The Visual Computer manuscript No.
(will be inserted by the editor)

Procedural Approach to Volumetric Terrain Generation

Aitor Santamaría-Ibirika · Xabier Cantero · Mikel Salazar · Jaime Devesa · Igor Santos · Pablo G. Bringas

Received: date / Accepted: date

Abstract The recent outbreak of indie games has popularized volumetric terrains to a new level, although video games have used them since decades. These terrains contain geological data, such as materials or cave systems. To improve the exploration experience and due to the large amount of data needed to construct volumetric terrains, industry uses procedural methods to generate them. However, they use their own methods, which are focused on their specific problem domains, lacking of customization features. Besides, the evaluation of the procedural terrain generators remains an open issue in this field since no standard metrics have been established yet.

In this paper we propose a new approach to procedural volumetric terrains. It generates completely customizable volumetric terrains with layered materials and other features (e.g., mineral veins, underground caves, material mixtures and underground material flow). The method allows the designer to specify the characteristics of the terrain using intuitive parameters. The method uses a specific representation for the terrain based on stacked material structures, reducing memory requirements. To overcome the problem in the evaluation of the generators, we propose a new set of metrics for the generated content.

Aitor Santamaría-Ibirika
Xabier Cantero
Mikel Salazar
Jaime Devesa
Igor Santos
Pablo G. Bringas
E-mail: {a.santamaria, xabier.cantero, mikel.salazar, jaime.devesa, isantos, pablo.garcia.bringas}@deusto.es

University of Deusto
Avda. de las Universidades, 24
48014 Bilbao, Bizkaia, Spain
Tel.: +34 944139064
Fax: +34 944139101

Keywords procedural generation · virtual worlds · volumetric terrain · terrain modelling

1 Introduction

Interactive graphic software has evolved in the last two decades due to several reasons, including the rapid hardware evolution (arguably the most important factor). These advances have expanded the possibilities of virtual world simulations. For instance, the level of detail of virtual environments has grown to almost-realistic levels. Therefore, a significant number of real time graphic applications make use of highly detailed virtual worlds, even with volumetric terrains.

Volumetric terrains are usually procedurally generated without human interaction. This process has two main advantages against the traditional design. First, it avoids terrain modeling steps, saving time and money. Second, it is possible to create interesting and unpredictable terrains to be explored (in some cases, even theoretically infinite).

Traditionally, terrains have been generated using a height matrix called heightmap. These techniques define the surface data, but they present limitations on the complexity of some structures (such as bridges or caves).

First approaches to procedural terrains were based on fractals [9, 4, 20]. These methods generate realistic terrains, but have low interaction possibilities. Miller [9] used a grid subdivision technique to generate them, subdividing a grid with the terrain heights recursively. Fournier et al. [4] and Voss [20] proposed different methods based on Perlin noise [15].

Current research in fractal terrain generation focuses on two main aspects: erosion simulation and higher design control.

Erosion simulation improves the realism of the results. Musgrave et al. [10, 11] simulated fluvial and gravitational

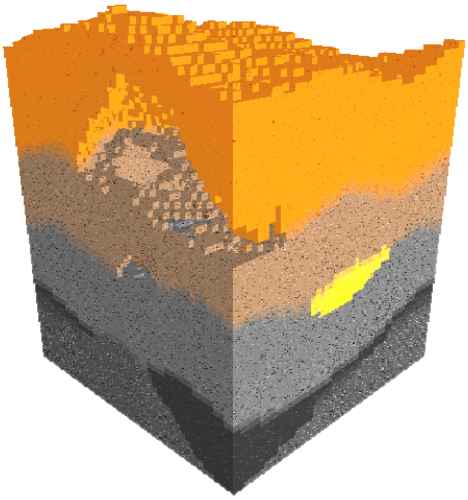


Fig. 1 A terrain generated with our method.

erosion and Olsen [13] analyzed some speed optimizations. Meanwhile, Benes and Frosbach [1] proposed a method to erode volumetric terrains.

However, the lack of control over generated results is an endemic problem in fractal generation. Kamal et al. [8] proposed a method to enhance the control of the designer, but it requires technical parameters. Doran et al. [3] used an agent-based approach to customize the terrain with additional parameters. On the other hand, Gain et al. [6], Zou et al. [21], Hnaidi et al. [7] and Carpentier and Bidarra [2] suggested innovative approaches based on direct painting on the terrain surface.

Albeit fractal methods have been the main trend in terrain generation, evolutionary algorithms can also be used for this objective. They provide designers with higher control over the process, adapting the results to different objectives [18, 17]. There are several evolutionary approaches, each one focused on specific goals such as playability or realistic appearance [14, 19, 5].

Some researches focused on the validation of those type of generators. Ong et al. [14] proposed four aspects that generators should meet (adaptive, innovative, scalable and intuitive). Saunder [16] defined the ideal generator with a set of desirable features (require low degree of human input, require high degree of human control, be intuitive, be able to generate realistic and recognizable terrains, produce terrains at arbitrary scale, be able to run in real time and be extensible to support new types of terrains). Doran and Parberry [3] proposed another set of properties for an ideal generator (novelty, structure, interest, speed and controllability).

However, to the best of our knowledge, there is no procedural approach in the literature to generate volumetric terrains. Besides, current industrial approaches are completely focused on their own problem domain and do not allow the customization of the results. Moreover, there is not a standard method to validate procedural terrain generators, existing a lack of metrics to evaluate this type of generators and their results.

Against this background, we propose a real time method to procedurally generate volumetric terrains. Our approach is highly customizable, enabling the designer to specify terrain materials, including their characteristics and their interrelations. In addition, we propose a new approach to represent the volumetric terrain data, based on stacked material columns, allowing us to greatly reduce the memory size required. Besides, there is a classic problem evaluating the quality of generators. The lack of metrics and procedures to measure highly important aspects (like the interest of the results or the obtained realism) disables the evaluation with no subjectivity factors. To avoid this problem in the validation of our method we propose a new set of metrics.

Summarizing, the main contributions of the paper are:

– A new customizable online method to generate procedural volumetric terrains.
– A new system to represent volumetric terrains.
– A new set of metrics and requirements to validate procedural generators.

The remainder of this paper is organized as follows. First, section 2 presents an overview of the contributions of this article. Next, section 3 explains the volume representation system and section 4 outlines the generation process. We present the results in section 5. Finally, section 6 presents the conclusion of our research, with an strengths and weaknesses analysis and recommendations for future work.

2 General overview

This section provides a general overview of the contributions of this paper.

2.1 Evaluation Metrics

Some authors have established a set of prerequisites a terrain generator must satisfy [3, 16, 14]. We synthesize them in a set of aspects which the procedural generators must meet. Additionally, we assign a metric to each aspect to allow the measurement of the requisite.

Innovation Ability of the procedural generator to obtain unexpected elements.

0. No innovation. The results are completely predictable.
1. The method generates the same elements in all the results, but with different properties.
2. The results have unexpected elements.

Structure Ability of the procedural generator to obtain recognizable coupled elements.

0. Random. The results are similar to the random noise.
1. The results have independent recognizable elements.
2. The results have recognizable coupled elements.

Interest Ability of the procedural generator to obtain interesting results to interact with.

0. The terrains are no explorable.
1. The terrains are explorable.
2. The terrains motivate the exploration.

Speed Speed of the generator obtaining the result.

0. Offline.
1. Hybrid approach. It is fast enough to be executed in the previous step to the interaction (such as a loading screen), but it can not be executed online.
2. Online.

Usability Ability of the procedural generator to hide technical aspects to the designer.

0. N/A. The method has not input parameters.
1. The parameters are related to the technique.
2. The parameters are related to the result.

Control Ability of the procedural generator to allow the designer to specify the generated contents features.

0. The designer can specify no aspect of the result.
1. The designer can specify some aspects of the result.
2. The designer can specify all the aspects of the result.
3. The designer can specify all the aspects of the result, correcting them iteratively.

Ampliability Ability of the procedural generator to obtain different types of results.

0. None. It is impossible to obtain other type of results.
1. The procedural generator needs internal changes to obtain other type of results.
2. It obtains other type of results by modifying the input.

Scalability Ability of the procedural generator to obtain results at different scales and level of detail.

0. The results are always at the same scale.
1. Scalable. The procedural generator can obtain results at different scales and level of detail.

Realism Ability to obtain plausible results.

0. Impossible in any reality. For example, random noise.
1. Plausible in other realities. For example, flying isles.
2. Based on our reality, but not a simulation (called *data free* approach by Natalie et al. [12].)
3. Simulation of our reality. (called *sparse-data* and dense-data approach by Natalie et al. [12].)

2.2 Our Method

The main objective of this paper is to obtain a valid generalist method to get volumetric terrains. To this end, based on the metrics established before, our method has to meet the requirements shown in Table 1. We choose the maximum value of each metric excluding control and realism. We choose a level 2 of control because a level 3 require an interactive process, making impossible to execute it online. Aditionally, we establish the required level of realism at 2 because this is a constant in the video game industry, due to the high requisites in computation for a realistic simulation.

To satisfy these requisites we propose a method with the following features:

- **Probabilistic generation:** Our approach is based on probabilities, making impossible the prediction of what element is going to be constructed next.
- **Underground layers, veins and caves:** Generated terrains are structured in underground layers formed with material mixtures. This feature allows the terrain to avoid sharp limits between materials, connecting them with realistic progressive changes. Additionally, our method generates caves and mineral veins among layers, motivating the exploration.
- **Real time generation:** Our approach is fast enough to generate the terrain online.
- **Designer control:** Previous characteristics are controllable by the designer. Input parameters allow the generation of multiple terrain types, such as realistic or fantastic terrains, at different scales. Additionally, these parameters are directly related to the terrain, avoiding technical concepts and being friendly for non-expert users.
- **Reduced memory size:** We use an optimized representation for the terrain, designed to accelerate the generation process and to reduce its size in memory.

3 Volume representation

Traditionally, volumetric terrains are represented by voxels. A voxel is the minimal unit of volume, organized in a three-dimensional regular grid, storing different data like color, hardness or density.

The main problem of this representation is the memory usage, because a $N_t \times M_t \times L_t$ terrain stores the values for $N_t \times M_t \times L_t$ voxels. This is not optimal for interactive environments, where the speed is a critical aspect.

To avoid this issue, our technique is based in Benes and Frosbach representation [1], dividing the volumetric data in three levels: chunks, columns and limits.

In our approach, the chunk is the highest unit of volume. We define it as a three-dimensional portion of terrain with a size of $N \times M \times L$ voxels. This size is the same for all chunks

Table 1 Requirements for a procedural generator.

Metric	Ideal value	Our requirement	Notes
Innovation	1, 2	2	The ideal value depends on the scope and the objective of the problem, and the desired level of heterogeneity between the results.
Structure	2	2	The results must always present recognizable and structured elements.
Interest	0, 1, 2	2	The ideal value depends on the scope and the objective of the problem and the area in which the results are going to be used.
Speed	0, 1, 2	2	The ideal value depends on the purpose of the problem. The results must be ready when they are needed.
Usability	2	2	The generator must always hide the technical aspects.
Control	2, 3	2	The ideal value should be 3 only if the method is offline.
Ampliability	2	2	The generator must be easily adapted to obtain different types of results.
Scalability	1	1	The ideal generator must be able to generate content at different scales.
Realism	$\geq$1	2	The ideal value depends on the objective of the problem and the area in which the results are going to be used.

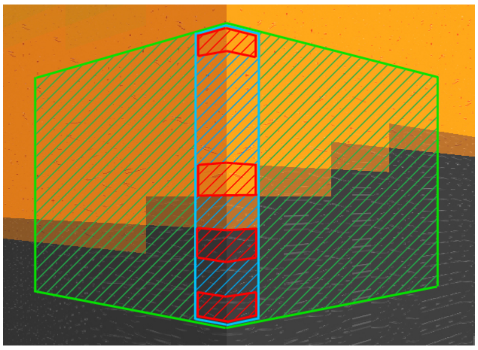


Fig. 2 Volume management example. This column has three limits: at the top, at the bottom and at the frontier between two different layers.

of the terrain, but it is variable and can be adapted for different needs. Figure 2 showcases a chunk in green.

Next, we subdivide the chunks into $N \times L$ columns in a regular grid. Each of these columns represents the data of M voxels. Figure 2 shows a column in blue.

Finally, the columns are formed by stacked material limits. These limits determine the properties of the volume at a specific height. Each column has at least 2 limits (the column top and bottom) and a maximum of M limits. Figure 2 shows an example of a column and its limits in red.

Using this representation of the volumetric data we can obtain any voxel dynamically. If the voxel is at the same chunk, column and height of one of the limits, the procedure generates it with the data stored in this limit. Otherwise, the method generates it interpolating between the limits surrounding the voxel. For example, if we want a voxel at height 2 from the blue column in figure 2, the procedure interpolates it between the two limits at the bottom.

This representation can store different types of data. Our terrains data is formed by the material composition (a list of materials with an associated concentration which determines the color and texture of the volume) and the material flow (internal material flow of the terrain represented as a three-dimensional vector).

4 Terrain generation

The terrain generation process can be divided into three main steps. First, the algorithm reads the input data and indexes the materials by their characteristics. Next, layers and veins are created. Layers can be indefinitely extended to generate the terrain in real time. Finally, the procedure converts the generated terrain into our volume representation method.

4.1 Input data

The designer must select the terrain characteristics, which can be done at design time in order to allow the generation process to be in real time. This data is divided in two main sets of parameters: material data and general parameters.

4.1.1 Material data

It describes terrain materials. This data is supplied in a configuration file with three sections.

First, basic material data is provided, which contains several data-fields for each material:

- **Name:** Material name.
- **Id:** Numerical unique identifier.
- **Color:** RGB color value.
- **Special parameters:** Parameters which establish some special characteristics of the material:
  - *Void*: Void materials determine empty voxels in the terrain.
  - *Base*: Base materials for the terrain, only located at the bottom layer.
  - *Alone*: It identifies non-mixing materials.
  - *Vein*: It identifies a material as a resource, structuring it into veins instead of layers.
  - *Cave*: This parameter identifies the materials to construct the caves. Different cave materials define different type of caves.

– **Textures:** Path to the material texture file.

Second, material data provides layer related characteristics, containing several data-fields for each material. All values are expressed in a 0-1 scale:

– **Area:** Only used with vein or cave materials. It specifies the vein or the cave area.
– **Depth:** The minimum, maximum and optimum depth of the elements formed with this material.
– **Thickness:** The minimum, maximum and optimum thickness of the layers, veins or caves formed with this material.
– **Roughness:** The minimum, maximum and optimum roughness of the elements formed with this material. For cave or vein materials it represents the constants for the fractal algorithm explained in section 4.2.2, affecting the shape roughness. Otherwise, it represents the superficial roughness.
– **Similarity:** It specifies the similarity of the shape between two consecutive layers.
– **Blend value:** It specifies the suitability of the material to be mixed with others.

Third, this file contains the relationships between materials. These data is provided as two data-fields per pair of materials. The values are expressed in a 0-1 scale if the two materials are compatible, and with a -1 value if they are not. So, if one of these values is 0.0, it means that those materials are not prone to be mixed, but they still can be blended by other reasons. These data-fields represent the affinity in the same layer (the affinity rate of the two materials in the same layer) and the affinity in the next layer (the affinity rate of the two materials in consecutive layers).

4.1.2 General Parameters

These global parameters, in some cases, transform relative values of the materials into absolute.

– **Terrain depth:** The distance between the surface and the bottom layer.
– **Layer thickness:** It converts thickness of each layer into an absolute value.
– **Cave and vein area:** It affects the size of the area of each vein and cave. It establishes the maximum length of the side of the square which bounds to the vein or cave.
– **Cave and vein density:** Densities of veins and caves in the terrain.
– **Roughness:** It converts the roughness parameter of each material into an absolute value.
– **Layer similarity:** It affects to the similarity parameter of each material.
– **Scale:** Scale of the terrain.
– **Cave and vein thickness:** It converts the thickness of each cave and vein into an absolute value.

4.2 Terrain generation

The first step of the generation procedure is the creation of layers, veins and caves. This step starts with the construction of the bottom layer, formed by materials with the base parameter. Next, it continues stacking more layers, veins and caves. The chance of generating a vein, a layer or a cave is determined by the vein density and the cave density parameters in the input data.

This process ends when the terrain depth reaches the value established in the input data.

4.2.1 Layer generation

First, the algorithm chooses the materials for the new layer. This process is performed by accessing the materials by their special parameters.

If this is the bottom layer, it will be composed of a mix of materials with the base parameter. Otherwise, the materials are chosen from a list based on the affinity of the materials of the last layer. To this extent, we get all possible materials, removing from the list any incompatibilities with the composition of the last layer. Then, we get the global affinity of each possible material:

$$A_i = \sum_{j=1}^{N} (a_{ij} \cdot c_j) \quad (1)$$

where A_i is the global affinity of i possible material, N is the number of materials in the last generated layer, a_{ij} is the affinity between the i possible material and the j material of the last layer, and c_j is the concentration of j material in the last layer.

Next, we get the probability of each material to be in the new layer:

$$P_i = A_i / (\sum_{j=1}^{N} A_j) \quad (2)$$

where P_i is the probability of the i possible material to appear, A_i is the global affinity of i possible material calculated in the last equation, N is the number of possible materials, and A_j is the global affinity of j possible material.

Then, the process obtains the main material of the new layer using a random value and the previously calculated probabilities. At this point, the layer has a probability of containing additional materials (being 0.5 for the second material, halving it consecutively for next materials). The algorithm includes the already selected materials to the materials of the last layer to calculate the new affinities.

Afterwards, we calculate the concentration of each material, which is the relative quantity of material located in the layer. The addition of all concentrations in each layer must be 1. If the layer is composed by two or more materials, the main material has a concentration randomly chosen within a 0.5 to 0.9 range. Remaining materials have a random concentration.

Secondly, we obtain the properties of the layer, calculated from the materials and their concentrations. In this step the relative thickness, similarity, blend value, and roughness are calculated:

$$V = \sum_{i=1}^{N} (v_i \cdot c_i) \tag{3}$$

where V is the property value, N is the number of materials in the layer, v_i is the property value of each material, and c_i is the concentration of each material.

Third, the algorithm generates the layer physically. This process uses Perlin noise [15] with the scale parameter provided by the designer as its frequency. Noise value is multiplied by the roughness of the layer and modified with the height of the last layer:

$$h = h_{-1} \cdot s + n \cdot (1 - s) \tag{4}$$

where h is the modified height, h_{-1} is the height of the last layer at the same point, s is the previously calculated similarity, and n is the Perlin noise value. This equation modifies the new layer, depending on the similarity, to look like the last layer.

This height value is relative to the height of the layer. To obtain the final height of the layer we add its thickness and the average height of the previous layer.

Finally, the method generates the underground material flow. The process generates the flow tangent to the surface of the layer rotating on Y axis as established by an horizontal 2D Perlin noise.

4.2.2 Vein generation

First, the algorithm establishes a starting bounding square and a location for the vein. The length of this square side is randomly generated between two values given as a global parameter in the input data: $VeinArea/2$ and $VeinArea$. The location is completely random, only limited by the size of the terrain plot.

Second, the method obtains the materials of the vein. This process is similar to the material selector of the layers having two main differences: the method only gets the materials of the last layer located under the region of the vein, and it selects the main material among materials with vein parameter.

Third, the algorithm calculates the vein properties (relative area, roughness and relative thickness) following the same process as the generation of the layer characteristics. The vein final bounding square is calculated multiplying the relative area by the base bounding square previously calculated.

Fourth, the method generates the shape of the vein by locating and scaling the relative shape into the final bounding square. The algorithm generates this relative shape using a fractal generator based on the midpoint displacement method, which has been extensively used for terrain generation [11].

Our fractal algorithm generates the relative shape locating a square in a two-dimensional space and displacing its vertices ($P1$, $P2$, $P3$ and $P4$) along the square diagonals (Figure 3(a)). Then, it takes a random point (P') from a segment, displacing it along the line formed by the new point (P') and the square center (Figure 3(b)). This process is performed recursively in all the segments, including the ones created by the new points.

The point displacement value is randomly obtained, with a maximum calculated as:

$$d_i = (d_{i-1}/SmoothConstant)/2 \tag{5}$$

where i is the recursive depth in the algorithm, d_i is the maximum displacement at i recursive depth, d_{i-1} is the maximum displacement at $i-1$ recursive depth (d_0 is a constant), and $SmoothConstant$ is a constant value that indicates the roughness loss in each iteration. The values for both constants are taken from the roughness parameter of the materials. The recursive depth of fractal algorithm is limited to 5 due to time performance. The minimum point displacement is $-d_i$.

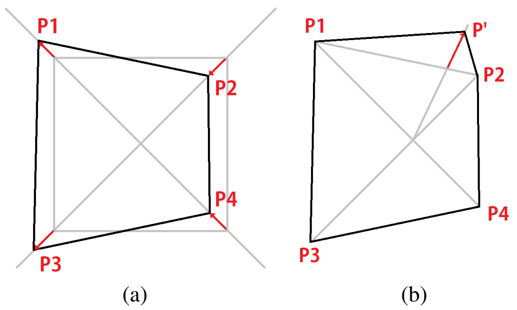


Fig. 3 Fractal algorithm for vein generation.

Fifth, we generate the heights of the vein, based on the minimum distance from each point to the fractal border. Both height and distance to the border are normalized, obtaining values in the range (0, 1). The method calculates the heights with a equation based on the circumference:

$$h_{xy} = |\sqrt{1-(d_{xy}-1)^2}| \quad (6)$$

where, h_{xy} is the height at $[x,y]$ point and d_{xy} is the distance from the $[x,y]$ point to the border. This equation is a simplification of one unit length radius circumference with the center at [1, 0] (maximum height in the center).

Then, our method multiplies these heights by the relative thickness and the global thickness of the vein. Then, it adds the height of the last layer to the result, getting the final height.

Finally, we generate the flow using the same process as the layer generation.

4.2.3 Cave generation

The cave generation process follows the same steps as vein generation process with some particularities.

First, as in the vein generation process, the algorithm establishes the base bounding square and the location. This process employs *CaveArea* instead of *VeinArea* to determine the size of the base bounding square.

Next, the method obtains the material of the cave and its properties. This process is performed in the same way as the vein material selection with two exceptions. The algorithm selects only one material, chosen within the set of materials with the cave special parameter defined. Then, the properties of the material are extracted, modifying the bounding square of the cave as in the vein generation process.

Finally, the method generates the final cave shape and heights with the same algorithms as the vein generation process.

4.2.4 Online approach

The previous sections show how the proposed method generates a terrain from scratch. The online generation has the peculiarity that it must be able to extend an already existing terrain. With our method, this process is very similar to the generation from scratch, except for the fact that the composition and properties of the layers are beforehand calculated.

First, the algorithm determines the number of veins and caves in the new terrain plot. This value is obtained randomly between 0.5 and 1.5 times the number of veins and caves in the original plot. While this process is perfomed against the number of veins and caves of the first plot of the terrain (which is generated using the density of veins and caves given in the input data), the density of those elements is preserved along the whole terrain.

Once the number of veins and caves is obtained, the method determines their position between the layers. This is a completely random process. The result of this step is a stack of layers, veins and caves.

Next, the method generates the elements of the stack beginning from the bottom. It extends the layers in the horizontal plane (along the X and Z coordinates) obtaining new heights using the Perlin noise algorithm. This step is the same as the layer generation from scratch process (see section 4.2.1), but the composition and properties of the layer are already calculated. The generation of veins and caves is performed following the same steps as before (see section 4.2.2).

4.3 Converting the terrain to a volume

This process converts the terrain composed by stacked layers, veins and caves to our representation, defining the mixed areas between layers. The algorithm multiplies the thickness by the blend value of each layer. The result is the height where the transition with the last layer ends. This process is responsible for removing sharp frontiers between layers, giving them a realistic appearance mixing their materials and flow.

5 Results

5.1 Speed & Size

We measure the time of the terrain generation using eight configurations. Table 2 shows the materials of these configurations (all of which have 1 void and 1 base additional materials). The value in brackets near the vein and cave materials is the density of veins and caves given in the input data for that configuration. All materials have the maximum affinity between them.

Table 2 Measured configurations.

	0	1	2	3
A	3 regular 0 cave 0 vein	3 regular 0 cave 1 vein(0.5)	3 regular 1 cave(0.5) 0 vein	3 regular 1 cave(0.25) 1 vein(0.25)
B	6 regular 0 cave 0 vein	6 regular 0 cave 2 vein(0.5)	6 regular 1 cave(0.5) 0 vein	6 regular 1 cave(0.25) 2 vein(0.25)

The method has been tested on an Intel(R) Core(TM) i7 950 CPU, running at 3.07GHz.

We generate 100 terrains of 5x5x5 chunks of 10x10x10 voxels with each configuration. Table 3 shows the average time and the standard deviation of the offline generation and the extension of those terrains with the online algorithm. Moreover, the table shows the average number of layers and elements (caves or veins) generated in the terrains. Additionally, we use a terrain generated by simple Perlin noise (PN in the table) as a baseline. This Perlin Noise process generates the internal flow and the material of each voxel.

Table 3 Time results.

	Offline average time (ms)	Offline standard deviation	Online average time (ms)	Online standar deviation	Avg. layers	Avg. elements
A0	319.14	41.94	310.59	32.20	4.65	0.00
A1	347.01	61.66	329.70	55.76	4.72	1.74
A2	334.85	53.49	312.20	44.75	4.81	1.55
A3	348.46	75.01	342.84	60.35	4.68	1.73
B0	324.36	46.85	309.24	32.29	4.75	0.00
B1	354.25	70.23	315.28	47.86	4.88	1.71
B2	354.30	62.27	316.40	46.28	4.79	1.90
B3	346.67	55.55	314.86	47.18	4.73	1.78
PN	61.25	14.46	61.25	14.46		

Figures 4 and 5 provides a visual comparison of the different tested configurations for online and offline set-ups. In both setups, generations of caves increases the computing overhead of our method whereas the most time-consuming configurable feature of our method is the generation veins. Even though, the method was able to generate terrains with both features online. It is also noticeable that the required time for generating terrains with a complex material configuration is not significantly higher than the time required with simple material configuration.

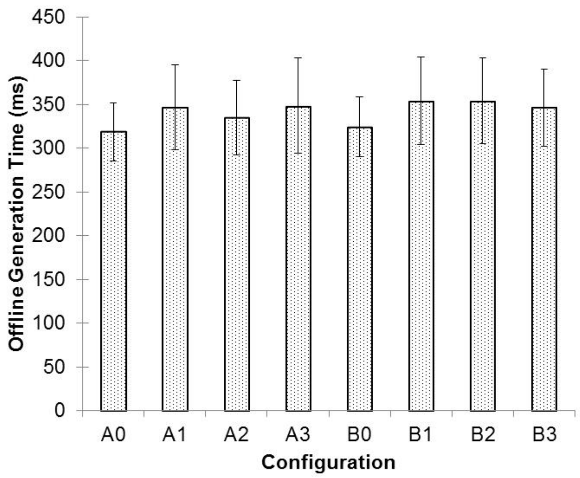


Fig. 4 Comparison of the different configurations of our method for the offline setup.

The average number of limits of the generated terrains is 84,376.42, while in a traditional volumetric terrain based on voxels and with the same dimensions as our generated terrains the number of voxels are 125,000.

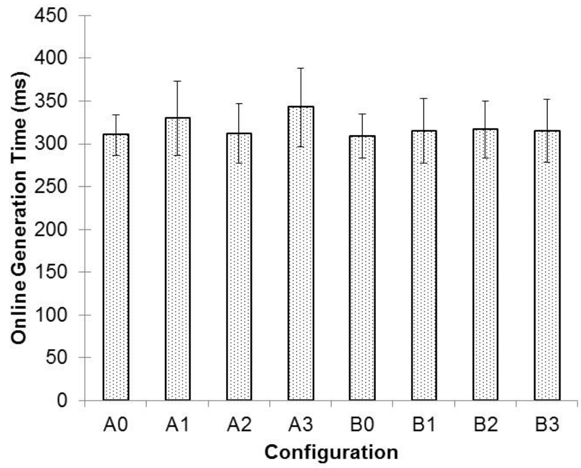


Fig. 5 A terrain generated with our method.

5.2 Realism

The proposed method is based on common knowledge, generating results similar to sedimentary terrains. Nonetheless, it is not based on real geological data and it does not simulate the real world. In this section we compare some results of our method against real terrains [1].

Figure 6 shows slices of real terrain representations (subfigures 6(a), 6(b) and 6(c)) and results of our method (subfigures 6(d), 6(e), 6(f) and 6(g)). It is obvious that our method does not generate simulations of real terrains, but it is possible to observe several similarities, such as stacked layers (all the subfigures) or propagated folds over the layers (subfigures 6(b) and 6(f)). In spite of this similarity, there are some real constructions that our method can not manage, such as the shown in subfigure 6(c). Figures 1 and 7 show more terrains generated with our method.

6 Conclusions

In this paper, we propose a method to generate volumetric terrains with mixed materials and natural and realistic appearance, giving the designer high control over the result. To store this terrain, we use an specific volume representation method that reduces the memory requirements. In addition, we propose a set of metrics and requisites to evaluate these type of generators.

Analyzing the results, our method seems to accomplish all the requisites established in Table 1 for several reasons:

– Innovation: The method uses a probabilistic method, so the elements are completely unexpected.

[1] Reproduced with the permission of the British Geological Survey ©NERC. All rights Reserved

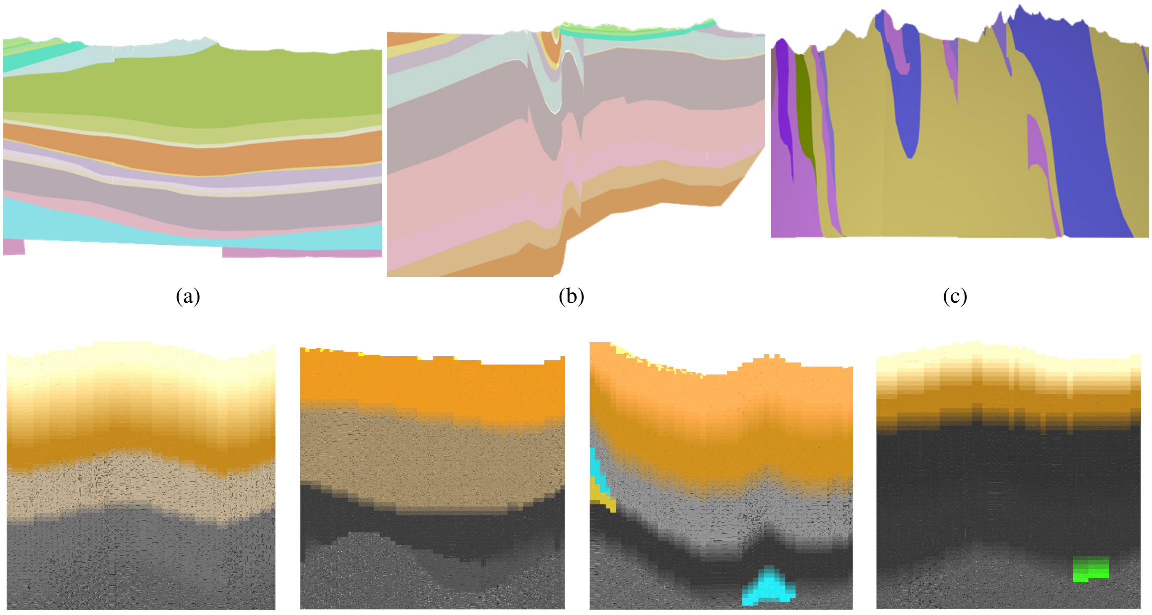


Fig. 6 Comparison against real world terrains.

- Structure: Generated layers, veins and caves are constructed in a coupled stack. Moreover, the probabilistic method uses material affinities to combine the structures. Additionally, the new elements have a similar shape to the last layer, depending on the similarity value of the materials forming it.
- Interest: The results are theoretically infinite, so they are prone to be explored. Moreover, the distributed resources over the terrain motivate this exploration.
- Speed: The method can run in real time.
- Usability: All the input is related to the terrain, avoiding technical concepts.
- Control: All aspects of the results can be specified by the designer, and the method is not interactive.
- Ampliability: Modifying the input data is possible to generate any type of layered terrain.
- Scalability: The scale of the result is determined with only one parameter.
- Realism: The generated terrains are not simulations of real world, but they are based on it.

To prove this accomplishment, more qualitative validations are needed, such as a questionnaire among users to measure the interest and the realism, or among designers to measure the usability and the control.

Finally, we conclude with an overview of the method particularities.

On one hand, the method presents some weaknesses. First, we observe that veins penalizes the execution time. However, the method is still able to run in real time. Second, our method only generates isolated caves, so terrains are not able to contain complex systems of caves, which is a usual feature of volumetric virtual worlds. Therefore, this aspect should be enhance in the future. Finally, terrains top surface is a simple Perlin noise. This problem could be tackled from any of the published methods for erosion simulation, being Benes and Frosbach [1] approach specially well suited as it applies this process to volumetric data.

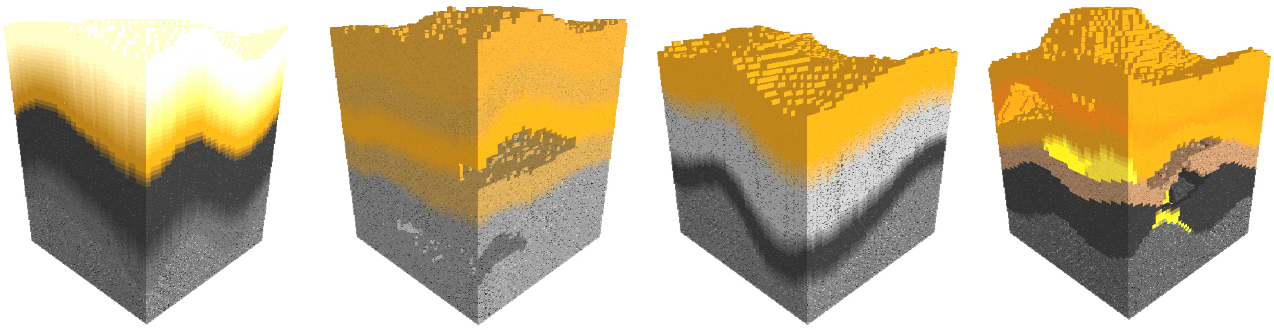
As future lines of work, we propose an increase of the terrain characteristics, such as complex cave systems or a more detailed surface. Additionally, we are working on more validations for the proposed method implementing it in a real video-game.

On the other hand, our method has several strengths. First, our method enables the designer to construct any type of terrain, being useful to several genre and settings of games. Second, the method combines the materials into layers, mixing them and interpolating the voxels, giving a natural appearance to the result. Last but not least, the memory requirements are reduced. These characteristics make easier the transmission of the terrain data in shared virtual worlds.

Acknowledgements We would like to thank Ivan García-Ferreira for his support over the research process.

References

1. Benes, B., Forsbach, R.: Layered data representation for visual simulation of terrain erosion. In: Computer Graphics, Spring Conference on, 2001., pp. 80–86. IEEE (2001)



(a) (b) (c) (d)

Fig. 7 Results obtained with our method.

2. de Carpentier, G.J.P., Bidarra, R.: Interactive gpu-based procedural heightfield brushes. In: Proceedings of the 4th International Conference on Foundations of Digital Games, FDG '09, pp. 55–62. ACM (2009)
3. Doran, J., Parberry, I.: Controlled procedural terrain generation using software agents. Computational Intelligence and AI in Games, IEEE Transactions on **2**(2), 111–119 (2010)
4. Fournier, A., Fussell, D., Carpenter, L.: Computer rendering of stochastic models. Communications of the ACM **25**(6), 371–384 (1982)
5. Frade, M., Vega, F., Cotta, C.: Evolution of artificial terrains for video games based on accessibility. In: Applications of Evolutionary Computation, *Lecture Notes in Computer Science*, vol. 6024, pp. 90–99. Springer Berlin Heidelberg (2010)
6. Gain, J., Marais, P., Straßer, W.: Terrain sketching. In: Proceedings of the 2009 symposium on Interactive 3D graphics and games, I3D '09, pp. 31–38. ACM (2009)
7. Hnaidi, H., Guérin, E., Akkouche, S., Peytavie, A., Galin, E.: Feature based terrain generation using diffusion equation. In: Computer Graphics Forum, vol. 29, pp. 2179–2186. Wiley Online Library (2010)
8. Kamal, K.R., Uddin, Y.S.: Parametrically controlled terrain generation. In: Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia, GRAPHITE '07, pp. 17–23. ACM (2007)
9. Miller, G.S.: The definition and rendering of terrain maps. In: Proceedings of the 13th annual conference on Computer graphics and interactive techniques, *SIGGRAPH '86*, vol. 20, pp. 39–48. ACM (1986)
10. Musgrave, F.K.: Methods for realistic landscape imaging. Ph.D. thesis, Yale University (1993)
11. Musgrave, F.K., Kolb, C.E., Mace, R.S.: The synthesis and rendering of eroded fractal terrains. In: Proceedings of the 16th annual conference on Computer graphics and interactive techniques, *SIGGRAPH '89*, vol. 23, pp. 41–50. ACM (1989)
12. Natali, M., Lidal, E.M., Parulek, J., Viola, I., Patel, D.: Modeling terrains and subsurface geology. In: Eurographics 2013, pp. 155–173. The Eurographics Association (2013)
13. Olsen, J.: Realtime procedural terrain generation. Tech. rep., University of Southern Denmark (2004)
14. Ong, T.J., Saunders, R., Keyser, J., Leggett, J.J.: Terrain generation using genetic algorithms. In: Proceedings of the 2005 conference on Genetic and evolutionary computation, pp. 1463–1470. ACM (2005)
15. Perlin, K.: An image synthesizer. SIGGRAPH Comput. Graph. **19**(3), 287–296 (1985)
16. Saunders, R.L.: Realistic terrain synthesis using genetic algorithms. Ph.D. thesis, Texas A&M University (2006)
17. Shaker, N., Yannakakis, G.N., Togelius, J.: Towards automatic personalized content generation for platform games. In: Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, pp. 63–68. AAAI Press (2010)
18. Togelius, J., De Nardi, R., Lucas, S.: Towards automatic personalised content creation for racing games. In: Computational Intelligence and Games, 2007. CIG 2007. IEEE Symposium on, pp. 252–259. IEEE (2007)
19. Togelius, J., Preuss, M., Yannakakis, G.N.: Towards multiobjective procedural map generation. In: Proceedings of the 2010 Workshop on Procedural Content Generation in Games, PCGames '10, pp. 3–8. ACM (2010)
20. Voss, R.F.: Random fractal forgeries. In: Fundamental algorithms for computer graphics, pp. 805–835. Springer (1985)
21. Zhou, H., Sun, J., Turk, G., Rehg, J.: Terrain synthesis from digital elevation models. Visualization and Computer Graphics, IEEE Transactions on **13**(4), 834–848 (2007)