camera.

- In addition to the GUI controls, a series of keyboard shortcuts have been implemented for better convenience.

- All available shortcuts are specified in the GUI next to the parameter which they control.

- By left clicking and dragging on the canvas, the mouse and its wheel can also be used to move the camera around and zoom in/out.

Scene selection
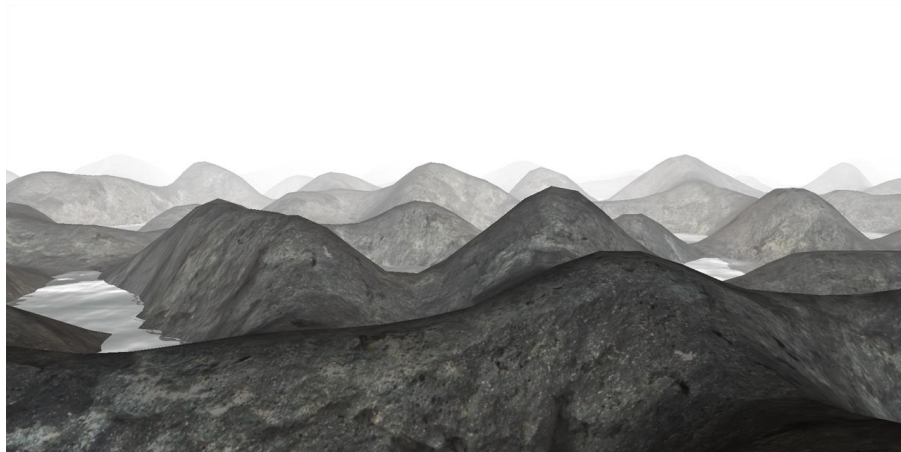
Foggy day
Featuring a thick fog that prevents the user from seeing the margins of the terrain matrix

Starry night
Featuring a darker fog *and* a textured skybox specially created for the project
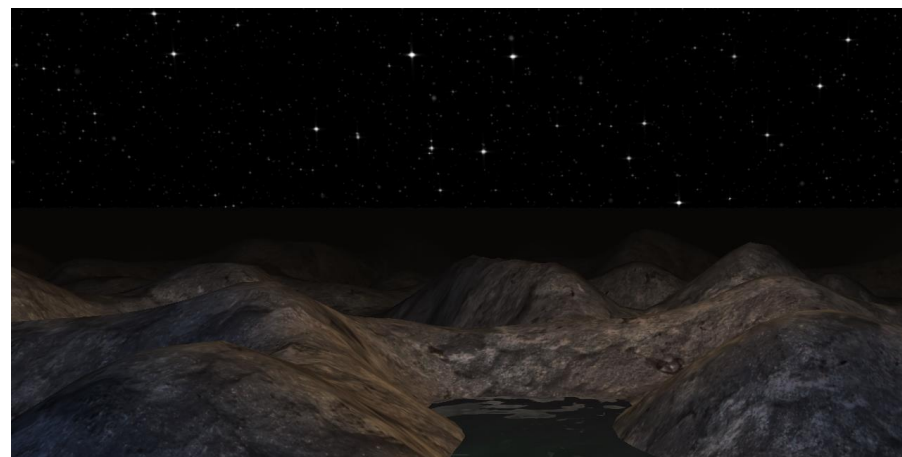
Water

Customizable water level

A slider in the graphical user interface allows the user to change the water level within reasonable limits (based on the current terrain height)
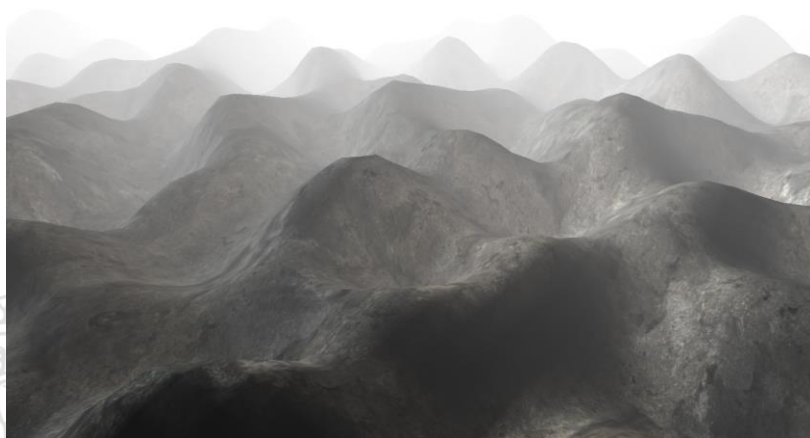
No water

The water shader can be easily toggled on and off by pressing 'R'

Optimizations

- Different measures have been taken in order to optimize the script's execution, among which:
  - **terrain scale**: the initial tile length and segments, equal to respectively 3000 and 1000, have been reduced to 500 each, resulting in a dramatic performance improvement; this change has been made virtually unnoticeable by adjusting the camera to an adequate position;
  - **tile generation**: instead of allocating new meshes every time a new tile is needed, the program updates the position and vertices of the tile that is going to be removed - resulting in a significantly smaller drop in the framerate when creating new terrain;
  - **scene fog, lights and textures** (e.g. the starry skybox) are similarly created only once when selected for the first time, then simply updated as the user chooses a new scene.

Optimizations

- Different measures have been taken in order to optimize the script's execution, among which:

    - **limited 'if's**: a series of flags used during development has been rewritten in more suitable ways in order to limit the number of 'if' checks at each update to only three;

    - **dictionaries**: all parameters have been stored in handy dictionaries for quicker editing, and the code has been carefully organized and commented for a better understanding;

    - **libraries**: JavaScript libraries have been imported in such a way that a local server is no longer needed to run the program (as had been the case in the first stages of development).

Conclusions

- The project efficiently integrates a number of 3D graphics and noise libraries, as well as many interesting demos and examples.

- It also has a fully functional GUI, completed by a series of keyboard shortcuts and controls that allow the user to move the camera and change many scene parameters.

- In conclusion, the project fully achieves its original aim to create an infinite terrain in the JavaScript programming language with WebGL and Three.js, and does so while also introducing additional features and keeping the user safe from immersion-breaking settings and controls, for an overall pleasant experience.

References

Project
[1] Inigo Quilez, "Raymarching terrains", 2002
[2] Stefan Gustavson, "Simplex noise demystified", Linköping University, 2005

[3] Ricardo Cabello, "three.js", 2010
[4] Joseph Gentle, "noiseJS", 2012
[5] Tatarize, "Olsen noise", 2016
[6] Google Data Arts Team, "dat.GUI", 2011
[7] Anatoliy, "Random color generator", 2009

[8] Three.js Fundamentals, 2017
[9] Stephan Baker, "Terrain Generation with Perlin Noise", 2017
[10] George Campbell, "Procedural terrain generation with blocks", 2019
[11] John Clay, "Learning 3D Graphics With Three.js | Dynamic Geometry", 2018
[12] John Clay, "Learning 3D Graphics With Three.js | Procedural Geometry", 2018
[13] John Clay, "Learning 3D Graphics With Three.js | Advanced Materials and Custom Shaders", 2018
[14] Alexandra Etienne, Jerome Etienne, "Casting Shadows", 2012
[15] Laurent Duveau, "Visual Studio Code and local web server", 2016

[16] AlteredQualia, "WebGL procedural terrain"
[17] Ricardo Cabello, "three.js examples – orbit controls", 2012

[18] Giles Hodges, "Mountain rock seamless texture", 2014
[19] Ricardo Cabello, "Water Normals Texture", 2014
[20] Anonymous, "Night Sky Background"

Slides
[21] Wikipedia, "Procedural generation"
[22] Procedural Content Generation Wiki
[23] Wikipedia, "Heightmap"
[24] Wikipedia, "Simplex noise"
[25] Paula Mihalcea, "Procedural terrain", 2020

Thank you for your attention!

The project can be viewed live at:
https://paulamihalcea.github.io/Procedural-Terrain.

The complete source code, along with this presentation, is available at:
https://github.com/PaulaMihalcea/Procedural-Terrain.

Press 'M' in the project's live page for your daily dose of memes.