Dijkstra-based Terrain Generation Using Advanced Weight Functions

Procedia Computer Science

Procedia Computer Science 101, 2016, Pages 152 – 160

YSC 2016. 5th International Young Scientist Conference on Computational Science

Dijkstra-based Terrain Generation Using Advanced Weight Functions

Kirill Golubev, Aleksander Zagarskikh, and Andrey Karsakov
ITMO University, Saint Petersburg, Russia
golubev1251@gmail.com, alazar.az@gmail.com, kapc3d@gmail.com

Abstract
Due to the growing popularity of consumer virtual reality devices, the new surge in the use of computer-generated terrain is already happening. That leads to the need to develop new fast and efficient methods of landscapes generation. However, lack of flexibility of user control over terrain generation in most popular modern terrain generation algorithms is a big part of terrain generation problem. In this paper, we present a new method for generating three-dimensional landscapes based on modified Dijkstra algorithm. The proposed method allows a user to set the initial location of landscape features and select or create weight functions that determine the appearance of the generated terrain. It has a lower computational cost compared to the closest analogs giving the equal quality of results and allows users to create various types of terrain, as well as to combine them together in one landscape.

Keywords: terrain generation, weight functions, virtual worlds, visualization

1 Introduction

Generation and visualization of virtual landscapes is an actual task in different areas, such as development of the educational and simulation environments, video games, the creation of decorations for movies and animations, as well as environments for simulation of various natural processes [1]. Due to the growing popularity of consumer virtual reality devices, the new surge in the use of computer-generated terrain is already happening. That leads to the need to develop new fast and efficient methods of landscapes generation.

Most of the modern generation algorithms generate terrains that require subsequent manual revision by designers or in need of automatic completion by sophisticated algorithms of erosion, climate and weather strains, like a thermal distortion of the landscape. Such approach is well suited for applications where it is necessary to create a single rather small area or when the landscapes are configured manually but is ill-suited for the generation of large terrain areas in a fully automatic mode.

doi:10.1016/j.procs.2016.11.019

This paper describes a new method of terrain generation based on Dijkstra algorithm and the use of advanced weight functions that allow to balance between initial user control and fully automated generation.

2 Related Works

The problem of the terrain generation has a long history and still a universal algorithm that can be used for any task does not exist. In the 1968 Mandelbrot and Van Ness described the fractal Brownian motion [2] and 15 years later Fournier et al. proposed a method for fractal Brownian surfaces generation based on midpoint displacement with linear time complexity [3]. In 1985 Perlin presented a gradient noise [4] to generate procedural textures, including height maps, that is still commonly used as a part of terrain generation pipeline. All these studies laid the foundation for the family of fractal methods based on the addition of the level of details using self-similarity. Despite the simplicity of these methods, they are still used for the terrain generation and underpin the popular powerful tools, such as World Machine [5] and Terragen [6].

Modern requirements for terrain generation algorithms have been formulated by Saunders [7]. However, the most popular algorithms do not meet requirements in “permit a high degree of human control” and “be able to reproduce a wide variety of recognizable terrain types and features realistically, in believable relationship to one another”. They generate uniform landscapes and locations that quickly become boring. Attempts to solve these issues lead to the unnecessary complication of the initial parameters and the need for active participation of the human in the terrain creation. Also, the development of the generation algorithms is always an attempt to strike a balance between these two extremes.

Further development of the landscape generators went in several directions. Kelley et al. [8] presented a method of terrain generation based on the stream erosion, and Musgrave et al. [9] developed the idea of using the hydraulic and thermal erosion. This determined the new direction of the generators based on erosion that was extended further by works of Nagashima [10], Benes et al. [11], Cordonnier et al. [12] and other authors. Using different types of erosion is one of the most popular ways to improve the look of the landscape. However, such methods are also not entirely satisfying the Sanders’ requirements, mainly due to the low performance. Furthermore, the implementation of erosion still requires the basic landscape and generation of this basic terrain must meet the same requirements. Brosz et al. in [13] introduced a method of details improving for low-quality landscapes using highly detailed real terrain data, which allows increase the realism of the terrain created in any manner without the use of erosion.

Another approach to landscape generation is to use patch-based and feature-based generators. These algorithms allow the user to mark the location of the main elements (features) of the landscape, and the algorithm completes the terrain mesh using this data. Belhadj and Audibert [14] proposed a method that uses precomputed ridge lines and rivers network and fills the rest of the landscape using midpoint displacement. Zhou et al. in their work [15] proposed a method for transporting real landscape elements on the user-defined features. Their method allows to create a detailed and nice-looking landscape but poorly solves the problem of the diversity of terrain features. Moreover, this method is in need of real landscape data that is not always possible. Also, a significant drawback is the high computational cost of this method, caused by the need not only to build the landscape but also to search for suitable areas in the existing height map.

In 2009, Gain [16] presented a method to create mountains relying on the custom profiles. It is well suited as an editing tool, but not for entirely automatic generation. Hnaidi et al. [17], developed the idea of Gain by filling in the landscape between feature lines using diffusion equation. Such method gives good results but requires advance preparation of detailed locations of features.

In [18] Rusnell et al. describe a method based on Dijkstra algorithm and blending of layers, which are generated in accordance with individual features. Method works significantly faster than Zhou's and, in some cases, able to produce results with better quality, but it has several drawbacks. They include the monotonous shape of slopes that belong to the same profile, as well as the procedure of blending that significantly increases the time of generation of the landscape.

The method proposed in this paper is based on Dijkstra's algorithm, the same approach as by Rusnell et al., but it uses other ways of calculating weights of the edges and performs all the computations on a single mesh that gives a significant gain in performance.

3 Method Description

The primary goal of the proposed method is to obtain a landscape in the form of a height map. To do this, we form intermediate weight maps that represent the impact of various landscape features. Then we define a weight function that receives values of all detail maps and data in a current node and calculates the height difference for this node. Next, the initial set of nodes is used as a frontier for the modified Dijkstra algorithm that uses weight function instead of static weights. The proposed method is summarized in Listing 1.

Listing 1: Dijkstra-based terrain generation method using advanced weight functions.

```
 1.     Define weight function delta ($m_1$, $m_2$, …, $m_n$, p)
 2.     Define detail maps $M_1$-$M_n$,
 3.     Define initial points of algorithm P.
 4.     Execute modified Dijkstra algorithm:
   a.     Q <- P
   b.     while(|q| > 0)
   c.       Q -> p
   d.       p.processed = true
   e.       for each n - neighbor of p
   f.         if (not p.processed)
   g.         d = delta($M_1$[p], $M_2$[p], …, $M_n$[p], p)
   h.               if (h[n] < h[p] - d) then
   i.           h[n] = h[p] - d
   j.             Q <- n
   k.           end if
   l.         end if
   m.       end for
   n.     end while
```

Positive weight function delta(m_1, m_2, … ,m_n, p) is declared on step 1. It can be either uniformly distributed random value or complex conditional sub-algorithm that computes height difference based on detail maps values and data of parent node p such as height, position on height map, parent of the node, spawn (node p_0 from P from which we come to p) and feature stats. By using different weight function behaviors allows to split generated landscape into biomes and combine different terrain types, such as dunes and cliffs.

At the second step, terrain detail maps are chosen. Different detail maps can be used for different purposes: from simple noises to define steepness or feature intensity, to biomes placement, road maps, shorelines definition, etc.

The third step sets the initial nodes that represent highest points of landscape features and used as initial values for Dijkstra algorithm. For examples in this paper, a uniform distribution of features was used, mostly with single points or Bezier-based lines. However, different approach can be used to create specific landscapes, for example, using hand-drawn features placement.

On the final step modified Dijkstra algorithm is running. It uses priority queue *Q* to store all nodes that ready to be processed. It checks its neighbors *n* for each node *p* taken from the queue. If the neighbor node is not processed by an algorithm yet, it calculates its new height *h[n]* as a difference between height of *p* and an edge weight *d* and updates it if the old value is lower. The weights of the edges are not set beforehand and are calculated dynamically on the basis of the weight function *delta* at the time of treatment to the edge. Point updates only if the new height is greater than the previously recorded values at this point.

The computational complexity of steps from 1 to 3 is O(n). Dijkstra's algorithm works for O(n*log(w)), where w is not more than 6*n (and in fact significantly less). Thus, the total running time is O(n*log(n)). The number of computations of the weight functions value is O (n) at the same time.

3.1 Weight Functions

Weight functions are a powerful way to alter landscape generated by the proposed method. Using different weight functions, it is possible to create hills, mountains, dunes or any other type of terrain, pack them into biomes and join them without any artifacts.

Weight function can differ from simple linear or uniform math functions using only vertex value from a height map, to complex sub-algorithms depending on detail map value, child and parent node relative position, etc. For example, dunes can be created by using basic constant multiplied by the scalar product of wind direction (specified in user settings) and direction to feature-spawn node (Figure 1). That weight function can be described in the following equation:

$$d = \delta_h * I\left(R\left(\vec{w} * \overrightarrow{\delta_{\mathrm{pos}}}\right)\right),$$

where δ_h is base weight that can be a constant, a noise value or somehow interpolated parent node height, I and R is interpolation and reverse interpolation $\left(\mathrm{I}: (0,1) \rightarrow (\delta_{\min}, \delta_{max}) and\ R: (-1,1) \rightarrow (0,1)\right)$, respectively $\vec{\mathrm{w}}$ is normalized wind direction and $\overrightarrow{\delta_{\mathrm{pos}}}$ is distance between current node and node from initial Dijkstra frontier.

As another example, canyons, terraces and crags can be created by setting higher weight values at specific nodes, determined by spawn-node distance or detail map values.

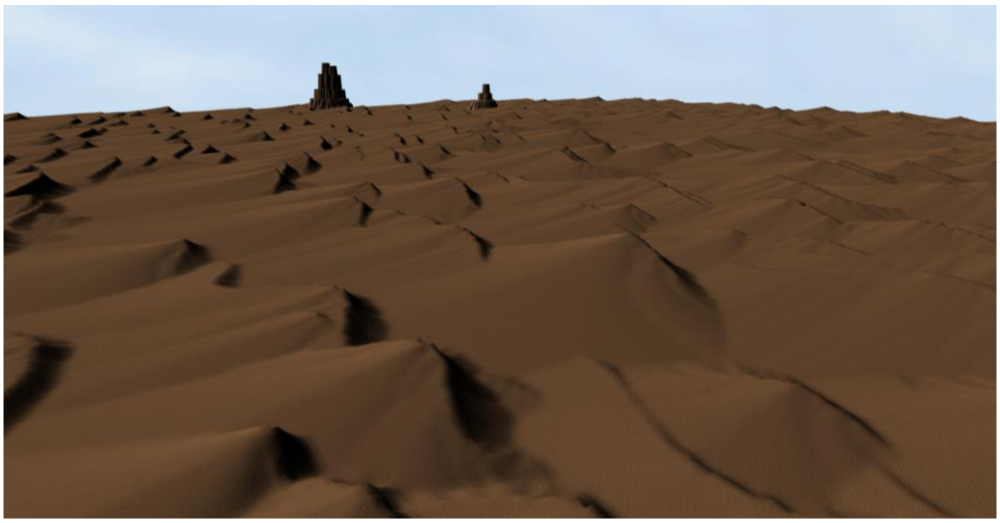

Figure 1: An example of the desert landscape, generated using the advanced weight functions and the shape of basic features.

3.2 Placement of Initial Points

Another way to customize generated landscape is to set specific basic nodes for Dijkstra algorithm. It can be set by the user or generated by the placement algorithm. If the initial set consists of randomly selected single points (mountain peaks), mountains look uniform and unnatural. In this case, the landscape can be improved by the user by setting not only points but lines, Bezier curves or more complicated shapes, like sine sum graphs that were used to produce the sides of the dunes at Figure 1. For the mountain landscape polylines of Bezier curves are well suited. To generate ridges, we took three-dimensional Bezier curve, placed it on the height map and added Perlin noise. After all these operations we set all of this points as initial nodes for the algorithm. Resulting landscape presented in Figure 2.

In general, for any type of weight function, it makes sense to choose individual initial set guided by the basic purpose of the landscape, real data, and user's fantasy.

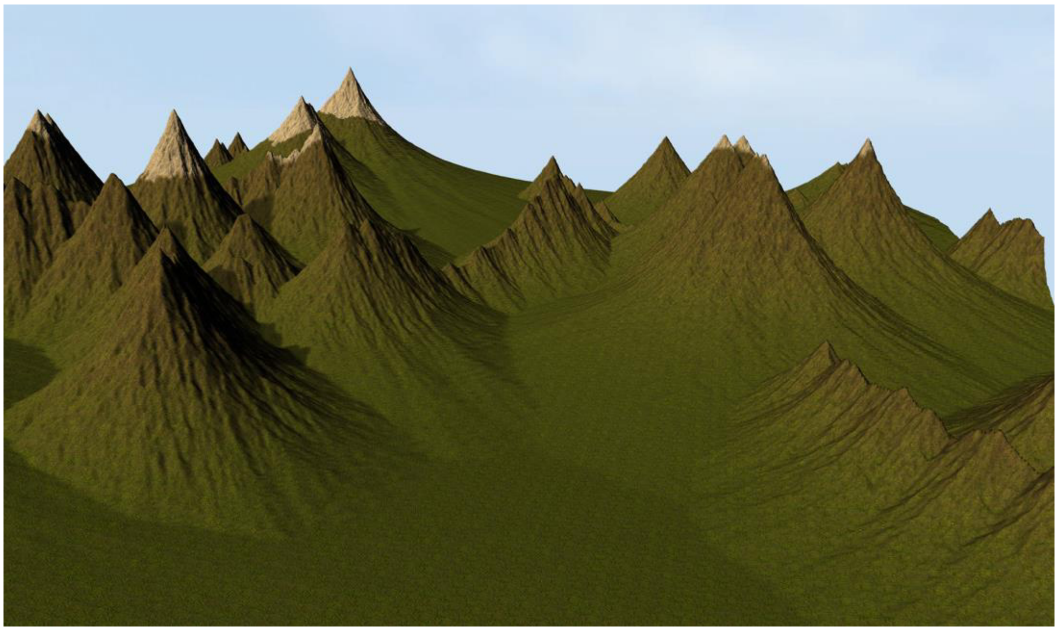
Figure 2: Mountain ridges generated using Bezier curves as an initial data.

4 Results and Performance Evaluation

We use Unity3D game engine and C# programming language to implement the proposed method and terrain component to visualize resulting landscape. All tests were performed on AMD Athlon(™) II X4 645 Processor 3.11GHz with 8 GB RAM and NVidia GeForce GT440 video card. Figure 3 shows a set of landscape examples generated by the proposed method and Table 1 shows a performance evaluation for this examples.

Example	Height map resolution	Features count	Computation time, sec	Weight function calls
1	512x512	128	3.17	825224
2	512x512	4	2.58	776535
3	1024x1024	48	12.35	3201594
4	2048x2048	2048	75.89	13077911

Table 1: Performance evaluation of proposed method for examples from Figure 3.

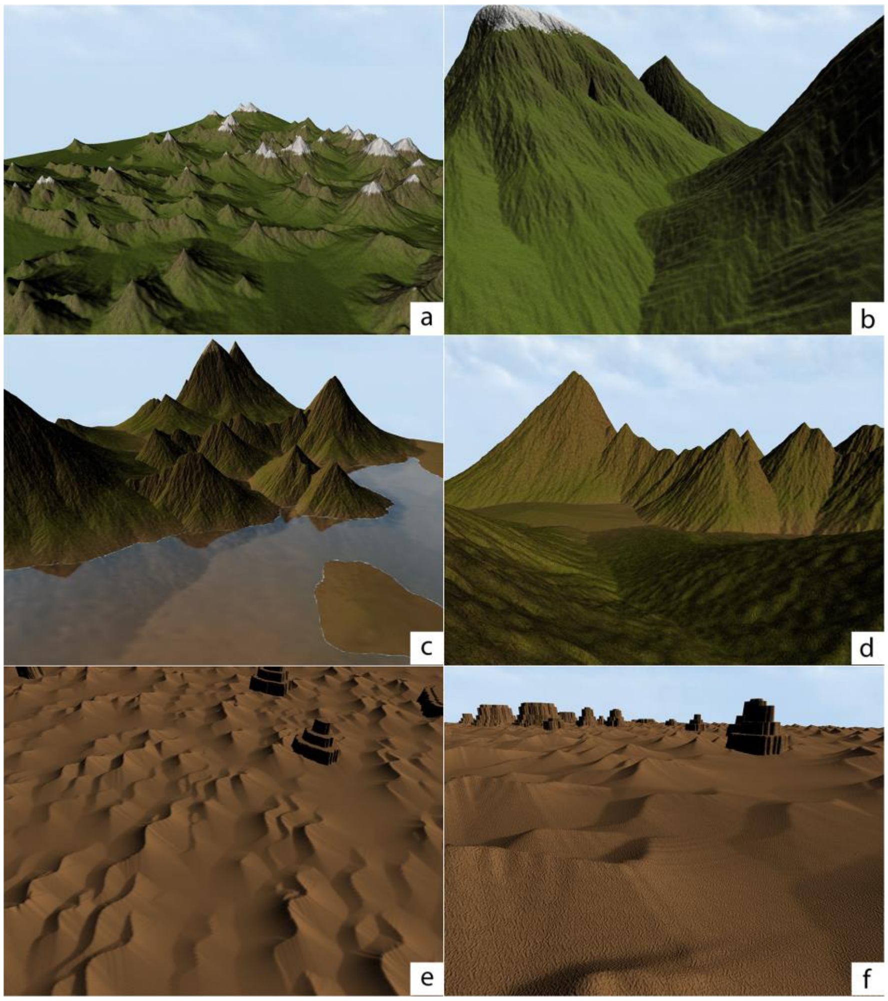

Figure 3: Screenshots of generated landscapes: a – (example 1) a big area landscape using low-resolution height map; b - (example 2) a small area (single mountain) with the same resolution as the example 1; c, d - (example 3) a complex landscape with different feature types, e, f - (example 4) a highly detailed desert landscape with a lot of small features like dunes and crags.

The first example (Figure 3a) demonstrates low-resolution landscape of big area and shows irregularity in mountain placement, untypical for most fully-automated terrain generators. The second example at Figure 3b shows big mountains created using a height map with the same resolution as the previous one and demonstrates the ability of our method to generate detailed landscapes. The third example at Figure 3c and 3d show complex landscape with different feature types, created on 1024x1024 resolution height map with a medium scale of mountains. The fourth example at Figure 3e and 3f demonstrates a desert, created on a big 2048x2048 height map and contains a lot of small features like dunes and crags.

For more precise evaluation of the proposed method complexity, we made measurements of weight function calls and computation time by varying height map size and initial features count and summarized it in Table 2.

It can be noticed that execution time grows even when weight function calls count decreasing at 512…2048 features count examples. From this we can conclude that the time required for the calculation of weight functions is significantly lower than for the implementation of a priority queue in Dijkstra

Height map resolution	Features count						
	2	8	32	128	512	1024	2048
512	700447 2.48 sec	740697 2.55 sec	804382 2.89 sec	824056 3.14 sec	825921 3.53 sec	814417 3.53 sec	817596 3.79 sec
1024	2802248 11.6 sec	2912436 11.44 sec	3397056 12.73 sec	3403763 13.04 sec	3289422 14.01 sec	3296215 15.19 sec	3271992 15.47 sec
2048	11326938 50.48 sec	11809117 48.66 sec	13440152 55.82 sec	13336477 58.26 sec	13240730 59.97 sec	13168505 62.96 sec	13193554 64.6 sec

Table 2: Performance of the proposed method for different initial parameters.

algorithm, which has O(log(n)) operation time per node that exceeds the constant time needed to compute weight functions. Thus, their complexity does not reduce the performance of the method.

As it is shown in Table 2, algorithm execution time does not depend on feature types or its amount a lot, but mostly on height map resolution. At the same time, its ability to create compound multi-featured terrain surpass closest analogs that are currently used for terrain generation.

4.1 Comparison with Rusnell`s Method

As was mentioned in Section 2, in [18] Rusnell et al. presented terrain generation method also based on Dijkstra algorithm, using separate Dijkstra runs for each landscape feature and then mixing the resulting height maps.

The method that we are proposing in this paper is also based on Dijkstra algorithm. However, instead of calculating the profiles for each generator node, we use weight functions. They are based on one or more features maps that are matched with the vertices on a height map. Also, in the presented algorithm, the height of the point is calculated in the first pass of the algorithm that can significantly reduce the landscape generation time. For some particular cases, the performance benefit of our method reaches a few dozen times. However, one pass execution makes landscapes coarser at the junctions of the slopes with different features and at the initial points.

Using the detail maps instead of the usual noise allows creating a much more detailed terrain. Particularly in Rusnell's method mountains have a pronounced symmetry, that can be easily solved using our method by setting detail maps. Figure 4 shows a comparison of landscapes generated by Rusnell et al. and our methods with same basic settings.

For full comparison, we present time and complexity stats for different landscape sizes for examples 1 and 2.

According to the results obtained in comparative tests, it can be seen that the Rusnell's method has significantly greater execution time at the same initial values and with the very close quality of the result. Also, execution time depends on features count almost linearly. Even while Rusnell's method gives smoother landscape, this big difference in time allows us to create better resolution height map with more complex weight function to achieve better results and still generate it faster than Rusnell's method.

	Method	Sample 1 (features count = 80)			Sample 2 (features count = 40)		
		512	1024	2048	512	1024	2048
Execution time, sec	Rusnell et al.	224.07	1121.04	4930.04	146.07	685.31	4967.06
	Proposed method	3.12	13.24	57.23	3.15	13.18	57.40
Weight fun. calls	Rusnell et al.	125992960	503644160	1938399232	64571392	258117632	1032134656
	Proposed method	956970	3832207	15465736	970772	3890438	15396538

Table 3: Performance comparison of the proposed method and method from paper by Rusnell et al. [18].

5 Conclusion and Future Work

In this paper we have presented a new Dijkstra-based method for virtual terrain generation, which can be easily parameterized by weight function to set appropriate terrain type and detail maps to control it more precisely. By varying these parameters and input data, it is possible to generate an enormous diversity of landscapes using the proposed method - from individual biomes to complex landscape structures.

There are many ways for the further development of proposed methods. Improved use of weight functions by storing multiple feature parameters in nodes and selecting more suitable ones can lead to better results on different features junctions. Even more, approximating different features heights on the same node can make junctions even better. Also, we can use feature data for texturing terrain mesh. All of this further method developments can bring a new effective terrain generation tool.

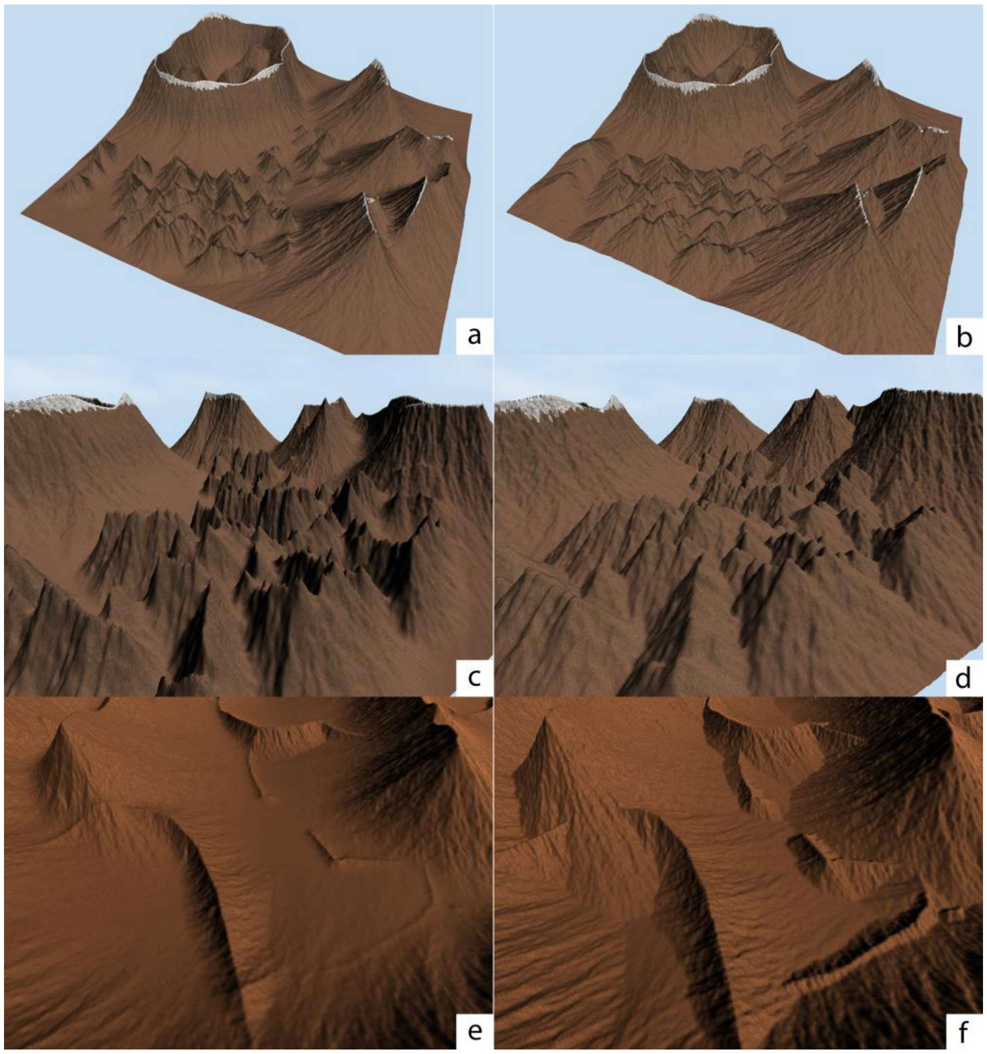

Figure 4: A comparison of landscapes built by Rusnell's method (left) and the proposed method (right) based on the same initial data: a-d – Sample 1; e-f – Sample 2.

Acknowledgments

This paper is financially supported by Ministry of Education and Science of the Russian Federation, Agreement #14.584.21.0015 (11.11.2015)

References

[1] Rastislav Tisovcík, “Generation and Visualization of Terrain in Virtual Environment,” Masaryk University, 2012.

[2] B. B. Mandelbrot and J. W. Van Ness, “Fractional Brownian Motions, Fractional Noises and Applications,” *SIAM Rev.*, vol. 10, no. 4, pp. 422–437, Oct. 1968.

[3] A. Fournier, D. Fussell, and L. Carpenter, “Computer rendering of stochastic models,” *Commun. ACM*, vol. 25, no. 6, pp. 371–384, Jun. 1982.

[4] K. Perlin, “An image synthesizer,” *ACM SIGGRAPH Comput. Graph.*, vol. 19, no. 3, pp. 287–296, Jul. 1985.

[5] L. World Machine Software, “World Machine.” [Online]. Available: http://www.world-machine.com/.

[6] Planetside Software, “Terragen 3.” [Online]. Available: http://planetside.co.uk/products/terragen3.

[7] R. L. Saunders, “Terrainosaurus: realistic terrain synthesis using genetic algorithms,” Texas A&M University, 2007.

[8] A. D. Kelley, M. C. Malin, and G. M. Nielson, “Terrain simulation using a model of stream erosion,” *ACM SIGGRAPH Comput. Graph.*, vol. 22, no. 4, pp. 263–268, 1988.

[9] F. K. Musgrave, C. E. Kolb, and R. S. Mace, “The synthesis and rendering of eroded fractal terrains,” *ACM SIGGRAPH Comput. Graph.*, vol. 23, no. 3, pp. 41–50, Jul. 1989.

[10] K. Nagashima, “Computer generation of eroded valley and mountain terrains,” *Vis. Comput.*, vol. 13, no. 9–10, pp. 456–464, Jan. 1998.

[11] O. Šťava, B. Beneš, M. Brisbin, and J. Křivánek, “Interactive terrain modeling using hydraulic erosion,” in *EuroGraphics Symposium on Computer Animation*, M. Gross and D. James, Eds. 2008, pp. 201–210.

[12] G. Cordonnier, J. Braun, M. Cani, B. Benes, É. Galin, A. Peytavie, and É. Guérin, “Large Scale Terrain Generation from Tectonic Uplift and Fluvial Erosion,” *Comput. Graph. Forum*, vol. 35, no. 2, pp. 165–175, May 2016.

[13] J. Brosz, F. F. Samavati, and M. C. Sousa, “Terrain Synthesis By-Example,” in *Advances in Computer Graphics and Computer Vision*, 2007, pp. 58–77.

[14] F. Belhadj and P. Audibert, “Modeling landscapes with ridges and rivers,” in *Proceedings of the 3rd international conference on Computer graphics and interactive techniques in Australasia and South East Asia - GRAPHITE ’05*, 2005, vol. 1, p. 447.

[15] H. Zhou, J. Sun, G. Turk, and J. M. Rehg, “Terrain synthesis from digital elevation models,” *IEEE Trans. Vis. Comput. Graph.*, vol. 13, no. 4, pp. 834–848, 2007.

[16] J. Gain, P. Marais, and W. Straßer, “Terrain sketching,” in *Proceedings of the 2009 symposium on Interactive 3D graphics and games - I3D ’09*, 2009, vol. 1, no. 212, p. 31.

[17] H. Hnaidi, E. Guérin, S. Akkouche, A. Peytavie, and E. Galin, “Feature based terrain generation using diffusion equation,” *Comput. Graph. Forum*, vol. 29, no. 7, pp. 2179–2186, Sep. 2010.

[18] B. Rusnell, D. Mould, and M. Eramian, “Feature-rich distance-based terrain synthesis,” *Vis. Comput.*, vol. 25, no. 5–7, pp. 573–579, May 2009.