Polynomial methods for fast Procedural Terrain Generation

Yann Thorimbert

November 3, 2016

Abstract

We present a new method allowing to generate 3D terrain significantly faster than prevailing fractal brownian motion approaches, while producing results of equivalent quality. The method is derived through a novel, systematic approach that generalizes to an arbitrary number of spatial dimensions and gradient smoothness. The results are compared, in terms of performance and quality, to efficient gradient noise methods widely used in the domain of fast terrain generation: Perlin noise and OpenSimplex noise. Finally, we propose to objectively quantify the degree of realism of the results by performing a fractal analysis of generated terrains, and to compare it to real terrain data.

1 Introduction

1.1 Motivations

Procedural terrain generation (PTG) methods have grown in number in the last decades due to the increasing performances of computers ; video games, movies and animation find obvious use of PTG, for terrain generation as well as for texture generation. However, a less evident use of PTG can be found in more practical domains, as for instance vehicle dynamics [6] or military training [31, 30], where accurate methods for emulating real terrain are of interest. More generally, one can also mention the use made of procedural coherent noise in the field of fluid animation [3, 18], which helps to improve performances of turbulence modeling. The main advantages of procedural noise compared to non-procedural noise generation is both an immensely decreased memory demand and an increased amount of content produced. The drawback usually occurs at two levels : accuracy, which reflects terrain realism, and performance, since the generation step may induce an additional processing effort compared to meshes which are simply loaded from files. What makes a PTG method more appropriate than another one is therefore dependent of the needed levels of performance and realism. For instance, PTG for video game usually targets performance rather than accuracy [20, 11] in order to run on home computers,

as long as the produced terrain resemble at least superficially real ones. In the other hand, PTG used for vehicle simulation is highly demanding in terms of accuracy and less in terms of performance. Whatever the application, the capability of the method to describe a self-similar pattern is a crucial point ; a central issue in PTG is the apparent fractal behaviour of many natural patterns that numerical models should mimic in order to produce realistic shapes [15, 7, 24]. It has to be noticed that, although we do not investigate them, some other approaches like stochastic subdivision [14] allow to produce realistic, non-fractal results.

1.2 Related works

Most popular approaches for low-dimensional PTG include methods from the family of random midpoint displacement such as diamond-square algorithm [8, 16] and methods from the family of gradient noise, such as Perlin noise [25] and simplex noise [26]. As a result of the number of parameters influencing the generated terrain, noises from this family can be easily improved in terms of quality, as done for instance in [22]. While simplex noise is found to be clearly more efficient than Perlin noise for dimensions higher than 3, the difference in terms of performances is slighter for low dimensions (see Section 5.2). Many recent efforts have been done to develop alternative methods that we shall review here. The use of cellular automata [11] and tile-based procedural generation [10] has shown to allow either fast, non-realistic terrain generation, either to be too slow for realistic-shaped height map generation. Evolutionary algorithms have been investigated to assist fractal terrain generation for video games, although not giving satisfactory results [28]. While tectonic-uplift [29], hydrology-based [4, 9] and example-based [34] approaches can reach a high level of realism, they are significantly slower than Perlin-like methods. In particular, the latter has serious limitations in terms of autonomous procedural generation, since it involves the synthesis of a template heightmap given by the user, whose features are transcribed in the generated map.

Midpoint displacement methods are usually more interesting than gradient noise methods in terms of pure performance, thanks to the fact that they do not need multiple passes corresponding to the different octaves. However, they suffer two serious drawbacks, namely their non-locality (the h value at a given coordinate depend on its neighboring points) and the quality of the generated terrain, as pointed out by [16]. Although most of the visual artifacts can be corrected by the more complex scheme presented in [14], wich largely differs from the initial and simple idea of midpoint displacement, the fully local nature of algorithms such as Perlin noise allows to save a large amount of memory compared to non-local schemes and, moreover, is embarrassingly parallel. In addition, the nature of the midpoint displacement algorithms is such as it provides less control parameters than Perlin or simplex method. Finally, as for cellular automata, the non-locality of midpoint displacement causes additional difficulties for assembling different chunks of terrain, as commonly done in video games.

As a result of the specificities of the models discussed above, Perlin-like and simplex-like approaches clearly appear to be the most appropriate choice for fast generation of realistic terrains, thanks to the good compromise they provides between performance and quality. To illustrate this, one can observe that most of the recent video games using massive procedural content generation make use of Perlin noise or variant ; Minecraft (24 millions units sold between october 2011 and october 2016 [17]) or No Man's Sky (1.5 million units sold in the first three months [19]) to name but two (as a way of comparison, Tetris has been sold 30 million times since 1989 [32]).

For all the reasons mentioned, we shall focus in this article on comparisons of the presented model with Perlin and simplex noise in two dimensions, mapping two spatial quantities (x, y) to a scalar value h. Note that OpenSimplex algorithm is used in this study to compare to the presented method, as it provides noise that is very similar to simplex noise and that it is not patented, unlike original simplex noise.

To conclude this review of existing methods, note that many algorithms as for instance cell noise [33] or erosion modeling [2, 20] are normally not used on their own but rather applied to results obtained by previously cited methods in order to increase accuracy. In addition, one can mention procedural generation of human structures like cities [23, 5], which can be associated with terrain generation for human-impacted landscapes. Although these additional methods are not discussed here, it should be noted that the models described in this article are suitable for their use.

The aim of this study is to build a novel, general method based on boundary-constrained polynomials, which enclose Perlin noise, and to derive an optimized model for producing 2D heightmap using a minimum number of operations per pixel.

2 Polynomial terrain generation model

2.1 Generalized boundary condition

Consider a D-dimensional domain called cell, and restricted from 0 to 1 in each axis of the space : $x_a \in [0, 1], \forall a \in [1, D]$. The position of a point in this cell is then characterized by a set of D values $\mathbf{X} \equiv (x_1, x_2, .., x_D)^T$. The set of points S for which all coordinates values are either 0 or 1 constitute the corner points of the cell. The cardinal number of S in D dimensions is equal to 2^D.

Let h be a function allowing to associate a height value $h(\mathbf{X})$ to each point of the domain. Here we want to impose a height value $h(\mathbf{s}_i)$ to each corner point $\mathbf{s}_i \in S$. Similarly, values for spatial derivatives of h can be imposed. Defining the partial mixed derivatives $h^{\mathbf{d}}$ as:

$$h^{\mathbf{d}} \equiv \frac{\partial^{d_1+d_2+..+d_D}}{\partial^{d_1}x_1 \; \partial^{d_2}x_2 \; .. \; \partial^{d_D}x_D} h, \tag{1}$$

one writes $h^{\mathbf{d}}(\mathbf{X}) = h^{\mathbf{d}}(\mathbf{s}_i)$ if $\mathbf{X} = \mathbf{s}_i$. Note that if the mixed derivative of order

n is continuous, then the mixed partial derivative is unaffected by the ordering of the derivatives. By defining as Equation (1) the mixed partial derivative for a given $\mathbf{d}$ to be unique, we make the assumption of continuous mixed partial derivatives.

As a last requirement, we want $h(\mathbf{X})$ to depend only on $h(\mathbf{s}_i)$ $h(\mathbf{s}_j)$ if $\mathbf{X}$ is located on the edge connecting the two corners $\mathbf{s}_i$ and $\mathbf{s}_j$, so that if this edge is shared with another cell, the interface between cells is smooth. In the sequels we will refer to this condition as the edge boundary condition. Defining $\mathbf{\Delta}_{ij} \equiv \mathbf{s_j} - \mathbf{s_i}$, it reads:

$$h\left(\mathbf{s_i} + k \cdot \mathbf{\Delta}_{ij}\right) = f(\mathbf{s_i}, \mathbf{s_j}) \; \forall k \in [0,1] \Leftrightarrow \|\mathbf{\Delta}_{ij}\| = 1, \tag{2}$$

where $f(\mathbf{s_i}, \mathbf{s_j})$ must be a differentiable function of $\mathbf{s_i}, \mathbf{s_j}$ only.

2.2 Polynomial definition

Many choices are possible at this point for the form of h ; we propose here to describe it as a multivariate polynomial of degree n:

$$h(\mathbf{X}) = \sum_{\mathbf{a} \in I} \left(c_{\mathbf{a}} \prod_{k=1}^{D} x_k^{a_k} \right), \tag{3}$$

where I is the set of all vectors on the form $(a_1, a_2.., a_D)$ such that $0 \leq a_k \leq n \; \forall k$.

A specific nomenclature may be defined for sake of clarity. Noting m the order of the highest constrained derivative, one can refer to a given cell configuration by DdMmNn. For instance, D2M1N4 stands for a cell of dimension 2, constrained on value and on first derivative, and whose h value is given by a polynomial of degree 4. Note that not all configurations make sense, as for instance D1M8N1, a polynomial for which the number of constraints is obviously higher than the number of coefficients.

Together with the value constraints and derivative constraints defined above, as well as the edge boundary condition Equation (2), Equation (3) leads to a system of linear equations which can be solved in order to determine polynomial coefficients, as long as the number of constraints does not exceed the number of coefficients, whose value is $(n+1)^D$. The number of constraints can be viewed as the number of corners times the number of constraints per corner. Since the number of different derivatives of degree m is the combinations with replacements of D elements on m length sequence, one can express the total number of constraints as:

$$2^D \cdot \sum_{i=0}^{m} \frac{(D+i-1)!}{i!(D-1)!} \leq (n+1)^D. \tag{4}$$

A cell whose polynomial and constraints configuration obey this inequality is then guaranteed to obey the specified constraints on corners points.

3 Special cases in two dimensions

We present in this section three special cases of 2D polynomials. First, we give the minimum polynomial that can generate 2D terrain with both height and gradient imposed, the D2M1N3 polynomial. We then briefly describe how usual Perlin noise correspond to an order 5 polynomial, and we finally show how to simplify D2M1N3 by assuming zero gradient on corners.

3.1 D2M1N3 polynomial

Let us now restrict to the case where $D = 2$ and where the derivative are constrained up to order 1. In this case, the number of constraints provided by Equation (4) is equal to 12, and the smallest degree n which allows the polynomial to respect the constraints is 3. In this configuration, Equation (3) simplifies to:

$$h(x, y) = \sum c_{ij} x^i y^j, \tag{5}$$

where i and j are integers comprised between 0 and 3, usual axes names x and y now stands for x_1 and x_2, and c_{ij} denotes polynomial coefficients c_{i_1,i_2}. Defining $f(x, y)$ and $g(x, y)$ as the x-component, respectively y-component of the gradient at position (x, y), one obtains:

$$f(x, y) = \sum_{i>0} i \cdot c_{ij} x^{i-1} y^j, \tag{6}$$

and

$$g(x, y) = \sum_{j>0} j \cdot c_{ij} x^i y^{j-1}. \tag{7}$$

We now define h_{ij} to be the requested height on corner at coordinates (i, j), so the height constraint reads $h(0, 0) = h_{00}$, $h(1, 0) = h_{10}$, and so on. Similarly, four conditions arise from the x-gradient conditions f_{ij} and four from the y-gradient conditions g_{ij}, defining a system of twelve linear equations.

A possible solution for this system of equations is:

$$c_{00} = h_{00}, \tag{8}$$

$$c_{10} = f_{00} \tag{9}$$

$$c_{22} = c_{33} = c_{32} = c_{23} = 0, \tag{10}$$

$$c_{20} = 3(h_{10} - h_{00}) - 2f_{00} - f_{10}, \tag{11}$$

$$c_{30} = f_{10} + f_{00} - 2(h_{10} - h_{00}), \tag{12}$$

$$c_{21} = 3(h_{11} - h_{01}) - 2f_{01} - f_{11} - c_{20}, \tag{13}$$

$$c_{31} = f_{11} + f_{01} - 2(h_{11} - h_{01}) - c_{30}, \tag{14}$$

$$c_{11} = h_{01} + h_{10} - h_{00} - h_{11} + f_{01} + g_{10} - g_{00} - f_{00}. \tag{15}$$

Note that for symmetry reasons, non-diagonal terms c_{ji} can all be deduced from c_{ij} by inverting all indices and replacing f by g. For instance, $c_{02} = 3(h_{01} - h_{00}) - 2g_{00} - g_{01}$. With these coefficients, it is easy to check that edge boundary condition is verified.

3.2 Perlin's polynomial

In Perlin's method, a grid of gradient values is generated and the height value of a subdomain is obtained by interpolating the height contribution of each corner's gradient. The smooth interpolation function S can be of any form as long as $S(0) = 0$ and $S(1) = 1$. It is most common to use polynomial of order 3 (smoothstep) or 5 (smootherstep) [27]. Smoothstep reads $S_3(x) = 3x^2 - 2x^3$ and is the lowest order polynomial to provide zero-derivative at $x = 0$ and $x = 1$ along with the condition aforesaid. The height value of a point in the domain then reads:

$$h(x, y) = h_0(x, y) + S(y) \cdot (h_1(x, y) - h_0(x, y)) , \tag{16}$$

with

$$h_0(x, y) = v_{00}(x, y) + S(x) \cdot (v_{10}(x, y) - v_{00}(x, y)) \tag{17}$$

and

$$h_1(x, y) = v_{01}(x, y) + S(x) \cdot (v_{11}(x, y) - v_{01}(x, y)) , \tag{18}$$

where $v_{ij}(x, y) = f_{ij} \cdot (x - i) + g_{ij} \cdot (y - j)$.

With our definition of multivariate polynomial degree, it appears that this scheme makes use of a polynomial of degree $2 + s$, where s is the degree of the chosen smoothstep polynomial. The consequent total order is at least 5, a value higher than for D2M1N3 polynomial presented above, which is of degree 3. However, due to the factorized form it offers, Perlin noise allows to gain computation steps compared to D2M1N3, resulting in better performances ; Zero-gradient D2M1N3 presented below, in the other hand, will need less operations than Perlin's polynomial.

3.3 Zero-gradient D2M1N3 polynomial

Forsaking the generality of D2M1N3 polynomial derived above, one can impose special gradient conditions in order to increase performances of the implementation of the polynomial. The condition reads $f_{ij} = g_{ij} = 0 \ \forall i, j$ and Equations (8-15) simplify, yielding the following expression for $h(x, y)$:

$$h(x, y) = h_{00} + S_3(x)\Delta x + S_3(y)\Delta y + A \left[S_3(x) \cdot y + S_3(y) \cdot x + xy \right] . \tag{19}$$

with $\Delta x = h_{10} - h_{00}$, $\Delta y = h_{01} - h_{00}$ and $A = h_{11} + h_{00} - h_{10} - h_{01}$, and as before S_3 is the third order smoothstep function. It is worth noting that the only cell-dependent terms in this expression are h_{00}, Δx, Δy and A ; this allows for important performance gain when using lookup tables for evaluating space-dependant terms (*i.e* terms involving x or y), since only cell-dependant terms have to be evaluated, as a result of their dependancy to boundary constraints, that are not known before terrain generation. The quality of the generated terrain is not affected in comparison with the generic version of the polynomial

(see Section 5.1). Note that S_3 could be replaced by any higher order smoothstep function S_i, in a very similar way as in Perlin noise.

Zero-gradient D2M1N3 polynomial is the minimum configuration (*i.e* the configuration that has the lowest number of coefficients) for smooth two-dimensional heightmaps. Indeed, $N = 3$ cannot be reduced since no polynomial of degree lower than 3 can take arbitrary height and derivative values at boundaries. In addition, $M = 1$ cannot be reduced by definition as we are seeking for smooth heightmaps, that are necessarily constrained on first derivative. Thus, the only way to reduce the number of coefficients is to impose special values at the constrained locations. Examining Eqs (8-15), it appears that setting $f_{ij} = g_{ij} = 0$ allows to cancel two coefficients and reduce the expression of the other ones.

3.4 Other polynomials of interest

The systematic approach described in Section 2 is general and can be applied for dimension 2 as well as any dimension D, although simplex-like algorithms have shown to be more efficient for high dimensions (see Section 1). Unidimensional equivalent to D2M1N3 is D1M1N3, which leads to smoothstep function $S_3(x) = 3x^2 - 2x^3$ if specific conditions $h(0) = 0$, $h(1) = 1$ and zero gradients at boundaries are required. A 3D equivalent to D2M1N3 configuration would be D3M1N3. In addition to the generality of dimension, constraints determine the quality of the generated terrain. If the accuracy of the terrain is a priority, one may consider to impose gradients constraints for higher orders. In particular, with a view to improve isotropy, one may consider to use $M = 2$ and impose second order, mixed derivative gradients $\partial^2 h/\partial_x^2$, $\partial^2 h/\partial_y^2$, $\partial^2 h/(\partial_x \partial_y)$ in addition to first order gradients. This would lead to D2M2N4 and D3M2N4 models.

4 Fractal noise

In order to produce convincing results, it is of crucial importance to observe and reproduce the apparent fractal nature of real terrains. This is achieved by adding together multiple layers of height maps commonly called *octaves*, generated by the same method but which lower in amplitude a_i as they grow in frequency f_i, with i the number of the octave. More specifically, $a_i \propto a^{-i}$ and $f_i \propto f^i$. When $a \approx f \approx 2$, the amplitude of the deformation is always proportional to the scale at which it is applied ; this is how self-similarity arise from the generated terrain. It is worth noticing that most authors refer to $1/f$ as the lacunarity and is then seen as the multipler of frequency between two successive octaves. Finally, a is often called persistence. Note that in this study these parameters are chosen once and for all before data is produced and do not influe the performance of an implementation. For all the results presented in this article, base frequency is equal to 2 and lacunarity is equal to 0.5, unless otherwise specified.

A property of D2M1N3 makes it a particular model compared to Perlin polynomial and zero-gradient D2M1N3 presented below ; while both of these

have, by construction, either zero gradient or zero height on the corners of the domain, generic D2M1N3 can take any arbitrary value on these locations. For this reason, one may define an additional parameter w which weights the value of gradient in order to tune the predominance of either height or gradient at the corners. For terrain generation, w is typically a constant, since the scale (*i.e* the octave) seems to have no influence on the slope of the added values, in first approximation. In the sequels we set $w = h_0/100$, where h_0 is the amplitude of polynomial at octave zero.

4.1 Time cost

We study in this paragraph the computational time T for generating a height map using the method presented in this study. Let us consider first the case of a single octave. For a domain of size R, the computational time of the evaluation of a polynomial over the domain is equal to the number of evaluation points times the cost of the evaluation of a single point. In d dimensions it reads $T_{eval} = C_{eval} \cdot R^d$, with C_{eval} the cost of a single point evaluation, whose value depends on the specific type of polynomial used. In addition, each polynomial coefficient needs to be computed once and for all before spatial evaluation. At a given octave level i, there are $2^{i \cdot d}$ different polynomials to initialize, leading to a creation cost equal to $T_{init} = C_{init} 2^{i \cdot d}$, where C_{init} is the time needed for deducing polynomial coefficients from boundary conditions, which again depends on the specific type of polynomial used. Note that C_{eval} and C_{init} are typically of the same order of magnitude.

The total cost $T = T_{eval} + T_{init}$, including the N octaves, is simply the sum of the cost for each octave from 0 to N. One obtains $T \approx C_{eval} N R^d + C_{init} 2^{Nd}$. Let us now consider the two-dimensional case. The previous expression may be misleading, as the second term usually becomes negligible compared to the first one, despite the exponential behaviour of the latter, since dimension $d = 2$, resolution $R \approx 10^3$ and $N \approx 7$ or smaller in most cases ; in this configuration, non-exponential term is of the order of 10^7 while the exponential one is of the order of $2^8 \approx 10^3$. Thus for usual, low number of octaves, the evaluation dominates the total cost of terrain generation, while polynomial initialization dominates the cost for high number of octaves. Performance test depicts this effect in Section 5.2. This behaviour is related to the fact that the contribution of each octave exponentially decays. As a result, the global change of shape of the generated terrain, from the point of view of a given scale, is exponentially smaller. This can be observed in Figure 1 and is the reason why the exponential component of the time complexity can be neglected in most cases, since the typical resolution used is large enough to make it much smaller than the linear component up to approximately 7 octaves.

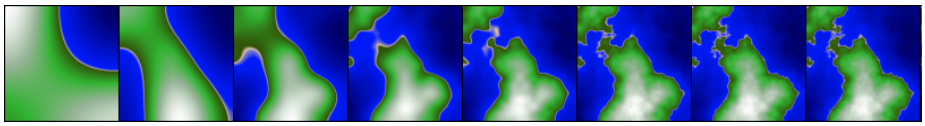

Figure 1: Example of map generated with 1 to 8 octaves, from left to right. Zero-gradient D2M1N3 model was used. As a result of the exponentially decaying height contribution, the global change of shape of the generated terrain, from the point of view of a given scale, is exponentially smaller.

5 Results

5.1 Visual comparison with other methods

An example of terrain produced with different methods is shown in Figure 2, along with its first and second order gradients. In addition to the three methods previously described, we also used Perlin's model with a fifth order interpolation polynomial $S_5(x) = 6x^5 - 15x^4 + 10x^3$, as well as OpenSimplex method as a way of comparison. No quality difference can be visually noted between models ; acceptable isotropy (*i.e* not visible for human eye) at all orders can be observed, although it can be pointed out that simplex model provides values that are distributed slightly more evenly, as it can be seen from the first order gradient norm.

Figure 3 depicts different ways to visualize coherent noise produced by zero-gradient D2M1N3 model, and in particular how 2D height map data is used to generate a 3D rendered mesh. Figure 3 provides different types of landscapes that can be obtained with coherent noise, for each method. Again, no quality difference can visually be noted between models. Finally, Figure 4 shows typical examples of coherent noise used to add turbulence to sine signal, in order to procedurally generate textures of marble, wood or stone, as first indicated in [25], as well as a cloud-like texture directly obtained from the raw noise.

5.2 Performance comparison with other methods

As seen in Section 4, D2M1N3 methods and Perlin noise have the same complexity in time, which is $\mathcal{O}(NR^2 + 2^{2N}) \approx \mathcal{O}(NR^2)$ up to $N \approx 8$ in two dimensions. However, their performances may substantially differ, since constants C_{eval} and C_{init} are different for each method. We shall estimate these values here. Let us examine Eqs. (8-15) first. For convenience, we consider purely space-dependant terms as having null cost, since they can be pre-computed and then accessed in constant time (this is equivalent to assume that the zoom cannot be adjusted once the execution starts). If this is not the case, they globally represent the same amount of effort in all polynomials whose order are similar, thus they can be ignored with reasonable accuracy. With these assumptions, one counts $a_g = 11$ additions and $m_g = 12$ multiplications for D2M1N3 polynomial. Equation (19) yields $a_z = 3$ additions and $m_z = 3$ multiplications for zero-gradient

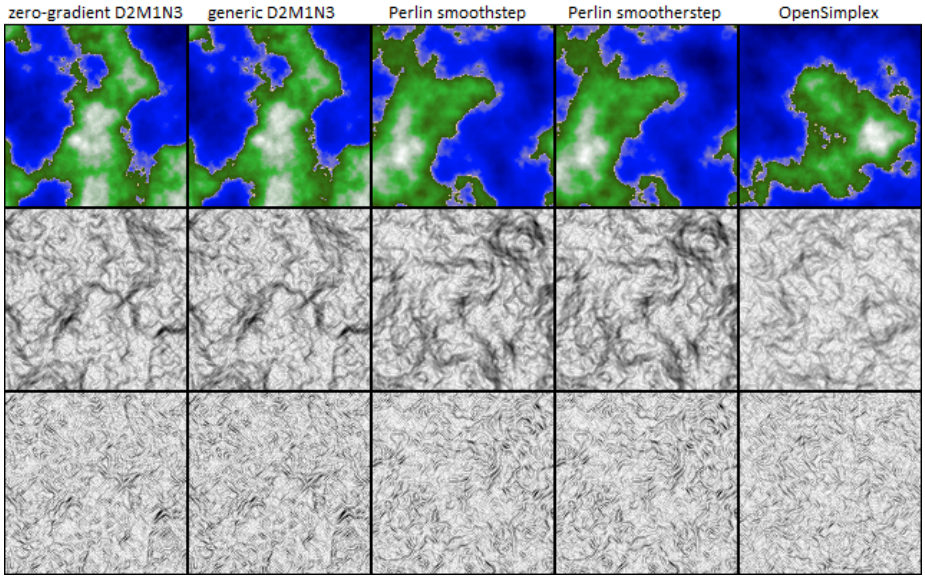

Figure 2: Example of terrain generated with generic D2M1N3, zero-gradient D2M1N3, third order smoothstep, fifth order smoothstep Perlin's model and OpenSimplex scheme on each column respectively. Height, height gradient norm and height second order gradient norm are represented on each line, for each corresponding column model. Specific color map was used to visualize terrain. Gray scale was used to visualize gradients, where white means zero and black means maximum value.

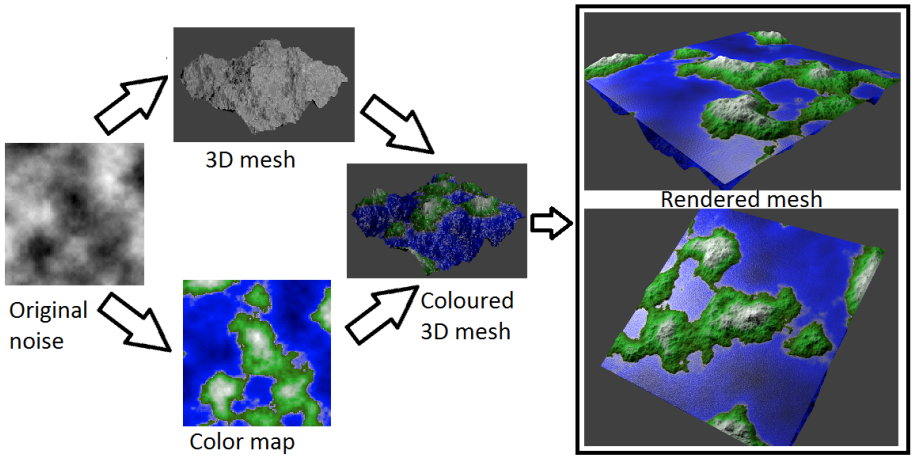

Figure 3: Process leading to 3D mesh, from raw noise to rendered mesh.

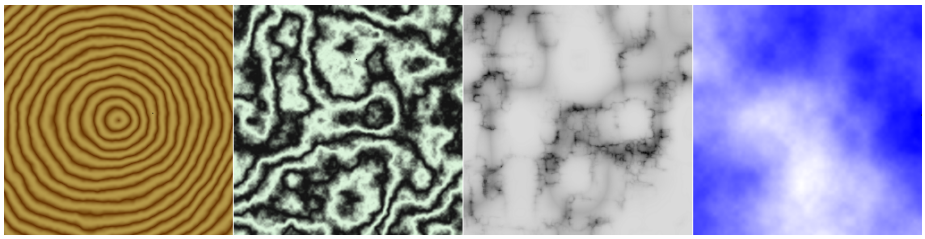

Figure 4: Examples of Wood, marble and cloud textures obtained with zero-gradient D2M1N3 model.

D2M1N3 polynomial. Finally, Perlin's scheme is found to include $a_p = 6$ additions and $m_p = 7$ multiplications. By way of comparison, examination of an efficient implementation of 2D simplex noise yields 6 additions and 10 multiplications [26]. Observing that $a_g \approx 4a_z$, $m_g \approx 4m_z$, $a_p \approx 2a_z$, $m_p \approx 2m_z$, one obtains that the ratios $C_{eval,z}/C_{eval,g} \approx 4$ and $C_{eval,z}/C_{eval,p} \approx 2$ are independant of the specific cost of addition and multiplication on the machine used. For this reason, zero-gradient D2M1N3 model is expected to run approximately twice faster than Perlin's model and four time faster than generic D2M1N3 model. This estimation tends to be weaker as the number of octaves becomes larger, as contribution of C_{init} tends to be significant.

Performances measurements have been done on four different home computers: Intel Core 2 Duo E6550 @ 2.33GHz (denoted Core2 Duo in the figures), Intel Core i5-5200U @ 2.20GHz, Intel Core i7-4500U @1.8GHZ, and Intel Core i7-4770 @ 3.40GHz. For each method, a C code and a Python code have been used (these codes are available as supplementary material). Execution time $T_m(N, R)$ for a given number of octaves N and resolution R have been obtained, for each method m and with each machine, by averaging the total execution time for 1000 terrain generations. Figure 5 shows the normalized execution time for zero-gradient D2M1N3, Perlin and OpenSimplex models as a function of the number of octaves N. The normalized execution time is obtained by dividing the execution time T by the execution time T_{Z0} for $N = 1$ with zero-gradient D2M1N3 method. The linear behavior of the execution time clearly indicates that the computational effort is dominated by C_{eval} up to $N = 9$. Figure 6 displays an example of the square root of the normalized execution time, this time as a function of the resolution R, with a fixed number of octaves $N = 3$. Again, it is evident that the cost is dominated by pixel evaluation at all tested resolutions. Finally, the time advantage of zero-gradient D2M1N3 method can be quantified by defining the speedup $S_m = T_m/T_Z$, where subscript T_m is the execution time for any method m and Z denotes D2M1N3 method. Figure 7 shows the different speedups obtained, taking into account all the acquired data for all machines and methods. It appears that C version of zero-gradient D2M1N3 method is on average 33 % faster than Perlin method, while Python version is more than twice faster. OpenSimplex implementations are approximately four times slower.

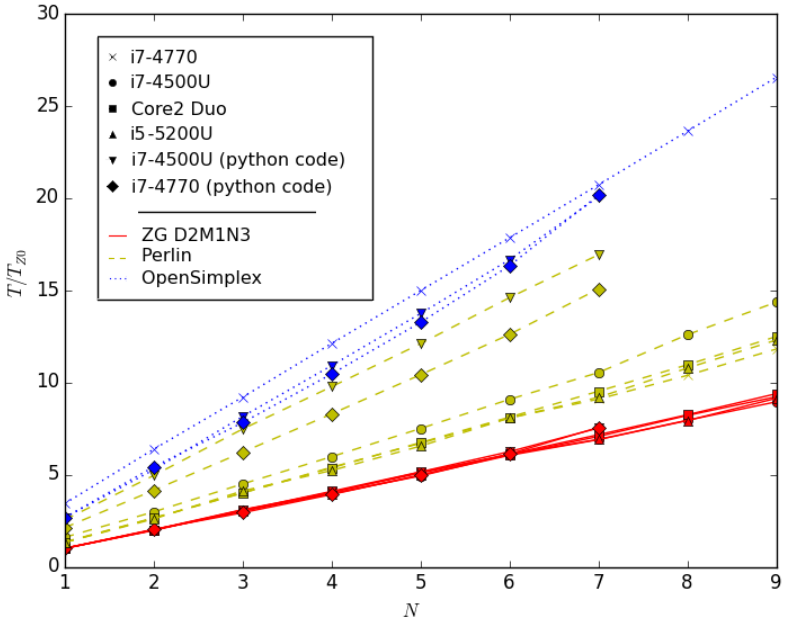

Figure 5: Measured normalized execution time as a function of the number of octaves N, for resolution $R = 1024$ pixels. Data is presented for all methods and for all machines tested. T is the measured execution time and T_{Z0} is the measured execution time for zero-gradient D2M1N3 method with $N = 1$ octaves. The linear behaviour show that the computational cost is dominated by pixel evaluation and that coefficients calculation is negligible for $N < 9$.

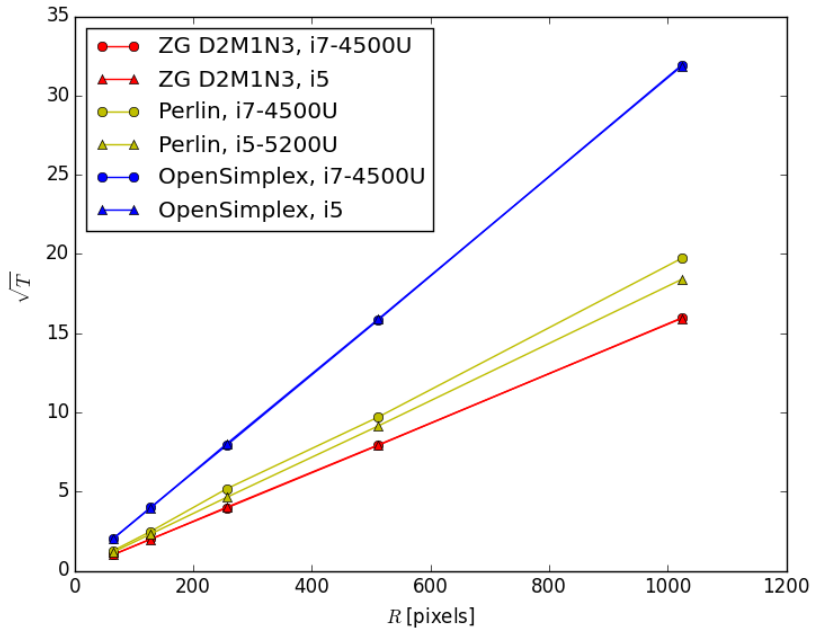

Figure 6: Example on two different machines of the square root of the measured execution time as a function of the resolution R, for $N = 3$ octaves. The expected quadratic behaviour in R shows that the computational cost is dominated by pixel evaluation C_{eval} and that coefficients calculation is negligible for common resolutions.

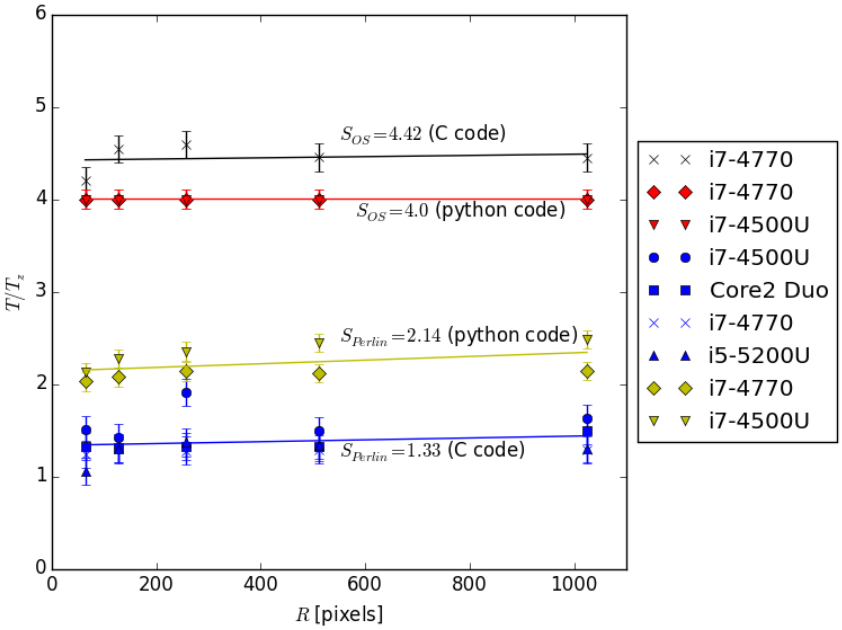

Figure 7: Speedup values of zero-gradient D2M1N3 over Perlin and OpenSimplex methods.

5.3 Fractal analysis as a measure of realism

The evaluation of how much a generated terrain is realistic undoubtedly depends on arbitrary choices ; a man who never saw an island in his life would judge a real island as 'unrealistic' compared to all landscapes he saw before. In the other hand, it seems reasonable and intuitive to base our evaluation on quantities that can be measured both in real and numerical terrains, and which can reflect in a simple manner the complexity of a given topological configuration. For this reason, fractal dimension is a classical choice to characterize landscapes, and coastlines in particular [15, 12]. We propose here to study the fractal dimension corresponding to the coastlines of the terrains generated with our model, and to compare it with real, natural terrains' fractal dimension.

Let L be the total length of a given coastline ; it is clear that its value depends on the length ϵ of the measuring tape : as it becomes smaller, more details can be measured and L therefore increases. This effect is known as Richardson effect and is reported in [15]. Fractal dimension D is defined as the quantity allowing to link ϵ to L by a power law : $L(\epsilon) = k \cdot \epsilon^{1-D}$, with k a constant. It follows that $\log(L) = (1-D)\log(\epsilon) + C$ with C a constant, thus the fractal dimension can be deduced from the slope of the function $\log(L)$, whose value is $1 - D$. A formal study of fractal dimension can be found in [1].

Before proceeding to a numerical measurement of D for a given data set, we shall convince ourselves that a superimposition of an infinity of 1D sinusoids with exponentially decreasing amplitudes and exponentially increasing frequency leads to a theoretical value $1 \leq D \leq 2$. Sin functions are considered in order to easily handle periodicity. We define the height function at octave i as $h_i(x) = a^{-i}\sin(f^i x)$, with real constants a and f. The crest resulting from the superimposition of all N octaves is:

$$H(x) = \sum_{i=0}^{N} h_i(x), \tag{20}$$

for $x \in [0, 2\pi]$. An example of the obtained crest is shown in Figure 8, for different number of octaves. Using that $\partial h_i(x)/\partial x = (f/a)^i \cos(f^i x)$ The total length of the crest can be expressed as:

$$L(N) = \int_0^{2\pi} \sqrt{1 + \left(\sum_{i=0}^{N} (f/a)^i \cos(f^i x)\right)^2}\, dx, \tag{21}$$

From the mathematical point of view, the actual length L of the crest is the length obtained with an infinite number of octaves. In the other hand, as previously said, we are looking for a relationship on the form $L(N) = k\epsilon^{1-D}$. Consequently, we are left with:

$$D = \lim_{(N,\epsilon)\to(\infty,0)} \frac{-\log L(N)}{\log \epsilon} + 1. \tag{22}$$

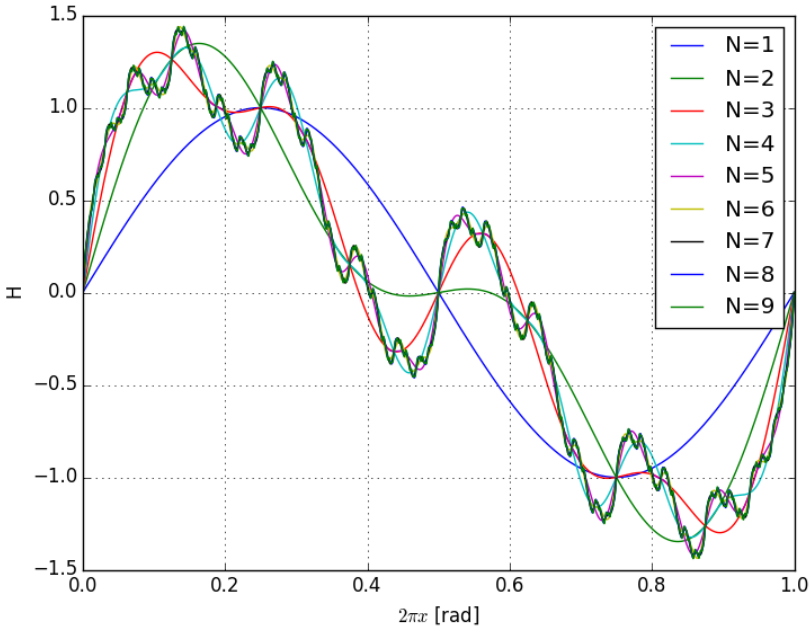

Figure 8: Superimposition $H(x)$ of sinusoids with increasing frequencies and decreasing amplitudes. Here $a = f = 2$.

Because of the difficulty to solve the integral in Equation (21), we provide in Figure 9 numerical evidences of the convergence value of D for different choices of parameters a and f.

In order to measure D for a given numerical terrain, we generate it once and for all with a given number of octaves N and a resolution $R = 1024$. We then choose several, exponentially decreasing values for ϵ, to which are associated squares of side length ϵ. By counting, for each given ϵ, how many squares are needed to cover all the coastline of the height map, one is able to deduce D from a linear regression of $log(L)$ plot, as explained above. Figure 10 depicts the process, for an example map of 512×512 pixels. In Figure 11 are displayed fractal dimension reported in [15] from Lewis Richardson's empirical work for South Africa, Germany land frontier and West coast of Britain ; these experimentally found quantities are compared to data generated with zero-gradient D2M1N3 method for different values of persistence. Figure 12 shows how D evolves as the number of octaves used for generating the height map grows, again for different values of persistence, as expected after studying the behaviour of fractal sinusoids. One can note that the results are essentially similar between zero-gradient D2M1N3, Perlin and OpenSimplex algorithms. It should be noted that even though D may continue to grow significantly, the

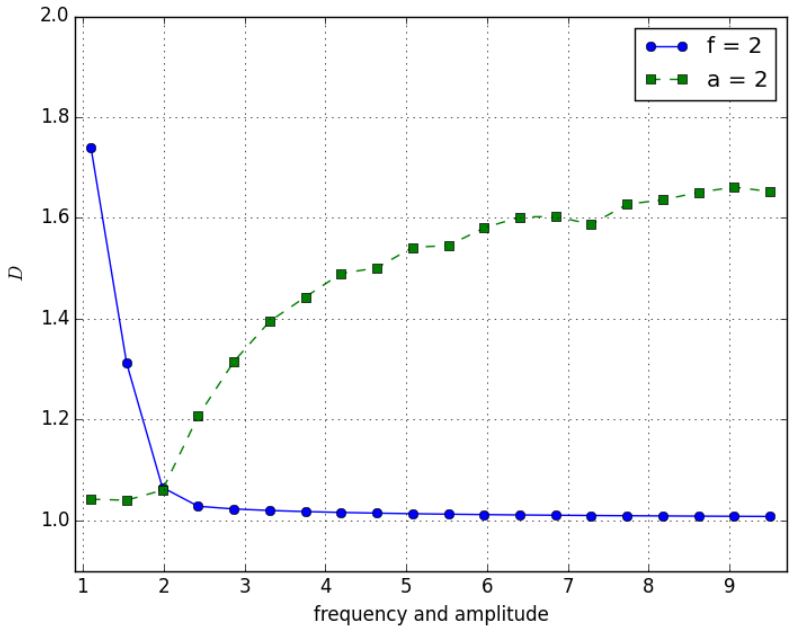

Figure 9: Fractal dimension D as a function of frequency f and amplitude a. As expected D is close to 2 when a is small, in other words when low octaves do not predominate much over high octaves. Reversely, D is close to 1 when low octaves almost entirely determines the shape of the crest.

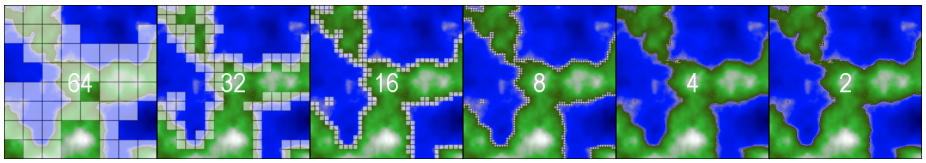

Figure 10: Example of box counting process used for computing fractal dimension of a generated map. Box size varies from 2 to 64 pixel and is indicated on each image. In this example, the terrain size is 512×512 pixels large.

human, visual appreciation of the result is limited at approximately 8 octaves for common model parameters, as argued below.

It is worth noting that fractal dimension of coastlines is not by itself a complete measure of terrain realism ; Koch snowflake has very little resemblance with real terrain, while its fractal dimension is 1.26 [12]. Therefore, it seems judicious to associate D with another criterium whose role is to quantify how much terrain is uneven. Although we do not investigate this point, we indicate here that Kullback-Leibler [13] divergence could be a suitable choice for measuring the similarity between height maps as a complement to fractal analysis.

6 Conclusion

Two main uses can be made of the model, either by using zero-gradient D2M1N3 model to increase performance compared to standard Perlin noise, or by using a higher-order scheme (and in particular constrained mixed derivatives) with the aim to arbitrarily constrain gradient. While the former correspond to typical video-games demand, the latter may find use in scientific and industrial research, as discussed in the introduction.

Strengths

The new method described allows to significantly improve performances of realistic terrain generation based of fractal brownian motion. In addition, the general model proposed allows to reach an arbitrary level of gradient smoothness. The scheme for noise generation consists in solving, once and for all at the theoretical level, a linear set of equations whose order depends on the number of desired constraints on gradients. We have shown that zero-gradient D2M1N3 model is the minimal configuration for smooth 3D polynomial terrain generation. Another benefit of the presented method is that, unlike simplex models, it is very similar to Perlin noise, which means that quality and performance improvements developed for Perlin noise such as [22, 21] can be straightforwardly applied to it.

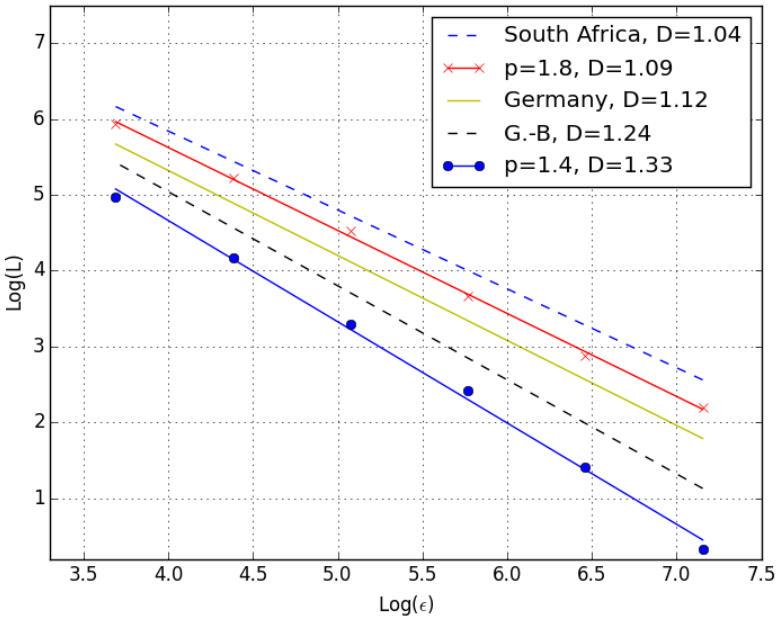

Figure 11: Example of computed fractal dimension for different values of persistence p and comparison with values for South Africa, Germany and Great-Britain coastline obtained by Richardson and reported in [15]. The plot display the logarithmic value of coastline length L as a function of the logarithmic value of the box size ϵ, and the fractal dimension is the corresponding gradient. Note that ordinate of the curves has been arbitrary chosen for convenience, as we are only interested in their slopes.

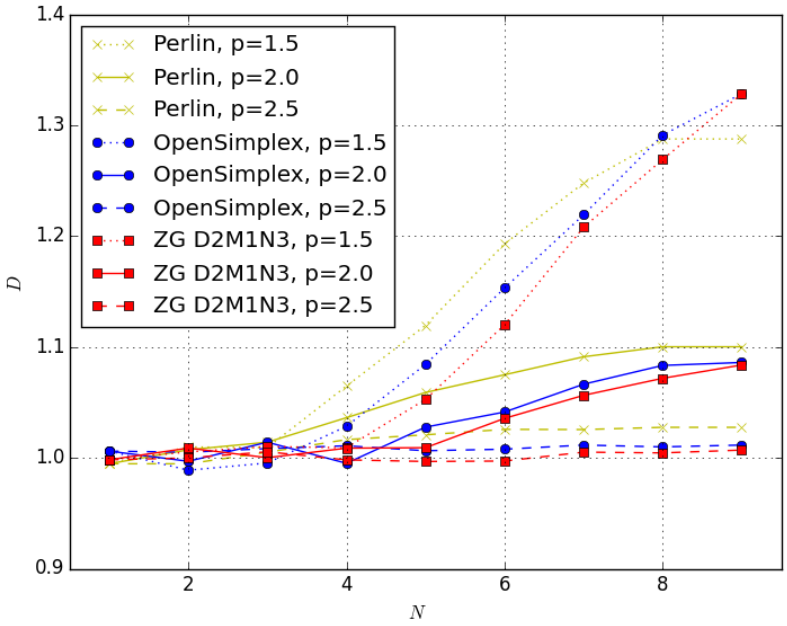

Figure 12: Computed fractal dimension D as a function of the number of octaves N used to generate the height map, for different values of persistence p. The results of zero-gradient D2M1N3 method are compared to Perlin and OpenSimplex models.

Limitations

The performance gain of the model is paid by the loss of intuition compared to Perlin polynomial, who can be seen as a simple interpolation of height values generated from corner gradients on a grid. Moreover, for dimensions higher than 4, simplex noise has proven to give better performances than Perlin model [26]. Finally, for generation of terrain including caves, a common approach is to use a 3D height map for which some values are interpreted as void or air ; in that case, a performance study comparing zero-gradient D3M1N3 model to Perlin and simplex methods should be performed in order to determinate how the performance advantage of zero-gradient method is diminished.

Acknowledgments

The author thanks Gregor Chliamovitch, Bastien Chopard and Paul Albuquerque for assistance with fractal analysis, and Mohamed Ben Belgacem for assistance with C implementations of the models.

References

[1] Michael Barnsley. *Fractals Everywhere*. Academic Press Professional, Inc., San Diego, CA, USA, 1988.

[2] B Benes and R Forsbach. Visual simulation of hydraulic erosion. *Wscg'2002, Vols I and Ii, Conference Proceedings*, pages 79–86, 2002.

[3] Robert Bridson, Jim Houriham, and Marcus Nordenstam. Curl-noise for procedural fluid flow. In *ACM SIGGRAPH 2007 Papers*, SIGGRAPH '07, New York, NY, USA, 2007. ACM.

[4] Guillaume Cordonnier, Jean Braun, Marie-Paule Cani, Bedrich Benes, ric Galin, Adrien Peytavie, and ric Gurin. Large Scale Terrain Generation from Tectonic Uplift and Fluvial Erosion. *Computer Graphics Forum*, 2016.

[5] Minh Dang, Stefan Lienhard, Duygu Ceylan, Boris Neubert, Peter Wonka, and Mark Pauly. Interactive design of probability density functions for shape grammars. *ACM Trans. Graph.*, 34(6):206:1–206:13, October 2015.

[6] Jeremy J Dawkins, David M Bevly, and Robert L Jackson. Evaluation of fractal terrain model for vehicle dynamic simulations. *JOURNAL OF TERRAMECHANICS*, 49(6):299–307, 2012.

[7] David S. Ebert, Steven Worley, F. Kenton Musgrave, Darwyn Peachey, Ken Perlin, and Kenton F. Musgrave. *Texturing and Modeling*. Academic Press, Inc., Orlando, FL, USA, 2nd edition, 1998.

[8] Alain Fournier, Don Fussell, and Loren Carpenter. Computer rendering of stochastic models. *Commun. ACM*, 25(6):371–384, June 1982.

[9] Jean-David Génevaux, Éric Galin, Eric Guérin, Adrien Peytavie, and Bedrich Benes. Terrain generation using procedural models based on hydrology. 32(4):143:1–143:??, July 2013.

[10] David Grelsson. *Tile Based Procedural Terrain Generation in Real-Time*. PhD thesis, 2014.

[11] Lawrence Johnson, Georgios N. Yannakakis, and Julian Togelius. Cellular automata for real-time generation of infinite cave levels. In *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*, PCGames '10, pages 10:1–10:4, New York, NY, USA, 2010. ACM.

[12] Jay Kappraff. The geometry of coastlines: a study in fractals. *Computers & Mathematics with Applications*, 12(3):655 – 671, 1986.

[13] S. Kullback and R. A. Leibler. On information and sufficiency. *Ann. Math. Statist.*, 22(1):79–86, 03 1951.

[14] J. P. Lewis. Generalized stochastic subdivision. *ACM Trans. Graph.*, 6(3):167–190, July 1987.

[15] Benoit B. Mandelbrot. *The Fractal Geometry of Nature*. W. H. Freeman and Company, 1977.

[16] Gavin S P Miller. The definition and rendering of terrain maps. *SIGGRAPH Comput. Graph.*, 20(4):39–48, August 1986.

[17] Minecraft sales. `https://minecraft.net/fr/stats/`. Online; accessed 20 October 2016.

[18] Rahul Narain, Jason Sewall, Mark Carlson, and Ming C. Lin. Fast animation of turbulence using energy transport and procedural synthesis. In *ACM SIGGRAPH Asia 2008 Papers*, SIGGRAPH Asia '08, pages 166:1–166:8, New York, NY, USA, 2008. ACM.

[19] No Man's Sky sales. `http://www.vgchartz.com/game/84663/no-mans-sky/`. Online; accessed 20 October 2016.

[20] Jacob Olsen. Realtime Procedural Terrain Generation. 2004. Technical Report, University of Southern Denmark.

[21] Ian Parberry. Amortized noise. *Journal of Computer Graphics Techniques (JCGT)*, 3(2):31–47, June 2014.

[22] Ian Parberry. Modeling real-world terrain with exponentially distributed noise. *Journal of Computer Graphics Techniques (JCGT)*, 4(2):1–9, May 2015.

[23] Yoav I. H. Parish and Pascal Müller. Procedural Modeling of Cities. *28th annual conference on Computer graphics and interactive techniques*, (August):301–308, 2001.

[24] Heinz-Otto Peitgen and Dietmar Saupe, editors. *The Science of Fractal Images*. Springer-Verlag New York, Inc., New York, NY, USA, 1988.

[25] Ken Perlin. An image synthesizer. *SIGGRAPH Comput. Graph.*, 19(3):287–296, July 1985.

[26] Ken Perlin. Noise Hardware. *SIGGRAPH Course Notes*, 2001.

[27] Ken Perlin. Improving noise. *ACM Trans. Graph.*, 21(3):681–682, July 2002.

[28] William L. Raffe, Fabio Zambetta, and Xiaodong Li. A survey of procedural terrain generation techniques using evolutionary algorithms. *2012 IEEE Congress on Evolutionary Computation*, pages 1–8, 2012.

[29] J R Rankin. The Uplift model Terrain generator. 5(2):1–8, 2015.

[30] Ruben M. Smelik, Tim Tutenel, Klaas Jan de Kraker, and Rafael Bidarra. Declarative terrain modeling for military training games. *Int. J. Comput. Games Technol.*, 2010:2:1–2:11, January 2010.

[31] Ruben M. Smelik, Tim Tutenel, Klaas Jan de Kraker, and Rafael Bidarra. A case study on procedural modeling of geo-typical southern afghanistan terrain. In *Proceedings of IMAGE 2009*, St.Louis, Missouri, jul 2009.

[32] Tetris sales. `http://www.vgchartz.com/game/4534/tetris/`. Online; accessed 20 October 2016.

[33] Steven Worley. A cellular texture basis function. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, pages 291–294, New York, NY, USA, 1996. ACM.

[34] Howard Zhou, Jie Sun, Greg Turk, and James M. Rehg. Terrain synthesis from digital elevation models. *IEEE Transactions on Visualization and Computer Graphics*, 13(4):834–848, July/August 2007.