Procedural Terrain Generation Based on Constraint Paths

THESIS

Presented in Partial Fulfillment of the Requirements for the Degree of Master of Science in
the Graduate School of The Ohio State University

By

Michael Andereck

Graduate Program in Computer Science and Engineering

The Ohio State University

2014

Master’s Examination Committee:

Professor Roger Crawfis, Advisor
  Professor Alan Price
  Professor Rephaniel Wenger
Abstract

Procedural terrain generation is a highly popular topic in computer science today with applications in video games, medical rehabilitation, land planning, and even military training. Many algorithms exist to create these terrains including fractal designs, physical simulations, and applying real-world data. These systems offer various levels of interactivity and user control. We introduce a constraint-based system for procedurally generating virtual terrains. The first constraint is based on user-designed paths which can be customized for the various needs of patients experiencing medical rehabilitation. Due to the specific needs of this application, we require that our terrain shape should not manipulate the heights specified by these paths. Additional constraints may be applied from real-world data, user-painted heights, and tile borders. Given a set of constraints our generative algorithm iteratively finds the best fitting terrain shape, interpolating between and beyond the specified points. With a combination of user interaction and faithful fitting of data, our algorithm provides a more friendly system for constrained virtual terrain generation.
Dedicated to my family...
Acknowledgements

I would like to thank my family for all their support through the years, my wife for putting up with me, Dr Roger Crawfis for helping to steer me in the right direction in my work, Alan Price and everyone at ACCAD for providing such a wonderful place to work during my years at OSU...
Vita

June 2003  
Dublin Coffman High School

June 2007  
B.S. Computer Science, Vanderbilt University

August 2007 to present  
GRA positions in The Computer Science and Engineering Department and The Advanced Computing Center for the Arts and Design, The Ohio State University

Summers 2009 to 2011  
Student Internships at Lawrence Livermore National Laboratory

Publications


Fields of Study

Major Field: Computer Science and Engineering
# Table Of Contents

Abstract .................................................................................. ii

Dedication ........................................................................ iii

Acknowledgements ................................................................ iv

Vita .................................................................................. v

Contents ........................................................................ vi

List of Figures ...................................................................... viii

Abbreviations .................................................................... ix

1 Introduction ...................................................................... 1
  1.1 Problem Statement ........................................................ 1
  1.2 Overview of Remaining Chapters ................................. 2

2 Review of Literature ........................................................ 3
  2.1 Generative Methods ....................................................... 3
  2.2 Using Real-World Data .................................................. 3
  2.3 Using Path Constraints ................................................... 4
  2.4 Motivation for Virtual Environments ............................... 5
  2.5 Tiling Methods ............................................................. 6

3 Path Generation .............................................................. 8
  3.1 Control Points .............................................................. 8
  3.2 Hermite Splines ........................................................... 9
  3.3 Character Position and Movement ................................. 10
  3.4 Constraint Points from Paths ......................................... 11
  3.4.1 Path Casting ........................................................... 12
  3.4.2 Sample Path Shapes ................................................... 13

4 Solving for Terrain Shape ............................................... 17
  4.1 Matching the Path ........................................................ 17
  4.2 Shaping The Terrain ..................................................... 18
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.2.1 Linear Least Squares</td>
<td>19</td>
</tr>
<tr>
<td>4.2.2 Corner Heights</td>
<td>21</td>
</tr>
<tr>
<td>4.3 Tiling</td>
<td>24</td>
</tr>
<tr>
<td>5 Other Constraints</td>
<td>27</td>
</tr>
<tr>
<td>5.1 Using Real-World Data</td>
<td>27</td>
</tr>
<tr>
<td>5.2 Painting Points</td>
<td>30</td>
</tr>
<tr>
<td>5.3 Combinations</td>
<td>34</td>
</tr>
<tr>
<td>5.4 Tile Borders</td>
<td>36</td>
</tr>
<tr>
<td>6 Implementation</td>
<td>38</td>
</tr>
<tr>
<td>6.1 Unity 3D</td>
<td>38</td>
</tr>
<tr>
<td>7 Conclusion</td>
<td>40</td>
</tr>
<tr>
<td>7.1 Conclusion</td>
<td>40</td>
</tr>
<tr>
<td>7.2 Future Work</td>
<td>41</td>
</tr>
</tbody>
</table>
List of Figures

3.1 Sample of a path using spheres as control points. ........................................... 9
3.2 A screenshot of the user walking through a built terrain. ............................... 11
3.3 Two mountains created from spiral paths ...................................................... 14
3.4 A single path can be used to shape a canyon and overlooks ......................... 15
3.5 The terrain shape from a large mountainside path. ........................................ 16

4.1 The terrain has been modified to match the desired path. ............................ 17
4.2 A compilation of the terrain shaping process ............................................... 20
4.3 Mismatched corner problem. ................................................................. 21
4.4 Four levels of the plane-fitting method. ...................................................... 22
4.5 Four corners of a square and four squares to a corner. ................................. 23
4.6 Corner Colors vs. Wang Colors. ............................................................... 25
4.7 A small example of 3-color corner tiling. Image from Lagae and Dutré, [1]. .... 26

5.1 Different styles of natural terrain sampling. ................................................... 28
5.2 Canyon and mountain data sampled. ......................................................... 31
5.3 A hill and slope specified by 3 lines .......................................................... 32
5.4 Terrain shapes generated with a few hand-painted lines. ............................... 33
5.5 A real-world canyon with a user path ....................................................... 35
5.6 Different styles of tile border interpolation. ............................................... 37

6.1 The Unity interface design for terrain editing. ............................................. 39
Abbreviations

**ACCAD** The Advanced Computing Center for the Arts and Design

**DEM** Digital Elevation Map

**GIS** Geographical Information System

**LOD** Level of Detail

**MDBU** Midpoint Displacement Bottom-Up

**OSU** The Ohio State University

**Virtual** A representation intended to approximate a real world or experience.

**Terrain** A landscape designed to represent the ground plane of a virtual world.

**Rasterize** To make a discrete representation of a continuous object.

**Height Map** A two-dimensional array of height values used to rasterize a terrain.
Chapter 1

Introduction

1.1 Problem Statement

Creating virtual terrains is an increasingly popular undertaking for computing applications of all kinds. Video games, medical rehabilitation, virtual tourism, land planning, and even military training motivate the creation of these terrains. This variety of applications leads to a wide range of design needs which has led to a diverse set of construction tools for these environments.

Video games require the ability to generate terrains which can be vast and which require contiguous accessible areas for the player, constraints on player access, as well as various game features. In a first-person-shooter game these features can include sniper locations, galleries, choke points, and arenas [2]. Controlling accessibility and using game design patterns are important in the generation of video game worlds.

Virtual environments are increasingly used in exercise-based video games and rehabilitation systems. As users perform exercises, they are able to receive real-time feedback and are entertained, leading to more enjoyment and engagement during the activity. In a rehabilitation scenario, we may have a patient who walks on a treadmill with the environment projected in front of them, mimicking the speed and orientation of the walker. This virtual environment is key to keeping the patient engaged in the rehab process. Another key is the ability of the rehabilitation operator to tune the exercise regimen to the needs of the patient. Needs vary from patient to patient and even from week to week. We cannot foresee all necessary path configurations ahead of time, so a customizable solution is required.
Virtual tourism and land planning are both based on real-world data which can be obtained from Digital Elevation Maps (DEM’s). Military training levels are similar in design to video game levels.

One way to customize paths and environments is full hand modeling. While this approach works in theory, in reality the cost of generating all the necessary environments by hand is prohibitive for all but the largest design companies. For example, a sample terrain which included a 2 kilometer path designed by hand took over 40 hours to model and texture. In this paper we present a system which can be used by a user with moderate knowledge of computer graphics and modeling to design a path and virtual environment of comparable scale and complexity in less than an hour.

We propose a system to design terrain constraints which can be used to expedite the production of these virtual environments. These constraints can be built from real-world data, user-defined paths and painted heights, as well as game-specific features. Once the constraints are in place, an environment is procedurally generated around them.

1.2 Overview of Remaining Chapters

The remainder of the paper is organized as follows. In Chapter 2 we look at work related to the idea of procedural terrain generation. In Chapter 3 we will look at the methods used for designing paths both for constraining the shape of the terrain and for controlling movement through it. The generation of terrain shape based on these paths is discussed in Chapter 4. In Chapter 5 we will look at the application of other types of constraints in the system, including the addition of height map data in Section 5.1, applying user-painted weights in Section 5.2, and tile border constraints in Section 5.4. In Chapter 6 we will present the implementation of the system. Chapter 7 will draw some conclusions from the resulting system, discussing successes, failures, and room for future work. Last is the bibliography.
Chapter 2

Review of Literature

Procedural terrain modelling has been an increasingly popular topic of research for the last thirty years. Our work brings together methods from various terrain and environment generation techniques. The most common techniques for synthesis include geometric modeling and physical simulation. With the increase in available resources, real-world data is often used as a source for terrain shape. Some work has been done in building terrains from different constraint systems. In the video game world, tiling is a common technique for designing levels. This chapter will discuss these methods and how they relate to our work.

2.1 Generative Methods

Geometric models are commonly based on fractal designs such as those used by Mandelbrot [3], Fournier [4], and Perlin [5]. Physical simulation techniques are largely based on erosion methods such as thermal erosion and fluvial erosion. Both are discussed by Musgrave et al [6]. These techniques can produce realistic looking terrains, but are often difficult to control.

2.2 Using Real-World Data

One method for recreating realistic terrains is proposed by Zhou et al [7] which applies real-world elevation data to user-defined control paths. Thus a mountain range or canyon can be conformed to follow any desired route. Several other works allow users to sketch
splines, rivers, mountains and other features and then generate terrains to match [8, 9]. These techniques are designed for the user to define terrain features such as ridges which are then directly built in to the terrain shape. Our constraint points may be ridges or valleys but are not limited to these descriptions.

A technique for constrained fractal terrains has been proposed by Belhadj [10]. This idea was created to reconstruct digital elevation maps by supersampling compressed elevation sets gathered by satellites or other means. They are also able to account for user sketches as constraints for terrain generation. They use a triangle-based Midpoint Displacement Bottom-Up process (MDBU) to fill in the gaps between known control points and those which require additional constraint. This technique does a fine job in filling in the gaps between user-defined control points, but has complicated parameter settings which would cause a steep learning curve for a novice user. In an earlier work [11], they allowed users to define mountain ridges and river valleys to give an initial shape to the terrain. This technique produces terrains by refining fractals between the constraints, but is not suitable for our needs as the terrain always slopes down from the ridges and up from the valleys. We want to use the shape of the path to inform the terrain builder of the slope of surrounding features without having the path designer specify whether they are on a ridge or in a valley. We want the user to design terrains with accessibility in mind.

2.3 Using Path Constraints

In our research we have found limited work which adapts the shape of a terrain to a pre-defined path. One author proposed the idea but in their actual work did not end up using paths [12]. Another work by Smelik et al [13] uses an integration of manual editing and procedural generation to create landscapes at low cost to the user but with more control than purely procedural methods. They also point out that procedural content is well-suited for large-scale operations while manual editing works well with small details, but it is difficult to bridge the gap with medium levels of detail and control. Their design is meant
to be edited continuously with live updating of features such as cities, rivers, and roads. Because of the interaction between these components, strict manually-designed features are not well-preserved upon procedural generation.

Another paper by Stachniak and Stuerzlinger [14] presents a method for conforming an existing terrain to the shape of a constraint which could be defined as a path or road. This method begins with a defined terrain and a path for constraint, and then seeks to take steps toward an optimal solution by deforming segments of the terrain under a gaussian kernel. While this technique can come close to an optimal solution, there is no guarantee of convergence. This technique involves manipulating an existing terrain, whereas we may be working from scratch. One more technique for applying local manipulation to an existing terrain is presented by Bruneton et al [15]. They begin with large-scale Geographical Information Systems (GIS) data and render using a level of detail (LOD) approach, focusing on local manipulation of the terrain from user-defined roads and rivers. Once again this system is designed for use with a pre-existing terrain.

Taking the opposite approach, there has been work done related to finding a suitable path using the terrain as the constraint [16]. Given a terrain with origin and destination points, Galin et al are able to generate a road connecting the two while considering obstacles including rivers and hills. Our work seeks to do the reverse: we are designing a path and then generating a terrain shape to conform to that path. In our case, the path is the key component because of its link to the therapy, while the environment is secondary.

### 2.4 Motivation for Virtual Environments

Immersing people in virtual environments using gameplay has been shown to motivate physical activity [17]. Significant work has been done in research on exergames, the design of video games used in conjunction with exercise equipment. Considerations for these games are discussed in [18]. Rewards for physical interaction has been shown to increase physical
activity while playing games without decreasing the enjoyment of the experience [19]. In a similar vein, Bianchi et al [20] demonstrated that body movement and physical activity not only increase the level of engagement for players by motivating a sense of presence in the digital world, but the experience became more rewarding for players as a result.

While it has been shown that games can serve to assist in promoting physical activity, they cannot provide all the motivation on their own. Chandra et al [21] showed that while patients enjoyed their individual exercise sessions more while using motivation through entertainment, they still wanted to see long-term progress numbers toward their overall rehabilitation.

Dimovska et al [22] demonstrate that adapting level difficulty based on previous performance can positively impact rehabilitation goals. Using a Wii Balance Board, they developed an adaptive skiing game which placed future gates based on the player’s ability as shown through prior gate passing. By adapting the difficulty to the patient ability, they are able to maintain better engagement of subjects because they are less prone to boredom in the case of easy levels, or frustration in the case of levels which are too difficult. A system which can be customized to the needs of a game can be valuable in motivating interaction with the environment and increasing replayability.

2.5 Tiling Methods

Tiling methods allow for the creation of large, complex environments composed of many smaller pieces. Once a tile set has been created, diverse environments may be created from different orderings and placements of individual tiles. These tiles can either be prefabricated or customized for each environment. Here are some papers which discuss tiling. Wang Tiles, first introduced by Wang in 1960 [23] are equal-sized squares with colors assigned to each side. The tiling is applied by placing a new tile only when the colors of the new tile match the existing colors on the adjacent edges of the existing tiles. Many updates to the original
Wang Tiles have been proposed over the years, notably a paper by Lagae and Dutré [1], proposing corner-colored tiles as opposed to colored edges. This proposal seeks to avoid the problem of tile corners causing discontinuities due to the lack of constraints on their diagonal neighbors. Grünbaum and Shephard, [24], include an algorithm for creating an aperiodic tiling of an infinite plane using a finite set of tiles. Stam, [25], proposes a system for applying this aperiodic tiling to planar texture synthesis using noise functions.
Chapter 3

Path Generation

Generating paths in the terrain helps to shape the terrain as well as ensuring accessible areas in the virtual environment. The path is defined by a set of control points which are user-placed. To interpolate between control points we use Hermite splines for a second-order continuity. Once the spline curve has been defined, we determine the shape of path geometry to be used as a constraint in Section 4 to calculate the shape of the terrain. This path is also available to guide user motion through the environment. In the case of medical rehabilitation we want to be able to control the level of difficulty for the patient through their exercise course, so constraining their movement to a specific path provides a solution.

3.1 Control Points

The main control the operator has in shaping the path is in the position of the control points. These points are shown as white spheres as in Figure 3.1. The operator has freedom to select a sphere and move it anywhere in the 3-dimensional world space of the game. Points may be added to the end of a path, inserted into the middle of the path, moved, and deleted. As points are manipulated, the path is updated for real-time feedback. One element which can aid in tuning the rigor of the workout is the display of path statistics such as the length of the path, the current grade, the maximum height, and the distance covered. These numbers can be seen when editing control points or when walking along the path as seen on the right side of Figure 3.1. Additionally the operator can choose an open path which can start and end at arbitrary locations, or a closed loop path.
3.2 Hermite Splines

With our control points in place, we need to interpolate between them to give shape to the path. Linear and quadratic interpolation do not provide the smoothness we require, so we use a cubic Hermite spline. We need the interpolation to be smooth because of the nature of the immersive environment. With a large, immersive display, jerky motion can cause disorientation of the user. If used for purposes other than walking in an immersive environment, other interpolations could be used.

Our Hermite spline interpolates the path shape between each consecutive pair of path control points. We will define a spline \( S \) as a piecewise polynomial real function, \( S : [a, b] \Rightarrow \mathbb{R}^3 \) with \( n \) sub-intervals. Sub-interval \( i \) is defined between control points \( p_i \) and \( p_{i+1} \) and its shape is defined by polynomial \( P_i : [0, 1] \Rightarrow \mathbb{R}^3 \), so that \( S(t) = P_i(t - i) \) where \( i = \lfloor t \rfloor \) and the parameter for \( P \) is restricted to the range \( [0, 1) \). We now define the parametrization of each polynomial, \( P_i \).

The Hermite interpolation requires two endpoints, \( p_i \) and \( p_{i+1} \) and two tangents. To obtain the starting tangent for \( P_i \), we use half the vector formed between the surrounding control
points: \( m_i = (p_{i+1} - p_{i-1})/2 \) and similar for \( p_{i+1} \) our tangent is \( m_{i+1} = (p_{i+2} - p_{i})/2 \). The automation of tangent vectors makes the system easier to use than requiring explicit tangents as in some drawing programs. Then for a given interval between control points, with interpolant \( t \epsilon [0,1] \), we find the position \( P_i(t) \) using the following equation:

\[
P_i(t) = a \ast p_i + b \ast m_i + c \ast p_{i+1} + d \ast m_{i+1}
\]  

where \( a = (2t^3 - 3t^2 + 1) \), \( b = (t^3 - 2t^2 + t) \), \( c = (-2t^3 + 3t^2) \), and \( d = (t^3 - t^2) \).

### 3.3 Character Position and Movement

The character of a game may be able to freely move but in the case of medical rehabilitation, the position of the viewer can be constrained to the defined path. In that case the position is determined by the parameters of the Hermite spline as in Equation 3.1, while the orientation of the character is based on an approximate tangent of that curve. This tangent is given as the normalized vector \( S(t + \epsilon) - S(t) \). Sampling the path slightly ahead of the current position (at \( t + \epsilon \)) allows us to give the user a look-ahead anticipation of upcoming curves. For this reason the path needs to be second-order continuous. The first order ensures that the character’s position does not jump from time to time, while the second order continuity means that the character’s view direction does not take an abrupt turn.

We need to ensure the user can traverse the path at a consistent speed, so we reparameterize the path as follows. We have established \( t \) as the parameter along the curve, and say \( \vec{s} = S(t) \) is the position in space. We would like the following to hold:

\[
\frac{d\vec{s}}{dt} = \delta
\]  

that is, the change in spatial position relative to the change in the path distance parameter should be a constant \( (\delta) \). We can sample the curve in parameter space to construct a mapping from \( t \) to \( \vec{s} \). Now consider a function \( \Phi : \mathbb{R} \Rightarrow \mathbb{R} \) such that \( \Phi(d) = t \). We use
the map to determine the translation for $\Phi$. The Hermite interpolation spline $S : \mathbb{R} \Rightarrow \mathbb{R}^3$ translates $S(t) = \vec{s}$. Combining those layers yields:

$$\vec{s} = S(t) = S(\Phi(d)) \quad (3.3)$$

which gives us the ability to increment $\vec{s}$ at a constant rate relative to $d$, satisfying Equation 3.2. This consistency allows the speed of the user on a treadmill to determine the rate of the user through the world.

### 3.4 Constraint Points from Paths

We are able to use path shapes to determine the shape of a terrain. These paths can be relatively flat for a nearly-planar terrain, or they can vary in height and space to create more
interesting features. Because the height of the path determines the height of the resulting terrain, the shape of the path can determine the overall shape of the terrain.

### 3.4.1 Path Casting

The geometry of the path is extruded along the path spline, using normals from the spline to determine the vertices of the geometry. Extruding geometry from the path spline allows us to apply the path heights to the height map for the terrain shape. The process of applying the continuous shape of the path geometry to the discrete structure of the terrain height map is known as rasterization.

Path casting requires that our path is located where we want it relative to the terrain. We then iterate over every point in the height map, casting a ray to check for intersections with any path geometry. The origin of the ray is determined by the position and scale of the terrain along with the resolution of the height map. The ray is then cast in the $-y$ direction from above the highest point on the path. If there is an intersection between the ray and the path, the height of that collision is applied to the terrain height map. Note that for the intersections to be enforced only on path geometry we must turn off collisions with the terrain itself. Otherwise path geometry below the current level of the terrain would be ignored. This would also lead to false path readings which would lock the positions of the entire terrain, avoiding the processing of terrain shape described in the next Chapter.

```plaintext
xStep = terrainSize_x / heightMapRes_x;
zStep = terrainSize_z / heightMapRes_z;
for (i : 0 to heightMapRes_x) {
    x = terrainPos_x + i * xStep;
    for (j : 0 to heightMapRes_z) {
        z = terrainPos_z + j * zStep;
        rayOrigin = (x, +inf, z);
        hit = CastRay(rayOrigin);
        if (hit) {
```
heightMap[i,j] = hit.pos.y;
}

3.4.2 Sample Path Shapes

At each level in the terrain fitting algorithm, we find the best fitting plane over a given area. This process begins with a plane over the entire area of the terrain. The second level solves each of four quadrants of the terrain. Then each of those quadrants is broken into four and solved again. At each level, the terrain is shaped based on the constraint path points left inside the square.

If we want to build a mountain with an accessible path, we can create a spiral shape which rises in the middle. Compare the two terrains shown in Figure 3.3 which have the same paths in the center forming the top of the mountain. The only difference in the paths used to create the two mountains is the path surrounding the peaks. Because of our terrain shape solver described in the next chapter, the terrain surrounding the mountains here is actually quite different.

The first mountain path 3.3c stops when it comes to the base of the mountain. When solving for the best fit planes in this map, the second level of the algorithm found planes which fit constraint points in each of four quadrants of the entire terrain. This means that one corner of each of these squares contained some of the mountain peak and some of the lower portions. Finding the plane which best fit those constraint points means finding a plane which slopes down, away from the mountain. Once that plane is established, the next iteration divides that plane into four more pieces. The outer quads now have no constraint points in them because they are all in the center of the terrain. So those portions of the terrain are set where they were, sloping away from the mountain peak.
Figure 3.3: Two mountains created from spiral paths which have different bases. If we simply have a spiral path which is highest in the center, we will form a mountain as in 3.3a to 3.3c. However, if there is a flat base path around the spiral, there will be a flat base terrain surrounding the mountain as in 3.3d to 3.3c.

In contrast, the path in Figure 3.3d to 3.3f continues and flattens outside of the base of the mountain. These additional constraints are taken into account when solving for the terrain shape. Now the second level of quads has a mountain peak and a flat outer area. This will cause a more level plane to be the best fit on the second iteration. On the third iteration we also see a difference as the path constraints extend out farther than in the first mountain, so the outer portions of the terrain continue to be flattened and are then removed from the influence of the central mountain.

Using a similar idea, we can create a canyon from a path if we place the ends above the middle as shown in Figure 3.4c. Then when the terrain is fit to the path, the lower middle
section forms the base of the canyon while the higher ends form overlooks. Because of the smoothness of the path around these areas, the top and bottom are both accessible, but the edges are too steep to climb.

On a larger scale, we built a terrain which took over 40 hours to model by hand. The design was a mountainside with a path running a loop up one side and down the other. The time invested in hand modeling included shaping the terrain, texturing various parts of it, and applying trees around the map. Once the detailed version of the terrain was created, it was possible to trace a trail around the terrain using our system in under an hour, an improvement of 40 times. This trail ended up using 250 control points to create a path 4 virtual kilometers in length. The resulting terrain is over 1.5km square with a resolution of $4096^2$, and takes around 30 seconds to iterate for its shape, paint the path texture, and populate with trees. The terrain shape is shown in Figure 3.5.
Figure 3.5: The terrain shape resulting from our large mountain path. The path length is nearly 4km, and the terrain is over 1.5km square with over 16 million points in the height map.
The main contribution of our work is the concept of moving from the shape of a user-defined constraints to fitting an entire terrain around those constraints. Our terrain is defined as a 2-dimensional height map. This map may be scaled in the X and Z directions to account for the potential range of scales on the user-defined path. We define the Y-axis as the vertical scale along which the height values vary. We begin by matching the terrain to the heights of the constraints. Next we start an iterative plane-fitting process to bring the terrain into alignment with the path.

4.1 Matching the Path

To begin the terrain defining process, we match the terrain to the user-defined height of the constraints. This sets up the height map for the processing described in the next sections.
The points we rasterize are then fixed so they do not move during the processing of the rest of the terrain.

For each point in the height map we check for the presence of path geometry at the same $XZ$ coordinate. If there is a collision with that geometry, we assign the corresponding height ($Y$) into the height map. See Section 3.4.1 for a description of the path to terrain casting. We also mark that position as a part of the path so that it is preserved as a terrain-path point when we perform the plane fitting in the next section.

At this point we also modify the texture of the terrain to indicate where the path is. When walking through the world, it is helpful to have a visible path to follow. To easily avoid $z$-fighting issues or other problems caused by imperfect matching between the path and terrain geometries, we simply apply the path texture to the terrain on the appropriate terrain-path positions.

### 4.2 Shaping The Terrain

Once we have rasterized the constraints, we can begin the process of matching the surrounding terrain to the shape they model. We need to use a method to fit the terrain to the path at multiple scales. Several prior techniques rely on local manipulation of existing terrain shapes [13, 14]. We do not want to constrain our path generation process to be driven by an existing terrain shape, but to be open for whatever the designer deems appropriate for the application. Our technique is to iteratively find a best-fit surface. By finding the best fit at various scales via a quadtree [26], we are able to provide slopes over the scale of the entire terrain as well as refine local details to conform to the height near each constraint. We describe the technique of matching a single quad in Section 4.2.1 and then how to combine different levels of the tree in Section 4.2.2.
4.2.1 Linear Least Squares

When we solve for a best-fit surface over a portion of the quad-tree, we need to find the coefficients to define the surface. In general we want to solve the overdetermined system defined by a set of \( m \) constraint points with \( n \) unknown coefficients.

\[
\sum_{j=1}^{n} X_{ij} \beta_j = y_i, \quad (i = 1, 2, \ldots, m).
\]  

(4.1)

Note that in the case of a planar equation, we will use 3 coefficients, giving us \( n = 3 \) and Equation 4.1 becomes:

\[
\beta_1 x_i + \beta_2 z_i + \beta_3 = y_i
\]

(4.2)

In general we may have more coefficients, \( \beta_1, \beta_2, \ldots, \beta_n \). In order to solve this system, we need \( m \geq n \). As we use a different position or change levels of the quadtree, \( m \) will change, reflecting a different set of constraint points in the target region. We can write Equation 4.1 in matrix form,

\[
X \beta = y
\]

(4.3)

where

\[
X = \begin{bmatrix}
X_{11} & X_{12} & \cdots & X_{1n} \\
X_{21} & X_{22} & \cdots & X_{2n} \\
\vdots & \vdots & \ddots & \vdots \\
X_{m1} & X_{m2} & \cdots & X_{mn}
\end{bmatrix}, \quad \beta = \begin{bmatrix}
\beta_1 \\
\beta_2 \\
\vdots \\
\beta_n
\end{bmatrix}, \quad y = \begin{bmatrix}
y_1 \\
y_2 \\
\vdots \\
y_m
\end{bmatrix}
\]

(4.4)

To find the best-fit surface, our objective is then to minimize the squared error term,

\[
E(\beta) = \sum_{i=1}^{m} |y_i - \sum_{j=1}^{n} X_{ij} \beta_j|^2 = \|y - X\beta\|^2
\]

(4.5)
We begin the terrain fitting process by finding the best fit plane over the entire terrain for the set of constraints (in this case the shape of the path), 4.2a. Subfigures 4.2b, 4.2c, and 4.2d show successive steps in our iterative planar matching algorithm. Each level introduces 4 solving planes for each plane from the level before. This terrain has a resolution of 513 pixels, so we will gain no new clarity by solving beyond 7 levels in our iterations.

Assuming the \( n \) columns of matrix \( \mathbf{x} \) are linearly independent, we can find a unique solution to the minimization problem, namely,

\[
\hat{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}
\]  

(4.6)

This technique can be used to solve for the best fitting surface of any polynomial or parabolic shape. We have implemented the system to solve for the best fit plane.

When we solve Equation 4.6 we have coefficients \( \beta_1, \beta_2, \) and \( \beta_3 \) to determine values of \( \mathbf{y} \) in Equation 4.1. This defines the slope of the best fit plane required for one square of the terrain as shown in Figure 4.2a. Clearly one plane cannot fit the entire path unless that path is entirely flat. To accommodate this, we iterate down to multiple levels of detail using a quadtree approach, refining the shape of the terrain at multiple resolutions. Figure 4.2
Figure 4.3: When the corners don’t match up, the terrain loses its realism.

demonstrates the refinement of a height map being fitted to a path. At each level, planes are fit to the constraint points they contain. Part 4.2a fits one plane over the full path. Each level of the tree we divide the fitting area of the previous level into quadrants and solve again. The second level solves over 4 planes. Figure 4.2b shows the third level and has fit 16 planes. Figure 4.2c represents the 5th level and has 256 planes, 4.2d with 1024. At each level, the area covered by each plane is reduced, creating a better fit height map.

4.2.2 Corner Heights

We want to refine the fitting of the terrain, starting at the largest scale and moving to the smallest. Blending from one level of quadtree planar matching to the next is tricky. If we do not match the edges of the plane squares, we obtain some clearly incorrect results as in Figure 4.3. We therefore need to maintain some information of the points used from one level to the next.
The first level of plane fitting is done over the entire terrain. This establishes the basic slope of the whole area. If the constraints have an overall inclination, we want to be able to maintain the feel that the terrain is on a hillside, rather than having it flatten on the edges beyond the scope of the user input. This step is shown for a sample path in Figure 4.2B. Next we divide the terrain into quadrants and find the best fit planes for each of these based on the terrain-path points contained in each as described in Section 4.2.1.

Beyond the first iteration of plane matching, there will be a few issues to consider. One is that adjacent planes share boundaries. If we do not match those boundaries, we obtain a result as in Figure 4.3. On each level of iteration, we only compute the heights on the corners of each square as indicated in red in Figure 4.4. In order to determine the required height of the corners, we use the following terminology. We will say that a given square, \( S \) of the quadtree has corners:

\[
S_{NW}, S_{NE}, S_{SE}, and S_{SW}
\]
And that a given corner may be associated with up to four adjacent squares:

\[ C_{NW}, C_{NE}, C_{SE}, \text{and} C_{SW} \] (4.8)

A corner may have fewer than four adjacent squares if it is on the boundary of the height map, but any internal corners will have four adjacent squares.

\[ Adj(C) = C_{NWSE}, C_{NESW}, C_{SENW}, C_{SWNE} \] (4.9)

The height of a given corner will be determined by the average heights of the four interior corners of the adjacent squares. That is:

\[ h_c = \frac{1}{|Adj(c)|} \sum_{c_i \in Adj(c)} c_i \] (4.10)

where \(|Adj(c)|\) represents the number of squares adjacent to corner \(c\). Note that this number may be less than 4 if the corner is on the edge of the terrain square.

Because of the contributions from multiple squares to a single corner, each square is not guaranteed to be planar. If the corner heights of adjacent squares differ greatly, the discrepancy will be resolved on further iterations. This in particular arises when the constraints are far from planar, such as the switchbacks found on the side of a steep slope. We may find multiple levels of a switchback path to be in the same square, meaning that a single
plane cannot be a good representative for that section. This issue is resolved on further refinement as seen in Figure 4.2b as compared to 4.2d.

We only apply the plane-fitting calculations on squares which contain enough constraint points to satisfy the requirement from Section 4.2.1 of an overdetermined system with \( n \) surface coefficients, that is \( m \geq n \). This creates a natural stopping point for our iteration. Once we have too few control points per square we will have an underdetermined system. For example, in a system solving for planes \((n = 3)\) in a 1024x1024 height map, we would be able to iterate 10 levels before we are down to 2x2 squares, giving a maximum number of control points at \( m = 4 \). Further iterations would not allow for solving of a best-fit plane.

As in Figure 4.4C the corner squares (grayed out) of the terrain do not contain any part of the path, so they are not used to solve for a best-fit plane and no longer contribute to surrounding control corners. This does not mean that the square is held static, however. If that square is still adjacent to any other active squares, its control corners may still be affected, leading to a change in its shape.

### 4.3 Tiling

Assembling a set of tiles is not difficult, but we must take care in order to assemble the set in such a way as to avoid discontinuities and periodic repetition. We discussed creating a set of tile border constraints in Section 5.4, and here we will demonstrate how to ensure that our tiles are placed correctly.

While it is straightforward to treat tile continuity across adjacent edges, the enforcement of corner tiles maintaining the same height is not necessarily enforced. We employ a system which uses corner heights as constraints to help define our edges, so the tiles can be thought of as corner-colored tiles. We can also consider them as Wang tiles with extra colors based on the number of corner-color combinations.
In Figure 4.6 we can see an example of translating a corner-height system into a regular wang-tile layout using corner-pairs. In this case, set of corner-colored tiles using 2 colors is analogous to a Wang-tile set using 4 colors. In general this translation can occur from a set using $n$ corner colors to a Wang tile set using $n^2$ edge colors. Because the original corner colors preserve height continuity across the diagonals of tiles, the corresponding edge colors will do the same. Note that this does not give us the freedom to place every combination of colored edges into a tile. There are rules to maintain the system established by the corner colors. In Figure 4.6b notice that there are no examples of a red edge adjacent to a blue edge. This is because the corner color associated with red is red and the corner color associated with blue is yellow. Since a corner cannot have both red and yellow applied, red and blue edges cannot share a common corner.

Assembling a full set of tiles is easiest to conceptualize using corner colors. Assigning the colors to the tiles takes place during the constraint placement of border height from Section 5.4. As seen in Figure 4.6a, 2 corner colors leads to a set of 16 tiles. In general, $n$ colors result in a full tile set of size $4^n$. With this full set of tiles, a random placement of corner colors may be made in the terrain assembly space, with individual tiles filled in naturally based on the four corner colors adjacent to each space.

In Figure 4.7 we see a small example of what a corner tiling looks like when assembled. Each corner on the grid is assigned one of the available corner colors, and the appropriate tile is then inserted into place. In this example from [1], the tile numbers are determined based on the corner colors. These indices can be thought of as a base-$C$ number where $C$ is the
number of available colors. Then the index is a 4-digit number, \( c_{NE}c_{SE}c_{SW}c_{NW} \) determined by each corner color. The decimal version of this index is found by the combination of color indices:

\[
((c_{NE} \cdot C + c_{SE}) \cdot C + c_{SW}) \cdot C + c_{NW}
\]

For a moderately-sized tiling, the appropriate tile can be found without much trouble.
Chapter 5

Other Constraints

In addition to designing terrains based on path shapes, we are able to provide constraints from an array of sources, including real-world height maps, user-painted heights, tile border constraints, and various combinations of those points.

5.1 Using Real-World Data

Real-world data can be taken from available Digital Elevation Maps (DEMs). This data is not as manipulable as drawing one’s own height map by hand, but if used properly can add a great amount of detail and realism to a scene. Popular DEM features include hills, mountains, and canyons, but sometimes they can be used to apply smaller-scale textures.

Points from these real-world examples can be applied to the constraint height map in several ways. In any case, the process of transferring data from the source DEM to the terrain’s height map follows the same procedure. Much like casting the path shape to the terrain geometry, we will be applying new constraints to the target height map. And as in the case of the path shape, these points will be static in the terrain shape processing. In this case, the translation is done based on the resolutions of the DEM data and the height map.

\[
\text{heightMap}[i,j] = \text{DEM}[m,n] \\
i = m \times scl_x + off_x \\
j = n \times scl_z + off_z
\] (5.1)
The scale and offset values between the two arrays can be manipulated to move and scale features of the DEM around the target terrain. The default translation is $scl = 1$ and $off = 0$, but they can be rearranged depending on the needs of a particular situation. If the mountain needs to be in the top right instead of the bottom left, the offset could be set to $0.5 \times heightMapRes$.

Figure 5.1 shows six different methods of sampling from a mountain height map.

**Use the full height map** 5.1a The original version of the height map may be what is desired. In this case it is possible to simply copy the full original DEM into the target
height map. Upon application this will result in the terrain appearing exactly as the DEM would in reality. If some flexibility is desired, it is also possible to use a large brush size to copy chunks of the DEM, but leave room for other areas to be filled in. Brush size is by default set to 3 pixels, but can be made as large as the full resolution of the height map.

**Grid Sampling**  5.1b Use a grid if a regular sampling of the DEM data is desired. Our grid sampling method allows the user to set \( n \) as the number of sample points they would like and then copies points from DEM to the height map in a \( \sqrt{n} \) by \( \sqrt{n} \) grid of evenly-spaced points.

**Random Sampling**  5.1c A similar method to grid sampling is simply taking random points sampled from the DEM. Again the user is able to specify how many points to sample. In both cases a greater number of sample points leads to a more faithful recreation of the source data, whereas fewer sample points allows for easier incorporation of other constraint data. For a demonstration of how the grid and random sampling methods affect the calculated terrain shape, see Figure 5.2.

**Hand Sampling**  5.1d Sampling data from all over the elevation model may not be what the user has in mind. In that case the user can choose how much and which parts of the DEM are copied to the terrain height map constraints. This technique is useful if the user wants to highlight some key feature of the source map such as a ridge or mountain profile. In this way the overall shape can be captured, but limiting the space of terrain constraints allows for new user-defined shape to take their place. The ability of the user to choose which parts of the map get translated could also be used to erase features from a map. By copying the surrounding terrain shape, our algorithm will fill in the gaps, but will not necessarily match the original, now missing, data. This would be especially obvious in the case of a standalone feature such as a solo mountain. If the user only copied the flat terrain surrounding the mountain, but excluded the higher data, there would be nothing to inform
the fitting algorithm to increase the height of the generated terrain, and the mountain would effectively have been erased.

**All Above** 5.1e In another intentionally selective process, the user can sample only the height values above or below a certain threshold. Using the mountain example again, if the user selected only the elevation data below a level close to the base of the mountain, the rest of the mountain shape would be excluded from translation. This would likely result in the solver finding the best-fit surface to be mostly flat in the space where the peak used to be. On the other end, it is possible to sample only the data above a certain height as is done in the Figure.

**Isocontours** 5.1f The last method we show for sampling heights from a digital elevation model is taking isocontours. These are lines which all have the same height value. By extracting these from the terrain, we can see how the solver reacts to having jumps between different areas of constrained height.

It is clear from Figure 5.2 that more sampled points leads to a more detailed terrain. There is at least one other interesting observation to make about the results from different sampling rates. That is the concept of the Nyquist frequency. The canyon terrain includes many more high-frequency features and therefore requires a much higher sampling rate for a faithful reconstruction of the original as compared to the mountain terrain which has a few lower-frequency features. The mountain is much more recognizable at 100 sample points than the canyon, which looks like a jumbled mess with that relatively low sample rate.

### 5.2 Painting Points

Users are able to include their own custom-drawn heights to influence the terrain shape. Drawing can be done using any image-processing program and then imported into our
Figure 5.2: Subfigures 5.2a to 5.2f show real-world canyon data, while Subfigures 5.2g to 5.2l show terrains built from constraints sampled from a mountain height map. The constraint points are sampled with various frequencies and techniques. Grid samples are regularly spaced throughout the map, while random samples are taken from anywhere. The more sample points are used, the more detail appears in the final terrain.

...system. Images used in the program should be grayscale images which represent a height map. The highest points possible will be represented as white while the lowest possible points will be black. The advantage to drawing a limited set of points is the lower investment of time to generate a full terrain shape as compared to explicitly applying height values to the entire terrain area. Higher points can be used to design ridges or mountain peaks, while lower points will form valleys, canyons, or bodies of water.
Figure 5.3: A hill 5.3b specified by 2 colors and 3 lines 5.3a, and a slope across the terrain 5.3d specified by 3 colors and 3 lines 5.3c.

The drawing of constraints by the user works in a similar fashion to those points constrained by a path and those specified from real-world data. In Chapter 4 we saw how the constrained points contribute to the overall shape of the terrain. In a basic manner the user could specify a mountain shape with only a few strokes of the pen: a few points or a line to specify the shape of the mountain peak, along with an arc or other lines to define the base of the mountain. Once those few lines have been placed, it is possible to leave the remaining shape to the algorithm as far as interpolating the height and terrain shape in between.

We can take advantage of the fact that the terrain solving algorithm will interpolate between and beyond constraint points. This enables us to build shapes like hills using only a few constraints. This also allows us to design slopes across entire terrains using just a few constraints. Examples of these uses can be seen in Figure 5.3.

In addition to simple terrain features, it is possible to use the drawing space to add more
Figure 5.4: The terrains in 5.4b, 5.4d, and 5.4f were generated from several hand-drawn lines, 5.4a, 5.4c, 5.4e. These lines only use two or three colors (heights), but the resulting terrains have many different features. These non-flat features arise from the solving method at each level of the quadtree.
specific game elements such as terrain-based choke points, sniper locations, and arenas. These are described in [2] as examples of design patterns in first-person shooter video games. Choke points are a strategic game element which force players to pass through a certain location, creating an efficient position to guard. Sniper locations are often elevated and provide a position where opponents can be seen, but where the player is well-guarded. An arena is a large open space which can hold numerous friends and foes leading to melee battles. These features can be implemented based simply on the shape and height of areas within the terrain. Figure 5.4 shows the creation of arena-shaped terrain, a sniper location, and a choke point designed using fewer than 20 constraint lines and only 2, 3, and 2 constraint colors, respectively. Because of the interpolation of solving surfaces, cliffs are constructed between the top and bottom heights. The influence of these surfaces beyond the strict scope of their position leads to irregular features both on the upper level and in the lower areas of each terrain shape.

5.3 Combinations

In order to obtain more interesting results, it is also possible to combine some of the techniques in a hybrid terrain description. This allows the user to include real-world data with custom paths or other constraints. These techniques can be useful when concerned about the accessibility of real-world terrains. It is possible to use hand tools to modify the shapes of real-world data, but our technique allows for the combination of custom lines and paths to be blended into the real terrain heights.

In Figure 5.5 we show an example of a canyon which would not be traversible by a game character due to the excessive slopes they would encounter. Rather than going through and manipulating the terrain shape by hand, the user can build a path using our system. We then sample points from the original canyon shaped terrain. When solved with just canyon points, we get the result shown in 5.5a. We can then apply the height of the path we have designed to help the character cross the canyon. This path applied directly on top of the
Figure 5.5: A real-world canyon is the source for the height map data. Part 5.5a is just the canyon data sampled and applied. Part 5.5b applied a user path directly to the terrain with no blending. Part 5.5c applies our interpolation to the new path constraints to help blend this accessible portion with the rest of the terrain.
canyon shape is shown in 5.5b. To make the path look more natural, we apply our terrain solving algorithm again. This time the constraints include the sampled canyon data as well as the heights of the path. When the path heights are blended with the original canyon terrain, we see a believable resulting terrain in 5.5c.

Because of the design of the iterative terrain solver, it is possible for the user to apply a combination of all kinds of constraints at once. There is a limitation that only one constraint applies per height map pixel because the terrain can only have one height in a given XZ location. Overlapping constraints will have to be applied based on which was specified most recently. Apart from that, there are no practical limitations on the constraint applicability.

### 5.4 Tile Borders

Building a set of tiles requires the ability to combine individual tiles into a set over a larger area. The most common description of tile edge compatibility derives from using colors to describe each edge, [27]. A limited set of colors is used to build a set of tiles with all combinations of color. A full set of tiles using \( n \) colors would result in \( 4^n \) tiles. Tile edge color represents the compatibility of the tiles with one another, that is matching colors should naturally align. In our case, the tile edge color corresponds to the height profile on the border of the tile. The algorithm for applying tiles is discussed further in Section 4.3.

In general the borders of individual tiles must match naturally or have a seam-stitching algorithm applied. We have set up a system of tile borders in which the user can choose corner heights and intermediate values as well. A few deterministic options are available which give a regular appearance to an array of tiles. In our system, we align the edges of each tile segment based on the height of the corners. As the corner heights are determined, the remaining height of the edges are determined by a choice of interpolation styles. Deterministic, regular edge patterns are apparent when assembling a set of tiles. A few examples of edges are shown in Figure 5.6. Deterministic edges such as those in 5.6a, 5.6b, or 5.6c,
naturally align during the tiling process because their shapes are consistent from tile to tile. That is, given the same corner heights, a linear interpolation will yield the same border every time. More interesting borders such as in 5.6d can provide a more natural landscape, but care must be taken to ensure that borders of the same color also share the same profile.

In our setting of constraints on tile edges, we limit the thickness of the constrained points around the edge so that enough freedom is allowed in shaping the remainder of the terrain tile. We want this constraint in order to maintain first-order continuity between tiles. That is, the edges are aligned and their slopes are equivalent on the edge. We find that a ribbon with a width of 2 is enough to maintain this level of continuity. For reasons of continuity, tile border constraints have precedence over other features of the terrain.
Chapter 6

Implementation

6.1 Unity 3D

We implemented the system using the Unity3D software package (http://www.unity3d.com). All scripting was done using C#. The basic Unity engine supports interactive 3D scenes. The scene we have constructed is built around the editing of terrains. We provide for the creation and shaping of paths, importing and applying real-world textures, drawing user-painted heights, applying border heights, and tiling created terrains.

The editing system consists of elements related to designing paths, including real-world data, painting user-desired heights, and adding border shapes. Editing the path consists of creating, deleting, and moving control points around the terrain space. The path interpolates between these points to fill in the proper path shape which can then be used to define the shape of the terrain. An editor for the user to select real-world data and to draw their own heights is available as is a method of creating tile borders.

Once the terrain is built, we are able to take advantage of the rendering and storing capabilities that Unity provides. Terrains are stored with information about their height maps, applied textures, and other objects such as trees and grass. Our scenes are able to run in real-time with interactive framerates over 90fps on a single core Intel i7 processor using under 750MB of memory for a terrain with a heightmap resolution of 16 million pixels and over 100,000 trees.
The user interface designed in Unity can be seen in Figure 6.1 where we have all the controls available. Menu selections from the "GUI Controls" box bring up various other control windows which allow the user to manipulate path points, build or reset the terrain, import real-world height maps and paint their own heights, add trees and signs, and walk along the path or walk freely through the environment. Additionally the user has the option to save the terrain they have created or to load one they saved earlier.
Chapter 7

Conclusion

7.1 Conclusion

We have presented a system for generating environments based on user-defined constraints. These constraints can be taken from user-shaped paths, real-world data, user-painted heights, and tile borders. The path generation technique allows for novice users to define a path shape which could fit the needs of a physical therapy patient or ensure accessibility for a video game character. The surrounding terrain is automatically generated to conform to the shape of the given constraints. Trees and textures are applied to the terrain to create a more immersive experience for the user.

The use of a few constraints as sufficient input provides an efficient pipeline for procedural terrain generation. This allows a level designer the freedom to explore with as much or as little input as they would like. It is possible to construct a terrain using just a few strokes of virtual paint in the constraint height map. It is also possible to spend hours editing a heightmap to have just the right features. In either case, the procedural terrain generation algorithm presented here is able to create a complete terrain which satisfies the constraints and shapes the remainder of the space automatically.

The ability to bring in constraints from different sources allows for great control over the appearance of features large or small, realistic or artificial, smooth or undulating.
7.2 Future Work

There are a number of additional ideas and features which would make for an even better assisted procedural terrain generation system. Real-time terrain updates would allow a much more efficient feedback loop for the user. Real-world data samples could be taken relative to the frequency of features for a more efficient sampling. Something could be done to combat the square line artifacts left in the generated terrains whether that would be adding noise to the result or processing un-constrained areas further. We could look into binary space partitioning as a method of dividing and conquering the iterative solving problem. Another possibility to adding more realism to the generated terrains would be to add small-scale features from actual terrains, so if the user built a mountain, the algorithm would apply the small features of a mountain to the terrain.
Bibliography


