Terrain Modeling from Feature Primitives

Jean-David Génevaux [1], Éric Galin [1], Adrien Peytavie [1], Éric Guérin [1], Cyril Briquet [1], François Grosbellet [1,2], Bedrich Benes [3]

[1] Université de Lyon, LIRIS, CNRS, UMR5205, Lyon, France
[2] Université de Limoges, XLIM, CNRS, UMR7252, Limoges, France
[3] Purdue University, West Lafayette, IN, USA

Abstract

We introduce a compact hierarchical procedural model that combines feature-based primitives to describe complex terrains with varying level of detail. Our model is inspired by skeletal implicit surfaces and defines the terrain elevation function by using a construction tree. Leaves represent terrain features and they are generic parameterized skeletal primitives such as mountains, ridges, valleys, rivers, lakes, or roads. Inner nodes combine the leaves and subtrees by carving, blending, or warping operators. The elevation of the terrain at a given point is evaluated by traversing the tree and by combining the contributions of the primitives. The definition of the tree leaves and operators guarantees that the resulting elevation function follows the Lipschitz property which speeds up sphere tracing and adaptive tessellation algorithms used to render the terrain. Our model is compact and allows for the creation of large terrains with a high level of detail using a reduced set of primitives. We show the creation of different kinds of landscapes and demonstrate that our model allows to efficiently control the shape and distribution of landform features.

Categories and Subject Descriptors (according to ACM CCS):
I.3.5 [Comp. Graph.]: Computational Geometry and Object Modeling—Surface, and object representations
I.3.6 [Comp. Graph.]: Methodology and Techniques—Graphics data structures, Interaction techniques

1. Introduction

Terrain modeling has been on the radar of computer graphics for more than three decades. However, despite the considerable progress towards developing efficient methods for modeling terrains, it still remains an open problem.

Existing terrain modeling techniques can be categorized into procedural, physics-based, sketch-based, and example-based. Erosion simulation and hydrology-based algorithms generate results that are geologically correct, but often lack controllability and take a long time to calculate. Sketch-based methods involve manual editing that can be tedious and example-based algorithms are limited by the provided input. An important problem is the scalability of the algorithms. The generated terrains usually represent only features at a single scale that are stored in a discrete regular height field. The height field is later converted into a mesh suitable for fast visualization with varying levels of detail. Moreover, there is no algorithm that would allow for an interactive editing of complex terrains with a high level of detail, in turn allowing for a quick and controllable placement of features using simple editing operations.

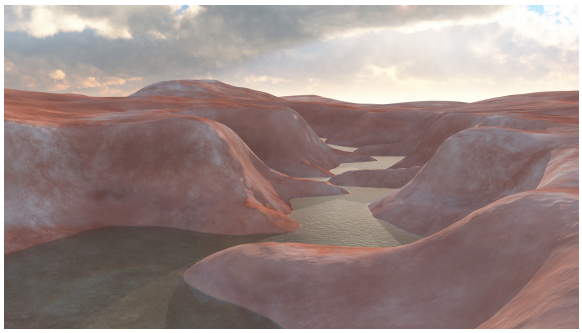

Figure 1: *Example of a scene created with our terrain representation. Thirty primitives were combined to define the terrain. The water elevation was defined by using an additional construction tree.*

In contrast, procedural methods are computationally efficient as they define the elevation of the terrain using a continuous procedural function. One limitation is that they do

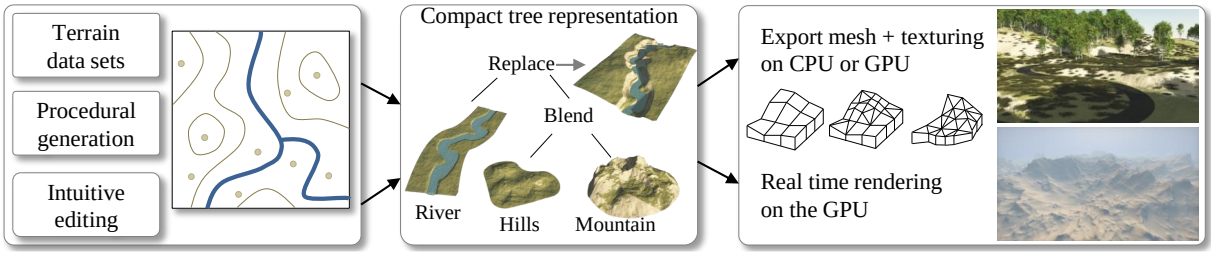

Figure 2: *Overview: using different inputs (procedural generation or terrain data), we create and allow an intuitive editing of a hierarchical terrain model. The construction tree can be evaluated on the CPU or on the GPU and exported as a mesh (regular/adaptative grid or TIN). We can also visualize terrains in real-time by using an adaptive polygonization [DIP14] or by an accelerated sphere tracing, both performed on the GPU.*

not provide control over the terrain features. A key observation of our work is coming from viewing real landscapes that are composed of *features* from different scales such as rivers, mountains, cliffs, or lakes. Additional terrain features arise from human activity and include roads and urban structures. The terrain could be created by composing the major features, preferably in an interactive manner, and completed by less important procedural content.

We introduce a novel compact hierarchical model that combines procedural primitives representing landform features. Our representation defines continuous terrains (Fig. 1) with varying level of detail, allows for a wide variety of terrains, and provides intuitive control over the terrain creation and editing. It is based on a tree structure that combines different kinds of primitives using operators.

This representation is compact, computationally efficient, and easy to implement, even on GPU. Moreover, the mathematical definition of the primitives and the operators provides us with foundations for fast visualization. The use of Lipschitz elevation functions allows us to visualize the terrains in real-time using an accelerated sphere tracing algorithm or polygonization techniques (Fig. 2). Because of this, our model is suitable for representing and intuitive editing of large scenes. Combined with the intuitive editing tools that focus on placement and distribution over landform features, the tree representation provides global and local control.

2. Related work

We relate our work to procedural approaches, physics-based methods, and interactive editing. For an overview of recent approaches in procedural modeling we refer to [STBB14] and terrain structures are surveyed in [NLP*13].

Physics-based techniques exploit the usage of water, temperature changes, or human activities as geomorphic agents. Probably the first to introduce erosion was [MKM89] and this work was extended by stabilization and hydraulic erosion in [Nag98, CMF98, BF02]. The majority of the above described techniques works with Eulerian approaches by using discrete regular heightfields, layered data structures [BF01], or a 3D volumetric data [BTHB06]. Lagrangian approaches that use smoothed particle hydrodynamics were combined with erosion in [KBKS09], and corrosion simulation has been introduced in [WCMT07]. Among the disadvantages of geomorphological algorithms is their low controllability. A related problem is their scalability due to the use of a discrete regular grid to represent the terrain. Because of high computational demands, these methods cannot be used to simulate large terrains with a high level of detail, even with the GPU [MDH07, VBHS11].

Interactive editing addresses terrain generation by using user intuition. Rusnell *et al.* [RME09] generate terrains by computing distances in a weighted graph. Zhou *et al.* [ZSTR07] combine 2D heightfield examples into the final terrain, but fails to generate results that are not exemplified by the input. Sketching approaches [GMS09, HGA*10, TGM12, TEC*14] and interactive terrain editing [PGMG09] provide good control over the resulting terrain, but can lead to results that are not geologically correct. Hybrid approaches that attempt to combine interactive editing with physics-based algorithms [SBBK08, VBHS11] are limited to small scenes and to editing existing terrains. As for erosion simulation and hydrologically-based approaches, sketch-based and example-based methods rely on a discrete regular grid and cannot be used to create large terrains with details.

Procedural techniques have been used in computer graphics for a long time. They are easy to implement and allow for the generation of a wide range of terrains by changing a few input parameters. The adaptive subdivision provides an intrinsic level of detail and belongs to noise-based procedural approaches that combines noise functions at various scales [EMP*98]. Although fractal-based methods can generate infinite terrains with unlimited precision, they are not easy to control and produce terrains without any underlying geographical structure.

Various algorithms have been developed to incorporate features such as rivers into the procedural terrains. Kelley *et al.* [KMN88] proposed a procedural method to gen-

erate watersheds by fractal interpolation. Prusinkiewicz *et al.* [PH93] combined context-sensitive L-systems with the midpoint displacement method to generate rivers in fractal terrains and Belhadj and Audibert [BA05] modified stochastic subdivision to constrains ridges and river curves generated by fractional Brownian motion. Teoh [Teo09] also started by producing the river network and fitting a terrain into it and Derzapf *et al.* [DGGK11] generated river networks on a planetary scale.

Recently, Génevaux *et al.* [GGG*13] used models inspired by geology and hydrology to generate models of terrains with embedded coherently placed rivers. While this work is inspired by the data structure they introduced, our representation is extended to include several new compact operators and primitives. Moreover, our model allows for intuitive edition and provides mathematical foundations based on the Lipschitz property to accelerate processing.

3. Construction tree representation

The terrain model is defined by a construction tree shown in (Fig. 3). The leaves of the tree act like primitives (Section 4), each generating a terrain portion containing similar landforms features. The internal nodes of the tree represent operators (Section 5) that combine and aggregate their subtree (each representing a part of the terrain, recursively composed of one or several primitives). The elevation of a point $\mathbf{p}$ depends on the evaluation of the hierarchical combination of every primitive in the tree.

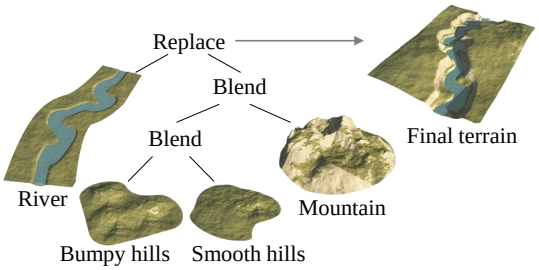

Figure 3: *A simplified construction tree representing a river carved into a hilly scene. Editing the scene usually consists in modifying the placement of the different primitives or in tuning the parameters of the different elevation functions.*

We define two functions in each node: an elevation function $f(\mathbf{p}) : \mathbf{R}^2 \to \mathbf{R}$ and a weight function $\alpha(\mathbf{p}) : \mathbf{R}^2 \to \mathbf{R}^+$. The weight function defines the way a terrain portion will be combined in its environment, when used with an operator. We define the compact support, denoted Ω_0, of the weight function α as:

$$\Omega_0 = \{\, \mathbf{p} \in \mathbf{R}^2 \;\mid\; \alpha(\mathbf{p}) > 0 \,\}.$$

The elevation function f is defined on Ω_0. Let $T > 0$ be a threshold value. The terrain is defined on the sub-domain denoted $\Omega_T \subset \Omega_0$ and defined by:

$$\Omega_T = \{\, \mathbf{p} \in \mathbf{R}^2 \;\mid\; \alpha(\mathbf{p}) \geq T > 0 \,\}.$$

An example in Fig. 4 illustrates this concept and represents the domains Ω_0 and $\Omega_T \subset \Omega_0$. Both domains Ω_0 and Ω_T can be composed of disconnected components. In the remainder of this paper, the region Ω_0 is removed for clarity, except whenever explicitly specified.

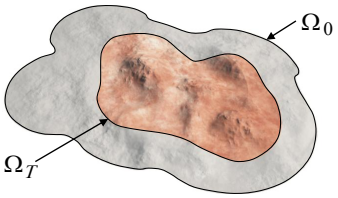

Figure 4: *Characterization of a terrain feature: for each node, Ω_0 defines the support of f and α, however the terrain only exists on $\Omega_T \subset \Omega_0$.*

In our implementation, we use $T = 1/2$. We define the terrain surface $\mathcal{T}$ as the set of points $(\mathbf{p}, f(\mathbf{p}))$ in space whose elevation $f(\mathbf{p})$ is defined from their projection $\mathbf{p}$ in Ω_T:

$$\mathcal{T} = \{\, (\mathbf{p}, f(\mathbf{p})) \in \mathbf{R}^3, \mathbf{p} \in \Omega_T \,\}.$$

The tree representation allows to adapt the evaluation of the elevation function f according to a continuous level of detail (Section 6). Moreover, the construction of the hierarchical model guarantees that the elevation functions f attached to the primitives and the operators satisfy the Lipschitz property. The weight functions are C^1. These mathematical properties provide us with mathematical foundations to accelerate the computations for visualization (Section 7).

Finally, the construction tree implements a bounding volume hierarchy that allows for efficient encoding of spatial relations. Thanks to this property, our model allows for an efficient rendering as shown in Section 7.

Normal vector evaluation may be needed by some operators *e.g.*, during rendering. The normal of the terrain at a given point $\mathbf{p}$ can be computed from the gradient of the elevation ∇f or by its discrete approximation.

$$\nabla f(\mathbf{p}) \simeq \frac{1}{2\varepsilon} \begin{pmatrix} f(\mathbf{p}+\varepsilon_x) - f(\mathbf{p}-\varepsilon_x) \\ f(\mathbf{p}+\varepsilon_y) - f(\mathbf{p}-\varepsilon_y) \end{pmatrix}.$$

Our framework implements exact computation of the gradient for each node type whenever possible. Moreover, when the evaluations of $f(\mathbf{p})$, $\alpha(\mathbf{p})$ and the gradient $\nabla f(\mathbf{p})$ are needed, we use an optimized query that simultaneously computes the values at the same time when traversing the tree structure and returns them together. This speeds up computations, not only because the tree is traversed only once, but also because the computation of several intermediate terms can be factored out.

4. Primitives

We use two different kinds of primitives: skeletal-based procedural primitives and image-based. Those two approaches allow for the creation and control of the landform features in different ways. The skeletal primitives are fast to render and have smaller memory requirements, but they need an explicit analytical elevation function associated, whereas the image-based primitives provide, with a memory cost, an easy way for an artist to integrate real data and/or precise user-crafted features into the scene.

4.1. Skeletal primitives

Skeletal primitives are inspired by the Constructive Solid Geometry and by the Blob-Tree model of [WGG99]. They are defined by a geometric skeleton (point, segment, curve, or contour) and a set of parameters that describe the elevation and the weight functions depending on the distance from the skeleton. We use different types of skeleton adapted to the representation of features: disc, curve, and contour.

Disc primitives represent a small portion of a terrain. The center $\mathbf{c}$ and the radius r describe their areas of influence (Fig. 5). The noise function referred to as $\eta(\mathbf{p})$ controls the local ground roughness. Let $\eta(\mathbf{p}) : \mathbf{R}^2 \rightarrow \mathbf{R}$ denote a two dimensional noise function, $\{a_i\}$ a set of decreasing amplitudes, and $\{s_i\}$ a set of increasing frequencies. The elevation function is defined as:

$$f(\mathbf{p}) = \mathbf{c}_z + \sum_{i=0}^{n-1} a_i \, \eta\big((\mathbf{p}-\mathbf{c})\, s_i\big)\,.$$

Figure 5: *Disc primitives allow to generate landforms in small circular areas. Various elevation functions can be used: a constant function will represent a plain or plateau, a turbulence will produce hills and a ridge-multifractals noise function [MKM89] will mimic mountains landscapes.*

Curve primitives are built from a piecewise curve skeleton Γ and from a set of profiles $\{c_i\}$ describing the cross sections perpendicular to the curve (Fig. 6). In our implementation, each cross section is stored as a one-dimensional piecewise quadric function to speed up computations.

The signed distance between $\mathbf{p}$ and the curve Γ is denoted $d(\mathbf{p})$, the projection of $\mathbf{p}$ on Γ is denoted $\mathbf{q}$. Let c be the cross section in $\mathbf{q}$, constructed by interpolation using the curvilinear coordinates on the curve Γ. We define:

$$f(\mathbf{p}) = \mathbf{q}_z + c \circ d(\mathbf{p})\,.$$

Rivers are created from a curve skeleton that defines the river path Γ. The profile curves $\{c_i\}$ represent the cross section of the river that corresponds to the river type (stream, river, meander, main stem). Confluences or braided river streams can be built in the same way using more complex cross section curves representing multiple channels, or by combining several rivers using the blending operator.

Roads rely on the same model as rivers but with different cross sections. The skeleton is a piecewise C^1 continuous curve approximation of clothoids that represent trajectories, whereas profile curves define the cross section of the road and are parametrized by the road width and the size of the carriageway. Road primitives also define the elevation of the terrain in the neighborhood of the road to allow a combination with other primitives in the terrain.

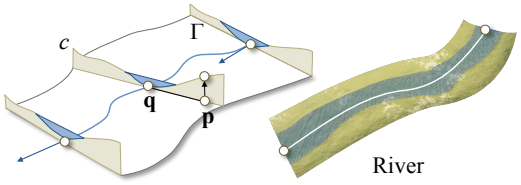

Figure 6: *Curve primitives allow us to create easily terrain features structured around trajectories such as roads or rivers. The curves can be constructed with lines or splines but, in order to speed up the evaluation, it is important that the distance and orientation functions are fast to compute.*

Contour primitives are created from a curve enclosing a center point $\mathbf{c}$, from an orientation vector $\mathbf{v}$ and from a set of profile curves $\{c_i\}$ that describes the elevation in radial direction. Each cross section is stored as a one-dimensional piecewise quadric function (Fig. 7). The elevation function f is defined as follows. Given a point $\mathbf{p}$, we compute its polar coordinates: the distance $d(\mathbf{p})$ and the angle $\theta(\mathbf{p})$ computed according to $\mathbf{v}$. The elevation of a point depends of a cross-section c that we compute by interpolation using angular sectors. We define:

$$f(\mathbf{p}) = \mathbf{c}_z + c \circ d(\mathbf{p})\,.$$

Those primitive resemble anisotropic distance functions for implicit sweep objects proposed in [CBS96].

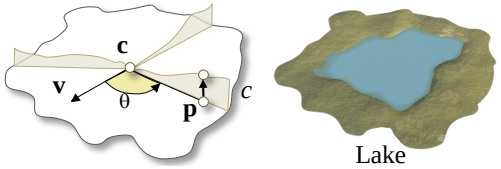

Figure 7: *More complex primitives can be defined using polygonal shapes. A lake is created by defining a set of increasing cross sections, radiating from the center* $\mathbf{c}$.

Weight functions are defined as C^1 continuous decreasing functions over compact support to limit the influence of the primitive and control the way they are combined by operators in the construction tree. Let $d(\mathbf{p})$ denote the distance between the evaluated point $\mathbf{p}$ and the skeleton. We define the weight function α as a composition of the distance with a smooth fall-off filter function g:

$$\alpha(\mathbf{p}) = g \circ d(\mathbf{p}) .$$

In our implementation, we use Wyvill's fall-off filter function [WGG99] which provides good range control:

$$g(x) = \begin{cases} \left(1 - \left(\frac{x}{r}\right)^2\right)^3 & \text{if } x < r , \\ 0 & \text{otherwise} . \end{cases}$$

4.2. Image primitives

Image-based primitives define specific and complex terrain features, such as detailed rivers or sand ripples, which would be difficult to create by using procedural skeletal primitives.

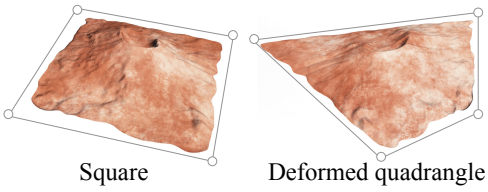

Figure 8: *Using real data (Mount St. Helens, U.S.A.) and mapping it onto a convex quadrangle that can be deformed is possible. For a given point in the quadrangle, we compute its parametrization using inverse bilinear interpolation and then get the elevation value contained in the image.*

The elevation and weight functions are computed by mapping a discrete height field and weight field images onto a given domain Ω for which the parametrization (u,v) is known. The evaluation of f and α is done by interpolating the data to guarantee a Lipschitz property for f and a C^1 continuity for α respectively. We implement a parametrization for convex quadrangles that allow us to easily use real height fields such as mountains depicted in Fig. 8.

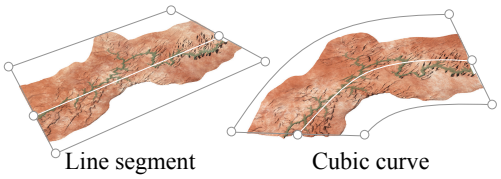

Figure 9: *Mapping raster images (Grand Canyon, U.S.A.) onto curves allow us to use complex geometries multiple times without visual artifacts due to tiling.*

We also propose a parametrization along curves that allows to create rivers and confluences of complex trajectories and surroundings (Fig. 9). In our implementation, complex rivers are created by connecting several curve image primitives. The different images defining the local elevation of the terrain match together at their borders in a seamless way.

5. Operators

Operators are internal nodes that combine the elevation and weight functions of their sub-trees, thus defining two new functions f and α. We consider binary nodes and denote the two sub-trees A and B.

5.1. Blending

This operator provides an effective way for creating large terrains with different features from a set of patches. The blending of two nodes A and B combines the elevation functions f_A and f_B according to the weight functions α_A and α_B and allows to aggregate two terrain primitives (Fig. 10).

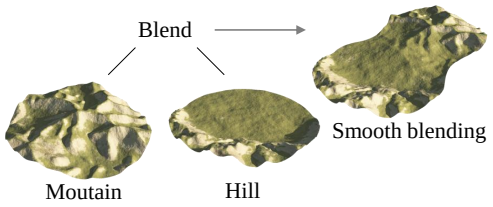

Figure 10: *The blending operator allows to aggregate several primitive in a smooth and continuous way. This node can be easily transformed in n-ary operator to work with a large amount of primitives.*

$$f = \frac{\alpha_A f_A + \alpha_B f_B}{\alpha_A + \alpha_B} , \qquad \alpha = \alpha_A + \alpha_B .$$

When two primitives are far away (their support Ω_A and Ω_B don't intersect), they do not have a mutual influence and the resulting terrain is the union of the two terrain patches produced by the sub-trees A and B. As the primitives get closer, their regions of influence intersect and they progressively blend together (Fig. 11).

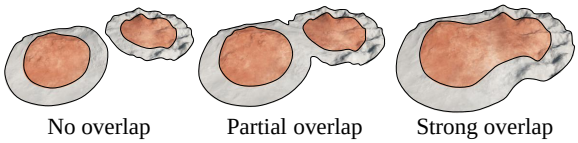

Figure 11: *Blending allows to extend the definition domain of the terrain. If the primitives don't intersect, the operator corresponds to the union. When primitives get closer, their support and elevation blend in a smooth way.*

The blending operator is commutative and associative and it can be extended to become an n-ary node. This property is enables us to optimize the memory footprint of the hierarchical structure my minimizing the depth of the tree. Combined to a grid acceleration structure, it is also used to speed-up the computations when many primitives blend together.

5.2. Replacement

This operator defines and locates specific terrain features such as lakes, mounds, rivers, or roads over an existing terrain. The operator continuously replaces its argument subtree A by another one B.

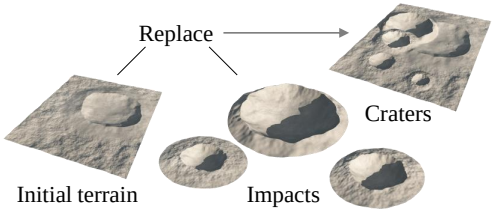

Figure 12: *The replacement operator allows to easily constraint the landscape. It also represents a notion of temporality: here, the last operand corresponds to the most recent crater impact.*

Replacement is an asymmetric operator and it can be used for example to place a water-course geometry such as rivers or lakes onto the terrain. It allows to easily sculpt road paths or create lunar landscapes (Fig. 12). The node A is replaced by B only when $\alpha_B > 0$:

$$f = (1-\alpha_B)\, f_A + \alpha_B\, f_B\,, \qquad \alpha = \alpha_A\,.$$

In general, we want to keep the weight of the left-hand operand, which is why we set $\alpha = \alpha_A$. Depending on the context, we may want to take into the account the influence of the weight function α_B and set:

$$\alpha = (1-\alpha_B)\,\alpha_A + {\alpha_B}^2\,,$$

which progressively replaces α_A by α_B.

5.3. Addition

This operator locally adds variations and details to the elevation of an existing terrain A according to the weight function α_B of the second argument B (Fig. 13).

$$f = f_A + \alpha_B\, f_B\,, \qquad \alpha = \alpha_A\,.$$

Similarly to the replacement operator, we want to keep the weight of the left-hand operand, which is why we set $\alpha = \alpha_A$. Another possibility is to take into account the influence of the weight function α_B, and use $\alpha = \alpha_A + {\alpha_B}^2$.

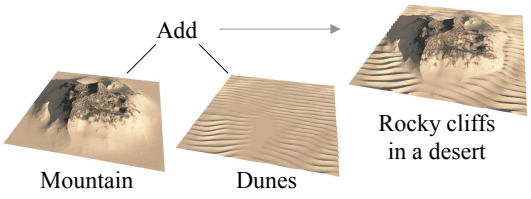

Figure 13: *The sand dunes are added to a mountainous landscape. Because we want them to appear at low altitude without masking the rocks, the weight function α_B is defined according to the mountain's elevation f_A and gradient ∇f_A*

5.4. Warping

This operator allows a distortion of the shape of a surface by warping both the elevation f and the weight function α (Fig. 14). A warp is a bijective C^2 function denoted as: $\omega(\mathbf{p}) : \mathbf{R}^2 \rightarrow \mathbf{R}^2$. The warping operator is defined as an unary node which can be evaluated as:

$$f(\mathbf{p}) = f_A \circ \omega^{-1}(\mathbf{p})\,, \qquad \alpha(\mathbf{p}) = \alpha_A \circ \omega^{-1}(\mathbf{p})\,.$$

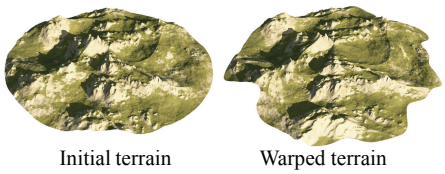

Figure 14: *It is possible to deform both primitive and terrain portions made of a combination of nodes. Deformation allows to eliminate the visual artifacts due to the repetitive usage of a single type of primitive across a large terrain.*

In some cases, we may want to compute the gradient of the elevation function, *e.g.*, for normal vector evaluation. The gradient of f becomes:

$$\nabla f(\mathbf{p}) = \nabla f_A \circ \omega^{-1}(\mathbf{p}) \times J_{\omega^{-1}}(\mathbf{p})\,.$$

In order to speed-up the computations, we store the Jacobian $J_{\omega^{-1}}$ for every warp function ω.

Affine transformations can be applied as specific warp operators. Although the effect is not different from applying the same transformations to the skeletons, the advantage is that it is possible to transform a complex terrain portion corresponding to a sub-tree composed of several primitives. Moreover, each primitive can be defined in its canonical position, orientation and scale that allow us to instantiate the primitive multiple times. Thus the construction tree becomes a directed acyclic graph (Fig. 15).

Affine transformation can be composed with deformations. For efficiency, consecutive affine transformations can be concatenated.

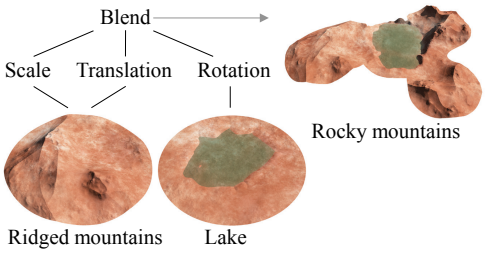

Figure 15: *Instantiation associated with transformation operators (translation, rotation, homothety) allow to use the same primitive several times in a scene.*

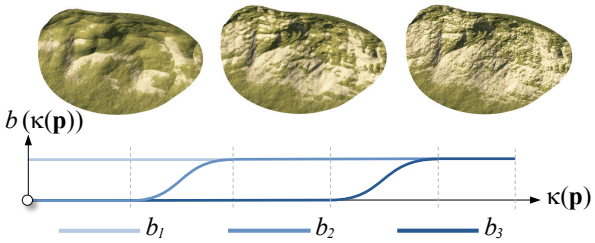

Figure 16: *A continuous LoD primitive: as the global level of detail κ increases, the filter functions b_i will, one by one, unravel smoothly the different details.*

6. Level of detail

Our hierarchical representation allows to adapt the computation of the elevation function according to a continuous required level of detail (LoD). Adapting the evaluation of f is done by using either specific operators that will chose which primitive to use, or with the help of parameterized primitives with a LoD system. We denote $\kappa(\mathbf{p}) : \mathbf{R}^2 \to [0,1]$ a continuous C^1 function defining the level of detail.

Level of detail operators are binary nodes whose sub-trees are evaluated according to the input of detail $\kappa(\mathbf{p})$. They are characterized by two threshold values: k_A and k_B with $0 \leq k_A < k_B \leq 1$ that define which sub-tree should be evaluated according to the needed LoD. The elevation function is defined as:

$$f(\mathbf{p}) = \begin{cases} f_A & \text{if } \kappa(\mathbf{p}) \leq k_A\,, \\ (1-t)\,f_A + t\,f_B & \text{if } k_A < \kappa(\mathbf{p}) \leq k_B\,, \\ f_B & \text{otherwise}\,, \end{cases}$$

with

$$t = 1 - \frac{k_B - \kappa(\mathbf{p})}{k_B - k_A}\,.$$

The weight function α is computed in the same way.

Continuous level of detail primitives are characterized by an elevation function $f(\mathbf{p})$ whose computation directly depends on the level of detail function $\kappa(\mathbf{p})$. Continuous LoD primitives automatically simplify their evaluation with the objective of speeding up computation as the level of detail decreases.

Our system implements different kinds of continuous level of detail primitives. One example is the disc terrain primitive whose elevation is defined by a sum of several noise functions of decreasing amplitude and increasing frequencies. When this primitive is queried with a low level of detail, we can only sum the lowest frequencies. Between two levels of detail, we interpolate the elevations to get a continuous and smooth transition. We define $f(\mathbf{p})$ as:

$$f(\mathbf{p}) = \mathbf{c}_z + \sum_{i=0}^{n-1} a_i\, \eta\big((\mathbf{p}-\mathbf{c})\,s_i\big) b_i \circ \kappa(\mathbf{p})\,.$$

The filter functions $b_i : [0,1] \to [0,1]$ are defined as smoothly increasing functions on the interval $[k_i^-, k_i^+]$ of the input level of detail $\kappa(\mathbf{p})$:

$$b_i(\kappa) = \begin{cases} 0 & \text{if } \kappa \leq k_i^-\,, \\ \big(1-t^2\big)^3 & \text{if } k_i^- < \kappa \leq k_i^+\,, \\ 1 & \text{otherwise}\,, \end{cases}$$

with

$$t = 1 - \frac{k_i^+ - \kappa}{k_i^+ - k_i^-}\,.$$

Continuous level of detail can be controlled by modifying the different definition intervals (Fig. 16). A typical set up for $\kappa(\mathbf{p})$ is to make it decrease the LoD when the distance from the evaluated point to the camera increases.

7. Visualization

We introduce an accelerated ray tracing method to visualize our terrain model (Fig. 17) taking advantage of the mathematical properties provided by the definition of the elevation function of the construction tree. Our approach is inspired by the sphere tracing method [Har94] which is a robust technique for ray tracing implicit surfaces. Unlike LG-surfaces [KB89] it does not require that the evaluated function is C^2 continuous. Instead, it requires only a bound on the magnitude of the derivative – that the function is C^0 continuous and obeys Lipschitz property. Thus, the derivative of the function does not need to be continuous or even defined.

Recall that a function $h(\mathbf{p}) : \mathbf{R}^3 \to \mathbf{R}$ is Lipschitz over a domain Ω if and only if there exists a positive constant λ such that:

$$\forall (\mathbf{p},\mathbf{q}) \in \Omega \times \Omega\,,\quad |\,h(\mathbf{p}) - h(\mathbf{q})\,| < \lambda \|\,\mathbf{p} - \mathbf{q}\,\|\,.$$

The Lipschitz constant λ is the minimum satisfying this equation. In practice, Lipschitz constants are overestimated by a Lipschitz bound, particularly for nodes in the tree whose sub-trees have known Lipschitz constants.

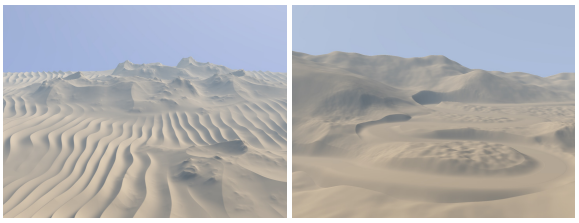

Figure 17: *Examples of terrains rendered in read-time using the accelerated sphere tracing algorithm. Using sphere tracing instead of adaptive tessellation allows us to render faster and without visual artifacts detailed and sharp geometries such as those sand dunes.*

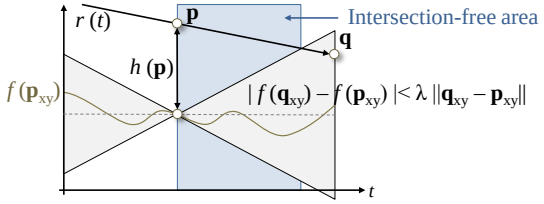

Figure 18: *Lipschitz constant* λ *of* f *and ray direction* $\mathbf{u}$ *provides us with a guaranteed intersection-free area.*

7.1. Sphere tracing

The sphere tracing algorithm consists in marching along the ray Δ from a camera towards the surface by an adaptive increment defined as $|h(\mathbf{p})|/\lambda$ that is sufficiently small to guarantee that we will not penetrate the surface. Let $r(t) = \mathbf{o} + t\,\mathbf{u}$ be the parametric ray equation (where $\mathbf{u}$ is normalized). The field function along the ray develops as $h(t) = h \circ r(t)$. We denote ε the threshold value from which we consider that we are so close to the surface that we intersect it. The algorithm can be written as:

1. Compute a Lipschitz bound λ of the function h and the interval of intersection between the ray and the domain $[t_-, t_+] = \Delta \cap \Omega$. Initialize t to t_-.
2. While $t < t_+$, compute $h(t)$.
3. If $|h(t)| < \varepsilon$ then we will consider that there is an intersection, otherwise increment t with $|h(t)|/\lambda$ and continue.

In the following sections, we show how we can take advantage of the tree definition to improve the algorithm and speedup computations.

7.2. Accelerated sphere tracing

Let $\mathbf{p}$ be a point of $\mathbf{R}^3$ where $\mathbf{p}_{xy}$ and $\mathbf{p}_z$ denote the x, y and z coordinates of $\mathbf{p}$. Recall that $f(\mathbf{p}_{xy})$ represents the terrain elevation at $\mathbf{p}_{xy}$. We define an implicit surface characterizing the terrain by its field function $h(\mathbf{p}) : \mathbf{R}^3 \to \mathbf{R}$ defined as:

$$h(\mathbf{p}) = \mathbf{p}_z - f(\mathbf{p}_{xy})\,.$$

Since the elevation function f is built from a hierarchical combination of compactly supported primitives, the local Lipschitz constants are usually smaller than λ. Smaller bounds can be computed by using standard space partitioning structures but require the structure traversal. We propose an accelerated sphere tracing algorithm, adapted to our model and based on the evaluation of a Lipschitz local bounds.

The algorithm consists in marching along the ray with an adaptive step that depends of the computation of the Lipschitz bound of the local neighborhood. At every step, we try to move forward by a candidate step of length δ and compute a stepping distance denoted as $s(t,\delta)$. The candidate distance defines an interval $[t\,, t+\delta]$ corresponding to a segment $[\mathbf{p}\,, \mathbf{q}]$ with $\mathbf{q} = \mathbf{p} + \delta\mathbf{u}$ being a point further down the ray (Fig. 18).

We query the tree structure to compute both the elevation $f(\mathbf{p})$ and the local Lipschitz bound $\lambda(t,\delta)$ along the segment. The function $h \circ r(t)$ develops as:

$$h \circ r(t) = \mathbf{o}_z + t\,\mathbf{u}_z - f(\mathbf{a}_{xy} + t\,\mathbf{u}_{xy})\,.$$

The derivative of $h \circ r(t)$ according to t is:

$$(h \circ r)'(t) = \mathbf{u}_z - ||\,\mathbf{u}_{xy}\,||\;\; f'(\mathbf{a}_{xy} + t\,\mathbf{u}_{xy})\,.$$

Depending on the sign of $\lambda(t,\delta)\;||\,\mathbf{u}_{xy}\,|| - \mathbf{u}_z$, two cases arise:

1. If $\lambda(t,\delta)\;||\,\mathbf{u}_{xy}\,|| - \mathbf{u}_z > 0$, an intersection-free area is detected thus it is possible to move forward with the distance:
$$\frac{h(t)}{(\lambda(t,\delta)\;||\,\mathbf{u}_{xy}\,|| - \mathbf{u}_z)}\,.$$
2. If $\lambda(t,\delta)\;||\,\mathbf{u}_{xy}\,|| - \mathbf{u}_z \leq 0$, no intersection can occur on the interval $[t\,, t+\delta]$ since the derivative is positive.

Therefore, we can move forward along the ray without intersecting the terrain surface by a stepping distance:

$$s(t,\delta) = \min\left(\delta,\, \frac{h(t)}{(\lambda(t,\delta)\;||\,\mathbf{u}_{xy}\,|| - \mathbf{u}_z)}\right).$$

Given an initial step δ, the algorithm proceeds in three steps:

1. Compute the interval $[t_-, t_+] = \Delta \cap \Omega$ and initialize t to t_-.
2. While $t < t_+$, compute $h(t)$ and $\lambda(t,\delta)$.
   2.1 If $h(t) < \varepsilon$, then an intersection is considered to be found in t.
   2.2 Otherwise increase t with increment $s(t,\delta)$.
3. Update the distance δ to $2\,\delta$.

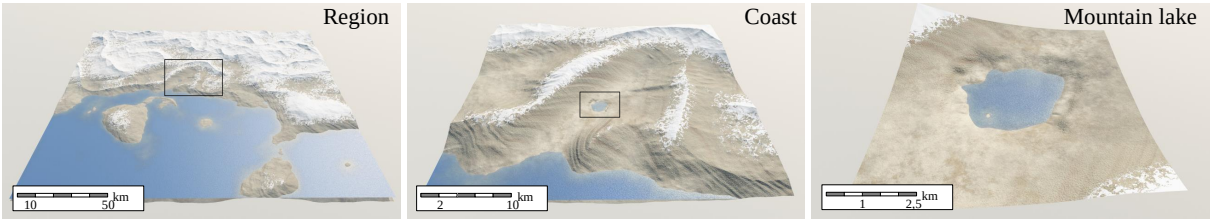

Figure 19: *Our model allows to represent scenes composed of primitives of various scales. In this example, the overall relief of the terrain has been generated by blending multiple noise primitives ; the more detailed mountain, located near the coastline, is an image primitive. Finally, the lake was produced by a combination of small procedural disc primitives and carved with the help of a replacement operator.*

Step 3 of the algorithm allows to constantly try to move forward by larger distances. The computation of $s(t,\delta)$ guarantees that this step length is intersection-safe.

7.3. Lipschitz constant computation

Our model provides an accurate Lipschitz constant evaluation for every type of node (primitives and operators). Thus, the Lipschitz constant λ of the root of the tree is computed by traversing the tree structure recursively. This means that for each node, we need to compute the minimum and maximum elevations and the Lipschitz constants of both the elevation and weight functions.

7.3.1. Primitives

The elevation functions of primitives always follow the Lipschitz property. For image based primitives, the elevation function is computed using a bi-linear interpolation of a height field. Consider the mapping onto a square: the elevation function f is Lipschitz and its constant is bounded by the maximum elevation difference between two neighbor samples divided by the distance l between those samples:

$$\lambda = \frac{1}{l} \max_{i,j\in[0,n-1]^2} \left(|\, f_{i,j} - f_{i+1,j} \,|, |\, f_{i,j} - f_{i,j+1} \,| \right).$$

When projected along a curve or on a quadrangle, this constant is modified by the deformation.

In our implementation, skeletal primitives are often built upon sum of gradient noises of various amplitudes and frequencies. The gradient noise function η has intrinsic Lipschitz properties, and we denote its bound λ_η. When several harmonics of noise are summed, we estimate the new bound by summing the Lipschitz constants as well.

$$\lambda = \sum_{i=0}^{n-1} \frac{a_i\, \lambda_\eta}{s_i}.$$

Thus level of detail skeletal primitives which are derived from skeletal primitive by filtering the terms according to the prescribed level of detail, speed up rendering as they provide smaller Lipschitz bounds.

The weight functions α are defined by $\alpha(\mathbf{p}) = g \circ d(\mathbf{p})$ where d is the Euclidean distance to a skeleton and g is the Wyvill's function. The Lipschitz constant of α is the maximum of the derivative of g. More details are presented in appendices.

7.3.2. Operators

For every operator, we compute its Lipschitz bound according to the bounds of its sub-trees λ_A and λ_B. For example, the Lipschitz constant λ of a blending node is defined as (see Appendix for proof materials):

$$\lambda \leq \max(\lambda_{f_A}, \lambda_{f_B}) + \max(\lambda_{\alpha_A}, \lambda_{\alpha_B}) \sup |\, f_A - f_B \,|,$$

where $\lambda_{f_A}, \lambda_{f_B}, \lambda_{\alpha_A}, \lambda_{\alpha_B}$ represents the Lipschitz constants for the elevation and weight functions of each sub-tree respectively.

8. Results and discussion

We have implemented our system in C++ with the use of OpenGL and GLSL. All examples in this paper were created on a desktop computer equipped with Intel® Core *i*7, clocked at 3 GHz with 16 GB of RAM. Photorealistic images with vegetation were produced by extracting the surface from a model and directly streaming it into MentalRay®. GPU evaluations were performed using GLSL 4.4 on an nVidia® GeForce GTX 670.

8.1. Edition

Our model allows to construct and manipulate complex and varying terrains composed of both large primitives defining large scale features (mountains, plains, plateaus) and smaller primitives that lets the user sculpt and edit details with arbitrary precision such as rivers, dunes, and lakes.

A construction tree is represented by a compact array of parameters. The structure is lightweight and experiments show (see Table 1) that we need between ten to a few hundred primitives per square kilometer to represent most of the major features (hills, mountains, rivers). Very small scale details are provided by manually tuned noise. We can enrich

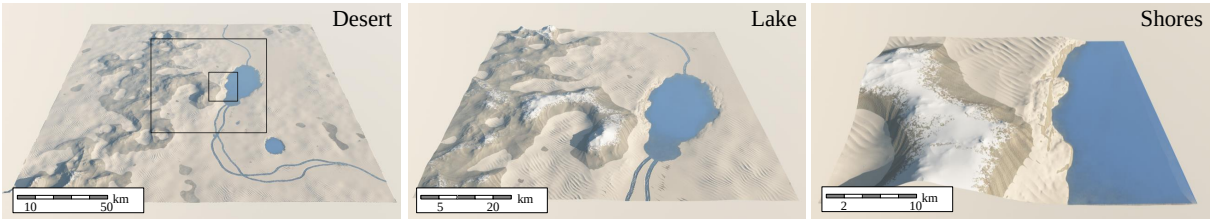

Figure 20: *Our representation, combined to textures and physically based shading, allows to create complex scenes. The construction tree of this scene uses only three types of primitives: the lake corresponds to Fig. 7, sand dunes to Fig. 13 and the mountains are several instances of Fig. 14. The water elevation was defined by an additional construction tree.*

each scene even more with additional primitives to represent really fine details such as footprints, tire track, sand ripples, mounds, sharp edges, leveling for trees, *etc*.

Our representation is based on the usage of procedural primitives. Using compact primitives that are geologically and spatially homogenous provides the user both local and global control. It is easy and intuitive for an artist to locally edit the terrain and add details, which allows to construct large scale scenes (Fig. 19) with a large amount of details (Fig. 21).

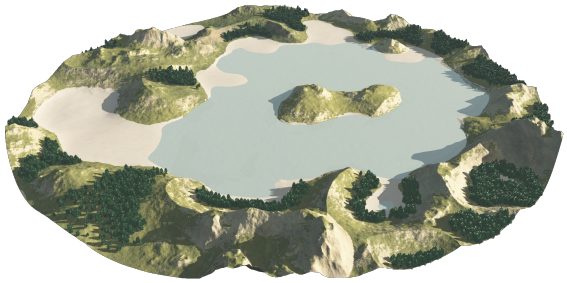

Figure 21: *Our model allows to construct complex scenes. Using additional construction trees, it is possible to represent different layers of materials (in this case, the sand and the water). The vegetation density is defined in the same way.*

Our model can be extended easily by adding new primitives or operators that answer specific needs for the user. In practice, developing a new node in the framework requires adding a few dozen lines of code in a new class.

8.2. Performance

Our model is based on a reduced set of generic primitives and operators whose elevation and weight functions $f(\mathbf{p})$ and $\alpha(\mathbf{p})$ can be implemented as generic functions in a shader program. The tree structure and the parameters of the different nodes in the tree are stored in two shader storage buffers objects (SSBO). The evaluation function is performed by parsing the first SSBO containing the tree structure and calling the generic functions with the corresponding data.

We have implemented two different techniques for visualizing on the GPU: an accelerated sphere tracing algorithm (described in Section 7) and adaptive quad-tree tessellation of [DIP14]. The parallel evaluation of the terrain tree combined to enhanced shading possibilities enables us to render images at interactive rates (Fig. 17).

Adaptive quadtree tessellation allows for a multi-resolution and crack-free terrain elevation surface generation with frustum culling using hardware tessellation and a distance-based level of detail selection criterion. Both the tessellation and the evaluation are done using the GPU.

On one hand, tessellation is fast and easy and allows for a foreseeable number of evaluations, thus, the rendering time is only dependent of the complexity of the tree. On the other hand, sphere tracing generates images of better quality, especially on fine and sharp primitives but the number of evaluations of the construction tree is larger and view-dependent.

The evaluation of the whole tree in order to visualize a complete complex terrain takes only a few seconds on the CPU and less than a second on a GPU. That allows us to explore the scenes interactively. The timing statistics of the scenes throughout the paper are shown in Table 1. For scenes composed of hundreds of primitives, the spatial pruning, thanks to the hierarchical set of bounding boxes, has a great influence on performance. The frequency pruning allows to avoid even more necessary evaluations and increases sphere tracing increments by diminishing Lipschitz constants.

8.3. Comparison to other models

Most existing data-structures for representing and generating terrains can be classified into two categories: discrete representations and function-based models. The discrete representations define the terrain in discrete points organized in a regular or an adaptive grid and the continuous elevation is obtained by a reconstruction that interpolates the elevations from the corresponding set of samples. The function-based

Scene	Surface	Sphere tracing		Regular mesh		Memory footprint		
		Calls of $f(\mathbf{p})$	Time	Calls of $f(\mathbf{p})$	Time	Primitives	Operators	Memory
Fig. 1	2×2	60M	182	1M	14	70	9	10
Fig. 3	1×1	55M	36	1M	6	20	3	3
Fig. 19	200×200	50M	60	4M	25	15	3	81
Fig. 20	100×100	79M	561	4M	150	50	25	7
Fig. 21	2×2	66M	280	16M	320	78	10	12
Fig. 22	0.6×0.6	52M	361	16M	560	148	9	19

Table 1: *Statistics for some terrain scenes. Computation times are measured in* ms*, memory cost in* kB*, surface in* km^2*. The sphere tracing algorithm renders images at* 1920×1080 *resolution. The scenes were created in* 15 *to* 55 *minutes. Most of the construction time was spent placing details. The memory cost of the image primitive used in Fig. 19 is about* 61 kB.

models directly define the elevation as a procedural or analytic function. Erosion simulations [MKM89], hydrology based techniques [BF01] as well as example or sketch based methods [GMS09] rely on a uniform or adaptive grid to generate the terrain. Therefore, they do not easily allow the creation of large terrains with a high level of detail as they are limited by the grid resolution and can represent only features at a single scale. In contrast, function-based models, such as fractal and noise based approaches [EMP*98], are not limited in terms of resolution, but they are often difficult to control. Also, they are difficult to combine with erosion models.

Although our compact hierarchical model allows for discrete input it belongs to function-based models. It provides a simple and efficient framework that combines different landform features with varying level of detail, allows for creating a vast variety of terrains, and provides intuitive control over the terrain creation and editing. Our model does not handle erosion simulation or hydrological correctness intrinsically because those methods rely on the computation of neighboring values which is not compatible with a *pure* evaluation approach. However, it could be combined with hydrologically-based generation techniques as presented in [GGG*13], and could be used in an interactive editor. An implementation of such editor, along with high level authoring tools, are beyond the scope of this paper, but would be an avenue for future work.

8.4. Limitations

Our method allows representation of 2D terrains without overhangs contrary to voxels or layered terrain representations. This is due to the nature of the underlying tree representation and its evaluation. However, it is possible to represent multiple layers of different materials, placed on top of each other, using additional tree structures. An example is shown as the sand and water in Fig. 20, 21 and 23.

We have shown several examples of node primitives. In order to include a new landscape feature, our approach would need to be extended with a new type of node. Adding new primitives or new operators can be easily done but it requires an analytical definition that represents the new landforms feature.

As mentioned above, our approach does not explicitly provide a location of a neighboring location and is therefore not directly suitable for methods that require this kind of information, such as erosion simulation or weathering.

9. Conclusion and future work

We have introduced a novel compact hierarchical procedural model that combines feature-based primitives to define complex continuous terrains with varying level of detail (Fig. 22). Inspired by the skeletal implicit surfaces and Constructive Solid Geometry, we define the elevation of each point as a function by using a construction tree whose leaves are primitives describing terrain features, and the inner nodes are the operations that aggregate and combine them together. The scene rendering is efficiently performed on the GPU by bounding the terrain to Lipschitz condition that allows for a fast GPU-oriented rendering or a GPU-oriented polygonization.

One of the interesting extensions would be to make our approach compatible with erosion models that have a great visual impact on landscapes. Currently we cannot use simulations, however we would like to test *pure* procedural erosion that does not need any neighboring evaluations such as the approach of [dCB09].

We think that the tree representation has the potential to become a foundation of many new works. Many existing procedural algorithms could rely on the hierarchical representation instead of working with regular heightfields. One way to show the potential of our method would be by creating a robust interactive editing tool that would work with the tree structure to allow an artist to construct a terrain scene in a fast, easy and intuitive way.

Figure 22: *Example of a winding road on a hilly terrain. The artist first authored a draft of the terrain scene by blending a hundred of disc primitives to define the overall landscape. The trajectory of the road was defined by using about 20 quadric curves and the replacement operator was used to level the terrain. Several disc primitives were placed along the trajectory of the road to locally adjust the elevation to make the integration of the road look more natural.*

Last but not least, it would be interesting to study how to efficiently combine our approach with other models of representation (structures that allow 3D terrain with caves and arches, animate fluids such as water, or models for building representation and generation).

Appendix: Lipschitz constants

Here we outline the computation of the Lipschitz constants for some operators and primitives.

Wyvill blending function. We use this function for defining the weight function $\alpha(\mathbf{p})$ of some skeletal primitives. Recall that Wyvill's C^2 blending function $g(x)$ is defined as:

$$g(x) = \begin{cases} \left(1 - \left(\frac{x}{r}\right)^2\right)^3 & \text{if } x < r, \\ 0 & \text{otherwise}. \end{cases}$$

For $x < r$, we have:

$$g'(x) = \frac{-6x\left(r^2 - x^2\right)^2}{r^6}.$$

Finding the Lipschitz constant of g consists in finding the maximum of g', thus we consider the second order derivative:

$$g''(x) = \frac{-6\left(r^2 - x^2\right)\left(r^2 - 5x^2\right)}{r^6}.$$

The second order derivative has two positive vanishing values of r and $r/\sqrt{5}$. Only the latter corresponds to a maximum value of $g'(x)$. The maximum absolute value of $g'(x)$ is obtained by evaluating $g'(r/\sqrt{5})$, thus $\lambda = 96\sqrt{5}/125\,r$. □

Blending operator. Let A and B the two sub-nodes of the blending operator. Let $\beta = \alpha_A/(\alpha_A + \alpha_B)$, the blending function definition may be written as:

$$f = \beta f_A + (1-\beta) f_B, \qquad \alpha = \alpha_A + \alpha_B.$$

The domain is defined by $\Omega_0 = \{\mathbf{p} \in \mathbf{R}^2 \mid \alpha(\mathbf{p}) > 0\}$.

Let λ_A and λ_B the Lipschitz constants of f_A and f_B. We consider the domain where f_A, f_B, α_A and α_B are C^1. The Lipschitz constant λ of f is defined as $\lambda = \sup_{\mathbf{p}\in\Omega} ||\nabla f(\mathbf{p})||$. The gradient ∇f can be written as follows:

$$\nabla f = \nabla\beta\,(f_A - f_B) + \beta\,\nabla f_A + (1-\beta)\,\nabla f_B.$$

Since $\beta \in [0,1]$, $||\nabla f|| \le \max(\lambda_A, \lambda_B) + \lambda_\beta \sup|f_A - f_B|$. We need to find the Lipschitz constant λ_β of β. We have:

$$\nabla\beta = \frac{1}{\alpha_A + \alpha_B}\Big((1-\beta)\,\nabla\alpha_A - \beta\,\nabla\alpha_B\Big).$$

Thus, $\lambda_\beta = \max(\lambda_{\alpha_A}, \lambda_{\alpha_B})$ and eventually:

$$\lambda \le \max(\lambda_A, \lambda_B) + \max(\lambda_{\alpha_A}, \lambda_{\alpha_B})\,\sup|f_A - f_B|.$$

Similar results are obtained with other operators. □

References

[BA05] BELHADJ F., AUDIBERT P.: Modeling landscapes with ridges and rivers: bottom up approach. In *GRAPHITE* (2005), pp. 447–450. 3

[BF01] BENES B., FORSBACH R.: Layered data representation for visual simulation of terrain erosion. In *Proc. of the 17th Spring Conf. on Computer Graphics* (2001), pp. 80–85. 2, 11

[BF02] BENES B., FORSBACH R.: Visual simulation of hydraulic erosion. In *Journal of WSCG* (2002), vol. 10, pp. 79–86. 2

[BTHB06] BENES B., TĚŠÍNSKÝ V., HORNYŠ J., BHATIA S. K.: Hydraulic erosion. *Comput. Animat. Virtual Worlds 17*, 2 (2006), 99–108. 2

[CBS96] CRESPIN B., BLANC C., SCHLICK C.: Implicit sweep objects. *Computer Graphics Forum 15* (1996), 165–174. 4

[CMF98] CHIBA N., MURAOKA K., FUJITA K.: An erosion model based on velocity fields for the visual simulation of mountain scenery. *Journal of Vis. and Comp. Anim. 9*, 4 (1998), 185–194. 2

[dCB09] DE CARPENTIER G. J. P., BIDARRA R.: Interactive gpu-based procedural heightfield brushes. In *Proc. of Foundations of Digital Games* (2009), pp. 55–62. 12

[DGGK11] DERZAPF E., GANSTER B., GUTHE M., KLEIN R.: River networks for instant procedural planets. *Comput. Graph. Forum 30*, 7 (2011), 2031–2040. 3

[DIP14] DUPUY J., IEHL J.-C., POULIN. P.: Quadtrees on the gpu. *GPU Pro 5: Advanced Rendering Techniques* (2014), 439–450. 2, 10

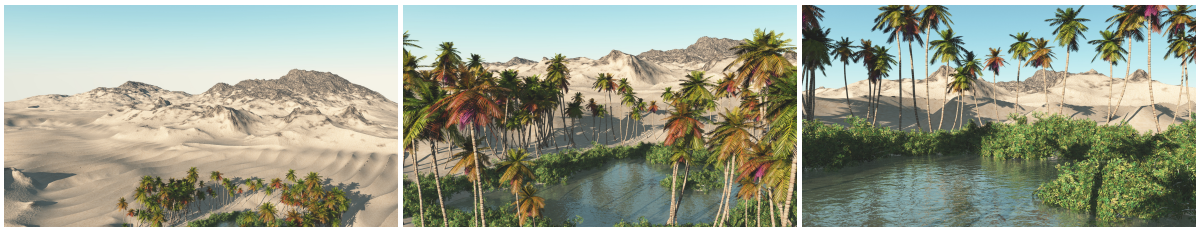

Figure 23: *The entire scene has been created with our system. We used a specific tree to represent the water body, and another one to define the density of vegetation around the oasis.*

[EMP*98] EBERT D., MUSGRAVE K., PEACHEY D., PERLIN K., WORLEY S.: *Texturing and Modeling: A Procedural Approach*. Academic Press Professional, 1998. 2, 11

[GGG*13] GÉNEVAUX J.-D., GALIN É., GUÉRIN É., PEYTAVIE A., BENEŠ B.: Terrain generation using procedural models based on hydrology. *ACM Trans. on Graph. 32*, 4 (2013), 143:1–143:13. 3, 11

[GMS09] GAIN J., MARAIS P., STRASSER W.: Terrain sketching. In *Proc. of I3D* (2009), pp. 31–38. 2, 11

[Har94] HART J. C.: Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer 12* (1994), 527–545. 7

[HGA*10] HNAIDI H., GUÉRIN É., AKKOUCHE S., PEYTAVIE A., GALIN É.: Feature based terrain generation using diffusion equation. *Computer Graphics Forum (Proceedings of Pacific Graphics) 29*, 7 (2010), 2179–2186. 2

[KB89] KALRA D., BARR A.: Guaranteed ray intersections with implicit surfaces. *SIGGRAPH Computer Graphics 23*, 3 (July 1989), 297–306. 7

[KBKS09] KRIŠTOF P., BENES B., KŘIVÁNEK J., ŠŤAVA O.: Hydraulic erosion using smoothed particle hydrodynamics. *Computer Graphics Forum 28*, 2 (2009), 219–228. 2

[KMN88] KELLEY A., MALIN M., NIELSON G.: Terrain simulation using a model of stream erosion. In *Proc. of SIGGRAPH* (1988), pp. 263–268. 3

[MDH07] MEI X., DECAUDIN P., HU B.: Fast hydraulic erosion simulation and visualization on GPU. In *Pacific Graphics* (2007), pp. 47–56. 2

[MKM89] MUSGRAVE F. K., KOLB C. E., MACE R. S.: The synthesis and rendering of eroded fractal terrains. In *Proc. of SIGGRAPH* (1989), pp. 41–50. 2, 4, 11

[Nag98] NAGASHIMA K.: Computer generation of eroded valley and mountain terrains. *The Visual Computer 13*, 9-10 (1998), 456–464. 2

[NLP*13] NATALI M., LIDAL E. M., PARULEK J., VIOLA I., PATEL D.: Modeling terrains and subsurface geology. In *Proceedings of Eurographics STAR* (2013), pp. 155–173. 2

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

[PH93] PRUSINKIEWICZ P., HAMMEL M.: A fractal model of mountains with rivers. In *Graphics Interface* (1993), pp. 174–180. 3

[RME09] RUSNELL B., MOULD D., ERAMIAN M. G.: Feature-rich distance-based terrain synthesis. *The Visual Computer 25*, 5-7 (2009), 573–579. 2

[SBBK08] ŠŤAVA O., BENES B., BRISBIN M., KŘIVÁNEK J.: Interactive terrain modeling using hydraulic erosion. In *ACM Siggraph/Eurographics Symposium on Comp. Anim.* (2008), pp. 201–210. 2

[STBB14] SMELIK R., TUTENE T., BIDARRA R., BENES B.: A survey on procedural modelling for virtual worlds. *Computer Graphics Forum 33*, 6 (2014), 31–50. 2

[TEC*14] TASSE F. P., EMILIEN A., CANI M.-P., HAHMANN S., BERNHARDT A.: First person sketch-based terrain editing. In *Graphics Interface* (2014), pp. 217–224. 2

[Teo09] TEOH S. T.: Riverland: An efficient procedural modeling system for creating realistic-looking terrains. In *Proc. of the ISVC: Part I* (2009), pp. 468–479. 3

[TGM12] TASSE F. P., GAIN J., MARAIS P.: Enhanced texture-based terrain synthesis on graphics hardware. *Computer Graphics Forum 31*, 6 (2012), 1959–1972. 2

[VBHS11] VANEK J., BENES B., HEROUT A., ŠŤAVA O.: Large-scale physics-based terrain editing using adaptive tiles on the GPU. *IEEE, Computer Graphics and Applications 31*, 6 (2011), 35–44. 2

[WCMT07] WOJTAN C., CARLSON M., MUCHA P., TURK G.: Animating corrosion and erosion. In *Proc. of the Eurographics Workshop on Natural Phenomena* (2007), pp. 15–22. 2

[WGG99] WYVILL B., GUY A., GALIN É.: Extending the CSG tree - warping, blending and boolean operations in an implicit surface modeling system. *Computer Graphics Forum 18*, 2 (1999), 149–158. 4, 5

[ZSTR07] ZHOU H., SUN J., TURK G., REHG J. M.: Terrain synthesis from digital elevation models. *IEEE Transactions on Visualization and Computer Graphics 13*, 4 (2007), 834–848. 2