Sparse representation of terrains for procedural modeling

Eric Guérin$^{1,2}$ Julie Digne$^{1,4}$ Eric Galin$^{1,3}$ Adrien Peytavie$^{1,4}$

1 Université de Lyon, CNRS 2 INSA-Lyon, LIRIS, UMR5205, F-69621, France
3 Université Lyon 2, LIRIS, UMR5205, F-69676, France 4 Université Lyon 1, LIRIS, UMR5205, F-69622, France

Abstract

In this paper, we present a simple and efficient method to represent terrains as elevation functions built from linear combinations of landform features (atoms). These features can be extracted either from real world data-sets or procedural primitives, or from any combination of multiple terrain models. Our approach consists in representing the elevation function as a sparse combination of primitives, a concept which we call Sparse Construction Tree, which blends the different landform features stored in a dictionary. The sparse representation allows us to represent complex terrains using combinations of atoms from a small dictionary, yielding a powerful and compact terrain representation and synthesis tool. Moreover, we present a method for automatically learning the dictionary and generating the Sparse Construction Tree model. We demonstrate the efficiency of our method in several applications: inverse procedural modeling of terrains, terrain amplification and synthesis from a coarse sketch.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Modeling – Natural Phenomena—Modeling – Procedural Modeling

1. Introduction

Despite the considerable progress towards efficient methods for modeling realistic terrains, generating large terrains with a high level of detail remains an open problem in Computer Graphics.

Existing terrain modeling techniques can be categorized into procedural, erosion-based, sketch-based, and example-based approaches. Erosion simulation and hydrology-based algorithms generate geologically correct models, but often do not provide user-control and are computationally demanding. Sketch-based methods involve manual editing that can be tedious while example-based algorithms are limited by the provided input exemplars. Furthermore, all those methods share a common major drawback: their scalability. Indeed, the generated terrains usually represent only features at a single scale that are stored in a discrete regular heightfield.

In contrast, fractal-based and function-based methods are computationally efficient as they define the elevation of the terrain using a continuous procedural function. Yet they fail to provide control over the terrain features.

A fundamental aspect of our work is coming from the observation that real landscapes can be described as a combination of a small set of landform features at different scales. This observation lies also at the foundation the sparse modeling theory, which has gathered a tremendous amount of research effort in various disciplines over the past ten years. The key idea of sparse modeling borrows from the compressive sensing theory [CRT06] which can be...
roughly summarized as follows: Given a set of signals there exists a space with low dimension especially well suited for their analysis. The goal of this theory is to find such a subspace, or, simply put, find a solution to some problem, denoising for example, while enforcing this solution to lie in the small subspace. By means of this theoretical framework, we propose a method to design realistic terrains from a well-chosen set of primitives.

For this purpose, we introduce a novel compact hierarchical sparse terrain model, called Sparse Construction Tree, that combines atoms taken in a dictionary that stores the different characteristic landform features. Our representation extends the function based model presented in [GGP+15] and defines continuous terrains with varying level of detail. It is based on a tree structure that combines different kinds of primitives using operators. Instead of authoring the terrain interactively by editing and combining landform primitives by hand, we propose a generic inverse procedural modeling approach that automatically learns and creates the dictionary of atoms, optimizes it with a view to reducing its memory footprint, and builds a Sparse Construction Tree from input elevation maps. Our method can handle synthetic (procedural) and real datasets, and any combination of multiple terrain models.

Our framework allows us not only to solve the inverse procedural modeling of terrain problem, but also to propose amplification techniques that can be applied to sketched input terrain or real world data-sets to automatically generate large scale and highly detailed terrain models. Although we do not focus on terrain compression, a benefit of our representation is the ability to generate large terrains with a high level of details with a small memory footprint. Finally, our sparse terrain representation lends itself for interactive editing of large terrains with a high level of detail. Combined with intuitive editing tools that focus on placement and distribution of specific landform features, the Sparse Construction Tree representation provides global and local control (Figure 1).

2. Related work

In this section, we review the existing approaches for terrain modeling and focus on the level of user-control, efficiency, realism and scale range. For a complete overview of recent approaches and representations used in terrain modeling we refer the reader to [NLP13].

Erosion simulation was introduced by [MKM89] and is often used as a post processing step to fractal and procedural generation methods with a view to adding realism. This work was later extended and refined in [Nag98, CMF98]. The majority of those techniques rely on discrete regular heightfields [CBC+16], layered data structures [BF01] or a volumetric data [BTHB06]. A major limitation of erosion-based techniques 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 using the GPU [MDH07, VBHS11].

Various volumetric approaches apply small-scale erosion models such as Voronoi-based block erosion for modeling cliffs [PGMG09], and spheroidal erosion for modeling of goblins [BFO+07]. Those methods rely on three dimensional structures such as voxels or material stacks, are computationally demanding and do not lend themselves for modeling large scenes.

**Interactive editing** techniques rely on high-level control tools to allow the generation of a complete terrain from a limited set of user-defined constraints. 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 [ŠBBK08, VBHS11] are also limited to small scenes and to editing existing terrains.

**Synthesis by example** approaches borrow from texture synthesis and combine realism and high-level user-control by generating new terrains from patches extracted from exemplars. A first method proposed in [ZSTR07] extract heightfield patches from a terrain exemplar and combines them according to a user-painted coarse map. These approaches were later improved to enable the combination of user-defined maps with silhouette strokes defining roughness and [GMS09, GMM15]. Sketched terrain silhouettes were used to deform an existing terrain to conform to a view from a certain viewpoint [TEC+14].

A common limitation of erosion simulation, sketch- and example-based methods is that they rely on a discrete regular grid and cannot create large terrains with details. Additionally, they do not allow for a precise user-control. In contrast, terrains that are mathematically defined as a combination of simple functions can be processed efficiently and have a theoretical infinite precision. Moreover, their representation and storage is compact.

**Procedural models** are generally based either on fractals or on a combination of noise-based functions and exploit the observation that terrain landform features repeat at different scales and different positions in space [EMP+98]. Several procedural subdivision techniques were proposed to automatically generate river networks [KMN88, PH93, BAO5, DGG11].

Although those algorithms can generate infinite landscapes with unlimited precision, they only provide an indirect global control and produce terrains without any underlying geographical structure. To improve user control of fractal-based methods, several algorithms generate terrains from feature curves that specify ridges and river networks [KMN88, HGA+10, GGG+13]. Another function-based approach [GGP+15] was recently proposed to define terrains as a hierarchical construction tree combining primitives with blending, carving and warping operators.

A major limitation of those procedural models is that there is no algorithm that would allow for simple and efficient editing of large scale complex terrains with high level of detail. The hierarchical construction tree described in [GGP+15] requires the user to focus on the definition and distribution of many different primitives by hand to create large terrains. In contrast, our sparse tree representation provides us with a means to automatically define a set of primitives by learning a dictionary from a set of exemplars, and automatically distribute them to generate large scenes.
Sparse Modeling. In our work, we propose to rely on sparse modeling to define a versatile model able to address the previous limitations. Sparse modeling aims at representing complex signals such as natural image patches as linear combinations over a small dictionary. These linear combinations are forced to be sparse, which means that each patch will only need a few subset of the dictionary atoms in its linear combination. Recent advances in sparse modeling led, among others, to new denoising methods [EA06], as well as texture synthesis [Pey09]. In geometry processing, the sparse approach also led to new developments such as surface reconstruction [XZZ*14] or surface compression [DCV14]. Sparse representation can also be linked to local surface representation and parameterization [ZDL*14, JBPS11].

Our method relies on the same tools that have been designed for signal processing and adapted to image and surface processing. Such tools had never been used for terrain modeling before. In this paper, we show that sparse models provide an efficient mathematical framework for solving complex problems such as inverse procedural modeling and terrain synthesis and amplification.

3. Overview

Breaking with usual terrain modeling methods, we propose a versatile framework that addresses several complex problems in a unified way. This framework is based on a novel concept, the Sparse Construction Tree, a description of the terrain generation process by sparsely combining a well-chosen small set of terrain dictionary atoms (Figure 2). To the best of our knowledge, it is the first time that the sparsity principle is used for terrain representation and synthesis. Armed with this Sparse Construction Tree, we can represent any kind of terrain in a compact way and handle large scale areas at a high precision. Moreover, our model lends itself for a variety of applications such as inverse procedural modeling, terrain amplification as well as multi-resolution terrain editing and control, in a unified manner.

The remainder of this paper is organized as follows. Section 4 defines the Sparse Construction Tree model and presents its construction. Section 5 presents several applications of this concept to inverse procedural modeling, terrain synthesis and amplification and compression. Finally Section 6 gives implementation details and shows the results of our novel terrain modeling framework.

4. The Sparse Construction Tree

Our terrain model is defined by a construction tree which extends the model presented in [GGP*15] by introducing several important new features. The leaves of the tree act like primitives, each generating a portion of the terrain containing similar landforms features on a compact support. The internal nodes are operators that combine and aggregate their sub-trees. Therefore a terrain is represented by a continuous functions \( f : \Omega \to \mathbb{R} \) and the elevation of a point \( p \) depends on the evaluation of the hierarchical combination of every primitive in the tree. The tree representation allows to adapt the evaluation of the elevation function \( f \) according to a continuous level of detail. Finally, the construction tree implements a bounding volume hierarchy that allows for efficient processing of elevation queries. We first give the formal definition of the Sparse Construction Tree, then we explain how to build it from an input terrain \( T \).

4.1. Model definition

A Sparse Construction Tree is organized as a weighted Directed Acyclic Graph (DAG) structure (Figure 3) whose edge weights represent coefficients in linear combination. The upper nodes of the tree consist in blending operators, whereas the \( n \) leaves of the tree are sparse primitives.

Primitives are entirely constructed from atoms at the bottom layer which represent the \( N \) atoms of a dictionary denoted as \( D \). Each primitive has only a few children corresponding to a sparse linear combination of the dictionary atoms. Such a primitive will be called Sparse Primitive. Let \( \Omega \subset \mathbb{R}^2 \) denote a compact subset of \( \mathbb{R}^2 \). A terrain is formally defined as a global elevation function \( f \)
defined on $\Omega$ with values in $\mathbb{R}$:

$$f(p) = \sum_{i=0}^{n-1} f_i(p) \alpha_i(p) / \sum_{i=0}^{n-1} \alpha_i(p) \quad p \in \Omega$$

where the $f_i$ are the $n$ sparse primitives with the corresponding weighting functions $\alpha_i$.

**Sparse primitives** are characterized by their elevation function $f_i$ and corresponding weighting function $\alpha_i$ defined over a compact support $\Omega$. As a sparse linear combination of dictionary atom functions. In our system, the $j^{th}$ sparse primitive is defined as a portion of terrain defined over a disk-shaped support parameterized by its center $c_i \in \mathbb{R}^2$, the elevation at its center $z_i$ and a vector of coefficients denoted as $\mathbf{X}_i = \{x_i^j\}$, with $j = 1, \ldots, n$ (Figure 3). The coefficients are used to define the elevation function $f_i$ as a linear combination of atomic elements, referred to as $d_i$:

$$f_i(p) = z_i + \sum_{j=0}^{N-1} x_i^j d_i(p - c_i)$$

The functions $\alpha_i$ are defined as $C^1$ continuous decreasing radial functions over a disc-shaped compact support of constant radius $R$, denoted as $\Omega_i = B(c_i, R)$. They limit the influence of the primitive and control the way they are combined by operators in the construction tree. We define the weight function as a composition of the distance to the center $c_i$ with a smooth fall-off filter function $g$:

$$\alpha_i(p) = g(||p - c_i||/R) \quad g(r) = \begin{cases} (1 - r^2)^3 & \text{if } r < 1 \\ 0 & \text{otherwise} \end{cases}$$

**A Dictionary** is a set of terrain atoms, i.e., portions of terrains that define the functions $d_i$. Atoms are defined as core continuous elevation functions that are linearly combined together to define the elevation of a primitive. They can be either data-based or function-based. In the first case, each atom represents a vectorized patch of an image. Data-based atoms are implemented as discrete regular grids of elevation values and their corresponding elevation functions are computed using bi-cubic interpolation. Function-based atoms they are expressed in a functional space as a linear combination of basis functions from $\mathbb{R}^2$ to $\mathbb{R}$ (Figure 4). In our system function-based atoms are defined as sums of ridged noise as described in [EMP'98]. In both cases, they can be expressed as a vector of coefficients of dimension $K$ that represents respectively the number of elevations samples, and the dimension of the function space. Recall that $N$ denotes the number of dictionary atoms represented as vectors. The dictionary can be represented as a $K \times N$ matrix $D$ where each column $d_i$ represents the parameters characterizing the atom.

In this work we will further assume that the vectors (resp. functions) $d_i$ are normalized with respect to the $L^2$ norm of vectors (resp. functions). Importantly enough, there is no constraint on the orthogonality of the dictionary atoms. Moreover, the fact that our model can work with either analytic or discretized atoms further increases the pliability of our model and will be exploited in particular for solving the inverse procedural modeling problem.

**A multi-resolution dictionary** is a set of two dictionaries denoted as $(H, L)$, high- and low-resolution, with the exact same number of atoms and such that there exists a one-to-one correspondence between the atoms of both dictionaries. Thus, given a decomposition over the low resolution dictionary $L$, the decomposition on the high resolution dictionary $H$ can be obtained simply by keeping the decomposition coefficients $X$ and replacing the atoms through the one-to-one correspondence.

**4.2. Sparse terrain construction**

Given an input terrain denoted as $T$, the goal is to build the corresponding sparse representation. This step is performed by extracting patches from the terrain, followed by a dictionary generation step. Recall that the elevation function $f$ of the terrain is defined as the weighted sum of sparse primitives $f_i$. We need to decompose the domain of the terrain into overlapping patches. The patches can be placed at different positions over the terrain, as long as they cover it completely. In our algorithm, we rely on disc-shaped primitives distributed over a regular grid (Figure 5). The distribution guarantees that the weighting function never vanishes onto the domain of the terrain:

$$\forall p \in \Omega : \sum_{i=0}^{n-1} \alpha_i(p) = \sum_{i=0}^{n-1} \alpha(p - c_i) \neq 0$$

Every primitive can use only a subset of the dictionary atoms:
\[ \|X_i\|_0 \] represents the number of nonzero coefficients, also called sparsity and later referred to as \( s \). Although sparsity is traditionally referred to as the \( \ell_0 \) norm, it is important to remember that it is not a norm, since it is not absolute homogeneous. By convention, we normalize the energy of each atom of the dictionary and remove its constant component.

Because only a few sparse primitives overlap for every point in the support \( \Omega \), the sparse construction tree lends itself for efficient acceleration schemes such as bounding volume hierarchies that allows for efficient elevation queries.

From a given input terrain \( T \), building a sparse construction tree consists first in extracting terrain patches and describing each of these patches using the dictionary and a given sparsity constraint \( s \).

**Patch extraction.** We want to express a small part of the terrain at a given position \( \mathbf{c}_i \) based on a weighting function \( \omega_i \) whose compact support is \( \Omega_i \). We first extract the constant elevation component of the local patch by computing its average value:

\[ z_i = \int_{\Omega_i} T(p) \, dp \, dy / \int_{\Omega_i} \, dp \, dy \]

We define the patch \( T_i \) as: \( T_i(p) = T(p + \mathbf{c}_i) - z_i \) which guarantees that the patches are all defined in the same local frame. The patch is centered so that the constant component is not encoded on the dictionary.

**4.3. Sparse decomposition**

Given a dictionary \( D \), representing a terrain involves extracting the patches \( T_i \) and computing their decompositions on the dictionary. More precisely, we want to express \( T_i \) as a linear combination of \( s \) dictionary atoms, \( s \) being the target sparsity constraint, while minimizing the representation error. Since such least-squares problems with \( \ell_0 \) penalty are known to be NP-hard, we must use an approximate algorithm to solve this problem.

**Orthogonal Matching Pursuit.** In this paper, we rely on the Orthogonal Matching Pursuit (OMP) algorithm [MZ93]. In a nutshell, the method consists in finding the atom that maximizes the projection, removing its projection and iterating until the target sparsity is obtained. Formally, the algorithm processes each signal \( T_i \) independently. For each \( T_i \), a set of used atoms indices \( \hat{\Gamma}_i \) is initialized to \( \emptyset \). At each iteration, the following steps are performed:

1. Select the coordinate \( \hat{k} \) leading to the smallest residual:

\[ \hat{k} = \arg \min_{k \in \hat{\Gamma} \cup \hat{\Gamma}_i} \| T_i - \mathbf{D}_{\hat{\Gamma} \cup \hat{k}} \tilde{\beta} \|_2^2. \]

2. Update the active set \( \hat{\Gamma} \) and the solution \( \mathbf{X}_i \) as:

\[ \hat{\Gamma}_i = \hat{\Gamma} \cup \{ \hat{k} \} ; \quad \mathbf{X}_i^T = \tilde{\beta} \]

and \( \mathbf{X}_i^j = 0 \), where \( \mathbf{X}_i^T \) (resp. \( \mathbf{X}_i^j \)) are the coefficients of vector \( \mathbf{X}_i \) with indices in \( \hat{\Gamma} \) (resp. \( \hat{\Gamma}_i \)).

The iterations stop when \( \hat{\Gamma} \) has reached the desired size, i.e. \( |\hat{\Gamma}| = s \). The output of the OMP algorithm process is depicted in Figure 6. The error decreases as the number of atoms increases which allows the algorithm to better reconstruct the input signal. At the end of this process, we have a decomposition of each signal on the given dictionary with the specified sparsity, and the terrain can be reconstructed from this decomposition.

**4.4. Dictionary learning**

The previous decomposition does not account for the possible redundancy in the initial terrain since \( s = n \). To optimize the dictionary with respect to a given terrain, we consider \( N < n \) and \( s < N \). In this setting, the dictionary contains a reduced number of atoms and each patch of the terrain is described as a linear combination of a reduced number of atoms.

For the sake of readability, we will identify the signals and dictionary atoms to their vector of coordinates: \( \mathbf{d}_i = (d_{ik}) \) and \( \mathbf{f}_i = (f_{ik}) \) with \( i = 1, \ldots, N \), \( k = 1, \ldots, K \). The dictionary learning problem is formally defined as a minimization problem:

\[
\min_{\mathbf{D} \in \mathbb{R}^{n \times K}, \mathbf{X} \in \mathbb{R}^{n \times s}} \| \mathbf{F} - \mathbf{D} \cdot \mathbf{X} \|_F^2 \quad \text{such that } \forall i = 1, \ldots, n, \| \mathbf{X}_i \|_0 \leq s
\]

\( \mathbf{F} \) is the \( K \times n \) matrix whose columns are the primitives of the terrain to represent, \( \mathbf{X} \) is the matrix of the decomposition coefficients and \( \mathbf{X}_i \) is the \( i^{th} \) column of \( \mathbf{X} \); i.e. the coefficients of the decomposition of \( f_i \) on \( \mathbf{D} \). The constraint ensures that each \( f_i \) uses at most \( s \) dictionary atoms for its decomposition.

Several algorithms exist for solving approximately problem 1. Among those, we rely on the K-SVD algorithm [AE90]. Starting
with an initial dictionary composed of randomly selected primitives in the set of \( n \) primitives, the algorithm alternates between the following two steps until the error converges or a given number of iteration has been reached.

1. Given the dictionary \( D \), find the best sparse representation \( X \) efficiently by learning the dictionary and the set of sparse coefficients. Thus, instead of storing a complete height map, we store the set of patch centers \( c_i \), the sparse coefficients \( X \) and the small dictionary \( D \). In our implementation, the patches centers \( c_i \) are located on a regular grid which enables us to avoid to store their positions explicitly. Moreover, since the coefficient matrix is sparse, it can be encoded efficiently by keeping track of the nonzero coefficients positions, thus storing only a few couples \( (i, x_i) \).

Let \( w \times h \) the size of the elevation data map \( T \). Assuming the grid step is half the patch radius, i.e., the patches overlap, the total number of values to store the map completely can be defined as \( t_0 = w \times h = nK/4 \) where \( n \) denote of the number of patches and \( K \) the size of each patch.

The representation of the same terrain using a dictionary \( D \) of size \( N \) with a sparsity \( s \) can be optimized by storing the dictionary \( D \) and the coefficients \( X \). The dictionary is represented by a \( N \times K \) dense matrix, thus requiring a storage of \( N \times K \) values. The coefficient matrix \( X \) is a \( n \times N \) sparse matrix, thus by encoding it in compressed row storage format, we only need to store \( n(2s+1)+1 \) values. In the general case, the coordinates of the patches centers should be stored. Since we rely on a distribution over a regular grid, only the height \( z_j \) should be stored to reconstruct the terrain. Thus, the total number of values is:

\[
t(s,N) = KN + n(2s+2) + 1
\]

Recall that the sparsity \( s \) (which is the number of nonzero coefficients \( x_i \)) is much smaller than the number of dictionary atoms \( n \) and the patch size \( K \), which are in turn far smaller than the number of terrain patches \( n \). Therefore we have \( s \ll N \ll n \), and \( N = O(K) \) which leads to a compact representation:

\[
t(s,N) = KN + n(2s+2) + 1 \ll nK/4
\]

In the remainder of this paper \( \rho = t(s,N)/t_0 \) will refer to the storage ratio between our sparse model and the original input terrain.

<table>
<thead>
<tr>
<th>Terrain</th>
<th>Size ( t_0 )</th>
<th>Sparsity ( s )</th>
<th>Ratio ( \rho )</th>
<th>PSNR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alpes (512 ( \times ) 512)</td>
<td>0.26M</td>
<td>1</td>
<td>14.9 %</td>
<td>23.7</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>20.0 %</td>
<td>26.4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>50.9 %</td>
<td>31.1</td>
<td></td>
</tr>
<tr>
<td>Italy (3600 ( \times ) 3600)</td>
<td>12.96M</td>
<td>1</td>
<td>4.9 %</td>
<td>31.7</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>9.7 %</td>
<td>34.2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>38.0 %</td>
<td>39.2</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Initial storage size \( t_0 \), sparsity \( s \), storage reduction ratio \( \rho \) and PSNR (in dB) of two terrains for different sparsity values. The computations were made with a dictionary of 128 atoms.

**Control parameters.** In this application, we focus on the quality of the reconstructed terrain and want to minimize the error. This can be achieved by using relatively small values for the patch size \( K \sim 10^2 \).

Table 1 reports the storage ratio \( \rho \) and the rate-distortion results for terrains encoded on dictionaries with various sizes and sparsity. Our method can represent large terrains at a small memory
cost with good PSNR even for very small sparsity values. Results demonstrate that large terrains of hundreds of thousands square kilometers at a resolution of a few meters can be represented efficiently by our Sparse Construction Tree in a few megabytes instead of gigabytes needed for the entire elevation map. The Italy terrain compressed with JPEG2000 using a compression ratio $\rho$ similar to the one obtained with a sparsity of 1 gives a PSNR of 20dB. Our approach compares favorably with a PSNR of 31.7 dB.

5.2. Inverse procedural modeling

Our method proposes a solution to the inverse procedural modeling of terrain problem. Given an input terrain represented by a discrete elevation map $\mathcal{T}$, we want to build a corresponding procedural generative model approximating it. We propose to use our Sparse Construction Tree construction algorithm operating with a function-based dictionary.

Let us consider a terrain represented by its matrix of discrete patches $\mathcal{F}$ as input and a set of functions given by sums of ridged noise functions as presented in Section 4. Sampling this set of functions leads to a (potentially large) analytical dictionary $\mathcal{D}$. Through the discretization of dictionary $\mathcal{D}$, we can define a discrete dictionary $\tilde{\mathcal{D}}$. Both dictionaries are by construction in one-to-one correspondence (Figure 7), a property that we want to exploit here.

The input heightfield is then decomposed over the discrete dictionary $\mathcal{D}$ using Orthogonal Matching Pursuit which gives a sparse matrix of coefficients $\mathbf{X}$. Finally using the correspondence between the discrete atoms and the analytical atoms we can reconstruct an analytical terrain $\tilde{\mathcal{F}}$ by computing:

$$\tilde{\mathcal{F}} = \tilde{\mathcal{D}} \cdot \mathbf{X}$$

A limitation of this process is that $\mathcal{D}$ (and thus $\tilde{\mathcal{D}}$) can be huge. To overcome this limitation we can use the constrained dictionary learning problem 2 to learn a dictionary while ensuring that the atoms are exact atoms of the noise. Hence we are able to revert any atom index to a procedural noise function.

**Functions-based primitives.** We compared the effectiveness of our inverse procedural algorithm by using different types of functions that are often used for generating synthetic terrains: Perlin noise, Ridged and Ridged multi-fractal noise. Experiments demonstrate that mountainous terrains are better approximated by ridged noise than by standard noise and gives better PSNR values (Figure 8). This result is coherent as ridged noise allows the generation of crests and ridges that appear in natural terrains.

**Control parameters** Our goal is to preserve the shape of the input terrain approximation, therefore we used small patch sizes $K \approx 10^7$. Since we rely on function-based atoms, we can sample them at any resolution. Depending on the type of the functions and the terrain to approximate, the user can adapt the value of the patch radius $R$ so that it should capture the details of the input terrain. We do not constrain the number of atoms $N$ because only the parameters of the functions-based atoms need to be stored.

5.3. Terrain synthesis and amplification

Given an input coarse terrain $\mathcal{T}$ and a high resolution exemplar $E$, terrain synthesis and amplification aim at producing a new detailed high resolution terrain that preserves the global aspect of the coarse map. The resolution of the initial terrain will be increased by adding details from the exemplar.

If $\mathcal{T}$ is a coarse user-painted sketch, we will call this process **Terrain Synthesis** whereas if the input is a low-resolution digital elevation map obtained from real-world data, we will call this process **Terrain Reconstruction**.

### Figure 7: Pipeline for the inverse procedural modeling of terrains: given an input elevation map and an analytic dictionary $\mathcal{D}$ we create a discrete representation of the dictionary $\tilde{\mathcal{D}}$, before performing the OMP over $\tilde{\mathcal{D}}$ and applying reconstruction process using $\tilde{\mathcal{D}}$.

### Figure 8: Reconstruction of a terrain with a sparsity of 1 from an analytical dictionary $\mathcal{D}$ extracted from a Perlin noise (left) and a ridged multi-fractal noise (right). The ridged noise better preserves the pikes and sharp ridges and has a smaller error (PSNR is 20.0 dB for the Perlin noise, 24.9 dB for the Ridged noise).

### Figure 9: Overview of the amplification/synthesis process: starting from an input low resolution terrain or user-painted sketch, we use the multi-resolution dictionary $\mathcal{D}$ in the OMP and reconstruction process to generate a new high resolution terrain.
Table 2: Statistics for both synthesis and amplification on parts of the terrains shown in Figure 12 and Figure 16. Storage size and timings (in seconds) are reported for different sizes of atoms. Sparsity was always set so as to 1 to provide a good reconstruction. Comparisons are made between two cases: using a complete dictionary or a dictionary learned from real terrain elevation maps.

<table>
<thead>
<tr>
<th>Terrain</th>
<th>Type of dictionary</th>
<th>Initial size</th>
<th>Final size</th>
<th>Ratio ρ</th>
<th>N</th>
<th>Learning time</th>
<th>Synthesis time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synthesis</td>
<td>Complete</td>
<td>64 × 128</td>
<td>4096 × 8192</td>
<td>9.6%</td>
<td>60000</td>
<td>0.0</td>
<td>25.5</td>
</tr>
<tr>
<td></td>
<td>Learned</td>
<td></td>
<td></td>
<td>0.48%</td>
<td>64</td>
<td>5.2</td>
<td>10.2</td>
</tr>
<tr>
<td>Amplification</td>
<td>Complete</td>
<td>32 × 32</td>
<td>8192 × 8192</td>
<td>19%</td>
<td>120000</td>
<td>0.0</td>
<td>84.0</td>
</tr>
<tr>
<td></td>
<td>Learned</td>
<td></td>
<td></td>
<td>0.23%</td>
<td>256</td>
<td>30.0</td>
<td>27.0</td>
</tr>
</tbody>
</table>

Terrain Amplification. In that case, the exemplar \( \mathcal{E} \) and the terrain \( \mathcal{T} \) should be consistent, i.e. the data should represent similar terrains. In general terrain synthesis targets authoring from a coarse sketch (Figure 1). In contrast, terrain amplification is useful for visualizing real terrains at a higher level of detail than the original sampling.

Figure 10: Multi-resolution dictionary construction.

The algorithm (Figure 9) can be decomposed in three steps:

1. A multi-resolution dictionary \((\mathbf{H}, \mathbf{L})\) is built from the high resolution exemplar \( \mathcal{E} \).
2. The input terrain \( \mathcal{T} \) is decomposed over the low-resolution dictionary \( \mathbf{L} \) yielding the sparse matrix \( \mathbf{X} \).
3. A high resolution Sparse Construction Tree \( \mathcal{S} \) is generated by combining the same matrix of coefficients \( \mathbf{X} \) and the high resolution dictionary \( \mathbf{H} \).

The first step builds a high resolution dictionary from the exemplar. Then, every atom in this dictionary is down-sampled to create a low-resolution dictionary such that \( \mathbf{H}_i \) matches \( \mathbf{L}_i \) for all \( i \in 1, \ldots, n \) (Figure 10). The down-sampling process is crucial to guarantee that the atoms of \( \mathbf{L} \) are smooth, i.e. contain less features and therefore form an appropriate basis for decomposing the low-resolution terrain \( \mathcal{T} \) over \( \mathbf{L} \), which is performed as the second step of the algorithm. Finally, we substitute the low-resolution atoms by the corresponding high-resolution atoms in the reconstruction yielding the final high resolution Sparse Construction Tree. Results (Figure 1) show that our method can be used to automatically transfer the details of the exemplar to the synthesized terrain.

Table 2 reports the memory cost ratio for storing a detailed height map compared to storing a low-resolution terrain and a high resolution exemplar. We also give the computation time for synthesizing or amplifying the terrain using either a complete dictionary (i.e. all the patches of the exemplar) or a dictionary learned using the optimization problem 2. Timings show that the synthesized terrain can be computed quite efficiently since the sparsity is \( s = 1 \), and even faster with a learned dictionary. Instead of storing the complete dictionary, it is more efficient to store only the exemplar which by definition contains all possible patches, which results in a reduced memory footprint, even in the case of a complete dictionary.

Control parameters In terrain synthesis and amplification applications, it is less crucial to capture the details of the input terrain or sketch exactly. Thus, we use large values for the patch size \( K \sim 10^3 \). The number of atoms in the dictionary can be set to large values \( N \sim 10^5 \) to increase amplification and synthesis possibilities.

5.4. Authoring and Control

The Sparse Construction Tree provides an efficient, consistent and unified framework for authoring complex and large terrains with a high level of detail. Its hierarchical architecture combining landform primitives with different operators allows to define entire regions in space as different sparse construction sub-tree models which can be combined together or with other procedurally defined primitives such as rivers, mountains or roads. Consistency comes from the fact that Sparse Primitives define a continuous elevation function \( f(\mathbf{p}) \) over their domain and can be evaluated naturally when recursively traversing the tree structure.

© 2016 The Author(s)

Computer Graphics Forum © 2016 The Eurographics Association and John Wiley & Sons Ltd.
The radius of the atoms $R$ provides the user with another intuitive means to control the shape of terrain produced by the Sparse Construction Tree. For small values of $R$, the shape of input terrain $T$ is almost preserved. As the radius increases, the influence of $T$ decreases while the impact of the exemplar increases (Figure 11). The size of the patch $K$ represents the number of values and is defined as $K = \lceil (4R/\delta)^2 \rceil$ where $\delta$ denotes the discretization of the patch. $\delta$ is provided for data-based terrains and chosen by the user for continuous function-based terrains.

Finally, scene editing can be performed efficiently by adding new nodes in the tree. Figure 13 shows how a river was created at the bottom of a canyon by simply combining the initial terrain with a carving operator and river primitives. The Sparse Construction Tree represents a sub-tree with 3 levels and modifications are performed by performing operations on top of this sub-tree. Authoring is simple and provides a direct local control that comes in addition to the global control offered by synthesis and amplification techniques.

6. Results and discussion

We implemented our framework in C++ and MatLab. Experiments were performed on a desktop computer equipped with © Intel Core i7, clocked at 3 GHz with 16GB of RAM. The output of our system was directly streamed into E-on Vue xStream © to produce the images shown throughout the paper (Figures 1, 12 and 16).

Evaluation. The Sparse Construction Tree model can generate and store large terrains with a high level of detail that would otherwise be impossible to store in memory. A key feature of our approach is its versatility: our model can be used to represent any kind of terrain in an extremely compact form. Moreover, it allows to reproduce different geomorphological styles and landform features in a unified and consistent framework.

Control. A major contribution of our approach is its simplicity. More importantly, the multi-resolution structure embedded in our Sparse Construction Tree provides the user with a control over synthesized terrain at different scales. It allows him to select styles among a set of multi-resolution dictionaries either derived from existing real terrain data or computed from procedurally defined primitives. This control is further augmented by the ability to add some landform features such as rivers (Figure 12) by simply adding new primitives and nodes in the tree. Building a dictionary from several terrains or by mixing analytical functions with real terrain data is also possible. Figure 14 shows different examples combining atoms from different sources. The algorithm automatically chooses which data-based or analytical atoms best fit the input terrain.

Performance. Our method generates a vector-based representation of large terrains (tens of thousands of square kilometers) at
a very high resolution (one meter) in a few seconds. A crucial aspect of our Sparse Construction Tree model is its native multi-resolution support. We can visualize the terrain at multiple scales, and we can use view-dependent clipping algorithms or resource-dependent strategies. The dictionary-based description of the generated terrain is extremely compact and allows the storage of large terrains as shown in Table 1. Most of the operations performed in our algorithms rely on linear algebra and can benefit from standard parallelization techniques.

**Figure 15:** Standard terrain synthesis techniques such as procedural noise applied to amplification (left) give unstructured results, whereas our sparse amplification method (right) generates coherent land-form features and patterns that conform to real terrain.

Comparison to other techniques. Procedural amplification techniques often consists in adding a multi-fractal noise whose amplitude depends on the altitude and the slope at every point of the input terrain. Figure 15 shows an input terrain (left) that has been amplified using such a method (center) and our sparse amplification technique (right). Our method compares favorably to noise-based procedural functions as it generates land-form features learned from real-data and produces details such as ridges that are coherent with the global shape of the terrain.

Our method extends the construction tree presented in [GGP*15] where landscapes were created by hand carefully setting and tuning the parameters of less than one hundred primitives. In contrast, our inverse procedural approach automatically generates large terrains with tens of thousands of primitives by learning from exemplars. Our approach is complementary to the river generation technique presented in [GGG*13] where landform primitives are defined as the result of a procedural growth process. It can be used as a post processing step after the large scale river network generation. Our approach has the advantage that the selection of primitives is automatically performed by learning the dictionary from real examples. Our approach compares favorably in terms of memory footprint: we can build complex terrains as a combination of a few elements taken in small dictionary, whereas the procedural river generation technique relies on the definition of tens of thousands primitive with many specific parameters.

Limitations. The main limitation of our approach is that it cannot guarantee that the generated terrains will be geomorphologically consistent. However, it is possible to partially overcome this problem by using hydrology consistent coarse elevation input terrains. While the amplification and the terrain synthesis processes do not guarantee that coherence will be preserved at smaller scales, we observed that the produced terrains were consistent as long as the radius $R$ of the weighting function $\phi_i$ was small enough with respect to the size of the landform features of the input terrain.

7. Conclusion

We have introduced a general framework for representing and generating terrains. Our model combines a hierarchical primitive-based tree structure with the sparse representation theory into a coherent and unified framework. It can represent large terrains at a very high resolution in an compact fashion. The final geometric model is a hierarchical sparse combination of atoms in a dictionary and can be augmented with a variety of primitives for representing different specific landform features at different scales including hills, mountains, valleys, and water-courses with highly detailed geometry at varying scales. Moreover, it can serve as a basis for inverse procedural modeling, terrain synthesis and amplification.

There are many possible extensions of this work. At the heart of our method lies a dictionary-based sparse representation: we could provide the user with enhanced control by allowing to specify and blend different terrain styles in the synthesis and amplification processes. Improving the speed of the method allows interactive editing: embedding our terrain representation in an interactive and intuitive editor to increase controllability would be worth investigating. Another possible extension would be to include the generation of vegetation in the same process. We could learn from exemplars and describe the distribution of trees and vegetation on the terrain by using a similar dictionary-based sparse model. Finally, our method is currently limited to elevation functions and cannot handle terrains with cliffs or overhangs. Generalizing our approach to model volumetric terrains with different materials is a promising avenue of research we are currently investigating.

Acknowledgments

This work is part of the project PAPAYA funded by the Fonds National pour la Société Numérique. The algorithms were implemented in the Arches framework supported by the LIRIS/CNRS. We would like to credit E-on software for providing Vue xStream for rendering our terrain models.

References


© 2016 The Author(s)

Computer Graphics Forum © 2016 The Eurographics Association and John Wiley & Sons Ltd.
Figure 16: This amplification example was generated using a multi-resolution dictionary created from a real data-set. The original digital elevation map was taken from Australia at ε = 1 km resolution. The final sparse-terrain model has a resolution of ε = 4 m and was generated by applying 4 successive steps of amplification with a resolution increase factor of 4, yielding a total amplification factor of 256. Procedural texturing and vegetation were applied to outline ground details such as ridges and erosion patterns generated by the amplification process.


