Adding Zoom Feature to Terrain Programmes

Miguel Frade[1], F. Fernandez de Vega[2] and Carlos Cotta[3]
[1]Instituto Politécnico de Leiria, Portugal, mfrade@estg.ipleiria.pt
[2]Universidad de Extremadura, Mérida, Spain, fcofdez@unex.es
[3]Universidad de Málaga, Málaga, Spain, ccottap@lcc.uma.es

Abstract— **The Genetic Terrain Programming technique is an evolutionary approach for the generation of artificial terrains. It is based on evolutionary design with Genetic Programming and allows designers to evolve terrains according to their aesthetic feelings or desired features. This technique evolves Terrain Programmes (TPs) that are mathematical expressions that can be executed as a procedural technique. This kind of techniques have two important traits which are their ability to compute a terrain with the desired zoom level and resolution. However, the present implementation of TPs only allows to choose the resolution. This paper discusses the hurdles that prevent the zoom feature in TPs and proposes a solution to them. To sustain the feasibility of our solution some results were obtained as a proof of concept.**

Key words— **genetic terrain programming, evolutionary systems, terrain program, zoom**

I. 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 and virtual reality. However, video games is the field where artificial terrain generation techniques are probably more prominent.

There are several traditional terrain generation techniques that can be divided into measuring, modeling and procedural techniques. The procedural techniques are based on programmatic generation and offer several advantages over the other ones [1]. Its representation is extremely compact and can be measured in *Kilobytes*, while others require *Megabytes* of storage. The procedural landscapes are unlimited in extent and can cover an arbitrarily large area without seams or unwanted pattern repetition. Some offer mathematical advantages that make them friendly for rendering and allow ray intersections to be calculated in a straightforward way [1]. They are also parameterised, which allow the generation of a family of related terrains. Finally they have no fixed resolution and can compute a terrain to be viewable at any zoom scale with the desired resolution. Some of the most popular procedural techniques are fractal algorithms, which are a sub-category of procedural techniques. These algorithms are the favourite ones by game's designers, mainly due to their speed and simplicity of implementation, in addition to the zoom and resolution features.

Procedural techniques have also their own limitations [1]. One disadvantage is their evaluation. This operation requires intense computations and can be very expensive (this is the classic trade-off of time versus space). But the main disadvantage of procedural techniques is the difficulty of modelling with them. It is very difficult or impossible to know how to modify them to achieve a certain local effect.

Frade et al. [2] proposed a new evolutionary approach designated GTP (Genetic Terrain Programming). This technique allows to overcome the modeling problem of procedural techniques as well as limitations of other traditional generation techniques. This approach consists of the combination of evolutionary art systems with GP to evolve mathematical expressions, designated TPs (Terrain Programmes), to generate artificial terrains as height maps. TPs can be executed as any other procedural technique. However, unlike other procedural techniques, TPs' current implementation does not allow to choose the desired zoom level [3]. This paper discusses the obstacles that prevent the zoom feature implementation and presents a solution to surpass them.

Section II introduces some background about resolution and zoom features of procedural techniques and the following section details the GTP technique. Section IV discusses the current limitation of TPs and presents some results of the proof of concept for the proposed solution. Finally, the conclusions and future work are presented on Section V.

II. Background

An important characteristic of procedural techniques is their ability to generate a scene with the required *resolution* and *zoom* level - in fact this is probably the main advantage of the procedural techniques over the other types of techniques.

Due to computers' digital nature they cannot truly represent continuous data. So, all continuous data must be sampled to discrete values. The amount of samples per unit determines the resolution, which is perceived by the user as the quality of the digitalised function. Fig. 1 and 2 represent a continuous function with two different samples rates and the correspondent result of that sampling. The higher the sampling rate, the better is the quality of the digitalised function. However, after a certain point there is no use to increase the amount of sampling because of the display medium limitation, or ultimately, due to the biological limitations of the human eyes. For instance, many LCD monitors can only display images up to 72 dpi (dots per inch). So, increasing the sampling rate beyond this limit will require more

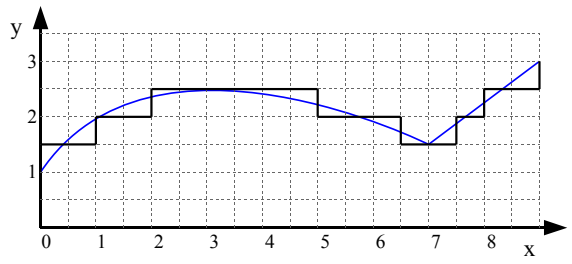

Fig. 1.
Example of a continuous function sampling, where the grid lines represent the sampling points.

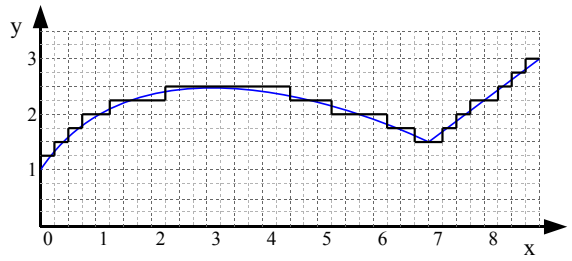

Fig. 2.
Example of the sampling of the same continuous function as in Fig. 1, but now with twice the samples - twice the resolution

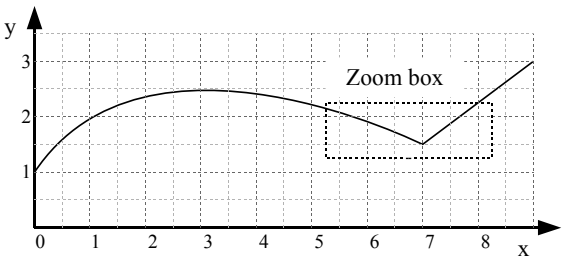

Fig. 3.
Sampling grid of a continuous functions before zoom

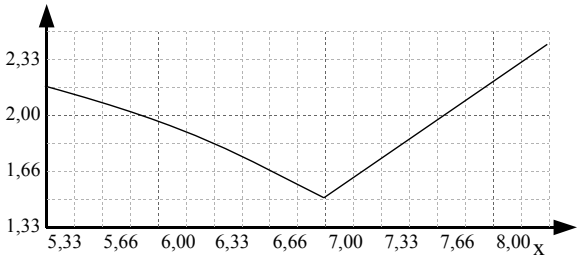

Fig. 4.
Sampling grid after a 3× zoom of the zoom box shown in Fig. 3

storage space but does not improve user's perceived quality.

If it is required to view more details than the ones allowed by the display medium limitations it is possible to resort to zoom. The zoom feature consists of narrowing the apparent angle of view of a scene, giving the sense of approximation. This feature can be achieved in procedural techniques by scaling and increasing the sampling rate. For example, Fig. 3 represents continuous function where the sampling rate is 2 (shown by the grid lines). To enlarge three times the area bounded by the zoom box, the sampling rate must be increased three times and the output must be scaled for the same resolution, as shown in Fig. 4. Notice the distance between each sample before and after the zoom, 0.5 units and 0.166 units respectively. However, they are represented in the output medium with the same distance between them due to the scaling. For easier illustration the examples used a function with just one input variable. Nonetheless, the same principles apply to any function independently of how many input variables it has.

It is thanks to the resolution and zoom properties present in procedural techniques, that computers can better simulate and represent continuous data.

III. Genetic Terrain Programming

The Genetic Terrain Programming (GTP) [2] technique is based on Aesthetic Evolutionary Design and was developed to address the weaknesses of existing terrain generation methods, allowing also the generation of aesthetic terrains. This technique lies in the combination of interactive evolutionary art systems with GP to evolve mathematical expressions, designated TPs, to generate artificial terrains as height maps. Those TPs can be used, like a procedural technique, to automatically generate different terrains with different resolutions with the same consistent features.

GTP relies on GP as evolutionary algorithm where the initial population is created randomly, with trees depth size limited initially to 6 and a fixed population size of 12 (see Table I). The number of generations is decided by the designer, who can stop the algorithm at any time. 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 [4] will be applied. Like in others IEC systems, the fitness function relies exclusively on designers' decision, either based on his aesthetic appeal or on desired features.

Accordingly to Bentley [5] 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, non-convergence 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

TABLA I
PARAMETERS FOR A GTP RUN

Objective:	Generate realistic or aesthetic terrains
Function set:	Functions from Table II, all operating on matrices with float numbers
Terminal set:	Terminals from Table III 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, after there are no tree size or depth 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 prabability: (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 offspring.
Termination:	Can be stopped at any time by the designer, the "best" individual is chosen by the designer

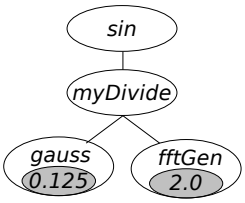

Fig. 5.
Example of a GP tree individual with two RECs, in grey ellipses

next generation should be limited. The extensive use of the crossover operator will make the population converge to a few solutions, thus leading to the loss of diversity of individuals and limiting the designer to explore further terrains.

Each GP individual is a tree composed by functions, listed in Table II, and height maps as terminals (see Table III). 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. Fig. 5 presents an example of a TP in tree form with two REC values represented in grey ellipses within the terminals.

While in [6], [7] the mathematical equations are used to calculate both the pixel value and its coor-

TABLA II
GP FUNCTION SET

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 $mod(h_1, h_2)$ otherwise
$mySqrt(h)$	returns $sqrt(abs(h))$
$negative(h)$	returns $-h$
$FFT(h)$	2-D discrete Fast Fourier Transform
$smooth(h)$	circular averaging filter with $r = 5$
$gradientX(h)$ $gradientY(h)$	returns the gradient (dh/dx or dh/dy) of a height map h. Spacing between points is assumed to be 1

TABLA III
GP Terminal Set

Name	Description
rand	map with random heights between 0 and 1
fftGen	spectral synthesis based height map, whose spectrum depends on a REC: $1/(f^{REC})$
gauss	gaussian bell shape height map, whose wideness depends on a REC
plane	flat inclined plane height map whose orientation depends on a REC within 8 values
step	step shape height map whose orientation depends on a REC within 4 values
sphere	semi-sphere height map whose centre location is random and the radius depends on a REC

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

The experiments conducted on [2] showed that to obtain aesthetic appealing terrains (regardless of their realism) it was required about 30 to 70 generations. However, to obtain TPs to generate terrains with specific features, such as mountains or cliffs the number of necessary generations varies widely until an acceptable result was obtained. The number of generations is highly dependent on the initial population and could vary between 10 to more than 100 generations. If, after a number of generations, an interesting result was not obtained the experiment was canceled and began again to avoid this way a long run.

IV. Terrain Programmes

In spite of the procedural nature of the TPs, the current implementation only allows to choose terrain resolution, but not zoom level [3]. This is a limitation that runs against the procedural advantages which we want to eliminate.

First, it is necessary to understand why this limitation exists. If we take a closer look to the GTP technique we can verify that the terminal set, shown in Table III, does not depend directly on (x, y) input variables. Each terminal generates a matrix of values, that represent a height map, whose coordinates will be dictated by the matrix position occupied by the height value. Without direct control over the (x, y) coordinates it is not possible to implement the zoom feature. The terminals *gauss*, *plane*, *step* and *sphere* can be easily rewritten to be directly dependent of (x, y). But the same does not hold for the *rand* and *fftGen* terminals.

As the name suggests, the *rand* terminal generates random numbers. Although we can fixate the random number seed to ensure the same values can be obtained as many times as desired, that behaviour is not enough to allow the implementations of the zoom feature. This happens for two reasons, first the random number function is not continuous. Second, the output of that function depends on the number of times it is called and not on an input variable. Fig. 6 shows the consequences of random numbers in the current TP implementation. When we increase the terrain resolution the resulting terrain is different, in spite of their similarities.

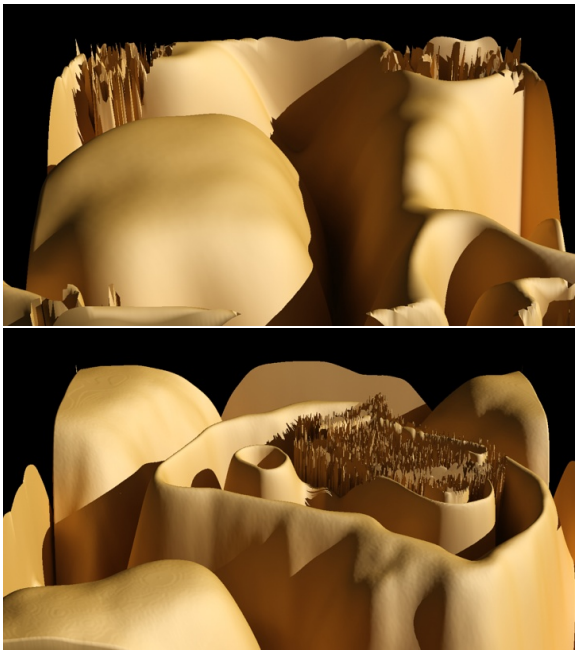

Fig. 6.
Two terrains generated by the same TP with different resolutions that show the problem of random numbers in the current implementation

The *fftGen* terminal is more complex, it is based on the Fourier transform. The theory of Fourier states that any function can be represented as a sum of sinusoidal terms. The Fourier transform takes a function from the spatial or time domain into the frequency domain, where it is represented by the amplitude and phase of a series of sinusoidal waves. Summing together the series of sinusoidal waves reproduces the original function - this is called the inverse Fourier transform [8] . The *fftGen* terminal starts by generating random frequency components (amplitude values), then a low band filter is applied to eliminate high frequency components. Finally the inverse Fast Fourier Transform (FFT) - an efficient algorithm to compute the discrete Fourier transform - is computed to convert the frequency components into altitudes [9]. The outcome of this terminal is a height map whose surface roughness can be controlled by the low band filter. The lower the filter value, the smoothest the surface is. This terminal presents two problems for the zoom implementation: first it is based on a random number generator thus

suffering from the same problems of the *rand* terminal. Additionally, even if we solve the random number issue, this terminal would still be an obstacle to the zoom feature due to the fact of working initially with components in the frequency domain. The inverse FFT algorithm requires a large set of points to convert them to the space domain, it does not allow the computation of a single point which is required to implement the zoom feature.

A. Proposed Solution

The main obstacle to implement the zoom feature are the *rand* and *fftGen* terminals. They must be replaced by terminals that depend directly from the (x, y) input coordinates, or simply eliminated. For the *rand* terminal we believe that it should be replaced, otherwise we would lost one important characteristic of a single TP to be able to produce a family of terrains - different terrains that share the same morphological characteristics. To replace the *rand* terminal we propose the use of *noise* functions, these kind of functions have been widely used in procedural textures for several years. Noise functions are stochastic functions whose ideal properties are [1]:

- the noise function must have a repeatable pseudo-random output, based on its inputs variables;
- the output range is known, namely from -1 to 1;
- the output is band-limited, with a maximum frequency of about 1;
- the noise function should not exhibit obvious periodicities or regular patterns. Pseudorandom functions are always periodic, but the period can be made very long and therefore the periodicity is not notable.
- the noise function is stationary, that is, its statistical character should be translationally invariant.
- the noise function is isotropic, that is, its statistical character should be rotationally invariant.

There are several noise functions available, such as: *Voronoi*, *Cell Noise*, *Perlin*, *Blender Noise*, among others. Although these functions share the statistical behaviour previously described, they produce different outputs. So, the question that arises is: which one should replace our *rand* terminal? Instead of replacing the *rand* terminal by a specific noise function, we propose to add all the noise functions, or at least the ones that differ more on their output, to our GP terminal set. This way we do not have to decide which noise function is the best for our propose, but also will alllow our technique to produce more diverse and hopefully more interesting TPs. All the noise functions depend on (x, y, z), but TPs only need two input coordinates. This can be easily solved by fixating the third coordinate to a value, such as $z = 0$.

Regarding the *fftGen* terminal its simple elimination is not desirable because most of the interesting TPs produced by the GTP have this terminal present at least once. The problem is to know which new terminal, or terminals, should replace it. We started our search by comparing the output of the *fftGen* terminal (see Fig. 7) to possible candidates. The best candidates we found to replace *fftGen* were procedural fractals for terrains [1]. Like the noise, there are also several procedural fractals:

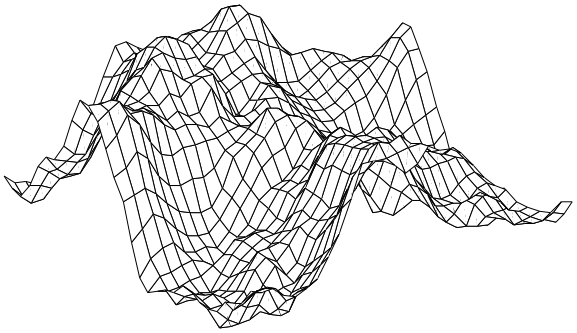

Fig. 7.
Sample output of the *fftGen* terminal

- distorted noise
- hetero-terrain
- multifractal
- hybrid multifractal
- ridged multifractal
- fBm (fractal Brownian motion)
- turbulence
- voronoi

None of the analysed fractals produced the same output of our *fftGen* terminal, but *fBm* was one that showed more similar results. The procedural fractals have many input variables, such as: H, Lacunarity and Octaves among others. These parameters could also be evolved, so it will be more beneficial to add them to the function set instead of the terminal set. Again, we believe that adding all the fractals to our function set will increase the diversity and the chances to produce more interesting TPs. However, this approach will break the compatibility with previously generated TPs. This means that changing the nodes from the previous TPs by new ones will produce different results. Anyhow, we believe that the advantages of the zoom feature brought by this approach will largely overcome the disadvantage of breaking retro-compatibility. Besides, we hope to easily produce many new interesting TPs with these changes on the GTP technique.

The solution we propose consists of changing the terminal set, shown on Table III, to the ones in Table IV. The *rand* terminal is replaced by the several new terminals discussed previously and the remaining terminals will be rewritten to depend directly from (x, y) input coordinates. The *fftGen* will be removed from the terminal set and the procedural fractal functions will be added to the function set to compensate its lost.

These changes have other implications on the GP function set shown in Table II. The functions $FFT(h)$, $smooth(h)$, $gradientX(h)$ and $gradientY(h)$ depend upon a large set of points.

TABLA IV
PROPOSED GP TERMINAL SET

Name	Description
gauss	gaussian bell shape
plane	flat inclined plane
step	step shape
sphere	semi-sphere shape
blender-noise *o-perlin* *i-perlin* *voronoi-F1* *voronoi-F2* *voronoi-F3* *voronoi-F4* *voronoi-F2-F1* *voronoi-crackle* *cell-noise*	noise functions

They cannot operate over a single (x, y) point and, to the best of our knowledge, there are no alternative implementations. As a first approach to this problem we will remove those functions from the functions set and see if the the new procedural fractal functions can, somehow, compensate them. Table V shows the changed function set. All the new procedural fractal functions have (x, y) input variables, but we do not count them for the n-arity. For now we want to preserve these input variables, so we will not allow the GP system to manipulate them.

B. Proof of concept

The proposed changes require the implementation of many new terminals and functions. In order to test the new ideas we decided to make a simple proof of concept. We choose Blender [1], an open source 3D modeling tool, because it has a Python [2] API which has interfaces for the new noise terminals and procedural fractal functions [3].

The next step was to chose one of the previous TPs and adapt it for the new terminal set and check both the aesthetic of the outcome and the zoom feature. Due to the lack of compatibility between some old and new terminals, the adaptation of most previous TPs resulted in non interesting terrains. The TP shown in Eq. (1) was an exception – see Fig. 8. We changed the $fftGen(3.00)$ by the $fBm(x, y, 0, 1.0, 1.97, 2)$ function. The input parameters of *fBm* were manually fine tuned to generate aesthetic terrains more similar to the ones produced prior to the modification.

$$TP = myLog(myLog(myLog(myLog(myLog(myLog(myLog(fftGen(3.00))))))) \ . \quad (1)$$

[1] http://www.blender.org
[2] http://www.python.org
[3] http://www.blender.org/documentation/248PythonDoc/Noise-module.html

TABLA V
PROPOSED GP FUNCTION SET

Name	Description
$plus(a, b)$ $minus(a, b)$ $multiply(a, b)$	arithmetical functions
$sin(a)$ $cos(a)$ $tan(a)$ $atan(a)$	trigonometric functions
$myLog(a)$	returns 0 if $h = 0$ and $log(abs(a))$ otherwise
$myPower(a, b)$	returns 0 if a^b is NaN or Inf, or has imaginary part, otherwise returns a^b
$myDivide(a, b)$	returns a if $b = 0$ and $a \div b$ otherwise
$myMod(a, b)$	returns 0 if $b = 0$ and $mod(a, b)$ otherwise
$mySqrt(a)$	returns $sqrt(abs(a))$
$negative(a)$	returns $-a$
$distorted_3$ $heteroTerrain_4$ $multiFractal_3$ $hybridMFracta_5$ $ridgedMFractal_5$ fBm_3 $turbulence_5$ $voronoi_2$	procedural fractal terrains (the subindices are the functions' n-arities)

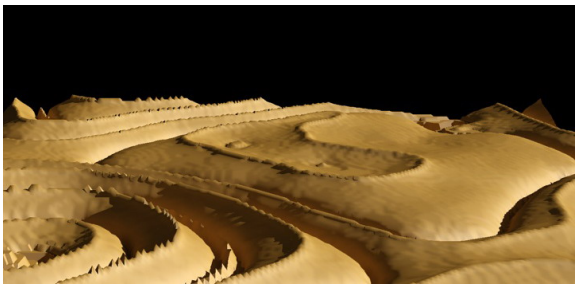

Fig. 8.
Sample of a terrain generated by TP in Eq. (1)

$$TP = myLog(myLog(myLog(myLog(myLog(myLog(fBm(x, y, 0, 1.0, 1.97, 2))))))) \ . \quad (2)$$

Fig. 9 and 10 show two different terrains generated by the TP shown in Eq. (2), with three zoom levels 50%, 100% and 200% (from top to bottom). Like the noise functions, the procedural fractal functions will give always the same output for the same (x, y) input values. So, in order to obtain a different terrain from the same TP, we change the starting values of (x, y) input range. Fig. 11 shows the last image of Fig. 9 with eight times less resolution. All these images show the top view of the terrains generated within Blender 3D and rendered with YafRay.

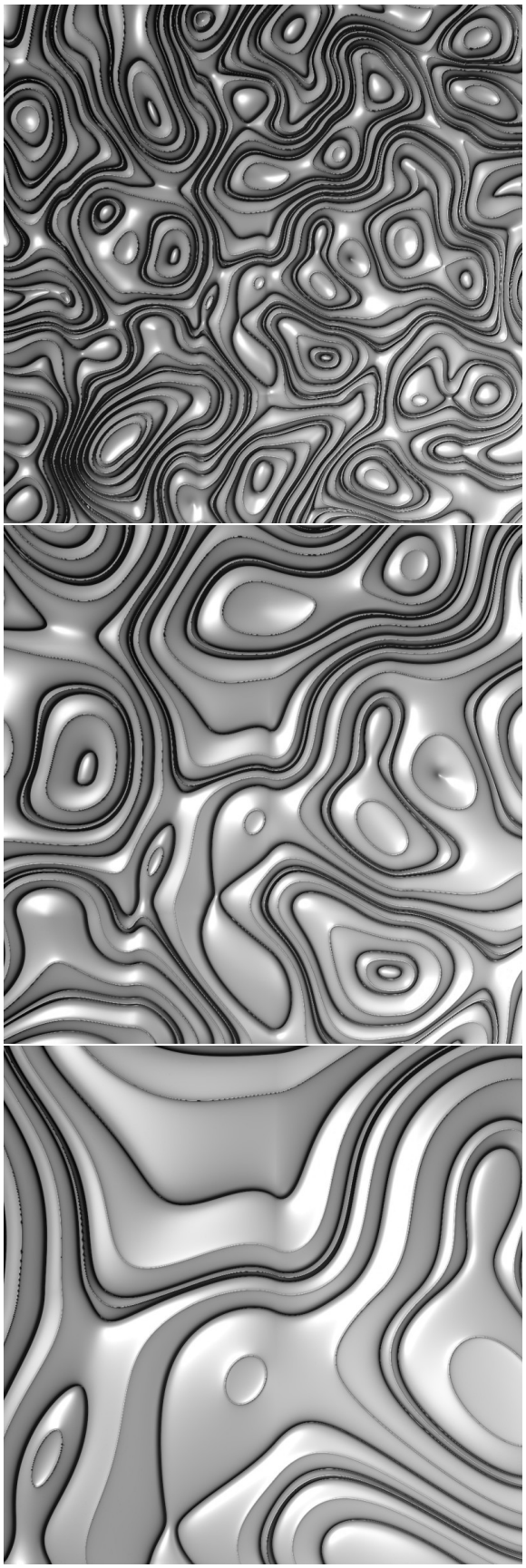

Fig. 9.
First sample of a terrain generated by TP in Eq. (2) with three zoom scales: 50%, 100% and 200%

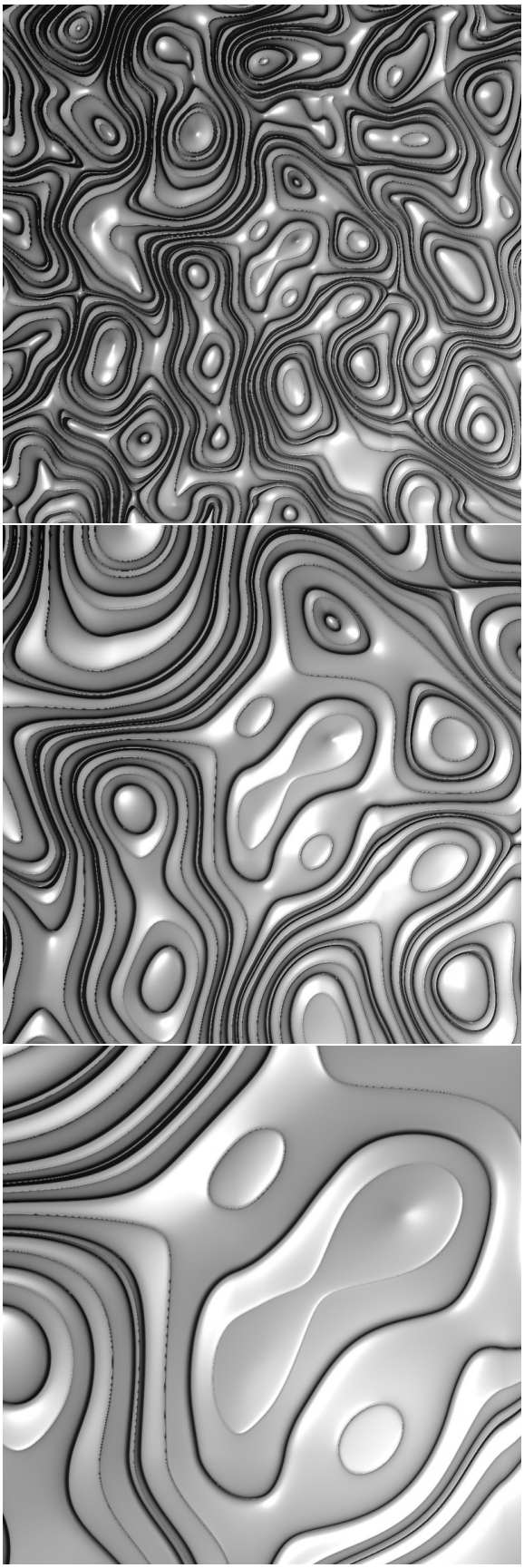

Fig. 10.
Second sample of a terrain generated by TP in Eq. (2) with three zoom scales: 50%, 100% and 200%

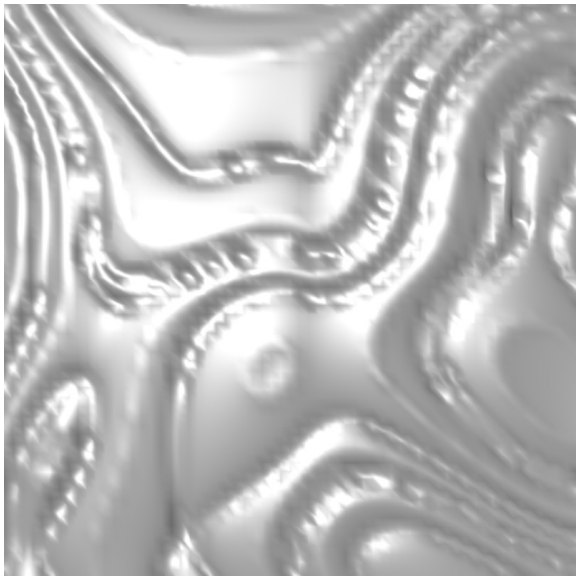

Fig. 11.
Image of the last terrain from Fig. 9, but generated with eight times less resolution

V. Conclusions

The GTP techniques evolves TPs which are mathematical expressions and can be executed as any other procedural technique. However, TPs did not allow the implementation of the zoom feature, which is one of the most important traits of procedural techniques. The obstacles to its implementation were identified and a solution to surpass them was proposed. A proof of concept was built to bring some insight to the feasibility of the proposed solution.

The obtained results encourage us to pursue the full implementation of the proposed solution. With this approach we expect to create many new TPs for both real looking and aesthetic terrains. These changes will allow the direct integration of TPs on a video game to automatically generate terrains. Nevertheless, the proposed solution brings some important modifications to our technique. The new TPs will not be compatible with the ones generated previously. Also the GP terminals will not be matrices and the image processing functions will be eliminated form the function set.

VI. Future Work

To continue the improvement of the GTP technique several future lines of investigation are suggested. One of them is the composition of a terrain through the use of several TPs, where the generated terrains will be joined on a credibly and smooth way. Another one is to add more features to GTP so that whole scenarios, including vegetation and buildings, can be generated by the same technique. Finally it will be also desirable to develop a fitness function to allow the automatic evolution of TP's and avoid designers fatigue, a common problem of interactive evolutionary applications [5].

Acknowledgements

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

References

[1] David Ebert, Kenton Musgrave, Darwyn Peachey, Ken Perlin, and Steven Worley, *Texturing and Modeling: A Procedural Approach*, Morgan Kaufmann, 3rd edition, 2003.

[2] Miguel Frade, F. Fernandez de Vega, and Carlos Cotta, "Modelling video games' landscapes by means of genetic terrain programming - a new approach for improving users' experience," in *Applications of Evolutionary Computing*, Mario Giacobini et al., Ed., Napoli, Italy, 2008, vol. 4974 of *LNCS*, pp. 485–490, Springer.

[3] Miguel Frade, F. Fernandez de Vega, and Carlos Cotta, "Genetic terrain programming - an aesthetic approach to terrain generation," in *Computer Games and Allied Technology 08*, Singapore, 2008, pp. 1–8.

[4] J. R. Koza, "Genetic programming. on the programming of computers by means of natural selection," *Cambridge MA: The MIT Press.*, 1992.

[5] Peter Bentley, *Evolutionary Design by Computers*, Morgan Kaufmann Publishers, Inc., CA, USA, 1999.

[6] Tatsuo Unemi, "SBART 2.4: breeding 2D CG images and movies and creating a type of collage," in *The Third International Conference on Knowledge-based Intelligent Information Engineering Systems*, Adelaide, Australia, 1999, pp. 288–291, IEEE.

[7] Tatsuo Unemi, "SBART 2.4: an IEC tool for creating 2D images, movies, and collage," in *Proceedings of 2000 Genetic and Evolutionary Computational Conference*, NV, USA, 2000, p. 153.

[8] Ronald N. Bracewell, *The Fourier Transform & Its Applications*, McGraw-Hill Science/Engineering/Math, 3 edition, 1999.

[9] Jacob 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.