Erosion, Self-Organization, and Procedural Modeling

by

Alexei Pytel

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2015

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Procedural modeling of natural objects such as coastlines and terrains in combination with their characteristic erosion features involves integration of appropriate physical models with the procedural approach and culminates in the development of physically-based simulations. I have invented a modeling paradigm for designing this type of simulations in a way that generalizes formation of complex relationships between erosion features, such as the tributary relationship. My generalization uses self-organization to define where erosion occurs and how it propagates rather than emphasizing the exact mechanism of erosion and the details of what happens during each erosion event. Propagation of state changes due to self-organization can also lead to emergence of fractal character, which is essential for modeling of natural objects, without explicit fractal synthesis. I successfully apply my methodology to procedural modeling of dunes, coastlines, terrains that undergo hydraulic erosion due to channel networks, and 3D channel networks that form underground.

Acknowledgements

I would like to thank my examination committee for their help: Drs. Stephen Mann, William B. Cowan, Justin W. L. Wan, Jonathan Li, and W. Randolph Franklin. I am grateful to my supervisor, Dr. Stephen Mann, for his guidance and support.

As a student of CGL, I have benefited from a great learning atmosphere and many research opportunities. I would be ignorant of many key concepts in computer graphics if the CGL faculty, especially Dr. William B. Cowan, did not share their knowledge with me.

Many thanks to student members of CGL and the HCI lab for creating a fun environment for learning and research: Elodie, Matt, and Tyler, for friendly discussions of research ideas; Andrew, Eugene, Kate, Marshall, Philippe, Richard, Tiffany, and Yasu, for lab events.

Thanks to David for suggestions on programming with OpenGL.

This research was partially funded by NSERC.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

When geometric representation of objects for use in computer graphics is achieved with procedural modeling, one of the modeling goals is typically the development of a construction procedure that encapsulates aspects of shape that are tedious or impossible to create by hand. An algorithmic solution to such modeling problems is desirable, since a well-designed construction algorithm offers the flexibility of generating similar shapes without changing the basic structure of the algorithm. One of the most popular examples of procedural modeling is that of fractal shapes, which can be difficult to draw without using a computer due to both the large amount of detail they may possess and because their defining mathematical properties need to be translated into algorithmic form for drawing.

Procedural construction of geometric models of physical objects is a closely related modeling problem. In a basic form, the problem can be reduced to constructing fractals, as fractals resemble many natural objects. For example, a random walk looks similar to a coastline, because both objects can be considered to be planar curves and have equal amounts of detail at different levels of magnification. However, a coastline construction based on random walks also has some undesirable properties: it may contain self-intersections and it can not create plausible outlying islands. In general, explicit fractal synthesis motivated by a resemblance between a physical object and a fractal can not capture aspects of the physical object that go beyond the resemblance, especially if the resemblance has no deep physical meaning. Similarly, the lack of physical interpretation also tends to be an obstacle for generalizing a construction procedure to include additional features of the physical object.

A better solution to the modeling problem integrates a physical model with the procedural approach. Intuitively, the appearance of an object might arise as a result of changes

it undergoes due to a natural process, such as erosion. A physical model of the process can provide a formulation of the behavior and a corresponding simulation, which can be incorporated into the construction algorithm of the procedural method. The difficulty in developing procedural construction algorithms using this scheme comes from the way the simulation component is abstracted from geometry. For example, it is possible that a physically-based simulation is formulated in 1D, when the goal is to construct a 2D geometric representation. This tends to be a problem with simulations built on reductionist formulations that lack a way to express any behavior that is not explicit in the underlying model. In general, the simulation may be structured based on a set of "rules" different from the construction rules used in the procedural modeling component.

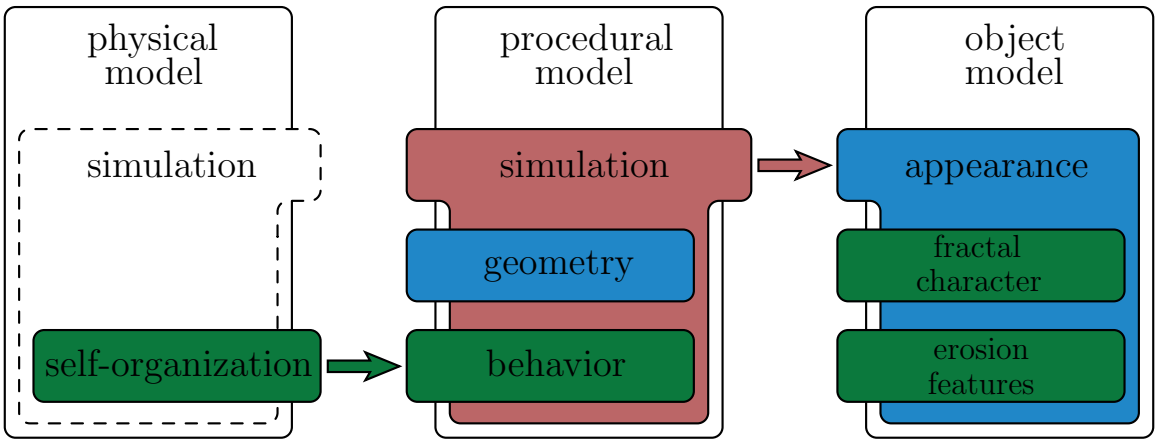


Figure 1.1: Flow of model development in procedural modeling with aspects of simulation and self-organization.

One of the main themes of my research is to remedy the disconnection between the physical and the procedural models by constraining the type of physical models to those that incorporate principles of self-organization. I consider most simulations to exhibit some aspects of self-organized behavior in the form of state-changing events that can cause other events. For example, in a simulation of the heat equation, when the temperature of a location is raised, that location propagates heat to its neighbors. Additionally, my approach requires the simulations to be directly usable as a construction algorithm in the procedural model, as illustrated in Figure 1.1. Since this requirement is met by simulations that are structured around propagation of state changes between neighbors in a discretization of a problem domain that eventually becomes the geometric representation of a given physical object due to emergent behavior, I consider this requirement to be a constraint on the type of self-organized behavior underlying the simulations. So, the scheme of Figure 1.1

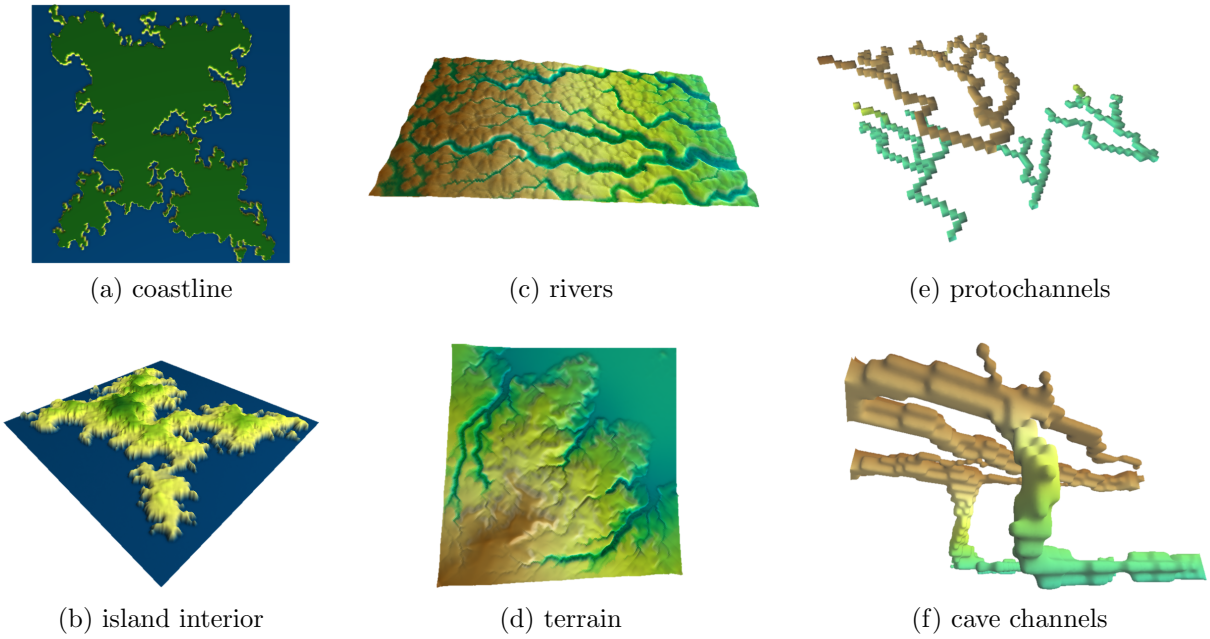
also implies the main step of my development process for modeling a given object, which is designing a simulation that acts on a geometric representation of the object and shapes it according to a self-organized formulation of its behavior.

The restricted type of self-organized simulations that I use in my approach is capable of constructing objects with fractal character in an implicit way, through emergent behavior. Therefore my approach can be an alternative to procedural modeling based on explicit fractal synthesis, but with a physical justification that links the construction procedure to the dynamics of the corresponding physical system. The property of the simulations that causes the fractal character to emerge is the propagation of state changes, which can distribute themselves at different length scales in a fractal manner. A physical theory of fractal dynamics called self-organized criticality (SOC) explains this behavior as the avalanching property of a class of physical systems. However, propagation of state changes similar to avalanching is not limited to the evolution of SOC systems. Therefore, in my modeling approach I extend avalanching to a general modeling paradigm for developing simulations that exhibit emergent behavior due to propagation of state changes.

In simulations of the effects of fluvial, thermal, and aeolian processes, the state changing avalanches take the form of avalanches of erosion, making construction of erosion-related features a natural application of my modeling approach. Simulations of erosion based on avalanching construct objects that exhibit both fractal character and erosion features, which is a combination that is difficult to achieve using techniques based on explicit fractal synthesis, due to their dissociation from physically-based models of erosion. The avalanching paradigm also has the benefit of abstracting erosion-related behavior, so that the modeling process for the physical systems that exhibit erosion can proceed in a unified manner.

In general, modeling of features produced by erosion is desirable in procedural modeling, because erosion features make models of natural landforms look realistic. This is another reason why I have focused on developing my modeling paradigm to emphasize erosion and also constrained the domain of my simulations to discretized 3D surfaces and volumes. The discretization, in combination with my concept of self-organization in simulations, forms the basis for a common procedural modeling framework that I have created to encompass my methodology. I have applied my framework to modeling of various types of landforms, such as dunes, coastlines, rivers, and caves. Here is a summary of my contributions:

1. I developed avalanching as a general modeling paradigm for self-organized simulations and designed a corresponding agent-based framework for implementing the simulations. I have shown that my methodology unifies the modeling process for



(a) coastline (c) rivers (e) protochannels

(b) island interior (d) terrain (f) cave channels

Figure 1.2: Simulation examples produced with my procedural modeling framework.

simulations of erosion-related features on terrains and inside porous rock. I also demonstrated additional benefits of avalanching, such as controlling the amount of folding in coastline shape and modeling rivers as channels that have width.

2. Among several coastline erosion models that I have implemented using my framework, I contribute a coastline erosion model that uses the avalanching paradigm to change the fractal character of the resulting coastline and control its "foldedness". My coastline model is also able to produce coastlines with mixed behavior and construct island interiors (Figure 1.2(a, b)).

3. I contribute a model of river-like erosion on terrains (Figure 1.2c) that reproduces several types of self-organized behavior: competition between emerging channels, tributary capture, convergent and divergent flow, and formation of primary and secondary valleys. My model is also able to produce wider conduits created by confluence of channels and realistically handle accumulation of water into flooded areas and lakes. The latter two features are new for self-organized simulations based on emergent behavior described by SOC.

4. I combine my coastline and terrain erosion models in a two-part method of modeling

terrains (Figure 1.2d). The first stage creates a terrain with controllable appearance and desirable fractal and hypsometric characteristics, while the second stage adds river-like channels and effects of diffusive and advective erosion caused by the flow of water in the channels. Both stages accomplish their tasks using formulations of erosions based on my avalanching paradigm. I evaluate the resulting terrains using spectral and hypsometric (area-altitude) analysis. The novelty of my method is two-fold: first, it is a procedural method that uses only self-organization principles to create terrains with hydraulic erosion features, and, second, it unifies modeling of erosion features associated with coastlines and rivers .

5. I contribute a two-stage model of subterranean channel growth due to dissolution. The first stage describes protochannel development (Figure 1.2e) and the second stage models further channel behavior (Figure 1.2f), generalized from my model of river-like erosion. The novel feature of the second stage of the model is its combination of self-organization of flow and pressure, which at the time of this writing makes my model the only of its kind, as well as the most expressive procedural model of cave-like channels in terms of its ability to represent a special type of zigzagging behavior commonly exhibited by cave channels.

A man of true science uses but few
hard words, and those only when
none other will answer his purpose;
whereas the smatterer in science
thinks, that by mouthing hard
words, he proves that he
understands hard things.

The World in a Man-of-War
Herman Melville

Chapter 2

Background and Previous Work

2.1 Fractals

Many naturally occurring landforms, such as coastlines and terrains, have prominent fractal characteristics. That is why, in a broad sense, all methods used to model such objects in computer graphics rely on fractal synthesis. However, in some modeling contexts it is possible to construct geometry by explicitly synthesizing a number of fractal objects whose fractal character is appropriate in a high-level way. This application of fractal synthesis is a part of a classical modeling approach, called ontogenetic modeling.

This chapter explains what it means for a shape to have a fractal character and describes several methods of explicit fractal synthesis that are suitable for ontogenetic modeling. The examples of fractal synthesis algorithms illustrate several issues that are relevant to the development of my own modeling approach, which creates fractals implicitly. First, spectral density is a useful tool for assessing such aspects of a fractal as roughness and self-affinity.

Second, fractal synthesis can suffer from several kinds of artifacts. Third, modeling based on fractal character alone does not provide for the semantics of features that are necessary to reproduce the effects of erosion.

2.1.1 Self-Similarity

It is possible to view a fractal as a set of geometric points. This interpretation makes it simple to state some essential relationships between fractals.

Definition 1. Relationships between fractals.

Equality relationship: fractals F and G are equal if they are made up of the same set of points up to a fixed translation or rotation.

"Part of" relationship: fractal G is a part of fractal F if the points of G are contained in F up to a fixed translation or rotation.

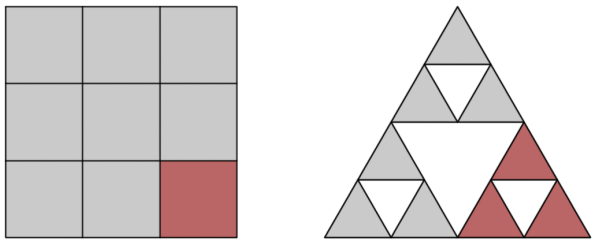
Scaling relationship: the set of points in a fractal F can be scaled by a nonzero factor s producing a new fractal $s \cdot F$.

Self-similarity: a fractal F is self-similar if it can be divided into several identical parts $G_1, G_2, \ldots, G_p$ and there is a scaling factor s such that $s \cdot G_i = F$ for each G_i.

Let M be a generalized measure of length in dimension D. In other words, M measures length for $D = 1$, area for $D = 2$, and so on. M is measured in the same units as ϵ^D, where ϵ is the ruler size. Consider measuring the area of the square in Figure 2.1a, which is self-similar with $p = 9$ and $s = 3$.

Since $s = 3$, using $\epsilon = \frac{1}{s} = \frac{1}{3}$ yields an area of $\left(\frac{1}{3}\right)^2$ for any one of the nine small squares. To sum the areas of the small squares and obtain the area of the original square, it is possible to either use the value of p, which is explicitly known due to self-similarity, or to recover p using an observation about the relationship between p and s. This observation is that $p = s^D$ for any D that is a positive integer.

It is clear that the square belongs to a class of objects that restrict the choice of p and s in such a way that D can only be a positive integer. A type of fractals called deterministic self-similar fractals (defined below) do not share this restriction. To define such objects, it is necessary to generalize M to work in a dimension that is a positive real number. Based on the example of measuring the area of the self-similar square, M must satisfy



(a) (b)

Figure 2.1: Self-similar objects: (a) $[0,1] \times [0,1]$, (b) diagram representing the Sierpinski triangle.

$$\begin{aligned} M(F) &= p \cdot M(G_i) \\ M(F) &= s^D \cdot M(G_i), \end{aligned} \tag{2.1}$$

where F and G_i are as in the definition of self-similarity. Note that the relationship $p = s^D$ is recovered, but now D can be a positive real number.

Definition 2. A *deterministic self-similar fractal* is a self-similar object such that if it is divided into p identical parts and it is self-similar to them with the corresponding scaling factor s, then its dimension is $D = \frac{\log p}{\log s}$ and D is a positive real number.

Figure 2.1b shows the first few steps of constructing the Sierpinski triangle, but it should be understood as if the construction was carried out completely. The entire construction is self-similar to the colored part with parameters $p = 3$ and $s = 2$. It is also self-similar to a single one of the shaded triangular regions with parameters $p = 9$ and $s = 4$. Thus the fractional dimension of the Sierpinski triangle is $\frac{\log 9}{\log 4} = \frac{\log 3}{\log 2} \approx 1.585$.

Deterministic self-similar fractals, such as the Sierpinski triangle, are defined using the exact relationship $p = s^D$, which necessitates that the whole object is self-similar to p identical parts, which become equal to the whole when scaled by a factor s. However, $p = s^D$ arises from a notion of measurement as described in Equation 2.1. This makes a further generalization possible.

Definition 3. A fractal is *statistically self-similar*, if for a given ruler size $\epsilon = \frac{1}{s}$ Equation 2.1 holds in an average sense, or, equivalently, the scaling relationship $p = \frac{1}{\epsilon^D}$ holds on average.

The above definition makes use of an explicit ruler size in place of the fixed parameter s. Similarly, it is possible to de-emphasize the parameter p. If F is a statistically self-similar fractal, then $M(F) \approx p \cdot \epsilon^D$ in a statistical sense. So it is possible to express the self-similarity relationship between F and its sub-parts, using $p \approx \frac{M(F)}{\epsilon^D}$. If F is a curve, this provides a way to measure its regular $D = 1$ length:

$$L(\epsilon) = \frac{M(F)}{\epsilon^D} \cdot \epsilon = C \cdot \epsilon^{1-D} \tag{2.2}$$

Thus, if F is a statistically self-similar curve, the plot of L measurements corresponding to a set of selected ruler sizes ϵ should show a power relationship. The slope of a log-log plot of the same data, called a Richardson plot, should be close to constant and can be used to find the value of D. One of the first examples of this procedure was to estimate the fractal dimension of the coast of Great Britain. Since the slope of the plot is nearly constant, it is possible to conclude that the coastline is a statistically self-similar curve. Kaye's text provides a more detailed discussion of using Richardson plots to analyze fractal curves [28]. I discuss fractal dimension of coastlines in the context of procedural modeling in Section 4.4.4.

2.1.2 Self-Affinity

Interpreting fractals as sets of geometric points makes it possible to give the points coordinate values. In a given coordinate system, the uniform scaling that characterizes self-similarity is the same as equal scaling along all defined axes. This suggests a generalization of the concept of self-similarity by allowing scaling that is not the same in all directions. One generalization of this kind is self-affinity, which is usually stated in a functional setting.

Observe that some fractals can be expressed as graphs of functions. The present discussion is limited to considering functions of just one or two variables. In a 1D setting, the graph of a function $f(x)$ is the set of points $\{(x, f(x)) \mid x \in U\}$, where U is the domain of f; in a 2D setting, the graph is $\{(x, y, f(x, y)) \mid (x, y) \in U\}$. Now, distinguish the direction of the ordinate axis and consider two ways to scale a graph:

$$(\vec{\boldsymbol{x}}, f(\vec{\boldsymbol{x}})) \mapsto (\vec{\boldsymbol{x}}, f(b\vec{\boldsymbol{x}}))$$
$$(\vec{\boldsymbol{x}}, f(\vec{\boldsymbol{x}})) \mapsto (\vec{\boldsymbol{x}}, cf(\vec{\boldsymbol{x}}))$$

To test f for self-similarity, set $c = \frac{1}{b}$, because the scaling should be uniform in all directions. Select a subset of the graph and apply both scaling operations to it: $(\vec{\boldsymbol{x}}, f(\vec{\boldsymbol{x}})) \mapsto (\vec{\boldsymbol{x}}, \frac{1}{b} f(b\vec{\boldsymbol{x}}))$. The scaled subset should match the whole graph (up to a translation), which can be written as $f(\vec{\boldsymbol{x}}) = b^{-1} f(b\vec{\boldsymbol{x}})$. A generalization of this property motivates the following definition of self-affinity. The text of Barabási and Stanley [2] contains a more detailed discussion.

Definition 4. A function f is *self-affine* if $f(\vec{\boldsymbol{x}}) = \left(\frac{1}{b}\right)^{\alpha} f(b\vec{\boldsymbol{x}})$. The function is statistically self-affine if the relationship holds in an average sense. The graph of f is a *(statistically) self-affine fractal*.

Calculating the fractal dimension D of a self-affine fractal is less straightforward than in the self-similar case. It is possible to cover the fractal with boxes whose size in D dimensions is ϵ^D and recover the relationship $p \approx \frac{M(F)}{\epsilon^D}$, which is characterized by D. However, using this relationship to estimate D can lead to an ambiguity based on the size of ϵ. Voss [68] discusses this issue in the context of fractional Brownian motion.

Mountains are an example of (statistically) self-affine fractals, because looking at mountains from the side and zooming away from the horizon makes the mountain profile appear flatter, suggesting unequal scaling in the height direction.

2.1.3 Fractional Brownian Motion

Definition 5. *Fractional Brownian motion (fBm)* is a statistically self-affine fractal that can be expressed as the graph of a function $B(\vec{\boldsymbol{x}})$ that satisfies

$$E(|\Delta B|) \propto ||\Delta \vec{\boldsymbol{x}}||^H, \tag{2.3}$$

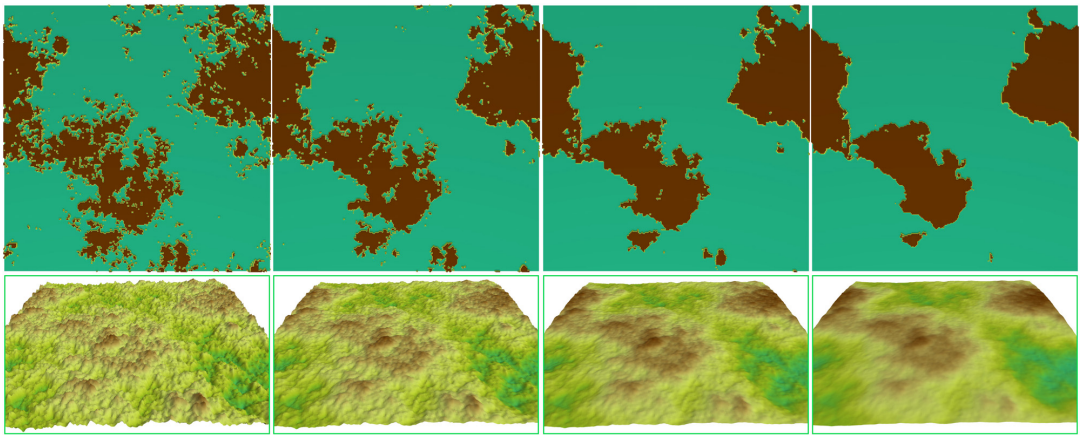
where E denotes expected value and $0 < H < 1$. In the special case when B is a function of one variable and $H = \frac{1}{2}$, B becomes a trace of the familiar version of Brownian motion, which is a random walk.

To see that fBm is self-affine consider that scaling $\vec{\boldsymbol{x}}$ to $b\vec{\boldsymbol{x}}$ scales $||\Delta \vec{\boldsymbol{x}}||^H$ to $b^H ||\Delta \vec{\boldsymbol{x}}||^H$. Since the factor b^H is a constant, the value of $B(\vec{\boldsymbol{x}})$ also becomes scaled by b^H since $B(\vec{\boldsymbol{x}})$ can be obtained from deltas via a summation. In other words, $H = \alpha$ from the definition of self-affinity.

Now consider the 2D case in which fBm is a surface. Walking along this surface from one point to another means taking several $\Delta \vec{\boldsymbol{x}}$ steps in the domain, finding the appropriate

values of ΔB, and summing them to find the destination point. Each step can be viewed as encountering a fault (a jump) in the surface that forces the walker to rise or descend by the corresponding amount ΔB. Indeed, one way to model an fBm surface is to start with a plane and apply a repeated faulting process to it, such that it selects an arbitrary line in the domain and raises the surface on one side of the line. If the location of the faults follows a Poisson distribution and the change in height across the faults has a Gaussian distribution, the result approximates a Brownian ($H = \frac{1}{2}$) fractal. Saupe and Voss provide more detail on this procedure [59, 68].

Mandelbrot proposes that synthesis of fBm traces can serve as a suitable model for the Earth's surface. The justification for this procedure is that the cumulative effect of faulting produced by the motion of tectonic plates is to create a pattern of height changes in a given direction that is similar to a trace of fBm [34].



(a) $H = 0.3$ $D = 1.7$ (b) $H = 0.5$ $D = 1.5$ (c) $H = 0.7$ $D = 1.3$ (d) $H = 0.9$ $D = 1.1$

Figure 2.2: The coastlines, or level sets, of fBm terrains. The bottom row shows fBm terrains with different parameters H and the top row shows a corresponding level set, as a set of islands. D is the fractal dimension of the coastline as a curve.

This type of fractal model for a planet suggests a unified perspective on coastlines and terrains from the point of view of using fractals to generate them. Given a terrain, coastlines can be produced by simply flooding it to a certain level. In other words, coastlines are level sets of terrains. Constructing a level set of a graph of a self-affine function removes all points except for those that are at a fixed height, which ends up removing the dimension in which the full graph has to be scaled differently for the self-affine relationship to hold. So a level-set of a self-affine fractal is self-similar. Therefore, it is consistent to model coastlines

as self-similar fractals and terrains as self-affine ones. Figure 2.2 illustrates this idea. In Section 4.3.4 I propose a combined method of modeling terrains and coastlines that also relies on the level set connection, but can produce additional hydraulic erosion features caused by flow of water over the terrain.

2.1.4 Spectral Character of fBm

Informally, the Fourier transform allows a function to be represented in terms of complex exponentials $e^{\mathbf{i}\theta} = \cos\theta + \mathbf{i}\sin\theta$. Such a representation is called a *spectral* or *frequency domain representation* and contains information about the frequency and amplitude of periodic signals (corresponding to the complex exponentials) that make up the behavior of the function in question. Since under sufficiently general conditions the original *space domain* function can be recovered with an inverse transform, the spectral representation can be considered to uniquely determine the function.

To apply the above concepts to fractals such as fBm, consider that those fractals have a functional representation and in practice are discretized on a grid. Therefore, let such a fractal be represented as a complex array of N uniformly spaced samples: $(f[0], \ldots, f[N-1])$. The frequency domain representation is also going to be a similar array of N points computed using the discrete version of the Fourier transform defined below.

Definition 6. The *Discrete Fourier Transform (DFT)* of $(f[0], \ldots, f[N-1])$ and its inverse are

$$\mathcal{F}(f)[m] = \sum_{n=0}^{N-1} f[n] e^{\frac{-2\pi\mathbf{i}nm}{N}} \tag{2.4}$$

$$\mathcal{F}^{-1}(f)[m] = \frac{1}{N} \sum_{n=0}^{N-1} f[n] e^{\frac{2\pi\mathbf{i}nm}{N}}. \tag{2.5}$$

Spectral density is a concept closely related to the Fourier transform. Although there are several alternative definitions of spectral density, I have chosen the following definition, because I intend to use the concept primarily in a qualitative way and in the setting of functions discretized on a grid.

Definition 7. For a discretized set of samples f the *spectral density* is

$$S(f)[i] = \frac{|\mathcal{F}(f)[i]|^2}{\Delta s}, \tag{2.6}$$

where $0 \leq i < N$, N is the number of samples, and Δs is the uniform spacing between the samples. $S(f)[i]$ also corresponds to frequency $\theta = i\Delta s$ and can be equivalently denoted as $S(f)(\theta)$.

This definition is sufficient to plot the spectral density and observe the important frequency-domain property of fBm stated in Theorem 2.1.1. Additionally, the spectral density plot can show whether a synthesized terrain contains all the detail its level of discretization allows (i.e., all of the possible frequencies are present). It can also reveal creasing artifacts in the space domain representation of the terrain.

Theorem 2.1.1. If an instance of fBm is discretized as a set of samples f, then for $\beta = 2H + 1$

$$S(f)(\theta) \propto \frac{1}{\theta^{\beta}}. \tag{2.7}$$

Because my definition of spectral density is not a common one, I include the proof of the theorem in Appendix A.1. The power falloff relationship provides a way to compare different ways of constructing fBm that are used in procedural modeling, as discussed in the following section. Similarly, I use spectral density plots when I analyze the results of my procedural modeling method, which involves generating approximately self-affine terrains.

2.1.5 Fractal Synthesis

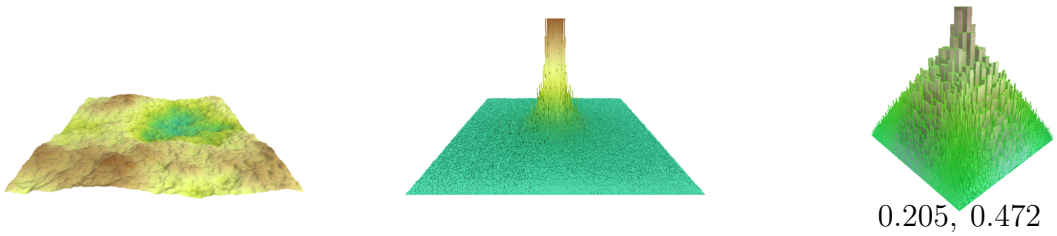
This section describes three methods of explicit fractal synthesis: spectral synthesis, random midpoint displacement, and noise-summing. I use the spectral synthesis method to illustrate Theorem 2.1.1 and introduce the triple-log plot of spectral density, which I use in subsequent chapters to visually assess whether a terrain possesses an approximately self-affine character (Sections 3.1.3, 4.3.2, 4.3.4, and 4.4.4). My examples of using the other two construction methods, random midpoint displacement and noise-summing, also show the utility of the spectral density plot and, together with a concluding discussion of ontogenetic modeling, serve to highlight a possible weakness of procedural modeling schemes based on fractals.

Discretization of a fractal on a grid combined with the recursive nature of fractal shape make it tempting to model levels of detail in a fractal explicitly, i.e., in a multi-resolution manner, which causes several artifacts, such as creasing (discussed further below and in Section 3.1.2). A major aspect of the design of my own modeling scheme is implicit

synthesis of fractal character that avoids using separate levels of detail. In my simulations, this is possible not only for terrains, but also for channel networks with streams of different size, which can appear to exist on separate levels of detail.

Spectral Synthesis

Voss [68] explains that fBm, as a class of random functions, is characterized by its spectral density. The effect of changing β in Equation 2.7 has an analogous effect to changing H from the definition of fBm (Figure 2.2). As a consequence of this relationship, one way to generate fBm is to synthesize the appropriate spectral density in the frequency domain and use the inverse DFT to obtain space domain samples. Saupe [59] describes this procedure in detail.



(a) (b) (c)

Figure 2.3: An fBm synthesis method. The plot in the center shows the magnitude of frequency domain samples (rescaled and cut off), which are generated first. Inverse DFT produces the result on the left. The triple-log plot on the right highlights a power relationship in the frequency domain. RMS error and R^2 value for a linear fit are stated at the bottom of the log plot.

Figure 2.3 demonstrates the construction procedure, which becomes statistically close to fBm with larger number of grid samples. Note that the frequency domain plot displays the magnitudes of frequency domain samples for convenience. These are proportional to $\frac{1}{f^{\beta_0}}$ with $\beta_0 = \frac{1}{2}\beta$. The power relationship is apparent in the triple-log plot as a linear falloff from its center. By comparison, the original plot of spectral density (Figure 2.3b) is more difficult to interpret, especially when it comes to the values corresponding to smaller frequences (away from the center) whose magnitude is small compared to the values in the center of the plot.

I use spectral analysis as a tool for assessing the fractal character of terrains, including those created procedurally with my method (Sections 4.4.4 and 4.4.5). To compare the

spectral density plots of different terrains more precisely, I fit a linear function to the spectral density data and compute two values that characterize the quality of the fit: RMS error and the coefficient of determination (Figure 2.3c). I state these two values at the bottom of every log plot of spectral density. Section 2.1.6 provides the details of the procedure.

Modern View of Random Midpoint Displacement

One of the basic methods of fractal synthesis is random midpoint displacement. It can generate a fractal mountain from a coarse shape by subdividing it and adding a random offset to the new vertices, generating increasingly finer levels of detail. In general such an approach has two shortcomings: the creasing due to the dependence of finer scales on the coarser ones and artifacts introduced by the subdivision process itself (without the random offsets). Miller explores these issues for several types of random midpoint displacement schemes, including a common one called diamond-square [37].

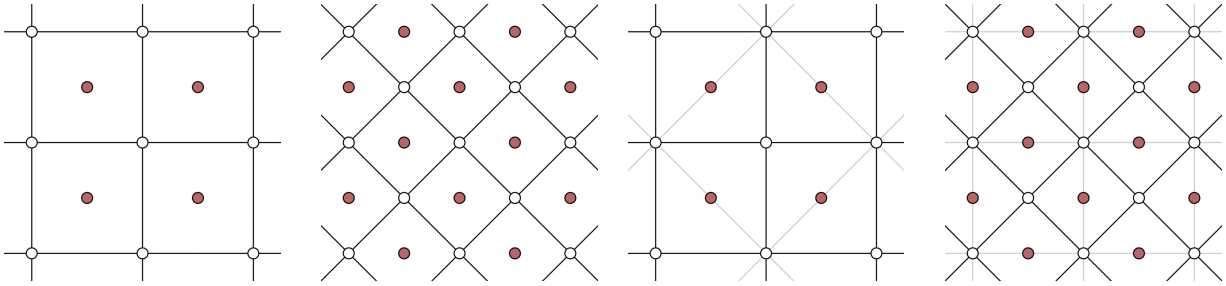


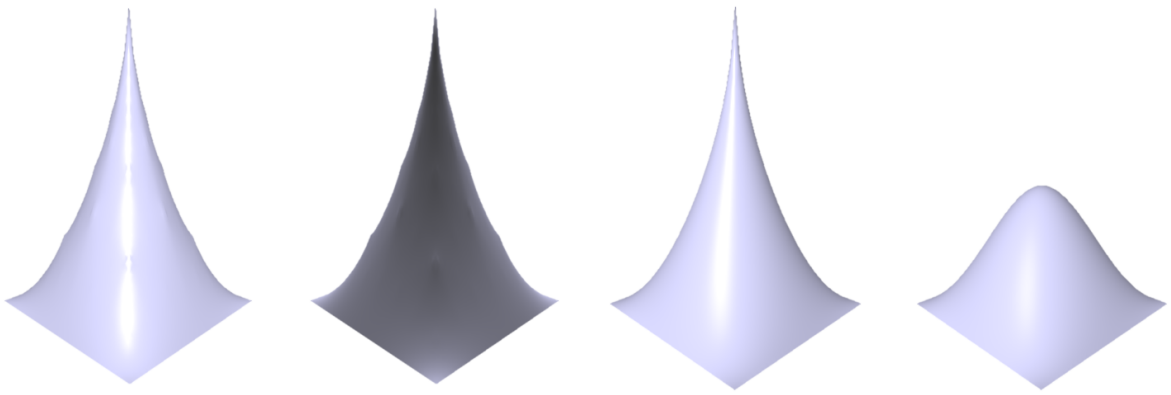
Figure 2.4: (a) the diamond-square construction adds new vertices (in red) inside squares then inside diamonds; (b) 4-8 subdivision adds new vertices (in red) in topologically the same way.

Figure 2.4a illustrates the diamond-square construction, which produces a new level of subdivison by adding new vertices inside grid squares followed by adding new vertices inside diamonds. Note that it is necessary to carry out both steps to produce the next finer subdivision level of a grid. Placement of the new vertices at the midpoint of the squares and diamonds contributes to several types of artifacts in the interpolating surface of the initial coarse shape: sharp peaks and regularly occurring bumps seen in Figure 2.5(a, b). I propose to update the algorithm with a better interpolant, which avoids these problems by

using 4-8 subdivision, introduced by Velho and Zorin [67]. This is different from Miller's solution to the problem, which substantially pre-dates the work of Velho and Zorin.

Figure 2.4b shows the result of splitting a grid according to 4-8 subdivision, which is topologically the same as the result of the diamond-square process. However, 4-8 subdivision adjusts the position of existing vertices when it adds new ones, resulting in an interpolant that is C^4 everywhere except at vertices of degrees other than 4 and 8, which are the only types of vertices created by the subdivision process. Degree 8 vertices appear because 4-8 subdivision treats a grid as having additional edges (shown in grey in Figure 2.4b) that split quadrilaterals into triangles in a parity-based pattern.

Compared with the grid-based definition of the diamond-square process the extra edges allow each step of 4-8 subdivision to use the same procedure (splitting of the grey edges in Figure 2.4b). Moreover, 4-8 subdivision is in a well-defined state after every step, which makes it possible to avoid increasing discretization density more than necessary due to applying the subdivision steps two at a time. One further advantage of 4-8 subdivision is that it readily extends to an adaptive scheme that increases subdivision density locally near fine features. I use the adaptive variant of 4-8 subdivision in Section 5.3.1.



(a) (b) (c) (d)

Figure 2.5: Comparison of the results of interpolation in the diamond-square construction and 4-8 subdivision: (a)-(b) front and back views of the result of the diamond-square construction, (c) result of 4-8 subdivision with the coarse shape frozen as in the diamond-square construction, (d) 4-8 subdivision applied everywhere in the interior of the grid.

Figure 2.5 compares the results of applying the diamond-square process (a, b) and 4-8 subdivision (c) to an initial coarse shape determined by a 3×3 grid of points, in which the

central point is raised above others. Figure 2.6 completes the fractal mountain modeling process by adding random displacement to the two interpolants. The diamond-square interpolant suffers from regularly occurring bumps and a sharp peak. Even if they can be hidden by random displacements, these artifacts are unavoidable consequences of poor interpolant shape. In contrast, the 4-8 interpolant does not produce the bumps and can also smooth away the sharp peak in the initial shape (Figure 2.5d). However, neither type of interpolant solves the fundamental problem of statistical dependence between fine and coarse detail in this type of construction.

As an example of using the triple-log density plot to assess scaling behavior of fractals, compare the plots in Figures 2.6d and 2.3c. The presence of the approximately linear falloff in Figure 2.6d demonstrates that random midpoint displacement is an approximation of fBm.

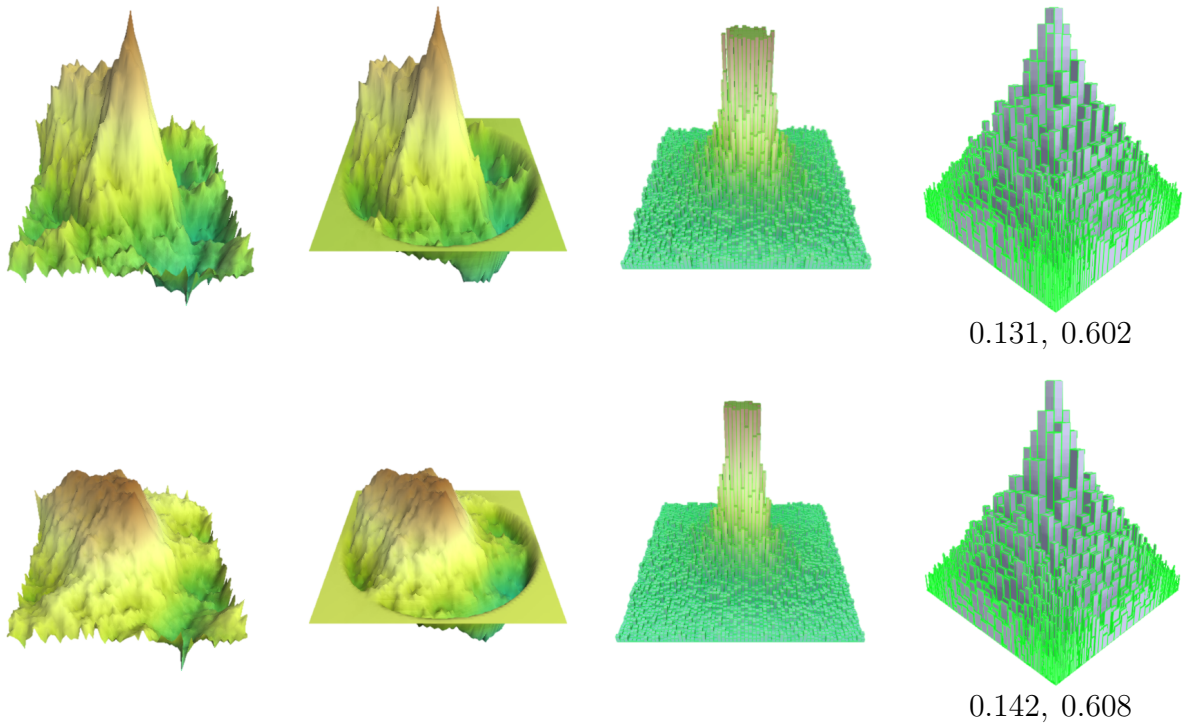


Figure 2.6: Fractal mountains constructed with the classic diamond-square process (top row) and the modernized algorithm that uses 4-8 subdivision (bottom row): (a) fractal mountain, (b) periodized version, (c) DFT of the periodized version (rescaled and cutoff), (d) triple log plot of DFT (rescaled).

Noise-Summing

A technique frequently used in computer graphics to approximate the results of more sophisticated methods, noise-summing starts with an irregular function and combines it with scaled down versions of itself in a multi-level way. The summation creates an approximate fractal character by introducing detail at different length scales. A common choice for the underlying function is a noise function, which is a special function with irregular bumps, not a fractal noise.

Figure 2.7 shows the result of applying the noise-summing procedure to Perlin's simplicial noise [22]. Summing Perlin's noise is a common procedure for approximating fBm [38].

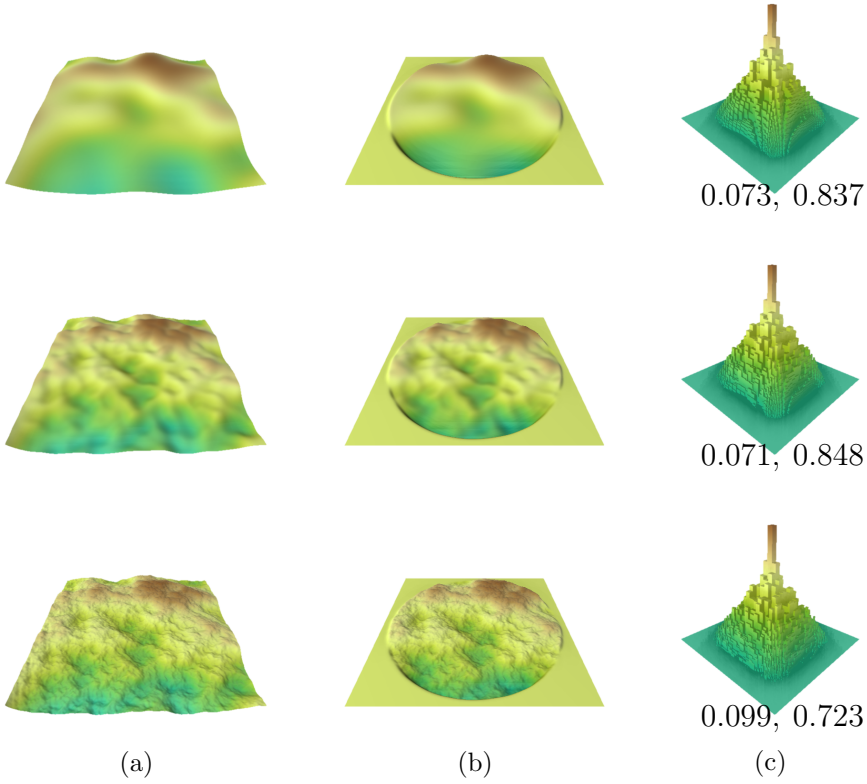


Figure 2.7: Fractal terrain constructed via summing varying numbers of noise levels (number of levels doubles in each row): (a) terrain, (b) periodized version, (c) triple log plot of DFT (rescaled).

The spectral density plot of the result exhibits the characteristic $\frac{1}{f^\beta}$ falloff, but does not contain many of the higher frequencies, which correspond to finer levels of detail (frequency increases towards the edges of the plot). This example demonstrates that the spectral density plot can be used to determine whether a construction procedure creates fine or coarse detail. It is desirable that a finished terrain contains as much detail as possible for its discretization density. In particular, the terrain will appear too smooth if it is missing detail corresponding to the higher frequencies.

Comparing Figures 2.3c and 2.7c shows that many levels of summing are needed to produce the highest frequencies possible at a given discretization density. Using so many levels of noise is computationally expensive. As a general implication for modeling, explicit

addition of one or several levels of detail using a procedure like noise-summing is unlikely to provide the object with a full range of detail that its discretization density can support.

Ontogenetic Modeling

Modeling a mountain or a terrain with an approximation to fBm produced with the diamond-square algorithm or noise summing is an example of the approach called *ontogenetic modeling*, i.e., modeling based on visible morphological character. This approach is advocated by Musgrave among others [38] for its ends-driven simplicity. Although the method is not physically-based, its use of fBm is fundamentally justified, because the shape of many natural objects contains $\frac{1}{f^\beta}$ noise [68].

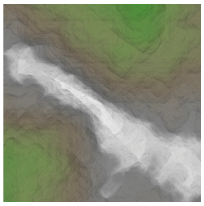


Figure 2.8: Illustration of the creasing artifact.

Explicit fractal synthesis used in ontogenetic modeling can be challenging to apply to problem settings that involve a diverse set of features. If these features appear to exist on different scales, it can be tempting to model them as a multi-level pyramid that can be later composited into a single object. However, Mandelbrot [34] warns that such human-recognized features can be illusory. Furthermore, combining distinct levels of detail can create an artifact called *creasing*.

Creasing appears because the combining can only be done differentially by offsetting the finer levels from the coarser ones. This introduces statistical dependences between the finer features, because they are combined with the same (or bordering) coarser features. The statistical dependence typically manifests itself visually as sharp creases, which can be made less apparent by smoothing. The effect of the shape of the coarsest level on the complete object can be especially noticeable. Figure 2.8 demonstrates a terrain, reproduced here from Section 4.4.5, that contains a visible grid of creases.

Another challenge for using fractal synthesis to model objects with a diverse set of features is related to modeling objects affected by erosion. Some erosion features can be related in complicated ways. For example, one stream can capture another as a tributary, resulting in a wider stream. Similarly, a river can create a lake by flooding a valley. The

semantics of such feature relationships can not be easily expressed in terms of fractals in a procedural way, even though it might be possible to model these features manually from pieces of fractals. However, a manual or automated process that naively combines fractals will be exposed to similar issues as modeling using explicit levels of detail.

Therefore, for my modeling approach, I have chosen a scheme (Chapter 3) that emphasizes physically-based creation of features, such as rivers, and creates approximate fractal character in an implicit way, as a secondary goal. Since my approach does not involve combination of multi-resolution features, it avoids the concomitant issues. However, the method may be unable to prevent artifacts such as creasing from arising due to extrinsic factors, such as initial conditions. That is why when I designed simulations based on my method and framework, I still had to consider issues related to ontogenetic modeling.

2.1.6 Spectral Analysis of Terrains

For a terrain to approximate fBm (i.e., to exhibit self-affine fractal character), the terrain's spectral density must approximate the characteristic spectral density falloff of fBm. I have developed the following procedure to compare spectral densities numerically. The results of the procedure are two numbers, RMS (root mean square error) and the coefficient of determination (R^2), which I report with every spectral density plot.

1. Collect the data $D = \{(r_0, z_0), \ldots\}$ from the logarithmic plot of spectral density.
2. Compute the least squares fit of D to a linear function L.
3. Find a transformation T that maps L to $z(r) = 1 - r$.
4. Compute the least squares fit of $T(D)$ to $z(r) = 1 - r$.
5. Report RMS and R^2 from the previous step.

Note that the data D (step 1) are in cylindrical coordinates, but the spectral density plots use rectangular coordinates. This limitation is due to the difficulty of drawing the necessary curved elements on a logarithmic cylindrical plot. So, it is necessary to apply the conversion $r_i = \sqrt{x_i^2 + y_i^2}$ to the raw data of the plot; the z coordinate is unchanged.

Additionally, some of the raw data of the plots have to be removed before the analysis. First, not all of the data are independent, because the frequency domain samples corresponding to the spectral density plot encode a real function. Roughly, about half of the data points have to be rejected for this reason, leaving 32770 for a 256×256 plot. Second,

data points with $z < \epsilon$ and data points in the 3×3 area in the center of the plot have to be similarly removed, because $\log 0$ is undefined. I choose $\epsilon = 0.001$, which typically results in about 0.03% data points rejected and never more than 3% with the following exception: in some cases the spectral density plots have missing higher frequencies, such as the plots in Figures 2.7, 4.23, and 4.26, so that about 6% of the data have to be removed, improving the fit for the remaining data.

For a base of comparison, I generated 33 fBm terrains with varying seeds and H in the range of $[0.5, 0.98]$. Voss states that $H = 0.8$ is a good choice for modeling of many natural phenomena [68]. The resulting range for RMS was $[0.199, 0.223]$; the range for R^2 was $[0.331, 0.526]$. So, to assess the falloff in a given plot of spectral density I carry out the analysis procedure and compute the RMS error and R^2 value. Values of RMS closer to 0 and values of R^2 closer to 1 indicate that the linear fit is better. However, a logarithmic plot of spectral density with RMS < 0.199 or $R^2 > 0.526$ suggests that the corresponding frequency domain data contains less noise than fBm.

2.2 Self-Organized Criticality

Self-organized criticality (SOC) is a theory of fractal dynamics in which a physical system approaches an attractor state that is scale-free. This state is also called a critical state of the system. Scale freedom is a concept related to self-similarity: when the features of a self-similar object are examined with a ruler of a given size (i.e., under different magnification levels), they are independent of the scale. That the critical state is scale-free implies that a SOC process creates geometric features on all scales of the objects that it acts on, or for all frequencies in the spectral representation. In general, avalanches of changes tend to occur, causing the changes to be distributed over a range of scales.

A pile of sand is one of the simplest examples of a physical system obeying the theory of SOC. The dynamics of this system cause the grains of sand to slip down steep slopes and the resulting changes to the shape of the sandpile propagate via avalanches. By the time the sandpile reaches the attractor state, the avalanches have propagated throughout the entire system (in a statistical sense) and the features of the resulting landscape are not tied to a specific length scale.

Duran [14] illustrates this behavior with the following 2D sandpile model. The sandpile is discretized into slabs of sand of a fixed size. Slipping of sand occurs when the addition or transport of a slab violates the sandpile's angle of repose. The slabs can form avalanches because the model requires a moving slab to transport the one directly underneath it. Otherwise, introduction of more sand would only cause slabs to move one at a time.

The main implication of SOC for modeling applications, such as procedural modeling of terrains, is the possibility of constructing shapes with fractal character in an implicit way via self-organization. In particular, slipping of material due to a slope constraint is a model of thermal erosion that causes material to break off and fall down due to weathering. The following list summarizes SOC properties and why they are valuable for procedural modeling.

Self-organization: The critical state of a SOC simulation is an attractor state. This is a benefit for modeling, as the solution to the geometric modeling problem is also the state to which the physical simulation converges. Additionally, self-organization de-emphasizes the importance of the initial conditions of a SOC simulation in producing the characteristic shape associated with the attractor state.

Link to dynamics: SOC simulates the evolution of certain physical systems by making an explicit connection to the dynamics of the systems. This is a major advantage over other modeling techniques that do not provide a justification for their method in terms of physics. In particular, SOC is suitable for modeling the shape of objects produced by fluvial and aeolian erosion.

Fractal character: SOC models produce objects that exhibit statistical self-similarity without a need for explicit fractal synthesis. In some modeling contexts, such as the sandpile example, this is all that is needed to capture the relevant aspects of the shape of the physical system being modeled. However, to make it possible to adjust the distribution of features in the attractor state in some way (such as by making one locality appear rougher, for example), it is necessary to re-introduce some notion of scale back into the system. Re-introducing scale generally means that the physics of the adjusted system no longer respects SOC. This type of adjustment is also difficult to control since dynamics, self-organization, and fractal character are inter-related in SOC.

Avalanching: The state changing events that lead a SOC system to the critical state occur in such a way that the probability of such an event occurring in a given location increases if similar events have previously taken place there. In other words, state changes are likely to cause chain reactions, or avalanches. I use this property to simulate SOC-like physically-based self-organization in new modeling contexts and adjust features that result from this process in useful ways.

2.3 Models and Simulations of Erosion and Material Transport

My main goal is to design a procedural modeling framework that can construct various types of erosion features in a common way. However, there exist many physical models of erosion, as transport of material due to erosion is subject to complicated dynamics, which are different in each specific case. Furthermore, not all conceptual models easily translate into algorithms for procedural modeling of geometry. This section discusses some previous work that includes both models of the physical phenomena related to erosion and modeling of the effects of erosion in a geometric sense. At a high-level my own modeling approach is a new way to develop a class of physically-based models of erosion for incorporation into simulations that construct geometry suitable for computer graphics applications.

2.3.1 Dunes

The development and evolution of aeolian sand dunes occurs due to the transport of sand grains by wind, primarily by a process called *saltation*. A sand particle undergoes saltation when the aerodynamic forces that it experiences cause it to lift, moving away from the surface. Eventually, the lift force diminishes and the particle sinks back to the surface in a characteristic parabola-like trajectory.

Kroy et al. [31] present a model of this process for isolated 1D dune profiles based on a sophisticated analysis of surface shear stress. This model shows how under certain assumptions a Gaussian hill can evolve into a characteristic dune shape. Specifically, the profiles can form a *slip face*, which is a steep slope on the lee side of the dune. The functional setting used in this model is a limitation that prevents it from being extended to a method capable of constructing 3D geometric models of dune fields.

Diniega et al. [12] take a similar approach to model interactions between dunes. As smaller dunes, which move faster, catch up to larger ones, the dunes combine into a complex, followed either by coalescence or ejection of a smaller dune in the downwind direction. This information allows dune interactions to be classified based on characteristic variables of the participating dunes. The final model of Diniega et al. operates on two separate scales: it computes characteristic variables for each dune at the scale of individual dunes and uses that information to determine what happens when the dunes collide at the scale of the entire dune field. This is a model of dune field behavior only and is not directly applicable to constructing dunes geometrically.

Werner [71] presents a 2D model in which the development of dunes occurs in a self-organized way, based on the emergent properties of simplified dynamics of sand transport. The essential feature of the proposed simulation is that each discrete slab of sand has a probability of being transported a fixed distance in the wind direction. At the end of the trajectory it can either be deposited or wind can transport it again. In a variation of this basic procedure, there is no sand transport out of a shadow zone, which is a location shielded from the wind due to the height of its neighbors. In addition to moving in the wind direction, sand can also slip downslope when the erosion and deposition events violate the angle of repose constraint by oversteepening a slope. This self-organized model of sand transport is capable of constructing dune field geometry that reproduces several different types of dunes found in nature. The simulated dunes also exhibit certain kinds of realistic behavior, such as smaller dunes moving faster than larger ones. Werner provides an interpretation of this phenomenon in terms of attractor dynamics, which suggests a decoupling of dune development from the small-scale aerodynamics of sand transport.

Onoue and Nishita [42] synthesize terrains with dunes and sand ripples by using a model that is similar to Werner's. However, the simulation of Onoue and Nishita computes a variable amount of sand and a trajectory of a variable length for each transport event. These computations involve a heuristic, which affects sand transport in a way similar to shadowing.

I have chosen to follow Werner's approach to support construction of dunes within my framework. The main reason is that more sophisticated reductionist approaches to modeling dunes are not directly applicable in a computer graphics context, as it is not their goal to construct a 3D shape that looks like a dune field. However, the reductionist models of dune physics are still helpful, since they provide information about the behavior of dunes.

2.3.2 Hydraulic Features

A large fraction of the literature on terrain generation is different from the focus of my research in one of two ways: emphasis on manual modeling techniques rather than procedural modeling and lack of emphasis on the synthesis of hydraulic erosion features. This distinction is necessary for a clear discussion of relevant previous work that follows, as procedural modeling frameworks combine ideas from many areas and can be difficult to classify. For instance, many interactive terrain modeling frameworks use noise (i.e., a building block of procedural modeling) to either generate additional surface detail or hide transitions between manually placed features [20, 65, 23]. Additionally, fractal synthesis

can be driven by elevation data, such as in the terrain generation method of Parberry [45], resulting in terrains with the same distribution of heights as natural eroded terrains, but without erosion features like rivers.

A key concept that characterizes interactive approaches to terrain modeling is the use of control curves to specify the shape of terrain features. Control curves have been used to directly control the shape of fractal terrains [63], as locations for placement of patches from exemplar terrains [72], as deformations in combination with voxel carving [20], and as constraint curves for parameters such as elevation and roughness [23]. Notably, the use of exemplar terrains in the method of Zhou et al. [72] allows the synthesized terrain to incorporate some erosion features. However, since the patches are small compared to the terrain the burden of modeling a large river network falls on the user. The control curve concept may be realized not only in terms of splines, but also in terms of a combination of discretized paths in a graph representing the terrain and cross-sectional profiles defining the shape of terrain features [57].

In contrast to the modeling frameworks discussed above, there is a body of previous work that contains true procedural modeling techniques that specifically emphasize formation of hydraulic erosion features, usually by simulation. Kaye presents some early work on simulating percolation of water through a 2D medium and random walks in a 3D porous solid [28]. This type of simulation does not involve erosion, but it does address the problem of modeling features associated with erosion, because it produces random walks that are reminiscent of rivers in the 2D case. The connection of this modeling problem to percolation theory is important historically, because it predates Mandelbrot's work on fractals.

On the other hand, it is possible to model river-like features in a purely synthetic way by using fractals. Mandelbrot [34] describes such an approach that uses *squigs*, which are a type of recursively-constructed curve. Prusinkiewicz and Hammel [49] integrate the squig construction with a fractal mountain construction in a terrain modeling system. Figure 2.9 shows a river-like squig that I generated using my shape grammar system [25].

Terrains created using affine fractals like in the essays of Saupe and Voss [59, 68] can serve as initial conditions for hydraulic simulations. For instance, the modeling method of Musgrave et al. [39] synthesizes a terrain with locally-varying fractal character and applies an explicit hydraulic erosion simulation to it.

Water simulation algorithms commonly appear as the main component of hydraulic erosion simulations. Musgrave's simulation is a type of water-column algorithm that assigns a volume of water to each node of a terrain and transports excess water from a location to all of its neighbors. The method of Roudier and Peroche [56] is similar, but transports excess

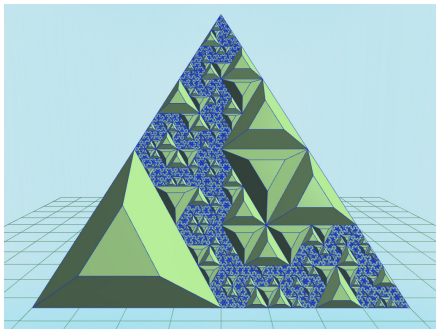


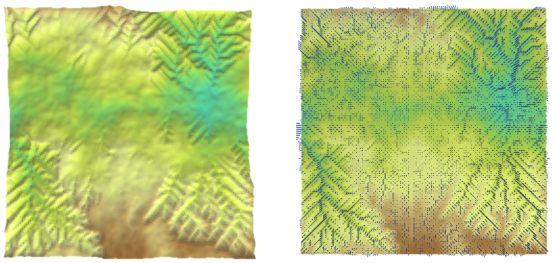
Figure 2.9: A river-like channel modeled using a squig.

water only to the lowest neighbor. However, when computing the associated amount of erosion Roudier and Peroche use additional parameters having to do with the stratification and faults in the underlying terrain. Additionally, the erosion computations rely on a graph of flow paths to determine the upstream locations for each node. This is similar to the area map concept, which is described further below.

In general, the water-column method stands out as a simple and versatile method of simulating shallow water flow that can be adapted to different situations. For instance, O'Brien and Hodgins [40] describe a variant of the algorithm that uses a formulation of water flow based on hydrostatics and apply it to animating splashing water. To compute water transport between neighbors in their version of the algorithm, Chiba et al. [9] use a velocity field obtained from particle motion, which makes their method more appropriate for mountain scenery, where the motion of water can vary a lot. Olsen [41] uses the water-column method as a practical game-oriented algorithm for constructing eroded terrains. Šťava et al. [69] use a hydrostatic scheme similar to that of O'Brien and Hodgins in an interactive GPU-based simulation of hydraulic erosion. Grønneløv and Jensen [21] extend the water-column algorithm for use with an adaptively subdivided terrain.

As an alternative to the water-column method, more sophisticated types of water simulation have also been used to model the effects of hydraulic erosion. For example, Krištof et al. [30] have demonstrated such a use of a water simulation based on smoothed particle hydrodynamics (SPH). In their approach, Chen et al. [8] develop a similar SPH-based simulation and also perform laboratory experiments of levee erosion that allow the computer simulation to be validated. Kamalzare et al. [27] enhance the SPH-based simulation with a model of hydraulic conductivity of the soil.

Modeling schemes of another type do not involve a bona fide water simulation and instead emphasize expected inter-relationships between the position of features like rivers



(a) (b)

Figure 2.10: Erosion model of Rodríguez-Iturbe and Rinaldo: (a) erosion acting on fBm terrain, (b) drainage area map displayed using bars that correspond to the logarithm of area.

and hills. The approach of Doran and Parberry [13] uses software agents to place features appropriately. Other approaches start by creating a plausible river network and then reverse engineering the terrain that fits it [4, 20, 65, 19].

One of the main ideas of a book by Rodríguez-Iturbe and Rinaldo [55] is that self-organized criticality is a valid new way of understanding the evolution of river channels. For a demonstration the authors present several versions of an algorithm for simulating a river network as a graph that encodes a "contributes water to" relationship. This graph, or *area map*, provides the information necessary to calculate the contributing area that drains at the location of each node. Computation of the drainage graph based on eight possible directions for each node of a regular grid is called the $D8$ *algorithm*. In the simulations of Rodríguez-Iturbe and Rinaldo discrete erosion events cause the nodes of the area map to become linked up as tributaries are captured. The authors go on to discuss the scaling properties and optimality of the emergent drainage network.

Figure 2.10 illustrates the erosion model of Rodríguez-Iturbe and Rinaldo according to my implementation (Section 4.4.5). Locations with the most contributing area are situated downstream, as seen in the visualization of the area map (the height of the bars is the logarithm of the drainage area). Note that rivers in the network are infinitesimally thin and that they stop upon reaching a local minimum. Additionally, many river segments are parallel because their direction is limited to one of eight possibilities according to the $D8$ algorithm.

Perron et al. [46] apply the area map concept in a more realistic way due to a PDE formulation for the amount of erosion at each site. This model of erosion can be either diffusion-dominated or advection-dominated. The authors study the growth of primary

and secondary valleys corresponding to these two cases.

As with other types of simulations, the implementation of hydraulic erosion in my framework bridges the gap between physical models, such as SOC and percolation, and constructing geometry for computer graphics applications. The SOC model of Rodríguez-Iturbe and Rinaldo is a suitable starting point to achieve this goal. However, as discussed by Tarboton [64], there are a few challenges with using area maps, such as handling of combinations of convergent and divergent flow and consistent classification of flow directions near local minima. The improved version of the $D8$ algorithm, called D_∞, splits the flow from each site between two of its neighbors, making grid artifacts less apparent. However, simulation of water collecting near a local minimum remains a major limitation for a terrain modeling application that needs to track the evolution of lakes, as opposed to the evolution of channel networks that area maps are suited for. One other modeling limitation is ambiguity in deriving the width of rivers from an area map.

To resolve these issues, I have chosen not to use the area map concept in my model and replace it with a specialized version of the water-column algorithm. This lets my simulation produce lakes that gradually fill up and become drained by rivers. Explicit simulation of water flow in my model also makes it possible for river networks to become wider downstream due to input from the tributaries. My water-column algorithm is also more flexible than $D8$ and D_∞, because it allows water to flow from one site to any combination of its neighbors. Finally, I have adopted a PDE formulation similar to that of Perron et al. to model diffusion-dominated erosion features.

The design of my framework relies on using agents to organize the specific simulations, including the hydraulic one. This relates my hydraulic erosion model to other terrain modeling frameworks (e.g., the method of Doran and Parberry [13]) that also use agents for constructing geometry. My model is also related to the modeling techniques that use fractal synthesis, although the fractal characteristics of the erosion features that my model produces arise in an implicit way due to self-organization.

2.3.3 Coastlines

One way to model coastlines is to synthesize fractal curves of an appropriate dimension. For example, a looping Brownian trail divides the plane into interior and exterior regions, despite any self-intersections. The border between these regions can serve as a coastline. Ward provides examples of this construction [70]. Rauch uses trails of fBm in a similar way [52]. However, as explained in Section 2.1.3, a more appropriate use of fBm in this context is to synthesize an fBm terrain and use its level sets as coastlines.

Sapoval et al. [58] propose the following model of coastline erosion based on SOC. In a grid of square finite elements that represent a region of land surrounded by ocean, the coastlines are identified in an implicit way as the subset of land blocks that border on ocean blocks. There is a force F that acts uniformly on each finite element of the coast and can erode it based on its intrinsic resistance R and degree of exposure to the ocean. As erosion events occur, the total perimeter of the coast L becomes longer, which attenuates F according to the formula

$$F(t) = \frac{f}{1 + \frac{gL(t)}{L(0)}}. \tag{2.8}$$

The parameters controlling the shape of the coast are constants f and g, as well as the distribution of R. Acceptable results can be achieved with uniformly random R values. The parameter f scales the force, while the parameter g scales the damping effect. The resulting shape of the coast is a statistically self-similar fractal and is an attractor state of the model.

Sapoval's model of coastlines uses self-organization in a similar way to Werner's model of dunes and Rodríguez-Iturbe's model of river basins. By selecting Sapoval's approach as a starting point for constructing islands in my framework, I have been able to use a unified infrastructure for modeling all three landform types. I have also been able to extend Sapoval's model in several ways that are useful in different modeling scenarios.

In particular, applying my avalanching modeling paradigm has allowed me to develop an extended model that can produce coastlines of different roughness. My approach also allows coastlines with different properties to develop side-by-side and plausible interiors for islands to be generated. I use a variant of my island interior generation algorithm to construct initial conditions for my method of modeling terrains with rivers, corresponding to the idea (described in Section 2.1.3) that coastlines are self-similar curves that are level-sets of self-affine terrains.

2.3.4 Caves and Subterranean Channels

One particularly difficult modeling problem associated with the effects of hydraulic erosion is procedural synthesis of channels that result from dissolution of carbonate rock, such as limestone, by subterranean water flow. This physical process gives rise to a type of landscape called *karst* and a class of caves called karstic caves. However, hydraulic erosion

in karst has several analogues, such as evacuation of melt water in ice and aeolian erosion in sandstone, which can produce pseudokarstic landforms with similar topology.

This makes modeling of general dissolution caves a broad problem that involves many different morphological sub-types, channels and systems of channels of different sizes, and a variety of behavior exhibited by the channels as they carve the material that surrounds them. Therefore, I distinguish two modeling sub-problems; first, the more abstract problem of simulating the behavior of the channels as they grow and link up according to such parameters as the presence of fractures in the surrounding rock, and, second, the problem of constructing a cave system in all of its detail, such as conduits with different cross-sections, dissolution features on the walls of conduits, and *speleothems*, or structures formed by deposition, such as stalagmites. This division into sub-problems parallels the convention used in some literature to refer only to enterable human-sized passages as caves.

Fractures that affect the permeability of the rock matrix that contains forming underground channels cause them to twist and turn in a manner not unlike that of rivers running above the ground. Furthermore, 3D channels link up into branching networks in a manner similar to the 2D case, although capture of tributaries is a more complex behavior in 3D (Section 2.4.2). One of the modeling approaches proposed by Boggus and Crawfis [6] approximates the branching appearance of 3D channels with a 2D simulation of river-like flow according to the area map concept (Section 2.3.2). As the scheme constrains any constructed channels to lie in a plane, the heightmap representing the floor of the channels is mirrored to create their ceiling.

Complexity of geometric data structures is a major obstacle to simulating more general channels that can form arbitrary loops in 3D and features like multiple levels of passages connected with waterfalls. Another modeling framework by Boggus and Crawfis [7] is based on a specialized data structure called a prismfield, which is a type of generalized heightmap capable of modeling overhangs. However, the framework is intended for sculpting caves manually without the benefit of procedural modeling. The manual modeling problem is perhaps better solved as part of a complete terrain modeling package, as exemplified by the Arches framework developed by Peytavie et al. [47]. Arches incorporates implicit surface modeling tools, so it can be used to create terrains with caves, and it also provides tools for sweeping out passages with complicated cross-sections and appropriately placing collapsed rock, which are necessary for modeling some types of cave passages.

The modeling framework of Cui et al. [11] retains procedural modeling capability, but dispenses with simulation of water flow. The framework represents both cave chambers and speleothems with voxels that are later polygonized into a mesh representation. The voxel-based approach proves to be suitable for creating a plausible cave environment consisting

of a tubular passage with a cross-section derived from a noise function, stalactites, and stalagmites. However, the level of detail is too great to use the method to construct a network of passages, and the method lacks a model for simulating the development of such a network in the first place. The framework of Cui et al. also does not model the growth of speleothems. The growth of stalactites and stalagmites has been simulated by Tortelli and Walter [66] using a simple approach that works directly on the polygonal representation of the speleothems.

I approach modeling of subterranean channels with an emphasis on procedural elements and self-organization, similar to how I model river-like channels (Sections 2.3.2 and 4.3.1). In particular, I use my avalanching modeling paradigm with a specialized water flow simulation to construct channel networks based on emergent behavior of individual channels, whose growth depends on their proximity to other channels. However, my model of the growth of the constituent subterranean channels is different from the 2D case, because of the effects of pressure that can cause water to creep upward along fractures or pores.

The novel part of my model for the 3D channels is combined self-organization of flow and pressure that matches rules of channel development known from hydrogeomorphology (Section 2.4.2). This formulation is sufficient to capture the most important aspects of the phenomenology of channel growth that reflect development history of the constituent channels and produce channel networks with complex topology. However, my model emphasizes channel behavior and does not also make provisions for the development of complicated cross-sections and speleothems. This allows me to implement the model using a uniform voxel discretization in my modeling framework and avoid management of level of detail issues. Furthermore, construction of speleothems may be treated as a separate problem that can be deferred to one of the existing models.

2.4 Hydrogeomorphologic Aspects of Erosion Features

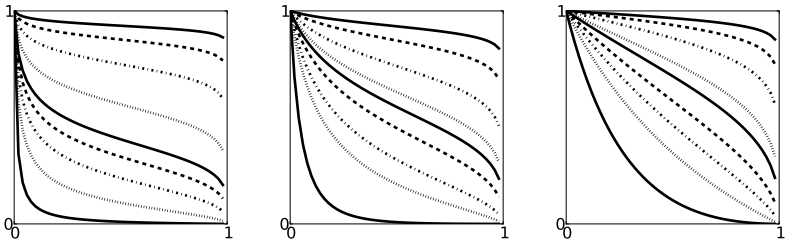
Procedural modeling of natural landforms can utilize a varying degree of insight into the shape of the landforms: from ontogenetic modeling that is based on visual appearance alone to modeling via simulation of relevant physical processes. Information about shape is valuable both for development of models and subsequent comparison of synthetic landforms to real ones. This section discusses two methods of evaluating the aggregate shape of landforms from hydrogeomorphology: hypsometric analysis of terrains modified by water and morphometric analysis of caves. Additionally, this section discusses the phenomenology of cave-like channel formation (i.e., the behavior of the channels that results in their characteristic shape). I use the aggregate shape properties to evaluate the landforms that

I produce with my simulations. The phenomenology of 3D channel formation serves as a guideline for the development of my model of subterranean channels.

2.4.1 Hypsometric (Area-Altitude) Analysis of Terrains

Hypsometric (area-altitude) plots can help understand the effects of erosion on the morphology of a landscape. The abscissa of the coordinate system of the plots is cross-sectional area lying above a given height; the ordinate is the height. Normalization of the values on the plots allows hypsometric curves to be compared without consideration for the dimensions of the terrains they represent. Strahler [62] designed a family of curves that closely correspond to most hypsometric plots obtained from real world data. The following equation produces the model curves using two parameters: the ratio of a to d and the exponent z.

$$f(x) = \left(\frac{d - ((d-a)x + a)}{((d-a)x + a)} \frac{a}{d-a} \right)^z \tag{2.9}$$



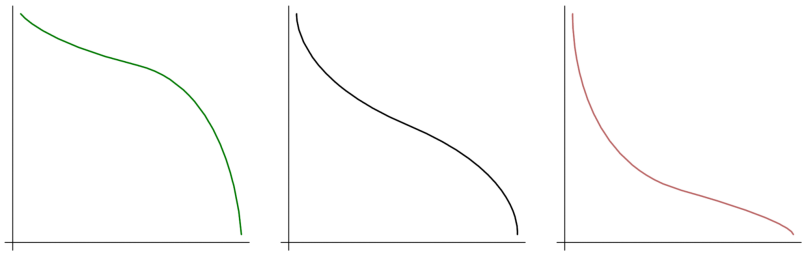
(a) (b) (c)

Figure 2.11: Families of hypsometric curves: (a) $\frac{a}{d} = 0.01$, (b) $\frac{a}{d} = 0.1$, (c) $\frac{a}{d} = 0.5$. Each plot contains curves for several representative values of z.

Figure 2.11 shows some representative hypsometric curves. Qualitatively, the curves tend to have a concavity in the upper part and a convexity in the lower part, although it is possible for a curve to be entirely convex or concave. The curves that transition from convexity to concavity approximately in the middle correspond to terrains that are near the equilibrium stage of erosion. Generally, evolution of terrains involves three broad stages: youthful, equilibrium (or mature), and old age. In terms of hypsometry, the equilibrium and old age stages are indistinguishable except in the situations when isolated rock outcroppings

called monadnocks are present. Therefore Strahler suggests classifying hypsometric forms as inequilibrium, equilibrium, and monadnock.

The inequilibrium category of curves has a larger integral than the equilibrium stage, while the monadnock category has a smaller integral. As monadnocks eventually collapse, normalization of the height and area restores the hypsometric curve to its equilibrium shape, which is roughly symmetric about the center of the curve. Figure 2.12 illustrates the three categories of curves.



(a) inequilibrium (b) equilibrium (c) monadnock

Figure 2.12: Qualitative categories of hypsometric forms.

Hypsometric curves can help evaluate the relationship between area and altitude in synthetic terrains produced with procedural terrain modeling applications. If the method used to model a terrain involves a simulation of hydraulic erosion, then the hypsometric curve of the terrain should qualitatively fit one of the three categories of Figure 2.12. Otherwise, the pattern of erosion on the terrain can not be considered to match a natural archetype. For example, terrains with hypsometric curves such as those shown in Figure 2.13 are unusual and can be considered unacceptable output for an algorithm that is expected to synthesize plausible terrains. I assess the hypsometric character of the terrains that I generate with my simulations in Sections 4.3.2, 4.3.4, and 4.4.4.

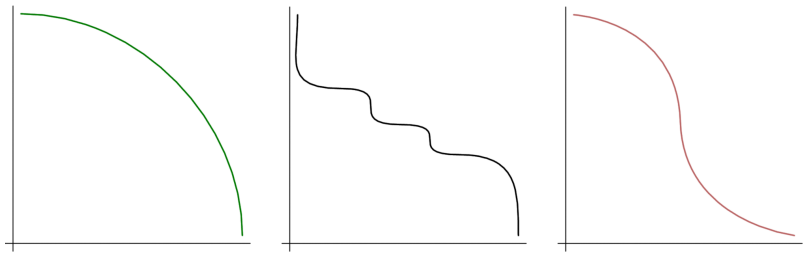


Figure 2.13: Examples of unusual hypsometric forms: (a) completely convex, (b) multiple inflection points, (c) convexity and concavity reversed.

2.4.2 Formation of Subterranean Channels by Dissolution

The scientific description of underground channels according to hydrogeomorphology accounts for many variations in the phenomenology of channel development that have been observed in karstic landforms associated with the presence of channels. These variations can be attributed to the effects of the structure and chemical composition of the rock matrix, as well as the patterns of flow between sources and sinks. From a procedural modeling perspective, not all such variables are of equal importance, as the modeling process tends to emphasize abstraction and generality. Therefore, my discussion of channel formation in this section is based on the text of Ford and Williams [17], but omits a lot of the detail that is not essential to formulating a procedural modeling perspective, such as the chemistry of dissolution. Additionally, I make the simplifying assumption that there is no variation in flow input (e.g., seasonal) and emphasize formation of channels below the watertable.

When water flows through a porous matrix it initially encounters substantial resistance and creates *protochannels*, which consequently may take a long time to grow. The flow inside the protochannels is increasingly impeded, as the pores surrounding each protochannel become filled with water and pressurized consistently with the flow from the protochannels' source. In the isotropic case (Figure 2.14a) the protochannels can be expected to grow radially to produce a wormiform network similar to experiments performed by McDuff et al. [36], who injected acid into carbonate rock and used CT scanning to image the dissolution patterns. However, it is more typical that protochannels develop in a preferred direction due to the presence of fracture planes in the rock, as well as the effects of gravity and pressure (Figure 2.14b).

Eventually, a protochannel may reach a fracture or cavity in the rock where resistance to flow is lower. Following this event, called *breakthrough*, flow inside the protochannel rapidly

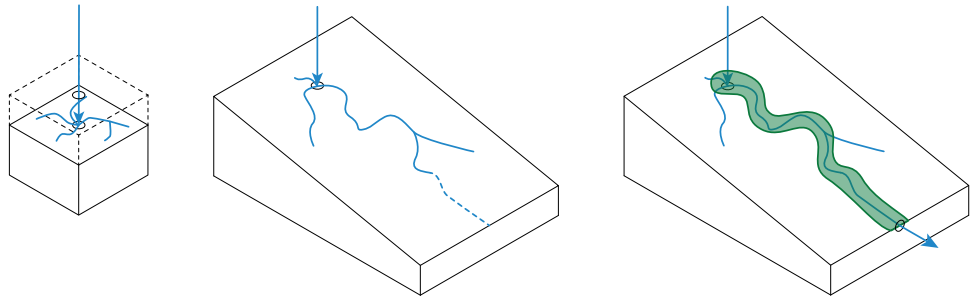


Figure 2.14: Protochannel development: (a) isotropic case, (b) growth, (c) breakthrough and eventual channel.

increases as the protochannel and the surrounding pores drain into the low resistance area. The concomitant erosion enlarges the protochannel, now referred to as a channel, and increases its carrying capacity, which causes more erosion, as shown in Figure 2.14c. If the source and the sink have a large capacity for flow, channel growth following breakthrough can remove much of the rock containing the original protochannels.

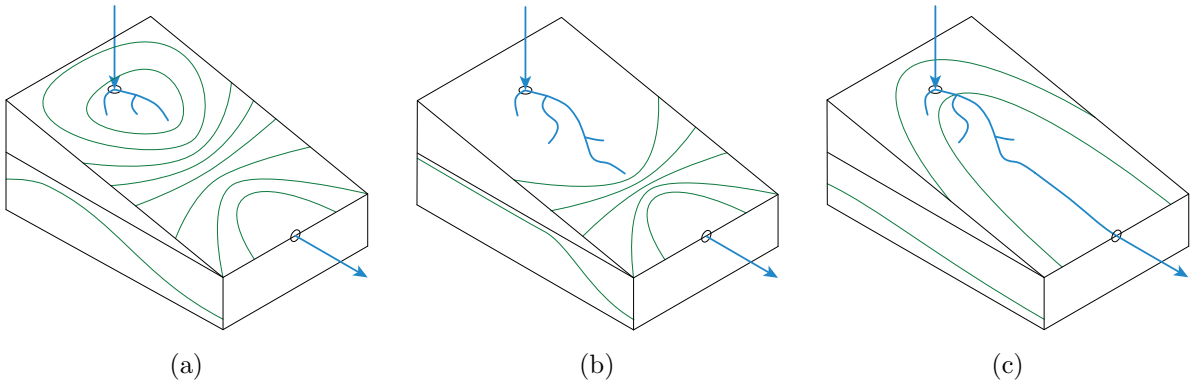
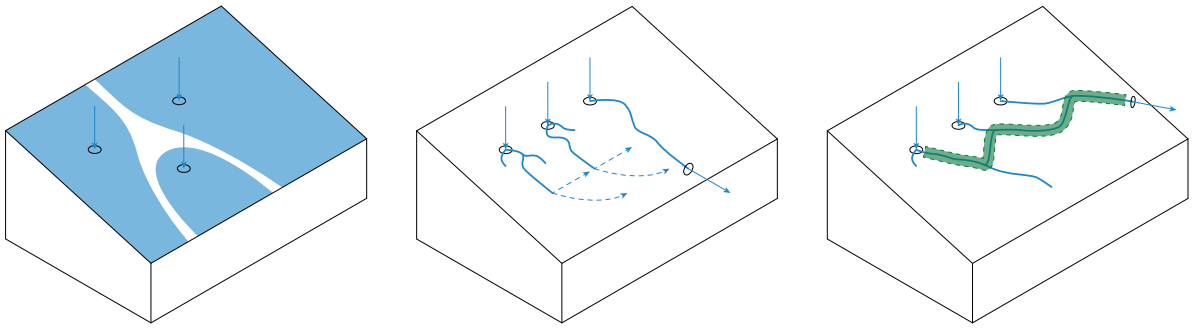


Figure 2.15: Protochannel behavior during breakthrough: (a) pressure grows around source, (b) protochannel nears low pressure zone around sink, (c) pressure drops after breakthrough.

Figure 2.15 illustrates the development of protochannels before and just after breakthrough in more detail. When the rock matrix impedes flow, it creates a differential of pressure between the pressurized region surrounding the source and the low pressure zone

around the sink. The figure shows this using idealized pressure isolines and a cross-section of the idealized pressure surface. As a protochannel breaks through, the pores in its vicinity drain to the sink and the pressure within the surrounding rock drops. However, as long as the channel is saturated, the pressure around the source will continue to be at least slightly larger than around the sink due to drag along the walls of the channel.

The effects of pressure also help to explain the behavior of systems with multiple inputs, as shown in Figure 2.16. Prior to breakthrough, pressurization about each source creates pressure zones that act as obstacles to protochannel development, since protochannels develop towards areas of low pressure. After breakthrough, the breaking through channel creates a pressure gradient around itself that may allow it to capture neighboring channels as tributaries after they redirect themselves according to the gradient. As the effect of decrease in pressure propagates, additional channels may become linked in a sequence. However, it is also possible that the sink's area of low pressure is large enough for several nearby channels to begin growing towards it without any of the channels increasing the extent of the low pressure zone first. This case typically occurs when the permeability of the rock matrix around the sink is high due to the presence of fractures, so that it remains at a pressure close to the sink's, as the low resistance to flow prevents formation of large pressure differentials. Figure 2.16b shows the two possible modes of development.



(a) (b) (c)

Figure 2.16: Interaction between channels: (a) pressure zones, (b) tributary capture, (c) phreatic loops.

In one situation that sometimes occurs in nature the rock contains bedding planes that are inclined from horizontal and additional fractures perpendicular to the bedding planes. Water enters along the bedding planes in an equally-spaced sequence of sources similarly to Figure 2.16(b, c) and initially forms a sequence of approximately parallel protochannels. When a propagating sequence of tributary captures occurs as described above, the channels

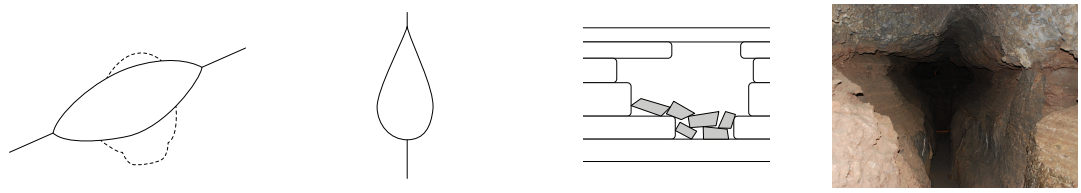
link up in a zigzag pattern that alternates between following bedding planes and fractures perpendicular to them (Figure 2.16c). A single instance of this pattern is called a *phreatic loop*, since it is usually found in the zone of rock below the watertable called the phreatic zone. This shape can also arise between any two nearby channels in the phreatic zone, as one breaks through to the other. Note that pressure causes water to flow up in one side of the zigzags.

More generally, as channels are guided by fractures and permeability of the rock matrix, their appearance can become visually similar to random walks. One of the first studies of cave morphology from the point of view of fractal analysis was undertaken by Laverty [32], who determined that the fractal dimension of several representative cave sections was between 1.0 and 1.5. This range is similar to that for coastline segments (Section 4.4.4) and includes 4/3, the fractal dimension of a self-avoiding 2D random walk.

However, the range does not include 5/3, the fractal dimension of a self-avoiding 3D random walk. There may be several reasons for this: first, procedures for estimating fractal dimension are approximate and rely on survey data, which do not perfectly describe the shape of the cave where multiple channels meet or channel cross-section is irregular. Second, Laverty sampled only five data sets. Third, caves of the type considered by Laverty are likely to be constrained vertically or horizontally. There may be caves of the maze type (not evaluated by Laverty) whose development is closer to a 3D random walk, but they also tend to be formed by water flowing in one general direction, for example by warm water rising due to a hot spring. Furthermore, Laverty's analysis concerns aggregate fractal character, so that the shape of an individual cave component can be arbitrary (e.g., close to a 3D random walk, or without fractal character).

Another aspect of channel formation is how the cross-sectional shape of the channels develops according to the structure of the rock matrix and the flow regime. Figure 2.17 illustrates some of the cross-section archetypes and provides some examples. Phreatic channels are saturated with water flow, so their walls erode nearly uniformly and their cross-section tends to be roughly circular. If the phreatic channel is situated along a bedding plane, or a joint (i.e., vertical fracture), then the shape is closer to an ellipse, or teardrop, respectively (Figure 2.17(a, b)). If a channel exposes multiple bedding planes, its cross-section contains rectangular elements (Figure 2.17c).

A more complicated cross-sectional shape can arise (dashed lines in Figure 2.17a) when a channel begins to develop according to a phreatic flow regime, but later its growth outpaces the amount of water arriving from the source. In this situation, the water flow carves a canyon in the floor of the channel. Reduction of water flow inside large channels can also lead to collapse of the ceiling, especially in stratified rock (Figure 2.17c). Such



(a) (b) (c)

(d) Cave of the Winds, CO

(e) Howe Caverns, NY

(f) Krueger's Cave, MN

(g) Howe Caverns, NY

(h) Mystery Cave, MN

Figure 2.17: Archetypes of cross-sectional shapes of channels: (a) formed at bedding plane, (b) formed at joint, (c) extended past several strata. Examples: (d) – (h).

complications make cross-sections poor descriptors of channel shape for modeling. While it is possible to classify the cross-section of the cave in (d), as a combination of the types in (a) and (c), the cross-sections are not well-defined in (e) and (f): (e) is an extremely wide space between bedding planes and (f) does not have a clear sense of direction, which is necessary to find a defining cross-section. Additionally, the cave passage in (g) demonstrates the complexity of the canyon type, which can sweep out a shape with an extremely variable cross-section.

2.4.3 Morphometric Analysis of Caves

Cave surveys provide most of the information regarding the geometric shape of cave passages. However, the complicated 3D shape of the passages causes many problems during exploration, measurement, and subsequent interpretation of the collected data. In particular, surveying involves taking measurements of cross-sections at survey stations in a process that discretizes the shape of the cave and can fail to capture its salient features, especially in places with complicated topology where the cross-sections are not always well-defined. Furthermore, cross-section measurements are typically limited to specification of a rectangular shape, leading to many ambiguous cases during subsequent reconstruction of

the cave as a digital model. For example, an O-shaped configuration of passages may be a chamber with a column in the middle or two channels that surround an obstacle, resulting in different estimations of length.

	unconfined		confined	
	range	average	range	average
network density $\left(\frac{m}{m^2}\right)$	$0.012 - 0.023$	0.017	$0.049 - 0.406$	0.167
areal coverage (%)	$3.77 - 8.55$	6.17	$13.56 - 58.51$	29.74
porosity (%)	$0.15 - 0.77$	0.40	$1.1 - 12$	5.0
$L : W$	$4676.41 - 218254$	61569.4	$798.06 - 135940$	16417
W (m)	$2.52 - 4.56$	3.77	$0.65 - 4.53$	2.04

Table 2.1: Morphometric analysis of Klimchouk [29] with optimized cave fields.

On the other hand, cave survey data can be used to reliably calculate a variety of aggregate shape measurements, or *morphometric indices*, that are sufficient for many comparisons between caves. Morphometric analysis is also applicable directly to cave plans, which is an advantage as most explored caves do not have digital surveys. Out of the different sets of morphometric indices discussed by Piccini [48] the ones used by Klimchouk [29] and Frumkin and Fischhendler [18] are the best fit for analysis of procedurally-generated caves. I have combined them into the following set.

A_{field}: The area of the cave field, or the polygon surrounding the plan map.

V_{block}: The volume of the cave block, estimated as the product of A_{field} and the vertical range of the cave.

Network density: The ratio of the cave's length to A_{field}.

Areal coverage: The ratio of the cave's area to A_{field}, expressed as a percentage.

Porosity: The ratio of the cave's volume to V_{block}, expressed as a percentage.

W: The average width of the cave's passages, estimated as the ratio of the cave's area to its total length.

$L : W$: The ratio of the cave's length to W.

These morphometric indices have been shown to be effective for making meaningful comparisons between some common cave types. For example, Klimchouk [29] found an

order of magnitude or a factor of 5 difference in network density, areal coverage, and porosity between two classes of caves: confined and unconfined (Table 2.1). Similarly, there are some apparent morphometric differences between the chamber and confined caves in the data of Frumkin and Fischhendler [18] although the authors have chosen to classify caves into types by defining chamber caves to have $L : W < 20$ (Table 2.2).

The indices can be both estimated from a cave plan and calculated more accurately from a 3D model, which is advantageous for two reasons. First, it should be possible to use any existing cave data for comparisons. Second, as length, area, and volume can be calculated more accurately from a 3D model, more accurate analysis can be made with these indices in the future when more digital surveys are made (i.e., the indices are future-proof). As for procedural models of caves, they already contain all the necessary information to determine these indices.

	chamber	confined
network density $\left(\frac{m}{m^2}\right)$	$0.01 - 0.25$	$0.07 - 0.35$
areal coverage $(\%)$	$20 - 80$	$10 - 40$
$L : W$	$2 - \mathbf{20}$	$\mathbf{20} - 4000$
$W\ (m)$	$2 - 100$	$0.9 - 8.0$

Table 2.2: Morphometric analysis of Frumkin and Fischhendler [18] with rectangular cave fields.

Klimchouk [29] notes that a passage that branches away from the major part of a cave is likely to increase A_{field} disproportionately, causing indices dependent on A_{field} to be significantly underestimated. So, instead of a rectangular area, he uses the area of a polygon that "reasonably closely embraces the plan of the cave" and argues that the resulting values are better for making comparisons despite some subjectivity in the definition of the bounding polygon. In the future, accuracy of comparisons can be likely improved if a standardized procedure for optimizing the shape of the cave field can be established. Until that time I believe that the best option is to compute all of the indices with both rectangular and optimized fields.

2.5 Multi-Agent Systems

The concept of an intelligent agent has different meanings depending on the area of application, such as artificial intelligence, distributed systems, and computer graphics. For

clarity, I give my own definition that fits the application of multi-agent systems (MAS) to the modeling that I am considering and the appropriate background literature.

An *agent* is a computational primitive that encapsulates the local state associated with a distinct position in a simulation space. The agent can make decisions based on that state and change itself or its neighborhood. Agents can communicate directly, particularly if they are neighbors in the simulation space, or indirectly through artifacts in the environment. A *system* of agents is a collection of agents designed for solving a particular problem via simulation of actions taken by the agents. The problem space can be separate from the simulation space, in which case a translation step is needed to turn the structure that the agents have formed in the simulation space into a solution of the concrete problem.

This kind of MAS resembles particle systems in computer graphics, but can exhibit more complicated behavior due to the additional interactions between the agents and the accumulation of state in their configurations and environment. The additional complexity of MAS allows for emergent behavior and makes it easier to encode rules needed for procedural modeling of geometry. However, the full power of expression may not be necessary in all cases, as demonstrated by one of the earliest examples of using a MAS for procedural modeling: a framework for modeling plants as particle system trails designed by Reeves [53].

Whether agent systems can serve as models of complex systems depends on the agents' interactive ability. In my definition of MAS, agents can interact with each other and their environment in simulation space according to the spatial position of each agent and the environment artifacts found in the agent's neighborhood. Benenson and Torrens [5] define a similar system of "geographic automata" with additional provisions that make it more convenient to model agent mobility. Interactions between the automata can successfully represent several types of behavior of humans and infrastructure entities in urban systems, including phenomena resulting from adaptation to spatial constraints.

Schlechtweg et al. [60] describe a MAS called RenderBots that is capable of generating images in a bottom-up way by composing elements such as stipples, hatches, and mosaic pieces. In this system the simulation space is an input image augmented with additional information, such as area masks and coordination artifacts for the agents. The simulation causes the agents to move and place the graphic elements that form the resulting image. Jones and Saeed [26] use MAS in a similar way in an image enhancement system called PixieDust. One difference between the two frameworks is that PixieDust includes provisions for modifying the underlying image based on the global features that emerge during the simulation.

Despite the local formulation of agent behavior encoded by a MAS, the global features

that emerge can contain significant information about the entire simulation space. For example, Lemmens et al. have used MAS to implement bee- and ant-based navigation algorithms [33]. In this case, the agent space is the environment to be navigated and the result of the simulation is a set of routes through that environment. The way in which the paths emerge is a primary example of *stigmergy*, or self-organization of agents using artifacts.

Mason et al. [35] propose the coalition paradigm to match configurations of agents in simulation space with compositions of features in the problem space. Coalitions are hierarchical groups of agents that contain a sufficient amount of information to construct an object in the problem space. By joining and leaving these groups, coalition-forming agents can synthesize artistic compositions of pre-defined elements.

My approach for modeling erosion features utilizes a MAS to provide the common framework for implementing different kinds of erosion simulations that rely on self-organization. Since the main goal of the simulations is to produce geometry that exhibits erosion features, the MAS in my system is specialized for working with that geometry as a concrete simulation space. In my 2D simulations of erosion on terrains, the terrain geometry is a grid-like graph whose nodes are vertices and edges correspond to triangle edges that connect the vertices. In this case, the agents are stored in the nodes of the graph and find their neighbors using the graph's edges. The graph's nodes alternate between having degrees four and eight, as in 4-8 subdivision (Section 2.1.5). In my 3D simulations, the geometry is a regular grid of voxels, which is interpreted as a graph whose nodes are the voxels and the edges are defined implicitly between each voxel and 26 voxels that are adjacent to it on the grid. In this case, the agents are stored in the voxels and use the voxels' indices on the grid to recover their neighbors.

Communication between agents in my MAS is similar to the other frameworks I have discussed in this section and can occur between agents at the same location (i.e., vertex or voxel), between neighbors, and between any two agents that can reach each other by searching the underlying graph. I have used both direct communication, in which one agent changes the state of another, and communication based on artifacts, which I have found to be convenient from an engineering point of view even though it is not fundamentally different from direct communication. The agents of my MAS can join different types of groups and define the properties of a geometric feature similar to coalition-forming agents, but do not make an explicit use of the coalition paradigm. Having done a case study of coalition-forming agents [50], I believe that coalition rules are redundant for controlling agents that already behave according to a physical model.

From a computational perspective, I have implemented a typical self-organized sim-

ulation of erosion by using the MAS framework to encode a set of algorithms that can be either combinatoric in nature (e.g., marking nodes of the problem domain graph for participation in simulation) or related to iteratively solving a discretized PDE. A group of agents can encapsulate either type of algorithm by using its constituent agents to store appropriate values and update them according to a common procedure. Note that the mapping between agent types and algorithms does not have to be one-to-one, especially if agents that play the role of communication artifacts are required. Nevertheless, the relationship between a group of agents and a procedure that modifies the underlying geometry makes it is unnecessary for more than one agent of each type to exist at a given location, which I assume as a simplification. The largest number of agent groups I have used is five in my terrain simulation that combines several algorithms involving coastlines, terrain level sets, water transport, and erosion (Section 4.3.4). The power of MAS to express diverse algorithms and encapsulate them so that they can be combined in different ways are key aspects of my MAS framework that make it particularly suitable for my method of procedural modeling. I discuss implementation of the framework further in Sections 3.2.2 and 3.2.3.

It's easy to invent unnecessarily
complex solutions.

Frans Kaashoek

Chapter 3

Simulation Framework

My modeling framework provides a general approach to constructing characteristic features of different kinds of erosion. The procedural nature of my approach involves several modeling sub-problems: development of conceptual or physically-based models of systems undergoing erosion, development of algorithms for constructing erosion features through simulation of the systems' behavior, and fitting the models to my simulation framework. I have observed that fractal character appropriate for erosion features can arise in an emergent way through simulations based on self-organization principles. Therefore, I have used self-organization to unify physical models of erosion and simulations in my modeling approach and developed avalanching, a pattern of emergent behavior, into a common modeling paradigm.

In the first part of the chapter, I discuss some aspects of fractal character appropriate for erosion features and the motivation for avalanching as the basis for my modeling framework. In the second part, I detail the infrastructure of my framework, which is based on a multi-agent system (MAS).

3.1 Motivation

Although the shapes of natural objects have a tendency to contain fractal character, there are numerous issues with using explicit fractal synthesis as the foundation of a procedural modeling approach. In particular, it may be a challenging task to determine how fractal features combine in one object and develop an appropriate modeling process that is also general enough to be applicable to other types of objects. Therefore, I develop a different concept, avalanching, as the foundation of my modeling approach. Avalanching produces fractal character in an emergent manner (i.e., implicitly) and leads to a modeling process that is based on simulating physical erosion processes in a common way.

3.1.1 Fractal Character

The geometric shape of real-world eroded landforms can take on various fractal characteristics. For example, it is possible to model a geographic feature as a fractal and estimate its fractal dimension using such an approach as the Richardson plot procedure for coastlines (discussed in Section 2.1.1). It is also possible to broaden the scope of the analysis and determine whether the landform exhibits a scaling relationship that fits the behavior of self-affine fractals (defined in Section 2.1.2).

However, there is an underlying problem with using fractal synthesis as a modeling approach: while real world objects may share some characteristics of their shape with certain types of fractals, it may not be the case that a description in terms of fractals captures their shape completely. This approach can especially suffer if it only depends on a single parameter like fractal dimension. Furthermore, fractal synthesis does not offer a common algorithm that can serve as a foundation for the type of modeling that I am considering, which can involve terrains with such diverse features as river channels and dunes. An attempt to make it suitably general can lead into the trap of using a superposition of human-recognized features in place of proper fractal scaling, which is something Mandelbrot cautions against doing [34].

Nevertheless, fractal analysis informs the design of my framework by requiring that the shape of the produced objects exhibits fractal scaling and respects scaling issues discussed below. In particular, the shape of many natural objects contains fractal noise whose spectral density varies as $\frac{1}{f^\beta}$ with $\frac{1}{f}$ noise being especially common [68]. So, even though my framework does not explicitly use fractal synthesis as the means of modeling, one of its aims is to produce objects that exhibit this type of noise. In this way my approach is similar to a general way of modeling with fractals called ontogenetic modeling (Section 2.1.5).

3.1.2 Scaling Issues

Generating detail at different levels of scale is a central concern when modeling fractal-like objects. First of all, natural objects exhibit fractal-like properties only at a range of scales. For example, the shape of coastlines is considered to be a self-similar fractal, but it is unclear what the shape is like on the scale of meters or centimeters. In other words, there is an intrinsic limitation on the smallest scales that can be considered to obey fractal theory.

However, for most modeling problems this limitation can be overlooked, as long as modeling detail on that range of scales is not part of the problem. Indeed, this lower limit on scale size parallels the limits of discretization density in modeling techniques that represent objects discretely. For this reason it is acceptable to consider an object to be a fractal if it displays fractal scaling on as few as three separate scales [38].

The situation can become more complicated when the range of scales is limited in another way. Some natural objects exhibit different kinds of fractal properties over different scale ranges. The scale at which a shift in behavior occurs is referred to as the *cross-over scale*. If the modeling problem involves a range of scales that includes a cross-over scale, then it is necessary to model the two *regimes* of behavior on each side of the scale.

The second issue is that different fractal-like behavior can coexist on the same scale by being spatially distributed, i.e., located on different parts of an object. For example, a coastline may be rougher (which is a sign of a higher fractal dimension) on one side of an island, than on another. One of the challenges in modeling such an object is that it is necessary to model the two types of behavior separately. It would be a mistake to start with a coarse shape where the two types of coastlines "diverged" and then apply the same differential changes to the entire island. This would result in an island with (statistically) the same properties along the entire length of its coastline, which could be verified by comparing any two stretches of the coastline at a scale much smaller than that of the starting coarse shape.

There exist many techniques for creating objects that exhibit fractal-like features over a range of scales. Some of them explicitly generate detail at different scales and combine them, but doing this tends to produce the *creasing* artifact. Creasing is most often associated with techniques that model each level of detail separately, resulting in a *multi-level pyramid* of features, and then combine the levels into a single object. What causes the artifact to appear is that the combining can only be done in a differential way: the finer levels are offset from the coarser levels. This introduces statistical dependences between the finer features, simply because they are combined with the same (or bordering) coarser

features. The statistical dependence typically manifests itself visually as sharp creases, which can be made less apparent by using sophisticated ways of combining the detail levels (see also Section 2.1.5).

Instead of a hierarchical top-down approach, some modeling problems can be solved in a bottom-up way. In particular, algorithms that simulate emergent behavior can produce fractal character and avoid creasing artifacts, at least in principle. A practical modeling problem solved using self-organized simulations is likely to require an amount of top-down control in the form of initial conditions that can be set according to some coarse shape. Although such constraints can introduce creasing, they can also serve as a natural way to combine different kinds of fractal behavior on the same object.

3.1.3 Relaxed Self-Organized Criticality and Avalanching

Many existing models of erosion demonstrate a common pattern of emergent behavior that involves erosion features becoming more pronounced over time. This is particularly true of the models that have an explicit link to SOC and its avalanching property (Section 2.2). Some examples of these models include sandpiles of Duran [14] (Section 2.2), coastlines of Sapoval et al. [58] (Section 2.3.3), and river networks of Rodríguez-Iturbe and Rinaldo [55] (Section 2.3.2). However, applications of SOC can not include modeling of terrains as a whole, because they can have self-affine features, while the fractal character of SOC is self-similar. Models of erosion such as the one of Šťava et al. [69] can be more flexible when it comes to fractal scaling, but they do not have an explicit link to avalanching.

In my erosion simulations I use avalanching with an additional parameter, which is a radius that represents a notion of scale. In the case of erosion on a terrain, an event that erodes a feature lying in the distinguished height direction also promotes further erosion of features situated within the neighborhood defined in the domain by the avalanching radius with the domain directions treated isotropically. The effect of my version of avalanching is to change the scaling of features in the radius from self-similar to self-affine, which makes it possible to extend self-organization principles of SOC to modeling erosion-related features of terrains. I give this modeling paradigm the following separate definition.

Definition 8. *Avalanching* (as a modeling paradigm) is an emergent behavior of state changing events that results from one event explicitly increasing the likelihood of a similar event occurring within a neighborhood of the original event.

In Figure 3.1 I provide two examples of relaxed-SOC erosion models that use avalanching to create approximately self-affine features. In the first model, oversteepening of local

slope causes material slippage, which can cause additional slippage at neighboring sites. I based this model on the behavior of sand in the model of Duran. In the second model, exposure of material causes it to corrode and disintegrate, exposing more material. I based this model on coastline erosion of Sapoval et al. As Figure 3.1 (a, c) illustrates, the sand-like model smoothes uniformly random initial conditions into a hilly terrain, while the corrosion-like model causes a flat initial surface to develop pits and cracks.

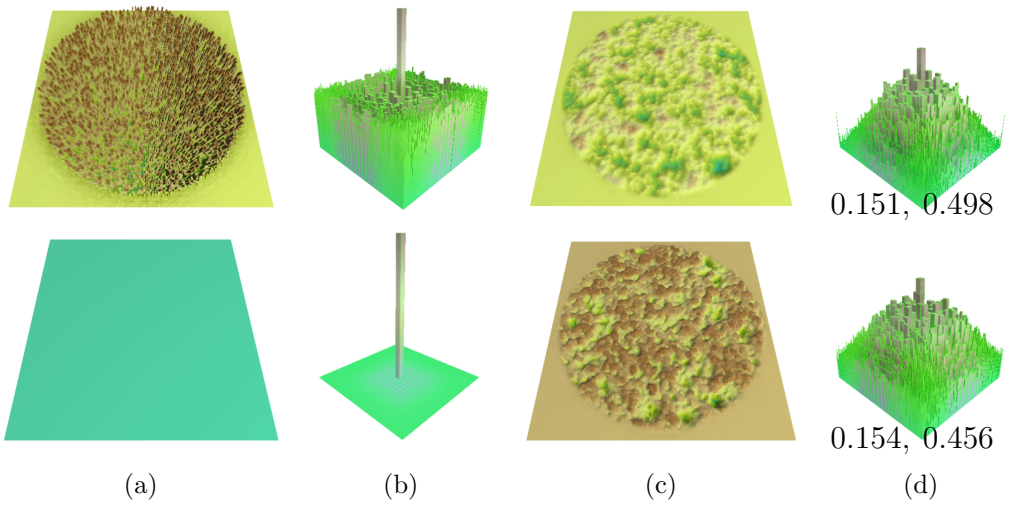


Figure 3.1: Results of a sand-like (top row) and corrosion-like (bottom row) relaxed-SOC simulations: (a) periodized initial conditions, (b) log plot of spectral density, (c) periodized result, (d) final spectral density.

The corresponding spectral density plots in Figure 3.1 (b, d) demonstrate that in both cases the shape of the resulting object exhibits approximately self-affine fractal scaling. The non-uniform scaling associated with self-affine behavior is due to the separation of the height direction from the isotropic domain neighborhood in my formulation of avalanching. The same kind of separation is responsible for self-affine terrains having self-similar coastlines (Section 2.1.3).

What is especially significant about Figure 3.1 (b, d) is that self-affine features arise despite the fact that the spectral density of one set of initial conditions is white noise (all possible frequencies are present equally) and the other set of initial conditions is flat (no frequencies are present). (Note that the central datum on the spectral density plots corresponds to the average height of the initial terrain.) Therefore, my avalanching paradigm can introduce a notion of scale into a system, both appropriately reshaping its spectral den-

sity and generating missing spectral frequencies. This independence from initial conditions is similar to the same property of SOC.

3.2 High-Level Design of the Framework

In addition to avalanching, which acts as its conceptual foundation, my generalized framework for modeling of landforms affected by erosion possesses two other main aspects. First, the framework has a model of computation that it uses to express simulations of different erosion processes. Second, the framework provides each simulation with a representation of landform geometry as a graph that encodes a 2D or voxel grid. This section discusses the development of these aspects of my framework and provides a list of features that are necessary for a complete implementation. Chapters 4 and 5 contain simulation examples, which illustrate the functionality of the framework concretely.

3.2.1 Towards a New Modeling Framework

Terrains are composed of geomorphologic features that can be formed through a complex combination of processes related to the action of tectonic plates, volcanoes, and glaciers, as well as different types of erosion. To model terrains at a reasonable cost procedural modeling frameworks tend to adopt a simplified view of landform evolution. Musgrave's terrain modeling framework [39] mentioned in Section 2.3.2 is an archetype of this approach. It generates a multifractal terrain and then erodes it using simulations of hydraulic and thermal erosion. From the perspective of ontogenetic modeling (Section 2.1.5) the initial conditions for the erosion simulation stand in for a combined result of tectonic processes, while the subsequent erosion creates realistic erosion features that are difficult to produce with fractal synthesis. From the perspective of physically-based modeling, the two-step model is over-simplified, because only the erosion step involves a model of a physical system. However, the first step may have a connection to Mandelbrot's explanation of fBm as a result of tectonic faulting (Section 2.1.3).

In my framework I simplify the terrain modeling problem in a new way by using a common approach based on self-organization to model different types of erosion. My idea is to express existing models of erosion in terms of avalanching (discussed in the previous section) and leverage that generalization to reduce the effort required to design individual simulations for creating features like coastlines and rivers. One of the challenges for the generalization comes from the disparate ways adopted by the existing modeling methods

for representing the features they model (Section 2.3). For instance, Sapoval et al. [58] represent their coastlines in an implicit way as the boundary between land and water squares on a grid, while Rodríguez-Iturbe and Rinaldo [55] represent their river networks as graphs.

A second challenge for the development of my framework is that the existing models tend to have an abstract view of the features they generate, because they aim to demonstrate certain physical phenomena instead. For example, in the coastline model of Sapoval et al. the shape of the coastline is less significant than its evolution into a shape that can resist erosion. Similarly, the width of rivers is disregarded by the river model of Rodríguez-Iturbe and Rinaldo, as it is concerned with the emergent properties of the river network as a whole. In respect to the design of my framework, any physically-based behavior inherited from these models is an advantage, but I have made extensions to the models to make the produced features more appropriate for computer graphics applications.

To summarize, the main goal of my framework is generalization of the modeling process for erosion-related features of terrains, using physically-based self-organization principles. As illustrations of using the common framework, I apply it to modeling of dunes, coasts, and rivers (Chapter 4). Additionally, I use my framework to design a two-part terrain modeling method similar to Musgrave's: the first part uses self-organization to create initial conditions for the second part that applies hydraulic erosion. This terrain modeling algorithm is an illustration of using only self-organization to model terrains (Section 4.3.4). Finally, I apply my framework to modeling of 3D cave-like passages, which is a modeling problem with many aspects that have not been previously considered in the realm of procedural modeling (Section 5).

3.2.2 Agent-Guided Procedural Modeling

In my simulation framework procedural modeling is *agent-guided*, i.e., carried out with the help of a specialized MAS (Section 2.5). In general, MAS can benefit procedural modeling tasks by making it easier to manipulate geometric features through an abstraction for defining their extents (boundaries, areas, or volumes). For example, the presence of a particular agent at a location on a mesh can select that site for participation in an algorithm for constructing a hill. In this way a MAS can encode rules for controlling the distribution of constructed features, as well as the relationships between them. Furthermore, the discrete agents of MAS map to controlled features naturally, as they are also discretized within the problem space.

This aspect of MAS plays a central role in the design of my modeling framework, which

executes erosion simulations on discretized domains, which are graphs representing terrain meshes in 2D and grids of voxels in 3D. Discretization in terms of agents is especially straightforward with some simulations, such as in the case of coastline erosion. In the coastline simulation the agents can mark the location of the coast, making it unnecessary to operate on the rest of the grid. Implementing simulations in terms of agents and their behavior also has the added benefit of standardizing interoperation of multiple algorithms, such as between simulations of different types of erosion on terrain.

As an alternative to algorithms of a combinatoric nature, a MAS can also express an iterative scheme for solving a PDE using the finite difference method. In this case it is necessary to instantiate an agent at each location inside a region in the problem domain that corresponds to the domain of the PDE. For example, I use a PDE formulation for the amount of erosion in my simulation of terrains undergoing hydraulic erosion due to channel networks. The ability of MAS to encapsulate and combine different types of computation is essential for my simulation-based approach to modeling, because of its procedural nature.

3.2.3 Infrastructure for Self-Organized Simulations

My modeling framework has two main components: a graph that represents a grid-based discretization of a given problem setting and a facility for using that graph as a simulation space for agent systems with customizable behavior. Discrete representations not based on grids, such as triangulated irregular networks, or TINs, are also compatible with MAS-based modeling, but I have chosen regular grids, since grids make it easier to solve PDEs numerically and uniform sampling is necessary for my spectral analysis procedure (Section 2.1.6). Furthermore, while the size of each triangle in a TIN may be optimized to represent the initial conditions of an erosion algorithm, the TIN is not likely to be optimal for the erosion features that the algorithm creates, necessitating frequent remeshing. However, I treat the grid discretization as a graph similar to how TINs are commonly represented, so that the vertices of the graph can correspond to the vertices of a mesh representation of a terrain or the voxels of a 3D volume representing porous rock.

The graph encodes connectivity using two directed edges for each pair of neighbors. In the 2D case, the edges correspond to the edges of triangles in the terrain mesh and are stored explicitly as part of the mesh data structure. In the 3D case, the edges of a voxel correspond to each of the 26 adjacent voxels and are not stored explicitly, as they can be efficiently recovered from the indices of each voxel (e.g., there are two directed edges between voxels $(0, 0, 1)$ and $(0, 0, 2)$). In certain simulations, it becomes necessary to augment graph edges with additional information such as the amount of water flowing

between a pair of adjacent nodes. I accomplish this by creating agents that store the extra data.

Agent-based simulations can use the vertices of the graph as a set of locations where the agents can situate themselves. Agents of the same type form groups that can store global information, which controls their members' behavior. For example, the global state can contain a parameter that determines how far each agent should search when it performs breadth-first search. The agents can also use the global storage to calculate the maximum value of a set of values that they store locally. I have found that it is sufficient to have at most one member of a group at each node, because each agent type corresponds (in a many-to-one way) to a particular computation that is to be performed at the node once for each iteration of a simulation. So, several types of agents can participate in a computation task, but there is no need to perform the task multiple times.

To facilitate design of specific erosion simulations that use self-organization in a common way, I have distinguished the following set of features as an infrastructure that all of the simulations can rely on.

Agent group membership: Agent groups keep agents with different behaviors separate and provide for sharing of global variables within a group, as well as inter-group communication.

Registration: Agents can register and de-register at each vertex. It is convenient to constrain an agent to be able to join only those locations where there are no agents from the same group already present. Agent motion through simulation space consists of registration and de-registration actions.

Agent lifetime: Existing agents can control where new agents enter the simulation. A parity mechanism protects newly created agents from acting on the iteration when they are created. Agents can also delete themselves from the simulation.

Communication: Agents can communicate directly by accessing other agents (of any type) by location, or indirectly by placing communication artifacts, which are types of agents that do not perform any actions.

Search primitives: To change or collect information about its environment, as well as to communicate with other agents, an agent can perform one of several search algorithms on the underlying graph. In the following list, the last two search primitives can apply a custom filter to the edges of the underlying graph. For example, an agent can execute a search that visits only uphill locations within a specified radius.

- neighbor search
- search based on a distance map computed via fast marching
- search along a ray with a given direction
- BFS (breadth-first search)

Quadratic fitting: Agents can obtain local first and second derivatives of a terrain by using a quadratic fitting procedure based on the method of Feddema and Little [15].

Sand transport: Agents can remove and add discretized slabs of sand to a terrain, potentially causing slippage.

Water-column algorithm: Agents have access to several versions of the water-column algorithm for simulating water flow.

Fast marching: An alternative iteration scheme for agents that is based on numeric approximation of front evolution. The scheme can be used to compute a distance map from several distinguished sites or structure neighbor-to-neighbor updates in general (see Appendix A.5, Section 5.4.1, and Section 5.2).

In any natural problem the actual conditions are of extreme complexity and the first step is to select those which have an essential influence on the result.

The Internal Constitution of the Stars
Arthur Stanley Eddington

Chapter 4

Modeling with Self-Organization and Avalanching

In this chapter, I present a number of self-organized simulations for modeling various landforms with erosion features, such as dune fields and river basins. I discuss the use of my avalanching paradigm in those modeling contexts and its advantages. I also summarize how I have implemented these simulations using my framework to illustrate how a common framework is useful for this entire class of simulations. In particular, I describe my method of avalanching-based terrain modeling and evaluate the terrains that I create using it.

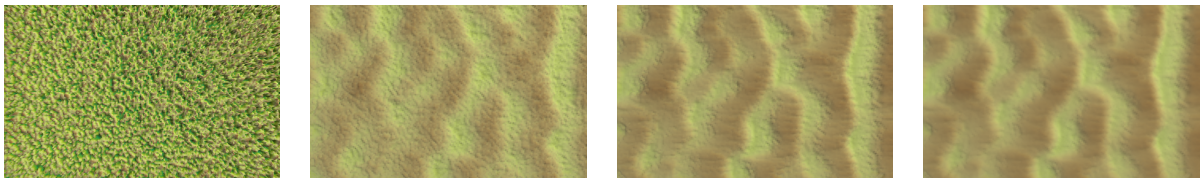
4.1 Dunes

My simulation of dunes is based on Werner's model and introduces my approach to procedural modeling using self-organization. By extending the model to use avalanching in

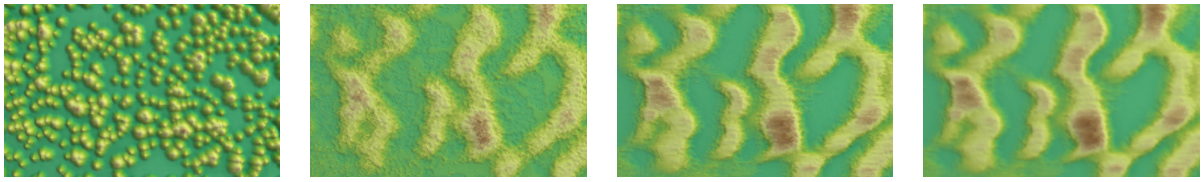
place of a geometric visibility test, I show that avalanching can serve as a foundation for simulations of other types of erosion.

4.1.1 Werner Dunes

Werner's model [71] for the development of dunes uses self-organization and is expressive as a modeling method, since it can produce different kinds of dunes easily. Section 2.3.1 contains a summary of the original model. I have implemented my own version of this model with two extensions. Figure 4.1 shows the results for two sets of initial conditions.



(a) First set of initial conditions (IC1)



(b) Second set of initial conditions (IC2)

Figure 4.1: Werner dune model: resulting dunes for two sets of initial conditions, colored based on height; each row contains the initial conditions, followed by dunes with a coarse slab size, post-processed results with a varying slab size, and the final result with extra smoothing (left to right).

The first modification concerns the discretization of the slabs of sand in the model. I believe that it is desirable for simulations to get a lot of work done in a small number of iterations. That is why my version of the dune simulation uses a comparatively large slab size at the start. However, I follow the main run with an additional phase that uses a diminishing slab size. Specifically, the extra phase consists of two iterations with half the original size, four with the quarter of the size, and so on, up to 2^k iterations with $\frac{1}{2^k}$ of the original size (for a small fixed k).

The second modification adds a final smoothing pass also consisting of a fixed number of iterations. I have added this pass because the discretization of the wind direction in the

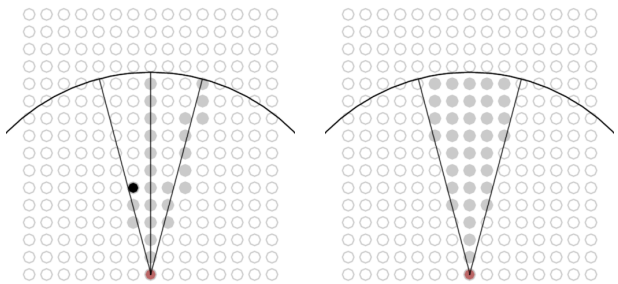
model results in noticeable grooves and smoothing hides this artifact. The smoothing in this pass uses a simple mass diffusion equation of the following form

$$\frac{\partial h(x,y)}{\partial t} = K \cdot \Delta h, \tag{4.1}$$

where $h(x,y)$ is the height at a given location (x,y), t corresponds to a discretized update step, K is a constant, and Δ stands for the Laplace operator. The idea for using such an equation with this type of dune model comes from Onoue and Nishita [42] who use a model similar to Werner's.

4.1.2 Dunes with Avalanching

My alternative model of dune development operates in a similar way to the extended Werner model from Section 4.1.1, except that it replaces the concept of shadowing with a different one. In the original model, transport of sand can not start at a location that is physically shadowed from wind by a higher sandpile along the ray in the upwind direction. This type of shadowing has two main disadvantages.



(a) (b)

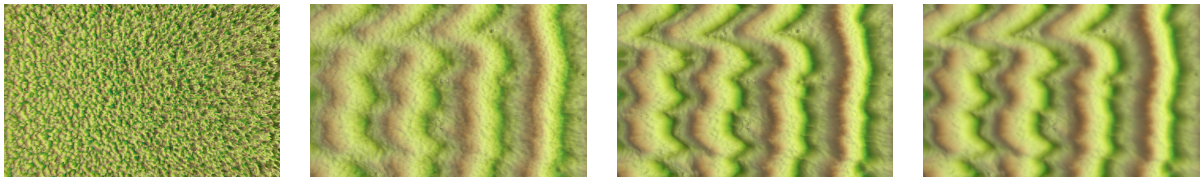
Figure 4.2: Shadowing and protection principles: (a) shadowing effect due to one site for three wind directions can be interrupted by an intervening location, (b) protection effect for a comparable wind regime without discrete wind directions.

First, it allows the discrete nature of the wind to have a bigger effect on the entire model. Since transport occurs in a direction parallel to the wind, the shape of the generated dunes contains grooves parallel to that direction. (Using more than one direction does not sufficiently hide the artifact, especially if the number of wind directions is not large, which

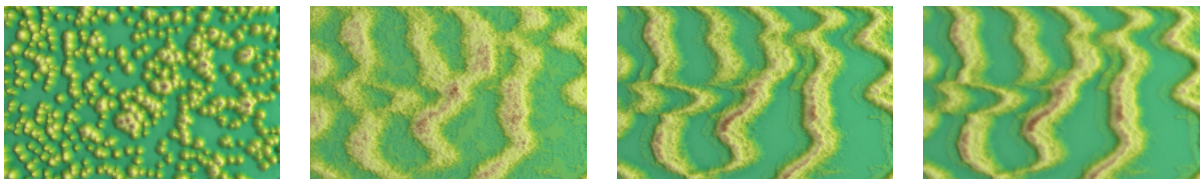
is a performance constraint.) The effect of shadowing is to eliminate sand transport next to some of these grooves, possibly making them even more pronounced.

Second, this type of shadowing leads to weak self-organization as the simulation runs. A dependency exists only between locations on a line and transport events to or from a given site may not affect any downwind locations at all, if there is an intervening bump of sufficient height. In other words, one transport event has a poor chance of having an avalanche effect. Dunes resulting from fewer transport events grow slower and appear to be flatter after a fixed number of iterations.

I propose a different avalanching mechanism to be used in the context of this problem. Instead of shadowing along a line, a specific site can contribute to the protection of any site located in a cone downwind of it. The dimensions of the cone of protection correspond to the wind regime. Protection of downwind locations is reduced by erosion and increased by deposition. The effect of protection is to prevent transport of sand by wind from downwind locations similar to shadowing based on the geometric test. The difference between the concepts of shadowing and protection is further illustrated in Figure 4.2.



(a) First set of initial conditions (IC1)



(b) Second set of initial conditions (IC2)

Figure 4.3: Dune model with more avalanching: resulting dunes for two sets of initial conditions, colored based on height; each row contains the initial conditions, followed by dunes with a coarse slab size, post-processed results with a varying slab size, and the final result with extra smoothing (left to right).

Figure 4.3 shows dunes created with the protection principle. In both types of simulations the maximum slope between a site and its neighbor is limited by sand's slope of repose (taken to be 33.0° for dry sand [31]). So one way to compare the shape of the new dunes with those produced by the old model is to count the number of sites with slope

within 10% of the slope of repose. I determined that the 10% range was necessary by trial and error: the error was too large for a smaller range and too many data were rejected for a larger range. As shown in Table 4.1, dunes based on the protection principle tend to be significantly steeper than those based on shadowing.

The most significant feature of protection-based dunes is that their shadow zones are determined in an emergent manner without the aid of a geometric visibility test. Table 4.1 also provides a comparison of the shadow zones in terms of the percentage of sites where the result of the shadow test based on the protection principle disagrees with the geometric test. The two shadowing principles can not agree exactly because the protection method considers a range of wind directions, while the shadowing method only uses three wind directions. Additionally, there is a substantial systematic error in the protection method, as it considers the peaks and troughs of the dunes to be in shadow. Following removal of sites on the peaks and troughs from consideration, % error values indicate that computation of emergent shadow zones based on avalanching can replace the geometric shadowing test for modeling purposes.

	% near repose
IC1	19.7
IC2	3.4

(a)

	% near repose	% error	% error without peaks/troughs	% error reduction
IC1	50.8	36.5	10.1	72.4
IC2	10.2	20.8	1.7	91.9

(b)

Table 4.1: Comparison of simulated dunes: (a) shadowing-based and (b) protection-based. Tables show % of sites within 10% of slope of repose and % error in protection-based shadow zones viewed as an approximation of geometric visibility. The sites in the peaks and troughs of the dunes are the largest source of disagreement.

The role played by avalanching in the two sets of dune simulations illustrates two key aspects of my approach to procedural modeling. First, I rely on avalanching to create complicated shapes from random initial conditions similar to the simulations in this section. The second set of simulations makes it especially clear that the resulting shape of dunes is exclusively due to avalanching, because I have formulated the new simulations to emphasize avalanching more and avoid using any other computational primitive such as a geometric visibility test. Second, in my method of procedural modeling I view avalanching as a general-purpose modeling tool and use it to achieve multiple goals such as creating initial terrains and also eroding them, which is similar to how I have used avalanching to replace the visibility test in this section.

4.1.3 Use of the Common Modeling Framework

The implementations of both of the dune simulations map naturally to the functionality of the common framework that I have developed. Each location in the problem domain contains an agent, which is responsible for the addition and deletion of sand at the site. The agents carry out the details of the erosion process by using the following two primitives.

Line search primitive: serves for finding upwind shadowing locations and the destination of each transport trajectory.

BFS search primitive: serves for maintaining the protection state during erosion and deposition; the visited locations are limited to a cone as appropriate.

The framework also calculates second partial derivatives for each domain location by locally fitting a quadratic surface. The dune simulations use these derivatives to implement the smoothing process described by Equation 4.1.

k	3
K	5.0
iter. of Eq. 4.1	10
sim. iterations	20
slab size	5.0
wind direction	$\pm\frac{\pi}{12}$
grid size	256×128

(a)

principle	wind strength	R_s	IC	time (s)
shadowing	140.0	140.0	IC1	41.6
			IC2	40.4
protection	80.0	120.0	IC1	67.3
			IC2	55.7

(b)

Table 4.2: Simulation parameters for dunes in Figures 4.1 and 4.3: (a) common parameters, (b) timing.

Table 4.2 lists simulation parameter values and timing numbers. Wind strength determines the distance that wind transports each slab of sand. In shadowing-based simulations, the search radius R_s determines how far each site searches in the upwind direction to test whether it is in shadow. In protection-based simulations, R_s determines the extent of the cone of protection in the downwind direction.

4.2 Coastlines

In this section I develop a coastline erosion model that simulates erosion events that take place at the land-ocean interface and cause the coastline to cave in. In my implementation

I use a discrete grid-like graph whose nodes can represent regions of land or ocean. Initially the graph encodes a square landmass surrounded by water. My framework visualizes the result of the modeling procedure as a mesh that differentiates land and water locations by appropriately setting their height and color.

My models are extensions of the coastline model of Sapoval et al. [58] introduced in Section 2.3.3. However, my discretization of the problem domain is different and I extend the formulation of erosion in significant ways so that I can apply my avalanching paradigm and solve new modeling problems. My avalanching principle changes the fractal character of the resulting coastlines and controls how folded they can become. I provide additional applications of my coastline models by using them to produce coastlines with mixed behavior and construct island interiors. The latter procedure also forms a part of my two-stage terrain modeling method described in Section 4.3.4.

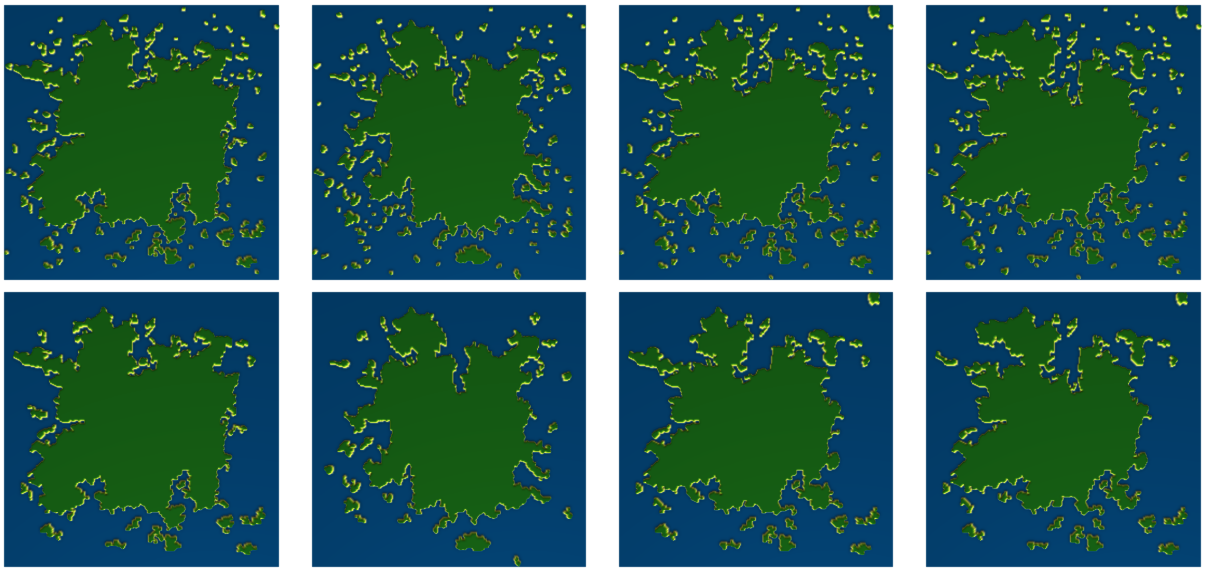
4.2.1 Sapoval Coastlines

Sapoval's model of coastline erosion introduced in Section 2.3.3 generates self-similar coastlines that result as a coast evolves under the action of a global erosion force. This process is consistent with SOC, as eroding pieces of the coast weaken their neighbors and cause erosion avalanches. The weakening of a coast segment is proportional to the increase in exposure due to the missing neighbors. The force of erosion is determined by Equation 2.8 reproduced below:

$$F(t) = \frac{f}{1 + \frac{gL(t)}{L(0)}}.$$

The changing perimeter of the coast $L(t)$ attenuates the force F. The parameter f scales the initial value of F. Decreasing the parameter g encourages the evolving coastline to become more folded to resist the erosion. However, to prevent F from being too strong, a decrease in g has to be paired with a decrease in f. In general, the parameters have to be selected carefully to avoid the two extremes when F is too high or too low and the coast, respectively, erodes completely or remains close to its initial shape (which is a square in the following examples). The model can respond drastically to small changes in the parameters, an effect that is amplified by the dependence of F on the entire shape of the coast. This is a weakness of the model, as the process of designing a coastline involves a lot of trial and error.

Figure 4.4 shows coastlines produced using the global perimeter model. Table 4.3 contains the corresponding values of parameters f and g. The model tends to produce a



(a) (b) (c) (d)

Figure 4.4: Islands constructed using erosion model based on global perimeter. Small islands are pruned in bottom row.

	(a)	(b)	(c)	(d)
f_0	0.495	0.476	0.460	0.444
g	0.180	0.150	0.130	0.110
grid size	256×256			

Table 4.3: Simulation parameters for Figure 4.4.

lot of small islands as a consequence of the attenuation of F by the combined perimeter of the entire coastline. The relative sizes of the small and large generated islands suggests that the scale of the entire construction is large in absolute terms (perhaps the scale of a continent). The impression is erroneous since the construction is scale-free and an increase in discretization density will cause even smaller islands to appear. It is difficult to use Sapoval's model to construct a single small island, because the evolution of a coastline under the model will tend to both lengthen the coastline and split off smaller islands.

As a first approximation to solving this problem, I followed a simulation based on Sapoval's model with a step that extracts islands whose perimeter is larger than a threshold value (24.0 grid units). My graph-centric representation of the problem setting helps with

this operation. Combined with the effect of discretization that hides detail smaller than the grid size, the resulting island models (shown in the bottom row of Figure 4.4) come closer to simulating the appearance of islands existing on a relatively small length scale. In other words, large islands must have abundant small features (like in the top row of Figure 4.4), so that removal of fine detail results in islands that appear to be smaller. A more complete solution to this problem should introduce a notion of scale into the behavior of the coastlines as they evolve (Section 4.2.3).

4.2.2 Coastlines Based on Local Perimeter

To better control the shape of the constructed islands, I changed F to be a local force dependent on a local formulation of the coast perimeter:

$$F(t) = \frac{f}{1 + gL_{R_p}(t)}, \tag{4.2}$$

where L_{R_p} is the length of the coast in a fixed radius R_p. The local perimeter and the local F for a piece of the coast C can be calculated in two ways: by considering any coast segment lying within R_p units of the location of C or by only considering coast segments that form a contiguous coast with C. Figure 4.5 shows islands resulting from both perimeter estimation strategies. The non-contiguous search tends to produce smaller islands like the global perimeter model, but they stay close to the coast of the mainland due to the limiting effect of the search radius R_p.

Besides controlling the creation of small islands, the local perimeter model determines the behavior of erosion avalanches locally, which has three additional benefits. First, the local model prevents coast segments on the opposite sides of an island from affecting each other's shape, which is more realistic than treating the entire coastline as a single unit. Similarly, the local model also allows coastlines with different parameters to evolve independently, which is a modeling problem considered in Section 4.2.4. Second, the local perimeter method makes it easier to select values for the parameters f and g, because if they result in a value of F that is too large for a significant part of the coast and it erodes, the large erosion event does not necessarily doom the remaining coastline. The parameter values for the islands in Figure 4.5 are given in Table 4.4.

The third benefit of the local perimeter model is an additional parameter for controlling the shape of the islands. A smaller value of R_p results in more erosion, because a coastline has less space for becoming folded in the circumscribed area. As R_p increases and the search

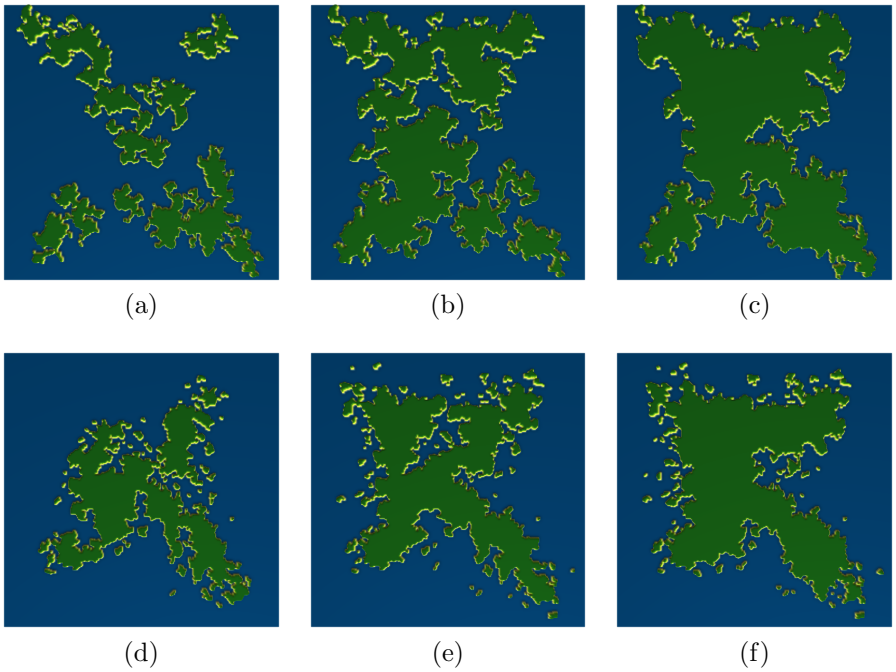


Figure 4.5: Islands constructed using erosion model based on local perimeter, which is calculated using contiguous (top row) and non-contiguous (bottom row) local search. Search radius increases from left to right.

	(a)	(b)	(c)	(d)	(e)	(f)
f_0	0.50					
g	0.20			0.15		
R_p	150	200	250	150	200	250
grid size	256×256					

Table 4.4: Simulation parameters for Figure 4.5.

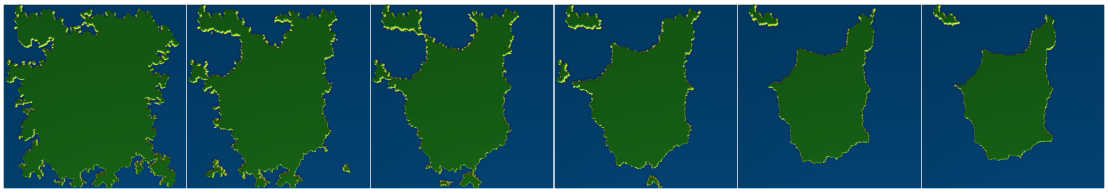
area covers the entire problem domain, the shape of the constructed coastline approaches the coastline generated by the global perimeter method.

On the other hand, my modification of the model to control erosion locally is not sufficient to control the length scale of the generated islands. The shape of the islands produced by the local perimeter model approximates the statistically self-similar (i.e., scale-free) fractal produced by Sapoval's global perimeter model. In both cases, the process that creates the fractal coastline depends on sampling its fractal properties (i.e., the perimeter

ratio) in an area, in a way similar to a procedure for estimating fractal dimension. The difference is that $R_p = \infty$ is a better approximation.

4.2.3 Coastlines with More Avalanching

My next modification of the model allows for the control of the length scale of the islands. This effect corresponds visually to coarseness of an island's coastline: a less folded coastline indicates that the island is smaller in absolute terms. Additionally, such a small island should not be surrounded by even smaller islands of the kind that the global perimeter model tends to produce, as discussed in Section 4.2.1. I achieve these goals through the introduction of additional avalanching into the model. Figure 4.6 shows a sequence of generated islands with increasingly flatter coastlines.



(a) (b) (c) (d) (e) (f)

Figure 4.6: Additional avalanching makes coast less coarse by weakening neighbors of erosion events. Amount of additional avalanching increases from left to right.

In my modified model, a segment of the coast can no longer withstand the erosion force F indefinitely due to all-or-nothing erosion events. Instead, the new model lets each site contain an amount of erodable material M, which plays the role of R for that site (initially $M = R$) and decreases each iteration. Each coastline site's M diminishes due to the action of F by an amount kF (I use $k = 1$) and due to nearby erosion events. Each time any site completely erodes ($M = 0$) it weakens its neighbors up to a fixed radius R_a away by decreasing their value of M. The decrease in M starts at a constant value a at the center of each erosion event and falls off to 0 at distance R_a. The effect of weakening in a radius is similar to spheroidal erosion modeled by Beardal et al. [3]. The values of parameter a used to produce the islands of Figure 4.6 are given in Table 4.5.

The additional scale-dependent avalanching makes coastlines less folded by causing more erosion near each erosion event, so that folds straighten back out. In other words the length of a coastline can no longer increase without bound and the resulting coastline shape is no longer scale-free. Since the generated coastlines can no longer reach the scale-free

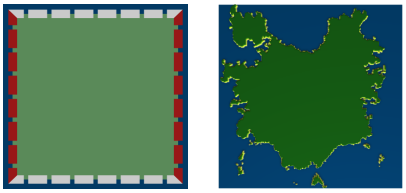
	(a)	(b)	(c)	(d)	(e)	(f)
a	0.0	0.05	0.075	0.10	0.15	0.175
R_a	25					
grid size	256×256					

Table 4.5: Simulation parameters for Figure 4.6.

state they continuously erode, which is consistent with Sapoval's interpretation of coastline dynamics in terms of SOC.

4.2.4 Extensions for Modeling Applications

Some real world coastlines are more folded on one side of a landmass than on the other. To produce such an effect with the coastline models of this section, it is necessary to prevent far away parts of one coastline from affecting a different coastline. This is made possible by my local perimeter model of Section 4.2.2. Figure 4.7 demonstrates a generated island with one type of coastline along its north and south boundaries and another along its west and east sides.

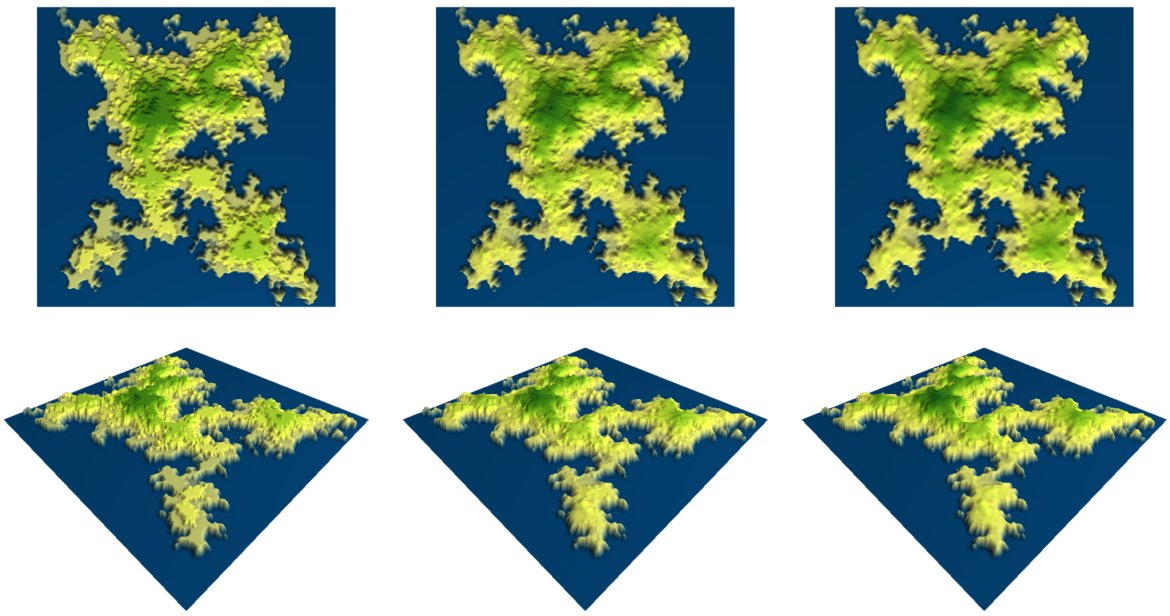


(a) (b)

Figure 4.7: Coastline simulations from Figures 4.6a and 4.6d combined to produce a single coastline: (a) parts of the coast that each simulation is responsible for, (b) result.

In a modeling application it is desirable to generate the interior of the islands as well. I extended the coastline models to keep track of the coastline as it evolves and designate certain previously existing coastlines as level sets of a terrain, with the newer coastlines corresponding to higher level sets. To generate the interior of an island, I restart coastline erosion with the currently existing coast as initial conditions. If the erosion stops due to the attenuation of F by local or global perimeter, my algorithm increases the value of f until the erosion resumes. As noted by Sapoval, the necessary change in F can be small due to avalanching that will happen as the erosion proceeds.

Finally, my algorithm interpolates heights in the interior of the island based on the level sets. The historic coastlines serve as plausible level sets according to the interpretation of terrain level sets as coastlines discussed in Section 2.1.3. This is possible because my algorithm treats generation of coastlines not as a problem of constructing curves, but as the higher dimensional problem of generating level sets of a terrain. Figure 4.8 shows three versions of a synthesized island terrain produced using nearest neighbor interpolation between the level sets, an interpolation method based on Hermite splines, and a method based on minimization of bending energy, which is generalized from interpolation used in natural cubic splines. Section 4.3.4 contains a more complete discussion of this terrain generation algorithm.



(a) (b) (c)

Figure 4.8: Interpretation of coastlines generated in the interior of islands as level sets. Different methods of interpolating heights between level sets are possible.

4.2.5 Use of the Common Modeling Framework

To implement the coastline erosion simulations of previous sections using my common modeling framework, I have represented coastlines as subgraphs of the problem domain

and used several types of agents to update the locations of the coastlines as they become eroded. The graph-centric approach of my framework restricts the graph nodes that have to be processed at each iteration to just those of the appropriate subgraphs. This is especially true in the case of simulations based on the global perimeter model and simulations that use the contiguous search to calculate local perimeter. Calculation of the local perimeter relies on the framework's BFS search primitive. The following list summarizes the types of agents I have used and their roles.

coast mark agent: an artifact agent that marks the graph nodes that belong to the current coastline. The agent stores the length that the local segments of the coastline contributes to the total perimeter. This agent type is also used to determine the total perimeter of each island when small islands are pruned in a post-processing step.

coast agent: always paired with a coast mark agent and responsible for destroying its part of the coastline and designating new coastline nodes by controlling the lifetime of all the associated agents. This agent type relies on the parameter values of a coastline erosion model to make erosion decisions.

artifact agent: an additional artifact agent that marks previously eroded locations, thus helping coast agents decide which of their neighboring nodes are located inland and should be designated as part of the new coastline following an erosion event. This type of agent also saves information about historic coastlines and helps with height reconstruction based on level sets.

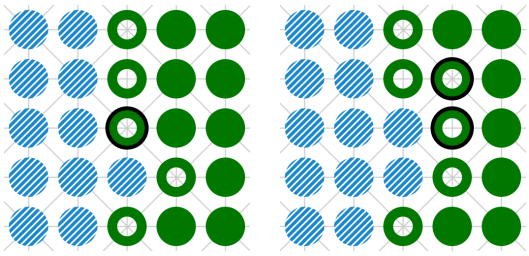


Figure 4.9: Simulation of coastline evolution using graphs and agents. Green circles are nodes associated with inland locations, blue circles are nodes that have eroded. Nodes circled in black change their state due to agent action: (a) before erosion event at one site, (b) after the event.

The purpose of artifact agents is to be a form of communication between other agents. After querying nearby locations for the presence of the two artifact agent types and their state, a coast agent can determine whether it should undergo erosion, as well as the neighboring locations that should become the new coastline. Figure 4.9 illustrates the deletion and creation of agents that occur during one erosion event. The nodes shown as unfilled circles describe the location of the coastline and contain the pair of a coast and coast mark agents. The previously eroded nodes contain artifact agents. As erosion occurs at a coastline node its labelling changes and a new artifact agent replaces its contents. In addition, erosion creates new coast agent pairs to designate previously empty inland locations that are nearby as parts of the new coastline.

4.3 Rivers and River Basins

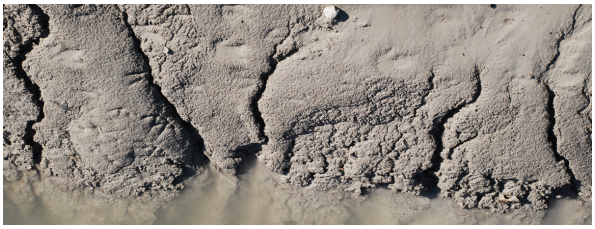
This section describes my model of hydraulic erosion that uses self-organization to create channels that capture tributaries, widen, and flood local minima. To simulate the totality of these behaviors, which has not been demonstrated by any existing model based on self-organization, I combine and extend the models of Rodríguez-Iturbe and Rinaldo [55] and Perron et al. [46]. In particular, application of my avalanching paradigm is an extension that allows channels to develop plausible width in my model.

I also contribute a more complete terrain modeling method that generates an initial terrain for my hydraulic erosion simulation, resulting in a two-stage procedure entirely based on self-organization of avalanching. The first stage of my method treats simulation of coastline erosion features as the higher order problem of generating appropriate level sets of the underlying terrain. In this way, my method uses self-organization to reproduce the relationship between coastlines and terrains that is also a part of the formulation of terrain modeling in terms of explicit fBm synthesis (Section 2.1.3). However, my method produces physically-based hydraulic erosion features that are not found in fBm and uses coastline and river erosion in combination that is novel compared to other simulation-based approaches. My method successfully reproduces phenomenology of hydraulic erosion (i.e., channels emerge and become linked), and synthesizes an eroded terrain with approximately affine scaling (Sections 2.1.4 and 3.1.3) and plausible hypsometry (Section 2.4.1).

My simulation of hydraulic erosion starts with a discretized heightmap that represents a terrain and adjusts the height of the terrain at each site using a PDE formulation. The amount of erosion tends to be higher in downstream areas and at sites where the local slope of the terrain is high. This simulation of erosion results in a network of channels on the surface of the terrain as water flow converges in a self-organized way. There is a

feedback relationship between the terrain and the channel network: the morphology of the terrain directs the flow of water, while the flow of water reshapes the terrain via erosion.

(a) roof



(b) mud bank

(c) sand hill

(d) sand pile

(e) sand pile

(f) limestone hill

(g) limestone hill

Figure 4.10: Photographs of erosion left by flow of water organized into channels.

Photographs of channels left by hydraulic erosion in Figure 4.10 give the general appearance of the erosion features that I aim to simulate. However, there is water in the channels of the terrains that I construct with my simulations. The appearance of rivers is not trivial to render, because the surface of the water lies on the surface of the terrain and requires additional geometry in the graphics pipeline. I provide more details about rendering rivers in Appendix A.2.

As for some particular visual attributes of channel networks, Figure 4.10a shows the result of nearly parallel flow on a flat substrate. Although distinct channels emerge they can run close to each other for some distance before linking up. By contrast, tributary

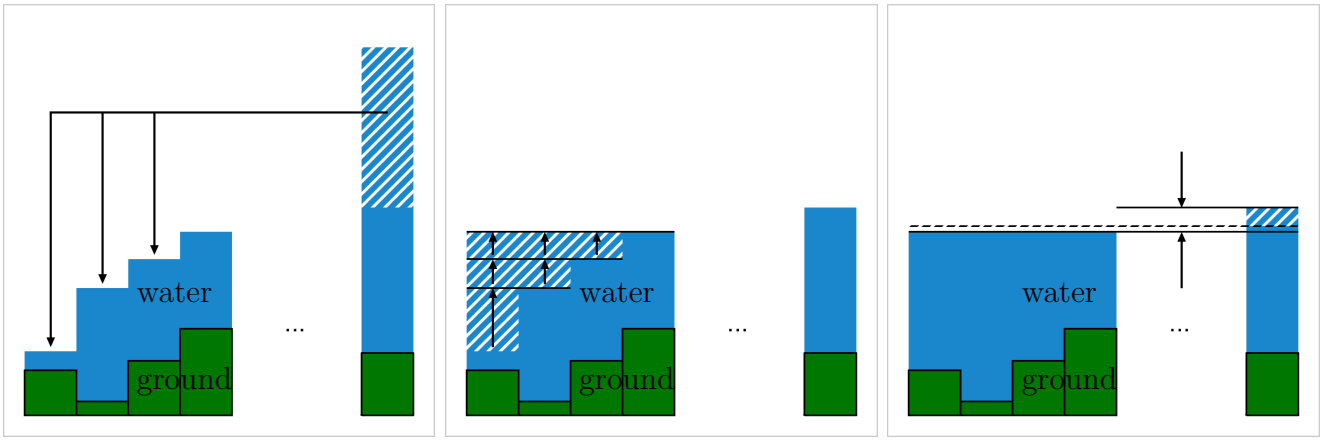
capture occurs sooner in Figure 4.10c, resulting in a network with more branching. The network in Figure 4.10b is reminiscent of a river network on a map. Rodríguez-Iturbe and Rinaldo [55] discuss this type of self-similar scaling behavior in the context of their SOC models of river basins.

4.3.1 Hydraulic Erosion Simulation

My approach to modeling hydraulic erosion relies on three main ideas. First, I use a specialized water-column algorithm to simulate the flow of water on a terrain. Second, I use the amount of local flow in a PDE formulation of erosion to determine how the terrain changes at each site. Third, I set up additional erosion avalanches to accelerate tributary captures and ensure that channels link up to form wider combined channels. The specialized water-column algorithm and the use of the avalanching paradigm are the main differences between my approach and the models of Rodríguez-Iturbe and Rinaldo [55] and Perron et al. [46] that also use self-organization. Besides modeling the width of the channels, the benefits of my approach also include the ability to model gradual flooding of local minima and eventual breakthrough of the resulting lakes to nearby channels. The existing models can not accurately determine the behavior of water near local minima of the terrain, because they use the area map concept in place of a water simulation. Additionally, algorithms based on the area map (e.g., the $D8$ algorithm) constrain drainage directions at each site in a way that my water-column algorithm does not (Section 2.3.2).

The water-column algorithm is one of the simplest methods of simulating water flow on a terrain (Section 2.3.2). It represents the level of water at each site on the terrain as the combined height of a column of water present at the site and the local height of the terrain. Neighboring sites exchange water to equalize their levels of water by following an iterative procedure. In my version of the water-column algorithm the water distribution strategy promotes concentration of water into channels by preventing diffusion, which is necessary for modeling channel formation. I provide a comparison of my algorithm with existing variants in Section 4.4.3.

Figure 4.11 illustrates the three main stages in the operation of my algorithm. In Figure 4.11a the algorithm selects a site with excess water, finds n of its grid neighbors that are lower, and sorts them in ascending order of combined ground and water height. In subsequent $n-1$ steps the algorithm takes water from the central site and uses it to raise the level of water in m lowest neighbors to match the total height of $(m+1)$-th neighbor for m in $[1, n-1]$, as shown in Figure 4.11b. As a final step, the algorithm checks whether the central site is still higher than its neighbors, in which case it equalizes all participating



(a) (b) (c)

Figure 4.11: Stages of my specialized water-column algorithm.

locations to a common level, which is the dashed line in Figure 4.11c. The distribution of water can stop at any step if the amount of water at the central site is exhausted. The algorithm tracks the amount of water received by each location during the distribution stage, which serves as the local magnitude of water flow in the rest of the simulation.

Perron et al. [46] use the following PDE formulation for the amount of erosion that occurs on the terrain represented by the function $z(x, y, t)$:

$$\frac{\partial z}{\partial t} = D\Delta z - K(A^m|\nabla z|^n - \theta) + U. \tag{4.3}$$

The two main terms of this equation are the diffusion term containing the Laplacian Δz and the advection term containing the gradient ∇z. The constants D and K respectively determine the amount of diffusion-based and advection-based erosion. The constant n is typically chosen to be close to 1.0. The constant m can take on a range of values, but, according to Rodríguez-Iturbe and Rinaldo, $m = 0.5$ has a physical significance because of a relationship between stress and flow depth. Perron et al. use 0 for the constant θ, so I do not include it in my formulation. The constant U represents fixed-rate uplift and is necessary for modeling the test topography in Figure 4.15.

The value of A is the area that contributes water to a given location on the terrain. The size of the contributing area determines the amount of water flow at the site. The simplest way to approximate A in the discretized setting is to count upstream nodes in an

area map, or a graph that encodes the "contributes water to" relationship. This procedure also discretizes the local flow direction, which leads to undesirable artifacts. Tarboton [64] discuss alternative methods of calculating the area map, but two fundamental limitations of the concept remain. First, area maps tend to emphasize either convergent or divergent flow making combinations of the types of flow difficult to produce. Second, the way area maps assign flow directions at sites near local minima of the terrain results in an arbitrary drainage direction for the corresponding lake. Moreover, the size of the lake is not well-defined, because area maps are based on the geometry of the terrain and do not involve any calculation for the current amount of water at a site.

In my simulation of erosion I adapt the equation of Perron et al. by replacing A with a quantity f derived from the flow computed by my water-column algorithm. The following equation is the result:

$$\frac{\partial z}{\partial t} = \begin{cases} \max(f w_\Delta \Delta z - f(w_d |D_{\vec{s}}| + w_0), z_{low} - z) & \text{if } z > z_{low}, \\ f w_\Delta \Delta z - f w_0 & \text{if } z \leq z_{low}. \end{cases} \tag{4.4}$$

The division into two cases ensures that advective erosion can not make a site lower than the height of its lowest neighbor z_{low}, since this type of erosion simulates the action of water running downhill. The constants w_Δ and w_d respectively weight the amounts of diffusion-based and advection-based erosion. $D_{\vec{s}}$ is the directional derivative in the direction of steepest descent. The constant w_0 is an optional fixed amount of erosion that occurs inside the channels in addition to the other types of erosion. I scale all of the erosion terms by f, so that the total amount of erosion is greater inside the channels that form. I do not except diffusion-based erosion from this scaling like Perron et al. under the assumption that the terrain outside the channels is covered by vegetation, which makes it resist diffusion.

I introduce the effect of avalanching into the calculation of f. My water-column algorithm provides the value f_c as the local magnitude of flow. Using f_c directly to calculate erosion will result in scale-free networks similar to those of Rodríguez-Iturbe and Rinaldo. In other words, river networks will have a graph-like (forest-like) appearance and in the discrete domain their width will be as narrow as the discretization density allows (see Figure 2.10). For rivers to have width, channels with large flows must collapse their banks and capture nearby flows. From the perspective of the avalanching paradigm, the erosion events leading to the emergence of a channel should cause further erosion in its vicinity. To achieve this effect my simulation contributes a fraction of f_c at a given site to all of its uphill neighbors within a search radius $R = f_c R_{max}$, where R_{max} is a fixed constant. The sum of contributed flows at each site forms an additional flow map f_a. I combine the two flow maps into a single map f for Equation 4.4 using the following formula:

$$f = (w_a f_a + w_c f_c)^m, \tag{4.5}$$

where w_a and w_c are weighting constants and m is a constant exponent similar to the exponent used with A, which f replaces.

From a modeling perspective, larger values for parameter w_Δ result in more smoothing along the channels. The parameter w_d, as well as the parameters used in calculating f, control the degree of channel incision into the terrain. The weights w_a and w_c in the equation for f determine the relative importance of avalanching around sites with large flows. The exponent m in the equation for f can make erosion in smaller channels comparatively more noticeable if it is less than 1.0. The parameter w_0 sets up additional erosion inside all channels at the rate proportional to f, which also brings out small channels.

To solve Equation 4.4 I discretize z on the simulation grid and approximate its first and second derivatives by locally fitting quadratic surfaces using the least squares method. This operation is a feature of my common simulation framework. Having obtained discretized z and its derivatives, I apply Gaussian iteration to achieve a forward Eulerian integration step. I find it convenient to let the the right hand side of Equation 4.4 determine the step size. In other words, the step size is absorbed into the parameters I use to scale the amount of erosion.

Figure 4.12 shows some results of my hydraulic simulation with parameter values given in Table 4.6. Each row of the figure contains a set of initial conditions, the resulting channel network, and a plot showing hypsometric curves of the underlying terrain before and after erosion. I discuss the hypsometry (introduced in Section 2.4.1) of the eroded terrains further in the following sections. The terrain in Figure 4.12c is especially significant, because it demonstrates the ability of my simulation to model convergent flow resulting from downhill water runoff, as well as divergent flow resulting from an obstacle.

	w_a	w_c	m	w_Δ	w_d	w_0	grid size
(a)	1.0	2.0	0.5	10.0	3.0	1.5	
(b)	2.0	1.0	1.0	10.0	3.0	0.0	128×256
(c)	2.5	1.5	0.5	5.0	3.0	0.5	

Table 4.6: Simulation parameters for Figure 4.12.

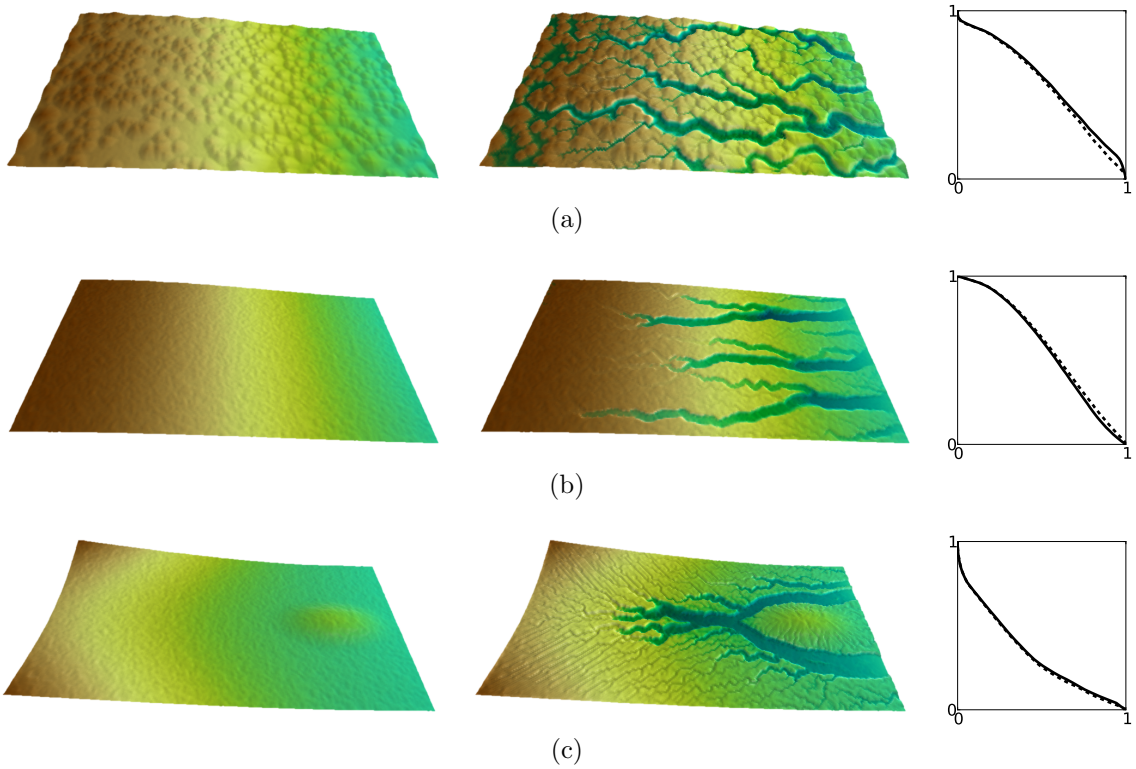


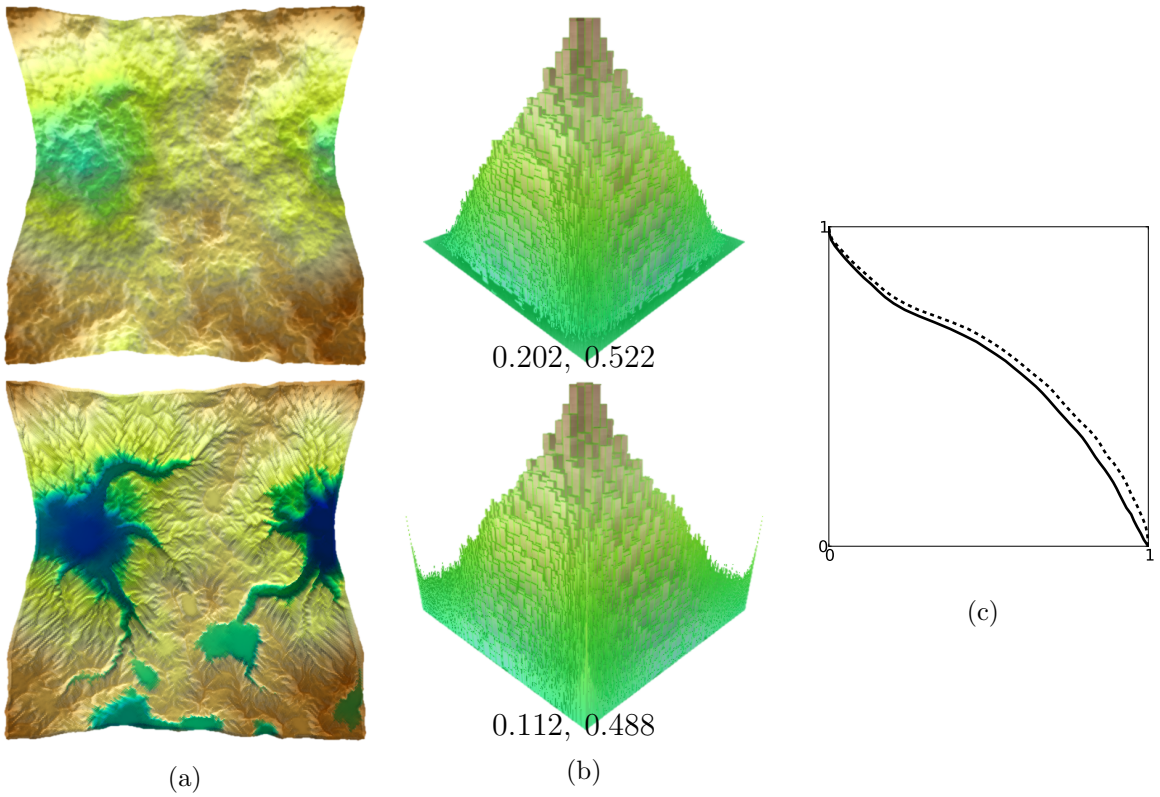
Figure 4.12: Each row contains the initial conditions, followed by the generated river network and the hypsometric curves of the topography before and after erosion (final curves are heavier).

4.3.2 Effect on Spectral Density and Hypsometry

To evaluate the effects of my erosion algorithm on terrains I investigated the changes it produces in the terrains' spectral density and hypsometry. These properties of terrains are relevant when extending an erosion algorithm to a complete procedural terrain modeling application. The spectral properties of the generated terrains should exhibit the power falloff characteristic of fBm as many natural terrains do (Section 2.1.3). The avalanching paradigm used in my erosion simulation can act as a source of the affine fractal scaling associated with the power falloff (Section 3.1.3). As for the terrains' hypsometric properties, they should tend to the hypsometric forms of Figure 2.12 (Section 2.4.1), especially the monadnock and equilibrium forms, as the amount of erosion increases.

Applying my erosion algorithm to synthetic terrains like in Figure 4.12 may not pro-

duce these properties if the effects of erosion do not overwhelm the shape of the initial terrain. In most situations like in Figures 4.12 and 4.13, the spectral density and the hypsometry of the initial terrain are not significantly altered. This makes it possible to use my simulation as a second stage in a terrain modeling method that begins by generating a terrain with the natural spectral and hypsometric properties. The subsequent erosion will add visually apparent erosion features like rivers to the terrain, but will not destroy its original properties. I develop such a terrain modeling algorithm in Section 4.3.4.



(a) (b) (c)

Figure 4.13: Effects of hydraulic erosion on fBm landscape: (a) – (b) terrain before and after erosion and corresponding log plots of spectral density, (c) hypsometric curves (darker curve for eroded terrain).

Figure 4.13 shows the result of applying my erosion simulation to an fBm-generated terrain. The spectral density plots in the figure demonstrate that the spectral properties of the terrain are not altered significantly, with most of the changes confined to the highest frequencies (away from the center of the plot). The hypsometric curves in the figure

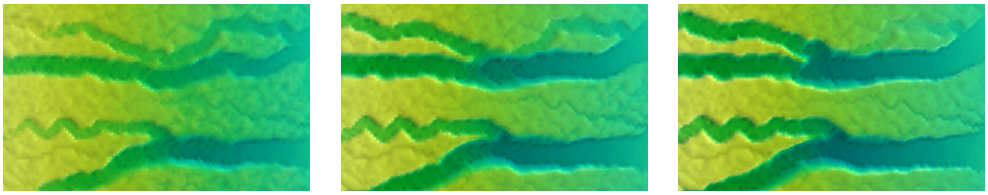
show that the curves are nearly identical before and after erosion, similar to the curves in Figure 4.12.

Figure 4.13c indicates that the hypsometric curve of fBm exhibits features appropriate for terrains in erosion equilibrium: concavity in the upper part, convexity in the lower part, and an inflection point. Therefore the aggregate appearance of fBm terrains is similar to that of eroded terrains, even though the former does not contain any erosion features. If the opposite were true, fBm would be less flexible as a modeling tool.

4.3.3 Emergent Behavior of Channel Networks

Effects of Avalanching

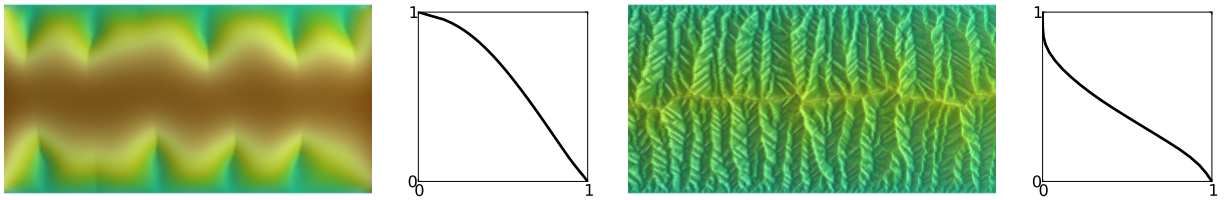
The characteristic shape of the channel networks created with my algorithm is mainly due to avalanching. Erosion occuring at a given site can make it lower than its neighbors and re-orient water flow towards it. This concentration of water flow can begin an avalanche of erosion at the current site and downstream of it. Erosion events can also propagate upstream, as a channel becomes more incised into the underlying terrain and the heights along the bottom of the channel equalize with its lowest point downstream.



(a) (b) (c)

Figure 4.14: Erosion avalanches reorient flow in a channel network resulting in tributary capture.

My method of calculating the approximation of flow f in Equation 4.4 also ensures that the erosion at each site can be scaled by nearby large flows. At the macroscopic level this behavior causes large rivers to become wider and capture tributaries in an area around them. Figure 4.14 illustrates this behavior.



(a) (b)

Figure 4.15: Modes of valley evolution differentiated by formation of secondary valleys: (a) diffusion-dominated, (b) advection-dominated.

Evolution of Valleys

The equation of Perron et al., which is the basis of my PDE formulation of erosion (Equation 4.4), has two extreme solutions corresponding to diffusion-dominated and advection-dominated erosion. The smoothing effect of diffusion-dominated erosion diminishes the formation of secondary incisions that branch out from primary ones. Although I have replaced some terms in the PDE with quantities derived from my water simulation, my formulation retains the ability to produce these two types of erosion as long as the diffusion term is not scaled by f. The scaling is a modeling decision necessary to allow smoothing only inside the channels, i.e., the area not protected by vegetation. However, to reproduce the model topographies of Perron et al. I allow smoothing to occur everywhere. Additionally, I include a constant uplift term needed for these topographies. These two changes result in the equation below.

$$\frac{\partial z}{\partial t} = \begin{cases} \max(w_\Delta \Delta z - f(w_d |D_{\vec{s}}| + w_0) + w_U, z_{low} - z) & \text{if } z > z_{low}, \\ w_\Delta \Delta z - f w_0 + w_U & \text{if } z \leq z_{low}. \end{cases} \tag{4.6}$$

Figure 4.15 shows the two model topographies with different types of erosion. Table 4.7 contains the corresponding parameter values. The figure also includes the hypsometric curves of the terrains. The hypsometry of the advection-dominated terrain shows the classic shape of the equilibrium evolution phase. On the other hand, the hypsometry of the diffusion-dominated terrain does not fit the classification. This provides another justification for my decision to restrict the effect of diffusion with the scaling term in my main equation and focus on modeling advection-dominated terrains.

	w_a	w_c	m	w_Δ	w_d	w_0	w_U
(a)	0.1	1.0	1.0	20.0	3.0	0.0	0.03
(b)	0.0	1.0	0.3	3.0	5.0	0.0	0.30

Table 4.7: Simulation parameters for Figure 4.15 and Equation 4.6.

4.3.4 New Method of Terrain Modeling

Simulating hydraulic erosion addresses only one part of the terrain modeling problem, because it can only add erosion features to an existing terrain. A complete terrain modeling method based on hydraulic erosion needs to incorporate a step to construct the initial conditions for the erosion algorithm. So far, the examples in Section 4.3.1 used purely synthetic initial terrains based on ramps and sinusoid shapes (although the use of sinusoid shapes can be partially justified as an approximation of synclines, i.e., folds in terrains). The example in Section 4.3.2 erodes an fBm terrain, which has a strong physical justification, but presents some modeling difficulties associated with generating fBm (Section 2.1.5). For example, fBm terrains created via spectral synthesis with an inverse Fourier transform are periodic and difficult to control. This is why I have designed a procedure based on self-organization to create terrains as a first stage for my hydraulic simulation. As Section 4.3.2 shows, my erosion algorithm does not significantly change the spectral and hypsometric properties of the underlying terrain, so the combined procedure benefits from having the separate initial stage that creates a terrain with a controllable appearance that also exhibits realistic fractal and hypsometric character.

The basic idea of the initial terrain algorithm comes from the erosion simulation of Section 4.2 and its application to generating island interiors discussed in Section 4.2.4. So, the first stage of my complete modeling method treats simulation of coastline erosion as the higher order problem of generating appropriate level sets of the terrain containing the coastlines. In this way, my method uses self-organization to reproduce the relationship between coastlines and terrains that is also a part of the formulation of terrain modeling in terms of explicit fBm synthesis (Section 2.1.3). However, my method produces physically-based hydraulic erosion features that are not found in fBm and uses coastline and river erosion in combination that is novel compared to other simulation-based approaches.

Besides providing a valuable connection between coastline and terrain modeling, my use of the coastline erosion algorithm in the interior of the underlying terrain makes sense in terms of the coastline model used by my algorithm. The coastline model is formulated in an abstract way and is also applicable to any situation where a front of material "caves in" due to exposure and weakening from its neighbors. Furthermore, flooding a terrain to

any level must produce a plausible coastline. Listing 4.1 outlines the first stage algorithm in pseudocode form.

Listing 4.1: First stage of terrain generation algorithm.

```
1  seed erosion fronts using a mask
2  label interior and exterior of fronts consistently (Section 4.2.5)
3  while uneroded sites remain {
4      iterate coastline erosion model
5      d = each site's BFS distance from last level set
6      D = maximum such distance
7      if (D > threshold) {
8          force erosion event at sites with d = 0
9          mark new level set
10     }
11     if (erosion front is stuck) perturb F (Equation 4.2)
12 }
13 assign heights to level sets linearly
14 interpolate heights between level sets
```

The mask in line 1 initializes each location on the terrain as either high or low, so that the erosion fronts begin at the boundary between the two types of locations and propagate into the interior of the regions composed of the high locations. I have chosen the local perimeter model for use in line 4 because its semantics allow for the possibility of mixing erosion fronts with different properties. However, I have not found it necessary to exercise fine-scale control over the shape of the level sets. Instead, the shape of the resulting terrain depends on the specified mask of initially eroded locations, which become the lowest points on the terrain. This mask is particularly suitable for setting up the areas of oceans and rivers.

Figure 4.16 shows a simple terrain generated with this algorithm. The initial seed is a single point in the center of the terrain. From a hypsometric point of view, this is an ideal situation for my algorithm, because a small set of seeds results in initial fronts with short perimeters, producing convexity in the hypsometric curve for low heights. This effect is due to small fronts having a higher probability of using the entire area allowed by the threshold distance in line 7. In contrast, fronts with large perimeters are likely to have a few locations where erosion avalanches rush into the interior quickly violating the threshold distance and causing a new level set to be created. The leftover sites increase in height as a result. As for a similar effect that produces concavity in the part of the curve corresponding to large heights, it is always guaranteed to occur, because level sets either climb up hills and become shorter or run out of area and create hills. The latter situation is apparent along the boundary of the terrain in Figure 4.16. Figure 4.17 shows terrains

generated with more interesting masks (painted by hand) and eroded with my hydraulic simulation. This set of terrains uses a grid size of 256×256. I discuss the terrains' spectral and hypsometric qualities in Section 4.4.4.

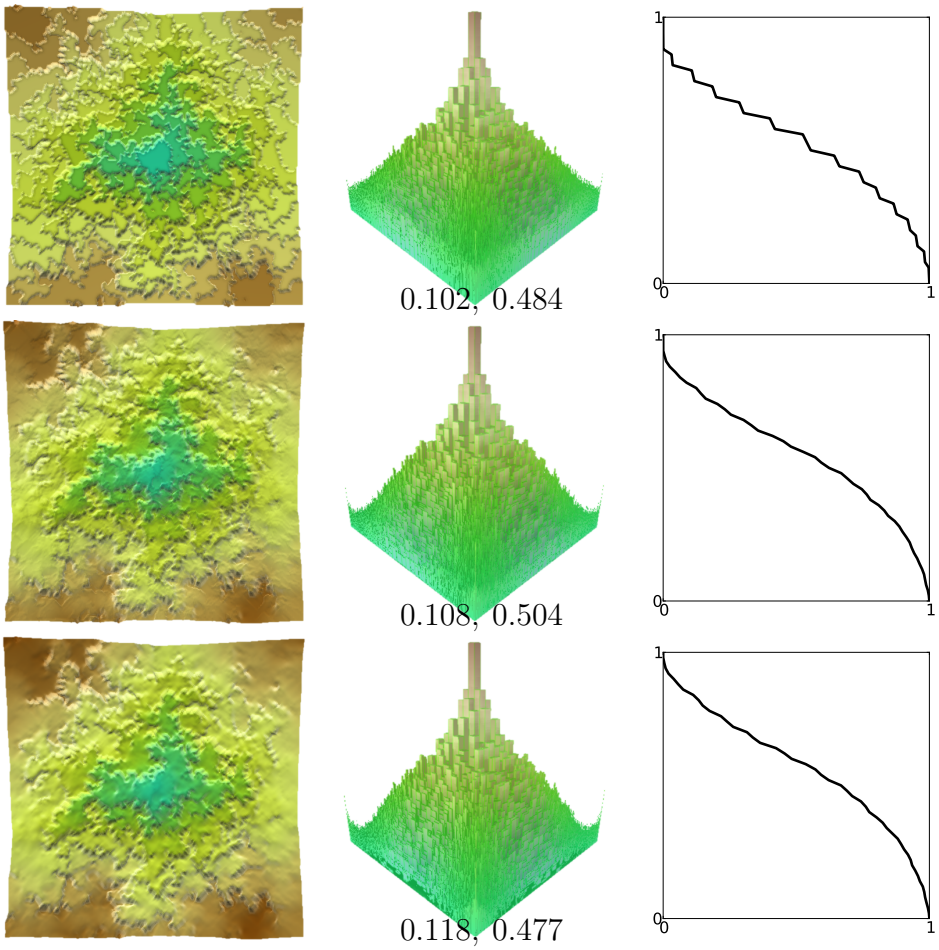
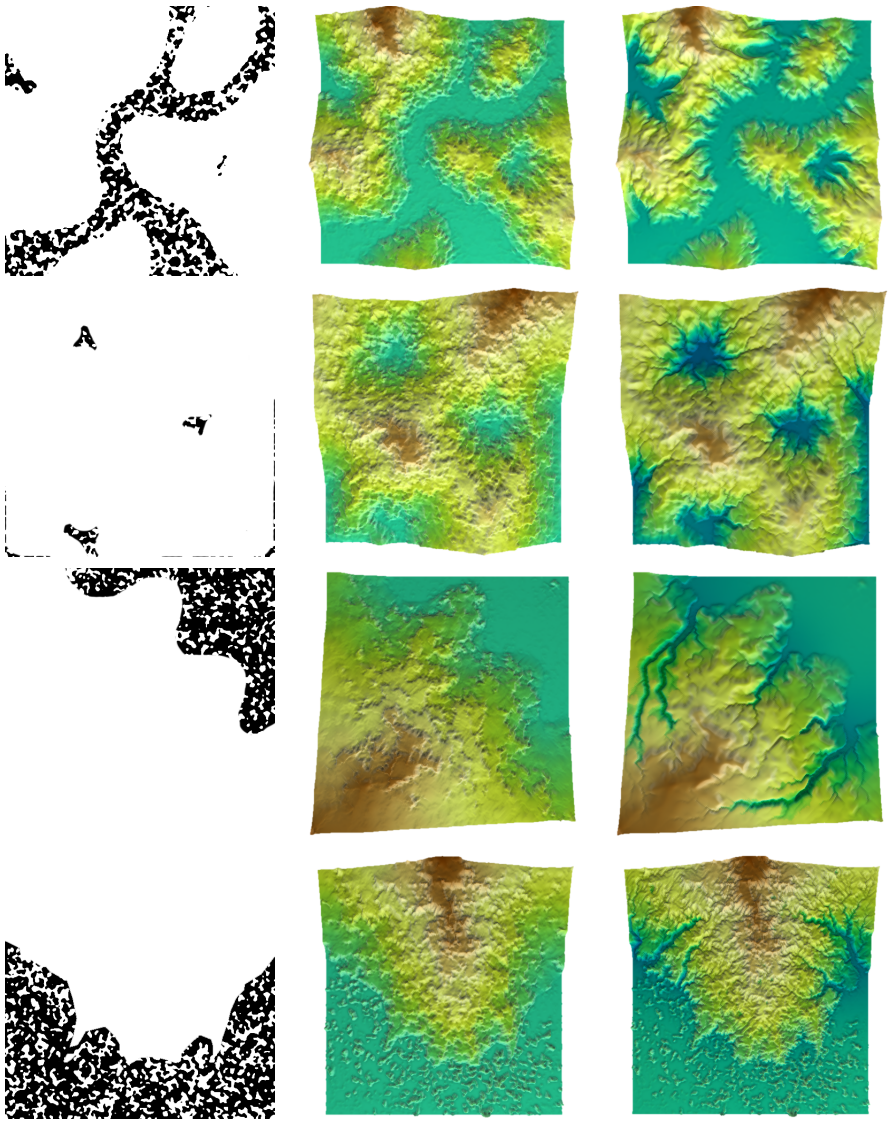


Figure 4.16: Comparison of interpolation strategies for use in the new terrain modeling algorithm (nearest, Hermite, and variational): (a) generated terrain, (b) log plot of spectral density, (c) hypsometric curve.

Figure 4.16 also compares different interpolation strategies for the height of the terrain between the level sets. The method of the top row sets unassigned heights to the lower of the two nearest level sets. The terrace-like stacking of the flat areas is apparent in

the generated terrain and the hypsometric plot. This illustrates the need for a better interpolation method unless the terraces are explicitly required for a specific kind of terrain. The method of the middle row uses a variant of Hermite interpolation similar to the procedure of Hormann et al. [24]. The method of the bottom row uses the result of the previous method as initial conditions and follows it with a relaxation procedure based on an energy formulation similar to the procedure of Szeliski and Terzopoulos [63]. The reduction in this approximate bending energy was 43.67%.



(a) (b) (c)

Figure 4.17: Additional examples of terrains constructed with the two-stage modeling method: (a) mask specifying level set seeds, (b) initial conditions for hydraulic stage, (c) final result.

4.4 Discussion of Terrain Simulations

In this section I discuss my methodology for developing self-organized simulations of erosion, especially the role played by avalanching and my simulation framework. Additionally, I carry out a detailed analysis of the water-column simulation component. I also analyze the results of my erosion simulations.

4.4.1 Use of Avalanching in Modeling

As I stated in Section 3.2.1, my modeling approach uses physically-based self-organization principles to generalize the modeling procedures for several types of geomorphologic features. In applying this approach to simulations of dunes, coastlines, and rivers I identified avalanching behavior in conceptual models of erosion, re-defined the conceptual models to emphasize self-organization (and in some cases model new types of behavior, such as channel widening), and finally designed appropriate simulations using my common framework. However, my approach does not merely re-define models of erosion in terms of avalanching, but can serve as a conceptual model by itself, due to its abstract view of erosion.

In particular, my model of hydraulic erosion on terrains (Section 4.3.1) uses avalanching to concentrate water into channels and promote tributary capture, while the amount of erosion around the developing channel network is determined by a PDE also used in the model of Perron et al. [46]. The self-organized behavior of the channels is a primary part of the model, while the PDE formulation of erosion, which by itself is a type of erosion model used in hydrogeomorphology, plays a secondary role. Furthermore, the use of avalanching in my hydraulic erosion model is closely related to the flow of information during its simulation: the channels communicate by competing for tributaries, tributary capture causes a channel to carry more water downstream, and incision of rivers into the terrain causes upstream locations to become drained. Therefore, the avalanching principle leads to a high-level model of the entire erosion process without the need to specify the details of water flow or how much material is removed. This allows avalanching to be used to design new types of simulations.

4.4.2 Evaluation of Framework

The final step of my modeling approach is to develop a simulation according to principles of self-organization by using appropriate facilities of my common simulation framework (Sec-

tion 3.2.3). To implement my simulations of dunes, coastlines, and rivers I have followed the process outlined below.

1. Represent the feature to be modelled in a graph-centric way.
    - The common framework maintains the graph that represents the entire problem domain, while the subgraphs corresponding to individual features are identified implicitly.
2. Identify the state variables to be tracked at the sites that belong to the features.
    - The common framework provides for a MAS-based structure for storing and updating the state, as well as communicating changes between neighbors.
3. Set up an iterative algorithm for updating the state variables to simulate evolution of the features.
    - The common framework provides various subroutines for computing updates and communicating them throughout the problem domain. These include computation of derivatives and different kinds of searches (Section 3.2.3).
4. Link the state variables to measurements (e.g., height) that can be visualized on a terrain.
    - The common framework displays the terrain, as well as markers that can be used for debugging.
5. Provide initial conditions for the simulation.

Many of the design steps rely heavily on facilities provided by the common framework showing how the framework can save work needed to design each specific simulation. I discussed above how each specific simulation that I have implemented uses the common code (Sections 4.1.3, 4.2.5, and 4.3.1). Speaking generally, a lot of the effort (steps 1, 2, and 4) is descriptive in nature and involves declaration and manipulation of variables. The only conceptually difficult part, which has to be unique for each simulation, is step 3. It requires the knowledge of the problem domain and an external definition of the processes that cause the features being modelled to evolve. However, the common framework provides computational building blocks needed to express the external concept in terms of state changing avalanches of the avalanching paradigm.

Since the code of the simulations based on the common framework involves a lot of trivial manipulation of variables (e.g., for setting up different types of initial conditions), the size of the simulations in lines of code is not the best measure of the effort involved

in creating them. Nevertheless, I have computed the ratio of specific to common lines of code for my simulations of dunes, coasts, and terrains with hydraulic erosion; respectively, they are: 0.36 : 1, 0.47 : 1, and 0.64 : 1. The size of code specific to each simulation is a reasonable proportion of the common code. In calculating these numbers I have not included any code related to visualization, which would skew the results in favor of the common framework. I have also not included the code needed for comparisons in Section 4.4.3.

Performance

Computational costs typically associated with physics simulations are substantial. Additionally, the design of my simulations prioritizes capture of complex emergent behavior over non-trivial performance enhancements. For these reasons, the modeling techniques that my framework encapsulates are not real-time.

Table 4.8 summarizes the performance of my simulations of coastline erosion. Recall that simulations in Figure 4.5 (a)–(c) use a search constrained to coastline locations, while the simulations in (d)–(f) visit a much larger number of sites in a breadth-first manner. This accounts for the large growth in runtime for the latter type of simulations. The design of simulations with extra avalanching also includes visiting locations in a radius to weaken them due to nearby erosion events. However, the weakening effect increases the amount of erosion per iteration, so the total runtime is lower, as exemplified by the runtime for the simulation in Figure 4.7.

	4.5a	4.5b	4.5c	4.5d	4.5e	4.5f	4.7
grid size	256×256						
runtime (s)	195	308	199	1444	2179	3724	78

Table 4.8: Performance of coastline erosion simulations.

To explore performance issues of my simulations further, I have implemented a version of the coastline simulations as a GPU-based algorithm (see Appendix B.1 for details). Fully leveraging GPU functionality required compromising the high-level model, but resulted in an algorithm that was able to produce acceptable results many orders of magnitude faster. However, direct comparison between the CPU and GPU coastline simulations is difficult and does not immediately suggest a way to also speed up my hydraulic erosion simulations.

Table 4.9 summarizes the performance of my simulations of hydraulic erosion. Stage 1 refers to the algorithm for constructing an initial terrain suitable for erosion, which is the

first step of my combined two-stage terrain modeling method. The algorithm is based on my coastline erosion model that uses local perimeter and a contiguous search (Figure 4.5 (a)–(c)), so the performance of stage 1 should be similar to those coastline simulations. Stage 2 refers to the algorithm for simulating hydraulic erosion, which creates river channels on top of a terrain. Its computational cost tends to be higher than that of stage 1, because it can take many iterations to erode the terrain and uses searching in a radius to produce an avalanching effect.

	4.12a	4.12b	4.12c	4.17 (first)	4.17 (third)
grid size	128×256			256×256	
stage 1 runtime	N/A			2m 0s	2m 36s
stage 2 runtime	4m 27s	8m 6s	7m 27s	3m 51s	8m 13s
stage 2 iterations	300	450	450	50	150

Table 4.9: Performance of hydraulic erosion simulations.

The performance numbers of Table 4.9 may be compared to the running time of simulations in World Machine, which is discussed in detail in Section 4.4.5. Similar to terrains produced with my method, a terrain can be created in World Machine in two steps: synthesis of a fractal terrain and simulation of erosion. However, simulation of erosion in World Machine does not include a simulation of water flow, which makes the algorithm much faster. A terrain like in Figure 4.26c with grid size 512×512 can be completed by World Machine in about 14 seconds with a single worker thread. A 256×256 terrain takes less than 2 seconds.

4.4.3 Water-Column Algorithms

Water-column algorithms have been used in both animation and erosion simulations. One version of the algorithm uses a water distribution strategy based on averaging and is exemplified by the work of Olsen [41]. I have implemented this type of water-column scheme based on both a uniform average and a weighted one. The main idea of the scheme is to transport a local excess of water to lower neighboring locations. Therefore, any local bumps in the surface of the water become smaller with every iteration. The following is high-level pseudocode for this type of water transport scheme:

Listing 4.2: Water-column algorithm based on averaging.

```
1 H = combined height at central site
2 for each neighbor {
```

```
    h = combined height at neighbor site
    if (h < H) {
        neighbor participates in distribution
    }
}

A = average combined height of participating sites
W = water at central site
X = min(W, H − A) // total amount of water to transfer

for each neighbor {
    if (neighbor participates) {
            f = fraction of X weighted uniformly or by height difference
            transport f*X from central site to neighbor
        }
}
```

The second type of water-column algorithm adds virtual pipes that connect the water columns and transports water based on a hydrostatic formulation of water flow inside of the pipes. The hydrostatic approach is exemplified by the work of O'Brien and Hodgins [40] and Šťava et al. [69]. The main difference of this scheme from the simpler one based on averaging is the explicit calculation of water flow (in volume per unit of time) inside the pipes, which produces changes in water levels inside the water columns. So this scheme has an explicit time step. Excess water is distributed to neighbors that are connected to the central site by pipes with positive outward flow. There is an additional complication that it can take time for a flow with large magnitude to reverse, so the direction of flow in the pipes may temporarily stay out of sync with geometry. The following is high-level pseudocode for my implementation of the scheme:

Listing 4.3: Water-column algorithm based on hydrostatic pipes.

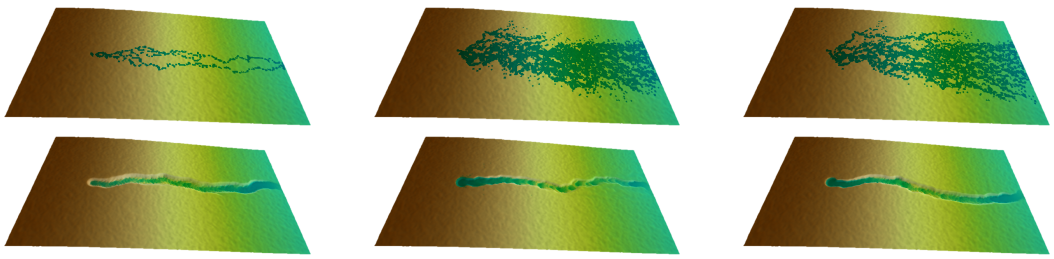
```
for each pipe {
    a = acceleration of water in the pipe based on hydrostatic pressure
    update flow in the pipe using an Eulerian integration step
}

V = volume flowing out of central site during the current time step
W = water at central site

for each pipe {
    scale back flows so that V does not exceed W
}

start a new pass
```

```
14  for each neighbor {
15      scatter outgoing volume updates to neighbors
16  }
17
18  start a new pass
19  update water column with the difference of incoming and outgoing volume
```

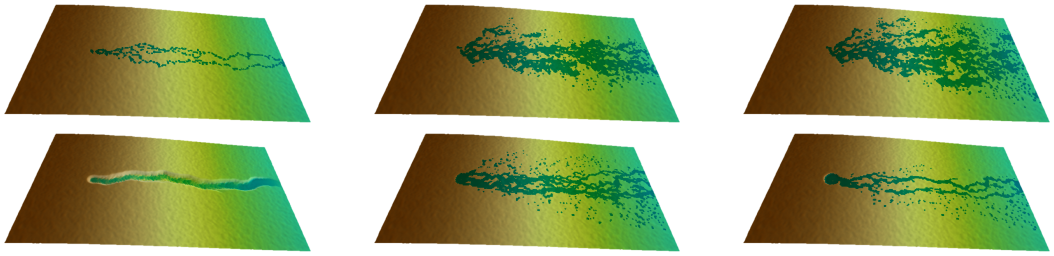


(a) (b) (c)

Figure 4.18: Comparison of water-column algorithms in combination with erosion: (a) special algorithm from Section 4.3.1, (b) distribution based on uniform average, (c) distribution based on weighted average. Water flows without causing erosion in the top row.

Figure 4.18 compares my specialized water-column algorithm from Section 4.3.1 with the version that is based on averaging. The test case is a sloping terrain from Figure 4.12b with a single source of water. In the top row the water flows without causing erosion and in the bottom row the water-column simulations work in combination with the erosion simulation from Section 4.3.1. In my algorithm lower locations are prioritized during water distribution in such a way that if the available water runs out, the remaining neighbors do not receive any water at all. In contrast, averaging-based algorithms will always contribute a fraction of the available water to all lower locations. That is why my algorithm concentrates water flow into proto-channels (these channels evolve into rivers if there is erosion) earlier than the averaging-based algorithms, which spread the water over a much larger area. However, combining both types of algorithms with erosion produces a single dominant channel. The placement of the channel differs slightly in each case: my algorithm identifies several potential paths that follow a sequence of local minima of the terrain (Figure 4.18a top) and all three versions result in different segments of the paths becoming the dominant ones.

Figure 4.19 compares my specialized algorithm (Figure 4.19a is reproduced for convenience) and the hydrostatic version. The time variable in the hydrostatic version can artificially slow down the simulation to an arbitrary degree relative to other versions of



(a) (b) (c)

Figure 4.19: Comparison of water-column algorithms in combination with erosion: (a) special algorithm from Section 4.3.1, (b) hydrostatic version, (c) hydrostatic version with five times more iterations.

the water-column algorithm. That is why Figure 4.19c shows the results of running the hydrostatic version for a larger number of iterations. I have found that the hydrostatic version with Δt close to 1.0 (s) produces water that is too "spiky" (i.e., the bumps in the surface do not have time to equalize due to the errors in Euler integration), while $\Delta t = 0.2$ (s) produces visually acceptable results. However, equalization of water based on the flow in the virtual pipes happens slower than the direct distribution of water in the other water-column algorithms. Consequently, when the hydrostatic algorithm operates together with the PDE formulation of erosion (Section 4.3.1) the water flow is slow to react to erosion events, preventing a single concentrated channel from emerging. Even after five times more iterations (about an order of magnitude longer in wall clock time) the result of hydrostatic simulation with erosion looks like the result of my specialized water simulation without erosion. So I conclude that it is impractical to combine the hydrostatic water-column algorithm with the PDE formulation of erosion of Section 4.3.1. Additionally, both the averaging and the hydrostatic schemes spread water into large areas surrounding the terrain's local minima, which makes them more dependent on discretization density than my specialized algorithm. If the input amount of water is held constant, then intermediate locations originating at a higher discretization level will take water away from locations originating at coarser levels; alternatively, the input amount of water may be proportionally increased, changing the volume of water moving through the system.

4.4.4 Evaluation of Results

Dunes

I have used my modeling framework to implement a variant of Werner's model of dune formation and reformulate the model to use avalanching in place of a geometric visibility test. The simplicity of the underlying model, which is based on transporting fixed amounts of sand according to the test, makes it apparent that in the reformulated simulation it is avalanching that is solely responsible for the complex emergent shape of the dunes. In other words, my dune simulations present an application of avalanching that motivates the development of my procedural modeling method, which has avalanching at its foundation. Additionally, implementing dune simulations within my framework allows me to demonstrate a larger diversity of avalanching-based simulations. However, the dunes that I have produced using my simulations are not significantly different from those produced by Werner, so I do not evaluate them.

Coastlines

I have been able to construct different kinds of coastlines using my framework and my extended model of coastline erosion. Two results are especially significant: a range of coastlines of varying roughness (Figure 4.6) and a combination of coastlines with different roughness (Figure 4.7). These results illustrate the ability of my method to directly control the shape of the coastlines in two ways that are necessary for modeling real world coastlines.

Richardson's [54] measurements of coastlines provide empirical evidence for two phenomena that correspond to these modeling problems. First, the roughness of coastlines can vary a lot. According to Richardson's data the fractal dimension of the coast of South Africa is only 1.02, while that of the west coast of Great Britain is 1.25. Second, a given stretch of a coastline can exhibit mixed fractal behavior. Richardson has found that the east coast of Great Britain behaves differently from the west one and its fractal dimension varies between 1.15 and 1.31.

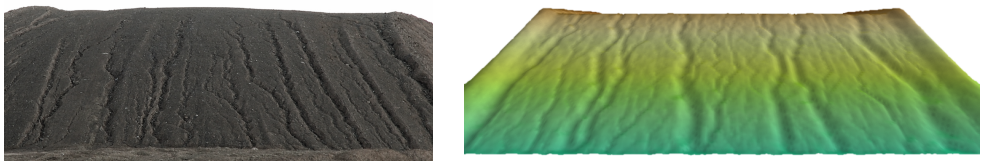
I do not evaluate the validity of the fractal dimension of the coastlines I produce with my method, because natural coastlines can have such a large range of dimensions. One of the largest dimensions reported is 1.52 for the southern part of Norway [16]. Passing any curve with fractal dimension between about 1.0 and 1.5 is too permissive for a test of validity, as the range includes most "reasonable" curves that can pass for a coastline. In contrast, fractals with dimension less than 1.0 are disconnected and fractals with dimension significantly bigger than 1.5 get too close to space-filling curves that do not look like

coastlines. The degradation of coastlines with increasing fractal dimension is apparent in Figure 2.2.

Terrains with Rivers

Rodríguez-Iturbe and Rinaldo [55] discuss several forms of behavior exhibited by real world water channels, such as tributary capture. However, the authors overlook some other types of behavior, such as gradual flooding of local minima and eventual breakthrough of the resulting lakes to nearby channels. This decision originates from limitations of the area map concept that Rodríguez-Iturbe and Rinaldo use in their simulations. Perron et al. [46] provide a similar view of the emergent properties of real world rivers in respect to the competition between nearby channels. They also state that erosion associated with the channels can be advection-dominated or diffusion-dominated.

In my model of hydraulic erosion, I have used my own water simulation with avalanching in place of the area map. Therefore, I have validated the behavior of channels formed with my model by showing that the types of behavior mentioned above are reproduced. Figure 4.14 shows channel capture and Figure 4.15 shows the two modes of valley evolution. Figure 4.12a and Figure 4.13a contain lakes that are drained with rivers. The latter type of behavior is an improvement relative to models of Rodríguez-Iturbe and Rinaldo and Perron et al. that use self-organization, but it should be reproducible with other existing simulations that emphasize fluid simulations, such as that of Šťava et al. [69].



(a) (b)

Figure 4.20: Comparison with channels on a sand hill.

Figure 4.20 shows my attempt to reproduce the appearance of a real channel network from one of the photographs in Figure 4.10. There are a lot of visual similarities between the channels on the synthetic hill and its real-world counterpart. I suggest this correspondence merely as a confirmation of the merits of my simulation, as comparisons between real and simulated imagery can suffer from subjectivity. However, the similarities suggest that my simulation is particularly suited for modeling channels in homogeneous substrates similar

to this hill. For applications that require modeling more varied terrains with different soil properties, the initial conditions for my simulations would have to be be augmented with zone information.

My terrain modeling method of Section 4.3.4 combines my hydraulic simulation with a method for generating initial conditions for it. I justify the latter addition as a necessity for modeling real terrains. Since I incorporate a step for specifying a mask for controlling the initial distribution of heights into the algorithm, it becomes possible to apply the hydraulic erosion simulation to a variety of topologies. This is illustrated in Figure 4.17.

To validate the generated terrains themselves, I analyze their spectral density and hypsometry. I confirm that the spectral density is close to a power falloff characteristic of affine fractal scaling, which is typical for my use of the avalanching paradigm. In some cases, the highest frequencies do not fit the pattern by either being filtered out or instead becoming too pronounced. However, I consider such violations acceptable, because they tend to happen when there is too much diffusion in the erosion or it creates many small-scale channels, respectively. The lowest frequencies can also break the falloff pattern, but they are not significant, as they capture the distribution of heights in the terrain at the coarsest level and can be skewed easily. For example, the central value on the spectral density plots corresponds to the average height of the terrain, which can be made arbitrarily large by changing the reference point relative to which the heights are measured.

As for the hypsometric curves of the terrains I generate, they tend to be qualitatively similar to the equilibrium and monadnock hypsometric forms. It makes sense that the inequilibrium stage, which may not exhibit distinct erosion features, is not typical for my terrains. The concavity in the upper part of the terrains' hypsometric curves results from a property of my algorithm for generating initial terrains via level sets. For a hill, the area above a level set drops off approximately quadratically. However, level sets with large perimeters tend not to use all of their available area, increasing the area above a given level set. Typically, this effect straightens out the hypsometric curve for heights in the middle of the range. Figure 4.21 shows the spectral density and hypsometry for terrains of Figure 4.17.

Some of the hypsometric curves in Figure 4.17 contain nearly horizontal segments in the parts of the plots that correspond to low heights. Such segments indicate the presence of nearly flat areas in the generated terrains, which arise because my erosion model tends to flatten a terrain to its minimum height (except for the effect of parameter w_0). In other words, the generated terrains are cut off at a minimum level. Alternatively, my model can be set up with more complicated boundary conditions similar to the terrains in Figure 4.15 to produce terrains that do not flatten and instead achieve equilibrium between uplift and

erosion. However, I have chosen not to tailor my model to the latter type of terrains, because they represent a special case of terrain evolution that takes place over especially long timescales[†].

[†]A shorter presentation of the material up to this point can be found in Computers and Graphics [51].

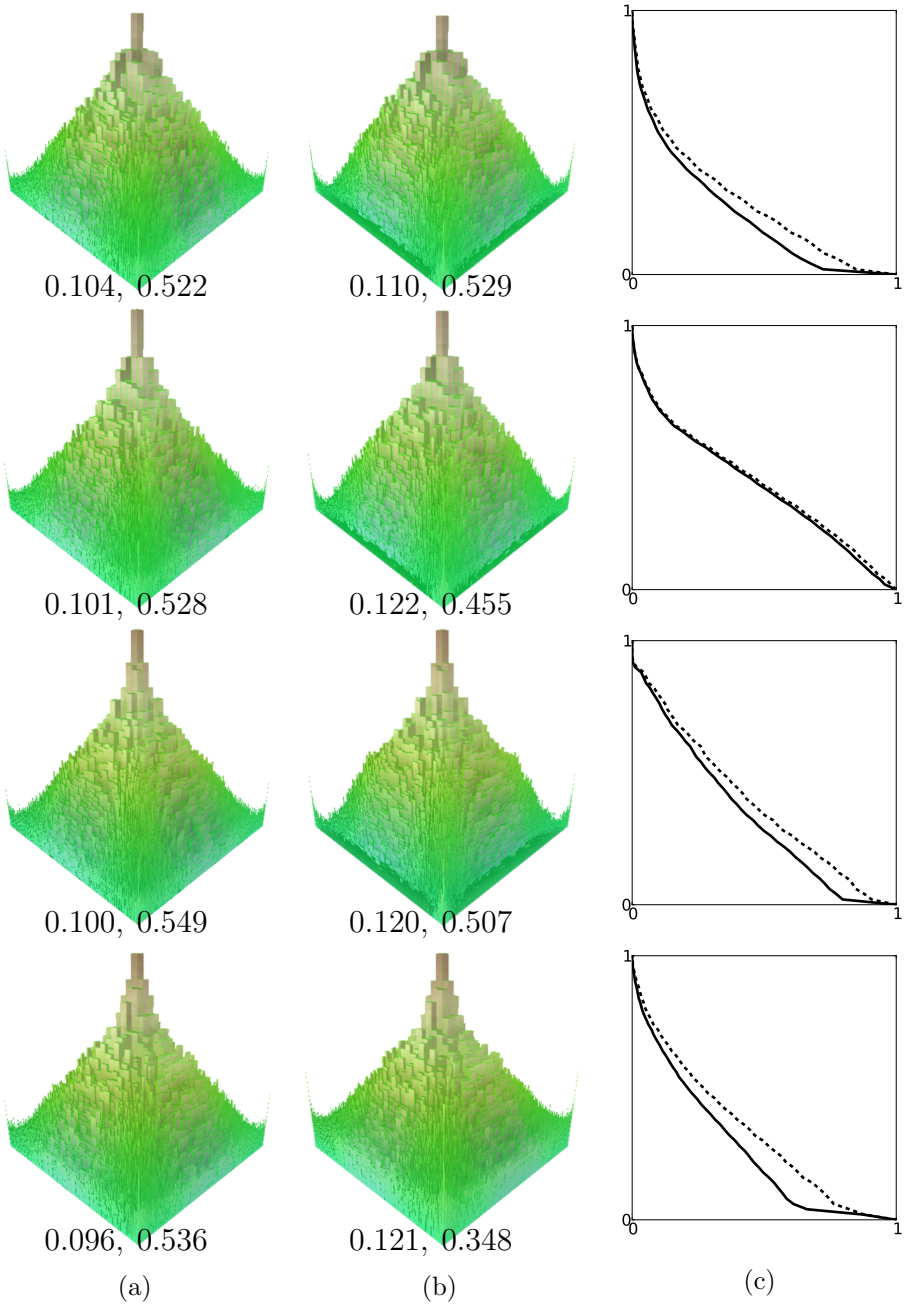


Figure 4.21: Hypsometry and spectral density of terrains from Figure 4.17: (a) spectral density before erosion, (b) spectral density after erosion, (c) hypsometric curve before and after erosion (final curve is heavier).

4.4.5 Terrain Analysis and Comparisons

I have made a combination of spectral and hypsometric analysis my procedure for evaluating terrains, because spectral analysis helps identify fractal character, while hypsometric analysis categorizes the effects of hydraulic erosion (when it is present). The two types of analysis supplement each other, because spectral density concerns the frequency domain properties of a terrain, while hypsometric analysis deals with the distribution of heights and areas in the spatial domain.

In this section, I apply my analysis procedure to several kinds of terrains, including natural terrain data sets and terrains created with proprietary procedural methods, to further illustrate each part of the evaluation procedure and its significance for modeling terrains procedurally. By applying the procedures to elevation data from southern Ontario, I confirm that natural terrains can possess spectral and hypsometric properties that I have aimed to achieve using my procedural modeling method. In the case of terrains synthesized using Terragen, the analysis shows that terrains can be indistinguishable in terms of their spectral density despite having different height distributions.

The examples in this section also serve as additional comparisons between my terrain modeling method of Section 4.3.4 and other existing methods of modeling terrains.

Terrains from Southern Ontario

I have selected four sets of elevation data from the Ontario Ministry of Natural Resources database for Southern Ontario [1]. Table 4.10 lists the geographic locations and extents of the datasets in order of increasing size. The pattern of elevations of the first two datasets is relatively extreme: the first one is a deep gorge (featuring a 41 meter waterfall) and the second one contains an unusually shaped depression. The remaining two datasets describe river valleys that I believe are likely to be representative of the majority of the surface of Ontario.

Figure 4.22 provides several types of visualizations of the datasets produced using ArcGIS software `10.2.2` for Desktop Advanced. The software suite includes ArcMap, which can be used to compile map-like representations of the data (first from the left in the figure), and ArcScene, which is used for 3D visualization (second from the left). The figure also includes the spectral and hypsometric plots created according to my analysis procedure.

The hypsometric curves of the first two datasets are mostly convex, confirming that the datasets are atypical for terrains undergoing hydraulic erosion (Section 2.4.1). The

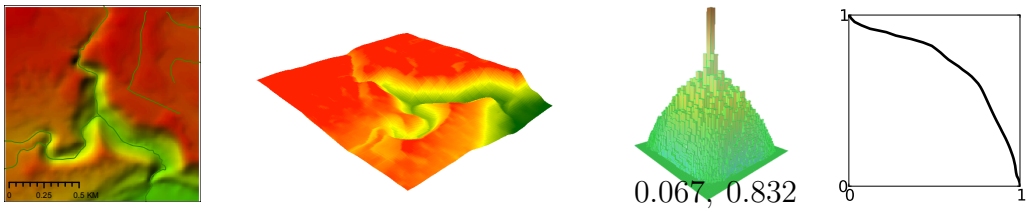
convex pattern can also be found in the terrain in Figure 4.12a, which is generated with my erosion model and a synthetic underlying terrain, and in the terrain in Figure 4.23c (below), which is generated with Terragen and the "canyonism" option. This suggests that terrains with convex hypsometry can arise easily both in procedural modeling and in the real world. However, in this type of terrains the effects of rock structure must predominate over the effect of hydraulic erosion, resulting in the creation of gorges and canyons in place of valleys.

The third and fourth dataset are composed predominantly of typical river valleys and exemplify the type of hydraulic erosion that I aimed to reproduce using my modeling method. Note that these two terrains possess spectral and hypsometric qualities that I have identified as desirable for procedural modeling. In particular, the hypsometric curve of the third terrain is of the monadnock type, while that of the fourth terrain is of the equilibrium type (Section 2.4.1).

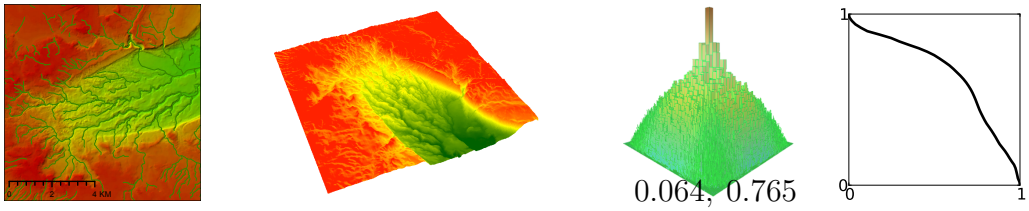
	extent ($km \times km$)	location
Webster's Falls	1.4×1.4	79°59′6″W and 43°16′19″N
Dundas Valley	9.17×9.17	80°3′3″W and 43°12′39″N
loc. near St. George	10.83×10.83	80°17′10″W and 43°11′0″N
Thames River	24.19×24.19	81°21′57″W and 43°7′56″N

Table 4.10: Geographic parameters of elevation datasets from Southern Ontario.

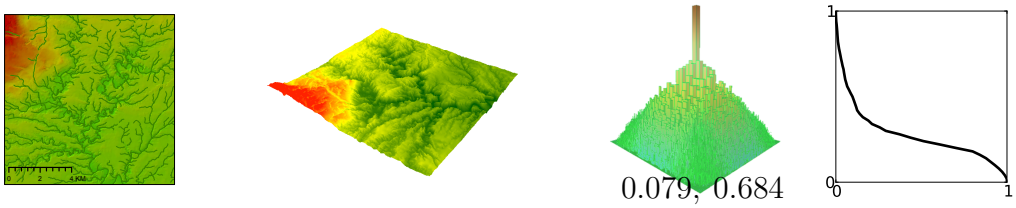
One way in which the natural terrains are different from the ones I have generated using my method (e.g., those in Figure 4.17 is that the river channels are deeply incised into the terrains where they cross the border of a dataset. In my model, erosion lowers the terrain to a local minimum within the terrain grid, as a consequence of Equation 4.4. The result is that a deep channel will flatten out if it reaches the border of the domain. That is why the hypsometric curves of the terrains in Figure 4.17 tend not to have a convexity in the lower part, but the curves of the terrains of the datasets in this section do. The easiest way to fix this problem is to expand the domain to be larger than necessary and cut out a part from the middle for analysis.



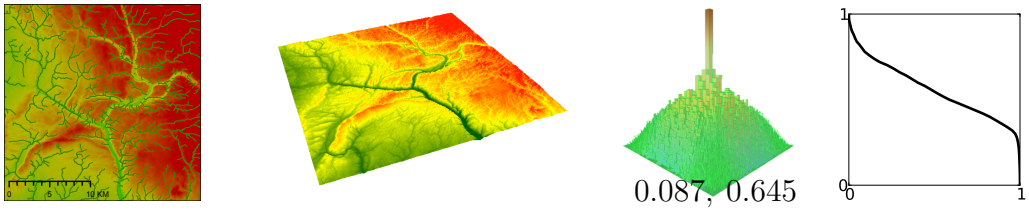
(a) Webster's Falls



(b) Dundas Valley



(c) location near St. George

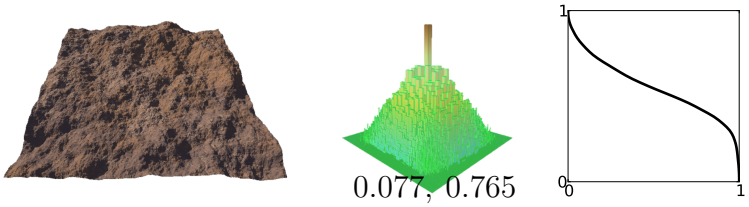


(d) Thames River

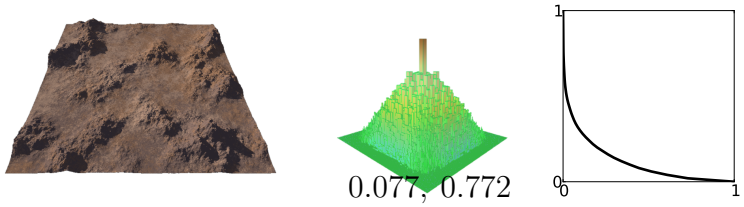
Figure 4.22: Analysis of elevation datasets from Southern Ontario. Left to right: map with waterlines, 3D visualization, log plot of spectral density, hypsometric curve.

Terragen Classic

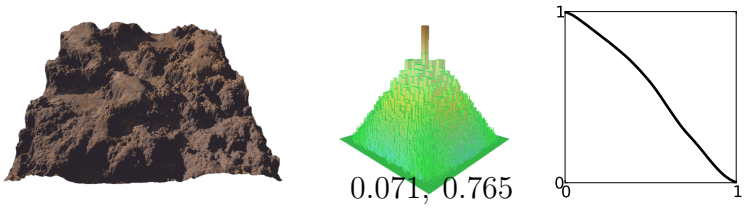
I have chosen to use Terragen Classic to generate a set of terrains for comparison with my method, because the software is extremely well-known in the graphics community. Conceptually, the program can be divided into two parts: a procedural modeling component for terrains and a rendering component that simulates such effects as atmospheric scattering. Since my primary concern is with procedural modeling, I have turned off advanced rendering effects in the visualizations that follow.



(a) Perlin



(b) Perlin with glaciation



(c) Perlin with canyonism

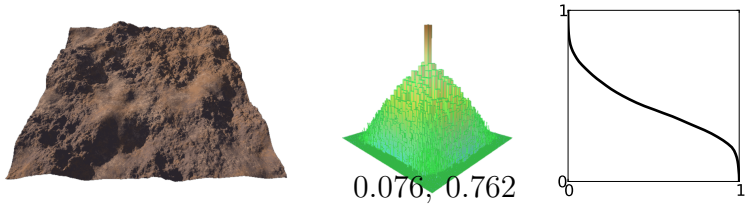
Figure 4.23: Analysis of terrains created with Terragen (part 1). Left to right: generated terrain, log plot of spectral density, hypsometric curve.

The procedural modeling component of the software consists primarily of explicit fractal synthesis algorithms, which are similar to the algorithms of Section 2.1.5. So analyzing terrains created with Terragen establishes a base of comparison between classical ontogenetic modeling and my method. Additionally, Terragen provides two options to modify

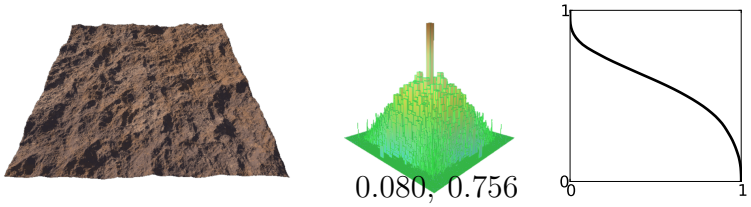
the results of "pure" fractal synthesis: glaciation, which simulates erosion in the low parts of the terrain, and canyonism, which has the opposite effect of increasing terrain elevation in high regions. Broadly speaking, the glaciation option produces results similar to large scale hydraulic erosion.

Application of my analysis procedures to terrains that are created with explicit fractal synthesis and modified according to the two options demonstrates how spectral and hypsometric properties of the terrains change in response to Terragen's simple model of erosion. Similarly, analysis of Terragen's multifractal terrains that exhibit spatially-varying roughness can also provide insight into the effects of erosion, which can locally smooth away rough features.

Figure 4.23 contains analysis results for three types of terrain: generated using Perlin noise (included for comparison with the others), generated with Perlin noise and modified using glaciation, and generated with Perlin noise and modified using canyonism. The first terrain is a simple approximation of fBm (see Figures 2.3 and 2.7). The effect of glaciation and canyonism on spectral density is negligible. This is most likely due to the modifications carrying out selective scaling of the terrain instead of flattening. However, the effect of the two options on hypsometry is profound and can be used to differentiate the results.



(a) multifractal Perlin



(b) ridged Perlin

Figure 4.24: Analysis of terrains created with Terragen (part 2). Left to right: generated terrain, log plot of spectral density, hypsometric curve.

Figure 4.24 contains analysis results for a multifractal Perlin terrain and a ridged Perlin terrain. The hypsometry and spectral character of these two terrains is similar to an

unmodified Perlin terrain. This is expected, because these types of fractals are typically used as slight improvements on the appearance of typical noise-based terrains: multifractal noise varies spatially and contains rougher and smoother regions, while noise with ridges attempts to differentiate valleys and peaks (they are mirrors of each other in typical noise terrains).

Erosion Model of Rodríguez-Iturbe

The erosion model of Rodríguez-Iturbe and Rinaldo [55] (discussed more in Sections 2.3.2 and 4.3) describes the formation of river networks and contains several limitations that prevent it from also being a method of procedural modeling of terrains. However, the model uses the area map concept in a way that is similar to how many terrain modeling methods operate (e.g., World Machine below). This is why I discuss the results of simulating the model of Rodríguez-Iturbe and Rinaldo here. My implementation of the model using my framework was straightforward and required a small amount of code, particularly because iterative computation of the area map is easily expressed using my paradigm of iteration with agents.

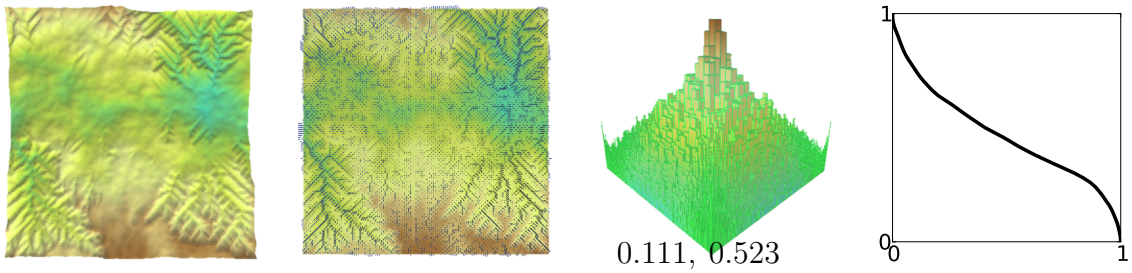
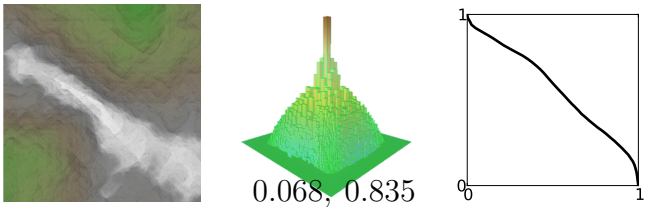


Figure 4.25: Erosion model of Rodríguez-Iturbe used for terrain modeling. Left to right: generated terrain, drainage area map, log plot of spectral density, hypsometric curve.

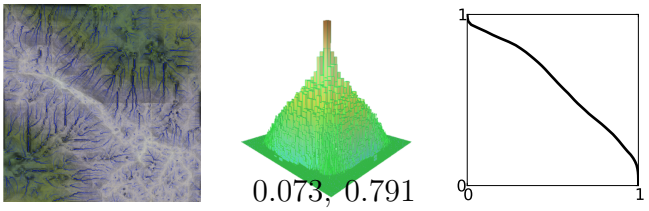
Figure 4.25 contains the result of simulating the model when applied to an fBm terrain, as well as a visualization of the area map and the results of spectral and hypsometric analysis. I have used an fBm terrain here, because the model makes no provisions for generating a starting terrain, which is one of its weaknesses that prevent it from being a procedural modeling method. A more serious drawback is that the rivers stop when they reach a local minimum. If this had not been the case, the rivers would flow down the hills at the top and bottom of the terrain and produce a substantial river flowing left or right. Additionally, the rivers are infinitesimally thin and do not create any lakes.

World Machine

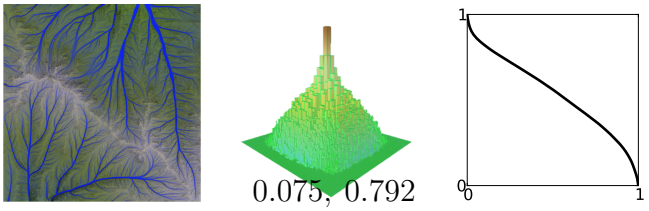
World Machine version `2.3` is a sophisticated terrain editor that combines several procedural modeling techniques with manual editing. The software can synthesize an infinite world using explicit fractal synthesis, as well as simulate erosion. The overall workflow follows a procedural paradigm based on a pipeline of modifications applied to a starting terrain. The manual component of the editing is carried out through placement of control curves and polygons (see Section 2.3.2 for a discussion of control curves in this context). Large scale river networks that span multiple terrain tiles are perhaps best modelled manually with control curves, as the procedural erosion in World Machine is applied separately to each tile. However, an optional post-processing step attempts to match the visual appearance of erosion features at tile borders using blending.



(a) ridged noise terrain



(b) terrain after channel erosion



(c) terrain after channel erosion with sedimentation

Figure 4.26: Analysis of terrains created with World Machine. Left to right: generated terrain (with flow lines in blue), log plot of spectral density, hypsometric curve.

Figure 4.26 shows the results of World Machine's erosion simulation. Note that the

starting terrain is also generated with World Machine (using preset parameters for ridged noise) and that it contains visible creasing (Sections 2.1.5 and 3.1.2). The presence of creasing and absence of high frequencies in the spectral plots suggest that World Machine is using a naive version of the random midpoint displacement method for fractal synthesis.

The channel erosion mode of the simulation displays a pattern of erosion that is extremely similar to that created according to the model of Rodríguez-Iturbe and Rinaldo (above) including the behavior of streams stopping once they reach a local minimum. World Machine solves this problem by simulating transport of sediment, but apparently without incorporating a true water flow simulation, which also would have solved the problem (water flow simulation is the approach I have taken in my modeling method). Without a water simulation World Machine can not produce any lakes based on simulated water flow.

The sedimentation simulation fills in local minima and allows the erosion simulation to produce visually pleasing flow networks, but at the expense of overly smoothing the terrain. The procedural pipeline capability of World Machine makes it possible to mask out the effect of sedimentation to preserve some sharp features, but overall the erosion simulation seems more suited to creating small rills on the sides of hills rather than river networks.

In conclusion, Terragen and World Machine contain many terrain modeling tools, but erosion features that the software can produce have limitations due to the lack of an explicit simulation of water flow like in my method. First, it is not possible to simulate a river flooding a depression and later draining out of the resulting lake. Second, the more extensive river networks produced using the sedimentation option in World Machine depend greatly on smoothing the terrain so that local minima are filled. In other words, World Machine's simulation depends on the assumption that the terrain erodes easily (or, alternatively, is subject to diffusion-dominated erosion (Figure 4.15a). So, erosion due to small, nearly parallel streams is the most general and realistic type of erosion that can be modeled using World Machine. In comparison, my method is more flexible and can produce all of the following features: erosion due to short parallel streams (Figure 4.15b), extensive river networks (Figure 4.12), and lakes (Figure 4.13).

In terms of my spectral analysis, I conclude that explicit fractal synthesis algorithms of Terragen and World Machine create approximations to fBm using noise summing or subdivision-based techniques. The resulting spectral densities tend to be less noisy than fBm and sometimes lack the highest frequencies. When I implemented my versions of these techniques, I observed similar results (Figure 2.6 and 2.7). The spectral densities of natural terrains that I analyzed in this section also tend to be less noisy than those of my

fBm experiment (Section 2.1.6). Terrains generated with my method, which is based on implicit fractal synthesis, are also slightly less noisy than fBm (Figure 4.21), but closer to fBm than the other fBm approximations or natural terrains.

It was not a primitive beginning or a slow evolution. It is as if the modern human soul has awakened here.

Cave of Forgotten Dreams
Werner Herzog

Chapter 5

Subterranean Channels

In this chapter I present my approach to modeling 3D channel networks that form underground due to dissolution (Sections 2.3.4 and 2.4.2). Although this modeling problem shares some similarities with modeling of hydraulic erosion on terrains (Section 4.3.1), there are no existing approaches based on procedural modeling and simulation that reproduce tributary capture behavior of subterranean channels, which is different in 3D due to the effect of pressure. I develop a simulation of this behavior by using my avalanching modeling paradigm to formulate a self-organized model of pressure and flow. Two additional components complete my procedural model of subterranean channel networks: first, an initial stage that simulates formation of protochannels and, second, a formulation of erosion that allows self-organized flow of the main model to create channels similarly to my simulations of 2D channels.

I use a grid of voxels as the discretization for my simulations of subterranean channel development. Each voxel element represents a unit of rock that can contain an amount of water corresponding to its porosity. Additionally, the voxel grid can use porosity to rep-

resent bedding planes and fractures in the rock matrix. When erosion increases porosity of the voxels inside channels, the faces of appropriate voxels become visualized via polygonization. Protochannel development creates initial channels beginning at voxels that contain sources of flow. Channel growth and linkage stage simulates flow from source voxels to sink voxels according to my self-organized model: flow concentrates into channels and causes erosion. Tributary capture can occur when a channel is able to drain the rock matrix in its vicinity and create a low pressure zone that redirects other channels.

Two aspects of my procedural model of channel development are particularly significant. First, the expressive power of my avalanching paradigm allows the model to reproduce a complicated pattern of tributary capture, called a phreatic loop, using emergent behavior. This is essential for simulating realistic channel development in 3D and so far has not been attempted in procedural modeling. Second, my model of pressure and flow provides for self-organization of pressure in a novel fashion by using semantics similar to avalanching in sand.

5.1 Challenges in Simulating Channel Behavior in 3D

Differences in problem domain representation and behavior of flow make it difficult to extend simulations of 2D channels on top of terrains to simulations of subterranean channels in 3D. In the 2D case, heightmaps serve as the discretization and flow is transmitted from higher to lower sites, causing erosion that lowers the surface of the terrain. So a high level model of 2D flow follows from the implied avalanching behavior that occurs when eroded sites receive more flow and cause more erosion in their vicinity. On the other hand, the growth of 3D channels is guided by fractures and permeability of the rock matrix, in which they are embedded. The 3D setting can be discretized using a grid of voxels, whose porosity values can represent both fractures and any other local variations in permeability. However, flow in the voxel grid does not necessarily concentrate in voxels with low porosity and the feedback behavior due to erosion as in 2D does not occur in the same way.

Figure 5.1 illustrates the differences between discretization based on heightmaps and voxels (for 2D and 3D channel simulations, respectively) in terms of transmission of water between neighbors that can occur during one iteration of a simulation. Water columns that represent the amount of water at a terrain site can have unlimited size, while the capacity of voxels is fixed. This constraint restricts which flow updates can be made. Assuming that the flow should occur in a "downhill" direction as in the figure, higher neighbors can always transmit water in the 2D case, but flow between voxels can encounter bottlenecks.

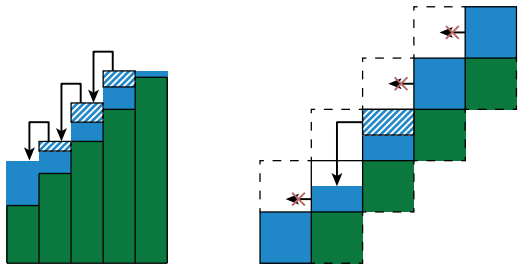
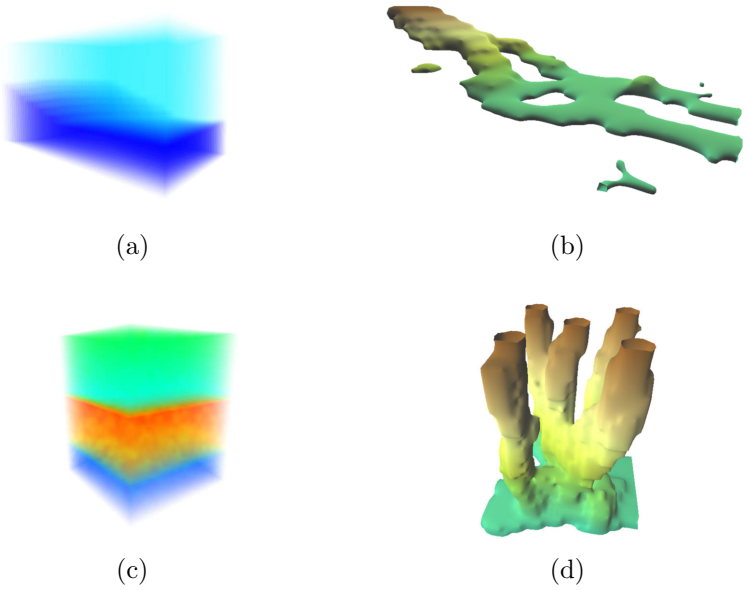


Figure 5.1: Differences in problem domain representation between 2D and 3D channel simulations: (a) flow based on water columns, (b) flow based on voxels.

Since the 3D simulation performs less work in one iteration on average, it is slower even without accounting for the faster asymptotic growth of the number of grid units in 3D.

The main challenge in developing a simulation of channel behavior in 3D is the need for a new model of water flow that can account for flow not only between higher and lower sites like in terrain simulations, but also according to a pressure gradient that may be in a vertical direction (Section 2.4.2). To motivate the formulation of a flow model suitable for 3D channel networks, I have adapted the water-column algorithm to use voxels similarly to water columns by separating the 26 neighbors of a voxel according to their relative vertical position. Any remaining capacity of the lower 9 neighbors is filled first, then the level of water in the 8 neighbors on the same level is equalized (Figure 4.11).

Simulation of erosion based on the above flow algorithm variant produces two types of channels, as shown in Figure 5.2. In the first example, the initial conditions consist of a single source and a rock matrix of variable permeability that contains an impermeable half-ramp half-block. Porosity is visualized as a range of colors, such that blue corresponds to lowest porosity and red to maximum porosity. The water from the source slides down the ramp as the model of water flow requires water to be transported to lower neighbors. In the second example, the initial conditions consist of five sources and several layers of rock with different porosity. The flow of water is in the vertical direction. Note that both examples exhibit flow concentration that guides channels to link up, as in simulation of water flow in terrains. However, water always falls through the rock matrix until it reaches a sink or an impermeable layer, which is equivalent to a terrain discretized using voxels. This limitation shows that a straightforward extension of the water-column algorithm is not a valid generalization to 3D as it can not express flow behavior that is substantially different from that on terrains. The fundamental limitation of the algorithm is that it



(a) (b)

(c) (d)

Figure 5.2: Generalization of water simulation from 2D to 3D: (a) half-ramp half-block, (b) channels produced by 1 source, (c) layered block, (d) channels produced by 5 sources.

compares heights of neighbors, which implies that transport of water occurs relative to a horizontal plane.

5.2 Model and Simulation

According to hydrogeomorphologic principles of channel development via dissolution (Section 2.4.2), I distinguish two stages of channel development: the protochannel stage and the channel growth stage. I make the simplifying assumption that protochannels and channels do not develop simultaneously (in different parts of the problem domain), although they can coexist until protochannels are eroded or enlarged. Therefore, I develop separate models for the two stages and simulate them in sequence. My main contributions are the model and simulation of the channel growth stage.

5.2.1 Protochannel Stage

The goal of the protochannel stage is to create physically-motivated initial conditions for the channel growth stage in the form of vestigial channels that grow out of each source, elongate according to rock permeability and pressure differentials, and have not yet broken through to a sink. As protochannels grow they compete for space by pressurizing the pores of the rock in their vicinity, slowing down the development of affected protochannels (Figure 2.16a). The first part of my model for protochannels accounts for this competition by letting each protochannel's source claim an area according to its pressure.

The second part of my model details the behavior of the protochannels as they grow. Since I am assuming that the breakthrough has not occurred yet, the sinks are inactive and the flow through the rock matrix is slow compared to the post-breakthrough flow. Additionally, the pre-breakthrough flow is further constrained, because each source has created a pressure zone of water-filled pores, which have to be displaced for flow to occur. I assume that displacement of water in the pores happens extremely slowly due to drag along the surface of the pores, so that pressure of interacting pressure zones can become nearly equalized. Therefore, only local changes in porosity and a small local gradient of flow affects the growth of each channel. I simulate the presence of an idealized small flow gradient using a set of protosinks on the boundaries of each source's pressure zone. In the isotropic case, this causes the protochannels to branch out in all directions. Clustering the protosinks results in protochannels that grow in a preferred direction.

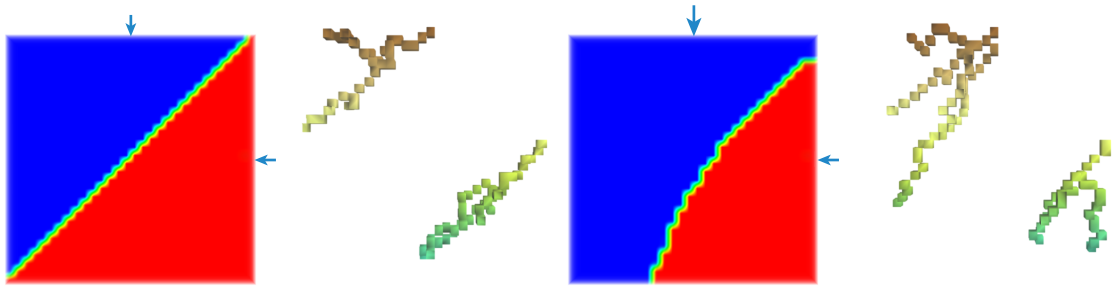


Figure 5.3: Pressure zones in protochannel development: (a)–(b) two sources with equal pressure, (c)–(d) pressure of one source is higher.

I implement the first part of my protochannel model by using fast marching (Appendix A.5) to approximate the extent of each pressure zone according to distance to closest source weighted by pressure. Using distance this way simulates the effect of a

larger pressure capturing more area before equalization, as illustrated in Figure 5.3. The increase in area allows the protochannel system of the source with higher pressure to also become larger. The following pseudocode outlines the necessary change to the fast marching algorithm (corresponding to line 18 of Listing A.6). The variable `max_pressure` is the largest pressure of protochannel sources.

Listing 5.1: Distance update in protochannel model.

```
1 // recompute distance for neighbors of minimal site P
2 for each neighbor N of P {
3     weight = max_pressure / P.source.pressure
4     new_distance = weight * distance(N, P.source)
5     if (new_distance < N.current_distance) {
6         update distance }
7 }
```

I implement the second part of my protochannel model using a variant of the Ford-Fulkerson flow network algorithm, which computes maximum flow in a network subject to capacity constraints [10]. I have chosen this algorithm, because according to my model the flow in protochannels is primarily affected by porosity, which acts like a capacity constraint. Additionally, the neighbor adjacency relationship in the voxel discretization extends the grid of voxels into a graph, similar to the graph-like representation of terrain grids in my framework. The flow network algorithm assigns maximal flow to a set of nodes lying between a source and a set of protosinks, whose location represents the direction of protochannel growth. Following the path of maximal flow from a source creates a single protochannel. I have found that to create a number of branching protochannels it is better to follow paths of maximal flow from the sinks instead. Otherwise, it is difficult to judge when branches should separate and the protochannels clump into a mass.

5.2.2 Channel Growth and Linkage Stage

My model for the channel growth stage describes the behavior of channels after breakthrough, as well as how they self-organize to form a channel network. Figures 2.15 and 2.16 suggest that channel development is guided by pressure differentials that each channel creates in its vicinity. To formulate a complete model suitable for procedural modeling, I detail three aspects of channel development: how pressure responds to erosion, how flow responds to pressure, and how flow causes erosion. The connection between flow and erosion is similar to the formulation that I use to simulate hydraulic erosion on terrains (Section 4.3.1). However, the effect of pressure requires a completely new model that combines pressure and flow, which I have developed using the avalanching paradigm.

Figure 5.4 illustrates how pressure responds to erosion in my model. In Figure 5.4a two channels develop side by side at a similar level of pressure, until one of them undergoes substantial erosion (e.g., breakthrough). Erosion causes a reduction in pressure as the rock matrix drains of water (Figure 5.4b). If the sharp jump in pressure between the lowered pressure and its original level in the second channel persists, then the rock between the two channels must have an infinite resistance to flow and the channels will never interact.

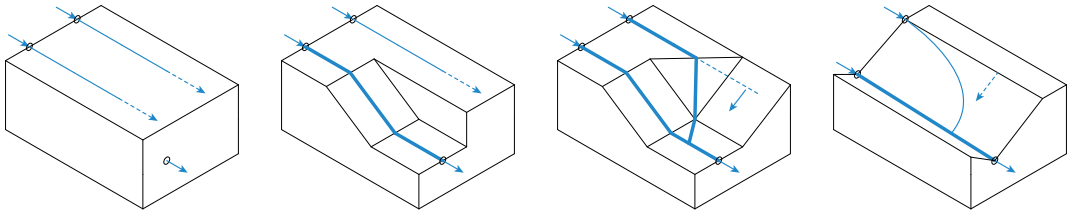


Figure 5.4: Model for interaction between channels: (a)–(c) pressure drops around one channel causing redirection of another channel; (d) more realistic pressure drop off around the main channel.

It is more realistic to consider the "wall" of rock between the channels to be penetrable, similarly to the rest of the rock carrying the channels. In this case, I assume that the resistance of the rock is such that pressure rises linearly with its thickness, leading to an adjustment in pressure between the two channels shown in Figure 5.4c. The adjustment in pressure can be interpreted in terms of a maximal rate of growth in pressure that the rock can support, so that excess pressure forces water out of the pores until the pressure is reduced. This behavior is similar to avalanching of sand grains on a slope that is steeper than the slope of repose (Section 2.2). In my model, I use two angles to differentiate the rate of growth in pressure inside channels and between them, as shown in Figure 5.4d. Reduction in pressure such that the pressure gradient points away from a channel is the main factor leading to linkage of channels in my model.

I implement the part of my model concerning pressure by adapting the fast marching algorithm (Appendix A.5), which is ideally suited to simulating propagation of fronts. When erosion causes a reduction in pressure at a given site, pressure gradient at nearby locations may become oversteepened and an avalanche of changes may radiate outward, like a front. The following pseudocode outlines the necessary changes to the fast marching algorithm (corresponding to line 18 of Listing A.6). Note that I separate pressure from T, so that pressure values can be initialized to those computed in the protochannel stage of my

model while T controls the order of site discovery. I discuss an alternative implementation in Section 5.4.1.

Listing 5.2: Pressure update in channel growth model.

```
1  // recompute pressure and T for neighbors of minimal site S
2  for each neighbor N of S {
3      choose applicable slope k
4      new_pressure = S.pressure +  k * distance(N, S)
5      if (new_pressure < N.current_pressure) {
6          update pressure
7          N.T = 0}
8  }
```

To have flow respond to pressure in my model, I use a water transport simulation similar to the generalization of the water-column algorithm discussed in Section 5.1. However, this time I use a distribution strategy based on a weighted average, because of the increased number of neighbors compared to the 2D case (Section 4.4.3). Unlike equalization of water relative to the horizontal, this distribution scheme also avoids defining a preferred direction and can represent flows between any set of neighbors (subject to the weighting). The following pseudocode, which is similar to Listing 4.2, provides the details.

Listing 5.3: Voxel-based water transport algorithm.

```
1  for each neighbor N of S {
2      if (N.capacity > N.contents && N.pressure < S.pressure) {
3          N participates in distribution}
4  }
5
6  distributed = 0
7
8  for each neighbor N of S {
9      if (N participates) {
10         update = weight(pressure difference) * (N.capacity - N.contents)
11         update -= max(distributed + update - S.contents, 0)
12         distributed += update
13         N.contents += update }
14 }
15 S.contents -= distributed
```

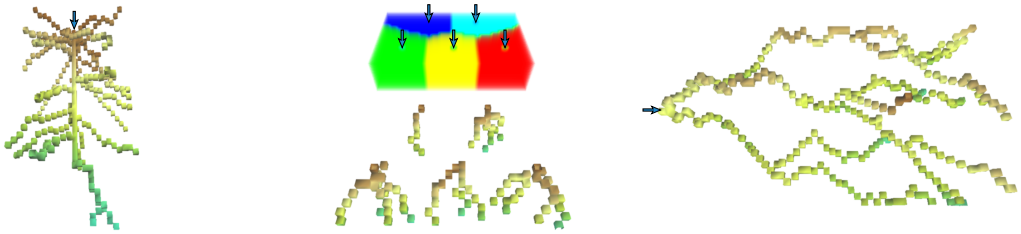
Finally, I formulate the relationship between flow and erosion similarly to hydraulic erosion on terrains, but with some simplifications. First, I calculate a flow value f from central and contributed flows, f_a and f_c, respectively, analogously to Equation 4.5:

$$f = (w_a f_a + f_c)^m. \tag{5.1}$$

Second, I use f to determine the corresponding amount of erosion in terms of the increase in porosity P. The following equation is a simplified form of Equation 4.4, in which I dispense with terms containing derivatives of P, because my discretization is coarse (i.e., each voxel contains only one sample). I do not incorporate terms based on the pressure gradient, which may also affect erosion, because my formulation of flow already results in more flow to neighbors with a greater difference in pressure.

$$\frac{\partial P}{\partial t} = \min(w_0 f, 1.0 - P). \tag{5.2}$$

5.3 Results

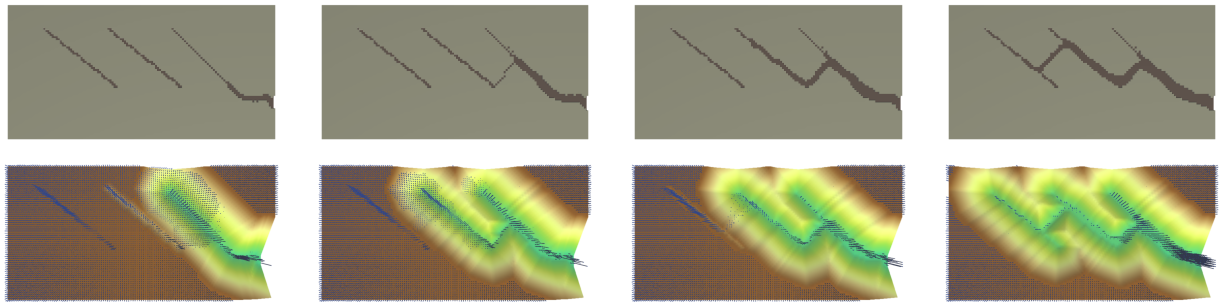


(a) (b) (c)

Figure 5.5: Protochannel simulation results. Sources are marked with arrows.

The protochannel stage of my simulation can construct several types of protochannel systems exemplified in Figure 5.5. Figure 5.5a shows protochannels that branch from a central "core" similar to experiments of McDuff et al. [36] (Section 2.4.2). The development of protochannels in Figure 5.5b encounters obstacles due to pressure zones (shown in different colors) of nearby inputs, according to a configuration with multiple ranks of inputs. Figure 5.5c contains a longer system of protochannels.

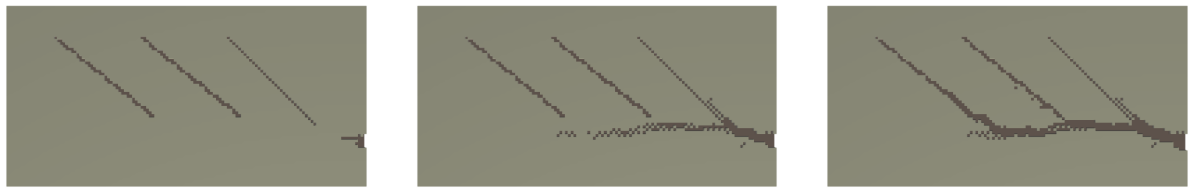
To illustrate the behavior of my simulation of the channel growth stage, I first provide examples using a 2D version of the simulation, which makes it easier to visualize pressure, flow, and the forming channels simultaneously. The top row of Figure 5.6 contains a sequence of snapshots of developing channels, while the bottom row contains the corresponding flow (blue bars) and pressure surface (visualized as a terrain with brown color mapping to the highest value). The initial conditions for this simulation are three protochannels constructed synthetically. There are three sources, which are located at



(a) (b) (c) (d)

Figure 5.6: Channel growth simulation in 2D. Top row: forming channels; bottom row: pressure and flow.

the tops of the protochannels, and one sink, which is located along the right boundary of the domain. Note that channel growth causes them to link up in a sequence by forming phreatic loop patterns (Section 2.4.2). This behavior is possible due to avalanching of pressure and flow that causes redirection of flow from one channel towards another.

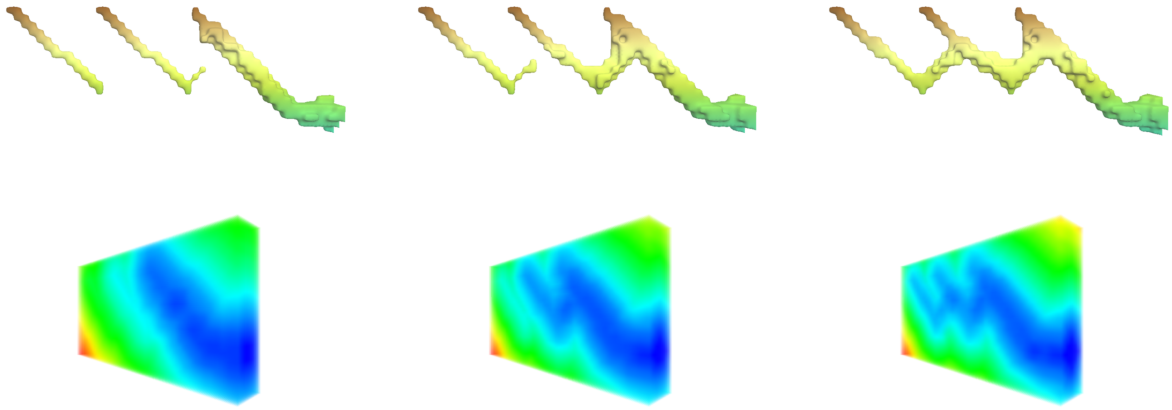


(a) (b) (c)

Figure 5.7: Alternative mode of channel growth (in 2D) without formation of phreatic loops.

Figure 5.7 demonstrates that my model of channel growth captures the alternative mode of channel development that does not result in formation of phreatic loops (Section 2.4.2). The alternative behavior occurs when the resistance of the rock matrix is low and the protochannels are able to reach the sink without first going through a neighbor. In my model resistance to flow corresponds to growth in pressure with rock thickness, which is different for rock with and without a present channel. Therefore, to specify low resistance in my model it is necessary to set the pressure growth rate for channel-free rock to be low. So the configuration of channels in Figure 5.7 emerges when the rates are set to 0.0025 :

0.001 (for rock without and with channels, respectively), as compared to **0.05** : 0.001 for Figure 5.6. However, the condition that the resistance of the rock matrix without channels is low also implies that development towards a sink will be preferable over linking up with a neighboring channel. My model supports this interpretation and the result of the simulation remains the same as long as the ratio of the rates is not changed. For example, the same channels emerge for rates set to 0.05 : **0.02**.



(a) (b) (c)

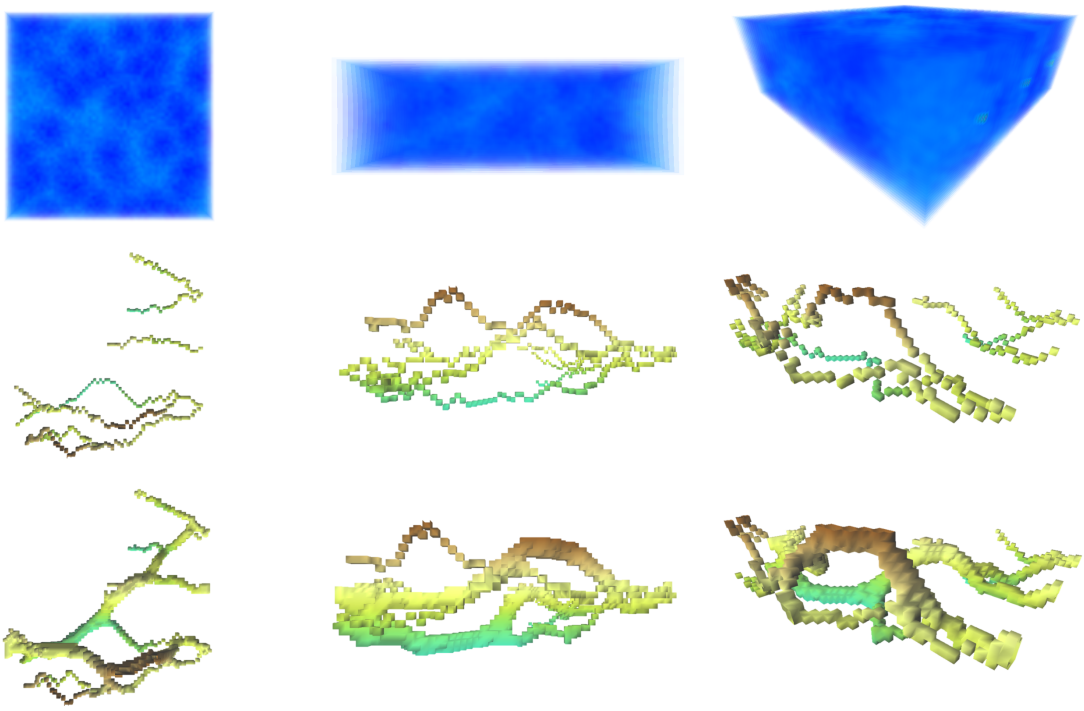
Figure 5.8: Channel growth simulation in 3D. Top row: forming channels; bottom row: pressure visualization.

Figure 5.8 shows the same pattern of channels as in Figure 5.6, but constructed using the 3D version of my simulation. Each snapshot of the simulation is accompanied by a volumetric visualization of pressure with blue and red corresponding to low and high pressure, respectively. I use my own polygonization algorithm to create the surface of the channels, which is defined in terms of the change in porosity between neighboring voxels (Appendix A.3).

5.3.1 Caves Created Using Two-Stage Simulation

I use my two-stage simulation of 3D channels to model some realistic caves. The complete process involves three steps. First, I create a set of initial conditions in the form of a voxel volume representing rock of varying porosity and containing several cells marked as

sources. Second, I apply my protochannel simulation, which is the first simulation stage. Third, I mark some cells as sinks and run the channel growth simulation, which is the second simulation stage. In the following figures, the result of the first stage is shown with additional erosion first, in order for the protochannels to be visible. In the second stage the porosity of the formed protochannels is reset to their original values, which are lower than the threshold for the empty test, so that the new channels become visible and grow due to the action of the channel erosion simulation only.

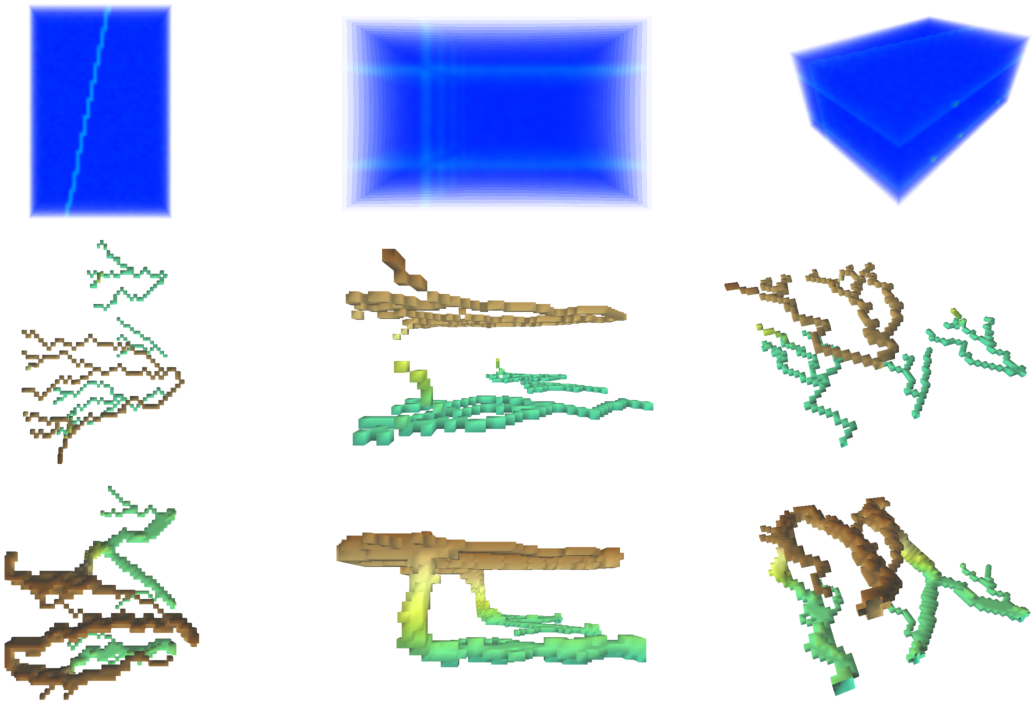


(a) top (b) side (c) angled

Figure 5.9: Cave with three sources that form channels towards a single sink area.

The cave shown in Figure 5.9 contains three sources that, in the first simulation stage, produce protochannels that branch away from the source locations. The second stage replaces the protochannels with larger channels that converge to a single sink area, which is composed of multiple cells. The channels corresponding to sources that are farther from the sink area are able to reach it due to being captured as tributaries. The initial porosity matrix contains spherical hill-like regions in which porosity decreases towards the center. These obstacles cause both types of channels to meander, producing several noticeable

arches (visible in the side and perspective views).

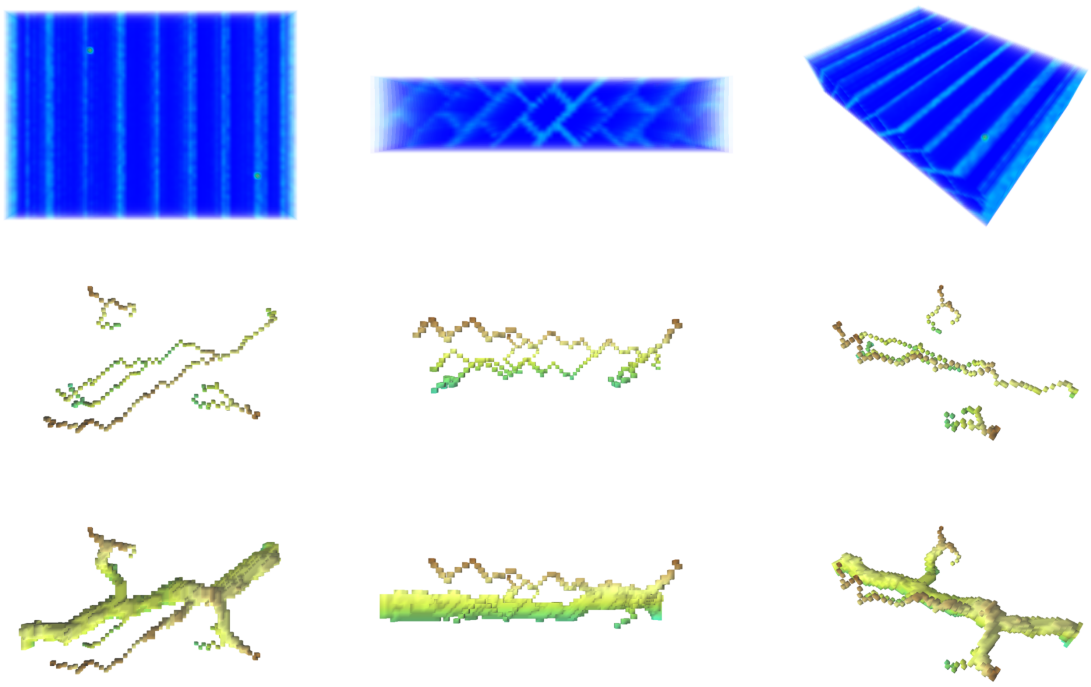


(a) top (b) side (c) angled

Figure 5.10: Cave with four sources, two sink areas, and a pattern of perpendicular fractures.

The cave shown in Figure 5.10 contains four sources and two sink areas. It develops according to the presence of two horizontal bedding planes and a perpendicular fracture plane, which cause both protochannels and cave passages to stay relatively planar as they follow the pattern of fractures. In particular, there are two nearly vertical passages that coincide with the location of the vertical fracture plane.

The initial conditions for the cave shown in Figure 5.11 represent a configuration of bedding planes and fractures similar to the rock matrix in Figure 5.10, but with a denser pattern of fractures and the bedding planes being inclined. The cave of Figure 5.11 forms primarily due to the action of a single source area (i.e., several neighboring voxels introducing water into the system), which is paired with a sink area in the opposite wall of the simulated volume. The result is a cave passage that is wider than those in the previous

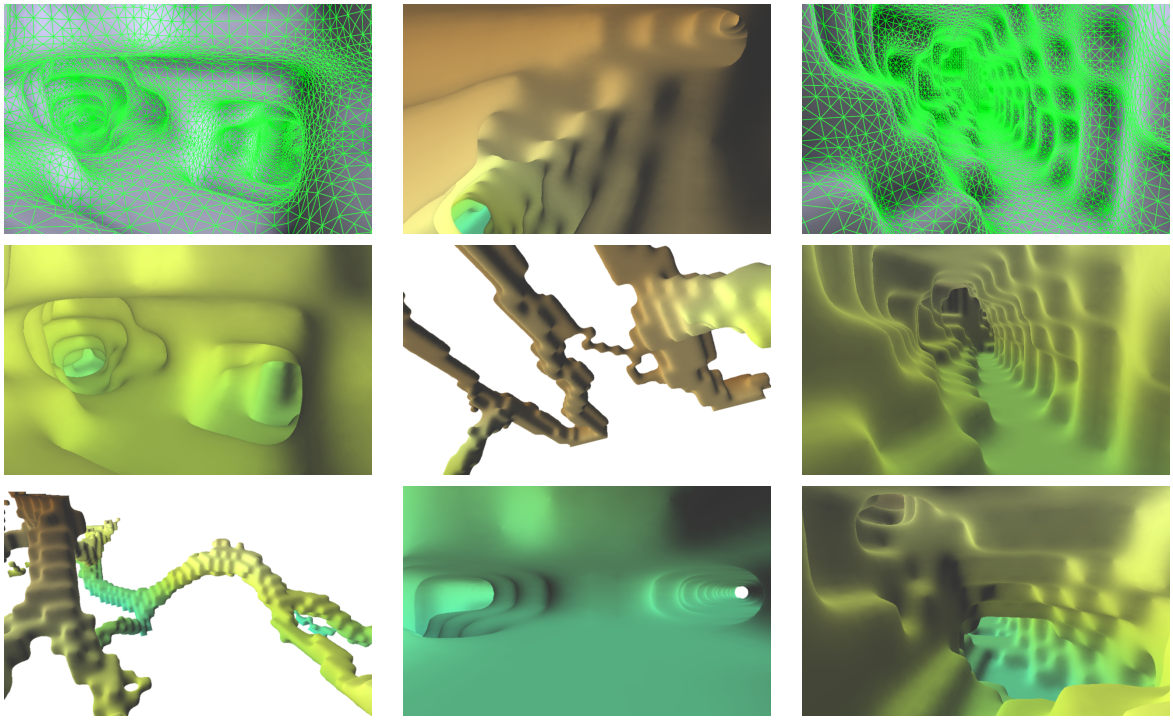


(a) top (b) side (c) angled

Figure 5.11: Cave with one source area, one sink area, two weaker sources, and a pattern of fractures and bedding planes.

figures. Additionally, there are two weak sources at the top of the simulated volume that produce tributaries of the main channel. In nature, the large underground source could be originating at a nearby lake, while the two weak sources could result from water collecting at the ground level and entering through the inclined bedding planes, which are exposed at the surface.

Figures 5.9–5.11 contain entire sets of cave passages, which may obscure each other. To better show places where individual passages meet, Figure 5.12 provides some close-up screenshots. The polygonizations of the cave surfaces in this set of images are adaptively subdivided (Section 2.1.5 and Appendix A.3) to further improve visualization. For example, the walls of a voxelized cave passage may contain two voxels that stick out into the passage interior and touch diagonally. In this case, subdivision of the polygonization pulls the surfaces of the voxels apart, which is a more plausible interpretation of the voxel data, because a cave passage should not have sharp cubical shapes obstructing it.



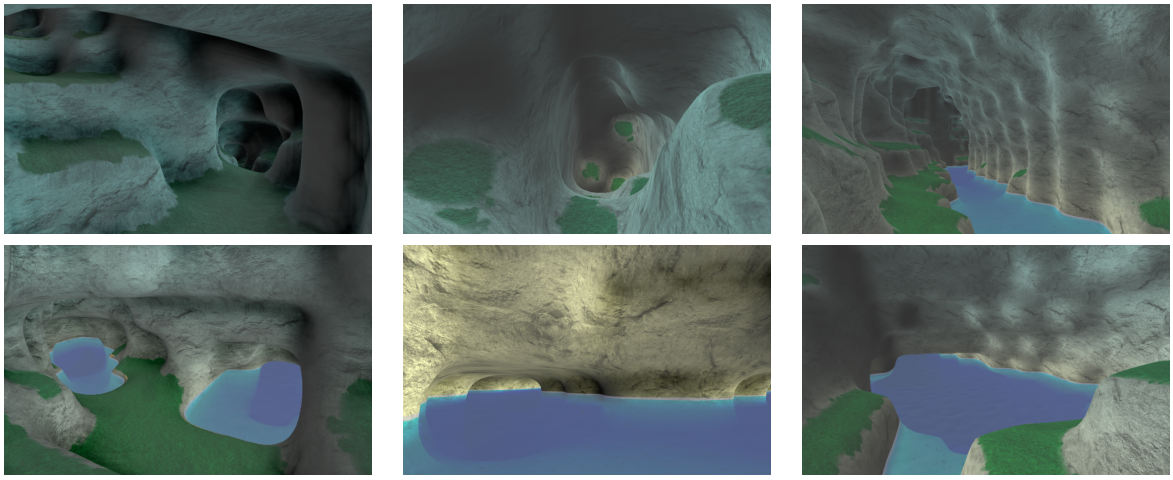
(a) Figure 5.9 (b) Figure 5.10 (c) Figure 5.11

Figure 5.12: Additional cave screenshots with subdivision.

Table 5.1 lists the simulation parameters for caves in Figures 5.9–5.11. The ratio of the rates of pressure growth outside and inside of channels ($P_{rate} : P_{rate}^{channel}$) is higher in this set of simulations than in the simulations of Figures 5.6 and 5.7, because the resolution of the voxel simulations is lower, while the channels should drain roughly proportional regions. Section 5.4.1 provides more information about the grid sizes of the cave simulations. The parameters w_0, w_a, and m are used in Equations 5.1 and 5.2, which determine the effect of flow on erosion in a similar way to the model of hydraulic erosion from Section 4.3.1.

Figure	P_{rate}	$P_{rate}^{channel}$	w_0	w_a	m
5.9	0.50	0.001	0.15	0.25	1.0
5.10	0.50	0.001	0.15	0.25	1.0
5.11	0.85	0.001	0.10	0.40	1.0

Table 5.1: Simulation parameters for modeled caves.



(a) Figure 5.9 (b) Figure 5.10 (c) Figure 5.11

Figure 5.13: Additional cave screenshots with textures and screenspace water.

Figure 5.13 shows a different type of visualization of the caves with texturing. The textures are applied procedurally based on slope and composited together based on masks that are generated using Perlin noise. In addition, some of the images feature a water surface, which is rendered in screenspace. The transparency and color of the water takes depth into account.

5.4 Discussion of Subterranean Channel Simulation

In this section I discuss the modeling process for my simulation of 3D channel networks and some of its similarities to simulations from Chapter 4. I also analyze the caves that I have generated with my simulation.

5.4.1 Use of Avalanching and Framework

As the result of my modeling process, my procedural model of subterranean channel development illustrates the benefits of avalanching as a modeling paradigm. Most importantly, avalanching puts an emphasis on abstraction of erosion mechanics, which makes it possible to use similar formulations of self-organized behavior in different contexts. For example,

avalanching behavior of erosion in my simulations of river-like and subterranean channels allows both types of channels to develop plausible width in analogous ways. Similarly, my conceptual model of self-organized pressure borrows ideas from avalanching in sand. In other words, the abstract nature of self-organization according to avalanching makes for an expressive modeling template. Although avalanching does not explicitly provide for the emergence of such a specific pattern of erosion as the phreatic loop, nevertheless avalanching does encode the emergent behavior of flow and pressure leading to its formation.

To accomodate the subterranean channel simulation I have added two new facilities to my common procedural modeling framework: fast marching (Appendix A.5) and voxel thinning (Appendix A.4). The main use of fast marching is to re-calculate pressure at each iteration of the simulation by making updates in a pattern similar to a front propagating through the problem domain. I have chosen fast marching, because it performs the updates in an optimal way using a heap that contains only a subset of all possible nodes in the domain. However, the agent-centric structure of my framework suggests an alternative way to express front propagation.

The alternative algorithm, outlined in pseudocode below, structures the state updates similarly to breadth-first search and visits all nodes in the domain to make each round of updates (like the majority of computation in the framework). Each iteration of the alternative algorithm is much more expensive than fast marching, because state updates occur only near the propagating front and visiting any other locations is a waste. However, the alternative method has two benefits: first, it dispenses with the heap, which makes it more amenable to parallelization, and second, it can lower pressure gradually in the loops in line 4 and 17 according to a function of time, which is the way I first implemented it. Although the two algorithms are similar in some ways, they illustrate two conceptually different ways to perform iteration within my framework that are useful in different contexts.

Listing 5.4: Computation of pressure using front propagation between neighbors.

```
1  initialize by flagging sink sites as DOMAIN | FRONT
2
3  while front can be advanced {
4      visit agents {
5      if agent flagged FRONT {
6          update pressure using neighbor with lowest pressure }
7      }
8      visit agents {
9          if agent flagged FRONT {
10             for each neighbor {
11                 if neighbor violates pressure constraint,
```

```
12                      not flagged DOMAIN and not flagged TRANSITION {
13                  flag neighbor TRANSITION }
14              }
15          }
16      }
17      visit agents {
18          if agent flagged FRONT {
19              if does not exist neighbor not flagged DOMAIN and not
                    flagged TRANSITION {
20                  unflag agent FRONT }
21          } else if agent flagged TRANSITION {
22              unflag agent TRANSITION
23              flag agent DOMAIN | FRONT
24          }
25      }
26  }
```

Performance

I have chosen to use a uniform voxel grid data structure for implementation of my 3D erosion simulations, so that they are similar to my simulations of erosions on terrains in terms of such computational primitives as visiting and updating neighbors. However, the growth of storage and computation costs for the voxel-based 3D domain representation can be mitigated with an adaptive space partitioning scheme, such as an octree. For example, the simulation in Figure 5.6 indicates that there are sites whose values are never updated, because no channels develop in their vicinity. Therefore, large regions of unaffected sites can be replaced with a single large site, that can be subdivided only if erosion makes it necessary.

Although it would be beneficial for simulation of larger domains, I did not attempt to enhance my simulations in this way for two reasons. First, the benefit of an adaptive space partition is not as great at coarse levels of discretization due to the higher cost of accessing neighbors and additional maintenance of the data structure. Second, changes to the discretization scheme would also have to be reflected in the MAS component of my simulation framework, so that the complex data structures would make the basic principles of my modeling methodology less clear. Nevertheless, in the future I would like to extend my modeling framework in several respects, especially its capability for modeling of large worlds (Section 6.3).

Figure	grid size	stage 1	stage 2	
		runtime	iterations	runtime
5.9	60×60×20	1m 43s	240	22m 30s
5.10	40×60×25	1m 3s	180	9m 39s
5.11	70×50×15	4m 6s	180	12m 41s

Table 5.2: Performance of cave simulations.

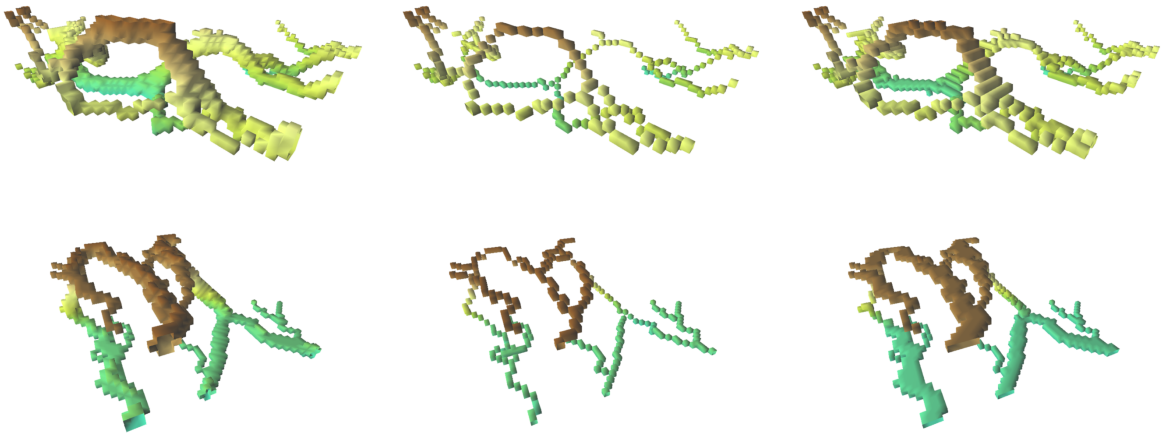
5.4.2 Evaluation of Results

Morphometric indices (Section 2.4.3) are a natural choice for analyzing procedurally generated caves, as they have been developed for the purpose of comparisons between caves. Futhermore, natural caves that have been studied can be classified into types according to their morphometric indices, suggesting that the indices are well-chosen parameters of shape. So comparison of the indices of generated caves to the value ranges in Tables 2.1 and 2.2 is a test that determines not only whether the generated caves possess some properties of real caves in general, but also whether the properties taken in combination fit a particular type of cave.

To apply the test, it is necessary to interpret the size of the generated caves in physical units, as some of the morphometric indices are not dimensionless. Computing W (average passage width) for a procedural cave yields a value in units of cell size that corresponds to the value of W in meters for the appropriate cave category. Comparison of the remaining indices can then rely on the interpretation of cell size based on W and also confirm that the unit conversion is sensible. I have used this procedure to define cell size to be 1 meter for my caves.

I compute the morphometric indices in several steps, using thinning (Appendix A.4). First, I apply a standard version of the thinning algorithm to the voxels of the cave, yielding a skeleton of the cave in the form of voxels that lie approximately in the center of the cave's passages. I reason that traveling along the skeleton voxels is approximately equivalent to moving along the survey centerline of a real cave. Therefore, I compute the length of the cave by summing up length contributions from each voxel of the skeleton in such a way that diagonally touching voxels contribute a length of $\sqrt{2}$ or $\sqrt{3}$, as appropriate. Figure 5.14 (center column) shows the voxel skeleton for some example caves.

Second, I apply thinning to the original voxels of the cave again, using only subiterations that remove voxels in the top and bottom directions. The result is a flattened version of the cave's passages that cuts approximately through their center. Note that voxels in this version of the skeleton do not have top or bottom neighbors. Similarly to computation of



(a) cave (b) skeleton (c) area skeleton

Figure 5.14: Thinning of voxelized caves used to compute morphometric indices.

length, I sum up area contributions from the voxels in the area skeleton. I use the neighbors of each voxel (in the 3×3×3 neighborhood) to determine an approximate normal to the surface represented by the voxel neighborhood. The direction of the normal determines the area contribution from each face of a voxel. Figure 5.14 (rightmost column) shows the area voxel skeleton for some example caves.

Third, I determine the cave field of the cave, which is an axis-aligned bounding rectangle, and optimize it, as suggested by Klimchouk [29] (Section 2.4.3). The area of the optimized cave field is better for making comparisons of morphometric indices between caves, but there is some subjectivity in how the optimized bounding polygon is determined. My procedure is to start with the rectangular cave field, draw an infinite vertical cylinder with a fixed radius R_{opt} at each voxel and only keep that voxel in the cave field if the cylinder contains a non-empty voxel. The result is an optimized cave field that fits the voxels of the cave more closely than a bounding rectangle. Morphometric indices derived from the area of the optimized cave field, which is computed according to my procedure, are comparable to the indices reported for real caves by Klimchouk, as he finds that areas of optimized cave fields are 2 to 5 times smaller than unoptimized ones, and this is true for my computation, as well.

Finally, it is trivial to compute the rest of the morphometric indices given length, area, and cave field values. Table 5.3 contains the results for several caves that I have created procedurally. The definition of cave field affects only the indices in the first three rows of

	Figure 5.9	Figure 5.10	Figure 5.11
network density $\left(\frac{m}{m^2}\right)$	0.317 (0.122)	0.462 (0.179)	0.277 (0.105)
areal coverage (%)	46.51 (17.94)	77.54 (30.05)	63.66 (24.05)
porosity (%)	5.59 (2.16)	8.74 (3.39)	14.06 (5.31)
$L : W$	193.5	191.1	79.25
W (m)	2.28	1.95	3.33
field R_{opt} (m)	12	8	8

Table 5.3: Morphometric indices of modeled caves. Values in parentheses use rectangular cave field.

the table: network density, areal coverage, and porosity. For these three indices, the values without parentheses use the optimized cave field definition and should be compared to Table 2.1; the values in parentheses use the rectangular cave field and should be compared to Table 2.2. The other two indices, length to width ratio and width, should be compared to both tables.

The morphometry of the procedurally generated cave in the first column of Table 5.3 matches well with both data sets for real caves in the confined category. However, the two real cave data sets have one dissimilarity: the range for the $L : W$ value starts and ends much higher in Klimchouk's data. As a result, $L : W$ for the procedural cave agrees only with the data from Table 2.2. Observe that as a cave system becomes longer L can increase indefinitely, while W can be expected to stay the same. Therefore, the dissimilarity between the two data sets for real caves can be explained by the different lengths of the caves that were analyzed. Analogously, the cave that I generated procedurally could better fit the first real cave data set with larger $L : W$ values if its length was increased, i.e., if the simulation was carried out on multiple blocks of voxels and the results stitched together. In that case, values of length, area, and volume would all increase linearly and the derived indices would be unchanged (e.g., $W = A : L$), except for $L : W$, because it is the only quantity that involves a non-linear function: $L : W = L^2 : A$.

The second procedurally generated cave also exhibits the same disagreement in $L : W$ and additionally its network density and areal coverage are too high as compared to the first real cave data set (13.8% and 32.5% increase compared to the maximum value of the range, respectively). In other words, the procedural cave (and likely the second real cave set relative to the first one, although they are difficult to compare due to the difference in their definition of cave field) has values of L and A that are slightly too high for its value of A_{field}, which means that the cave has relatively long passages that are clustered together. This situation is characteristic for confined caves and my procedural cave example does

feature a pattern of planar fractures that guide the cave's development.

The morphology of the third generated cave is slightly different from the others, as it consists of one dominant channel, compared to the more elaborate channel networks of the other two caves. Therefore, the third cave has the smallest $L : W$ ratio, especially since it also has the largest value of W. As before, the value of $L : W$ is too low for the first data set of real caves, but not for the second one. More importantly, the compact nature of the third cave also causes it to have a lower network density, but higher areal coverage and porosity. Optimization of the cave field in this situation brings $A : A_{field}$ closer to one and can also have the same effect on $V : V_{block}$ unless the passages of the cave are spread in the vertical direction. This is why the areal coverage and the porosity of the third cave slightly exceed (8.8% and 17% increase compared to the maximum value of the range, respectively) the values for the first data set of real caves, which uses the optimized cave field value.

The caves created with my simulation possess morphometric indices that match the confined category of both data sets of real caves. The indices of generated caves fit the value ranges of the second data set, but have a tendency to exceed some values of the first data set. The larger values compared to the first data set can be explained as the result of either a large number of channels clustering in a small volume or the optimized cave field being able to shrink around a compact channel network. Since longer caves can be expected to spread out more, it is no surprise that the generated caves match the second data set (longest cave 3540m) better than the first (longest cave 188km). Furthermore, comparison of the generated caves with the first data set would likely improve if the generated caves were longer. Note that if the goal is only to categorize the caves, the amounts by which some of the values are exceeded are not significant, as the differences between index values of different categories of caves can be as large as an order of magnitude.

I conclude that the caves produced with my simulation possess morphometric indices that are appropriate for the confined cave category. The confined character is likely the product of my decision to include more sources than sinks in my simulations to force channels to link up and produce more interesting caves. Additionally, my simulation causes growing cave passages to realistically exploit weaknesses in the rock matrix, such as fractures, which also has a net effect of preventing unconfined behavior.

Chapter 6

Conclusion

6.1 Contributions and Themes

I have explored an approach to modeling that combines physical simulations and procedural modeling by using principles of self-organization as the common element. In particular, my use of self-organization requires both development of a model of the emergent behavior of a physical system and formulation of a related concrete simulation that constructs a geometric representation of the modeled objects. The modeling process includes my avalanching modeling paradigm and common simulation framework, which I designed to take advantage of avalanching. Because avalanching has a natural connection to erosion, I have demonstrated my modeling approach by applying it to modeling of the effects of erosion. Figures 6.1 and 6.2 present my contributions at a glance.

I have solved several new modeling problems using my avalanching paradigm and framework. First, I have developed a model of self-organized channel networks that exhibit widening of channels and flooding of low areas, where the former is a direct consequence of avalanching. The flooding effect is produced by a water-column simulation, which can be part of the model because of abstraction of erosion dynamics due to avalanching. The main idea behind my modeling process for channel networks is to encapsulate behavior like tributary capture with avalanching and have a water simulation that can respond to erosion by concentrating water flow into the emerging channels.

Second, I developed a coastline model that uses avalanching to produce coastlines of varying degree of foldedness. Because of the abstract nature of avalanching, this model also extends to modeling of the interior of islands using the level set interpretation of

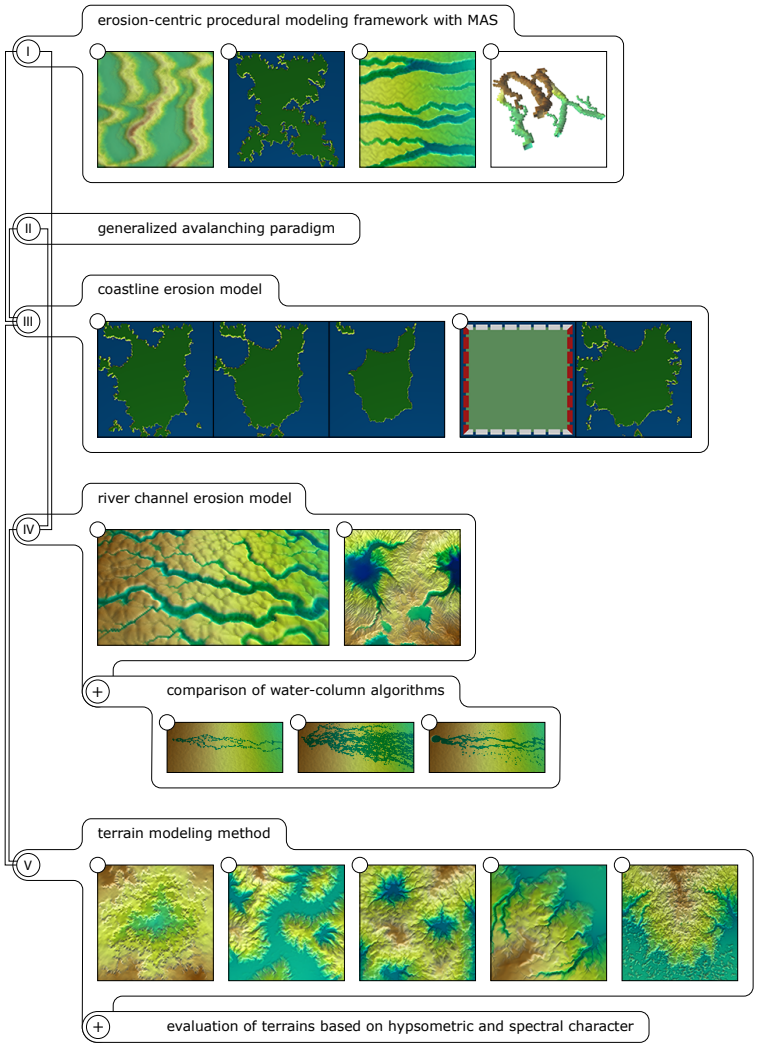


Figure 6.1: Summary of contributions (Part 1 of 2).

coastlines on terrains. Third, I have combined coastline and terrain modeling to achieve a procedural model with two novel features: construction of terrains with hydraulic erosion features using only self-organization and unification of hydraulic erosion features related to coastlines and rivers. Additionally, my model based on avalanching causes coastlines to be self-similar level sets of self-affine (except for the effect of rivers) terrains, which is motivated by the same relationship in explicit fractal synthesis of terrains, but achieved using implicit fractal synthesis using avalanching.

Fourth, I used avalanching to generalize my model of 2D channel networks to 3D channel networks. One of the main features of the generalization is a new model of water flow due to a self-organized formulation of pressure. The use of self-organization in this context makes my procedural model of cave-like channels unique.

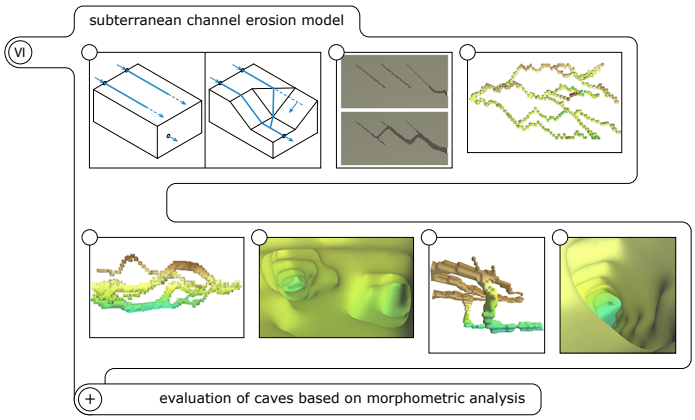


Figure 6.2: Summary of contributions (Part 2 of 2).

My demonstrations of modeling with avalanching use self-organization almost exclusively, owing to self-organization in erosion. In other words, the modeling power of avalanching comes from putting emphasis on where erosion occurs and how it propagates rather than the exact mechanism of erosion and the details of what happens during each erosion event. Therefore, one common theme in my modeling approach is repeated usage of avalanching to encapsulate different aspects of a model. For example, in my combined terrain modeling method avalanching creates initial conditions and channel erosion; similarly, in my model of subterranean channels avalanching determines pressure changes and simulates erosion.

Use of the avalanching paradigm as an abstraction for erosion-related behavior also motivates unification of the modeling process for the physical systems that exhibit erosion, such as dunes, coastlines, etc. This has been one of my primary goals in designing a common procedural modeling framework for use with avalanching. The agent-based formulation of the framework is helpful in achieving this goal, as the agent paradigm is based on propagation of information between neighbors, which has the expressive power necessary for simulation of erosion with avalanching. However, avalanching is the key to generalization, because it offers abstraction that stimulates development of erosion models using common principles.

6.2 Observations about Modeling

In the process of developing my modeling paradigm I have made several observations regarding the use of simulations and fractal synthesis in the context of procedural modeling. This section details these observations, which I believe can be of fundamental importance in applying the procedural method effectively.

6.2.1 Procedural Modeling

I have found that several aspects of procedural modeling are particularly desirable in a procedural modeling framework. First, local state and computation similar to a MAS paradigm provide a foundation for self-organized simulations, including simulations of erosion. This type of computation can numerically solve PDEs, which are essential for expressing the effect of erosion on the geometry of a landscape. Furthermore, the quantities used in a PDE can themselves be computed using a simulation-based approach, as I have shown with Equation 4.4 and the water-column algorithm.

Second, additional use of procedural modeling to produce initial conditions for a simulation, such as in my two-stage terrain modeling method, makes it easier to model complex objects. In general, a procedural model can be effective at capturing a particular behavior that leads to the formation of features like rivers with tributaries, but creating a complex landscape calls for modeling a composition of many kinds of features. I believe that this aspect of the procedural modeling problem requires the use of additional simulation stages, as I have shown with my simulations.

Third, division of physically-based simulations into components allows additional models to be formulated through combination, enhancing the versatility of the procedural approach. For example, I have implemented several construction algorithms by having a water simulation component, which can effect changes to geometry only in combination with a formulation of erosion. So I have been able to follow the same high-level approach to model rivers with valleys, terrains with coastlines, and 3D cave-like passages.

Fourth, some methods of fractal synthesis are more suitable for a particular modeling scheme than others. Since my modeling framework is based on simulation, I have formulated a method to synthesize fractals implicitly, as a by-product of self-organized simulations. Furthermore, my spectral analysis procedure shows that the result of some fractal synthesis algorithms, especially approximations to fBm such as noise-summing, have a tendency to be less noisy than fBm, similar to natural terrains. This deserves further investigation.

6.2.2 Simulation Parameters

Use of parameters in procedural modeling that involves physically-based simulations is problematic, because poorly-designed simulations may "work" only for a certain set of parameter values. While I was developing my simulations, I tried to understand the effects of the parameters I used, so that I could avoid the situation when a simulation produces poor results for reasonable parameter values. Once I settled on the final versions of my simulations, I ran any given simulation at most a few times, as it was necessary for modeling an object for one of the figures. I tried my best not to "fish" for good parameter values or use different random seeds to find a better output.

In realistic modeling contexts, specification of an object's appearance may not be based only on specific parameter values and in many cases initial conditions may be more important. For example, to create a model of the specific hill with erosion in Figure 4.20, I needed to specify the rough shape of the hill manually as initial conditions. The subsequent simulation qualitatively changed the appearance of the hill to include realistic patterns of erosion, but the same effect could have been accomplished with many different settings for erosion, especially according to a casual observer.

6.3 Future Work

In the future I would like to incorporate my simulation framework into a more general content creation system and also develop additional functionality that can facilitate further experimentation with the modeling process. I hope that by adding new features in the following areas I might be able to discover more applications of my procedural modeling method.

Grid freedom: One of the ultimate goals of procedural modeling is the ability to author large, high resolution virtual worlds. However, the computational demands of physical simulations associated with a grid-like domain tend to increase rapidly with grid resolution. Use of massively-parallel GPU architectures increases the ceiling on the practical range of resolutions, but it is not a fundamental solution to the scaling problem. It also comes at the expense of an additional stage of encoding data for submitting it to the GPU and some constraints on the algorithm that are necessary to make it GPU-friendly. Newer procedural modeling frameworks show that for certain modeling problems an alternative approach to scaling is possible, which incorporates

top-down design elements and does not require a grid. I believe that my procedural modeling framework can be effectively extended to modeling of extremely large worlds via a hybrid approach that will combine a simulation on a coarse grid with an additional stage based on self-organization that will derive resolution-free detail from the results of the simulation and construct output geometry with any given resolution.

Flexible specification of initial conditions: While my modeling method emphasizes procedural elements it also requires non-trivial initial conditions to be specified, especially when a simulation occurs in multiple stages (Figure 4.17). Top-down control via constraints in initial conditions can extend the procedural approach so that it becomes easier to apply to a given modeling problem. However, a system for constructing and applying the constraints must be sufficiently general and combine with the procedural framework effectively, or the combination of the two will not gain in flexibility.

Experiment infrastructure: Procedural models, particularly ones with physics-based elements, tend to have a lot of parameters. Both when developing such models and when using them to construct a desired object, it is necessary to appreciate the effect of the parameters by experimenting many times with their values. This is particularly important for multi-parameter models, because it can be difficult to judge the relative importance of a single parameter or whether a combination of parameter values can cause the model to transition from one type of behavior to another (e.g., valley evolution modes in Section 4.3.3). So future development of my framework can benefit from an additional layer of infrastructure that can facilitate exploration of model parameter space via sets of scripted virtual experiments with different parameter valuations. Such an experimentation module can be of general use in scientific computation research and can be futher extended to incorporate multiparameter optimization techniques.

Procedural detail generation: A fully-realized procedural virtual world does not only consist of a terrain or a cave domain, but also many associated elements such as rock piles and vegetation. These elements can be discrete such as grass on a terrain or they can be blended with the underlying geometry, such as stalagmites in caves. There exist many algorithms for procedural placement of objects, but for use with my framework, it would be desirable to develop one that relied on similar self-organization principles and information that could be collected from running my simulations, instead of being a completely independent algorithm.

APPENDICES

Appendix A

Additional Details

A.1 Spectral Density of fBm

Theorem 2.1.1. *If an instance of fBm is discretized as a set of samples f, then for $\beta = 2H + 1$*

$$S(f)(\theta) \propto \frac{1}{\theta^{\beta}}.$$

Proof. This proof is adapted from a continuous version by Saupe [59]. It considers the 1D setting for simplicity.

Let f consist of N samples that are spaced Δt apart. Define a new set of N samples $g[i] = \frac{1}{b^H} f[i]$ that are spaced $\Delta t_g = \frac{1}{b}\Delta t$ apart. In other words, g rescales f consistently with the self-affinity property of fBm (for scaling factor b and exponent H).

Now observe that by definition of DFT, $\mathcal{F}(g) = \frac{1}{b^H}\mathcal{F}(f)$. This is because only ordinate values are involved in the definition. As for spectral density, it is defined using quantities in both of the scaling directions, so that $S(g)(\theta) = S(f)(\theta)$ for $\theta = i\Delta s_g$.

There is a relationship between spacing in the time and frequency domains: $\Delta t \Delta s = \frac{1}{N}$. Using it and $\Delta t_g = \frac{1}{b}\Delta t$, the remaining variable Δs_g is determined: $\Delta s_g = b\Delta s$.

Putting everything together yields

$$S(f)(bi\Delta s) = S(f)(i\Delta s_g) = S(g)(i\Delta s_g) = \frac{|\mathcal{F}(g)[i]|^2}{\Delta s_g} =$$

$$= \left(\frac{1}{b^H}\right)^2 \frac{|\mathcal{F}(f)[i]|^2}{b\Delta s} = \frac{1}{b^{2H+1}} S(f)(i\Delta s).$$

For the next step it is necessary to interchange the roles of b and i by making $i = 1$. Then b is a variable and θ can be obtained as $\theta = b\Delta s$.

$$
\begin{aligned}
S(f)(bi\Delta s) &= \frac{1}{b^{2H+1}} S(f)(i\Delta s) \\
S(f)(b\Delta s) &= \frac{1}{b^{2H+1}} S(f)(\Delta s) \\
S(f)[b] &= \frac{1}{b^{2H+1}} S(f)[1] \\
S(f)[b] \propto \frac{1}{b^{2H+1}} \text{ or } S(f)(b\Delta s) &\propto \frac{1}{(b\Delta s)^{2H+1}}
\end{aligned}
$$

□

A.2 River Rendering

Rendering of terrains is a well-studied problem. Common approaches focus on texturing a single triangle mesh by procedurally combining different kinds of textures and lighting effects. However, it is not trivial to integrate the rendering of rivers into such a framework.

In my simulations, the amount of water is known at each mesh vertex and the rivers are defined as locations where the column of water exceeds a threshold that corresponds to the depth of the smallest channel left by erosion. My first attempt at rendering the rivers was a procedural scheme that added the color of water to each triangle of the terrain based on the height of water at the triangle's vertices. This approach suffered from interpolation artifacts and made the channels appear too flat. Re-lighting the channels to simulate depth would be difficult without an explicit water surface.

My final approach to rendering rivers starts by extracting the geometry of the channels as an explicit mesh that is offset from the mesh of the underlying terrain by the height of water. The offset mesh accurately represents the surface of a river that runs down the side of a hill with a curving profile. The second step of my rendering algorithm is to use the water mesh in a rendering stage that post-processes the result of rendering the terrain by modifying the terrain color in a way that simulates the presence of water of a given depth. Figure A.1 presents a high-level overview of the process.

Listings A.1 and A.2 show partial GLSL code of the vertex and fragment shader for water, respectively. The code for functions `calc_pos` and `extinct_color` is not essential for describing the rendering pipeline and I have omitted it. The function `calc_pos` reconstructs fragment position in eye coordinates from its depth. The function `extinct_color`

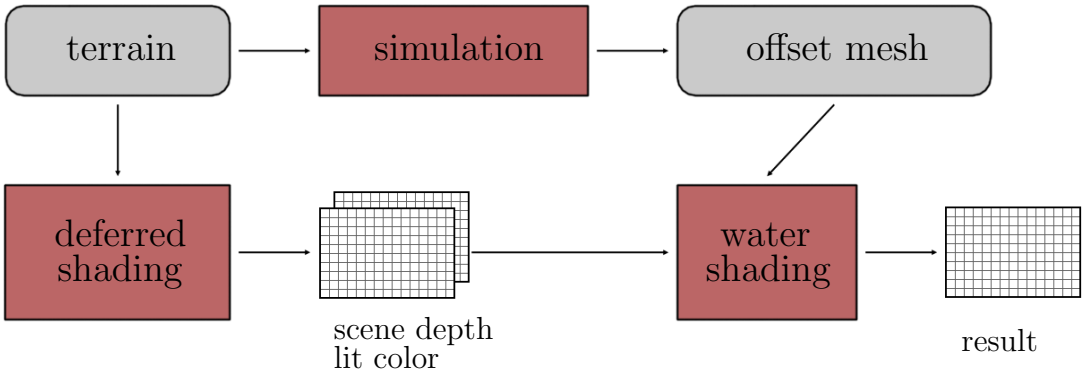


Figure A.1: Pipeline for river rendering.

computes the effective color of the ground as seen through the layer of water covering it. The extinction of other colors is what makes the water appear to be a darker blue the deeper it is.

Listing A.1: GLSL vertex shader code for rendering water.

```
1  varying vec3 normal_EC;
2  varying vec3 pos_EC;
3
4  void main()
5  {
6      normal_EC = normalize(gl_NormalMatrix * gl_Normal);
7      pos_EC = vec3(gl_ModelViewMatrix * gl_Vertex);
8      gl_Position = gl_ModelViewProjectionMatrix*gl_Vertex;
9
10     vec2 screen = vec2(gl_Position.xy) * vec2(1.0 / gl_Position.w);
11     gl_TexCoord[0].st  = 0.5*screen + vec2(0.5);
12 }
```

Listing A.2: Partial GLSL fragment shader code for rendering water.

```
1  uniform vec2 viewport;      // w,h of the render target
2  uniform vec4 unproject;    // near, far, s = cot(fov_y/2), aspect
3
4  varying vec3 normal_EC;
5  varying vec3 pos_EC;
6
7  void main()
8  {
```

```
 9     vec4 landc = textureRect(tunitc, viewport * gl_TexCoord[0].st);
10
11     vec3 pos_land_EC = calc_pos(gl_TexCoord[0].st);
12     if (pos_land_EC.z < 0.001 - unproject.y) discard;
13     float dist = length(pos_land_EC - pos_EC);
14
15     vec4 waterc = extinct_color(landc, dist);
16     float wbdr = smoothstep(WFADE_LOW, WFADE_HIGH, dist);
17
18     gl_FragData[0] = mix(landc, waterc, wbdr);
19 }
```

A.3 Voxel Polygonization

To develop my 3D erosion simulations I have used my own algorithm for visualizing the voxel data as a triangular mesh. The voxels represent cubes of material that are eroded to varying degrees. So a simple way to construct a coarse visualization is to represent each voxel that is eroded more than a given threshold with a cube. However, in such a representation a channel made of side-by-side eroded voxels will be segmented by extraneous cube faces, which can clutter the visualization. Moreover, the topology of the channel will be incorrectly represented with multiple cubes and coincident (non-manifold) faces. The goals of my visualization algorithm are to manage the topology of the constructed mesh correctly and to create a coarse representation that can be subdivided to achieve smoother shading.

A.3.1 Main Algorithm

In the first stage of my algorithm each full voxel (i.e., eroded less than a threshold value) is represented with eight vertices and twelve triangles that implicitly connect the vertices in the shape of a cube. Additionally, each vertex maintains a reference list that allows neighboring cells to share their vertices. The algorithm merges vertices that belong to coincident faces of neighboring full voxels by deleting one set of vertices and using their reference lists to update all vertex pointers that used the deleted vertices. Then the algorithm uses the remaining vertices to create two triangles for all faces that lie on the boundary between a full and an empty cell. The triangles are added to a half-edge mesh representation. The end result is that side-by-side occupied voxels appear to merge together into a larger solid, while occupied voxels that are touching only diagonally stay separate.

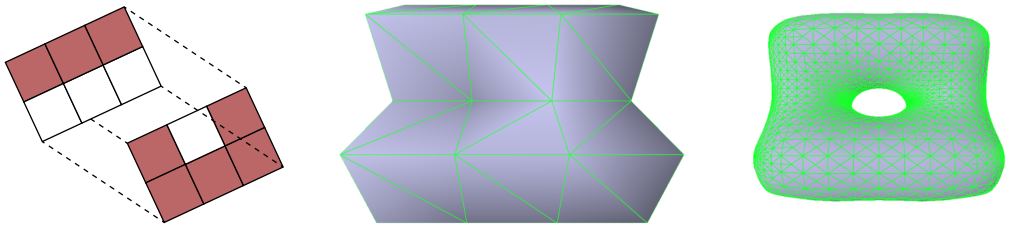
Listing A.3: The main stage of voxel polygonization.

```
1  polygonize() {
2    for each voxel {
3        create 8 vertices and 8 reference lists
4        each vertex initially contains itself in its reference list
5    }
6    for each voxel {
7        look at 3 faces: +X, +Y, +Z
8        if face is between two full voxels {
9              call merge_vertex on the 4 vertices of the face
10       }
11   }
12   for each voxel {
13       look at all 6 faces
14       if voxel is full and face is shared with an empty neighbor {
15             add (once) 4 vertices of the face to the mesh
16             add 2 triangles of the face to the mesh
17       }
18   }
19   complete mesh boundary
20   clean up vertex pointers and reference lists
21 }
22
23 merge_vertex() {
24   delete_list = reference list of the vertex to be deleted
25   neighbor_list = reference list of the other (neighbor) vertex
26   for each r in delete_list {
27       point r to neighbor vertex
28       add r to neighbor_list
29   }
30   delete orphaned vertex and reference list
31 }
```

A.3.2 Non-Manifold Edge and Vertex Removal

The voxelization algorithm must address two topological issues. First, the half edge mesh can link a triangle to the wrong edge in the case of diagonally-touching cells and erroneously create a non-manifold edge. Figure A.2 illustrates the scenario. Second, there is one situation when the merging in the main stage is too aggressive and produces a non-manifold vertex with two vertex loops. Figure A.3 displays the issue and the solution.

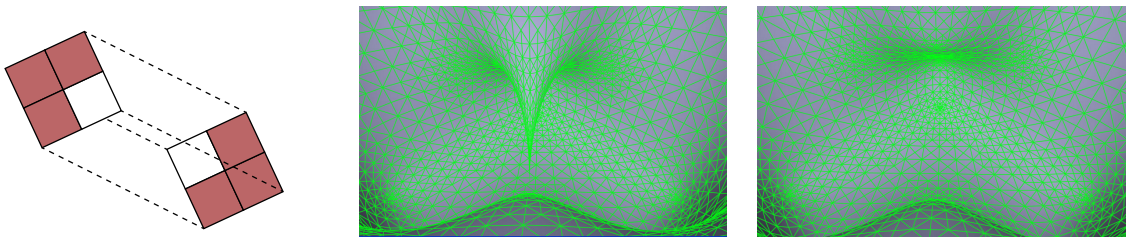
Non-manifold edge removal takes place during construction. When triangles are added to the mesh, a correct edge match must be found if it exists. This is accomplished with a



(a) (b) (c)

Figure A.2: Configuration that can produce a non-manifold edge: (a) two layers of voxels represented as 2D squares, (b) resulting mesh with coincident edges, (c) subdivided mesh.

hashmap of existing edges keyed by their midpoint. A correct match has the appropriate endpoints and has one half-edge missing. This simple matching procedure is sufficient for voxelization, as the surface becomes "closed up" around one of the coincident edges after all triangles are added for one voxel.



(a) (b) (c)

Figure A.3: Configuration that can produce a non-manifold vertex: (a) two layers of voxels represented as 2D squares, (b) interior view of mesh with error, (c) interior view of corrected mesh.

Non-manifold vertex removal takes place during a post-processing stage that follows the execution of the main stage of the voxelization algorithm. This part of the algorithm works directly on the half-edge mesh representation to make the necessary changes. In a half-edge mesh each edge has two half-edges that point in opposite directions. A vertex has a link to one half-edge that it can use to visit all edges that are incident on it (i.e., its vertex loop) and all of its neighbor vertices.

Listing A.4: Non-manifold vertex removal.

```
1 bool vertex_loop_match(vertex * v, half_edge * h)
```

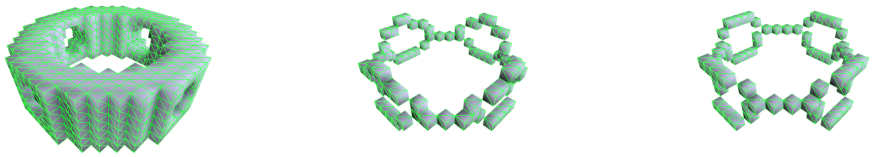
```
2  {
3    go around v's vertex loop {
4        H = current half edge
5        if (H == h) return true;
6    }
7    return false;
8  }
9
10 fix_nm_vertices() {
11   for each vertex v {
12       go around v's vertex loop {
13             c = current half edge
14             n = v's neighbor vertex on the loop
15             if (!vertex_loop_match(n, pair(c)))
16             {
17                   if (n is not marked) {
18                         create replacement vertex R
19                         point R at pair(c)
20                         mark n
21                         point the start of half-edges on n's loop to R
22                   }
23             }
24   }
25   update mesh with newly created vertices
26   reset vertex marks
27 }
```

A.4 Voxel Thinning

In many applications a geometric object may be voxelized on a grid of white and black voxels, where the set of black voxels B represents the approximate volume of the object. After voxelization, it is possible to extract a voxelized medial skeleton of the object using an operation called voxel thinning, in which black voxels are removed subject to topological constraints. These constraints depend on the connectivity of the voxel space, defined as an equivalence relationship based on neighbor adjacency.

On a cubic grid a given voxel has at most 26 neighbors and the corresponding type of connectivity is denoted N_{26}. Connectivity with all neighbors except the ones on the corners of the 3×3×3 cube is denoted N_{18} and connectivity with neighbors in the six primary directions (north, south, east, west, up, and down) is N_6. Typically, black and

white voxels are considered to have different connectivity with one of the most common combinations being $(26, 6)$, or N_{26} connectivity for black voxels and N_6 for white ones.



(a) (b) (c)

Figure A.4: Voxel thinning: (a) initial voxelized object, (b) skeleton produced by the original algorithm of Palágyi et al. [44], (c) skeleton produced by an algorithm variant.

Adjacency between black and white voxels gives rise to several relationships. A *border point* is a black point that is 6-adjacent to a white point. An *isolated point* is a black point that has no black neighbors. During thinning only border points can be removed, since doing otherwise would open up a hole in the interior of the object and change the topology. More generally, the topology will not change as long as only *simple points* are removed. The following two theorems allow a thinning algorithm to find simple points locally using a set of criteria and iteratively remove them.

Theorem A.4.1 (Kong and Rosenfeld, 1989)**.** Simplicity can be decided locally by testing a criterion that applies to 3×3×3 voxel cubes.

Theorem A.4.2 (Saha and Chaudhuri, 1994)**.** A black point p is simple in a $(26, 6)$ voxel space if and only if the following conditions hold:

1. $N_{26}(p) \cap (B \setminus \{p\})$ is not empty (p is not an isolated point);
2. $N_{26}(p) \cap (B \setminus \{p\})$ is 26-connected in itself;
3. $(\mathbb{Z}^3 \setminus B) \cap N_6(p)$ is not empty (p is a border point);
4. $(\mathbb{Z}^3 \setminus B) \cap N_6(p)$ is 6-connected in $(\mathbb{Z}^3 \setminus B) \cap N_{18}(p)$.

Palágyi et al. provide an algorithm that implements the thinning procedure according to the four criteria that characterize simple points [44]. The skeleton of the algorithm is listed in pseudocode form below. Each iteration consists of several subiterations that only remove border points exposed on one side (north, south, etc.). Figure A.4 shows

the results of applying this algorithm on a ring-like object with four holes. Note how the topology is preserved during thinning. Figure A.4c shows the result after I have adjusted the algorithm's subiteration structure to prioritize border points based on the number of exposed sides. I have found that this variant of the algorithm produces slightly more regular skeletons.

Listing A.5: Voxel thinning.

```
1  do {
2  subiteration: thin in direction U
3  subiteration: thin in direction D
4  subiteration: thin in direction E
5  subiteration: thin in direction W
6  subiteration: thin in direction N
7  subiteration: thin in direction S
8  } while (progress);
9
10 subiteration() {
11   foreach p in voxel space
12     if (test(p))
13           add p to candidate list
14
15   foreach element in candidate list
16     if (test(element))
17           erase element
18 }
```

A.5 Fast Marching

The boundary between two or more materials situated side by side can evolve as the zones containing the materials deform or one material changes into another. Theoretical and computational aspects of representing this behavior form the study of propagating interfaces, or fronts. Numeric techniques for approximating moving interfaces have many applications in computational physics. One possible formulation of front evolution is based on the level set method, which makes the formulation especially amenable to numeric approximation and offers a straightforward generalization from the 2D setting to 3D.

An $(N-1)$-dimensional interface is represented with a surface $\Gamma(t)$, where t is the time variable. The interface, which is a set of closed curves in the 2D setting, starts as a given shape $\Gamma(0)$ and undergoes motion in the normal direction based on a speed function F, which can encode local and global properties of the front, as well properties

of the environment that are independent of the front. As Γ evolves, its shape can become complex, incorporating sharp creases and changes in topology. Fortunately $\Gamma(t)$ can be adequately represented as the zero level set of an N-dimensional function $\phi(\vec{\boldsymbol{x}}, t)$:

$$\Gamma(t) = \{\vec{\boldsymbol{x}} | \phi(\vec{\boldsymbol{x}}, t) = 0\}. \tag{A.1}$$

This is an Eulerian formulation, because the underlying system of coordinates is fixed. Using this formulation the fronts can be found as a solution to the equation

$$\phi_t + F|\nabla\phi| = 0. \tag{A.2}$$

If F is always positive or negative, the equation can be re-stated in a simpler form without the time variable. In this case the function $T(\vec{\boldsymbol{x}})$ that gives the time at which the front reaches the location $\vec{\boldsymbol{x}}$ is well-defined (i.e., single-valued). The front $\Gamma(t)$ can be re-stated as $\{\vec{\boldsymbol{x}} | T(\vec{\boldsymbol{x}}) = t\}$ and can be found by solving:

$$|\nabla T|F = 1. \tag{A.3}$$

This is equivalent to the eikonal equation as long as F depends only on position. Furthermore, in the special case when F is constant, the solution gives signed distance from $\Gamma(0)$, which makes the following algorithm a simple way to obtain a grid of samples of signed distance from a given shape.

Sethian [61] provides the following algorithm, called fast marching, for numerically solving Equation A.3 on a discrete grid. It involves computing new values of T in a narrow band around known values with the updates taking place in an "upwind" order, from smaller values of T to larger ones, which is similar to Dijkstra's algorithm.

Listing A.6: Fast marching for propagating interfaces.

```
1  for each grid point {
2      initialize alive points to points on the initial front
3      T = 0 for alive points
4      initialize narrow band points to neighbors of alive points
5      initialize far away points to all other points
6      T = inf for far away points
7  }
8
9  for each narrow band point {
10     d = distance to closest alive point
11     T = d / F(i, j)
```

```
12  }
13
14  while narrow band is not empty {
15      P = narrow band point with smallest T
16      mark P as an alive point
17      mark neighbors of P that are not alive as narrow band points
18      recompute T for neighbors of P that are not alive
19   }
```

A.6 Hardware and Software

Table A.1 lists the hardware and software that I used for my project's implementation. I wrote my code in C++ and compiled it as a 32-bit application.

CPU	AMD Athlon 4400+ 2.5 GHz
GPU	Nvidia 7800 GTX
OpenGL	`2.0.1`
GLSL	`1.10`

Table A.1: Hardware and software specifications.

Appendix B

Alternative Models

B.1 Coastline Erosion on GPU

In this section I re-formulate a coastline erosion model from Section 4.2 to be more friendly for procedural modeling applications. The alternative model captures the same physical behavior as the original models (i.e., coastlines become folded to resist erosion), but allows more freedom when it comes to defining what constitutes a coastline and calculating its length. An implementation of the new model on GPU offers a substantial performance increase. I follow up performance analysis of the GPU-based simulation by re-implementing it on the CPU.

B.1.1 GPU Computing Concepts

A major part of the realtime rendering pipeline can be viewed more abstractly as a process that reads data from input buffers, performs computation, and writes the result to output buffers. The input and output buffers are typically distinct, as the extreme parallelism of GPU computing relies on processing multiple data simultaneously. Accessing multiple data to compute a single result is called a *gather* operation, while outputting multiple data is called a *scatter* operation. In principle, scatter operations can be re-factored into multiple gather operations, which is preferable for achieving optimal performance.

Similarly, performance considerations typically constrain the locations of the data accessed during gather operations. For example, computing the sum of the elements of a rectangular array should be formulated in terms of multiple gather operations that can

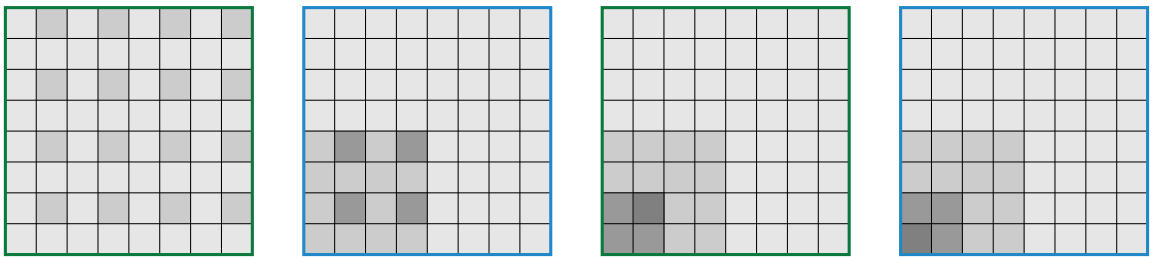


Figure B.1: GPU reduction operation with two buffers.

be performed independently. Such a series of gather operations is called a *reduction*. Figure B.1 illustrates the process with two buffers. At each iteration, the GPU reads $2n \times 2n$ locations from the input buffer and writes $n \times n$ results to the output buffer. To compute each result the GPU performs a 2×2 gather operation. The input and output buffers exchange their roles at the end of each iteration.

Reductions and basic gather operations are sufficient as primitives for solving many computational problems. However, there can be different platforms that provide such GPU-based computational facilities. Owens et al. [43] provides a more complete introduction to GPU computing.

I have implemented these operations using fragment shaders operating on floating point rectangular arrays. Mapping general GPU computing tasks to parts of the standard rendering pipeline has several advantages, such as more intuitive use of GPU resources and control over thread coherence. The main disadvantage of the approach is the difficulty of adapting existing code to run on the GPU. The resulting algorithm that uses the GPU pipeline can be much faster, but some features of the original algorithm have to be compromised.

B.1.2 Restatement for GPU Computation

In Section 4.2 I have discussed several coastline erosion models that describe how discrete sections of the coast erode due to the action of the ocean. Simulating these models involves such tasks as using neighbors to determine the degree of exposure to the ocean, computing the perimeter of the coast, and updating the state of coastline sites affected by avalanching. Some of these tasks map well to gather and reduction primitives from the previous section. However, the avalanching in my most sophisticated coastline model is a scatter operation that can potentially affect locations that are far away. Implementing this operation on the GPU would require a prohibitive number of texture reads in a fragment shader.

Computation of the local perimeter seen by an individual coastline site is also a challenge. However, interpreting the model in terms of sampling coastline roughness (Section 4.2.2) suggests approximating the effect of local perimeter by using global contribution to the perimeter from fixed nearby regions. Stopping the reduction process early provides this information.

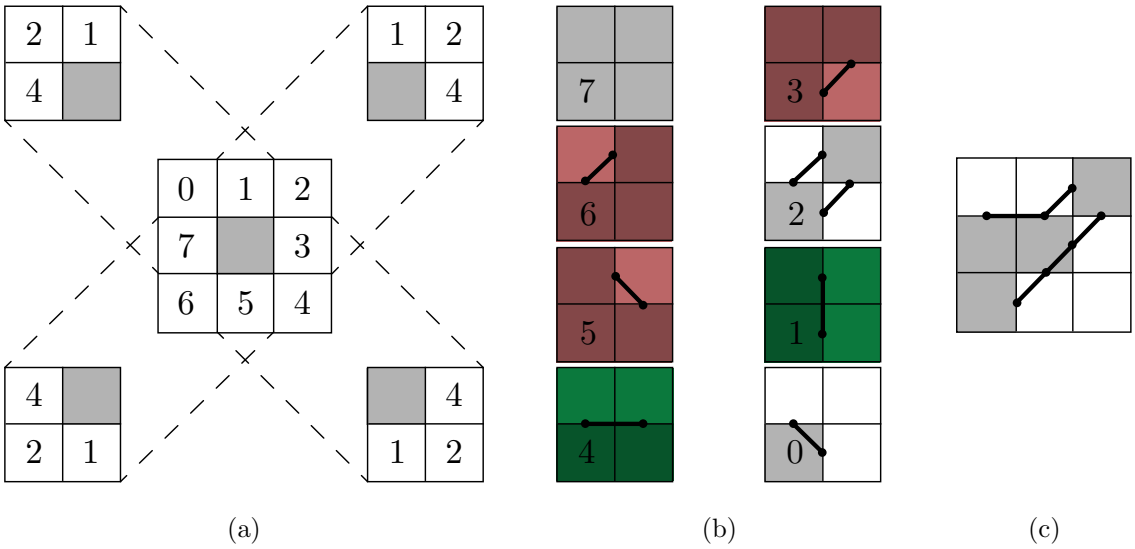


Figure B.2: Perimeter calculation for GPU coast model: (a) neighborhoods around each site, (b) configurations with coastline interpretation, (c) example for a complete 3×3 neighborhood.

To make the GPU-friendly model complete it is also necessary to define how the length of the coastline can be obtained from the discretized land-ocean interface. The models of Section 4.2 identified coastline sites on a discrete grid and connected them with graph edges, summing the lengths of the edges to yield perimeter contributions. However, the computational aspect of the new model limits it to operating on the discrete grid as it is presented in textures to shader programs.

Figure B.2 describes a scheme that I have developed to obtain coastline edges from a pixel grid in a consistent way. The algorithm examines a 3×3 neighborhood of each pixel and separately examines each of the overlapping 2×2 quadrants. Since land areas have value 1 and sea areas have value 0, the contents of the 2×2 areas can be represented with a binary number. Each number corresponds to a configuration of coastline segments, selected in such a way that the overlap between neighboring 3×3 and 2×2 areas produces a

contiguous set of segments that consistently separates water areas from land areas. Therefore configurations of pixels can be interpreted as perimeter contributions in terms of two types of segment lengths: straight and diagonal. The computation simplifies to evaluation of a few simple conditionals as shown in Listing B.1.

Listing B.1: Local contribution to perimeter for GPU model.

```
1  float func_side(int i) {
2     bool test1 = (i==1);
3     bool test4 = (i==4);
4
5     return float(test1) + float(test4);
6  }
7  (...)
8
9  //          top              topleft          left
10 cases[0] = region[1] + 2 * region[0] + 4 * region[7];
11
12 (...) // three more cases
13
14 for (int i=0; i < 4; ++i) {
15   side_total += func_side(cases[i]);
16   diag_total += func_diag(cases[i]);
17 }
18
19 side_total *= site.r;
20 diag_total *= site.r;
21
22 gl_FragData[0] = vec4(site.r, side_total, diag_total, site.a);
```

Figure B.3 shows an overview of the complete GPU algorithm for simulating the new version of the local perimeter erosion model. The procedure uses four floating point buffers whose channels encode the following information: contents of the site (land or water), contributions to the global perimeter, and intrinsic resistance of the site R (Section 2.3.3). The perimeter shader program calculates each site's contribution to the perimeter based on its 3×3 neighborhood. The reduction stage sums up the contributions until the total perimeter is known for each sector of a $2^k \times 2^k$ grid. For example, a 2×2 grid corresponds to dividing the problem domain into quadrants and calculating the perimeter of each separately, producing four output values (stored in a 2×2 grid). Finally, the erosion shader uses the previous state of the simulation and the results of the reduction to change some of the land sites to water sites based on Equation 4.2. When the erosion shader accesses perimeter contributions for the fixed grid, they need to be bilinearly interpolated.

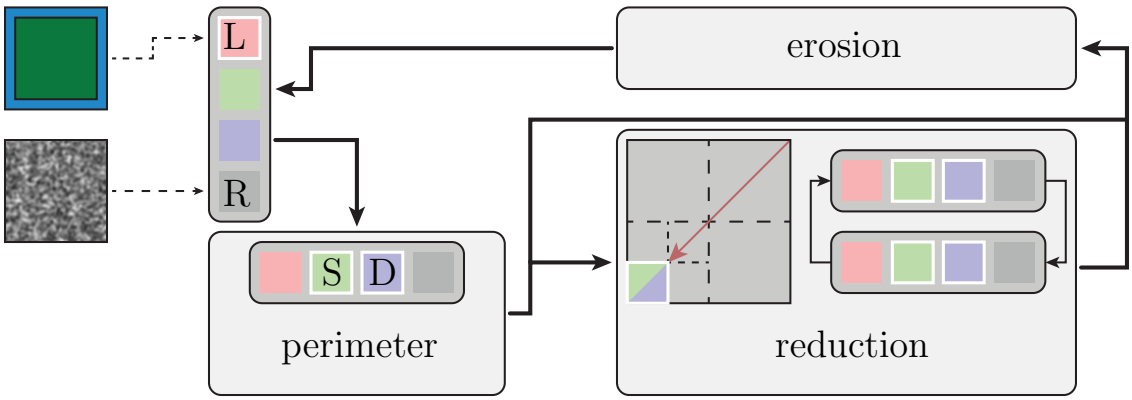
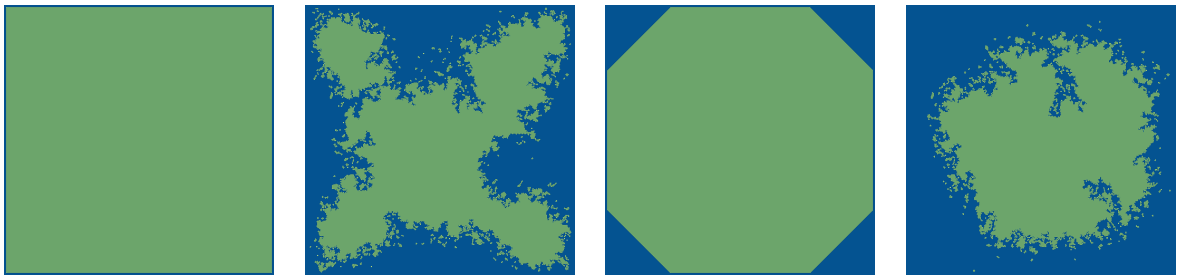


Figure B.3: GPU implementation of coastline simulation.

B.1.3 Results

Figure B.4 shows some results of simulating the GPU-friendly model for grid size of 512× 512. For the final rendering both sea and land pixels are re-colored slightly, in order to give them a more variable appearance. The land pixels start out as green and become more sandy as their exposure to sea increases; the sea pixels become darker when they are surrounded by land.

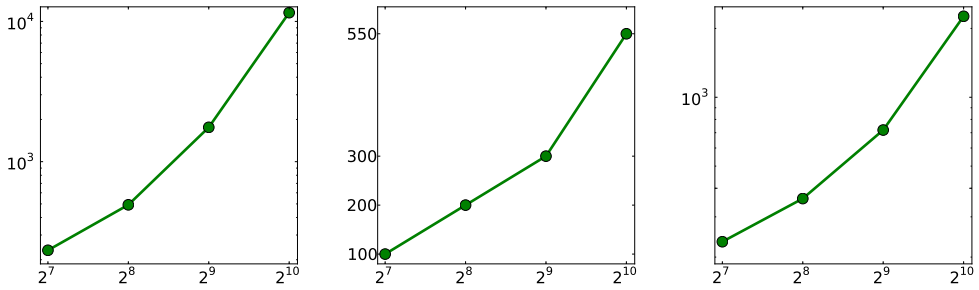


(a) initial conditions (b) result (c) initial conditions (d) result

Figure B.4: Results of GPU coastline erosion simulation.

Performance of the new simulation is substantially faster than that of its CPU counterpart, which was on the order of minutes to hours (Section 4.4.2). The plots in Figure B.5 visualize several performance aspects of the new simulation. The total runtime is shown in Figure B.5a. However, an increase in discretization density not only creates more sites for processing but also requires more iterations for the coast to become sufficiently folded for

resisting erosion. The increase in the number of iterations is shown in Figure B.5b. The plot in Figure B.5c avoids including the extra iterations in the runtime by measuring the time necessary to complete ony the first 100 iterations.



(a) runtime (ms) (b) iterations (c) runtime for 100 iterations (ms)

Figure B.5: Performance of GPU coastline erosion simulation for grid sizes from 128×128 to 1024×1024.

I used a GPU with 24 pixel shader pipelines to measure performance. The runtime measurement also includes some of the cost of setting up the computation buffers (some of the GPU resources are not allocated until the computation begins), but it is lower than resubmitting geometry in the CPU version of the algorithm. Note that all iterations on the GPU perform the same amount of work, but this is not true for the CPU version. Table B.1 lists parameters for the simulations used in the perfomance plots. The effects of the parameters are essentially the same as in Section 4.2.2. The number of iterations is the smallest multiple of 50 greater than the number of iterations required for convergence. A test to determine exactly when the coastline stops evolving would cost a roundtrip every iteration and greatly increase the total runtime (however, there are better options on newer hardware).

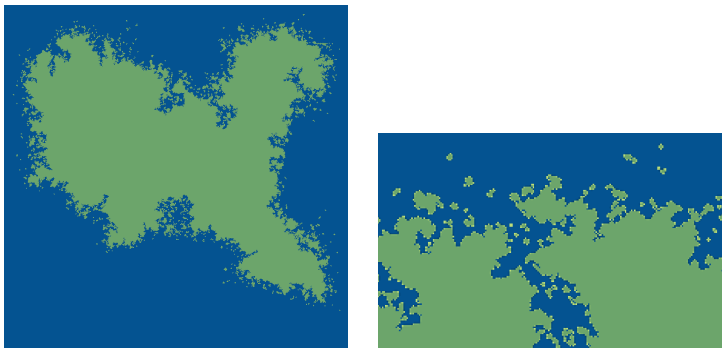
The new model based on GPU computation is much faster and potentially has wider applications for procedural modeling, as it operates by processing what is essentially binary image data. Following coastline evolution the resulting land and sea areas are encoded in textures and can be immediately used in an image synthesis pipeline, for example by acting as a mask for removing objects outside the irregular boundary symbolized by the coastlines. However, the new model has a slight drawback in comparison to CPU-based models that use more accurate perimeter calculation.

The GPU model uses a relatively small number of evenly-spaced samples for the perime-

Figure	grid	f_0	g	offset	iterations
B.5	128×128		0.230	2	100
B.5	256×256		0.175	2	200
B.4d & B.5	512×512	0.5	0.150	2	300
B.5	1024×1024		0.147	3	550
B.4b	512×512		0.120	2	250

Table B.1: Simulation parameters for GPU coastline simulations.

ter, making the initial shape of the coastline slightly noticeable in the final result. This effect is visible in Figure B.4, which shows initial configurations that are more rounded turning into more rounded final coastlines. However, the issue is isolated to the coarse level of detail of the image and can additionally be hidden by randomized behavior of the coastline due to the resistances R. Figure B.6 shows two more results where the coarse-level dependency on the initial shape is less apparent.

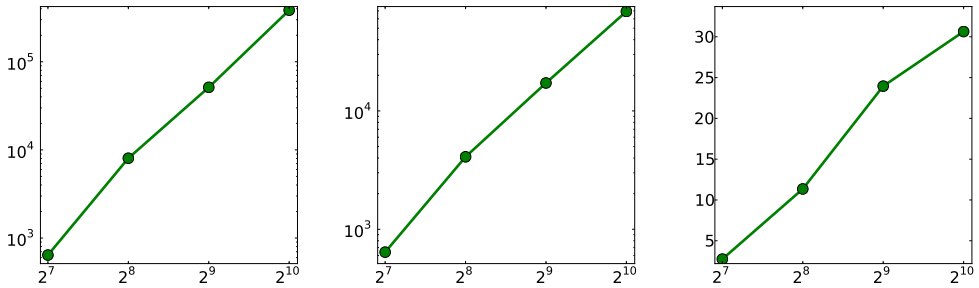


(a) (b)

Figure B.6: Closer look at dependence on initial conditions in GPU coastline simulation: (a) situation when dependence is less apparent, (b) coarse-level dependency not present in closeup.

I have also re-implemented the GPU simulation on the CPU for a better assessment of performance gains due to changes in the conceptual model, as opposed to parallelism of GPU architecture. This third generation simulation is serial and replaces summation based on the reduction operation with simple loops. The asymptotic complexity of both subroutines is the same, but summation via simple loops benefits from not having to write out many intermediate results to main memory (this extra work is justified for a reduction operation because it is executed by more than one thread). Figure B.7 contains

performance statistics collected by submitting the same workloads as in Figure B.5 for the GPU version. The performance gain of the second generation GPU version over the third generation CPU version is roughly between 3 and 30 for these workloads as shown in Figure B.7c. This puts the third generation CPU version between the original CPU models of Section 4.2 and the second generation GPU version in terms of performance, as expected.



(a) runtime (ms)

(b) runtime for 100 iterations (ms)

(c) gain of GPU over CPU version

Figure B.7: Performance of third generation CPU coastline erosion simulation for grid sizes from 128×128 to 1024×1024.

References

[1] Digital Elevation Data and Digital Map of Southern Ontario. Canadian Geospatial Data Resources. Toronto, ON: The Ontario Ministry of Natural Resources, 2008. Available: University of Waterloo Geospatial Centre. Accessed on 09/26/2014.

[2] A.-L. Barabási and H. E. Stanley. *Fractal Concepts in Surface Growth*. Cambridge University Press, 1995.

[3] M. Beardal, M. Farley, D. Ouderkirk, J. Smith, C. Rheimschussel, M. Jones, and P. Egbert. Goblins by Spheroidal Weathering. Eurographics Workshop on Natural Phenomena, pages 1–8, 2007.

[4] F. Belhadj and P. Audibert. Modeling Landscapes with Ridges and Rivers: Bottom Up Approach. GRAPHITE, pages 447–450. ACM, 2005.

[5] I. Benenson and P. M. Torrens. *Geosimulation: Automata-based Modeling of Urban Phenomena*. John Wiley & Sons, Ltd, 2004.

[6] M. Boggus and R. Crawfis. Procedural Creation of 3D Solution Cave Models. Technical report, Ohio State University, 2009. TR19.

[7] M. Boggus and R. Crawfis. Prismfields: a Framework for Interactive Modeling of Three Dimensional Caves. In *ISVC*, pages 213–221, 2010.

[8] Z. Chen, C. S. Stuetzle, B. M. Cutler, J. A. Gross, W. R. Franklin, and T. F. Zimmie. Analyses, Simulations, and Physical Modeling Validation of Levee and Embankment Erosion. Geo-Frontiers, pages 13–16, 2011.

[9] N. Chiba, K. Muraoka, and K. Fujita. An Erosion Model Based on Velocity Fields for the Visual Simulation of Mountain Scenery. *Journal of Visualization and Computer Animation*, 9:185–194, 1998.

[10] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*, pages 643–664. The MIT Press, 2nd edition, 2001.

[11] J. Cui, Y.-W. Chow, and M. Zhang. Procedural Generation of 3D Cave Models with Stalactites and Stalagmites. *International Journal of Computer Science and Network Security*, 11(8):94–101, 2011.

[12] S. Diniega, K. Glasner, and S. Byrne. Long-time Evolution of Models of Aeolian Sand Dune Fields: Influence of Dune Formation and Collision. *Geomorphology*, 121:55–68, 2010.

[13] J. Doran and I. Parberry. Controlled Procedural Terrain Generation Using Software Agents. *IEEE*, 2(2):111–119, 2010.

[14] J. Duran. *Sands, Powders, and Grains: An Introduction to the Physics of Granular Materials*. Partially Ordered Systems. Springer, 2000.

[15] J. Feddema and C. Little. Rapid World Modeling: Fitting Range Data to Geometric Primitives. volume 4 of *IEEE Conference on Robotics and Automation*, pages 2807–2812, 1997.

[16] J. Feder. *Fractals*. Plenum Press, 1988.

[17] D. Ford and P. Williams. *Karst Hydrogeology and Geomorphology*. John Wiley & Sons, Ltd, 2007.

[18] A. Frumkin and I. Fischhendler. Morphometry and Distribution of Isolated Caves as a Guide for Phreatic and Confined Paleohydrological Conditions. *Geomorphology*, 67:457–471, 2005.

[19] J.-D. Génevaux, É. Galin, E. Guérin, A. Peytavie, and B. Beneš. Terrain Generation Using Procedural Models Based on Hydrology. *SIGGRAPH*, 32(4):143:1–143:13, 2013.

[20] G. Greeff. Interactive Voxel Terrain Design Using Procedural Techniques. Master's thesis, Stellenbosch University, 2009.

[21] T. A. Grønneløv and A. E. Jensen. Procedural Planetary Landscapes with Continuous Level of Detail. Master's thesis, Technical University of Denmark, 2008.

[22] S. Gustavson. Simplex Noise Demystified. Technical report, Linköping University, 2005.

[23] H. Hnaidi, E. Guérin, S. Akkouche, A. Peytavie, and E. Galin. Feature Based Terrain Generation Using Diffusion Equation. *Computer Graphics Forum*, 29(7):2179–2186, 2010.

[24] K. Hormann, S. Spinello, and P. Schröder. C^1-continuous Terrain Reconstruction from Sparse Contours. Vision Modeling and Visualization, pages 289–297, 2003.

[25] J. Huang, A. Pytel, C. Zhang, S. Mann, E. Fourquet, M. Hahn, K. Kinnear, M. Lam, and W. Cowan. An Evaluation of Shape/Split Grammars for Architecture. Technical report, University of Waterloo, 2009. CS-2009-23.

[26] J. Jones and M. Saeed. Image Enhancement — An Emergent Pattern Formation Approach via Decentralised Multi-agent Systems. *Multiagent Grid Systems*, 3(1):105–140, 2007.

[27] M. Kamalzare, T. S. Han, M. McMullan, C. S. Stuetzle, T. F. Zimmie, B. M. Cutler, and W. R. Franklin. Computer Simulation of Levee Erosion and Overtopping. Geo-Congress, pages 1851–1860, 2013.

[28] B. H. Kaye. *A Random Walk Through Fractal Dimensions*. VCH Publishers, 1989.

[29] A. Klimchouk. Unconfined Versus Confined Speleogenetic Settings: Variations of Solution Porosity. *International Journal of Speleology*, 35(1):19–24, 2006.

[30] P. Krištof, B. Beneš, J. Křivánek, and O. Šťava. Hydraulic Erosion Using Smoothed Particle Hydrodynamics. *EUROGRAPHICS*, 28(2), 2009.

[31] K. Kroy, G. Sauermann, and H. J. Herrmann. Minimal Model for Sand Dunes. *Physical Review Letters*, 88(5):054301, 2002.

[32] M. Laverty. Fractals in Karst. *Earth Surface Processes and Landforms*, 12(5):475–480, 1987.

[33] N. Lemmens, S. de Jong, K. Tuyls, and A. Nowe. A Bee Algorithm for Multi-Agent Systems: Recruitment and Navigation Combined. Proceedings of ALAG, an AAMAS workshop, 2007.

[34] B. B. Mandelbrot. *The Fractal Geometry of Nature*. W. H. Freeman, 1982.

[35] K. Mason, J. Denzinger, and S. Carpendale. Negotiating Gestalt: Artistic Expression by Coalition Formation between Agents. 5th International Symposium on Smart Graphics, Frauenwoerth Cloister, Germany, August 2005.

[36] D. McDuff, S. Jackson, C. Shuchart, and D. Postl. Understanding Wormholes in Carbonates: Unprecedented Experimental Scale and 3D Visualization. *Journal of Petroleum Technology*, 62(10):78–81, 2010.

[37] G. S. P. Miller. The Definition and Rendering of Terrain Maps. *SIGGRAPH*, 20(4):39–48, 1986.

[38] F. K. Musgrave. A Brief Introduction to Fractals. In *Texturing and Modeling: A Procedural Approach*, pages 429–445. Elsevier Science, 2003.

[39] F. K. Musgrave, C. E. Kolb, and R. S. Mace. The Synthesis and Rendering of Eroded Fractal Terrains. SIGGRAPH, pages 41–50. ACM, 1989.

[40] J. F. O'Brien and J. K. Hodgins. Dynamic Simulation of Splashing Fluids. Computer Animation, pages 198–205, 1995.

[41] J. Olsen. Realtime Procedural Terrain Generation. Technical report, University of Southern Denmark, 2004.

[42] K. Onoue and T. Nishita. A Method for Modeling and Rendering Dunes with Wind-ripples. Pacific Graphics, pages 427–428, 2000.

[43] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips. GPU Computing. *Proceedings of the IEEE*, 96(5):879–889, 2008.

[44] K. Palágyi, E. Balogh, A. Kuba, C. Halmai, B. Erdöhelyi, E. Sorantin, and K. Hausegger. A Sequential 3D Thinning Algorithm and Its Medical Applications. In *Information Processing in Medical Imaging*, pages 409–415, 2001.

[45] I. Parberry. Designer Worlds: Procedural Generation of Infinite Terrain from Real-World Elevation Data. *Journal of Computer Graphics Techniques*, 3(1):74–85, 2014.

[46] J. T. Perron, W. E. Dietrich, and J. W. Kirchner. Controls on the Spacing of First-Order Valleys. *Journal of Geophysical Research*, 113, 2008. F04016.

[47] A. Peytavie, E. Galin, J. Grosjean, and S. Merillou. Arches: a Framework for Modeling Complex Terrains. *Computer Graphics Forum*, 28(2):457–467, 2009.

[48] L. Piccini. Recent Developments on Morphometric Analysis of Karst Caves. *Acta Carsologica*, 40(1):43–52, 2011.

[49] P. Prusinkiewicz and M. Hammel. A Fractal Model of Mountains with Rivers. Graphics Interface, pages 174–180, 1993.

[50] A. Pytel. Creating Artistic Compositions Using Coalition-Forming Intelligent Agents. Technical report, University of Waterloo, 2008. CS-2008-23.

[51] A. Pytel and S. Mann. Self-Organized Approach to Modeling Hydraulic Erosion Features. *Computers & Graphics*, 37(4):280–292, 2013.

[52] E. Rauch. Fractal Brownian Islands Exhibit. `http://groups.csail.mit.edu/mac/users/rauch/islands/`. Accessed on 10/19/2011.

[53] W. T. Reeves. Particle Systems — a Technique for Modeling a Class of Fuzzy Objects. *ACM Transactions on Graphics*, 2(2):91–108, 1983.

[54] L. F. Richardson. The Problem of Continuity. *General Systems Yearbook*, 6:139–187, 1961.

[55] I. Rodríguez-Iturbe and A. Rinaldo. *Fractal River Basins*. Cambridge University Press, 1997.

[56] P. Roudier and B. Peroche. Landscapes Synthesis Achieved through Erosion and Deposition Process Simulation. *Eurographics*, 12(3):375–383, 1993.

[57] B. Rusnell, D. Mould, and M. Eramian. Feature-rich Distance-based Terrain Synthesis. *Visual Computer*, 25(5–7):573–579, 2009.

[58] B. Sapoval, A. Baldassarri, and A. Gabrielli. Self-Stabilized Fractality of Seacoasts through Damped Erosion. *Physical Review Letters*, 93(9), 2004.

[59] D. Saupe. Algorithms for Random Fractals. In *The Science of Fractal Images*, pages 71–113. Springer-Verlag, 1988.

[60] S. Schlechtweg, T. Germer, and T. Strothotte. RenderBots — Multi Agent Systems for Direct Image Generation. *Computer Graphics Forum*, 24:283–290, 2005.

[61] J. A. Sethian. Theory, Algorithms, and Applications of Level Set Methods for Propagating Interfaces. Technical report, University of California, Berkeley, 1995.

[62] A. N. Strahler. Hypsometric (Area-Altitude) Analysis of Erosional Topography. *Geological Society of America Bulletin*, 63(11):1117–1142, 1952.

[63] R. Szeliski and D. Terzopoulos. From Splines to Fractals. *SIGGRAPH Computer Graphics*, 23(3):51–60, 1989.

[64] D. G. Tarboton. A New Method for the Determination of Flow Directions and Upslope Areas in Grid Digital Elevation Models. *Water Resources Research*, 33(2):309–319, 1997.

[65] S. T. Teoh. RiverLand: An Efficient Procedural Modeling System for Creating Realistic-Looking Terrains. ISVC, pages 468–479. Springer-Verlag, 2009.

[66] D. M. Tortelli and M. Walter. Modeling and Rendering the Growth of Speleothems in Real-time. In *GRAPP*, pages 27–35, 2009.

[67] L. Velho and D. Zorin. 4–8 Subdivision. *Computer-Aided Geometric Design*, 18(5):397–427, 2001.

[68] R. F. Voss. Fractals in Nature: From Characterization to Simulation. In *The Science of Fractal Images*, pages 21–69. Springer-Verlag, 1988.

[69] O. Šťava, B. Beneš, M. Brisbin, and J. Křivánek. Interactive Terrain Modeling Using Hydraulic Erosion. SCA, pages 201–210. Eurographics Association, 2008.

[70] M. Ward. Fractal Islands. `http://davis.wpi.edu/~matt/courses/fractals/islands.html`. Accessed on 10/19/2011.

[71] B. T. Werner. Eolian Dunes: Computer Simulations and Attractor Interpretation. *Geology*, 23(12):1107–1110, 1995.

[72] H. Zhou, J. Sun, G. Turk, and J. M. Rehg. Terrain Synthesis from Digital Elevation Models. *IEEE Transactions on Visualization and Computer Graphics*, 13(4):834–848, 2007.