Vis Comput (2009) 25: 573–579
DOI 10.1007/s00371-009-0332-6

ORIGINAL ARTICLE

Feature-rich distance-based terrain synthesis

Brennan Rusnell · David Mould · Mark Eramian

Published online: 3 March 2009

Abstract This paper describes a novel terrain synthesis method based on distances in a weighted graph. A height field is determined by least-cost paths in a weighted graph from a set of generator nodes. The shapes of individual terrain features, such as mountains, hills, and craters, are specified by a monotonically decreasing profile describing the cross-sectional shape of a feature. The locations of features in the terrain are specified by placing the generators; secondary ridges are placed by pathing. We show the method to be robust and easy to control, even making it possible to embed images in terrain shadows. The method can produce a wide range of realistic synthetic terrains such as mountain ranges, craters, cinder cones, and hills. The ability to manually place terrain features that incorporate multiple profiles produces heterogeneous terrains that compare favorably to existing methods.

Keywords Terrain synthesis · Natural phenomena modelling · Path planning · Pareidolia

B. Rusnell (✉) · M. Eramian
University of Saskatchewan, 110 Science Place, Saskatoon, SK S7N 5C9, Canada
e-mail: brennan.rusnell@usask.ca

M. Eramian
e-mail: eramian@cs.usask.ca

D. Mould
Carleton University, 1125 Colonel by Drive, Ottawa, ON K1S 5B6, Canada
e-mail: mould@scs.carleton.ca

1 Introduction

Synthetic terrains are widely used in applications such as computer games. Currently, fractal and physical erosion models are the dominant terrain generation methodologies. However, such methods are difficult to control and lack easy-to-use parameters. Herein we present a novel terrain synthesis method which provides the artist with improved user control over both terrain feature placement and terrain style. It also facilitates the synthesis of terrains whose shadows embed hidden images for the creation of pareidolia effects. Pareidolia refers to the phenomenon where a vague or imperfect sensory input is mistakenly interpreted as something familiar, such as a human face.

Our terrain synthesis method employs path planning [16] to produce a height field: a 2D scalar field where the field value is interpreted as vertical distance.

The advantages of the proposed method are its controls and heterogeneous results. Features are indicated by user-drawn strokes (Fig. 1) that depict the location of ridges and peaks. Feature shape is specified by cross-sectional profiles such as those in Fig. 2. Heterogeneous terrains are easy to create because a diverse set of strokes and profiles can be used in a single scene.

The paper is organized as follows. Section 2 reviews previous methods of terrain synthesis. Section 3 details the proposed path planning algorithm for terrain synthesis. Section 4 describes a method for creating pareidolia effects, and Sect. 5 shows some terrains synthesized with the proposed methodology. Section 6 contains concluding remarks and a discussion of future work.

2 Related work

Use of fractals in terrain synthesis is common, especially fractional Brownian motion (fBm) [6, 8]. Fournier introduced the midpoint displacement method [6], an alternative to fBm, which recursively subdivides an interval and generates a value at the midpoint. Prusinkiewicz and Hammel [13] noted that mountains and rivers can be expressed by similar mechanisms and were able to synthesize a mountain landscape with a river in a single process. Musgrave [5, 9, 10] noted that the statistical character of fBm surfaces is the same everywhere and developed a new synthesis model called noise synthesis which features local control over surface frequencies. Additionally, ridged multifractals (RMF) [5] is a well-known terrain synthesis method that addresses the homogeneity of fBm by producing heterogeneous terrains with valleys at varying altitudes. However, RMF cannot synthesize specific features such as craters and cinder cones. Furthermore, RMF is controlled by difficult-to-use parameters, making feature placement a tedious process. In contrast, our method allows hand-drawn feature placement.

Physically accurate erosion models also received attention. Kelley et al. simulated the erosion of stream networks [7], and Musgrave et al. simulated hydraulic and thermal erosion [9, 10]. Nagashima [11] improved upon these initial models by creating valleys and mountains as a function of erosion due to river flows, rainfall, and weathering. Chiba et al. [3] added ridge and valley lines to both fBm and midpoint displacement terrains by simulating the topography of eroded mountains based on velocity fields of water flow. Recent work has looked at increasing controllability [1], generalizing earlier work to simulate a variety of phenomena [2], and at optimizing physically accurate models for real-time applications [12].

A parallel branch of research in terrain synthesis focused on adapting texture synthesis models to height field synthesis [4]. Our method is similar to those of Zhou et al. [18], Szeliski and Terzopoulos [15], and Worley [17]. Zhou et al. presented an example-based system for terrain synthesis, adapting algorithms for nonparametric texture synthesis. In their approach, patches from an input height field are used to generate new terrain, and synthesis is guided by a user-sketched feature map. Szeliski and Terzopoulos combine variational splines and stochastic fractals to produce realistic, controllable terrains. Unfortunately, difficult-to-use parameters of a deformational energy functional are used for local control over continuity of the splines. Szeliski and Terzopoulos's work and our proposed method both use a sparse set of known elevation values and algorithmically determine the remaining elevations, in the former case by interpolating using splines, and by extrapolating using least-cost paths in the latter. Worley's cellular textures [17] use Euclidean distances from randomly positioned points as a basis for texture synthesis. In our work, distances are least-cost paths through a weighted graph and can be computed from structures that need not be points. Zhou et al.'s, Szeliski and Terzopoulos's, and our work aim to address the issue of terrain feature control. Our method uses a different underlying approach, is not limited to the features found in an input terrain, accepts hand-drawn feature placement and terrain style, and makes it easy to create heterogeneous terrains by placing many diverse features within a single scene.

3 Our algorithm

Our goal is to create a cost field that can be interpreted as a height field. The cost field is calculated by determining shortest paths in a weighted graph from a set of generator nodes. Sets of generator nodes define the individual features in the terrain; Dijkstra's algorithm is used to find shortest paths between generator nodes and non-generator nodes, and the resulting path costs define the terrain height.

The shapes of individual terrain features, such as mountains, hills, and craters, are specified by a monotonically decreasing *profile* that describes the cross-sectional shape of a feature. Sets of generator nodes specify the locations of the features in the terrain. Our algorithm uses global parameters for mean graph edge weight μ_w (controlling overall terrain steepness) and maximum edge weight deviation r (controlling overall terrain roughness), though the usage of profiles can alter their influence (Sect. 3.1). Our terrain generation method is summarized in Algorithm 1.

Algorithm 1 Terrain Synthesis Algorithm

Input: mean edge weight μ_w, maximum edge weight deviation r, generator locations g, sea level scale s, and terrain profiles p

Output: Height field

1. Initialize graph G, consisting of nodes N and edges E, with mean edge weight μ_w and maximum edge weight deviation r
2. Mark nodes at locations g as generator nodes
3. Calculate sea level cost c_s as the product of s and the maximum cost of a generator node
4. Create a scaling function for each profile in p
5. Apply Dijkstra's algorithm with frontier consisting of all generator nodes
6. Store resulting approximate cost field $C_a(n)$
7. For each individual feature f in g:
   (a) Apply Dijkstra's algorithm (using a modified search space based on $C_a(n)$ and c_s) with frontier consisting of only f's generator nodes
   (b) Blend f's height field into final height field

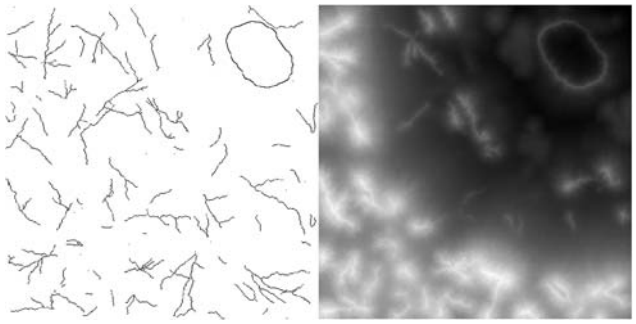

Fig. 1 Generator placement (*left*) and resulting height field (*right*) for the crater render presented in Fig. 3a

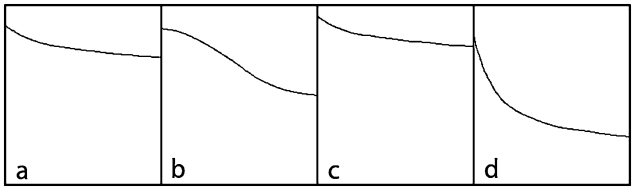

Fig. 2 The four hand-drawn profiles used in the crater render presented in Fig. 3a. From *left* to *right*: mountain, hill, exterior crater, and interior crater profiles

The nodes of a graph are arranged in an 8-connected grid corresponding to the spatial points on a height field. Edge weights w_e are given by $w_e = \mu_w + r \cdot v_r$; i.e., the weights are uniform random values with a mean of μ_w and a maximum deviation of $\pm r$ (the random variable v_r is uniformly distributed over the range $[-1, 1]$), where $0 \leq r < \mu_w$ (Step 1 of Algorithm 1).

With each graph node n we store a cost c_n, a scaling term $w_s(c)$ (a function of cost), a feature identification number f_{id}, a profile identification number p_{id}, and the generator node n_g on the least-cost path that includes n. Feature IDs must be unique to each feature, but many features may use the same profile ID. The costs of generator nodes, which govern the overall height of the feature, are user-specified. Smaller costs indicate higher elevation. The cost, c_n, of a non-generator node, n, is the cost of the shortest path to n from a generator node n_g, as determined by Dijkstra's algorithm. The cost of the ith node on the path $n_1, n_2, \ldots, n_k$ is given by $c_{n_i} = c_{n_{i-1}} + w_e \cdot w_s(c_{n_{i-1}})$, where $n_1 = n_g$, $c_{n_1} = c_{n_g}$, w_e is the weight of the edge from n_{i-1} to n_i, and $w_s(c_{n_{i-1}})$ is the node scaling value for n_i. Node scaling values are derived from the profile associated with n_g (see Sect. 3.1) and are used to enforce the shape of the profile on the landscape. When profiles are not used, $w_s(c) = 1$ for all c.

Step 2 of Algorithm 1 proceeds by setting the cost (c_n), feature ID (f_{id}), and profile ID (p_{id}) of generator nodes. Each feature's generator nodes share the same f_{id} and p_{id} value, and the locations and costs of these nodes collectively define the location and topography of the feature. For example, Fig. 1 shows the locations of generator nodes for the crater result presented in Fig. 3a. In Step 3 of Algorithm 1 the sea level cost c_s is the product of the largest cost of a generator node and s ($s \geq 0.0$), an aesthetic parameter controlling the maximum node cost.

Step 4 of Algorithm 1 creates from each user-provided profile the corresponding node scaling function $w_s(c)$ (see Sect. 3.1). Steps 5–6 create an approximate cost field $C_a(n)$ which is used to guide the termination of Dijkstra's algorithm. The approximate cost field is created by running Dijkstra's algorithm using all generator nodes (and their associated profiles) as the initial frontier. Step 7a then individually generates the height fields for each feature. Therein, Dijkstra's algorithm is halted when the cost of a node is sufficiently close to the sea level cost c_s, or is a sufficiently small fraction of $C_a(n)$. This prevents traversal of nodes with negligible contribution, saving computation. Step 7b blends together the height fields from individual features to eliminate discontinuities where features meet (see Sect. 3.2). The result is a height field such as that in Fig. 1.

3.1 Scaling by profile

Terrain profiles offer user control over feature shape by specifying local slope as a function of path cost. Profiles can be hand-drawn or procedurally generated but must be monotonically decreasing to avoid ill-defined path costs arising from negative edge weights. Figure 2 shows the four hand-drawn profiles used in the crater result presented in Fig. 3a. The profile is scaled vertically to the range $[0, c_s]$, the slope s_m of this profile is calculated as a function of cost c, and the scaling function $w_s(c) = (s_m(c)/\mu_w)$ is constructed. The scaling function replaces the slopes and heights computed from edge weights with user-sketched slopes and heights.

As Dijkstra's algorithm searches the graph, $w_s(c)$ is calculated for every visited node. However, we do not use the raw node cost as the parameter c. If we did, the upper part of the profile might never be used. Instead, the raw node cost is used to determine a new cost c_m, allowing for the inclusion of the full profile. The cost c_m used as the argument to the scaling function is calculated by linearly interpolating between zero and sea level. Thus, c_m is a function of the current node n's cost c_n, its source node n_g's cost c_{n_g}, and sea level cost c_s (1). This framework allows for the inclusion of any custom profile.

$$c_m = \frac{c_n - c_{n_g}}{c_s - c_{n_g}} \cdot c_s \tag{1}$$

3.2 Blending

Our algorithm creates an individual height field for each feature, M features total. The individual height fields are then

blended into a single height field with a modified version of the blending function proposed by Singh and Fiume [14]. The final height of a node h_f is a function of its costs c_i in the i-th feature and sea level cost c_s. We may terminate Dijkstra's algorithm with unvisited nodes; such nodes have a contribution of zero. Equation (2) gives the blending function whose behavior varies with the bias b from a simple average of the heights at each node when $b = 0$, approaching $\max\{(c_s - c_i)\}$ (the maximum height) as b approaches ∞.

$$h_f = \frac{\sum_{i=1}^{M} (c_s - c_i)^{b+1}}{\sum_{i=1}^{M} (c_s - c_i)^b} \tag{2}$$

4 Pareidolia

Our method provides sufficient control over the synthesized terrain that we can create pareidolia effects like the *face on Mars* phenomenon, where images appear in the shadows of terrains. The synthesis of such terrains is controlled by an input shadow image. We modify an initial base terrain by adding *primary terrain features* that cast the desired shadows and *secondary terrain features* that complement the primary features. With both the primary and secondary terrain features defined, Dijkstra's algorithm is used to calculate the costs of all non-generator nodes, yielding a final height field.

The inputs are a binary *shadow image* and a base terrain. The dark regions of the shadow image indicate desired shadows; we also derive *anti-shadow* regions, which show where shadows should be absent. The anti-shadow regions surround the shadow regions. We begin by choosing a light location that minimizes the maximum shadow length in the input shadow image. Next, we record the start and end locations of each shadow segment with respect to the light direction. Primary terrain features are created by placing generator nodes at the start of each shadow segment. The costs of these generator nodes are given by $c_e - m/\tan\theta$, where θ is the light tilt angle from the vertical, m is the length of the shadow segment, and c_e is the base terrain cost at the end of the shadow. The quotient $m/\tan\theta$ provides the height difference needed to cast a shadow of length m from a light tilt angle θ.

For increased realism and a more natural result, we populate otherwise vacant regions with secondary features: additional terrain features such as peaks, ridges, and hills. Secondary features are placed so as to avoid anti-shadow regions and minimize the amount of change to shadows cast by primary features. Path planning is used to construct secondary features. First, a new noncontiguous set of secondary generator nodes s_g is chosen randomly; these nodes cannot reside in anti-shadow regions. Their cost is constrained by a lower bound, intended to prevent changes to existing shadows and to avoid introducing steep, irregular features. The lower bound is $c_e - m/\tan\theta$. In shadow regions, c_e is the cost at the end of the shadow segment, and m is the distance to the end of the shadow; this prevents the placement of tall features that can extend an existing shadow. Outside shadow regions, m is the distance to, and c_e is the cost at the closest shadow segment; this provides a smooth transition to a lower bound of zero (the maximum height).

Next, another new noncontiguous set of nodes s_s is chosen randomly, differing from s_g only in location. Dijkstra's algorithm is then run using both s_g and the primary feature nodes as the initial frontier. The secondary features are defined as the least-cost paths between s_s and all previous generator nodes; the nodes of a given path P share the same f_{id} and p_{id} as the generator node from which P originates. The cost of the nodes of P can be a function of the existing node cost, or can be provided from a completely separate terrain synthesis process. In either case, each node's cost must be greater than its lower bound.

Dijkstra's algorithm is used to calculate the costs of all non-generator nodes. However, if a given node's cost exceeds the cost of the base terrain, the base terrain cost is used instead. This ensures that the cost difference between the start and end of a shadow is maintained. The final height field is computed from the final cost values.

5 Results

Figure 3 showcases some results, selected to show the controllability of the method and the diversity and realism of the features. Also note that the results are robust to differing structures within the same scene. Images were rendered with Terragen (www.planetside.co.uk/terragen) on a Pentium 4 2.80 GHz processor with 1 GB RAM. All height fields were created at a resolution of 512×512, except for Fig. 4, which was 256×256. All terrains took less than 130 s to synthesize; Fig. 4 completed in 10 s. Dijkstra's algorithm terminates when either the cost of the current node n is within 0.5 units of c_s or when the height of n is less than 5% of the height of $C_a(n)$. A summary of parameter values and synthesis time for each height field is given in Table 1.

Figure 3a shows features not commonly seen in existing methods, such as craters, and demonstrates that features

Table 1 Summary of parameters for each result in Fig. 3: mean edge weight μ_w, maximum mean edge weight deviation r, sea level scaling value s, blending bias b, number of profiles n_p, and synthesis time t

Render	μ_w	r	s	b	n_p	t
Crater	1.5	1.0	1.1	4	4	36 s
Image-based	N/A	N/A	5.0	3	0	128 s
Cinder cone	1.0	0.5	3.0	3	3	7 s
Foothills	0.75	0.375	2.5	3	3	7 s

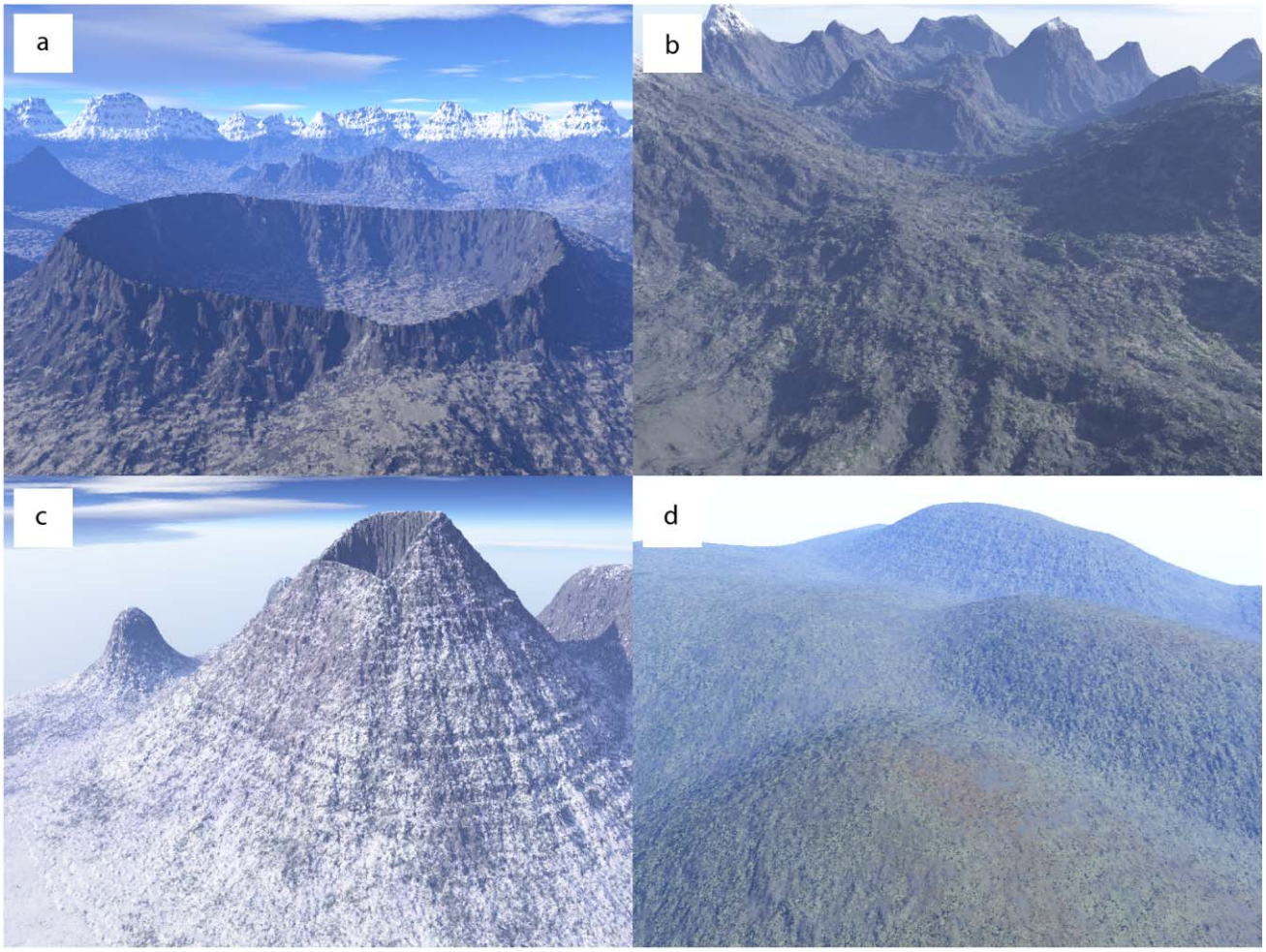

Fig. 3 Select results synthesized via the proposed methodology. Image (**a**) shows a crater and mountain range, image (**b**) shows a mountainous terrain synthesized using input textures for edge weights and terrain feature placement, image (**c**) shows a cinder cone volcano, and image (**d**) shows rolling foothills

with diverse structure and varying altitude can be included in a single terrain. The majority of the scene contains tall, rough features, requiring larger values for r, μ_w, and s. The rough appearance of the background mountain range was accomplished by densely populating the region with peaks and ridges that contain small initial costs, and using profiles with a steep slope (Fig. 2a). We wanted to fill the midground with valleys and smooth regions, and hence placed fewer features with larger costs. The low-altitude hills used a gradually sloping profile (Fig. 2b) and had costs close to sea level. The crater required two profiles: one for the smooth, rounded interior (Fig. 2d) and another for the rough, steep exterior (Fig. 2c).

The terrain in Fig. 3b was synthesized using input images for both terrain feature placement and edge weights. Intensity edges of an input image provide terrain features, and the brightness of another image determines edge weights. Using images in this way reduces the need for manual input such as feature location, μ_w, and r; we found that the most striking terrains result from relatively stochastic, high-contrast images. The relatively lengthy synthesis time results from the dense set of features derived from the input image.

The smooth exterior of the cinder cone in Fig. 3c required a small value for r, but the desired scene topography required steep input profiles. The high altitude of the cinder cone and surrounding features required a large value for s. An input profile created the smooth, rounded basin, and careful initialization of the rim's generator node costs created its sloping profile. Secondary features were subject to the same roughness and cost constraints.

Figure 3d contains rolling hillsides. The addition of this render to the results set demonstrates the diversity of the proposed method; the gentle roll of the hills contrasts the fairly jagged features seen in other images. The hills were synthesized using low values for r and μ_w; a larger value for s helped define the local minima between hills. Three different gradually sloping input profiles provided some variation between the features.

Fig. 4 A synthesized terrain whose shadows contain the word *TEXT* (*top*) and the same scene with different camera and light placement (*bottom*), making the embedded word difficult to see

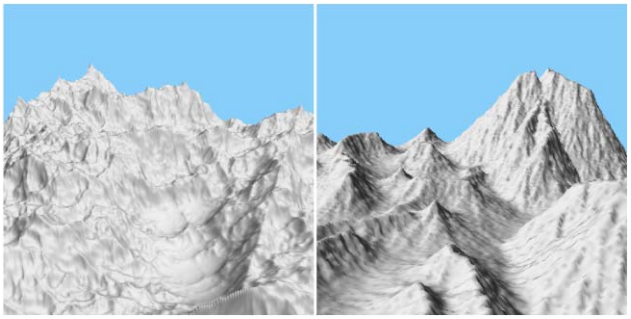

Fig. 5 A comparison of a terrain synthesized by the ridged multifractal process (*left*) and a terrain synthesized by the proposed method (*right*)

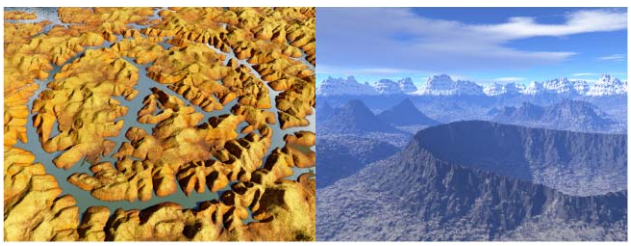

Fig. 6 A comparison of a terrain synthesized by Zhou et al.'s method (*left*) and a terrain synthesized by the proposed method (*right*)

Figure 4 shows an example of pareidolia synthesized according to the process given in Sect. 4. The word *TEXT* is embedded in the shadows. The top image of Fig. 4 shows the terrain from an aerial perspective, making the embedded word easy to see. The bottom image shows the same terrain with different camera and light placement, where the embedded word is difficult to see.

Figure 5 shows a closeup comparison of an RMF height field (left) and a terrain synthesized via the proposed method (right). This comparison, along with Fig. 3, emphasizes the ability of our method to create extended features such as ridges and specific features such as craters and cinder cones. Ridged multifractals allow a more limited range of features.

Figure 6 shows a comparison of Zhou et al.'s method (left) and a terrain synthesized via the proposed method (right). While the left-hand terrain is rather homogeneous, the right-hand terrain shows diverse features. It is worth noting that Zhou et al.'s method cannot introduce new small-scale features not present in the input texture, whereas the proposed method allows new features through the addition of a new feature stroke and profile.

6 Conclusions and future work

We presented a new, robust terrain synthesis method, capable of generating a wide variety of features with straightforward user control over feature shape and location. Future work could address alternative profile generation techniques and the incorporation of bodies of water such as lakes and rivers.

Alternative profile generation techniques could include the development of new functions for edge weights, including functions that are dependent on node location, edge direction, current node attributes such as f_{id}, and/or edge weight w_e. Bodies of water could be handled by halting the recursion of Dijkstra's algorithm when a body is reached. A separate pass of Dijkstra's algorithm with generator nodes defining the shoreline of the body (similar to crater construction) would provide its costs and would be needed to avoid the effects of blending. In summary, the proposed method is robust and easy to control. The resulting terrains are realistic and compare favorably to those created by existing methods.

References

1. Benes, B., Forsbach, R.: Visual simulation of hydraulic erosion. In: WSCG, pp. 79–94 (2002)
2. Benes, B., Tesinsky, V., Hornys, J., Bhatia, S.K.: Hydraulic erosion. Comput. Animat. Virtual Worlds **17**(2), 99–108 (2006)
3. Chiba, N., Muraoka, K., Fujita, K.: An erosion model based on velocity fields for the visual simulation of mountain scenery. J. Vis. Comput. Animat. **9**(4), 185–194 (1998)
4. Dachsbacher, C.: Interactive terrain rendering—towards realism with procedural models and graphics hardware. Ph.D. thesis, University of Erlangen-Nuremberg (2006)
5. Ebert, D.S., Musgrave, F.K., Peachey, D., Perlin, K., Worley, S.: Texturing and Modeling: A Procedural Approach, 3rd edn. Morgan Kaufmann, San Mateo (2002)
6. Fournier, A., Fussell, D., Carpenter, L.: Computer rendering of stochastic models. Commun. ACM **25**(6), 371–384 (1982)
7. Kelley, A.D., Malin, M.C., Nielson, G.M.: Terrain simulation using a model of stream erosion. In: Proceedings of SIGGRAPH '88, pp. 263–268 (1988)
8. Mandelbrot, B.B.: The Fractal Geometry of Nature. Freeman, New York (1982)
9. Musgrave, F.K.: Methods for realistic landscape imaging. Ph.D. thesis, Yale University (1993)
10. Musgrave, F.K., Musgrave, F.K., Kolb, C.E., Mace, R.S.: The synthesis and rendering of eroded fractal terrains. In: Proceedings of SIGGRAPH '89, pp. 41–50 (1989)
11. Nagashima, K.: Computer generation of eroded valley and mountain terrains. Vis. Comput. **13**(9–10), 456–464 (1997)
12. Neidhold, B., Wacker, M., Deussen, O.: Interactive physically based fluid and erosion simulation. In: Eurographics Workshop on Natural Phenomena, pp. 25–32 (2005)
13. Prusinkiewicz, P., Hammel, M.: A fractal model of mountains with rivers. In: Proceeding of Graphics Interface '93, pp. 174–180 (1993)
14. Singh, K., Fiume, E.: Wires: A geometric deformation technique. In: Proceedings of SIGGRAPH '98, pp. 405–414 (1998)
15. Szeliski, R., Terzopoulos, D.: From splines to fractals. SIGGRAPH Comput. Graph. **23**, 51–60 (1989)
16. Winston, P.H.: Artificial Intelligence, 3rd edn. Addison–Wesley, Reading (1992)
17. Worley, S.: A cellular texture basis function. In: Proceedings of SIGGRAPH '96, pp. 291–294 (1996)
18. Zhou, H., Sun, J., Turk, G., Rehg, J.M.: Terrain synthesis from digital elevation models. IEEE Trans. Vis. Comput. Graph. **13**(4), 834–848 (2007)

Brennan Rusnell received his Honours B.Sc. with High Distinction in Computer Science in 2006. Brennan is currently a M.Sc. student at the University of Saskatchewan. He is supervised by Dr. David Mould and Dr. Mark Eramian. He is interested in procedural modelling, image processing, and non-photorealistic rendering.

David Mould received his Ph.D. from the University of Toronto in 2002. Dr. Mould is currently an Assistant Professor in the School of Computer Science at Carlton University. He is also an Adjunct Professor in Computer Science at the University of Saskatchewan. He is interested in procedural modelling, image processing, and non-photorealistic rendering.

Mark Eramian received his B.Sc. in Computer Science and Geophysics in 1997 and his Ph.D. in Computer Science in 2002, both from the University of Western Ontario, Canada. Dr. Eramian is currently an Associate Professor in the department of Computer Science and an associate member of the Division of Biomedical Engineering at the University of Saskatchewan, Canada. His research interests are in the area of image processing and computer vision, particularly medical image analysis and computer assisted diagnosis.