EDEN PROJECT

Créer un univers complet

L'article était originellement publié sur notre dev blog hébergé par Hits PlayTime

(Version en Français - Page 3)

Creating an entire universe

The article was originally published on our dev blog hosted by Hits PlayTime

(English version - Page 12)

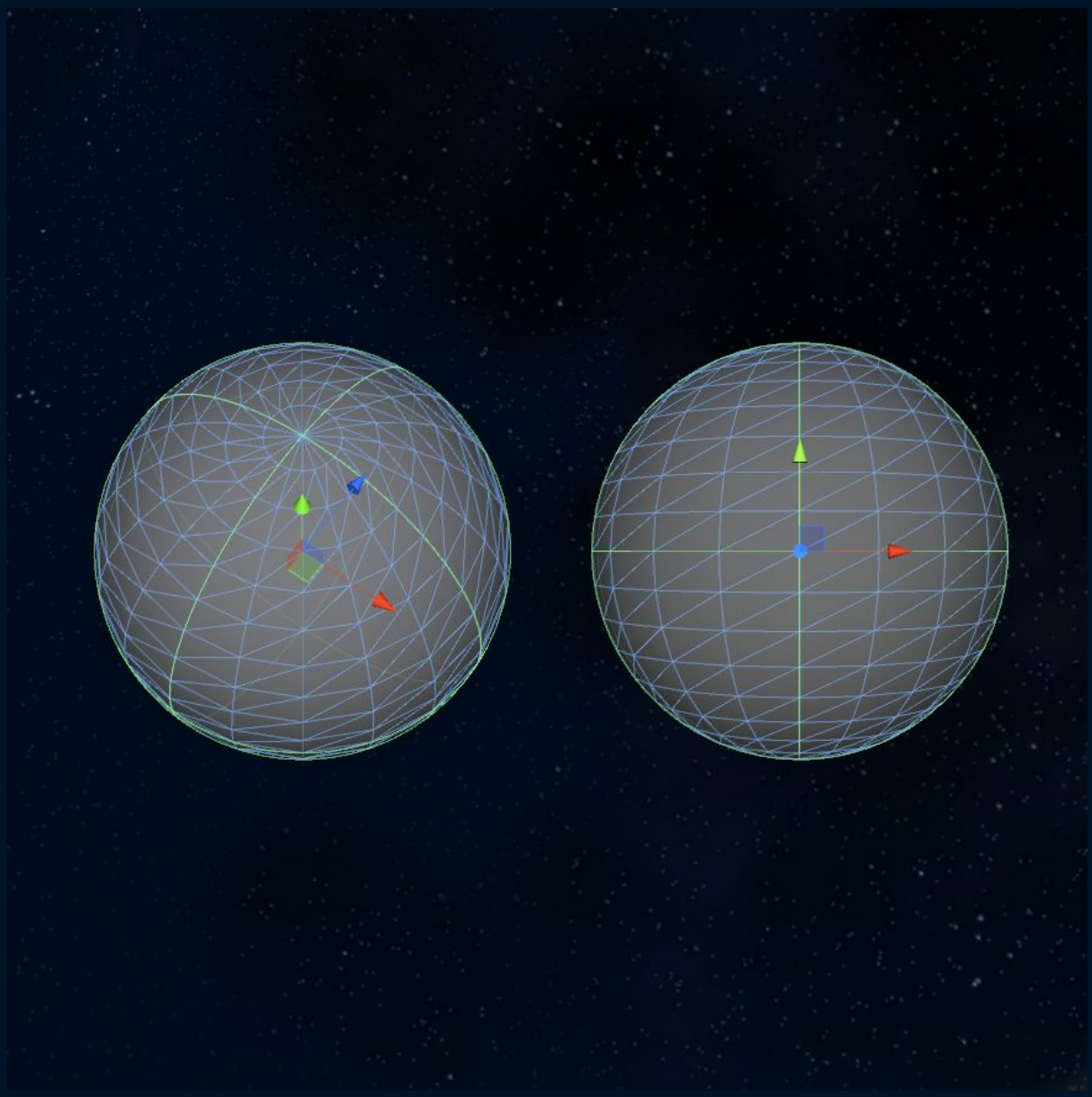
Créer un univers complet

Le projet étant un jeu en ligne à thème spatial, nous voulions dès le début créer un univers entier, ouvert, où chaque joueur aurait sa propre planète à gérer, où chaque monde serait différent des autres.
Nous nous sommes donc penché sur la génération procédurale pour créer les planètes de notre jeu.

Nous avons commencé par demander conseil à notre professeur Gil Damoiseau, qui a créé un générateur d'univers d'une toute autre envergure.
http://www.ignishot.com/grand-designer.html

Grâce à son aide et aux liens qu'il nous a fourni, nous avons pu comprendre dans les grandes lignes le principe de la génération procédurale.

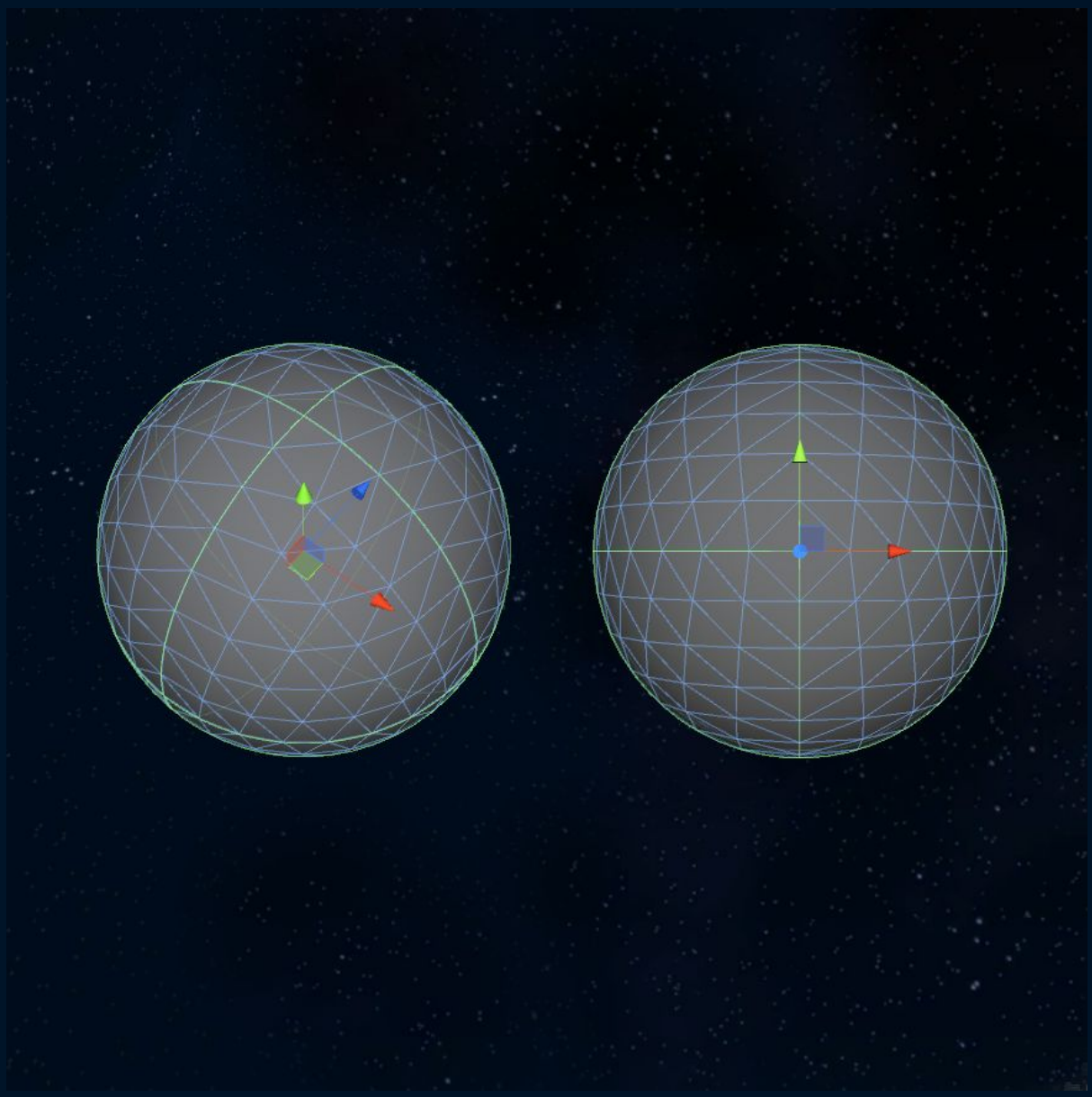
Il nous fallait tout d'abord un mesh de base, une sphère dans notre cas, pour créer le relief de la planète.
Le problème avec la sphère de base d'Unity (version 4) est que la répartition des vertex n'est pas homogène.



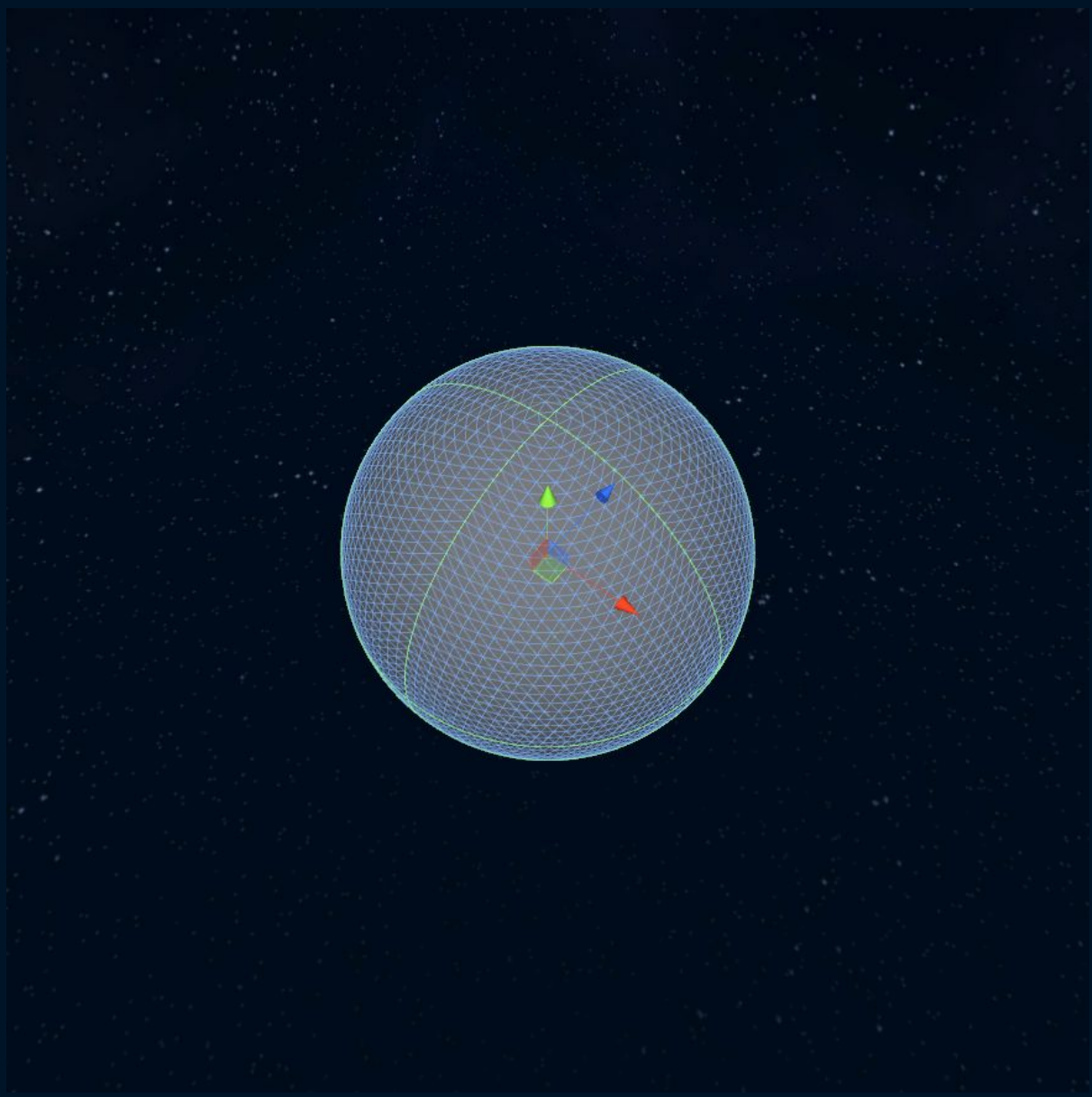
Sphère par défaut d'Unity 4, la concentration des vertex est plus fortes sur les pôles

Il y a une forte concentration des vertex aux pôles de la sphère, ce qui poserait problème par la suite car le relief aurait une résolution trop élevée aux pôles par rapport au reste de la sphère.

Nous avons donc dû commencer par générer notre propre sphère, avec une résolution des vertex homogène.
Nous nous sommes inspirés de ce tutoriel pour créer une sphère à base d'octaèdres.
http://www.binpress.com/tutorial/creating-an-octahedron-sphere/162



Sphère à base d'octaèdres, la concentration des vertex est beaucoup plus homogène



La même sphère avec une plus grande résolution

Notre sphère de base ayant maintenant un positionnement régulier des vertex, nous pouvons déformer ceux-ci afin de générer le relief de la planète.

Pour cela, nous avons utilisé la fonction Perlin Noise d'Unity, qui nous permet de récupérer une valeur pseudo-aléatoire en fonction d'une position, valeur qui reste cohérente d'une position à une autre.
http://docs.unity3d.com/ScriptReference/Mathf.PerlinNoise.html

Nous récupérons ainsi les positions X, Y et Z de chaque vertex de la sphère pour les passer en paramètre de notre fonction qui utilise le Perlin Noise.
Cette fonction nous renvoie une valeur entre 0 et 1, cohérente dans l'espace. Nous utilisons cette valeur comme base pour créer notre relief.

Nous utilisons le Perlin Noise sur chaque vertex pour obtenir une valeur pseudo-aléatoire

Chaque vertex est déplacé en utilisant la valeur pseudo-aléatoire obtenu.

En combinant plusieurs Perlin Noise, à des fréquences différentes, nous pouvons obtenir plusieurs environnements, des montagnes, des océans, des plaines, des volcans, des déserts, etc…

Création du relief final

Nous avons maintenant le relief de notre planète, reste à lui appliquer ses textures correctement.

Nous utilisons ici les Custom Shaders d'Unity.
http://docs.unity3d.com/Manual/SL-ShaderPrograms.html

Dans notre shader, nous testons la hauteur du relief, ainsi que son inclinaison.
Ainsi, quand le relief est plat, nous y appliquons la texture de plaine, sauf à partir d'une certaine hauteur, où ça sera de la neige.
Quand le relief est en pente, nous appliquons la texture de sable, sauf encore une fois à partir d'une certaine hauteur, où ça sera une texture de roches qui sera appliquée.

Nous utilisons ces tests pour appliquer les textures de chaque environnement.

Les textures sont appliquées grâce à un custom shader

On utilise le principe du Triplanar Mapping pour appliquer correctement les textures à la planète.
http://gamedevelopment.tutsplus.com/articles/use-tri-planar-texture-mapping-for-better-terrain--gamedev-13821

Cela nous permet d'éviter les problèmes d'étirement de textures et de dépliage des UVs.

Enfin nous calculons dans le shader, via la position du soleil et via les normales du relief, l'éclairage cartoon de la planète, le rendu final qu'elle aura à l'écran.

Nous rajoutons une sphère pour simuler l'eau des océans, et une dernière pour l'atmosphère.

Rendu final de la planète

Nous avons donc maintenant un système complet de génération de planète, de sa modélisation et son texturing jusqu'à son lighting.

Ainsi toutes les planètes seront différentes. Elles sont générées en temps réel, cela représente des millions de combinaisons possibles, des millions de mondes explorables.

Chaque joueur aura sa planète qui lui sera propre, il devra la coloniser, l'exploiter correctement, la protéger des autres joueurs pour enfin en coloniser de nouvelles et étendre son empire !
Chaque partie sera différente, rendant l'expérience de jeu unique !

4 types de planètes générées procéduralement

Nous vous parlerons très prochainement du système que nous avons utilisé pour créer les fonctionnalitées en ligne de notre jeu !

N'hésitez pas à nous suivre sur Facebook et Twitter.

L'équipe Eden Project.

Alexandre Meunier - alexandremeunier.com

Johan Lambot - johanlambot.com

Julien Hemmerlé - niouhop.com

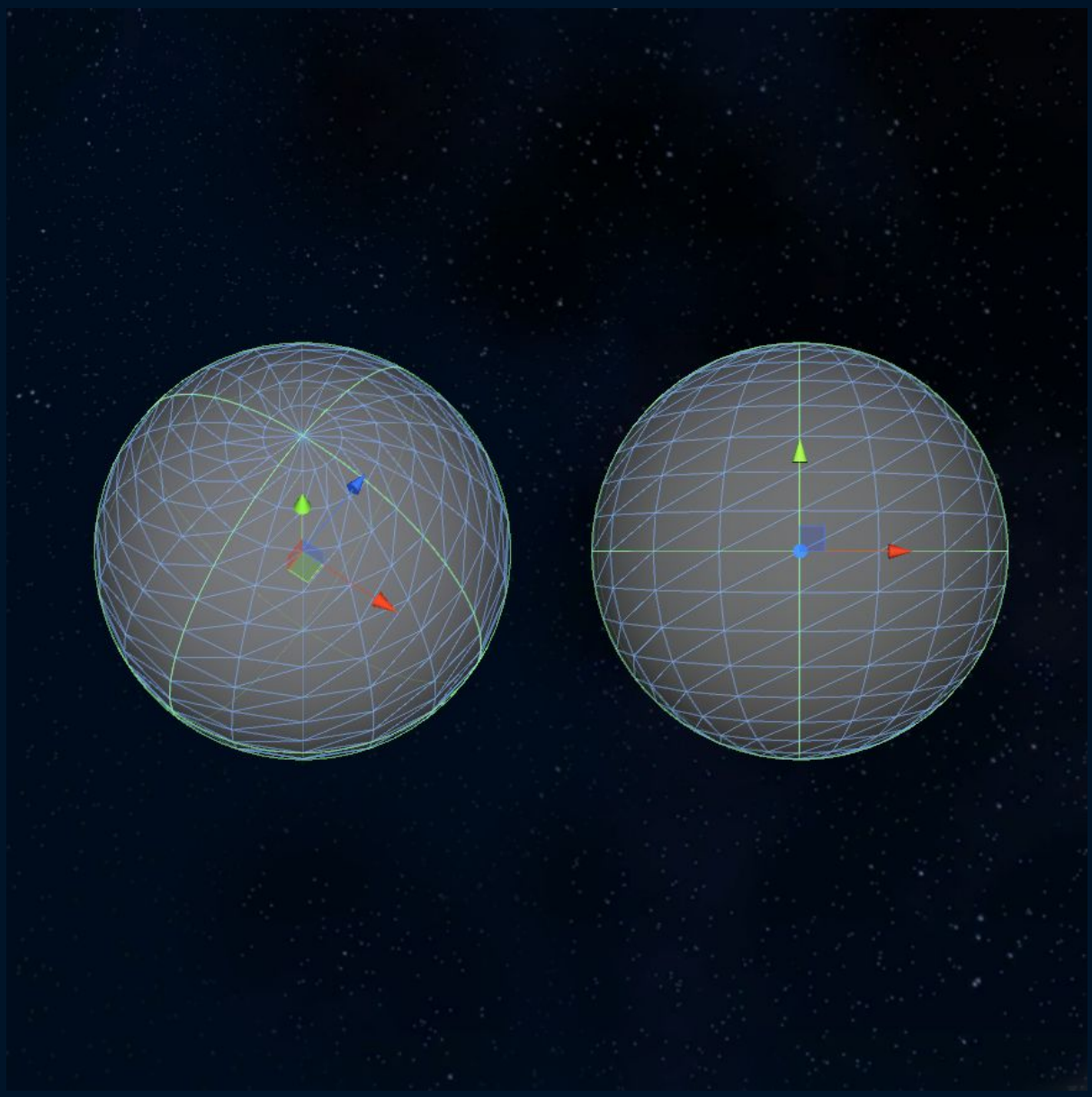
Creating an entire universe

Our project is an online space game. From the beginning, we wanted to create a entire universe in which every player has his own planet to manage and each world would be different from others.
So we thought that procedural generation is the best way to create all the planets in our game.

We started by asking for help to our professor Gil Damoiseau, who created a great procedural generator universe.
http://www.ignishot.com/grand-designer.html

With his help and the links he gave us, we were able to understand the main principles of procedural generation.

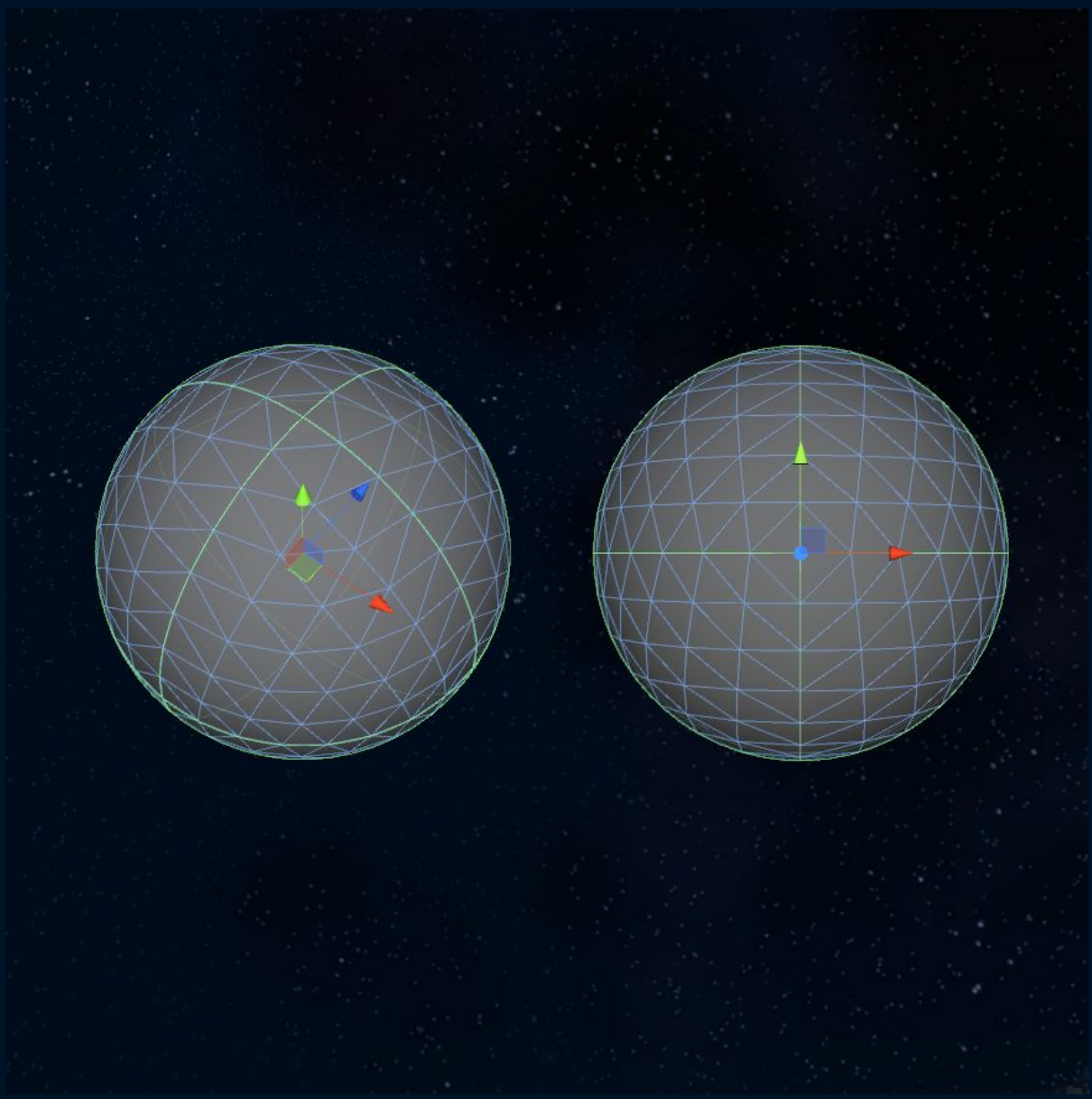
At first, we needed a sphere mesh to create the relief of the planet.
The problem with the basic sphere of Unity (version 4) is that the distribution of the vertex is not homogeneous.



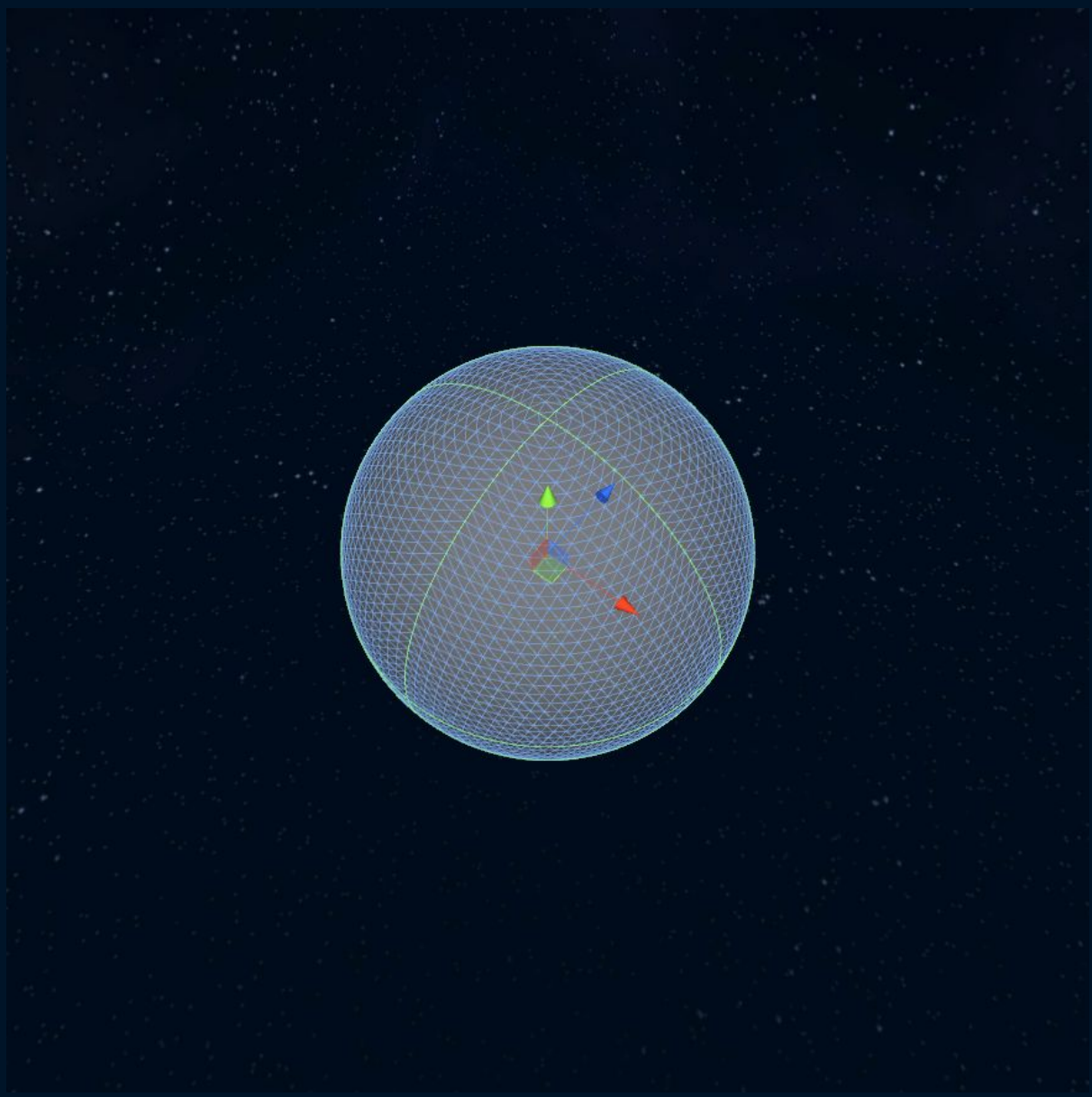
Unity 4 default Sphere, the concentration of the vertex is strongest on the poles

There is a high concentration of vertices at the poles of the sphere, which will be a problem for the next step because the relief will have a higher resolution at the poles than the rest of the sphere.

So we started to generate our own sphere with a homogeneous vertex resolution.
We were inspired by this tutorial to create a sphere based on a octahedron.
http://www.binpress.com/tutorial/creating-an-octahedron-sphere/162



Octahedron sphere, concentration vertex is much more homogeneous



The same sphere with a higher resolution.

Our basic sphere having regular positioning of vertex, we can now distort them to generate the relief of the planet.

For this, we used the Perlin Noise Unity feature, which allows us to retrieve a pseudo-random value based on a position value that remains coherent from one position to another.
http://docs.unity3d.com/ScriptReference/Mathf.PerlinNoise.html

We pass the positions X, Y and Z of each vertex of the sphere as a parameter of our function that uses the Perlin Noise.
This function gives us a value between 0 and 1, coherent in space. We use this value as the basis for creating our terrain.

We use the Perlin Noise on each vertex to get a pseudo-random value

Each vertex is moved using the pseudo-random value..

By combining several Perlin Noise at different frequencies, we can get multiple environments, mountains, oceans, plains, volcanoes, deserts, etc ...

Creating the final relief

Now, we have the relief of our planet, it remains to apply textures correctly.

We use the Unity custom shaders.
http://docs.unity3d.com/Manual/SL-ShaderPrograms.html

In our shader, we test the height of the relief, and its inclination.
Thus, when the terrain is flat, we apply the plain texture, except from a certain height, where it will snow texture.
When the terrain is inclined, we apply the texture of sand, except again from a certain height, where it will be a rock texture that will be applied.

We use these tests to apply the textures of each environments.

Textures are applied through a custom shader

The principle of triplanar mapping is used to correctly apply textures to the planet.
http://gamedevelopment.tutsplus.com/articles/use-tri-planar-texture-mapping-for-better-terrain--gamedev-13821

This allows us to avoid texture stretching problems and unfolding UVs..

Finally we calculate in the shader, using the position of the sun and normal terrain, the cartoon lighting of the planet.

We add a sphere to simulate the ocean water, and another for the atmosphere.

Final render of the planet

So now we have a full planet generation system, from its modeling and texturing to its lighting.

Thus all planets will be different. They are generated in real time, it represents millions of possible combinations, millions of explorable worlds.

Each player will have his own planet, he should colonize it, exploit it properly, protect it from the other players, to finally colonize other planets and expanding his empire!
Each game will be different, making an unique gaming experience!

4 types of planet generate procedurally

We will speak very soon of the system that we used to create the online functionalities of our game!

Feel free to follow us on Facebook and Twitter.

The Eden Project team.

Alexandre Meunier - alexandremeunier.com

Johan Lambot - johanlambot.com

Julien Hemmerlé - niouhop.com