Feature-Rich Distance-Based Terrain Synthesis

A Thesis Submitted to the
College of Graduate Studies and Research
in Partial Fulfillment of the Requirements
for the degree of Master of Science
in the Department of Computer Science
University of Saskatchewan
Saskatoon

By
Brennan Rusnell

Permission to Use

In presenting this thesis in partial fulfilment of the requirements for a Postgraduate degree from the University of Saskatchewan, I agree that the Libraries of this University may make it freely available for inspection. I further agree that permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis work or, in their absence, by the Head of the Department or the Dean of the College in which my thesis work was done. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of Saskatchewan in any scholarly use which may be made of any material in my thesis.

Requests for permission to copy or to make other use of material in this thesis in whole or part should be addressed to:

Head of the Department of Computer Science
176 Thorvaldson Building
110 Science Place
University of Saskatchewan
Saskatoon, Saskatchewan
Canada
S7N 5C9

Abstract

This thesis describes a novel terrain synthesis method based on distances in a weighted graph. The method begins with a regular lattice with arbitrary edge weights; heights are determined by path cost from a set of generator nodes. The shapes of individual terrain features, such as mountains, hills, and craters, are specified by a monotonically decreasing profile describing the cross-sectional shape of a feature, while the locations of features in the terrain are specified by placing the generators. Pathing places ridges whose initial location have a dendritic shape. The method is robust and easy to control, making it possible to create pareidolia effects. It can produce a wide range of realistic synthetic terrains such as mountain ranges, craters, faults, cinder cones, and hills. The algorithm incorporates random graph edge weights, permits the inclusion of multiple topography profiles, and allows precise control over placement of terrain features and their heights. These properties all allow the artist to create highly heterogeneous terrains that compare quite favorably to existing methods.

Acknowledgements

I would like to thank Dr. David Mould and Dr. Mark Eramian for their continued support and guidance as my supervisors. This work would not have been possible without their aid.

Contents

List of Tables

List of Figures

List of Abbreviations

1D	one-dimensional
2D	two-dimensional
3D	three-dimensional
CC	Creative Commons
CPU	central processing unit
DEM	Digital Elevation Model
ED	edge detection
EW	edge weight
fBm	fractional Brownian motion
GPU	graphics processing unit
ID	identification
IMG	Imaging, Multimedia, and Graphics
I/O	input/output
MRF	Markov Random Field
NASA	National Aeronautics and Space Administration
OpenGL	Open Graphics Library
RMF	ridged multifractal
SBM	sketch-based modelling
U of S	University of Saskatchewan
USA	United States of America
USGS	United States Geological Survey

Chapter 1

Introduction

The term *terrain* describes the physical or surface features of a tract of land. The aesthetics of terrains have been well captured in photography and well researched in computer graphics. Landscapes were photographed from the advent of photography in the mid-19th century because they are a static medium well suited for the approximately eight hour exposure times of the first cameras. In 2002, London et al. observed that "in addition, people were interested in faraway places; they traveled to them when they could and bought travel books and pictures when they could not" [47]. Examples of such pictures, which capture the Earth's beauty, are shown in Figure 1.1 [1, 38].

The advent of computer graphics ushered in a transition from 2D to 3D; in addition to 2D imagery such as landscape photography, the synthesis of 3D models depicting such scenes was now possible. The ability to realistically synthesize and render, or capture, such models has been increasing as computer hardware and software become more sophisticated. Computers can be used as a tool to synthesize and capture terrains. Methods for terrain synthesis have been increasing in realism, resulting in an increased ability to capture the Earth's beauty. This thesis presents a new terrain synthesis method, the goal of which is to create realistic terrains that capture this beauty.

A sub-domain of computer graphics is concerned with the realistic synthesis of terrains. Terrain synthesis involves the creation of a model that approximates a terrain resulting from erosional and geological processes. It is an important problem because synthetic terrains are widely used in film and video games. Such models can depict realistic terrains found in nature, or completely fictional terrains. The goal of terrain synthesis algorithms is to be a realistic method that is able to synthesize a variety of features, such as peaks, ridges, hills, and valleys, within a single terrain. Furthermore, it must be easy to control feature placement and shape.

(a) (b)

Figure 1.1: Two landscape images. Left, a photo of Banner Peak [38], located at Thousand Island Lake in California's Sierra Nevada. Right, a photo of Bicaz Canyon [1], located in north-east Romania.

Terrain models can be created by hand, or procedurally by terrain synthesis algorithms that have been developed over the past few decades. Creating models by hand involves using modelling software such as 3D Studio Max (Autodesk, Inc.), Maya (Autodesk, Inc.), Blender (Stichting Blender Foundation), and Lightwave (NewTek, Inc.). Terrain synthesis algorithms include procedural methods such as fractional Brownian motion (fBm) and the ridged multifractal (RMF) terrain model, physically-based methods that simulate erosion, and image-based methods that, among other techniques, patch terrain samples together.

Existing terrain synthesis methods suffer from one or more of the following drawbacks: are difficult to control, produce homogeneous results, or are time consuming. Homogeneous results are undesirable because realistic terrains are heterogeneous: foothills reside at the base of jagged mountains, faults can occur in flat terrain. Procedural and physically-based methods produce realistic, heterogeneous terrains, but are difficult to control. Image-based methods require an initial set of terrains and synthesize a new terrain that is similar to the examples. New terrain is synthesized, but the input sample terrain must be obtained from some other process, such as the aforementioned procedural and physically-based methods, or satellite elevation data. Furthermore, the synthesized terrain contains only the features of the samples – new features cannot be generated and the result is a relatively homogeneous terrain. The creation of terrain by hand can produce realistic results and provides user control over feature placement, but the method requires much human time and skill. The existing literature leaves room for improvement in terrain synthesis, providing an opportunity for new methods to emerge.

1.1 Solution

This thesis addresses the shortcomings of existing methods. It does so by presenting a new procedural terrain synthesis method, based on distances in a weighted graph, that provides control over feature placement and shape. It can synthesize a wide variety of terrains found in nature, such as mountains, hills, and valleys. Furthermore, it can synthesize a variety of terrains that existing methods cannot; such features include craters, faults, and volcanoes. Feature placement can be manually or procedurally controlled; manual feature placement involves the creation of marking in an input image. Feature shape is controlled by hand-drawn and/or procedural *profiles*; a *profile* describes the cross-sectional shape of a feature and offers user control over terrain feature shape. This high level of control also facilitates the synthesis of terrains whose shadows embed hidden images for the creation of pareidolia effects. Pareidolia refers to the phenomenon where a vague or imperfect sensory input is mistakenly interpreted as something familiar, such as a human face.

The algorithm incorporates random graph edge weights, permits the inclusion of multiple topography profiles, and allows precise control over placement of terrain features and their heights. These properties all allow the artist to create highly heterogeneous terrains that compare quite favorably to existing methods.

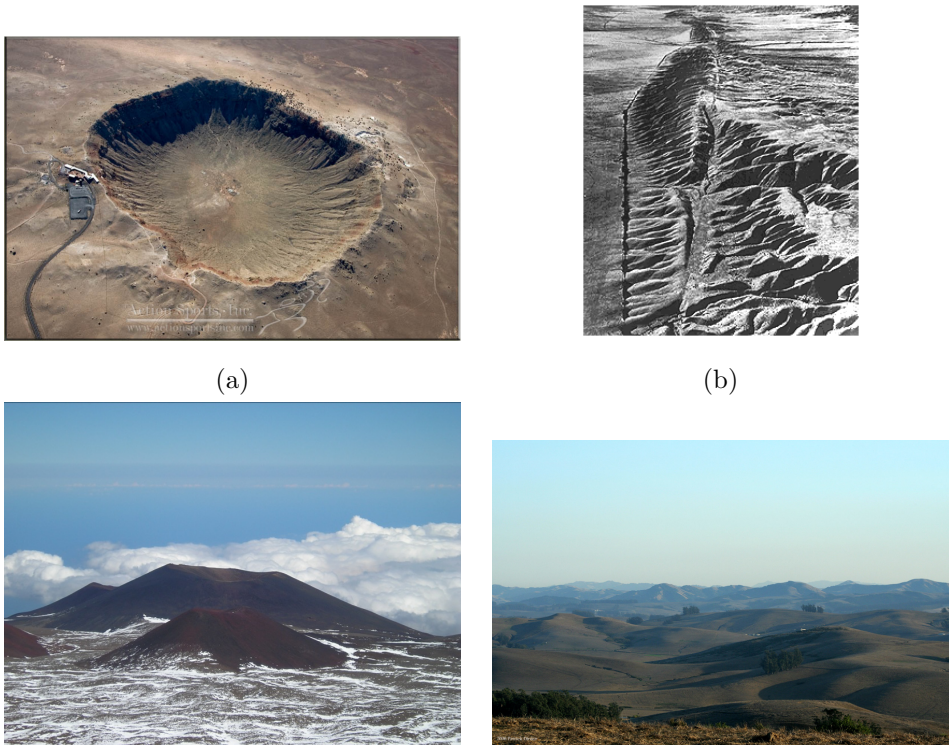
1.1.1 Reference Terrains

Figure 1.2 shows reference photos for the crater, fault, cinder cone volcano, and rolling hills renders in Chapter 4. These images serve as a basis of comparison for terrain shape: craters form large bowl-shaped depressions, cinder cone volcanoes have a flat top with a smaller bowl-shaped depression at the peak, faults form unnatural, sharp breaking features at the Earth's surface, and hills are smooth and rounded. When comparing the synthesized terrains to the provided reference photos, the reference image is considered to be an exemplar of the feature. We are not attempting to synthesize a clone of the reference image, in terms of feature location and height, but we are attempting to synthesize features that are also sensible exemplars. We are interested in what ways it reflects the ideal exemplar, and in what ways it does not. It is important that a terrain synthesis method capture mountains, hills, and valleys, though such a method should not be limited to them. Figure 1.2 shows formations that are also desirable to capture. Such formations, as well as others, are presented in Chapter 4.

1.2 Overview

Chapter 2 of this thesis begins with a discussion of related and previous work in the areas of procedural-based, physical-based, and image-based terrain synthesis methods. Attention will then turn to path planning and sketch-based modelling. Chapter 3 focuses on general terrain synthesis using the proposed method. The section begins with an overview of the method, followed by a thorough description of its inputs, parameters, and profile creation. The chapter concludes with an in-depth discussion of the proposed algorithm.

Chapter 4 focuses on the results created using the proposed method. The results are categorized as: realistic, image driven, or sketch-based. Realistic terrain synthesis consists of topographies found in nature such as craters, volcanoes, faults, hills, mountain ranges, tower karsts, and lunar landscapes. Image driven terrain synthesis consists of terrains whose features are placed using edge detection, and/or whose edge weights are calculated according to an input texture. In sketch-based terrains, coloured pixels in an input image indicate feature location, and each colour corresponds to an input profile.



(a) (b)

(c) (d)

Figure 1.2: These four reference images are a subset of the final set of reference images used in this work. Image (a) [6] shows a crater, image (b) [86] shows the San Andreas Fault, image (c) [35] shows a cinder cone volcano, and image (d) [26] shows some rolling hills.

Motivation for each result is given – why each terrain was made, what each terrain demonstrates, how each terrain was made, and specific input/parameter values to achieve each result will be presented. Also, each realistic terrain is accompanied by a reference photo for comparison.

Next, Chapter 5 introduces the phenomenon of pareidolia. Two terrains synthesized to embed imagery in their shadows are presented and discussed. The final chapter of the thesis, Chapter 6, formulates conclusions and summarizes the main contributions of this work. Ideas for future consideration are also presented. These ideas, among others, can increase the diversity of synthesized terrains by incorporating more features, such as bodies of water and rock strata, and by increasing the heterogeneity of terrains via different edge weight calculations.

Chapter 2

Related Work

2.1 Procedural Terrain Synthesis Methods

Much of the early terrain synthesis literature concentrates on procedural methods. A procedural method is an algorithm that specifies some facet of a computer-generated model [29]. Early work on procedural terrain synthesis methods began with the generation of fractal shapes.

Mandelbrot [49] termed *fractal* to mathematically describe certain natural phenomenon. Fractal refers to a family of shapes that are self-similar. If an object is self-similar, each piece of the object is geometrically similar to the whole – the degree of irregularity and/or fragmentation is identical at all scales. In other words, a fractal is "a geometrically complex object, the complexity of which arises through the repetition of a given form over a range of scales (or sizes)" [29]. Fractals introduce a fractal dimension D: a D-dimensional self-similar object can be divided into N smaller copies, each of which is scaled down by a factor of r, where $N = 1/r^D$. Fractal dimension extends the idea of integer based Euclidean dimensions to real-numbered values. The decimal part of D, also known as the *fractal increment*, determines the visual complexity of the fractal, and the non-decimal part of D indicates the underlying Euclidean dimension of the fractal. For example, as D increases from 2 to 3, the resulting shape progresses from locally planar to locally occupying some volume of a 3D space. Fractals are a language for describing shapes and phenomena common in nature. However, such shapes and phenomena are fractal only over some limited range of scales, making them band-limited.

Mandelbrot [49, 51] then proposed fractional Brownian motion (fBm), or $1/f^\beta$ surfaces; the spectral exponent β controls the fractal dimension of the function. Fractional Brownian motion is the archetypal fractal procedural method [29] and is a member of the class of *fractional noises* [51]. Fournier et al. defines such noises as, "the contribution of each frequency to the power spectrum is nearly inversely proportional to the frequency" [33]. Musgrave's [56] formuation for fBm is given in Algorithm 1. This formulation is applied at each point p in the terrain, and is controlled by three parameters: the Holder exponent H, lacunarity, and octaves. Terrain smoothness increases with H and small-scale detail increases with octaves. Lacunarity provides the gap between successive frequencies and Musgrave notes that in most cases, lacunarity can be fixed at 2.0; a value of 2.0

means that each successive frequency is doubled. Also, Perlin noise [68] is used in Algorithm 1 because it is monochromatic (single frequency), stationary (invariant under translation), and isotropic (invariant under rotation). In Perlin noise, integer points of a lattice or grid are assigned a random gradient value and non-integer points are defined by interpolation via a cubic function. However, the value of the function at the integer points is zero. Algorithm 1 maintains the previously mentioned inverse relationship between power spectrum contribution and frequency. Unfortunately, it is statistically homogeneous and isotropic, and is difficult to control – fBm does not provide a method for controlling where terrain features reside, and understanding how a change to H and/or lacunarity affects the final terrain is a difficult process.

Algorithm 1 fBm(p, H, lacunarity, octaves)

$value \leftarrow 0$
for $i \leftarrow 0$ to $octaves$ **do**
  $value \leftarrow value + Noise(p) \cdot lacunarity^{-H \cdot i}$
  $p \leftarrow p \cdot lacunarity$
end for
return $value$

In contrast, Mandelbrot generated fBm using Poisson faulting. Poisson faulting involves the application of Gaussian random displacements such as faults, or step functions, to a plane or sphere at Poisson distributed intervals; Poisson faulting is well suited for producing naturally occurring rough surfaces, such as mountains, with little overhead. Voss [84] called such surfaces fractal forgeries of the natural world and noted that their success plays an important role in the rapid establishment of fractal geometry as a new scientific discipline and graphic technique.

Like Mandelbrot, Fournier et al. [33] wanted to represent natural irregular objects and phenomena with minimal overhead. The problem was solved by modelling objects, such as fBm terrains, as sample paths of stochastic processes. Additionally, Fournier et al. introduced a realistic, visually satisfactory approximation to fBm that can be computed in less time than the exact solution. A major advantage of Fournier et al.'s approximation is that it facilitates surface computation to arbitrary levels of detail without increasing the storage overhead. Unfortunately, the method still suffers from the same drawbacks as fBm: homogeneous results and no control over terrain feature placement.

Other approximations include Fourier filtering [67, 84], midpoint displacement [33], and successive random additions [84]. Fourier filtering works by taking the Fourier transform of a 2D Gaussian white noise, multiplying it in frequency space with an appropriate filter, and interpreting the inverse Fourier transform of the product as a height field. A transfer function proportional to $1/f^{\beta/2}$ must be used [84]; a transfer function specifies the spatial or temporal frequency relationship between the input and output of a system.

Midpoint displacement, or recursive subdivision, methods are an approximation to fBm which recursively subdivide an interval and generate a scalar value at the midpoint. These scalar values, or displacements, are distributed according to a Gaussian distribution [50]. Once the value at a point is determined, it remains fixed. Furthermore, at each stage of the process, only half of the points are determined more accurately [84]. Unfortunately, midpoint displacement methods generate features that catch the eye and, by their length and straightness, make the rendered scene look unnatural. This property is known as the creasing problem [50]. More specifically, points generated at different stages have different statistical properties in their neighbourhoods, which often leave a visible trace that does not disappear as more stages are added. Additionally, the final result of midpoint displacement methods is highly dependent on the displacements at early intervals. In contrast, successive random additions adds randomness to all points at each stage of a recursive subdivision process [84] and is useful for continuously variable levels of detail. Unfortunately, terrain feature control is still an issue in both midpoint displacement and successive random addition methods.

Similar in construction to midpoint displacement methods, Lewis [46] argued that spectral modelling provides a more powerful and intuitive perceptual characterization of random processes than does the fractal model. In spectral modelling, the user is able to control the spectral properties of the synthesized random functions. As a result, Lewis presented generalized stochastic subdivision, the basis of which is a midpoint estimation problem: given two samples of the noise, a new sample midway between the two is estimated as a weighted sum of the known noise values from the previous stages of the construction, in some neighborhood. This generalized construction was suitable for generating a variety of perceptually distinct, high-quality random functions, including those with non-fractal spectra and directional or oscillatory characteristics. However, feature placement control still remained an issue and the implementation was more complex than that of other methods.

Mandelbrot [50] then observed and addressed the symmetry of valleys and mountains resulting from the symmetry of the Gaussian distribution used in midpoint displacement methods. The solution involved replacing the Gaussian with suitable non-symmetric distributions, such as the the gamma distribution. In a gamma distribution, most of the area under the density function is located near the origin, and the density function drops gradually as x increases (in a 2D plot); examples of its use include queuing models [85]. Using a gamma distribution resulted in an increase in realism.

In addition to addressing the symmetry of valleys and mountains, Mandelbrot [50] also noted that a basic defect in past fractal forgeries of landscapes is the failure to include river networks. In response to this defect, Mandelbrot presented a fully random combined model of rivers and of mountains based on midpoint displacements and fractal curves. Midpoint displacements provide the relief along watersheds, and fractal curves provide the relief along the rivers; watershed refers to a ridge of land that separates adjacent river systems. More specifically, an input map showing the locations of rivers and watersheds is used to guide/constrain this random model. Mandelbrot's method provided terrain feature control, but it was used only for rivers and watersheds. Furthermore, the randomness of the model limited this level of control.

Another contribution by Musgrave et al. [55, 56] came in response to a known drawback of fBm. They noted that the statistical character of fBm surfaces is the same everywhere, an undesirable property that does not capture the majority of topographies found in nature. As a result, Musgrave et al. presented a new synthesis model called noise synthesis which provides locally independent control of the frequencies composing the surface, and thus local control of fractal dimension D. In Musgrave et al.'s work, each point in the terrain is determined procedurally, independently of its neighbours. An example of point evaluation for fBm is given in Algorithm 1. The evaluation of points is a distinguishing property of the noise synthesis method. Musgrave et al. approximated eroded landscapes by varying the fractal dimension D with altitude, as well as other functions. Their model incorporated arbitrary lacunarity, varied terrain smoothness and asymmetry, and addressed the issue of creases and periodicity in previous works. Periodicity is commonly seen in terrain synthesis methods based on Fourier filtering [55]. Musgrave et al.'s work was an major step forward which addressed many important issues in previous works. Unfortunately, feature control was not addressed and remained an issue.

Musgrave then formulated another important contribution to procedural terrain synthesis known as the ridged multifractal (RMF) terrain model [56]. The RMF model is a well known terrain synthesis method that addresses the homogeneity of fBm by producing heterogeneous terrains with valleys at varying altitudes. However, the RMF model is not adept at synthesizing extended features such as ridges, and lacks various specific features such as craters and cinder cones. Furthermore, the model is controlled by difficult to use parameters, making feature placement a tedious process.

Similar to Mandelbrot's combined model of rivers and mountains, Prusinkiewicz and Hammel [70] addressed the problem of combining fractal mountains with rivers. As a result, they presented a partial solution that incorporated a squig-curve model of a river's course into the midpoint displacement model for mountains. Mandelbrot [49] used *squigs* to refer to a family of fractal shapes. Some of these fractals are self-avoiding and nonbranching curves while others are loops or trees. Mandelbrot noted that the simplest squig-curve "is a model of a river's course, patterned after the well-known pictures in geology or geography that show the successive stages of a river

that burrows into its valley, defining its course with increasing precision". Both squig-curve and midpoint displacement models were observed to be expressed by similar context-sensitive rewriting mechanisms. As a result, a mountain landscape with a river could be generated using a single integrated process. However, terrain synthesis itself was not addressed, and the known drawbacks to midpoint displacement methods still existed.

In addition, Sapozhnikov and Nikora [74] proposed a model for river networks based on a random walk. Their approach produced individual streams that display self-similar behavior at shorter lengths and self-affine behavior at longer lengths; self-affine refers to a fractal whose components are scaled by different amounts in each direction. Similar behavior was also observed for simulated river networks. Again, actual synthesis of the surrounding terrain was not addressed, so known issues in current terrain synthesis methods still existed.

An important attribute of our work is user control over terrain feature placement. User control over terrain feature placement has been addressed by Szeliski and Terzopoulos [82]. Szeliski and Terzopoulos combine variational splines and stochastic fractals to produce realistic, controllable terrains; the splines serve as the interpolant between the input elevation values and provide the fractal shape. Unfortunately, local control over the continuity of the spline is provided via difficult to use parameters of a deformational energy functional. Szeliski and Terzopoulos's work and this thesis both use a sparse set of known elevation values and algorithmically determine the remaining elevations, in the former case by interpolating using splines, and in the latter case by extrapolating using least-cost paths.

Much of the recent work on procedural terrain synthesis has focused on efficient processing. This is in response to the increase in CPU and GPU processing power. The work by Musgrave et al. [55, 56] has remained largely unchanged. Select works include that of Dollins [27] and Schneider [75]. Dollins worked on the authoring and emulation of highly interactive, large-scale synthetic environments and Schneider developed a new GPU method for real-time editing, synthesis, and rendering of infinite landscapes exhibiting a wide range of geological structures. However, the problem of control still remains an issue with previous procedural methods. Also, specific features such as craters, cinder cones, and faults, cannot be synthesized with these methods.

2.2 Physical Terrain Synthesis Methods

Physically-based synthesis methods simulate actual erosional processes. Most algorithms in this category can each be divided into four distinct and independent steps [9]. These methods proceed by 1) distributing water on the terrain's surface, 2) eroding the underlying terrain structures, 3) transporting the material captured in the water, and 4) depositing this material. Because the steps are independent, any step can be run an arbitrary number of times to simulate heavy rains, or dry

seasons. Additionally, many physically-based methods are inspired by Musgrave's [55, 56] simple physical erosion model that simulates hydraulic and thermal erosion processes to create global stream/valley networks and talus slopes; talus slopes consist of rock debris located at the base of a cliff or steep mountain slope [53].

Prior to Musgrave's work, Kelley et al. [42] simulated the erosion of stream networks on an initially uneroded surface. In this context, stream erosion simulation was based upon empirical geomorphology models; empirical geomorphology is the study of landscape development [53]. Nagashima [57] then improved upon these erosion models. Valley and mountain terrains were created in this work and were based on erosion due to river flows, rainfall, and weathering. Terrains with differing surfaces were created by adjusting the erosion intensities of rainfall and thermal weathering.

Chiba et al. [14] noted that terrains created using fBm or midpoint displacement cannot create clear ridge and valley lines, both of which are important characteristics of mountains resulting from erosional processes. To solve this problem, they presented a quasi-physically based method for simulating the topography of eroded mountains based on velocity fields of water flow.

Benes and Forsbach [8] developed a new data structure for the visual simulation of terrain erosion. Their representation was inspired by real geological measurements and acted as a good trade-off between height fields and memory demanding voxel representations. The classical erosion algorithm by Musgrave [55, 56] was supported by this new representation and a new property could be simulated: their algorithm could capture the conversion of non-deposed material (very dense) to dust (less dense).

Benes and Forsbach [9] followed this work by developing a new algorithm for hydraulic terrain erosion. Their main goal was to provide a technique which is inspired by physics and allows for a high level of control. Unfortunately, their method was non-interactive. Neidhold et al. [61] addressed this issue by developing an interactive system for physically based fluid and erosion simulation. The key attribute of this work was that the artist is able to influence the erosion process in real-time by changing the simulation parameters or by applying additional water to the scene.

Finally, Benes et al. [10] generalized earlier work [9] on modelling hydraulic erosion. This generalization was made possible by using ideas from fluid mechanics. Specifically, the model was based on the Navier-Stokes equations, which provided the dynamics of velocity and pressure. The resulting model was fully 3D and was able to simulate a variety of phenomena including meandering streams, low hill sediment wash, natural water springs, and receding waterfalls.

The previously described erosional-based methods focus on the erosion process itself and are usually quite complex [9]. Furthermore, the techniques are not well related to physics – they use a large number of constants that influence each other and a numerically stable implementation of these techniques is not easy to formulate. Furthermore, some algorithms tend to result in

water oscillating between two states as a result of the simple underlying water transport model. In contrast, more control has been added over the years, such as in the work by Neidhold et al. [61]. In relation to procedural and image-based terrain synthesis, the results possible with these methods are more accurate because they are based on physics engines. However, designing a terrain via water simulation parameters and water volume is not as easy to use as simply placing features via markings in an input image. Furthermore, non-erosional features, such as craters and cinder cones, have yet to be addressed by these methods.

2.3 Image-Based Terrain Synthesis Methods

Image-based terrain synthesis methods begin with an initial set of terrains and synthesize a new terrain that is similar to the examples. New terrain is synthesized, but a major drawback to these methods is that the input sample terrain must be obtained from some other process, such as the aforementioned procedural and physically-based methods, or satellite elevation data. Unfortunately, the final set of terrain features is limited to those in the provided example textures – new features cannot be synthesized. Furthermore, the resulting terrains tend to be homogeneous because of the methods used to incorporate the input textures, such as non-parametric and patch-based sampling. The common representation for terrains in these methods is a height field: a 2D scalar field where the field value is interpreted as vertical distance.

For terrain synthesis, Arakawa and Krotkov [5] worked on surface reconstruction with range data; range data refers to natural terrain patterns acquired by a laser rangefinder. They initially estimated the fractal dimension of natural surfaces given range data and then reconstructed natural surfaces using this estimate and the given sparse range data. Their approach extended fBm to incorporate the sparse data, and imposed roughness constraints in order to create natural surfaces.

Dachsbacher et al. [18, 19] presented a method to synthesize or grow height-fields from an initial input field (they demonstrate their model using Efros and Leung's *Texture Synthesis by Non-Parametric Sampling* [30]) and describe how it can be adapted to height field synthesis. Non-parametric sampling [30] grows a texture by matching the neighbourhoods of the pixel to be synthesized with pixels in the input texture. Textures are modelled as a Markov Random Field (MRF): the probability distribution for the brightness of a pixel depends on the brightness values of its spatial neighbourhood. The neighbourhood is a square region about a given pixel, and it is assumed that the input texture is regular at high spatial frequencies and stochastic at low spatial frequencies.

Dachsbacher et al. treat a height field as a texture and apply non-parametric sampling to synthesize new terrain, but note that their method is not restricted to Efros and Leung's work and simply use it as a proof of concept. Similar to Arakawa and Krotkov's work, the use of non-parametric sampling allows gaps to be extended or filled in satellite elevation data and allows transitions to be computed between different procedural models.

As previously mentioned in Section 2.1, Szeliski and Terzopoulos [82] addressed user control over terrain feature placement. An image-based method that also addresses user control is the work of Zhou et al. [92]. Their work presented an example-based system for terrain synthesis. Specifically, patches from a provided sample height field are joined using graph cuts to generate new terrain. The synthesis is guided by a user-sketched feature location map that specifies where terrain features occur in the resulting synthetic terrain. This synthesis method addresses the issue of feature control, but still suffers from the same problems as other image-based methods: dependence on an input sample terrain, new terrain features cannot be synthesized, and relatively homogeneous results. These dependencies limit the range of results possible with Zhou et al.'s method. Furthermore, heterogeneous terrains, resulting from non-stationary textures, are difficult to synthesize using patch-based sampling. User control has been addressed, but usage of an input sample terrain severely limits the diversity of the results.

Further image-based work involved that of Brosz et al. [11] and Yu and Chang [90]. Brosz et al. developed a terrain synthesis technique that uses an existing terrain to synthesize new terrain. This is accomplished by first using multiresolution analysis to extract the high-resolution details from existing models. Then, these details are applied to the original terrain to increase its resolution, retaining large-scale characteristics of the original terrain.

Yu and Chang developed shadow graphs for surface reconstruction – an image-based method that motivated the synthesis of terrains for pareidolia effects in Chapter 5. In their work, a set of images from a fixed viewpoint is taken as input, as well as the height values for a small number of pixels. Each image contains the shadows cast by some unknown surface from a unique light source location. The method then generates a height field for the unknown surface using these inputs. Specifically, the shadow graph is built from shadow constraints for which an upper bound at each pixel can be derived using the provided absolute height values; these height values can be recovered from other approaches such as stereo processing. Finally, the results from shape-from-shading are made consistent with the aforementioned upper bounds using a constrained optimization procedure. Chapter 5 introduces the synthesis of terrains that embed imagery in their shadows. Both this thesis and the work of Yu and Chang rely on input images for the creation of pareidolia effects and surfaces, respectively. However, shadow graphs use input images as the sole source of constraints – many images are required to generate enough constraints for a given surface. In Chapter 5, this thesis uses one constraint image for pareidolia effects, but additional constraints result from the fact that

the final terrain must look realistic. In contrast, Yu and Chang place relatively little emphasis on what the reconstructed surface looks like, so long as it accurately represents the shadow data.

2.4 Path Planning

The terrain synthesis method proposed in Chapter 3 creates a height field using path planning. Winston [87] describes path planning as the process of finding a least-cost traversal through a weighted graph; this thesis uses Dijkstra's algorithm [64] to calculate such least-cost traversals. Prior to this work, path planning has not been used for terrain synthesis. The majority of the literature attempts to solve robot navigation. However, path planning has recently been used as an artistic tool [48, 89]. Both types of applications will be discussed in turn. However, before these applications are presented, Dijkstra's algorithm is discussed.

Dijkstra's algorithm finds the shortest path in a weighted graph G, consisting of nodes N and edges E; the cost of a node n is represented by c_n and the weight of an edge e is represented by w_e. Our application of Dijkstra's algorithm in this thesis is described in Chapters 3 through 5. The algorithm maintains a set of nodes T, rooted at a generator node n_g, that consists of all nodes with a tentative cost; the set T is known as the frontier. The algorithm begins by setting $c_{n_g} = c_{init}$ and the cost of all remaining nodes to ∞, where c_{init} is the initial cost of the generator node. Then, the least-cost node n_o is removed from T and each edge e incident to n_o is examined. If the cost to n_e, going though n_o via edge e, is less than its existing cost, c_{n_e} is updated to reflect the smaller cost and n_e is added to T. This process repeats until T is empty. Dijkstra's algorithm is given in Algorithm 2.

Select examples of path planning for robot navigation include the work of Gewali et al. [36]. In their work, Gewali et al. considered the terrain navigation problem in a 2D polygonal subdivision consisting of obstacles, free regions (travelling without cost), and regions in which cost is proportional to distance traveled. The problem of interest was a generalization of the planar shortest path problem in the presence of obstacles.

Pai and Reissell [65] described an approach to motion planning for mobile robots on natural, non-homogeneous terrain. The path planning algorithm uses a non-scalar path cost measure based on the sorted terrain costs along the path and can be incorporated into standard global path search algorithms. The non-scalar path cost measure attempts to model terrain roughness, causing robots to avoid peak edge costs even if it means taking an alternate route with a higher total cost. This would typically result in a longer, more winding path.

Shortly thereafter, Chen [13] discussed different contexts for solving geometric shortest path problems, such as finding paths connecting different locations in geometric space and paths that optimize a given cost function. De Carvalho et al. [22] then described a complete coverage path

Algorithm 2 Dijkstra's Algorithm

$c_{n_g} \leftarrow c_{init}$
$c_n \leftarrow \infty, \forall n \in N \backslash n_g$
$T \leftarrow \{n_g\}$
while T not empty **do**
  $n_o \leftarrow$ remove minimum cost node from T
  for each edge e incident to n_o **do**
    $c \leftarrow c_{n_o} + w_e$
    $n_e \leftarrow$ node connected to n_o via e
    if $c < c_{n_e}$ **then**
      $c_{n_e} \leftarrow c$
      $T \leftarrow T + \{n_e\}$
    end if
  end for
end while

planning and guidance methodology for a mobile robot. The novelty of the proposed approach was the capability of the path planner to deal with prior mapped or unexpected obstacles.

Amato et al. [3] evaluated different distance metrics and local planners for planning the motion of rigid objects in 3D workspaces. Also, a new local planning method was developed which often outperformed similar, competing methods. Choset [15] focused on coverage path planning algorithms for mobile robots, and Kuffner [43] presented techniques for rigid body path planning whose paths need to meet specific criteria (eg. various distance metrics).

Performance of these algorithms quickly became important because path planning was being applied in real-time, such as video games [20]. Niederberger et al. [62] addressed the need with a fast/real-time and robust path planning algorithm for generic static terrains with polygonal obstacles. Similarly, Kallmann [41] developed techniques for efficiently computing collision-free paths in a triangular planar environment. A triangular planar environment facilitated much more efficient planning, when compared to grid-based environments.

Path planning as a method for image and model synthesis has recently been explored by, among others, Davis [21], Xu and Mould [89], and Long and Mould [48]. Similarly, Worley [88] created cellular textures, a texture synthesis method based on distances from points in an image. Our terrain synthesis method is based on least-cost paths – a more complex function of distance. This thesis uses Xu and Mould's path planning algorithm [89] for the synthesis of terrains. Least-cost paths to every non-generator node, from a set of generator nodes, are used to define the field value at each index in the height field. Each of the aforementioned artistic methods is now presented.

Davis addressed the problem of creating mosaics in the presence of moving objects. Existing methods have focused on capturing static scenes, but Davis's work remained unbiased by movement (i.e. image registration accounted for translation and rotation) and avoided blurred areas due to moving objects; standard compositing techniques produce a blurred image in moving regions. Of particular importance is Davis's method of compositing images. In the compositing stage, each source image is compared to the mosaic created thus far and the best path dividing the overlapping section is found using Dijkstra's algorithm [64]; edge weights are drawn from the intensity difference in the overlapping section. On one side of this path the pixels from the mosaic are preserved; on the other, previous information is discarded in favor of samples from the current source image.

Long and Mould presented a procedural method for modelling stylized dendritic structures based on path planning. Their method included the partial non-scalar distance metric previously introduced by Pai and Reissell [65]. This non-scalar distance metric was used because it avoids taking short cuts over high cost areas, a characteristic of the traditional distance metric consisting of accumulated cost along the path. The avoidance of such short cuts preserves high frequency details that are important for artistic effects.

Similar to the work of Long and Mould, Xu and Mould presented a method for creating geometric models of dendritic forms. Their work first generates a regular lattice with random edge weights, then finds least-cost paths through the lattice using Dijkstra's algorithm. Multiple paths from a single generator are connected into a single dendritic shape. Alternatively, path costs can be used to segment volumes into irregular shapes. In this case, each lattice node stores the generator node it is closest to; the closest generator node is updated as Dijkstra's algorithm searches the graph. Then, a region is created for each generator, consisting of all the points nearest that node. Shapes are defined as the mesh marking the boundary of one of these regions.

Finally, the terrain synthesis method presented in this thesis calculates distances for terrain synthesis; distances are least-cost paths, from a set of generator nodes, in a weighted graph. Worley's cellular textures [88] uses distances as well, only for texture synthesis. Cellular textures are created by calculating Euclidean distances from randomly positioned points. These distances partition the resulting scalar field into cellular regions, where all the points within each region are closer to its defining point than any other point. These regions are exactly the regions given by a Voronoi diagram. Worley then extends this notion to calculating the distance between a given location and the random point that it is n^{th} closest to. In contrast, this thesis defines distances as least-cost paths through a weighted graph, and such distances can be computed from structures that need not be points.

The interpretation of path cost as a complex function of distance from a one or more generator nodes, along with control over generator placement, makes path-planning an ideal candidate for terrain synthesis (see Chapter 3). Terrain height can be interpreted as a function of distance from

terrain features such as ridges and mountain tops. Shortest paths are a mechanism for evaluating a complex function of distance that incorporates information embedded in a graph's edge weights, such as terrain roughness and style.

2.5 Sketch-Based Modelling

An image that defines the locations of terrain features such as peaks, ridges, and hills, is provided as input to the method proposed in this thesis. Thus, the markings in this image can be thought of as a method for model creation and manipulation. Sketch-based modelling (SBM) deals with the conversion of a user's freeform markings into 2D curves, or 3D models. Notable works include that of Akeo et al. [2], Igarashi et al. [40], and Zeleznik et al. [91]. Sketch-based modelling is not limited to 3D model creation though. It can also be used, among other applications, for 3D surface editing [44, 91] (based on a 2D painting technique), specifying 2D curves [7], and in 3D I/O devices [23, 24].

Akeo et al. allows users to scan real sketches into the computer where perspective lines, vanishing lines, and 3D cross sections are added to the digital sketch to aid in the creation of the resulting 3D shape. Finally, the scanned data is then projected onto the 3D mark-up to complete the process. Igarashi et al. provide a sketching interface for quickly and easily designing freeform models and were inspired by Zeleznik et al. [91] and Eggli et al. [31]. Their system uses real-time pen-and-ink rendering [52] for input.

Zeleznik et al. combine the rapid exchange of ideas in sketching and the detail of 3D computer modelling systems. The user sketches the important properties of any of a variety of 3D primitives and, following simple placement rules, the corresponding 3D primitive is instantiated in the 3D scene. Zeleznik et al. also used a variety of direct-manipulation interaction techniques for transforming 3D objects, similar to the work of Snibbe et al. [78] and Strauss and Cary [81]. Additionally, Zeleznik et al. exploited some simple flexible constrained manipulation techniques that are similar to those of Bukowski and Séquin [12].

Snibbe et al. developed a framework for creating interactive 3D environments. Their framework resulted in the development of *3D widgets*, or objects, called *racks* that encapsulate 3D geometry and behavior and control over other objects in the scene. *Racks* consist of a bar specifying the axis of deformation and multiple handles attached to the bar specify additional deformation parameters. Strauss and Cary presented an object-orientated toolkit for developers of interactive 3D graphics applications that facilitates techniques such as direct object manipulation. Picking an object consists of finding a path to the frontmost object under the cursor and manipulation consists of operations such as one-axis scale and one-axis translate.

Bukowski and Séquin described a software framework to aid in designing and implementing manipulation behaviors for objects in a 3D virtual environment. This framework disambiguates the mapping of 2D cursor motion to 3D object motion, determines a valid and desirable final location for the 3D objects, and relocating objects actively look for nearby objects to associate and align themselves with. A desirable final location and nearby object associations attempt to satisfy some of the physical realities of the environment. However, Zeleznik et al. had less semantic information than Bukowski which limited the automation of constraint generation. Therefore, the user is occasionally required to sketch constraints, similar to Gleicher [37], in addition to geometry. Gleicher integrated constraint and direct manipulation approaches for geometric modelling. Specifically, snapping techniques, such as grids, and constraint techniques are combined; grids establish geometrical relationships and constraint techniques maintain them. Constraints can then be edited by directly manipulating objects to show how they are to move (the desired effect updates the relationship), eliminating the need to refer to the constraint itself.

Pugh [71] developed a system similar to the work of Igarashi et al. and Zeleznik et al. that uses a constraint based approach to derive 3D geometry from 2D sketches. Unfortunately, the method was slow, difficult to implement, and could interpret only line drawings of objects with planar faces that correspond to a general view – a view where a small change in the view direction makes correspondingly small changes in the line drawing.

Interactive surface deformations via model painting, similar to Zeleznik et al.'s and Igarashi et al.'s ability to edit geometry, was addressed by Lawrence and Funkhouser [44]. Here, interactive manipulation and physical simulation are combined with a painting interface that gives the user direct, local control over a physical simulation. The painting interface allows the user to define the instantaneous surface velocity of a model, and by interactively simulating this velocity, the user can deform the surface. However, similar to using water as a tool for model manipulation in physically-based terrain synthesis [61], editing via velocities is not an easy process – mapping a velocity field to a specific model is difficult (eg. what is a chair's velocity field?). Furthermore, the underlying physical simulation model can cause the method to perform slowly at higher polygon counts.

Finally, Singh and Fiume [77] developed an implicit modelling technique called Wires. This work was inspired by armatures used by sculptors, in which *wire curves* give definition to an object and shape its deformable components. *Domain curves* were also introduced as a method for defining the domain of deformation about an object. Wires are bound to an object, directly reflect an object's geometry, and provide a coarse geometric representation of an object that can be created through sketching. A wire deformation is independent of the complexity of the underlying object and the animator can interact with the wires without ever having to deal directly with the object representation itself. Of particular interest is the method in which Singh and Fiume combine

deformations resulting from multiple wires whose domains intersect. Their solution is used in this thesis to blend individual terrain features together. This blending step removes creases/seams in the resulting synthetic terrain. The blend proposed by Singh and Fiume varies with a scalar blending bias b. The average of the deformations is calculated when $b = 0$, and the method converges to the maximum deformation for larger values of b. Their blending formulation has two desirable properties: 1) in a region with only one wire, the result is the deformation of just that wire and 2) when several wires produce the same deformation, the result is the deformation of any one of those wires. These properties are a nice fit for blending terrain features together. Furthermore, b can be re-interpreted as a method for controlling terrain roughness because larger values for b create more creases/seams in the resulting terrain.

The reviewed SBM literature consists of transforming input markings into 3D objects, followed by varying levels of object and constraint manipulation. The work presented in this thesis takes markings from an input image and, along with a description of terrain profile, converts them into terrain features – the markings indicate the location of peaks and/or ridges. Individual terrain features are then blended together according the work of Singh and Fiume [77]. However, this thesis does not facilitate the direct manipulation of the resulting synthetic terrain. This is an area for future consideration.

2.6 Summary

Work related to this thesis can be categorized as: Procedural Terrain Synthesis Methods, Physical Terrain Synthesis Methods, Image-Based Terrain Synthesis Methods, Path Planning, and Sketch-Based Modelling. Procedural terrain synthesis methods can synthesize heterogeneous terrains but lack direct control over feature placement (see Algorithm 1) and cannot render specific terrain features such as craters and cinder cones. Physical terrain synthesis methods create a more realistic result, but still lack direct control over feature placement. Furthermore, the final synthetic terrain is a function of an initial water distribution. Manipulating a model via a water distribution is not easy to do. Image-based terrain synthesis methods require input example height fields, but have addressed the problem of controlling terrain feature placement. Unfortunately, the quality of the resulting terrain is highly dependent on the quality of the input height field and these methods require another terrain synthesis method to be used before a new terrain can be created. Furthermore, image-based methods cannot synthesize new features – they are limited to the features present in the input textures. Path planning has shown promise in artistic/graphical applications, but has not been used for terrain synthesis.

Terrain height as a function of distance from terrain features such as ridges and mountain tops makes path planning a promising candidate for terrain synthesis. Finally, the success of input markings in sketch-based modelling makes the use of markings for terrain feature placement an attractive solution to the problem of terrain feature control.

The terrain synthesis method presented in this thesis addresses the known shortcomings of existing terrain synthesis methods. Terrain features are placed via easy-to-use input markings and the proposed solution incorporates random graph edge weights, permits the inclusion of multiple topography profiles, and allows precise control over placement of terrain features and their heights. These properties all facilitate the creation of highly heterogeneous terrains. Path planning over a weighted graph facilitates the combination of diverse features, and edge weights embed terrain roughness and style. Finally, the method does not require an input sample terrain that will serve as an exemplar for the resulting terrain, making the proposed algorithm an attractive new option for terrain synthesis.

Chapter 3

Terrain Synthesis

3.1 Overview

This chapter introduces the proposed terrain synthesis algorithm. The algorithm uses path planning to create synthetic terrains. Careful initialization of edge weights and generator node costs in a weighted graph G, consisting of nodes N and edges E, results in the creation of a cost field that can be interpreted as a height field. The nodes in N are arranged in an 8-connected grid corresponding to the spatial points on a height field and the edges in E are undirected. The aforementioned cost field is calculated by determining a series of shortest paths in G from a set of generator nodes, which define both the location and height of terrain features such as ridges and peaks. We use the term *stroke* to refer to a contiguous set of generator nodes that collectively define the height and location of a single terrain feature. A height field represents the synthesized terrain because it is simple to implement and has minimal storage overhead. This work focuses on control over terrain feature placement and terrain heterogeneity, not on synthesis speed, thus a height field is an appropriate choice for representing terrains.

Our method is similar to those of Zhou et al. [92], Szeliski and Terzopoulos [82], and Worley [88]. Zhou et al. presented an example-based system for terrain synthesis, where patches from an input height field and a user-sketched feature location map are used to synthesize new terrain. Szeliski and Terzopoulos combine variational splines and stochastic fractals to produce realistic, controllable terrains. Finally, Worley's cellular textures [88] use Euclidean distances from randomly positioned points as a basis for texture synthesis. Zhou et al.'s, Szeliski and Terzopoulos's, and our work aim to address the issue of terrain feature control. Furthermore, both Szeliski and Terzopoulos's and our work use a sparse set of known elevation values and algorithmically determine the remaining elevations. However, our method uses a different underlying approach based on distances in a weighted graph, similar to Worley's cellular textures. In addition, our work is not limited to the features found in an input terrain, accepts hand-drawn feature placement and terrain style, and makes it easy to create heterogeneous terrains by placing many diverse features within a single scene.

In our approach, smaller costs correspond to greater heights. This relationship results from the fact that our method provides user control over the location of terrain features such as ridges and peaks. Shortest path algorithms such as Dijkstra's algorithm assign a cost to all non-generator nodes. The cost of a node n is the least-cost path J's cost (sum of edge weights) to n from one or more generator nodes. As a result, the cost of the generator node that J originates at will be less than the cost of n. The previously mentioned relationship between cost and height is required to convert such lower-cost generator nodes into taller features such as peaks and ridges.

This chapter discusses the proposed terrain synthesis algorithm in detail. A simple example terrain consisting of 25 peaks and 38 ridges is used to illustrate inputs, parameters, and decisions made in this research. Section 3.2 introduces the inputs and parameters to the method and their impact on G, and Subsection 3.2.1 introduces a novel method for terrain shape control using profiles. Finally, Section 3.3 discusses each step of the proposed algorithm, as well as the the creation of scaling functions (Subsection 3.3.1) and the blending of individual height fields (Subsection 3.3.2).

3.2 Inputs and Parameters

The creation of the renders in Chapter 4 requires three inputs and seven parameters which provide control over the synthesized terrain. We distinguish between an input and parameter as follows: a parameter identifies a software setting – a scalar value (though it can be an enumerated or vector quantity) that informs how the system operates. An input identifies data that the system relies on, such as an image or scalar field. Each input and parameter is summarized in Table 3.1. It is shown that terrain feature placement and shape are easy to control using the provided inputs and parameters.

The input F is typically provided as an image buffer such as that in Figure 3.1. The input P can also be provided as image buffers (see Figure 3.2), but explicit sets of $(x, f(x))$ pairs can be provided as well.

The first input F represents every generator node used to synthesize the current terrain. Sets of contiguous nodes are called *strokes* and define the location of a single feature in the resulting terrain. Connected component labelling assigns each stroke a unique ID/label, the generator nodes associated with a given stroke define the corresponding feature's location, and the cost of these nodes define the initial height of the feature. The method in which the locations and costs of nodes in F is determined is application dependent. Their locations can be manually determined using a paint program or procedurally determined (see Chapter 4), and their costs can be calculated by any means; good choices for node cost include any of the procedural, physical, or image-based terrain synthesis methods discussed in Chapter 2. Figure 3.1 shows the locations of the generator nodes for the running example terrain in this chapter.

Table 3.1: Summary of inputs and parameters to the proposed terrain synthesis algorithm. Inputs consist of F, P, and $C_g(n)$. Parameters consist of μ_w, r, s, b, ϵ_s, p_f, and $min\{t_s\}$.

Input/Parameter	Description
F	Generator node locations
P	Terrain profiles
$C_g(n)$	Generator node costs
μ_w	Mean edge weight
r	Maximum edge weight deviation
s	Sea level scale
b	Blending bias
ϵ_s	Sea level cost threshold
p_f	Approximate terrain threshold
$min\{t_s\}$	Minimum scaling value threshold

Figure 3.1: This figure depicts a binary image where black pixels correspond to the location of generator nodes. This figure visualizes the input F used for the running example terrain in this chapter.

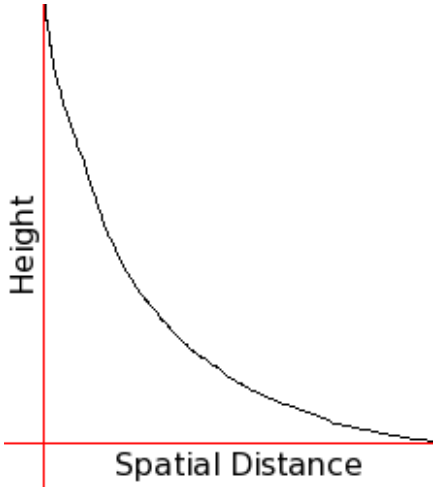


Figure 3.2: The hand-drawn profile used in the synthesis of the running example terrain. The labelled axes are added for interpretation and are not part of the profile.

The second input P refers to every *profile* used in the current render. A *profile* describes the cross-sectional shape of a feature and offers user control over terrain feature shape. Profiles are discussed in Subsections 3.2.1 and 3.3.1. Figure 3.2 shows the hand-drawn profile used in this chapter's running example terrain.

The third and final input $C_g(n)$ provides the cost for every generator node. These costs define the initial height of the terrain features, before the application of Dijkstra's algorithm. The method in which these costs are created is application dependent; they can be user-specified, they can be calculated using any terrain synthesis method that produces a height field, or the user can sketch input for heights (initial costs) along each stroke. For example, the majority of the results in Chapter 4 use fBm for $C_g(n)$.

The first parameter μ_w is the mean weight of the edges in E and controls overall terrain steepness. The method requires $\mu_w > 0.0$ and as μ_w increases, so does terrain steepness. Figure 3.3 shows a sample terrain with different values for μ_w. The second parameter r is the maximum edge weight deviation in E and controls overall terrain roughness. The method also requires that $0.0 \leq r < \mu_w$ to ensure that all edge weights are positive. As r approaches μ_w, terrain roughness increases. Figure 3.4 shows a sample terrain with different values for r.

The third parameter s is called the *sea level scale* and it is assumed that $s \geq 0.0$. Equation 3.1 shows that the product of the sea level scale and the maximum cost of a generator node c_{max} in F produce the *sea level cost* c_s.

$$c_s = s \cdot c_{max} \tag{3.1}$$

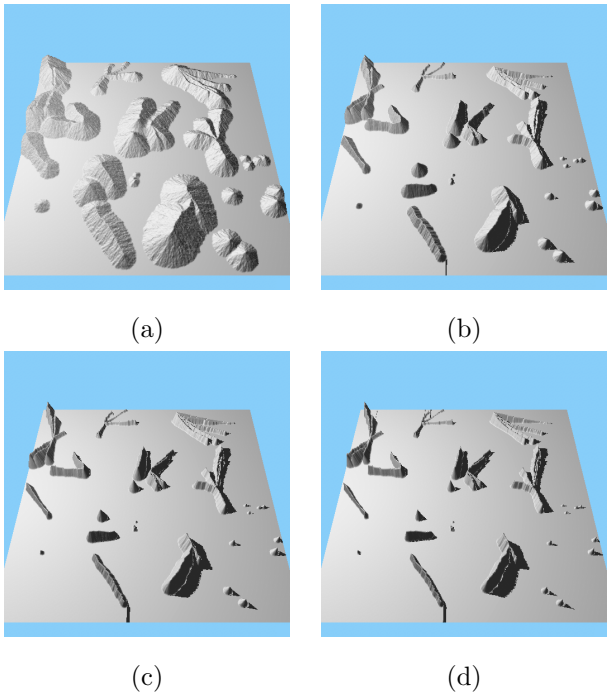


Figure 3.3: Effect of parameter μ_w on terrain steepness. From (a) to (d): $\mu_w = 1.0$, $\mu_w = 2.0$, $\mu_w = 3.0$, and $\mu_w = 4.0$. All other variables are constant ($s = 1.1$, $r = 0.5$, and $b = 3$). To emphasize the effect of μ_w, no profiles were used to shape the terrain; fBm was used to calculate the cost of generator nodes. Terrain steepness increases with μ_w.

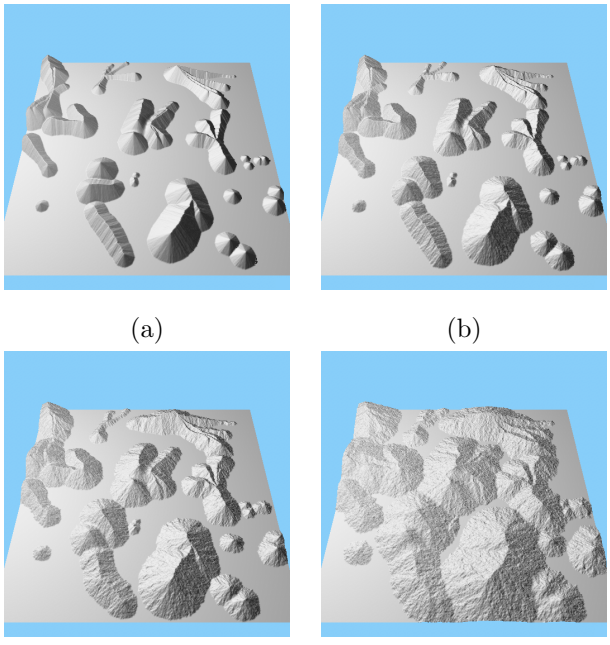


Figure 3.4: Effect of parameter r on terrain roughness. From (a) to (d): $r = 0.1$, $r = 0.37$, $r = 0.64$, and $r = 0.9$. All other variables are constant ($s = 1.1$, $\mu_w = 1.0$, and $b = 3$). To emphasize the effect of r, no profiles were used to shape the terrain; fBm was used to calculate the cost of generator nodes. Terrain roughness increases with r.

More specifically, the sea level scale s serves to adjust the computed sea level cost c_s – the maximum permitted cost in the graph. The sea level scale is an aesthetic parameter that determines the maximum permissible height in the synthesized terrain. It controls the maximum height because of the relationship between cost and height, which is now discussed. The final height h of a node is calculated as a function of its cost c, and c_s, as shown in Equation 3.2.

$$h = (c_s - c) \tag{3.2}$$

Figure 3.5 shows a sample terrain with different values for s, and Figure 3.6 visualizes the relationship between cost and height. Figure 3.6 illustrates how smaller costs are converted into larger heights using $s = 1.0$, $c_{max} = 55$, $c_s = 55$, along a profile through a single generator node with cost $c = 10$; the maximum cost of the profile is 50. After the application of Equation 3.2, the generator has a height of 45 units and the minimum height of the profile is 5 units. However, if $s = 1.2$, then $c_s = 66$. As a result, the height of the generator will be 56 units – increasing s by 0.2 increases the height of the generator and its profile by 11 units.

The assumption that $s \geq 0.0$ is made so that the sea level cost c_s is a positive value; negative costs are problematic for Dijkstra's algorithm. However, the behavior of $s \geq 1.0$ differs from that of $s < 1.0$. When $s \geq 1.0$, every generator has a cost less than or equal to c_s. Thus, all generator nodes will be visible in the final terrain because they will have a height greater than or equal to the sea level height. In contrast, when $s < 1.0$, all generator costs greater than or equal to c_s will be clipped at a cost of c_s; the cost of these nodes is clipped and the portions of the strokes they define are not included in the final synthetic terrain. Figure 3.5a shows a terrain synthesized with $s = 0.50$.

The fourth parameter b controls the bias when blending individual features together. Blending is discussed in Section 3.3. The fifth and sixth parameters are the sea level cost threshold ϵ_s and approximate terrain threshold p_f, respectively. These parameters control the termination of Dijkstra's algorithm and are also discussed in Section 3.3. Finally, the seventh parameter $min\{t_s\}$ specifies the minimum value returned by a scaling function embedding a profile's slope in the weights of edges in E (see Subsections 3.2.1 and 3.3.1).

3.2.1 Profile Creation

The shape of individual terrain features, such as mountains, hills, and craters, are specified by a monotonically decreasing profile. Terrain profiles P are an input to Algorithm 3.3 that offer user control over feature shape. Profiles can be hand-drawn or procedurally generated, but must be monotonically decreasing to avoid negative edge weights which can lead to ill-defined path costs.

The goal of a profile is to replace the slopes and heights computed from edge weights with user-sketched slopes and heights. More specifically, a profile causes the final shape of terrain feature f_i to

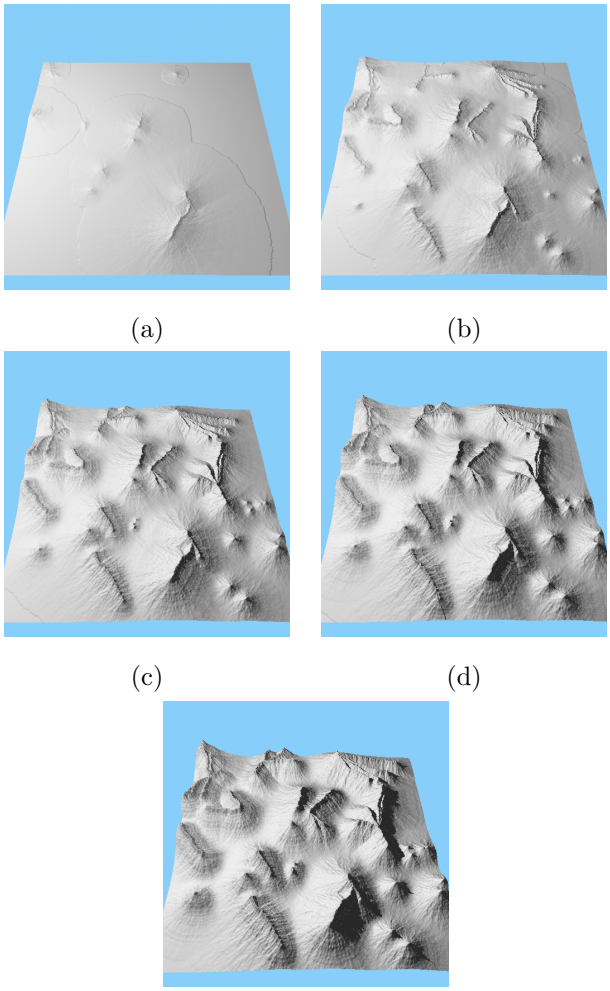
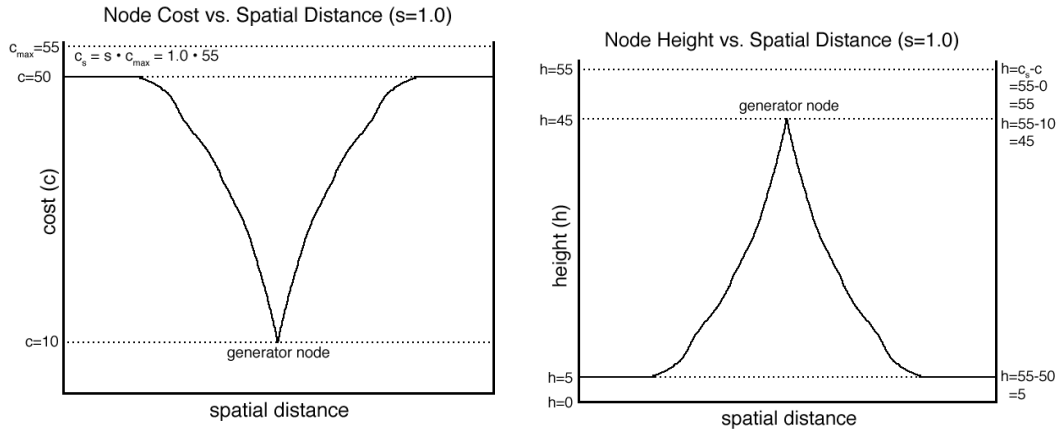


Figure 3.5: Effect of parameter s on maximum terrain height. From (a) to (e): $s = 0.5$, $s = 1.0$, $s = 1.33$, $s = 1.66$, and $s = 2.0$. All other variables are constant ($\mu_w = 2.0$, $r = 1.0$, and $b = 3$). One profile was used to shape the terrain and fBm was used to calculate the cost of generator nodes. Terrain height increases with s and image (a) shows that generator nodes whose cost exceeds or is equal to half of the maximum cost of a generator node will not be included in the final synthetic terrain.



(a) (b)

Figure 3.6: Visualization of the conversion from node cost to node height. In both images, $s = 1.0$ and $c_{max} = 55$; the sea level cost $c_s = 55$ (see Equation 3.1). Image (a) shows a profile through a single generator node with a cost of 10. The maximum cost of this profile is 50. Image (b) shows that the corresponding height of this node is 45 units and the minimum cost of this profile is 5 units; the conversion from cost to height is calculated using Equation 3.2.

be similar to the shape of profile p_j with which it is paired. This is accomplished by forcing the slope of a least-cost path originating at a generator in f_i to be similar to the slope of p_j. The conversion of an input profile to a scaling function that enforces this slope is detailed in Subsection 3.3.1.

3.3 Algorithm

The proposed terrain synthesis method is summarized in Algorithm 3. The algorithm is simple and incorporates random graph edge weights, permits the inclusion of multiple topography profiles, and allows precise control over placement of terrain features and their heights. These properties all allow the artist to create highly heterogeneous terrains. Furthermore, it addresses many of the known deficiencies in existing methods (Section 2.6).

The algorithm begins by initializing F (step 1 of Algorithm 3). The nodes of G are arranged in an 8-connected grid corresponding to the spatial points on a height field. Edge weights w_e are initialized as shown in Equation 3.3: the weights are uniform random values with a mean of μ_w and a maximum deviation of $\pm r$ (the random variable v_r is uniformly distributed over the range $[-1, 1]$) where $0.0 \leq r < \mu_w$.

$$w_e = \mu_w + r \cdot v_r \tag{3.3}$$

Algorithm 3 Terrain synthesis algorithm summary

Input: mean edge weight μ_w, maximum edge weight deviation r, generator locations F, sea level scale s, terrain profiles P, generator node costs $C_g(n)$, blending bias b, sea level cost threshold ϵ_s, approximate terrain threshold p_f, and minimum scaling value threshold $min\{t_s\}$

Output: Height field

1. Initialize graph G, consisting of nodes N and edges E, with mean edge weight μ_w and maximum edge weight deviation r
2. Mark nodes at locations F as generator nodes with costs $C_g(n)$
3. Calculate sea level cost c_s as the product of s and the maximum cost of a generator node
4. Create a scaling function, with a minimum scale of $min\{t_s\}$, for each profile in P
5. Apply Dijkstra's algorithm with frontier consisting of all generator nodes
6. Store resulting approximate cost field $C_a(n)$
7. For each individual feature f in F:

a) Apply Dijkstra's algorithm (using a modified search space based on $C_a(n)$, c_s, ϵ_s, and p_f) with frontier consisting of only f's generator nodes

b) Blend f's height field, as a function of b, into the final height field

With graph node n we store a cost c_n, a scaling term $w_s(c)$ (a function of cost), a feature identification number f_{id}, a profile identification number p_{id}, and the generator node n_g on the least-cost path that includes n. Feature IDs must be unique to each stroke, but many strokes may use the same profile ID. The costs of generator nodes, which govern the initial height of the feature, are user-specified. Smaller costs indicate higher elevation. The cost, c_n, of a non-generator node, n, is the cost of the shortest path to n from a generator node n_g, as determined by Dijkstra's algorithm. Equation 3.4 defines the the cost of the i-th node on the path $n_1, n_2, \ldots, n_k$, where $n_1 = n_g$, $c_{n_1} = c_{n_g}$, w_e is the weight of the edge from n_{i-1} to n_i, $w_s(c_{n_{i-1}})$ is the node scaling value for n_i, and k is the number of nodes on the path.

$$c_{n_i} = c_{n_{i-1}} + w_e \cdot w_s(c_{n_{i-1}}) \tag{3.4}$$

Node scaling values are derived from the profile associated with n_g (see Section 3.3.1) and are used to enforce the shape of the profile on the landscape. When profiles are not used, $w_s(c) = 1$ for all c.

Step 2 of Algorithm 3 proceeds by setting the cost (c_n) according to $C_g(n)$. The feature ID (f_{id}) and profile ID (p_{id}) of generator nodes are also set at this time. Each contiguous set of generator nodes, that collectively define the height and location of a single terrain feature, defines a *stroke*. Each stroke's generator nodes share the same f_{id} and p_{id} values, and the costs and locations of these nodes collectively define the initial height and location of the feature. Figure 3.1 shows the locations of such nodes for the running example terrain in this chapter.

In step 3 of Algorithm 3 the sea level cost c_s is the product of the input scaling value s ($s \geq 0.0$) and the maximum cost c_{max} of any generator node. This calculation is given in Equation 3.1. Setting c_s below c_{max} will prevent some features from appearing in the synthesized terrain. As already discussed in Section 3.2, c_s provides the maximum permissible cost in the graph. Furthermore, it aids in converting cost to height (Equation 3.2), and in limiting the search space of Dijkstra's algorithm. The termination of Dijkstra's algorithm is discussed next.

Step 4 of Algorithm 3 creates from each user-provided profile the corresponding node scaling function $w_s(c)$ (see Subsection 3.3.1). Steps 5–6 create an approximate cost field $C_a(n)$ which is used to guide the termination of Dijkstra's algorithm. The approximate cost field is created by running Dijkstra's algorithm using all generator nodes, and their associated profiles, as the initial frontier. As previously mentioned in Section 2.4, the frontier consists of all nodes with a tentative cost. The approximate cost field is visualized in Figure 3.7. The profiles are included in this step to define each feature's shape. Step 7a then individually generates the cost fields for each feature – a cost field is created for each feature using only the associated stroke's generator nodes. Therein, Dijkstra's algorithm is halted when the cost of a node is within ϵ_s of c_s, or is a sufficiently small fraction p_f of $C_a(n)$. Figure 3.8 visualizes the search space of four select strokes from the running example terrain in this chapter. Black pixels collectively define a stroke and feature location, and

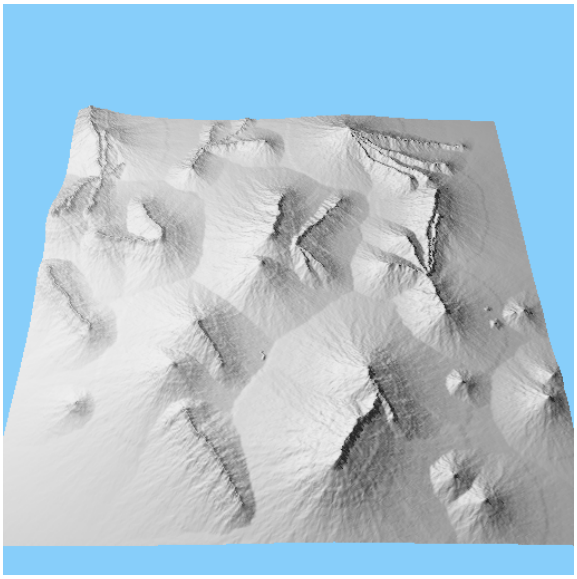


Figure 3.7: An approximate terrain synthesized without blending. Discontinuities appear where least-cost paths from differing generators are nearly identical in cost. Input parameter values: $\mu_w = 2.0$, $r = 1.0$, $s = 1.1$, and $b = 3$. Generator node locations F is given in Figure 3.1 and P is shown in Figure 3.2.

blue pixels indicate halting due to cost being within ϵ_s of c_s. Red pixels indicate halting due to cost being smaller than p_f percentage of the approximate cost field and green pixels indicate that both conditions have been met simultaneously. Halting Dijkstra's algorithm in this fashion prevents traversal of nodes that contribute negligible cost, saving computation. Step 7b blends together the height fields from individual features. Blending eliminates the discontinuities that appear where least-cost paths from differing generators are nearly identical in cost (see Section 3.3.2). The result is a height field such as that in Figure 3.9.

3.3.1 Scaling by Profile

Terrain profiles offer user control over feature shape by specifying local slope as a function of path cost. Profiles can be hand-drawn or procedurally generated, but must be monotonically decreasing to avoid ill-defined path costs arising from negative edge weights. Figure 3.2 shows the profile used in the running example terrain; Figure 3.11 uses a different profile.

However, profiles are not required. Profiles aid in the creation of heterogeneous terrains because several different profiles can be used in a single render. Profile removal forces raw edge weights to be used for terrain shape and the heterogeneity of the resulting terrain is limited, in part, to the heterogeneity of the edge weight calculation process, such as that in Figure 3.10. Applying the same random edge weight calculation to every edge in the graph can result in homogeneous terrains, but other functions for edge weight calculation are considered in Chapters 4 and 6.

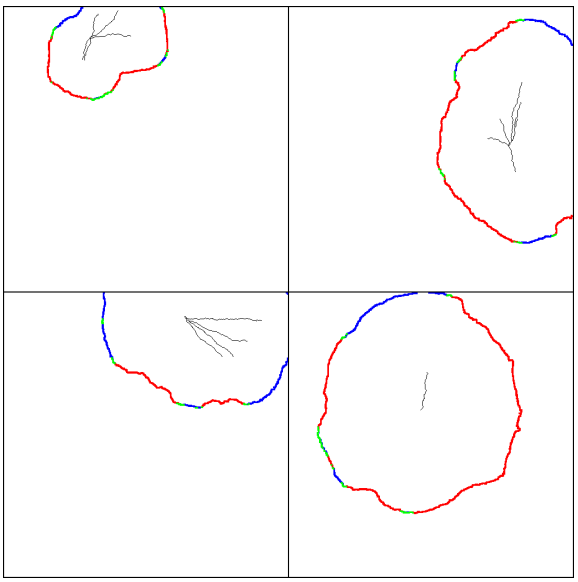


Figure 3.8: This figure shows the search space for Dijkstra's algorithm in four select strokes from the running example terrain in this chapter. Black pixels collectively define a stroke and feature location, and blue pixels indicate halting due to cost being within ϵ_s of c_s. Red pixels indicate halting due to cost being smaller than some small percentage p_f of the approximate cost field and green pixels indicate that both conditions have been met simultaneously. Each coloured pixel has been scaled to 3×3 for visualization purposes.

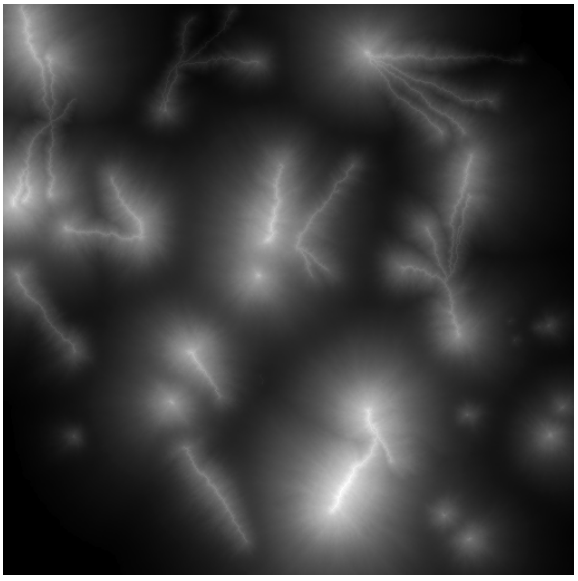


Figure 3.9: The final height field resulting from Algorithm 3 for the running example terrain in this chapter. Input parameter values: $\mu_w = 2.0$, $r = 1.0$, $s = 1.1$, and $b = 3$. Generator node locations F is given in Figure 3.1 and P is shown in Figure 3.2.

To construct a scaling function, the profile is first flipped vertically about the x-axis and is then scaled vertically to the range $[0, c_s]$. The profile is flipped vertically because of the relationship between cost and height provided in Equation 3.2. Scaling vertically to the range $[0, c_s]$ ensures that the scaling function will adequately cover all possible costs. Next, the slope $s_m(c)$ of the profile is calculated as a function of cost c. The scaling function $w_s(c)$, also a function of cost c, enforces the profile's slope by multiplying a given node's outgoing edge weights by the value of the scaling function at its cost (Equation 3.4). The calculation for $w_s(c)$ is given in Equation 3.5.

$$w_s(c) = s_m(c)/\mu_w \tag{3.5}$$

If there are multiple occurrences of a given cost c in the provided profile, the average $w_s(c)$ for c is stored. The average is taken to produce a single scaling value at c that accounts for the difference in profile slopes at c. The scaling function replaces the slopes and heights computed from edge weights with user-sketched slopes and heights.

It is important to note that the scaling function is defined as the slope $s_m(c)$ divided by the mean edge weight μ_w. Division by μ_w ensures that some edge weight randomness is preserved after the given edge weight is multiplied by $w_s(c)$. Recall from Table 3.1 that μ_w is the mean edge weight and that edge weights can deviate a maximum of $\pm r$ from this value; actual edge weights are calculated according to Equation 3.3. If the slope was divided by an edge's weight w_e, then this weight would cancel out when multiplying $w_s(c)$ and w_e in Equation 3.4. The division by μ_w preserves some of the edge weight deviation from μ_w. This means that the roughness of the resulting synthetic terrains is not lost during this process.

If a given node n's scaling value $w_s(c)$ is less than a minimum scaling value threshold $min\{t_s\}$, $w_s(c)$ is updated by linear interpolation between the value of the scaling function at the next lowest and greatest scaled profile costs (compared to n's cost c_n). To avoid re-assigning a cost less than $min\{t_s\}$, the value of the scaling function at the next lowest and greatest costs must be greater than $min\{t_s\}$. This check avoids the addition of small edge weights to least-cost paths that create artificially flat areas in the resulting cost field. Boundary cases ($c_n < 0.0$ or $c_n > c_s$) are not considered because negative node costs are not permitted in G and Dijkstra's algorithm is halted when the cost of a node is sufficiently close to c_s (see Section 3.3).

As Dijkstra's algorithm searches the graph, $w_s(c)$ is calculated for every visited node. If a node n's cost c_n is not present in the scaled profile, $w_s(c)$ is calculated by linear interpolation between the value of the scaling function at the next lowest and greatest profile costs (compared to c_n). Furthermore, the costs that $w_s(c)$ is evaluated over are modified so that every feature will fully depict its profile regardless of its overall height. If the raw cost of a node is used, the upper part of the profile might never be incorporated. Therefore, the raw node cost is used to determine a new cost c_m, allowing for the inclusion of the full profile.

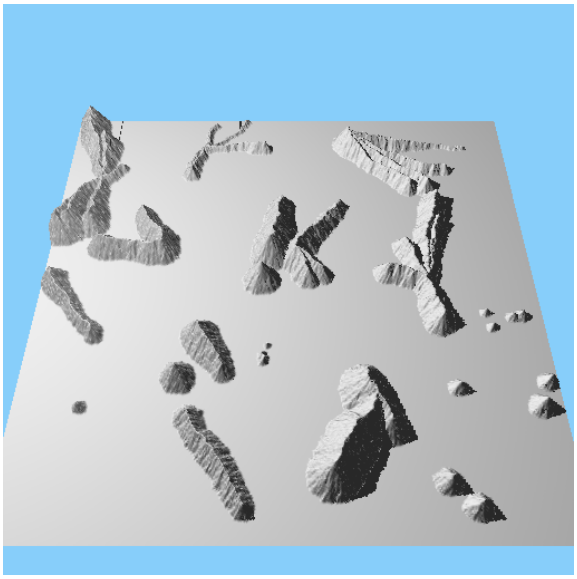


Figure 3.10: A synthesized terrain that does not use any input profiles. Input parameter values: $\mu_w = 2.0$, $r = 1.0$, $s = 1.1$, and $b = 3$. Generator node locations F is given in Figure 3.1.

The cost c_m used as the argument to the scaling function is calculated by linearly interpolating between zero and sea level. Thus, c_m is a function of the current node n's cost c_n, its source node n_g's cost c_{n_g}, and sea level cost c_s (Equation 3.6).

$$c_m = \frac{c_n - c_{n_g}}{c_s - c_{n_g}} \cdot c_s \tag{3.6}$$

Figure 3.11(a–b) shows a hypothetical scenario that uses raw node costs. It is evident from this figure that features with larger costs exhibit the slope near the bottom of the profile. Figure 3.11(c–d) visualizes the usage of c_m.

The profile framework allows for the inclusion of any custom profile, providing user control over the shape of synthesized terrains. Furthermore, profiles can be hand-drawn in any graphics painting software, making profile creation an easy process.

3.3.2 Blending

Algorithm 3 creates an individual height field for each feature, M features total. The individual height fields are then blended into a single height field with a modified version of the blending function proposed by Singh and Fiume [77]. Blending avoids the synthesis of terrains such as that in Figure 3.7. Such terrains exhibit noticeable seams/discontinuities that appear where least-cost paths from differing generators are nearly identical in cost. Sometimes these seams are desirable, such as in a v-shaped valley, but having control over the presence of the seam is more desirable.

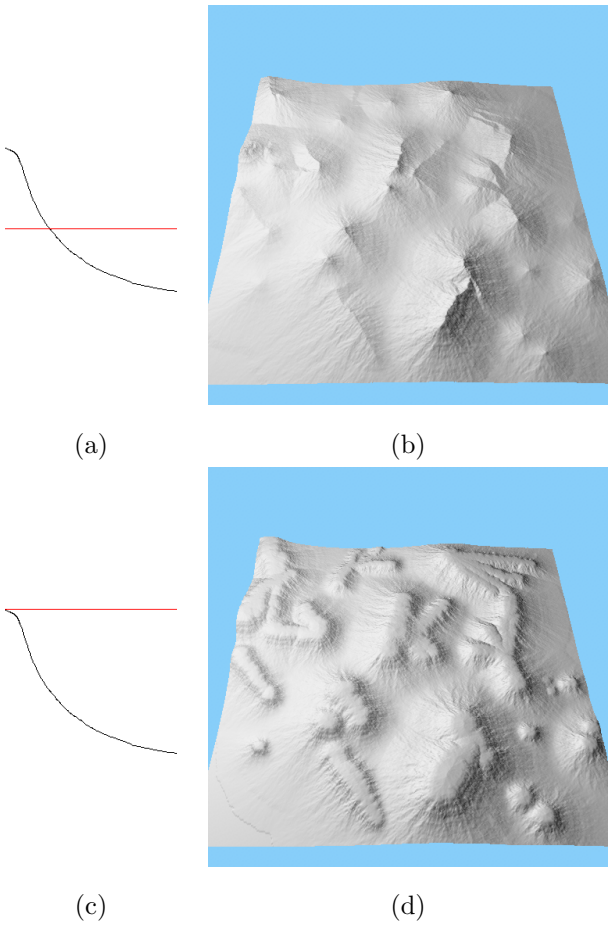
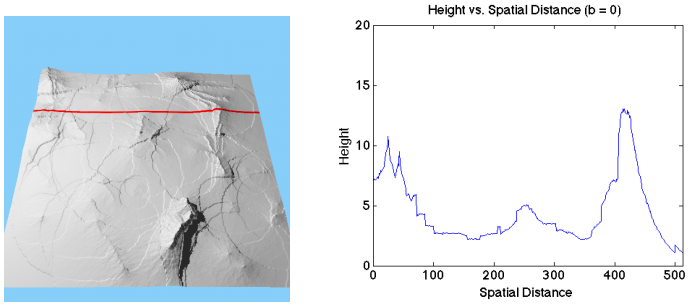


Figure 3.11: Images (a–b) show the result of using raw node costs for the calculation of $w_s(c)$. Image (a) visualizes that raw node cost usage may never index the entire profile. A node whose cost is that of the red, horizontal line will incorporate only profile slopes that exist below this line; profile slopes above this line will not be incorporated. Image (b) shows a synthetic terrain resulting from raw node cost usage. Images (c–d) show the result of using modified node costs c_m for the calculation of $w_s(c)$. Image (c) visualizes how Equation 3.6 ensures that the full profile will be indexed using c_m, regardless of starting cost. The first node on a least-cost path will always have a value of c_m equal to the cost of the red, horizontal line. This ensures that every feature indexes into the full profile. Image (d) shows a synthetic terrain resulting from c_m usage. Input parameter values: $\mu_w = 2.0$, $r = 1.0$, $s = 1.1$, and $b = 3$. Generator node locations F is given in Figure 3.1 and P is shown in image (a,c).

The final height h_n of a node n is a function of its costs $c_{n,i}$ and the sea level cost c_s. The quantity $c_{n,i}$ is the cost of node n according to feature i, $1 \leq i \leq M$. We may terminate Dijkstra's algorithm with unvisited nodes; such nodes do not contribute a height to the final blend. Equation 3.7 gives the blending metric whose behavior varies with the bias b from an average of the heights at each node when $b = 0$, approaching $max\{(c_s - c_i)\}$ (the maximum height) as b increases. Recall that Equation 3.2 shows how costs are converted to heights.

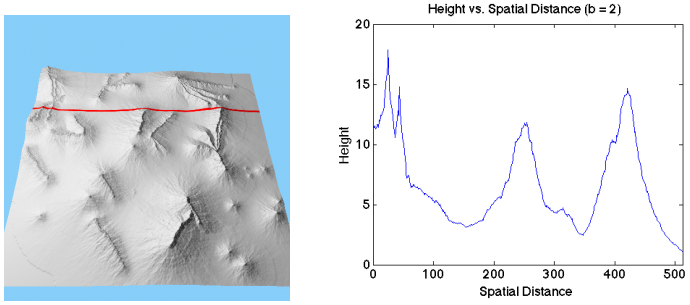
$$h_n = \frac{\sum_{i=1}^{M}(c_s - c_i)^{b+1}}{\sum_{i=1}^{M}(c_s - c_i)^{b}} \tag{3.7}$$

As previously mentioned, we may terminate Dijkstra's algorithm with unvisited nodes, and such nodes do not contribute a height to the final blend. The final height h_n of a node n is calculated as a function of the features that visited n; this calculation is given in Equation 3.7. Discontinuities can arise at the boundary dividing the nodes visited by a feature from those not visited. Inside the boundary, the feature's cost is included in the weighted average, and outside, the feature is excluded. Since a feature has a small height near the boundary, excluding it from the calculation can discontinuously increase the average. For larger b, larger heights have more weight, reducing the visibility of the discontinuity; however, for small b, the discontinuities are quite evident, and can be clearly seen in Figure 3.12(a–b). The effect of bias b for $b = 0$, $b = 2$, $b = 4$, and $b = 8$ is visualized in Figure 3.12. However, when an appropriate value for b is used ($b = 3$ or $b = 4$), the result of applying Algorithm 3 is a height field such as that in Figure 3.9.



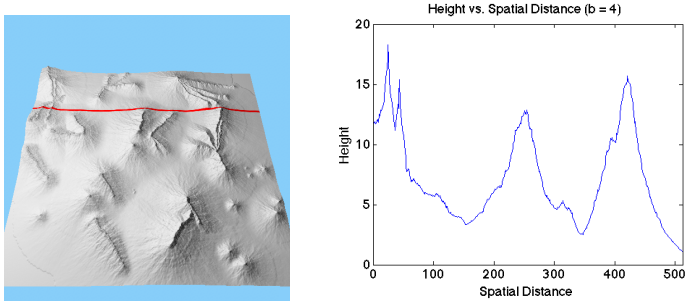
(a)

(b)



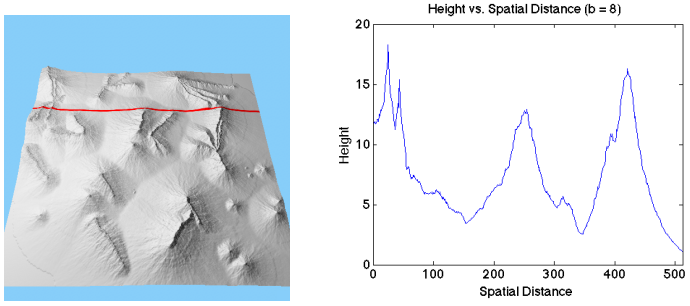
(c)

(d)



(e)

(f)



(g)

(h)

Figure 3.12: A terrain synthesized using different values for b. Images are grouped in the following pairs: (a–b) $b = 0$, (c–d) $b = 2$, (e–f) $b = 4$, and (g–h) $b = 8$; each plot visualizes the cross-section given by the red horizontal line in the corresponding terrain. The red line has a width of 5 pixels to aid in visualization, and the cross-section passes through the middle. All other variables are constant ($\mu_w = 2.0$, $r = 1.0$, and $s = 1.1$). As b increases from $b = 0$ to $b = 8$, the final height at each location goes from an average to the maximum height at that location.

Chapter 4

Terrain Synthesis Results

4.1 Overview

This chapter presents the results of terrain synthesis using Algorithm 3, as well as methodology evaluation and an assessment of our algorithm's performance. The results can be categorized as follows: realistic, image driven, and sketch-based. Realistic terrain synthesis produces terrains found in nature such as craters, image driven terrain synthesis consists of terrains whose features are placed and/or shaped using an input texture, and sketch-based terrains are where coloured pixels in an input image indicate strokes, providing feature location and feature profile. Some of the above sections are further categorized with respect to the synthesis of each result. For example, in the realistic terrain category, the canyon, layered, tower karst, hill, and mountainous landscapes were all synthesized using a combination of manual and procedural input strokes. In contrast, the crater, cinder cone volcano, and lunar landscapes selectively apply profiles and selectively add terrain heights in order to create the crater effect.

The evaluation of our methodology involves a comparison to the RMF terrain model and the work of Zhou et al. [92] – the current state-of-the-art in procedural and controllable terrain synthesis methods, respectively. The assessment of our algorithm shows that per-stroke cost is consistent across height field resolutions when feature density is fixed; per-stroke cost refers to the area that Dijkstra's algorithm searches when synthesizing each feature individually.

Feature locations (F from Section 3.2) are both procedurally and manually created as warranted by the situation. Generator nodes for some ridges are formed by running Dijkstra's algorithm from select generator nodes and then selecting a random node (uniformly distributed over the graph) and tracing back on the least-cost path to the originating node. The process is used to extend the location of existing features and does not add new features; the nodes on the least-cost path contain the same feature and profile ID as the originating node – they are part of the same stroke. This creates ridges with a dendritic shape. For more elaborate results that combine multiple feature styles, pixel colour in an input image defining F indicates which profile provides its style. This scenario is discussed in Section 4.4. Also, reference photos are provided for each result in Section 4.2 as real examples of the type and character of the terrain element(s) being synthesized.

Algorithm 3 was coded in C++ (compiled using G++ 4.2.3) and executed in Mandriva 2008.1 running on a Pentium 4 2.80GHz processor with 1GB RAM. Unless otherwise indicated, a height-field resolution of 512×512 was used, and values of $p_f = 0.05$ (height-field percentage threshold) and $\epsilon_s = 0.5$ (sea level proximity threshold) were used to terminate Dijkstra's algorithm. Additionally, the minimum scaling value threshold $min\{t_s\} = 0.001$ avoids the addition of small edge weights to path costs that create artificially flat areas in the resulting cost field.

The default method for calculating generator node costs is fBm using $H = 3.0$, Lacunarity=2.0, and Octaves=2. As previously mentioned in Section 2.1, the parameter H is known as the Holder exponent [55]. The value of H determines the fractal dimension of the roughest areas (the smaller H is, the rougher the terrain becomes); using $H = 3$ produces a smooth terrain. Lacunarity is the gap between successive frequencies, and octaves is the number of frequencies in the fBm terrain. The calculation for fBm takes place over a 4×4 grid scaled to the height-field resolution – the step size along each axis is $0.0078 = 4.0/512$ for a 512×512 height field. This step reduces the number of integer points where Perlin noise is zero (see Algorithm 1); the size of the grid controls the frequency of the fBm surface. Furthermore, the resulting fBm values are scaled by 30.0 so that the resulting heights are appropriate for the chosen height field resolution. Deviations from these parameters are noted where applicable. Finally, all renders were created by importing the resulting height fields into Planetside's Terragen [79] terrain rendering system with procedural textures (determined by height and slope) overlaid on top of the terrain geometry.

For each result, its inputs and parameters (Section 3.2), cost assignments to generator nodes, and running time are presented. Complete details, including inputs and parameters, for each result are provided in Appendix A. With the exception of sketch-based terrains, all parameters are hard-coded (completely defined within the code). The selected results show the ease of use of the method, and the diversity and realism of the features. Furthermore, the results can accommodate differing structures within the same scene, resulting in heterogeneous terrains. As each result in Section 4.2 is presented, a comparison between it and its reference photo is made. When comparing the synthesized terrains to the provided reference photos, the reference image is considered to be an exemplar of the feature. We are not attempting to synthesize a clone of the reference image in terms of feature location and height, but we are attempting to synthesize features that are also sensible exemplars. We are interested in what ways it reflects the exemplar, and in what ways it does not. Furthermore, desirable characteristics of edge weight textures used in Section 4.3 are discussed as well.

4.2 Realistic Terrains

Realistic terrain synthesis produces terrains found in nature such as craters, volcanoes, faults, hills, mountain ranges, and tower karst and lunar landscapes. A reference photo, inputs, parameters, and outputs are provided for the results in Subsection 4.2.1, while subsequent results show only the reference photo, generator node locations F, and a final render. All reference photos, with the exception of images [58, 59, 60] and [86], are used under the Creative Commons (CC) license. Images [58, 59, 60] and [86] are non-copyrighted works provided courtesy of NASA and USGS, respectively.

4.2.1 Valleys

V-shaped and u-shaped valleys are commonly seen in mountain ranges and larger groups of foothills, so it is desirable to realistically synthesize these features. Reference photos are provided in Figure 4.1 [72, 73]. Figure 4.1 shows the characteristics of such valleys: v-shaped valleys contain a noticeable discontinuity between the two opposing ridges, whereas u-shaped valleys contain a smooth, round transition at this point. Two opposing ridges were manually placed, and their strokes are visualized in Figure 4.2. To distinguish between the valley types, two different profiles were used, as shown in Figure 4.3(a–b); all other parameters (with the exception of bias b) were identical.

Two profiles were used per result; the profile in Figure 4.3c is applied when Dijkstra's algorithm searches the non-valley regions. The profile in Figure 4.3a creates the discontinuity seen in a v-shaped valley by terminating with a slope larger than zero, whereas the the profile in Figure 4.3b terminates with a near zero slope, facilitating the smooth, round transition seen in a u-shaped valley. The flatter, smoother topography of the non-valley regions is created using the profile in Figure 4.3c. Feature identification is visualized in Figure 4.4a and this information is used in Step 7 of Algorithm 3.

Generator node costs were assigned using fBm, which are visualized in Figure 4.4b. The final renders and height fields in Figure 4.5 were achieved using the previously mentioned inputs and parameters, along with $\mu_w = 2.0$, $r = 1.5$, and $s = 2.0$; the OpenGL renders are included to emphasize the valley topographies. A larger scaling value s helped define the opposing ridges, a bias value of $b = 15$ was used for the v-shaped valley, and $b = 3$ was used for the u-shaped valley. Increasing b reduced the effect of blending, which is ideal for the discontinuity of v-shaped valleys. The synthesis took 7 seconds for the v-shaped valley and 8 seconds for the u-shaped valley.

The striking characteristic of v-shaped and u-shaped valleys is the profile of the valley that separates the two or more opposing features. Figure 4.1a shows a noticeable discontinuity where the opposing features meet, whereas Figure 4.1b does not contain this discontinuity. Also, erosional

(a) (b)

Figure 4.1: a) Reference photos for a v-shaped [73] and (b) u-shaped [72] valley.

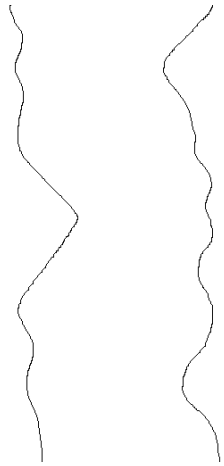
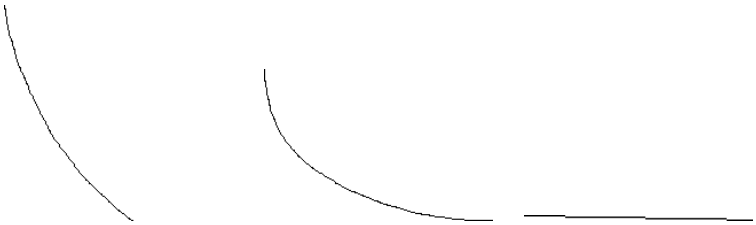
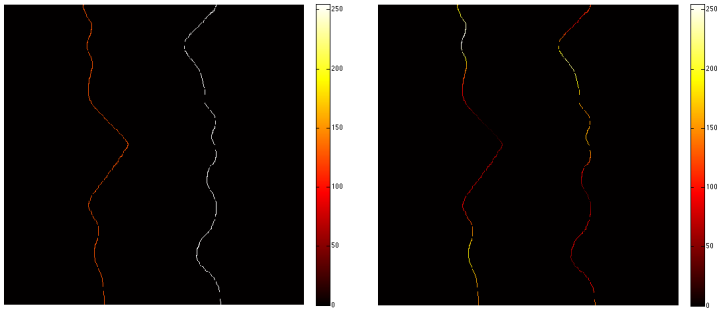


Figure 4.2: Visualization of generator node locations F for the v-shaped valley result.



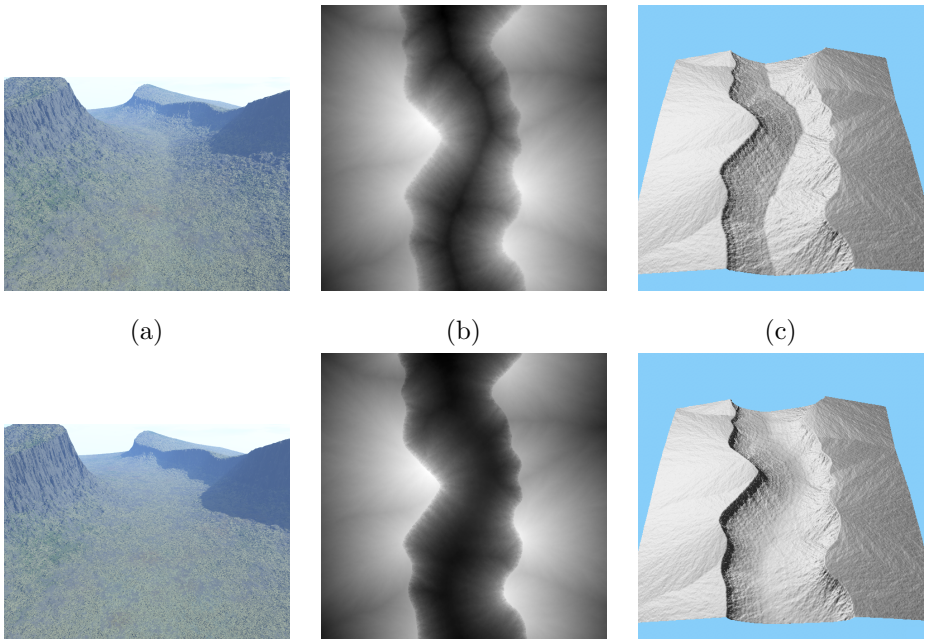
(a) (b) (c)

Figure 4.3: (a) Profiles for a v-shaped and (b) u-shaped valley. Image (c) provides the profile for the non-valley regions and is used in both renders.



(a) (b)

Figure 4.4: (a) Visualization of feature label and (b) cost for the v-shaped and u-shaped valley results. False colouring has been used to increase label and cost visibility. The colourbar associated with each image maps pixel colour to a pixel intensity in the interval [0,255]; lower pixel intensity corresponds to smaller cost.



(a) (b) (c)

(d) (e) (f)

Figure 4.5: Final renders, height fields, and OpenGL renders for the v-shaped and u-shaped valleys. Images (a–c) are the final render, height field, and OpenGL render for the v-shaped valley, respectively. Images (d–f) are the final render, height field, and OpenGL render for the u-shaped valley, respectively.

forces have caused the terrain to be relatively smooth near this meeting point. The synthesized results in Figure 4.5 accurately showcase such features. The provided profiles create the v-shape or u-shape valley and the chosen blending function helps shape the terrain near the middle of the valley. The proposed method synthesizes valleys that reflect the exemplars in Figure 4.1.

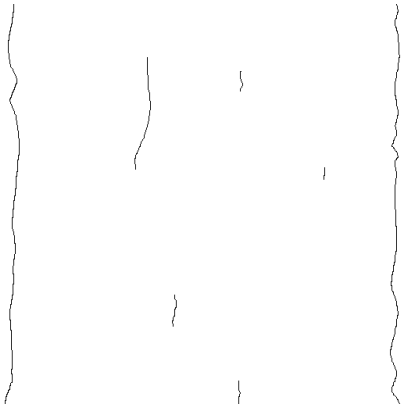
4.2.2 Canyon, Layered, Tower Karst, Hill, and Mountainous Landscapes

Canyons are common in the southern United States of America (USA); the Canyon de Chelly National Monument, located in Arizona, USA, is an example. A reference photo of this canyon is provided in Figure 4.6a [69] which shows the characteristics of canyons; erosional processes, typically river streams, carve through a relatively flat terrain creating steep-sided gorges. Canyons often contain pillar-like structures of denser rock that are less prone to erosion.

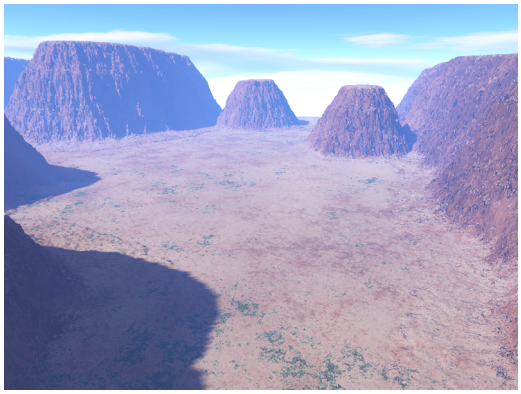
To synthesize Figure 4.6c, seven flat areas were manually placed, as shown in Figure 4.6b. A single profile was used to create a consistent erosional process, as shown in Figure A.3b. The profile's steep slope creates the steep-sided gorges. Feature identification is visualized in Figure A.3c, and generator node costs were assigned using fBm and are visualized in Figure A.3d. The final render in Figure 4.6c and height field in Figure A.3f result from these inputs, parameters, and processes, along with $\mu_w = 2.0$, $r = 1.5$, $s = 2.0$, and $b = 4$. A larger scaling value s was needed to create the deep gorge, and the synthesis completed in 7 seconds.

The provided reference canyon in Figure 4.6a shows that canyons contain steep gorges with a relatively flat basin, isolated tall features can reside within the interior of the canyon, and rock strata is present in the steep sides of the gorge. The synthesized result in Figure 4.6c demonstrates that the proposed method can create the steep gorge walls, rock strata, and isolated features. Unfortunately, not all canyon features can be captured using the proposed method. Features that overhang lower features are not possible in a height-field representation because only one height value can exist at each index in the field. Even though Figure 4.6c is able to produce the steep gorge walls, many more profiles would be needed to match the feature diversity seen in Figure 4.6a. Additionally, a given profile is incorporated in all directions. Thus, individual features, such as the isolated taller features within the canyon interior, tend to be isotropic; this is an undesirable property for the gorges. The gorges should not be isotropic – one side should be relatively flat, the other should be steep. There are two ways to circumvent this situation: 1) identify the canyon exterior and apply a different profile in that region, or 2) place exterior generator nodes, adjacent to the generators defining the gorge, and assign a different profile ID to these nodes. Unfortunately, both solutions require more work by the user. Though synthetic canyons require additional work to match the feature diversity of the reference image and to avoid isotropy, the proposed method synthesizes canyons that reflect the exemplar.

(a)



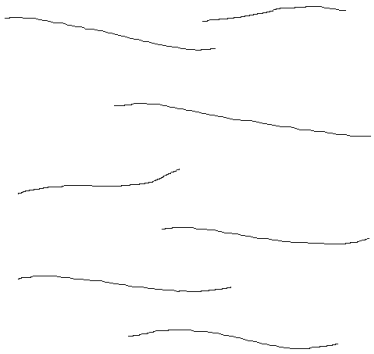
(b)



(c)

Figure 4.6: (a) Reference photo [69], (b) generator node locations F, and (c) final render for the canyon result.

(a)



(b) (c)

Figure 4.7: (a) Reference photo [32], (b) generator node locations F, and (c) final render for the layers result.

Visually, distant objects become increasingly abstract; detail is lost. Such arrangements of features, specifically layers of features, that elicit this visual effect are easy to create with the proposed method. Seven features were manually placed, and the corresponding strokes are shown in Figure 4.7b. A single profile (Figure A.4b) was used to create a uniform terrain style, such as that in Figure 4.7a. The profile begins with a steep slope and quickly descends, resulting in steep features. Feature identification is visualized in Figure A.4c, and generator node costs were assigned using fBm ($H = 1.0$ and Octaves=8) evaluated over a 6×6 grid. These costs are visualized in Figure A.4d. The final render in Figure 4.7c and height field in Figure A.4f result from these inputs, parameters, and processes, along with $\mu_w = 2.25$, $r = 1.25$, $s = 1.75$, and $b = 4$. A larger scaling value s was needed to create the taller ridges, and the synthesis completed in 13 seconds.

Series of roughly parallel mountain ranges result in scenic landscapes such as that in Figure 4.7a. This arrangement of features results in a layered look, and Figure 4.7c shows that such scenes are easy to synthesize using the proposed method. The render in Figure 4.7c results from drawing a series of curved line segments that are roughly parallel, as in Figure 4.7b, and using the profile

depicted in Figure A.4b. The rough topography of the ridges is accomplished by lowering the value of H in fBm to 1.0. The parameter H controls terrain roughness; smaller values of H create a rougher surface. The proposed method synthesizes layered mountain ranges that reflect the exemplar in Figure 4.7a.

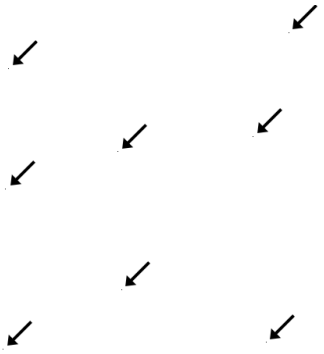
Karst landscapes are formed by the underground erosion of rocks, such as limestone and marble, that dissolve in water. They display distinctive surface attributes, with sinkholes being the most common; sinkholes are small to medium sized closed depressions that form where the roof of a subterranean cave collapses [53]. Such terrains are beautiful, but are difficult to synthesize using existing procedural methods because of their unique shape and style.

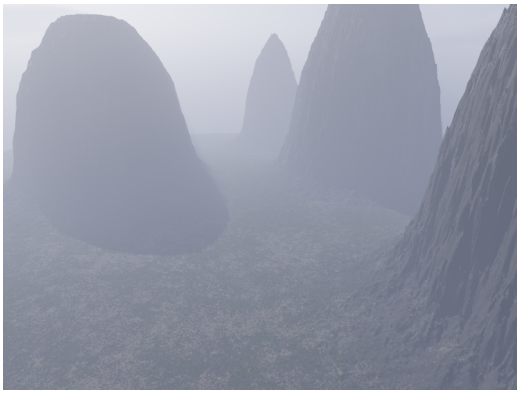
Fortunately, the proposed method requires minimal input to create these effects. The reference photo in Figure 4.8a [83] shows a tower karst. Tower karst are characterized by steep-sided formations between sinkholes and are formed in regions where vertical joints (natural cracks) control the collapse of subterranean caves [53]. Eight features were manually placed, and the corresponding strokes are shown in Figure 4.8b. Figures A.5(b–d) show the three profiles that were used to create a similar, yet varying landscape; profile ID was calculated as feature ID number modulo three. The sharing of a steep, rounded profile among the terrain features provides this similarity. Feature identification is visualized in Figure A.5e, and generator node costs were assigned using fBm (see Figure A.5g). The final render in Figure 4.8c and height field in Figure A.5i result from these inputs, parameters, and processes. Additional parameter values were $\mu_w = 0.5$, $r = 0.35$, $s = 4.0$, and $b = 3$. A larger scaling value s was needed to create the deep valleys seen in Figure 4.8c and the synthesis completed in 7 seconds.

Tower karst are characterized by large, round, and isolated foothills, as seen in Figure 4.8a. Of particular importance are the height and style of these terrains, and their level of isolation. Many of these foothills can reside near one another, but deep valleys typically reside between their respective peaks. Furthermore, they have a rougher topography than that of the hills in Figure 4.9a. Such landscapes can be accurately synthesized using the proposed method. Input profiles control feature style, and the careful selection of the sea level cost facilitates the synthesis of deep valleys between the foothills. Due to the steepness of these features, overhanging features are not possible to synthesize for the same reason as the canyon result. Furthermore, care must be taken to avoid the isotropic features seen in Figure 4.8c. Within a given feature, its profile is the same in all directions and this results in an isotropic feature. However, this can be alleviated by assigning adjacent generator nodes different profile IDs. With a few minor exceptions, the proposed method synthesizes tower karst landscapes that reflect the exemplar in Figure 4.8a.

Rolling hills are a common topography on earth that results from erosional forces such as rain and wind. Therefore, existing physically-based methods can accurately synthesize such topographies. Procedural methods such as fractional Brownian motion can create this effect as well. One

(a)





(b) (c)

Figure 4.8: (a) Reference photo [83], (b) generator node locations F, and (c) final render for the tower karst landscape result.

way to create hills via fBm is by using a small number of octaves; the higher frequencies associated with the upper octaves are not required to synthesize hill shapes. Some hills are shown in Figure 4.9a [26]. Six features were manually placed, and their corresponding strokes are shown in Figure 4.9b. Figures A.6(b–d) show three similar, yet different profiles. Their differences decrease the homogeneity of the resulting synthetic terrain. Also, the profiles are rounded and smooth, and are assigned to a stroke's generators via feature ID number modulo three, in order to add diversity to the hills. This assignment is visualized in Figure A.6f. Feature identification is visualized in Figure A.6e.

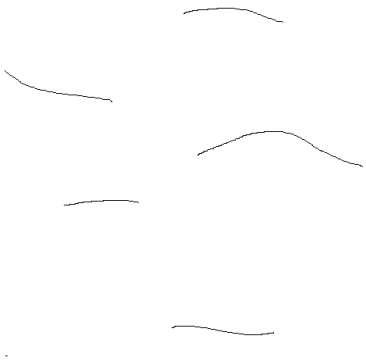
Generator node costs were calculated using fBm evaluated over a 3×3 grid. Then, the generator node costs at the locations visualized in Figure 4.9b are calculated by scaling fBm within a cost range of $[0.0, 20.0]$. The final render in Figure 4.9c and height field in Figure A.6i result from the previously mentioned inputs, parameters, and processes, along with $\mu_w = 0.75$, $r = 0.375$, $s = 2.5$, and $b = 3$. A larger scaling value s helped emphasize the local minima between hills, and the synthesis completed in 7 seconds.

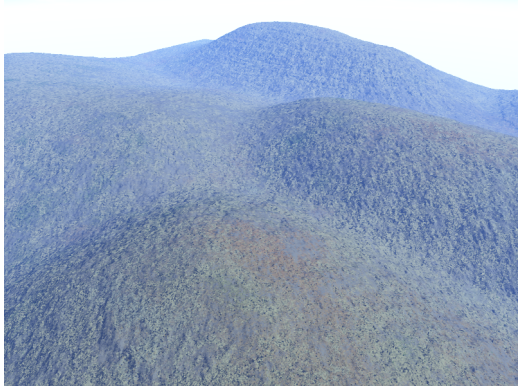
Hills are characterized by their smooth rounded shape – there are minimal discontinuities in the terrain. Figure 4.9c shows such a terrain. The hills are smooth and rounded, and the blending function ensures a smooth transition between features. The terrain style is controlled by providing profiles that are smooth and rounded, as well as edge weights that have a smaller deviation from the mean edge weight. The proposed method synthesizes hills that reflect the exemplar in Figure 4.9a.

Mountain ranges are another common topography, and they result from tectonic forces and erosion. Therefore, existing physically-based methods can accurately synthesize mountain ranges. Additionally, fBm and the RMF terrain model can achieve this as well by using a large number of octaves and a smaller value of H. Some mountains are shown in Figure 4.10a [17]. One hundred features were randomly placed with the constraint that they cannot appear in the dark region of Figure A.7b. The final placement is given in Figure 4.10b. The terrain depicts a series of mountain ranges, whose styles are controlled by the three profiles in Figures A.7(c–e). These profiles range from smooth and rounded to sharp and steep, producing a heterogeneous terrain. However, the smooth profiles used here are not as smooth as the previously mentioned hill profiles. Furthermore, profiles were assigned to a stroke's generators via feature ID number modulo three, in order to create this diversity. This assignment is visualized in Figure A.7g. Feature identification is visualized in Figure A.7f.

The initial costs of these peaks were calculated using fBm directly. The ridges in this result were formed by running Dijkstra's algorithm from select generator nodes and then selecting another set of random nodes (uniformly distributed over the graph) and tracing back on the least-cost path to the originating set. Running Dijkstra's algorithm assigns a cost to every ridge node. These costs were scaled by 50% in order to create well defined ridge lines. The final render in Figure 4.10c and

(a)



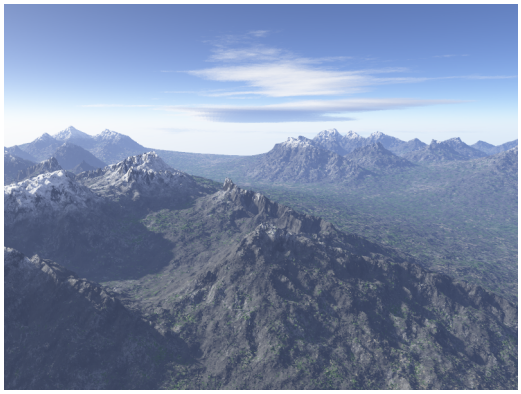


(b) (c)

Figure 4.9: (a) Reference photo [26], (b) generator node locations F, and (c) final render for the hills result.

(a)

(b)



(c)

Figure 4.10: (a) Reference photo [17], (b) generator node locations F, and (c) final render for the mountain range result.

height field in Figure A.7j result from the aforementioned inputs, parameters, and processes, along with $\mu_w = 1.0$, $r = 0.75$, $s = 1.01$, and $b = 4$. A smaller scaling value s increases the diversity of feature altitude by allowing some features to be near sea level, and the synthesis completed in 24 seconds.

Mountain ranges contain many tall, rough peaks and ridges that are heterogeneous, such as that in Figure 4.10a. They are usually grouped into distinct regions and stretch over large areas of land. Note that the initial location of ridges has a dendritic shape and that the features are heterogeneous: peak and ridge altitude vary, as well as their shape. Figure 4.10c shows a mountain range synthesized using the proposed methodology. There are three distinct regions of mountains, and peak and ridge altitude and shape varying across the scene. The resulting heterogeneous height field is aided by the use of multiple steep, yet varying profiles.

Edge weight randomness aids in the rough appearance of the features. The proposed method synthesizes mountain ranges that reflect the exemplar in Figure 4.10a.

4.2.3 River Terrace

River terraces are formed via renewed downcutting of a valley. Downcutting creates a new, narrower floodplain at a lower elevation than the original one. The surface of the older floodplain becomes a terrace on either side of the new floodplain [53]. An example is shown in Figure 4.11a. Existing procedural terrain synthesis methods struggle at replicating this effect because of a terrace's unique shape. Fortunately, the proposed method makes this an easy process. Three non-black feature regions were manually placed, and the corresponding strokes are shown in Figure 4.11b. Figure A.8d shows that a single profile was used to produce downcutting. The steep profile creates the steep transition between terraces. Feature identification is visualized in Figure 4.11b, but generator node costs could not be calculated using fBm. The location of each terrace was calculated as a function of cost from the dendritic path in Figure A.8a (the dendritic path has a cost of zero). More specifically, the costs from this path are quantized; the initial, larger set of costs is mapped to a smaller set of costs. The region of each cost in the smaller set is determined as a percentage range of the maximum cost from the dendritic path. These ranges are $(87.5\%, 100.0\%]$, $(50.0\%, 87.5\%]$, and $(12.5\%, 50.0\%]$; each region is assigned a cost of 0.0, 40.0, and 80.0, respectively. The final render in Figure 4.11c and height field in Figure A.8g result from the previously stated inputs, parameters, and processes, along with $\mu_w = 1.0$, $r = 0.5$, and $s = 1.1$. A smaller scaling value s allows the newest floodplain to appear close to sea level, and a bias value of $b = 8$ was used to bias towards local maxima and emphasize the transition between terraces. The synthesis completed in 8 seconds.

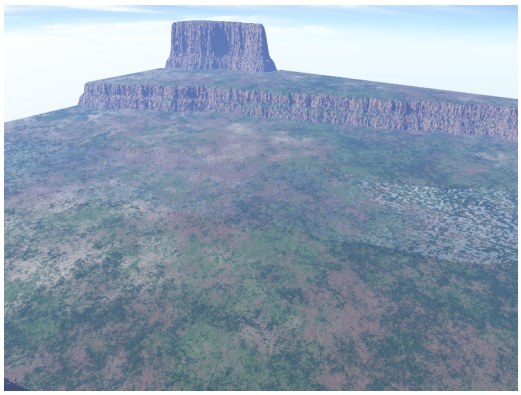
River terraces are composed of multiple flat regions at varying altitude, as seen in Figure 4.11a. The key attribute is to be able to synthesize flat regions at different altitudes, and Figure 4.11c shows that the proposed method can synthesize such regions. The included steep profile produces the steep transitions between the flat regions, and the randomness in edge weights results in rough transition zones. The proposed method synthesizes river terraces that reflect the exemplar in Figure 4.11a.

4.2.4 Fault

Faults result from bodies of rock sliding past one another. Physically-based methods have yet to simulate this process. Furthermore, procedural methods such as fBm and the RMF terrain model have difficulty forming the required long ridge lines. However, the proposed method can synthesize them. The San Andreas Fault is shown in Figure 4.12a [86]. Six features (one fault feature plus five surrounding features) were semi-automatically placed to create the desired effect,

(a)

(b)



(c)

Figure 4.11: (a) Reference photo [63], (b) generator node locations F, and (c) final render for the river terrace result.

and their corresponding strokes are shown in Figure 4.12b. Constraints on their placement were manually provided, but their initial dendritic shape was automatically generated; these constraints are discussed in the following paragraph. Figures A.9(c–d) show that two profiles were used: one for the fault, and one for the surrounding features. Two profiles were used to help differentiate the formation process: faulting results in sharp breaks in the Earth's crust, not in the surrounding hills. Feature identification is visualized in Figure A.9f.

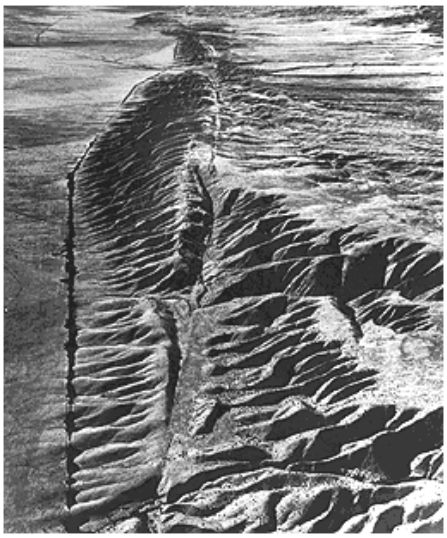
Not all costs were calculated using fBm though. Generator costs were assigned in two phases: 1) fault costs and 2) surrounding generator costs. Fault costs were assigned in two passes. The first pass creates the lip of the fault. The lip is created by selecting the nodes who have a path length of 8 from the feature in Figure A.9b; Dijkstra's algorithm assigns a path length greater than zero to each non-generator node. All nodes on the path in Figure A.9b have an initial cost of zero. Seventy-five ridges that emanate from these nodes are formed. These nodes must be within 80 units of a lip node and their cost is taken as 35% of the existing node cost. The surrounding features, which consist of 5 peaks and 6 ridges, have their initial cost calculated via fBm and are not permitted to appear within 75 units of a fault feature. The final render in Figure 4.12c and height field in Figure A.9i result from the previously mentioned inputs, parameters, and processes, along with $\mu_w = 1.5$, $r = 1.0$, $s = 1.01$, and $b = 3$. A smaller scaling value s allows the fault to appear close to sea level, and the synthesis completed in 10 seconds.

The faulting process creates very distinct features, as seen in Figure 4.12a. The prominent features are a dip (resulting in multiple local minima) near the fault itself, with many smaller ridges emanating from the fault. Furthermore, the faulting process results in a rougher terrain closer to the fault. Figure 4.12c shows a synthesized fault using the proposed method which accurately captures the aforementioned characteristics: a dip is seen along the fault line, many ridges emanate from the fault, and terrain roughness increases as distance to a ridge decreases. The proposed method synthesizes faults that reflect the exemplar in Figure 4.12a.

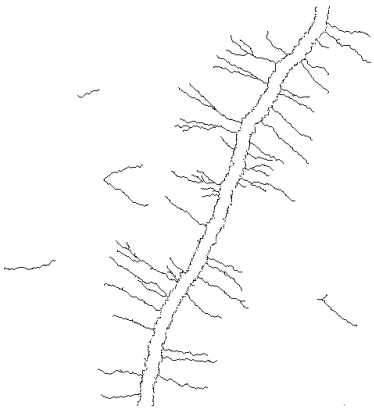
4.2.5 Crater, Cinder Cone Volcano, and Lunar Landscapes

Craters are formed via non-erosional processes, and it has yet to be shown that physically-based terrain synthesis methods can create such terrain features. Furthermore, their unique shape makes them difficult to capture using existing procedural terrain synthesis methods. Fortunately, the proposed method can synthesize such features.

An example crater is given in Figure 4.13a [6]. To create the render in Figure 4.13c, 100 features were both procedurally and manually placed (see Figure 4.13b). Figures A.10(c–f) show that four profiles were used to create the resulting heterogeneous terrain. The profiles range from steep to gentle, smooth slopes. The profile shown in Figure A.10e is always used when Dijkstra's algorithm searches the interior of the crater, to guarantee the crater's shape. Also, blending ignores the costs



(a)



(b)

(c)

Figure 4.12: (a) Reference photo [86], (b) generator node locations F, and (c) final render for the fault result.

of non-crater features inside the crater to guarantee the shape. Additionally, feature identification is visualized in Figure A.10g, and cost assignment depends on feature type.

In this result, two fBm height fields are used to define the initial costs of features: one for the background mountain range and midground foothills (fBm_M), and one for the remaining features (fBm_F). The fBm_M height field is sampled over a 12×12 grid and the resulting values are scaled by 10.0; this increases high-frequency detail. The resulting fBm_F values are scaled by 10.0 as well. In either case, cost assignment depends on feature type. The midground foothills (24 features) in Figure A.10a are assigned an initial cost of fBm_M scaled between $[50\%, 80\%]$ of the maximum fBm_M value $max\{\text{fBm}_M\}$. Scaling within this percentage range consists of normalizing fBm_M so that its values are within $[0, 1]$. Then, each normalized value v_n is scaled within the percentage range $[p_1, p_2]$ at its location, resulting in the scaled value v_s. This scaling process is given in Equation 4.1.

$$v_s = p1 \cdot max\{\text{fBm}_M\} + (p2 \cdot max\{\text{fBm}_M\} - p1 \cdot max\{\text{fBm}_M\}) \cdot v_n \tag{4.1}$$

The mountain range generator nodes (40 features) are assigned an initial cost of fBm_M scaled between $[0\%, 25\%]$ of $max\{\text{fBm}_M\}$. The foreground hills (35 features) are assigned an initial cost of fBm_F scaled between $[80\%, 100\%]$ of the maximum fBm_F value.

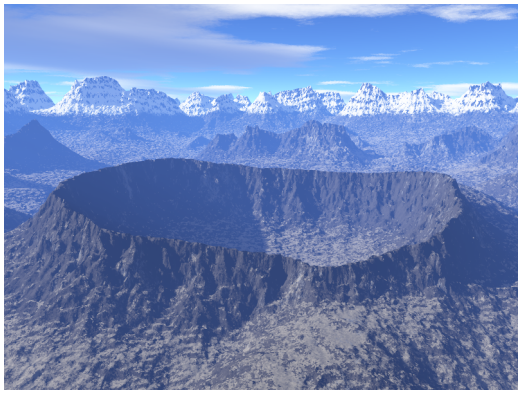
The crater generator nodes's costs require a more complicated process. An approximate cost field is calculated as per step 6 of Algorithm 3 and the average of the node costs in Figure A.10b minus 25.0 is calculated and stored as μ_c. Then, the final cost of these nodes is calculated as fBm_F scaled within the the range $[-0.2 \cdot \mu_c, 0.2 \cdot \mu_c]$. These costs are visualized in Figure A.10i.

Scaling fBm within the given ranges facilitates the placement of features at varying altitudes whose topography is a function of a fractal process. The final render in Figure 4.13c and height field in Figure A.10k result from the aforementioned inputs, parameters, and processes, along with $\mu_w = 1.5$, $r = 1.0$, $s = 1.1$, and $b = 4$. A smaller scaling value s ensured that the larger costing generator nodes were close to sea level, and the synthesis completed in 36 seconds.

Craters create a discontinuity at the Earth's surface. The crater impact results in a round, smooth basin and steep, rough walls. Furthermore, a ridge typically forms at the boundary of the impression/impact. These features are seen in Figure 4.13a. The synthesized crater in Figure 4.13c accurately captures these features. The rounded basin is accomplished with the use of a profile that terminates with a slope close to zero, edge weight randomness provides the rougher crater walls, and the application of Dijkstra's algorithm helps to ensure that the crater boundary forms a small ridge. The summation of edge weights in Dijkstra's algorithm makes the crater boundary locations local height maxima. The proposed method synthesizes craters that reflect the exemplar in Figure 4.13a.

(a)

(b)



(c)

Figure 4.13: (a) Reference photo [6], (b) generator node locations F, and (c) final render for the crater landscape result.

Cinder cone volcanoes are another non-erosional feature that have yet to be synthesized via physically-based terrain synthesis methods. Specifically, it has yet to be shown that such methods provide controls that govern the formation of the crater at the volcano's pinnacle. Similarly, the unique shape of cinder cone volcanoes makes them difficult to capture using existing procedural terrain synthesis methods.

An example cinder cone volcano is given in Figure 4.14a [35]. The volcano's crater was manually placed and the three surrounding features were procedurally placed, as shown in Figure 4.14b. Figures A.11(c–e) show that three profiles were used to create the resulting heterogeneous terrain. The profiles in Figure A.11(c,e) define the exterior and interior of the cinder cone, respectively. The shallow slope near the center of Figure A.11e's profile creates the rounded basin of the cinder cone volcano. The profile in Figure A.11d defines the shape of the features surrounding the cinder cone volcano. Finally, feature identification is visualized in Figure A.10f.

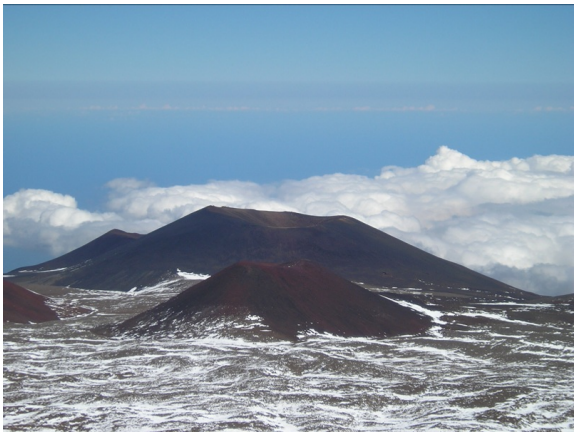
The costs of generator nodes are defined using fBm, and cost assignment depends on feature type. The cost of the cinder cone volcano generator nodes is calculated as a normalized 1D Gaussian function (situated along the x-axis) with the standard deviation σ a function of the maximum width W_{max} of the stroke in Figure A.11b. This calculation is given in Equation 4.2.

$$\sigma = (2.0 \cdot W_{max})/6.0 \tag{4.2}$$

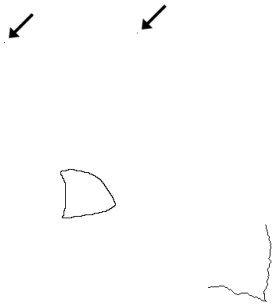
The origin is defined as the minimum x-value of a generator node in Figure A.11b. The Gaussian value is then scaled by 40.0 and fBm scaled within $[-10.0, 10.0]$ is added to this product. The surrounding features are assigned an initial cost of fBm scaled within a range that is a function of the maximum cost $max\{c\}$ of a cinder cone generator node. This cost range is $[1.50 \cdot max\{c\}, 2.0 \cdot max\{c\}]$. Furthermore, these features cannot reside within 128 units of a cinder cone feature. The initial costs of the surrounding feature ridges are defined as 50% of the existing node costs; all generator node costs are visualized in Figure A.11h.

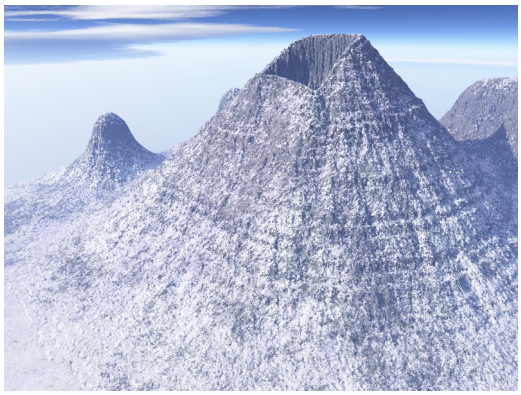
The scaling of fBm facilitates user control over the altitude of features in the final terrain. The final render in Figure 4.14c and height field in Figure A.11j result from the aforementioned inputs, parameters, and processes, along with $\mu_w = 1.0$, $r = 0.5$, $s = 2.0$, and $b = 3$. A larger scaling value s ensured that the features were well above sea level, and the synthesis completed in 7 seconds.

The exemplar cinder cone volcano in Figure 4.14a shows a basin at the peak of the volcano. This basin results from past and/or current volcanic activity and the exterior of a cinder cone volcano is characterized by a near constant slope. A cinder cone volcano synthesized using the proposed method is given in Figure 4.14c. This terrain accurately captures the previously mentioned features: a basin resides near the peak of the cinder cone volcano and the slope of the volcano's exterior is close to constant. Two profiles provide the desired terrain styles. The proposed method synthesizes craters that reflect the exemplar in Figure 4.14a.



(a)





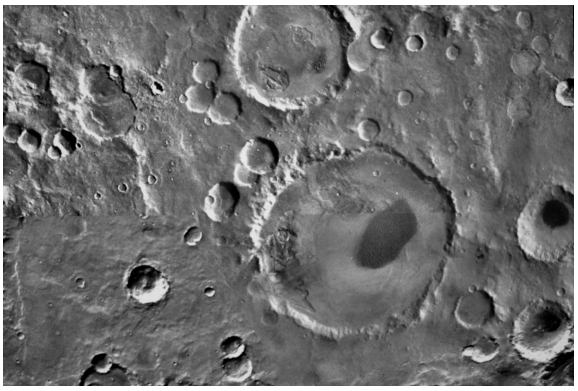
(b) (c)

Figure 4.14: (a) Reference photo [35], (b) generator node locations F, and (c) final render for the cinder cone result.

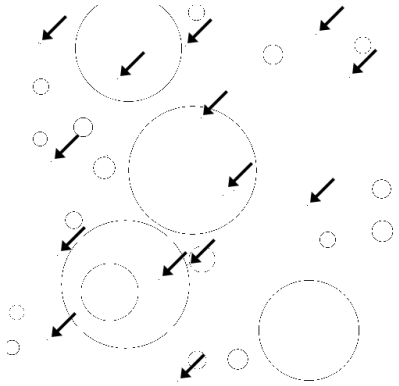
The last result in this subsection is a lunar landscape. Lunar landscapes typically depict numerous craters in close proximity to one another, as shown in Figure 4.15a [59]. For similar reasons to the crater landscape in Subsection 4.2.5, lunar landscapes are difficult to synthesize using existing methodologies. Thirty-eight features were both procedurally and manually placed, and their corresponding strokes are shown in Figure 4.15b. Figures A.12(b–e) show the profiles for the crater exterior, procedurally located hills, manually located hills, and crater interior, respectively. The profile in Figure A.12b is used to create the crater lip, the profiles in Figures A.12(c–d) are used to create the rounded hills, and the profile in Figure A.12e is used to create the bowl-shaped interior of the craters. Furthermore, Dijkstra's algorithm uses only the profile in Figure A.12e when searching the graph in crater regions, and costs from non-crater features do not contribute to blending in the crater's interior. This ensures that the craters will have a rounded profile in their interior. Feature identification is visualized in Figure A.12f.

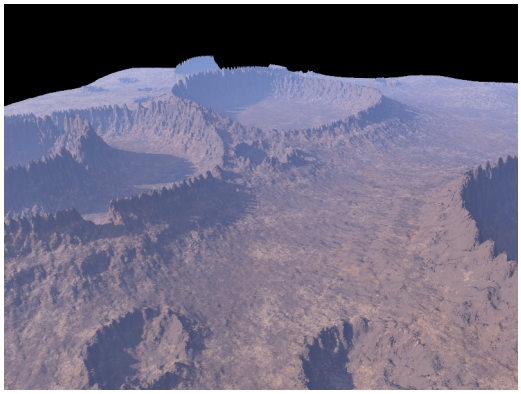
Crater generator nodes have their cost defined by fBm scaled within a specific cost range, but this cost range depends on crater size. Large craters have a radius in the interval of $[10.7, 85.3]$ units, and generator node costs for these craters are calculated by scaling fBm within a cost range of $[0.0, 12.5]$ units. Small craters have a radius in the interval of $[8.5, 14.2]$ units, and generator node costs for these craters are calculated by scaling fBm within a cost range of $[12.5, 18.75]$ units; the hills's generator costs are calculated using the same cost range as the small craters. The costs of all generator nodes are visualized in Figure A.12h. Regardless of crater radius, the actual radius is calculated as a uniform random value within the provided interval. These inputs create the final render in Figure A.12i. The final render in Figure 4.15c and height field in Figure A.12j result from the previously stated inputs, parameters, and processes, along with $\mu_w = 1.5$, $r = 1.0$, $s = 1.01$, and $b = 4$. A smaller scaling value s ensured that the craters with small radii were near sea level, and the synthesis completed in 18 seconds.

Lunar landscapes are not subject to the erosional processes that exist on Earth. Instead, they are characterized by craters of varying shape, area, and depth. The impact that creates the crater typically results in a smooth basin, and a ridge at the crater boundary. However, this ridge is less noticeable in small craters because the impact is relatively small for such craters. Furthermore, regions that do not contain craters are typically characterized by subtle rolling hills. A reference lunar landscape is given in Figure 4.15a and a lunar landscape synthesized using the proposed method is given in Figure 4.15c. The synthesized terrain contains multiple craters, some of which are nested, that vary in their shape, area, and depth. The crater basins are relatively smooth, and the smaller crater ridges are less prominent. Finally, non-crater regions are populated with subtle rolling hills. Adding more variety to crater shape can be accomplished by manually drawing the desired shapes or by adding a noise function to the circle equation presently used for crater shape. The proposed method synthesizes lunar landscapes that reflect the exemplar in Figure 4.15a.



(a)





(b) (c)

Figure 4.15: (a) Reference photo [59], (b) generator node locations F, and (c) final render for the lunar landscape result.

4.2.6 Musgrave Landscape

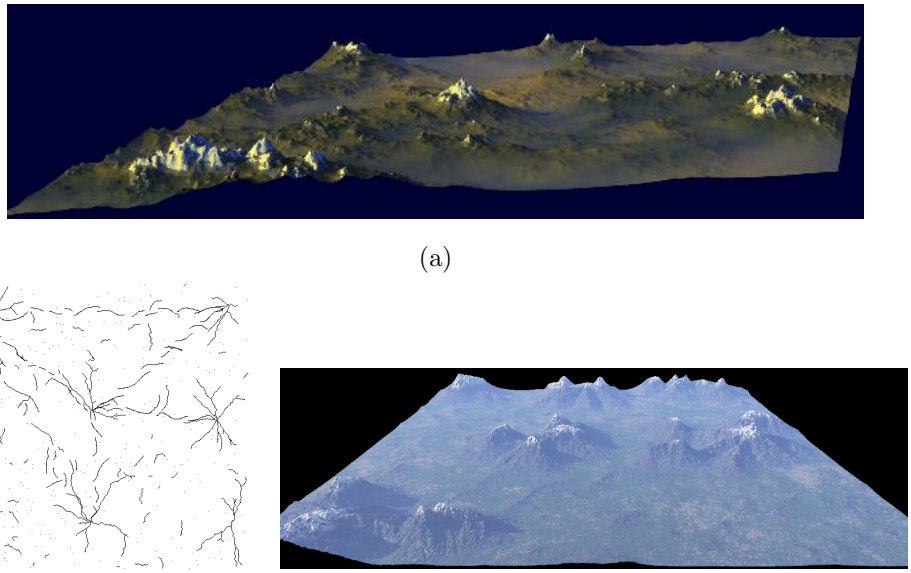
Musgrave's noise synthesis was able to produce the terrain in Figure 4.16a [29]. A similar terrain was pursued using the proposed synthesis method. Numerous features (402) were both procedurally and manually placed, and their corresponding strokes are shown in Figure 4.16b. Figures A.13(c–e) show that three profiles were used to create the resulting heterogeneous terrain. The profiles range from steep to rounded, smooth slopes. Feature identification is visualized in Figure A.13f, and cost assignment depends on feature type.

In this result, fBm (Octaves=6) is sampled over a 3×3 grid and the resulting values are scaled by 60.0. Furthermore, all ridge generator node costs are calculated as 37.5% of the existing node cost. Each generator node (46 features) in Figure A.13b is assigned a cost of fBm scaled between $[60\%, 85\%]$ of the maximum fBm value $max\{\text{fBm}\}$. The generator nodes (6 features) in Figure A.13a are assigned a cost of fBm scaled between $[40\%, 60\%]$ of $max\{\text{fBm}\}$. The lower altitude feature generators (150 features) are assigned a cost of fBm inverted and scaled between $[60\%, 85\%]$ of $max\{\text{fBm}\}$, while the low altitude hill generators (200 features) are assigned a cost of fBm scaled between $[85\%, 100\%]$ of $max\{\text{fBm}\}$. These strokes and costs are visualized in Figure A.13(g–h). Scaling fBm within the given intervals facilitates the placement of features at varying altitudes. The final render in Figure 4.16c and height field in Figure A.13j result from the aforementioned inputs, parameters, and processes, along with $\mu_w = 1.5$, $r = 1.0$, $s = 1.01$, and $b = 4$. A smaller scaling value s ensured that the larger costing generator nodes were close to sea level, and the synthesis completed in 27 seconds. Lastly, all heights were scaled by 45% to more closely approximate the reference render; scaling was performed in Terragen.

As previously mentioned in Subsection 4.2.2, mountain ranges contain many tall, rough peaks and ridges that are heterogeneous. They are usually grouped into distinct regions and stretch over large areas of land. An example mountain range with these features is given in Figure 4.16a. Similar to the previous mountain range result, Musgrave's synthetic terrain contains peaks and ridges at varying altitude. Furthermore, the shape of these features vary, creating a realistic terrain. Figure 4.16c shows a mountain range synthesized using the proposed methodology. Feature location in our terrain is similar to that in Musgrave's terrain. Our method is able to synthesize terrain features that vary in shape and altitude, and the usage of three varying profiles aids in a heterogeneous result. Additionally, edge weight randomness aids in the rough appearance of the features. The proposed method synthesizes mountain ranges that reflect the exemplar in Figure 4.16a.

4.3 Image Driven Terrains

Image driven terrain synthesis consists of terrains whose features are placed using edge detection, and/or whose edge weights are calculated according to an input texture. The results can



(a) (b) (c)

Figure 4.16: (a) Reference photo [29], (b) generator node locations F, and (c) final render for the Musgrave landscape result.

accommodate differing structures within the same scene, resulting in heterogeneous terrains. Subsections 4.3.1 and 4.3.2 demonstrate the effect of edge detection (ED) for feature placement and input texture pixel intensity for edge weights (EW), respectively. Subsection 4.3.3 demonstrates their combined effect and is referred to as ED/EW.

Edge detection on an input texture is used to define feature location in Subsection 4.3.1; edge weights are controlled by μ_w and r. The intensity of pixels in an input texture is used to calculate edge weights in Subsection 4.3.2; generator nodes in F are procedurally created. The results presented in Subsection 4.3.3 differ only in the input textures used for edge detection and edge weight calculation; the content of the input textures can vary in contrast, structure, and frequency. More specifically, all other parameters in Subsection 4.3.3 share the following parameter values: $s = 5.0$, $b = 3$, edge weights are scaled within the range $[0.25, 2.75]$, ridge generator costs are defined as 35% of the existing node cost, the number of ridges is calculated as a quarter of the number of connected components, and generator node cost comes directly from fBm. A Sobel operator is used to detect edges in the input texture. A Sobel operator was used because it is easy to implement, and is used as a proof of concept – any desired edge detection method can be used in practice. The threshold t_g on the gradient magnitude (for edge detection) is taken to be the halfway point between the minimum $grad_{min}$ and maximum $grad_{max}$ gradient magnitude. This calculation is given in Equation 4.3.

$$t_g = grad_{min} + 0.5 \cdot (grad_{max} - grad_{min}) \tag{4.3}$$

Furthermore, the input textures used in edge weight calculations eliminates the need for profiles, μ_w, and r. Each result is now discussed. An input texture and/or detected edges, as well as generator node locations and a final render accompany each result.

4.3.1 Edge Detection

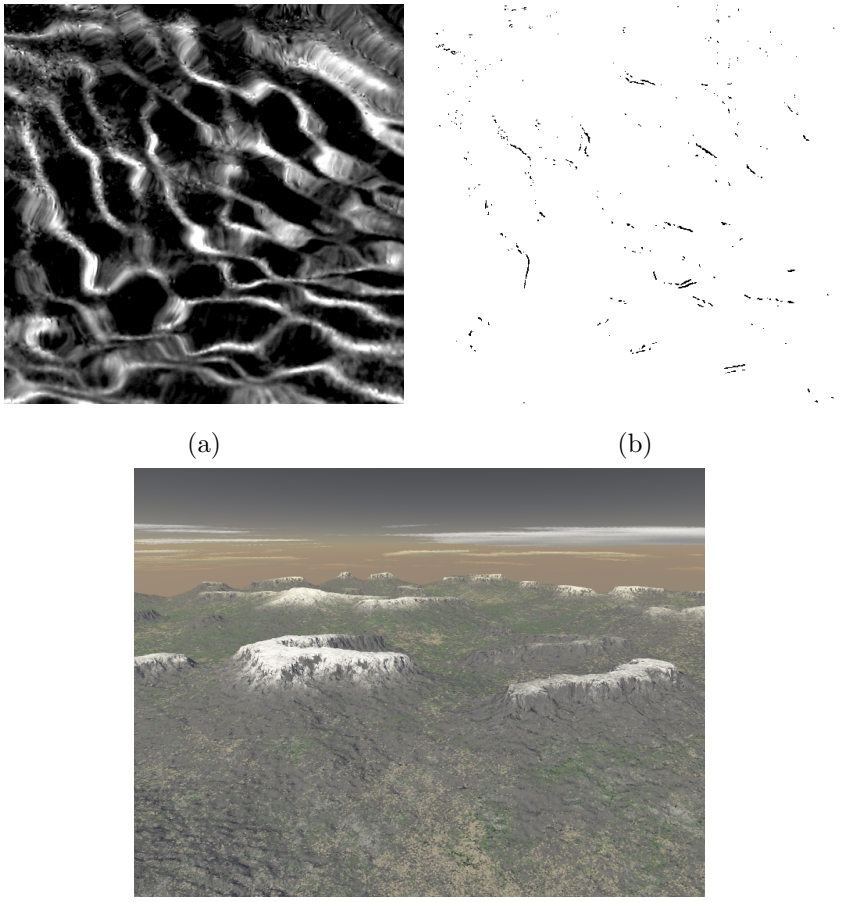
In this work, procedurally generated features are formed using the ridge synthesis process described in Section 4.1. However, features can also be placed via edge detection. A Sobel edge detector is applied to Figure 4.17a [76] to produce Figure 4.17b. The result is 288 features that resemble a mountain range. Then, 50 ridges whose generator costs are 30% of the existing node costs are created. A single profile was used to create a consistent terrain style, as shown in Figures A.14c. The profile creates a steep slope near the top of features, and gradually reaches a shallow slope near sea level. Feature identification is visualized in Figure A.14d, and generator node costs were assigned using fBm and are visualized in Figure A.14e. The final render in Figure 4.17c and height field in Figure A.14g result from the previously mentioned inputs, parameters, and processes, along with $\mu_w = 1.0$, $r = 0.75$, $s = 1.01$, and $b = 4$. The synthesis completed in 1 minute 11 seconds.

4.3.2 Edge Weights

In this work, edge weights are calculated according to Equation 3.3 in Chapter 3. However, edge weights can also be created via an input texture. Figure 4.18b [4] was converted to grayscale using the formula in Equation 4.4 to convert RGB color values to intensity values.

$$gray = 0.3 \cdot red + 0.59 \cdot green + 0.11 \cdot blue \tag{4.4}$$

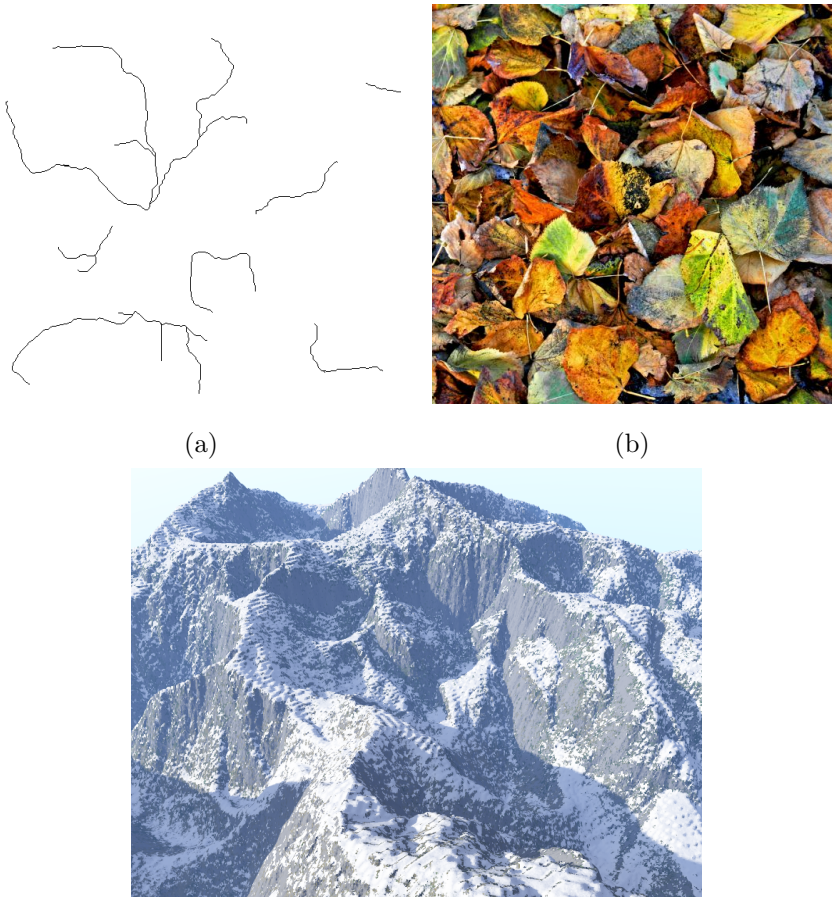
Then, each gray value is scaled within the range $[0.25, 2.5]$ and the corresponding node in the graph has this value assigned to its outgoing edges's weights. A range of $[0.25, 2.5]$ was chosen so that a variety of edge weights would be encountered in the graph, providing terrain heterogeneity. The interval $[0.25, 2.5]$ means that between any two adjacent nodes, the minimum height difference will be 0.25 units and the maximum height difference will be 2.5 units. This range of possible height differences facilitates edge weight variety and terrain heterogeneity. For example, a particular least-cost path consisting of edge weights near 0.25 will create a relatively gentle path, whereas a least-cost path consisting of edge weights near 2.5 will create a relatively steep path. Such paths co-existing in the same scene facilitates terrain heterogeneity. The usage of an input texture for edge weights eliminates the need for profiles, μ_w, and r. Figure 4.18a visualizes the input F. Generator locations were determined procedurally and costs were calculated using fBm; these costs are visualized in Figure A.15d. Also, feature identification is visualized in Figure A.15c. The final render in Figure 4.18c and height field in Figure A.15f result from these inputs, parameters, and processes, along with $s = 2.5$ and $b = 4$. The synthesis completed in 12 seconds.



(a) (b)

(c)

Figure 4.17: (a) Input texture [76], (b) detected edges, and (c) final render for the edge detection result.



(a) (b)

(c)

Figure 4.18: (a) Generator node locations F, (b) input texture [4], and (c) final render for the edge weights result.

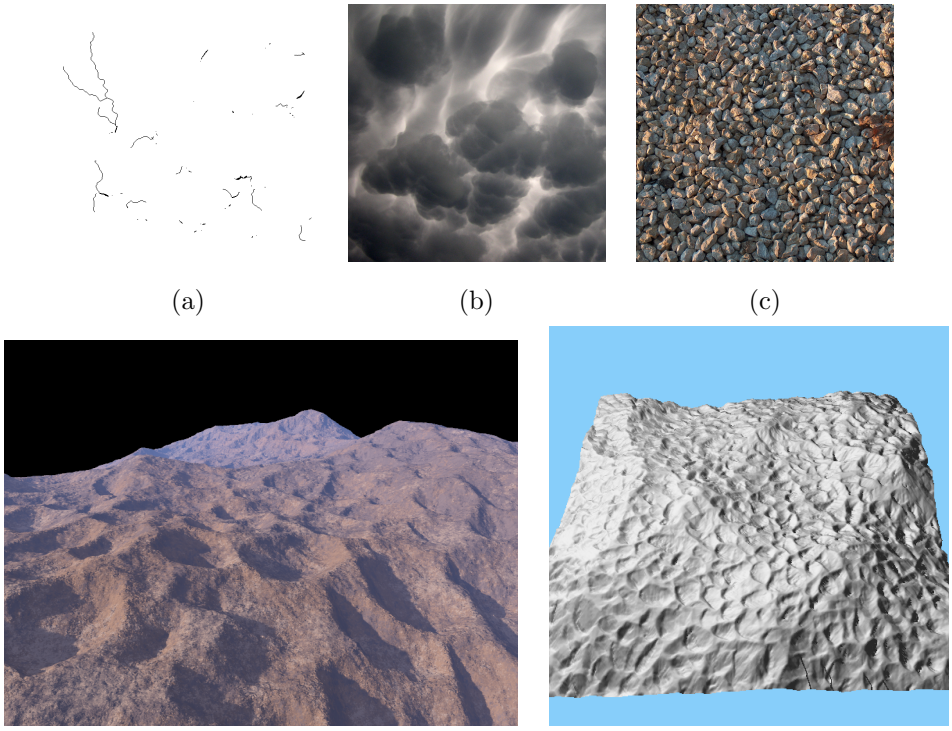
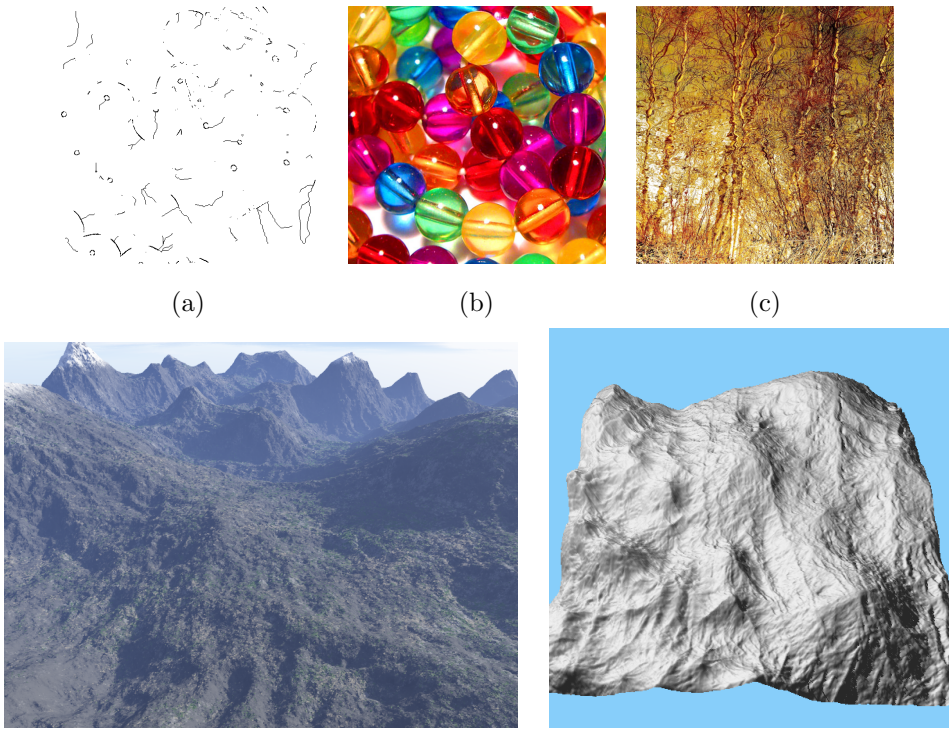


Figure 4.19: (a) Generator node locations F (includes 14 accompanying ridges), (b) edge detection texture [28], (c) edge weight texture [93], (d) final Terragen render, and (e) final OpenGL render for the first ED/EW result.

4.3.3 Edge Detection and Edge Weight Results

In the first combined result, generator node locations F are calculated by performing edge detection on Figure 4.19b [28] and synthesizing 14 accompanying ridges. The resulting set of locations F is visualized in Figure 4.19a. Edge weights are determined according to Figure 4.19c [93]. Specifically, the gray value of each pixel in Figure 4.19c is calculated according to Equation 4.4, and is scaled within the previously mentioned lower and upper edge weight bounds. Feature identification is visualized in Figure A.16e and the costs of generators are visualized in Figure A.16f. The final renders in Figure 4.19(d–e) and height field in Figure A.16i result from these inputs, parameters, and processes. The synthesis completed in 1 minute 3 seconds.

In the second combined result, generator node locations F are calculated by performing edge detection on Figure 4.20b [25] and synthesizing 57 accompanying ridges. The resulting set of locations F is visualized in Figure 4.20a. Edge weights are determined according to Figure 4.20c [16]. Specifically, the gray value of each pixel in Figure 4.20c is calculated according to Equation 4.4, and is scaled within the previously stated lower and upper edge weight bounds. Feature identification



(a) (b) (c)

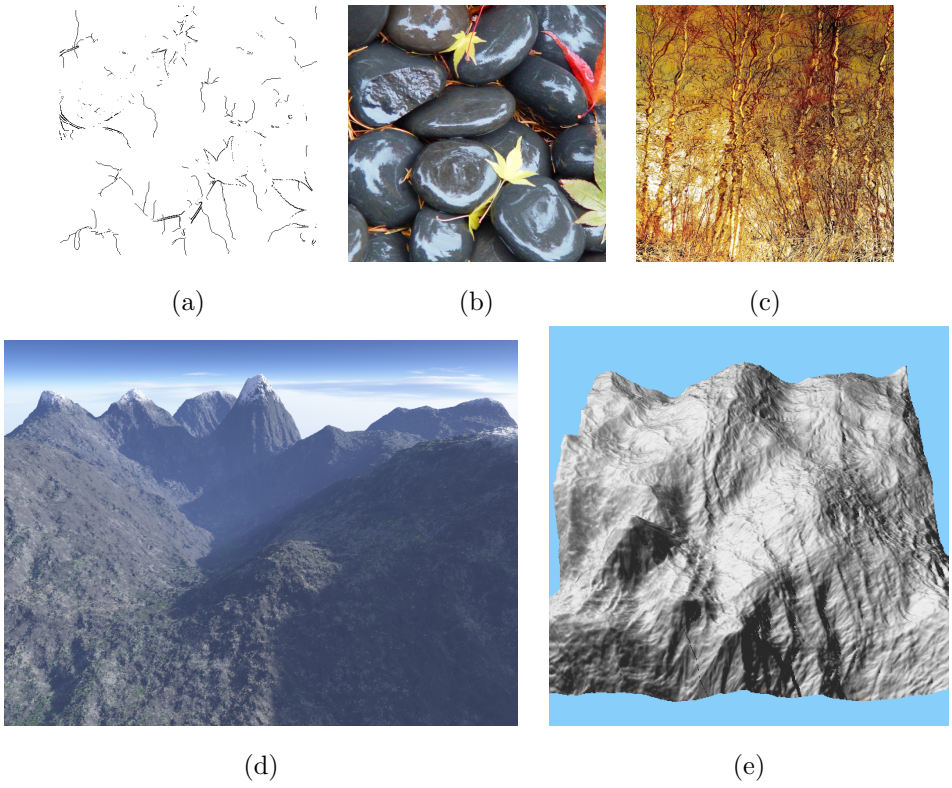
(d) (e)

Figure 4.20: (a) Generator node locations F (includes 57 accompanying ridges), (b) edge detection texture [25], (c) edge weight texture [16], (d) final Terragen render, and (e) final OpenGL render for the second ED/EW result.

is visualized in Figure A.17e and the costs of generators are visualized in Figure A.17f. The final renders in Figure 4.20(d–e) and height field in Figure A.17i result from these inputs, parameters, and processes. The synthesis completed in 2 minutes 8 seconds.

In the third combined result, generator node locations F are calculated by performing edge detection on Figure 4.21b [34] and synthesizing 86 accompanying ridges. The resulting set of locations F is visualized in Figure 4.21a. Edge weights are determined according to Figure 4.21c [16]. Specifically, the gray value of each pixel in Figure 4.21c is calculated according to Equation 4.4, and is scaled within the aforementioned lower and upper edge weight bounds. Feature identification is visualized in Figure A.18e and the costs of generators are visualized in Figure A.18f. The final renders in Figure 4.21(d–e) and height field in Figure A.18i result from these inputs, parameters, and processes. The synthesis completed in 2 minutes 15 seconds.

In the fourth, and final, combined result, generator node locations F are calculated by performing edge detection on Figure 4.22b [25] and synthesizing 57 accompanying ridges. The resulting set of locations F is visualized in Figure 4.22a. Edge weights are determined according to Figure 4.22c [39]. Specifically, the gray value of each pixel in Figure 4.22c is calculated according



(a) (b) (c)

(d) (e)

Figure 4.21: (a) Generator node locations F (includes 86 accompanying ridges), (b) edge detection texture [34], (c) edge weight texture [16], (d) final Terragen render, and (e) final OpenGL render for the third ED/EW result.

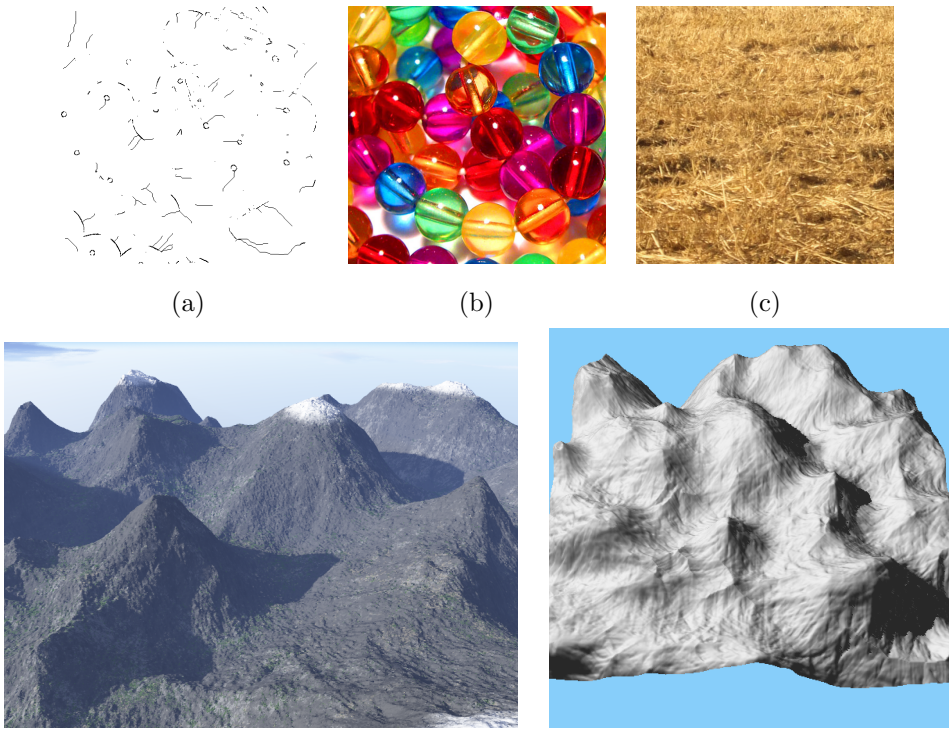


Figure 4.22: (a) Generator node locations F (includes 57 accompanying ridges), (b) edge detection texture [25], (c) edge weight texture [39], (d) final Terragen render, and (e) final OpenGL render for the fourth ED/EW result.

to Equation 4.4, and is scaled within the previously stated lower and upper edge weight bounds. Feature identification is visualized in Figure A.19e and the costs of generators are visualized in Figure A.19f. The final renders in Figure 4.22(d–e) and height field in Figure A.19i result from these inputs, parameters, and processes. The synthesis completed in 1 minute 15 seconds.

4.3.4 Embedded Imagery

The ability to manually place terrain features makes it possible to synthesize terrains whose dominant features depict recognizable images and/or words. Three results were created to demonstrate this effect: University of Saskatchewan (U of S) text, the U of S Imaging, Multimedia, and Graphics (IMG) lab logo, and the lambda (λ) symbol.

The synthesis of these results are governed by common input/parameter values. Specifically, $s = 2.5$ and $b = 3$; μ_w, r, and profiles are not used because each result uses Figure 4.23 [66] for the edge weight calculation process. The process is identical to that in Subsection 4.3.2. The gray value at each pixel is scaled so that it is within a range of $[0.25, 2.0]$. Furthermore, the costs of all generator nodes are determined by scaling fBm within a cost range of $[0.0, 20.0]$. All ridge

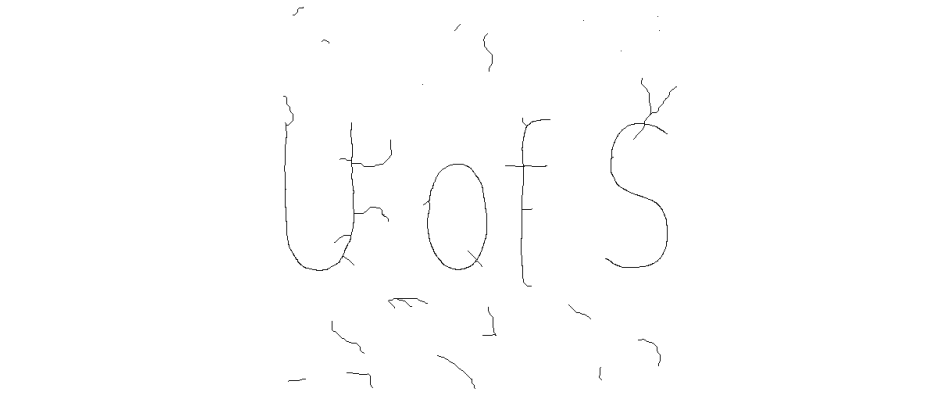
Figure 4.23: Visualization of the common input image [66] for edge weights in the embedded imagery terrains. This image is converted to grayscale and used for edge weight calculations.

generator nodes have a cost equal to 35% of the existing node cost (recall that Dijkstra's algorithm is applied for ridge synthesis). However, generator node locations F vary between the results. Finally, connected component labelling is performed to group and label strokes, which facilitates the blending of each feature (step 7 of Algorithm 3).

The first result that demonstrates a terrain with embedded imagery is the text "U of S". Figure 4.24a visualizes the manually determined generator node locations F which create the "U of S" text, as well as the procedurally synthesized surrounding strokes; Figure A.20c visualizes the connected component labelling. Weights were calculated as a function of the grayscale version of Figure 4.23, and generator node costs are visualized in Figure A.20d. The result of applying Algorithm 3 with the previously stated inputs and parameters produces the renders in Figure 4.24(b–c) and the height field in Figure A.20g. It took 9 seconds to synthesize the scene.

The second result that demonstrates a terrain with embedded imagery is the IMG logo. Figure 4.25a visualizes the manually determined generator node locations F which create the IMG logo, as well as the procedurally synthesized surrounding strokes; Figure A.21c visualizes the connected component labelling. Weights were calculated as a function of the grayscale version of Figure 4.23, and generator node costs are visualized in Figure A.21d. The result of applying Algorithm 3 with aforementioned inputs and parameters produces the renders in Figure 4.25(b–c) and the height field in Figure A.21g. It took 10 seconds to synthesize the scene.

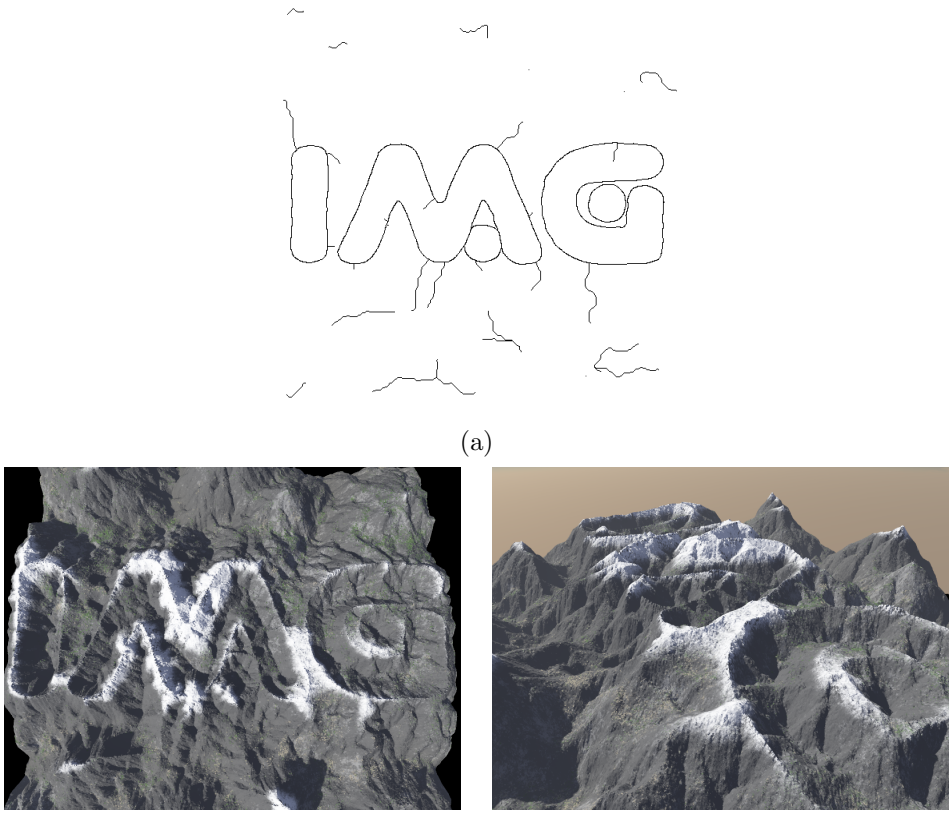
The third and final result that demonstrates a terrain with embedded imagery is the λ symbol. Figure 4.26a visualizes the manually determined generator node locations F which create the λ symbol, as well as the procedurally synthesized surrounding strokes; Figure A.22c visualizes the connected component labelling. Weights were calculated as a function of the grayscale version of Figure 4.23, and generator node costs are visualized in Figure A.22d. The result of applying Algorithm 3 with the previously mentioned inputs and parameters produces the renders in Figure 4.26(b–c) and the height field in Figure A.22g. It took 9 seconds to synthesize the scene.



(a)

(b) (c)

Figure 4.24: (a) Generator node locations, (b) final overhead render, and (c) final non-overhead render for the "U of S" text result.

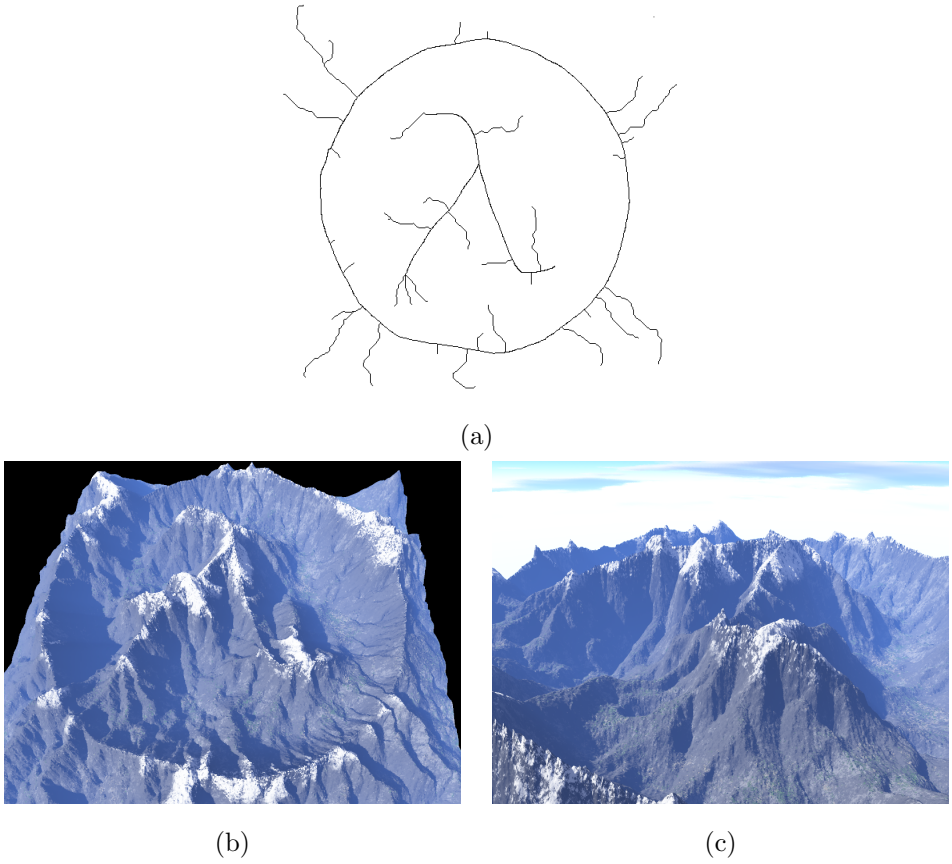


(a)

(b)

(c)

Figure 4.25: (a) Generator node locations, (b) final overhead render, and (c) final non-overhead render for the IMG logo result.



(a)

(b) (c)

Figure 4.26: (a) Generator node locations, (b) final overhead render, and (c) final non-overhead render for the λ symbol result.

Section 4.3 presented six figures (not including Subsection 4.3.4) that use edge detection for generator node locations F and/or pixel intensity for edge weights. Specifically, Figure 4.17 uses only edge detection, Figure 4.18 uses only pixel intensity for edge weights, and Figures 4.19 through 4.22 use both. These results, especially those in the latter set, suggest that structured high-frequency, high-contrast content could be better for edge weights than either unstructured high-frequency content (the purely random edge weights calculated in Equation 3.3), structured content with only localized high-frequency content (i.e. containing large smoothly varying regions), or low-contrast content (i.e. containing a small range of intensity values). The structure in the input texture used to calculate edge weights can be very apparent in the synthesized terrain, as in Figure 4.19, or it can be subtle, as in Figures 4.20 through 4.22. In either case, the structure adds additional detail and features for free; these effects are not explicitly modelled by hand.

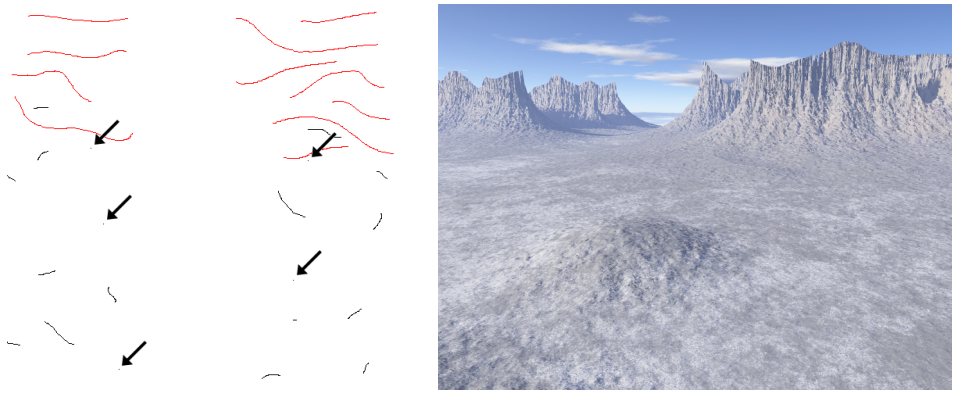
Different texture styles can be used as input for edge detection and/or edge weights. Textures can be categorized as follows [45]: regular, near-regular, irregular, near-stochastic, and stochastic. Regular textures have extremely high regularity of both texton placement and texton shape/structure, whereas stochastic textures do not. Near-regular, irregular, and near-stochastic categorize textures that are between these two extremes. The homogeneity of the resulting terrain is, among other attributes, a function of the input texture. For example, regular textures result in homogeneous terrains because they are fairly stationary and isotropic. Such a terrain is given in Figure 4.19. Our results indicate that texture stationarity and isotropy can create highly specialized terrains, but such textures should be avoided for general terrain synthesis.

However, textures that are relatively stochastic and high in contrast aid in the synthesis of heterogeneous terrains because of the increase in texture randomness and the texture's high dynamic range. Both of these properties result in highly varied edge weights that span the entire edge weight range. Examples of these textures are given in Figures 4.20 through 4.22. It is difficult to state that the usage of input textures with structured high-frequency, high-contrast content is better, because evaluating aesthetics can be highly subjective. However, the usage of input textures for edge weights offers a promising alternative to Equation 3.3 that is able to produce realistic terrains.

4.4 Sketch-Based Terrains

In sketch-based terrains, coloured pixels in an input image indicate feature location, and each colour corresponds to an input profile. This type of input shows that it is easy to customize realistic terrains.

The first sketch-based terrain contains hills in the foreground, a mountain range in the background, and a valley running through the center of the scene. Thirty-one features were manually placed as shown in Figure 4.27a. The background mountains correspond to red pixels and are



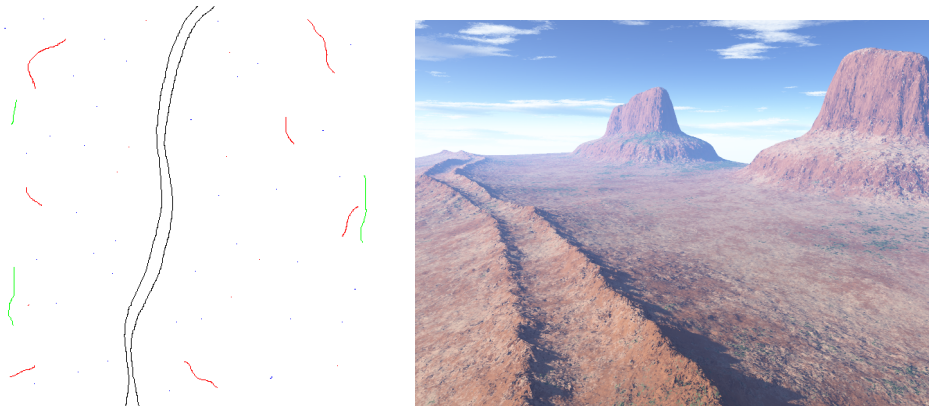
(a) (b)

Figure 4.27: (a) Generator node locations F and (b) final render for the sketch-based #1 result. Arrows are not part of the input and are merely visual aids for single node generators which are otherwise difficult to see.

created using the steep profile in Figure A.23b. The foreground hills correspond to black pixels and are created using the profile in Figure A.23c. These two profiles provide the desired steepness and smoothness in the aforementioned features, and create a heterogeneous result. The costs of the generator nodes are calculated using fBm (Octaves=6) sampled over a 12×12 grid and the resulting values are scaled by 40.0. Red pixels use fBm scaled within a percentage range of $[0\%, 35\%]$ of the maximum fBm cost. Black pixels use a percentage range of $[85\%, 100\%]$. The final render in Figure 4.27b and height field in Figure A.23g result from the previously stated inputs, parameters, and processes, along with $\mu_w = 1.5$, $r = 0.75$, $s = 1.25$, and $b = 3$. A smaller scaling value s places the foreground hills near sea level, and the synthesis completed in 12 seconds.

The second sketch-based terrain contains approximately parallel ridge lines running through the center of the scene, with accompanying hills and rock formations. The scene contains features that unintentionally resemble Devils Tower (located in Wyoming, USA). Fifty-eight features were manually placed as shown in Figure 4.28a. The ridge lines correspond to black pixels and are created using the profile in Figure A.24b. The tiered rock formations correspond to green pixels and are created using the profile in Figure A.24c. The hill features correspond to red pixels and are created using the profile in Figure A.24d, and the symmetric hill features correspond to blue pixels and are created using the profile in Figure A.24e. The provided profiles help create the desired terrain styles and a heterogeneous result.

The costs of the generator nodes are calculated using fBm (Octaves=6) sampled over a 12×12 grid and the resulting values are scaled by 48.0. Black pixels use fBm scaled within a percentage range of $[67.5\%, 85\%]$ of the maximum fBm cost. Green pixels are scaled within $[0\%, 30\%]$, red pixels are scaled within $[92.5\%, 97.5\%]$, and blue pixels are scaled within $[97.5\%, 100\%]$. The final



(a) (b)

Figure 4.28: (a) Generator node locations F and (b) final render for the sketch-based #2 result. Image (a) contains multiple single node generators, but arrows are not used to allow for a clearer visualization of F.

render in Figure 4.28b and height field in Figure A.24i result from the aforementioned inputs, parameters, and processes, along with $\mu_w = 1.5$, $r = 0.75$, $s = 1.1$, and $b = 3$. A smaller scaling value s places the hill features near sea level, and the synthesis completed in 9 seconds.

The third sketch-based terrain contains two opposing mountain ranges with a valley running through the center of the scene. Thirty-eight features were manually placed as shown in Figure 4.29a. The mountains correspond to black pixels and are created using the profile in Figure A.25b. The rough, steep features in the valley correspond to red pixels and are created using the profile in Figure A.25c. The smooth hills in the valley correspond to blue pixels and are created using the profile in Figure A.25d. The provided profiles help create the desired terrain styles and a heterogeneous result.

The costs of the generator nodes are calculated using fBm (Octaves=6) sampled over a 12×12 grid and the resulting values are scaled by 60.0. Black pixels use fBm scaled within a percentage range of $[0\%, 25\%]$ of the maximum fBm cost. Red pixels are scaled within $[65\%, 85\%]$, and blue pixels are scaled within $[85\%, 100\%]$. The final render in Figure 4.29b and height field in Figure A.25h result from the previously mentioned inputs, parameters, and processes, along with $\mu_w = 2.0$, $r = 1.0$, $s = 1.01$, and $b = 3$. A smaller scaling value s places the valley features near sea level, and the synthesis completed in 11 seconds.

The fourth, and final, sketch-based terrain contains foothills of varying altitude and style. Fifteen features were manually placed as shown in Figure 4.30a. The foothill running vertically through the center of the scene, and the larger horizontal foothill in the foreground correspond to black pixels and are created using the profile in Figure A.26b.

(a) (b)

Figure 4.29: (a) Generator node locations F and (b) final render for the sketch-based #3 result. Arrows are not part of the input and are merely visual aids for single node generators which are otherwise difficult to see.

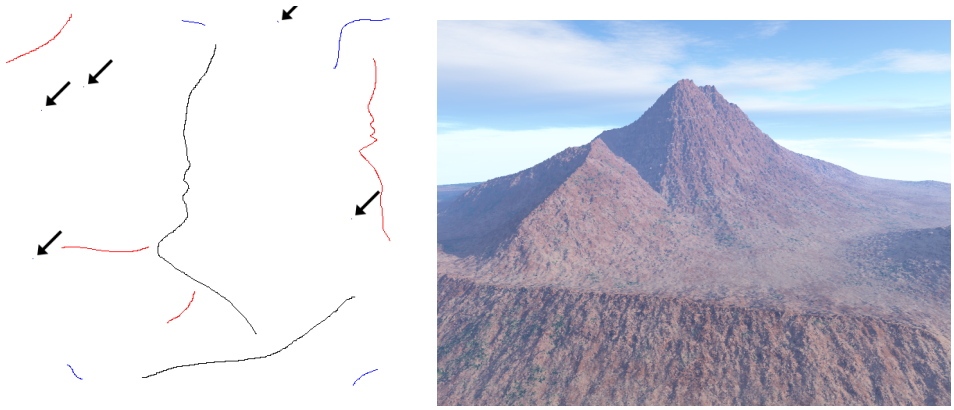
The surrounding features are composed of red and blue pixels which are created using the profiles in Figures A.26c and A.26d, respectively. The provided profiles help create the desired terrain styles and a heterogeneous result.

The costs of black and red pixels are calculated as their distance to the center of the image scaled by 85.0. Blue pixel costs are calculated as the minimum of this distance-based cost and a random value within the cost range of $[80.75, 85.0]$. The final render in Figure 4.30b and height field in Figure A.26h result from the aforementioned inputs, parameters, and processes, along with $\mu_w = 1.5$, $r = 0.75$, $s = 1.01$, and $b = 3$. A smaller scaling value s places the foothills near sea level, and the synthesis completed in 7 seconds.

4.5 Methodology Evaluation

The proposed terrain synthesis method is now compared to the RMF terrain model and the work of Zhou et al. [92] – the current state-of-the-art in procedural and controllable terrain synthesis methods, respectively. A brief comparison to physically-based methods is presented as well.

Physically-based synthesis methods simulate actual erosional processes. This fact is both their strength and weakness. The resulting terrains are formed by the same processes that govern the Earth's topography, but the resulting terrains are limited by the accuracy of these physical models. The results are realistic, but their range is limited: terrain features such as craters, cinder cone volcanoes, and lunar landscapes have yet to be synthesized. Furthermore, control over terrain placement and style is difficult because the majority of physically-based methods begin by distributing water on the terrain's surface [9]. Height field manipulation via water distribution is



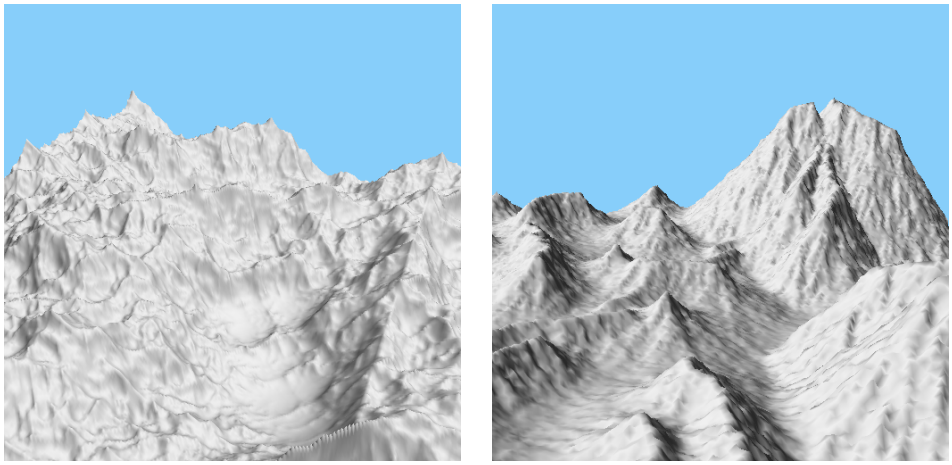
(a) (b)

Figure 4.30: (a) Generator node locations F and (b) final render for the sketch-based #4 result. Arrows are not part of the input and are merely visual aids for single node generators which are otherwise difficult to see.

not easy. As a result, the terrains in Subsection 4.3.4 and Section 4.4 are very difficult to replicate. Finally, these methods require an initial terrain to erode, requiring the use of another synthesis method. Physically-based methods produce realistic results, but the possible range of results is relatively small.

In comparison, the proposed method has shown that it can easily synthesize features such as craters (Figure 4.13c), cinder cone volcanoes (Figure 4.14c), and lunar landscapes (Figure 4.15c). Feature placement can be explicitly provided via a hand-drawn input image, and terrain style can be controlled using hand-drawn profiles. Allowing hand-drawn inputs makes terrain placement and style an easy and controllable process. Lastly, heavy reliance on an initial terrain is not required.

The RMF terrain model is a well known procedural terrain synthesis method that addresses the homogeneity of fBm by producing heterogeneous height fields with valleys at varying altitudes. Figure 4.31 shows a close up comparison of a RMF height field (Figure 4.31a) and a terrain synthesized via the proposed method (Figure 4.31b). This comparison, along with the synthesized terrains in Sections 4.2 through 4.4, emphasizes the ability of our method to create extended features such as ridges and specific features such as craters, cinder cone volcanoes, and lunar landscapes. Ridged multifractals allow a more limited range of features. However, this comparison also shows that the RMF terrain model has more high-frequency detail than Figure 4.31b. Our method is able to incorporate such detail, but it comes at an added cost to the user. The RMF model can be used directly for the costs of generators, but incorporating this detail into hand-drawn profiles is more difficult. The user can attempt to approximate the detail by hand, or such detail can be added to the profile in a post-processing step. However, the profile must be monotonically decreasing. In either case, the addition of this detail is easier in the RMF model.



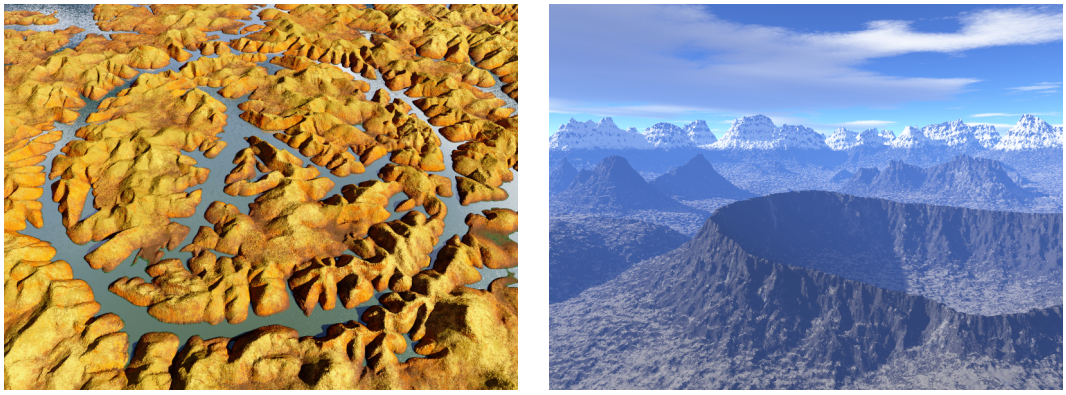
(a) (b)

Figure 4.31: A comparison between a terrain synthesized by the ridged multifractal process (a) and a terrain synthesized by the proposed method (b).

The RMF terrain model is controlled by difficult to use parameters: H, lacunarity, octaves, offset, and gain. The parameter H determines the fractal dimension of the roughest areas (the smaller H is, the rougher the terrain becomes), lacunarity is the gap between successive frequencies, octaves is the number of frequencies in the RMF model, offset raises the terrain from sea level, and gain controls the weighting of successive contributions by the previous signal. It is not easy to determine how these parameters control the resulting terrain, which makes feature placement and terrain style control a tedious process. Lastly, due to the multiplicative nature of the RMF terrain model, it is prone to divergence. In comparison, the proposed method provides easy to use, hand-drawn feature placement and terrain style is controlled using hand-drawn profiles.

Zhou et al. [92] presented an example-based terrain synthesis method to address the issue of feature placement. In their work, patches from a sample terrain are joined using graph cuts to generate new terrain. The synthesis is guided by a user-sketched feature location map that specifies where terrain features occur in the resulting synthetic terrain. Their work addresses a recurring problem in terrain synthesis: control over feature placement. However, features not present in the input terrain cannot be synthesized – the final terrain feature set is limited to the terrain features in the provided input. Lastly, patch-based sampling was designed for stationary signals making heterogeneous terrains more difficult.

Additionally, a correct patch size is difficult to calculate, and it is a sensitive parameter: a small patch size may not contain many terrain features, limiting the diversity of the final terrain. Large patch sizes may include more terrain features, but the final terrain can be homogeneous (less room for additional patches) and may have difficulty aligning with the provided feature loca-



(a) (b)

Figure 4.32: A comparison between a terrain synthesized Zhou et al.'s method (a) and a terrain synthesized by the proposed method (b).

tion map. Furthermore, the sample terrain requires the use of another terrain synthesis method, whether it be a Digital Elevation Model (DEM), or the output of an existing physically-based or procedurally-based terrain synthesis method. Figure 4.32 shows a comparison of Zhou et al.'s method (Figure 4.32a) and a terrain synthesized via the proposed method (Figure 4.32b). This comparison demonstrates the difficulty of heterogeneous terrains in their method, resulting from the exclusion of new features and the use of a single patch from the provided sample terrain.

4.6 Algorithm Performance

Showing that per-stroke cost is consistent demonstrates that Algorithm 3 scales appropriately, assuming that over large scales, the density of features is constant. Per-stroke cost refers to the area that Dijkstra's algorithm searches when synthesizing each feature individually. This section provides a brief theoretical argument as to why per-stroke cost is consistent when feature density is fixed. The hypothesis that per-stroke cost is consistent when feature density is fixed results from the way in which Dijkstra's algorithm terminates. Dijkstra's algorithm is terminated if the current node cost is within a certain threshold from the sea level cost or if the current node height is less than some percentage of the approximate terrain height at that location. If the density and distribution of features is similar between a small and large input texture, near-constant effort per feature will result because of way Dijkstra's algorithm is terminated. The thresholds that terminate Dijkstra's algorithm will be satisfied in similar locations relative to the originating stroke, though boundary conditions and initial generator cost will result in small differences in the termination frontier for different terrain sizes; Dijkstra's algorithm will terminate at the edges of the underlying graph in smaller terrains, but will explore past these edges in larger terrains. Furthermore, initial generator

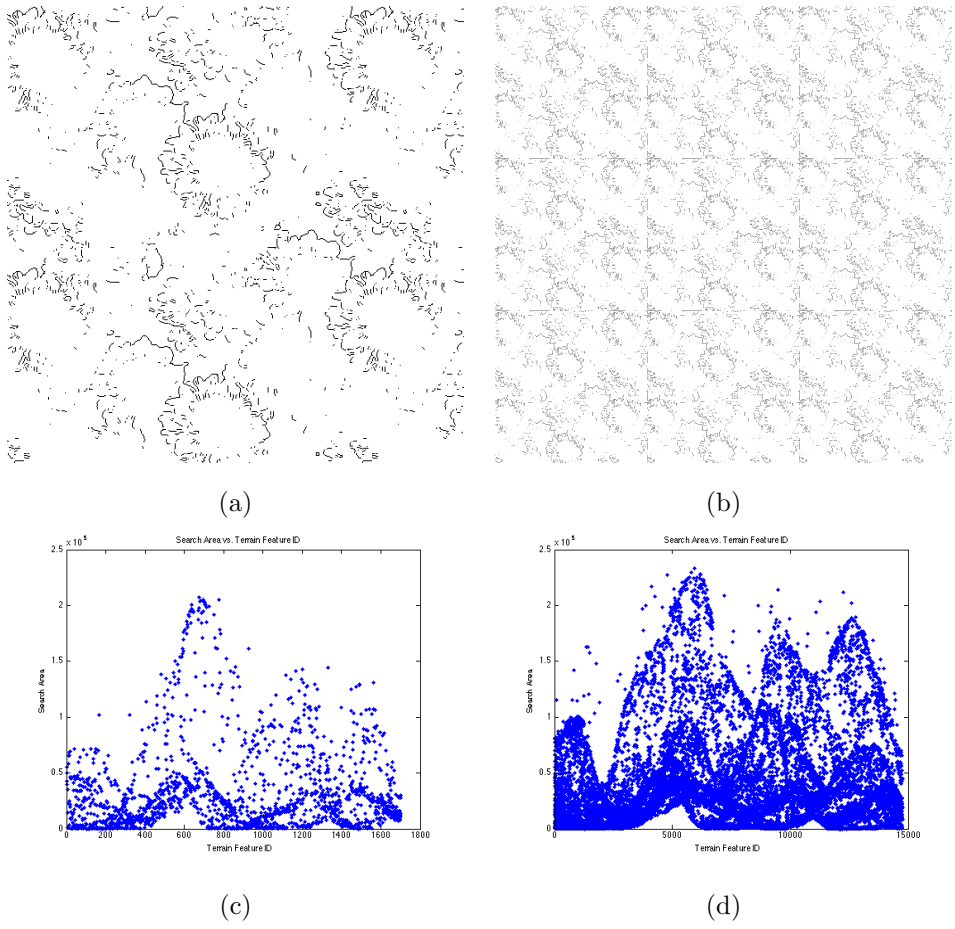


Figure 4.33: (a) Generator node locations for the 512×512 terrain, (b) generator node locations for the 1536×1536 terrain, (c) scatter plot visualizing the 512×512 terrain per-stroke cost, and (d) scatter plot visualizing the 1536×1536 terrain per-stroke cost in the first performance example. Images (c–d) show that there are only small deviations in the maximum per-stroke cost as the number of terrain features increases.

cost will impact how quickly that feature's profile reaches the sea level cost and/or becomes smaller than some percentage of the approximate terrain height. However, the increase in this constant (effort per feature) will be much smaller than the increase in terrain size.

To validate our claims, two tests were carried out to determine if the per-stroke cost is consistent across terrain size. Each test measured the average per-stroke cost on a 512×512 terrain and on a 1536×1536 terrain. Feature placement was provided using edge detection, similar to the process outlined in Subsection 4.3.1. To ensure that the small and large textures are statistically the same, the 512×512 texture was tiled to create the 1536×1536 texture.

The feature locations for the first test are given in Figure 4.33(a–b). The 512×512 terrain contained 1703 features and took 4 minutes 39 seconds to synthesize. On average, each feature's

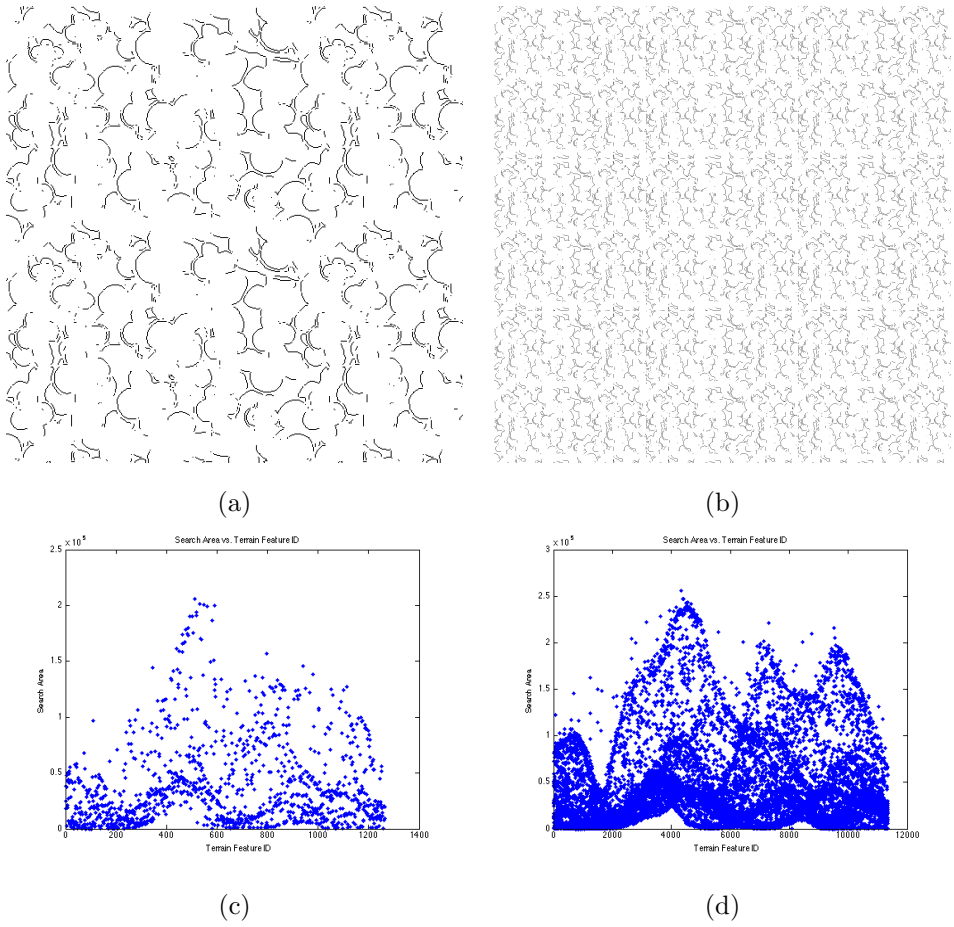


Figure 4.34: (a) Generator node locations for the 512×512 terrain, (b) generator node locations for the 1536×1536 terrain, (c) scatter plot visualizing the 512×512 terrain per-stroke cost, and (d) scatter plot visualizing the 1536×1536 terrain per-stroke cost in the second performance example. Images (c–d) show that there are only small deviations in the maximum per-stroke cost as the number of terrain features increases.

area was 35463.38 pixels. A scatter plot visualizing each feature's area is given in Figure 4.33c. The 1536×1536 terrain contained 14758 features, and on average, each feature's area was 42545.88 pixels. This terrain took 55 minutes 1 second to synthesize. A scatter plot visualizing each feature's area is given in Figure 4.33d. The increase in mean per-stroke cost in the larger terrain results from two factors: 1) features near graph boundaries in the 512×512 terrain are allowed to expand past these boundaries after image tiling, and 2) initial generator cost varies across each tile. However, the increase in mean per-stroke cost is far less than the increase in feature number or terrain size. The ratio between the two mean per-stroke costs is 0.83 and the two scatter plots in Figure 4.33(c–d) show that the magnitude of the feature areas is consistent across terrain resolution.

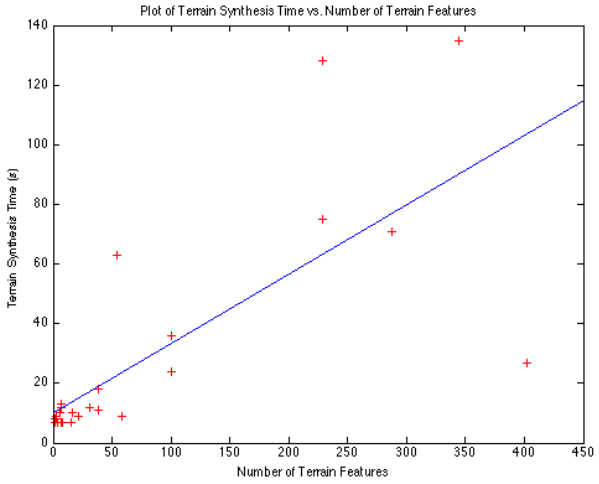


Figure 4.35: Plot of terrain synthesis time (s) vs. total number of terrain features for all resulting synthetic terrains.

The feature locations for the second test are given in Figure 4.34(a–b). The 512×512 terrain contained 1264 features and took 3 minutes 48 seconds to synthesize. On average, each feature's area was 38650.05 pixels. A scatter plot visualizing each feature's area is given in Figure 4.34c. The 1536×1536 terrain contained 11349 features, and on average, each feature's area was 46889.64 pixels. This terrain took 47 minutes 55 seconds to synthesize. Like the first test, the increase in mean per-stroke cost is far less than the increase in feature number or terrain size. A scatter plot visualizing each feature's area is given in Figure 4.34d. The ratio between the two mean per-stroke costs is 0.82. Furthermore, the two scatter plots in Figure 4.34(c–d) show that the magnitude of the feature areas is consistent across terrain resolution.

Figure 4.35 plots terrain synthesis time vs. total number of terrain features for all resulting synthetic terrains. The figure shows that in general, synthesis time increases with feature count, supporting the claim that the algorithm scales appropriately and is dependent on the total number of terrain features. Though the 1536×1536 terrains take 55 minutes 1 second and 47 minutes 55 seconds for the two examples, respectively, a good approximate terrain is available in 6 seconds and 7 seconds for a 512×512 terrain (based on the two presented examples), and in 3 minutes 13 seconds and 3 minutes 44 seconds for a 1536×1536 terrain. For rapid turnaround such as initial sketching and large-scaled feature layout, the approximate terrain will often suffice, and the expensive blending stage can be left until a fairly late stage in the synthesis process. Furthermore, a quality/time tradeoff is available because the algorithm can be sped up by picking different approximate height and sea level cost thresholds, at the expense of introducing seams.

However, the two tests show that the proposed algorithm scales properly. In the first test, the 512×512 terrain synthesized in 4 minutes 39 seconds and the 1536×1536 terrain synthesized in 55 minutes 1 second – the larger terrain took 12.61X longer. In the second test, the 512×512 terrain synthesized in 3 minutes 48 seconds and the 1536×1536 terrain synthesized in 47 minutes 55 seconds – the larger terrain took 11.83X longer. Considering the slight increase in mean per-stroke cost in the larger terrains and the fact that the larger terrain is 9X larger (than the smaller terrain), this increase in synthesis time is appropriate. Furthermore, the amount of increase was consistent between the two tests (12.61 vs. 11.83) and the mean per-stroke cost ratios (between the smaller and larger terrains) were also consistent (0.83 vs. 0.82), showing that neither of the examples were specially invented to bias the results.

Chapter 5

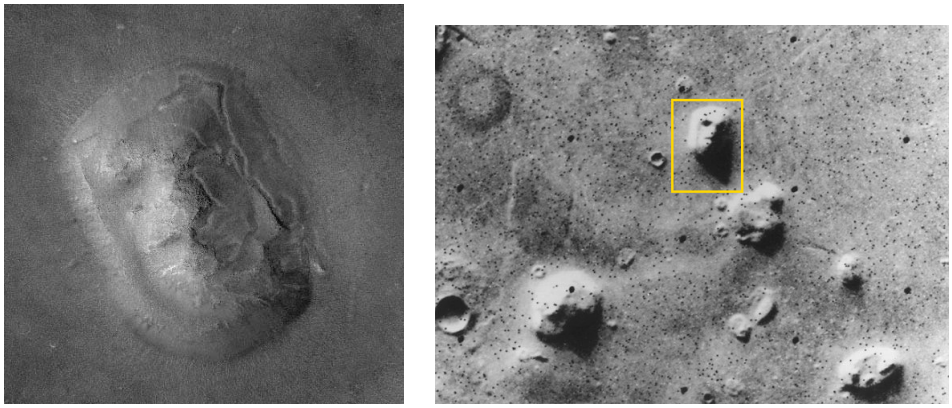
Pareidolia

5.1 Overview

Pareidolia refers to the phenomenon where a vague or imperfect sensory input is mistakenly interpreted as something familiar, such as a human face. Common examples include images of animals or faces in clouds, or hidden messages in music played in reverse. A well documented [58, 60] example of the pareidolia effect is the *Face on Mars*, as depicted in Figure 5.1. When the sunlight hits the surface of one of the Cydonian mesas on Mars at a specific angle and direction, the terrain in Figure 5.1a [60] casts a shadow resulting in the human face seen in Figure 5.1b [58].

The terrain synthesis method presented in Chapter 3 provides user control over feature placement. Motivated by the *Face on Mars* phenomenon, synthesizing terrains whose shadows resemble an image and/or word is a creative, artistic way to demonstrate user control over feature placement. Furthermore, this chapter combines feature placement using an input image, and a novel method for the initialization of generator cost. This combination makes it easy to cast shadows that resemble an image and/or word. The creation of terrain from shadow is similar to Yu and Chang's [90] work on surface reconstruction via shadow graphs. The difference here is that shadow graphs use input images as the sole source of constraints for surface creation – many images are required to generate enough constraints for a given surface. In this chapter, we only use one input image, but additional constraints are generated because the final terrain must look realistic. Furthermore, Yu and Chang place relatively little emphasis on what the reconstructed surface looks like, so long as it accurately represents the shadow data.

This chapter presents a novel terrain synthesis algorithm for pareidolia effects. The method is currently a proof of concept – additional terrains that realistically capture more shadow regions with more varied shape are required to establish this as a robust technique. Early results such as those in Figure 5.2 indicate that further work is required to make the technique more robust; too many terrain features are isolated and jagged, producing an unrealistic final terrain. Also, the non-shadow regions are too flat. An updated method would incorporate additional features with heights that do not produce shadows (from the given light direction). The target shadow images are taken from the work of Mould and Grant [54].



(a) (b)

Figure 5.1: These two images show the famous *Face on Mars* phenomenon. When the sunlight hits the surface of one of the Cydonian mesas on Mars at a specific angle and direction, the terrain in Image (a) [60] casts a shadow resulting in the human face seen in Image (b) [58].

Section 5.2 introduces the inputs and parameters to the method and their impact on G, and Section 5.3 discusses each step of the proposed algorithm. Section 5.4 concludes by showing two renders produced via the proposed method (Algorithm 4): one terrain embeds the word "TEXT" in its shadows and the other terrain embeds the image of a smiley face in its shadows.

5.2 Inputs and Parameters

The creation of the pareidolia renders in Section 5.4 requires three inputs and three parameters which provide control over the synthesized terrain. Each of the inputs and parameters are summarized in Table 5.1. It is shown that embedding images and/or words in shadows for pareidolia effects is easy to achieve using the provided inputs and parameters.

The three inputs are 1) binary *shadow image* I, 2) base terrain $T_b(n)$, and 3) secondary feature generator costs $C_{sec}(n)$. The binary *shadow image* I indicates the desired shadow regions; dark pixels indicate shadow. The use of I is discussed in Section 5.3. The base terrain $T_b(n)$ acts as an initial terrain to which terrain features will be added, and is seen in regions where no features have been added. The base terrain can be synthesized using any of the existing terrain synthesis algorithms, but it is recommended that Algorithm 3 in Chapter 3 be used to ensure consistency between $T_b(n)$ and the added features. Furthermore, the shadow area introduced in $T_b(n)$ should be reduced to avoid large changes to the desired shadows. However, this shadow area should not be eliminated, to maintain a realistic result. A flat base terrain would stand in contrast to the synthesized features and look artificial. Thus, a rough base terrain with many smaller features is

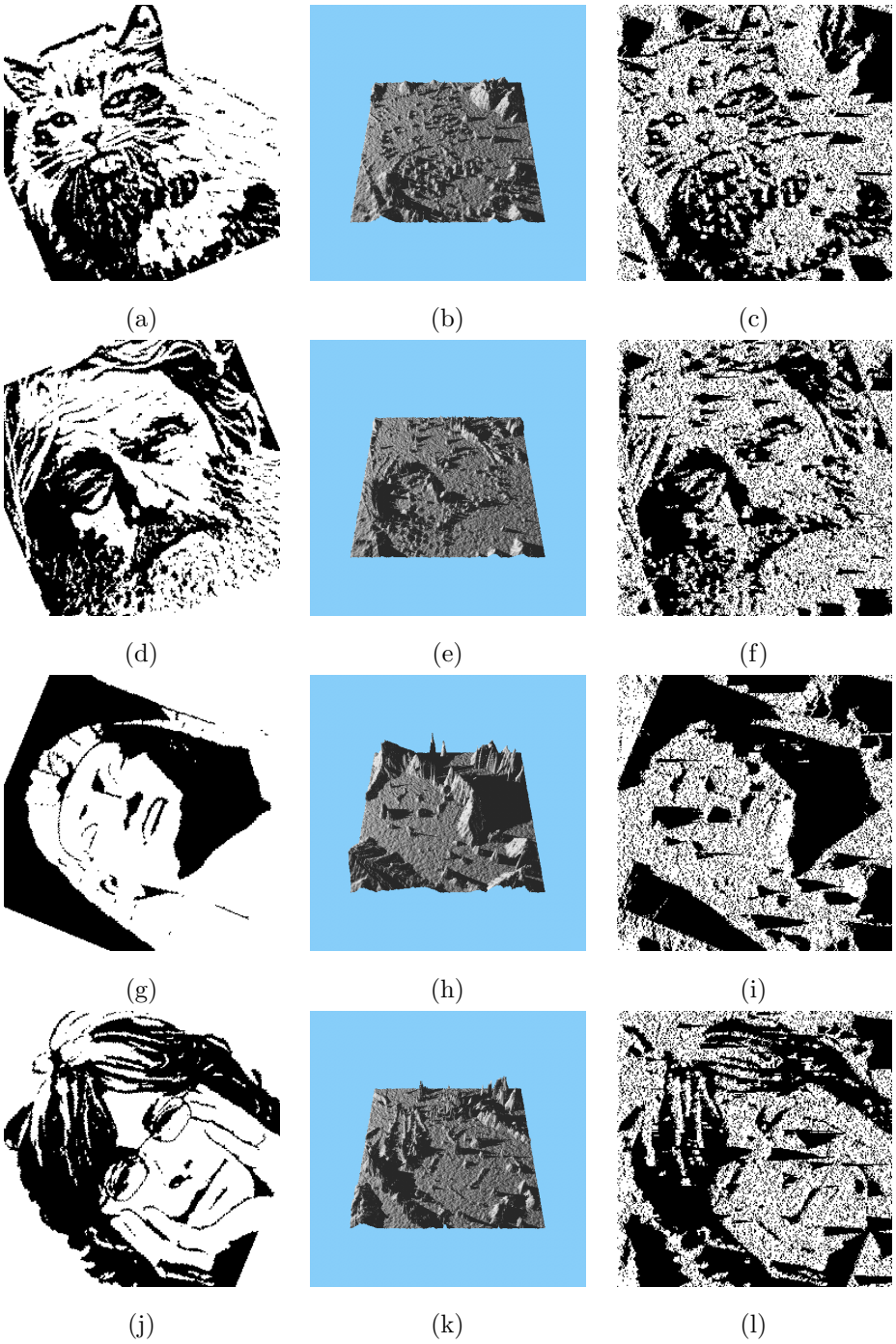


Figure 5.2: This figure shows the terrains and shadow regions resulting from the use of multiple target shadow images [54]. The order in each row is: target shadow, OpenGL render, and shadows cast by the final terrain. These results indicate that further work is required to make the technique more robust; too many terrain features are isolated and jagged, producing an unrealistic final terrain. Also, the non-shadow regions are too flat. An updated method would incorporate additional features with heights that do not produce shadows (from the given light direction).

Table 5.1: Summary of and inputs and parameters to the proposed terrain synthesis algorithm for pareidolia effects. Inputs consist of I, $T_b(n)$, and $C_{sec}(n)$, and parameters consist of μ_w, r, and θ.

Input/Parameter	Description
I	Binary *shadow image*
$T_b(n)$	Base terrain
$C_{sec}(n)$	Secondary feature generator costs
μ_w	Mean edge weight
r	Maximum edge weight deviation
θ	Light tilt angle

desirable. The reduction of $T_b(n)$'s shadow area is with respect to the lighting conditions used for primary and secondary terrain feature placement (see Section 5.3); if the shadows provided in I are large, then smaller changes to them are less noticeable. In this work, the shadow area was reduced by using numerous generator nodes. Densely packed generator nodes prevent the formation of tall, isolated features that can cast long, undesirable shadows. The base terrains for the "TEXT" and smiley face renders are presented in Section 5.4. Finally, the costs of secondary feature generators are provided via $C_{sec}(n)$. Secondary features complement the primary features and increase the realism of the resulting terrain; the use of $C_{sec}(n)$ is discussed in step 7 of Algorithm 4.

The three parameters are 1) mean edge weight μ_w, 2) maximum edge weight deviation r, and 3) light tilt angle θ (from the vertical). Specifically, μ_w is the mean weight of the edges in E and controls overall terrain steepness. The method requires $\mu_w > 0.0$ and terrain steepness increases with μ_w. The second parameter r is the maximum edge weight deviation in E and controls overall terrain roughness. The method also requires that $0.0 \leq r < \mu_w$ and as r approaches μ_w, terrain roughness increases.

The third parameter is the light tilt angle θ – the angle at which the light source points downward towards the origin. The tilt angle serves two purposes: it partially determines the light direction $\vec{v}$, and it limits the maximum height in the resulting synthetic terrains. Figure 5.3a visualizes θ in a 2D scene where a terrain feature of height h casts a shadow m units long at a tilt angle of θ degrees from the vertical. In the proposed method, m and θ are known and h is calculated according to Equation 5.1.

$$h = m/tan\theta \tag{5.1}$$

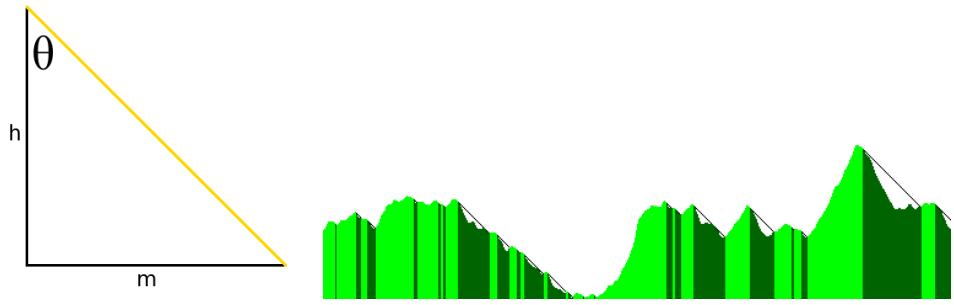


Figure 5.3: Two 2D scenes that visualize shadow casting. Image (a) shows a terrain feature of height h casting a shadow m units long at a tilt angle of θ degrees from the vertical. In the proposed method, m and θ are known and h is calculated according to Equation 5.1. Image (b) generalizes these principles to an arbitrary terrain; black lines visualize the shadow line and dark green terrain is in shadow. Furthermore, peaks can exist within shadows. The shadows cast by such features are completely contained within another shadow and can thus be ignored.

Figure 5.3b generalizes these principles to an arbitrary terrain; black lines visualize the shadow line and dark green terrain is in shadow. This figure also shows that local maxima can exist within shadows. The shadows cast by such local maxima are completely contained within another shadow and can thus be ignored.

The value of θ controls the maximum height in the resulting synthetic terrains. Specifically, as θ increases, h decreases (for fixed m). When synthesizing a scene, care should be taken in selecting θ. It should be provided such that the maximum height in the final height field is appropriate for the height-field resolution (i.e. if I contains longer shadows, then θ should be larger). Terrains whose maximum height is far greater than the width/height of the scene produces unrealistic results. The value for θ also depends on terrain type – a mountainous terrain will have a larger maximum height and should therefore use a smaller value for θ.

5.3 Algorithm

The synthesis of terrains whose shadows embed words and/or imagery must balance two competing goals: realism vs. adherence to desired shadows. The shadows should be recognizable, but the terrain should look realistic. Two synthetic terrains with pareidolia effects are presented that address these competing goals. The first render involves the word "TEXT" and the second render involves the image of a smiley face. An overview of the method is given in Algorithm 4.

The algorithm begins by initializing graph G using μ_w and r (step 1 of Algorithm 4). This process is identical to that outlined in Section 3.3. Next, step 2 calculates the light direction $\vec{v}$ such

Algorithm 4 Terrain synthesis algorithm summary for pareidolia effects

Input: Binary shadow image I, base terrain $T_b(n)$, mean edge weight μ_w, secondary feature generator costs $C_{sec}(n)$, maximum edge weight deviation r, and light tilt angle θ

Output: Height field

1. Initialize graph G, consisting of nodes N and edges E, with mean edge weight μ_w and maximum edge weight deviation r
2. Calculate light direction $\vec{v}$
3. Rotate I according to $\vec{v}$ and store as I_r
4. Determine anti-shadow regions
5. Determine shadow start and end locations
6. Determine primary terrain features using θ
7. Determine secondary terrain features using $C_{sec}(n)$
8. Initialize generator node locations F as all primary and secondary terrain feature node locations
9. Apply Dijkstra's algorithm with frontier consisting of nodes at locations F
10. Convert resulting cost field into the final height field

that the maximum shadow length $max\{m\}$ in I is minimized in the projected light direction. The formation of $\vec{v}$ is guided by the light tilt angle θ and the above minimization procedure. First, the y-value of the light source location $\vec{s}$ is determined according to Equation 5.1. The y-value replaces h, and m is chosen to be large enough so that the orthographic projection model is accurate; all light rays are assumed to be parallel. Next, $\vec{s}$ iterates through a set of predetermined locations about the scene at the calculated y-value. The light direction $\vec{v}$ is calculated in Equation 5.2. The light source $\vec{s}$ points downward towards the origin resulting in $-\vec{s}$.

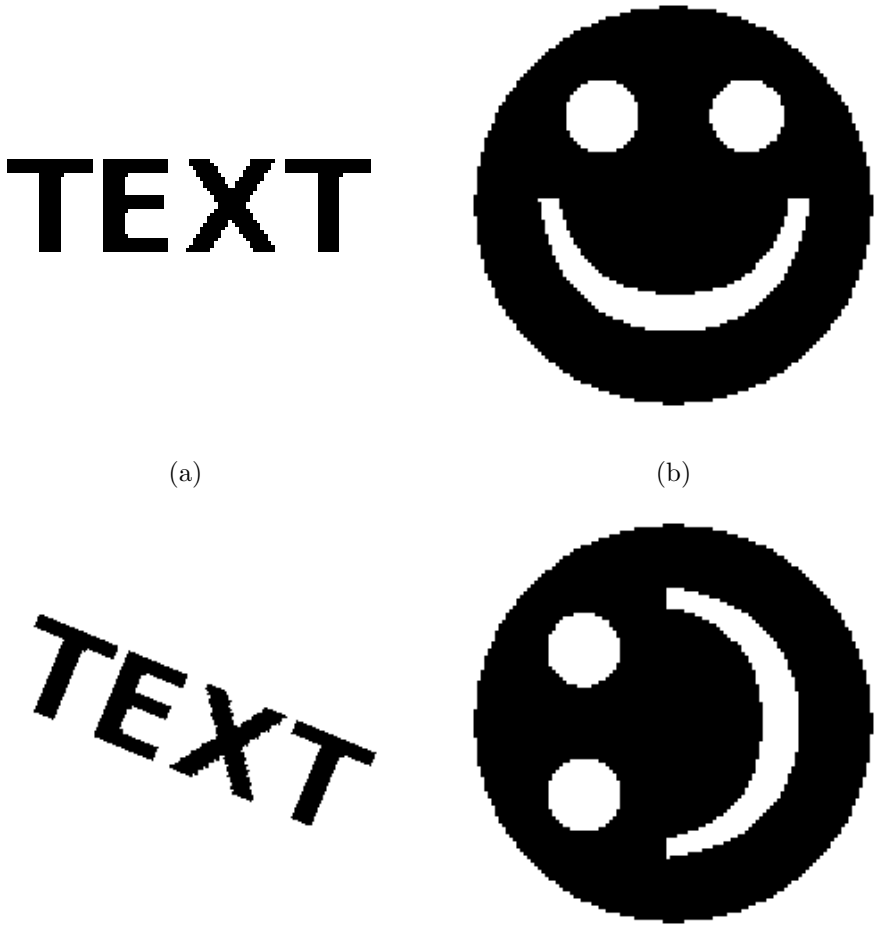
$$\vec{v} = -\vec{s}/\|\vec{s}\| \tag{5.2}$$

Minimizing $max\{m\}$ begins by iterating through the set of predetermined light source locations. At each light source location, I is scanned in the projected light direction and the maximum shadow length is recorded. In this context, the shadow length is the longest contiguous set of black pixels in I. The light direction $\vec{v}$ that results in the shortest maximum shadow length is selected as the final light direction. These steps are taken to help maintain an appropriate valued height field. According to Equation 5.1, as m increases, h also increases (for fixed θ). Thus, minimizing the maximum shadow length ensures that the maximum terrain height is also minimized.

Step 3 accounts for the light direction $\vec{v}$. There are two ways to account for this value: 1) process I in the projected light direction or 2) rotate I, forming I_r, such that processing it in scan-line order is identical to processing it in the projected light direction. The benefit of option 2 is that it prevents aliasing that results from naively using option 1. Thus, option 2 was used in this work. The shadow image I for the "TEXT" and smiley face renders is given in Figure 5.4(a–b) and their rotated versions are provided in Figure 5.4(c–d).

Step 4 of Algorithm 4 establishes *anti-shadow* regions in proximity of the shadow regions. *Anti-shadow* regions show where shadows should be absent and serve to contrast the shadow regions. Step 7 outlines how anti-shadow regions are enforced. Visualizations of anti-shadow regions are provided in Figure 5.5. Step 5 determines the start and end locations of each shadow segment. Each segment is found by scanning I_r in scan-line order and marking where a shadow begins (start nodes) and ends (end nodes); this is why I is rotated to form I_r. These locations help determine the cost of start nodes; the start nodes define the primary features which cast the shadows given by I_r. Figure 5.6 shows both the start and end node locations for the "TEXT" and smiley face renders.

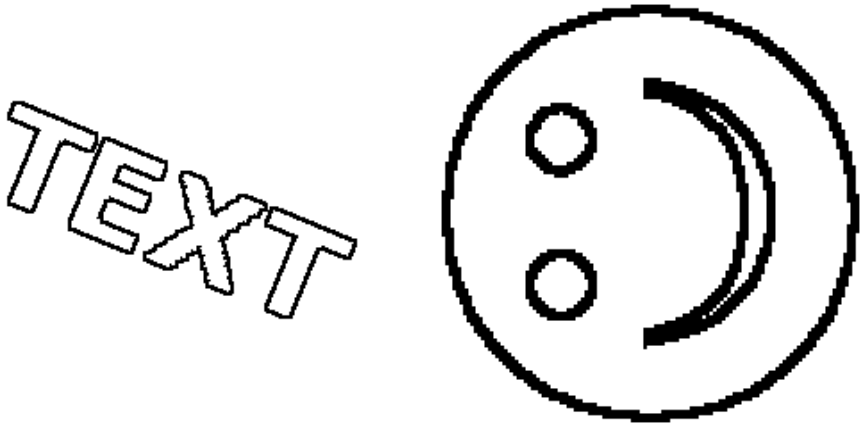
Step 6 creates the terrain features which cast the desired shadows stored in I_r. These features are known as *primary terrain features*. The goal of this step is to define the height of the features that cast the desired shadows.



(a) (b)

(c) (d)

Figure 5.4: Images (a,c) show the original (I) and rotated (I_r) shadow images used in the "TEXT" render, and images (b,d) show the original and rotated shadow images used in the smiley face render. The original images were rotated so that processing them in scan-line order is equivalent to processing their non-rotated version in the projected light direction.



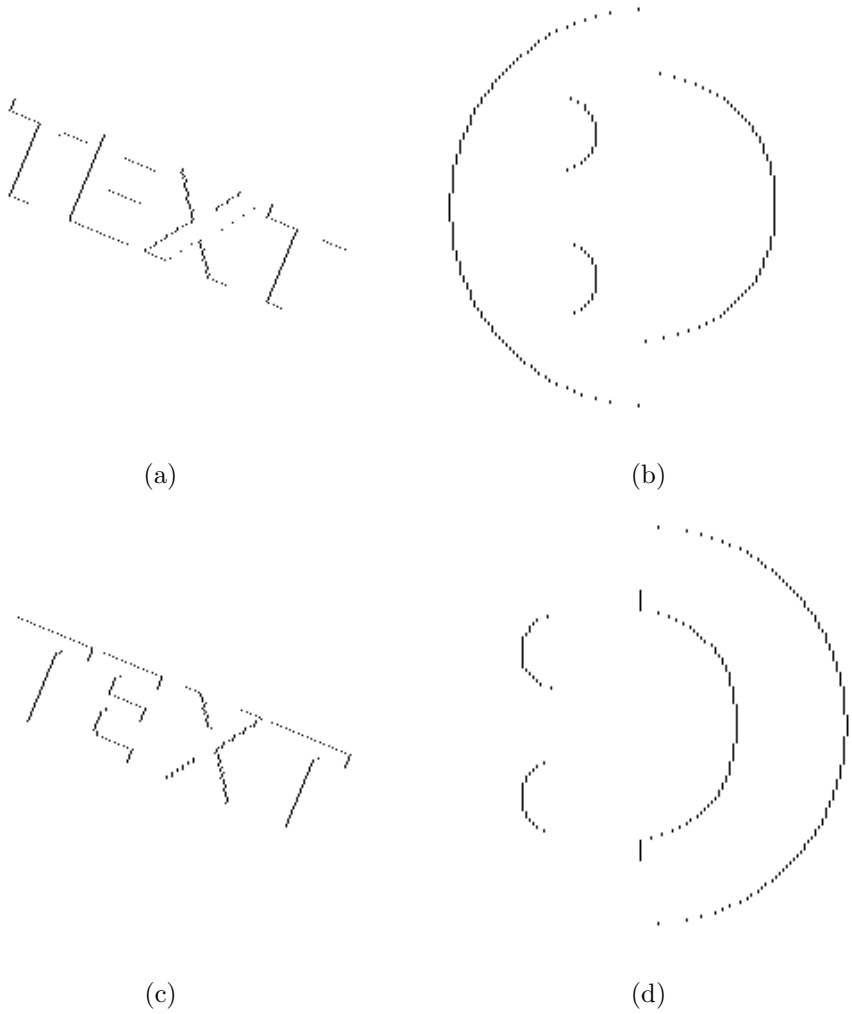
(a) (b)

Figure 5.5: These two images visualize the *anti-shadow* regions corresponding to Figure 5.4c and Figure 5.4d, respectively.

Though the presented examples differ in how this is accomplished, both methods define the height of the shadow casting features to be h (as per Equation 5.1) units more than the height of the terrain where the shadow terminates. This step is further detailed in Subsections 5.3.1 and 5.3.2.

Step 7 creates complementary terrain features called *secondary terrain features* for a more natural result. Secondary features include peaks, ridges, and hills. They can be placed manually or procedurally, and should populate vacant terrain regions. Their inclusion makes the scene more realistic and natural. These features avoid anti-shadow regions and minimize the amount of change to shadows cast by primary terrain features. They should avoid anti-shadow regions to keep the anti-shadow regions free of shadow, which helps define the shadows containing the embedded imagery. It can be argued that ridges parallel to the to the light direction may create only small shadows. However, their profiles in the perpendicular direction may cause unwanted shadows and secondly, controlling the exact location of ridges via Dijkstra's algorithm is not trivial because of the usage of random edge weights. Minimizing the change to shadows cast by primary shadows means that secondary feature heights should not be allowed to exceed the height of the yellow diagonal line in Figure 5.3a. Exceeding this height will lengthen the shadow – an undesirable result.

The presence of secondary features in anti-shadow regions is minimized by preventing either a single generator node (for peaks) or the start and end generator nodes (for ridges) from residing in those regions. Shadow modification is limited by ensuring that the initial costs of secondary features's generator nodes lie between a lower bound b_b and upper bound b_t, calculated on a per node basis. The base terrain serves as b_t, and b_b depends on the location of a given node. The calculation of b_b can depend on distance d to the closest shadow segment from the current location



(a) (b)

(c) (d)

Figure 5.6: Images (a) and (b) visualize the start node locations for the "TEXT" and smiley face renders, respectively. Images (c) and (d) visualize the end node locations for the "TEXT" and smiley face renders, respectively.

k, light tilt angle θ, base terrain cost c_b, or the minimum cost of a start node c_{min}. Pseudocode for the calculation of b_b is given in Algorithms 5, 6, and 7.

Algorithm 5 CalculateLowerBoundShadow(k, θ, c_b)

$p \leftarrow$ getShadowEnd(k) {find location where shadow ends}
$d \leftarrow$ length($k - p$)
return $c_b - d/tan\theta$

Algorithm 6 CalculateLowerBoundAntishadow(k, c_{min})

$s \leftarrow$ getAntishadowStart(k) {find location where anti-shadow starts}
$e \leftarrow$ getAntishadowEnd(k) {find location where anti-shadow ends}
$t_{start} \leftarrow$ cost(s) {base terrain cost at location s}
$t_{end} \leftarrow$ cost(e) {base terrain cost at location e}
$len_r \leftarrow$ length($k - s$)/length($e - s$)
if a lower bound does not exist at both s and e **then**
  return $t_{start} - (t_{start} - c_{min}) \cdot len_r$
else if a lower bound exists at both s and e **then**
  $b_{start} \leftarrow$ cost(s) {lower bound cost at location s}
  $b_{end} \leftarrow$ cost(e) {lower bound cost at location e}
  return $b_{start} - (b_{start} - b_{end}) \cdot len_r$
else if no lower bound at s and a lower bound at e **then**
  $b_{end} \leftarrow$ cost(e)
  return $t_{start} - (t_{start} - b_{end}) \cdot len_r$
else {lower bound at s and no lower bound at e}
  $b_{start} \leftarrow$ cost(s)
  return $b_{start} - (b_{start} - c_{min}) \cdot len_r$
end if

We begin by calculating b_b in shadow regions (Algorithm 5), and proceed by calculating b_b in the anti-shadow regions (Algorithm 6). The lower bound in unconstrained regions (i.e. not shadow and not anti-shadow) is calculated last (Algorithm 7). As previously mentioned, the lower bound in shadow regions prevents the height of secondary features from exceeding the height of the yellow diagonal line in Figure 5.3a. In anti-shadow regions, the goal is to linearly interpolate between the cost of the base terrain or lower bound at the start and end locations of the anti-shadow; if both costs are known at a given location, the lower bound cost is used. This provides a smooth lower bound transition between known costs. This approach is used in both the "TEXT" and smiley face renders, except the eyes/mouth in the smiley face render have a lower bound that exceeds

Algorithm 7 CalculateLowerBoundUnconstrained(k, θ, c_{min})

$s \leftarrow$ getNearestAntishadowB(k) {find location where anti-shadow starts before k}
$e \leftarrow$ getNearestAntishadowA(k) {find location where anti-shadow ends after k}
$t_{start} \leftarrow$ cost(s) {base terrain cost at location s}
$t_{end} \leftarrow$ cost(e) {base terrain cost at location e}
$len_r \leftarrow$ length$(k - s)$/length$(e - s)$
if s and e are valid locations **then**
  $b_{start} \leftarrow$ cost(s) {lower bound cost at location s}
  $b_{end} \leftarrow$ cost(e) {lower bound cost at location e}
  return $b_{start} - (b_{start} - b_{end}) \cdot len_r$
else if s is not valid and e is a valid location **then**
  return $max(c_{min}, (length(e - s)/tan\theta) \cdot len_r - t_{end})$
else if s is valid and e is not a valid location **then**
  $b_{start} \leftarrow$ cost(s)
  return $b_{start} - (b_{start} - c_{min}) \cdot len_r$
else {both s and e are invalid locations}
  return c_{min}
end if

$c_b - d/tan\theta$ by a small amount (0.5 units) to bring those regions out of shadow. In this case, d is defined to be the distance from the current node to the end of the eye/mouth.

Finally, if we are in an unconstrained region, b_b approaches c_{min} according to $c_b - d/tan\theta$ at locations preceding an anti-shadow region; this calculation avoids the placement of features whose shadows will extend into the anti-shadow region. If the current location follows an anti-shadow region, we linearly interpolate between the lower bound cost at the start of the unconstrained region and c_{min}. The manner in which the lower bound is set in unconstrained regions prevents tall features from residing beside short features – such discontinuities are unnatural. This provides a smooth transition to the minimum cost c_{min}. The process is detailed in Algorithm 7.

For consistency with the primary features, b_b is taken as the maximum of the calculated lower bound and c_{min}. The visualizations of b_b for both the "TEXT" and smiley face renders are given in Figure 5.7. The exact cost of a secondary feature's generator node is then calculated as $C_{sec}(n)$, or fBm in this case, scaled between $[b_b, b_t]$ at that node. Scaling fBm consists of normalizing fBm into the interval $[0, 1]$ and using the normalized values to index into the interval $[b_b, b_t]$. If the normalized fBm value is zero, then the cost of the generator node is b_b; if the normalized fBm value is one, then the cost of the generator node is b_t.

(a) (b)

Figure 5.7: Visualization of the lower bound b_b in the (a) "TEXT" and (b) smiley face renders.

Step 8 of Algorithm 4 takes the generator node locations from both the primary and secondary terrain features and combines them to form the final set of generator node locations F. In Step 9, Dijkstra's algorithm is applied with the frontier consisting of the nodes at locations in F. When Dijkstra's algorithm is applied for these renders, it is not halted as in Algorithm 3. Dijkstra's algorithm is not halted because a single parallel pass with all nodes is used to calculate the resulting cost field. This single pass is identical to the creation of the approximate terrain in step 5 of Algorithm 3. However, blending is not performed in Algorithm 4 because it modifies the height field where least-cost paths from differing generators are nearly identical in cost. The blending process can change the height difference between shadow start and end points. Changing this difference can undesirably shorten or lengthen shadows. Furthermore, constant profiles were used to maintain the generator costs of the primary terrain features (see Section 5.4). These decisions were made to minimize the number of factors that could change the desired shadows.

Finally, step 10 converts the cost field resulting from Dijkstra's algorithm and the base terrain $T_b(n)$ into the final height field. Specifically, each node's cost c is converted to height h_c by subtracting c from the maximum height $max\{h\}$ of any generator node in the graph. In our implementation, each node stores a cost and a height; $max\{h\}$ has a cost of zero. The conversion from cost to height is given in Equation 5.3.

$$h_c = max\{h\} - c \tag{5.3}$$

However, the maximum of h_c and the height at the corresponding location in $T_b(n)$ is taken as the final height. This process is repeated for every node in the graph, resulting in the final height field.

5.3.1 TEXT

The primary terrain features for the "TEXT" render are created in a straightforward manner. The goal is to assign a cost to each start node in Figure 5.6a so that it casts a shadow that ends at its corresponding end node in Figure 5.6c. Figure 5.3a visualizes the process: if we can calculate the distance m between the start and end node for each shadow segment, we can calculate the cost for each start node. The cost c_o of a start node is given in Equation 5.4 and is a function of the cost c_b of the base terrain $T_b(n)$ at the end node location, distance m, and light tilt angle θ.

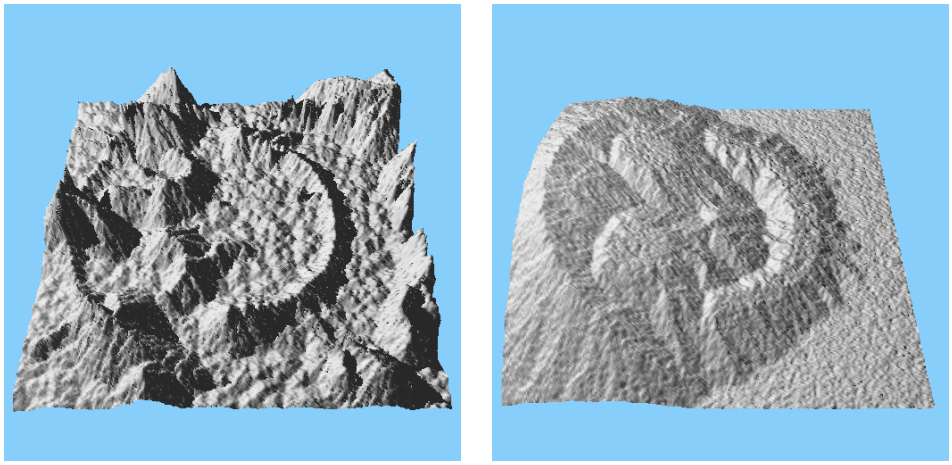
$$c_o = c_b - m/tan\theta \tag{5.4}$$

Equation 5.4 is applied to every start node. The minimum cost of a start node is stored so that secondary terrain features have a consistent altitude (step 7 of Algorithm 4). The results of this process can be seen in Section 5.4.

5.3.2 Smiley Face

The creation of primary terrain features for the smiley face render requires a different methodology compared to Subsection 5.3.1. This is because the smiley face render contains non-shadow areas (the eyes and mouth) surrounded by shadowed areas (the head). A differing approach is motivated by the fact that the application of Subsection 5.3.1's methodology to the smiley face render results in unnatural, jagged features, as seen in Figure 5.8a. In the case of the smiley face, the interior non-shadow areas can be interpreted as breaking up a longer shadow segment beginning at the top of the head and ending at the bottom of the head. In the previous approach (Subsection 5.3.1), local height maxima are defined at the start of every shadow segment. Since interior non-shadow areas result in more shadow segments, there is an increase in the number of local height maxima. This increase results in the aforementioned unnatural and jagged terrain features. As a result, an iterative approach was developed to avoid these drawbacks. Figure 5.8a shows the smiley face render synthesized using the "TEXT" method. The majority of this terrain is dominated by steep, jagged features which clash with the base terrain. However, using an iterative approach avoids such features and produces the terrain in Figure 5.8b. This terrain contains shallower features that closely resemble the base terrain, resulting in a more realistic result.

The aforementioned iterative approach involves the use of a homotopic tree [80]. Each depth/level in the tree stores region IDs. Regions in I_r are stored at different depths in the tree. The depth that a given region appears at is a function of its surroundings. The parent of a given level (with the exception of the root level) consists of the regions that surround the current level's regions. If a given level contains multiple regions, each region has a different ID for easy identification. Such an arrangement is shown in Figure 5.9. This organizational hierarchy facilitates the iterative processing required to render the smiley face.



(a) (b)

Figure 5.8: (a) OpenGL render of the smiley face terrain synthesized via the methodology for "TEXT" and (b) OpenGL render of the smiley face terrain synthesized via an iterative approach.

The iterative algorithm begins by assigning costs to the start nodes associated with Figure 5.9c. The goal is to synthesize terrain features whose shadows cast a shape identical to that in Figure 5.9c. The cost of these nodes are calculated in the exact same manner as the generators of primary terrain features for the "TEXT" render (Equation 5.4). At this point, we have a single ridge that casts a circular shaped shadow corresponding to the smiley face's head. In order to synthesize the non-shadowed areas corresponding to the eyes and mouth (Figure 5.9d), a different approach is taken. Random locations in the eye and mouth regions have their cost set so that they are just above the head's shadow line. An example shadow line is the gold, diagonal line in Figure 5.3a. As each level in the tree is processed, the hierarchical organization makes it easy to locate the features from the previous level. This is necessary, for example, for determining the cost of the shadow line. The iterative approach produces the smiley face renders seen in Section 5.4 and avoids the synthesis of unnatural, jagged features that result from applying the process outlined in Subsection 5.3.1. However, this iterative approach also introduces other unnatural features, which are discussed in Section 5.4.

5.4 Results

Results of Algorithm 4 are now presented. Again, the synthesis of terrains whose shadows embed words and/or imagery must balance two competing goals: realism vs. adherence to desired shadows. The shadows should be recognizable, but the terrain should look realistic. The results show that the proposed method was able to balance these competing goals, though the smiley face render

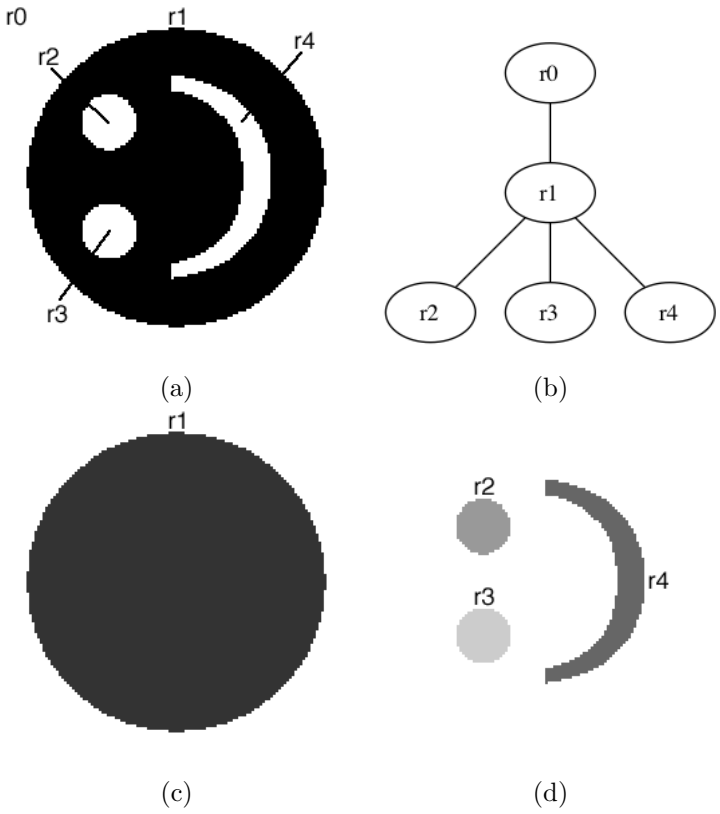
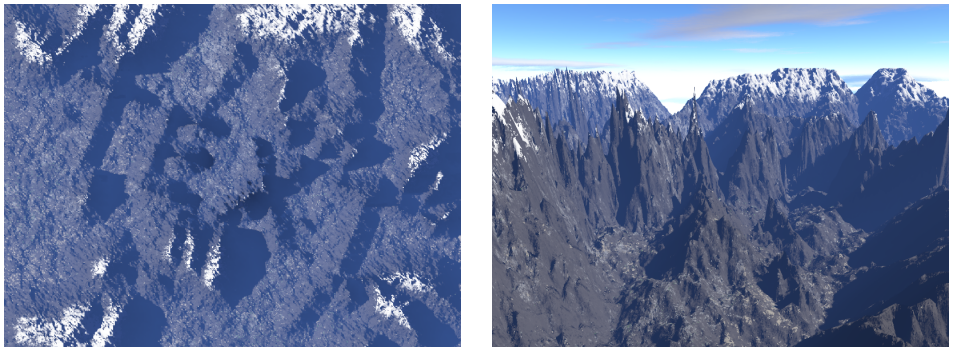


Figure 5.9: Visualization of the homotopic tree for the smiley face render. Image (a) shows an annotated shadow image. The regions are background $r0$, head $r1$, right eye $r2$, left eye $r3$, and mouth $r4$. Image (b) shows the corresponding homotopic tree. Image (c) shows the smiley face's head and image (d) shows the face (the eyes and mouth). Each region has a different label (grayscale value) that facilitates region identification. The labels in each image are not part of the input and are merely visual aids for image regions.



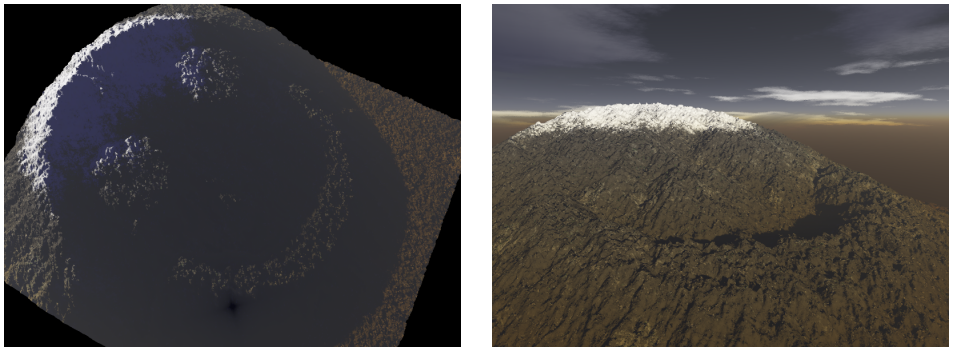
(a) (b)

Figure 5.10: Terrain whose shadows spell the word "TEXT" for a pareidolia effect. Image (a) gives an overhead perspective to emphasize "TEXT" and image (b) shows the same scene with different camera and light placement, making the embedded word difficult to see.

leaves room for improvement. Figure 5.10 shows the "TEXT" render and Figure 5.11 shows the smiley face render.

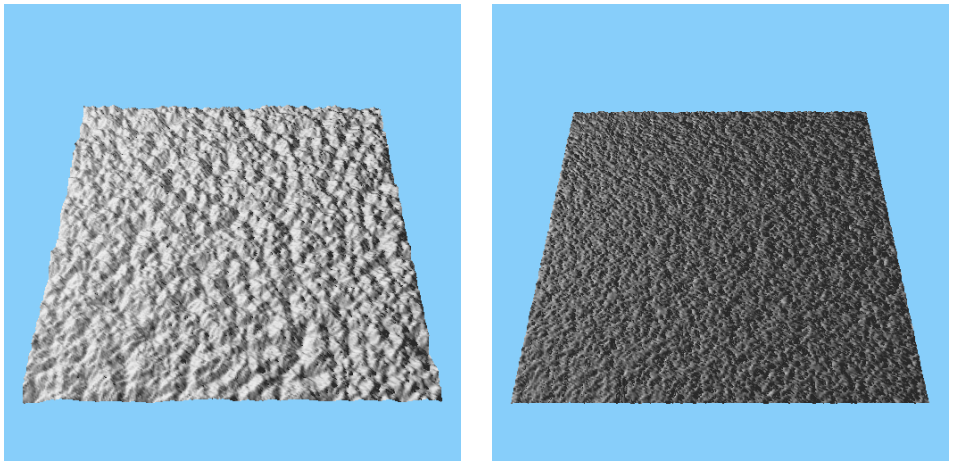
The base terrains for each render are given in Figure 5.12. The "TEXT" base terrain was generated using 1311 randomly (uniformly distributed over the height field or cost range) generated peaks with a randomly assigned cost in $[0.0, 20.0]$ and 200 randomly generated ridges with a cost equal to 25% of the original cost. Recall that ridges are formed by running Dijkstra's algorithm from the peak generator nodes and then selecting a random node and tracing back on the least-cost path to the originating node. Original cost refers to the cost assigned by this pass of Dijkstra's algorithm. The underlying graph for this base terrain had $\mu_w = 2.0$, $r = 1.5$, though the edges were scaled by 50% of their original weight to create a smoother base terrain. The smiley face base terrain was generated using 31457 randomly generated peaks with a randomly assigned cost in $[0.0, 20.0]$ and 4800 randomly generated ridges with a cost equal to 25% of the original cost. The underlying graph for this base terrain had $\mu_w = 2.0$, $r = 1.5$, though the edges were also scaled by 50% of their original weight to create a smoother base terrain. The smiley face base terrain is smoother than the "TEXT" base terrain because the topographies of the primary and secondary features in these renders are different. The "TEXT" terrain contains steeper and rougher features, whereas the smiley face render is smoother in appearance.

The final renders (Figures 5.10 and 5.11) used different parameters than the base terrains. The number and location of start and end nodes for each render is visualized in Figure 5.6. The "TEXT" render had $\theta = 45.0$, $\mu_w = 2.0$, and $r = 1.5$.



(a) (b)

Figure 5.11: Terrain whose shadows create a smiley face for a pareidolia effect. Image (a) gives an overhead perspective to emphasize the smiley face and image (b) shows the same scene with different camera and light placement, making the embedded image more difficult to see.



(a) (b)

Figure 5.12: Base terrains for the "TEXT" and smiley face renders. Image (a) shows the base terrain for "TEXT" and image (b) shows the smiley face's base terrain.

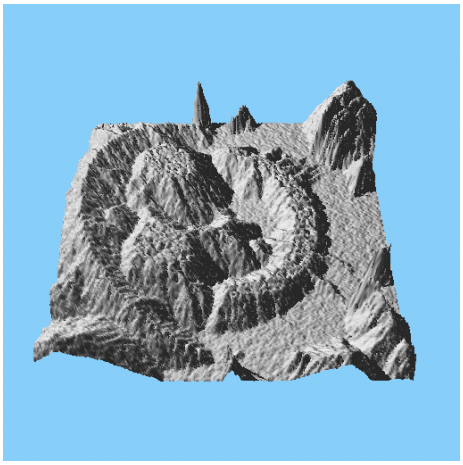


Figure 5.13: Smiley face render that incorporates steeper terrain features, as well as complementary terrain features that appear in the surrounding area. This terrain was synthesized using the proposed iterative method, and the steeper terrain features were accomplished using $\mu_w = 2.0$ and $r = 1.5$.

The light location was $\vec{s} = (-295.64, 320.0, 122.46)$ which resulted in $\vec{v} = (0.65, -0.71, -0.27)$. Its secondary features consisted of 10 peaks and 75 ridges, and fBm (H=3.0, Lacunarity=2.0, and Octaves=2) normalized within $[0.0, 1.0]$ was used to determine the exact cost in $[b_b, b_t]$ at each node.

The smiley face render had $\theta = 75.0$, $\mu_w = 1.0$, and $r = 0.75$. The light location was $\vec{s} = (0.0, 320.0, -320.00)$ which resulted in $\vec{v} = (0.0, -0.71, 0.71)$. Its secondary features consisted of 10 peaks and 75 ridges. Also, 16384 nodes were added in the eyes and mouth to make these regions non-shadowed. Finally, fBm (H=3.0, Lacunarity=2.0, and Octaves=2) normalized within $[0.0, 1.0]$ was used to determine the exact cost in $[b_b, b_t]$ at each node. Unfortunately, the smiley face terrain in Figure 5.11 contains some unnatural ridges and is overly smooth. Recall that the smiley face method randomly places generator nodes in the eyes and mouth regions in order to bring these locations out of shadow. This is effective for establishing the interior non-shadow regions but causes the eye and mouth regions to be relatively smooth and flat. For consistency, the rest of the terrain is synthesized in a similar fashion. Though terrain heterogeneity is desirable, too much diversity can look unnatural (see Figure 5.13).

Per-node profiles were required in order to maintain the costs of primary terrain generators after the application of Dijkstra's algorithm. If the difference in cost between adjacent nodes exceeds the edge weight between them, Dijkstra's algorithm will redefine the larger cost. Therefore, a constant profile was used for each generator node to avoid this scenario. A constant profile means that the profile's slope is constant. Specifically, a scalar value storing the maximum steepness between the current generator node and a node from the same feature with a larger cost was used as the

constant profile. All edges on a given path J have their weight multiplied by this scalar value – all edge weights are multiplied by the scalar value associated with the generator node that J originates at. This step ensures that the calculated costs are maintained after running Dijkstra's algorithm.

Both renders were synthesized on a height field with a resolution of 256×256. Algorithm 4 was implemented in C++, and was executed in Mandriva Linux 2008.1 running on a Pentium 4 2.80GHz processor with 1GB RAM. The "TEXT" render took 10 seconds to synthesize, and the smiley face render took 12 seconds to synthesize.

Algorithm 4 produces realistic terrains that embed words and/or imagery in their shadows. The method is easy to use and can handle varying types of input shadow images. The inclusion of μ_w and r facilitate terrain diversity and the limited restrictions on secondary terrain features allow for the synthesis of realistic terrains. However, it is still a proof of concept. Simple shadows, that do not contain interior non-shadowed areas, are handled according to the process outlined in Subsection 5.3.1. More complex shadows, that contain interior non-shadowed areas, are handled according to the process outlined in Subsection 5.3.2. Unfortunately, this iterative approach introduces a tradeoff: it can avoid the rough, jagged features seen in Figure 5.13, but introduces artificial looking plateaus in non-shadow regions such as the eyes and mouth (see Figure 5.8b).

Chapter 6

Conclusions and Future Work

6.1 Conclusions

Terrain synthesis involves the creation of a model that approximates a terrain resulting from erosional and geological processes. It is an important problem because synthetic terrains are widely used in film and video games. Because terrains found in nature result from complex erosional and geological processes, synthesizing terrains is a difficult task. Existing terrain synthesis methods are either difficult to control, produce homogeneous results, are time consuming, or require an existing height field that will serve as an exemplar for the resulting terrain.

This thesis presents a procedural algorithm based on path planning for terrain synthesis. Terrains are synthesized by calculating the least-cost path cost for every non-generator node from multiple generator nodes. The proposed algorithm provides control over feature placement, terrain profile, mean edge weight, maximum edge weight deviation, and input stroke costs. Terrain profiles allow the user to easily control the terrain style. The mean edge weight controls the steepness of the terrain and the maximum edge weight deviation controls terrain roughness. Using existing terrain synthesis methodologies for the calculation of input stroke costs provides an easy way to incorporate realistic height fields – the input stroke costs define the initial topographies of the features they define. The level of control provided in this work creates different terrain styles that realistically model the various topographies found in nature. This control also facilitates the synthesis of terrains whose features embed imagery in their shadows or dominant features. The main contributions of this thesis are the introduction of path planning for terrain synthesis, control over feature placement and terrain style, the inclusion of imagery and text in terrain features and shadows, and the ability to synthesize a great range of different terrain structures.

Chapter 3 details the proposed algorithm. Specifically, Algorithm 3 begins by initializing a graph. The edge weights in this graph are calculated according to a user provided mean edge weight and maximum edge weight deviation. With the graph initialized, a set of generator nodes is identified and the maximum permitted cost in the graph is established. Next, a scaling function is created for each input profile; a scaling function replaces the slopes and heights computed from edge weights with user-sketched slopes and heights. With the generators and scaling functions

defined, an approximate cost field is created. This cost field is used in the last step of Algorithm 3; the last step terminates the search of Dijkstra's algorithm when the individual terrain features are synthesized and blended together. As previously mentioned, terrains are synthesized by calculating the least-cost path cost for every non-generator node from multiple generator nodes; a stroke is a contiguous set of generator nodes that collectively define the height and location of a single terrain feature. The algorithm proposed in Chapter 3 creates a realistic height field subject to the input constraints on terrain feature location and style.

The 26 resulting terrains presented in Chapter 4 show that it is easy produce a diverse set of realistic and artificial terrains with our method. An artificial terrain's dominant features depict recognizable images and/or words. The method is particularly adept at producing mountain ranges, hills, craters, cinder cone volcanoes, and faults. However, bodies of water, canyons, tower karsts, and river terraces are not as easy to synthesize. These findings are qualitatively supported by comparing each result to a reference photo. The reference photo is included as an exemplar of the feature, and the comparison is meant to show that our synthesized results are also sensible exemplars of the desired terrains. The chapter also shows that using input textures for terrain feature placement and/or edge weights can produce realistic terrains. Our findings suggest that structured high-frequency, high-contrast input textures are desirable.

In addition to comparing our results to reference photos, we also evaluate our method by comparing it to existing methods. The evaluation shows that terrain feature placement with our method makes it easier than, or as easy as, that of existing methods. Our algorithm does not produce homogeneous terrains like that of fBm because it incorporates random graph edge weights, permits the inclusion of multiple topography profiles, and allows precise control over placement of terrain features and their heights. These properties all allow the artist to create highly heterogeneous terrains. Extended features, which are problematic for the RMF terrain model, are easy to synthesize in our method as well. Additionally, designing terrains in the proposed system is efficient; this system does not heavily rely on an existing height field that serves as the foundation for new terrain, and the complex simulation programming involved in physically-based models is not required.

Chapter 4 ends with a quantitative measurement of our algorithm's performance. It is shown that per-stroke cost, where cost refers to the area that Dijkstra's algorithm searches, is approximately constant, assuming the density of features is constant at larger height field resolutions. Furthermore, at a resolution of 512×512, the terrains presented in Chapter 4 took 28.08 seconds on average to render, though the minimum and maximum render times were 7 seconds and 135 seconds, respectively. Figure 6.1 shows a histogram of the terrain synthesis times for these results.

Chapter 5 introduces terrains that create a pareidolia effect, but the method is currently a proof of concept. These are terrains whose dominant features embed imagery in their shadows. Embedding imagery in the shadows of terrains means that the usage of fBm, or any existing

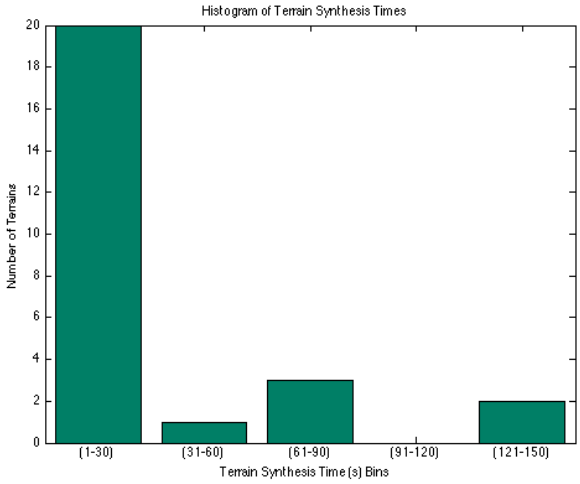


Figure 6.1: Histogram of the terrain synthesis times for all presented results.

methodology, for the cost of generator nodes is no longer applicable because a great deal of control over the heights of certain generator nodes is required. A shadow with a length of m units requires the shadow casting feature to be $m/tan\theta$ units taller than the end of the shadow, where θ is the light tilt angle from the vertical. However, complementary features known as secondary features are not synthesized according to this process because they do not cast recognizable shadows. The method then ends with the synthesis of the final height field, which is created in a manner similar to that of the approximate terrain in Algorithm 3.

Two results are shown in Chapter 5. One terrain contains the words "TEXT" in its shadows, and the other contains the image of a smiley face. The "TEXT" render is able to produce a realistic result while accurately embedding the word in its shadows. Unfortunately, the smiley face render contains some unnatural ridges and is overly smooth. It is smooth because the proposed iterative method was developed to avoid the jagged, rough topography that is characteristic of the "TEXT" terrain. Recall that the smiley face method randomly places generator nodes in the eyes and mouth regions in order to bring these locations out of shadow. This is effective for establishing the interior non-shadow regions but causes the eye and mouth regions to be relatively smooth and flat. For consistency, the rest of the terrain is synthesized in a similar fashion. Though terrain heterogeneity is desirable, too much diversity can look unnatural. The iterative smiley face approach introduces a tradeoff; it can avoid the rough, jagged features seen in the "TEXT" result, but introduces artificial looking plateaus in non-shadow regions.

6.2 Future Work

One possibility for future work would be to incorporate bodies of water. Currently, bodies of water can be added to any of the aforementioned results by simply selecting a water level height and rendering water at that height. However, this is a post-process, meaning bodies of water are not explicitly modelled. A possible solution for bodies of water is similar to the process for synthesizing craters. The user could manually draw a coastline, and when Dijkstra's algorithm searches this region, a profile corresponding to the body of water would be used, regardless of a node's profile ID. Furthermore, Dijkstra's algorithm would not terminate near sea level in order to synthesize features below sea level.

Profile resolution is another avenue to explore in future work. When the profile is read, it is scaled within a cost of zero and the sea level cost. Unfortunately, many costs are undefined in the scaled profile if its y-axis resolution is significantly lower than the sea level cost. This situation makes it difficult to determine the slope at the undefined costs. To remedy the situation, the slope is calculated by linearly interpolating between the neighbouring profile locations with a defined slope. The combination of missing data and linear interpolation introduces artifacts into the scaled profile (which calculates the scaling term $w_s(c)$). These artifacts can be desirable because they can produce topographical features such as rock strata. This means that the user does not have to explicitly model such features by hand.

The blending function used in this work could also be investigated. The results shown in Chapter 4 use a constant blending bias value b over the entire height field. To quickly summarize, as b increases, the final terrain height approaches the maximum height at each location in the height field. When $b = 0$, the average height at each height field location is taken. However, b does not have to be constant. Spatially varying b could produce smoother, more blended terrains in some regions, and rougher, less blended terrains in others. The overall goal of spatially varying b would be to increase the heterogeneity of the synthesized terrains. Another aspect of the blending function that could be investigated is determining which features contribute to the final blend. When synthesizing the crater and lunar landscape results in Subsection 4.2.5, height values from non-crater features do not contribute to the final height values in the crater interiors. The selective inclusion of heights produces the impact effect seen in craters. However, it could also be used to isolate local minima, such as bodies of water. Further analysis of the application of this effect could allow for the explicit modelling of valleys and bowl-shaped depressions.

Three final areas to explore could be alternative edge weight calculation methods, further refinement of the resulting synthetic terrain, and moving the pareidolia terrains in Chapter 5 from a proof of concept framework to a more robust and realistic framework. Section 4.3 explored the usage of textures for edge weights. Our early findings indicate that relatively stochastic and high in

contrast textures produce realistic terrains, but exploring other texture properties could result in additional realistic or artificial terrains. Here, artificial refers to martian and/or fictional topographies. It would also be interesting to see how easy it is to incorporate more features into the final synthetic terrain. Questions such as "Will the blending function realistically merge new features?", and "Is there utility in post-processing the terrain?" would be addressed. This sort of iterative refinement could be beneficial in the stages of terrain prototyping because the addition of a single feature requires a relatively small overhead, and it would allow for continuous visual feedback. In Chapter 5, we mentioned that pareidolia terrains are currently a proof of concept. To establish our method as a robust technique, it needs to realistically capture more shadow regions with more varied shape. Furthermore, the non-shadow regions are too flat. Future work would like to address these issues to not only increase the realism of the resulting synthetic terrains, but to capture a wider variety of shadow regions. Such additions to the method could be used to create realistic terrains that subtly incorporate hidden images for applications such as advertising.

References

[1] Luke Addison. Bicaz canyon 1. Yahoo Flickr, June 2008. `http://www.flickr.com/photos/1uk3/2581389806/`; Access Date: February 17, 2009.

[2] M. Akeo, H. Hashimoto, T. Kobayashi, and T. Shibusawa. Computer graphics system for reproducing three-dimensional shape from idea sketch. *Comput. Graph. Forum*, 13(3):477–488, 1994.

[3] Nancy M. Amato, O. B Bayazit, Lucia K. Dale, Christopher Jones, and Daniel Vallejo. Choosing good distance metrics and local planners for probabilistic roadmap methods. Technical report, Texas A & M University, 1998.

[4] Francisco Antunes. Leaves. Yahoo Flickr, December 2007. `http://www.flickr.com/photos/franciscoantunes/2110518922/`; Access Date: February 17, 2009.

[5] Kenichi Arakawa and Eric Krotkov. Fractal modeling of natural terrain: Analysis and surface reconstruction with range data. *Graph. Models Image Process.*, 58(5):413–436, 1996.

[6] Walter G. Arce. Barringer meteor crater. Yahoo Flickr, April 2007. `http://www.flickr.com/photos/walterarce/467202606/`; Access Date: 2008; No Longer Available.

[7] Thomas Baudel. A mark-based interaction paradigm for free-hand drawing. In *UIST '94: Proceedings of the 7th annual ACM symposium on User interface software and technology*, pages 185–192, 1994.

[8] Bedrich Benes and Rafael Forsbach. Layered data representation for visual simulation of terrain erosion. In *SCCG '01: Proceedings of the 17th Spring conference on Computer graphics*, page 80, 2001.

[9] Bedrich Benes and Rafael Forsbach. Visual simulation of hydraulic erosion. In *WSCG*, pages 79–94, 2002.

[10] Bedrich Benes, Vaclav Tesinsky, Jan Hornys, and Sanjiv K. Bhatia. Hydraulic erosion. *Computer Animation and Virtual Worlds*, 17(2):99–108, 2006.

[11] John Brosz, Faramarz F. Samavati, and Mario Costa Sousa. Terrain synthesis by-example. *Advances in Computer Graphics and Computer Vision: International Conferences VISAPP and GRAPP 2006*, 4(2):58–77, 2007.

[12] Richard W. Bukowski and Carlo H. Séquin. Object associations: A simple and practical approach to virtual 3D manipulation. In *SI3D '95: Proceedings of the 1995 symposium on Interactive 3D graphics*, pages 131–214, 1995.

[13] Danny Z. Chen. Developing algorithms and software for geometric path planning problems. *ACM Comput. Surv.*, page 18, 1996.

[14] N. Chiba, K. Muraoka, and K. Fujita. An erosion model based on velocity fields for the visual simulation of mountain scenery. *The Journal of Visualization and Computer Animation*, 9(4):185–194, 1998.

[15] Howie Choset. Coverage for robotics - a survey of recent results. *Annals of Mathematics and Artificial Intelligence*, 31(1-4):113–126, 2001.

[16] Denis Collette. These birches are making water in my wild river...!!! :))). Yahoo Flickr, April 2008. `http://www.flickr.com/photos/deniscollette/2449484303/`; Access Date: February 17, 2009.

[17] Storm Crypt. Southern California mountain range. Yahoo Flickr, July 2008. `http://www.flickr.com/photos/storm-crypt/2715992898/`; Access Date: February 17, 2009.

[18] Carsten Dachsbacher. *Interactive Terrain Rendering - Towards Realism with Procedural Models and Graphics Hardware*. PhD thesis, University of Erlangen-Nuremberg, 2006.

[19] Carsten Dachsbacher, Martin Meyer, and Marc Stamminger. Heightfield synthesis by non-parametric sampling. In *Vision, Modeling and Visualization*, pages 297–302, 2005.

[20] Ian Lane Davis. Warp speed: Path planning for star trek: Armada. In *AAAI Spring Symposium Technical Report (2000 AAAI Spring Symposium)*, pages 18—21, 2000.

[21] J. Davis. Mosaics of scenes with moving objects. In *CVPR '98: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 354–360, 1998.

[22] R. Neumann de Carvalho, H.A. Vidal, P. Vieira, and M.I. Ribeiro. Complete coverage path planning and guidance for cleaning robots. In *Industrial Electronics, 1997. ISIE '97., Proceedings of the IEEE International Symposium*, volume 2, pages 677–682, 1997.

[23] Michael F. Deering. Holosketch: A virtual reality sketching/animation tool. *ACM Trans. Comput.-Hum. Interact.*, 2(3):220–238, 1995.

[24] Michael F. Deering. The holosketch VR sketching system. *Commun. ACM*, 39(5):54–61, 1996.

[25] Lucy Dell. Rainbow beads. Yahoo Flickr, September 2006. `http://www.flickr.com/photos/98662646@N00/236184995/`; Access Date: February 17, 2009.

[26] Patrick Dirden. Rolling hills. Yahoo Flickr, October 2006. `http://www.flickr.com/photos/sp8254/282061496/`; Access Date: February 17, 2009.

[27] Steven C. Dollins. *Modeling for the Plausible Emulation of Large Worlds*. PhD thesis, Brown University, 2002.

[28] Dominic. Hay bales. Yahoo Flickr, September 2008. `http://www.flickr.com/photos/dominart/2895914399/`; Access Date: February 17, 2009.

[29] David S. Ebert, F. Kenton Musgrave, Darwyn Peachey, Ken Perlin, and Steven Worley. *Texturing and Modeling: A Procedural Approach*. Morgan Kaufmann, third edition, 2002.

[30] Alexei A. Efros and Thomas K. Leung. Texture synthesis by non-parametric sampling. In *ICCV '99: Proceedings of the International Conference on Computer Vision-Volume 2*, page 1033, 1999.

[31] Lynn Eggli, Ching-Yao Hsu, Beat D Bruderlin, and Gershon Elber. Inferring 3D models from freehand sketches and constraints. *Computer-Aided Design*, 29(2):101–112, 1997.

[32] Michael Foley. Himalayan hills, Nepal. Yahoo Flickr, January 2007. `http://www.flickr.com/photos/michaelfoleyphotography/340954037/`; Access Date: February 17, 2009.

[33] Alain Fournier, Don Fussell, and Loren Carpenter. Computer rendering of stochastic models. *Commun. ACM*, 25(6):371–384, 1982.

[34] Chris Fry. River rocks at the Japanese garden. Yahoo Flickr, November 2006. http://www.flickr.com/photos/chrisjfry/309586187/; Access Date: February 17, 2009.

[35] David Galvan. volcanoes.jpg. Yahoo Flickr, October 2005. http://www.flickr.com/photos/dgalvan/57921165/; Access Date: February 17, 2009.

[36] L. Gewali, A. Meng, J. S. Mitchell, and S. Ntafos. Path planning in $0/1/\infty$ weighted regions with applications. In *SCG '88: Proceedings of the fourth annual symposium on Computational geometry*, pages 266–278, 1988.

[37] Michael Gleicher. Integrating constraints and direct manipulation. In *SI3D '92: Proceedings of the 1992 symposium on Interactive 3D graphics*, pages 171–174, 1992.

[38] Jon Higgins. Sunrise, Banner Peak from 1000 Island Lake, Ansel Adams wilderness, Sierra Nevada. Yahoo Flickr, February 2007. http://www.flickr.com/photos/jon_higgins/388014714/; Access Date: February 17, 2009.

[39] Jason Hunter. Mammatus clouds. Yahoo Flickr, May 2007. http://www.flickr.com/photos/coreburn/487357814/; Access Date: February 17, 2009.

[40] Takeo Igarashi, Satoshi Matsuoka, and Hidehiko Tanaka. Teddy: A sketching interface for 3D freeform design. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 409–416, 1999.

[41] Marcelo Kallmann. Path planning in triangulations. In *Proceedings of the Workshop on Reasoning, Representation, and Learning in Computer Games, International Joint Conference on Artificial Intelligence (IJCAI)*, pages 49–54, 2005.

[42] Alex D. Kelley, Michael C. Malin, and Gregory M. Nielson. Terrain simulation using a model of stream erosion. In *SIGGRAPH '88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, pages 263–268, 1988.

[43] James J. Kuffner. Effective sampling and distance metrics for 3D rigid body path planning. In *Robotics and Automation, Proceedings. ICRA '04. 2004 IEEE International Conference*, volume 4, pages 3993–3998, 2004.

[44] Jason Lawrence and Thomas Funkhouser. A painting interface for interactive surface deformations. *Graph. Models*, 66(6):418–438, 2004.

[45] Seungkyu Lee. PSU near-regular texture database. Pennsylvania State University. http://vivid.cse.psu.edu/texturedb/gallery/; Access Date: February 17, 2009.

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

[47] Barbara London, John Upton, Ken Kobre, and Betsy Brill. *Photography.* Prentice Hall, seventh edition, 2002.

[48] Jeremy Long and David Mould. Dendritic stylization. *The Visual Computer*, 2008.

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

[50] Benoit B. Mandelbrot. *Fractal Landscapes Without Creases and With Rivers, in The Science of Fractal Images*, chapter Appendix A, pages 243–260. Springer-Verlag, 1988.

[51] Benoit B. Mandelbrot and John W. Van Ness. Fractional Brownian motions, fractional noises and applications. *SIAM Review*, 10(4):422–437, 1968.

[52] Lee Markosian, Michael A. Kowalski, Daniel Goldstein, Samuel J. Trychin, John F. Hughes, and Lubomir D. Bourdev. Real-time nonphotorealistic rendering. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 415–420, 1997.

[53] Stephen Marshak and Donald R. Prothero. *Earth: Portrait of a Planet*. W. W. Norton & Company, 2001.

[54] David Mould and Kevin Grant. Stylized black and white images from photographs. In *NPAR '08: Proceedings of the 6th international symposium on Non-photorealistic animation and rendering*, pages 49–58, 2008.

[55] F. K. Musgrave, C. E. Kolb, and R. S. Mace. The synthesis and rendering of eroded fractal terrains. In *SIGGRAPH '89: Proceedings of the 16th annual conference on Computer graphics and interactive techniques*, pages 41–50, 1989.

[56] F. Kenton Musgrave. *Methods for Realistic Landscape Imaging*. PhD thesis, Yale University, 1993.

[57] Kenji Nagashima. Computer generation of eroded valley and mountain terrains. *The Visual Computer*, 13(9–10):456–464, 1997.

[58] NASA. A face on Mars. NASA Images, July 1976. `http://nasaimages.org/luna/servlet/detail/NVA2~4~4~7500~108026:A-Face-On-Mars`; Access Date: February 17, 2009.

[59] NASA. Ancient martian highlands. NASA Images, 1977. `http://nasaimages.org/luna/servlet/detail/nasaNAS~20~20~120972~227675:Ancient-Martian-Highlands`; Access Date: February 17, 2009.

[60] NASA. Highest-resolution view of "face on Mars". NASA Images, April 2001. `http://nasaimages.org/luna/servlet/detail/NVA2~13~13~23084~123625:Highest-Resolution-View-of--Face-on`; Access Date: February 17, 2009.

[61] B. Neidhold, M. Wacker, and O. Deussen. Interactive physically based fluid and erosion simulation. In *Eurographics Workshop on Natural Phenomena*, pages 25–32, 2005.

[62] Christoph Niederberger, Dejan Radovic, and Markus Gross. Generic path planning for real-time applications. In *CGI '04: Proceedings of the Computer Graphics International*, pages 299–306, 2004.

[63] Lerma Olayres. 70610024. Yahoo Flickr, May 2005. `http://www.flickr.com/photos/lolay/14607205/`; Access Date: February 17, 2009.

[64] Joseph O'Rourke. *Computational Geometry in C*. Cambridge University Press, second edition, 1998.

[65] Dinesh K. Pai and L. M. Reissell. Multiresolution rough terrain motion planning. Technical Report TR-94-33, University of British Columbia, 1994. citeseer.ist.psu.edu/pai94multiresolution.html.

[66] Richard Parmiter. Confetti. Yahoo Flickr, November 2008. `http://www.flickr.com/photos/parmiter/2990899203/`; Access Date: February 17, 2009.

[67] Heinz-Otto Peitgen and Dietmar Saupe, editors. *The Science of Fractal Images*. Springer-Verlag New York, Inc., 1988.

[68] Ken Perlin. An image synthesizer. *SIGGRAPH Comput. Graph.*, 19:287–296, 1985.

[69] David Porter. Canyon de Chelly - Spider Rock. Yahoo Flickr, July 2007. `http://www.flickr.com/photos/davidaporter/847833343/`; Access Date: February 17, 2009.

[70] Przemyslaw Prusinkiewicz and Mark Hammel. A fractal model of mountains with rivers. In *Proceeding of Graphics Interface '93*, pages 174–180, 1993.

[71] David Pugh. Designing solid objects using interactive sketch interpretation. In *SI3D '92: Proceedings of the 1992 symposium on Interactive 3D graphics*, pages 117–126, 1992.

[72] Jon Ragnarsson. Hola Valley, Norway. Yahoo Flickr, August 2007. http://www.flickr.com/photos/jonragnarsson/1206975922/; Access Date: February 17, 2009.

[73] William Rowlands. Valley of redemption. Yahoo Flickr, February 2007. http://www.flickr.com/photos/wrowlands/402244682/; Access Date: February 17, 2009.

[74] V B Sapozhnikov and V I Nikora. Simple computer model of a fractal river network with fractal individual watercourses. *Journal of Physics A: Mathematical and General*, 26(15):623–627, 1993.

[75] Jens Schneider, Tobias Boldte, and Ruediger Westermann. Real-time editing, synthesis, and rendering of infinite landscapes on GPUs. In *Vision, Modeling and Visualization 2006*, pages 145–152, 2006.

[76] D. Sharon. Free rippled water texture for layers. Yahoo Flickr, November 2008. http://www.flickr.com/photos/pinksherbet/3002844223/; Access Date: February 17, 2009.

[77] Karan Singh and Eugene Fiume. Wires: A geometric deformation technique. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 405–414, 1998.

[78] Scott S. Snibbe, Kenneth P. Herndon, Daniel C. Robbins, D. Brookshire Conner, and Andries van Dam. Using deformations to explore 3D widget design. *SIGGRAPH Comput. Graph.*, 26(2):351–352, 1992.

[79] Planetside Software. Terragen. Planetside Software, September 2005. http://www.planetside.co.uk/terragen/; Access Date: February 17, 2009.

[80] Milan Sonka, Vaclav Hlavac, and Roger Boyle. *Image Processing, Analysis, and Machine Vision*. Thomson Engineering, third edition, 2007.

[81] Paul S. Strauss and Rikk Carey. An object-oriented 3D graphics toolkit. *SIGGRAPH Comput. Graph.*, 26(2):341–349, 1992.

[82] R. Szeliski and D. Terzopoulos. From splines to fractals. *SIGGRAPH Comput. Graph.*, 23:51–60, 1989.

[83] Tym. Li River cruise. Yahoo Flickr, October 2006. http://www.flickr.com/photos/tym/277135885/; Access Date: February 17, 2009.

[84] Richard F. Voss. *Random Fractal Forgeries*, pages 805–835. Springer-Verlag, 1985.

[85] Dennis Wackerly, William Mendenhall, and Richard L. Scheaffer. *Mathematical Statistics with Applications*. Duxbury Press, sixth edition, 2001.

[86] Robert E. Wallace. The San Andreas fault. USGS. http://pubs.usgs.gov/gip/earthq3/; Access Date: February 17, 2009.

[87] Patrick Henry Winston. *Artificial Intelligence*. Addison-Wesley, USA, third edition, 1992.

[88] Steven Worley. A cellular texture basis function. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 291–294, 1996.

[89] Ling Xu and David Mould. Modeling dendritic shapes using path planning. In *GRAPP 2007, Proceedings of the Second International Conference on Computer Graphics Theory and Applications*, number 2, pages 829–838, 2007.

[90] Yizhou Yu and J. T. Chang. Shadow graphs and surface reconstruction. In *ECCV '02: Proceedings of the 7th European Conference on Computer Vision-Part II*, pages 31–45, 2002.

[91] Robert C. Zeleznik, Kenneth P. Herndon, and John F. Hughes. SKETCH: An interface for sketching 3D scenes. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 163–170, 1996.

[92] Howard Zhou, Jie Sun, Greg Turk, and James M. Rehg. Terrain synthesis from digital elevation models. *IEEE Transactions on Visualization and Computer Graphics*, 13(4):834–848, July/August 2007.

[93] Roberto Zingales. Gravel 1280x1024. Yahoo Flickr, January 2007. `http://www.flickr.com/photos/filicudi/364756371/`; Access Date: February 17, 2009.

Appendix A

Complete Results

A.1 Overview

This chapter presents complete results for each of the terrains synthesized in Chapter 4. Each terrain's inputs, parameters, outputs, and running times are provided; the symbol f_t refers to the total feature count. This information is summarized in Table A.1. Unless otherwise noted, fBm is calculated using $H = 3.0$, Lacunarity=2.0, and Octaves=2. The calculation takes place over a 4×4 grid scaled to the height-field resolution – the step size along each axis is $0.0078 = 4.0/512$ for a 512×512 height field. Furthermore, the resulting fBm values are scaled by 30.0. Finally, arrows are used in some figures to aid in the visualization of certain strokes, and false colouring is used in profile ID and select cost images to emphasize the assignment of profiles and costs to strokes.

A.2 Realistic Terrains

A.2.1 V-Shaped Valley

The v-shaped valley was synthesized in 7 seconds using $\mu_w = 2.0$, $r = 1.5$, $s = 2.0$, $b = 15$, and $f_t = 2$. Generator node locations F are provided in Figure A.1a and the included profiles are given in Figures A.1(b–c); the profile in Figure A.1c is used when Dijkstra's algorithm explores non-valley regions. Generator costs were assigned using fBm, which is visualized in Figure A.1e. These inputs and parameters create the final render in Figure A.1f.

A.2.2 U-Shaped Valley

The u-shaped valley was synthesized in 8 seconds using $\mu_w = 2.0$, $r = 1.5$, $s = 2.0$, $b = 3$, and $f_t = 2$. Generator node locations F are provided in Figure A.2a and the included profiles are given in Figures A.2(b–c); the profile in Figure A.2c is used when Dijkstra's algorithm explores non-valley regions. Generator costs were assigned using fBm, which is visualized in Figure A.2e. These inputs and parameters create the final render in Figure A.2f.

A.2.3 Canyon

The canyon was synthesized in 7 seconds using $\mu_w = 2.0$, $r = 1.5$, $s = 2.0$, $b = 4$, and $f_t = 7$. Generator node locations F are provided in Figure A.3a and the included profile is given in Figure A.3b. Generator costs were assigned using fBm, which is visualized in Figure A.3d. These inputs and parameters create the final render in Figure A.3e.

A.2.4 Layers

The layers result was synthesized in 13 seconds using $\mu_w = 2.25$, $r = 1.25$, $s = 1.75$, $b = 4$, and $f_t = 7$. Generator node locations F are provided in Figure A.4a and the included profile is given in Figure A.4b. Generator costs were assigned using fBm ($H = 1.0$ and Octaves=8) evaluated over a 6×6 grid. These costs are visualized in Figure A.4d. These inputs and parameters create the final render in Figure A.4e.

A.2.5 Tower Karst Landscape

The tower karst landscape was synthesized in 7 seconds using $\mu_w = 0.5$, $r = 0.35$, $s = 4.0$, $b = 3$, and $f_t = 8$. Generator node locations F are provided in Figure A.5a and the included profiles are

Table A.1: Summary of each result's input and parameter values used in the proposed terrain synthesis algorithm: mean edge weight μ_w, maximum edge weight deviation r, generator node locations F (provided as figure numbers), sea level scaling value s, profiles P (provided as figure numbers), number of profiles n_p, blending bias b, total feature count f_t, and synthesis time t (in seconds). For conciseness, SBM replaces sketch-based modelling, and ED/EW replaces edge detection and edge weights.

Result	μ_w	r	F	s	P	n_p	b	f_t	t
V-shaped valley	2.0	1.5	A.1a	2.0	A.1(b–c)	2	15	2	7 s
U-shaped valley	2.0	1.5	A.2a	2.0	A.2(b–c)	2	3	2	8 s
Canyon	2.0	1.5	A.3a	2.0	A.3b	1	4	7	7 s
Layers	2.25	1.25	A.4a	1.75	A.4b	1	4	7	13 s
Tower Karst landscape	0.5	0.35	A.5a	4.0	A.5(b–d)	3	3	8	7 s
Hills	0.75	0.375	A.6a	2.5	A.6(b–d)	3	3	6	7 s
Mountain range	1.0	0.75	A.7a	1.01	A.7(c–f)	4	4	100	24 s
River terrace	1.0	0.5	A.8c	1.1	A.8b	1	8	3	8 s
Fault	1.5	1.0	A.9a	1.01	A.9(c–d)	2	3	6	10 s
Crater landscape	1.5	1.0	A.10(a–b)	1.01	A.10(c–f)	4	4	100	36 s
Cinder cone	1.0	0.5	A.11(a–b)	2.0	A.11(c–e)	3	3	4	7 s
Lunar landscape	1.5	1.0	A.12a	1.01	A.12(b–e)	4	4	38	18 s
Musgrave landscape	1.5	1.0	A.13(a–b)	1.01	A.13(c–e)	3	4	402	27 s
Edge detection	1.0	0.75	A.14a	1.01	A.14c	1	4	288	71 s
Edge weights	N/A	N/A	A.15a	2.5	N/A	0	4	7	12 s
ED/EW #1	N/A	N/A	A.16a	5.0	N/A	0	3	54	63 s
ED/EW #2	N/A	N/A	A.17a	5.0	N/A	0	3	229	128 s
ED/EW #3	N/A	N/A	A.18a	5.0	N/A	0	3	344	135 s
ED/EW #4	N/A	N/A	A.19a	5.0	N/A	0	3	229	75 s
"U of S" text	N/A	N/A	A.20a	2.5	N/A	0	3	22	9 s
IMG logo	N/A	N/A	A.21a	2.5	N/A	0	3	16	10 s
λ symbol	N/A	N/A	A.22a	2.5	N/A	0	3	3	9 s
SBM #1	1.5	0.75	A.23a	1.25	A.23(b–c)	2	3	31	12 s
SBM #2	1.5	0.75	A.24a	1.1	A.24(b–e)	4	3	58	9 s
SBM #3	2.0	1.0	A.25a	1.01	A.25(b–d)	3	3	38	11 s
SBM #4	1.5	0.75	A.26a	1.01	A.26(b–d)	3	3	15	7 s

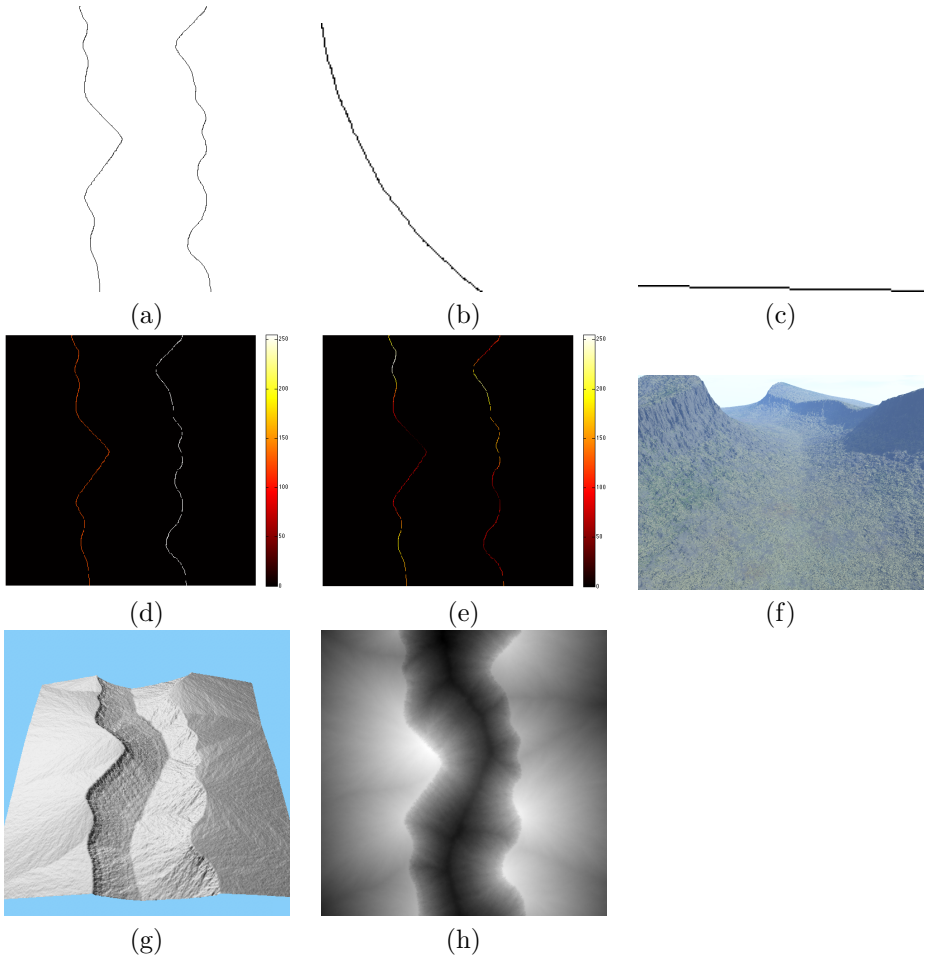


Figure A.1: Input and output for the v-shaped valley result: (a) generator node locations F, (b–c) profiles P, (d) feature ID, (e) generator node costs, (f) final render, (g) final OpenGL render, and (h) final height field. False colouring has been used in images (d–e) to increase label and cost visibility. The colourbar associated with each image maps pixel colour to a pixel intensity in the interval [0,255]; lower pixel intensity corresponds to smaller cost.

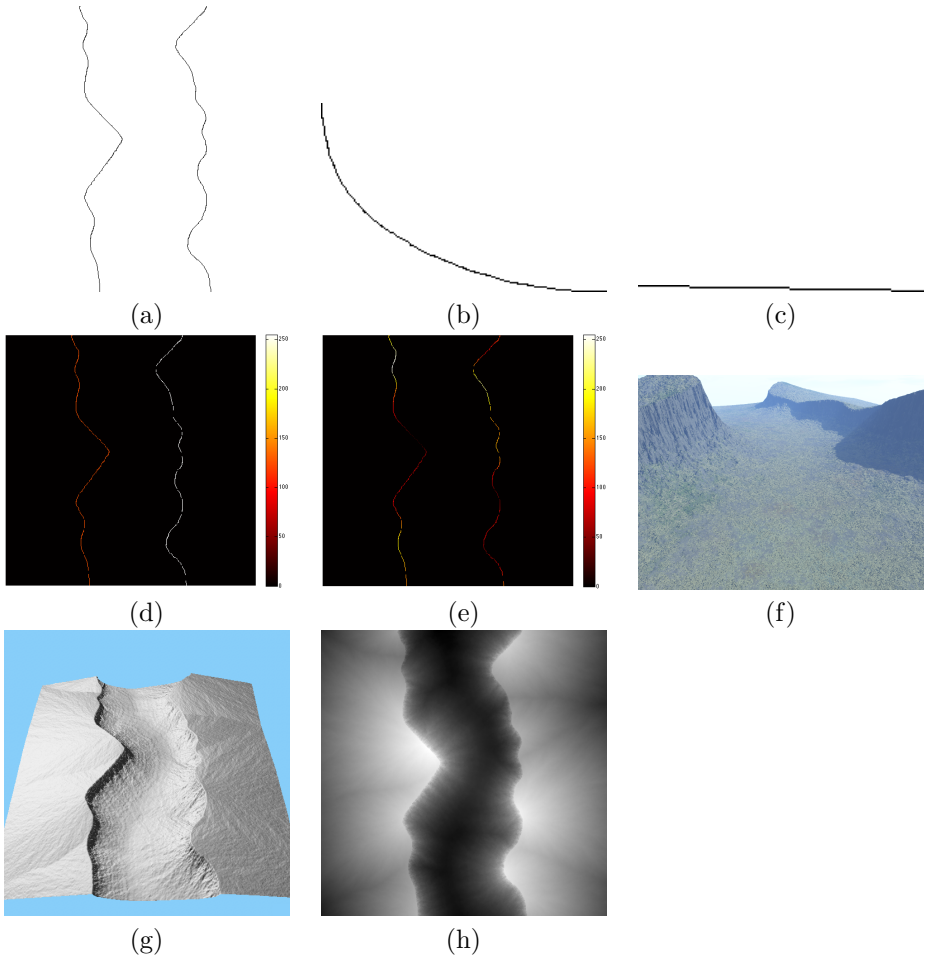


Figure A.2: Input and output for the u-shaped valley result: (a) generator node locations F, (b–c) profiles P, (d) feature ID, (e) generator node costs, (f) final render, (g) final OpenGL render, and (h) final height field. False colouring has been used in images (d–e) to increase label and cost visibility. The colourbar associated with each image maps pixel colour to a pixel intensity in the interval [0,255]; lower pixel intensity corresponds to smaller cost.

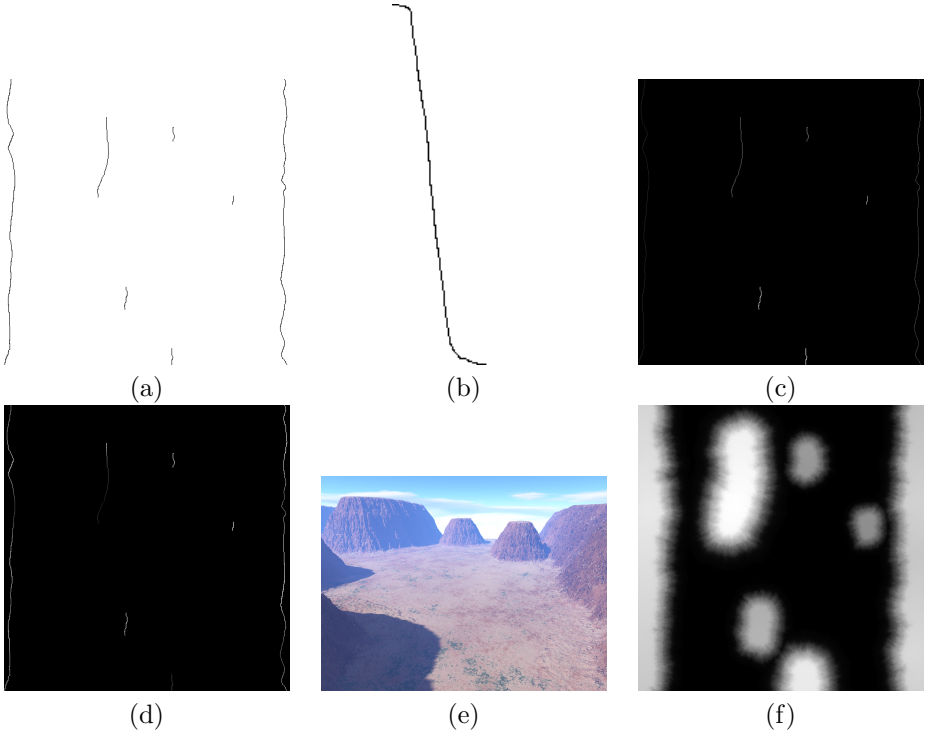


Figure A.3: Input and output for the canyon result: (a) generator node locations F, (b) profiles P, (c) feature ID, (d) generator node costs, (e) final render, and (f) final height field.

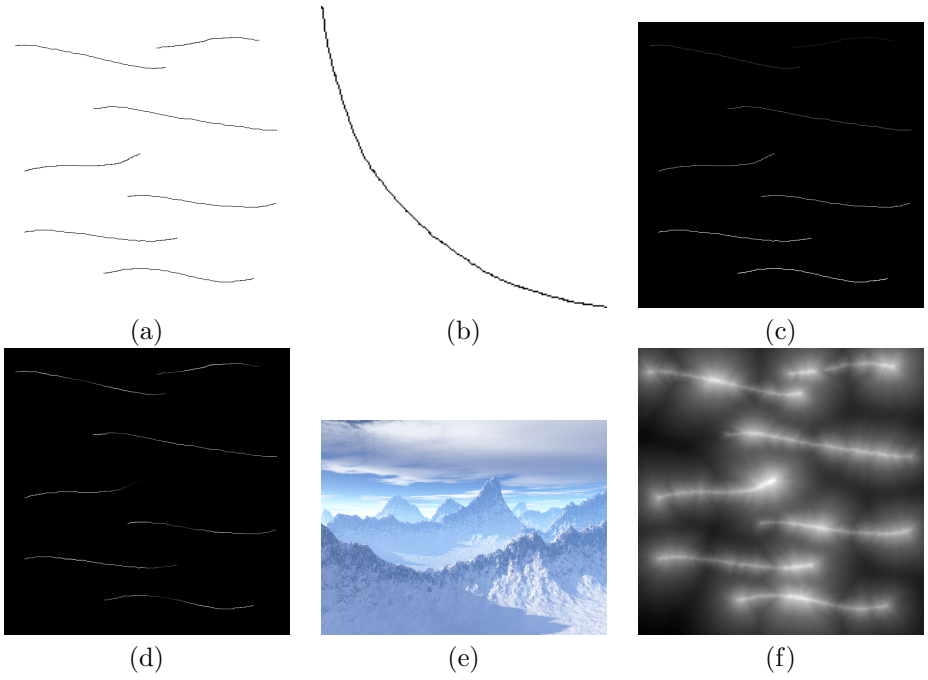


Figure A.4: Input and output for the layers result: (a) generator node locations F, (b) profiles P, (c) feature ID, (d) generator node costs, (e) final render, and (f) final height field.

given in Figures A.5(b–d); profile ID was calculated as feature ID number modulo three. Generator costs were assigned using fBm, which is visualized in Figure A.5g. These inputs and parameters create the final render in Figure A.5h.

A.2.6 Hills

The hills result was synthesized in 7 seconds using $\mu_w = 0.75$, $r = 0.375$, $s = 2.5$, $b = 3$, and $f_t = 6$. Generator node locations F are provided in Figure A.6a. The locations of the hills were manually placed and their initial costs were calculated using fBm evaluated over a 3×3 grid. Specifically, the generator node costs at the locations visualized in Figure A.6a are calculated by scaling fBm within a cost range of $[0.0, 20.0]$. The costs of all generator nodes are visualized in Figure A.6g, and profile ID is calculated as feature ID number modulo three, in order to add diversity to the hills. Red pixels in Figure A.6f correspond to Figure A.6b, green pixels correspond to Figure A.6c, and blue pixels correspond to Figure A.6d. These inputs and parameters create the final render in Figure A.6h.

A.2.7 Mountain Range

The mountain range result was synthesized in 24 seconds using $\mu_w = 1.0$, $r = 0.75$, $s = 1.01$, $b = 4$, and $f_t = 100$. Generator node locations F are provided in Figure A.7a. The locations of the mountain peaks are randomly generated such that they do not reside in a black region of Figure A.7b. The initial costs of these peaks are calculated using fBm directly. Ridge node generator costs are defined as 50% of the original node cost. The costs of all generator nodes are visualized in Figure A.7h, and profile ID is calculated as feature ID number modulo three, in order to add diversity to the mountain ranges.

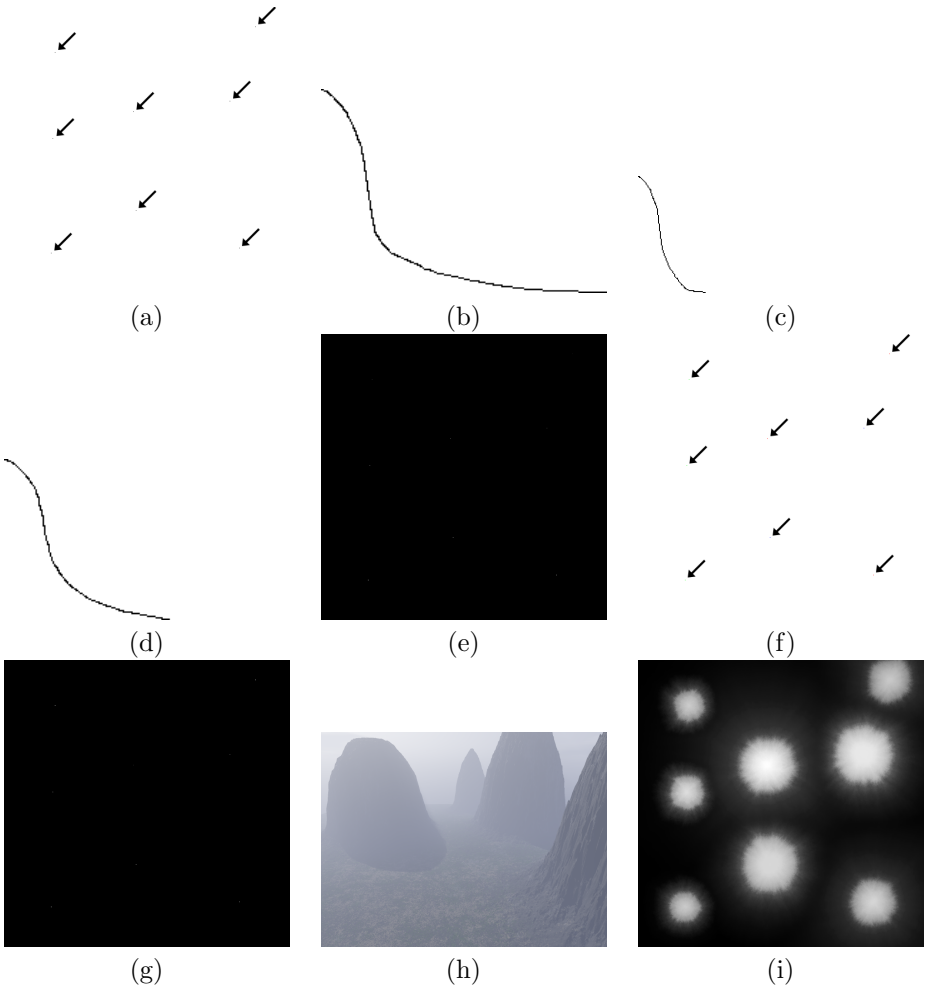


Figure A.5: Input and output for the tower karst landscape result: (a) generator node locations F, (b–d) profiles P, (e) feature ID, (f) profile ID, (g) generator node costs, (h) final render, and (i) final height field. Arrows in (a,f) are not part of the input and are merely visual aids for single node generators which are otherwise difficult to see.

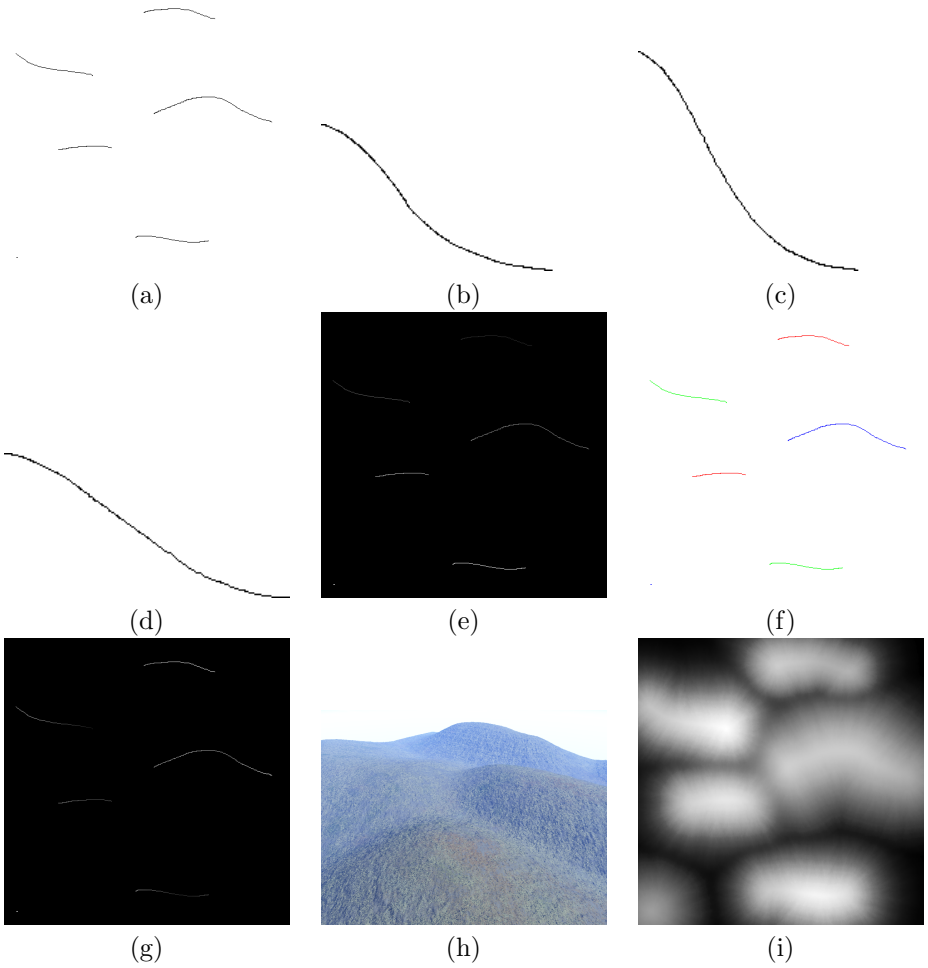


Figure A.6: Input and output for the hills result: (a) generator node locations F, (b–d) profiles P, (e) feature ID, (f) profile ID, (g) generator node costs, (h) final render, and (i) final height field. Image (f) uses false colouring to visualize profile assignment.

Green pixels in Figure A.7g correspond to Figure A.7c, blue pixels correspond to Figure A.7d, and purple pixels correspond to Figure A.7e. These inputs and parameters create the final render in Figure A.7i.

A.2.8 River Terrace

The river terrace was synthesized in 8 seconds using $\mu_w = 1.0$, $r = 0.5$, $s = 1.1$, $b = 8$, and $f_t = 3$. Generator node locations F are provided in the non-black regions in Figure A.8c and the included profile is given in Figure A.8b. Generator costs were assigned as a function of cost from the dendritic path in Figure A.8a; this path has a cost of zero. More specifically, the costs from this path are quantized; the initial, larger set of costs is mapped to a smaller set of costs. The region of each cost in the smaller set is determined from a percentage range of the maximum cost from the dendritic path. These ranges are $(87.5\%, 100.0\%]$, $(50.0\%, 87.5\%]$, and $(12.5\%, 50.0\%]$; each region is assigned a cost of 0.0, 40.0, and 80.0, respectively. These inputs and parameters create the final render in Figure A.8f.

A.2.9 Fault

The fault was synthesized in 10 seconds using $\mu_w = 1.5$, $r = 1.0$, $s = 1.01$, $b = 3$, and $f_t = 6$. Generator node locations F are provided in Figure A.9a and the included profiles are given in Figures A.9(c–d). Red pixels in Figure A.9e correspond to Figure A.9d and blue pixels correspond to Figure A.9c. Generator costs were assigned in two phases: 1) fault costs and 2) surrounding generator costs. Fault costs were assigned in two passes. The first pass creates the lip of the fault. The lip is created by selecting the nodes who have a path length of 8 from the path in Figure A.9b; Dijkstra's algorithm assigns a path length greater than zero to each non-generator node. All nodes on the path in Figure A.9b have an initial cost of zero. Seventy-five ridges that emanate from these nodes are formed. These nodes must be within 80 units of a lip node and their cost is taken as 35% of the existing node cost. The surrounding features, which consist of 5 peaks and 6 ridges, have their initial cost calculated via fBm and are not permitted to appear within 75 units of a fault feature. These inputs and parameters create the final render in Figure A.9h.

A.2.10 Crater Landscape

The crater landscape was synthesized in 36 seconds using $\mu_w = 1.5$, $r = 1.0$, $s = 1.1$, $b = 4$, and $f_t = 100$. Generator node locations F are provided in Figures A.10(a–b). Figure A.10a defines the location of the surrounding procedural mountain peaks and hills and Figure A.10b manually defines the location of the crater. The included profiles are given in Figures A.10(c–f) and their usage it shown in Figure A.10h. Red pixels in Figure A.10h correspond to Figure A.10f, green pixels correspond to Figure A.10c, and purple pixels correspond to Figure A.10d. It is important to note that the profile shown in Figure A.10e is always used when Dijkstra's algorithm searches the interior of the crater, and blending ignores the costs of non-crater features inside the crater. This ensures that the rounded basin is produced.

In this result, two fBm height fields are used to define the initial costs of features: one for the background mountain range and midground foothills (fBm_M), and one for the remaining features (fBm_F). The fBm_M height field is sampled over a 12×12 grid and the resulting values are scaled by 10.0; this increases high-frequency detail. The resulting fBm_F values are scaled by 10.0 as well. In either case, cost assignment depends on feature type. The midground foothills (24 features) in Figure A.10a are assigned an initial cost of fBm_M scaled between $[50\%, 80\%]$ of the maximum fBm_M value $max\{\text{fBm}_M\}$. The mountain range generator nodes (40 features) are assigned an initial cost of fBm_M scaled between $[0\%, 25\%]$ of $max\{\text{fBm}_M\}$. The foreground hills (35 features) are assigned an initial cost of fBm_F scaled between $[80\%, 100\%]$ of the maximum fBm_F value. The crater generator nodes's costs require a more complicated process. An approximate cost field is calculated as per step 6 of Algorithm 3 and the average of the node costs in Figure A.10b minus 25.0 is calculated and stored as μ_c. Then, the final cost of these nodes is calculated as fBm_F

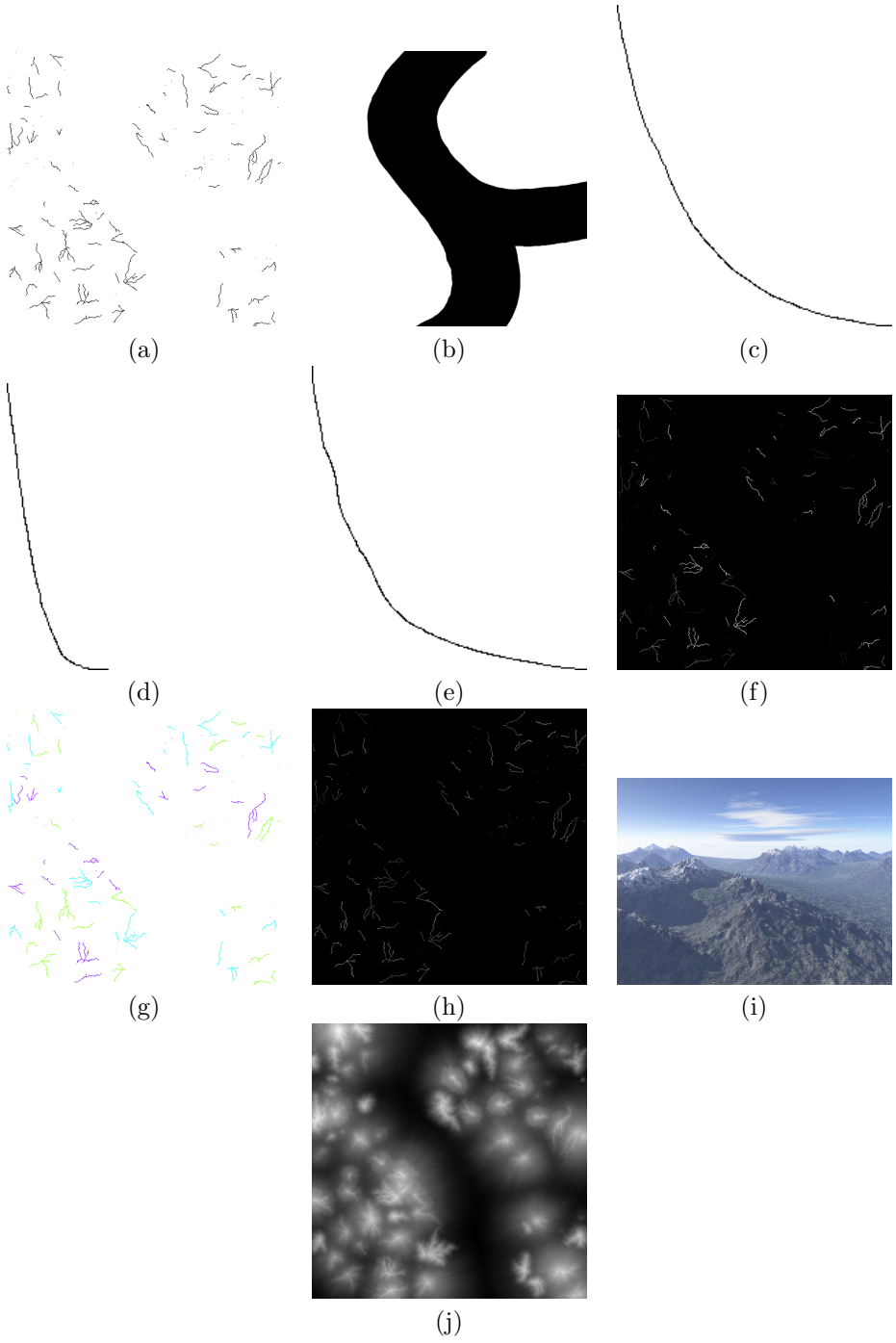


Figure A.7: Input and output for the mountain range result: (a) generator node locations F, (b) valley regions, (c–e) profiles P, (f) feature ID, (g) profile ID, (h) generator node costs, (i) final render, and (j) final height field. Image (g) uses false colouring to visualize profile assignment.

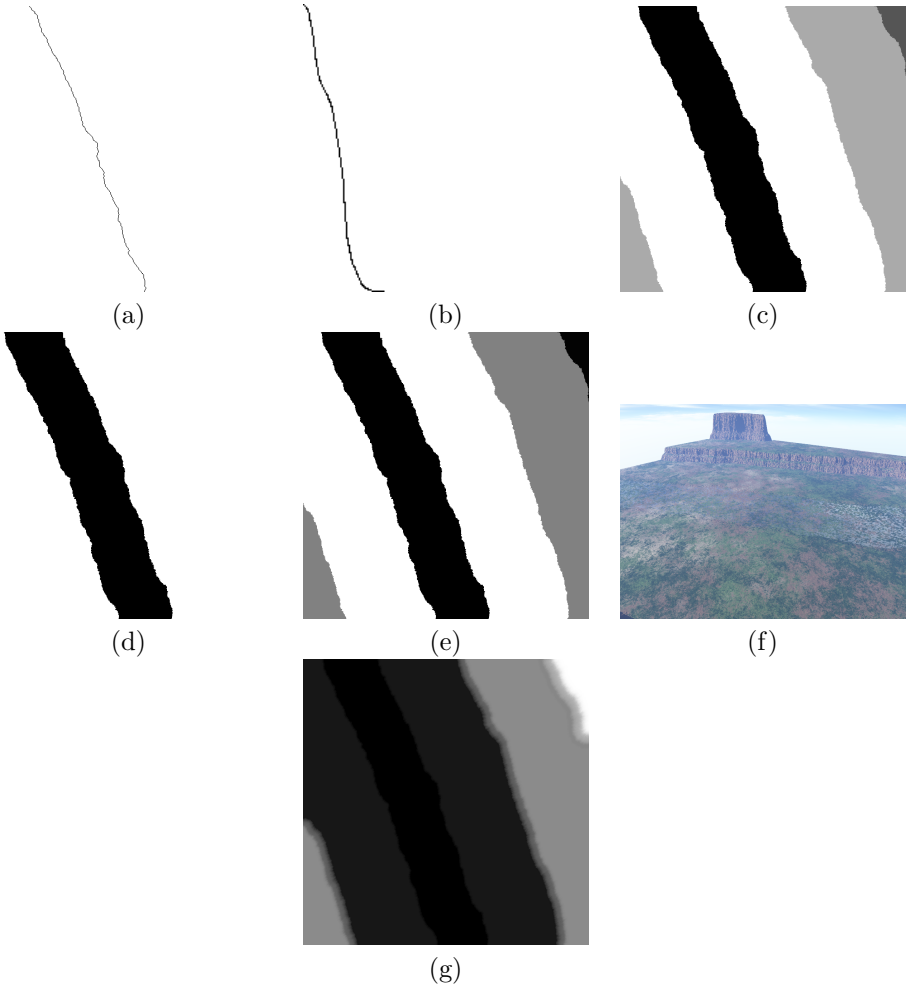


Figure A.8: Input and output for the river terrace result: (a) initial generator node locations F, (b) profiles P, (c) feature ID, (d) profile ID, (e) generator node costs, (f) final render, and (g) final height field.

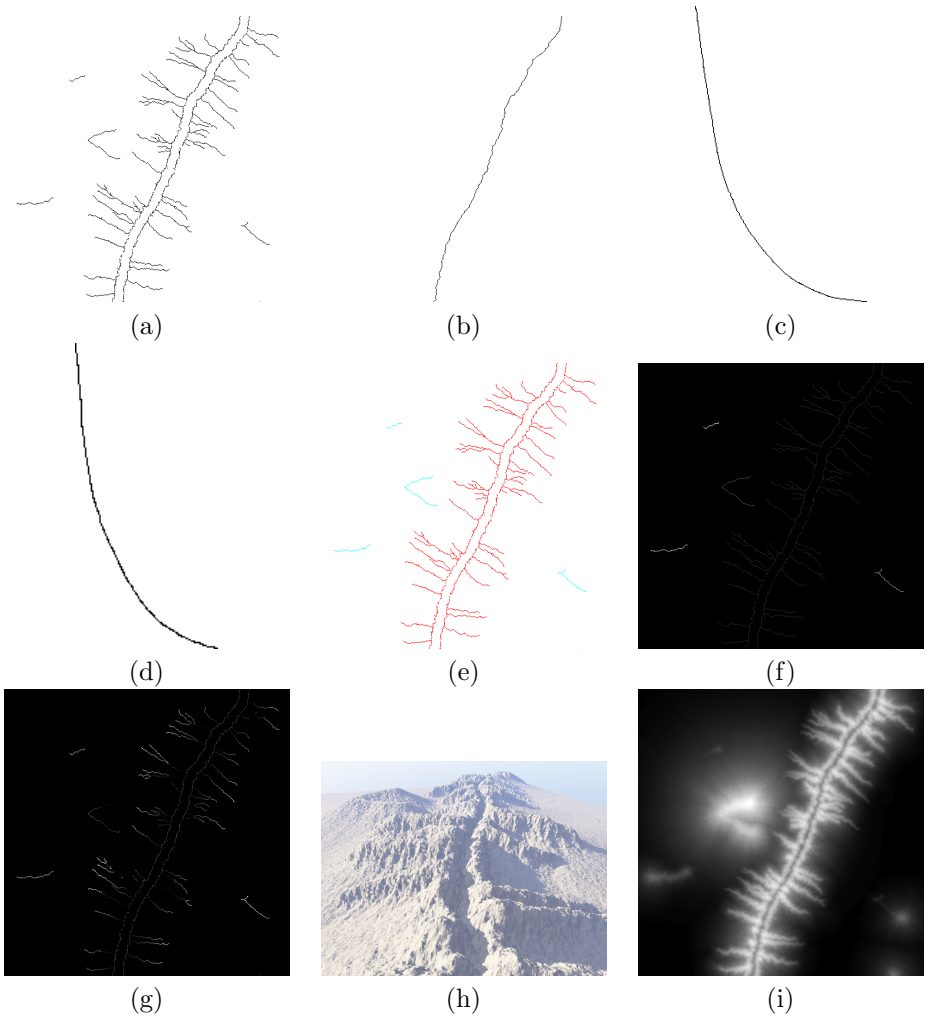


Figure A.9: Input and output for the fault result: (a) generator node locations F, (b) initial path, (c–d) profiles P, (e) profile ID, (f) feature ID, (g) generator node costs, (h) final render, and (i) final height field. Image (e) uses false colouring to visualize profile assignment.

scaled within the the range $[-0.2 \cdot \mu_c, 0.2 \cdot \mu_c]$. The costs of all generator nodes are visualized in Figure A.10i. These inputs and parameters create the final render in Figure A.10j.

A.2.11 Cinder Cone Volcano

The cinder cone volcano was synthesized in 7 seconds using $\mu_w = 1.0$, $r = 0.5$, $s = 2.0$, $b = 3$, and $f_t = 4$. Generator node locations F are provided in Figures A.11(a–b). Figure A.11a visualizes every stroke used and Figure A.11b manually defines the location of the cinder cone volcano. The included profiles are given in Figures A.11(c–e) and their usage it shown in Figure A.11g. Green pixels in Figure A.11g correspond to Figure A.11c and blue pixels correspond to Figure A.11d; the profile in Figure A.11e is used whenever Dijkstra's algorithm searches the crater region. The costs of generator nodes are defined using fBm, and cost assignment depends on feature type. The cost of the cinder cone volcano generator nodes is calculated as a normalized 1D Gaussian function (situated along the x-axis) with the standard deviation σ a function of the maximum width W_{max} of the stroke in Figure A.11b. This calculation is given in Equation A.1.

$$\sigma = (2.0 \cdot W_{max})/6.0 \tag{A.1}$$

The origin is defined as the minimum x-value of a generator node in Figure A.11b. The Gaussian value is then scaled by 40.0 and fBm scaled within $[-10.0, 10.0]$ is added to this product. The surrounding features are assigned an initial cost of fBm scaled within a range that is a function of the maximum cost $max\{c\}$ of a cinder cone generator node. This cost range is $[1.50 \cdot max\{c\}, 2.0 \cdot max\{c\}]$. Furthermore, these features cannot reside within 128 units of a cinder cone feature. The initial costs of the surrounding feature ridges are defined as 50% of the existing node costs. The costs of all generator nodes are visualized in Figure A.11h. These inputs and parameters create the final render in Figure A.11i.

A.2.12 Lunar Landscape

The crater landscape was synthesized in 18 seconds using $\mu_w = 1.5$, $r = 1.0$, $s = 1.01$, $b = 4$, and $f_t = 38$. Generator node locations F are provided in Figure A.12a. Thirty-eight features were both procedurally and manually placed, as shown in Figure A.12a. Figures A.12(b–e) show the profiles for the crater exterior, procedurally located hills, manually located hills, and crater interior, respectively. The profile in Figure A.12b is used to create the crater lip, the profiles in Figures A.12(c–d) are used to create the rounded hills, and the profile in Figure A.12e is used to create the bowl-shaped interior of the craters. These associations are visualized in Figure A.12g: blue pixels correspond to Figure A.12b, dark green pixels correspond to Figure A.12c, and light green pixels correspond to Figure A.12d. Furthermore, Dijkstra's algorithm uses only the profile in Figure A.12e when searching the graph in crater regions, and costs from non-crater features do not contribute to the final blend in the crater's interior. This ensures that the craters will have a rounded profile in their interior. Feature identification is visualized in Figure A.12f.

Crater generator nodes have their cost defined by fBm scaled within a specific cost range, but this cost range depends on crater size. Large craters have a radius in the interval of $[10.7, 85.3]$ units, and generator node costs for these craters are calculated by scaling fBm within a cost range of $[0.0, 12.5]$ units. Small craters have a radius in the interval of $[8.5, 14.2]$ units, and generator node costs for these craters are calculated by scaling fBm within a cost range of $[12.5, 18.75]$ units; the hills's generator costs are calculated using the same cost range as the small craters. The costs of all generator nodes are visualized in Figure A.12h. Regardless of crater radius, the actual radius is calculated as a uniform random value within the provided interval. These inputs and parameters create the final render in Figure A.12i.

A.2.13 Musgrave Landscape

The Musgrave landscape was synthesized in 27 seconds using $\mu_w = 1.5$, $r = 1.0$, $s = 1.01$, $b = 4$, and $f_t = 402$. Generator node locations F are provided in Figures A.13(a–b). Figure A.13a defines

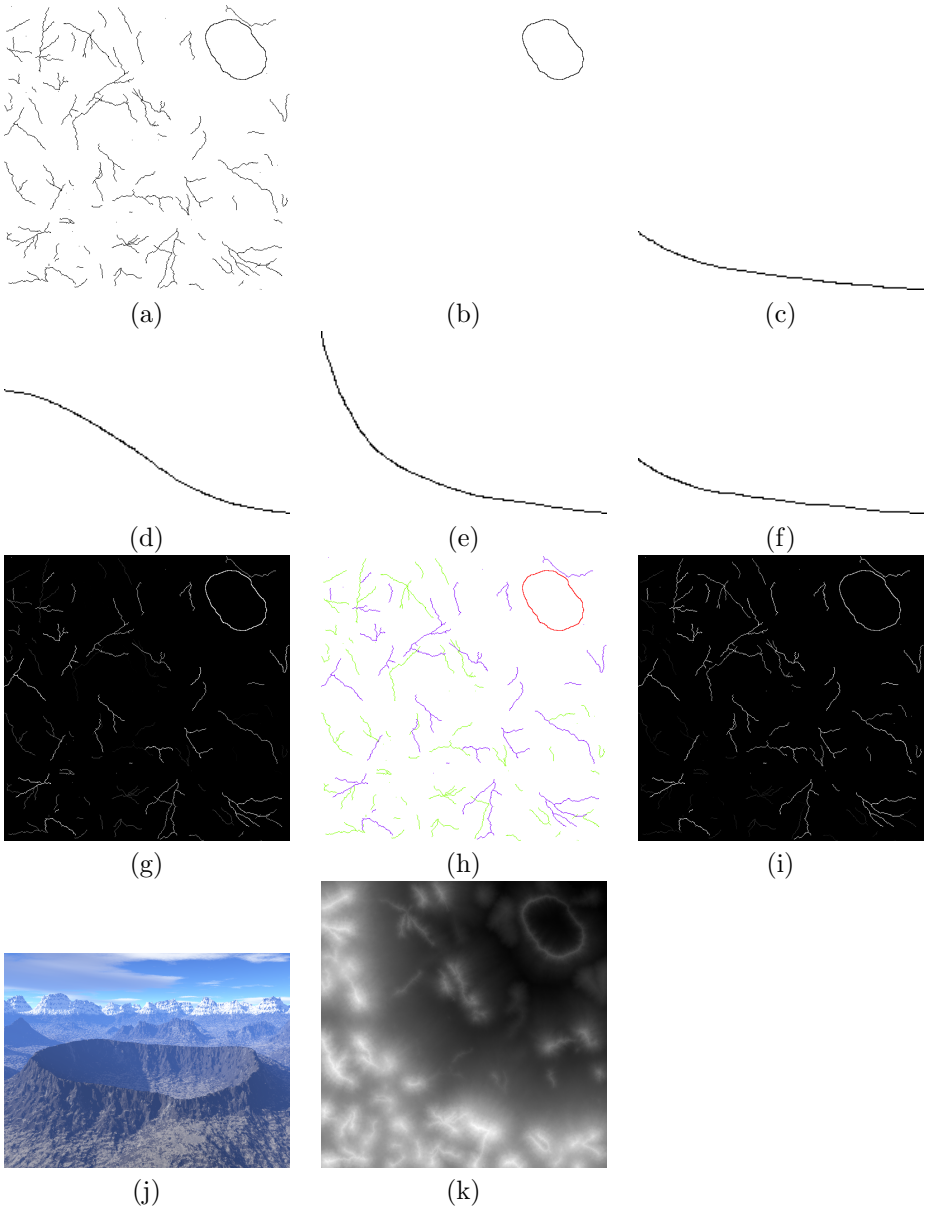


Figure A.10: Input and output for the crater landscape result: (a–b) generator node locations F, (c–f) profiles P, (g) feature ID, (h) profile ID, (i) generator node costs, (j) final render, and (k) final height field. Image (h) uses false colouring to visualize profile assignment.

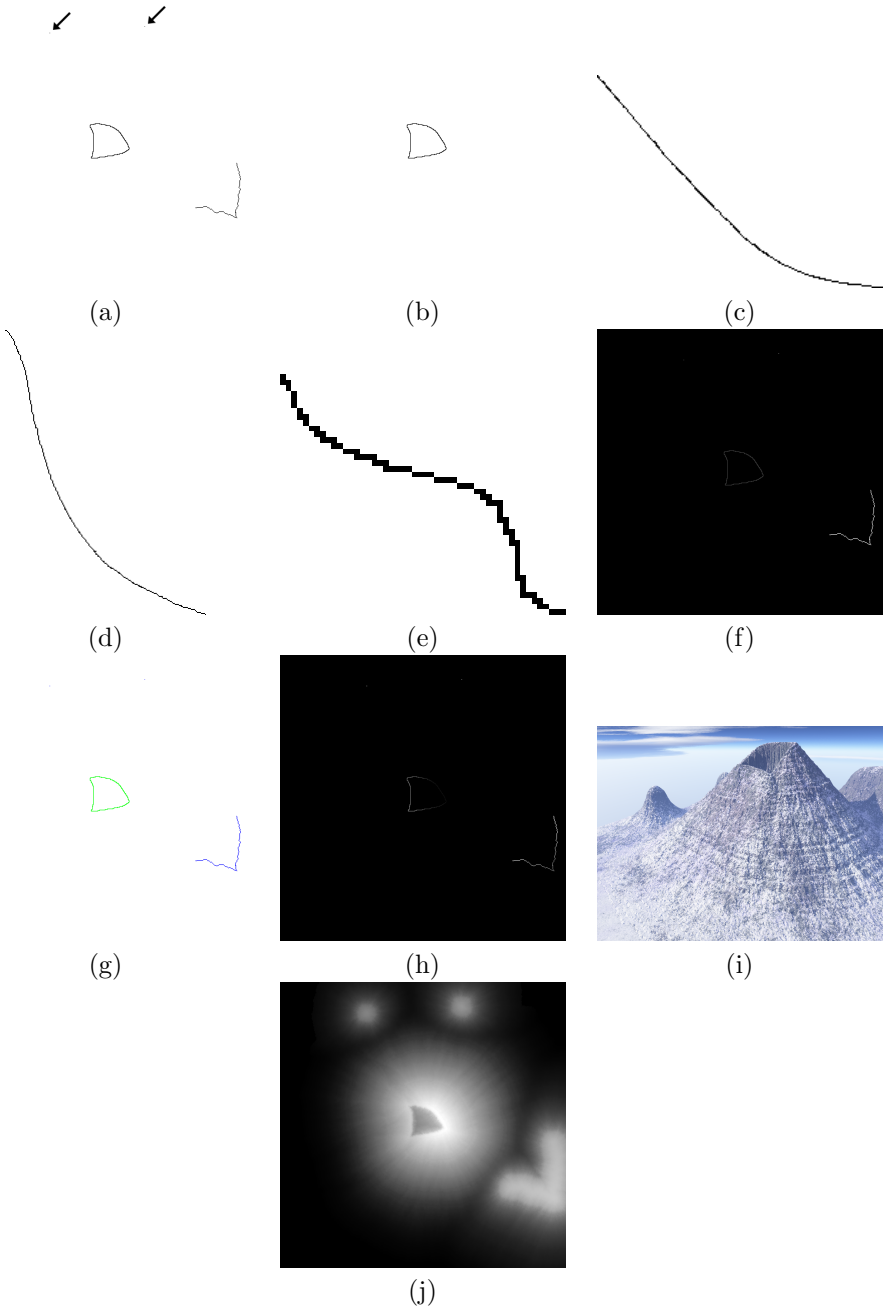


Figure A.11: Input and output for the cinder cone result: (a–b) generator node locations F, (c–e) profiles P, (f) feature ID, (g) profile ID, (h) generator node costs, (i) final render, and (j) final height field. Arrows in (a) are not part of the input and are merely visual aids for single node generators which are otherwise difficult to see. Image (g) uses false colouring to visualize profile assignment.

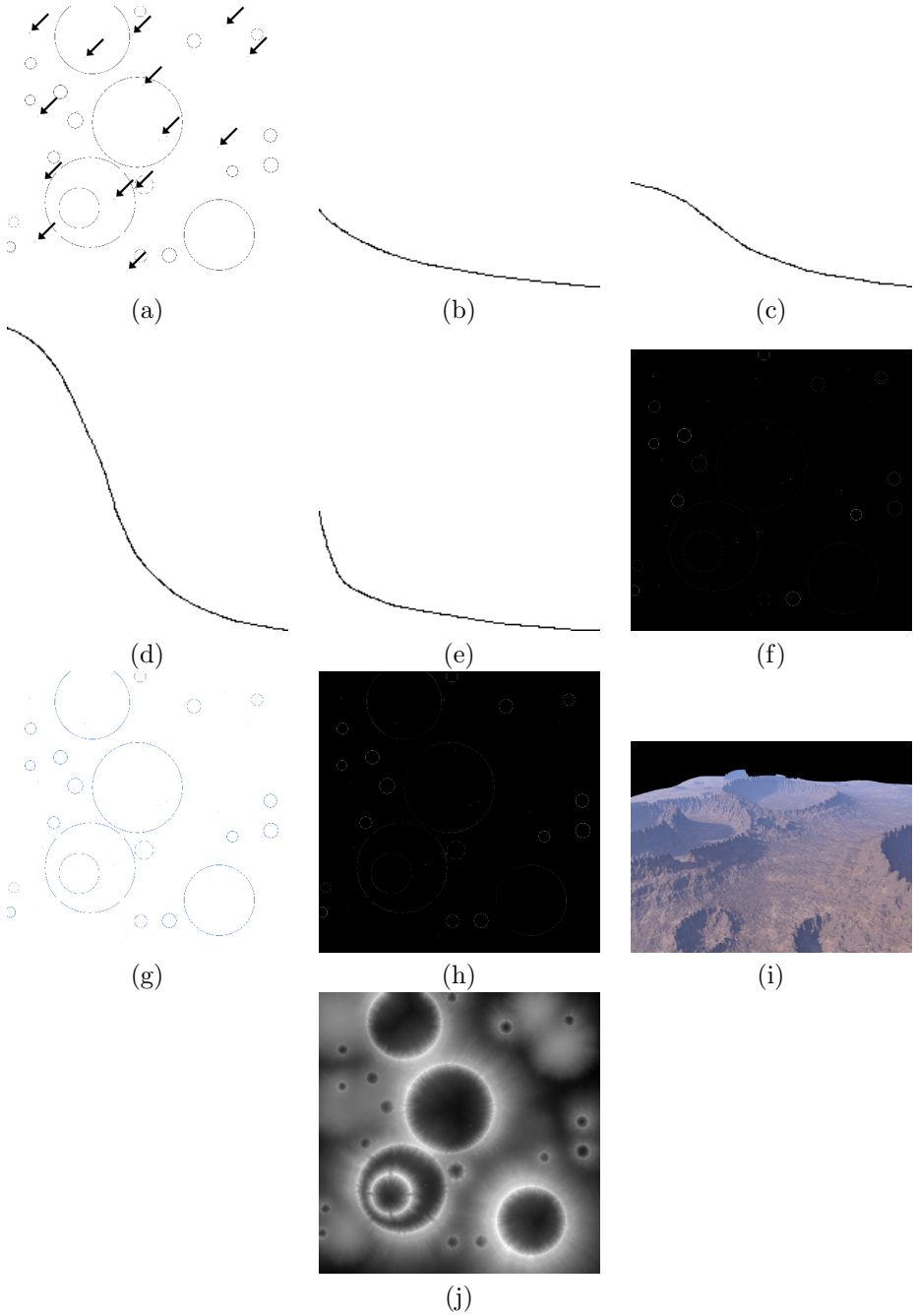


Figure A.12: Input and output for the lunar landscape result: (a) generator node locations F, (b–e) profiles P, (f) feature ID, (g) profile ID, (h) generator node costs, (i) final render, and (j) final height field. Arrows in (a) are not part of the input and are merely visual aids for single node generators which are otherwise difficult to see. Image (g) uses false colouring to visualize profile assignment.

the mountain peaks and Figure A.13b defines the lower elevation ridges that connect the peaks. Furthermore, complementary features such as peaks and ridges are procedurally located, as shown in Figure A.13f. Initial ridge node costs are calculated as 37.5% of the existing node cost. The included profiles are given in Figures A.13(c–e) and their usage is shown in Figure A.13g. Red pixels in Figure A.13g correspond to Figure A.13c, green pixels correspond to Figure A.13d, and blue pixels correspond to Figure A.13e. In this result, fBm (Octaves=6) is sampled over a 3×3 grid and the resulting values are scaled by 60.0. However, cost assignment depends on feature type. Each generator node (46 features) in Figure A.13b is assigned a cost of fBm scaled between $[60\%, 85\%]$ of the maximum fBm value $max\{\text{fBm}\}$. The generator nodes (6 features) in Figure A.13a are assigned a cost of fBm scaled between $[40\%, 60\%]$ of $max\{\text{fBm}\}$. The lower altitude feature generators (150 features) are assigned a cost of fBm inverted and scaled between $[60\%, 85\%]$ of $max\{\text{fBm}\}$; the low altitude hill generators (200 features) are assigned a cost of fBm scaled between $[85\%, 100\%]$ of $max\{\text{fBm}\}$. The costs of all generator nodes are visualized in Figure A.13h and the above inputs and parameters create the final render in Figure A.13i. Lastly, all heights were scaled by 45% to more closely approximate the reference render; scaling was performed in Terragen.

A.3 Image Driven Terrains

A.3.1 Edge Detection

The edge detection result was synthesized in 71 seconds using $\mu_w = 1.0$, $r = 0.75$, $s = 1.01$, $b = 4$, and $f_t = 288$. Generator node locations F are provided in Figure A.14a and the 50 accompanying ridges are visualized in Figure A.14e. Ridge generator node costs were calculated using 30% of the existing node cost. Terrain style was provided via the included profile seen in Figure A.14c. Edge detection was performed on Figure A.14b [76] using a 3×3 Sobel edge detector. The threshold t_g on the gradient magnitude (for edge detection) is taken to be the halfway point between the minimum $grad_{min}$ and maximum $grad_{max}$ gradient magnitude. This calculation is given in Equation A.2.

$$t_g = grad_{min} + 0.5 \cdot (grad_{max} - grad_{min}) \tag{A.2}$$

Generator costs were assigned using fBm, which is visualized in Figure A.14e. These inputs and parameters create the final render in Figure A.14f.

A.3.2 Edge Weights

The edge weights result was synthesized in 12 seconds using $s = 2.5$, $b = 4$, and $f_t = 7$. Edge weights are calculated using Figure A.15b [4], eliminating the need for μ_w and r. Generator node locations F are provided in Figure A.15a. This figure also includes 20 ridges, whose generator node costs were calculated using 50% of the existing node cost. Terrain style was provided via Figure A.15b; profiles were not used. Specifically, edge weights are assigned the normalized gray value at each pixel in Figure A.15b scaled within $[0.25, 2.5]$. Figure A.15b is converted to grayscale using Equation A.3.

$$gray = 0.3 \cdot red + 0.59 \cdot green + 0.11 \cdot blue \tag{A.3}$$

This results in all edges of a given node having the same weight. Generator node costs were assigned using fBm, which are visualized in Figure A.15d. These inputs and parameters create the final render in Figure A.15e.

A.3.3 Edge Detection and Edge Weights #1

In this result, generator node locations F are calculated by applying a 3×3 Sobel edge detector to Figure A.16b [28] and synthesizing 14 accompanying ridges. The threshold t_g on the gradient magnitude is calculated as per Equation A.2, and these edges are visualized in Figure A.16d. The resulting set of locations F is visualized in Figure A.16a. Edge weights are determined according to Figure A.16c [93]. Specifically, the gray value of each pixel in Figure A.16c is calculated according

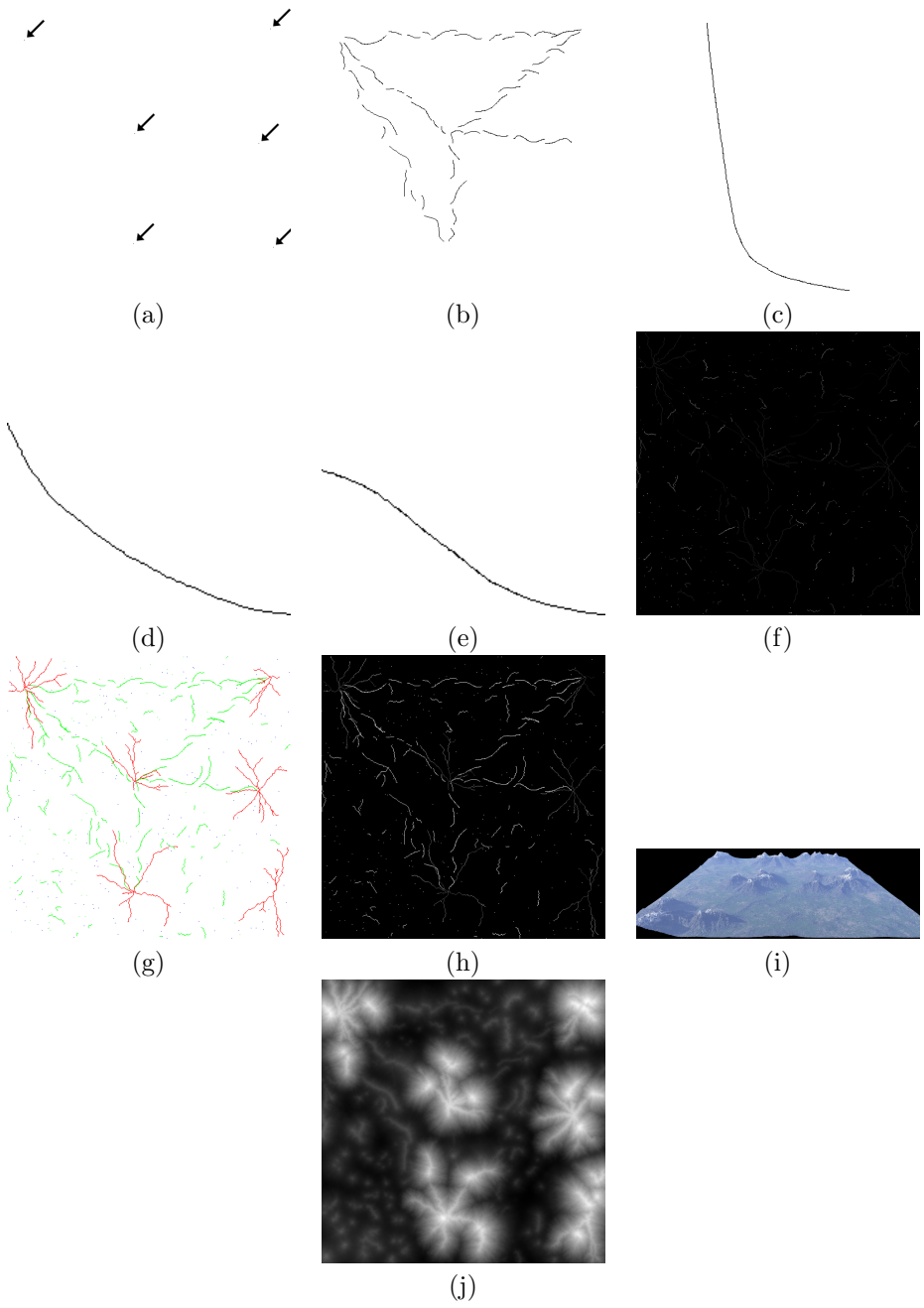


Figure A.13: Input and output for the Musgrave landscape result: (a–b) manually placed generator node locations F, (c–e) profiles P, (f) feature ID, (g) profile ID, (h) generator node costs, (i) final render, and (j) final height field. Arrows in (a) are not part of the input and are merely visual aids for single node generators which are otherwise difficult to see. Image (g) uses false colouring to visualize profile assignment and contains both manually and procedurally created strokes.

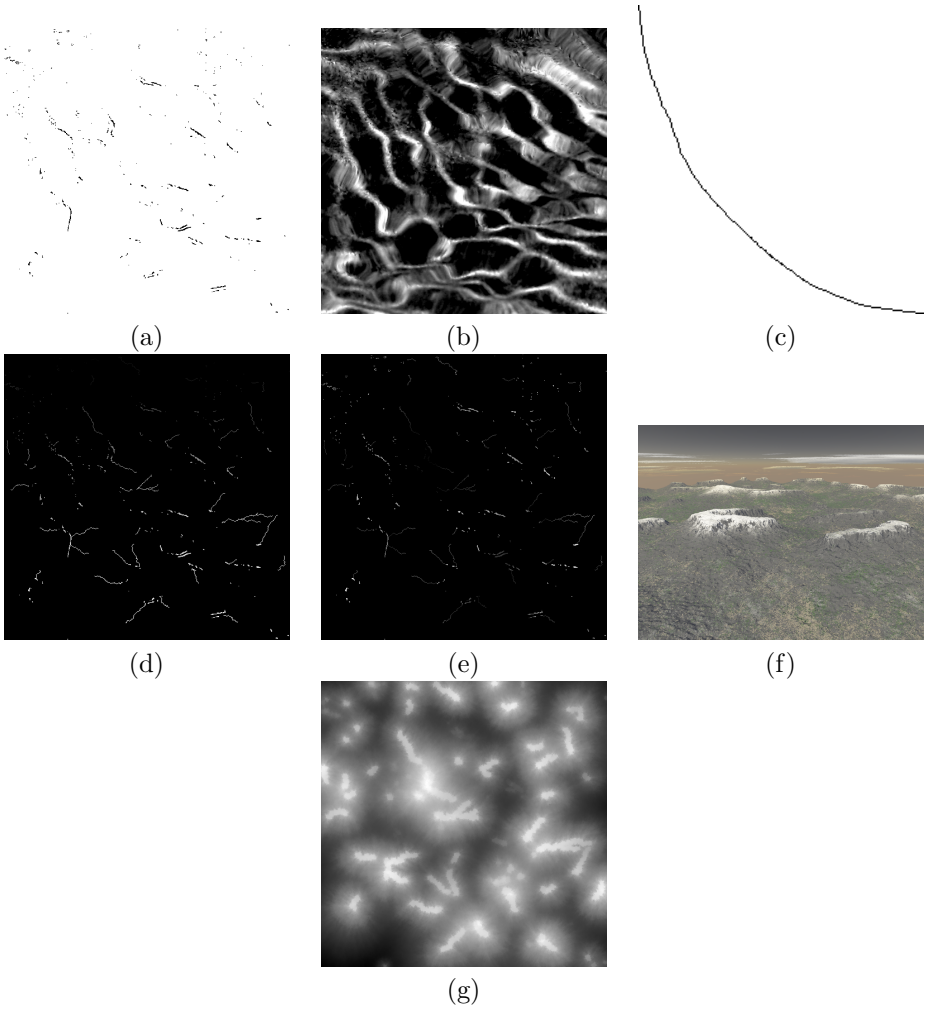


Figure A.14: Input and output for the edge detection result: (a) generator node locations F, (b) input texture [76], (c) profiles P, (d) feature ID, (e) generator node costs, (f) final render, and (g) final height field.

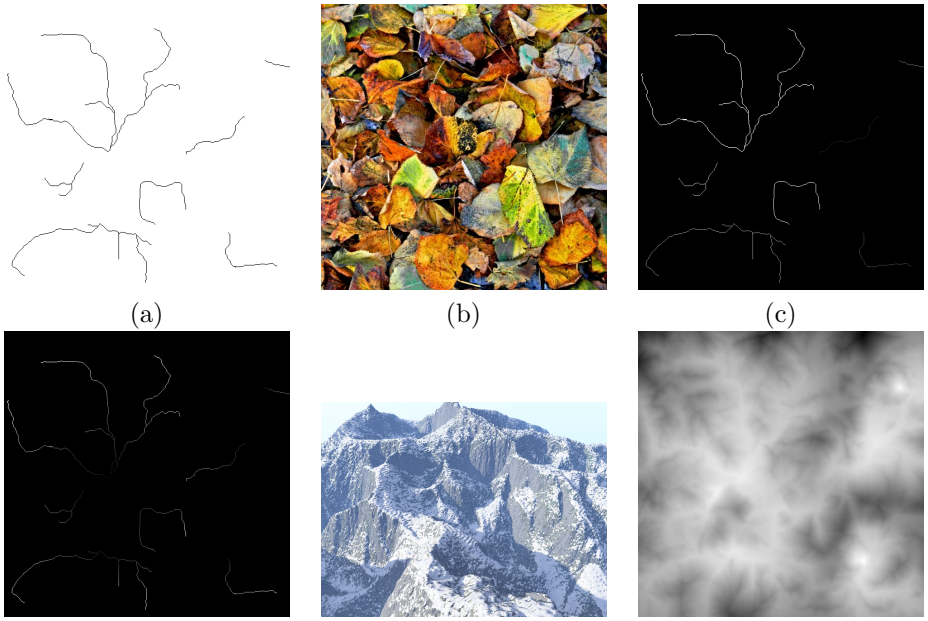


Figure A.15: Input and output for the edge weights result: (a) generator node locations F, (b) input texture [4], (c) feature ID, (d) generator node costs, (e) final render, and (f) final height field.

to Equation A.3, and is scaled within a range of $[0.25, 2.75]$. Feature identification is visualized in Figure A.16e and the costs of generators are visualized in Figure A.16f. The final render in Figure A.16g and height field in Figure A.16i result from these inputs, parameters, and processes. The synthesis completed in 1 minute 3 seconds.

A.3.4 Edge Detection and Edge Weights #2

In this result, generator node locations F are calculated by applying a 3×3 Sobel edge detector to Figure A.17b [25] and synthesizing 57 accompanying ridges. The threshold t_g on the gradient magnitude is calculated as per Equation A.2, and these edges are visualized in Figure A.17d. The resulting set of locations F is visualized in Figure A.17a. Edge weights are determined according to Figure A.17c [16]. Specifically, the gray value of each pixel in Figure A.17c is calculated according to Equation A.3, and is scaled within a range of $[0.25, 2.75]$. Feature identification is visualized in Figure A.17e and the costs of generators are visualized in Figure A.17f. The final render in Figure A.17g and height field in Figure A.17i result from these inputs, parameters, and processes. The synthesis completed in 2 minutes 8 seconds.

A.3.5 Edge Detection and Edge Weights #3

In this result, generator node locations F are calculated by applying a 3×3 Sobel edge detector to Figure A.18b [34] and synthesizing 86 accompanying ridges. The threshold t_g on the gradient magnitude is calculated as per Equation A.2, and these edges are visualized in Figure A.18d. The resulting set of locations F is visualized in Figure A.18a. Edge weights are determined according to Figure A.18c [16]. Specifically, the gray value of each pixel in Figure A.18c is calculated according to Equation A.3, and is scaled within a range of $[0.25, 2.75]$. Feature identification is visualized in Figure A.18e and the costs of generators are visualized in Figure A.18f. The final render in

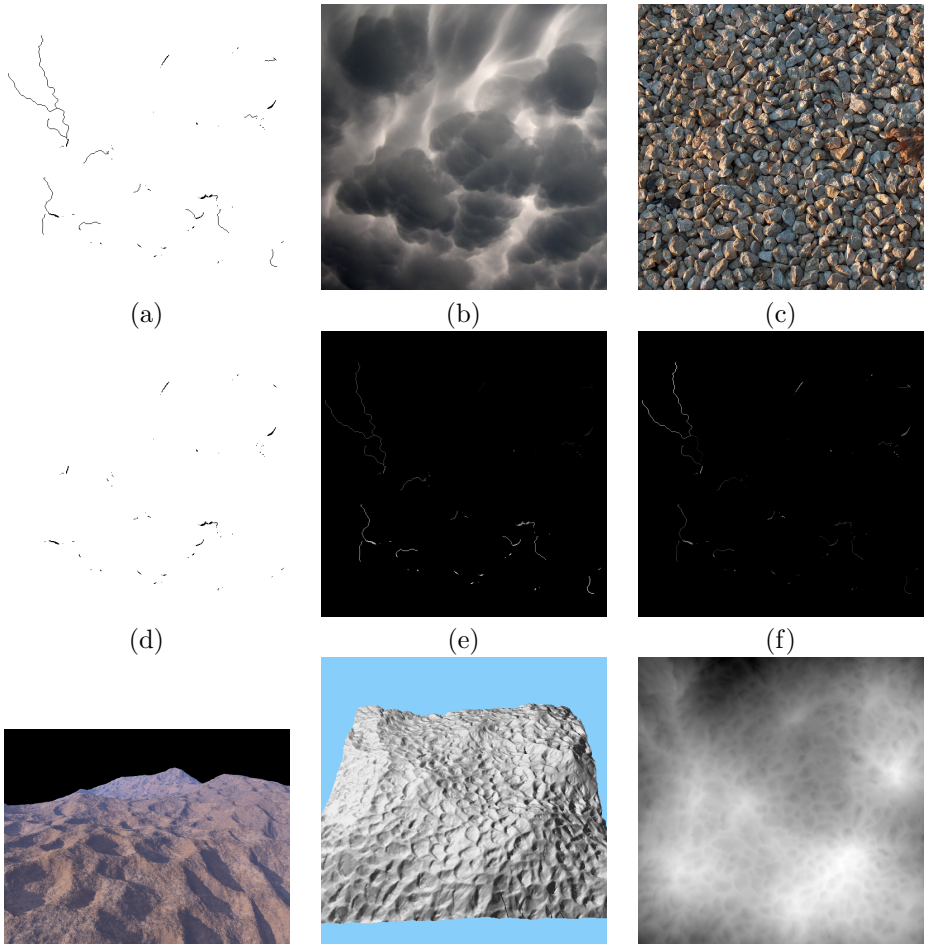


Figure A.16: Input and output for first ED/EW result: (a) generator node locations F, (b) edge detection texture [28], (c) edge weight texture [93], (d) detected edges, (e) feature ID, (f) generator node costs, (g) final Terragen render, (h) final OpenGL render, and (i) final height field.

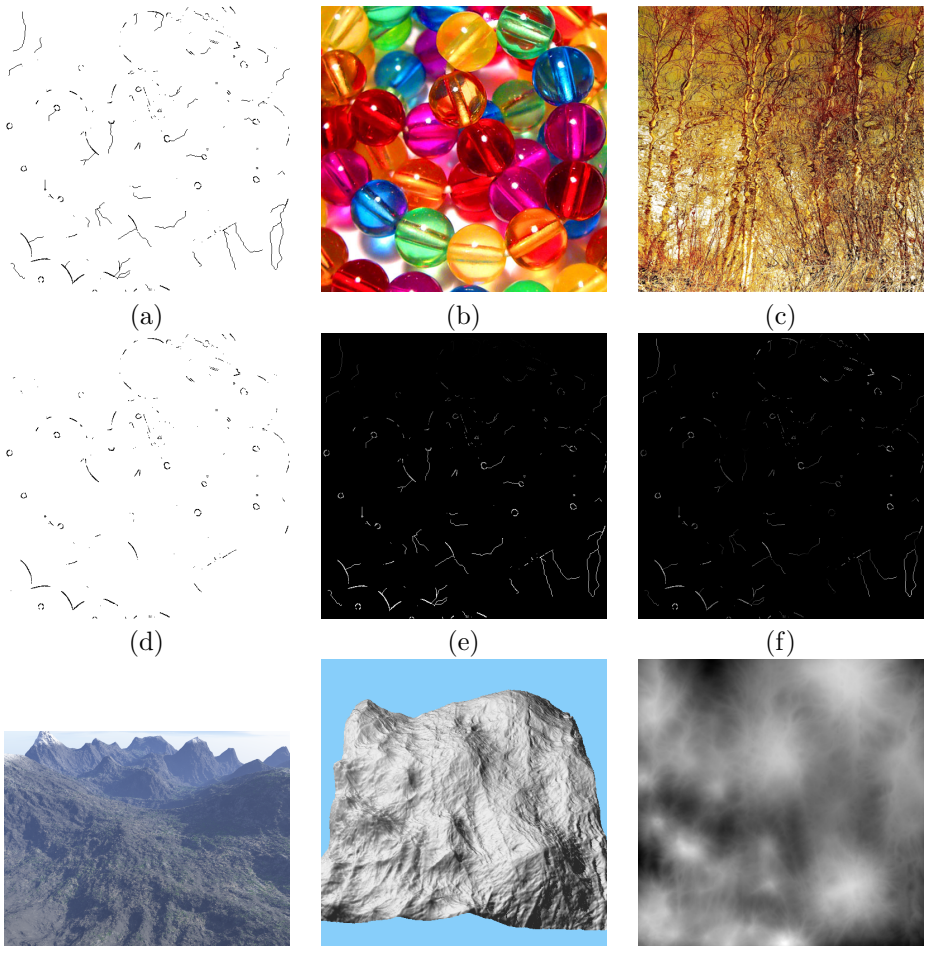


Figure A.17: Input and output for second ED/EW result: (a) generator node locations F, (b) edge detection texture [25], (c) edge weight texture [16], (d) detected edges, (e) feature ID, (f) generator node costs, (g) final Terragen render, (h) final OpenGL render, and (i) final height field.

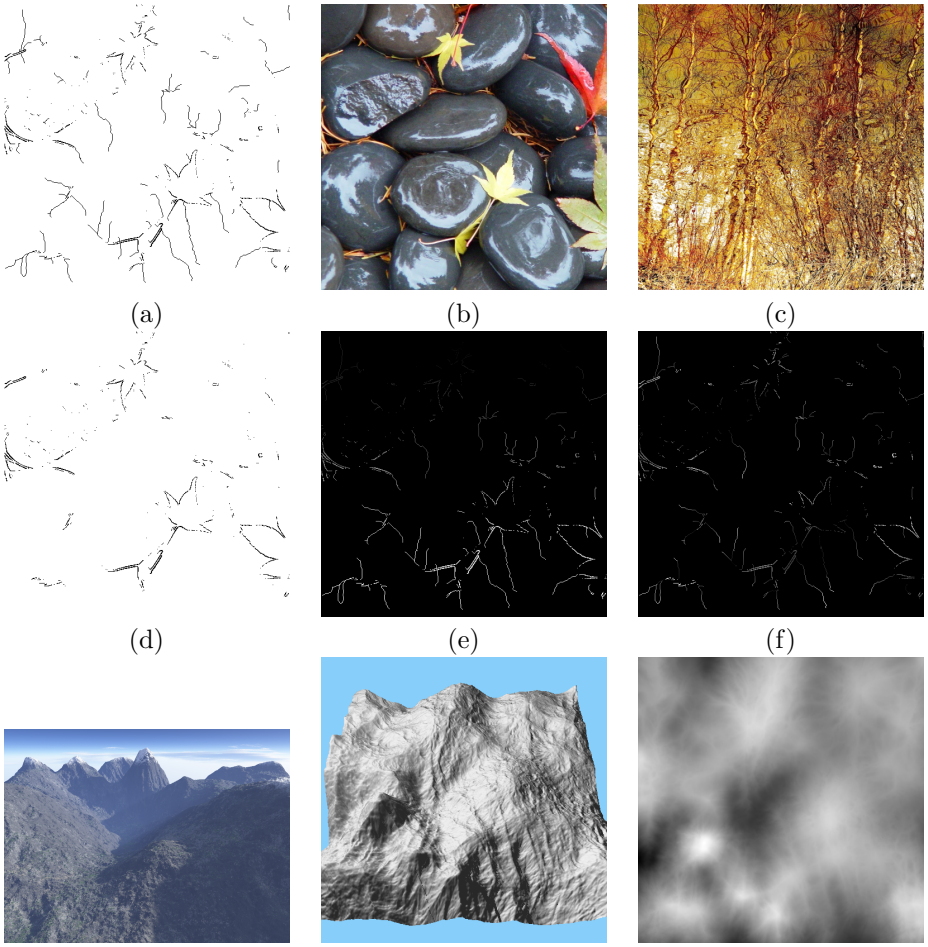


Figure A.18: Input and output for third ED/EW result: (a) generator node locations F, (b) edge detection texture [34], (c) edge weight texture [16], (d) detected edges, (e) feature ID, (f) generator node costs, (g) final Terragen render, (h) final OpenGL render, and (i) final height field.

Figure A.18g and height field in Figure A.18i result from these inputs, parameters, and processes. The synthesis completed in 2 minutes 15 seconds.

A.3.6 Edge Detection and Edge Weights #4

In this result, generator node locations F are calculated by applying a 3×3 Sobel edge detector to Figure A.19b [25] and synthesizing 57 accompanying ridges. The threshold t_g on the gradient magnitude is calculated as per Equation A.2, and these edges are visualized in Figure A.19d. The resulting set of locations F is visualized in Figure A.19a. Edge weights are determined according to Figure A.19c [39]. Specifically, the gray value of each pixel in Figure A.19c is calculated according to Equation A.3, and is scaled within a range of $[0.25, 2.75]$. Feature identification is visualized in Figure A.19e and the costs of generators are visualized in Figure A.19f. The final render in Figure A.19g and height field in Figure A.19i result from these inputs, parameters, and processes. The synthesis completed in 1 minute 15 seconds.

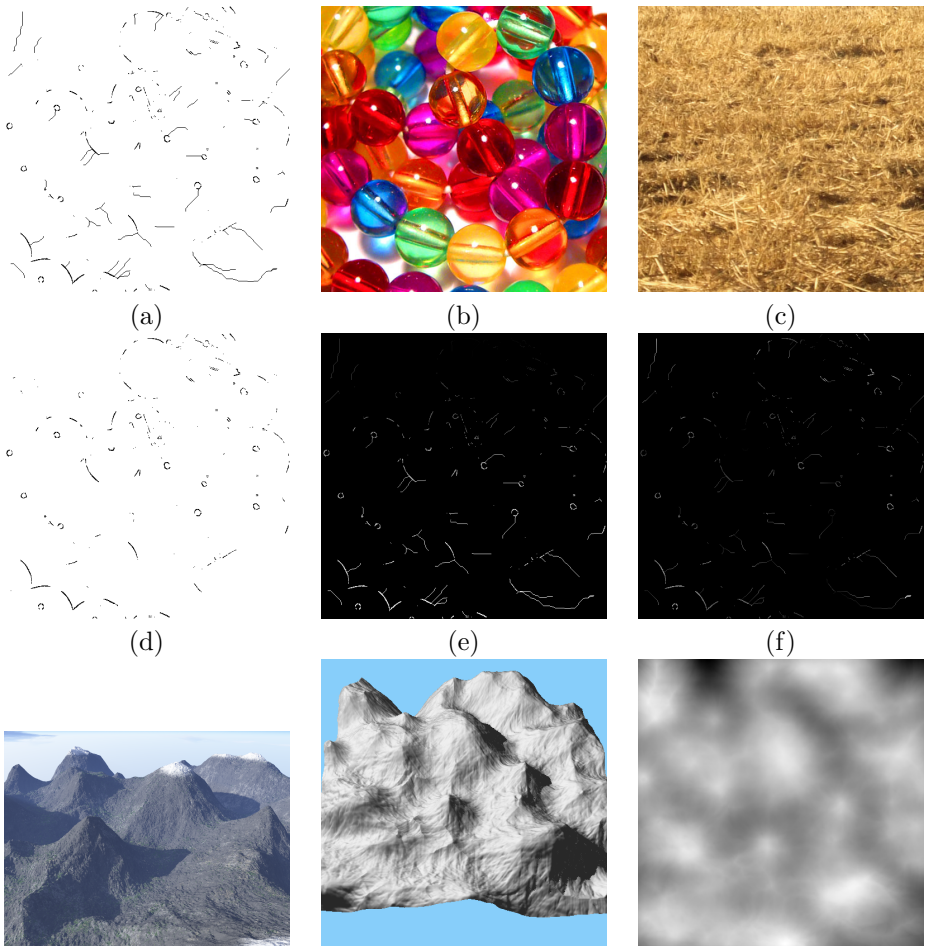


Figure A.19: Input and output for fourth ED/EW result: (a) generator node locations F, (b) edge detection texture [25], (c) edge weight texture [39], (d) detected edges, (e) feature ID, (f) generator node costs, (g) final Terragen render, (h) final OpenGL render, and (i) final height field.

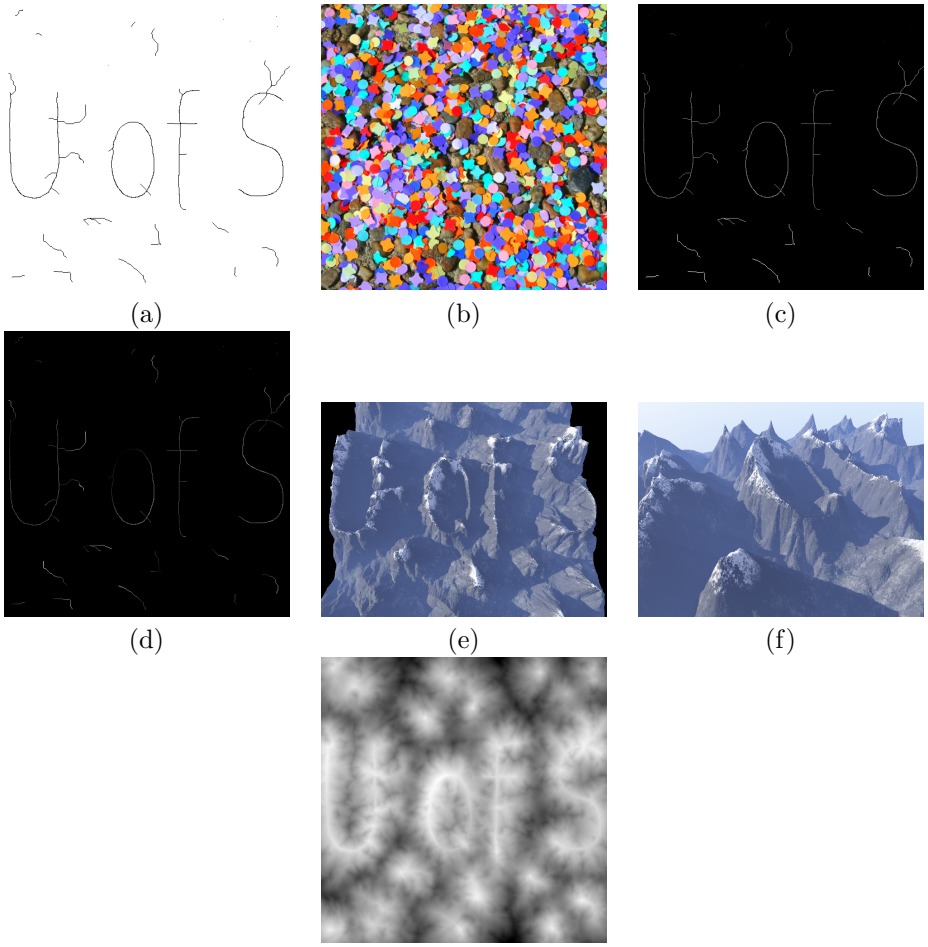


Figure A.20: Input and output for the "U of S" text result: (a) generator node locations F, (b) input texture for edge weights [66], (c) feature ID, (d) generator node costs, (e–f) final renders, and (g) final height field.

A.3.7 "U of S" Text

The "U of S" text result was synthesized in 9 seconds using $s = 2.5$, $b = 3$, and $f_t = 22$. The parameters μ_w and r are not used because an input texture provides the edge weights. Generator node locations F are provided in Figure A.20a and the image that provides the edge weights is given in Figure A.20b [66]. Specifically, the gray value at each pixel is scaled so that it is within a range of $[0.25, 2.0]$. Furthermore, the costs of all generator nodes are determined by scaling fBm within a cost range of $[0.0, 20.0]$. All ridge generator nodes have a cost equal to 35% of the existing node cost (recall that Dijkstra's algorithm is applied for ridge synthesis). These inputs and parameters create the final renders in Figures A.20(e–f).

A.3.8 IMG Logo

The IMG logo result was synthesized in 10 seconds using $s = 2.5$, $b = 3$, and $f_t = 16$. The parameters μ_w and r are not used because an input texture provides the edge weights. Generator node locations F are provided in Figure A.21a and the image that provides the edge weights is given in Figure A.21b [66]. Specifically, the gray value at each pixel is scaled so that it is within a range

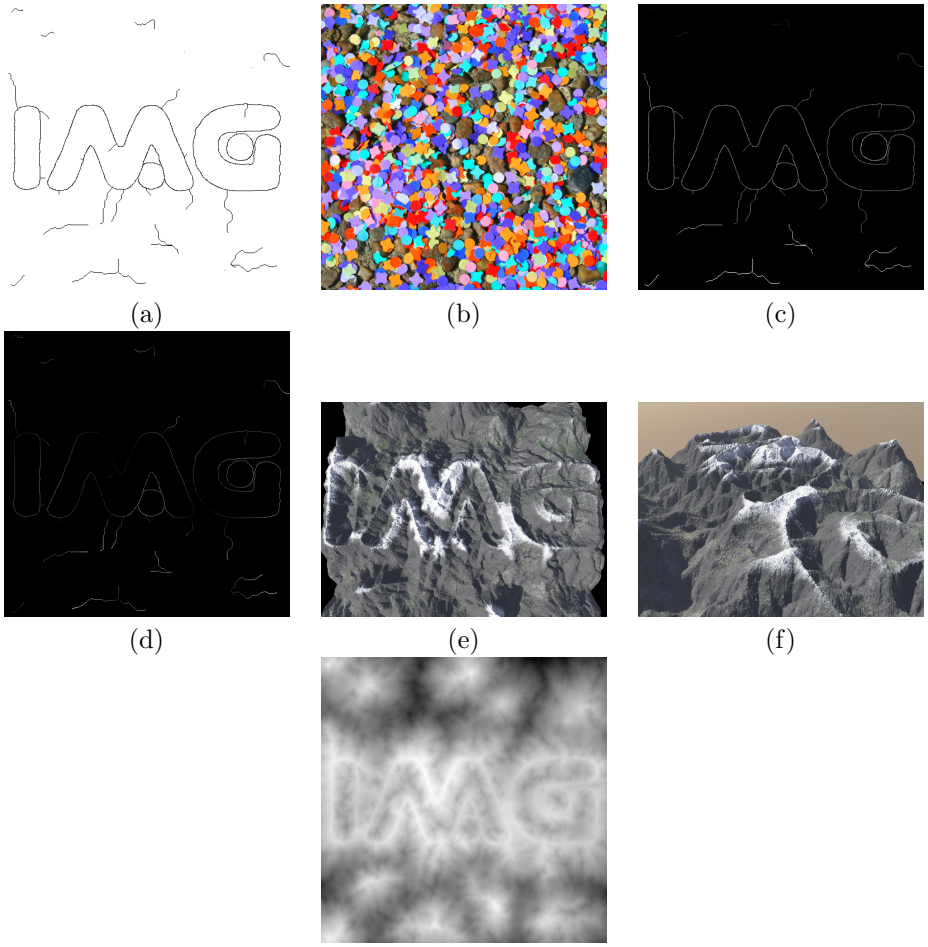


Figure A.21: Input and output for the IMG logo result: (a) generator node locations F, (b) input texture for edge weights [66], (c) feature ID, (d) generator node costs, (e–f) final renders, and (g) final height field.

of $[0.25, 2.0]$. Furthermore, the costs of all generator nodes are determined by scaling fBm within a cost range of $[0.0, 20.0]$. All ridge generator nodes have a cost equal to 35% of the existing node cost (recall that Dijkstra's algorithm is applied for ridge synthesis). These inputs and parameters create the final renders in Figures A.21(e–f).

A.3.9 λ Symbol

The λ symbol result was synthesized in 9 seconds using $s = 2.5$, $b = 3$, and $f_t = 3$. The parameters μ_w and r are not used because an input texture provides the edge weights. Generator node locations F are provided in Figure A.22a and the image that provides the edge weights is given in Figure A.22b [66]. Specifically, the gray value at each pixel is scaled so that it is within a range of $[0.25, 2.0]$. Furthermore, the costs of all generator nodes are determined by scaling fBm within a cost range of $[0.0, 20.0]$. All ridge generator nodes have a cost equal to 35% of the existing node cost (recall that Dijkstra's algorithm is applied for ridge synthesis). These inputs and parameters create the final renders in Figures A.22(e–f).

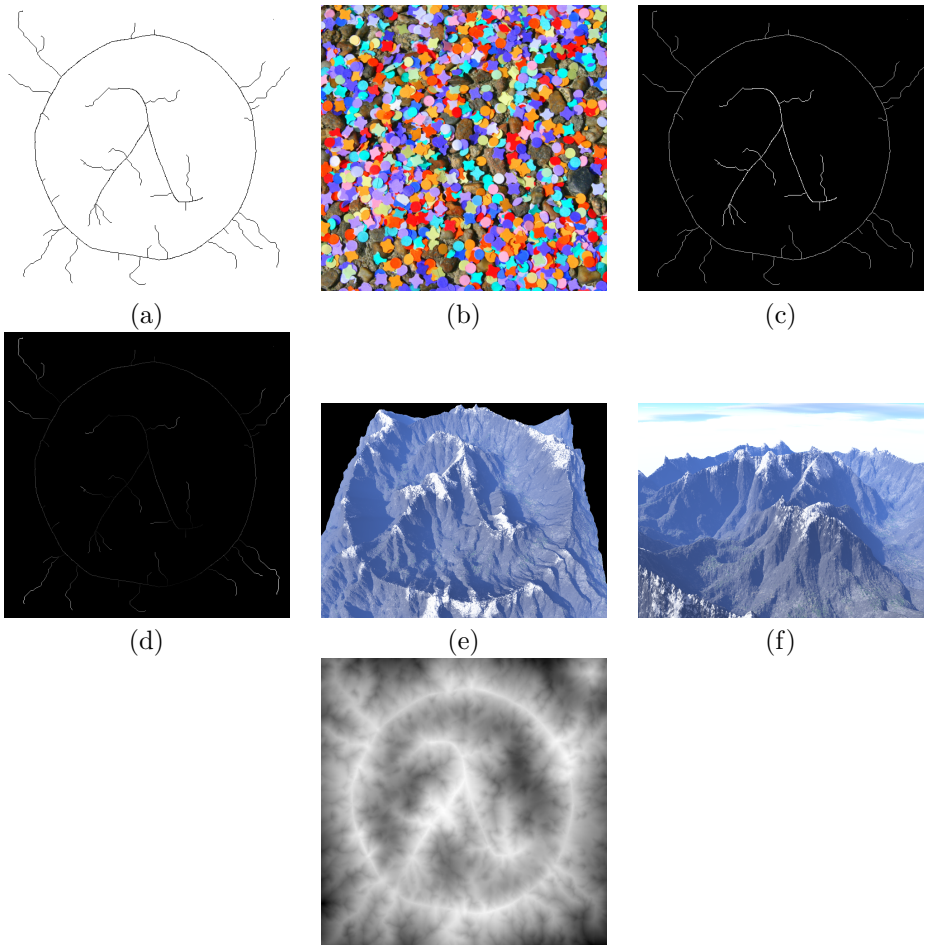


Figure A.22: Input and output for the λ symbol result: (a) generator node locations F, (b) input texture for edge weights [66], (c) feature ID, (d) generator node costs, (e–f) final renders, and (g) final height field.

A.4 Sketch-Based Terrains

A.4.1 Sketch-Based Terrain #1

The sketch-based #1 result was synthesized in 12 seconds using $\mu_w = 1.5$, $r = 0.75$, $s = 1.25$, $b = 3$, and $f_t = 31$. Generator node locations F are provided in Figure A.23a and were placed manually. Red pixels use the profile in Figure A.23b and black pixels correspond to Figure A.23c. The costs of the generators associated with these pixels are calculated using fBm (Octaves=6) sampled over a 12×12 grid; the resulting values are scaled by 40.0. Red pixels use fBm scaled within a percentage range of $[0\%, 35\%]$ of the maximum fBm cost. Black pixels use a percentage range of $[85\%, 100\%]$. The costs of all generator nodes are visualized in Figure A.23e. These inputs and parameters create the final render in Figure A.23f.

A.4.2 Sketch-Based Terrain #2

The sketch-based #2 result was synthesized in 9 seconds using $\mu_w = 1.5$, $r = 0.75$, $s = 1.1$, $b = 3$, and $f_t = 58$. Generator node locations F are provided in Figure A.24a and were placed manually. Black pixels use the profile in Figure A.24b, green pixels use Figure A.24c, red pixels use Figure A.24d, and blue pixels use Figure A.24e. The costs of the generators associated with these pixels are calculated using fBm (Octaves=6) sampled over a 12×12 grid; the resulting values are scaled by 48.0. Black pixels use fBm scaled within a percentage range of $[67.5\%, 85\%]$ of the maximum fBm cost. Green pixels are scaled within $[0\%, 30\%]$, red pixels are scaled within $[92.5\%, 97.5\%]$, and blue pixels are scaled within $[97.5\%, 100\%]$. The costs of all generator nodes are visualized in Figure A.24g. These inputs and parameters create the final render in Figure A.24h.

A.4.3 Sketch-Based Terrain #3

The sketch-based #3 result was synthesized in 11 seconds using $\mu_w = 2.0$, $r = 1.0$, $s = 1.01$, $b = 3$, and $f_t = 38$. Generator node locations F are provided in Figure A.25a and were placed manually. Black pixels use the profile in Figure A.25b, red pixels use Figure A.25c, and blue pixels use Figure A.25d. The costs of the generators associated with these pixels are calculated using fBm (Octaves=6) sampled over a 12×12 grid; the resulting values are scaled by 60.0. Black pixels use fBm scaled within a percentage range of $[0\%, 25\%]$ of the maximum fBm cost. Red pixels are scaled within $[65\%, 85\%]$, and blue pixels are scaled within $[85\%, 100\%]$. The costs of all generator nodes are visualized in Figure A.25f. These inputs and parameters create the final render in Figure A.25g.

A.4.4 Sketch-Based Terrain #4

The sketch-based #4 result was synthesized in 7 seconds using $\mu_w = 1.5$, $r = 0.75$, $s = 1.01$, $b = 3$, and $f_t = 15$. Generator node locations F are provided in Figure A.26a and were placed manually. Black pixels use the profile in Figure A.26b, red pixels use Figure A.26c, and blue pixels use Figure A.26d. The costs of black and red pixels are calculated as the distance to the center of the image scaled by 85.0. Blue pixel costs are calculated as the minimum of this distance based cost and a random value within the cost range of $[80.75, 85.0]$. The costs of all generator nodes are visualized in Figure A.26f. These inputs and parameters create the final render in Figure A.26g.

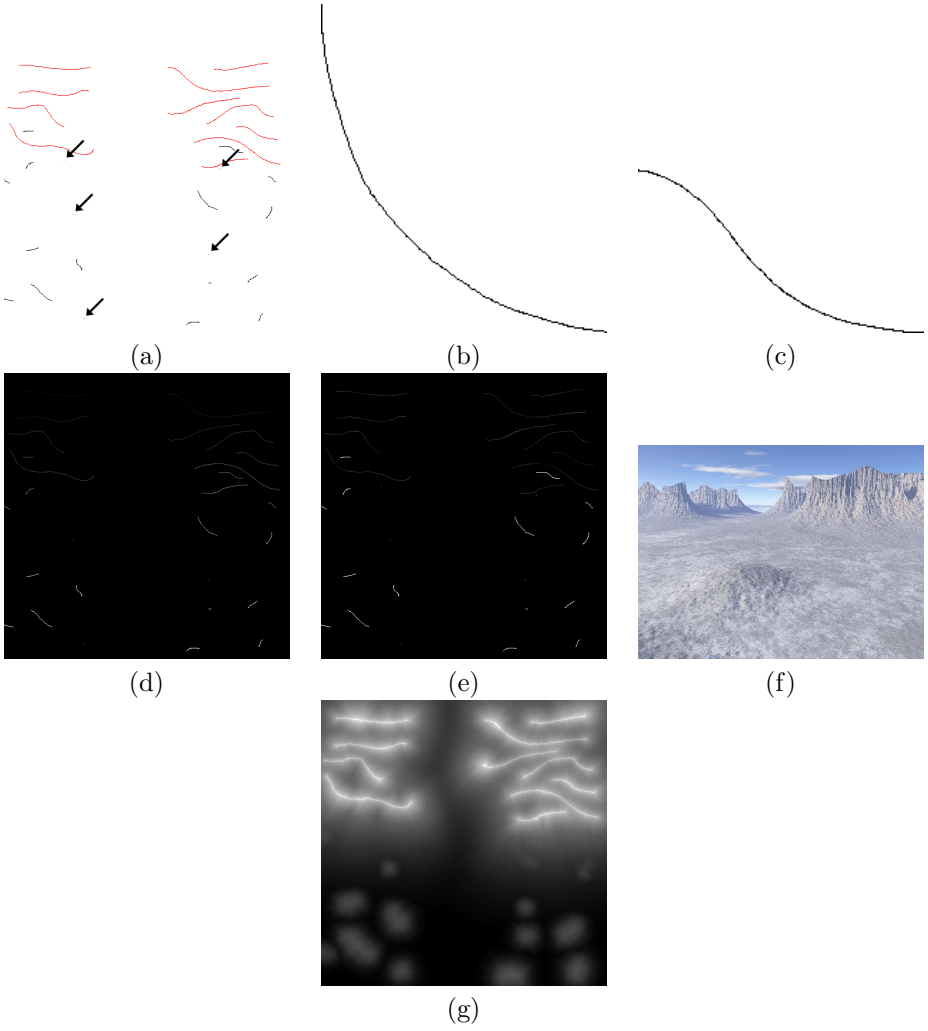


Figure A.23: Input and output for the sketch-based #1 result: (a) generator node locations F, (b–c) profiles P, (d) feature ID, (e) generator node costs, (f) final render, and (g) final height field. Arrows in (a) are not part of the input and are merely visual aids for single node generators which are otherwise difficult to see.

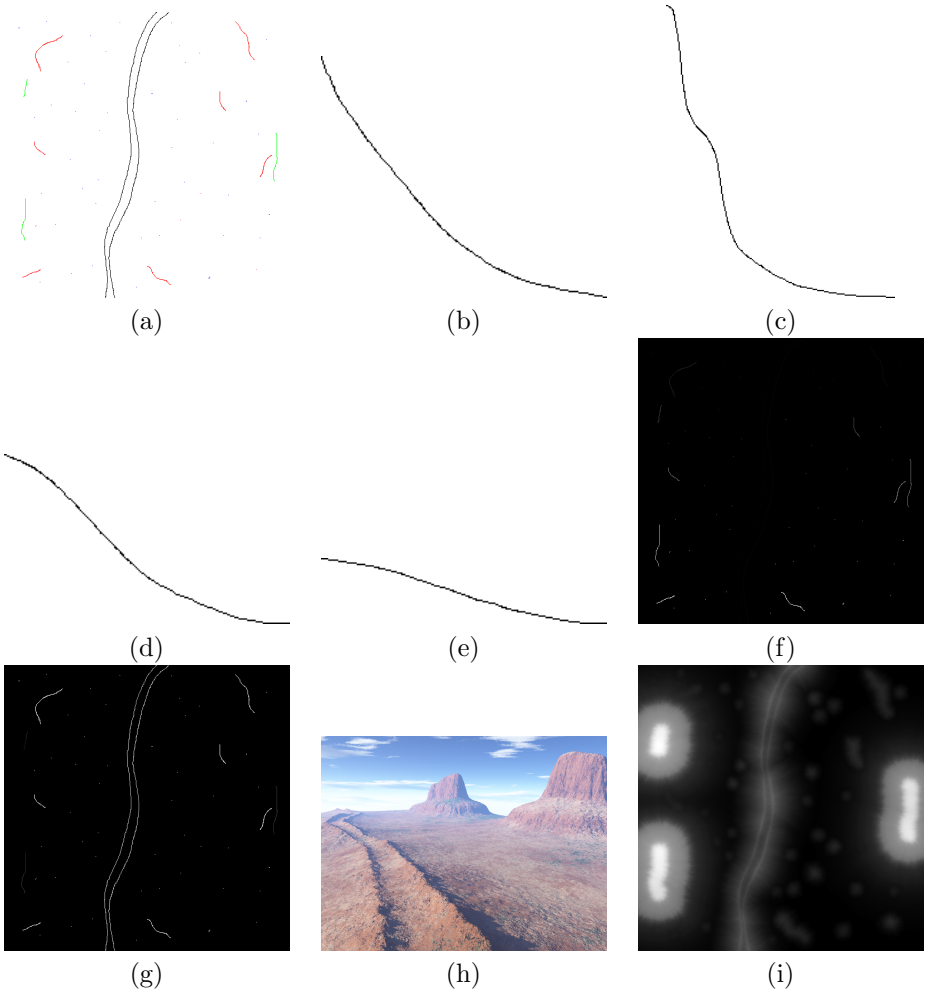


Figure A.24: Input and output for the sketch-based #2 result: (a) generator node locations F, (b–e) profiles P, (f) feature ID, (g) generator node costs, (h) final render, and (i) final height field. Image (a) contains multiple single node generators, but arrows are not used to allow for a clearer visualization of F.

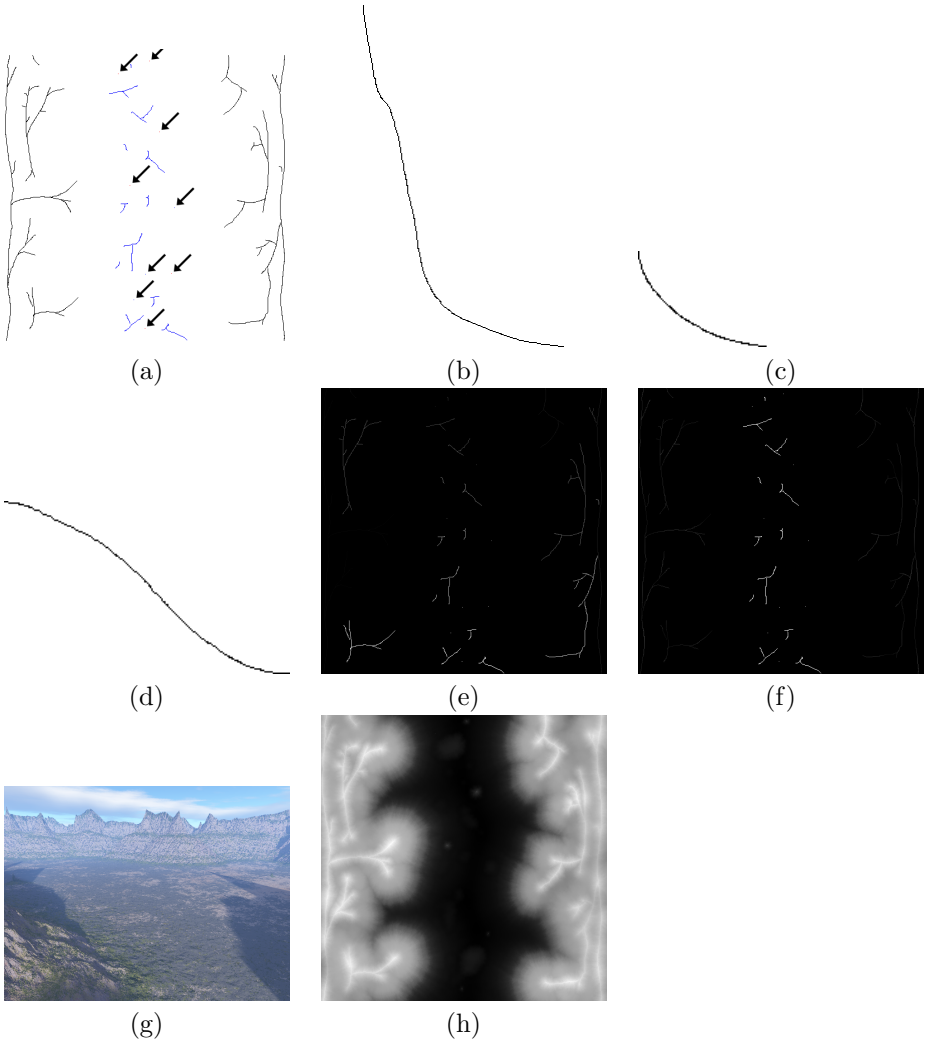


Figure A.25: Input and output for the sketch-based #3 result: (a) generator node locations F, (b–d) profiles P, (e) feature ID, (f) generator node costs, (g) final render, and (h) final height field. Arrows in (a) are not part of the input and are merely visual aids for single node generators which are otherwise difficult to see.

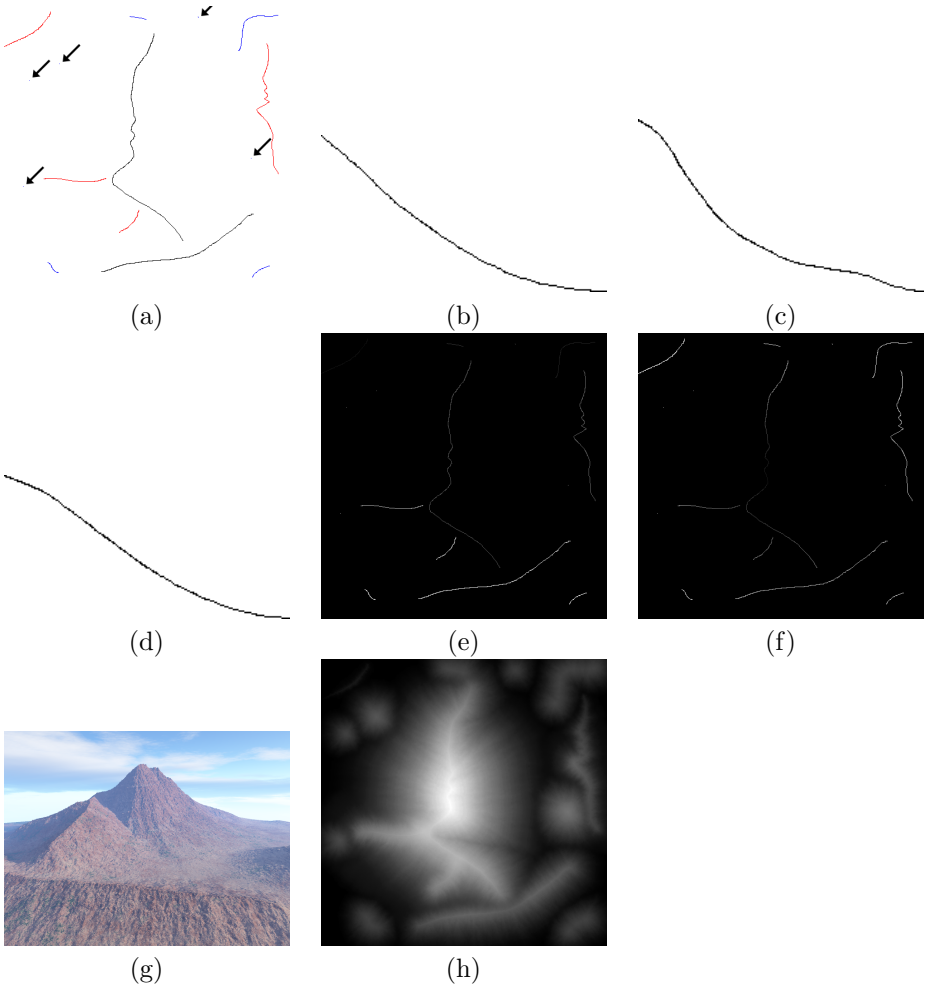


Figure A.26: Input and output for the sketch-based #4 result: (a) generator node locations F, (b–d) profiles P, (e) feature ID, (f) generator node costs, (g) final render, and (h) final height field. Arrows in (a) are not part of the input and are merely visual aids for single node generators which are otherwise difficult to see.