Procedural Terrain

Final project for class CS6491 Computer Graphics

Sebastian Weiss
November 25, 2015

***Abstract*—Abstract**

I. Objective

Terrain is a topic of endless discussion in computer graphics. It has to cover a large area of the world and at the same time it has to provide enough details for close-up shots. It is extremely time consuming to create a large-scale terrain that is also interesting when viewed closely.

In this project I present a framework for generating terrain, from a coarse grained height map to vegetation generation. The focus lies on combining existing techniques for performing the single tasks. The result should be an island because it provides a natural border of the world.

Applications of procedural terrain include computer games, movies and geographic visualization and simulation. The project was heavily inspired by a talk about "The Good Dinosaur" from Pixar Animation Studios.

II. Related Work

A lot of work has been done already in the field of generating height maps. There are in general three different approaches to this problem. Generating the height map completely from scratch using perlin noise, fractals, voronoi regions, software agents or genetic algorithms are described in [1], [2] and [3]. These models often lack realism because of the missing physically and geographic background. This is addressed in a second approach, hydraulic erosion models, as described in [4] and [5]. They start with existing height maps and increase the realism by simulating fluid to erode the terrain and forming rivers in the end. The other way is used in [6], here we start with a river network and build the terrain with respect to the river flow. The last approaches copes with the lack of control in the previous described methods. They define the terrain by control features provided by the user, see [7] and [8]. On a 2D-scope, [9] should be mentioned because it describes a complete new approach not using height map grids, but voronoi regions as base primitives.

After the terrain is created, it must be populated with vegetation. Rendering of grass is described in e.g. [10] and [11]. Trees must be generated first and for that, [12] and [13] should be mentioned.

III. Overview

The framework consists of the following steps that are executed one after another:

1) generating an initial map using voronoi regions, chapter IV
2) editing the terrain by user-defined features, chapter V
3) simulate water and erosion to increase the realism, chapter VI
4) define biomes and populate the scene with grass and trees, chapter VII

IV. Polygonal Map

As a first step, we have to come up with a good initial terrain to help the user with the terrain features. These techniques are taken from [9].

A. Voronoi Regions

We could start with a regular grid of squares or hexagons, but using a Voronoi diagram provides more randomness.

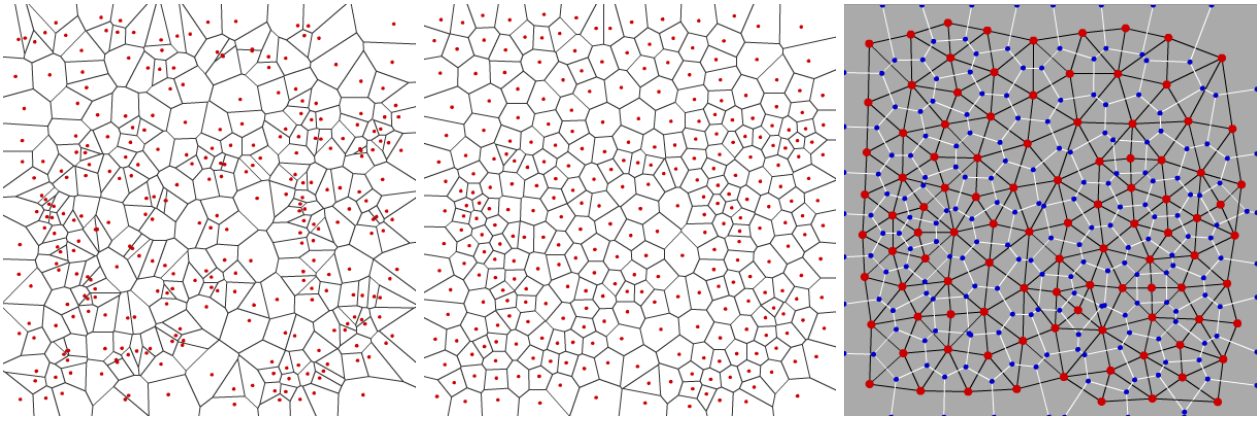
At the beginning, we generate random points, called center points, in the plane from -1 to 1 in x and y coordinate and compute the Voronoi diagram (Fig. 1a).

Because the shapes of the polygons are too irregular, we perform a relaxation step by replacing each center point by the average of the polygon corners (Fig. 1b). Applying this step several times results in more and more uniform polygons. In practice, two or three iterations provide good results.

B. Graph representation

For further processing of the polygons, we have to build a graph representation (Fig. 1c). The graph datastructure consists of the following three classes:

```
class Center {
    int index;
    Vector2f location;
    boolean water;
    boolean ocean;
    boolean border;
    Biome biome;
    float elevation;
    float moisture;
    float temperature;
    List<Center> neighbors;
    List<Edge> borders;
    List<Corner> corners;
}
```



(a) initial region (b) relaxation (c) graph representation

Figure 1: Initial polygon map

```
class Edge {
    int index;
    Center d0, d1; //Delaunay
    Corner v0, v1; //Voronoi
    Vector2f midpoint;
    int riverVolume;
}

class Corner {
    int index;
    Vector2f point;
    boolean ocean;
    boolean water;
    boolean coast;
    boolean border;
    float elevation;
    float moisture;
    float temperature;
    int river;
    List<Center> touches;
    List<Edge> incident;
    List<Corner> adjacent;
}
```

The class Center stores the center of a voronoi region, and the vertices of the voronoi polygons are represented by Corner. Because of the duality between the Voronoi Diagram and Delaunay triangulations, an edge connects both two corners (as an edge of a voronoi polygon) and two centers (as an edge in the delaunay triangulations). The other properties of the three classes are needed in the next step.

C. Island shape

As a next step, we have to define the general shape of the island. A simple way to get the desired shape is to combine perlin noise[1] with a distance from the center. Let (x, y) be a point and we want to know if it is in the water or not.

$$noise := \sum_{i=0}^{n} \text{noise}(x\ o_b\ o_s^i, y\ o_b\ o_s^i)\ a_b\ a_s^{-i} \qquad (1)$$

[1]http://mrl.nyu.edu/ perlin/doc/oscar.html

$$water(x, y) := noise < offset + scale\ (x^2, y^2) \qquad (2)$$

The function $water(x, y)$ returns true iff the point (x, y) is in the water. The function noise(x,y) returns the value of the perlin noise at the specified position.

There are many values to tweak: n defines the number of octaves that are summed together, o_b specifies the base octave, o_s the octave scale, a_b the base amplitude and a_s the amplitude scale. $scale$ defines the influence of the distance to the center of the map versus the perlin noise and $offset$ is the specifies a base island size. In our experiments, we use the following values: $n = 4, o_b = 6, o_s = 2, a_b = 0.5, a_s = 2.5, scale = 2.2, offset = -0.2$.

This function is evaluated for every corner (stored in the water-property). A center is labeled as water if at least 30% of the corners are water. The result can be seen in Fig. 2a

To distinguish between oceans and lakes, we use a flood fill from the border of the map and mark every water polygon on the way as 'ocean' until we reach the coast. Corners are marked as 'coast' if they touch centers that are ocean and land.

D. Elevation

After creating the initial island shape, we assign elevation values to every corner and center (Fig. 2b). For that we start from every coast corner and traverse the graph using breath-first along the corners. When traversing over land, we increase the elevation and over the sea we decrease the elevation. By that we obtain mountains and a continental shelf. Only when walking along or over lakes, we do not increase the elevation. This leads to flat areas that are later filled with water.

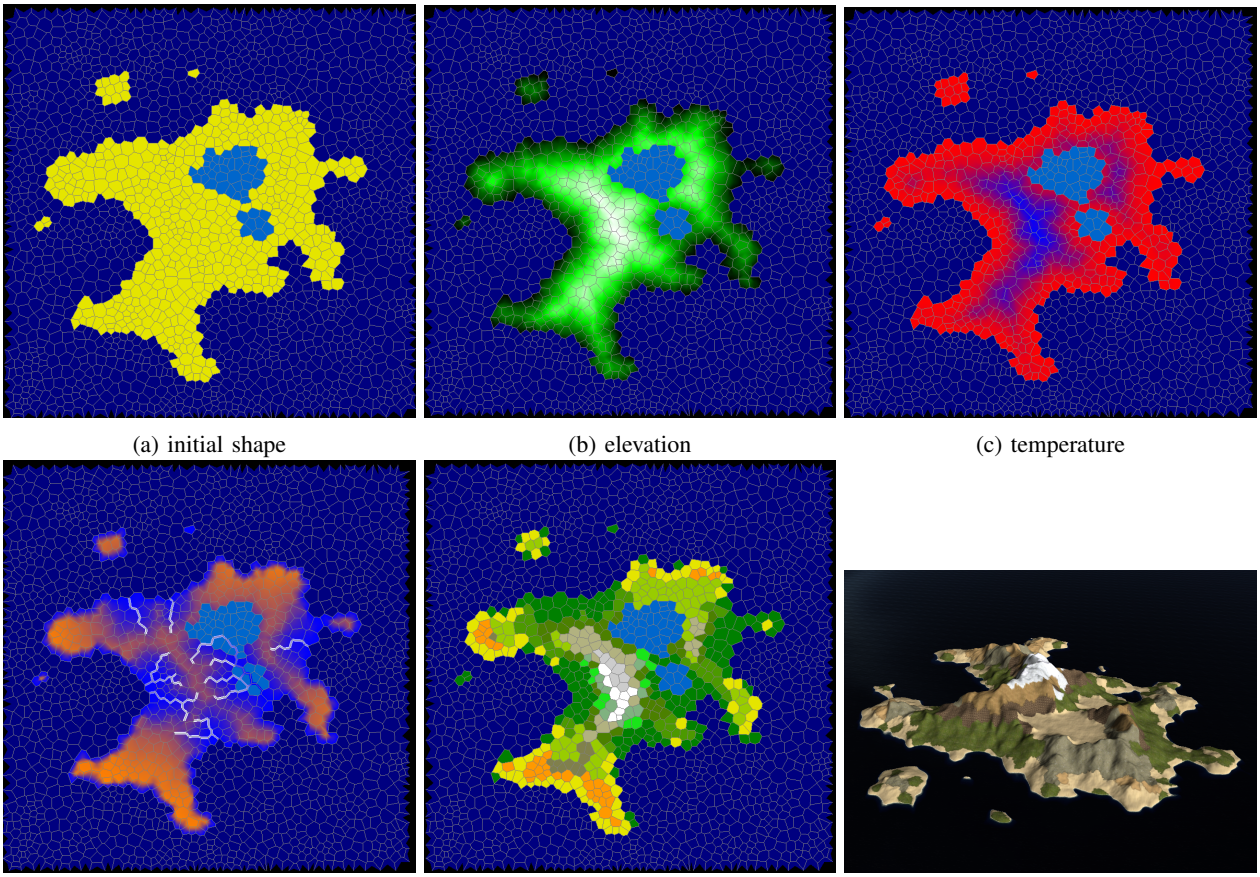
From the elevation, we directly derive the temperature: The higher it is, the colder it is. (Fig. 2c).

Until now, the elevation and temperature were defined for corners. To get these values for centers as well, we simply average them.

E. Moisture

To obtain moisture values, we start with creating rivers.

First we select random corners that are not coast or water and from there we follow the path of the steepest descent



(a) initial shape (b) elevation (c) temperature

(d) moisture (e) biomes (f) generated heightmap

Figure 2: Elevation and Moisture

until reaching the ocean. At each step, we increase the amount of water the river carries. We then initial the moisture value of the river corners with the water amount. To achieve more distributed rivers, you might start a new river only at corners that are at least one or two steps away from an existing river.

Second we perform a breath-first-search starting from river corners. At each step, we decrease the moisture by a multiplicative factor of e.g. 0.8 (works well in our experiments). By that, the rivers spread their moisture over the land depending on the amount of water they carry.

Third we average again the moisture of each corner to obtain the moisture per center. The result can be seen in 2d.

As a last step, we assign a biome to each center based on its temperature and moisture (Fig. 2e). This acts as a starting point for chapter VII. More details on the biomes are described in section VII-A.

F. Generating the height map

1) Heightmap: The next steps in the processing pipeline all require a heightmap. A height map is just a rectangular grid of values where each cell stores the height at this point. We also use the same datastructure to store moisture and temperature values.

Notation: Let h be the heightmap. Then we identified the cell at position i, j with either $h_{i,j}$ or $h[i, j]$, depending on the readability. When coordinates lie outside the map, we clamp them. When the coordinates are not integers, but lie between cells, we perform a bilinear interpolation between the neighboring cells.

2) Base elevation: Since we already display the polygon graph from the previous steps using triangle meshes, we can just render the elevation mesh from step IV-D to obtain a base elevation of the generated terrain.

However, the straight corners of the polygonal cells are still visible. Therefore we have to distort the height map by replacing each height with the height of the cell an offset away. The offset is defined by a perlin noise. For more details on this, see chapter "Combination and perturbation" in [2]. The increase the height difference between flatlands and mountains, we apply the following scaling to the height values:

$$h \leftarrow signum(h) \cdot |h|^{1.5} \tag{3}$$

3) Adding noise: The terrain still looks very boring, we need more noise to create hills and mountains. Therefore we add a multi fractal noise to the heightmap. The equation is taken from Chapter 4.3 of [8].

$$N(x, y) := A(x, y) \sum_{k=0}^{n} \frac{\text{noise}(r^k x, r^k y)}{r^{k(1-R(x,y)}} \tag{4}$$

noise is again the perlin noise function, r is the octave factor (here $r = 2$) and n is the number of octaves (we use $n = 5$). The scalarfield A defines the amplitude of the noise at the given terrain position and R the roughness (how the octaves are mixed together). In our experience we compute A and R based on the biomes using the values from table I. The noise

Biome	A	R
Snow	0.5	0.7
Tundra	0.5	0.5
Bare	0.4	0.4
Scorched	0.7	0.3
Taiga	0.4	0.3
Shrubland	0.5	0.2
Temperate desert	0.1	0.1
Temperate rain forest	0.3	0.2
Deciduous forest	0.3	0.4
Grassland	0.4	0.5
Tropical rain forest	0.3	0.2
Tropical seasonal forest	0.3	0.2
Subtropical desert	0.3	0.6
Beach	0.3	0.5
Lake	0.2	0.2
Ocean	0.2	0.1

Table I: Noise properties

value N is then simply added to the final height map.

The result can be seen in Fig. 2f. (The seed used is 658175619cff)

G. User interaction

In our implementation, the user can modify the elevation, temperature and moisture values starting from the presented initial values. By that, the user can create custom island shapes, raise mountains, build valleys and define the biomes by editing the temperature and moisture values.

With the terrain created in this step, we can proceed to the next step.

V. Terrain Features

Starting from the terrain from the previous chapter, the user has now the ability to fine-tune the terrain features.

VI. Hydraulic Erosion

TODO

VII. Vegetation

TODO

A. Biomes

VIII. Conclusion and future work

TODO

References

[1] J. Doran and I. Parberry, "Controlled procedural terrain generation using software agents," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 2, no. 2, pp. 111–119, 2010.

[2] Jacob Olsen, "Realtime procedural terrain generation," 2004.

[3] Teong Joo Ong, Ryan Saunders, John Keyser, John J. Leggett, "Terrain generation using genetic algorithms," 2005.

[4] Bedrich Beneš, "Real-time erosion using shallow water simulation," 2007.

[5] X. Mei, P. Decaudin, and B.-G. Hu, "Fast hydraulic erosion simulation and visualization on gpu," in *15th Pacific Conference on Computer Graphics and Applications (PG'07)*, pp. 47–56.

[6] Jean-David Genevaux, Eric Galin, Eric Guerin, Adrien Peytavie, Bedrich Benes, "Terrain generation using procedural models based on hydrology," 2013.

[7] Flora Ponjou Tasse, Arnaud Emilien, Marie-Paule Cani, Stefanie Hahmann, Adrien Bernhardt, "First person sketch-based terrain editing," 2014.

[8] H. Hnaidi, E. Guérin, S. Akkouche, A. Peytavie, and E. Galin, "Feature based terrain generation using diffusion equation," *Computer Graphics Forum*, vol. 29, no. 7, pp. 2179–2186, 2010.

[9] Amit Patel, "Polygonal map generation for games," 2010. [Online]. Available: http://www-cs-students.stanford.edu/~amitp/game-programming/polygon-map-generation/

[10] Kurt Pelzer, "Rendering countless blades of waving grass," in *GPU Gems*.

[11] K. Boulanger, "Real-time realistic rendering of nature scenes with dynamic lighting," 2005.

[12] Adam Runions, Brendan Lane, Przemyslaw Prusinkiewicz, "Modeling trees with a space colonization algorithm," 2007.

[13] J. Weber and J. Penn, "Creation and rendering of realistic trees," in *the 22nd annual conference*, S. G. Mair and R. Cook, Eds., pp. 119–128.