Feature-based Volumetric Terrain Generation

Michael Becher*, Michael Krone†, Guido Reina‡, Thomas Ertl§
Visualization Research Center (VISUS), University of Stuttgart, Germany

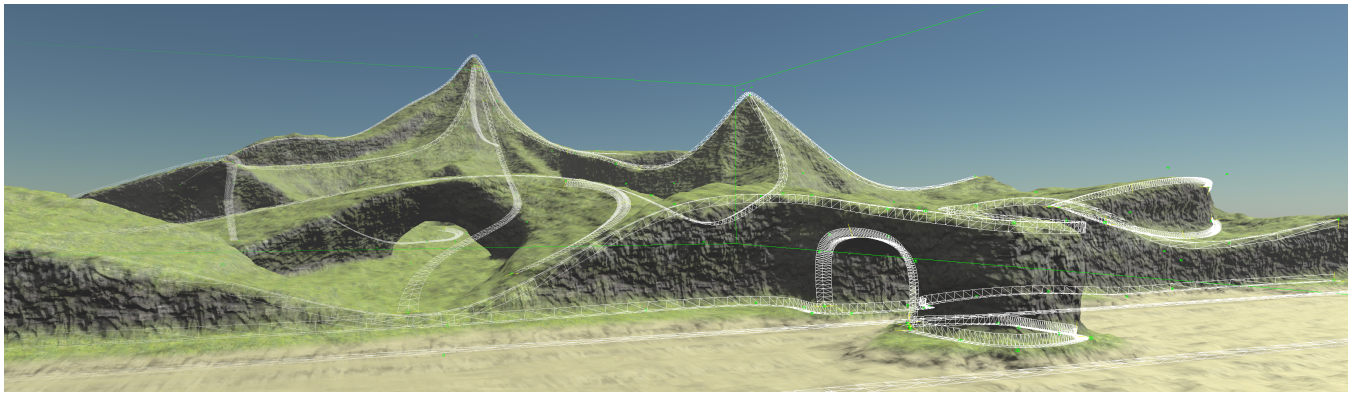

Figure 1: *A coastline modelled with our method. The Feature Curves that we placed to model the scene are blended over the image taken from our modelling application. Note the overhanging rock faces and the arch in the front, which are not possible using a traditional two-dimensional heightfield.*

Abstract

Two-dimensional heightfields are the most common data structure used for storing and rendering of terrain in offline rendering and especially real-time computer graphics. By its very nature, a heightfield cannot store terrain structures with multiple vertical layers such as overhanging cliffs, caves, or arches. This restriction does not apply to volumetric data structures. However, the workflow of manual modelling and editing of volumetric terrain usually is tedious and very time-consuming. Therefore, we propose to use three-dimensional curve-based primitives to efficiently model prominent, large-scale terrain features. We present a technique for volumetric generation of a complete terrain surface from the sparse input data by means of diffusion-based algorithms. By combining an efficient, feature-based toolset with a volumetric terrain representation, the modelling workflow is accelerated and simplified while retaining the full artistic freedom of volumetric terrains. All stages of our method are GPU-accelerated using compute shaders to ensure interactive editing of terrain.

Keywords: terrain, interactive modelling, volumetric, spline-curves, diffusion algorithms, GPU

Concepts: •**Computing methodologies** → **Parametric curve and surface models;** *Mesh models; Volumetric models;*

*michael.becher@visus.uni-stuttgart.de
†michael.krone@visus.uni-stuttgart.de
‡guido.reina@visus.uni-stuttgart.de
§thomas.ertl@vis.uni-stuttgart.de

I3D '17, February 25-27, 2017, San Francisco, CA, USA
ISBN: 978-1-4503-4886-7/17/03…$15.00
DOI: http://dx.doi.org/10.1145/3023368.3023383

1 Introduction

The study and depiction of landscapes has a long-standing history in traditional arts such as painting[1] and photography as well as in literature. It is no surprise that virtual landscapes have been present in the field of computer graphics since its early days. As early as 1980, animated short *Von Libre* by Loren Carpenter explored a mountain scene depicting fractal terrain [Carpenter 1980]. In the real-time computer graphics of the 1970s and early 1980s, virtual landscapes were still mostly limited to basic silhouettes drawn by vector graphics or rather coarse 2D raster graphics. But with the rapid increase of processing power, more advanced computer graphics algorithms, and the advent of hardware-accelerated 3D computer graphics, the quality of virtual terrain improved likewise. Over the decades, many approaches to rendering were explored, including bitmap sprites and voxel graphics. Eventually, most real time applications settled on using 3D polygonal meshes to render detailed terrain surfaces. Behind the scenes, 2D heightfields (or heightmaps) were established as a common representation for terrain. To this date, heightmaps remain the standard solution for rendering terrain in some of the most popular, commercial game engines including Unreal Engine 4 [Epic Games, Inc. 2016], CryEngine V [Crytek 2016] and Unity [Unity Technologies 2016]. However, overhanging terrain structures or caves simply cannot be represented by a heightmap without further extensions. Even terrain without overhangs, but with steep slopes, cliffs or any arbitrary terrain structures that stretch out vertically, can cause issues due to stretched-out textures or quads and triangles with high aspect ratio. Such restrictions do not apply to terrain representations in a three-dimensional domain. It is possible to represent and store any terrain structure, including in particular overhanging structures, caves or generally terrain with several vertical layers, in a volumetric data structure. Corresponding research, frameworks or extensions to existing graphics engines are indeed available. However, manual creation and editing of such volumetric terrain representations often becomes a tedious task if every cubic meter has to be manually *painted* with a brush (or rather blob) tool. In many cases it is pre-

[1] Evidence suggests landscape paintings date back to as early as Ancient Greece [Hugh Honour, John Fleming]

ferred to generate volumetric terrain data by procedural algorithms. This, on the other hand, limits the control the user can exercise over specific regions of the terrain.

We present an efficient and intuitive method for controlled generation and fast editing of volumetric terrain. We propose to apply concepts and modelling tools that have previously been used to create terrain heightmaps to volumetric representation. The immediate intention is to let users model prominent, large-scale terrain features that define the overall appearance of the landscape. Inspired by the use of curve-based tools for heightmap creation, Feature Curves, i.e. parametric 3D spline curves, are used to model the terrain and place constraints in the scene (see Figure 1). Editing of large-scale terrain features is thereby reduced to adjusting a small amount of curve control vertices and constraint properties. We developed a multi-stage GPU-accelerated computational pipeline that generates a volumetric terrain representation based on the Feature Curves. The pipeline works on a specified domain containing a set of discrete vector and scalar fields. Voxelization is used to transfer the (sparse) information defined by the Feature Curves into these fields. Using a diffusion process, a dense normal vector field is obtained and then used to approximate a signed distance field for a terrain surface that matches the constraints given by the Feature Curves.The terrain surface is then extracted and, based on user-controlled parameters, medium to high resolution details are added by simple, procedural means to create a visually pleasing result.

Our main contribution is a novel method for generating a terrain surface from a limited set of prominent terrain features with the following notable properties:

- Fast and intuitive modelling of terrain features by means of parametric curves;
- No restrictions to the vertical layout of the terrain in general (allowing overhanging terrain and caves) by using a volumetric computational domain;
- Interactive, real-time editing of terrain

The rest of the paper is structured as follows. After a brief overview of related work in the field of terrain modelling in Section 2, we introduce our modelling primitives in Section 3. In Section 4, we present our multi-stage computational pipeline to generate arbitrary terrain surfaces and discuss each stage in detail. Extensions of our method and an application prototype are presented in Section 5 and 6, respectively. In Section 7 we discuss the results of our method with regard to performance, capabilities, and quality. Section 8 concludes the paper and discusses possible future extensions.

2 Related Work

There are many efficient methods for modelling and editing terrain that range from fully procedural methods to methods combining sparse or dense user input with an automated terrain generation process. An overview of different methods used to model and represent terrain is given in the recent work of Natali *et al.* [Natali et al. 2013]. Different techniques are classified into data-oriented scenarios and workflow-oriented ones and a comparison of their abilities is made. With regard to the classification of Natali *et al.*, the method we present is a workflow-oriented, data-free technique.

Procedural methods are a popular approach to generating terrain surfaces as they generally require little input to create realistic landscapes. A comprehensive overview of general procedural approaches in computer graphics was given by Ebert *et al.* [Ebert et al. 2002]. A survey focusing on methods used to procedurally generate terrain was given by Smelik *et al.* [Smelik et al. 2009]. Some procedural approaches that are specifically aimed at terrain generation model real-world effects, such as hydraulic erosion, that have a large impact on the appearance of landscapes [Šťava et al. 2008]. Génevaux *et al.* presented a method based on hydrology that merges procedurally generated terrain with a user-defined river network [Génevaux et al. 2013].

Methods combining sparse user input with an automatic, and often procedural, terrain generation are of particular interest for our work. Sketch-based methods allow the user to sketch complex silhouettes and then generate a terrain surface that matches the input sketches [Gain et al. 2009][Tasse et al. 2014]. While conceptually similar to the idea of our methods, the sketches are usually made in a first person view whereas our method works in world space and furthermore also defines the slope around a terrain feature. Hnaidi *et al.* presented a framework for modelling terrain based on parametric spline curves [Hnaidi et al. 2010]. The *Feature Curve* concept they introduced for modelling terrain features has been a notable inspiration for this paper. However, in contrast to our method, the technique they presented is used to create heightmaps. Emilien *et al.* focus on the modelling of waterfalls and combine procedural methods with user-controlled tools [Emilien et al. 2014]. Although also using heightmaps, they achieve overhanging terrain by vertical vertex displacement. A framework with volumetric terrain generation in mind has been presented by Peytavie *et al.* [Peytavie et al. 2009]. They combine material layers stored in a volumetric datastructure with an implicit surface representation and offer high level tools for sculpting the terrain surface.

A more generic approach to modelling arbitrary objects using parametric curves was presented by Singh and Fiume [Singh and Fiume 1998]. Like our method, they propose to manipulate curves to implicitly modify the surface of objects rather than to directly manipulate the surface representation itself. However Singh and Fiume use curves to deform existing geometry of objects, whereas our method generates completely new geometry that satisfies the constraints given by our curve-based modelling primitives. Adding and manipulating additional curves will not deform the previously generated geometry but will simply cause a complete regeneration of the geometry with a potentially different topology.

3 Modelling Primitives

Rather than directly shaping and modifying parts of the terrain surface (e.g. by using brush-based tools), we propose to place primitives in the scene that describe and constrain the terrain surface in their vicinity and from which the terrain surface is propagated into the whole domain. While this is a relatively straightforward concept, we introduce some novelty by applying it to an otherwise unrestricted three-dimensional computational domain.

Our primary modelling primitive are Feature Curves, a concept previously used in the context of terrain modelling by Hnaidi *et al.* [Hnaidi et al. 2010]. The term is derived from the idea that curve-based modelling primitives are used to describe prominent terrain features such as ridgelines, river beds, or cliffs. Due to the shape of many terrain features, like the aforementioned ones, parametric curves are a good fit to design a generic modelling tool for describing such features. Our concept of a Feature Curve shares the same basic ideas, the exact definition and properties however differ in several aspects from those of Hnaidi *et al.*. Most prominently, we define our Feature Curves in three-dimensional space and are thus able to describe almost arbitrary terrain surface structures including, in particular, structures that are impossible to describe with a heightfield as they are compromised of several vertical layers.

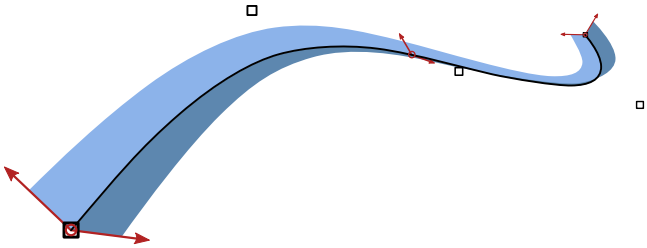

Figure 2: *A visual representation of a Feature Curve with five control vertices (black boxes) and three Constraint Points (red circles). At each Constraint Point, the bitangent vectors are displayed. Two ribbons matching the bitangent constraints are traced along the curve.*

3.1 Feature Curves

Our Feature Curves are constructed by augmenting regular, three-dimensional, cubic B-Splines with additional attributes. These attributes are gathered in so-called *Constraint Points* that are placed on and associated with – we also say attached to – a specific Feature Curve. A Constraint Point p attached to a Feature Curve $\mathcal{F}$ is defined as a tuple $\left(b_l, b_r, \alpha, \beta, t\right)$ where $b_l, b_r \in \mathbb{R}^3$ are the left- and right-hand bitangent vector, both α and β are scalar values used for noise generation and the scalar value t denotes the position of the Constraint Point on $\mathcal{F}$ in the curve's parameter space. The vectors b_l and b_r are perpendicular to the curve's tangent vector at the curve point given by t and control the slope of the terrain surface. Formally, a Feature Curve is then defined as

$$\mathcal{F}(t) = \left(\mathbf{B}(t), \mathcal{P}(t), s\right)$$

where $\mathbf{B}$ is a cubic B-spline, $\mathcal{P}$ is a parametric function that interpolates between constraint points in a given set P, and s is a boolean value that determines whether $\mathcal{F}$ acts as a seed primitive for the terrain surface or simply as a guidance primitive. Thus, a point on a Feature Curve is a tuple of a position in 3D space (given by the evaluation of a B-spline), a tuple of interpolated constraint attributes and the seed property of the Feature Curve.

A Feature Curve requires a minimum of two Constraint Points attached to the curve's endpoints, with an arbitrary number of additional Constraint Points added in between. Except for the bitangent vectors, the attributes given by the Constraint Points are linearly interpolated along the curve. Due to the curve's arbitrary shape and orientation in 3D space, the interpolation of bitangent vectors is non-trivial. A consistent interpolation is possible by first computing a rotational transformation matrix from known curve tangent vectors and then applying this matrix to the bitangent vectors.

A pair of ribbons joined at one edge as seen in Figure 2 serves as the visual representation of a Feature Curve. Each ribbon can be thought of as the result of tracing the left- or right-hand bitangent vector along the curve by interpolating them between Constraint Points.

3.2 Proxy Geometry

So far, Feature Curves are defined as continuous parametric curves. However, with respect to the terrain generation process presented in the upcoming section, a representation that can be evaluated fast and conveniently in object space is desired. Furthermore we aim to find a representation that could be shared by different kinds of additional modelling primitives. Since modelling primitives are entities placed and arranged in the scene, proxy geometry is useful to give visual feedback of their placement and shape, especially the shape of the ribbons as depicted in Figure 2. Therefore, we decided to employ a triangle mesh as input for the terrain generation pipeline because it is most convenient to handle as shared representation both for rendering and for evaluation in the pipeline. Recall the visual representation of a Feature Curve as a pair of connected ribbons. To create a triangle proxy mesh for Feature Curves as depicted in Figure 2, Feature Curves are first discretized into piecewise linear segments. For each line segment, the left- and right-hand bitangent vector is interpolated at the endpoints and used to create two possibly distorted rectangles, that extend away from the line segment. Curve position and shape is implicitly stored in the vertex positions of the proxy mesh. The remaining constraint properties are stored as additional vertex attributes alongside a per-vertex normal vector which is derived from the curve tangent at the segment's endpoints and the interpolated bitangent vectors.

4 Volumetric Terrain Generation Pipeline

The workflow we propose lets the user arrange a set of high-level modelling primitives in the scene from which a terrain surface mesh is automatically generated. This is handled by a multi-stage GPU-accelerated computational pipeline that is described in detail in this section (see also Figure 3).

4.1 Overview

As one of the key goals of our method is to create terrain surfaces with several vertical layers, the terrain generation pipeline operates on a volumetric computational domain. All modelling primitives within this domain are processed by the pipeline as input. These modelling primitives implicitly define a surface, which in the following we refer to as the target terrain surface. In order to retrieve an explicit representation of this target surface, we make the basic assumption that it is predominantly smooth, that is the second derivative of its normal vectors equals zero. Furthermore, recall that by definition of the modelling primitives, the target terrain surface passes through the location of surface modelling primitives and in the vicinity of both surface modelling primitives and guidance ones, the surface complies to the normal vectors given by the nearest modelling primitives. From these properties it ensues that one can compute a dense normal vector field using the normal vectors given by the modelling primitives as initial values for a diffusion process. Since this vector field conforms to the target surface properties (i.e. at the location of modelling primitives it is equal to the given surface normals and smooth everywhere else), it is also an implicit description and most notably a dense one for any potential terrain target surfaces. By combining it with information on points in the domain that are known to be located on the target terrain surface, an explicit representation of this target surface is computed.

The terrain surface generation is restricted to a limited domain that is defined by a discrete, three-dimensional, uniform, regular grid. All stages of the pipeline operate on a set of discrete vector- and scalar fields aligned to that grid. To store the information computed during the pipeline execution, the following fields are defined within the computational domain:

- $\mathcal{N}$: A vector field containing surface normal vectors
- $\mathcal{B}$: A vector field containing surface bitangent vectors
- $\mathcal{S}$: A scalar field storing terrain surface information
- $\mathcal{R}$: A vector field containing 2D vectors that store amplitude and roughness parameters for a noise generator

For the core terrain generation method, the normal vector field $\mathcal{N}$ is sufficient, while the bitangent vector field $\mathcal{B}$ is only required by

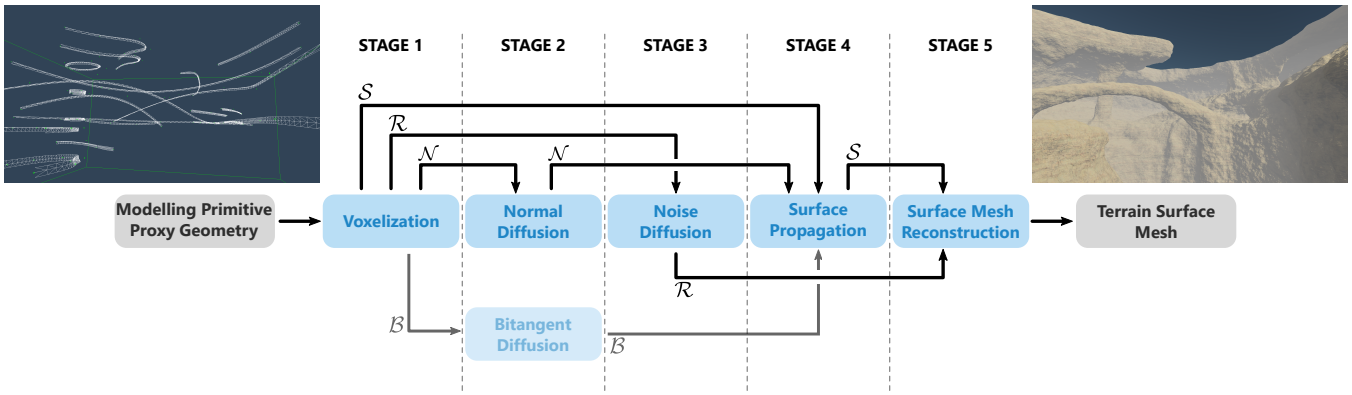

Figure 3: *The terrain generation pipeline as implemented in our method. Gray boxes depict in- and output of the pipeline. Blue boxes depict stages as described in Section 4.1. Arrows indicate data flow, i.e. fields, between stages.*

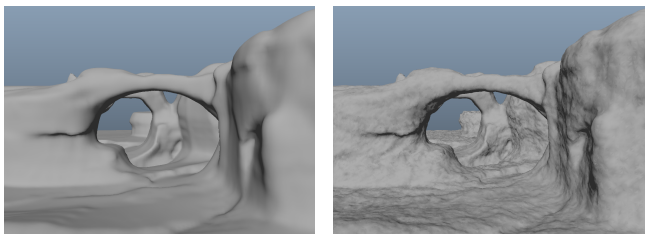

Figure 4: *A comparison between the terrain geometry before applying surface displacement based on the noise parameters stored in $\mathcal{R}$ (left) and afterwards (right).*

an optional stage and an alternative algorithm. We also refer to $\mathcal{N}$ and $\mathcal{B}$ as guidance fields. The scalar field $\mathcal{S}$ stores a distance field that serves as an intermediate terrain representation. The parameters stored in $\mathcal{R}$ are used to generate a noise field that we use to add additional high-frequency detail to the generated terrain surface (see Figure 4). Since storing the high-frequency noise signal would require are very high resolution, only noise generation parameters are stored in $\mathcal{R}$. The complete process of generating a terrain mesh from the sparse user input is split into the following five (plus one optional) stages, also shown in Figure 3.

1. **Voxelization** – Input modelling primitives are converted to the computational domain.
2. **Guidance field diffusion** – A dense normal vector field (and optionally a bitangent one) is computed using diffusion.

2a. **Feature Curve expansion** – *Optional.* The user-defined Feature Curves are automatically expanded along the guidance fields.

3. **Noise field diffusion** – A dense field of noise parameters is computed using a diffusion algorithm.
4. **Surface propagation** – Surface field values are propagated along the guidance field.
5. **Surface mesh reconstruction** – An iso-surface is extracted from the final surface field as a triangle mesh.

4.2 Voxelization

In the initial stage of the terrain generation pipeline, the input modelling primitives are converted from their vector-based definition in continuous 3D space to the discrete computational domain. Simply speaking, we use the triangle mesh proxy geometry generated from the parametric curves instead of the parametric representation itself. That way, the input of the pipeline is generalised to triangle meshes, which allows to not only to process the proxy meshes of our modelling primitives but also generic meshes, as for example a pre-modelled asset containing a section of terrain or a mesh created from a heightmap.

The proxy meshes of all relevant modelling primitives have to be discretized into a three-dimensional, regular, uniform grid, a process that is often referred to as voxelization. We chose a generic approach and simply determine all intersected grid cells for each triangle using a geometric intersection test. Since a grid cell can be intersected by multiple triangles with different normal and bitangent vectors as well as different noise generation parameters, the values of all intersecting triangles are first simply gathered per grid cell. Once all triangles are processed, the gathered values are averaged per voxel and written to the $\mathcal{NB}$ and $\mathcal{R}$ respectively. Note that if a cell is intersected by normals from both ribbons of a Feature Curve, normals derived from both bitangent vectors are averaged. The normal field is therefore smoothed along the spine of a Feature Curve. Special attention is required to average bitangent vectors. All left-hand and right-hand bitangent vectors in the cell are first averaged separately, resulting in two bitangent vectors $\hat{b}_l$ and $\hat{b}_r$. Then the vector b_{avg} that maximizes $|\hat{b}_l \cdot b_{avg}| + |\hat{b}_r \cdot b_{avg}|$ is computed. To that end, the averages are either added up if the angle between the two averaged vectors is smaller than $90°$, or subtracted otherwise.

For the upcoming sections, let $\Omega_N \subset \mathcal{N}$ and $\Omega_B \subset \mathcal{B}$ be the subset of voxels that have normal and bitangent vector information stored during the voxelization, $\Omega_R \subset \mathcal{R}$ the subset of voxels that have noise parameters stored and $\Omega_S \subset \mathcal{S}$ be the subset of voxels corresponding to cells that are intersected by at least a single triangle of a surface modelling primitive.

4.3 Normal field diffusion

After the voxelization stage, the normal vector field $\mathcal{N}$ is only sparsely populated. In order to use it for computing a closed terrain surface, it has to be densely populated. Recall that we require the target terrain surface to comply to the constraints given at Feature Curve locations and assume it to be smooth everywhere else. These properties should be reflected by the normal vector field. The problem of computing a smooth and densely populated vector field from sparse input shares similarities to gradient vector flow, a method for computing a dense vector field based on image gradient by minimizing an energy functional [Xu and Prince 1997]. However, when

applied to our problem, the method can be simplified to applying the smoothness term of the energy functional to $\mathcal{N} \setminus \Omega_N$. It follows that one can obtain a suitable dense normal vector field by solving a diffusion equation on the sparsely populated field $\mathcal{N}$ using the voxels in Ω_N as initial boundary conditions. To retain edges on the surface to at least some extent, we opt for using anisotropic diffusion. To that end, the following discretized anisotropic diffusion equation, as presented by Perona and Malik [Perona and Malik 1990], is applied to $\mathcal{N} \setminus \Omega_N$ in an iterative computation:

$$\begin{aligned}\mathcal{N}^{t+1}_{i,j,k} = \mathcal{N}^{t}_{i,j,k} + \frac{1}{6}\Bigg(& c_E \cdot \left[\frac{\partial \mathcal{N}^t}{\partial x}\right]^+_{i,j,k} + c_W \cdot \left[\frac{\partial \mathcal{N}^t}{\partial x}\right]^-_{i,j,k} \\ &+ c_U \cdot \left[\frac{\partial \mathcal{N}^t}{\partial y}\right]^+_{i,j,k} + c_D \cdot \left[\frac{\partial \mathcal{N}^t}{\partial y}\right]^-_{i,j,k} \\ &+ c_N \cdot \left[\frac{\partial \mathcal{N}^t}{\partial z}\right]^+_{i,j,k} + c_S \cdot \left[\frac{\partial \mathcal{N}^t}{\partial z}\right]^-_{i,j,k}\Bigg),\end{aligned}$$

where $\left[\frac{\partial}{\partial x}\right]^+_{i,j,k}$ denotes the finite difference operator using a forward difference in x-direction at the location (i, j, k), while $\left[\frac{\partial}{\partial x}\right]^-_{i,j,k}$ denotes the backward difference operator. To evaluate the finite differences at the boundaries of the domain, homogeneous Neumann boundary conditions are used for $\mathcal{N}$. We define the coefficients c_E, c_W, c_U, c_D, c_N and c_S as a function g of the negative dot product of adjacent voxels in $\mathcal{N}$mapped to $[0, 1]$, e.g.:

$$\begin{aligned} c_E &= g\left(\frac{1 - \left((\mathcal{N}^t_{i,j,k}) \cdot \mathcal{N}^t_{i+1,j,k}\right)}{2}\right), \\ c_W &= g\left(\frac{1 - \left((\mathcal{N}^t_{i,j,k}) \cdot \mathcal{N}^t_{i-1,j,k}\right)}{2}\right), \\ c_U &= \ldots \end{aligned}$$

The coefficients c_U, c_D, c_N and c_S are defined analogously for the two remaining coordinate axes. We choose function g as a subquadratic function known from literature [Perona and Malik 1990]:

$$g(x) = \frac{1}{1 + (\frac{x}{k})^2}$$

The choice of parameter k depends on the desired amount of smoothing across edges and is empirically set to a fixed value of 0.01. While we found this to produce good results, this parameter could be added to the parameter set of Constraint Points to give additional control to the user. Note that for all voxels that have not yet been set to a meaningful value, i.e. $\mathcal{N}_{i,j,k} = \vec{0}$, the function g will evaluate to 1 due to the dot product being 0. In this case, the anisotropic diffusion equation simplifies to Laplace's equation [Evans 1998]. We deliberately use this property in order to propagate normal vectors to regions of the field where no meaningful coefficient for the anisotropic diffusion can be computed yet.

4.4 Noise field diffusion

Like the guidance vector fields, the noise field $\mathcal{R}$ is only sparsely populated after the voxelization and the initial values have to be propagated into the whole domain. As we assume the field to be smooth as well, we again use a diffusion process. However, we do not require the diffusion process in $\mathcal{R}$ to be edge-preserving and therefore simply solve Laplace's equation to obtain a dense field.

4.5 Surface propagation

Once $\mathcal{N}$ is computed, we can proceed with determining the surface field $\mathcal{S}$. $\mathcal{S}$ shall result from a signed distance function with regard to the target terrain surface. Obviously, at this point we do not have a surface as input to derive $\mathcal{S}$ but rather we want to compute $\mathcal{S}$ by different means in order to reconstruct the target surface from it. The basic idea is to propagate the surface information within $\mathcal{S}$ starting from the values stored in Ω_S during the voxelization stage. To compute $\mathcal{S}$ based only on the initial values in Ω_S and the guidance field $\mathcal{N}$, we define a function that approximates a voxel's distance value based on its local neighbourhood.

Let $\mathcal{NB}_{i,j,k}$ be the set of coordinates adjacent to the location (i, j, k) in the domain, i.e. $(i\pm1, j, k)$, $(i, j\pm1, k)$ and $(i, j, k\pm1)$. Given a cell c_x with $x \in \mathcal{NB}_{i,j,k}$, assume that a target surface intersecting c_x is locally planar and its orientation is given by $\mathcal{N}_x$. Then we can model this surface as a plane defined by the midpoint of c_x and the normal vector $\mathcal{N}_x$. The signed distance d between the midpoint of cell $c_{i,j,k}$ and this plane is obtained by a simple point-plane distance calculation. In case the target terrain surface actually intersects c_x, i.e. $\mathcal{S}_x = 0$, the obtained distance value d is assigned to $\mathcal{S}_{i,j,k}$ unless a smaller distance is computed for one of the other neighbour cells. If, on the other hand, c_x is not intersected by the target surface but the distance to the nearest surface point is already known and stored in $\mathcal{S}_x$ we simply assign the sum $\mathcal{S}_x + d$ to $\mathcal{S}_{i,j,k}$. If c_x is neither intersected by the surface nor a valid distance is known yet, no value based on c_x is assigned to $\mathcal{S}_{i,j,k}$ at this point. Due to the construction of $\mathcal{N}$, this is a legitimate approximation to the distance value at location (i, j, k). The surface normal points in the direction of the nearest surface and is therefore a reasonable indicator for determining whether a voxel is closer or further away from a surface than the reference voxel. Recall that $\Omega_S \subset \mathcal{S}$ is the subset of $\mathcal{S}$ that is filled during the voxelization. Let $\partial\Omega_S \subset \mathcal{S} \setminus \Omega_S$ be the boundary between Ω_S and $\mathcal{S} \setminus \Omega_S$. Every value $\mathcal{S}_{i,j,k} \in \partial\Omega$ is set according to the following function and inserted into Ω_S.

$$\mathcal{S}_{i,j,k} = \min_{x \in \mathcal{NB}_{i,j,k}} \left(\mathcal{S}_x + ((i, j, k) - x) \cdot \frac{\mathcal{N}_{i,j,k} + \mathcal{N}_x}{2}\right)$$

Once $\Omega_S = \mathcal{S}$, a signed distance field for our target surface has been successfully estimated. Note that the actual calculation uses an interpolated normal located between (i, j, k) and x in order to reduce the error introduced by assuming the surface to be locally planar. Each value $\mathcal{S}_{i,j,k}$ is set only once. An iterative computation that tries to improve the distance approximation over time proved to be unstable, as modelling the surface contour as locally planar introduces an error that can lead to two neighbouring voxels continually de- or increasing each other's value. Further research into a more elaborate model of the local surface contour might improve this behaviour and the resulting distance approximation.

Because of discontinuities in $\mathcal{N}$as well as the discrete nature of $\mathcal{N}$, the surface propagation tends to suffer from noise as it assumes the target surface to be locally planar. By applying a Gaussian filter to $\mathcal{S}$, the noise is suppressed to a certain degree before $\mathcal{S}$ is used as input for the final pipeline stage.

As a side note, we also explored an alternative approach to propagating the surface information in $\mathcal{S}$. Here, a curve is traced through the guidance vector field similar to a streamline. If this streamline originates from a point known to be on the target surface, we assume that every voxel intersecting the streamline also intersects the target surface. Ultimately this approach proved to be less reliable, partly because it uses the biased bitangent vector field $\mathcal{B}$.

4.6 Surface mesh reconstruction

In the final stage of the pipeline an explicit surface representation for efficient rendering is reconstructed from the implicit representation given by $\mathcal{S}$. We deliberately use polygonal meshes as this are efficient to render on GPUs and supported by most rendering

pipelines. The reconstruction of a surface mesh from $\mathcal{S}$ is equivalent to computing an iso-surface of the scalar field. As our method computes $\mathcal{S}$ as a signed distance field, the iso-value is set to zero. We use the well-known Marching Cubes algorithm [Lorensen and Cline 1987] to extract the iso-surfaces as a triangle mesh. Note that in-between two Feature Curves that have a similar normal direction and are located above one another (*w.r.t.* their normal vectors), a sign change in $\mathcal{S}$ occurs and an iso-surface is extracted. This behaviour supports the generation of a plausible surface in areas not explicitly defined by modelling primitives. For example, if the roof of a cave is not explicitly modelled, it will be implicitly created where the positive distance values propagated from the cave floor meet the negative distance values propagated from the terrain surface above the cave.

During this stage, the noise parameters stored in $\mathcal{R}$ are also embedded into the vertex data of the extracted mesh, so that we can compute a corresponding high-frequency noise signal during rendering. We apply this noise as vertex displacement to obtain additional small-scale details on the surface mesh. Furthermore, surface normal vectors that are required for lighting and procedural texturing of the extracted mesh are also computed and embedded in the vertex data. To obtain normal vectors that consistently match the extracted surface mesh, we compute the normals as the gradient of $\mathcal{S}$ instead of using the values stored in $\mathcal{N}$.

5 Optional Extensions of our Method

To improve usability and scalability of our approach, we also implemented two extensions that are described in the following.

5.1 Feature Curve Expansion

As an additional, optional stage in the pipeline it is possible to automatically expand the Feature Curves along the guidance fields $\mathcal{N}$ and $\mathcal{B}$. This decreases the open space not covered by any user-defined Feature Curves and potentially reduces the ambiguity of the terrain surface in some areas. A Feature Curve is expanded by making small steps in the direction of the cross product of normal and bitangent vector. After some distance is covered, a new control vertex is added to the curve and the process is stopped once we either step out of the computational domain or get too close to another Feature Curve. After all Feature Curves have been expanded, the guidance field $\mathcal{N}$ is updated to match the expanded Feature Curves.

To achieve a stable expansion, a densely populated, smooth bitangent field is required. We consider the bitangent vector field to be smooth if the dot product of adjacent vectors is either close to one or close to minus one, meaning that the sign is ignored. A diffusion process as used for $\mathcal{N}$, which basically computes the mean vale of a local neighbourhood, is not applicable. It is possible to formulate the computation of the most suitable bitangent vector for any given voxel based on a local neighbourhood as an optimization problem. However, to avoid a costly computation and a possibly more complicated implementation, a more straightforward function to compute a suitable bitangent vector is desirable. We settled on approximating a possible solution for a given voxel by several successive, pair-wise averaging operations using its local neighbourhood. As our averaging operation is neither commutative nor associative, changing the order in which vectors are averaged potentially changes the results. Thus, $\mathcal{B}$ is biased in some direction.

5.2 Large Terrain using Volume Bricking

So far the computational domain is limited to a single grid of rectangular shape, treating all areas within the domain uniformly. While this allows for clear definitions and facilitates the understanding of the terrain generation pipeline's core structure, in practice it leads to a potential waste of processing power. Regions where the terrain surface is not present at all or that are far away from the virtual camera do not need to be computed at full detail, that is at full grid resolution. Especially by taking advantage of adaptive resolution for distant regions, overall larger terrain becomes feasible.

To increase the flexibility of our approach, the domain can be subdivided into sub-regions called bricks. Just like the whole domain before, the individual bricks are rectangular uniform grids and each brick comes with its own set of fields $\mathcal{N}$, $\mathcal{B}$, $\mathcal{R}$, and $\mathcal{S}$. However, the resolution of the individual bricks can vary and the resolution of the domain as a whole is thus no longer uniform. As the computations of the terrain generation pipeline are now distributed over multiple bricks, global solutions (*w.r.t.* the whole domain) for the guidance field diffusion, the noise parameter diffusion, and the surface propagation need to be computed. To achieve this, the bricks have to exchange values with adjacent bricks at the shared boundaries, meaning that the boundary conditions of computations have to be modified to access values from adjacent bricks. Furthermore, the exchanged values have to be updated after each iteration, requiring synchronisation between bricks. Another issue is the reconstruction of the actual surface mesh. If the mesh reconstruction is simply performed for all bricks individually, a separate terrain surface mesh is constructed for each brick. Because the Marching Cubes algorithm operates only in between the voxels of $\mathcal{S}$, a small, empty gap remains at the boundary of each brick and consequently in between adjacent bricks. The resulting overall landscape gets visually subdivided into per-brick patches. To close these gaps, additional geometry has to be inserted in between bricks where applicable. Note that the resolution of adjacent bricks can differ and simply performing additional Marching Cubes operations to create the additional geometry will result in visible seams in the mesh. Methods for generating seamless geometry from multiresolution voxel data are known and can be used to remedy this [Lengyel 2010].

6 Implementation of Application Prototype

As a proof of concept of the presented terrain generation method, we implemented a prototype application. It is written in C++ and heavily relies on OpenGL (4.3+) for general purpose computations on the GPU (GPGPU) as well as for the graphical presentation. All stages of the terrain generation pipeline use GPU acceleration to achieve interactive frame rates during editing of the terrain.

To correctly average over all intersecting triangles, the voxelization stage is split into a gathering and averaging step. First, the triangle intersection tests are performed in parallel and the results are gathered in a per-voxel linked list using the Shader Storage Buffer and Atomic Counter features [Yang et al. 2010]. In the averaging steps, the information stored per voxel is then averaged correctly. In each iteration of the iterative algorithms used in the normal field diffusion and surface propagation stage, the result for all voxels is computed in parallel, while the iterations itself are performed sequentially to allow synchronisation between iterations. The number of iterations depends on the grid resolution and should guarantee that information can propagate across the whole domain. The surface reconstruction uses a GPU implementation of Marching Cubes that uses Histogram Pyramids [Dyken et al. 2008]. The vertex displacement based on the noise generated from the parameters stored in the vertex data is performed during the rendering of the terrain surface using hardware tessellation. We can export the final surface mesh including vertex displacement using Transform Feedback.

Texturing of the mesh is performed using tri-planar projection based on the surface normals computed during the surface mesh ex-

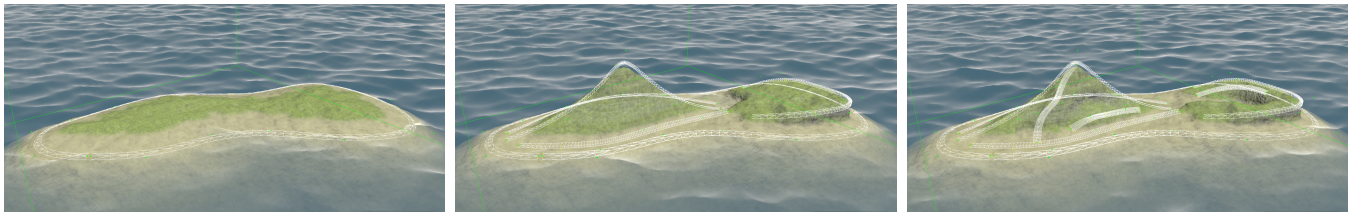

Figure 5: *Three stages of modelling a simple oceanic island. From left to right: Start by modelling the island's shoreline. Then add Feature Curves to define the island's topology, in this case a small mountain and cliff. Finally, add some details to the major terrain features by placing additional Feature Curves.*

traction stage [Nguyen 2007]. Because the terrain can feature multiple vertical levels, the tri-planar projection has to be extended to use different textures for the projection along the positive and negative up axis. Otherwise, ground textures like grass, snow or sand will appear on the underside of overhangs and the roof of caves, which is usually undesired. The world up vector has to be given by the rendering application or scene context, as the volumetric terrain itself has no reasonable concept of an up direction.

The application offers a basic user interface to freely create terrain surfaces and edit all properties included in our approach. The workflow of our method is centered around the arrangement of Feature Curves and offers the following functionality:

- Add and insert control vertices and Constraint Points anywhere on a curve
- Move control vertices and manipulate bitangents directly in the 3D viewport
- Switch between real-time and on-demand updates of the generated terrain
- Import and export Feature Curves as well as export the final terrain surface mesh

7 Results and Discussion

In this section, we analyze our method with regard to the capabilities and the quality of the resulting terrain surface. We will also show examples of more complex landscapes that demonstrate the benefits of our method with regard to terrain features that cannot be represented by heightmaps. Furthermore, we evaluate the performance of our terrain generation pipeline.

7.1 Capabilities

Figure 5 shows how a simple oceanic island can be modelled in three steps. The images were captured directly from our prototype application and a grid resolution of $128 \times 64 \times 128$ was used for the scene. We started with a single Feature Curve that defines the shoreline of the island. We then modelled the major ridge line of a small mountain with a single Feature Curve and used two more curves to define the base of the mountain. Using another Feature Curve, we added a cliff to the other side of the island. In this case we did not need a second curve to model the bottom of the cliff (as seen in Figure 1), because the curve is close enough to the shoreline and our method automatically bridges the gap. Finally, we added some more details to the mountain and cliff with a few additional short Feature Curves.

Apart from the slightly overhanging cliff, the oceanic island makes little use of the key advantage of our method. To better demonstrate the capabilities of our method, we recreated the famous Double Arch located in Arches National Park, USA. The result is shown on the left in Figure 7. We used our application to model the scene

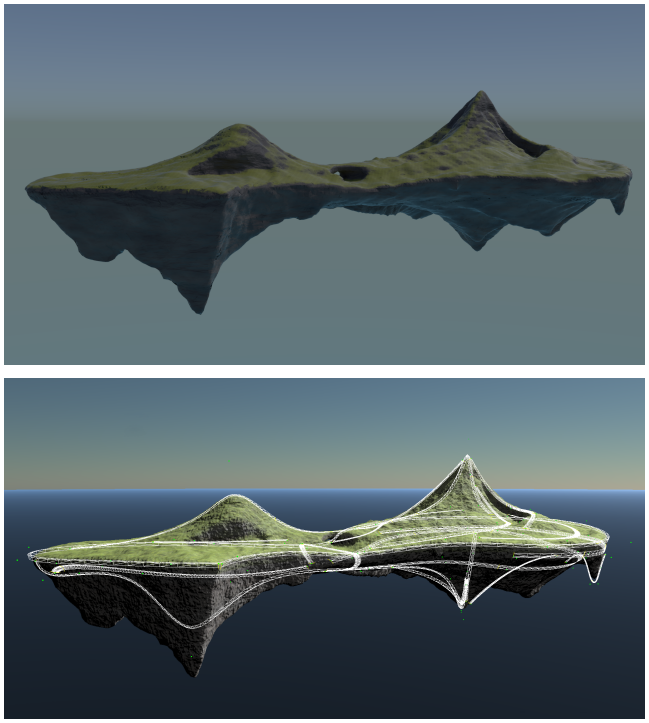

Figure 6: *Top: A fictional landscape consisting of floating islands. Our methods allows to consistently model both the topside and bottom of the islands using a single multi-purpose toolset. Bottom: A screenshot of the floating island in our prototype application showing the Feature Curves used to create the terrain surface.*

and then imported the generated terrain surface mesh into Blender 2.76b [Blender Foundation 2016] for shading and rendering. Only 18 Feature Curves were necessary to model the scene. For the terrain generation, a grid resolution of $256 \times 64 \times 256$ was used. While our method has some difficulties in recreating small surface details and especially jagged rocks and cracks, it easily allows to model the primary arches and vertical rock faces. Overall, we observed that our methods tends to create smooth surfaces, due to the diffusion process used to create the normal vector field.

We also demonstrate the possibilities our method offers for the creation of fictional landscapes. The upper image in Figure 6 shows an island floating in the sky. This scene was also first modelled in our prototype application and then exported to Blender for rendering. We again used a grid resolution of $256 \times 64 \times 256$ and arranged 16 Feature Curves to cover all possible angles of the island. We also deliberately left some open space to have it automatically filled in by our method. This often results in organic and interesting shapes,

Figure 7: *Left: A recreation of the Double Arch rock formation in Arches National Park, USA. The geometry was modelled in our prototype application and then imported into Blender for shading and rendering. Right: The photograph used as a template for the modelling. Photograph 'Rock Arch 1' courtesy of Fred Moore.*

as can be seen in Figure 6. On the bottom of the left half of the floating island, the geometry matching the Feature Curve is hidden behind geometry that was generated due the region being relatively unconstrained.

The three examples showcase that our method can be used create complex, high-quality terrains with only few Feature Curves. During content creation, Feature Curves can be adapted intuitively to obtain a desired shape of the landscape or new Feature Curves can be added to introduce additional details. As the Double Arches example shows, our method can also be used to easily recreate existing landscapes (e.g. using a photograph or a drawing as template). In contrast to traditional heightfield-based terrain generation, our method allows to model layered structures like arches, caves or overhanging cliffs, which require a volumetric representation.

7.2 Performance

The overall performance of the terrain generation almost exclusively depends on the grid resolution. The number of Feature Curves only impacts the performance of the voxelization stage, while the complexity of the generated terrain has a minor impact on the performance of the surface reconstruction stage. The increase in execution time with increasing grid resolution matches theoretical expectations. Doubling the grid resolution in all dimensions increases the size of the computational domain by a factor of 8. Additionally, the numbers of iterations performed in the guidance field diffusion, noise field diffusion, and surface propagation has to be increased by a factor of two. Overall, this yields an expected increase in execution time by a factor of 16, which can also be observed in the performance measurements shown in Table 1. Our results show that a grid of size $128 \times 64 \times 128$ can still be computed in slightly less than a quarter of a second on an AMD HD7870, but for larger grids, additional optimizations will be necessary to reach interactive frame rates during terrain editing.

Table 1: *Computation times of the individual stages of the terrain generation pipeline at different grid resolutions, as well as for the complete pipeline. All timings were measured on an AMD HD7870.*

Pipeline stage	64×32 $\times 64$	128×64 $\times 128$	256×128 $\times 256$
Reset	0.38 ms	1.04 ms	6.36 ms
Voxelization	1.18 ms	1.67 ms	8.36 ms
Guidance field diffusion	9.24 ms	119.34 ms	1,840.74 ms
Noise field diffusion	4.00 ms	47.48 ms	705.31 ms
Surface propagation	3.32 ms	40.36 ms	565.77 ms
Surface reconstruction	5.01 ms	22.11 ms	147.10 ms
Complete pipeline	23.13 ms	232.00 ms	3,273.54 ms

8 Conclusion and Future Work

We proposed a novel approach to modelling and generating almost arbitrary terrain surfaces from sparse user input data, specifically aiming to enable the creation of terrain with arbitrary vertical layout, such as arches, caves, or overhanging cliffs. Our method combines the advantages of a volumetric terrain representation with the ability to quickly and efficiently edit large-scale terrain features by means of Feature Curves, a modelling tool based on three-dimensional parametric curves.

We first explained how parametric curves are outfitted with the necessary information for defining a terrain surface. Parametric curves are already used in the field of terrain modelling, however, we extended the existing approached by applying them to a volumetric terrain representation. Our proposed computational pipeline operates on a three-dimensional domain and combines voxelization, diffusion, and iso-surface extraction algorithms to generated a terrain surface from a set of Feature Curves. As the voxelization front-end of our pipeline is generalised to triangle meshes, the pipeline can be easily extended to import complete meshes from other applications, or generated from terrain heightmaps, in the future. We explained optional extensions that we added to the pipeline, including how to handle distributed computations in a subdivided domain for the purpose of improved flexibility and ultimately larger computational domains. The results showed that our method is capable of recreating complex terrain features that slope beyond the vertical, including overhanging rock faces, arches and the underbelly of a fictional floating island. Generating larger landscapes with many Feature Curves and a combination of numerous different terrain features is possible and we believe that curve-based modelling tools are suitable for modelling large-scale terrain features. Finally, our GPU-accelerated application prototype allows for interactive editing of reasonably sized terrain regions, making it a suitable method for modelling terrain in real-time applications such as game engine level editors.

Several aspects could be improved in future work. To reinforce Feature Curves as a multi-purpose terrain modelling tool, additional constraint properties could be added, including for example terrain surface material and vegetation density or more complex noise pa-

rameters (e.g. aimed at geologically motivated models). We will also implement support for a wider range of modelling primitives, specifically for the import of whole meshes. The normal field diffusion and surface propagation would benefit from further basic research into more accurate models of the local terrain surface contour. Finally, the performance of the computationally intense diffusion process used in the terrain generation pipeline could be improved, for example by utilizing a multigrid method. Such an implementation could furthermore be incorporated into a coarse-to-fine level-of-detail scheme for the terrain using volume bricking.

References

BLENDER FOUNDATION, 2016. Blender - open source 3d creation suite.

CARPENTER, L. C. 1980. Computer rendering of fractal curves and surfaces. *SIGGRAPH Comput. Graph. 14*, 3, 109–.

CRYTEK, 2016. Cryengine terrain editor.

DYKEN, C., ZIEGLER, G., THEOBALT, C., AND SEIDEL, H.-P. 2008. High-speed marching cubes using histopyramids. *Computer Graphics Forum 27*, 8, 2028–2039.

EBERT, D. S., MUSGRAVE, F. K., PEACHEY, D., PERLIN, K., AND WORLEY, S. 2002. *Texturing and Modeling: A Procedural Approach*, 3rd ed. Morgan Kaufmann Publishers Inc.

EMILIEN, A., POULIN, P., CANI, M.-P., AND VIMONT, U. 2014. Interactive Procedural Modelling of Coherent Waterfall Scenes. *Computer Graphics Forum*, 1 – 15.

EPIC GAMES, INC., 2016. Unreal engine 4 landscape outdoor terrain.

EVANS, L. C. 1998. *Partial Differential Equations*. American Mathematical Society.

GAIN, J., MARAIS, P., AND STRASSER, W. 2009. Terrain sketching. In *Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games*, ACM, I3D '09, 31–38.

GÉNEVAUX, J.-D., GALIN, E., GUÉRIN, E., PEYTAVIE, A., AND BENES, B. 2013. Terrain generation using procedural models based on hydrology. *ACM Trans. Graph. 32*, 4, 143:1–143:13.

HNAIDI, H., GURIN, E., AKKOUCHE, S., PEYTAVIE, A., AND GALIN, E. 2010. Feature based terrain generation using diffusion equation. *Computer Graphics Forum (Proceedings of Pacific Graphics) 29*, 7, 2179–2186.

HUGH HONOUR, JOHN FLEMING. *A World History of Art*. Laurence King Publishing.

LENGYEL, E. 2010. Transition cells for dynamic multiresolution marching cubes. *J. Graphics, GPU, & Game Tools 15*, 2, 99–122.

LORENSEN, W. E., AND CLINE, H. E. 1987. Marching Cubes: A high resolution 3d surface construction algorithm. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, ACM, SIGGRAPH '87, 163–169.

NATALI, M., LIDAL, E. M., PARULEK, J., VIOLA, I., AND PATEL, D. 2013. Modeling terrains and subsurface geology. *Proceedings of EuroGraphics 2013 State of the Art Reports (STARs)*, 155–173.

NGUYEN, H. 2007. *GPU Gems 3*. Addison-Wesley Professional.

PERONA, P., AND MALIK, J. 1990. Scale-space and edge detection using anisotropic diffusion. *IEEE Trans. Pattern Anal. Mach. Intell. 12*, 7, 629–639.

PEYTAVIE, A., GALIN, E., GROSJEAN, J., AND MERILLOU, S. 2009. Arches: a framework for modeling complex terrains. *Computer Graphics Forum 28*, 2, 457–467.

SINGH, K., AND FIUME, E. 1998. Wires: A geometric deformation technique. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, ACM, New York, NY, USA, SIGGRAPH '98, 405–414.

SMELIK, R. M., DE KRAKER, K. J., TUTENEL, T., BIDARRA, R., AND GROENEWEGEN, S. A. 2009. A survey of procedural methods for terrain modelling. In *Proceedings of the CASA Workshop on 3D Advanced Media In Gaming And Simulation (3AMIGAS)*, 25–34.

TASSE, F. P., EMILIEN, A., CANI, M.-P., HAHMANN, S., AND DODGSON, N. 2014. Feature-based terrain editing from complex sketches. *Comput. Graph. 45*, C, 101–115.

UNITY TECHNOLOGIES, 2016. Unity manual - terrain engine.

ŠT'AVA, O., BENEŠ, B., BRISBIN, M., AND KŘIVÁNEK, J. 2008. Interactive terrain modeling using hydraulic erosion. In *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '08, 201–210.

XU, C., AND PRINCE, J. L. 1997. Gradient vector flow: A new external force for snakes. In *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*, IEEE, 66–71.

YANG, J. C., HENSLEY, J., GRÜN, H., AND THIBIEROZ, N. 2010. Real-time concurrent linked list construction on the gpu. In *Computer Graphics Forum*, vol. 29, Wiley Online Library, 1297–1304.