Controllable Procedural Terrain Synthesis using Curve Networks

by

Maryam Ariyan

A thesis submitted to the Faculty of Graduate and Postdoctoral Affairs
in partial fulfillment of the requirements for the degree of

Master
of
Computer Science

School of Computer Science
at

Carleton University

Ottawa, Ontario
January, 2015

Abstract

We present a controllable procedural technique for the synthesis of detailed terrains. We generate terrain based on a sparse curve network representation, where interconnected curves are distributed in the plane and can be procedurally assigned height. The user controls the placement and elevation of peaks. We employ path planning to procedurally generate irregular curves around peaks. Optionally, the user can specify base signals for the curve heights. Then we assign height to curves using biased random walks with controlled probability distributions, a process which we show produces signals with distinct shapes. The structure of a curve network partitions space into individual patches. We interpolate patch heights using mean value coordinates, after which we have a complete terrain heightfield.

Our algorithm enables users to get prominent features with lightweight interaction. Increasing the density of curves and roughness of curve elevation adds detail to the synthetic terrains. The curves in a network are organized into a hierarchy, where the major curves are created first and the secondary curves constructed at later stages are affected by changes made to the curves constructed at earlier stages. We use this hierarchy to control terrain elevation by modifying peak heights or curve shapes. We can locally evaluate the elevation in patches for obtaining arbitrary resolutions or in terrain editing. Our approach is capable of producing variety of landforms, with prominent ridges and distinct shapes.

This thesis is dedicated to the memory of my mother, Minoo Baghbanzadeh. My thoughts are with you every day.

Acknowledgments

I would like to state my sincerest gratitude to my Supervisor Dr. David Mould and thank him for his continued support and patience throughout this research. This work wouldn't have been possible without his guidance. Also, many thanks to my friends and family for their unconditional support and being so understanding during my studies. I would also like to thank GRAND for funding this work, as well as Carleton University for providing a great environment for study and research.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Terrains are visible in artistic paintings, computer games, and CGI movies. Traditionally, methods for terrain synthesis were controlled manually by the user. Modern examples of manual modeling tools are Maya [1] and Blender [4]. These manual tools provide complete control over the output model but demand a high degree of user involvement and expertise.

Procedural techniques employ algorithms to produce 3D models and textures with little or no input [11]. The characteristics of the resulting models are controlled with a set of parameters. Terragen [45] is commercial software with synthesis engines to generate terrain procedurally. In Terragen the user changes the shape of terrains by editing a set of parameters provided within the interface. The challenging aspect of procedural terrain synthesis is to control the result based on the input parameters.

Sketch-based modeling (SBM) methods construct 3D models based on freeform user sketches. SBM methods employ interpolation techniques to generate terrain surfaces [5,19, 23] and optionally add noise in a post-processing step. Sketch-based techniques demand less user involvement than manual methods and, compared to procedural methods, provide better control over terrain feature placement. However, the interpolated terrain surfaces exhibit less detail than the inherently rough-looking real-world terrains.

Early methods for terrain synthesis were mainly procedural, but, over time, hybrids of procedural and sketch-based techniques have become more popular. Purely procedural methods may have no input and generate models from random parameters. In sketch-based methods, a user explicitly defines terrain characteristics by sketching ridge and silhouette strokes. Olsen [39] presents a hybrid procedural and sketch-based technique and mentions that an ideal realistic terrain should exhibit low average slope and high standard deviation values. To retain terrain realism, Olsen interpolates the surface with a noise frequency

(a) Distinct silhoettes: photo taken by Flickr user *yedman* [66]

(b) Prominent detail along the ridges: photo taken by Frank Kovalchek [18]

(c) Rough surface detail: photo taken by Ken Lund [29]

Figure 1: Detailed features in real-world mountains

higher than the user-sketched contour strokes. Rusnell et al. [49] present another hybrid technique that defines the locations of terrain features such as peaks, ridges, and hills from an image map that contains sketched feature strokes. Then they interpret this input image as parameter constraints for a path planning procedural technique to generate terrain models.

Realistic terrains exhibit rough surfaces, prominent ridges, and distinct silhouettes. Figure 1 shows images of real-world mountain landscapes. These images show a couple of noteworthy details on terrain surfaces that we think are important for terrain modelers. The first and most important feature, which many of the current sketch-based techniques already cover, is control over the silhouette of terrains. Figure 1a shows the distinctive silhouettes of the Fitz Roy mountain. A modeler should be able to control the shape of height displacements along ridges and silhouettes. Figure 1b shows structural and detailed ridges next to the main peaks. Such details are cumbersome for a user to specify manually or by sketching. Figure 1c shows a distinct ridge profile along with rough surface detail throughout the mountain photo. The presence of prominent features pointed out through the images of Figure 1 are essential in realistic synthetic terrains.

(a) Reference landscape image **(b)** A detailed user sketch

Figure 2: A detailed user sketch contains a sparse representation of the terrain such as the position of peaks and valleys, the horizontal and vertical ridge profiles

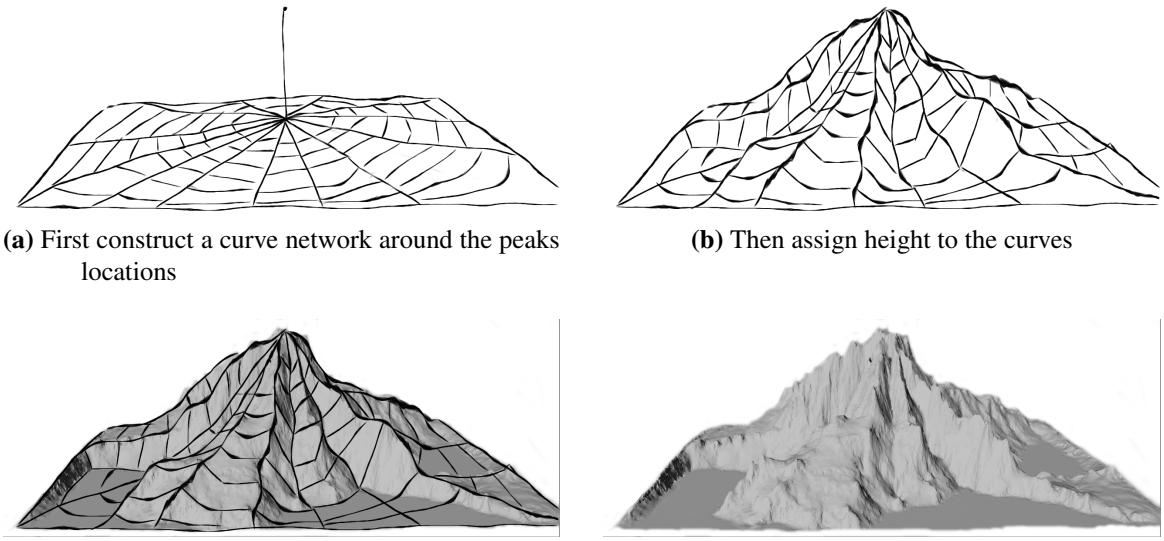
1.1 Problem statement

Our goal is to make controllable and realistic landforms with minimal user intervention. Terrains exhibit substantial detail. There is a trade-off between user effort and the presence of detail when modeling terrain using manual methods. We can reduce manual effort and get detailed models by using procedural methods, but we need a mechanism to gain high-level control over the terrain structure.

We can summarize the problem statement as the search for a controllable procedural method which computes a detailed synthetic terrain with prominent ridges. We consider ridges to be the main type of large-scale structures present in real terrains. We focus on finding a way to get structural detail while not imposing additional effort for the user or losing control of the placement of terrain features.

1.2 Overview of solution

To investigate the best approach for the synthesis of controllable yet detailed terrains, we conducted a qualitative experiment where we asked users to sketch the terrain landscape in Figure 2a. The goal was to see which parts of the landscape features seem to the user



(a) First construct a curve network around the peaks locations

(b) Then assign height to the curves

(c) Then interpolate patches bounded by curves

(d) The heightfield result

Figure 3: An overview of our terrain synthesis methodology

to be large-scale important characteristics. The users did not all provide the same level of detail in the sketch. Even in the sketched image of Figure 2b, which is among the most detailed sketches, the user did not complete all the ridges. The sketched result in Figure 2b exhibits vertical and horizontal ridges around the peak of the mountain. Figure 2b shows that detailed user sketches contain silhouettes, horizontal and vertical ridges, and the outline of valleys in terrains.

The method that we present controls the shape of height profiles such as the distinct silhouettes as seen in Figure 1a, procedurally generates structural details such as the vertical ridges in Figure 1b, and produces roughness along prominent ridges as seen in Figure 1c. Figure 3 illustrates an overview of our method for constructing heightfields from location of sparse input peaks. First we construct a closed network of curves around points with known elevation. Then, after estimating heights for all points on the curves, we interpolate patches bounded by curves.

Our method operates as follows: We take as input a scattered set of peaks with known elevation. Then we use a path planning approach to construct a network of curves in the area between the peaks. We procedurally compute the location of valleys between the peaks by extending paths from all the peaks and finding the locations where paths from different peaks meet. The horizontal curves in Figure 3 dictate the shape of terrain off the vertical curves. We create the vertical curves by distributing endpoints on the valley borders and

connecting paths from these endpoints to peaks in the vicinity of the valleys. We construct the horizontal curves, which are placed on the contours of a peak, by finding least-cost paths between pairs of vertical curves.

This network of curves provides a sparse spatial representation of terrain, where all the points are either part of a curve or inside patches between curves. Since the curves are created in a specific order, we get a hierarchy, which enables altering the structure of terrains by only changing the elevation of a sparse set of points in higher levels in the hierarchy. Based on this hierarchy, the elevation of valleys affects the height of the ridges on the curves connected to the modified valleys.

After we procedurally generate the (x, y) coordinates of the curve points in the network, we compute the heights of all the curve points. For each curve, we assign height profiles independently. We first calculate the elevation of valleys based on distances to closest peaks. Then we use 1D interpolation to compute the elevation of vertical curves which connect peaks to valleys. Then we compute the height of the remaining curves which connect to the vertical ridges using a 1D interpolation of endpoint heights. To add details to the terrain, we employ random walks to get a rough 1D signal for heights along the curves. In Chapter 3 present a novel technique to get biased random walks with distinct shapes.

We interpolate the elevation of the remaining points once we find the heights of all the curves in the network. The curve network partitions space into patches surrounded by parts of curves. Every point in a terrain is either on a curve or in an isolated patch enclosed by the curves. This composition enables us to employ mean value coordinates [16] to locally evaluate the height of each patch based on the height of curves surrounding the patch.

The curve network is a sparse map, which serves as a skeleton that controls terrain elevation. Curve networks can be used for locally editing terrain elevation by just changing the height of some points on the curves and re-interpolating only the patches bounded by the affected curves.

1.3 Statement of contributions

This work makes the following contributions:

- We present a hybrid procedural and sketch based modeling technique, which takes the placement and height of peaks and procedurally computes the elevation of the rest of the terrain. Optionally, the user can provide more detail such as a base shape

for the height of curves, and a parameter to control the density of the curves in the network.

- The network contains a large distribution of curves which is exhausting to set up manually. We create curve networks procedurally by employing multiple iterations of path planning to get a hierarchy of curves. The elevation of points in the plane can be controlled by the placement of a sparse set of points in higher levels in the hierarchy.

- A curve network is a sparse terrain representation and allows us to control elevation based on the height of peaks or procedurally generated valleys and ridges in the terrain. We can add details to the resulting terrains by increasing the density of curves. Also, since a curve network partitions space into isolated patches, we can locally evaluate the elevation in a patch for objectives such as obtaining arbitrary resolution or terrain editing.

- We present a method biased random walks with distinct shapes based on a user-sketched probability distribution. These biased random walks exhibit extremely detailed signals which we use for procedurally assigning height to curves. Anecdotally, the curves resemble terrain formations seen in the natural world.

1.4 Thesis Outline

Chapter 2 reviews previous work in terrain synthesis and the different trends introduced over time. We review related and previous work in the area of procedural-based terrain synthesis and then we review sketch-based techniques. Chapter 3 focuses on our novel terrain synthesis technique and begins with an overview of the method, followed by a thorough description of different elements of the algorithm.

In Chapter 4 we show several terrain models constructed from our method. We first explore the roles of different elements of our algorithm. Then we present variety of synthetic terrain landforms from our approach along with real-world photo examples. Then we compare our method with related terrain synthesis methods and analyze the performance of our algorithm. In Chapter 5 we summarize the main contributions, and then present ideas for future work.

Chapter 2

Terrain Synthesis: State of the Art

2.1 Overview

Early procedural modeling techniques modeled terrain using fractal shapes. Later, erosion-based methods were introduced to construct eroded terrains. Other types of procedural terrain synthesis techniques are example-based methods, which produce synthetic terrain by resampling features visible in input elevation data, and distance-based methods, which compute terrain height based on distances from a set of source points. Sketch-based techniques generate synthetic terrain models from sparse input such as sketched ridges and silhouettes.

This chapter provides a review of previous work in terrain synthesis and the different trends introduced over time. First we review different types of terrain representations. Then we describe procedural synthesis techniques, followed by a discussion of sketch-based methods. We conclude with a summary of advantages and drawbacks of the explained terrain synthesis techniques.

2.1.1 Terrain representations

A heightfield [35] is the most common data representation used for terrains. A heightfield is a 2D matrix of elements such that each cell contains the height of a point above a 2D plane. Heightfield data structures represent the surface layer of terrain and the entire heightfield contains only one layer of material.

Benes et al. [2] introduced a layered data representation that provides a heightfield surface along with a voxel layer on each interpolated point. Voxels are the 3D equivalent of 2D pixels and carry height values per element. Layered data representation is a modified

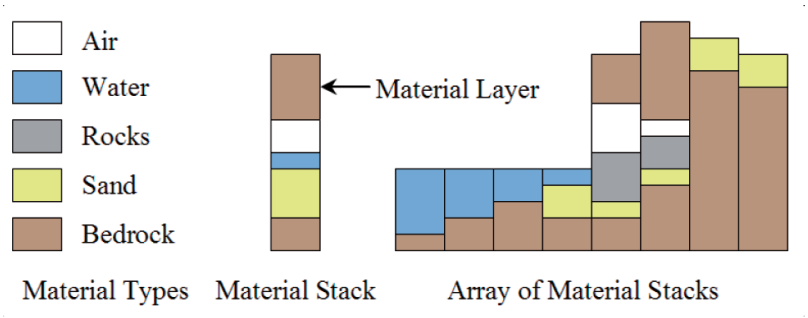


Figure 4: Overview of the material layers and material stack data structures used by Peytavie et al. for modeling arches [43]

type of heightfield. Each layer represents a different material. The layers are summed up to calculate the final elevation value per cell. Stava et al. [55] employ the layered data structure for editing and GPU implementation of terrain.

Heightfields cannot represent all types of terrain. Features such as arches and caves require a data structure that can provide more than one height value on each point of the 2D plane. Peytavie et al. [43] use a modified two-dimensional grid of stacked material to model arches. Figure 4 illustrates the material layers and material stack data structures.

Some researchers in terrain synthesis have employed voxels, meshes, and tree data structures. Valette et al. [61] use voxels to create a model for the formation of cracks. Fournier et al. [17] employ meshes to model terrain using an adaptive midpoint displacement algorithm which we discuss in Section 2.2.2. Genevaux et al. [20] use a tree data structure to represent terrain landscapes in an erosion-based technique that simulates hydraulic erosion. The tree leaves store terrain patches and the inner nodes carry operations such as blending, adding, or carving over the leaf nodes.

We described different types of data structures used for terrain representation in order of popularity. The most common data structure used amongst procedural and sketch-based methods are heightfields. Heightfields cannot represent all types of terrain. However, a heightfield is memory efficient compared to the rest of the data structures. We employ heightfields in our terrain synthesis approach, as described in Chapter 3.

2.2 Procedural techniques

Procedural methods are popular because of their ability to create detailed models of natural irregular objects and phenomena, including terrain. In this section first we discuss procedural noise synthesis techniques which have been used in terrain modeling techniques. Then we review fractal-based techniques which are the oldest procedural methods used for terrain synthesis. Next we describe erosion-based modeling techniques which construct eroded terrains by procedural methods such as simulation of erosion or construction of river networks. Then we review example-based techniques which employ texture synthesis methods to get synthetic terrain from a small input terrain texture. Lastly we describe distance-based methods which compute heightfields based on distances to a sparse set of input feature points.

2.2.1 Noise-based synthesis

Three procedural techniques for noise synthesis are Perlin noise [42], Worley noise [64], and random walk. Perlin and Worley noise have been used in texture synthesis and modeling of natural phenomena such as in terrain synthesis. In the terrain synthesis literature random walks have been used to generate river networks [28].

We can evaluate a noise value for points in 3D space using the Perlin noise [42]. For each discretized point in space, random orientations are set and the intermediate values are interpolated using splines. Perlin noise produces stationary and nearly isotropic noise signals. In computer graphics, Perlin noise has many applications, from stochastic solid texturing to adding small-scale detail to models of natural phenomena such as clouds [54] and terrains [23].

Worley noise [64] is a noise function which takes random source points in 3D space and then computes a noise value for all the points as a function of distances to the n-th closest source point in space. The Worley function is in the following format:

$$W(x) = \sum_{i=1}^{n} c_i \cdot d_i(x)$$

where $W(x)$ is the noise function for point x in the plane, $d_i(x)$ is distance from x to the i-th closest source, and each c_i is a constant. Different sets of constants define different distinct textures. Worley noise creates cellular textures such as animal patterns and rocks.

According to Bunde and Havlin [7], an unbiased random walk is a signal generated by

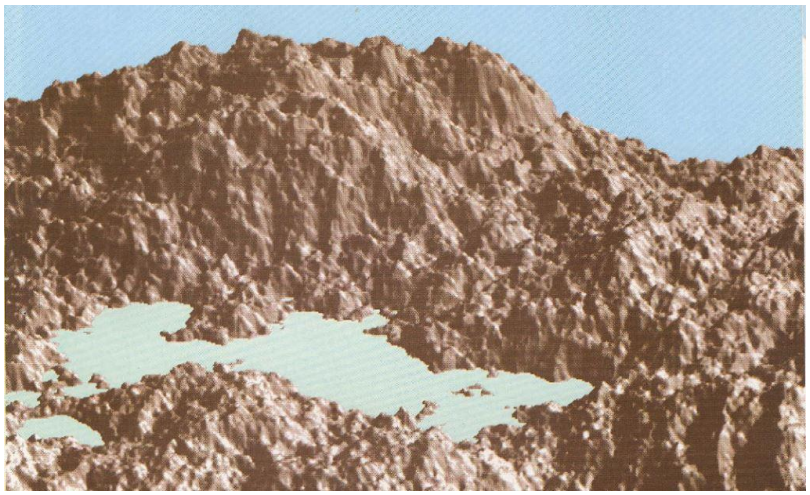


Figure 5: Fractal terrain generated by Mandelbrot [31]

recording the displacement of a drunken walk with a discrete probability of either $+1$ or -1 at each time step [7]. The signals generated with this drunkard's walk have a 50% likelihood of a positive increment and a 50% chance of a negative increment. In Chapter 3 we construct distinct shapes of random walks by selecting random increment values according to specific probability distributions.

2.2.2 Fractal-based methods

Fractal shapes exhibit self-similarity at all scales. Fractals have been used to represent natural irregular phenomena that contain details at different scales, such as clouds and terrain [11]. Mandelbrot was the first to study fractal shapes and described fractional Brownian motion (fBm) as a technique for getting fractal shapes [31, 33]. Figure 5 shows a fractal landscape from Mandelbrot [31]. FBm is a random process that creates fractals. The random value at each point of a fractal shape is computed by summing up signals with different frequencies.

Signals at each frequency of a random fractal can be generated with noise synthesis. Musgrave et al. [35] create fractal-based terrain by employing Perlin noise as the basis for generating fBm signals. They interpreted the spatial displacement in fBm signals as height and explored ways to control the fractal shape of terrains by a set of procedural parameters. For each point on the terrain, the process is controlled by three parameters: the fractal increment parameter H which controls the smoothness of fBm signals, octaves which is

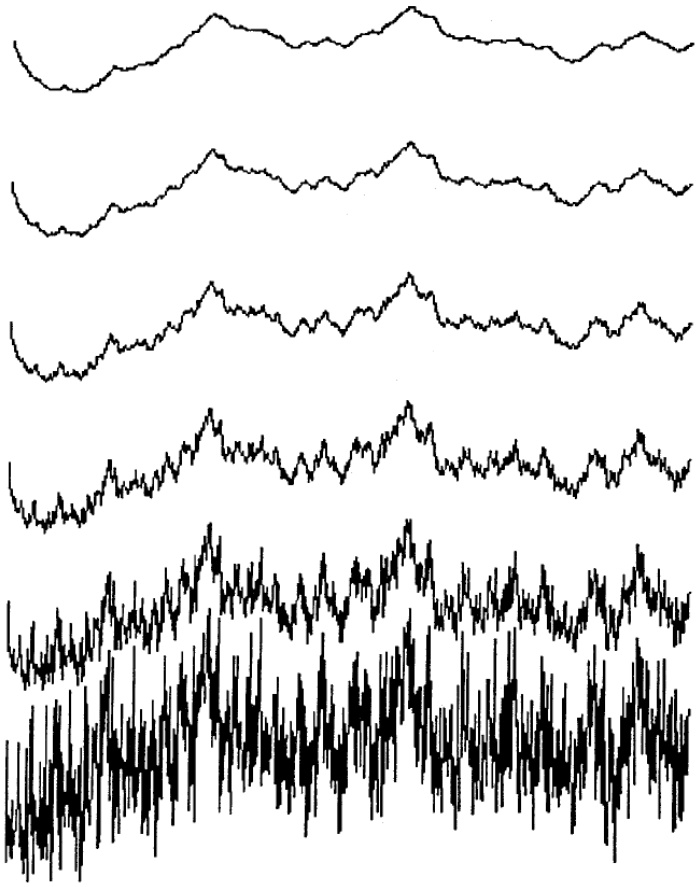
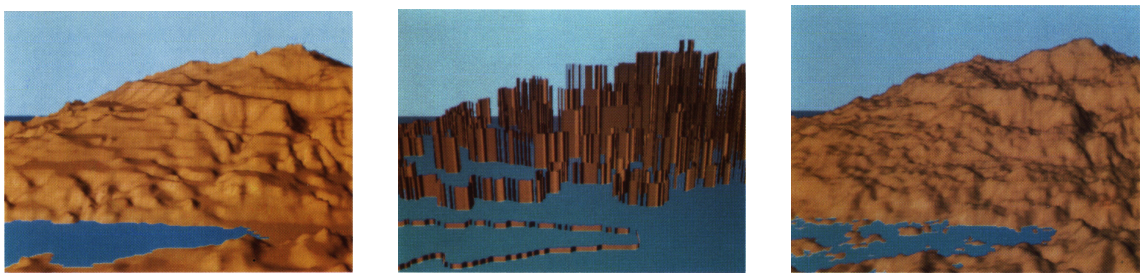


Figure 6: The effect of H on the fractal shape, from values 1.0 to 0.0 in decrements of 0.2; image taken from Ebert et al. [11]

the number of frequencies summed-up in an fBm signal, and Lacunarity L which is the gap of frequency between successive octaves. The fBm equation to compute height values is as follows:

$$h(x) = \sum_{i=1}^{n} N(x \cdot \gamma^{H \cdot i})/L^{i} \tag{1}$$

where n is the number of octaves and $N(x)$ is the Perlin noise function. As i increases, amplitude is reduced by a factor of L and frequency increases by a factor $\gamma^{H \cdot i}$. The fBm signal gets smoother as H approaches 1. Figure 6 illustrates the effect of H on the fractal shapes. In Chapter 3 we use Equation 1 to generate terrains, replacing Perlin noise with a different noise function which is based on a random walk.



(a) Input elevation data **(b)** Contours **(c)** Result

Figure 7: Szeliski and Terzopoulos [57] construct terrain by interpolation of select contours from input elevation data and fractal noise generation.

Fournier et al. [17] presented an adaptive midpoint displacement algorithm to approximate fractal noise for the synthesis of stochastic models. To do this, Fournier et al. recursively subdivide triangles by randomly selecting and then displacing midpoint height values. Such polygonal subdivision techniques are easy to implement and fast but produce undesirable crease artifacts. Lewis [30] presents a refinement of random midpoint displacement and reduce the crease artifacts by adding correlated Gaussian noise instead of displacing each midpoint independently. The approach of Lewis produces high frequency noise and has general control over shape but is computationally expensive.

Szeliski and Terzopoulos [57] employ a relaxation algorithm which can take arbitrary constraints in its energy minimization function to generate constrained fractals. They take a sparse set of known elevation values such as the one shown in Figure 7b, and then determine the remaining elevation using spline interpolation. Their work was a step forward towards feature-based terrain generation but the process, which they use to locally reduce the smoothness of fractals, is difficult to control.

Fractal modeling is an efficient way to procedurally generate terrain but lacks realism because it is homogeneous and isotropic which is not the case for real terrains. Musgrave [34] explains that hilltops are more rounded than valleys and valleys get filled up with material and are shallower than the surroundings. He also points out that fBm terrains lack river networks and do not exhibit erosional features. Based on these observations Musgrave introduced a mechanism for adding erosional factors to terrain models. He creates heterogeneous terrain models on top of fractal-based terrains by using thermal weathering and fluvial erosion processes. His work guided research into procedural synthesis of erosion-based terrains which we discuss next.

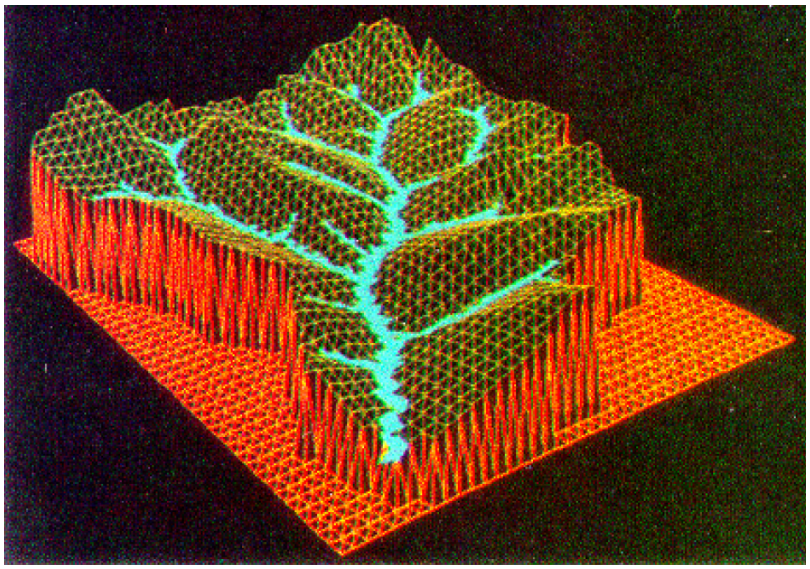


Figure 8: Terrain synthesis based on the placement of watersheds [28]

2.2.3 Erosion-based methods

Fractal terrains carry isotropic and homogeneous characteristics which are unsuitable for the synthesis of eroded terrain surfaces. Also, maintaining rivers with purely fractal-based techniques is a difficult task since realistic looking river networks need to flow from higher to lower surfaces. Kelley et al. [28] were the first to procedurally synthesize terrain with watersheds. They first generate a 3D watershed network and then interpolate the remaining points to generate hills. In this approach tributaries with different scales connect to a watershed network. Figure 8 shows a synthetic terrain by Kelley et al. based on a watershed network which is comprised of a stream and its tributaries.

Musgrave et al. [35] proposed an alternative approach to creating river channels by simulating water erosion in fractal terrains. They simulate river networks in a process which divides modeling into first terrain generation and then erosion simulation. Musgrave et al. demonstrate a noise synthesis method based on Perlin noise for creating fractal terrains. The erosion process is subdivided into a thermal and hydraulic process.

Hydraulic erosion involves depositing water (rain) on vertices of the height field and allowing the water and sediment suspended in the water to move to any lower neighboring vertices. Hydraulic erosion produces valleys and drainage networks. Thermal weathering on the other hand, creates talus slopes of uniform angles with a relaxation process. Talus slope is the large-scale shallow slope at the end of valleys on eroded terrain surfaces which

is fill-in from rocks falling down from higher elevations. Thermal weathering wears down steep slopes and creates talus slopes at their feet. Musgrave et al. apply gravitational water diffusion to simulate hydraulic erosion and then apply thermal weathering to simulate soil transportation.

The work of Musgrave et al. [35] started a trend for synthesis of eroded terrains. Sapozhnikov [50] used a random walk to generate a set of fractal river networks of various sizes. Prusinkiewicz and Hammel [46] used midpoint displacement methods with a non-branching river generation method by Mandelbrot [32] to create mountain landscapes with rivers. However, the synthetic terrains of Prusinkiewicz and Hammel exhibit asymmetric valleys and lack tributaries. Nagashima [36] eroded valleys with an ad-hoc approach rather than using physically-based simulation, generating a fractal terrain and then applying erosion of river flow, waterfall, and weathering on top of it. Doran and Parberry [9] employ software agents to generate landscapes. They construct a complete terrain in three phases: coastline, landform, and erosion. Each stage is completed using a specific software agent. Benes et al. [3] presented another erosion-based approach that simulates hydraulic erosion. Their work provides control over the resulting features but follows a slow process. Later Stava et al. [55] presented an interactive physically-based erosion method. This method simulates the movements of river flow and transportation of rock based on the physics of hydraulic erosion. All the erosion-based methods that we presented in this section include erosion factors for generating realistic terrain and add realism by including drainage networks for the simulation of hydraulic erosion. The approaches of Peytavie et al. [20,43,44], Teoh [59], and Kelley et al. [28] first generate river networks and based on that create the complete terrain but the rest have an opposite order. The main problem in the erosion-based methods is that the outcome is difficult to control. Also, the methods using thermal erosion suffer from performance issues.

2.2.4 Example-based methods

Example-based methods generate terrain based on a sample input terrain texture. The input to example-based terrain synthesis methods is real-world data such as Digital Elevation Models (DEM). Example-based terrain generation methods are adapted from texture synthesis methods which generate new textures which resemble patterns from a sample input texture.

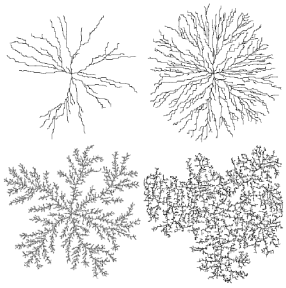
Efros and Leung [13] synthesize new texture by resampling local structure present in input images. They present an algorithm that synthesizes an image one pixel at a time.

Figure 9: Brosz et al. [6] employ the user-created base terrain in the left along with the reference target terrain in the middle to construct the example-based terrain in the right image

At each stage, they find all neighborhoods in the input texture which are similar to the neighborhood of a pixel, and then randomly select one of these neighborhoods; the center pixel is copied to the output texture as a newly synthesized pixel. The work of Efros and Leung is computationally expensive. Wei and Levoy [62] presented a faster algorithm that employs an acceleration technique for finding neighborhoods similar to the neighborhood of the pixel being processed. Once a matching neighborhood is found, they can extend their texture by one pixel by copying the relevant pixel from the input sample. Both the approaches of Efros and Leung [13] and of Wei and Levoy [62] operate on a pixel-by-pixel basis. Choosing a small mask size makes the algorithms more vulnerable to choosing wrong neighborhood area, thus producing output that would no longer resemble the input image sample. To address this shortcoming, Efros and Freeman [12] introduce a patch-based technique. After searching the sample texture for neighborhoods, Efros and Freeman merge patches rather than pixels to the output texture. This patch-based approach has better computational performance and improved stability in generating correct texture results.

Brosz et al. [6] construct heterogeneous terrain by using the texture synthesis method of Efros and Freeman. Brosz et al. use a base terrain and a target terrain to produce a final synthetic result. As Figure 9 shows, the base terrain contains user-created large-scale terrain features and the target terrain provides small-scale terrain details which can be taken from DEMs. Brosz et al. first construct a map of best matches between target and base, then refine the base terrain to match the map constraints. This method extracts and combines details with different resolutions in base and target terrain. Later Zhou et al. [67] present another example-based approach based on an input texture from DEMs. They combine a DEM heightfield with a sketch-map of the user's desired terrain features. Zhou et al. present the first terrain synthesis method that provides control over large-scale

(a) Rusnell et al. [49] use path planning to procedurally generate rough terrain



(b) Xu and Mould [65] use path planning to model dendritic shapes

Figure 10: Application of path planning in the modeling of irregular phenomena

terrain feature placement using sketched strokes. The main drawback in the work of both Zhou et al. and Brosz et al. is that the diversity of terrain styles depends on the databases of input DEMs. These methods produce high-quality results, but depend on a database of terrain features.

2.2.5 Distance-based methods

Distance-based terrain synthesis methods interpret heights from a set of input feature points as function of distances. Worley [64] presented a noise function that, given a set of randomly positioned feature points, partitions space into array of cells such that each contain one feature point. Worley noise produces textures that resemble terrain features such as mountain ranges and craters. Peytavie et al. [44] use a distance-based approach for constructing piles of rock by smoothly eroding the boundary of Voronoi cells while preserving the contact points between neighbor cells. This technique is capable of modeling different rock types, from sharp gravel to round pebbles and is fast enough for real-time modeling of scenery.

Path planning is the task of finding least-cost paths through a weighted graph [63]. Xu and Mould [65] generate dendritic paths by employing multiple passes of Dijkstra's path planning algorithm. Figure 10b shows four dendritic shapes from Xu and Mould. Rusnell et al. [49] employ path planning for terrain synthesis to find distances to a set of input feature nodes with known elevation in a weighted graph and then assign heights as a function of distances to the feature points. Algorithm 1 is taken from the work of Rusnell [48].

Algorithm 1 Terrain synthesis based on least-cost paths to feature points with known height and location indicated as source nodes in the connected graph $G(N, E)$ [48]

function DIJKSTRA
    $T \leftarrow$ empty min heap
    for all $n \subseteq N$ in graph $G(N, E)$ **do**
        if n is a source node **then**
            $c_n \leftarrow h_{max} - h_n$
            Push node n into heap T
        else
            $c_n \leftarrow \infty$
        end if
    end for
    while T not empty **do**
        Pop node n_0 with minimum cost from heap T
        for all edge e incident to n_0 **do**
            $c \leftarrow c_{n_0} + w_e$
            $n_e \leftarrow$ node connected to n_0 via e
            if $c < c_{n_e}$ **then**
                $c_{n_e} \leftarrow c$
                push n_e into heap
            end if
        end for
    end while
end function

function GETHEIGHTS
    for all $n \subseteq N$ in graph $G(N, E)$ **do**
        $h_n = h_{max} - c_n$
    end for
end function

Figure 10a shows a synthetic mountain range from Rusnell.

The work of Rusnell [48] provides user control over terrain feature placement and extrapolates height values outwards from feature points. In this thesis we present a distance-based algorithm motivated by Algorithm 1. Our algorithm first generates a network of closed curves around input peak locations based on path planning and then estimates elevation of points on curves as a function of distances from peak locations. In this process we add procedural detail along the ridges outwards from user-specified peaks and then we interpolate the remaining points.

2.3 Sketch-based techniques

Sketch-based interfaces for modeling (SBIM) follow a three-step pipeline: sketch acquisition, filtering, and interpretation [40]. In the first step, the user sketches an input. Then the filtering step involves cleaning or transforming the input into a complete sketch. Then the final step of the pipeline passes the sketch as input to an operation which constructs a 2D shape or 3D model. Olsen et al. [40] discuss the role of perception in sketching interfaces. When a user sketches the contours of an object, the imagined view of the user for that object is unlikely to be an "accidental view" [24]; on the contrary, the object as seen from many other views likely has a similar silhouette to the user-sketched contour. Based on this observation, sketch-based interfaces interpolate objects that have similar silhouette curves to the user-sketched contours as seen from other viewpoints. Compared to manual tools such as Maya, sketch-based modeling produces less specific results since the user controls only a sparse set of sketched contours.

Igarashi et al. [26] presented Teddy, the first sketch-based interface for 3D object modeling. Teddy supports basic tasks such as deleting, extrusion, cutting parts, or adding new parts to the existing object. Unfortunately, Teddy is limited in the versatility of model shapes and only constructs smooth models.

Some sketch-based terrain synthesis techniques are also example-based [6, 10, 67]. Dos Passos and Igarashi [10] present LandSketch, an example-based technique that fits terrain landscapes to silhouette strokes as seen from a first person viewpoint. Tasse et al. [58] also present a sketch-based and example-based terrain editing technique from the first person point of view. Tasse et al. also explore methods to estimate depth information from t-junctions visible in the sketched contours. All of these methods share the problem common in all example-based techniques which is that the diversity of terrain results depends on the

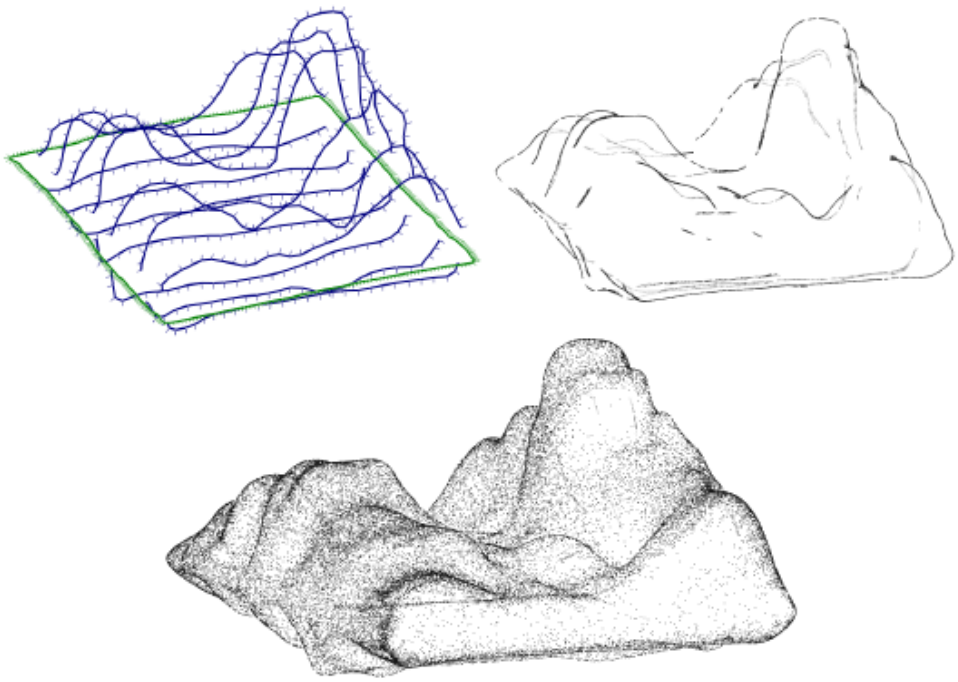


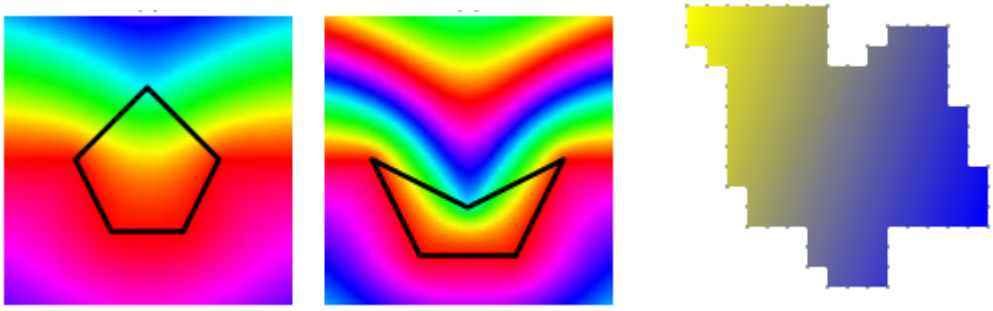
Figure 11: Brazil et al. [5] use RBF interpolation to fill the gaps between the sketched contours to model terrain

existing database used for initializing the terrain.

Sketch-based interfaces employ interpolation techniques to fit surfaces along strokes. Brazil et al. [5] employ RBF interpolation to generate 3D closed objects. Figure 11 shows an RBF interpolation which fits a terrain surface from 22 input strokes using the method of Brazil et al.

A discrete Poisson solver [41] is a popular interpolation technique. The Poisson solver finds the solution of a linear system that includes Poisson equations and equations representing constraints on the value and gradients for points in the sketched strokes. The stroke locations and values are given in the sketch; the Poisson solver determines a global solution that includes values for all points. The Poisson equation minimizes changes in the gradient of the interpolated surface, yielding a smooth interpolation.

Many of the existing sketch-based terrain synthesis methods use a discrete Poisson solver for height interpolation based on sparse strokes. Gain et al. [19] present a sketching framework for terrain synthesis that allows the user to sketch ridge contours and silhouettes of landscapes from different 3D views. To change the roughness of an existing terrain, the user selects a closed region and provides a noise profile for the region by sketching a rough



(a) Interpolating hue values at polygon vertices using mean value coordinates on a convex and a concave polygon [27].

(b) The interpolated hue values for a patch of a curve network based on points on the curve surrounding the patch

Figure 12: MVC interpolation examples

stroke. The user can also delete or add new terrain to the landscape by adding new strokes. For a given silhouette stroke, the program automatically constructs terrain with boundary lines and shadows that can be modified later by user. Hnaidi et al. [23] similarly employ a Poisson solver to obtain a full terrain from sketched input. They generate a variety of terrain features such as ridgelines, riverbeds, hills, and cracks.

Farbman et al. [15] show that Poisson interpolation is slower than mean value coordinates (MVC) interpolation. Floater [16] presented MVC as a direct solution to interpolation. MVC can be used to interpolate the interior of a polygon, whose vertices have known values. In MVC, the value of each point in the interior of a polygon is calculated based on a weighted sum of the values of the polygon vertices. The weight of each vertex polygon is a function of the distance from the point with unknown value in the interior to the vertex. MVC is a direct solution to interpolation and provides interpolated surfaces with arbitrary resolution. However, strokes need to have closed shapes for MVC to be applicable. Figure 12a shows a colored interpolation provided by Ju et al. [27] using MVC on convex and concave polygons. Each vertex on the polygon carries one value per RGB channel. Based on the known values of the polygon vertices, Ju et al. compute value of each channel for the points inside the polygons from MVC interpolation. In Figure 12b we illustrate the same experiment for an arbitrary region with known color values on the closed curve boundary.

We employ MVC as part of our terrain synthesis approach. Prior to this work, MVC

has not been used for terrain synthesis. In our approach we construct a network of interconnected curves which partition space with individual patches. The boundary of each patch is points on the curve network. Once we compute height values for all the points in the curve network, then we employ MVC to interpolate height values for all the points in each patch bounded by the curves.

2.4 Summary

In this chapter, we reviewed terrain synthesis methods. Although a heightfield is the most common terrain representation, certain shapes such as caves and arches cannot be represented by a heightfield. In procedural modeling techniques, synthetic terrains are controlled by a set of parameters, whereas in sketch-based techniques, user-sketched strokes control the shape of terrains.

The early terrain synthesis methods were fractal-based. Then Musgrave [35] started a trend that simulates erosion to add realism to synthetic terrains. Erosion-based terrain synthesis methods exhibit a trade-off between physical accuracy and synthesis speed and are slower than other procedural techniques. On the other hand, example-based methods add realism by resampling terrain features from input elevation data. The main drawback of example-based techniques is that terrain variety is restricted by the diversity of features in the existing input databases.

Sketch-based interfaces provide an easy framework for terrain feature placement since they allow user to sketch ridges and silhouettes. However, two major problems exist when analyzing sketched input. The first is that the ridge depth is underconstrained in the sketch; the second is we need a method to automatically complete partially occluded ridges. In addition to problems relating to sketch analysis, there is the issue of how to structurally control terrain roughness. Many of the existing sketch-based techniques employ diffusion-based methods for fitting surfaces in the gaps between sparse input and present synthetic terrains with low level of detail compared to the procedural techniques.

The method we present in this thesis generates detailed terrain based on the placement of scattered peaks and is capable of adding prominent ridges to sparse set of input peaks. Although we do not present a sketch-based approach, our method can be regarded as the back-end of a sketch-based interface for terrain synthesis.

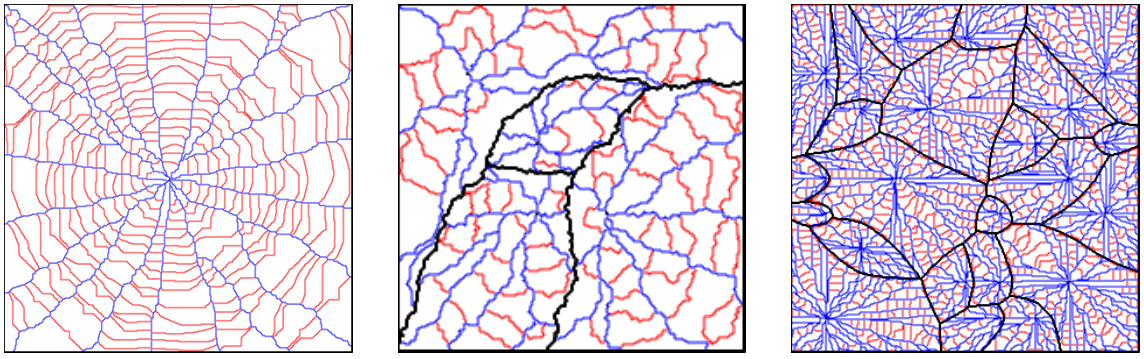
Chapter 3

Algorithms

In Chapter 2 we reviewed past procedural and sketch-based modeling (SBM) techniques for terrain synthesis. Although procedural methods can produce detailed terrain, the user is more restricted than in SBM methods to control the outcome of modeling. In sketch-based methods, smooth interpolation techniques are used to fill the gaps between the sparse input height values. These smooth interpolation techniques generate terrain with less detail than is apparent in procedurally generated synthetic terrain. In our approach, similar to SBM methods we start off with a sparse set of input data such as the placement and elevation of individual peaks. In contrast with the SBM methods, we add structural detail in the gaps between the scattered input peaks.

The images in Figure 13 show the top-view of two real-world mountains. The clouds around these two mountains illustrate an approximately constant elevation level around the peaks. Figure 13 shows sets of irregular ridges which connect peaks to their surroundings. We present an algorithm which creates mountains based on the placement and appearance of mountain peaks, and then procedurally adds ridge detail to the gaps between the peaks. We take as input the position, height, and average outwards slope of a set of scattered individual peaks. The user can also specify the shape of ridges outwards from peaks. The placement of the individual peaks dictates the path of ridges in the vicinity, since the peaks are starting points for the path of ridges. The blue paths in the images of Figure 14 illustrate synthetic ridges extending outwards from scattered peaks. The black borders in Figure 14 show the boundaries between the neighborhoods of the peaks. In this thesis we refer to the closed paths bordering individual peak positions in space as *valley borders*, the area inside the valley border of each input peak position as a *mountain region*, and irregular paths starting from a peak and ending on a valley border as a *primary curve*. In Figure 13

Figure 13: Photographs from the top view of mountains taken from NASA [38]



(a) Single peak curve network **(b)** Curve network **(c)** Dense curve network

Figure 14: Three sample curve networks

the blue paths show primary curves, the black borders show valley borders, and each area partitioned by valley borders is an individual mountain region. A mountain region contains a single peak that extends primary curves towards valley borders.

We refer to the elevation along a curve as a *height profile*. For each curve, we can employ a 1D interpolation to set the height profile, given that the elevation of endpoints are known. For all the primary curves, one endpoint is an input peak and the other endpoint is contained in the valley borders. We estimate height values for points on valley borders with a distance-based approach which gets the nearest peak position for each valley border and assigns height according to the distance walked and the elevation of the peak. For each valley border point, as the distance to the closest peak gets larger, the height difference with the peak elevation increases. In the images of Figure 14 we also show red paths in between

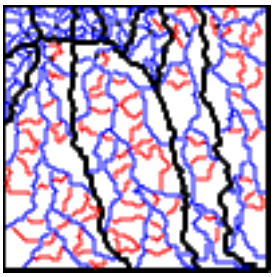
pairs of primary curves. We refer to each path connecting a point in a primary curve to another point on a different primary curve as a *secondary curve*. Once assigned height, the secondary curves control the shape of terrain in perpendicular direction to the primary curve paths. We can compute the height profile of each secondary curve once the elevation of all points on the primary curves are calculated, since the endpoints of all the secondary curves are points on the primary curves.

We refer to the interconnected structure of paths, as seen in the images of Figure 14, as a *curve network*. A curve network partitions space into isolated *patches* with borders from points on the curves. All of the black, blue and red points in the images of Figure 14 account for all of the points on the curve network and the white points are part of the patches. So far, we explained that based on peak heights we compute valley border heights, then we compute primary curve heights based on height of peaks and valley borders, and then we calculate secondary curve heights from height of peaks, valley borders, and primary curves. Once we assigned height to all the points in the curve network, we interpolate the height of the patches using mean value coordinates so the height of each patch depends only on the heights along the patch border.

3.1 Overview

Our algorithm consists of three main steps. In the first step we construct a curve network, in the second step we assign height profiles separately to each path segment of a curve network, and in the third step we interpolate height in patches surrounded by the curves based on the computed height values of the points on the curve network. Figure 15 shows a synthetic terrain from our approach next to a photo of a mountain range. We first constructed the curve network in Figure 15a, then we calculated the height profiles of curves inside the curve network, and then interpolated patch heights to get the heightfield in Figure 15b. Figure 15c illustrates that the area around mountain peaks contains ridges. Figure 15d shows that the synthetic terrain from our approach also carries ridge detail in the area between the peaks.

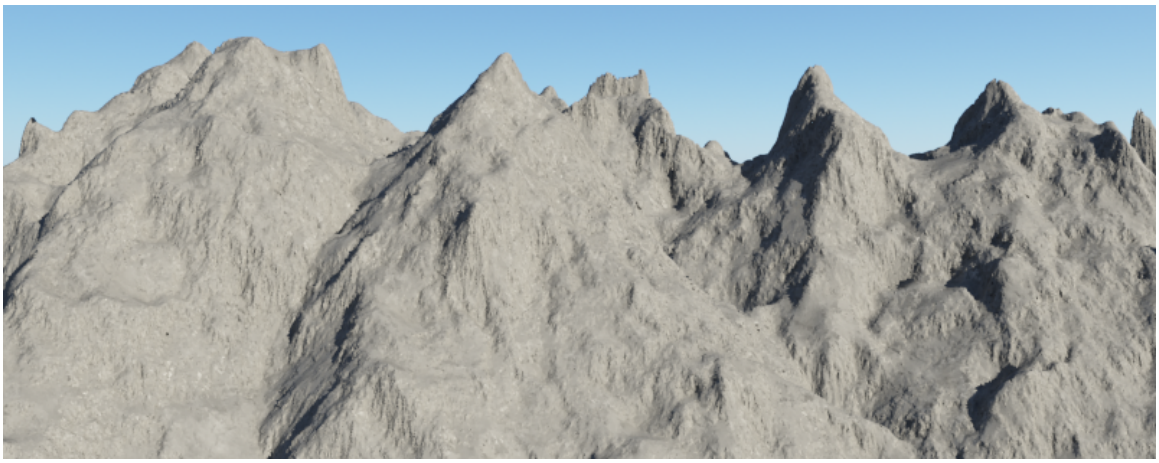
In this chapter we first describe a new cost setting for computing least-cost paths based on Dijkstra's algorithm. Then we employ our path planning approach to construct curve networks. Then given a curve network we assign height to curves. We use random walks with custom probability distributions to get distinct shape of height profiles. Finally we discuss the use of existing interpolation techniques for computing elevation in the patches.



(a) Curve network

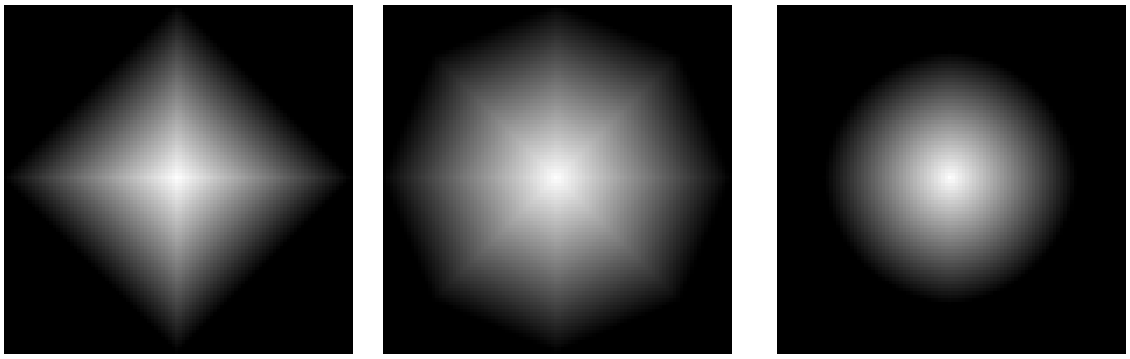
(b) Heightfield

(c) Major peaks of the Teton range: photo taken by Flickr user *tlindenbaum* [60]



(d) Synthetic terrain constructed based on the elevation and location of ten scattered input peaks

Figure 15: Curve network, heightfield, reference photo, and final render of a mountain range



(a) Based on Algorithm 1 **(b)** Based on Algorithm 2

Figure 16: Visualization of least-cost values from a single source node in the middle of image, using different cost calculation methods

3.2 Curve network generation

In this section we explain how to generate curve networks based on the placement of individual input peaks. Our goal is to be able to control height profiles both in the direction of ridge growth from peaks and in the direction perpendicular to ridge growth. We use Dijkstra's path planning algorithm to create a mountain region around each peak and to construct primary and secondary curves. First we explain ways to control the shape of least-cost paths when using Dijkstra's algorithm. Then we explain how to partition space into separate mountain regions such that each region contains a single peak and then we assign the boundaries between different mountain regions as valley borders. Then we explain how to construct primary curves for each peak such that the path of curves extend from a peak until they meet on the valley borders. Finally in Section 3.2.4 we explain how to construct secondary curves perpendicular to primary curve path growth, enabling us to complete the curve network.

3.2.1 Path planning

Rusnell et al. [49] were the first to employ path planning for terrain synthesis. They computed least-cost traversals in a connected graph using Algorithm 1 to get the minimum distance of all nodes from a set of input feature points and then assigned a height to every node using $h_n = h_{max} - cost_n$ where h_{max} is the height of the tallest feature point. Based on Algorithm 1, Rusnell et al. minimize the distance walked in a weighted graph from the

closest feature point in the format of $(dx + dy)$ where dx is the sum of distances walked in x-direction and dy is the sum of distances walked in y-direction. However, for the diagonal edges in an 8-connected graph, Rusnell et al. compute the distance walked in the format of $\sqrt{dx^2 + dy^2}$. The images of Figure 16a visualize the calculated least-cost paths from a point in the center of the images using Algorithm 1 in a 4-connected graph for the left-hand image and in an 8-connected graph for the right-hand image. The spurious vertical, horizontal, and diagonal lines in the images of Figure 16a are lattice artifacts from the structure of the graphs.

We compute least-cost paths through a weighted graph in a slightly modified way than in Algorithm 1. For each node, the distance walked through edges of the graph in Algorithm 2 is accumulated in the form $\sqrt{dx^2 + dy^2}$. Unlike the approach of Rusnell et al., each edge in our method carries two weights: one in the x direction and the other in the y direction. Figure 16b visualizes the least-cost paths from the source node in the middle of the image computed using Algorithm 2 in a 4-connected graph. Figure 16b illustrates that the modified cost settings in Algorithm 2 removes the lattice artifacts visible in Figure 16a. To get the least-cost paths in Figure 16b, for each node n in the graph, the final cost is:

$$c_n = b_n + \sqrt{dx^2 + dy^2} \tag{2}$$

where b_n, which we refer to as the base cost, is the cost of the closest source node to n. Note that source nodes have known cost values and all other nodes will get their costs based on distances to source nodes.

The natural way to combine b_n, dx, and dy to compute the cost of nodes is to add the base cost to the incremental distance, i.e., the root square of distances walked. Figure 17a shows a rendered view of the heightfield in Figure 16b in which we used Equation 2 to compute the costs.

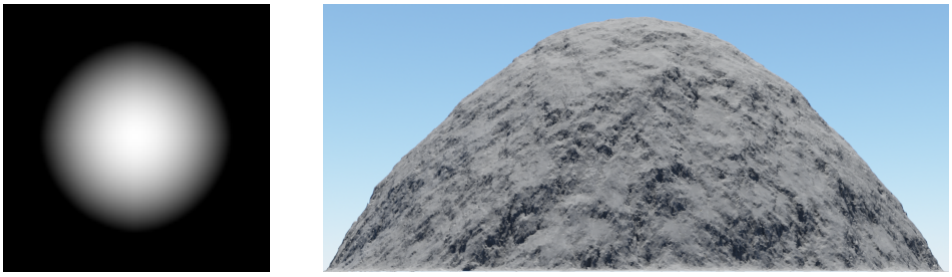
Another way to compute least costs, based on the tuple of b_n, dx, and dy, is by assuming the base cost b_n is zero and by incorporating the base cost into the square root as if the base cost is a result of an initial distance walked of $\sqrt{dx_0^2 + dy_0^2}$. The alternative way for computing the cost of nodes is summarized in the following equation:

$$c_n = \sqrt{b_n^2 + dx^2 + dy^2} \tag{3}$$

Algorithm 2 Terrain synthesis based on least-cost paths to feature points with known height and location indicated as source nodes in the connected graph $G(N, E)$, based on a customized distance function

function XY-DIJKSTRA
    $T \leftarrow$ empty min heap
    for all $n \subseteq N$ in graph $G(N, E)$ **do**
        if n is a source node **then**
            $b_n \leftarrow h_{max} - h_n$
            $dx_n \leftarrow 0$
            $dy_n \leftarrow 0$
            $c_n \leftarrow b_n$
            Push node n into heap T
        else
            if n not a blocking node **then**
                $c_n \leftarrow \infty$
            end if
        end if
    end for
    while T not empty **do**
        Pop node n_0 with minimum cost from heap T
        for all edge e incident to node n_0 **do**
            $dx \leftarrow dx_{n_0} + wx_e$
            $dy \leftarrow dy_{n_0} + wy_e$
            $c \leftarrow b_{n_0} + \sqrt{dx^2 + dy^2}$
            $n_e \leftarrow$ node connected to n_0 via e
            if n_e not a blocking node and $c < c_{n_e}$ **then**
                $b_{n_e} \leftarrow b_{n_0}$
                $dx_{n_e} \leftarrow dx$
                $dy_{n_e} \leftarrow dy$
                $c_{n_e} \leftarrow c$
                push n_e into heap
            end if
        end for
    end while
end function

(a) For each $n \subseteq N : c_n = b_n + \sqrt{dx^2 + dy^2}$



(b) For each $n \subseteq N : c_n = \sqrt{b_n^2 + dx^2 + dy^2}$

Figure 17: Cone vs. Dome-shaped mountain

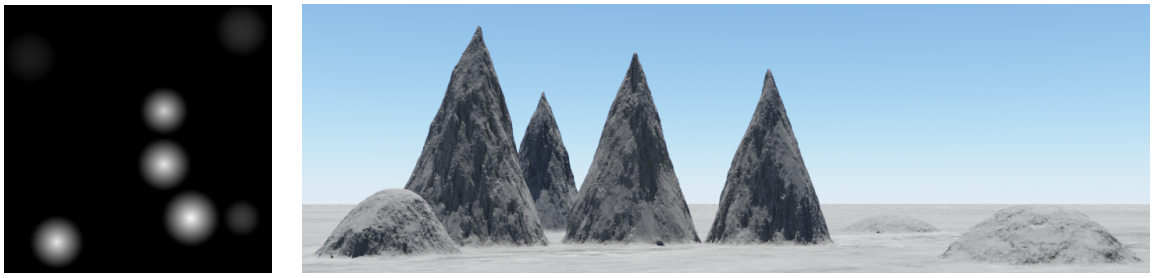
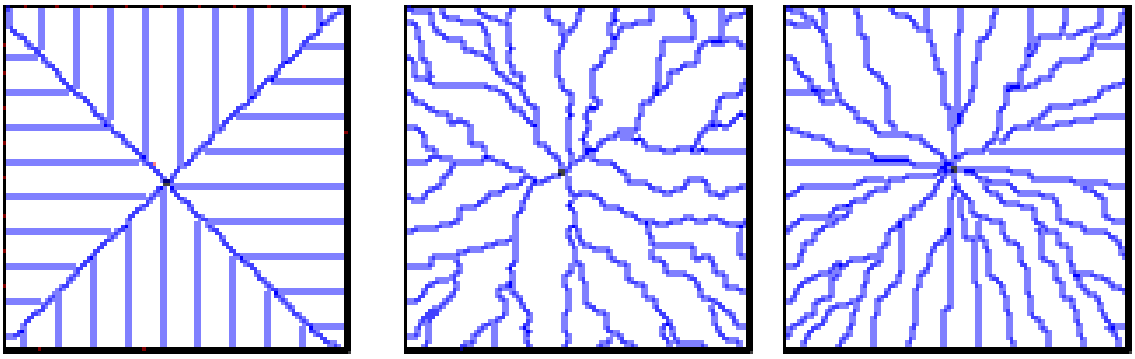


Figure 18: A heightfield generated using the cost setting in Equation 3 in a graph with multiple source nodes.

where b_n is the cost of the single source node in the middle of the left image in Figure 17b.

These two ways produce different effects, as shown in Figure 17. The difference between Figure 17a and Figure 17b is only in the way we compute cost values. This example shows that the cost setting in Equation 2 produces cone-shaped mountains, but when using the cost setting in Equation 3, selecting a non-zero cost value for b_n results in a dome-shaped mountain. Figure 18 shows a heightfield generated using the cost setting in Equation 3 in a graph with multiple source nodes. The shorter mountain peaks in the right-hand image of Figure 18 look like dome-shaped mountains.

Figure 19a shows a set of shortest paths from endpoints on the edges of the image to the



(a) Constant edge weights **(b)** Random edge weights

Figure 19: Visualization of shortest paths connecting scattered endpoints on the edge of the images to a source node in the center of image, using different edge weight distributions

source node in the middle of the image. These paths are constructed using path planning in a graph with constant edge weight distribution. Rusnell et al. [49] employed different edge weight distributions to construct different types of terrain. Figure 19b shows irregular paths constructed on the same graph but with random edge weight distribution.

The cost change from traversing an edge is computed based on the weight of the edge. In Algorithm 1 the cost change is w_e. In Algorithm 2 however, each edge in the graph should carry two edge weights: wx_e, the weight along the x-direction, and wy_e , the weight along the y-direction. In Algorithm 2 the cost change is a function of wx_e and wy_e. When employing Dijkstra's algorithm $|w_e| = \sqrt{wx_e^2 + wy_e^2}$, which is the magnitude of weight change via edge e, cannot be zero. However, either of wx_e or wy_e can be zero. We can multiply specific slope values to paths growing out from each source node to get mountains that have different slopes per mountain peak in the same graph or to construct mountains with different slopes in per direction.

Algorithm 2 can also prevent paths from passing certain nodes in the graph. We refer to the nodes that block path growth as *blocking nodes*. Blocking nodes are marked with a flag. Any node that gets pushed into the heap is either a source node or will have a path through the connected graph to a source node. We do not allow blocking nodes to be pushed into the heap so that we can stop path growth past blocking nodes.

3.2.2 Mountain region and valley border generation

Given a set of input peaks with known location and height, we determine the elevation of the remaining points based on the height of the peak closest to each node. We partition space into different mountain regions such that for every node in the region, the closest peak is the one inside that region. For example, we constructed the partition in Figure 20b such that the height of the points colored in red depend on the elevation of the peak inside that region.

To partition space into separate mountain regions, we assign a unique peak ID to each peak node and insert all peak points into the heap described in Algorithm 2. Then for each node, we store the ID of the closest peak when traversing the graph through the least-cost paths. We employ the peak ID property of the nodes to partition the graph into separate mountain regions. Figure 20a shows a visualization of least-cost values after employing a pass of path planning with multiple peak points, each with different slope values. A shallower slope for a peak allows the mountain region around the peak to be wider than the rest. Figure 20b shows a combination of mountain regions with different widths constructed from varying the peak slopes which get multiplied with the cost in the path planning process. The mountain regions constructed from larger peak slopes are smaller.

Valley borders contain points where height profiles starting from each peak terminate. A profile gradually decreases from a peak until the path reaches a valley border. The position of valley borders as illustrated in black in Figure 20b is where the two nodes with different peak identifiers meet. To get a one-pixel-thick valley border we first get all the nodes with an edge that connects two nodes with different peak IDs. Then, for each such edge we assign the incident node with the smaller peak ID as a valley border.

3.2.3 Primary curve generation

After we construct the valley borders, we employ boundary tracing on each mountain region to get an ordered sequence of the boundary. Then to get endpoints of primary curves in each mountain region, we take every n-th node along the boundary, where n is smaller for denser arrangement of primary curves around a peak. The yellow points in Figure 21a show the placement of a scattered set of endpoints along the boundary of a mountain region. To complete the process of primary curve generation, we trace paths back from each endpoint to the peak in that mountain region.

Each *curve element* is an ordered path of connected points in the curve network, where

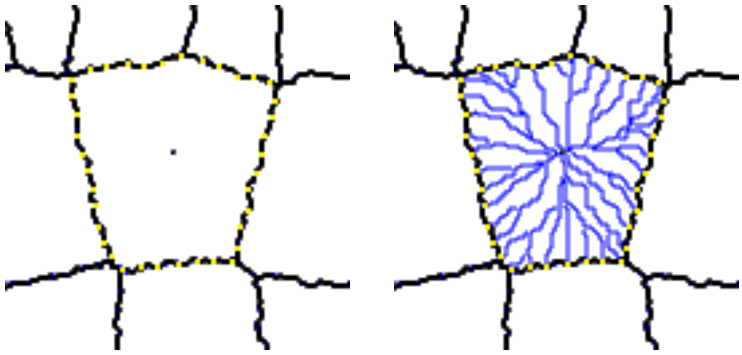
(a) Least-cost values **(b)** Mountain regions **(c)** Valley borders

Figure 20: Mountain region and valley border generation using least-cost traversal in a graph from multiple randomly spaced source nodes

both ends of the curve element are attached to other parts of the network. The curve network is segmented into a series of interconnected curve elements such that a point on the network only belongs to a single curve element. Figure 21b shows that paths from points on the valley border to the peak exhibit forked shapes. Since we want the points on the curve network to be unique to a single curve element, not all primary curve elements can be a complete path from an endpoint on the valley border to a peak point. Instead, each primary curve element can be part of the path that connects a peak point to a valley border, which is not already assigned to another primary curve. In the cases where paths from two different endpoints branch in before they meet at the peak, for one of them we assign the entire path to the peak as a single primary curve element, and for the other one we assign the path from the endpoint to the point where the two paths branch in as the other primary curve.

3.2.4 Secondary curve generation

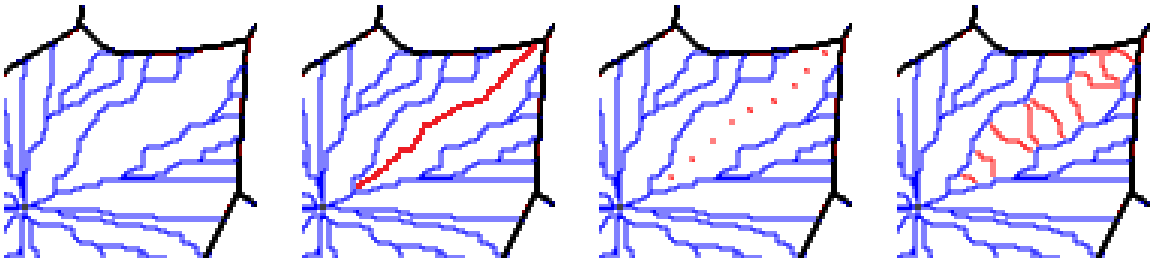
A curve network consists of primary and secondary curves. Each secondary curve connects two points on different primary curves. Secondary curve generation consists of a 4-step process. Figure 22 illustrates the four steps for generating a set of secondary curve elements between a pair of primary curves. We first add all primary curve nodes to the heap and set a blocker flag for all valley border and peak nodes and then compute least-cost paths using Algorithm 2. In the process of finding least-cost paths, we pass the ID of the primary curve element of the source nodes in the heap to all the nodes in a least-cost path starting from



(a) Place scattered endpoints on valley borders

(b) Find paths from each endpoint back to the source node

Figure 21: Primary curve generation on a single mountain region



(a) Start paths from primary curves

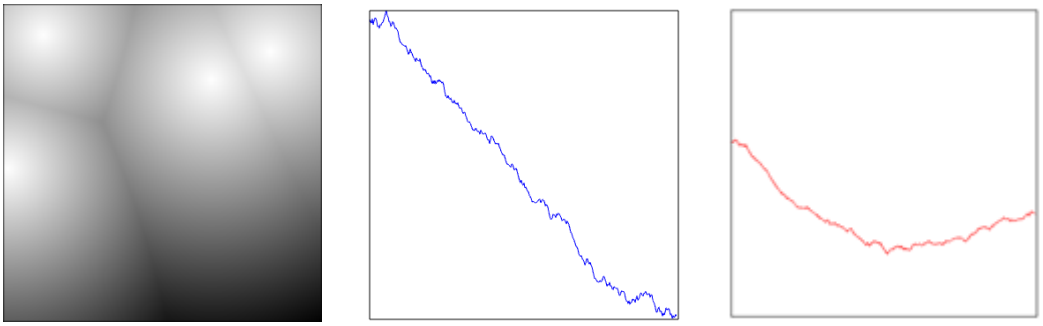
(b) Find edges connecting nodes with different primary curve IDs

(c) Sample the edges

(d) Trace paths back to each primary curve

Figure 22: Visualization of secondary curve generation in the region between two neighbor primary curves

a source. Then we look up the graph for edges that connect nodes with different primary curve IDs. The red points in Figure 22b show the location of these edges of interest between a pair of primary curves. As shown in Figure 22c, we sample these edges of interest based on a step size. The least-cost paths for each pair of node connecting to these edges of interest connect to two different primary curve elements. In Figure 22d we illustrate the paths traced back to primary curves for each of the sampled edges in Figure 22c. We assign the ordered sequence of points connecting two different primary curves as a single secondary curve path. Note that in Figure 22 we only illustrate the process for the region enclosed by two primary curves, although this process constructs all the secondary curve elements at once.



(a) Valley border estimates **(b)** Primary curve estimates **(c)** Secondary curve estimates

Figure 23: Profile estimates for points on valley borders, primary curves, and on secondary curves

The endpoints of secondary curve elements are points on primary curves and the endpoints of primary curves are points on peaks, valley borders or points that branch off a path between peaks and valleys. So far, we explained how to construct the different parts of a curve network. The next step of our algorithm is to assign height profiles to the curve elements of a curve network.

3.3 Height assignment inside a curve network

In Section 3.2 we constructed curve networks based on the placement of peaks and computed the (x, y) values for every point in the curve network. In this section we describe how to get the height coordinate for all points on the curve network.

The irregular paths of primary curves connect valley borders to peaks and we control the height profile of primary curves using peak and valley border elevations. We use distances to peaks to estimate valley border heights, so then we can interpolate heights of primary curves from the height of peaks and valley borders. We can determine the height of secondary curves once the heights of points on all primary curves are known.

Figure 23a shows an example heightfield with multiple peaks. We use the heights calculated from least-cost paths to estimate valley border elevation. Figures 23b and 23c show example height profiles that we use for the primary and secondary curve elements, respectively. For the points on a primary curve, we look for non-monotonic height profiles that go downwards from a peak until the path reaches a valley border. The height profile for the secondary curves implies the shape of terrain in the paths perpendicular to ridge growth

from peaks. The height profile shown in Figure 23c, which dips down in the middle of the profile, represents a terrain with prominent ridges on the primary curves.

We interpolate heights along the path of a curve element by referring to heights of each curve's endpoints and by employing a specific 1D interpolation technique. The simplest solution is linear interpolation:

$$h(t) = (1 - t) \cdot h_{ep_1} + t \cdot h_{ep_2}$$

where h_{ep_1} and h_{ep_2} are the two endpoint heights and t is a value between 0.0 and 1.0. We can use other kinds of interpolation such as spline interpolation. However, we prefer to construct more stochastic height profiles in order to add roughness to terrain. To get a rough interpolation, we employ random walk signals for the height profile of curve elements in the network.

To generate irregular height profiles, we obtain a single curve profile by linear interpolation of two separate random walk signals, each starting from one endpoint of a curve element and finishing at the other end. Equation 4 shows the linear interpolation formula for this two-way profile generation technique:

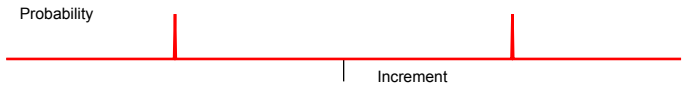
$$h(t) = (1 - t) \cdot R_1(t) + t \cdot R_2(1 - t) \tag{4}$$

where R_1 and R_2 are the two random walk signals generated from each endpoint of the given curve element. We constructed the profiles in Figure 23b and 23c by interpolating two random walk signals.

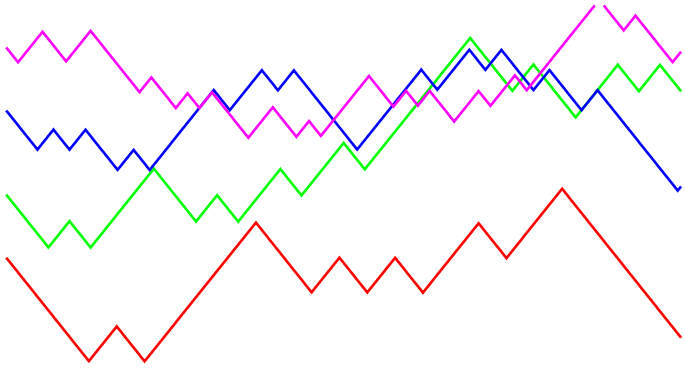
For the purpose of height profile generation, we project the time dimension of random walk on the (x, y) plane of a curve element path and interpret the value of signal as height. In the remainder of this section we present a technique to produce distinct shapes of random walks by controlling the probability distribution function (PDF) of displacements in each step of a random walk.

3.3.1 Random walk profile generation

Consider the "drunkard's walk", also known as the classic unbiased random walk, as explained in Chapter 2. In a drunkard's walk there is a 50% likelihood of a positive increment



(a) The drunkard's PDF



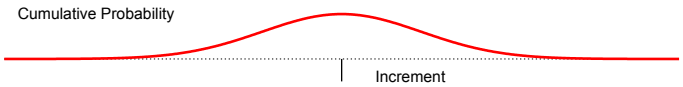
(b) Four drunkard's walks

Figure 24: The drunkard's PDF and four random walk signals generated from the PDF

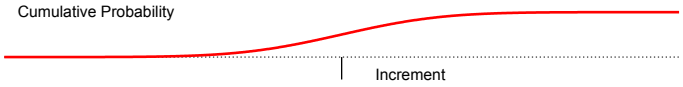
and a 50% chance of a negative increment. Figure 24a shows the PDF of the drunkard's walk and Figure 24b illustrates four drunkard's walks based on the given PDF. The x axis for the PDFs indicates the random increments, and the y axis represents the probability. For the random walks, the x axis shows the spatial displacement and the y axis depicts height. Also, the vertical line in the middle of the x axis of the PDFs marks the zero coordinate of the axis.

We use discrete approximations of an underlying PDF, represented as a histogram of the probability that an increment lies within a given range, uniformly sampled. In the drunkard's walk PDF, two bins carry non-zero values, one on the positive side and one on the negative side. In this thesis we show that we can also use other shapes of PDFs to construct random walks, each with distinct shapes. Figure 25a shows four random walks generated from a Gaussian PDF. The expected mean of all these random walks remain constant because of the symmetry in the Gaussian PDF with respect to the zero coordinate.

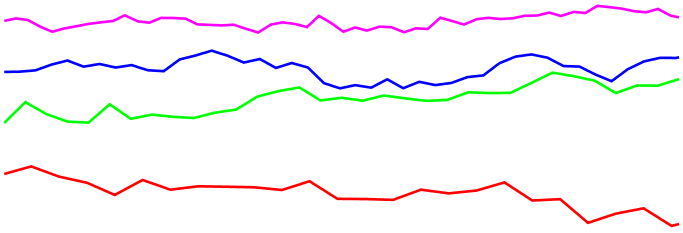
Signals generated with non-symmetric PDFs are referred to as biased random walks. Figure 26b shows some biased random walks generated from the sketched PDF in Figure 26a. The expected mean value of a probability distribution is the average of the random



(a) A Gaussian PDF



(b) Cumulative distribution of a Gaussian PDF



(c) Four random walk signals constructed from a Gaussian PDF

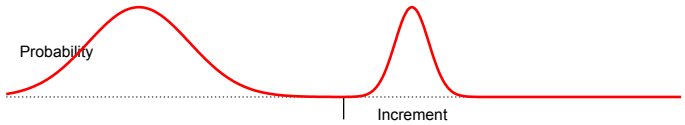
Figure 25: A Gaussian PDF and four random walk signals generated from the PDF

numbers generated over a large number of samples. The equation for calculating the expected mean value of a probability distribution function is:

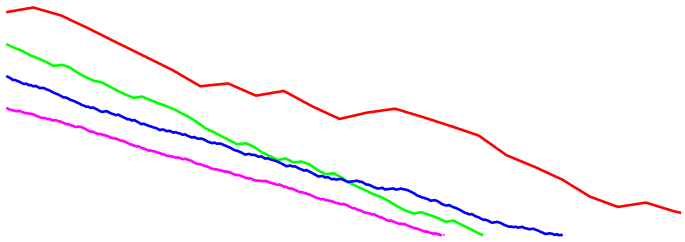
$$E(x) = \sum_{i=1}^{n} (x_i \cdot p_i)$$

where p_i is the probability of x in the PDF and n is the number of sampled bins in the discrete representation of a custom PDF. A cumulative distribution function (CDF) describes the probability to find values less than or equal to a certain value of x [7]. Figure 25b illustrates the CDF of the Gaussian PDF in Figure 25a. Since all cumulative distributions are monotonically increasing, they are invertible. Thus, we can generate a random number based on the input PDF. We employ this inverse of a CDF at each step of a random walk to set a vertical displacement.

To get a random walk signal from a PDF, we first compute samples on a lattice and



(a) A sketched PDF



(b) Four random walk signals constructed from a sketched PDF

Figure 26: Four random walk signals generated from a sketched PDF

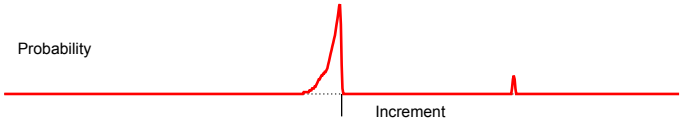
Algorithm 3 The usage of CDFs in populating heights on the lattice of random walks

1: $S[0] \leftarrow$ the initial height value
2: **for** $n = 1 \rightarrow N - 1$ **do**
3: $\quad inc \leftarrow$ a random increment from the inverse of the CDF
4: $\quad S[n] \leftarrow S[n-1] + inc$
5: **end for**

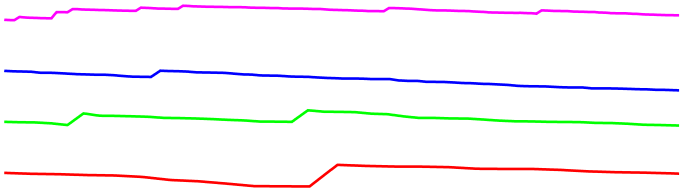
store in array S. Algorithm 3 shows how to compute array S. This Algorithm works by adding a random increment from the inverse of the CDF at each step of a random walk, storing the cumulative sum in array S. Now that we have S we are able to evaluate $R(x)$ for any off-lattice x, by linearly interpolating values in S, using the equation below:

$$R(x) = (x - \lfloor x \rfloor) \cdot S[\lfloor x \rfloor + 1] + (1 - (x - \lfloor x \rfloor)) \cdot S[\lfloor x \rfloor] \tag{5}$$

where $\lfloor x \rfloor$ and $\lfloor x \rfloor + 1$ are the points on the lattice.



(a) A sketched PDF



(b) All octaves

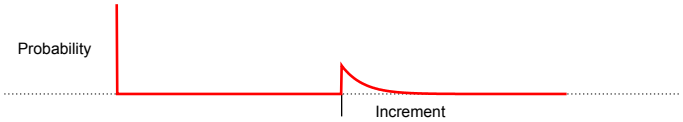
(c) The fBm sum

Figure 27: The random walk octaves and the fBm sum generated from a biased PDF

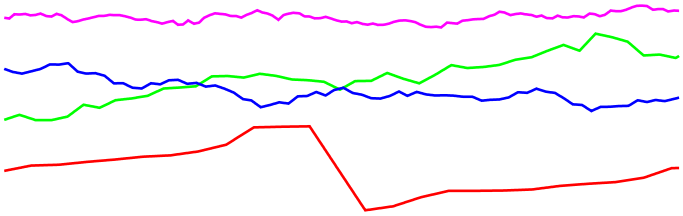
3.3.2 Random walk fBm

As discussed in Chapter 2, Musgrave et al. [35] constructed fractal terrains by interpreting the vertical displacement of fBm signals as height. Musgrave et al. [35] employed Perlin noise function, denoted by N in Equation 1, as the basis for generating fBm signals. Their technique produces homogeneous and isotropic terrain. In our work, we slightly modify Equation 1 in that, instead of Perlin noise, we employ random walk signals as the noise basis:

$$h(x) = \sum_{i=1}^{n} R_i(x \cdot \gamma^{H \cdot i})/L^i \tag{6}$$



(a) A sketched PDF

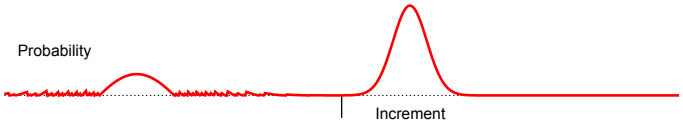


(b) All octaves

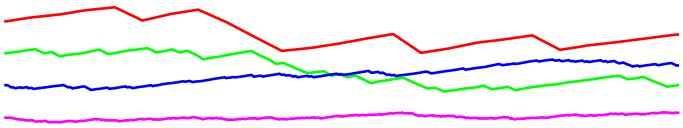
(c) The fBm sum

Figure 28: An fBm signal generated using random walks from two different PDFs

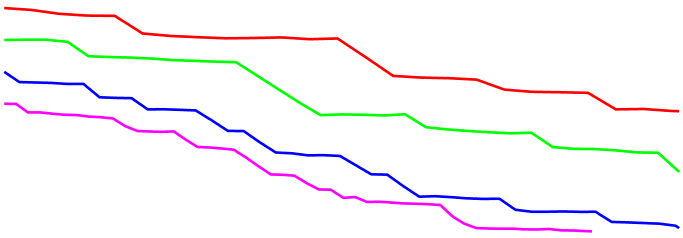
where R_i is a random walk function producing signals with different frequencies for each octave i in an fBm. For every R_i we have to precompute a corresponding S_i using Algorithm 3 to be able to evaluate the height of any point in the random walk. Therefore, there is no random-access ability in Equation 6 as there is with the Perlin noise function. Musgrave et al. [35] discussed that the noise function should be stationary. Therefore, we use random walks with random increments generated from PDFs with expected mean of zero for the noise signals R_i in Equation 6.



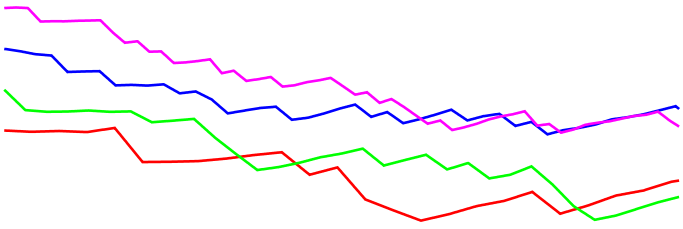
(a) Corrected version of the sketched PDF in Figure 26a



(b) Four random walk signals constructed from a corrected PDF



(c) Four random walk signals constructed from a corrected PDF $+k$



(d) Four random walk signals constructed from a corrected PDF $+f(k)$

Figure 29: Using corrected PDFs to construct different types of random walk signals

To get stationary random walks we can directly employ symmetric PDFs as the basis of R_i. We can also employ user-sketched PDFs to construct biased random walks for R_i, in the case that the PDFs have zero mean. Alternatively, for PDFs with non-zero expected mean, we can compute stationary random walks, based on linearity of expectation, by subtracting the constant mean of the sketched PDF, k_0, from each random increment. This is equivalent to constructing a random walk from a new PDF with expected mean of zero, shifted by k_0 with respect to the x coordinate. However, correcting the PDFs this way will alter the ratio of probabilities that user sets for a slope value.

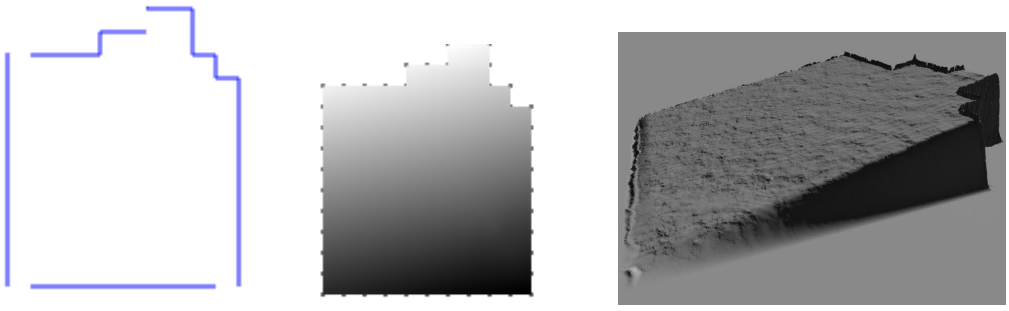
Imagine a user sketched a PDF with a distribution where positive values are close to zero and negative values are further from zero. Then the random walks constructed will exhibit a steeper slope for negative slopes compared to the positive ones. Correcting a PDF based on linearity of expectation will not keep the ratio of probabilities that the user sketched for the different values in the PDF. Instead, we propose another method to correct the user-sketched PDF which preserves the shape of random walk signals generated.

To get a corrected PDF with expected mean of zero from an asymmetric PDF, we first calculate the expected mean for all negative and all positive values separately. We need to decrease a small fraction from all the bins of whichever side of the coordinate which the expected mean is larger, until the expected mean of both sides even up within some threshold. We decide on a small decrement and start on the first bin until the two sides of the coordinate are in balance; if we reach the last bin before the sides even up, we go back to the first bin and proceed from there.

The reason we iteratively decrease probability is that we want to make sure we reduce the same ratio of probability from all bins in the negative or positive coordinate, whichever exhibits a larger expectation in the sketched PDF. The result of this process is a corrected PDF which produces random walks with zero mean and exhibits the preserved ratio of probabilities from the sketched PDF. Figure 29a illustrates the corrected PDF of the sketched one in Figure 26a and Figure 29b shows random walk profiles generated from this corrected PDF.

The fBm signal in Figure 27c is produced by summing up the four random walks in Figure 27b with the same PDF shown in Figure 27a. In Appendix A we show a set of PDFs and their corresponding random walks and fBm sum. The diagrams in Appendix A indicate that different PDFs produce distinct shapes of biased random walks.

So far, we explained an approach which constructs stochastic fBm signals with distinct shapes, by changing the PDF of the random walks separately for each octave. This will



(a) Curves around the patch **(b)** The interpolated patch **(c)** The rendered patch

Figure 30: MVC patch interpolation

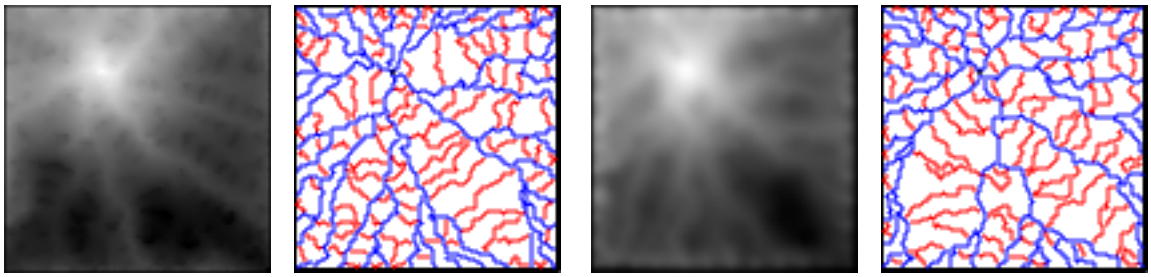
translate into an fBm signal where all octaves have similar shapes to one another. To get other types of fBm signals, where at a certain scale, the shape of signal is different from the rest, we can employ a random walk from different PDFs for each octave. In Figure 28 the lowest-frequency random walk is generated using the illustrated biased PDF whereas the other three frequencies are generated using a Gaussian distribution PDF.

In some cases we need to construct base profiles with user-specified shapes. We add fine-scale detail to these base profiles by superimposing random walks. Figure 29 illustrates three sets of random walks from the PDF in Figure 26a. We use linearity of expectation to construct the random walks in figures 29c and 29d where we add $f(k)$, which is based on a base profile, to the slope values at each step interval of the random walks. Figure 29b shows random walks from the same PDF used for Figure 29c and 29d but with no added base profile.

In this section we explained how to construct structure-aware height profiles for the curve elements in a curve network, given that the curve element endpoints have known elevation. For each curve element, we linearly interpolate two random walks each starting from one endpoint of the curve. Also, we explained how to fine-scale detail to a base profiles using random walks. In the next section, we discuss how to get the height of the remainder of points based on the height values of points in a curve network.

3.4 Height interpolation around the curve network

At this point, we have the (x, y, h) coordinates of all the points on the curve network. To get a complete heightfield, we need to calculate the height of the remaining points with

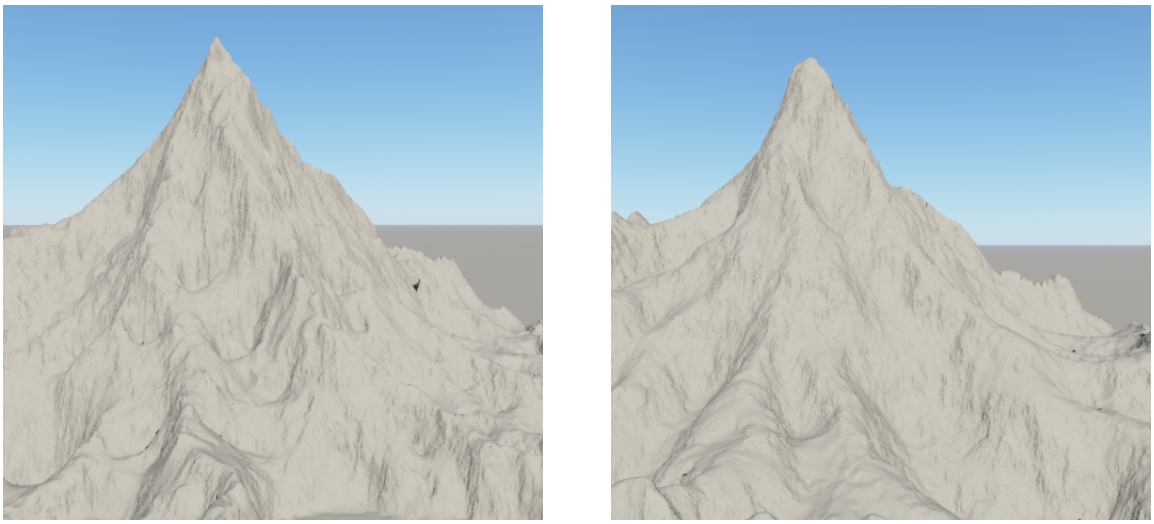


(a) Heightfield generated with MVC interpolation

(b) Curve network of the MVC interpolated terrain

(c) Heightfield generated with Poisson interpolation

(d) Curve network of the Poisson interpolated terrain



(e) Terrain synthesis using MVC interpolation

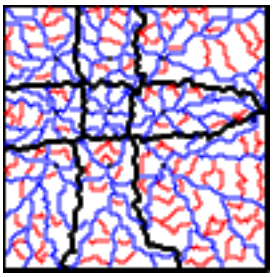
(f) Terrain synthesis using Poisson interpolation

Figure 31: Comparison of patch interpolation between Poisson and MVC

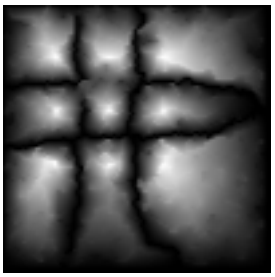
unknown height, located in the patches bounded by curves in the network.

One way to get a completed heightfield is to interpolate each patch independently based on the known heights of patch boundaries. We interpolate each patch separately using mean-value coordinates (MVC) [16]. In Chapter 2 we described MVC in slightly more detail. Our MVC patch interpolation technique is based on the implementation of Hormann and Floater [25]. When using MVC, we need a closed polygon around the area in which we want to interpolate. We use boundary tracing [21] to find an ordered list of the border of each patch.

Figure 30a shows a close up view of a patch surrounded by parts of four different curves illustrated in blue; Figure 30b shows the interpolated patch interior based on the height of the boundary points on the blue curves. Figure 30c shows a Terragen render of



(a) Boundaries



(b) Heightfield

(c) Reference photo taken by Rodrigo Soldon [47]

(d) Rendered

Figure 32: Example result

the interpolated patch in Figure 30b. MVC interpolates a smooth interior surface for each patch. Note that the patch surface in Figure 30b is rendered with small-scale noise added by Terragen.

The terrain in Figure 31e is interpolated with MVC. Another way to compute the remaining heights is to interpolate all the patches at once, based on height of all the points in the curve networks, using a discrete Poisson solver. Figure 31f shows a mountain with patches interpolated using a discrete Poisson solver, where the value of each point on the curve network is passed as a value constraint to a system of linear equations. Figure 31e has sharper peaks and ridges which we consider desirable for the purpose of generating rough terrain.

Once we change the height profile of only some of the points in a curve network, when using MVC, we only need to update the height of patches in the vicinity of the updated curve network points. Conversely, in the Poisson-based interpolation technique, a change to any point on the curve network alters the height of all the points.

3.5 Summary

Our algorithm is composed of three main components: curve network generation, profile construction, and patch interpolation. We first generate curve networks around each input peak. Then we assign height profiles on each curve element based on the values of their endpoints and a random walk based approach that controls the shape of profiles. Then we use the height of points on the curve network to interpolate heights for the remaining of points in the patches. Figure 32 shows an example terrain constructed using our algorithm. The peaks in this synthetic model have different heights and each mountain exhibits varied silhouette shapes outwards from the peaks.

In the next chapter, we provide more terrain examples created using our methodology. We discuss the roles of each element in our algorithm in producing varied shapes of terrain. Then we provide terrain models to compare with a set of real-world images and synthetic terrains generated using other approaches.

Chapter 4

Results

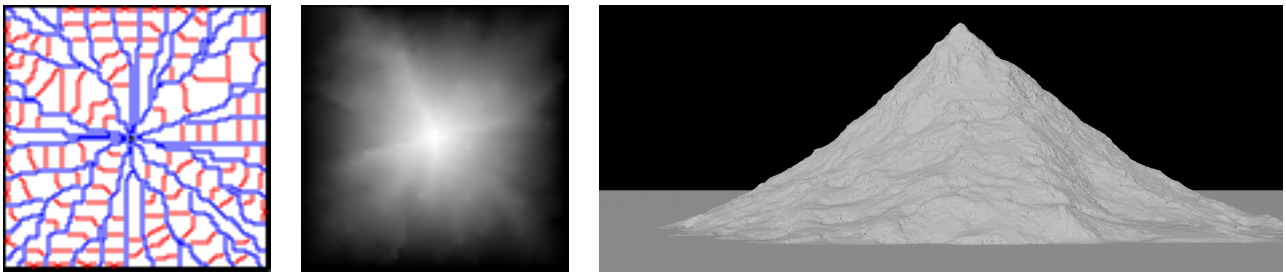
4.1 Overview

In this chapter we present synthetic terrains that are produced using our approach. First we discuss the role of different elements of our algorithm in getting varied shapes of terrain. Then we present terrains along with real-world terrain images to show the versatility of our approach in generating realistic terrains. We then compare our approach with related methodologies which have either similar approaches to the problem of terrain synthesis or exhibit comparable control over the shape of terrains. Then we provide the timing of the different elements of our algorithm.

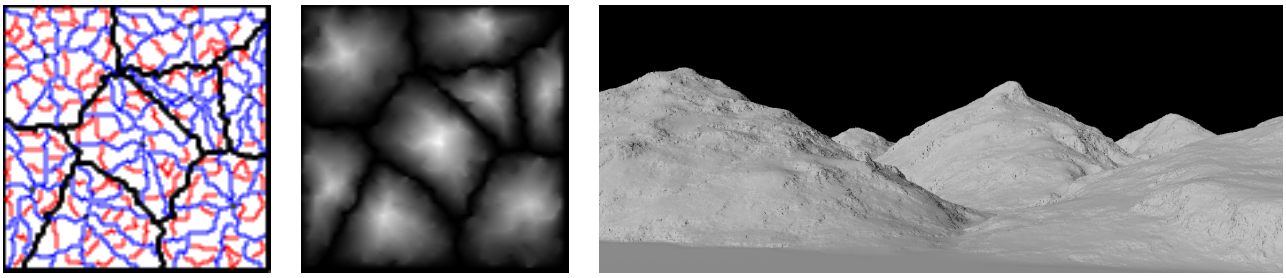
We implemented our algorithm in C++ running on a 64-bit Intel dual Core i7 CPU 2.40 GHz (and 1.9 GHz) with 8 GB RAM. The output from our program is a greyscale heightfield, where the lighter pixels represent higher elevation. We imported heightfields into Terragen [45] to render terrains. The renders contain small-scale noise with no textures on top of the surfaces unless otherwise specified. We present the curve network used to construct the terrain next to each heightfield visualization.

4.2 Role of algorithm elements

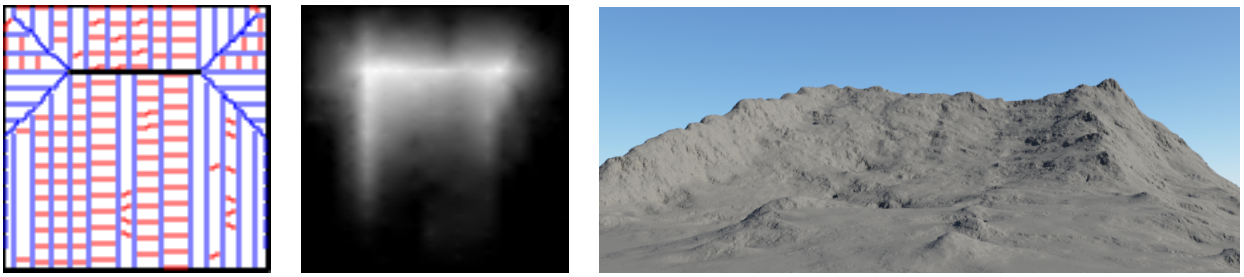
In our algorithm we first construct curve networks, then we assign height profiles on curve elements, and then interpolate heights in the patches. We get variety in terrains shapes by constructing different curve networks, by modifying height on valley borders, by controlling the density of curve elements, and by assigning random walks with custom PDFs or by superimposing PDFs on simple curves as profiles.



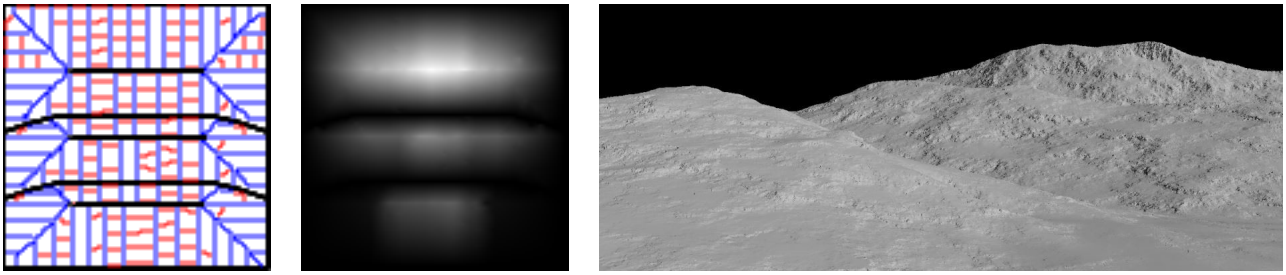
(a) Terrain with a single peak



(b) Terrain with multiple peaks

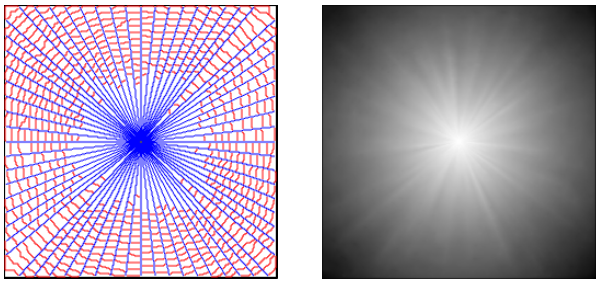


(c) Terrain with a single ridge

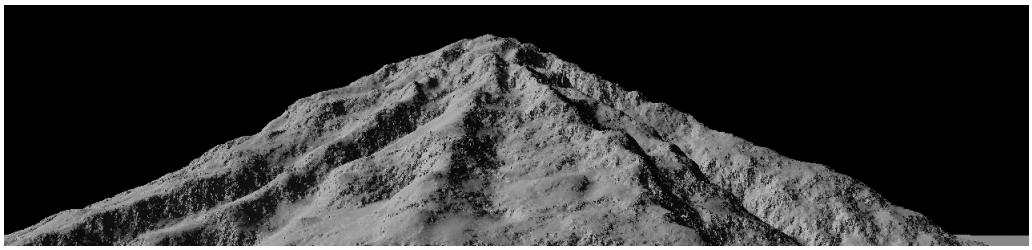


(d) Terrain with multiple isolated ridges

Figure 33: Varied curve network types (left) along with their resulting heightfield (center) and terrain (right)



(a) Curve network **(b)** Heightfield

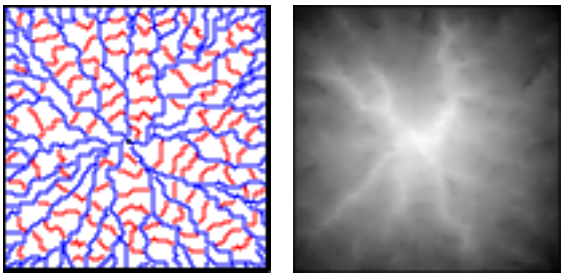


(c) Heightfield render

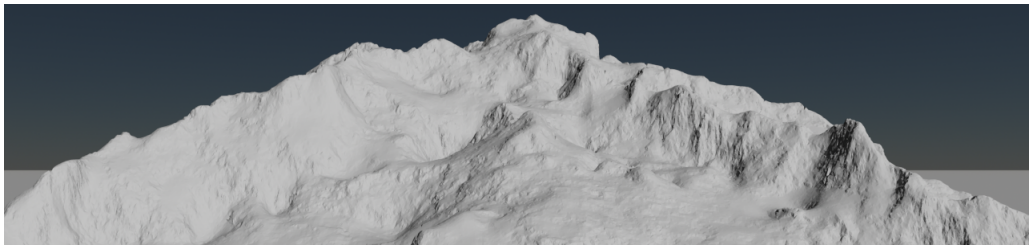
Figure 34: A single-peak terrain constructed by displacing the height of valley border points

The input to curve network generation is the position, slope, and height of a set of scattered peaks. In Figure 33 we illustrate the effect of different curve network compositions. Figure 33a shows a single-peak curve network that contains of valley borders on the edges of the image; Figure 33b shows another curve network with multiple peaks that contain valley border points along the boundaries between the scattered peak points. For the curve network in Figure 33a, we automatically generated a hierarchy of curves that grow out to the edges of the image.

Although in this thesis we mainly concentrate on generating curve networks around scattered peak positions, curve networks can also be constructed around ridges. In figures 33c and 33d we show two terrains constructed from ridge inputs rather than single peak points, on each mountain region in their curve networks. Similar to the single peak terrain in Figure 33a, the terrain in Figure 33c contains a single mountain region with valley borders along the edges of the image, whereas the terrain in Figure 33d consists of three isolated ridges partitioned into separate mountain regions. To get curve networks for ridge inputs, we assign the same peak ID to all ridge points in a connected path. Then we can



(a) Curve network **(b)** Heightfield



(c) Heightfield render

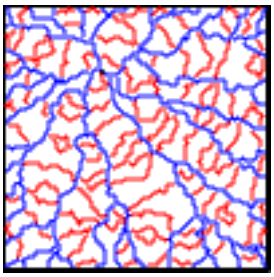
Figure 35: A dendritic terrain constructed on a dense network of curves with random walk profiles

partition space based on the closest peak ID of each point in the image.

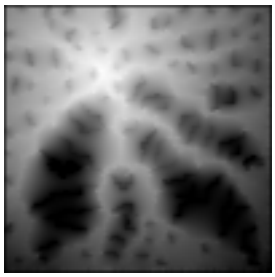
We constructed all the terrains in Figure 33 by assigning a height of zero to all the valley border points. For the terrain in Figure 34, we applied a sine wave displacement along the valley borders. The resulting terrain shows prominent ridges outwards from the peak whereas the terrains of Figure 33 do not contain such ridges. This example shows a way to add ridge detail on a mountain peak when all of the other components of our algorithm such as the curve network and the curve profiles have minimal complexity.

Another way to get structural detail is by increasing the density of curves in a curve network. The single-peak curve network in figure 35a has a high density of curves and the resulting terrain in figure 35c exhibits roughness throughout the surface and contains a distinct set of ridges outwards from the peak. This roughness is due to the large number of patches in the network and because each patch is interpolated from rough random walks assigned to the curves.

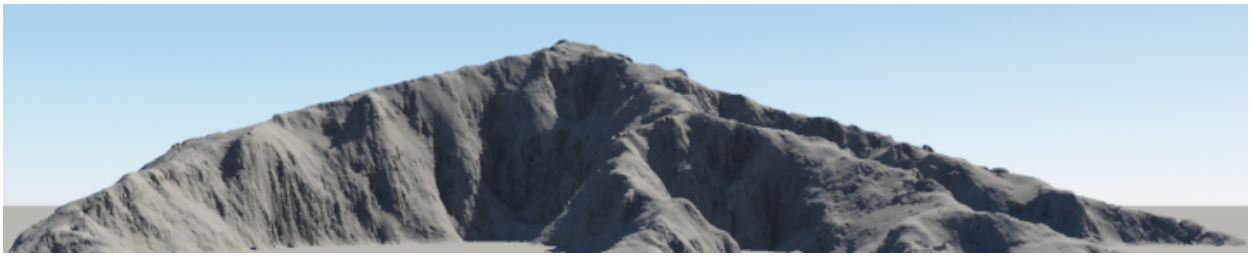
Figure 36c shows a single-peak mountain that contains monotonically decreasing ridge profiles and exhibits sudden height decrease perpendicular to ridges. We linearly interpolate the height of the primary curves in the curve network of Figure 36a whereas for the secondary curves we assign a random walk profile that dips down precipitously in the middle. This example illustrates that we can get specific shapes for a mountain both along the



(a) Curve network



(b) Heightfield



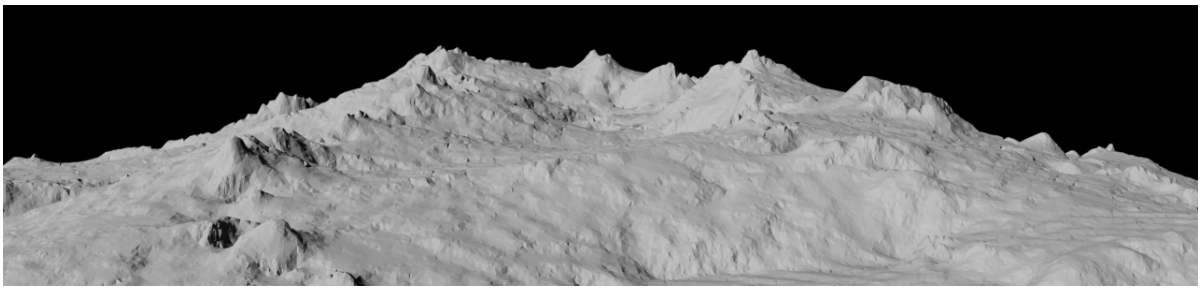
(c) Heightfield render

Figure 36: A single-peak terrain with prominent ridges, constructed by setting a base profile that dips down fast for secondary curves in the network

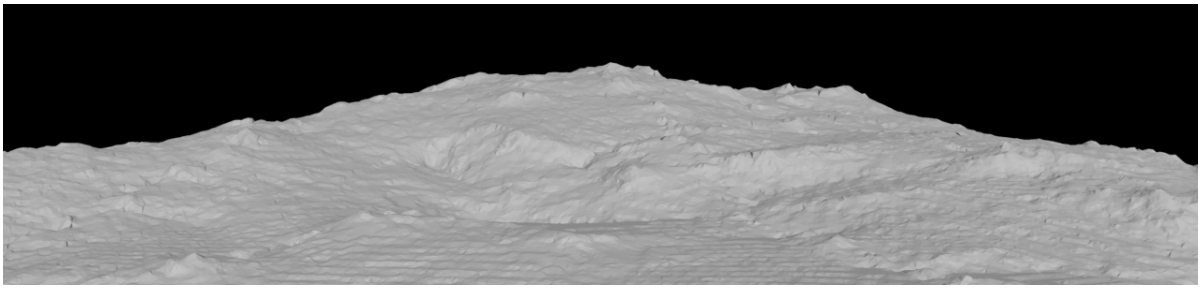
vertical and horizontal ridges, by assigning random walk profiles with distinct shapes to primary and secondary curve elements, respectively.

In Chapter 3 we explained how to construct biased random walks from sketched PDFs. Figures 50 through 60 in Appendix A show a set of biased random walks generated from sketched PDFs. Appendix A shows the role of PDFs in generating different shapes of biased random walks. In figures 37, 38, and 39 we compare the shape of terrains constructed with the same curve network, but with different random walk PDFs used for their primary curve profiles.

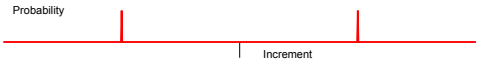
The example random walks constructed from PDF A shown in Figure 37e appear more ragged compared to the example random walks from PDF B in Figure 37f. The two single-peak mountains in Figure 37, were constructed from the same network shown in Figure 37h but with different random walk PDFs assigned to the primary curves. As expected, Terrain A appears more ragged and rough compared to Terrain B, since the random walk profiles in Terrain A are assigned from PDF A as opposed to Terrain B which employs random walks with PDF B on the primary curves.



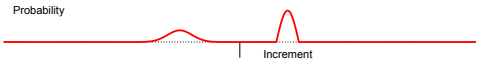
(a) Terrain A



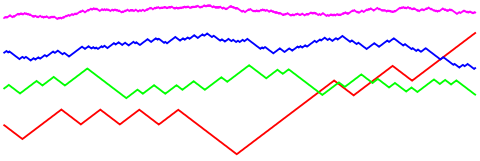
(b) Terrain B



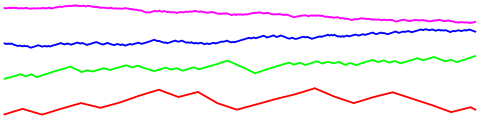
(c) Primary curve profile PDFs in Terrain A



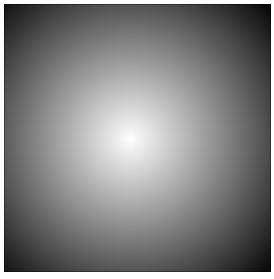
(d) Primary curve profile PDFs in Terrain B



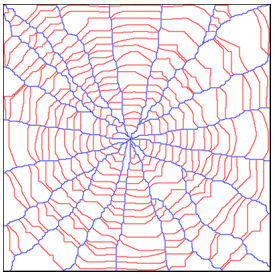
(e) Example profiles from PDF A



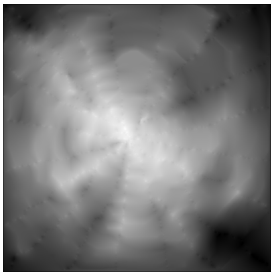
(f) Example profiles from PDF B



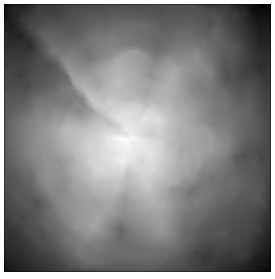
(g) Height estimates



(h) Curve network

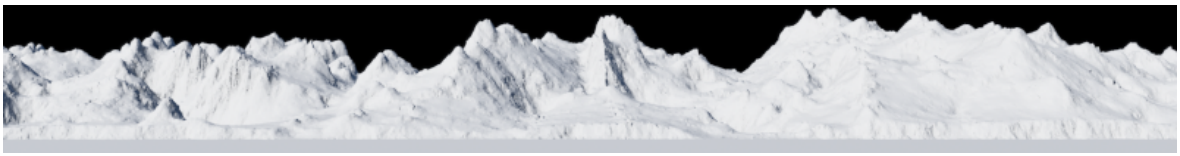


(i) Heightfield A

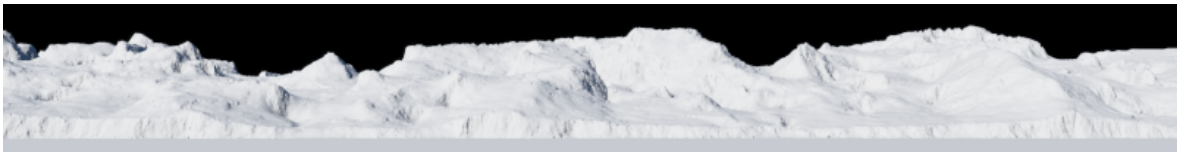


(j) Heightfield B

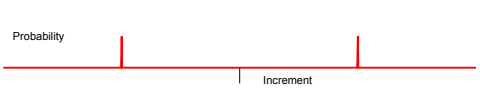
Figure 37: Two terrains generated with different primary curve PDFs – Both terrains use the same curve network, the same height estimates on valley borders, and a linear height interpolation on the secondary curves



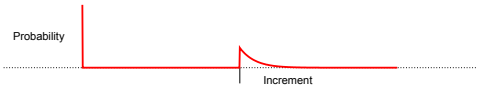
(a) Terrain C



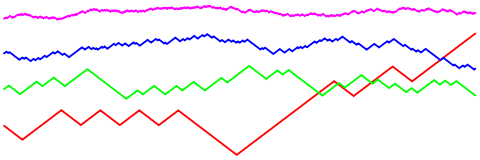
(b) Terrain D



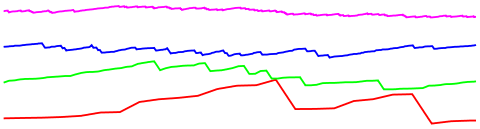
(c) Primary curve profile PDFs in Terrain C



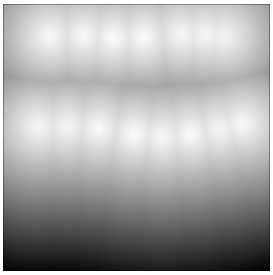
(d) Primary curve profile PDFs in Terrain D



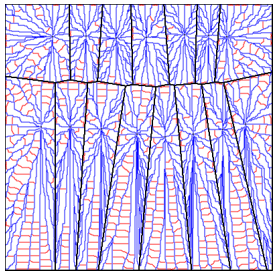
(e) Example profiles from PDF C



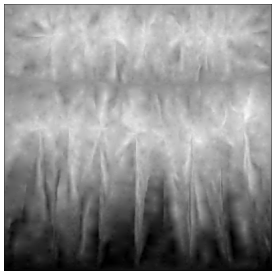
(f) Example profiles from PDF D



(g) Height estimates



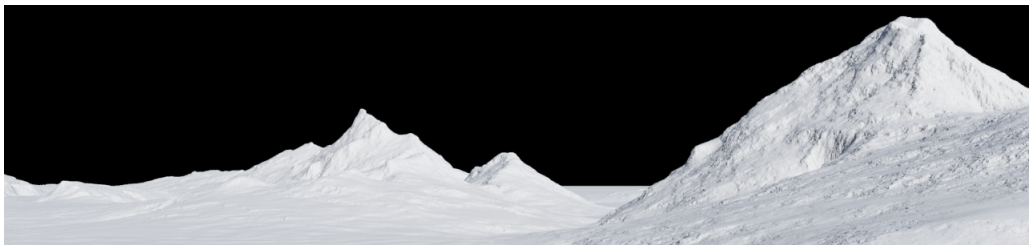
(h) Curve network



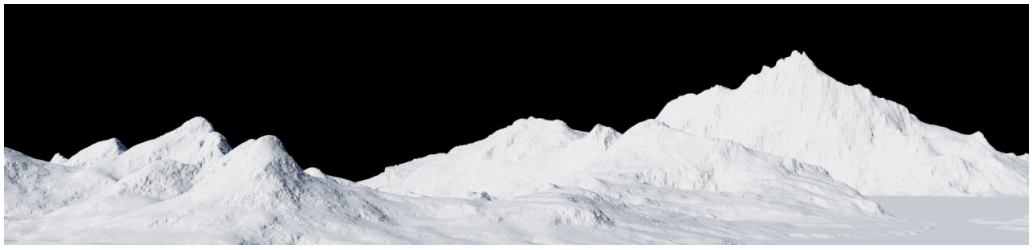
(i) Heightfield C

(j) Heightfield D

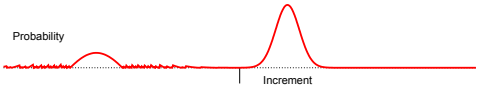
Figure 38: Two mountain ranges generated with different primary curve PDFs – Both terrains use the same curve network, the same height estimates on valley borders, and a linear height interpolation on the secondary curves



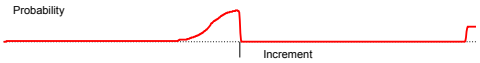
(a) Terrain E



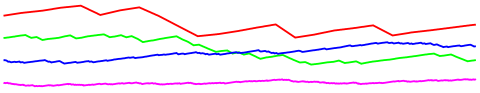
(b) Terrain F



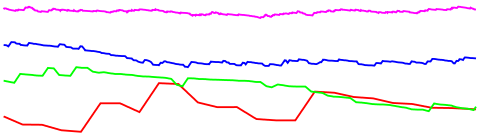
(c) Primary curve profile PDFs in Terrain E



(d) Primary curve profile PDFs in Terrain F

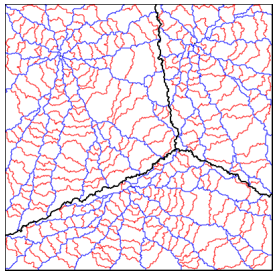


(e) Example profiles from PDF E

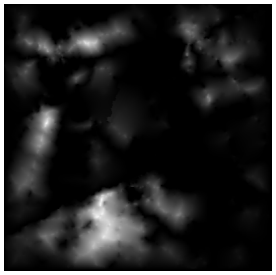


(f) Example profiles from PDF F

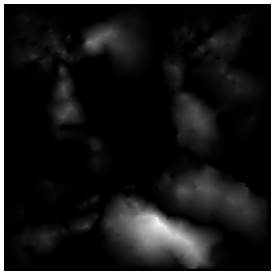
(g) Height estimates



(h) Curve network



(i) Heightfield E



(j) Heightfield F

Figure 39: Two terrains with multiple isolated ridges. Each terrain uses different primary curve PDFs on the same curve network. Height estimates for all valley border points and all three peaks are zero. The remaining points with higher elevation than the peak and valleys are shown in the terrain renders.

Figure 38 shows terrains C and D, two mountain ranges with different random walk PDFs on their primary curves. The PDF for terrain C, which is the same for terrain A, constructs ragged random walks, but the PDF for terrain D constructs plateau-shaped random walks. Comparing terrains C and D shows that we can get two very different shapes of mountain ranges, with the same curve networks and height estimates on valleys and peak, by changing the PDF of random walks assigned to the curve elements.

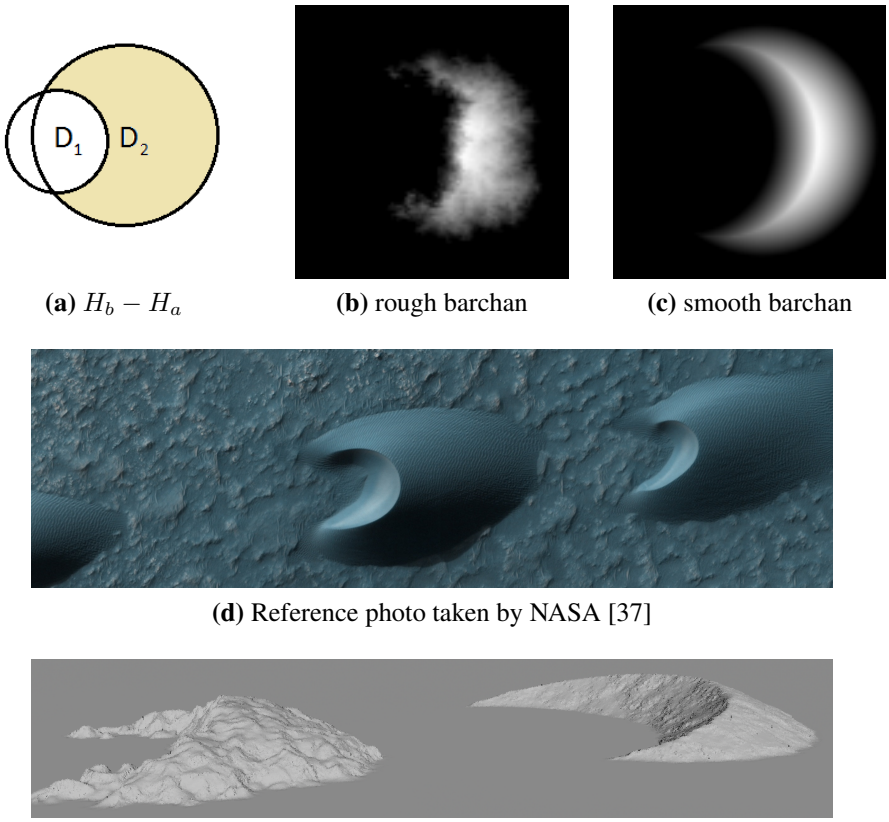
Figure 39 shows another pair of mountain ranges with ridges constructed from different random walk PDFs on their primary curves. We generated both terrains E and F in Figure 39 by assigning height of zero to all peaks and valley borders in the curve network of Figure 39h. Therefore, any point with an elevation higher than zero in terrains E and F, shows parts of a random walk primary curve taller than both the peaks and valleys. Terrain E in Figure 39a contains ridges with similar slopes whereas terrain F in Figure 39b exhibits a wider range of slope variation on the ridges. We can see this difference also by comparing the example random walks in figures 39e and 39f which are generated from PDF E and F, respectively. Therefore, we can use a curve network representation and control the shape of ridges by analyzing the profiles in 1D space.

In this section we discussed different ways in which we can use the hierarchy present in a curve network to control the shape of terrain. We showed that changes in attributes such as elevation of valley border points, curve density, and profile roughness, alters the shape of terrains constructed with the same curve network. In the next section we compare terrains from our algorithm against real-world terrain images to show the diversity of the terrains that we can produce.

4.3 Variety of formation

In this section we compare a set of synthetic terrains from our approach with real-world photographs. We show the curve network used for constructing each terrain along with a heightfield visualization. Our aim is to show that our resulting terrains resemble terrain features visible in the photos.

Constructive solid geometry (CSG) operations [53] allow modelers to generate solid object surfaces from existing models using boolean operators. We can employ CSG operations on terrains constructed from our algorithm to get varied types of shapes like barchan landforms.



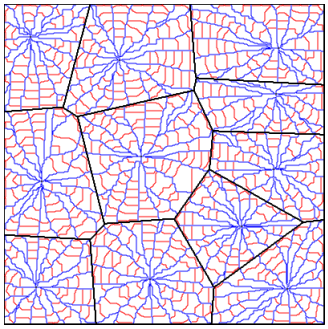
(a) $H_b - H_a$ **(b)** rough barchan **(c)** smooth barchan

(d) Reference photo taken by NASA [37]

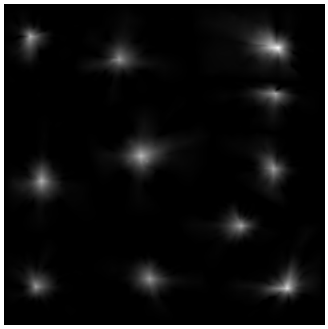
(e) Heightfield render

Figure 40: We employ CSG operations on two domes D_1 and D_2 in part **(a)** to obtain each barchan shown in part **(e)**.

Figure 40d shows a photo of two barchan dunes on the surface of Mars [37]. To synthesize the barchans in Figure 40d, we first constructed two single-peak dome-like terrains, called D_1 and D_2, with different peak steepnesses and elevations. We applied a steeper peak slope and a higher elevation to terrain D_1 in figure 40a, and positioned the peak of terrain D_1 to the left of the peak of D_2. Then we assigned the neighborhood height values from equation $H_{final} = max(0.0, H_{D_2} - H_{D_1})$. The right-hand barchan in Figure 40e resembles the barchan dunes in Figure 40d. The barchan on the left-hand side of Figure 40e has a more eroded surface than the one on the right. To construct an eroded barchan, we assign a random edge weight to the graph used for constructing the domes. We can also generate craters and volcanos by employing the same kind of CSG operations on single peak terrains.

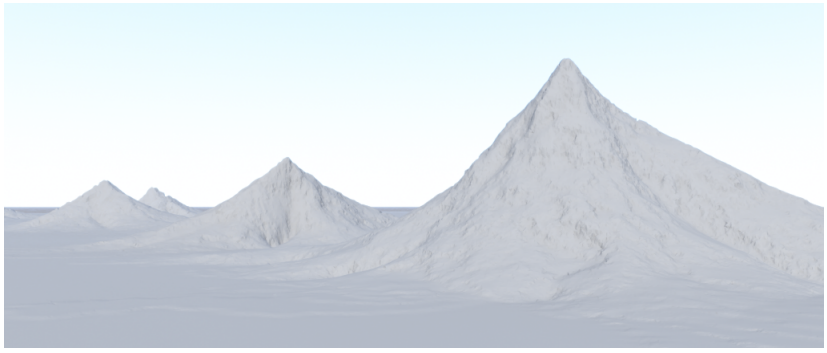


(a) Curve network



(b) Heightfield

(c) Reference photo taken by Allan Shimada [52]



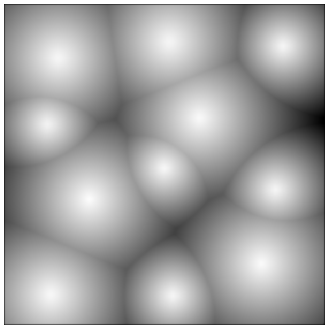
(d) Heightfield render

Figure 41: Volcanic mountain ranges. We add roughness by increasing curve density in the network and by superimposing random walks on top of the simple base profiles on primary curves

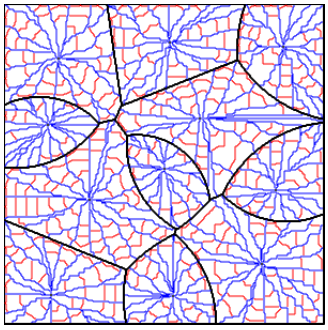
Figure 41c shows the Aleutian islands, a chain of volcanic mountains in Alaska, and Figure 41d shows a synthetic volcanic mountain range similar to the Aleutian islands. We placed a simple monotonically decreasing profile on the primary curves of the curve network in Figure 41a to get the synthetic terrain in Figure 41d. We employed different base profile for the primary curves to construct a terrain similar to the karst mountains in Gulin, China [22] shown in Figure 42d. The base profile for the karst towers dips down fast in the middle of the path from peaks to valleys, and becomes flat at the near the valleys. In both synthetic terrains of figures 41d and 42e, we get intermediate-scale detail along the paths of curves in the network while obtaining mountains with the desired profiles outwards from peaks.

Figure 43 shows an eroded mountain in the Himalayas along with a synthetic terrain containing distinct ridges. The terrain in Figure 43d contains many details throughout the surface such as irregular and distinct ridges and ravines visible at the feet of the mountain, even though the terrain was created from the simple curve network in Figure 43a. Figure 44c shows a mountain range from Karakoram, Pakistan [56], along with a closeup of terrain C from Figure 38. Terrain C and the Karakoram mountain range exhibit similar ragged ridge shapes. This synthetic mountain range contains prominent ridges which are part of the curve network. These examples show that the structure of profiles used on the curve elements dictate the overall shape of terrains that are produced.

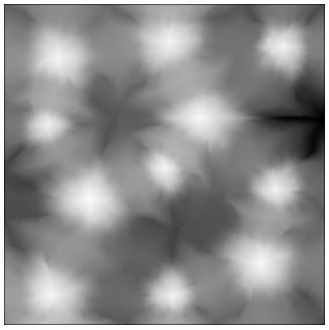
Figure 45c shows a photo of scattered peaks distributed along a nearly straight line. Figure 45d shows a synthetic terrain similar to the reference photo of Figure 45c. To construct the curve network in Figure 45a, we first select a scattered set of peaks in a connected path of ridge points and then generate a mountain region per peak. The synthetic ridged terrain in Figure 45d has an irregular collection of peaks distributed on a line, a jagged silhouette, and steep ridges in the foreground outwards from the scattered set of peaks. Both this synthetic image and the ones shown in figures 33c, 33d, and 39 illustrate different ways in which we can generate ridged terrains. In the first two, we assigned the entire ridge as a single ridge input and constructed a curve network around the ridge, but in Figure 39 we ignore the shape of the input ridge and instead keep the coordinates of local peaks scattered in the ridge. The advantage in selecting scattered peaks is that then all the curves connect to local peaks rather than arbitrary points on the ridge.



(a) Height estimates

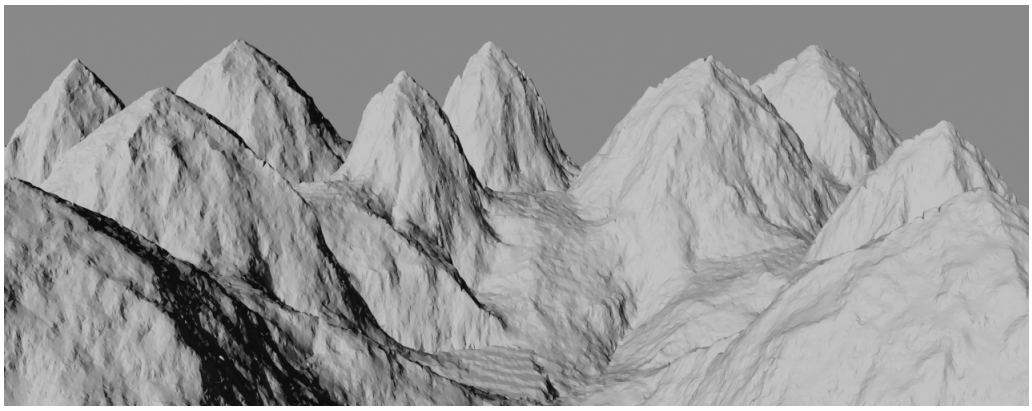


(b) Curve network



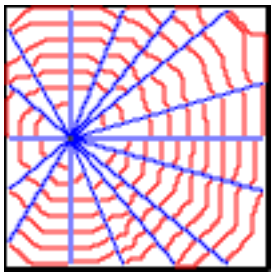
(c) Heightfield

(d) Reference photo taken by Flickr user *Hanumann* [22]

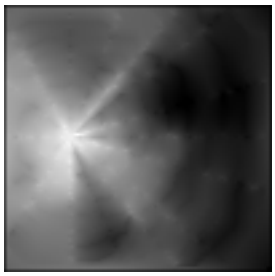


(e) Heightfield render

Figure 42: Karst mountains. We employ a simple base profile onto which we superimpose random walks for the primary curves in the network.

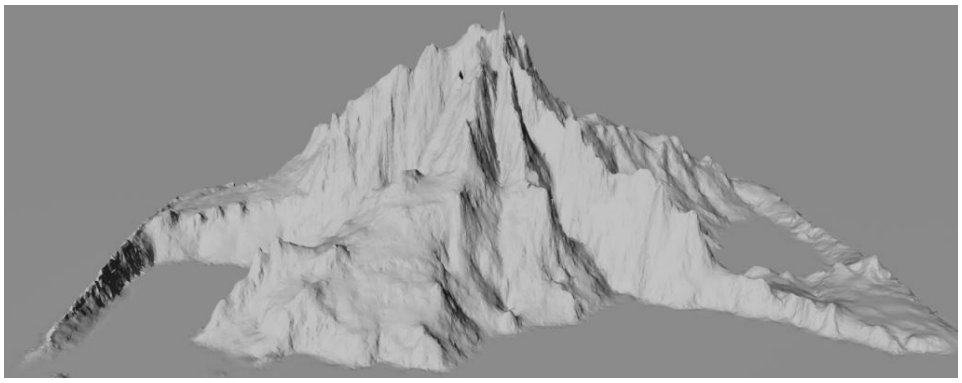


(a) Curve network



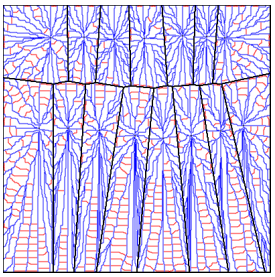
(b) Heightfield

(c) Reference photo taken by Flickr user *chriscom* [8]

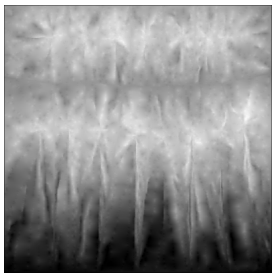


(d) Heightfield render

Figure 43: We constructed a synthetic mountain similar to a realistic mountain in the Himalayas by employing a two-sided log profile that dips down steeply in the middle.

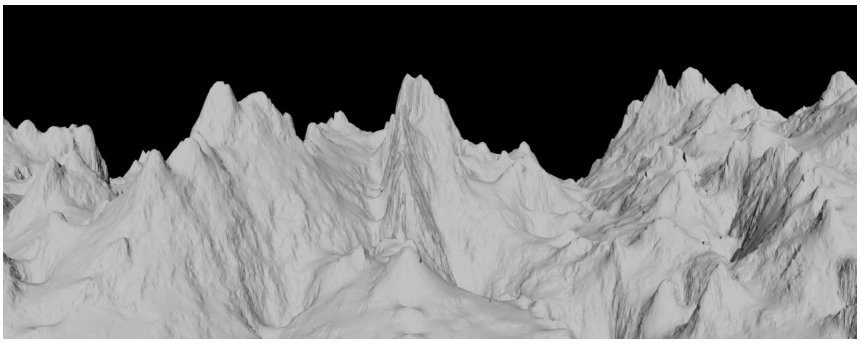


(a) Curve network



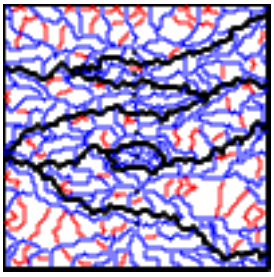
(b) Heightfield A

(c) Reference photo taken by Stefanos Nikologianis [56]



(d) Closeup of Terrain C in Figure 38a

Figure 44: We employ drunkard's walks on primary curves in the network to get a synthetic mountain range similar to the Karakoram mountains in Pakistan [56].



(a) Curve network

(b) Heightfield

(c) Reference photo taken by El Guanche [14]

(d) Heightfield render

Figure 45: A synthetic terrain with scattered peaks distributed on a line similar to a realistic terrain.

4.4 Algorithm performance

In this section we evaluate the timing statistics regarding computation of curve networks, construction of height profiles, and interpolation of patches. The grid length indicates the width and height of the square 4-connected graph generated to construct a curve network. Table 1 includes the empirical computation times for generating a curve network. Table 1 provides the computation times for the Dijkstra's pass for finding the main three elements in a curve network: (a) the mountain regions, (b) the primary curves elements, and (c) the secondary curve elements. The total curve network construction timing accounts for the disclosed path planning timings and an additional time to analyze paths found from Dijkstra's passes to setup the curve elements and closed border sequences of patches and mountain regions as explained in Chapter 3.

Table 1: Curve network generation timing

Grid length	Mntn regions	Num curves	Valley Dijkstra	Primary Dijkstra	Secondary Dijkstra	Patch border computation	Total timing
100	1	192	1.0s	1.0s	0.9s	3.7s	8.0s
200	1	296	4.3s	5.0s	4.0s	13.4s	34.8s
300	1	113	11.5s	10.1s	9.9s	15.6s	1m 2.3s
300	3	144	10.4s	15.1s	17.0s	21.2s	1m 25.0s
400	1	413	20.1s	20.0s	18.3s	55.6s	2m 18.2s
400	8	1241	17.7s	17.7s	17.0s	2m 28s	3m 53.6s

Table 1 shows that the total time spent in the three Dijkstra's passes for constructing a curve network in a 400 x 400 grid takes less than a minute. Our Dijkstra's path planning in Algorithm 2 is similar to the one presented by Rusnell et al. in Algorithm 1. Rusnell et al. indicated that their implementation of Dijkstra's path planning takes 55 minutes to synthesize a 1536 x 1536 terrain and takes 4 minutes and 39 seconds to synthesize a 512 x 512 terrain. In the Primary Dijkstra passes in Table 1, we set peak and valley border nodes as blocking nodes and additionally set primary curve nodes as a blocking nodes in the Secondary Dijkstra pass. These extra calculations cumulate to the computation timing for these three Dijkstra passes compared to the Dijkstra pass presented by Rusnell et al.

Table 2: Profile generation and patch interpolation timing

Grid length	Curve elements	Patches	Profile assignment	Patch interpolation
100	59	69	$< 1s$	11s
200	153	167	$< 1s$	1m 2s
300	226	247	$< 1s$	3m 16s
400	999	1254	2s	3m 18s
400	312	347	$< 1s$	6m 58s

Table 1 shows that the boundary tracing implementation of our algorithm for computing patch borders is noticeably time consuming. The boundary tracing algorithm is not a very time consuming process but there is some issue in our custom implementation. This part of our algorithm is responsible for finding an ordered sequence of the closed border of each patch in the curve network. The remaining computation time of curve network generation, not shown in Table 1, is responsible for setting up labels for curve elements, valley borders, and mountain regions in the network.

Table 2 provides the timing for patch interpolation and height profile assignment on curve networks with the specified number of curve elements and patches. Profile assignment accounts for a small fraction of terrain synthesis timing. Table 2 illustrates that for the same grid length size, a larger density of curve elements and patches in a curve network speeds up the time to interpolate all the patches.

Other than the computation times presented in Tables 1 and 2, we need to consider the manual effort to set the parameters for constructing different types of terrain. The curve network generation stage requires a user to place the position and elevation of peaks. The height of valley borders are computed automatically based on distances to the closest peaks. In the profile generation stage, the user can choose to provide profiles for the curves or the system can automatically construct profiles by linear height interpolation or by random walks from existing PDFs in the database. The user can sketch a random walk PDF or select from the existing PDFs to specify the shape of curve networks. Additionally, the user can set a base profile for the curve elements onto which we superimpose random walks. The patch interpolation stage is automatic. Therefore, the user can either procedurally make terrains by specifying as little as just the placement and elevation of the peaks, or make

more specific terrains by providing more information to the system.

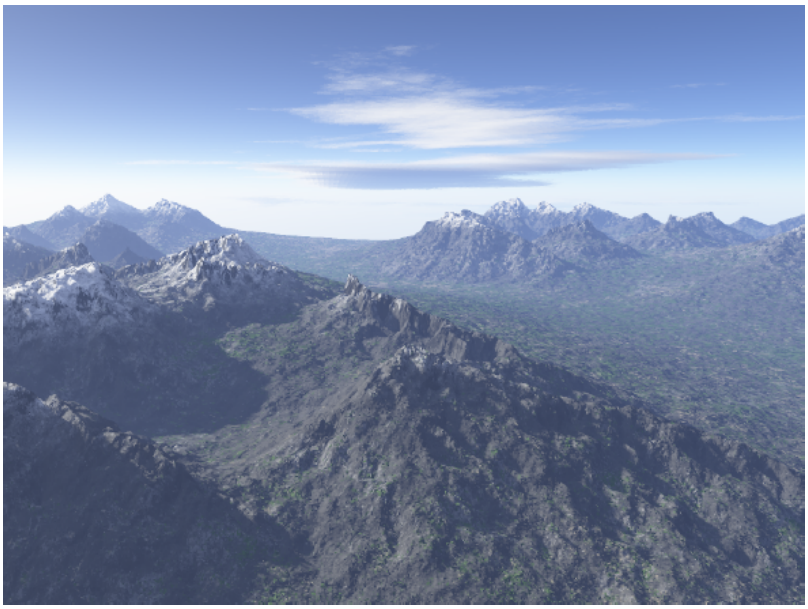
We provide a new mechanism for terrain synthesis that can be broken down to three main components. Each component can be executed independently without affecting the internal procedure of the other two components. We can get varied types of results by assigning different height profiles on curve elements on the same curve network and with the same patch interpolation method. Performance-wise, we save computation time because of the terrain variation achieved without having to process all three component interdependently.

However, there is a problem in the custom implementation of the boundary tracing responsible for finding the patch borders which accounts for the bulk of the curve network generation timing as seen in Table 1. Also, the statistics in Table 2 suggest that for large patch sizes, the MVC interpolation algorithm taken from Hormann and Floater performs poorly. We can speed up the patch interpolation by employing the mean-value seamless cloning algorithm proposed by Farbman et al. [15] which approximates MVC.

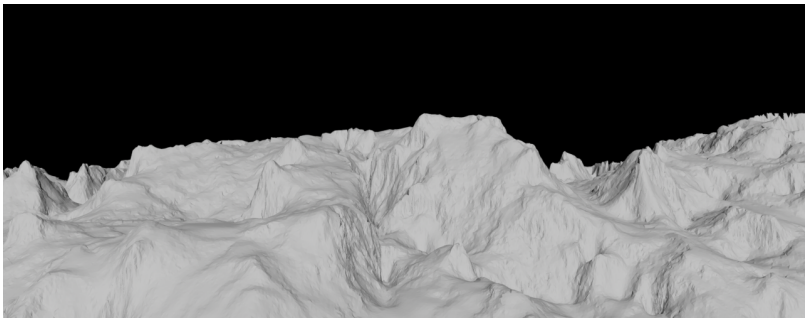
4.5 Comparison with related work

In terrain synthesis, there is a trade-off between level of detail, user involvement, and control over the resulting terrain. In procedural methods, the synthetic terrains usually are detailed and demand a low level of user involvement for setting up the procedural parameters, but the resulting terrain features are not fully controllable. In sketch-based techniques there is moderate control over the placement of resulting terrain features, but the visible structural detail depends on the amount of user involvement.

The main focus in this research has been to find a way to get structural detail while not imposing additional effort for the user or losing control of the placement of terrain features. In our hybrid method, we gain control over the resulting terrain by allowing user to determine the location of the peaks and to optionally specify the height profiles of ridges based on custom PDFs. The combination of procedural curve network generation with random walk profile construction on the curves adds a high degree of structural detail to our resulting terrains. However, random elements in our approach make the position of procedurally generated ridge detail uncertain.

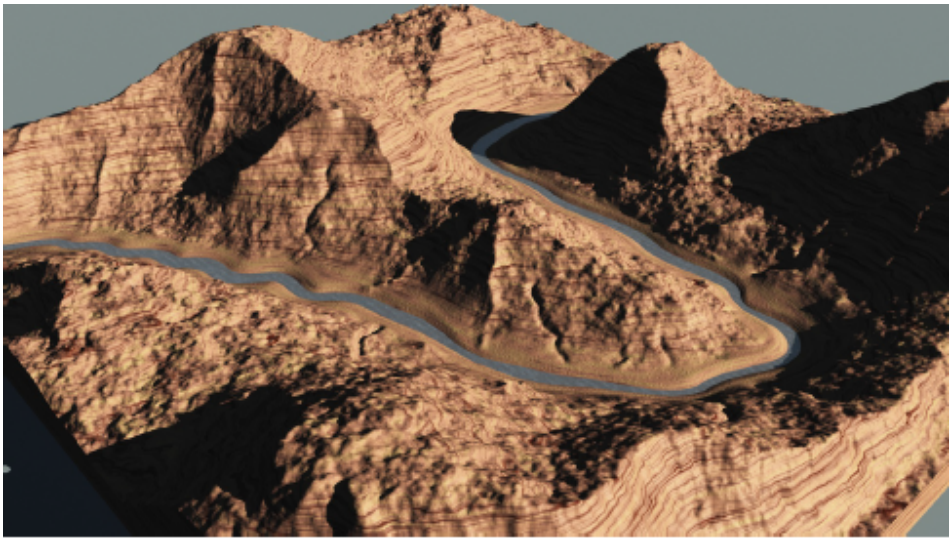


(a) Synthetic terrain from Rusnell et al. [49]

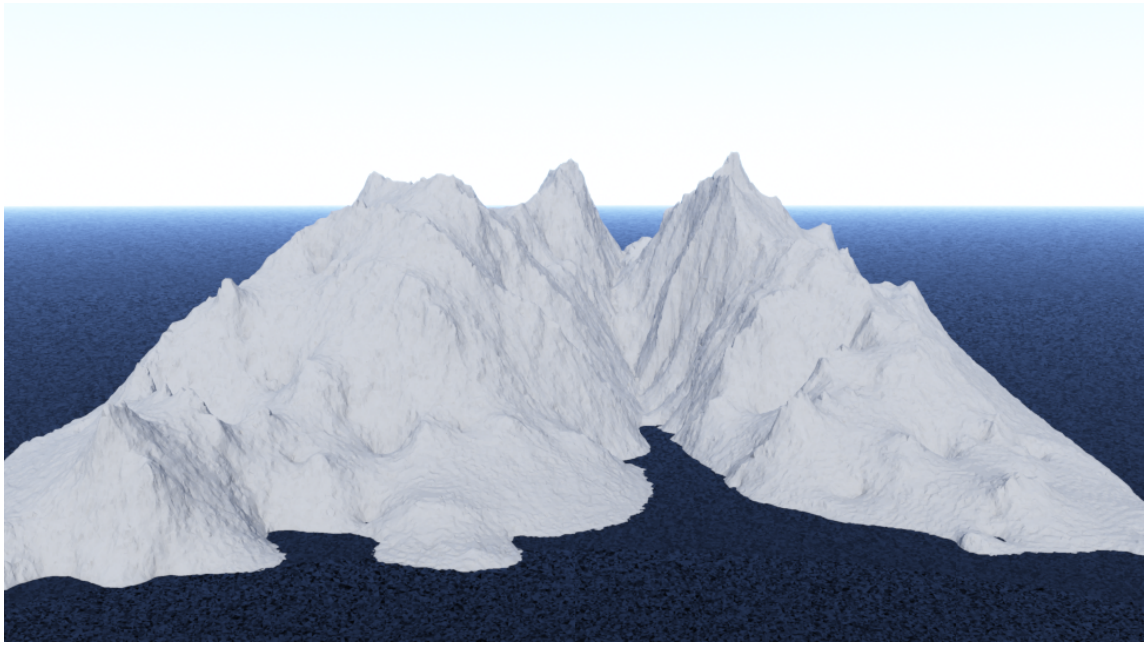


(b) Closeup of Terrain D in Figure 38b

Figure 46: We employ biased random walks on the primary curves of a curve network to generate a plateau with smooth ridges, which is hard to get from the approach of Rusnell et al. [49].

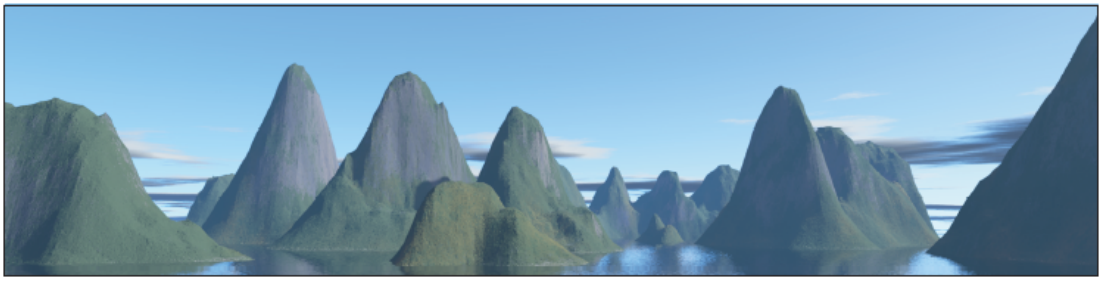


(a) Synthetic terrain from Hnaidi et al. [23]

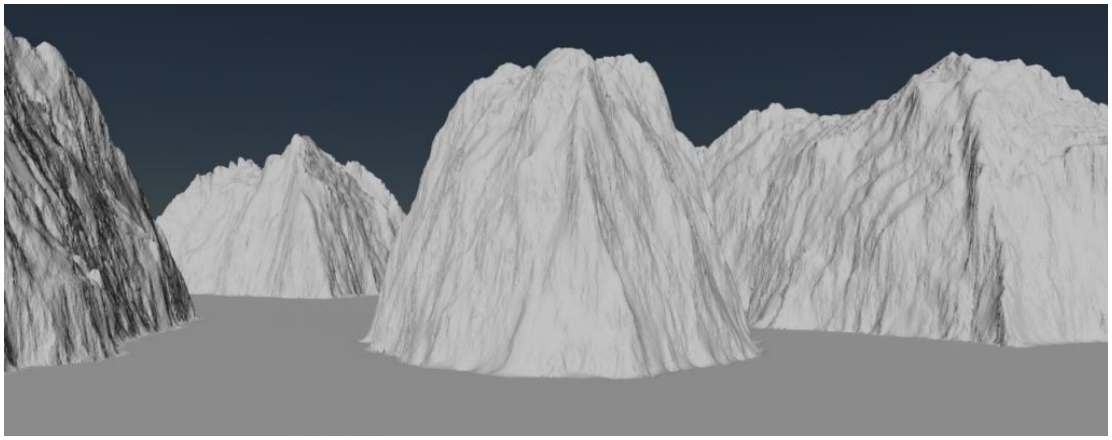


(b) Synthetic terrain from our method

Figure 47: The approach of Hnaidi et al. [23] constructs terrain based on feature curves but the results look more homogeneous than ours, in that we can get sharper peaks and flatter surfaces



(a) Synthetic terrain from Gain et al. [19]



(b) Synthetic terrain from our method

Figure 48: In the resulting terrains of Gain et al. [19], there are no surface details but we can add procedural detail around mountain peaks.

In this section we compare our algorithm with some of the related terrain synthesis approaches described in Chapter 2. Because our approach is not example-based or erosion-based we do not make comparisons with the techniques that fall into these categories. Our procedural approach for constructing curve networks is influenced by the work of Rusnell et al. [49]. We also compare our approach with the sketch-based approaches of Hnaidi et al. [23] and Gain et al. [19], since they are modern techniques for generating controllable terrain.

The criteria for evaluating user control over resulting terrains is the ability for terrain feature placement. All four methods provide user control to some extent. However, in the approach of Gain et al. the control is restricted to the sketching of terrain silhouettes. In the approach of Hnaidi et al. the user can also specify terrain features such as riverbeds, cliffs, and ridges. Rusnell et al. also provides the user the ability to specify the placement of ridges and peaks. Our way of providing user control is similar to the approach of Rusnell

et al. We can also control intermediate-scale features such as shape of ridges connected to peaks in the terrain, by making changes to the profile of the curves in a curve network. We employ biased random walks with custom PDFs to construct fBm profiles with distinctively varied shapes.

A shortcoming in the method of Rusnell et al. is the occasional shortcuts that happen to some input feature points along a Dijkstra's pass, which results in an uncertainty of keeping all feature points. Due to the nature of Dijkstra's path planning algorithm, shortcuts occur to source nodes whose cost through the shortest paths is smaller than the initial cost if the edge weights are not set carefully in the graph. We separate height profile assignment from feature map construction to prevent such shortcuts from happening. We use a similar distance-based approach to the one presented by Rusnell et al., but we finalize the placement of different kinds of feature points on separate stages of curve network generation to avoid unwanted shortcuts in the paths with higher priorities: we first finalize the paths of valley borders, then the placement of primary ridges curves, and lastly the placement of paths for secondary ridge curves. Rusnell et al. employ sketched profiles to assign heights to points outwards from feature points. Their sketched profiles have simpler characteristics than the random walk profiles that we assign to curves. We illustrate this difference by comparing two plateau landscapes, one constructed by Rusnell et al., in figure 46a, and another by our approach, in Figure 46b. Note that the plateau in Figure 46b is a closeup of Terrain D in Figure 38b.

Hnaidi et al. [23] present a terrain synthesis approach that has strength in variety of formation, from generating hills and cliffs to ridge lines. They employ control curves to set positions and gradients for terrain features. Figure 47a shows a terrain with complex features from Hnaidi et al. where the constraints such as riverbed, ridge, and peak placement were defined based on sketched input. The curves in the approach of Hnaidi et al. are produced through sketching, whereas in our approach the curves are generated procedurally and the curve profiles are assigned based base profiles onto which we superimpose random walks. The resulting terrain from Hnaidi et al. in Figure 47a looks more homogeneous than ours in Figure 47b. The terrain in Figure 47b exhibits heterogeneous structures such as sharper peaks on the top of mountains and flatter surfaces on lower elevations of the terrain.

Gain et al. [19] present another sketch-based terrain synthesis method. They model terrains based on sketches of silhouette curves. User can also modify boundary curves surrounding the silhouettes. The terrain from Gain et al. in Figure 48a, has no structural detail

off the ridges and is completely smooth. The problem of getting smooth terrain shapes is not only apparent in this one image. They use a diffusion-based approach integrated in their algorithm, which constructs smooth terrain, but most real terrains are rough and detailed. They however, procedurally add small-scale noise as a post-process on top of terrain surfaces. Compared to the results from Gain et al. [19], our method produces more details which are integrated into the rest of the terrain. In figure 48 we illustrate a comparison between our method and the approach of Gain et al. Figure 48b shows many narrowly spaced near-vertical ridges which are procedurally added detail that we incorporated into our terrain surfaces, whereas Figure 48a shows no such detail in the terrain from Gain et al.

Table 3: Comparison to related work

Method	Structural detail	User control	User effort	Computer effort
Rusnell et al. [49]	✓	✓	✓✓	✓
Hnaidi et al. [23]	✓	✓✓	✓	✗
Gain et al. [19]	✗	✓	✓	✗
Our method	✓✓	✓	✓✓	✗

Table 3 summarizes the comparison of our method with the aforementioned methods based on our judgment, showing where our work stands with respect to the state of the art. The method of Gain et al. has control over the placement of terrain silhouettes but lacks structural detail off the silhouettes. The work of Hnaidi et al. provides control over the placement of a variety of terrain features, but generating complex and detailed terrains demands a high degree of user involvement. The approach of Rusnell et al. constructs rough terrain with little user intervention and provides control over the placement of peaks and ridges. Compared to the work of Rusnell et al., our method provides better high level control over the placement of valleys and peaks which affect the elevation in the ridges. Also since the process of profile assignment and curve network generation in our method are separated, a user can re-evaluate the appearance of terrain containing the same infrastructure of ridges, valleys, and peaks in a curve network but with different types of random walk profiles with custom PDFs.

Making a fair comparison of the computer effort between our method and the sketch-based methods of Gain et al. and Hnaidi et al. is not possible since they do not provide

complete timing statistics for their methods. However, we estimate that their computer effort is high since both methods employ a discrete Poisson solver to interpolate the surface between the sketched strokes. In contrast, we employ MVC using the algorithm by Hormann and Floater [25]. Farbman et al. [15] provide an approximation to MVC interpolation and report that their technique performs much faster than Poisson interpolation.

Table 4: Characterization of inputs for our approach and the related methods

Method	Large-scale control	Detail
Rusnell et al. [49]	silhouette strokes	Perlin noise
Hnaidi et al. [23]	feature strokes	Perlin noise
Gain et al. [19]	few types of feature strokes	edge weights, possibly random
Our method	peaks, valleys, and ridges	PDFs (heights), edge weights (paths)

We characterize the input for our approach and the related methods in Table 4. Hnaidi et al. and Gain et al. use Perlin noise and Rusnell et al. use random edge weights to get fine-scale detail. We explored more ways to add structural detail: one by finding irregular paths for ridges, using random edge weights, and another way by assigning rough height profiles for the ridges using custom PDFs. Amongst the four techniques, the approach of Hnaidi et al. cover the widest variety of large-scale feature types in their resulting terrains.

Our main focus in this research was to get structural detail in the resulting terrains. Rusnell et al. produce rough terrain, but have no intermediate structural detail of the kind we add by assigning random walks to curves in the network. The sketch-based techniques of Gain et al. and Hnaidi et al. use Perlin noise to add roughness to the synthetic terrains. However, Perlin is a trivial noise that is isotropic and self-similar throughout the terrain and adds no structural detail. The detail present in the aforementioned sketch-based techniques is taken from the sketched strokes of the placement of terrain features, meaning that to get more complex terrains, the user needs to add more input strokes. In our method the user effort for creating detailed terrain is much less since the curves and the height profiles are procedurally generated.

The major strength in our algorithm is that we can add procedural detail by adding more feature curves and higher complexity than people normally sketch. Also, a curve network representation enables control over the large-scale features of resulting terrains; changing the elevation of a single peak point can immediately alter the height of the points

in a mountain region containing the peak, or changing the profile of a single primary curve alters the height of all the secondary curves attached to the primary curve and all the patches in the vicinity of the affected curves. Therefore, based on the hierarchy imposed by a curve network we can control or edit the elevation of points in the terrain.

4.6 Limitations

In this section we briefly discuss the shortcomings of our approach. Our method does not allow a complete catalog of terrain features to be created, due to the limitations of heightfields and the special cases that we do not cover. Also, controllability and performance is not as good as it could be.

Rivers have monotonically decreasing profiles and branch in until they reach a body of water such as the ocean. Our proposed method does not account for river paths. In future, a feasible solution for including river paths in terrain synthesis using curve networks, would be by finding least-cost paths to a couple of points on the valley border and by assigning monotonically decreasing profiles to these paths.

Our current approach for constructing barchans and volcanic mountains is by CSG operations but we would prefer direct synthesis. The hierarchy in our method gives higher priority to the primary curves than to the secondary curves meaning that the height of primary curve points affects the elevation in the secondary curves connected to a primary curve. This hierarchy is suitable for constructing different types of mountain ranges. Changing the curve hierarchy can help generate a wider range of mountains such as volcanic mountains where the elevation in contours around a peak affects the height of ridges connected vertically to the contours.

Since the output in our method is a heightfield, our approach is incapable of synthesizing caves or arches. Also, random walk signals do not cover all types of shapes since there is no ability for coordinated changes. Therefore, our algorithm cannot model shapes such as bubbly lava which have local coherence.

In our algorithm, we construct profiles based on input PDFs. A more direct approach is to be able to retrieve PDFs from silhouette or ridge samples. This alternative approach can contribute to an example-based method which detects PDF profiles from the ridges and silhouettes of sample images in a database and then constructs synthetic terrains similar to the input terrain.

Our method adds structural ridge detail in terrain but the patch surfaces are smoothly

interpolated. We would like to have finer-scale roughness. The sketch-based methods add Perlin noise to get surface roughness. Our terrains are segmented into separate interconnected patches. One way to add structural roughness using our technique is to further subdivide the patches before interpolating each patch.

We need to improve the performance for our algorithm to be suitable for real-time terrain synthesis or editing. Our patch interpolation is based on the implementation of Hormann and Floater [25]; we can accelerate the patch interpolation by employing the approximate algorithm proposed by Farbman et al. [15].

4.7 Summary

In this chapter we discussed different ways in which we control the shape of terrains, from altering the elevation of valley border points to changing the random walk PDFs of curves in the network. Then we compared synthetic terrains from our method against a set of real-world photos. We showed that we can use base profiles for the curves to generate mountain ranges such as karst towers and volcanic mountain ranges and that we can use CSG operations to construct complicated terrains such as barchans, volcanos, and craters from dome-shaped terrains.

We also showed two different ways to construct curve networks from ridge inputs. In one way we grow interconnected path of curves around input ridges, and in another way we only keep a scattered set of peaks from the input ridge to construct a curve network. In the former curves, can connect to any point on the input ridge, whereas in the later, the curves only connect to the peaks in the ridge. Therefore, we can select any of these two approaches for creating a curve network around a ridge input, depending on how we want the shape of details outwards from the ridge to look like.

Many existing sketch-based terrain synthesis methods use diffusion-based algorithms to interpolate surfaces around the feature curves. We can also employ MVC interpolation since the curve networks decompose the terrain into series of closed patches. MVC allows us to interpolate each patch locally and the smaller the patches are, the rougher the terrain surface gets.

We showed that with the same curve network structure, we can construct different terrain shapes by just changing the random walk PDFs applied for populating heights to curve elements. The user draws PDFs to create new types of random walks. Our resulting terrains show that compared to Rusnell et al., we can get better control over the placement

and elevation of procedurally added features such as valleys and ridges. In our approach the user can control placement and appearance of peaks, and specify profiles for the curves in the network. We superimpose biased random walks on basic profiles to get varied and realistic terrains.

Chapter 5

Conclusion

In this research we explored procedural techniques for the synthesis of detailed yet controllable terrain. We presented a method with control over feature placement and constructed structural detail in the gaps between sparse input of peaks. We first construct a curve network which partitions space into small patches and consists of series of interconnected paths, called curve elements. We add roughness to the height profiles of each curve element individually based on random walk. Then we interpolate each patch in the curve network based on the local paths bounding the patch. Our method separates the problem of curve network construction from height profile assignment to each curve. Therefore, we can use the same curve network to produce different kinds of terrain shapes by modifying the 1D profiles of curves.

Our method controls both large-scale and fine-scale detail of terrain models. We can control the placement of peaks and other terrain properties such as roughness and ridge shapes and we can produce structural and prominent ridge detail around input peaks and ridges. We generate realistic rough terrains by increasing the density of curves or by making rough random walk curve profiles in the gaps between the scattered input peaks. We can control the shape of ridges outwards from peaks by specifying base profiles for curves onto which we superimpose smaller-scale random walks. Since we employ mean-value coordinations for patch interpolation, we can get surfaces with arbitrary resolution and can get local evaluation of patch heights.

In our work, we made the following contributions:

- We present a hybrid sketch-based and procedural modeling technique which allows the user to place peak positions and to optionally specify custom PDFs for the random walk profiles.

- We present a method that procedurally constructs a curve network with irregular paths given the placement of a sparse set of peaks. The procedurally generated curves in the network are placeholders for the location of ridges.
- We also procedurally generate profile shapes from custom random walks based on the provided PDFs which results in added structural detail on the ridges of the terrains.
- We introduce curve networks for terrain synthesis. The hierarchy present in the curve network, along with the ability to locally evaluate heights in the patches, makes our method a suitable approach for terrain synthesis and editing.

Our method has some shortcomings. The curve network does not contain a separate type of curves to represent river paths. Also, since the output to our method is a height-field we cannot model features such as arches or caves. We cannot use random walks for generating profiles that have coordinated height changes such as the ones seen in lavas. In addition, although we can generate structural terrain detail but since the surface of each patch is smoothly interpolated based on its surrounding curves, we do not get small-scale roughness throughout the terrain surface. To have a real-time application in terrain synthesis and editing, we need to improve the performance in the patch interpolation stage of our approach. Lastly, to enhance the user effort, we need to find a technique that automatically retrieves the PDFs from sample random walks rather than having the user explicitly specify the custom PDFs.

There is a degree of uncertainty inherent from the randomness in our distance-based procedural technique. Because of the random elements in our algorithm, the outcome is not necessarily predictable. If the input is ridges, then we either pass the ridge profiles and get local peaks and lose exact profile between the local peaks, or we pass the entire ridge profile and lose the exact paths back to mountain peaks.

5.1 Future work

- We demonstrated strategies to construct paths outwards from peaks. A complementary approach is to explore other path planning strategies to incorporate river networks into curve networks.
- To able to construct a family of profiles similar to a sketched ridge or silhouette, we ask a user to sketch a PDF. Instead we should explore ways to automatically retrieve

Figure 49: Generating coarse paths [65]

a PDF from a sketched ridge profile.

- We presented curve networks for terrain synthesis. Next we need to show that we can utilize curve networks for terrain editing as well. We need to implement a sketch-based interface, where the user selects a curve and changes profiles accordingly and then the system locally evaluates the affected patches again using MVC.

- As we see in figure 49, we can use path planning to construct larger resolution of existing paths [65]. An interesting future direction is to show we can zoom in on existing curve networks by generating high-resolution paths for each curve. Since we can also generate arbitrary resolution of the terrains with MVC patch interpolation, zooming in on a curve network can translate into getting higher resolution terrain for the selected patches in the view of a camera.

- No existing research has focused on procedurally controlling the roughness of different patches in a terrain. We proposed an approach that based on the size of patches and the roughness of patch borders, controls the global roughness of terrain. Schaefer et al. [51] explained regular lofting, which interpolates a smooth surface inside patches in a curve network using b-splines. Another area for further research is to explore procedural techniques that produce rough lofting of the patches in a curve network. Rough lofting would be a method in which we can add roughness to different patches locally.

This thesis covered the topic of terrain synthesis. Although further enhancement is needed, our work is a step forward and the terrains from our approach have more detailed structure and are more convincing than other recent synthetic terrains.

List of References

[1] Autodesk Inc. Maya. http://www.autodesk.com/maya. [Accessed: 2014-10-30].

[2] Bedřich Beneš and Rafael Forsbach. Layered data representation for visual simulation of terrain erosion. In *Computer Graphics, Spring Conference on, 2001.*, pages 80–86. IEEE, 2001.

[3] Bedřich Beneš and Rafael Forsbach. Visual simulation of hydraulic erosion. *Journal of WSCG*, 10(1–3):79–86, 2002.

[4] Blender Foundation. Blender. http://www.blender.org/. [Accessed: 2014-10-30].

[5] Emilio Vital Brazil, Ives Macedo, Mario Costa Sousa, Luiz Henrique de Figueiredo, and Luiz Velho. Sketching Variational Hermite-RBF implicits. In *Proceedings of the Seventh Sketch-Based Interfaces and Modeling Symposium*, pages 1–8. Eurographics Association, 2010.

[6] John Brosz, Faramarz Samavati, and Mario Costa Sousa. Terrain synthesis by-example. In *Advances in Computer Graphics and Computer Vision*, pages 58–77. Springer, 2007.

[7] Armin Bunde and Shlomo Havlin. *Fractals and disordered systems*. Springer-Verlag New York, Inc., 1991.

[8] chriscom (Flickr user). Everest (left) and Nuptse seen from the Kala Pathar. https://www.flickr.com/photos/chrigu/206608437. [Accessed: 2014-10-30].

[9] Jonathon Doran and Ian Parberry. Controlled procedural terrain generation using software agents. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(2):111–119, 2010.

[10] Vladimir Alves dos Passos and Takeo Igarashi. LandSketch: A first person point-of-view example-based terrain modeling approach. In *Proceedings of the International Symposium on Sketch-Based Interfaces and Modeling*, pages 61–68. ACM, 2013.

[11] David Ebert, Forest Kenton Musgrave, Darwyn Peachey, Ken Perlin, and Steven Worley. *Texturing & modeling: A procedural approach*. Morgan Kaufmann, 2003.

[12] Alexei Efros and William Freeman. Image quilting for texture synthesis and transfer. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 341–346. ACM, 2001.

[13] Alexei Efros and Thomas Leung. Texture synthesis by non-parametric sampling. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 2, pages 1033–1038. IEEE, 1999.

[14] El Guanche. Isle of Navarino, Southern Chile. `https://www.flickr.com/photos/el_guanche/4080885249/in/photolist-3aikEe-4wPaQE-7dBB1v-4wPdKf`. [Accessed: 2014-10-30].

[15] Zeev Farbman, Gil Hoffer, Yaron Lipman, Daniel Cohen-Or, and Dani Lischinski. Coordinates for instant image cloning. In *ACM Transactions on Graphics (TOG)*, volume 28, page 67. ACM, 2009.

[16] Michael Floater. Mean value coordinates. *Computer aided geometric design*, 20(1):19–27, 2003.

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

[18] Frank Kovalchek. Na Pali coast. `https://www.flickr.com/photos/72213316@N00/4257969869`. [Accessed: 2014-10-30].

[19] James Gain, Patrick Marais, and Wolfgang Straßer. Terrain sketching. In *Proceedings of the 2009 symposium on Interactive 3D graphics and games*, pages 31–38. ACM, 2009.

[20] Jean-David Génevaux, Eric Galin, Eric Guérin, Adrien Peytavie, and Bedřich Beneš. Terrain generation using procedural models based on hydrology. *ACM Transactions on Graphics (TOG)*, 32(4):143, 2013.

[21] Rafael Gonzalez and Richard Woods. *Digital image processing*. Prentice Prentice Hall, 2008.

[22] Hanumann (Flickr user). Karst mountains, Xingping Scenic Area, Guilin, China. `https://www.flickr.com/photos/40883475@N00/5273963020`. [Accessed: 2014-10-30].

[23] Houssam Hnaidi, Eric Guérin, Samir Akkouche, Adrien Peytavie, and Eric Galin. Feature based terrain generation using diffusion equation. In *Computer Graphics Forum*, volume 29, pages 2179–2186. Wiley Online Library, 2010.

[24] Donald Hoffman. *Visual intelligence: How we create what we see*. WW Norton & Company, 2000.

[25] Kai Hormann and Michael Floater. Mean value coordinates for arbitrary planar polygons. *ACM Transactions on Graphics (TOG)*, 25(4):1424–1441, 2006.

[26] Takeo Igarashi, Satoshi Matsuoka, and Hidehiko Tanaka. Teddy: A sketching interface for 3D freeform design. In *ACM SIGGRAPH 2007 courses*, page 21. ACM, 2007.

[27] Tao Ju, Scott Schaefer, and Joe Warren. Mean value coordinates for closed triangular meshes. In *ACM Transactions on Graphics (TOG)*, volume 24, pages 561–566. ACM, 2005.

[28] Alex Kelley, Michael Malin, and Gregory Nielson. Terrain simulation using a model of stream erosion. In *ACM SIGGRAPH Computer Graphics*, volume 22, pages 263–268. ACM, 1988.

[29] Ken Lund. Fire Canyon, Nevada. `https://www.flickr.com/photos/kenlund/11004255604`. [Accessed: 2014-10-30].

[30] John Lewis. Generalized stochastic subdivision. *ACM Transactions on Graphics (TOG)*, 6(3):167–190, 1987.

[31] Benoit Mandelbrot. The fractal geometry of nature/revised and enlarged edition. *New York, WH Freeman and Co., 1983, 495 p.*, 1, 1983.

[32] Benoit Mandelbrot. Squig sheets and some other squig fractal constructions. *Journal of Statistical Physics*, 36(5-6):519–539, 1984.

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

[34] Forest Kenton Musgrave. *Methods for realistic landscape imaging*. PhD thesis, Yale University, 1993.

[35] Forest Kenton Musgrave, Craig Kolb, and Robert Mace. The synthesis and rendering of eroded fractal terrains. In *ACM SIGGRAPH Computer Graphics*, volume 23, pages 41–50. ACM, 1989.

[36] Kenji Nagashima. Computer generation of eroded valley and mountain terrains. *The Visual Computer*, 13(9):456–464, 1998.

[37] NASA. Barchan dunes on Mars. `http://apod.nasa.gov/apod/ap120422.html`. [Accessed: 2014-10-30].

[38] NASA's Earth Observatory web site. Islands of the four mountains. `http://earthobservatory.nasa.gov/IOTD/view.php?id=82588`. [Accessed: 2014-10-30].

[39] Jacob Olsen. Realtime procedural terrain generation. Technical report, Department of Mathematics and Computer Science (IMADA), University of Southern Denmark, 2004.

[40] Luke Olsen, Faramarz Samavati, Mario Costa Sousa, and Joaquim Jorge. Sketch-based modeling: A survey. *Computers & Graphics*, 33(1):85–103, 2009.

[41] Patrick Pérez, Michel Gangnet, and Andrew Blake. Poisson image editing. In *ACM Transactions on Graphics (TOG)*, volume 22, pages 313–318. ACM, 2003.

[42] Ken Perlin. An image synthesizer. *ACM SIGGRAPH Computer Graphics*, 19(3):287–296, 1985.

[43] Adrien Peytavie, Eric Galin, Jérôme Grosjean, and Stephane Merillou. Arches: A framework for modeling complex terrains. In *Computer Graphics Forum*, volume 28, pages 457–467. Wiley Online Library, 2009.

[44] Adrien Peytavie, Eric Galin, Jerome Grosjean, and Stephane Merillou. Procedural generation of rock piles using aperiodic tiling. In *Computer Graphics Forum*, volume 28, pages 1801–1809. Wiley Online Library, 2009.

[45] Planetside Software. Terragen. `http://planetside.co.uk/`. [Accessed: 2014-10-30].

[46] Przemyslaw Prusinkiewicz and Mark Hammel. A fractal model of mountains and rivers. In *Graphics Interface*, volume 93, pages 174–180. Canadian Information Processing Society, 1993.

[47] Rodrigo Soldon. Dedo de Deus. `https://www.flickr.com/photos/soldon/12746825585`. [Accessed: 2014-10-30].

[48] Brennan Rusnell. Feature-rich distance-based terrain synthesis. Master's thesis, University of Saskatchewan, Saskatoon, 2009.

[49] Brennan Rusnell, David Mould, and Mark Eramian. Feature-rich distance-based terrain synthesis. *The Visual Computer*, 25(5-7):573–579, 2009.

[50] VB Sapozhnikov and Vladimir Nikora. Simple computer model of a fractal river network with fractal individual watercourses. *Journal of Physics A: Mathematical and General*, 26(15):L623, 1993.

[51] Scott Schaefer, Joe Warren, and Denis Zorin. Lofting curve networks using subdivision surfaces. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 103–114. ACM, 2004.

[52] Allan Shimada. Alaska, Aleutian islands. `https://www.flickr.com/photos/noaaphotolib/5277232633`. [Accessed: 2014-10-30].

[53] Peter Shirley, Michael Ashikhmin, and Steve Marschner. *Fundamentals of computer graphics*. CRC Press, 2009.

[54] Jos Stam. *A multi-scale stochastic model for computer graphics*. PhD thesis, University of Toronto, 1991.

[55] Ondřej Šťava, Bedřich Beneš, Matthew Brisbin, and Jaroslav Křivánek. Interactive terrain modeling using hydraulic erosion. In *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 201–210. Eurographics Association, 2008.

[56] Stefanos Nikologianis. Brakk Towers, Karakoram, Pakistan. `https://www.flickr.com/photos/snikologiannis/5981012968`. [Accessed: 2014-10-30].

[57] Richard Szeliski and Demetri Terzopoulos. From splines to fractals. In *ACM SIGGRAPH Computer Graphics*, volume 23, pages 51–60. ACM, 1989.

[58] Flora Ponjou Tasse, Arnaud Emilien, Marie-Paule Cani, Stefanie Hahmann, and Adrien Bernhardt. First person sketch-based terrain editing. In *Proceedings of the 2014 Graphics Interface Conference*, pages 217–224. Canadian Information Processing Society, 2014.

[59] Soon Tee Teoh. Riverland: An efficient procedural modeling system for creating realistic-looking terrains. In *Advances in Visual Computing*, pages 468–479. Springer, 2009.

[60] tlindenbaum (Flickr user). Grand Tetons National Park, Wyoming, United States. `https://www.flickr.com/photos/lindenbaum/330315184`. [Accessed: 2014-10-30].

[61] Gilles Valette, Stéphanie Prévost, Laurent Lucas, and Joël Léonard. A dynamic model of cracks development based on a 3D discrete shrinkage volume propagation. In *Computer Graphics Forum*, volume 27, pages 47–62. Wiley Online Library, 2008.

[62] Li-Yi Wei and Marc Levoy. Fast texture synthesis using tree-structured vector quantization. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 479–488. ACM Press/Addison-Wesley Publishing Co., 2000.

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

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

[65] Ling Xu and David Mould. Modeling dendritic shapes using path planning. In *Proceedings of GRAPP 2007*, page 2936. INSTICC, 2007.

[66] yedman (Flickr user). Fitz Roy. `https://www.flickr.com/photos/27101885@N08/4608060304`. [Accessed: 2014-10-30].

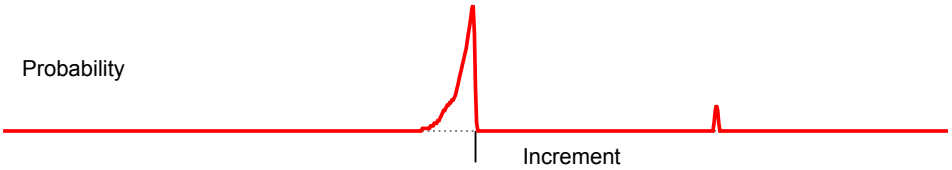
[67] Howard Zhou, Jie Sun, Greg Turk, and James Rehg. Terrain synthesis from Digital Elevation Models. *Visualization and Computer Graphics, IEEE Transactions on*, 13(4):834–848, 2007.

Appendix A

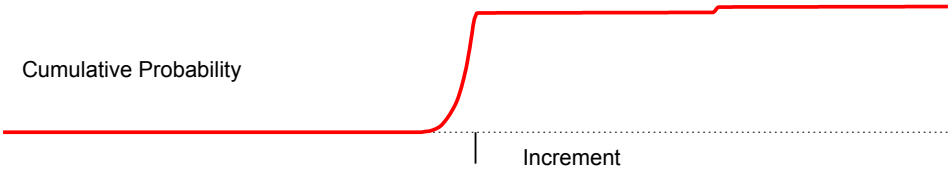
Random walk profiles from different PDFs

In Chapter 3 we explained how to construct biased random walks from sketched PDFs. Figures 50 through 60 in this chapter show a set of biased random walks generated from sketched PDFs. In these figures we also include the fBm sum of the four different random walks as well as the CDF. The figures presented in this chapter, show that altering the PDFs results in the generation of biased random walks with distinct shapes.

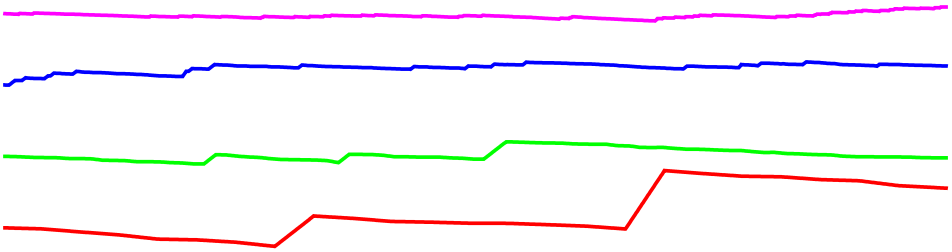
For each figure in this chapter, part (a) shows the sketched PDF, part (b) illustrates the derived CDF, part (c) depicts four random walk octaves generated from the CDF, and part (d) displays the fBm sum of the four example octaves. The x axis for the PDFs and the CDFs indicates the random increments, and the y axis represents the probability and the cumulative probability, respectively. For the random walk fBms and the octaves, the x axis shows the spatial displacement and the y axis depicts height. Also, the vertical line in the middle of the x axis of both the PDFs and the CDFs marks the zero coordinate of the axis.



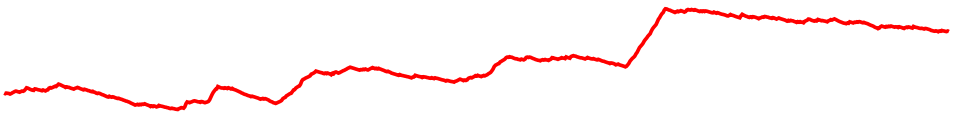
(a) The sketched PDF



(b) The derived CDF

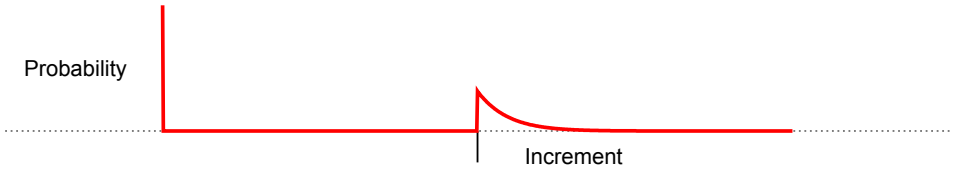


(c) Four example octaves with random increments from the provided PDF

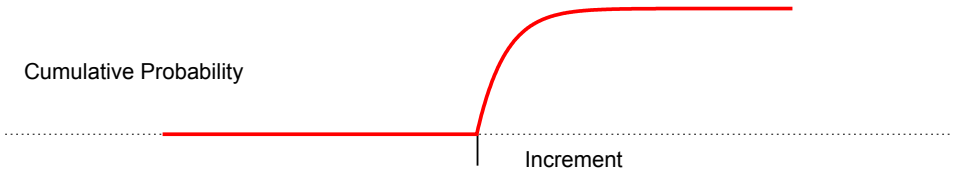


(d) The fBm sum of the four octaves

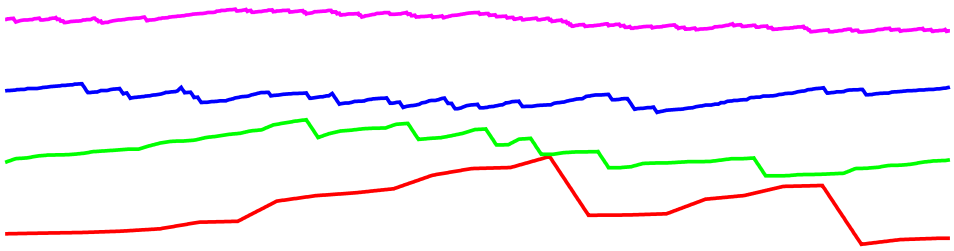
Figure 50: Example 1



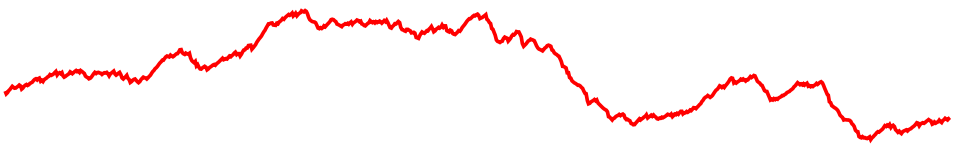
(a) The sketched PDF



(b) The derived CDF

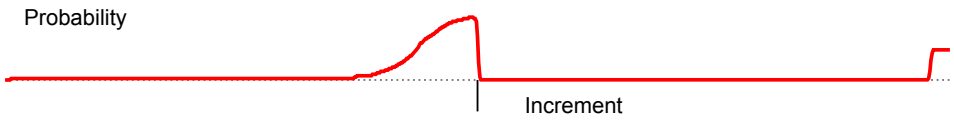


(c) Four example octaves with random increments from the provided PDF

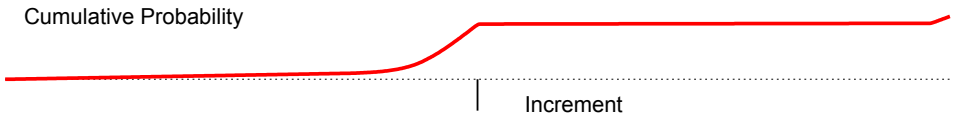


(d) The fBm sum of the four octaves

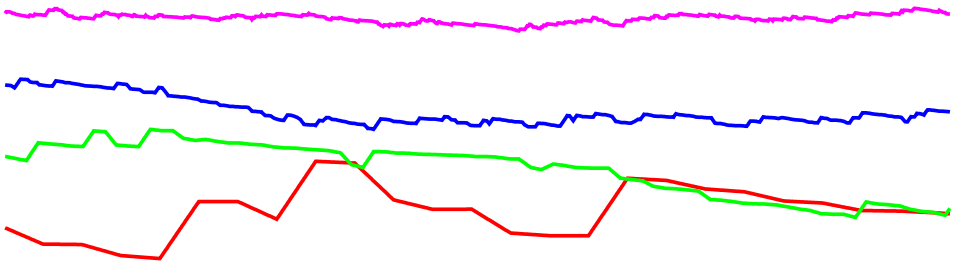
Figure 51: Example 2



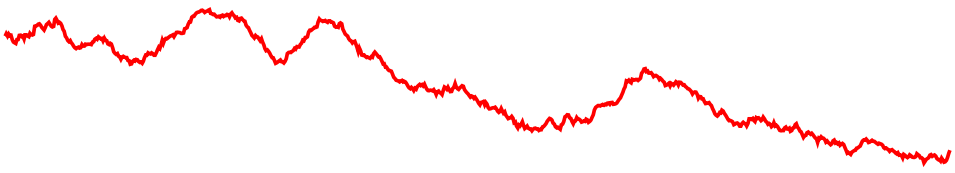
(a) The sketched PDF



(b) The derived CDF

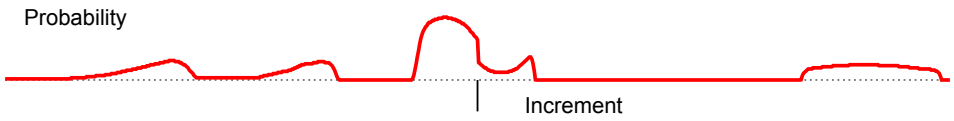


(c) Four example octaves with random increments from the provided PDF

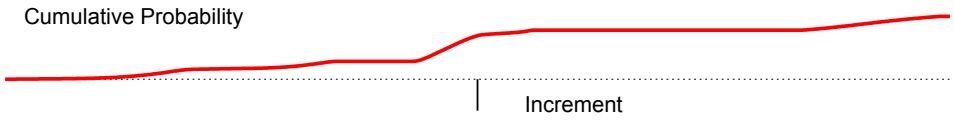


(d) The fBm sum of the four octaves

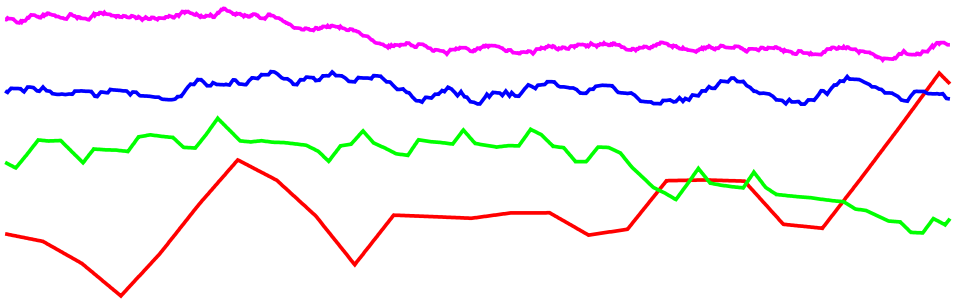
Figure 52: Example 3



(a) The sketched PDF



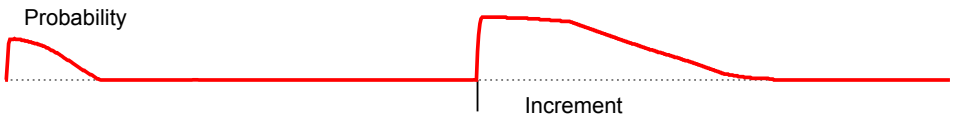
(b) The derived CDF



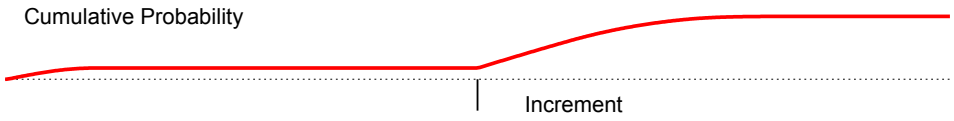
(c) Four example octaves with random increments from the provided PDF

(d) The fBm sum of the four octaves

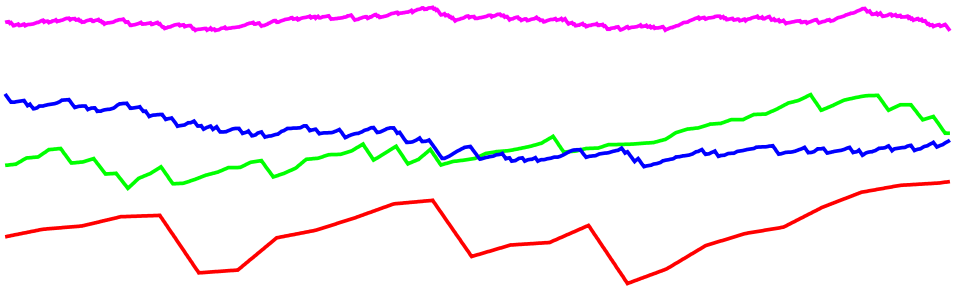
Figure 53: Example 4



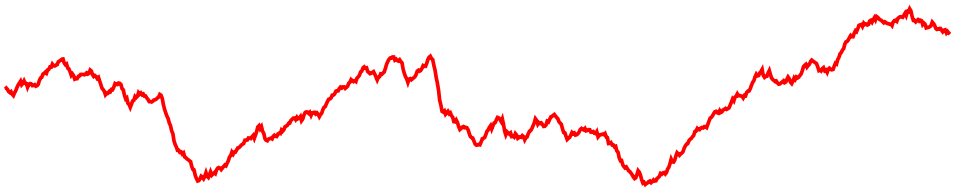
(a) The sketched PDF



(b) The derived CDF

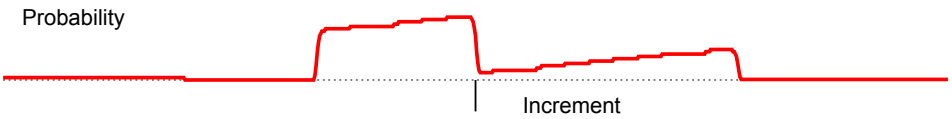


(c) Four example octaves with random increments from the provided PDF

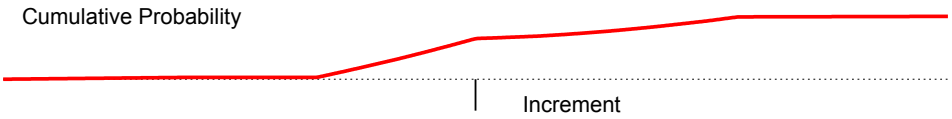


(d) The fBm sum of the four octaves

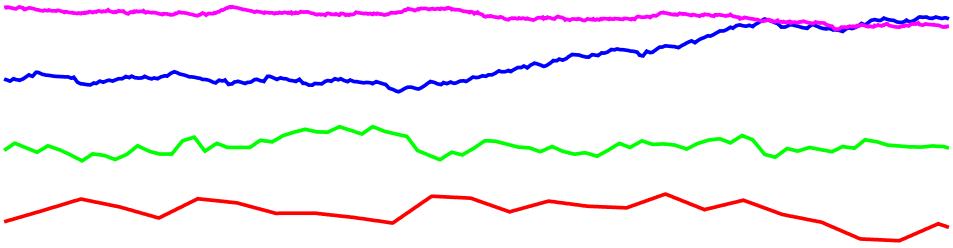
Figure 54: Example 5



(a) The sketched PDF



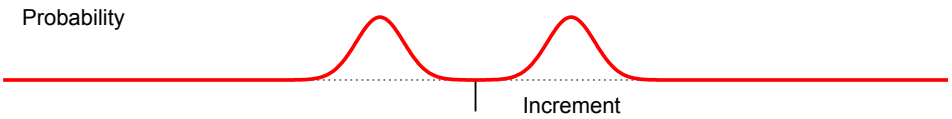
(b) The derived CDF



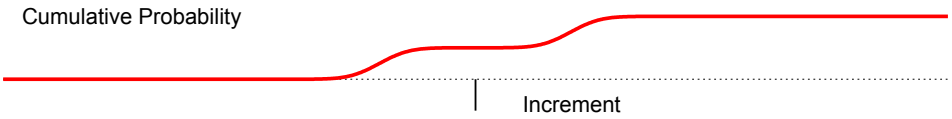
(c) Four example octaves with random increments from the provided PDF

(d) The fBm sum of the four octaves

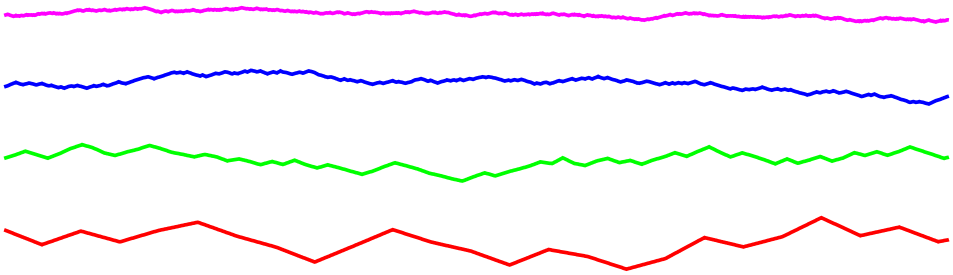
Figure 55: Example 6



(a) The sketched PDF



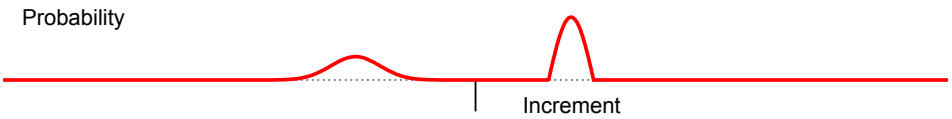
(b) The derived CDF



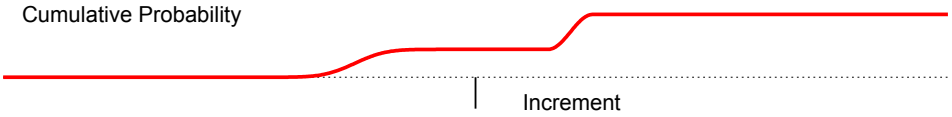
(c) Four example octaves with random increments from the provided PDF

(d) The fBm sum of the four octaves

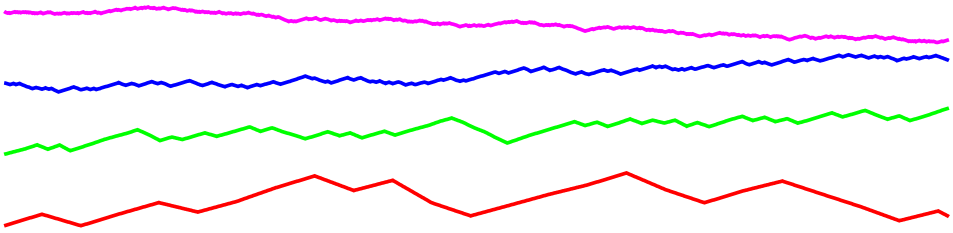
Figure 56: Example 7



(a) The sketched PDF



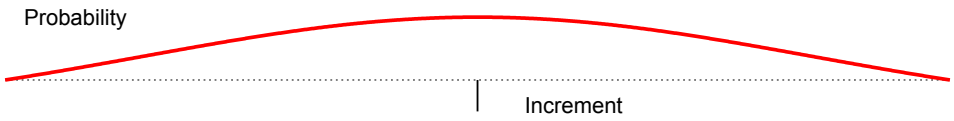
(b) The derived CDF



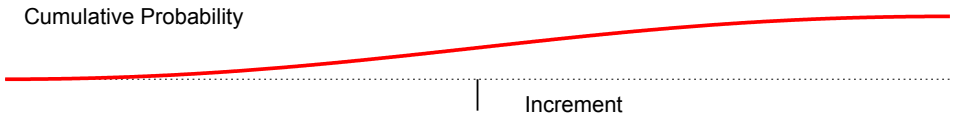
(c) Four example octaves with random increments from the provided PDF

(d) The fBm sum of the four octaves

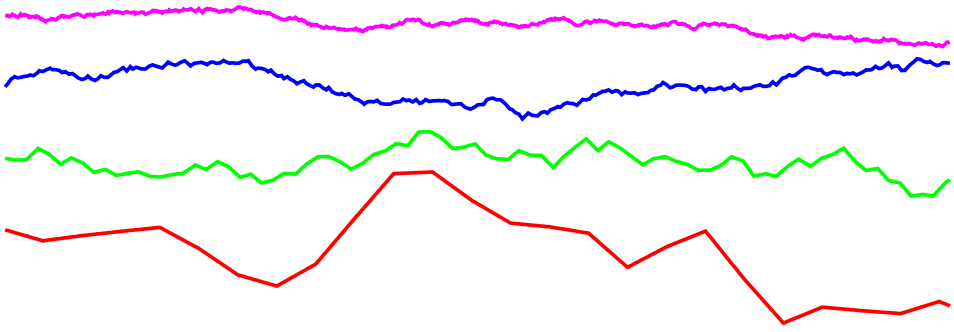
Figure 57: Example 8



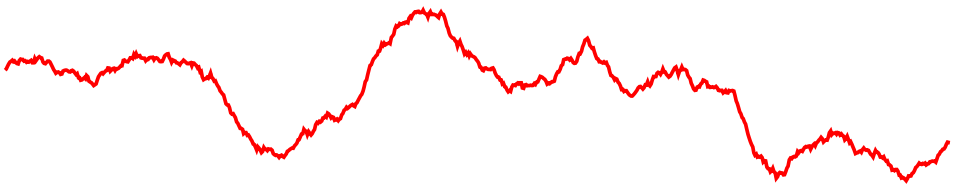
(a) The sketched PDF



(b) The derived CDF

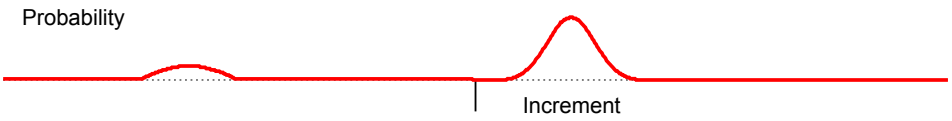


(c) Four example octaves with random increments from the provided PDF

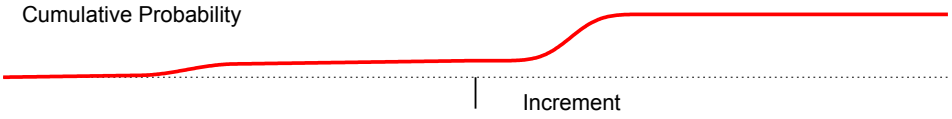


(d) The fBm sum of the four octaves

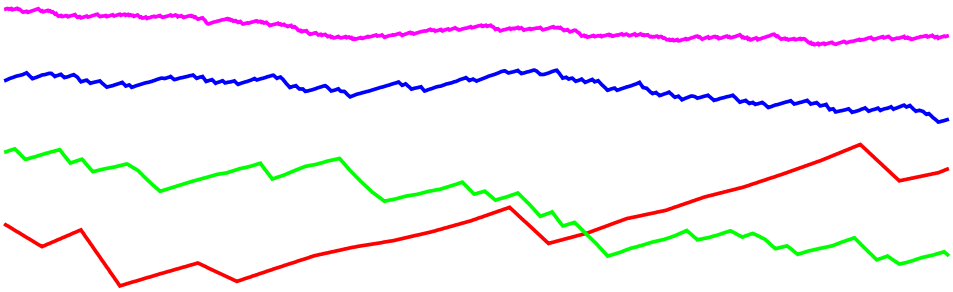
Figure 58: Example 9



(a) The sketched PDF



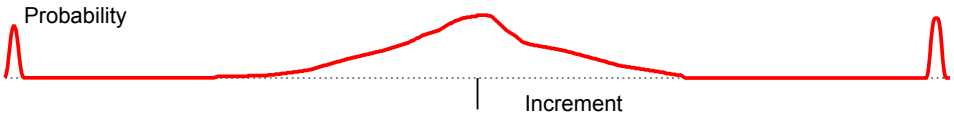
(b) The derived CDF



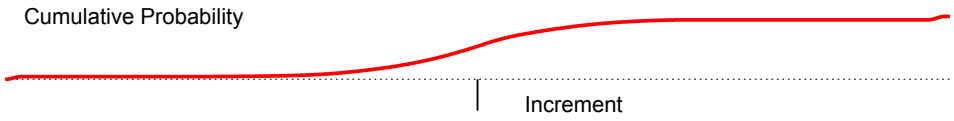
(c) Four example octaves with random increments from the provided PDF

(d) The fBm sum of the four octaves

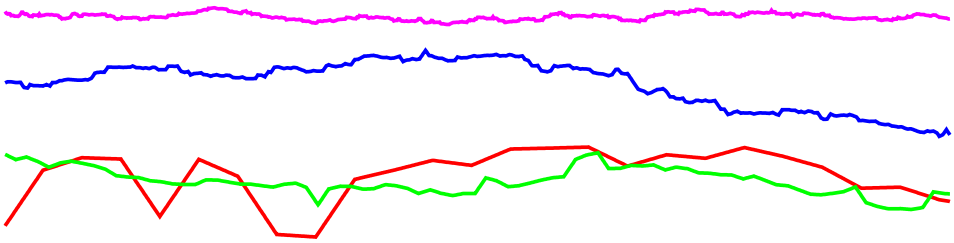
Figure 59: Example 10



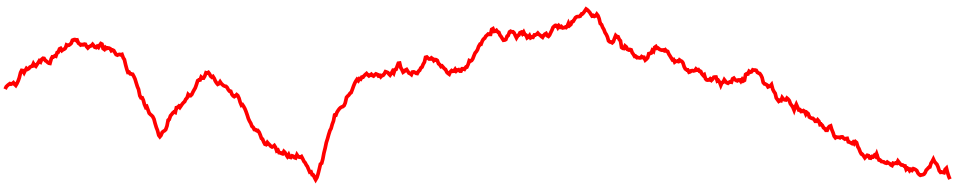
(a) The sketched PDF



(b) The derived CDF

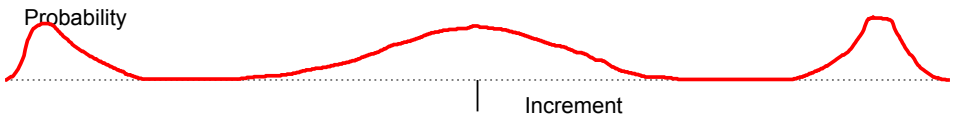


(c) Four example octaves with random increments from the provided PDF

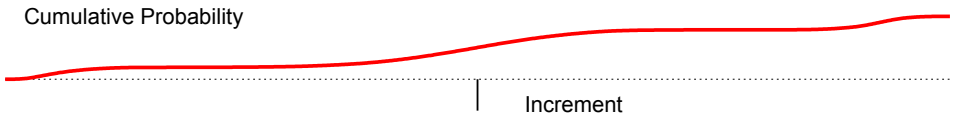


(d) The fBm sum of the four octaves

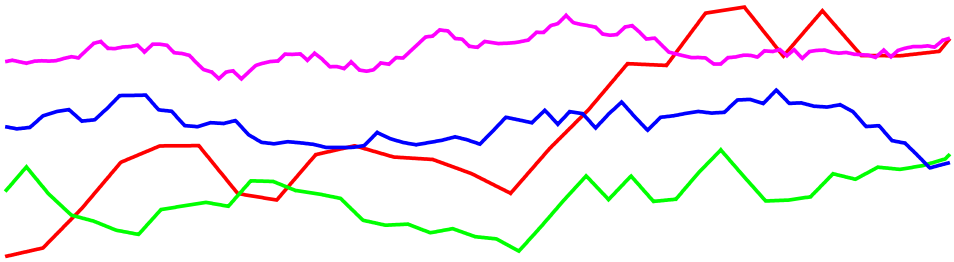
Figure 60: Example 11



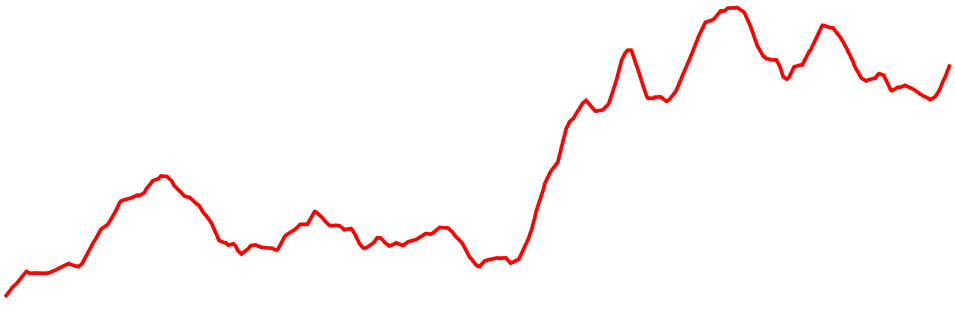
(a) The sketched PDF



(b) The derived CDF



(c) Four example octaves with random increments from the provided PDF



(d) The fBm sum of the four octaves

Figure 61: Example 12