DOI: 10.1111/j.1467-8659.2010.01806.x

Pacific Graphics 2010
P. Alliez, K. Bala, and K. Zhou
(Guest Editors)

Volume 29 (2010), Number 7

Feature based terrain generation using diffusion equation

Houssam Hnaidi[1], Eric Guérin[1], Samir Akkouche[1], Adrien Peytavie[1], Eric Galin[2]

[1]LIRIS - CNRS - Université Lyon 1, France [2]LIRIS - CNRS - Université Lyon 2, France

Abstract

This paper presents a diffusion method for generating terrains from a set of parameterized curves that characterize the landform features such as ridge lines, riverbeds or cliffs. Our approach provides the user with an intuitive vector-based feature-oriented control over the terrain. Different types of constraints (such as elevation, slope angle and roughness) can be attached to the curves so as to define the shape of the terrain. The terrain is generated from the curve representation by using an efficient multigrid diffusion algorithm. The algorithm can be efficiently implemented on the GPU, which allows the user to interactively create a vast variety of landscapes.

Categories and Subject Descriptors (according to ACM CCS): Computer Graphics [I.3.7]: Three-Dimensional Graphics and Realism—

Keywords: Terrain modeling, vector graphics, gradient and height field diffusion.

1. Introduction

There are numerous applications that make use of synthetic terrain: landscape design, flight simulators, battleground simulations, feature film special effects and computer games. Very often the terrain is either the dominant visual element or plays a central part in the application.

Over the years, researchers have made considerable progress towards developing efficient methods for generating and editing synthetic terrains. There are different approaches to generating synthetic terrains. Fractal landscape modeling and physical erosion simulation methods can automatically generate realistic terrain models, but often lack intuitive tools for controlling the placement and the shape of the landforms features. In contrast, terrain generation techniques from features [RME09] or images [ZSTR07] as well as sketching approaches [GMS09] provide a better and more intuitive control over the resulting terrain.

Existing constraint based editing techniques have several weaknesses however. In general, the characterization of the height field in the vicinity of the constraints is either random or difficult to control [GMS09, RME09] or depends on the characteristics of another terrain image [ZSTR07].

In this paper, we present a compact vector-based model to efficiently and accurately control the terrain generation process. The terrain is characterized by a set of control curves representing landform features such as ridge lines, cliffs or river beds. The control curves define the local elevation and slope of the terrain, as well as some parameters characterizing the roughness of its surface. The terrain is generated to match the elevation and gradient constraints attached to the curves using a multigrid diffusion equation [OBW*08]. Details are created by a parameterized noise function controlled by the diffused amplitude and roughness attributes. Therefore, our method allows to finely tune the parameters of the terrain by setting tangent, elevation and noise constraints in the neighborhood of user-controlled feature curves.

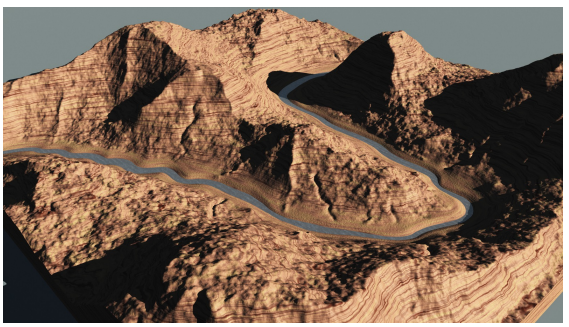

Figure 1: *A complex terrain generated with our method*

© 2010 The Author(s)
Journal compilation © 2010 The Eurographics Association and Blackwell Publishing Ltd.
Published by Blackwell Publishing, 9600 Garsington Road, Oxford OX4 2DQ, UK and 350 Main Street, Malden, MA 02148, USA.

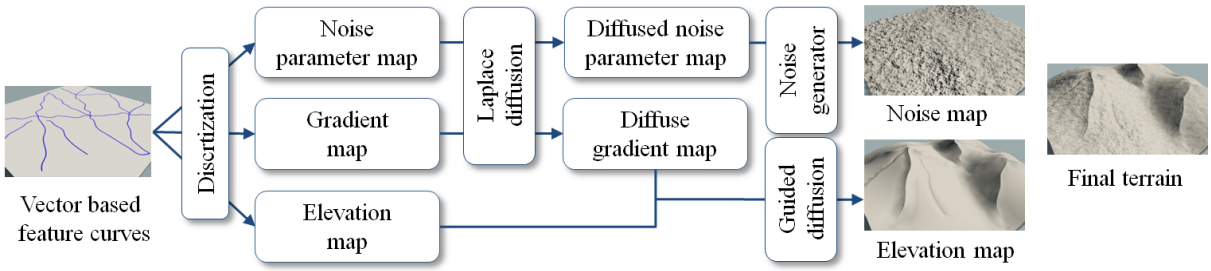

Figure 2: *Generating the terrain requires: the rasterization of the vector based constraints into roughness, elevation and gradient maps, the diffusion of elevation, gradient and noise parameters, the synthesis of the noise and smooth height fields and the final combination of the two elevation maps.*

Our paper makes the following major contributions. Firstly, we describe a compact representation to control the procedural terrain generation (Section 3). Secondly, we present an automatic generation algorithm based on a multigrid diffusion algorithm (Section 4). The generated terrain fits the elevation and gradient constraints attached to the featured curves. Moreover our approach allows us to create landform features without ridge lines such as hills. We show that our method can be accelerated by implementing the multigrid parameter and elevation diffusion algorithm on the GPU (Section 5). Finally, we show that our approach provides a seamless framework for authoring and generating terrains at different resolutions (Section 6). Our vector-based representation of control curves allows us not only to sketch terrains with a very limited number of curves, but also to incrementally add as many details as needed by refining and adding new constraints to the curves.

2. Related work

There are different approaches to generating synthetic terrains: fractal landscape modeling, physical erosion simulation and terrain synthesis from landform features or sample terrain patches.

Fractal based methods: Several stochastic subdivision techniques, such as random midpoint displacement [FFC82] or square-square subdivision schemes [Mil86, Lew87] were first proposed to generate artificial terrains. Those methods produce realistic terrains but suffer from a lack of control. [ST89] addressed this problem by combining deterministic splines and stochastic fractals into constrained fractals. An overview may be found in [EMP*98]. A midpoint displacement based method was introduced in [Bel07] with several techniques to set local constraints and generate the terrain. With the recent advances in algorithms and graphics hardware, GPU methods for real-time editing and rendering of procedural terrains have been proposed [SBW06].

Erosion simulation: Erosion simulation is another approach to synthesizing terrains based on models of landscape formation and stream erosion. It is often used as a refinement step after a height field is generated. With appropriately-tuned parameters, these techniques can generate realistic terrains. Early methods adapted the fractal rules with specific changes to produce eroded terrains [KMN88, PH93]. Physically-based erosion approaches approximate natural terrain formation by simulating the erosion of stream networks. Some material is dissolved and transported by the water flow, and finally deposited at another location. The water movement is determined by the local gradient of the terrain in a simple diffusion algorithm [RPP93, Nag98, BF02].

Those methods are based on a simple diffusion model which is not accurate enough to describe the water movement and sediment transportation. More complex and accurate fluid simulation methods were integrated into the erosion process [CMF98]. Recently, hydraulic erosion simulation techniques have been optimized so as to take full advantage of the high parallel computing power offered by today's graphics processing units [NWD05, MDH07, SBBK08].

Controlled editing and sketching techniques: both procedural and physical erosion techniques require complex parameter tuning to obtain specific terrains. Therefore, several image-based editing [Lew84, PV95] and constraint based techniques have been proposed to provide the user with more intuitive control over the synthesized terrain. Material maps were also proposed to sculpt three-dimensional terrains featuring arches, overhangs or even caves [PGMG09].

An example-based system for terrain synthesis was proposed in [ZSTR07]. By combining patches from a sample image and a user-sketched feature map, the method can synthesize realistic terrains mixing the desired terrain features with the details of the sample height field. Another sketching method was introduced in [GMS09] to interactively generate a terrain that matches the sketched constraints using multi resolution surface deformation. [RME09] presented an original terrain synthesis method based on the computation of distances in a weighted graph. The height field is computed by a least-cost path algorithm in a weighted graph

© 2010 The Author(s)
Journal compilation © 2010 The Eurographics Association and Blackwell Publishing Ltd.

from a set of generator nodes. Terrain features such as mountains, hills or craters are specified by monotonically decreasing profile curves which define their cross-sections.

3. Terrain modeling primitives

In this section, we present the control curves that are used to describe and control the shape of landforms in the scene. There is a vast variety of landforms in nature, which makes them difficult to model in a consistent way. We propose to use vector-based control curves as a generic modeling primitive for representing a vast variety of landform features, including single or multiple ridge lines, riverbeds, hills, cracks and cliffs. Given a set of control curves with the corresponding elevation, gradient and noise constraint parameters attached to them, the overall generation process proceeds as follows (Figure 2):

- Computation of the elevation, gradient, noise amplitude and roughness constraint maps by a rasterization step.
- Diffusion of the noise, gradient and elevation constraints in the maps using Laplace diffusion.
- Computation of a smooth elevation map using a guided diffusion controlled by the diffused gradient map.
- Generation of the detail map of the terrain using the diffused noise parameters.

The final height field representing the terrain is obtained by adding the smooth elevation map and the detail map.

3.1. Feature curves

Feature curves, denoted as C, are defined as two-dimensional vector-based piecewise Bézier cubic splines (Figure 3). We attach a set of constraint points $\mathcal{P}_i$ to every curve C. Every constraint $\mathcal{P}_i$ is defined by the following attributes:

$$\mathcal{P}_i = ((h_i, r_i), (a_i, b_i, \theta_i, \varphi_i), (\mathcal{A}_i, \mathcal{R}_i), u_i)\ i \in [1, m]$$

u_i denotes the linear coordinate along the spline. It is not compulsory to attach all the types of constraints to a curve. Constraints are grouped into three categories: elevation constraints (h_i, r_i), angle constraints $(a_i, b_i, \theta_i, \varphi_i)$ and noise constraints $(\mathcal{A}_i, \mathcal{R}_i)$. θ_i and φ_i refer to the slope angle whereas a_i and b_i denote the extent of the slope constraint on both sides of the curve (Figure 3). h_i represents the height constraint and r_i denotes the radius of the plateau on both sides of the feature curve. Noise constraints are used to control the noise generator parameters. Our noise generator is controlled by two parameters: amplitude denoted $\mathcal{A}$ and roughness denoted $\mathcal{R}$.

The user can choose to attach only one category, or two or all the categories of constraints to the curve. Figure 3 shows an example of a feature curve defined in (x, y) plane with four control points. The first and the last control points of the curve are constraint points and we have put another constraint point along the curve at $u = 0.3$. The Bézier spline curve is used to define the projection of a ridge line or a riverbed on the (x, y) plane whereas constraints are used to define elevations, gradients, and noise parameters along it.

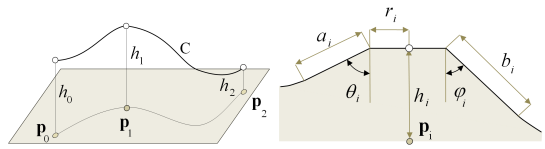

Figure 3: *Control curve and associated geometric constraints (elevation and angle)*

Two constraint points at least are attached to a curve, at extremities. Between constraint points, each attribute is linearly interpolated along the curve except for elevation h_i which is interpolated with a cubic spline to avoid unnatural piecewise linear ridge lines.

3.2. Terrain modeling

Feature curves provide the user with a generic and efficient modeling tool for defining and controlling the landforms of the terrain (Figure 4).

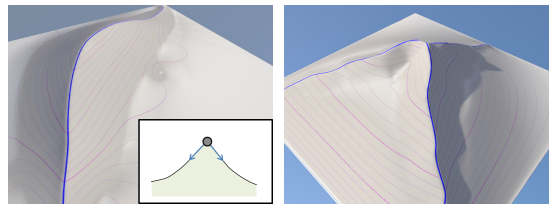

Figure 5: *Ridge line feature curve (left) and multiple ridge lines (right)*

Ridge line: A ridge line can be obtained by a single feature curve (Figure 5 left). Elevation constraints indicate altitudes on the ridge line while slope angle constraints describe the degree of inclination at each side. Small elevation constraint radius are often chosen in this case.

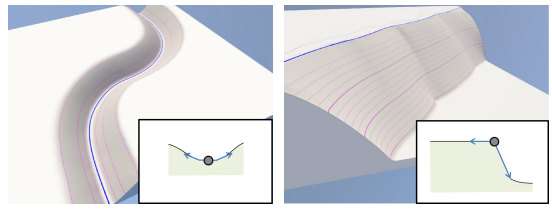

Figure 6: *Riverbed (left) and cliff (right) feature curves*

© 2010 The Author(s)
Journal compilation © 2010 The Eurographics Association and Blackwell Publishing Ltd.

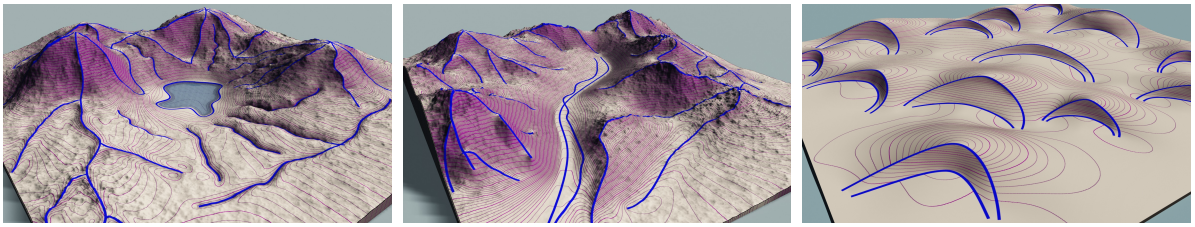

Figure 4: *Different kinds of landscapes (lake, mountain and desert) created with* 45*,* 59 *and* 26 *control curves respectively.*

Multiple ridge line: Sometimes, a mountain does not have a single ridge line. In our system, it is possible to combine multiple ridge lines that have a common peak point with the same elevation constraint and whose direction and slope angle constraints are different (Figure 5).

Riverbed: A riverbed can also be obtained by a single feature curve (Figure 6 left). Elevation constraints have a small range and elevation constraint radius indicates the width of the riverbed. Slope angle constraints are slightly upwards with a small associated radius.

Cliff: A cliff is formed with a single feature line combining two radically opposite slope angle constraints at each side (Figure 6 right).

Hill: A hill is obtained by setting horizontal slope angle constraints at each side of the feature curve (Figure 7 left). This constraint implies a null gradient and thus forces points in the neighborhood to be at the same elevation.

Cracks: A crack is obtained by setting two rising slope angle constraints at each side of the feature curve (Figure 7 right). In this case, there is no need for setting strict elevation constraint.

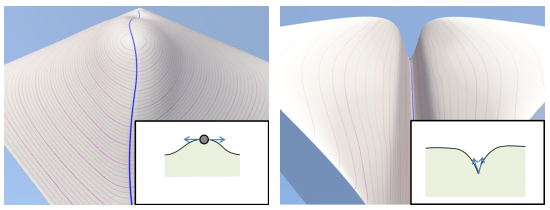

Figure 7: *Hill (left) and crack (right) feature curves*

Noise control: Two parameters allow the definition of constraints on amplitude and roughness of the noise. Figure 8 shows the influence of the amplitude parameter (left) and the roughness one (right). In this example, only noise constraints have been used.

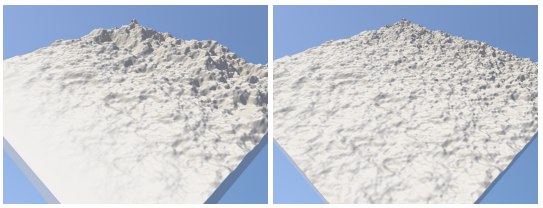

Figure 8: *Amplitude and roughness control of noise*

4. Terrain generation algorithm

In this section, we present the generation of the height field from the set of user-defined feature curves.

4.1. Rasterization

User-defined feature curves are vector graphics that are resolution-independent. The diffusion method relies on a raster representation of data. Therefore, a rasterization preprocessing is compulsory to represent constraints (elevation, noise parameters, slope angle) onto images.

The curve is first discretized along it and approximated into piecewise linear parts. At each linear coordinate, the whole parameter set is linearly interpolated from constraints points except for the elevation constraint where a cubic interpolation is used to avoid unnatural piecewise linear ridge lines on mountains. Then, at each side, quadrangles are generated in the direction of the normal whose lengths are set from the interpolated radii values r, a and b.

The colors on the vertices of the quadrangles are drawn from the interpolated constraint values. For strict constraints, quadrangles are painted with uniform values of h, $\mathcal{A}$ and $\mathcal{R}$. For slope angle primitives, the vertex color is set to its corresponding interpolated value along the curve and is set to 0 at the end of the quadrangle so as to avoid gradient discontinuities and artifacts. Gradient information is represented as a combination of the normal direction to the feature curve and the norm of the gradient. These values are inverted when the feature curve does not contain strict elevation constraints. In this way, the propagation of the gradient behaves better and requires less iterations in the multigrid solver.

© 2010 The Author(s)
Journal compilation © 2010 The Eurographics Association and Blackwell Publishing Ltd.

Feature curves can be placed anywhere so they can intersect themselves. Strict constraints (noise parameters and elevation) intersection can be solved automatically by averaging all the constraint values. However, gradient intersection is a more complex problem because conflicting gradients can have completely antagonistic directions. This problem is even harder when the number of intersecting gradients increases.

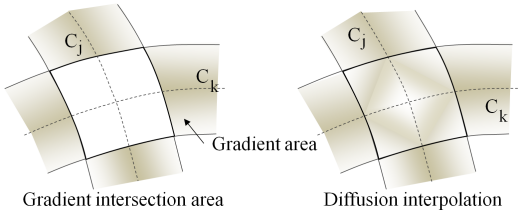

Figure 9: *Gradient intersection problem resolution*

In order to have a generic solution to this problem, we leave intersecting areas empty and perform a Laplace diffusion to fill in holes. Figure 9 illustrates the intersection problem when two feature curves C_j and C_k are intersecting. The diffusion method naturally tends to smooth the gradient field (Figure 9 right) and eliminates discontinuities.

4.2. Diffusion

Our diffusion method is based on [OBW*08]. Other diffusion methods for image editing were introduced in [MP08, PGB03]. Although these methods offer very effective image editing tools, they use a pixel based representation not suitable in the context of terrain generation. As the methods cited above, our method uses a multigrid solver to solve diffusion equation with an intensive use of the GPU capabilities. Several papers present such techniques for exploiting the GPU in this context [GWL*03, BFGS03].

Partial differential equations: We use several types of equations: Laplace (order 2), gradient (order 1) and identification (order 0). The first one is used when no information on gradient is provided. The Laplacian of the field F is simply vanishing: $\Delta F = 0$, where F is a scalar field defined in $\mathbb{R}^2$. This equation is used to express that the field is smooth.

When we want to guide the diffusion by a gradient direction drawn from the slope angle constraints, we use this gradient equation: $\nabla F = \mathbf{G}$, where $\mathbf{G} = (G_x, G_y)$ is the gradient vector field we want to impose. Note that we do not use Poisson partial differential equations because this would require the computing of the divergence of the gradient. This operation induces the loose of the gradient direction and is not suitable in our case. Additionally, Poisson equation simplifies to a Laplace equation when we set a null gradient constraint and thus forbids the creation of horizontal angle constraints.

When we want to set directly a constraint on the value of F we use a direct identification equation: $F = \mathcal{F}$.

To solve diffusion equations, we need firstly to discretize them on a rectangular 2D grid. For this end we use a forward Euler integration scheme.

Because we want a tight control over the result, several equations can be concerned at the same point. This over-constrained system is solved using a combination of Jacobi relaxations terms in the iterative solver (see section 5.2).

4.3. Noise generation

Any kind of noise generator can be plugged into our system provided that it can be driven by two scalar fields: amplitude $A(x,y)$ and roughness $R(x,y)$. Mapping the roughness parameter directly on a Perlin noise generator through the frequency and feature size parameters breaks the continuity and gives unnatural ring effects. In our method we generate a multi fractal noise based on a Perlin noise $S(x,y)$:

$$N(x,y) = A(x,y) \sum_{k=0}^{n} \frac{1}{r^{k(1-R(x,y))}} S(r^k x, r^k y)$$

where $n+1$ determines the number of octaves and r is the lacunarity. In our implementation, we use a 4 octaves multi fractal noise generation ($n = 3$ and $r = 2$). Our Perlin noise is based on a procedural deterministic noise generator. The roughness coefficient R does not influence the different scales of the multi fractal combination, it only changes the way they are blended. When setting $R = 1$, the different scales are blended equally, whereas lower values of R will damp the blending coefficient of higher frequencies.

5. Implementation details

In order to reach interactive generation performances, our implementation makes intensive use of the GPU. This section explains how we represent data with textures and the different shaders we use.

5.1. Data representation

All the data involved in our method after rasterization can be represented by images. These images are processed as textures by fragment shaders.

Several constraint maps can be multiplexed on a single texture using the different components. Elevation and noise parameters constraints are gathered into the RGB components of a single RGBA texture image. The last alpha component of the texture is used to indicate which constraints have been set on the different areas.

In the remainder in this paper $\mathbf{n} = (\mathbf{n}_x, \mathbf{n}_y)$ will refer to the normalized normal direction to the feature curve. The components of $\mathbf{n}$ and the gradient norm are stored in the red, green and blue components of another RGB texture.

© 2010 The Author(s)
Journal compilation © 2010 The Eurographics Association and Blackwell Publishing Ltd.

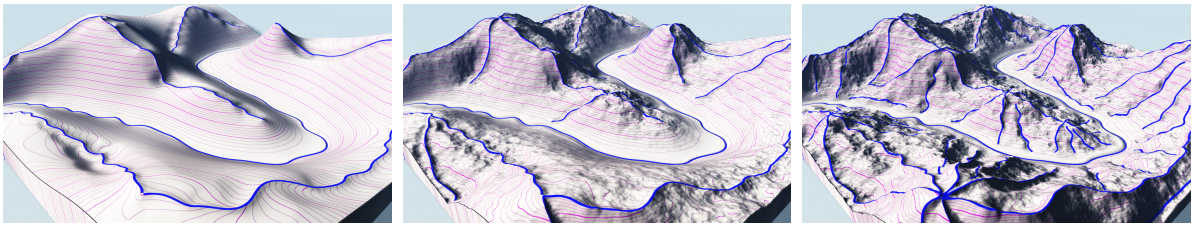

Figure 10: *Incremental editing of a canyon, with* 15*,* 30*, and* 74 *control curves.*

5.2. Multigrid implementation

We use a multigrid algorithm to solve the linear system obtained by discretization. Method from [OBW*08] is used and adapted to our problem for solving the multigrid. This algorithm is computed in interactive time for a square grid at a resolution range from 512^2 to 2048^2 with the aid of GPU capabilities. Multigrid method consists in solving a coarse version of the system and obtaining low frequency solution components, and refining the domain to adjust higher frequency components. A Jacobi relaxation is used to solve each multigrid level, and we limit the number of relaxation iterations to reach good performances. Let l denote the number of grid levels in the multigrid structure, 0 will refer to the coarsest level and $l-1$ the finest level. At each level i, we make $5(l-i)$ iterations in the Jacobi relaxation process as advocated in [OBW*08].

Let $0 \leq \alpha \leq 1$, $0 \leq \beta \leq 1$ and $0 \leq \alpha+\beta \leq 1$. The Jacobi relaxation term is calculated by a weighted combination of the three types of equations:

$$\begin{aligned} F^{k+1}(i,j) = \alpha F_L^{k+1}(i,j) \quad &+ \quad \beta F_G^{k+1}(i,j) \\ &+ \quad (1-\alpha-\beta) F_I^{k+1}(i,j) \end{aligned}$$

Where each term is calculated as follows:

1. Laplace equation. This equation is locally linearized by averaging the neighborhood points:

$$\begin{aligned} F_L^{k+1}(i,j) \quad = \quad & \tfrac{1}{4}\Big(F^k(i+1,j)+F^k(i,j+1) \\ & +F^k(i-1,j)+F^k(i,j-1)\Big) \end{aligned}$$

2. Gradient equation. The norm of the gradient is added to the gradient direction neighbor:

$$F_G^{k+1}(i,j) = F_N^k(i,j) + G(i,j)$$

The neighbor value F_N is calculated by combining the two points in the normal direction of the feature curve:

$$F_N(i,j) = \mathbf{n}_x^2 F^k(i-\text{sign}(\mathbf{n}_x),j) + \mathbf{n}_y^2 F^k(i,j-\text{sign}(\mathbf{n}_y))$$

3. Identification. This is the simplest equation:

$$F_I^{k+1}(i,j) = \mathcal{F}(i,j)$$

Sometimes, points in the grid can be concerned by several equations simultaneously. For example, in gradient constraint areas, we set $\alpha = \beta = 1/2$. In this way, Jacobi iterations tend to satisfy both the gradient equation and the Laplace equation at the same time. In strict elevation constraint areas, we set $\alpha = \beta = 0$. If we want to approximate (and not interpolate) elevation constraints, we can set $\alpha > 0$ and the solution will be smoother. But this breaks edges on features and not suitable in our case. Everywhere else, we set $\alpha = 1$ and $\beta = 0$.

Five scalar fields have to be calculated with a standard Laplace diffusion algorithm: the amplitude of the noise $\mathcal{A}$, the roughness of the noise $\mathcal{R}$, the gradient norm and the components of n. The scalar field that represents the altitude h is calculated by using a guided diffusion algorithm. α and β values are determined by feature curves and constraints positions.

We first diffuse the gradient norm and the components of n in order to fill in holes in intersecting gradient areas. Then the diffused gradient is used to diffuse the altitude h. The noise parameters $\mathcal{R}$ and $\mathcal{A}$ are diffused in the same shader for performance issue because they are algorithmically identical. The only difference is that the guided diffusion has to calculate the gradient direction neighbor. Altitude and noise parameters are multiplexed on the different components of a single texture image.

5.3. Noise generation

The input signal generator is based on a bilinearly interpolated Perlin noise generated procedurally on the GPU. To guarantee that this signal is deterministic and gives always the same result on a given (x,y) input, we compute pseudo-random numbers with bit operations on x and y.

6. Results

We have implemented our method into a modeling application coded in C++. All the images shown throughout the paper were created using this application. Renderings were performed using Mental Ray on the procedurally textured meshed produced by our technique.

Realism Our method creates realistic landscapes and compares favorably in terms of efficiency and quality to recent existing editing and sketching techniques [GMS09,

© 2010 The Author(s)
Journal compilation © 2010 The Eurographics Association and Blackwell Publishing Ltd.

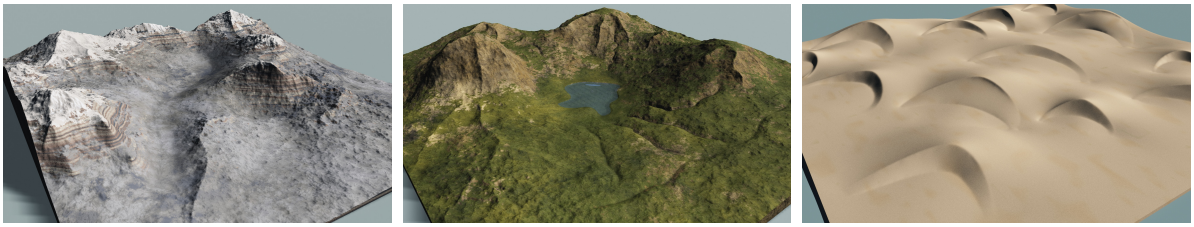

Figure 11: *Final mountain, lake and desert scenes rendered with procedural textures.*

RME09]. The main reason for this is that our approach provides a good control over the landform features. Moreover, control curves are a generic tool which can be used to create a vast variety of different kinds of terrains, such as volcanic islands (Figure 12), mountain and desert sceneries (Figure 11), canyons with complex riverbeds (Figure 1) and even hilly landscapes.

Model	Features	Size
Mountain	59	2.89
Lake	45	2.26
Canyon	74	4.48
Hills	55	2.25
Volcanic Island	48	2.95
Desert	26	1.30

Table 1: *Terrain statistics: number of feature curves and data structure size (in kB).*

Control A very interesting and powerful aspect of our approach is its simplicity and control. Our vector based model provides the user with a compact and resolution independent representation which can be edited and manipulated easily. Table 1 reports some statistics corresponding to the terrains shown throughout the paper.

Another interesting feature of our method is that is provides a seamless framework that bridges the gap between sketching and accurate editing for authoring complex terrains. Figure 10 illustrates this ability with three different steps in the creation of a canyon. The first sketch was created in less than 3 minutes. It took almost 45 minutes to carefully edit all the details needed to produce the final model. In that particular case, most of the time was spent tuning the noise parameters to adapt the roughness of the terrain to the slope. A straight forward improvement to speed up the design of the scene would consist in directly computing the amplitude and roughness parameters from the slope constraints.

Efficiency We implemented our algorithms as GLSL shader on an NVidia 8800 GTS graphic card. In terms of memory, our algorithm requires 10 grids of size $n \times n$. In our system, those grids are stored as 32 bits floating point textures. Thus, for a 1024×1024 grid size, we use 40MB of GPU memory. Table 1 and 2 report various statistics as well as timings for generating terrains at different resolutions. Figures demonstrate that the number of feature curves has a small influence over the overall terrain generation time, whereas the grid size has a strong influence over the generation time. Timings show that our method runs at interactive rates, even for large grid sizes.

Model	Computing Time		
Grid resolution	512^2	1024^2	2048^2
Mountain	0.187	0.270	0.670
Lake	0.185	0.261	0.690
Canyon	0.216	0.339	0.811
Hills	0.188	0.266	0.611
Volcanic Island	0.168	0.275	0.675
Desert	0.171	0.253	0.635

Table 2: *Timings (in seconds) for generating the terrain at* 512^2, 1024^2 *and* 2048^2 *resolution.*

All the rendered sceneries were generated with a 1024×1024 grid resolution.

Comparison with other techniques Compared to [GMS09], the terrain characteristics are incrementally edited but not stored. In contrast, our method relies on an underlining vector based representation and generation algorithms which enable us to create original features such as rivers or multiple ridge lines. Moreover, our multigrid GPU implementation compares favorably to existing methods.

7. Conclusion

In this paper we introduced a new method based on a multigrid diffusion of constraints for procedurally generating realistic terrains. Our approach provides the user with simple, intuitive and efficient tools for controlling the landforms of the terrain with a reduced number of feature curves. Remarkably, feature curves can handle the definition of ridge

© 2010 The Author(s)
Journal compilation © 2010 The Eurographics Association and Blackwell Publishing Ltd.

Figure 12: *A volcanic island and the corresponding feature curves.*

lines as well as river beds, lake and sea shores in a consistent and generic ways. Moreover, our method can also generate smooth hills and valleys very easily. Our optimized GPU implementation allows the generation of complex terrains in interactive time, even at large resolutions (2048×2048). Our technique lends itself for both real time sketching and editing of highly detailed terrain models.

References

[Bel07] BELHADJ F.: Terrain modeling: a constrained fractal model. In *Afrigraph* (2007), pp. 197–204.

[BF02] BENES B., FORSBACH R.: Visual simulation of hydraulic erosion. In *Journal of WSCG* (2002), vol. 10, pp. 79–86.

[BFGS03] BOLZ J., FARMER I., GRINSPUN E., SCHRÖDER P.: Sparse matrix solvers on the GPU: conjugate gradients and multigrid. *ACM Trans. Graph. 22*, 3 (2003), 917–924.

[CMF98] CHIBA N., MURAOKA K., FUJITA K.: An erosion model based on velocity fields for the visual simulation of mountain scenery. *Journal of Visualization and Computer Animation 9*, 4 (1998), 185–194.

[EMP*98] EBERT D., MUSGRAVE K., PEACHEY D., PERLIN K., WORLEY S.: *Texturing and Modeling: A Procedural Approach*. Academic Press Professional, 1998.

[FFC82] FOURNIER A., FUSSELL D. S., CARPENTER L. C.: Computer rendering of stochastic models. *Commun. ACM 25*, 6 (1982), 371–384.

[GMS09] GAIN J., MARAIS P., STRASSER W.: Terrain sketching. In *Proceedings of the 2009 symposium on Interactive 3D graphics and games* (2009), pp. 31–38.

[GWL*03] GOODNIGHT N., WOOLLEY C., LEWIN G., LUEBKE D. P., HUMPHREYS G.: A multigrid solver for boundary value problems using programmable graphics hardware. In *Graphics Hardware* (2003), pp. 102–111.

[KMN88] KELLEY A., MALIN M., NIELSON G. M.: Terrain simulation using a model of stream erosion. In *Proceedings of SIGGRAPH* (1988), pp. 263–268.

[Lew84] LEWIS J.-P.: Texture synthesis for digital painting. In *Proceedings of SIGGRAPH* (1984), pp. 245–252.

[Lew87] LEWIS J. P.: Generalized stochastic subdivision. *ACM Trans. Graph. 6*, 3 (1987), 167–190.

[MDH07] MEI X., DECAUDIN P., HU B.: Fast hydraulic erosion simulation and visualization on GPU. In *Pacific Graphics* (2007), pp. 47–56.

[Mil86] MILLER G. S. P.: The definition and rendering of terrain maps. In *SIGGRAPH* (1986), pp. 39–48.

[MP08] MCCANN J., POLLARD N. S.: Real-time gradient-domain painting. *ACM Trans. Graph. 27*, 3 (2008).

[Nag98] NAGASHIMA K.: Computer generation of eroded valley and mountain terrains. *The Visual Computer 13*, 9-10 (1998), 456–464.

[NWD05] NEIDHOLD B., WACKER M., DEUSSEN O.: Interactive physically based fluid and erosion simulation. In *Eurographics Workshop on Natural Phenomena* (2005), pp. 25–32.

[OBW*08] ORZAN A., BOUSSEAU A., WINNEMÖLLER H., BARLA P., THOLLOT J., SALESIN D.: Diffusion curves: a vector representation for smooth-shaded images. *ACM Trans. Graph. 27*, 3 (2008).

[PGB03] PÉREZ P., GANGNET M., BLAKE A.: Poisson image editing. *ACM Trans. Graph. 22*, 3 (2003), 313–318.

[PGMG09] PEYTAVIE A., GALIN E., MERILLOU S., GROSJEAN J.: Arches: a Framework for Modeling Complex Terrains. *Computer Graphics Forum (Proceedings of Eurographics) 28*, 2 (2009), 457–467.

[PH93] PRUSINKIEWICZ P., HAMMEL M.: A fractal model of mountains with rivers. In *Graphics Interface* (1993), pp. 174–180.

[PV95] PERLIN K., VELHO L.: Live paint: painting with procedural multiscale textures. In *Proceedings of SIGGRAPH* (1995), pp. 153–160.

[RME09] RUSNELL B., MOULD D., ERAMIAN M. G.: Feature-rich distance-based terrain synthesis. *The Visual Computer 25*, 5-7 (2009), 573–579.

[RPP93] ROUDIER P., PEROCHE B., PERRIN M.: Landscapes synthesis achieved through erosion and deposition process simulation. *Computer Graphics Forum 12*, 3 (1993), 375–383.

[SBBK08] ŠŤAVA O., BENEŠ B., BRISBIN M., KŘIVÁNEK J.: Interactive terrain modeling using hydraulic erosion. In *ACM Siggraph / Eurographics Symposium on Computer Animation* (2008), pp. 201–210.

[SBW06] SCHNEIDER J., BOLDTE T., WESTERMANN R.: Real-time editing, synthesis, and rendering of infinite landscapes on GPUs. In *Conference on Vision, Modeling, and Visualization* (2006), pp. 153–160.

[ST89] SZELISKI R., TERZOPOULOS D.: From splines to fractals. In *Proceedings of SIGGRAPH* (1989), pp. 51–60.

[ZSTR07] ZHOU H., SUN J., TURK G., REHG J. M.: Terrain synthesis from digital elevation models. *IEEE Trans. Vis. Comput. Graph. 13*, 4 (2007), 834–848.

© 2010 The Author(s)
Journal compilation © 2010 The Eurographics Association and Blackwell Publishing Ltd.