


## Procedural Modelling of Terrains with Constraints

Cristina Gasch  · Miguel Chover ·  
Inmaculada Remolar · Cristina Rebollo

Received: date / Accepted: date

**Abstract** Terrain is an essential part of any outdoor environment and, consequently, many techniques have appeared that deal with the problem of its automatic generation, such as procedural modeling. One form to create terrains is using noise functions because its low computational cost and its random result. However, the randomness of these functions also makes it difficult to have any control over the result obtained. In order to solve the problem of lack of control, this paper presents a new method noise-based that allows procedural terrains creation with elevation constraints (GPS routes, points of interest and areas of interest). For this, the method establishes the restrictions as fixed values in the heightmap function and creates a system of equations to obtain all points that they depend this restrictions. In this way, the terrain obtained maintains the random noise, but including the desired restrictions. The paper also includes how we apply this method on large terrain models without losing resolution or increasing the computational cost excessively. The results show that our method makes it possible to integrate this kind of constraints with high accuracy and realism while preserving the natural appearance of the procedural generation.

**Keywords** terrain modelling · procedural generation · Perlin noise · GPS routes

---

✉ Cristina Gasch  
Institute of New Imaging Technologies, Universitat Jaume I, 12006 Castelln, Spain.  
E-mail: cgasch@uji.es

Miguel Chover  
E-mail: chover@uji.es

Inmaculada Remolarr  
E-mail: remolar@uji.es

Cristina Rebollo  
E-mail: rebollo@uji.es

## 1 Introduction

The representation of natural environments is essential in a wide range of applications, such as geographic information systems [41], flight simulations or videogames. The terrain is a crucial part of these outdoor environments and its representation has been widely analyzed in the literature [39,42]. Some studies search to extract real data [23], but this limits the results to the real world and it is not always possible to obtain all real data to be able to represent them. That is why automatic terrain generation is very important, a field of research still active. Taking advantage of their fractal dimensions [24,38], many studies have appeared in the literature that address the automation of its creation process [40], mainly using procedural modelling. This modelling method has been applied to represent many features related to terrains, including roads, trees, villages, cityscapes [9,18,30], or even to represent atmospheric phenomena [37]. According to the work of Smelik et al. [45], procedural modelling were initially based on subdivision processes, in which the height map was iteratively created and, in order to create the elevation, a controlled amount of randomness was added in each iteration.

The first subdivision algorithm on which is based procedural modelling is known as the midpoint offset method [10,26], in which a new point is generated in every iteration. The elevation of this new point is obtained as a result of the average of the elevations of its neighbours, with the addition of a random displacement which decreases with each iteration. Many other studies, mostly based on noise generators such as the Perlin method [32,33] have been done in which the height map is obtained by a turbulence function. Perlin noise is fast, but it tends to create terrains that are uniform and random, often requiring significant post-processing to add interesting features. There are some works that deal with this problem, such as the one presented by Parberry [31] that uses a variant of Perlin noise, called value noise, to generate geotypical procedural terrain based on a spatial analysis of elevation data extracted from GIS data. The advantage of the approaches based on noise generators is that the computational cost is low, due to the fact that every point can be calculated independently of its neighbours.

However, all the methods described above have problems to controlling the generated terrain. Although some of them allow small constraints to be applied to the initial parameters, such as the maximum height, the procedural creation of a terrain that fits more specific constraints is difficult to make efficiently. Elements such as roads and rivers require a precise control to be able to include them in the final height map. Generally, these elements have specific forms and characteristics that condition the ground shape and including them in the terrain without control can be cause unrealistic results. For example, roads cannot be vertical, rivers do not flow upwards, and so forth. To represent large terrains with different geometric features, Bruneton et al. [4] combined a digital elevation model with layered geographic information system (GIS) vector data. Thus, they were able to obtain precise boundaries of roads, rivers, lakes and fields and enforce vector data constraints in the terrain.

According to the work of Smelik et al. [45], the solution provided by the studies that attempt to solve this problem is mostly based on the trial-and-error method. Some of them even need to indicate where the modifications have to be made, something that may not be trivial for users, since it is almost impossible to reliably predict how each operation will affect the final result. Many commercial tools have appeared that allow the user to manually model, generate, and modify the terrain [50,55,53,22]. However, procedural terrain modelling needs to be controllable in order to simplify content creation.

The initial problem to be solved was the integration of real GPS routes on procedurally generated terrains. Some applications, such as the video games for training, require to include real GPS routes and the rest of the terrain could be generated randomly. These applications also need to use automatic creation methods that do not need the interaction of the user during the creation process.

After studying the previous works, it has been found that noise-based works allow for variance in the final results, but at the moment there is none that includes previously established data on their results. Considering the problem, it was decided to create a new noise-based method to take advantage of its randomness and its characteristics, such as low computational cost but that allows to include precise restrictions in the final result.

The method solves an equation system that includes these constraints and generates a custom height map. The terrains generated with this new method are similar in appearance to those created with common noise functions, but including these previously established constraints, that the user can change, to design a random and at the same time custom terrain. This method computes noise functions with constraints that can be applied to any problem where the user needs to have some kind of control on the result.

The rest of the work is organized as follows. Section 2 show the previous work. Section 3 explains our method as well as certain concepts needed to understand it. Section 4 considers the creation of large terrain models. Next, Section 5 offers the results obtained. Finally, conclusions are presented in section 6.

## 2 State of the art

One of the most used terrain generating methods is the procedural modelling. It is mainly based on the fact that terrains are self-similar [24] and the most of works that manage this generation method use noise functions to obtain a random edition of the terrain height values [26,27].

Some of these works have included some constraints to condition the resulting terrain. According to the moment in which these constraints are applied, they can be mainly classified in two groups: methods that generate a terrain that includes the desired elements, and methods in which the final terrain is obtained by applying the required constraints to an already created terrain.

## 2.1 Generation of terrain with constraints

In order to create a realistic terrain automatically, one of the most important characteristics is to be able to generate it including constraints imposed by the user, so that it fits the terrain to be represented as much as possible. For this purpose, different methods have been developed.

Some studies are based on the modification of the method of terrain generation using mid-point displacement [10,26]. Kamal et al. [21] present a technique that generates a mountain by taking its elevation and base propagation as a constraint. Belhadj [1] presents a more general method in which a set of values constrain the mid-point displacement process. The constraints consist of scattered points of elevation provided by a satellite or other geological data acquisition sources.

A larger-scale method is the one proposed by Doran et al. [8], in which, using agents, constraints are added during the generation of the terrain, thereby creating geographical features such as mountains, beaches, etc. Although it allows for control over the terrain relief, it does not allow us to know the accurate position where these occurrences take place [45].

Some works initially generate the constraints and then model the terrain based on them. This is the case of the procedural approach presented by G enevaux et al. [14] that uses river networks for large terrain modeling. The landscape is generated by combining procedural terrain and river patches with blending and carving operators, but the user lacks fine control over the result.

The work of G enevaux et al. [15] requires the user to define and distribute many different primitives by hand. They combine feature-based primitives and hierarchical procedural modelling to generate the terrain, including rivers. Gu erin et al. [16] extend this model and present a hierarchical sparse terrain model that defines continuous terrains with varying level of detail. The approach represents terrains as elevation functions built from a sparse combination of automatically distributed primitives and, linear combinations of land-form features stored in a dictionary.

Recently, Gasch et al. [11] present a method of automatic terrain creation with constraints that force some height values to concur with a series of points determined by a GPS route. The limitations of this method are given by the size and complexity of the routes, since the accuracy diminished as they became bigger. This fact restricts its use to artificial routes, as tests carried out using real routes result in unrealistic maps.

## 2.2 Adaptation of the terrain to constraints

The following methods adapt an already created terrain to include elements such as roads or rivers. They can be classified depending on the user interaction required.

### *2.2.1 Results not controlled by the users*

This group considers works in which, although users can interactively apply some processes to a terrain to add its natural elements, they do not have any control on the final result.

A realistic way to modify terrain to simulate soil erosion caused by air or water is by using physics-based algorithms. Kelley et al. [19] propose a method based on empirical erosion models used in geomorphology to simulate the erosion of stream networks, adjusting the surface by triangulation and fractal interpolation. With the aim of improving the representation of relief, the approach proposed by [27] calculates the erosion of the water on the surface depending on the amount of water that would fall in each vertex of a fractal height field and the influence of the thermal weathering. They use a global simulation model. The work of Nagashima [28] presents an algorithm that adjusts valleys and mountains to the rainfall erosion and thermal weathering by using earth layer patterns on their surfaces. Neidhold et al. [29] combine a fluid simulation with an erosion algorithm. This fact enables to create eroded terrain interactively by changing simulation parameters or applying additional water to the scene. All these algorithms produce simulations that are difficult to control and unpredictable, and interactive editing cannot be done for large areas of terrain because calculations require a lot of time and memory, and a high computational cost.

To accelerate the simulation of terrain erosion, some works downloaded the simulation to a GPU [25,48]. Due to memory requirements and GPU limitations, these methods were not suitable for large terrain modeling. In order to simulate large-scale terrain affected by physics-based erosion, Vanek et al. [52] use adaptive tiles on the GPU. They divide the terrain into tiles of different resolutions according to their complexity. Then, each tile is stored as a mip-map texture and a mip-map pyramid is built for each tile. During the simulation process, they use the appropriate level of detail depending on the changes in the terrain.

Kristof et al. [20] present a hydraulic erosion simulation using the Smoothed Particle Hydrodynamics (SPH), which implies lower memory usage and better performance for large scale simulations as compared to earlier approaches. A stochastic and interactive simulation of the impact of various geomorphological agents on terrain erosion in conjunction with the vegetation simulation is presented in the work of Cordonnier et al. [5]. During the simulation process, the input elevation map is modified using a discrete layer model in which different geomorphological and ecological events modify the data stored in these layers.

Some studies allow for the inclusion of roads in the terrain, such as the one proposed by Stachniak et al. [47]. This method requires the initial approximation of the terrain and a function that looks for an acceptable set of deformation operations to apply to a random terrain in order to adapt it to the constraints. The problem is that it is computationally expensive. The work presented by Zhou et al. [58] allows roads, rivers, crests or valleys to be gener-

ated. They use a sketch generated by the user that, when linked to the height map of the terrain, produces the terrain following that crest-shaped drawing. This method allows a large terrain to be modified, although the final height cannot be predicted accurately.

### *2.2.2 Results controlled by the users*

This group involves methods where the users have control over the final result of the terrain appearance.

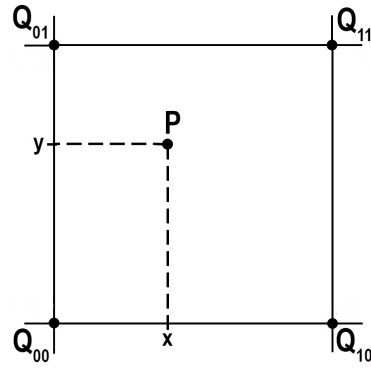
Based on Perlin noise, Scheneider et al. [44] introduce an editing environment where it is possible to edit the terrain by interactively modifying the base functions of the noise generator. Their method replaces the Perlin noise grid with a set of user-drawn greyscale images. Gain et al. [12] allow the user to draw the silhouette and the boundaries of a mountain chain through the propagation of noise, the terrain grows until the constraint is reached. To facilitate user interaction and make it more intuitive, Smelik et al. [46] propose a method in which a top view of the map is drawn, covering both heights and soil material. Rusnell et al. [43] base the terrain synthesis method on the interpretation of a weighted graph generated from a set of generator vertices.

The work presented by Hnaidi et al. [17] allows a designer to draw 3D curves to control the shape of the terrain generated. Each curve is enriched with different properties, such as elevation and slope angle, which become constraints during the modelling process. Bradbury et al. [3] present a sketch-based toolset that allows the manipulation of virtual terrains. The method works with a layer-based interface. Each layer includes three sub-layers showing, respectively, the high, medium and low frequency bands of the image. The user can mix these bands and apply different edits to each, as desired. The method for the creation of islands developed by Puig-Centelles et al. [35,36] is also included in this category, in which the resulting island created in real time can be observed through an interface.

Other methods use interfaces to allow the user to modify the terrain [2] or even to carve them in real time [6]. In these approaches, users paint a height-mapped terrain directly in 3D by applying simple terrain raising brushes and also GPU-based brushes that generate several types of noise in real time. The model proposed by Tasse et al. [49] combines the sketching interface and patch-based terrain generation in a hybrid system that gives wider control to the user and produces realistic terrains.

In an attempt to bring interactivity and user control in the synthesis of the terrain, the work of [13] combines user-defined maps with silhouette strokes used to define roughness. They use parallel pixel-based texture synthesis to interactively create the terrain from a database of height field examples, and gain control over the height and slope of the terrain by means of constraint points and curves. This method does not capture the large-scale terrain shape characteristics of real landscapes.

In 2017, Zhang et al. [57] presented a method to procedurally generate rivers and then adapt their routes according to a set of image features that are



**Fig. 1** Graphical representation of the terms involved in obtaining  $\text{Noise}(P)$ .

extracted by user interaction. The method adapts the initially generated rivers to these features by applying an iterative optimization in order to increase the similarity between the resulting rivers and the images.

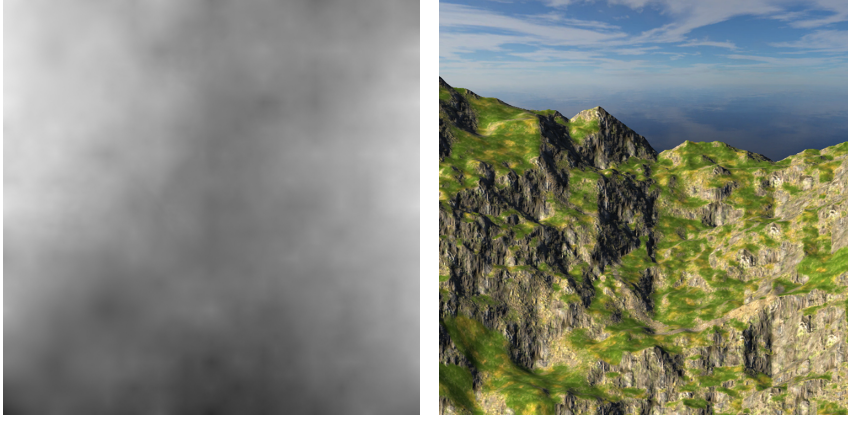
Modifications made through user interaction have the advantage that it is much easier to know what the final result will be like, or at least facilitates trial and error. However, the disadvantage is that to get accurate results, the user has to invest time in previously creating the terrain [45].

After reviewing all these works, it can be inferred that there is no method that generates a realistic terrain using procedural modelling that includes constraints defined by the user with a high accuracy. Moreover, in a simple way and without the need for high knowledge for to obtain the final result, being necessary the intervention of the user only at the beginning. In order to address this problem, the work described in the following sections is proposed.

### 3 Procedural generation of terrain with constraints

The presented work not only creates a height-map that includes some previously given features, but they are combined with some randomly generated heights for the rest of the terrain, that makes it possible to have a natural appearance. This randomness is generated by means of a noise function. After analyzing some noise generators, Perlin [32,33] has been used, since its low computational cost allows its use in real time and, in addition, its result maintains the pseudo-random appearance of nature.

According to the work of Perlin, the calculation of the noise in a point typically involves three steps. Initially, a two-dimensional grid is created, and a random value in the range of  $[0, 1]$  is assigned to each node ( $Q$ ). The next step is to determine in which cell in the grid is the point  $P$  to be calculated. Then the distance is obtained between the point  $P$  and every node  $Q$  of that cell. In a two-dimensional grid, this will require the computation of 4 distance



**Fig. 2** Procedural texture obtained with the Perlin method and its terrain representation.

vectors. The noise at the given point is finally obtained as the interpolation between the values computed at the vertices of the cell.

Considering a cell with four grid nodes  $Q_{00}$ ,  $Q_{01}$ ,  $Q_{10}$ ,  $Q_{11}$ , the noise value of a point  $P$  is based on a bilinear interpolation from the noise of the values of these nodes. Figure 1 illustrates the points and grid nodes involved in the process. The formula of this bilinear interpolation is presented in Equation 1.

$$\begin{aligned} Noise(P) => x \cdot y \cdot Q_{00} + x \cdot (1 - y) \cdot Q_{01} \\ + (1 - x) \cdot y \cdot Q_{10} + (1 - x) \cdot (1 - y) \cdot Q_{11} \end{aligned} \quad (1)$$

Turbulence is obtained at a point  $Turbulence(P)$  by considering the noise value of that point  $P$  on some levels, which are produced by iteratively dividing the grid by 2. If  $i$  is the number that controls the quantity of noise levels that have been taken into account to calculate the turbulence, then Equation 2 shows the final function proposed by Perlin to calculate it.

$$Turbulence(P) => \sum_i \frac{Noise(P \cdot 2^i)}{2^i} \quad (2)$$

Considering the values assigned to every point, a grey-scale image is generated for a determined turbulence. Terrain generation is achieved by interpreting this image as a height map so that the colour of each pixel is used as the value of the terrain elevation. An example of a procedural texture and the terrain that it generates is shown in Figure 2.

Perlin method [32] generates random values for each of the  $Q$  nodes involved in calculating the noise of  $P$ , so the value obtained in that point is also random. This work introduces constraints into certain points, for instance  $P$  has a value conditioned by the terrain features that has to appear in the final representation. In this case, the  $Q$  values that have an influence in  $P$  cannot be



randomly approximated. Instead of obtaining its value from the values of  $Q$ , our method changes the roles and the values that must be calculated are the points  $Q$  that influence  $P$ . This calculation is possible, establishing a system of equations using all equations obtained from all fixed values that are required for the given restrictions.

### 3.1 Case study

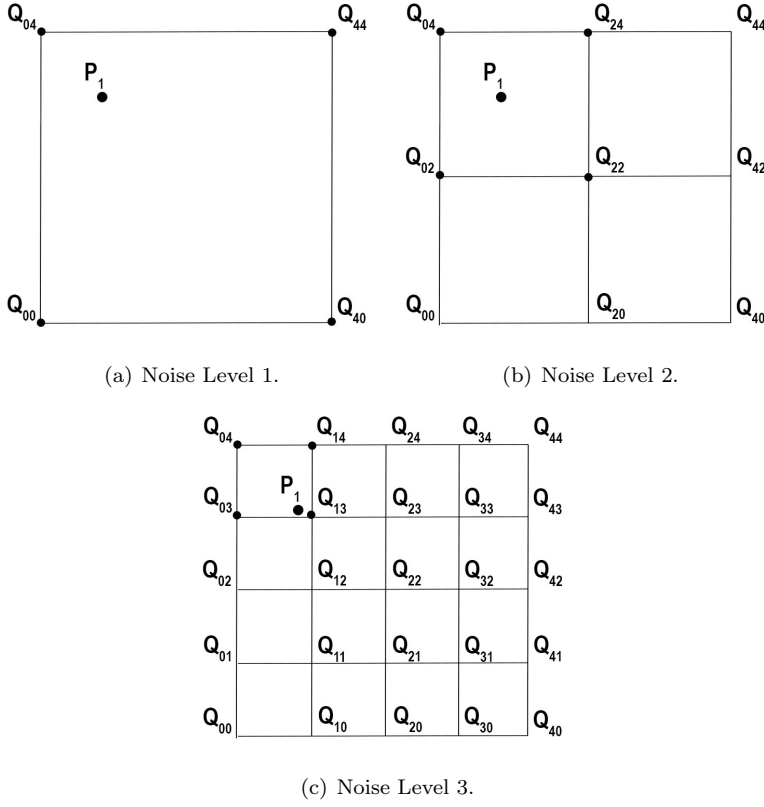
In order to make it easier to understand the method, the process of establishing the equation system is explained through an example. In this case, the turbulence is obtained combining three noise levels.

Figure 3 illustrates the data of the example. Initially, the value of the noise at point  $P_1$ ,  $Noise(P_1)$  is established with vertices ( $Q_{00}, Q_{04}, Q_{40}$  and  $Q_{44}$ ) on the first noise level (Figure 3(a)). According to Equation 1, these four vertices are necessary to obtain the noise of  $P_1$ . On the second level (Figure 3(b)), the grid is divided and five more vertices appear ( $Q_{02}, Q_{20}, Q_{22}, Q_{24}$  and  $Q_{42}$ ). However, only four of all the current vertices condition the value of  $P_1$  to obtain the  $Noise(P_1)$ . They have been marked with a circle to differentiate them from the rest. Finally, on the last considered level (Figure 3(c)), all the other vertices that complete the grid appear. This last level is where the vertices have the least distance between them, thereby giving the highest resolution. As on previous level, only three of these new values that need to be found intervene in obtaining  $Noise(P_1)$ , ( $Q_{03}, Q_{04}$  and  $Q_{13}$ ) in addition to  $Q_{14}$ .

Taking into account the equation shown in Equation 2, the turbulence  $Turbulence(P_1)$  is obtained by adding the noise on each of the three levels considered. The final formula is detailed in Equation 3, where the height value of the constraint is  $C\_height$ ,  $a_i, b_i, c_i$  and  $d_i$  replace  $xy, x(1-y), (1-x)y$  and  $(1-x)(1-y)$  respectively, from Equation 2. Finally, ten  $Q$  vertices condition the value of the turbulence at that point, leaving fifteen that do not intervene and which, as in the Perlin method, will receive a random value.

$$C\_height = \sum_{i=0}^3 \frac{Noise(P_1 \cdot 2^i)}{2^i} \Rightarrow \frac{a_1 \cdot Q_{00} + b_1 \cdot Q_{04} + c_1 \cdot Q_{40} + d_1 \cdot Q_{44}}{2^0} + \frac{a_2 \cdot Q_{02} + b_2 \cdot Q_{04} + c_2 \cdot Q_{22} + d_2 \cdot Q_{24}}{2^1} + \frac{a_3 \cdot Q_{03} + b_3 \cdot Q_{04} + c_3 \cdot Q_{13} + d_3 \cdot Q_{14}}{2^2} \quad (3)$$

In the example used, only one point has been used as a constraint. However, a real case usually have a points set, for example a path with  $n$  points. This



**Fig. 3** Vertices involved in obtaining the turbulence of three noise levels at point  $P_1$ .

fact makes it possible to have the equation system 4 composed by the equation that obtains every point in the trajectory.

$$\left\{ \begin{array}{l} C\_height\_P_1 = \sum_i \frac{Noise(P_1 \cdot 2^i)}{2^i} \\ \cdot \\ \cdot \\ \cdot \\ C\_height\_P_n = \sum_i \frac{Noise(P_n \cdot 2^i)}{2^i} \end{array} \right. \quad (4)$$

This example clearly shows that there is an unbalanced number of unknown-known terms involved in the equation system:  $n$  known points on the trajectory versus  $(2^{m-1} + 1) \times (2^{m-1} + 1)$  vertices that need to be found, being  $m = 3$  in this case. In order to balance this difference without adding more unknown values to the equations system, more restriction points have to be added in the trajectory of existing points. To obtain these points in a coherent way, these new values have been obtained by making a curve equation with the points

from which we start. This allows to obtain with great precision additional points that do not change the trajectory of the points set to represent and allow to solve the system. The additional points number added depends on the unbalanced numbers of unknown-known terms. Therefore, points are added between the pairs of nearby points of each set until the number of equations is equal or greater than the number of unknown values.

Points are added in such a way as to force all the ones that form the trajectory to be equidistant ( $d_{min}$ ). Having more points in the path with known values makes the equation system have more equations without increasing the number of values to obtain.

But it must be kept in mind that the new added equations can be equivalent to other equations of the system, that is, with linearly dependent results that have an infinite set of solutions. This happens when the new added point is placed too close to the existing ones. To avoid this problem, the point is discarded and only valid turbulence equations are added to the equations system. For the new valid points, as has been done with the constraints points, its equation from its height value and its turbulence are obtained.

As a result of the resolving process, the values of the Q vertices that directly affect the constraint points are obtained. However, the process still leaves some Q vertices without value. This is due to the fact that they have not been used to obtain the value of the constraints. To finish the process, like the conventional Perlin method, is assigned a random value to these vertices without value. In this way, the Perlin turbulence matrix thus obtained contains the constraints values given initially, but keeping the randomness in the parts that do not affect to these constraints. The pseudocode of the implementation of all this process is shown in Algorithm 1.

#### 4 Large terrain models

The number of noise levels involved in obtaining the turbulence limits the size of the grid. If  $m$  levels are considered, the resolution of the obtained grid will be  $((2^{m-1} + 1) \times (2^{m-1} + 1))$ . According to this, the resolution of the grid in the previous example, where 3 levels of noise were considered, is  $5 \times 5$  nodes. The number of points of the final terrain depends on the resolution of the grid, so the larger a terrain is, the more noise levels have to be considered.

Some considerations have to be taken into account when large terrains are going to be generated. On the one hand, the size of the terrain determines the distance between the nodes on the grid that represent it. If this distance is too large, all the constraints that fall below this minimum distance will be ignored, thus losing precision in the result. To avoid this loss of information, grids with a great number of points would have to be used in large areas, which would require using a high number of noise levels. Keep in mind that the equations size obtained by each constraint increases proportionally with each calculated noise level. So the computational cost of solving the equations system does too.

---

**Algorithm 1** Process to obtain the terrain with constraints.

---

```

//Obtain all turbulence equations for the system.
eq_n = 0
for all constraints do
    eq_n = eq_n + 1
    vector_equations(eq_n) = compute_equation(turbulence, C_height)
end for

//If there are more unknowns points than equations add new
//points and obtain its equation
for eq_n less unknown_number do
    eq_n = eq_n + 1
    equation(eq_n) = compute_equation(turbulence, C_height_new_point)
end for

Solve equation system(vector_equations)

//Get the value of nodes that do not influence the constraints
for all Q_nodes do
    if Q_node no value then
        Q_node = rand_value[0 - 1]
    end if
end for

```

---

Taking these facts into account, large terrain are then represented by dividing large areas into smaller tiles, so that the resolution of them falls within the optimal result. Once the map has been divided, the method solves each tile independently. It must be taken into account that when they are joined again, the values of the vertices at the edges must be the same. Due to this fact, the method solves every one by establishing these values of the vertices at the edges as new constraints, because they have to match with the corresponding vertices of the adjacent tiles. This ensures that the height of the edges is equal in neighbouring tiles.

## 5 Results and Discussion

In order to test the method presented here, several experiments were conducted using a computer with an Intel Core i7-6700k 4.00 GHz processor. As already mentioned, the method uses restrictions formed by a set of points. These points can form routes (synthetics or georeferenced), areas such as lakes or concrete forms, further this points can be used to set heights of interest, such as elevations and low ground areas. To test the results obtained, the three types of data sets mentioned have been used. Their characteristics have been detailed in subsection 5.1. During the experimentation, the MATLAB function LU factorization has been used to solve the system of linear equations, because of its computational efficiency.

**Table 1** Detail of the data used to test the presented method.

Route	Number of points	Normalized height (m)
Simple	43	13
Snake	192	50
First route	443	65
Second route	668	122
Third route	421	44
Fourth route	208	36.5
Lambda	12,154	20
Flower	23,217	40

### 5.1 Details of the used constraints

Different points sets have been used to test the results of the method. They can be classified in two groups according the terrain that is going to represent: simple or large terrains. Table 1 describes the number of points of each route and the normalized height, that is, the difference in meters between minimum and maximum height of terrain.

Regarding the first group, two different points sets have been considered: hand-made and special figures. Starting from the hand-made paths, two routes have been designed to test the method: one of them called *Snake*, because its form (Figure 4(a)), and other one composed by a low number of points, that has been called *Simple* to identify it (Figure 4(d)).

Some works related to the topic use synthetic forms that represent special figures to test the results. So, two of them have been used to comprove that this method can get this constraints kind. Both are black and white figures, which the points that form the figure have been extracted, so that the method uses them as restrictions. They have been obtained from two research works related to the topic: the first one, called *Flower*, from the work presented by Yoon and Lee [56] and the second one, called *Lambda*, from the work from Zhou et al. [58]. All the points sets in this group have been represented immersed in terrains with a resolution of  $256 \times 256$ .

The problem to be solved by this work asked to be able to use GPS routes as a restriction. Therefore, in this second group of point sets, four real routes extracted from Wikiloc [54] have been used, a mashup where geo-referenced outdoor routes and points of interest from around the world are stored and shared. This routes represent real routes and areas in the province of Castellón (Spain). They are more complex paths, composed by a higher number of points, with some interest points and areas. Table 2 completes the information about these routes. The considered resolution of the grid that represent the terrain has been  $1024 \times 1024$ , so the process of dividing the grid into 16 tiles has been performed to represent them.

**Table 2** Details of the interest points and areas associated with the real paths.

Route	Interest points	Area to be included
First route	0	No
Second route	0	No
Third route	9	No
Fourth route	12	Circle: 38 points

## 5.2 Visual results

The visual results obtained by applying the presented method to the different routes are shown in the following figures. Figure 4 illustrates the results for the simplest paths and the synthetic figures and finally Figure 5 for the real paths. All of them have been distributed in three columns. The first column shows a bird's eye view of the path or the set points. The second one shows the turbulence generated for the corresponding set of points, where its location has been marked when it was not clear. Finally, the third column shows the terrain representation associated to the turbulence generated. The set of points have also been highlighted in these images.

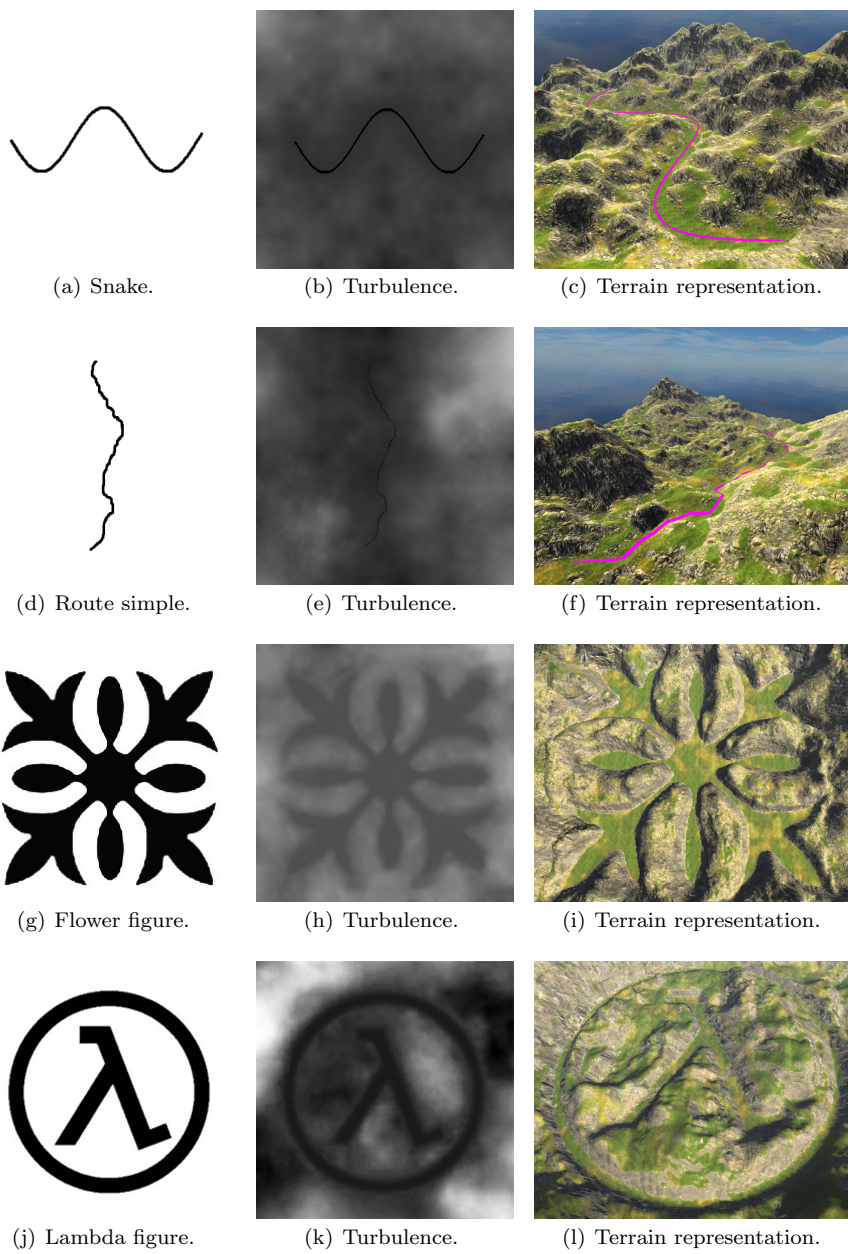
## 5.3 Efficiency and accuracy

The study was carried out by analyzing aspects related to efficiency, precision and realism. The experimental results have been divided according to the classification of the paths, and they can be seen in Table 3 and Table 4.

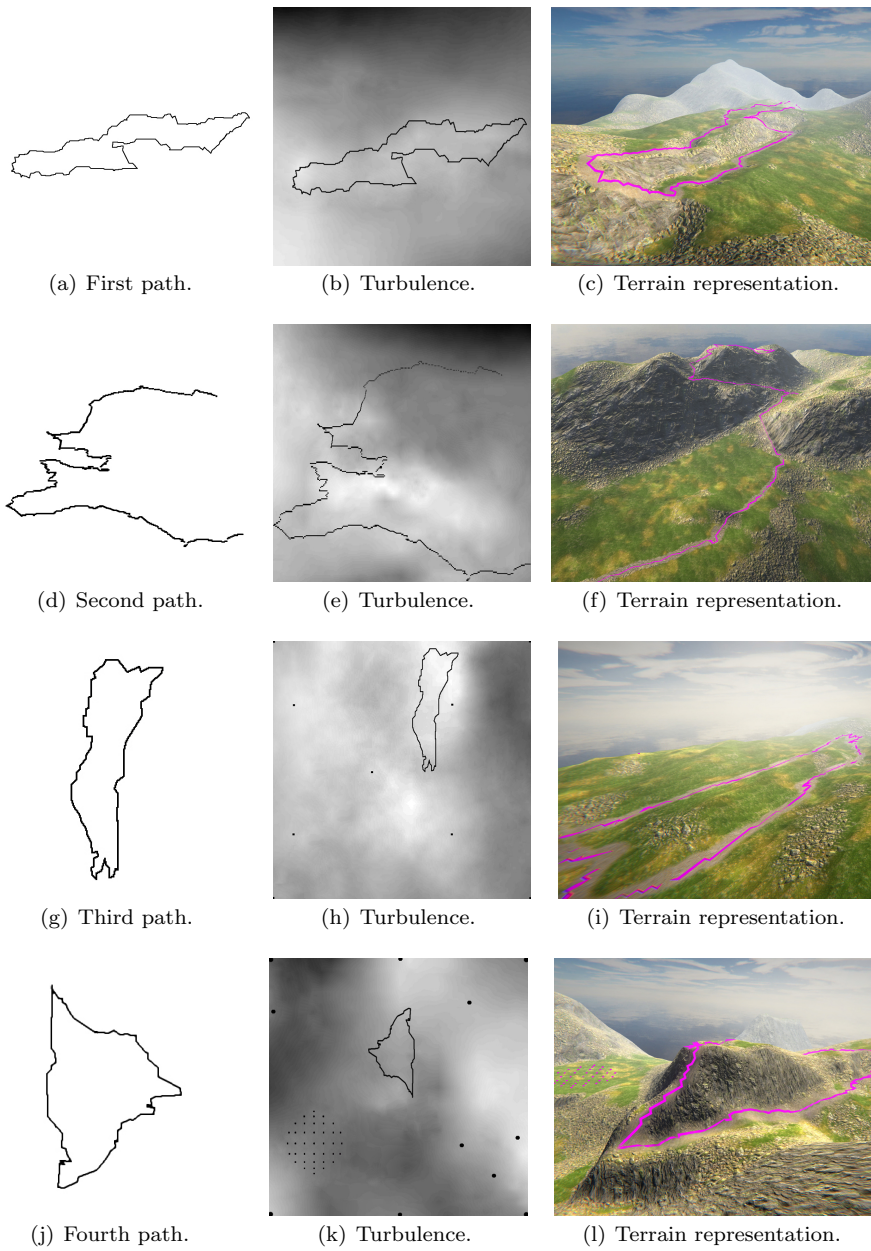
Regarding efficiency, every terrain was obtained computing grids of  $256 \times 256$ . Only the computation of the terrain that contain the real paths have required to generate tiles, because the resolution of the final terrain was  $1024 \times 1024$ . Then, the seconds required to compute the tiles only appear in Table 4 for these routes.

The grids of  $256 \times 256$  were obtained with three different numbers of noise levels (6, 7 and 8), which yield different levels of grid resolution ( $32 \times 32$ ,  $64 \times 64$  and  $128 \times 128$ ). The number of equations was the same for all the levels in every experiment. This implies that the points used as constraints in the edges are more separated in the lower resolutions. Finally, the tables show the total time spent on calculating the terrain (*Total Cost*).

To analyze the introduced error of the method, the values of the obtained height map were compared to the original values of the route, with an threshold of 0.025%. The final value for the error is obtained from all the points on the route with values that match the map obtained or which are within the permitted error. This accuracy is also shown in the tables.



**Fig. 4** Visual results obtained for the simplest paths and the synthetic figures.



**Fig. 5** Visual results obtained for the real paths.



**Table 3** Time cost (seconds) and error introduced (%) when generating terrains that include the hand-made routes and the special figures.

Route	Number Equations	Noise Levels	Total Cost (s)	Error (%)
Snake	192	6	0.12	16.46
		7	0.21	0
		8	0.54	0
Route simple	83	6	0.07	25.28
		7	0.15	0
		8	0.46	0
Flower	14,895	6	15.37	24.29
		7	29.92	0.37
		8	61.21	0.14
Lambda	12,158	6	11.76	28.52
		7	28.03	0.11
		8	58.16	0

**Table 4** Partial and total cost (seconds) and the error introduced (%) generating large terrains that contain the real paths.

Route	Number Equations	Noise Levels	Tile Cost (s)	Total Cost (s)	Error (%)
First	1979	6	1.01	16.82	12.92
		7	4.26	69.02	8.83
		8	10.80	174.91	5.37
Second	2204	6	1.79	29.20	11.55
		7	5.21	84.95	7.87
		8	11.91	193.28	3.09
Third	1966	6	0.81	12.99	5.9
		7	4.03	65.43	2.77
		8	10.64	172.12	0.58
Fourth	1794	6	0.23	5.01	6.29
		7	3.31	53.72	3.08
		8	9.87	161.02	0.21

#### 5.4 Realism

The Unity 3D game engine [51] was used to see the final representation of the height maps obtained by the method. Figure 6 shows the results obtained for the second real path. The terrain has been textured to improve the realism and the data set used as constraints in the experiments are marked in pink in the figures. These marks show that the generated terrain is adapted to the established constraints.



**Fig. 6** Close view of the resulting terrain that includes the Second real path.

### 5.5 Different iterations over the same paths

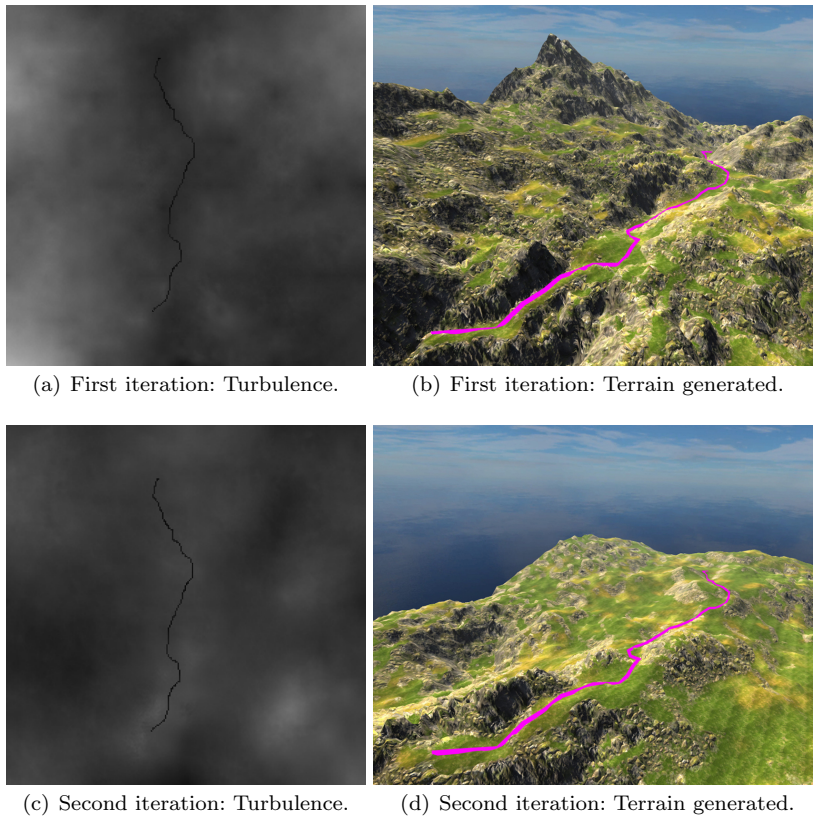
Apart from the data included in the terrain, a large part of the data are randomly obtained according to the traditional noise function. However, applying different iterations to the same path does not return the same terrains. This is because the method forces the data to be included in the terrain to have an accuracy, but as the distance of the points of the grid to the path increases, these constraints do not have any influence on the points and they are completely random.

An example of this situation is shown in Figure 7. The path used in this experiment has been the one called *Simple*. Two iterations of the method have been performed with this path and the terrains generated have been different in every one.

### 5.6 Validating the presented method

Noise-based procedural terrain generators [50] do not allow the use of precise constraints but often have some attributes that their modification allows to alter the obtained result. The most common attributes to vary are the modification of the octaves, modifying the level of detail of the result, the frequency, which allows varying the roughness of the terrain and the persistence, which modifies the height of the crests and the wave valleys. In order to analyze the method presented and to validate that in addition to meeting the proposed objectives it also allows the variations specified by these other methods without losing precision in the restrictions, it has been tested by modifying these attributes. Figure 8 show the results obtained when modifying the attributes. The path used to test this validation has been the one called *Simple*.

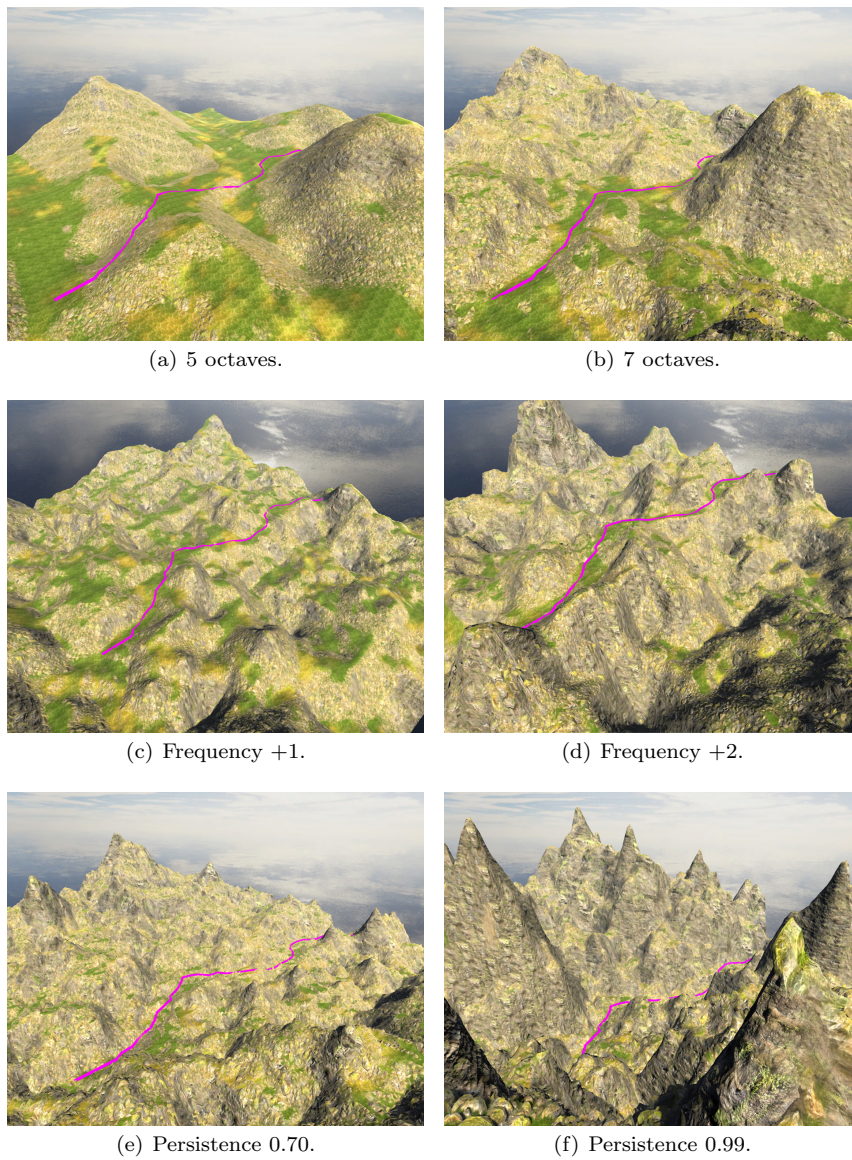
First of all, the number of octaves has been changed to obtain different results of the obtained turbulence. Figure 8 shows the terrains generated by



**Fig. 7** Results obtained after performing different iterations of the method taking the path called *Simple* as input data.

establishing the number of octaves to 5 and 7. It can be appreciated in the images that the level of detail of the terrain increases at the same time this attribute does. Next, the frequency has been modified. This attribute varies the distance between the points in a level of noise, so, as can be appreciated in the Figure 8, the higher the frequency is, the more abrupt the resulting terrain results.

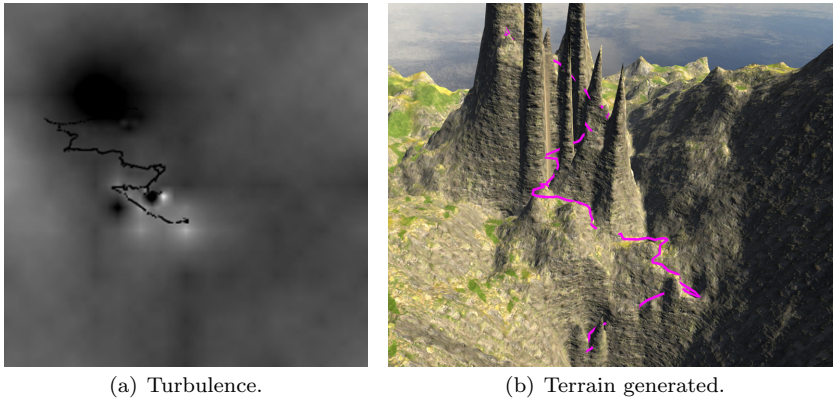
Finally, the persistence factor indicates the weight of every level of noise in the turbulence. As the level of noise increases, its weight is reduced to the half. Increasing the value of the persistence, the final result will have higher roughness because the higher levels of noise will have more influence in the final turbulence. Figure 8 shows the results obtained by varying the persistence value from 0.70 to 0.99.



**Fig. 8** Results obtained modifying the number of octaves, the frequency and the persistence.

### 5.7 Cases of failure

The method needs that the set of points that is going to be included in the final terrain meets some conditions to obtain a good result. If the points that form it are too closely situated or consecutive points represent really different



**Fig. 9** Example of a case of failure.

locations, the terrain obtained will be not valid. An example of this situation can be observed in Figure 9.

To avoid this situation, the routes have to be pre-processed to ensure a good distribution of the point data and a coherent sequence of point locations. Then, a good result can be obtained by applying this terrain generation method.

## 5.8 Discussion

As shown in the results, the method presented here is capable of generating large terrains that can be adapted to certain features using procedural modelling. This procedural modelling includes some previously fixed data that represent certain constraints and adds to them the random data that characterize this modelling method. The time spent on obtaining the turbulence map that represents the maps is not high taking into account that this process is not required to be performed in real time: in fact it took 3 minutes in the case of generating  $1024 \times 1024$  maps formed by tiles with a resolution of  $256 \times 256$ . This makes the method suitable for quick generation and later visualization in appropriate real-time applications such as game engines or simulators.

Regarding the accuracy of the data to be included in the final approximation, the generated terrain includes the routes with a precision of more than 85%, in most cases being above 90%. If the accuracy does not reach 100%, except in the case of accepting a major error, it is due to the final joining of the tiles. Sometimes, a small deviation takes place at some of the join points. This is not appreciable to the naked eye, but it affects the value of the accuracy.

The realism of the terrains can be observed in the Figure 6, where close views can be appreciated. There, it is shown the good quality of the results. Moreover, different executions of the method give as result to different terrains, as it has been shown in Figure 7. This is one of the advantages of being based

on the procedural modelling, the possibility of obtaining different terrains that fit the constraints established by the user.

## 6 Conclusions

This work presents a new procedural modelling method to generate terrain that allows to combine in a same representation some given data with other randomly generated. Some point set constraints are taken as input, such as GPS routes, figures or points situated in interest zones, and the method produces a random terrain that includes these constraints with a high degree of precision. The random part of the terrain is obtained by solving an equation system based on a noise function, which generates a procedural texture that is used as a height map. The noise generator method has been the presented by Perlin. It traditionally allows for variations in the final result but not the inclusion of precise data.

Results show that the terrains generated include the constraints in their final representation with a high level of realism, and ensuring a smooth continuity between the input data and the randomly generated heights. Moreover, the method maintains the random nature of the Perlin noise, because different terrains result from every run of the method. All the process is performed, as results support, with a low computational cost.

In the tests, it has been demonstrated that our method is able to produce large terrains. In these cases, terrains are divided into tiles, each of them is obtained separately, and they are joined at the end. Visual representations of the generated turbulence maps show the good quality of the results that include the given routes and the interest points, without the appearance of jumps in the terrain or changes that are too sudden.

As future work, the line of research that we are working on is addressed to the procedural generation of vegetation, in order to improve the realism of the terrains that are generated. Two main lines are being studied: on the one hand, the procedural placement of the plants, according to their botanical characteristics and the features of the generated terrain and, on the other hand, the procedural representation of the life of the plants based on inequations. Plants will be located on the terrain and in every iteration of the procedural system different representations of them will be shown depending on the time they have been growing.

## Acknowledgments

This work was supported by the Spanish Ministry of Science and Technology (Project TIN2016-75866-C3-1-R) and the Universitat Jaume I research project (UJI-B2018-56).

## References

1. Belhadj F. Terrain modeling: a constrained fractal model. In AFRIGRAPH 07: Proceedings of the 5 Conference on Computer Graphics, Virtual Reality, Visualisation and Interaction in Africa 2007; 197–204.
2. Bernhardt A, Máximo A, Velho L, Hnaidi H, Cani MP. Real-time terrain modeling using cpu-gpu coupled computation. In SIBGRAPI 11: Proceedings of the 24th Conference on Graphics, Patterns and Images 2011. 64–71.
3. Bradbury G.A, Amati C, Mitchell K, Weyrich T. Frequency-Based Creation and Editing of Virtual Terrain. Proceedings of the 11th European Conference on Visual Media Production, London, UK, 1314 November (2014). ACM: New York, NY, USA,
4. Bruneton E, Neyret F. Real-time rendering and editing of vector-based terrains. Computer Graphics Forum, Wiley, 2008, Special Issue: Eurographics 08, 27 (2), pp.311-320.
5. Cordonnier G, Galin E, Gain J, Benes B, Guerin E, Peytavie A, Cani M.P. Authoring landscapes by combining ecosystem and terrain erosion simulation. ACM Trans. Graph 2017. 36 (4), 12 pages.
6. De Carpentier G.J, Bidarra R. Interactive gpu based procedural heightfield brushes. In FDG 09: Proceedings th International Conference on the Foundations of Digital of the 4 Games 2009.
7. Dey R, Doig J.G, Gatzidis C. Procedural feature generation for volumetric terrains. SIGGRAPH 17 Posters, July 30 - August 03, 2017, Los Angeles, CA, USA.
8. Doran J, Parberry I. Controlled procedural terrain generation using software agents. IEEE Transactions on Computational Intelligence and AI in Games 2 2010. 2, 111–119.
9. Emilien A, Bernhardt A, Peytavie A, Cani M.P, Galin E. Procedural generation of villages on arbitrary terrains. Visual Computer 28(68) 2012, 809-818.
10. Fournier A, Fussell D, Carpenter L. Computer rendering of stochastic models. Communications of the ACM 25 (6) 1982: 371–384.
11. Gasch C, Chover M, Remolar I, Rebollo C. Procedural Modeling of Terrain from GPS Routes. XXVI Spanish Computer Graphics Conference 2016.
12. Gain J, Marais P, Strasser W. Terrain sketching. In I3D 09: Proceedings of the Symposium on Interactive 3D Graphics and Games 2009. 31–38.
13. Gain J, Merry B, Marais P. Parallel, realistic and controllable terrain synthesis. Comp. Graph. Forum 2015. 34, 2, 105116.
14. G enevaux J, Galin E, Gurin E, Peytavie A, Benes B. Terrain Generation using Procedural Models based on Hydrology. ACM Transactions on Graphics. Proceedings of SIGGRAPH 2013.32(4):13.
15. G enevaux J, Galin E, Peytavie A, Gurin E, Briquet C, Grosbellet F, Benes B. Terrain Modelling from Feature Primitives. Computer Graphics Forum 2015. 34(6): 198-210.
16. Gu erin E, Digne J, Galin E, Peytavie A. Sparse representation of terrains for procedural modeling. Computer Graphics Forum 2016. 35(2): 177-187.
17. Hnaidi H, Gu erin E, Akkouche S, Peytavie A, Galin E. Feature based terrain generation using diffusion equation. In Computer Graphics Forum: Proceedings of Pacific Graphics 2010. 29: 2179–2186.
18. Hou, F, Qin, H, Qi, Y. Procedure-based component and architecture modeling from a single image. Vis. Comput 2016. 32(2), 151-166.
19. Kelley A, Malin M, Nielson G. Terrain Simulation Using a Model of Stream Erosion. Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques 1988. 263–268.
20. Kristof P, Benes B, Kivanek J, Stava O. Hydraulic erosion using smoothed particle hydrodynamics. *Computer Graphics Forum* 2009. 28 (2).
21. Kamal K.R, Uddin Y.S. Parametrically controlled terrain generation. In GRAPHITE 07: Proceedings of the 5 International Conference on Computer Graphics and Interactive Techniques in Australia and Southeast Asia 2007. 17–23.
22. Bundy Soft. Available online: <http://http://www.bundysoft.com/L3DT/>. [Accessed on 30 may 2018].
23. Li W, Han D, Li H, Wang X, Zhu J. Extraction of digital terrain model based on regular mesh generation in mountainous areas. Multimedia Tools and Applications 2018. 77:62676286.

24. Mandelbrot B.B. *The fractal geometry of nature*. W. H. Freeman, New York 1983.
25. Mei X, Decaudin P, Hu B.G. Fast Hydraulic Erosion Simulation and Visualization on GPU, Pacific Graphics, IEEE Computer Society, 2007. 47-56.
26. Miller G.S.P. The definition and rendering of terrain maps. In SIGGRAPH 86: Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques 1986. 20(4): 39-48.
27. Musgrave F.K, Kolb C.E, Mace R.S. The synthesis and rendering of eroded fractal terrains. In SIGGRAPH 89: Proceedings of the 16th Annual Conference on Computer Graphics and Interactive Techniques 1989. 19: 41-50.
28. Nagashima K. Computer generation of eroded valley and mountains terrains. *The Visual Computer* 13 1997: 456-464.
29. Neidhold, B, Wacker, M, Deussen, O. Interactive Physically based Fluid and Erosion Simulation. *Proceedings of Eurographics Workshop on Natural Phenomena 2005*, 1: 25-32.
30. Pajarola, R, Gobbetti, E. Survey of semi-regular multiresolution models for interactive terrain rendering. *Vis. Comput* 2007. 23(8), 583-605.
31. Parberry I. Designer Worlds: Procedural Generation of Infinite Terrain from Real-World Elevation Data. *Journal of Computer Graphics Techniques (JCGT)* 2014. 3(1):74-85.
32. Perlin K. An Image Synthesizer. In SIGGRAPH'85: Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques 1985. 19: 287-296.
33. Perlin K. Improving noise. In Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH) 2002. 681-682.
34. Peytavie A, Galin E, Grosjean J, Merillou S. Arches: a Framework for Modeling Complex Terrains. *Computer Graphics Forum (Proceedings of Eurographics)*, 2009. 28: 457-467.
35. Puig-Centelles A, Varley P.A.C, Ripollés O, Chover M. Automatic Terrain Generation with a Sketching Interface. *Proceedings of the 17th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG 09)*. 2009.
36. Puig-Centelles A, Varley P.A.C, Ripollés O, Chover M. Automatic Terrain Generation with a Sketching Tool. *Multimedia Tools and Applications* 2014. 70: 1957-1986.
37. Puig-Centelles A, Ripollés O, Chover M. Creation and Control of Rain in Virtual Environments. *The Visual Computer* 2009. 25(11) :1037-1052.
38. Prusinkiewicz P, Hammel M. A Fractal Model of Mountains with Rivers. In *Proceedings of Graphics Interface'93* 1993. 174-180.
39. Ramos F, Chover M, Ripollés O, Granell C. Continuous Level of Detail on Graphics Hardware. In *Proceedings of the 13th International Conference on Discrete Geometry for Computer Imagery*. 460-469 (2006)
40. Rebollo C, Remolar I, Chover M, Ramos J.F. A Comparison of Multiresolution Modelling in Real-Time Terrain Visualisation. *Springer-Verlag Berlin Heidelberg* 2004. 3044: 703-712.
41. Rebollo C, Remolar I, Gumbau J, Chover M. Three-dimensional trees for virtual globes. *International Journal of Digital Earth* 2014. 789-810.
42. Ripollés O, Ramos J.F, Puig-Centelles A, Chover M. Real-time tessellation of terrain on graphics hardware. *Computers & Geosciences* 2012. 147-155.
43. Rusnell B, Mould D, Eramian M. Feature-rich distance-based terrain synthesis. *The Visual Computer* 2009 25: 573-579.
44. Scheneider, J, Boldte, T, Westermann, R. Real-Time Editing, Synthesis, and Rendering of Infinite Landscapes on GPUs. In *Vision, Modeling and Visualization* 2006. 145-152.
45. Smelik R, Tutenel T, Bidarra R, Benes B. A survey on procedural modeling for virtual worlds. *Computer Graphics Forum* 2014. 33(6): 31-50.
46. Smelik R, Tutenel T, De Kraker K.J, Bidarra R. Interactive creation of virtual worlds using procedural sketching. In *Proceedings of Eurographics 2010: Short Papers* 2010.
47. Stachniak S, Strzlinger W. An algorithm for automated fractal terrain deformation. *Computer Graphics and Artificial Intelligence* 2005. 1: 64-76.
48. Stava O, Benes B, Brisbin M, Krivanek J. Interactive Terrain Modeling Using Hydraulic Erosion. In *Eurographics/SIGGRAPH Symposium on Computer Animation* 2008. 201-210.
49. Tasse F.P, Gain J, Marais P. Enhanced texture-based terrain synthesis on graphics hardware. *Computer Graphics Forum* 2012. 31, 6. 19591972.
50. PlanetSide Software. Available online: <https://planetSide.co.uk/>. [Accessed on 28 June 2018].



- 
51. Unity 3D. Available online: <https://unity3d.com/>. [Accessed on 2 June 2018].
  52. Vanek J, Benes B, Herout A, Stava O. Large-scale physics-based terrain editing using adaptive tiles on the GPU. *Computer Graphics and Applications*, IEEE 31, 6 (Nov 2011), 3544.
  53. E-on Software. Available online: <https://info.e-onsoftware.com/more-info-vue>. [Accessed on 18 June 2018].
  54. Wikiloc. Routes of the world. Available online: <https://es.wikiloc.com>. [Accessed on 28 June 2018].
  55. WM WorldMchine Software. Available online: <https://planetside.co.uk/>. [Accessed on 18 July 2018].
  56. Yoon J.C. Lee IK. Stable and controllable noise. *Graphical Models* 70, 5. 2008. 105-115.
  57. Zhang J, Wang C, Qin H, Chen Y, Gao Y. Procedural modeling of rivers from single image toward natural scene production. *The Visual Computer* 2017 <https://doi.org/10.1007/s00371-017-1465-7>.
  58. Zhou H, Sun J, Turk G, Rehg J.M. Terrain synthesis from digital elevation models. *IEEE Transactions on Visualization and Computer Graphics* 2007. 13(4): 834-848.