Hindawi Publishing Corporation
International Journal of Computer Games Technology
Volume 2009, Article ID 125714, 13 pages
doi:10.1155/2009/125714

Research Article

Breeding Terrains with Genetic Terrain Programming: The Evolution of Terrain Generators

Miguel Frade,[1] F. Fernandez de Vega,[2] and Carlos Cotta[3]

[1] *Escola Superior de Tecnologia e Gestão (ESTG), Instituto Politécnico de Leiria, Alto do Vieiro, 2411-901 Leiria, Portugal*
[2] *Centro Universitario de Mérida, Universidad de Extremadura, C/Sta Teresa de Jornet 38, 06800 Mérida, Spain*
[3] *ETSI Informática (3.2.49), Universidad de Málaga, Campus de Teatinos, 29071 Málaga, Spain*

Correspondence should be addressed to Miguel Frade, mfrade@estg.ipleiria.pt

Received 14 June 2008; Accepted 24 November 2008

Recommended by Abdennour El Rhalibi

Although a number of terrain generation techniques have been proposed during the last few years, all of them have some key constraints. Modelling techniques depend highly upon designer's skills, time, and effort to obtain acceptable results, and cannot be used to automatically generate terrains. The simpler methods allow only a narrow variety of terrain types and offer little control on the outcome terrain. The Genetic Terrain Programming technique, based on evolutionary design with Genetic Programming, allows designers to evolve terrains according to their aesthetic feelings or desired features. This technique evolves Terrain Programmes (TPs) that are capable of generating a family of terrains—different terrains that consistently present the same morphological characteristics. This paper presents a study about the persistence of morphological characteristics of terrains generated with different resolutions by a given TP. Results show that it is possible to use low resolutions during the evolutionary phase without compromising the outcome, and that terrain macrofeatures are scale invariant.

1. Introduction

Artificial terrain generation techniques are an important facet of graphical applications that attempt to represent a real or an imaginary world. Among those applications are computer animation, architecture, virtual reality, and video games (for a more extensive list of examples check the Virtual Terrain Project, http://www.vterrain.org/Misc/Why.html). On the virtual terrain field, much of the research has been focused on how to accelerate the visualisation of large terrains to achieve interactive frame rates. As a result, many level of detail (LOD) algorithms have been developed [1–3], which increase rendering speeds by using simpler versions of the geometry for objects that have lesser visual importance such as those far away from the viewer [4]. This approach have been successfully applied on many video games, a field where artificial terrain generation techniques are probably more prominent.

On the other hand, terrain generation has received less attention in the literature. Fractal-based techniques are still the most prevalent, specially on video games, in spite of the several generation techniques existing today (see Section 2). This happens because of their speed, ease of implementation, and ability to create irregular shapes across an entire range of LODs. Nevertheless, these techniques allow only a confined variety of terrain types [2, 5], little control on the outcome, and are only focused on the generation of realistic terrains. Although this is important, it might prevent designers from achieving their goals when they attempt to represent an alien or an exotic looking terrain. Fractal-based techniques do not allow designers to express their full creativity or to evolve a terrain accordingly to their aesthetic feelings rather than realism. The terrain novelty might have a positive impact on a product's target audience and increase their interest. The Genetic Terrain Programming (GTP) technique [6] allows the evolution of Terrain Programmes (TPs) based on aesthetic evolutionary design with Genetic Programming (GP). For a specific resolution, it is known the ability of those TPs to generate a family of terrains—different terrains, but with coherent morphological features. This paper presents a set of experiments to study the perseverance of terrain morphological features across different resolutions. This is

a desired characteristic by video games' designers, as it will enable them to adapt the terrain to the required processing power, without recurring to additional algorithms. This property will also help to improve performance during the TPs' evolutionary phase.

Section 2 introduces some background about the traditional terrain generation techniques and their main constrains. It also presents an overview of evolutionary systems applied to terrain generation. Section 3 describes the GTP technique, the developed tool, and the achieved results. Finally, the conclusions and future work are presented on Section 4.

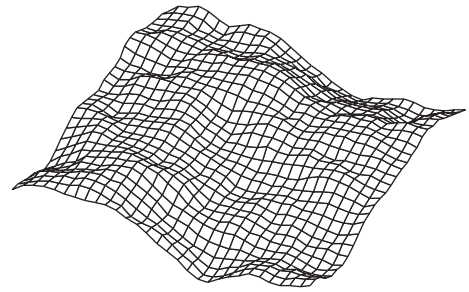


Figure 1: A discrete height map example.

2. Background

Artificial terrain generation has been addressed by several researchers for a long time, and therefore many techniques and algorithms have been developed. To establish a base line of comparison with real algorithms, Saunders [7] proposed the following list of traits that an ideal terrain generation algorithm should have

(i) low requirements of human input,

(ii) allow a high degree of human control,

(iii) to be completely intuitive to control,

(iv) produce models at arbitrary levels of detail,

(v) fast enough for real-time applications,

(vi) to be able to generate a wide variety of recognisable terrain types and features,

(vii) to be extensible to support new types of terrain.

Some of the listed characteristics are in tension with one another, such as low requirements of human input and high degree of human control. This means that all terrain generation techniques had to choose priorities and made some compromises regarding their traits. Another important attribute of a terrain generation technique is how it represents a terrain. The chosen data structure will influence the way the terrain is built, the available tools to manipulate it, and might affect also the terrains features that can be represented. Height maps are probably the most common method used to represent terrains, although other data structures exist. Formally, a height map is a scalar function of two variables, such that for every coordinate pair (x, y) corresponds an elevation value h, as shown in (1). In practice, a height map is a two-dimensional rectangular grid of height values, where the axis values are spaced with regular intervals valid over a finite domain (see Figure 1). The most common data structure to represent them is 2D arrays filled with the elevations values:

$$h = f(x, y). \tag{1}$$

The regular structure of height maps is their main advantage, since it allows the optimisation of operations such as rendering, collision detection, and path finding. The render of huge height maps in real time is now possible due to the creation of several continuous level of detail (CLOD) algorithms [1–3], which render highly visible areas of the terrain with detailed geometry, using progressively simpler geometry for more distant parts of the terrain. Collision detection is greatly simplified if one of the objects is a height map, because only a few surrounding triangles need to be checked for collision. A second advantage is the fact that height maps are compatible with grey-scale images (if the heights values are normalised). This means that image processing and computer vision techniques may be used to construct, modify, and analyse terrain models represented as height maps. For example, a height map can be stored, imported, or exported using an image file format, or a filter can be applied to smooth a rough terrain. Finally, Geographic Information Systems (GIS) use height maps to represent real-world terrain, which are commonly built using remote sensing techniques such as satellite imagery and land surveys. This is another advantage due to the significant amount of real-world terrain models available to work with.

The main limitation of height maps is the inability to represent structures where multiple heights exist for the same pair of coordinates. So, height maps are inherently unable to represent caves, overhangs, vertical surfaces, and other terrain structures in which multiple surfaces have the same horizontal coordinates. Fortunately, only a small percentage of natural terrain fall into this category, and this limitation can be overcome by using separate objects placed on top of the terrain model. A second disadvantage of height maps is that it has a finite uniform resolution, which means there is no simple way to handle a terrain with different local levels of details. If the resolution is chosen to match the average scale of the features in the terrain, then any finer-scale features will be simplified or eliminated. Conversely, if the resolution is chosen to be high enough to capture the fine-scale features, areas containing only coarse features will also be captured at this same high resolution, an undesirable waste of space and processing time. Ideally, a terrain representation for terrain generation would either be infinite in resolution, or else would adaptively increase its resolution to accommodate the addition of fine-scale details, rather than requiring a prior decision about resolution. A third disadvantage of height maps is its inadequacy to represent terrain on a planetary scale. Rectangular height maps do not map directly to spheroid objects; usually a two-pole spherical projection is used, and in those cases the density of height field points will be substantially greater in areas near the poles than at

those near the equator. For the purpose of our technique, the advantages of height maps outcome their shortcomings, though.

2.1. Traditional Terrain Generation Techniques. Traditional techniques for terrain generation can be categorised into three main groups: (1) measuring, (2) modelling, and (3) procedural. Next, we briefly review each of these techniques.

(1) Measuring techniques gather elevation data through real-world measurements, producing digital elevation models (http://rockyweb.cr.usgs.gov/nmpstds/demstds.html). These models are commonly built using remote sensing techniques such as satellite imagery and land surveys. One key advantage of measuring techniques lies in the fact that they produce highly realistic terrains with minimal human effort, although this comes at the expenses of the designer control. In fact, if the designer wants to express specific goals for the terrain's design and features, this approach may be very time-consuming since the designer may have to search extensively for real-world data that meet her targeted criteria.

(2) Modelling is by far the most flexible technique for terrain generation. A human artist models or sculpts the terrain morphology manually using a 3D modelling programme (e.g., Maya, 3D Studio (http://www.autodesk.com/fo-products), or Blender (http://www.blender.org)), or a specialised terrain editor programme (e.g., the editors that ship with video games like Unreal Tournament 2004 (http://www.mobygames.com/game/unreal-tournament-2004), SimCity 4 (http://simcity.ea.com/about/simcity4/overview.php), or SimEarth (http://www.mobygames.com/game/simearth-the-living-planet)). The way the terrain is built is different depending on the features provided by the chosen editor, but the general principle is the same. With this approach, the designer has unlimited control over the terrain design and features, but this might be also a disadvantage. By delegating most or all of the details up to the designer, this technique imposes high requirements on the designer in terms of time and effort. Also the realism of the resulting terrain is fully dependent on the designer's skills.

Finally, (3) procedural techniques are those in which the terrains are generated programmatically. This category can further be divided into physical, spectral synthesis, and fractal techniques.

Physically-based techniques simulate the real phenomena of terrain evolution trough effects of physical processes such as erosion by wind (http://www.weru.ksu.edu), water [8], thermal [9], or plate tectonics. These techniques generate highly realistic terrains, but require an in-depth knowledge of the physical laws to implement and use them effectively. Physically-based techniques are also very demanding in terms of processing power.

Another procedural approach is the spectral synthesis. This technique is based on the observation that fractional Brownian motion (fBm) noise has a well-defined power spectrum. So random frequency components can be easily calculated and then the inverse fast Fourier transform (FFT) can be computed to convert the frequency components into altitudes. The problem of using this technique for simulating real-world terrain is that it is statistically homogeneous and isotropic, two properties that real terrain does not share [9]. Furthermore, it does not allow much control on the outcome of terrains' features.

Self-similarity is the key concept behind any fractal technique. An object is said to be self-similar when magnified subsets of the object look like (or identical to) the whole and to each other [10]. This allows the use of fractals to generate terrain which still looks like terrain, regardless of the scale in which it is displayed [11]. Every time these algorithms are executed they generate a different terrain due to the incorporated randomness. This class of algorithms is the favourite one by game's designers, mainly due to their speed and simplicity of implementation. There are several tools available that are predominantly based on fractal algorithms, such as Terragen (which is a hybrid fractal/modelling tool) (http://www.planetside.co.uk/terragen) and GenSurf (a mapping tool for Quake 3 Arena video game) (http://tarot.telefragged.com/gensurf). However, generated terrains by this techniques are easily recognised because of the self-similarity characteristic of fractal algorithms. Besides, not all terrain types exhibit the self-similar property across all scales. For example, both photos from Figure 2 are from Death Valley (Calif, USA), but seen at very different scales. On Figure 2(a) is a close-up of cracked dried mud in a creek from the Death Valley and on Figure 2(b) is a satellite image of the same region. As is easily verified, in this case there is no self-similarity between the two scales of these terrain photos. Although these algorithms present some parameters that can be tweaked to control, for example, the roughness, the designer does not have control on the resulting terrain features.

2.2. Evolutionary Terrain Generation Techniques. Evolutionary algorithms (EAs) are a kind of bioinspired algorithms that apply Darwin's theory [12] of natural evolution of the species, where living organisms are rewarded through their continued survival and the propagation of its own genes to its successors. There are four main classes of EAs: genetic algorithms (GAs) [13], evolutionary strategies [14], GP [15], and evolutionary programming [16]. Evolutionary algorithms can be seen as search techniques [17]. They are able to achieve good solutions to many types of problems, thanks to their flexibility and adaptability to different search scenarios. This characteristic is the key factor of success in such diverse fields as engineering, art, biology, economics, marketing, genetics, operations research, robotics, social sciences, physics, and chemistry. Apart from their use as optimisers, evolutionary algorithms have also been used as an experimental framework to validate theories about biological evolution and natural selection, particularly through work in the field of artificial life [18].

Evolutionary design is a branch of evolutionary computation which has its roots in three different disciplines: computer science, evolutionary biology, and design. Evolutionary design has taken place in many different areas over the last decade. Designers have optimised selected parts of

(a) (b)

Figure 2: Images of Death Valley: (a) "cracked mud on the way to the borax haystacks," by *redteam*, Creative Commons license, (b) a satellite image from NASA (public domain). On this example, there is no self-similarity between the two scales of this region.

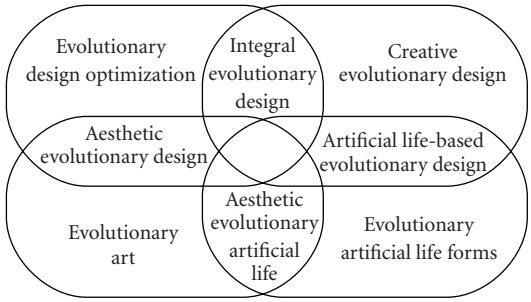


Figure 3: Evolutionary design categories.

their designs using evolution; artists have used evolution to generate aesthetically pleasing forms; architects have evolved new building plans from scratch; computer scientists have evolved morphologies and control systems of artificial life. Evolutionary design can be divided into four main categories [19]: evolutionary design optimisation, creative evolutionary design, evolutionary art, and evolutionary artificial life forms. However, some author's work may be included in two or more categories creating four overlapping subcategories shown in Figure 3.

Evolutionary art systems are similar in many ways. They all generate new forms or images from the ground up (random initial populations); they rely upon a human evaluator to set the fitness value of an individual based on subjective evaluation, such as aesthetic appeal; population sizes are very small to avoid user's fatigue and allow a quick evaluation, and user interfaces usually present a grid on the screen with the current population individuals, allowing the user to rank them. However, they differ on their phenotype representations [20].

GP has been the most fruitful evolutionary algorithm applied to evolve images interactively. Karl Sims used GP to create and evolve computer graphics by mathematical equations. The equations are used to calculate each pixel [21]. He created several graphic art pieces including *Panspermia* and *Primordial Dance,* and also allowed visitors to interact with his interactive art system at art shows and exhibitions. His *Galapagos* (http://www.karlsims.com/galapagos/index.html) is an L-system-based interactive evolutionary computation (IEC) system that allows visitors to create their own graphic art through their interaction.

Tatsuo Unemi developed Simulated Breeding ART (SBART) [22, 23], an IEC graphics system open to public. *SBART* uses GP to create mathematical equations for calculating each pixel value and its (x, y) coordinates. As GP nodes, *SBART* assigns the four arithmetic fundamental operators $(+, -, \times,$ and $\div)$, *power*, *sqrt*, sin, cos, log, exp, min, and max. The terminal nodes are constants and variables. Three values at each pixel are calculated using one generated mathematical equation by assuming that the constants are 3D vectors consisting of three real numbers, and the variables are a 3D tuple consisting of $(x, y, 0)$. The three calculated values are regarded as members of a vector (hue, lightness, and saturation), and are transformed to RGB values for each pixel. These three values are normalised to values in $[-1, 1]$ using a saw-like function. The *SBART*'s functions were expanded to create a collage [24]. A human user selects preferred 2D images from 20 displayed images at each generation, and the system creates the next 20 offspring. Sometimes exporting/importing parents among multiple *SBART* instances is allowed. This operation is iterated until the user obtains a satisfactory image.

In Neuro Evolutionary Art (NEvAr) [25], of Machado and Cardoso, the function set is composed mainly of simple functions such as arithmetic, trigonometric, and logic operations. The terminal set is composed of a set of variables x, y, and random constants. The phenotype (image) is generated by evaluating the genotype for each (x, y) pair belonging to the image. In order to produce colour images, NEvAr resorts to a special kind of terminal that returns a different value depending on the colour channel—red, green, or blue—that is being processed. This tool focuses on the reuse of useful individuals, which are stored in an image database and led to the development of automatic seeding procedures.

To the best of our knowledge, Ong et al. [26] were the first authors to propose an evolutionary approach to generate terrains. They proposed an evolutionary design optimisation technique to generate terrains by applying genetic algorithms to transform height maps in order to conform them to the required features. Their approach breaks down the terrain

generation process into two stages: the terrain silhouette generation phase, and the terrain height map generation phase. The input to the first phase is a rough 2D map laying out the geography of the desired terrain that can be randomly generated or specified by the designer. This map is processed by the first phase to remove any unnaturally straight edges and then fed to the second phase, along with a database of preselected height map samples representative of the different terrain types. The second phase searches for an optimal arrangement of elevation data from the database that approximates the map generated in the first phase. Since the height map generation algorithm is inherently random, the terrains generated from two separate runs of the algorithm will not be the same, even if they use the same map.

We proposed a new technique, based on aesthetic evolutionary design, designated GTP [6]. Our approach consists on the combination of interactive evolutionary art systems with GP to evolve mathematical expressions, designated TPs, to generate artificial terrains as height maps. GTP relies on GP as evolutionary algorithm, which creates mathematical expressions as solutions (further details are presented on Section 3).

2.3. Genetic Programming. Genetic programming (GP) is an evolutionary computation (EC) technique that automatically solves problems without requiring the user to know or specify the form or structure of the solution in advance. More precisely, GP is a systematic domain-independent method for getting computers to solve problems automatically starting from a high-level statement of what needs to be done. In GP, a population of computer programmes is evolved. Generation by generation, a population of programmes is stochastically transformed into new, hopefully better, populations of programmes [27]. Due to its heuristic nature, GP can never guarantee results. However, it has been used successfully in many areas, such as [17] artificial life, robots and autonomous agents, financial trading, neural networks, art, image and signal processing, prediction and classification, and optimisation.

Algorithm 1 shows the basic steps of GP. The generated programmes are run for evaluation (Line 3) and compared with some ideal. This comparison is quantified to give a numeric value called fitness. The best programmes are chosen to breed (Line 4) and produce new programmes for the next generation (Line 5). The primary genetic operators used to create new programmes from existing ones are the following:

(i) *crossover*: the creation of a child programme by combining randomly chosen parts from two selected parent programmes,

(ii) *mutation*: the creation of a new child programme by randomly altering a randomly chosen part of a selected parent programme.

In GP, programmes are usually expressed as trees rather than as lines of code. For example, Figure 4 shows the tree representation of the programme $\max(x + x, x + 3 * y)$. The variables and constants in the programme (x, y, and 3) are leaves of the tree, or terminals in GP terminology. The arithmetic operations ($+$, $*$, and max) are internal nodes called functions. The sets of allowed functions and terminals together form the primitive set of a GP system.

For those who wish to learn more about GP, the book *A Field Guide to Genetic Programming* from Poli et al. [27] has a very good introduction to GP. A thoroughly analysis on this topic is provided on the book *Genetic Programming—On the Programming of Computers by Means of Natural Selection* by Koza [15], the main proponent of GP who has pioneered the application of Genetic Programming in various complex optimisation and search problems.

3. Genetic Terrain Programming

Current terrain generation techniques have their own advantages and disadvantages, as detailed in Section 2. Notwithstanding the importance of real-looking terrains, none of the existing methods focused on generating terrains accordingly to designers' aesthetic appeal. The main goals of GTP are to address the weaknesses of existing methods, allowing also the generation of aesthetic terrains. Thus, providing a *better* way of generating virtual terrains for a broad range of applications, with a special emphasis on video games.

In light of the idealised terrain generator, the goals of GTP are (in order of decreasing importance) as follows:

(1) capable of generating diverse features and terrain types, both aesthetic and realistic,

(2) extensibility,

(3) intuitive to control,

(4) automated generation with arbitrary resolution,

(5) low requirements of human input.

To achieve these goals, we use aesthetic evolutionary design with GP, where the phenotypes are terrains represented as height maps. This approach consists of a guided evolution, through interactive evolution, according to a specific desired terrain feature or aesthetic appeal. The extensibility and ability to generate diverse features and terrain types are assured by the GP. The diversity of solutions is directly dependent on the GP terminal and function sets. So, the extensibility feature can be easily achieved by adding new functions and terminals. The designer will guide the terrains evolution, performing this way the control of the outcome, by selecting which ones he prefers for his specific goals. Consequently, the software tool will be easy and intuitive to use with low input requirements. The outcome of the interactive evolution will be TPs, which are mathematical expressions with incorporated randomness. Those TPs can be used, like a procedural technique, to automatically generate different terrains with different resolutions and the same consistent features.

3.1. Method. The initial population is created randomly, with trees depth size limited initially to 6 and a fixed population size of 12. The number of generations is decided by the designer, who can stop the algorithm at any time.

1: Randomly create an initial population of programs from the available primitives
2: **repeat**
3: Execute each program and ascertain its fitness
4: Select one or two program(s) from the population with a probability based on fitness to participate in genetic operations
5: Create new individual program(s) by applying genetic operations with specified probabilities
6: **until** an acceptable solution is found or some other stopping condition is met (e.g., a maximum number of generations is reached)
7: **return** the best-so-far individual

Algorithm 1: Genetic programming basic algorithm [27].

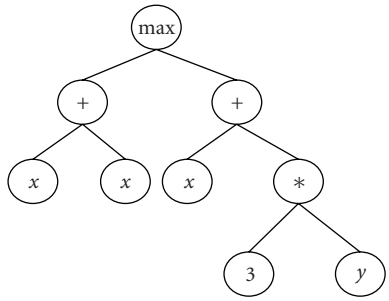


Figure 4: GP tree representation of $\max(x + x, x + 3 * y)$.

Table 1: GP functions.

Name	Description
plus(h_1, h_2) minus(h_1, h_2) multiply(h_1, h_2)	Arithmetical functions
sin(h) cos(h) tan(h) atan(h)	Trigonometric functions
myLog(h)	Returns 0 if $h = 0$ and $\log(abs(h))$ otherwise
myPower(h_1, h_2)	Returns 0 if $h_1^{h_2}$ is NaN or Inf, or has imaginary part, otherwise returns $h_1^{h_2}$
myDivide(h_1, h_2)	Returns h_1 if $h_2 = 0$ and $h_1 \div h_2$ otherwise
myMod(h_1, h_2)	Returns 0 if $h_2 = 0$ and $\text{mod}(h_1, h_2)$ otherwise
mySqrt(h)	Returns sqrt ($abs(h)$)
negative(h)	Returns $-h$
FFT(h)	2D discrete Fourier transform
smooth(h)	Circular averaging filter with $r = 5$
gradient $X(h)$ gradient $Y(h)$	Returns the gradient (dh/dx or dh/dy) of a height map h. Spacing between points is assumed to be 1

The designer can select one or two individuals to create the next population, and the genetic operators used depend upon the number of selected individuals. If one individual is selected, only the mutation operator will be used. In case the designer chooses to select two individuals, both the standard crossover and mutation operators [15] will be applied (see Table 3). Like in others IEC systems, the fitness function relies exclusively on designers' decision, either based on his aesthetic appeal or on desired features.

According to Bentley [20], the designer is likely to score individuals highly inconsistently as he might adapt his requirements along with the evolved results. So, the continuous generation of new forms based on the fittest from the previous generation is essential. Consequently, nonconvergence of the EA is a requirement. Evolutionary art systems do not usually use crossover operators on their algorithms, because EAs are used as a continuous novelty generators, not as optimisers. Therefore, in our algorithm, the use of two individuals for breeding the next generation should be limited. The extensive use of crossover operator will converge the population to a single solution, leading to the loss of diversity and limiting the designer to explore further forms.

Each GP individual is a tree composed by functions, listed in Table 1, and height maps as terminals (see Table 2). Most terminals depend upon a random ephemeral constant (REC) to define some characteristics, such as the spectrum value of *fftGen*. All terminals have some form of randomness, which means that consecutive calls of the same terminal will always generate a slightly different height map. This is a desired characteristic because we want to be able to create different terrains by each TP, but we want them to share the same features. All terminals generate surfaces that are proportional to the side size of the height map. This ensures that the terrain features of a TP are scale invariant. Figure 5 shows height maps of size 30×30 generated by terminals *fftGen*, *gauss*, *step*, and *sphere*. Except *rand*, all terminals depend upon a random ephemeral constant (REC) to define some characteristics. REC is a special terminal that creates values randomly which remain constant until it disappears from the GP tree due to the use of a genetic operators. Figure 6 presents an example of a TP in tree form with two REC values represented in grey ellipses within the terminals.

While in [23, 24], the mathematical equations are used to calculate both the pixel value and its coordinates, in *GenTP*

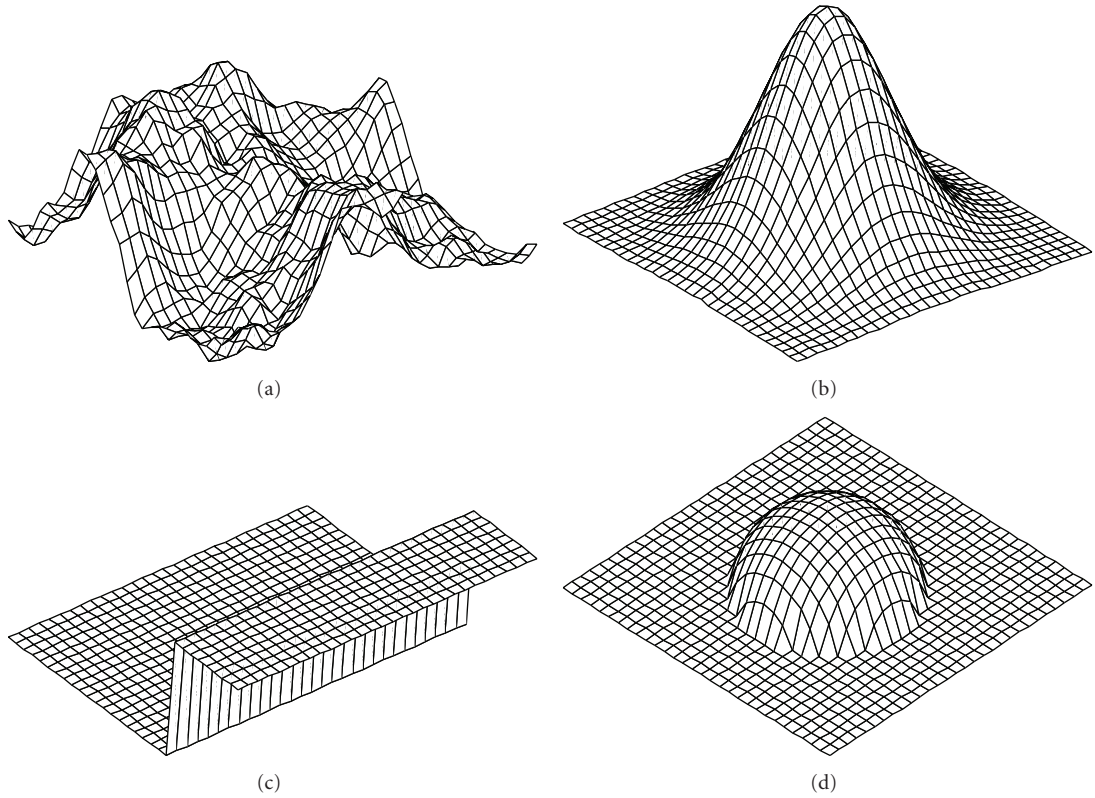


Figure 5: Examples of height maps terminals *fftGen*, *gauss*, *step*, and *sphere*.

Table 2: GP terminals.

Name	Description
rand	Map with random heights between 0 and 1
fftGen	Spectral synthesis-based height map, whose spectrum depends on an REC: $1/(f^{\mathrm{REC}})$
gauss	Gaussian bell shape height map, whose wideness depends on an REC
plane	Flat inclined plane height map whose orientation depends on an REC within 8 values
step	Step shape height map whose orientation depends on an REC within 4 values
sphere	Semisphere height map whose centre location is random and the radius depends on an REC

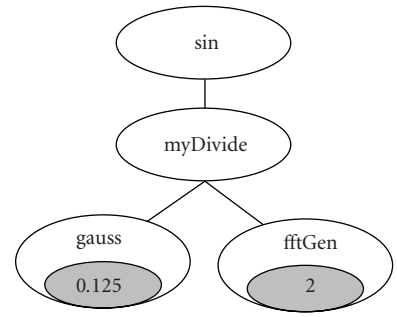


Figure 6: Example of a GP tree individual with two RECs (in grey ellipses).

only the height will be calculated. The (x, y) coordinates will be dictated by the matrix position occupied by the height value.

In GTP, the 12 individuals of the population must be executed during the interactive evolutionary phase to be evaluated by a designer, which will choose the TPs for the next generation. This means that using high resolution on this phase will consume more time, and the application will be less responsive. An additional variable (that will be denoted as s) is introduced in all terminals to control the resolution during the TP execution. The axis values in the terminals' functions are discrete with regular intervals, and the variable s controls the spacing between axis values by specifying the height map grid size, which covers a predefined area. The greater is the s value, the lesser is the distance between each grid point and greater is the resolution.

3.2. GenTP Tool. To implement this new technique, we developed Generator of Terrain Programs (GenTP) [28], an application developed with GPLAB

Table 3: Parameters for a GTP run.

Objective	Generate realistic or aesthetic terrains
Function set	Functions from Table 1, all operating on matrices with float numbers
Terminal set	Terminals from Table 2 chosen randomly
Selection and fitness	Decided by the designer accordingly to desired terrain features or aesthetic appeal
Population	Fixed size with 12 individuals; initial depth limit 6; no tree size limits; random initialisation
Parameters	If 2 individuals are selected: 90% subtree crossover and 10% mutation; if just one individual is selected: 50% mutation (without crossover)
Operators	Three mutation operators are used with equal probability: (1) *Replace mutation* where a random node is replaced with a new random tree generated by the grow method; (2) *Shrink mutation* where a random subtree (S) is chosen from the parent tree and replaced by a random subtree of S; (3) *Swap mutation* where two random subtrees are chosen from the parent tree and swapped, whenever possible the two subtrees do not intersect. One crossover operator is used: *subtree crossover* where random nodes are chosen from both parent trees, and the respective branches are swapped creating two offsprings
Termination	Can be stopped at any time by the designer, the "best" individual is chosen by the designer

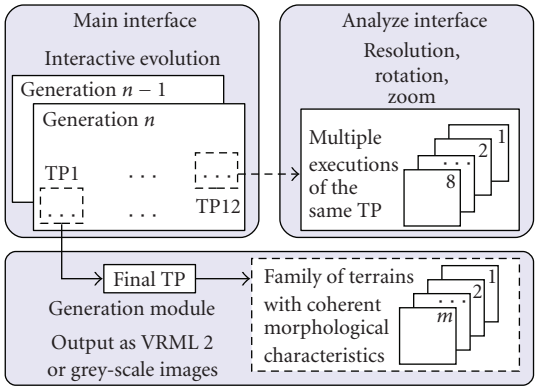


Figure 7: *GenTP*'s functional modules.

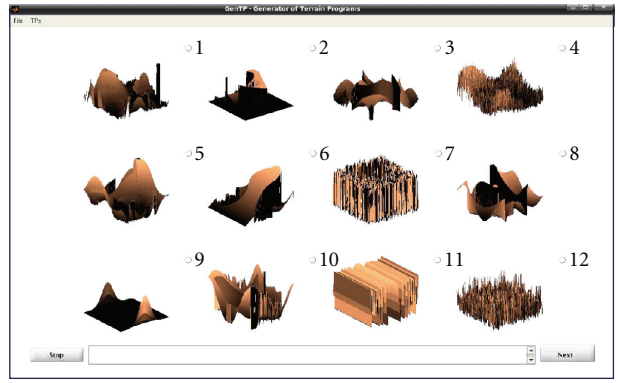


Figure 8: *GenTP* main user's interface.

(http://gplab.sourceforge.net/), an open source GP toolbox for MATLAB (http://www.mathworks.com/). *GenTP* has three functional modules (depicted in Figure 7):

(i) interactive evolution,

(ii) analyse,

(iii) generation.

The interactive evolution module is where the GP is implemented, and the designer chooses the desired terrains for the next generation, for the analyse or generation modules. Figure 8 shows the graphical user interface (GUI) of *GenTP*'s main interface, which is the visible part of the interactive evolution module. The 12 individuals of current population are represented as 3D surfaces and displayed in a 3×4 grid. Each TP is evaluated to produce a height map of size 100×100 to be displayed to the designer. The height map size can be changed, but should be kept small otherwise it might have a negative impact in the tool responsiveness. We will return to this point later on Section 3.3.

The *GenTP* main GUI allows a designer to select one or two individuals to create the next population generation. The number of selected TPs will influence their evolution. If just one TP is selected—only the mutation operator will be applied—the next generation will present few variations from the selected individual, and the TP will evolve slowly. On the other hand, if the designer opts to select two individuals, the next generation will present more diversity and the evolved TPs can change their look more dramatically.

On the bottom of the main GUI, the designer can see the TP mathematical expression that generated the selected terrain and save it on a text file or database. This option will allow the integration of TPs, as a procedural technique, to produce terrains for example on a video game.

Although the main interface serves its purpose, some times it is difficult to see all TP features due the display angle used to show the generated terrain. It may be also difficult to inspect small details of a generated terrain, and it is not possible to test the TP's features perseverance across multiple executions. For these reasons, it might be difficult for the designer to chose the TPs for the next generation. To

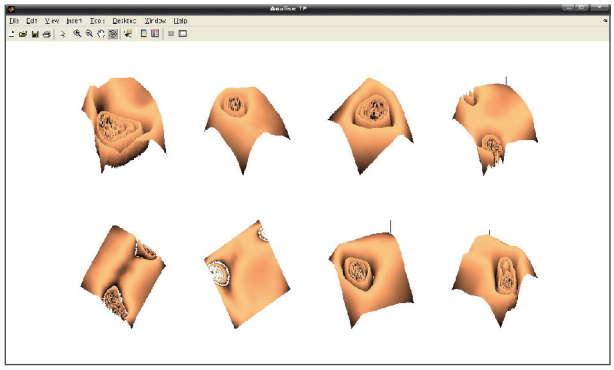
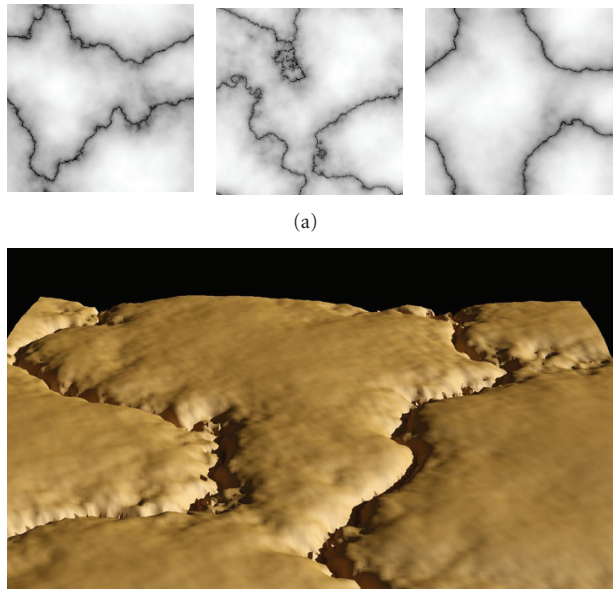


Figure 9: *GenTP* analyse user's interface.



(a)

(b)

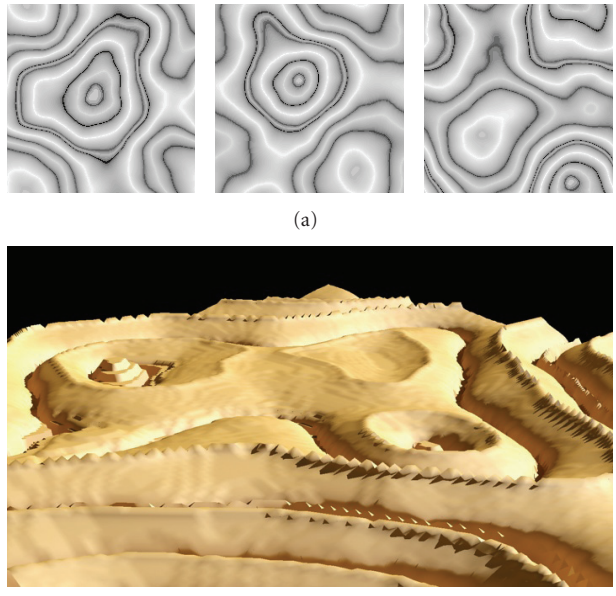
Figure 10: Family of terrains from TP in (2): (a) represented as grey-scale images, (b) rendered with 3D studio.

solve these limitations, the analyse module was added to our application. This new functionality opens a new window, see Figure 9, and performs 8 consecutive executions of the TP selected from the main interface. To allow a more detailed analysis of the TP characteristics, this interface allows the designer to rotate, zoom, and change the terrains resolution. This way the designer has more information about a TP to decide if it will be selected, or not, for the next generation.

When the designer achieves the desired TP, then he can save it in a file, or can pass it to the generator module. This module is responsible for the generation of height maps, as many as desired, from the selected TP. Those height maps can be saved as VRML 2.0 permitting its import from other applications, such as 3D modelling and render tools.

3.3. Experimental Results. Our previous work [6] has shown the ability of our technique to evolve TPs capable of generating a family of height maps (different terrains that share the same morphological characteristics). Figure 10 shows terrains from a TP evolved with river beds in mind, and Figure 11 shows terrains from a TP evolved to obtain a family of aesthetic terrains. All these results were obtained with a fixed resolution of 200×200.

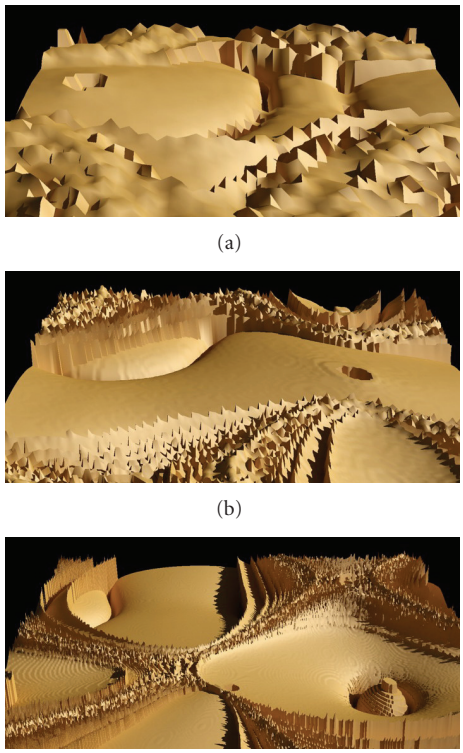
An experiment was conducted to test the perseverance of terrain features across several resolutions and the consequent impact in generation time on our evolutionary tool [29]. A set of TPs was chosen to generate terrains with grid sizes from 50 to 450 with increments of 50. To perform these tests, it was necessary to modify the terminals in order to include the variable s to specify the resulting height map size.



(a)

(b)

Figure 11: Family of terrains from TP in (3): (a) represented as grey-scale images, (b) rendered with 3D studio.

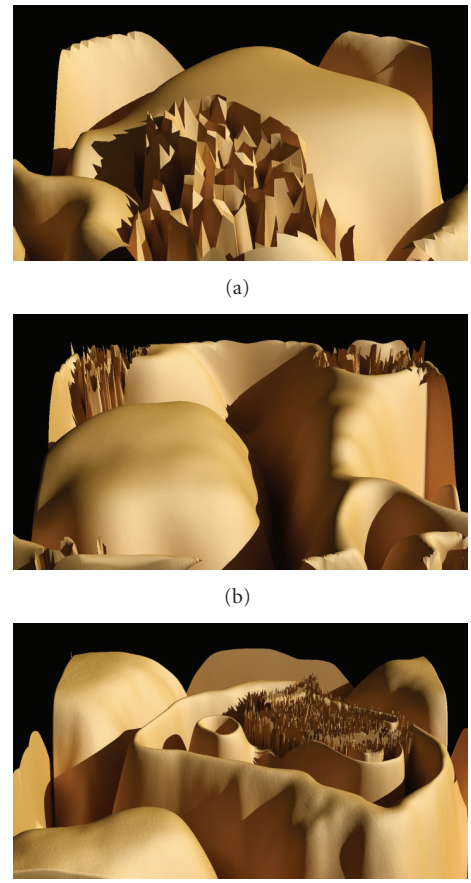


(a)

(b)

(c)

Figure 12: Exotic terrain generated by TP in (4), with resolutions 50×50, 150×150, and 450×450.



(a)

(b)

(c)

Figure 13: Exotic terrains generated by TP in (5), with resolutions 50×50, 150×150, and 450×450.

Figures 12, 13, 14, and 15 present the results of TP's shown in (4), (5), (6), and (7) at three different resolutions with grid sizes of 50×50, 150×150, and 450×450.

$$\begin{aligned} \text{TP} = {} & \text{myLog}\,(\text{myLog}\,(\cos(\text{minus}\,(\mathit{fftGen}\,(2.00), \\ & \mathit{fftGen}\,(3.75))))) \end{aligned} \tag{2}$$

$$\begin{aligned} \text{TP} = {} & \text{myLog}\,(\text{myLog}\,(\text{myLog}(\text{myLog}\,(\text{myLog}\,(\\ & \text{myLog}(\mathit{fftGen}\,(3.00))))))) \end{aligned} \tag{3}$$

$$\begin{aligned} \text{TP} = {} & \text{myLog}\,(\text{myLog}\,(\text{myMod}\,(\text{myLog}\,(\mathit{fftGen}\,(s, 3.75)), \\ & \text{myLog}\,(\text{myLog}\,(\mathit{fftGen}\,(s, 4.25)))))) \end{aligned} \tag{4}$$

$$\begin{aligned} \text{TP} = {} & \text{myPower}\,(\cos(\text{myDivide}\,(\text{myLog}\,(\text{smooth}(\\ & \mathit{fftGen}(s, 2.75))),\ \text{myMod}\ (\sin(\mathit{fftGen}\,(s, 0.50)), \\ & \text{myDivide}\,(\text{myLog}\,(\text{smooth}\,(\mathit{fftGen}\,(s, 2.75))), \\ & \text{myMod}\,((\sin(\mathit{fftGen}\,(s, 0.50))), \mathit{fftGen}\,(s, 2.25)))))) \end{aligned} \tag{5}$$

$$\begin{aligned} \text{TP} = {} & \text{multiply}(\,\sin(\mathit{fftGen}(s, 3.00)), \text{smooth}(\\ & \text{multiply}(\sin(\cos(\sin(\cos(\text{multiply}(\mathit{fftGen}\,(s, 1.75), \\ & \mathit{fftGen}\,(s, 0.75))))))), \mathit{fftGen}\,(s, 0.50)))) \end{aligned} \tag{6}$$

$$\begin{aligned} \text{TP} = {} & \text{plus}\,(\mathit{fftGen}\,(s, 2.00), \\ & \text{smooth}\,(\text{myMod}\,(\text{gauss}\,(s, 0.75), \\ & \cos(\mathit{fftGen}\,(s, 1.00))))) \end{aligned} \tag{7}$$

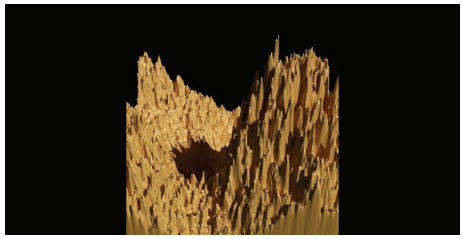
In these experiments, all TPs have preserved their main features independently of the chosen grid size. Due to the inherent randomness embedded in terminals, consecutive calls of the same TP will always generate a slightly different height map. This is a desired characteristic, that can be controlled by fixating the random number seed. Note that when generating terrains at different resolutions, the amount of necessary random numbers will vary accordingly with the chosen resolution. This explains the differences from terrains at different resolutions generated by the same TP.

Figure 16 shows the average time of 10 executions of each TP at each grid size on a Pentium Core 2 Duo at 1.66 GHz with 2 GB of RAM. As expected, the generation time increases at a quadratic pace with the increase of the number of grid points, for example, for TP 7 from 18.4 millisecond at 50×50 to 1066.0 millisecond at 450×450. The generation time also increased, as anticipated, with the number of TP's nodes.

The values presented on Figure 16 are the times for generating each individual. The time to generate the entire population must be multiplied by the population size. For TP 5 (with 17 nodes) with a resolution of 450×450, each individual takes 3.122 seconds. So, to generate an entire population of 12 indviduals, 37.464 seconds will be

(a)

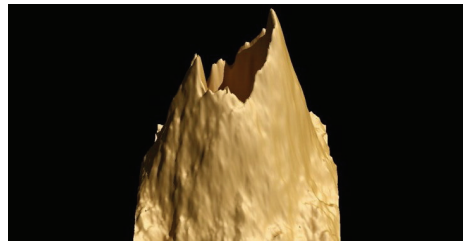
(b)



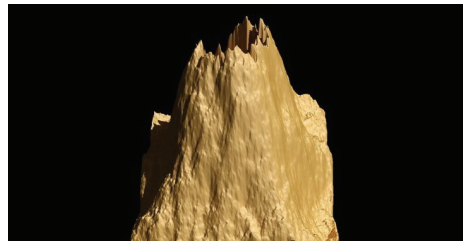
(c)

Figure 14: Mountains generated by TP in (6), with resolutions 50×50, 150×150, and 450×450.

(a)



(b)



(c)

Figure 15: Volcanoes generated by TP in (7), with resolutions 50×50, 150×150, and 450×450.

needed. A delay of this magnitude is not negligible and will have a negative impact on the response time of interactive application such as our tool. According to Card et al. [30] and Testa and Dearie [31],

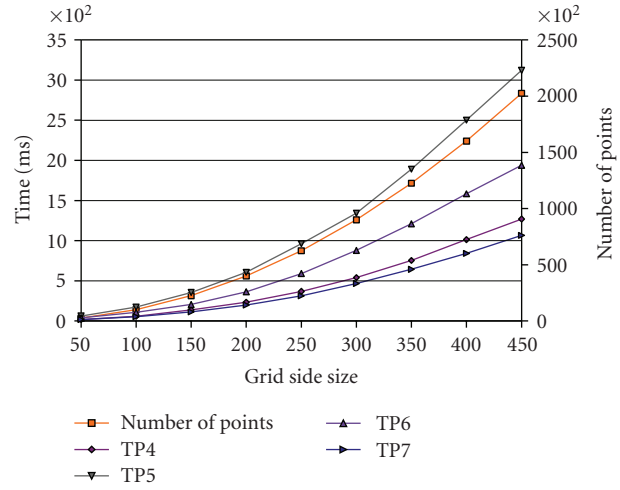
(i) 0.1 second is about the limit for having the user feel that the system is reacting instantaneously, meaning that no special feedback is necessary except to display the result;

(ii) 1.0 second is about the limit for the user's flow of thought to stay uninterrupted, even though the user will notice the delay. Normally, no special feedback is necessary during delays of more than 0.1 but less than 1.0 second, but the user does lose the feeling of operating directly on the data;

(iii) 10 seconds are about the limit for keeping the user's attention focused on the dialogue. For longer delays, users will want to perform other tasks while waiting for the computer to finish, so they should be given feedback indicating when the computer expects to be done. Feedback during the delay is especially important if the response time is likely to be highly variable, since users will then not know what to expect.

The response times should be as fast as possible to keep designer's attention focused on the application. Our goal is to keep generation times for the entire population around 1 second and never exceeding 10 seconds. The generation time depends on the chosen resolution and on each individual number of nodes, which tends to increase with the number of GP generations, a phenomenon known as bloat [27]. So, from the responsiveness point of view, the use of the lowest resolutions for the evolutionary phase is better. However, if the used resolution is too low the output might not represent all the terrain features, specially small details, and force the designer to use the analyse window more often. This will increase the time needed by the designer to choose the best terrain at each generation and consequently the overall time to achieve the desired terrain. A compromise must be made between the terrain resolution for the evolutionary phase and the application responsiveness. From our set of experiences we found the grid size of 100×100 to be the best settlement. Short generation times will be also advantageous for the future implementation of automated terrain evolution.

Once the TP is designed it can be used on a video game like any other procedural technique. On this case, the time for generating a new terrain is negligible, given that it will be generated before the game begins, during the "load" period.



Grid size	Time (ms)			
	1 (TP4)	2 (TP5)	3 (TP6)	4 (TP7)
50	18.5	61.4	37.7	18.4
100	59.6	171.9	106.9	53.9
150	136.4	355.1	205.3	113.1
200	233.2	605.7	362.9	198.6
250	368.3	962.5	590.1	312.6
300	539.8	1342.3	878.7	466.8
350	753.5	1889.8	1210.3	643.4
400	1012.7	2501.1	1583.6	841.3
450	1269.8	3122.1	1939.1	1066.0
GP nodes	8	17	13	7

Figure 16: Terrain generation times versus grid sizes.

4. Conclusions and Future Work

This paper presented the GTP technique which allows the evolution of TPs to produce terrains accordingly to designers' aesthetic feelings or desired features. Through a series of experiments we have shown that the feature persistence is independent of the chosen resolution. This means that during the evolutionary phase low resolutions can be used without compromising the result. Consequently less time will be required for our evolutionary tool, enabling it to be more responsive, which is an important characteristic on interactive tools. Additionally, the resulting TPs can be incorporated in video games, like any other procedural technique, to generate terrains, with the same features, independently of the chosen resolution.

Some game publishers require that all players have the same game "experience" if they make the same choices. They want to measure or obtain quality control both on the user experience side as well as on the development and testing end. This requirement seems to contradict our goal of achieving different terrains with the same TP. However, if a TP is incorporated on a video game as a procedural technique, our technique can deliver two levels of control regarding randomness. First, a specific TP will always generate terrains with the same features, this means that in spite of the present randomness those terrains are similar and not completely random. Second, if full control over the final terrain is required the seed for the random number generator can be kept the same across separate runs of the TP, allowing the same terrain to be regenerated as many times as desired.

The TPs' scale invariance showed in our results preludes the implementation of a zoom feature. Fixating the random number generator seed is not enough to implement this feature due to the variation of the amount of necessary random numbers accordingly with the zoom. Besides, some terminals, like *rand* and *fftGen*, are not based on continuous functions. Other improvement to our technique will be the composition of a terrain through the use of several TPs, previously stored on a database, where the generated terrains will be joined on a credibly and smooth way. This will allow the control over localised terrain features. We also want to implement the GTP technique as a Blender plug-in to increase both the flexibility and the target audience for our technique.

The search for a terrain with a specific feature might be a tiresome endeavour on interactive evolutionary applications [20]. Therefore, it is desirable to automate, as much as possible, the task of evaluating TPs to avoid designers fatigue. We plan to develop fitness functions to perform the automatic evaluation of TPs accordingly to a feature by means of statistic measures. Another future work will be the inclusion of more features in our technique in order to generate full landscapes including textures, vegetation, and buildings.

Acknowledgment

The third author acknowledges the support of MICINN under project TIN2008-05941.

References

[1] M. Duchaineau, M. Wolinsky, D. E. Sigeti, M. C. Miller, C. Aldrich, and M. B. Mineev-Weinstein, "ROAMing terrain: real-time optimally adapting meshes," in *Proceedings of the 8th IEEE Visualization Conference (VIS '97)*, pp. 81–88, IEEE Computer Society Press, Phoenix, Ariz, USA, October 1997.

[2] F. Losasso and H. Hoppe, "Geometry clipmaps: terrain rendering using nested regular grids," *ACM Transactions on Graphics*, vol. 23, no. 3, pp. 769–776, 2004.

[3] S. Li, X. Liu, and E. Wu, "Feature-based visibility-driven CLOD for terrain," in *Proceedings of the 11th Pacific Conference on Computer Graphics and Applications*, pp. 313–322, IEEE Computer Society Press, Canmore, Canada, October 2003.

[4] J. H. Clark, "Hierarchical geometric models for visible-surface algorithms," in *Proceedings of the 3rd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '76)*, p. 267, Philadelphia, Pa, USA, July 1976.

[5] B. Pelton and D. Atkinson, "Flexible generation and lightweight view-dependent rendering of terrain," Tech. Rep. COEN-2003-01-22, Department of Computer Engineering, Santa Clara University, Santa Clara, Calif, USA, 2003.

[6] M. Frade, F. F. de Vega, and C. Cotta, "Modelling video games' landscapes by means of genetic terrain programming—a new approach for improving users' experience," in *Proceedings of*

the EvoWorkshops on Applications of Evolutionary Computing, M. Giacobini, A. Brabazon, S. Cagnoni, et al., Eds., vol. 4974 of *Lecture Notes in Computer Science*, pp. 485–490, Springer, Naples, Italy, 2008.

[7] R. L. Saunders, *Terrainosaurus: realistic terrain synthesis using genetic algorithms*, M.S. thesis, Texas A&M University, College Station, Tex, USA, 2006.

[8] A. D. Kelley, M. C. Malin, and G. M. Nielson, "Terrain simulation using a model of stream erosion," in *Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '88)*, pp. 263–268, ACM, Atlanta, Ga, USA, August 1988.

[9] J. Olsen, "Realtime procedural terrain generation—realtime synthesis of eroded fractal terrain for use in computer games," Department of Mathematics and Computer Science (IMADA), University of Southern Denmark, 2004.

[10] H. O. Peitgen, H. Jürgens, and D. Saupe, *Chaos and Fractals: New Frontiers of Science*, Springer, New York, NY, USA, 2nd edition, 2004.

[11] R. Voss, "Fractals in nature: characterization, measurement, and simulation," in *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '87)*, Anaheim, Calif, USA, July 1987.

[12] C. Darwin, *On the Origin of Species by Means of Natural Selection*, John Murray, London, UK, 1859.

[13] J. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, Mich, USA, 1975.

[14] T. Bäck, F. Hoffmeister, and H. P. Schwefel, "A survey of evolution strategies," in *Proceedings of the 4th International Conference on Genetic Algorithms*, pp. 2–9, San Diego, Calif, USA, July 1991.

[15] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press, Cambridge, Mass, USA, 1992.

[16] L. J. Fogel, A. J. Owens, and M. J. Walsh, *Artificial Intelligence through Simulated Evolution*, John Wiley & Sons, New York, NY, USA, 1966.

[17] W. B. Langdon and A. Qureshi, "Genetic programming—computers using "natural selection" to generate programs," Tech. Rep. RN/95/76, University College London, London, UK, 1995.

[18] T. S. Ray, "Evolution, ecology and optimization of digital organisms," 1992.

[19] P. Bentley, "Aspects of evolutionary design by computers," in *Advances in Soft Computing: Engineering Design and Manufacturing*, Springer, New York, NY, USA, 1998.

[20] P. Bentley, *Evolutionary Design by Computers*, Morgan Kaufmann, San Francisco, Calif, USA, 1999.

[21] K. Sims, "Artificial evolution for computer graphics," in *Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '91)*, pp. 319–328, ACM, Las Vegas, Nev, USA, August 1991.

[22] T. Unemi, "A design of multi-field user interface for simulated breeding," in *Proceedings of the 3rd Asian Fuzzy and Intelligent System Symposium (AFSS '98)*, pp. 489–494, Masan, Korea, June 1998.

[23] T. Unemi, "SBART 2.4: breeding 2D CG images and movies and creating a type of collage," in *Proceedings of the 3rd International Conference on Knowledge-Based Intelligent Information Engineering Systems (KES '99)*, pp. 288–291, Adelaide, Australia, August-September 1999.

[24] T. Unemi, "SBART 2.4: an IEC tool for creating 2D images, movies, and collage," in *Proceedings of the Workshop on Genetic and Evolutionary Computational Conference*, p. 153, Las Vegas, Nev, USA, July 2000.

[25] P. Machado and A. Cardoso, "NEvAr—the assessment of an evolutionary art tool," in *Proceedings of the Symposium on Creative & Cultural Aspects and Applications of AI & Cognitive Science (AISB '00)*, G. Wiggins, Ed., Birmingham, UK, April 2000.

[26] T. J. Ong, R. Saunders, J. Keyser, and J. J. Leggett, "Terrain generation using genetic algorithms," in *Proceedings of Genetic and Evolutionary Computation Conference (GECCO '05)*, pp. 1463–1470, ACM, Washington, DC, USA, June 2005.

[27] R. Poli, W. B. Langdon, and N. F. McPhee, *A Field Guide to Genetic Programming*, Lulu, Morrisville, NC, USA, 2008.

[28] M. Frade, F. F. de Vega, and C. Cotta, "Gentp—uma ferramenta interactiva para a geração artificial de terrenos," in *Proceedings of the 3rd Iberian Conference in Systems and Information Technologies (CISTI '08)*, M. P. Cota, Ed., vol. 2, pp. 655–666, Ourense, Spain, April 2008.

[29] M. Frade, F. F. de Vega, and C. Cotta, "Genetic terrain programming—an aesthetic approach to terrain generation," in *The Annual International Conference & Symposium on Computer Games and Allied Technology (CGAT '08)*, pp. 1–8, Singapore, April 2008.

[30] S. K. Card, G. G. Robertson, and J. D. Mackinlay, "The information visualizer, an information workspace," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '91)*, pp. 181–186, ACM, New Orleans, La, USA, April-May 1991.

[31] C. J. Testa and D. B. Dearie, "Human factors design criteria in man-computer interaction," in *Proceedings of the Annual Conference*, vol. 1, pp. 61–65, ACM, 1974.