Procedural 3D Planet Generation in WebGL

Riccardo Grazzi, Cosimo Casini

Università di Firenze, Scuola di Ingegneria

Laurea Magistrale in Ingegneria Informatica

December 17, 2016

Introduction

UNIVERSITÀ DEGLI STUDI FIRENZE
DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

- In this work we discuss the idea of a 3D Planet Generator that uses the technique of Procedural Generation.
- The main idea is to use a noise function to model the characteristics of a real planet like mountains and clouds.
- We have achieved this goal using the WebGL and the ThreeJS library.

Procedural 3D Planet

Project steps

UNIVERSITÀ DEGLI STUDI FIRENZE
DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

1. Find a noise function
2. Apply the noise function to a sphere for terrain generation
3. Texturing and shadow mapping
4. Add elements as water and clouds
5. Implement dynamic Level Of Detail (LOD) for the terrain

Procedural Generation

UNIVERSITÀ DEGLI STUDI FIRENZE
DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

- Procedural content generation is the algorithmically generation of data using a pseudo-random process that results in an unpredictable range of possible values.
- The main use of this technique is the creation of procedural textures and worlds for videogames.
- The noise that we are trying to achieve must be very different from white noise to be visually appealing.
- There are many different kinds of noise functions that can be used. All of them try to smooth pseudo-random values to make the noise more interesting.

Diamond Square Noise

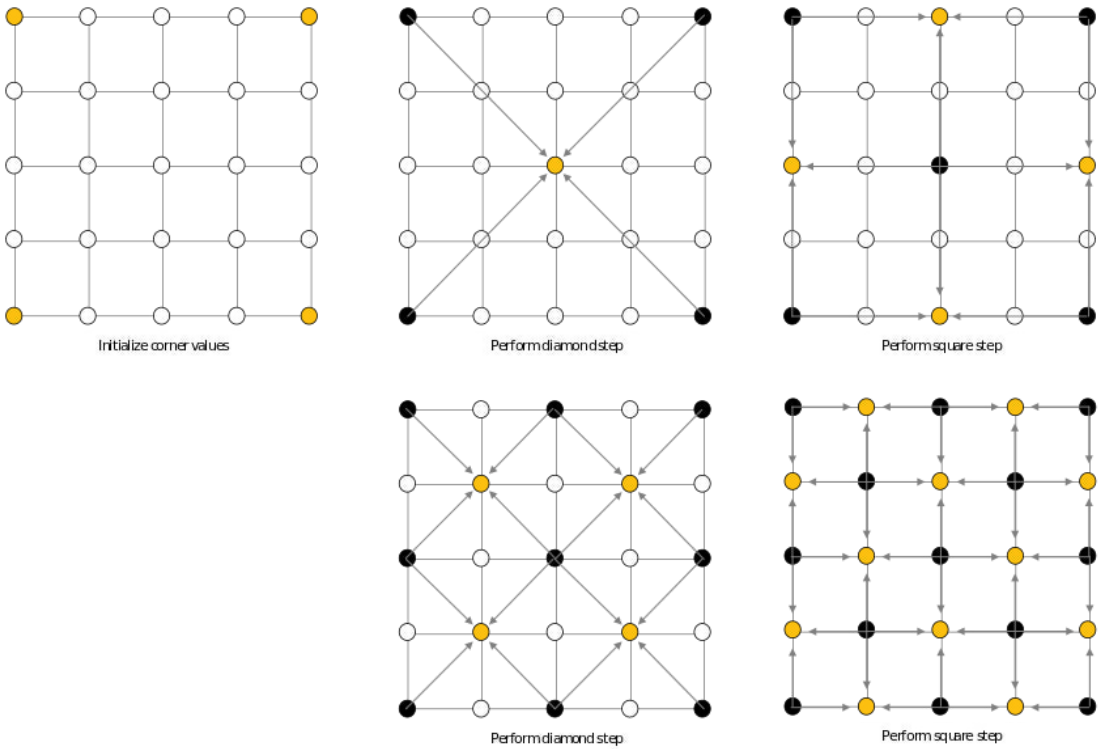
UNIVERSITÀ DEGLI STUDI FIRENZE
DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

- Used mainly to create 2D heightmaps. Given 2^{n+1} points on a planar grid with the 4 corners' noise values already set, the algorithm operates as follows:
  1. **Diamond Step:** set the centre point by averaging the four corner noise values and adding or subtracting a random amount of noise.
  2. **Square Step:** set the four mid points between the corners (i.e. North, South, East and West), by averaging the two noise values on either side of the point in question and adding or subtracting a slightly smaller random amount of noise.
  3. These 2 steps are iterated until all the pixels are perturbed in a top-down fashion, decreasing the random quantities at each step.

Diamond Square Noise Properties

UNIVERSITÀ DEGLI STUDI FIRENZE
DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

1. Very fast compared to the other techniques.
2. Depends too much on the first steps and the initial points.
3. The top-down generation is very limiting in some contexts.
4. It's difficult to apply to non-plane geometries.



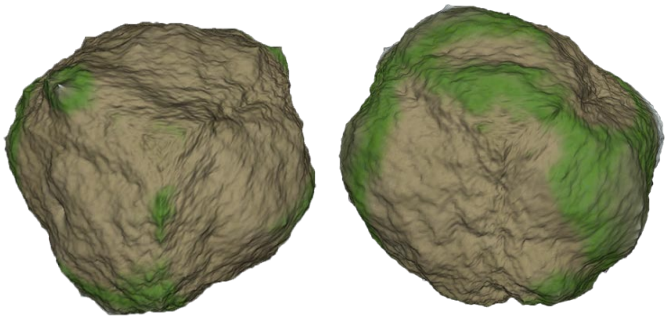
Diamond Square Noise applied to a Sphere

UNIVERSITÀ DEGLI STUDI FIRENZE
DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

- The Diamond Square algorithm can't be directly applied to a sphere. But it can be applied to every face of a cube successively converted in a sphere.

  Notice: the value of every point on each cube edge must be the same for each face to ensure continuity.

- **Issue:** the cube can be still visibile due to the shape of the triangles near edges and corners and to the fact that only the edge points of nearby faces are considered when the noise is generated.

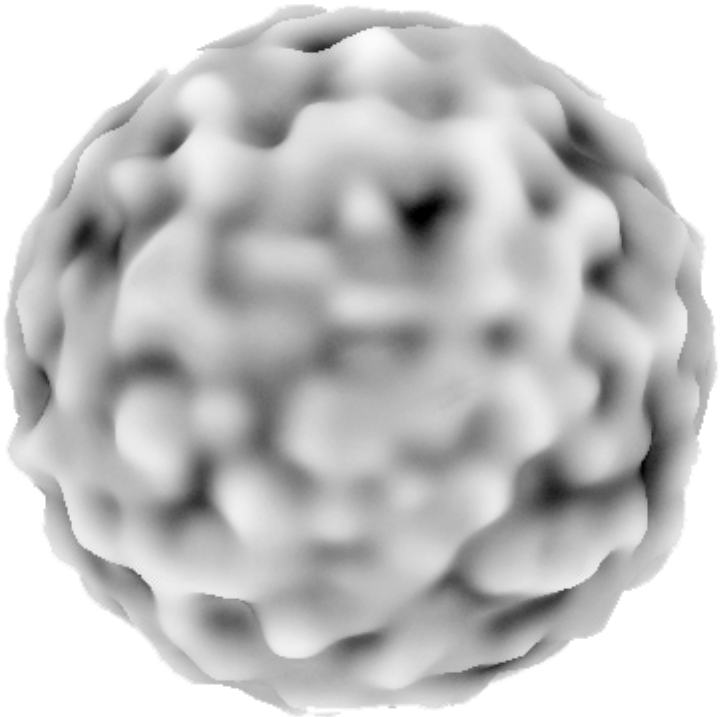


UNIVERSITÀ DEGLI STUDI FIRENZE
DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Perlin Noise

- It's a gradient noise developed by Ken Perlin in 1983.
- It's based on 3 steps:
  1. **Grid definition:** define an n-dimensional lattice grid. At each point of the lattice (4 for 2D, 8 for 3D) assign a pseudo-random gradient vector (lattice gradient).
  2. **Dot Product:** computation of dot products between the cell gradients and the distance vector from the given point to the lattice points.
  3. **Interpolation:** between the 2^n dot products previously computed. The node contributions could be given by different functions of the distance between the node and the point. These can be linear, higher order polynomials (that give better results) or similar. In this project we use a **tricosine** function (approximates a more expensive **tricubic**).

Perlin Noise



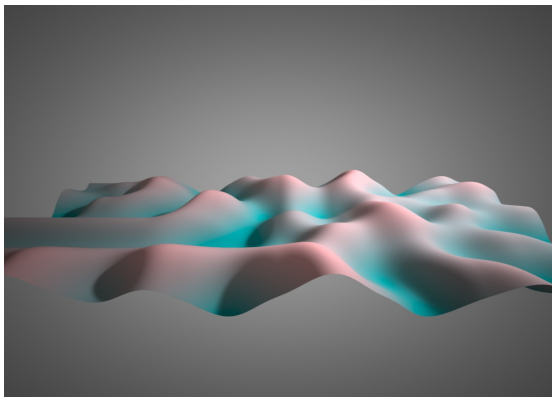
Perlin noise applied to a sphere

Fractional Brownian Motion

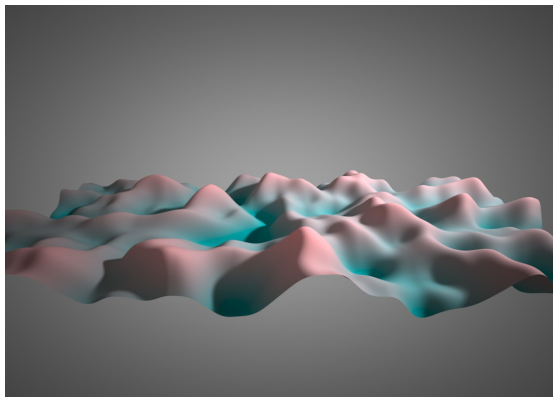
UNIVERSITÀ DEGLI STUDI FIRENZE
DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

- Perlin Noise alone gives a nice smooth transition between random values, but it's not similar to the characteristics that we see in the real world.
- Take mountains for example. If seen from afar they shows certain features (tops and peaks) that are somewhat similar if seen more closely (stones and rocks).
- From here comes the idea of **summing** noise at different scales to recreate these features.

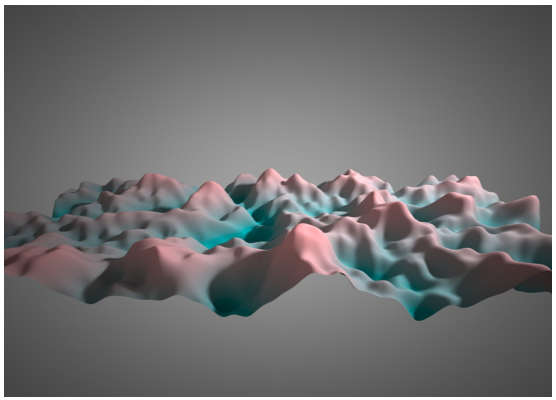
FBM - Different numbers of levels



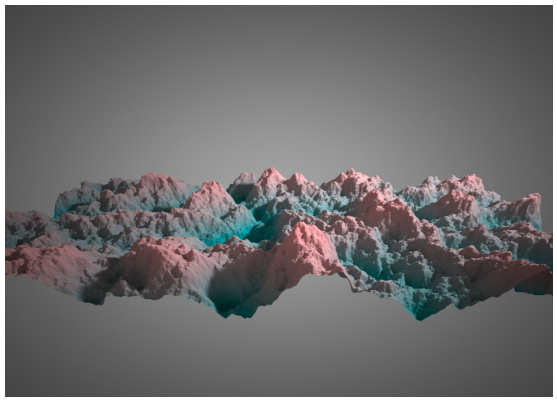
1 Level



2 Levels



3 Levels



10 Levels

Fractional Brownian Motion Function

UNIVERSITÀ DEGLI STUDI FIRENZE
DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

- Noise with low frequency and high amplitude represents the largest terrain features, while high frequency low amplitude noise represents smaller ones. The more the **levels** of noise, the more detailed the terrain.

```
FBMNoise(x, y, z, resolution, smoothness, lacunarity, levels):
    var amplitude = getAmplitude(smoothness,levels);
    var res = resolution; var c=0;
    for(var l =0; l < levels; l++)
        res =res*lacunarity;
        var noise = perlinNoise(x, y, z, res);
        c += noise*amplitude;
        amplitude/=smoothness;
   return c;
```

Fractional Brownian Motion

UNIVERSITÀ DEGLI STUDI FIRENZE
DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

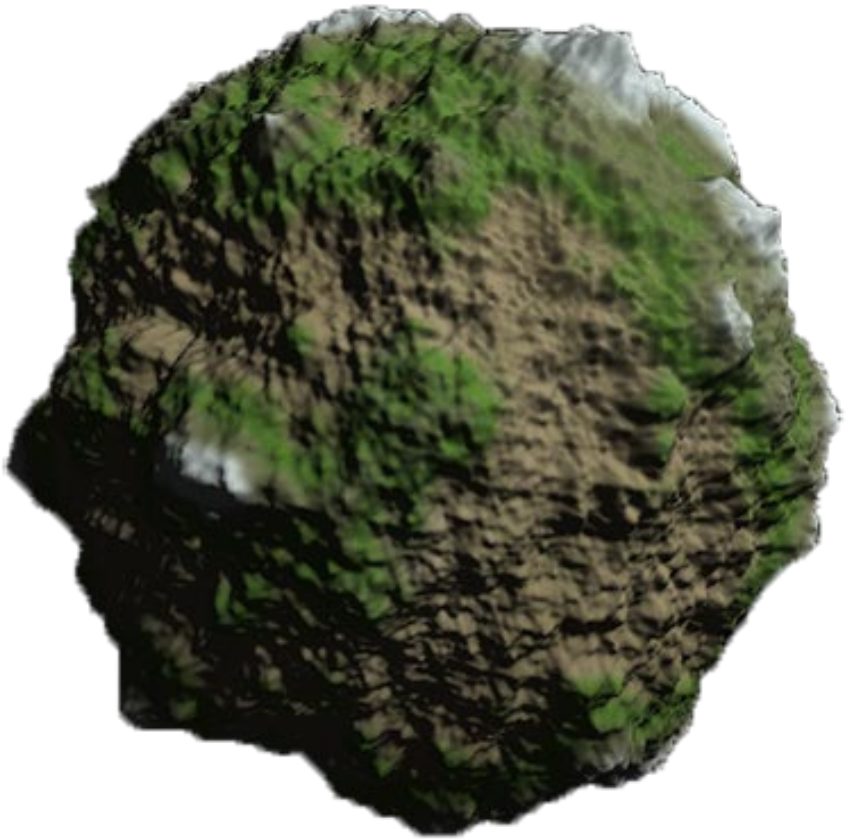
- **Resolution:** represents the lowest frequency and the lattice dimension at the first noise level.
- **Lacunarity:** determines how quickly the frequency increases for each successive level (if *lacunarity* = 2 the levels are called "octaves").
- **Smoothness:** determines how quickly the amplitude decreases for each successive level.
- **getAmplitude(s,l)** sets the amplitude of the first octave to be:

$$\frac{s^{l-1}}{\sum_{i=0}^{l-1} s^i} \tag{1}$$

With this equation the resulting noise is constrained between [-1,1] as the single level noise.

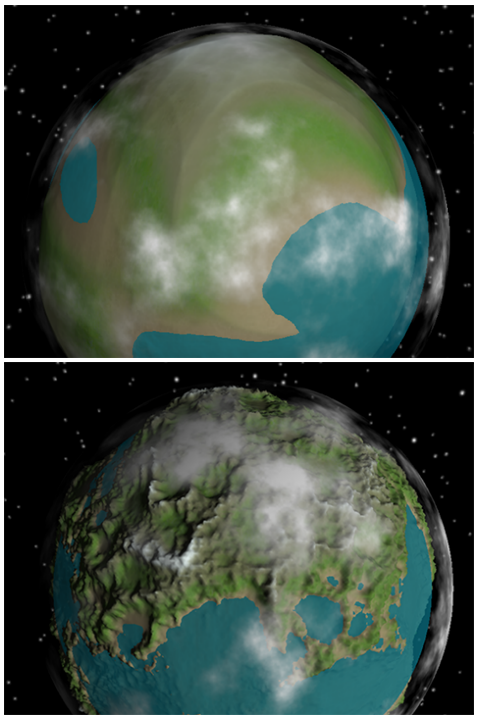
Fractional Brownian Motion

UNIVERSITÀ DEGLI STUDI FIRENZE
DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

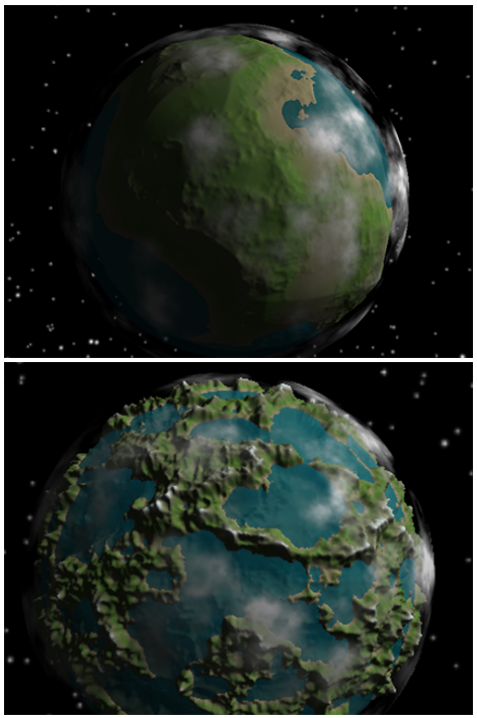


FBM noise applied to a sphere

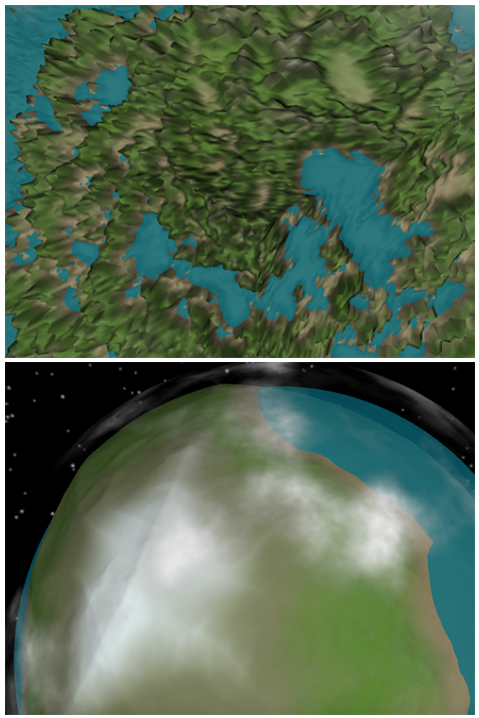
Noise Parameters Examples



Lacunarity: 1.5, 3.0



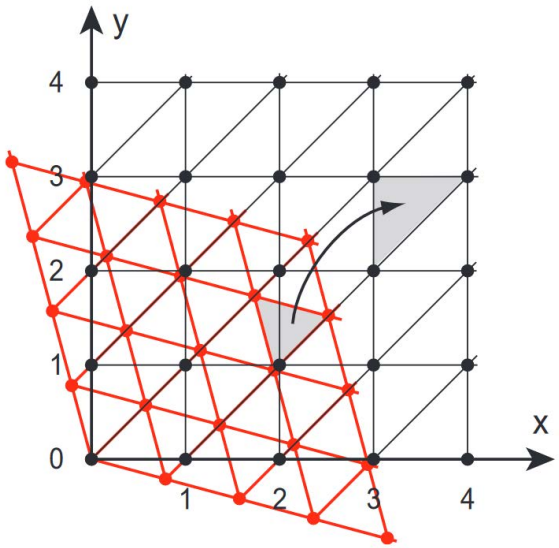
Resolution: 0.3, 3.0



Smoothness: 1.2, 2.5

Simplex Noise

- Variant of Perlin noise with a lower computational overhead in higher dimensions and a well defined and easy to compute gradient.
- Instead of using an hypergrid, the Simplex noise divides the space into simplices. The computational cost decreases from $O(2^n)$ to $O(n+1)$, where n is the number of dimensions.



Simplices 2D into a grid cell

Simplex Noise Steps

UNIVERSITÀ DEGLI STUDI FIRENZE
DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

- For each point of the noise with coordinates x, y, z there are four steps, given the number of dimensions n:
  1. **Coordinate Skewing:** convert the points with the formula:

$$\begin{aligned} x' &= x + (x + y + z) * F3 \\ y' &= z + (x + y + z) * F3 \\ z' &= z + (x + y + z) * F3 \end{aligned} \tag{2}$$

$$F3 = \frac{\sqrt{n+1} - 1}{n}$$

This is a transformation to a coordinate system in which some of the simplices form an hypercube with edges parallel to the cartesian axis

Simplex Noise Steps

UNIVERSITÀ DEGLI STUDI FIRENZE
DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

2. **Simplical subdivision:** determinate in which simplex the point falls by sorting the coordinates.
3. **Unskewed displacement vector:** unskewing the coordinate using the inverse of skewing formula, then subtract the input coordinate. This vector, called "unskewed displacement vector", is used to find the distance d from the point to the simplex vertices.
4. **Gradient definition and Kernel summation:** each simplex vertex is added back to the skewed hypercube's base coordinate to compute the gradient *grad*. For each vertex kernel contribution is determined using the equation:

$$(max(0, r^2 - d^2))^4 * (< \Delta x, \Delta y, ... > \cdot < grad.x, grad.y, ... >) \quad (3)$$

Where r^2 is usually set to either 0.5 or 0.6.

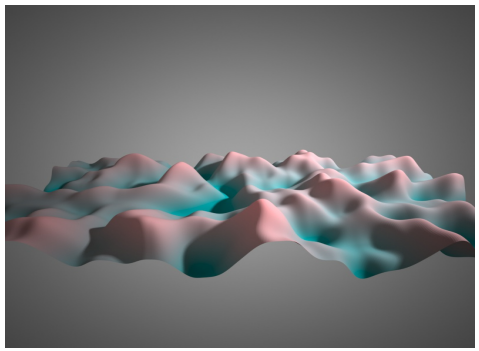
Ridged Multifractal Noise

UNIVERSITÀ DEGLI STUDI FIRENZE
DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

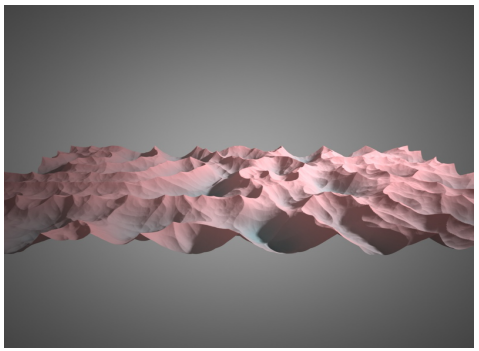
- To improve the quality of the Perlin/Simplex noise, and add features like mountain ridges, we can apply the following transformation given a single level noise value f in the [-1, 1] range:

  1. pf is the noise at the previous octave.
  2. $w = min(max(pf * 2, 0), 1)$
  3. $f = (1 - |f|)^2 * w$
  4. $f = f * 2 - 1$

  and repeat the process for each successive level.



Perlin Noise

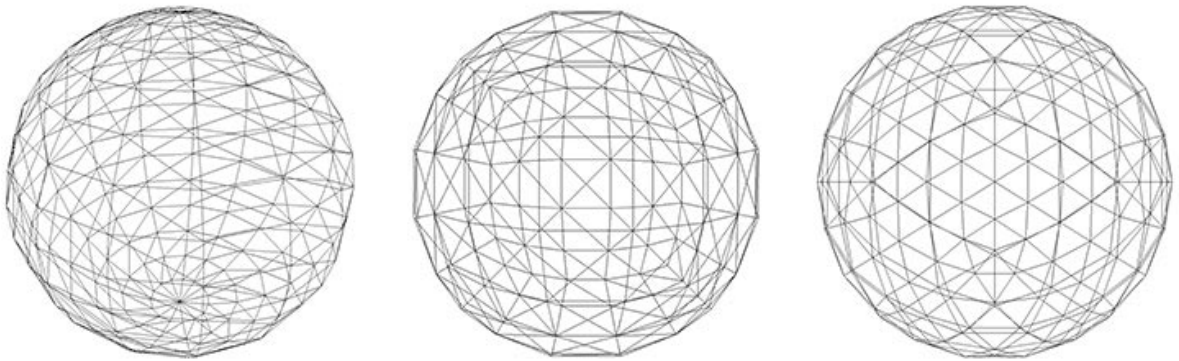


Ridged Multifractal Noise

Sphere Vertices

UNIVERSITÀ DEGLI STUDI FIRENZE
DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

- What are the best ways to set vertices onto a sphere?
  1. **Parallels and meridians**
     - Pros: Easy and simple.
     - Cons: densely distributed near the poles, artefacts can be noted when the points are perturbed or if a planar texture is applied.
  2. **"Spherified Cube"**
     - Pros: no poles problem, simple texturing (one square texture per face), perfect for grid-based noise algorithms.
     - Cons: corner artefacts due to different vertex densities.
  3. **Icosahedron**
     - Pros: vertex almost evenly distributed.
     - Cons: grid-based noise algorithms can't be directly applied.



Vertex perturbation

- Given the sphere vertices and a noise value per vertex in the [-1,1] range, we can **perturb** the sphere points accordingly.
- Let $\vec{p}$ be the point vector w.r.t the sphere center, we have

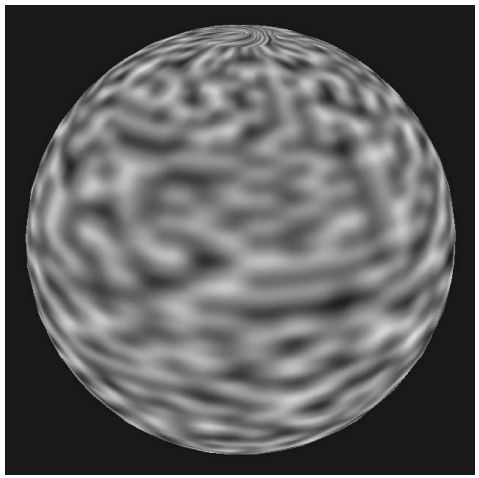
$$\vec{p} = \vec{p} + \frac{\vec{p}}{||\vec{p}||} h\, r \tag{4}$$

where h is the scalar noise value and $r \in [0, \infty)$ is the terrain's roughness.

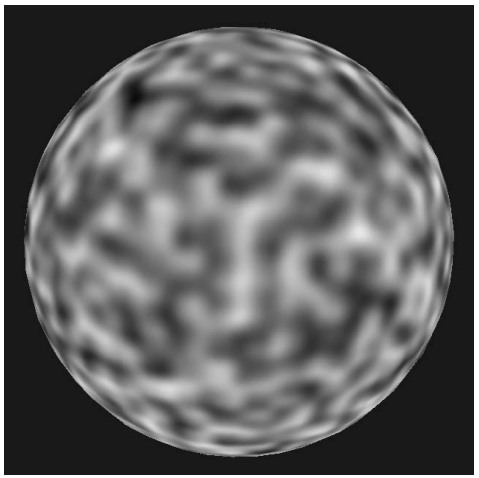
Vertex perturbation

UNIVERSITÀ DEGLI STUDI FIRENZE
DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

- Regarding Perlin, Simplex and Ridge noise even if the sphere surface is 2-dimensional we used the 3D version of the noise computed using the 3D spherical coordinates to avoid artefacts that come from the application of 2D noise to a sphere.



Perlin Noise 2D



Perlin Noise 3D

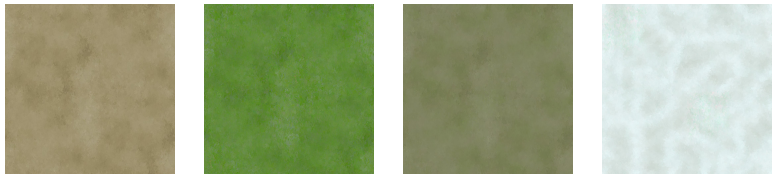
Surface Normals

UNIVERSITÀ DEGLI STUDI FIRENZE
DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

- After vertex perturbation, the new surface normals have to be computed. There are several ways to do that:
  - **Per face Normals** → Very poor quality shadows.
  - **Per face Normals + interpolated vertex normal** → Better shadows but difficult to compute in Vertex Shader.
  - **Using the noise function and the tangent, bitangent method** → Difficult to apply to a sphere, noise function must be calculated for at least 2 more points (Perlin and Simplex noise only).
  - **Using the noise gradient** → The gradient must be defined analytically in all the sphere points (Simplex noise only).
- We mainly used the second method in this project, but we also tried the last two with unpleasing results.

Texturing

- To simulate the earth terrain we use four different textures, each one representing a different biome: sand, grass, rock and snow.



- The final colors are set in the fragment shader, where the different textures are Hermite interpolated depending on the "height" of the terrain.
- On a perturbed sphere the "height" is the difference between the distance from the point to the sphere center and the sphere radius. Lower height values correspond to sand, higher to snow, passing through grass and rock.

Issues: Stretching of the textures: $\rightarrow$ Triplanar texturing.

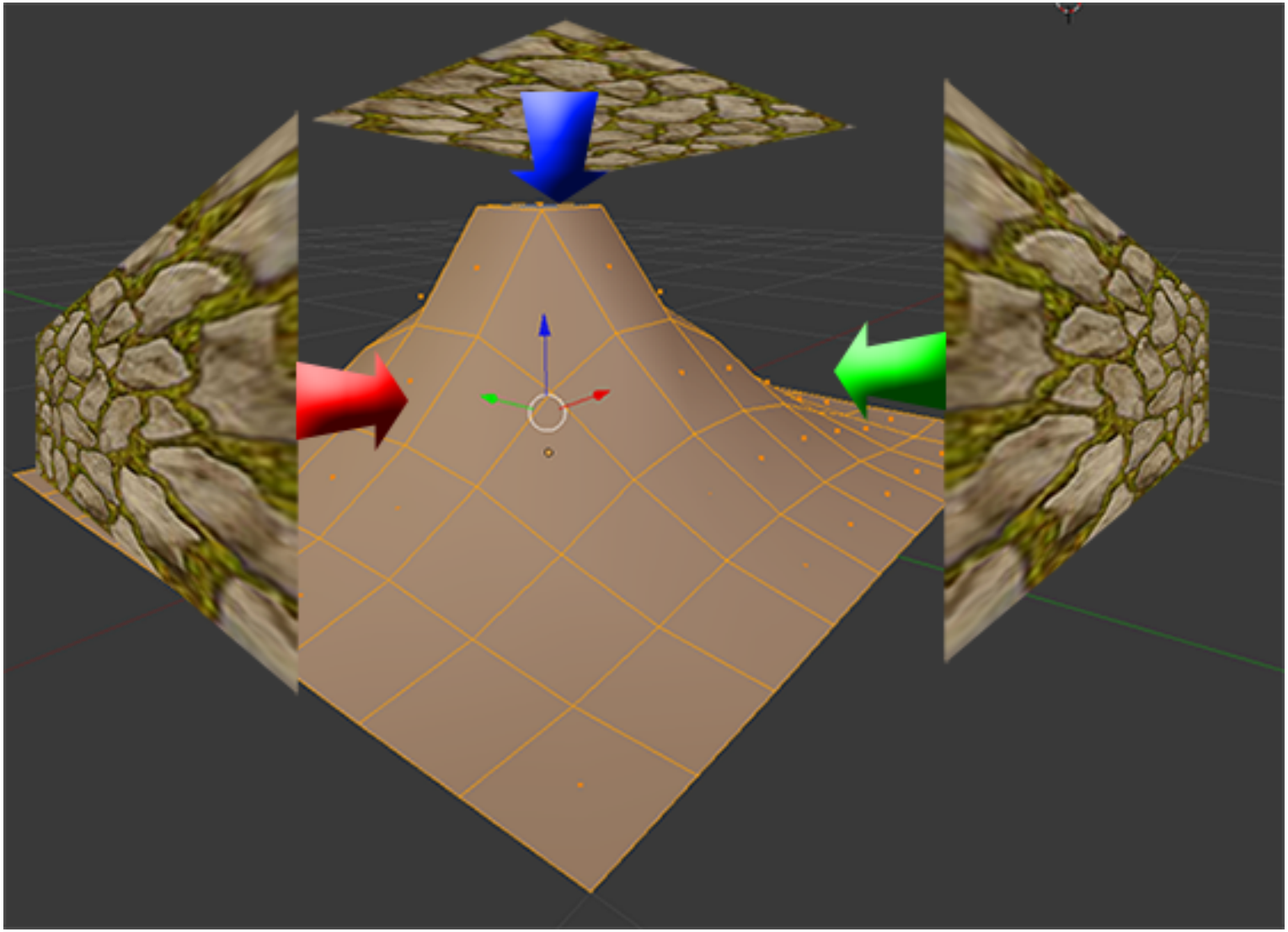
Triplanar Texturing

UNIVERSITÀ DEGLI STUDI FIRENZE
DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

- Triplanar mapping uses the same texture 3 times, one for each cartesian axis (x, y, z). Each point on the surface linearly interpolates contributions for each texture depending on the surface normal: if the normal points more to the x-axis direction, for example, the corresponding texture contribution will be higher.
- In this way the stretching of the textures caused by the direct application of a square texture to the planet disappears, even if some artefacts could still be noted.
- This technique is used in combination with the previous, by repeating it 4 times for each of the biome textures and interpolating the results as before.

Triplanar Texturing

UNIVERSITÀ DEGLI STUDI FIRENZE
DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE



Water Rendering

UNIVERSITÀ DEGLI STUDI FIRENZE
DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

- To render the water we reuse Triplanar Texturing adding the idea of moving waves.
- The water is initially represented as a blue sphere with radius equal to the planet sphere. The texture used is a normal texture in which each RGB value represents a 3D normal of a water surface point. For each sphere point, the normal is computed by summing contribution from the normal texture. These contributions come from Triplanar Texturing and from various periodic functions that simulate the moving waves.
- Shadows and light reflections are computed in the fragment shader using the normals.
- This is a simple way to achieve the moving waves effect on a planet but is affected by the problems of Triplanar Texturing and the fact that we are using a single texture.

Water Rendering

UNIVERSITÀ DEGLI STUDI FIRENZE
DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Water Sphere with Triplanar Normal Texturing

Clouds Rendering

UNIVERSITÀ DEGLI STUDI FIRENZE
DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

- To simulate clouds we used a noise texture applied to a sphere with radius equals to $r * roughness$ where r is the radius of the planet sphere.

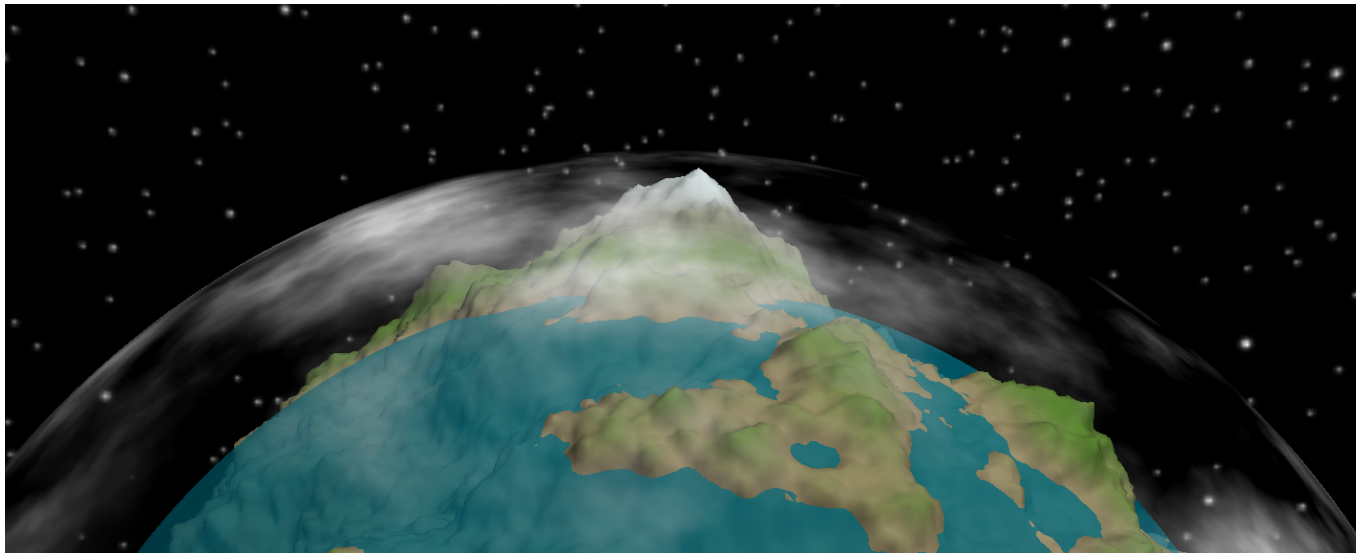
- The texture is procedurally generated using 3D Perlin noise directly into the final fragment shader, instead of having that previously made in a different render. This method overcomes a limit of WebGL, the lack of 4D texture support, to simulate the motion of the clouds, without even the need to store those very large textures in memory.

- Instead of using the 4D version of Perlin noise, and setting the 4th dimension as time, we used the 3D version and moved and rotated the spherical surface into the noise lattice through time, achieving similar effects of the expensive 4D Perlin noise at a much smaller cost.

Clouds Rendering

UNIVERSITÀ DEGLI STUDI FIRENZE
DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

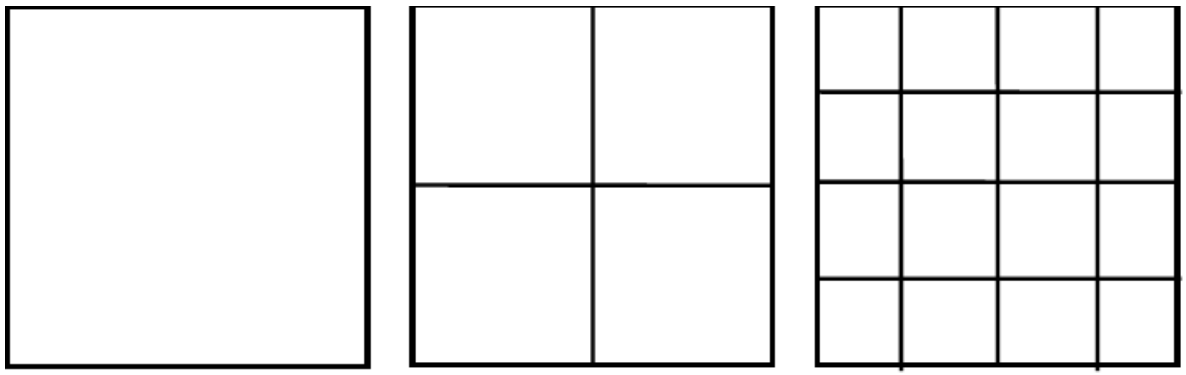
- In the fragment shader the final color is computed as follows:
  - Sum to the base grey color components the same scalar value from the noise.
  - Compute alpha values: higher values of noise correspond to higher alphas. Changing the opacity is a way to achieve the clouds' transparency.



Chunked LOD

UNIVERSITÀ DEGLI STUDI FIRENZE
DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

- To implement **Level Of Detail**, we used the chunked LOD technique. Six quad-tree are created for each face of the "spherified cube".
- Every quad-tree represents a 2D flat grid starting on $(0, 0)$ and ending in $(1, 1)$. A mapping function, converts each point on the grid in the corresponding 3D sphere point given the index of the face it belongs to (from 1 to 6) and the sphere radius.
- Tree's nodes represent parts of the grid: the root covers the full grid, it's four children divide the grid in four squares and so on.



Quad-Tree Traversing

UNIVERSITÀ DEGLI STUDI FIRENZE
DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

- Each node is represented by it's center (for the root $(0.5, 0.5)$) for which the corresponding perturbed vector on the sphere is computed.
- For every render cycle the quad-trees are traversed until the nodes to be drawn are reached. Those are selected depending on the distance d from the camera to the perturbed vector of the node center. The node to be drawn is the first in the path, starting from the root, to satisfy:

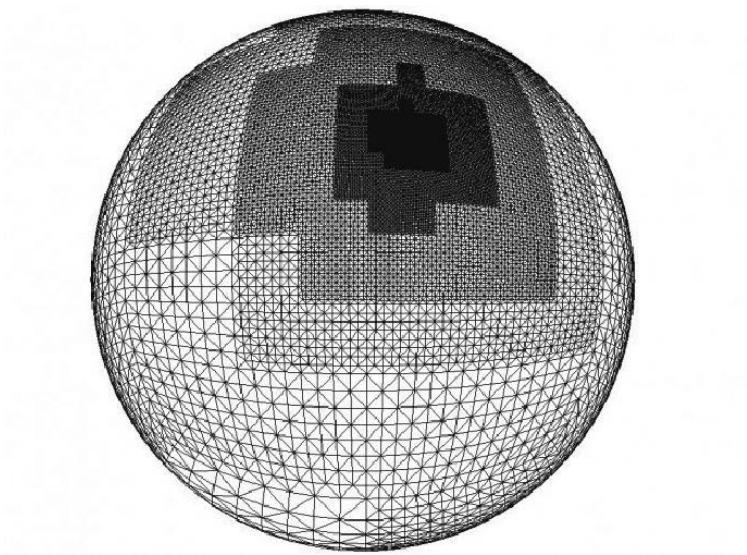
$$\frac{C * radius}{2^l} < d \tag{5}$$

Where C is a constant and l is the depth of the node in the quad-tree.

Chunk Drawing

UNIVERSITÀ DEGLI STUDI FIRENZE
DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

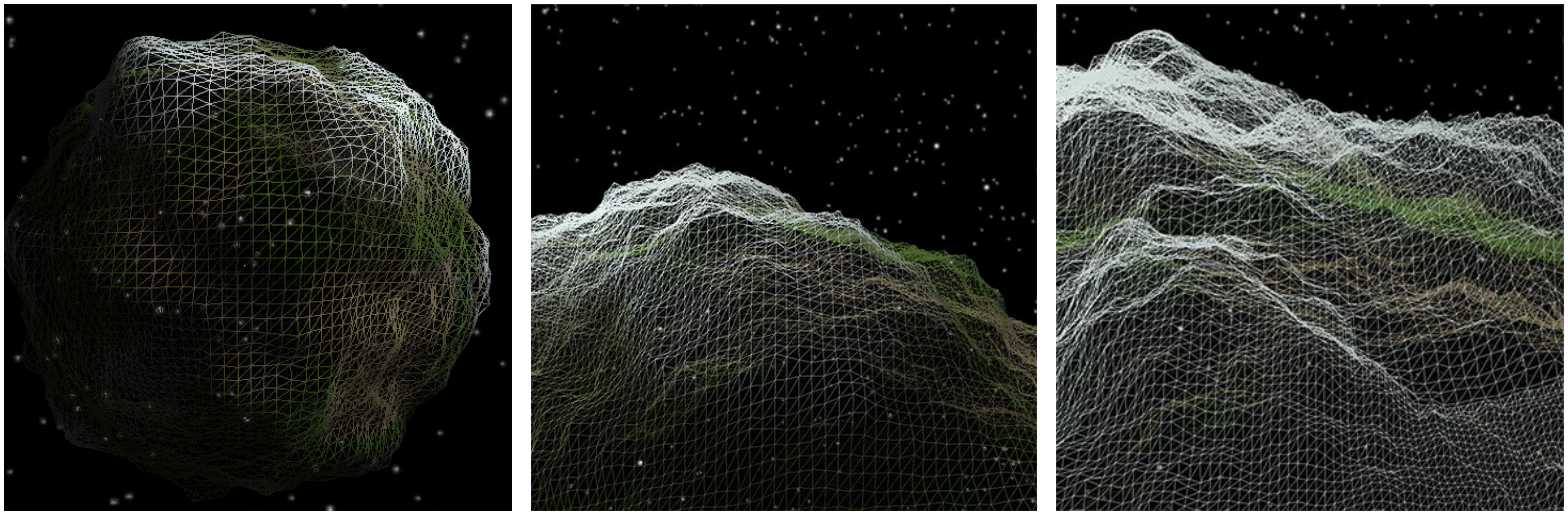
- For each node to be drawn, a Plane geometry (the *chunk*) corresponding to it's position on the 2D grid is generated.
- Each point of this geometry is then projected onto the sphere and perturbed by the noise function.
- only one node per path is drawn, thus at each render cycle all the other nodes in the path must be removed from the scene.



Quad-Sphere wireframe example without noise

Chunked LOD Result

UNIVERSITÀ DEGLI STUDI FIRENZE
DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

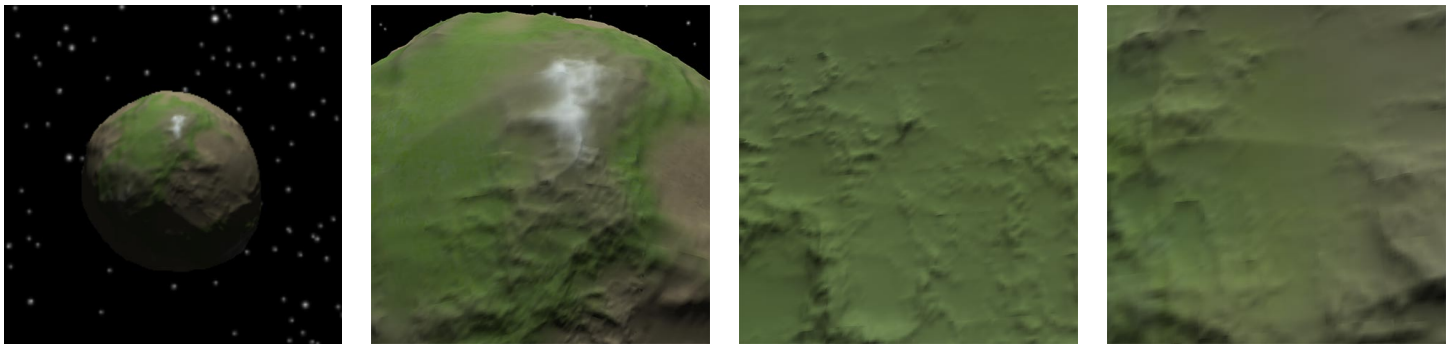


Wireframe LOD at different distances from the planet

Chunked LOD Properties

UNIVERSITÀ DEGLI STUDI FIRENZE
DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

- Top-Down generation: no need to know all the "height values" in advance.
- T-Junction can be avoided through "skirts".
- "Spherified cube" artefacts.
- Normal artefacts: Per face Normals + interpolated vertex normal must be calculated for each node separately, leading to shadows discontinuity at the node edges. These are not present with the "noise gradient" method.



Optimizing LOD

UNIVERSITÀ DEGLI STUDI FIRENZE
DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

- Compute the noise for all the sphere points is an expensive process, especially for Chunked LOD where is executed at runtime for each frame in which the camera position changes. In WebGL this issue is even more relevant due to the single thread nature of the javascript engine.
- Moving the noise calculation on the GPU vertex shader can bring sensitive changes in performances, but there are few downsides:
  - Nodes to be drawn must still be decided on CPU, so the nodes' centers must be computed on the CPU.
  - It's difficult to compute normals: the vertex shader cant access the position of neighboring points.
  - If the GPU is obsolete, performance can be equal if not worse the CPU solution.
- We implemented both approaches on the code, but obtained visually worse normals on the GPU one.

Conclusions and further developments

UNIVERSITÀ DEGLI STUDI FIRENZE
DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

- This project shows how to generate a planet procedurally, without the need of a preexisting map of the surface. Using WebGL allows the code to be ran on almost every browser, even on mobile, but it also limits the performance.
- Further improvements might be:
  - Improve the GPU implementation of the terrain noise with a more accurate normal calculation.
  - Implement Javascript **workers** for the quad-tree traversing.
  - Use a better measure for which nodes have to be drawn in the quad-tree (consider where the camera is pointing or how rough is the chunk).
  - Plenty more: atmosphere, vegetation, better controls and so on.