A Multi-Layered Map Analysis Approach to Tree Species Distribution in Terrain Generation

Israel Nebot Dominguez

V1 (23/10/2018)

Abstract

Generation of realistic ecosystems should take into consideration the several parameters that belong to a certain species and determine where the tree distribution will take place in a natural way to increase the fidelity of the simulation.

This paper presents a method to simultaneously analyze the ecological factors of a certain zone given the maps that describe its humidity, temperature and height (although it could be expanded to use more variables to better tailor them to different species requirements or increase accuracy).
These values are compared to those of a certain species and the optimal growth zones for that species are sketched. Then, using a slightly modified Poisson Disc Distribution algorithm the trees are spread evenly throughout that zone allowing for a certain degree of overlapping.

Keywords: terrain, generation, visualization, ecosystem simulation.

1 Introduction

Thanks to the advancements in satellite imagery we are now able to obtain very detailed maps of the different ecological factors of a region that shed a light on how species distribution takes place.
Excluding random external factors that affect the placement and growth of these species, it can be seen that they mostly obey the constraints under which they grow, and these aspects can be observed and quantified.

Taking this into account the problem of life-like species distribution can be divided in two different entities that play a role in it:

1) A certain species of trees with certain ecological parameters only under which it will grow if the terrain suits its needs.
2) A map (or series of maps) that defines the variance of a given parameter throughout a terrain.

From a computer simulation standpoint the first entity of the pair follows an object-oriented programming schematic; creating a base object for a species and instantiating different species with specific parameters allows for a rich ecosystem.
The second entity will consist of as many maps as parameters the species (or the species instance with the most parameters in the case of custom parameters) has. This means that for a simple case of temperature, humidity and height the second entity would need three different maps that the program could read from.

2 Program overview

The proposed program takes a certain instance of a species into consideration with *n* parameters and analyzes the *n* layers to find a zone of overlapping known as the growth zone.

Species 1	
Temperature range	20 - 35 °C
Humidity range	20 - 50 %
Height range	200 - 500 m.

Fig. 1: Example of a fictional species instance that the program would analyze.

The first step is to supply the *n* maps (which for the sake of simplicity will all be power of two sized images of the same size) to the program and specify the different colors that appear while ranging them from, for example, more humid to less humid.
Thus a map legend is created: a dictionary data structure with a color key and a numerical range value. The environmental values for each color are stored in it.
Once the program can know the range in which every pixel of the map is the following algorithm is ran to obtain a list of possible coordinates (X,Y) where the trees will be placed:

```
For every position x in the image's rows:
    For every position y in the image's columns:
        If the pixel's value at (x,y) in the temperature,
        humidity and height maps is within the valid
        range of the species:
            Add (x,y) to a list of possible spots.
```

Fig. 2: Simplified algorithm that analyzes the different maps.

If all the data an image contains is needed this algorithm is necessarily ran in O(n^2) time, n being the width of any of the maps. A first approach would be to iterate the three images one after another, but knowing they all have the same size a cross analysis can be made to get the data from the same pixel at the same time.

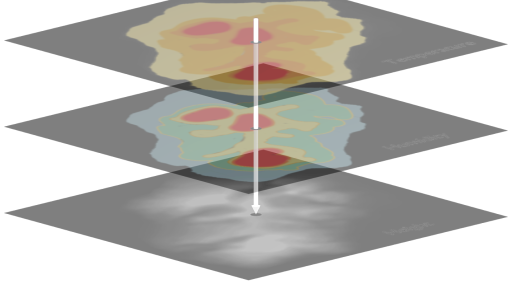

Fig. 3: Visual representation of the cross-map analysis.

A singly linked list data structure with a pointer to the last element can be used in order to provide O(1) insertion times and cut down complexity in this step.
The only extra computational cost the algorithm runs into is retrieving the range in which a given pixel is. This is done by searching in the aforementioned map legend dictionary.
This cost, however, is negligible due to the number of colors found in a map's legend being very small, hence the total cost is kept at O(n^2) time.

Once the list that represents the growth area has been obtained a slightly modified Poisson Disc Distribution (PDD in advance) algorithm is ran to obtain the final positions of the trees.

By creating a base tree class, generating instances of trees from a certain tree species and using it as a PDD sample every instance can hold its own radius as a variable that the algorithm will read from to finetune its behaviour.
The algorithm to distribute the trees throughout their growth area would be that proposed by [B07] with a few modifications:

- Use the radius of the crown of the tree instance as the radius around the sample to create the annulus inside which new samples will be looked for.
- To allow for some overlapping and a bit of clustering to achieve visually pleasing results, when comparing the distance between a new sample and the closest instead of using the radius to accept or discard it use the radius +- an overlapping percentage.

The base algorithm would run in O(n) time as explained in [B07].

Distribution of tree species will then take place by using the areas obtained in the first step as a parameter to accept or reject samples given by the PDD. Finally, a list holding the instantiated objects will be the only data structure kept in memory in order to represent the terrain.

3 Results

The aforementioned program has been implemented in the game engine Unity3D (2018.2.11f1).
Its development started with the implementation of the flexible PDD algorithm. A first approach generates a flat terrain and populates it with trees following what has been explained above:

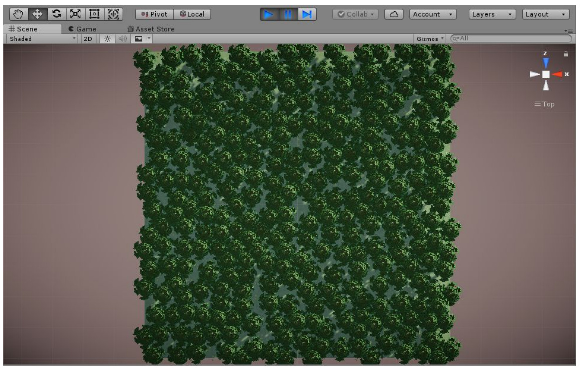

Fig. 4: Top down view of the terrain populated with trees.

By means of a first person character controller a more involved exploration of the environment can take place:

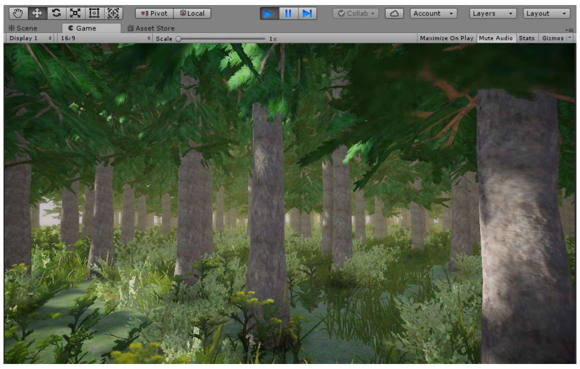

Fig. 5: First person view of the generated forest.

As it can already be seen, the implementation of this algorithm that allows for some flexibility yields realistic (or, at the very least, aesthetically pleasing) results.
The next step to be taken is only generating trees inside a given area or, conversely, deleting from memory and not rendering those outside it.

Thus a rejection sampling method is used: the total population of trees (a 100m x 100m area of trees) is used as the base distribution and a 3D mesh with a collider is used to detect samples inside its area; those outside are eliminated from the scene.
This rejection takes place in runtime. As soon as a tree is instantiated a script is ran right away to test its position by checking if it collides with the mesh that determines the area, and it is immediately deleted from the scene if this condition isn't met.

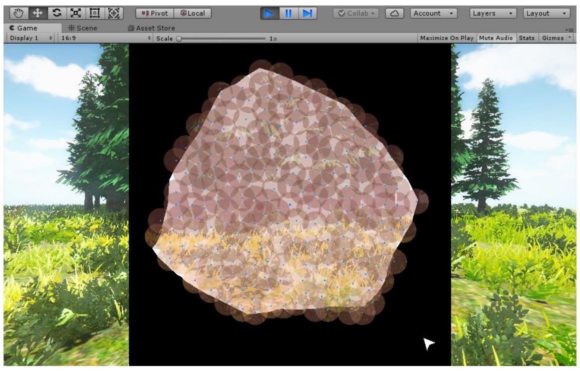

Fig. 6: Top-down view of the aforementioned area with the distribution of species belonging to it.

This algorithm is ran for every species of trees with their own areas and parameters in order to obtain the final terrain. As it can be seen in Fig. 6 the trees aren't strictly contained inside of the area sketched by the mesh and are also placed in its immediate surroundings. This will add to the effect of a slight gradient between species to increase visual fidelity.

For every species of trees ran the algorithm will check for the samples that have already been placed to avoid for overlapping and clustering of species.

At the end of the execution of this algorithm all previously used data structures are cleaned to preserve memory resources, since the final species distribution of tree models is stored in an array inside of the Unity Engine used for hierarchy construction and scene rendering.

Bibliography

Martin Weier, André Hinkenjann, Georg Demme, Philipp Slusallek, *Generating and Rendering Large Scale Tiled Plant Populations.* Journal of Virtual Reality and Broadcasting, Volume 10(2013), no. 1, ISSN 1860-2037

References

[B07] Robert Bridson, *Fast Poisson Disk Sampling in Arbitrary Dimensions*, Article from University of British Columbia