Biome placement in a procedurally generated planet

David Miranda Pazos and Roberto Villarejo Fernández

1 Introduction

Procedural planet generation does not require any complex technique to create a minimally interesting planet. However, creating a unique and vivid planet, worthy of today's standards in games is hard and time-consuming, and usually involves human intervention (as in artist time of generating the planet manually). This is expensive, both in terms of time and money. This project aims to help in that problem.

In this paper, we suggest a biome placing system meant to be used offline for game developers to speed up the content generation. These biomes are highly customizable and based on any number of parameters, they can be easily adjusted, and changes can be seen almost instantly. The result of that offline process would be a file containing the final model as well as data about the biomes, which besides the model data only needs an additional unsigned short per triangle.

2 Layered data generation

The process of generating the planet is composed by a sequence of simple steps that belong to 2 main areas: geometry generation and property data computation. Data (such as the temperature of each region of the sphere) will be stored in multiple vectors in a per triangle or per vertex basis and can be later accessed for either biome placement or the generation of more complex layers of data.

2.1. Inflated cube

Representing data in 3D is usually a problematic topic especially when it comes to rounded objects as the sphere. There are multiple models resembling a sphere 2 of the main ones being spherical-coordinates-based sphere (Figure 1) or geodesic spheres (An icosahedron with further divisions in each triangle).

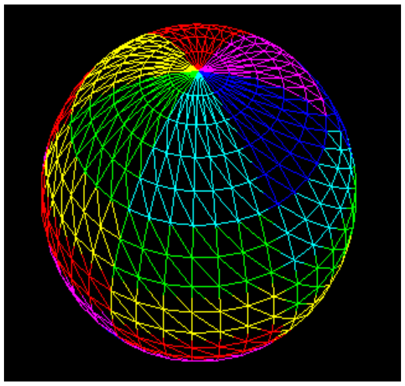


Figure 1 – Spherical coordinates

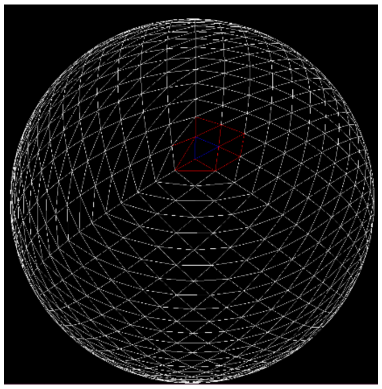


Figure 2 – Inflated Cube

However, those two options were not the best for the project: In the spherical-coordinates-based sphere geometry is not evenly distributed (being more noticeable in the poles) and for the geodesic sphere, accessing on it becomes tricky (especially when it comes to knowing the neighboring triangles to each other). Because of that, the geometry of the planet is represented as an inflated cube (Figure 2). This makes it much easier to distribute evenly the data and accessing it. Moreover, thanks to this distribution knowing the neighbors of each triangle becomes easier.

Nonetheless, getting those neighbors could be interpreted in more than one way, because of that the figure 3 below describes what a neighbor is in our logic. Even though getting them is trivial, since the data is represented as a cube map, refer to figure 4, the corners are especially tricky involving up to three different faces to be considered in which a triangle has neighbors.

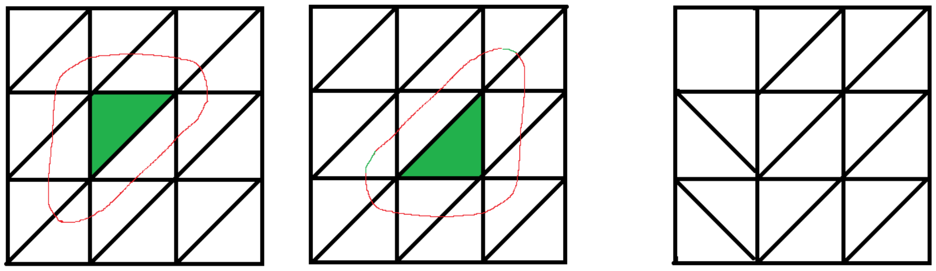
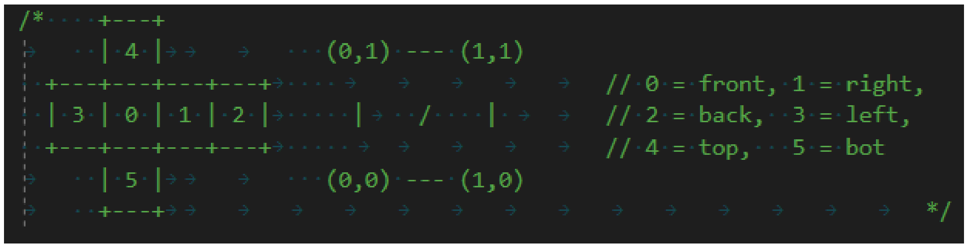


Figure 3 – Neighbors



```
/*    +---+
      | 4 |       (0,1) --- (1,1)
  +---+---+---+---+                   // 0 = front, 1 = right,
  | 3 | 0 | 1 | 2 |      |   /   |    // 2 = back,  3 = left,
  +---+---+---+---+                   // 4 = top,   5 = bot
      | 5 |       (0,0) --- (1,0)
      +---+                                                   */
```

Figure 4 – Cube faces breakdown

2.2. *Noise Map / Height Map*

To generate homogenous noise maps for later interpretation, a Perlin noise-based [Perlin 85] [Perlin 02] generator is used. The generator itself is based on multiple layers to generate more interesting and normalized maps. In few words, it uses a multisampling process based on some parameters that increment or decrement between each sampling and keep adding to the final value to later be normalized. The first layer using this method is the height map, which defines the height of each triangle or vertex in the model.

2.3. *Temperature Map*

There are two types of temperature defining the final temperature on a given point: Latitude temperature and internal temperature. Latitude temperature assumes that the maximum temperature will be in the equator and will decrease towards the poles. This decrease can be adjusted by an inverse cubic attenuation equation (1). The internal temperature in the planet is computed using the previously computed height map but inversing the value to favor low temperature in high points and high temperature in lower points. Which is more realistic. The final temperature is computed as a weighted sum of the above two values which gives more focus to the latitude temperature in the poles and more to the internal in the equator.

2.4. Coast Map

The geometry in the map is separated into water and land by a simple check of whether the height of the region is above or below the sea level (in this case, the sea level is a constant throughout the whole sphere, a floating-point value between 0 and 1).
The coastline is composed by all the triangles defined as “land” which are also surrounded by at least a triangle of "water”. This data is later used compute the minimum distance to the coastline from any point in the map (distance that will later be used to define some of the biomes such as the Deep Ocean which is a mass of water found far away from the coastline). Additionally, depending on the triangle being on water or land the distance is going to be stored in a positive or negative version.

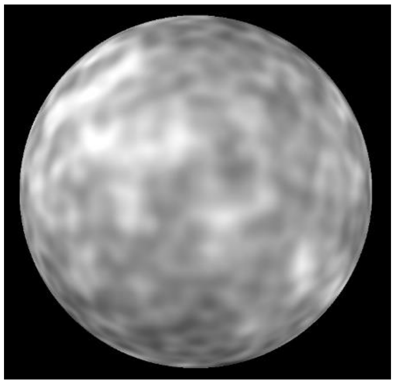


Figure 5 – Noise map

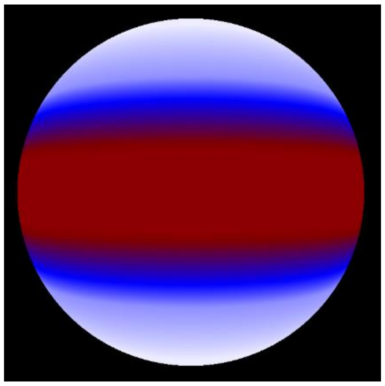


Figure 6 – Latitude temperature

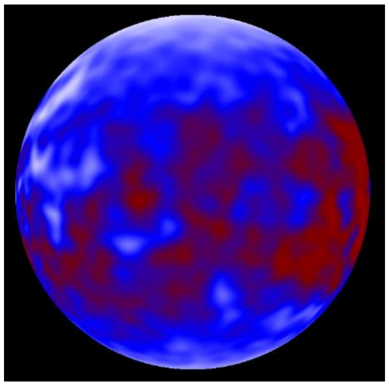


Figure 7 – Absolute temperature

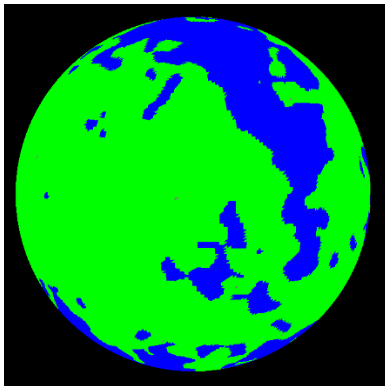


Figure 8 – Land/Water distinction

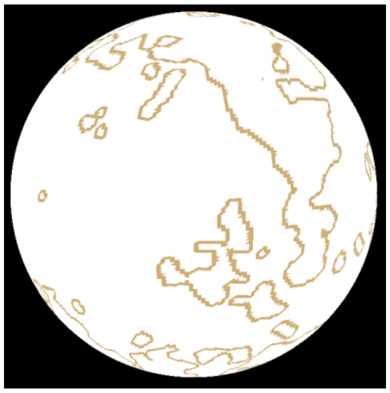


Figure 9 – Coast recognition

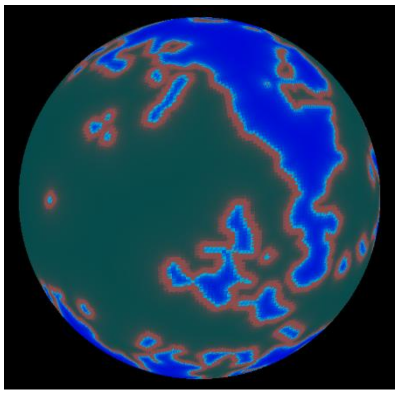


Figure 10 – Coastal distance

3 Biomes

Biomes are defined based on a few parameters (height, temperature, costal distance...) that may be extracted from the data layers and the geometry of the planet. An example of a biome defined in this fashion are the tropical beaches: They should be places on land, close to the water (0 min distance, small max distance), with a high temperature in the region and with not much height variation.

Once all biomes are defined for the planet we are working at, it is time to generate the final version. However, there might be collisions between 2 different biomes in the same area, for example, in the same triangles that a Tropical Beach can appear, a Tropical Forest may also appear (if defined as an area of high temperature and somewhat close to water). For this reason, collision resolution is needed.

3. 1. Exploring the neighbors

Whenever a collision occurs, there is a certainty that all related biomes can be placed in the triangle, but a decision needs to be made in a sensible way, not just deciding randomly. Since us humans prefer patters and smoothness, the approach we propose, is to decide based on the surrounding information.

First, useless information needs to be filtered out. In the context of the project that would be ignoring all the biomes that the triangle we are currently exploring cannot take, even if it appears in most of the surrounding cells. For example, a water tile in the coast of a tropical area that can take both the “coastal waters” or “warm ocean” biome. It might be surrounded by a huge area of tropical beach, but it does not make sense that the “tropical beach” biome is the final choice for a water tile.

Once the data is properly filtered what there is left is just keeping track of the number of occurrences each biome had and choosing the most frequent one. However, there is still a chance that after the whole process does not result in a conclusive answer. Even if it seems unlikely at first, this happens all the time.

The main trigger of this issue: A huge extension where 2 biomes that are not too specific in their properties overlap. Many triangles in the overlapping area would share both their possible biomes as well as having all their neighbors being of the same type, resulting in both choices having the same number of occurrences. To fix this, we rely on a process of multiple iteration.

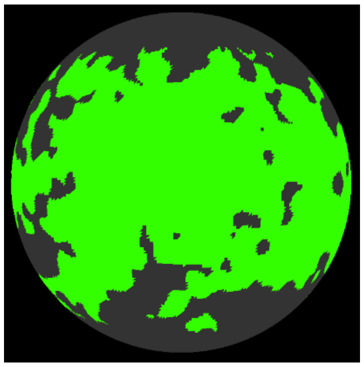


Figure 11 – Biome A

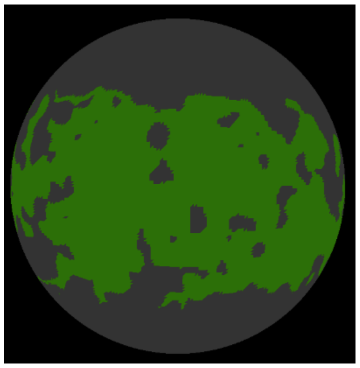


Figure 12 – Biome B

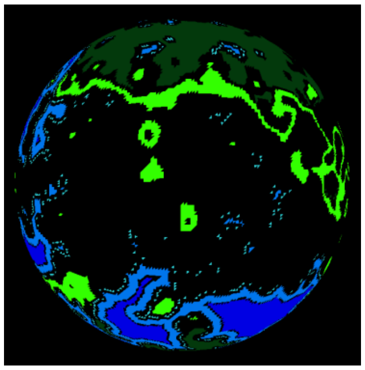


Figure 13 – In black, the collisions

3. 2. Multiple iterations

The aim of each iteration is simply to get a definitive answer for as many triangles as possible. Some of the triangles in big areas of uncertainty cannot be set from the beginning, however, after computing the ones that are, we can start filling the rest.
After some iterations most of the triangles will get a definitive answer. In case there are some that did not, we can arbitrarily choose one option. One possibility (we got nice looking results with it) is to just choose the one with the highest/lowest biome ID. This is done so that all triangles that are in the same situation (struggling to choose between the same 2-3 biomes), which are usually placed close-by in the planet, get a consistent result. If a random selection were applied instead the result would look weird at best if not totally wrong.

3. 3. Applying weights to each biome

Another option to avoid collisions is to apply a weight to each biome (thought to be used along with the other ideas). When exploring the neighbors and computing which is the most frequent biome among them, a value may be multiplied to number of appearances each biome had. This way we can ensure that when there is a rare biome (has lots of conditions / hard to happen at the same time / very restrictive conditions) it is more likely to be chosen over any other biome that may share some of the properties.
The value that is multiplied is usually a floating-point value, however, the value after the operation should be rounded to the closest integer value. This way definitive answers are only chosen once the margin is big enough to be worth considering and leaving further collision resolution to the ideas mentioned in the "Multiple iterations" section.

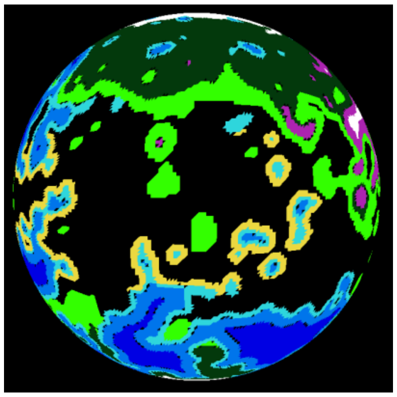


Figure 14 – Same weight, 1 iteration

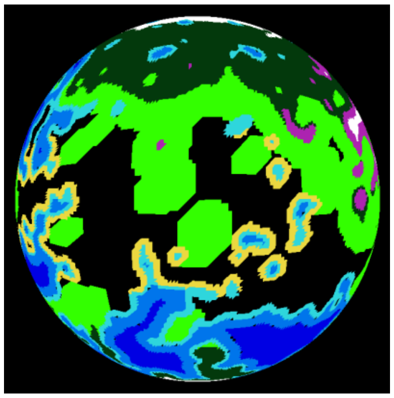


Figure 15 – Same weight, 5 iterations

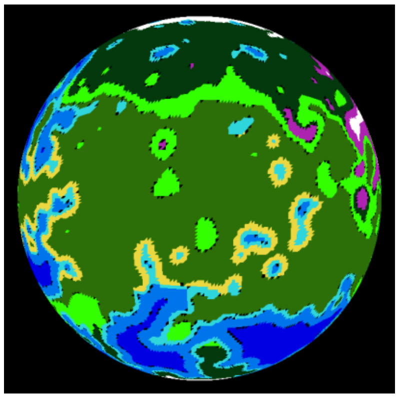


Figure 16 – Different weight, 1 iteration

3. 4. Implementation details

A single unsigned short (16 bits) may be used to represent the biome data for each single triangle. In this case it worked since the planet was expected to use less than 16 biomes otherwise it can be easily replaced with a 32-bit variable. Each bit position will represent one biome, therefore, when more than one biome meets the criteria for a triangle, as many bits as needed may be turned on.

When filtering out data for the neighbors (as mentioned in 3.1) it can be simply done using the & bitwise operation. The current triangles biome data (a set of 0s and 1s as explained above) can be used as a mask to extract the relevant biome data from the network and using the masked values to increase the counters for each biome.

A double buffer approach is recommended to avoid unwanted behaviors. When performing the iterations to solve collisions, if the data for each triangle is updated automatically and it can be used for the computations of the next triangle then the algorithm will highly depend on the order of the triangles (and traversal). Instead of that, a buffer can be used to write the data onto while we read from the original source of data, and later be overwritten once all triangles have been processed.

4 Final result

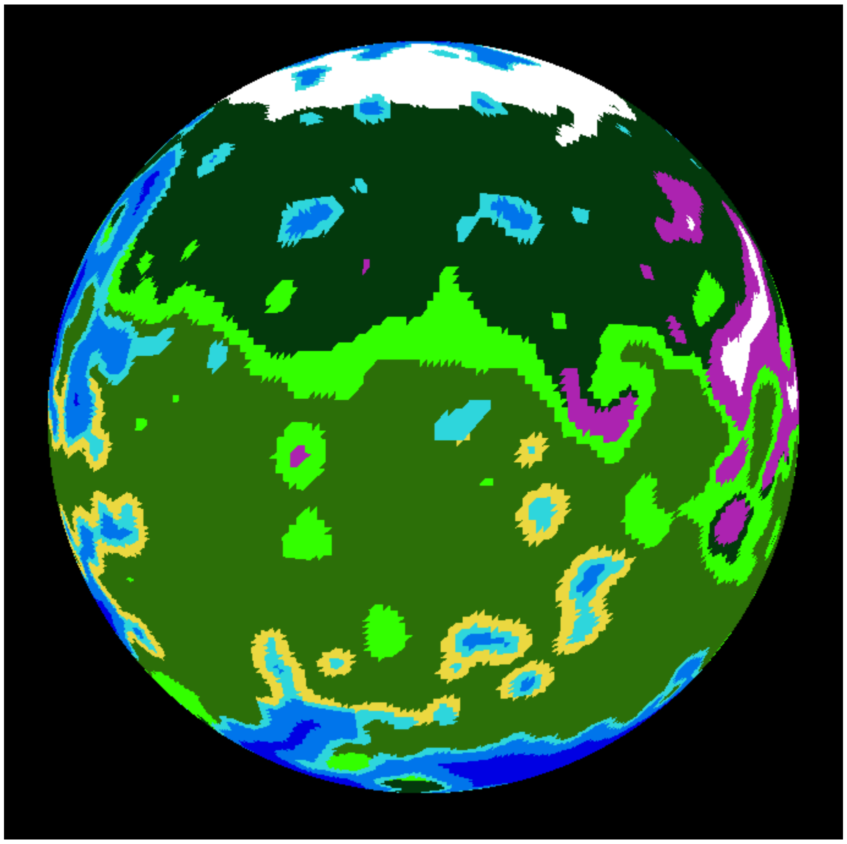


Figure 17 – Different weights, 5 iterations, arbitrarily decide at the end of last step (highest ID)

5 Selecting points of interest

All the layers of data generated for the planet do not only serve the purpose of defining the different biomes. Coupled with some new noise maps or other kinds of information they could also help at placing other interesting elements for the game such as cities, ruins, nests of animals...

In the demo attached to the project, cities are placed to illustrate this technique. To decide whether a city should be placed or not, distance from coast, distance from other cities and a new noise map is used. The first step is to know if that point meets the threshold for the distance from the coast and from any other city. Secondly, and most importantly, the new noise map is interpreted. Assuming the previous two conditions were met and if the hot spot of the noise map is on top of a biome which can handle a city it is placed.

6 Future work

At this moment, the project only handles one point of information per triangle. This means that the generated data could be limited in some cases in which subterranean or even atmosphere-related biomes are required. A fix for these issues would consist of generating more than one biome per triangle and classifying them in layers, such that the subterranean biome of a triangle would be taken from a set, the superficial biome from another one and so on.

7 Conclusion

Biome generation based on a few properties and with proper collision resolution allows for fast iteration times with high level of customization. Nevertheless, for this project to be usable in large scale games it should be coupled with proper manual editing to rule out any weird looking parts of the map. In the end, when creating content for a game quality will usually be preferred over quantity; for this reason, it is important to craft smart tools that allow faster iteration rather than fully automated ones that might not fit the game in the intended way.

8 References

[Perlin 85] Ken Perlin. An Image Synthesizer. In Computer Graphics (Proceedings of ACM SIGGRAPH 85), 24.3.
[Perlin 02] Ken Perlin. Improving Noise. ACM Transactions and Graphics (Proceedings of SIGGRAPH 2002) 21(3), pp. 681-682