2009-11-18

Feature-based Interactive Terrain Sketching

Daniel B. Adams
Brigham Young University - Provo

Feature-based Interactive Terrain Sketching

Daniel Adams

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computer Science

Brigham Young University

December 2009

ABSTRACT

Feature-based Interactive Terrain Sketching

Daniel Adams

Department of Computer Science

Master of Science

Procedural generation techniques are able to quickly and cheaply produce large areas of terrain. However, these techniques produce results that are not easily directable and often require artists to edit the results by hand to achieve the desired layout. This paper proposes a sketch-based system for controlling fractal terrain that allows for a wide variety of terrain feature types. Artists sketch features rather than constrained points or elevations. The system is interactive, provides quick on-demand previews of the terrain, and allows for iterative design modifications. Interaction between features is handled in a realistic fashion. An arbitrary vertex insertion order midpoint displacement algorithm is also described which provides the necessary flexibility and constraints for the terrain generation system.

Contents

Chapter 1

Introduction

Outdoor terrain is often used as a backdrop in film and television and is a key part of many interactive applications. Creating the 3D terrain models is a tedious and time-consuming process. As technology has improved, the realism expected by movie and game audiences has also increased, resulting in higher costs to studios.

Several algorithms currently exist to automatically generate terrain, saving time and money. These systems typically take a few parameters, sometimes sampling existing data, and produce a completed terrain map. However, none of the approaches currently available give enough control and feedback to the artist creating the terrain. The layout of the terrain is often of great importance to directors, especially for game designers. While some systems allow control over a small set of terrain features during the creation of the terrain, artists are typically still required to spend large amounts of time editing the generated terrain to fit a desired layout.

We present a system to generate terrain from user sketches of the layout. Unlike previous methods, this system provides a quick, interactive interface and allows a variety of terrain features. Users are able to focus on placing complete features rather than constraining individual points.

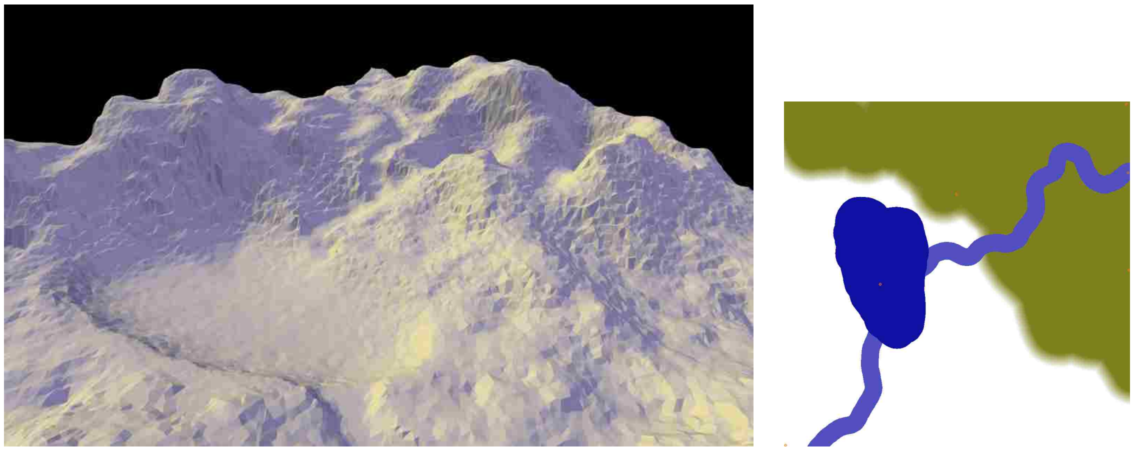


Figure 1.1: Preview of a lake and river (left) and the corresponding sketch (right).

The main interface for this system is a paint program with different brushes representing different feature types. For example, users are able to draw a brush stroke to represent the path of a river, or paint a region to represent a level section of terrain. Each brush has its own set of user-adjustable parameters. These parameters control properties such as the width of a road or the height of a mountain ridge. In addition to feature brushes, a few parameter brushes give the user control over the general parameters of the fractal terrain, including height of hills and the amount of variation in terrain elevation. The user is able to generate a preview of the terrain at any time. Figure 1.1 shows a user's sketch and the a preview of the resulting terrain.

It is important that interactions between various feature strokes are handled in a realistic fashion. We introduce two main mechanisms for handling the feature interactions. First, we set up a rigid priority system for terrain feature placement order. Second, before generating the points, we build a list of all the potential vertices as a place to store information for later use.

Additionally, we introduce a new arbitrary vertex insertion order midpoint displacement algorithm that gives us the flexibility needed to combine our feature generation with the fractal terrain generation.

Chapter 2

Related Works

2.1 Midpoint Displacement

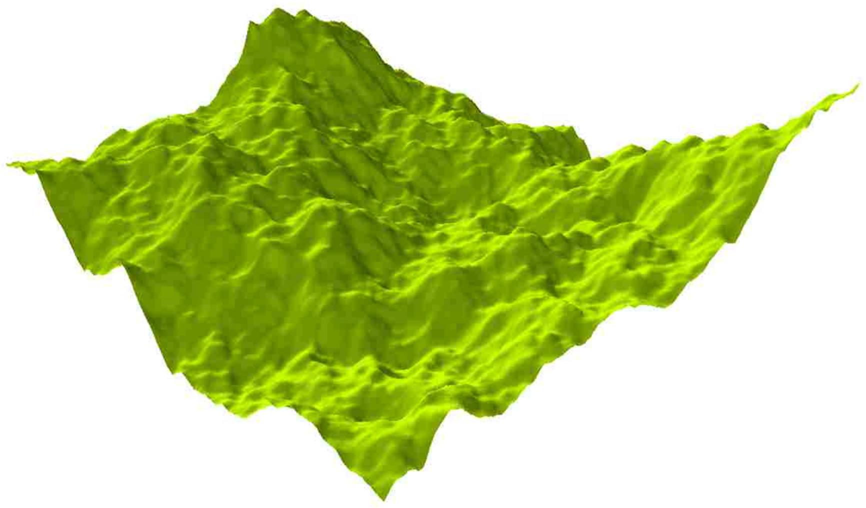


Figure 2.1: Terrain generated by the square-diamond midpoint displacement algorithm.

In [FFC82], the midpoint displacement algorithm is introduced as a method of generating fractal terrain. The midpoint displacement algorithm consists of recursively subdividing regions of the terrain, adding noise to the center point each time. It is used frequently because of its simplicity in generating a shape that roughly resembles hills and valleys (see figure 2.1). [Mil86] discusses artifacts that can be produced by the previously used subdivision methods, referred to as triangle-edge and square-diamond, and introduces a new subdivision method called square-square subdivision that reduces these artifacts. One of the major problems with the methods described in these papers is that they provide the user with no control over the layout of the final terrain. [MKM89] presents a noise synthesis method for generating terrain similar to that generated by the midpoint displacement algorithm. Noise functions of varying frequencies are successively combined until the terrain has the desired level of detail. Like midpoint displacement terrain, feature placement is not controllable.

2.2 Constrained Fractal Terrain

Recognizing the need to control the output of the fractal terrain algorithms, several papers have discussed ways to introduce user specified constraints. [ST89] presents a method that approximates the desired terrain with a constrained spline mesh with fractal terrain overlaid. While general shapes can be controlled using this technique, fine details are still not constrainable. The method given in [VM96] allows for exact constraints in the terrain, but requires a high number of constraining points to precisely control the results. An inverse to the midpoint displacement algorithm is presented in [BA05] and refined in [Bel07] which solves most of the issues found in previous approaches to constrained terrain. It finds the points in the grid that would have been settled before the constrained points in the traditional midpoint displacement algorithm and calculates appropriate values for these "parent" points. Once the parent points have been settled, the remaining points can be filled in normally.

Each of these approaches to generating constrained terrain shares the common requirement that the constraints must be completely specified before the terrain is generated. Additionally, each of these techniques produces terrain that is confined to a grid. Our arbitrary vertex insertion order technique allows for more flexibility by allowing new constraints or even modifications at any stage in the process and is not limited to grid vertex arrangements.

2.3 Other Approaches to Advanced Procedural Terrain Features

Stachniak and Stuerzlinger take the opposite approach to the systems previously described [SS05]. They first generate the fractal terrain, then deform the necessary portions of it to fit constraints, while maintaining the terrain's general appearance. Though their method produces good results, it is too slow for an interactive system.

Prusinkiewicz and Hammel present a technique for producing rivers in a midpoint displacement terrain system [PH93]. They add a simple context sensitive grammar that allows each region in the terrain map to affect its neighbors. This passes information about the river through the regions as they are subdivided. The rivers produced by this method are perfectly flat and artifacting occurs in the hills near rivers.

[ZSTR07] describes a technique to produce terrain with mountain ridges or river valleys that follow sketched curves. This system splices together patches from height maps of existing terrain, resulting in terrain that is much more realistic than fractal terrain. However, the algorithm requires existing sample terrain, handles only ridges or valleys, and is able to constrain the terrain only at a large scale.

Chapter 3

The Feature-based Interactive Terrain Sketching System

In order to overcome the weaknesses of current fractal terrain generation systems, we propose the Feature-based Interactive Terrain Sketching system. This system provides two major contributions. The first is the arbitrary vertex order midpoint displacement algorithm that provides the necessary flexibility for generating fractal terrain with user specified features. The second is a sketch interface that allows the user to iteratively sketch terrain features and view rapidly generated terrain models.

3.1 Arbitrary Vertex Order Midpoint Displacement

One of the major hurdles in placing our terrain features into fractal terrain is the fixed order in which points are added in the traditional midpoint displacement algorithm—

each point must be placed after the vertices that form its surrounding region. We need a method for generating the fractal terrain that is able to handle modifications and constraints during the construction of the heightmap.

We propose a new midpoint displacement technique that overcomes the rigidity of the standard techniques. The arbitrary vertex order midpoint displacement algorithm allows for the addition of constraints at any time in the generation of the terrain along with the ability to modify previously settled points.

The general idea of the midpoint displacement algorithm is to repeatedly subdivide a region, displacing the current height of the center point of each portion up or down by some random amount. The amount the point is allowed to move is a function of the size of the region. As the regions become smaller, the amount the new center points move is reduced. In our algorithm, we follow the general midpoint displacement procedure, but in order to allow for arbitrary vertex order, we introduce new methods of determining the current height of a point and calculating the variability of the new point.

3.1.1 Determining the Current Height of a Point

Instead of relying on a fixed grid as in the traditional algorithms, the modified algorithm maintains a triangulation of all the points already added to the map. To find the current height of the new point, the system simply looks at the height of the triangulation at the location of the new point. The Delaunay triangulation is particularly well suited to this because it avoids long skinny triangles. For speed considerations we use a 2D Delaunay triangulation instead of a 3D version. The height is ignored in computing the triangulation. Because points are added one at a time, we use the incremental Delaunay algorithm presented in [GS85]. A good implementation of this algorithm can be found in [Lis94]. Note that while the traditional algorithm forces a grid style heightmap, this modification allows us to use a non-grid vertex arrangement.

3.1.2 Calculating the Variability of a Point

In the traditional algorithm, the variability at a point is related to the distance from that point to the previously settled points—less variation is allowed as the size of the region shrinks. The modified midpoint displacement algorithm calculates the new points' variability as a function of the distance to the nearest settled point using equation 3.1.

$$V = \begin{cases} \dfrac{D^P \cdot V_{max}}{{D_{max}}^P} & \text{if } D < D_{max} \\ V_{max} & \text{if } D \geq D_{max} \end{cases} \qquad \begin{aligned} &\text{where} \\ &V = \text{variability of a new point} \\ &D = \text{distance to the nearest point} \\ &V_{max} = \text{maximum variability} \\ &D_{max} = \text{maximum distance} \\ &P = \text{curve power} \end{aligned} \tag{3.1}$$

The maximum variability, maximum distance, and curve power are the fractal parameters that define the shape of the terrain. The maximum variability represents the variability of a point in an empty triangulation and corresponds to the variability of the largest region in the traditional algorithm. The maximum distance parameter is the distance at which a point is considered to be added to a "new" triangulation. This roughly corresponds to the distance between corners of the grid in the traditional algorithm, and affects the width of the generated hills. The curve power adjusts how fast the variability diminishes when the new point is near an existing point. This affects the general smoothness of the terrain. Figure 3.1 shows a graph of the variability according to this function. Just like in the traditional algorithm, the new point is displaced up or down from its current height according to a random percentage of the variability. The parameters are not required to be constant across the terrain; each vertex may have its own value for each of these parameters.

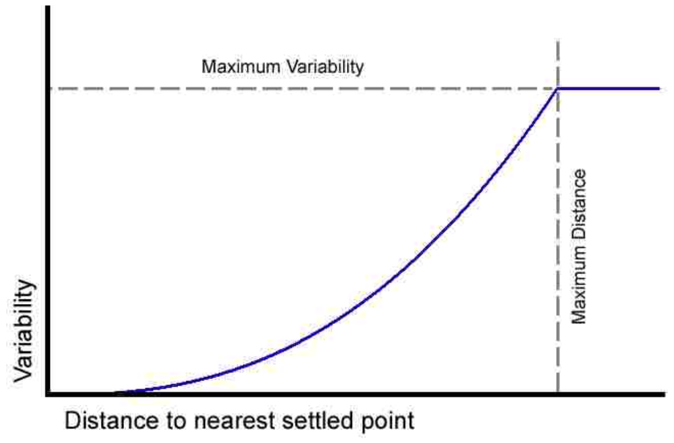


Figure 3.1: Variability as a function of the distance to the nearest settled point in the terrain mesh. The shape of this curve is defined by the maximum variability, maximum distance, and curve power parameters.

The properties of a Delaunay triangulation make finding the nearest point trivial. These properties guarantee that after the point has been added to the triangulation, the nearest neighboring point will be connected to the point in question by a triangle edge. Because the 2D Delaunay triangulation is based only on horizontal relationships, we can insert the point into the triangulation, find the distance to the nearest connected point, and later set the final height for the vertex.

It is important to note that this modified midpoint displacement algorithm does not allow for every possible insertion order. For example, if points are added in scanline order (such as left to right, top to bottom), the variability of every point would always be at its minimum, resulting in nearly unchanging terrain. In general, points added too close together may result in smoother than intended terrain while points added in a reasonably random order will maintain a correct appearance. It is important that this is taken into consideration as terrain features are added to the terrain by the system.

3.1.3 Post Process Filters

To gain additional control over the appearance of the terrain and to reduce the artifacts inherent to midpoint displacement terrain (see [Mil86]), we perform Gaussian blurs and median filters on the height values in the terrain. The Gaussian blur provides a general smoothing effect to the terrain. The median filter reduces sharp noise and can give a weathered look to the terrain. Using these filters to create smooth terrain gives better results than adjusting the curve power parameter alone. We allow different kernel sizes for the filters at each point in the terrain.

3.1.4 Summary

We use a Delaunay triangulation to maintain the current state of the terrain as a triangulated mesh at all times. This allows us to easily find the current height of the terrain at any given point. The point is moved a random amount according to a function of the distance to the nearest point in the mesh, with nearby points limiting the amount a point can move more than distant points. We further control the appearance of the fractal terrain by applying blur and median filters.

3.2 Feature Creation

The second major contribution of this work is the ability to add a variety of features to the terrain in an interactive fashion. Included in the feature set are lakes, rivers, terraces, roads, ridges, and fixed points. Any feature can be placed at any location in the environment. The system ensures that overlapping or intersecting features are handled realistically. The following sections describe how the features are generated in the system.

Terrain features are created by inserting specific vertices into the midpoint displacement mesh at a specified height instead of a random height. The vertices added are those that define the extreme boundaries of the features' shapes. For example, when creating a lake, vertices of a constant height are added to define the shoreline. Other vertices are placed at a lower elevation in the interior of the lake's region to define the lake bed. These vertices may be inserted at any time, but are usually added early in the midpoint displacement process, since that is when they have the most effect on the shape of the terrain. Points that fall within the feature's region but are not explicitly added as constraints will be filled in later by continuing the modified midpoint displacement algorithm as usual. In addition to adding new vertices, the heights of existing vertices may be modified as more features are created.

3.2.1 Potential Points

Before beginning the terrain synthesis process, we create a list of all of the points we may want to add to the terrain. We call these points "potential points". We use these potential points to store any information that has been calculated for a vertex before it is actually added to the terrain mesh. The data stored in the potential points includes midpoint displacement parameters and information about nearby terrain features. Potential points that have been added to the terrain mesh are marked as settled and their elevation is stored. While it is not strictly required, we fix all of our potential points to a grid arrangement, as this makes it trivial to find potential points in a given region.

Because we use the potential points to store information about features as they are created, every point added to the final mesh must come from our list of potential points. However, not every potential point must be added to the terrain. For example, we can produce a faster preview of the terrain by using only a random subset of the potential points.

3.2.2 Feature Interaction

It is important to the believability of the generated terrain that the terrain features interact well with other nearby features. As examples, intersecting roads must meet at the same elevation and it is more plausible for a river to carve a pass through a mountain ridge than it is for the river to flow up and over the top of the ridge. Our system uses two main mechanisms to simplify the interaction rules involved in generating the various features.

First, a rigid priority system is enforced on the feature types. Features are always added to the terrain in order of descending priority. The priority assigned to each is determined by a combination of the flexibility of its resulting shape and the difficulty involved in determining the shape. We create the terrain features in the following order: Fixed points, lakes, rivers, terraces, roads, ridges.

Second, each feature stroke is able to claim all the potential points in its region as belonging to that feature. Each potential point has a flag that indicates whether or not it has been claimed by a terrain feature. This flag is necessary because not every potential point in a feature's region is added to the mesh when defining that feature's shape and may be added later by the midpoint displacement process. With a few exceptions described below, new features do not modify points that have been claimed by previously generated features. Since high priority feature types are added to the terrain first, and therefore claim the points first, they are given control over the shape of the terrain when two or more feature strokes overlap.

3.2.3 Minimal Changes to Terrain for Sketch Edits

The focus of this system is to provide an interactive experience for the artist, allowing for an iterative design process. It is expected that an artist will sketch several terrain features, decide which look good and which do not, then make appropriate changes. As the user makes changes to one portion of the sketch, other areas of the terrain should not be affected (see figure 3.2). Our system takes several steps to make this possible.

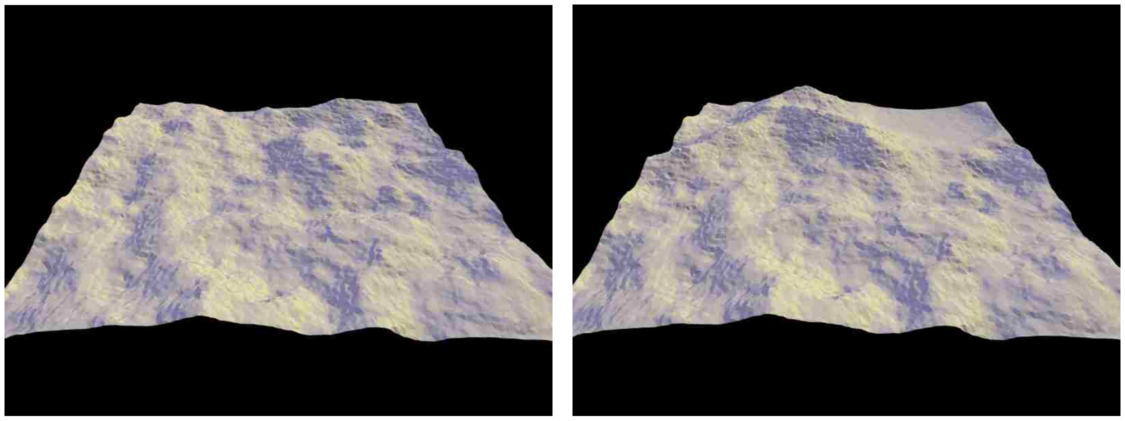


Figure 3.2: Minimal changes from sketch edits. Although ridges and a lake have been added to the back of the terrain in the right image, the front portions of the terrain remain unaffected.

First, the displacement variability of each point is determined by the order the vertices are inserted. In order to maintain consistent variability we must maintain consistent insertion order. To do this, we create a list of pointers to each of the potential points in the terrain and use the list order as our vertex insertion order. The list is shuffled to give us the required insertion order randomness. Once the insertion order is determined, it is left unchanged.

Second, the random value used as the variability percentage for displacing each point is stored along with the other data for that potential point and left unchanged between terrain generations. These random numbers are the primary determiner for where hills appear in the terrain. As the sketch is changed in certain locations, the insertion order is changed, thereby changing the variability at those locations. The nature of the midpoint displacement algorithm adjusts the terrain to fit. Farther from the sketch changes, the variability is determined more by the unchanged features and our constant insertion order. This, along with the random number reuse for variability, means that those distant hills remain unchanged.

Third, several feature types use noise functions in their creation. A single large set of 1D midpoint displacement noise data is generated and stored before the terrain is first generated. We use a random monotonically increasing noise function in the creation of rivers as described in section 3.2.4, which is also pre-generated and stored. Each feature stroke will use a predefined portion of this function. When a user paints a stroke of a feature type that needs either noise function, we store a random value that acts as a starting index for a segment of the given noise function.

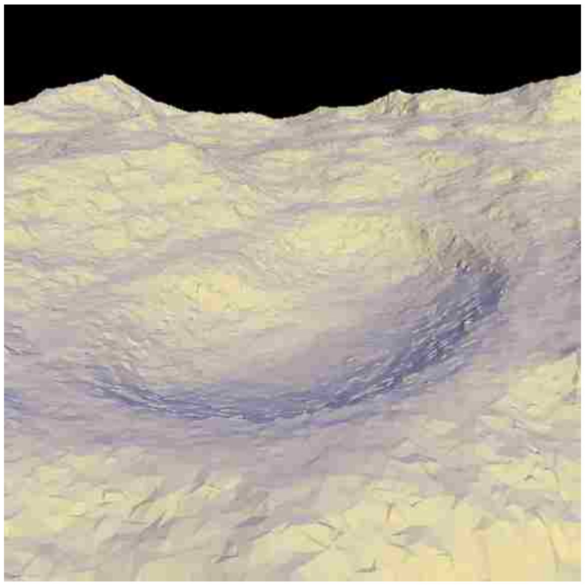
Finally, during the creation of certain feature types, we measure or modify randomly selected terrain locations. For example, we read the current height of the terrain at these random locations to determine the appropriate elevation for a terrace. Alternatively, we insert vertices at random locations within each feature's region to control the terrain shape in lakes and mountain ridges. To avoid changing which points are used, we preselect a subset of our potential points to be used in these cases.

3.2.4 Feature Type Creation Details

Fixed Points

Like other terrain constraint systems, we allow the user to specify the exact height of a particular location. Unlike other systems, however, this is not intended to be the primary method of defining the terrain layout. It is expected that users will place at most a small number of fixed points in the terrain to give a rough shape to the terrain that other features can follow. For example, placing a high elevation fixed point in each of the northern corners and a low elevation fixed point in each of the southern corners gives the terrain an overall south facing slope that is maintained even after adding other features. User-specified fixed points are the first vertices inserted into the terrain mesh.

Lakes



User Parameters
Depth Scale - scaling factor applied to the depth of each point in the lake
Initial Variability - amount of random height assigned to each terrace point
Standard Fractal Parameters

Figure 3.3: Image of a generated lake (left) and user-definable parameters on the lake brush (right).

The first step in creating the lakes is to create a signed distance map that represents the distance of each point from the shorelines. This map is used later to calculate the heights of the points we insert into the terrain. Interior points have negative values and exterior points have positive values. For speed reasons, we use Manhattan distance as opposed to other distance metrics, such as Euclidian distance. This map is created for all lakes simultaneously so that nearby lakes interact properly. Also note that as we create each lake, any overlapping lakes are combined into a single lake.

Once the signed distance map is generated, we begin the construction of each individual lake, starting with the shoreline. Regularly spaced points are selected along the border of the lake region. We find the current height of the terrain for each of the border points and select the minimum value, which is used as the elevation of the shoreline for the lake. We then insert each of these border points into the terrain at the determined elevation. Using the lowest point of the shore in the existing terrain

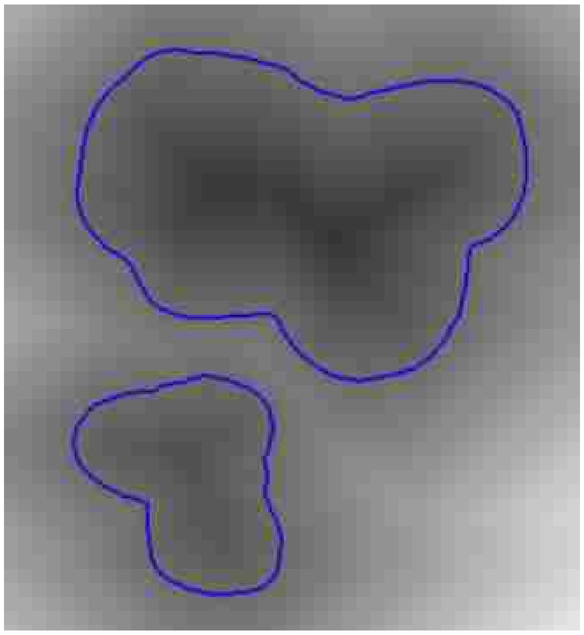


Figure 3.4: Visual representation of the signed distance map used in the creation of lake features. The blue outlines represent the shoreline and have a distance of 0. Image intensity represents the distance outwards from the shorelines.

improves the likelihood that nearby hills slope down towards the lake. Optionally, the system can see if the user has placed a fixed point within the lake's region and use the height of that fixed point as the height of the shoreline. This allows the user to directly specify the elevation of the lake.

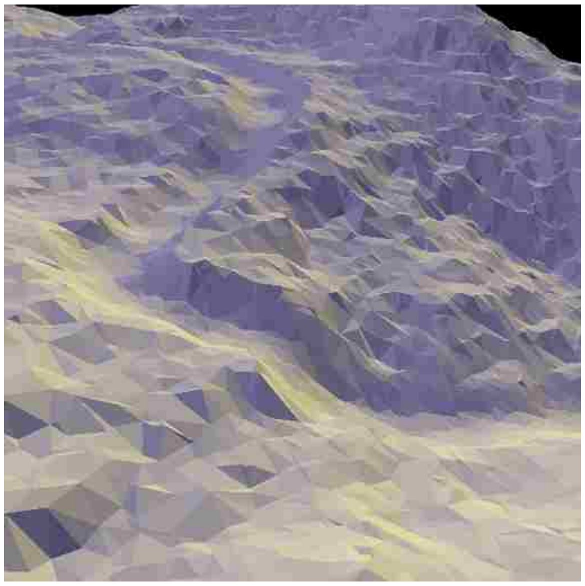
Next, we insert points to form the lake bed. This is done by inserting randomly selected points into the interior of the lake region at calculated heights. As mentioned above, these points are preselected and are reused each time the terrain is generated to ensure the lake is shaped the same each time. There are several possible methods to determine the height of each point that is inserted, each method is a function of the distance of the point to the shoreline as found in our signed distance map. As one option, we can use the value in the signed distance map directly. This results in lakes that have lake beds that slope consistently downwards towards a pointed bottom. As another option, we can use the distance, but clamp it to some maximum depth resulting in flat-bottomed lake beds. As a third option, we can scale the distance map values with a falloff to provide more rounded lake beds. Whichever function we apply to the distance, we add the result to the height we selected for the shoreline to determine the height of each inserted point. In addition to adding the random points to form the lake bed, we must also check for any points that were inserted before the lake was created and adjust their heights to match.

After the lake bed is built, we add points outside the lake border to create an elevated shore. This is done primarily to guarantee that the lake will correctly hold water. To do this, we add randomly selected vertices just outside the lake in much the same way as we do to create the lake bed. However, instead of adding the signed distance map value to the shoreline elevation, we add the signed distance map value to the current height of the point to be inserted. Note that we must find the current height of all of the points to be added before we actually add any of them or we change the terrain and therefore the current heights. Again, we use our preselected random points, inserting them at the calculated raised height. We also check again for any preexisting points and adjust them accordingly.

Finally, we store information about the lake for future use. Every potential vertex found within the lake's region is marked as claimed. The fractal parameter values specified by the lake brush are stored in the points and the fractal parameters are marked as overridden. The shoreline height is saved as the water height for each of the potential points in the lake region for use when creating rivers.

Rivers

After all of the lakes have been built, we generate the rivers one at a time. We begin by computing the elevation of the river's water surface for each point along the stroke. This elevation is constrained at various locations. The surface elevation at the start and end of the user's stroke are constrained to the current height of the terrain at the corresponding locations as read from the terrain mesh. The surface elevation



User Parameters
Width - width of the river
Depth - depth of the river
Width Noise - amount the noise function affects the river's width

Figure 3.5: Image of a generated river (left) and user-definable parameters on the river brush (right).

is also constrained to the height of any existing water areas that the river stroke crosses. The water height stored during the construction of lakes is used here. Rivers will also store their surface heights as they are built for use by subsequent rivers. To compute the surface height of segments of the river that remain unconstrained, we use a portion of a pre-generated monotonic noise function, scaled to fit the constraints at each end of the unconstrained segment. This monotonic noise function gives a natural appearance while ensuring that the river flows in only one direction between contrained sections.

We build the monotonic noise function in much the same way as we do a 1D midpoint displacement noise function. To start, our minimum domain position is assigned the minimum value in the range, and the maximum domain position is assigned the maximum range value. These extent values are chosen arbitrarily—since the function will later be scaled to fit terrain elevations, the only stipulation is that the chosen maximum value is larger than the chosen minimum value. Just as in the midpoint displacement algorithm, we recursively subdivide our domain in half, moving the midpoint of the segment up or down. However, instead of basing the variability of the midpoint on the size of the domain as in the midpoint displacement algorithm, we base the variability on the range of the current segment of the function. The midpoint is allowed to move at most halfway towards the value at either end of the segment (see figure 3.6). As mentioned in section 3.2.3, this noise data is calculated once and stored for reuse each time we generate the terrain.

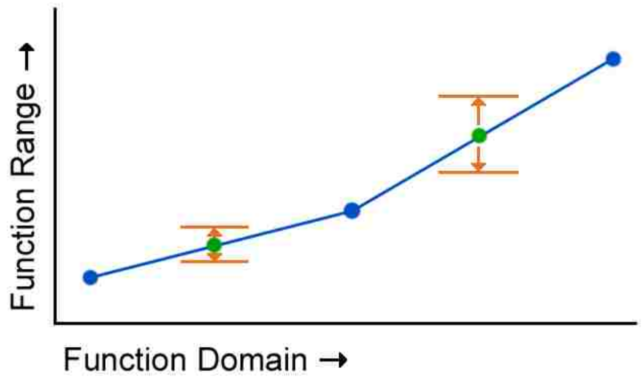


Figure 3.6: Generation of a monotonic noise function by recursive subdivision. As new points (shown in green) are added at the midpoint of each recursively subdivided segment, they are moved up or down by a random amount at most halfway towards either neighboring point's value.

Once we have calculated the water surface height for each point along the river stroke, we can insert vertices into the terrain mesh to represent the river bed and shores. At each stroke point, we find a vector perpendicular to the river direction. At a distance of half the river's width from the stroke point in the direction of the perpendicular vector, we place a shore point on each side of the river stroke. We use the stroke point's surface height as the elevation for each shore point. To give depth to the river and define the river bed, we insert one vertex at the location of the stroke point and another vertex halfway between the stroke point and each shore point. The height of each of these is calculated as the surface height at the stroke point minus the river depth.

It is important to note that each of these points must come from our set of potential points. We find and use the potential point nearest the calculated location for each vertex we insert for the river.

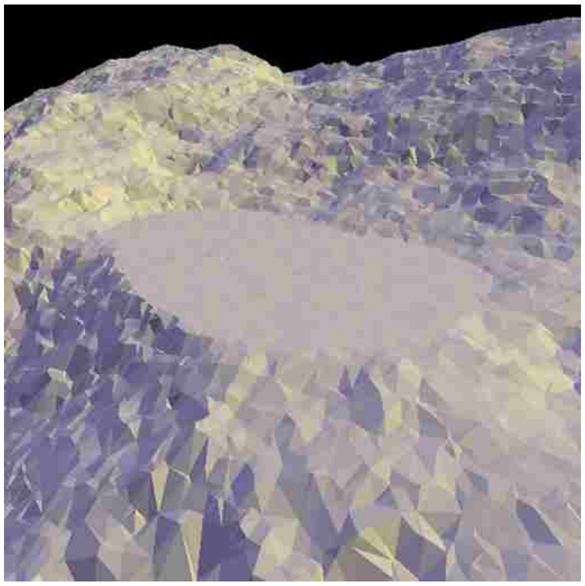
As we place the vertices for each river stroke point, we must also adjust the elevation of any existing terrain vertices that we cross. This allows the river to cut through lake and river shores, ensuring proper interaction between this river and previously constructed rivers and lakes. To do this, we find the quadrilateral region bounded by the two shore points corresponding to the current stroke point and the two shore points corresponding to the previous stroke point. Any vertices previously placed in the terrain mesh inside the quadrilateral that are currently above the elevation of the river bed are lowered accordingly.

While we scan the potential points for those that need to be lowered, we also mark all potential points that fall in the quadrilateral region as claimed. We also override the fractal parameters on each of these points to consistent and appropriate values. We use parameter values that give a smooth, low-variation appearance for our river beds.

Note that the described methods of handling feature interaction—constraining the new river's elevation to the elevation of previous features and lowering existing shorelines—only cover interaction with features that have been previously generated. They do not cover river strokes that self-intersect. Any self-intersecting river strokes should be first divided into three separate river strokes at the intersection point to remove the self-intersection. These three new river strokes are the segment from the beginning of the stroke to the self-intersection point, the looped segment that starts and ends at the self-intersection point, and the segment from the self-intersection point to the end of the stroke.

To improve the appearance of the river, we modulate the width of the river at each stroke point according to the 1D midpoint displacement noise function that has been previously generated. Note that this is a standard midpoint displacement noise function and is not the monotomic noise function we use for the river elevations. The amount we scale the width is a user configurable parameter for each river stroke.

Terraces



User Parameters
Initial variability - amount of random height assigned to each terrace point
Standard Fractal Parameters

Figure 3.7: Image of a generated terrace (left) and user-definable parameters on the terrace brush (right).

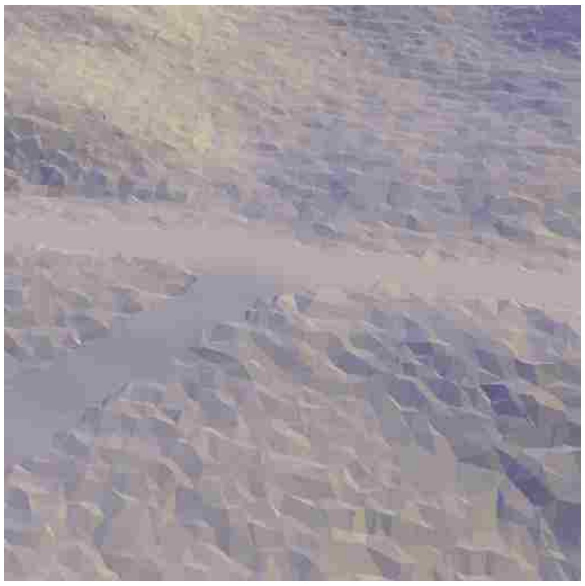
Terraces, as defined in our system, are simply level areas of terrain. Figure 3.7 shows an example of a terrace. To create the terraces, we first calculate the elevation of the terrace. If there are any fixed points placed by the user within the region of the terrace, the average elevation of the fixed points is used as the terrace elevation. If not, we check to see if any of our preselected random locations fall within the terrace region and use the average current height of those locations in our terrain mesh. If none of our random points fall within the region of the terrace, as may be the case in small terrace regions, we sample the terrain height at the locations of the terrace's border points. Just as we did with the lakes, we use an evenly spaced subset of border points.

Next, we insert vertices of the determined height at the border locations (again using the evenly spaced subset of border points) and the preselected random locations. If desired, the actual height of these inserted points may be adjusted randomly according to a user-defined variability parameter. To do this, we use the stored random variability percentage for each inserted point.

Finally, we update all potential points that fall within the terrace's region, marking them as claimed and overriding the fractal parameters to match the terrace brush's fractal parameters.

Interactions between the terrace and other features are handled as follows. Any two overlapping terraces are combined into a single terrace. Regions of potential points that have already been claimed by previously generated features are subtracted from the terrace's region before the terrace is created.

Roads



User Parameters
Width - width of the road

Figure 3.8: Image of a generated road (left) and user-definable parameters on the road brush (right).

See figure 3.8 for an example of a generated road. We begin the construction of each road by reading the current height from the terrain at each stroke point.

These heights are smoothed by averaging each point with its two neighboring stroke point heights. As in the river generation process, once we have the elevation of each stroke point, we place a number of points along the vector perpendicular to the stroke direction. However, for roads, we place each of the points at the same elevation. We also find it helpful to insert more points along each perpendicular vector than we do with rivers. Like rivers, each point placed along the perpendicular vector must be one of our preexisting potential points, so we pick the potential point nearest to each calculated location. If a potential point has already been claimed by another feature, we leave it unchanged.

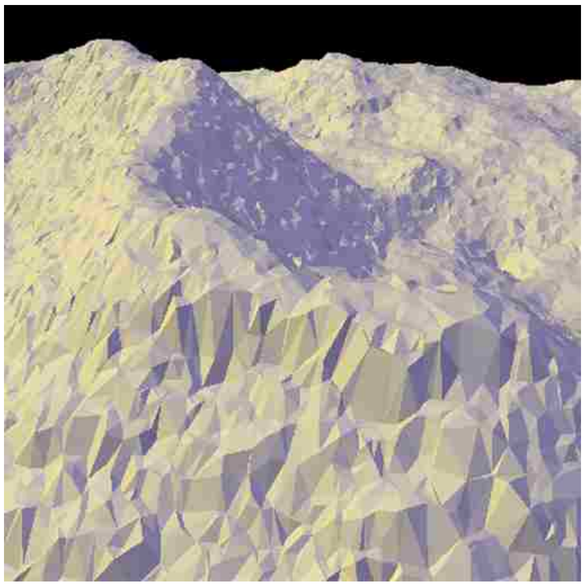
When roads curve too quickly or at road intersections, points of inconsistent heights may be placed together, resulting in a very rough surface. It is possible to use the same technique for adjusting the height of existing points as we do for rivers, but we have found it usually sufficient to simply apply a strong median filter to the road.

Just like other feature types, once the vertices have been added to the terrain, we mark all of the points in the road's region as claimed. We also override the fractal parameters for the points in the road's region. We set the variability to be near zero and the median filter and blur kernel sizes to relatively high values.

Mountain Ridges

Ridges are created by adding height in the shape of a mountain ridge to existing terrain. In this way, we can keep the basic shape of the underlying terrain while quickly forming large mountains. Figure 3.9 shows an example mountain ridge.

We begin by creating a single height map that represents the heights of all the ridges that will be added later. An example heightmap is shown in figure 3.10 and pseudocode is given for this process in algorithm 1. At each stroke point for each ridge, we calculate the height for the stroke point. To produce a more realistic



User Parameters
Height - height of the ridge
Slope - slope of the mountain from the ridgeline to the base
Ridge Variability - amount the ridgeline's height is varied using the 1D MDA function
Standard Fractal Parameters

Figure 3.9: Image of a generated mountain ridge (left) and user-definable parameters on the ridge brush (right).

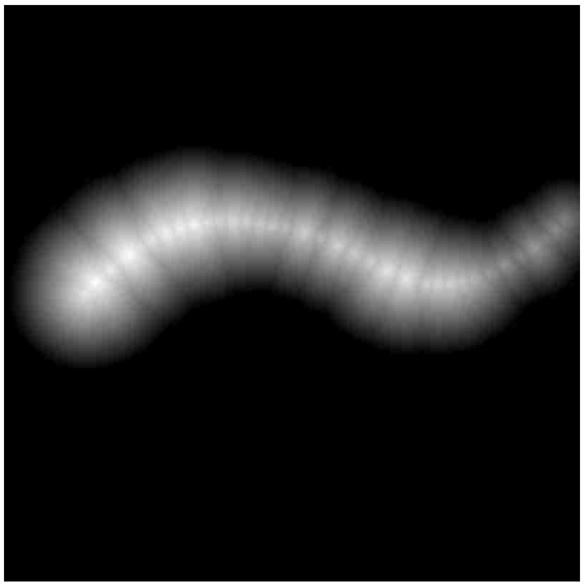


Figure 3.10: Heightmap generated from a ridgeline stroke.

Algorithm 1 Ridgeline heightmap generation

```
1: initialize each position in $heightMap$ to 0
2: for each ridge stroke do
3:   for each point $p$ in the stroke do
4:     if stroke point location is not claimed by a previous feature then
5:       // calculate the final height of the ridge at this point
6:       $mdaAmount \leftarrow mdaValues[strokePointIndex + startIndex]$
7:       $heightAdjust \leftarrow heightParam \cdot mdaAmount \cdot mdaInfluenceParam$
8:       $height \leftarrow heightParam + heightAdjust$
9:       $influenceRadius \leftarrow height/slopeParam$
10:      for each potential point that has been claimed by another feature do
11:        if distance to claimed potential point $\leq influenceRadius$ then
12:          $influenceRadius \leftarrow$ distance to claimed potential point
13:        end if
14:      end for
15:      $height \leftarrow influenceRadius \cdot slopeParam$
16:      // update the heightmap for this stroke point
17:      for $x \leftarrow p.x - influenceRadius$ to $p.x + influenceRadius$ do
18:        for $y \leftarrow p.y - influenceRadius$ to $p.y + influenceRadius$ do
19:          $dist \leftarrow$distance$((x, y), (p.x, p.y))$
20:          $heightMap[x, y] \leftarrow$max$(heightMap[x, y], height - dist \cdot slopeParam)$
21:        end for
22:      end for
23:    end if
24:  end for
25: end for
```

profile, we apply our pre-generated 1D midpoint displacement function data to the ridgeline height (lines 6-8). Each ridge is given a different random starting position in our midpoint displacement data (appears in Algorithm 1 as $StartIndex$).

We then find the maximum range of influence for each point by dividing the point's height by the ridge slope parameter (line 9). If there are any points in the terrain that have already been claimed by other features within the range of influence, we reduce the height of the point so that the claimed point is just outside the range of influence. This is done by calculating a new influence then adjusting the height (lines 10-15). This height reduction gives us plausible ridgelines along the user's strokes that still interact properly with other features. At every point in the heightmap within each stroke point's final range of influence, we take either the current value in the heightmap or the stroke point's influence at the point, whichever is greater (lines 17-22).

As we generate the heightmap, we override the fractal parameters of the potential points that fall within the range of any stroke point's influence. The new fractal parameters come from the user's brush parameters.

Once the heightmap is complete for all of the ridges, we store the current terrain height at each location we will be placing a vertex. These locations include the stroke points (if they are not already claimed by other features), preselected stored random locations, and any vertices that have already been placed in the terrain. Note that we only care about locations that correspond to positive values in our heightmap. Finally, we place vertices into the terrain at each of these locations, adding the value in the heightmap to their previous elevations.

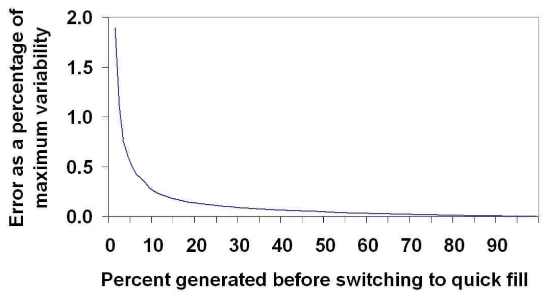
3.2.5 Speed Optimizations

Because of the time complexity of the incremental Delaunay triangulation algorithm, the modified algorithm used in our system is slower than the traditional algorithm.

However, in an interactive system, the user does not generally need the full quality result during the creation process—a lower quality preview that is indicative of the final result is usually sufficient.

For the preview terrain, the first optimization is to simply not place all of the grid points. We have found that using 7%–8% of our grid points is a good balance of speed to quality for terrain previews. The actual percentage of points placed for the preview is user-controlled. Note that the post-process filters must account for holes in the point grid when creating these partially generated meshes.

After a significant portion of vertices have been placed, each new vertex has a relatively small distance to its nearest neighbor. For these small distances, both the variability and derivative of the variability are low. This means that at this point in the process, adding a new vertex to the graph affects the terrain much less. Once a set percentage of points have been placed, we assign heights to the remaining points in the terrain without adding them to the Delaunay graph. The previous height at the location of the new points is still determined from the Delaunay graph but the insertion costs are avoided and the size for the $O(n)$ lookup does not increase with each new point when using this optimization. The final mesh triangulation is calculated using a faster single-pass method.



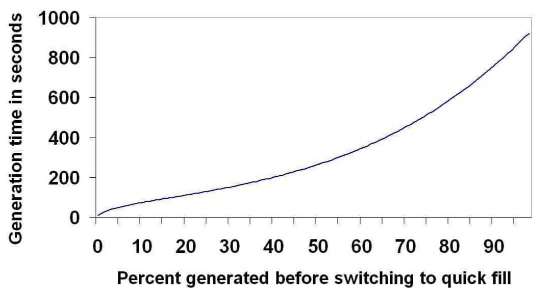


Figure 3.11: Effects on error (left) and generation time (right) of switching to the quick fill optimization after adding a given percentage of points using the regular fill method.

Figure 3.11 shows a graph of the root mean square error of the terrain elevation and the terrain generation time when using this optimization. The error is given as a percentage of the maximum variability parameter. The “percentage generated” axis represents the amount of vertices that are inserted into the Delaunay triangulation before switching to the optimized method of only reading from the triangulation. When switching to the optimization after 7%, the generation time for one million faces drops from 16 minutes to approximately 1 minute with a root mean square error equal to 0.38% of the maximum variability. The error is visible as a slightly rougher terrain surface as seen in figure 3.12, but does not affect the large scale shape of the terrain. The effect is significantly lessened by the post-process blur and median filters.

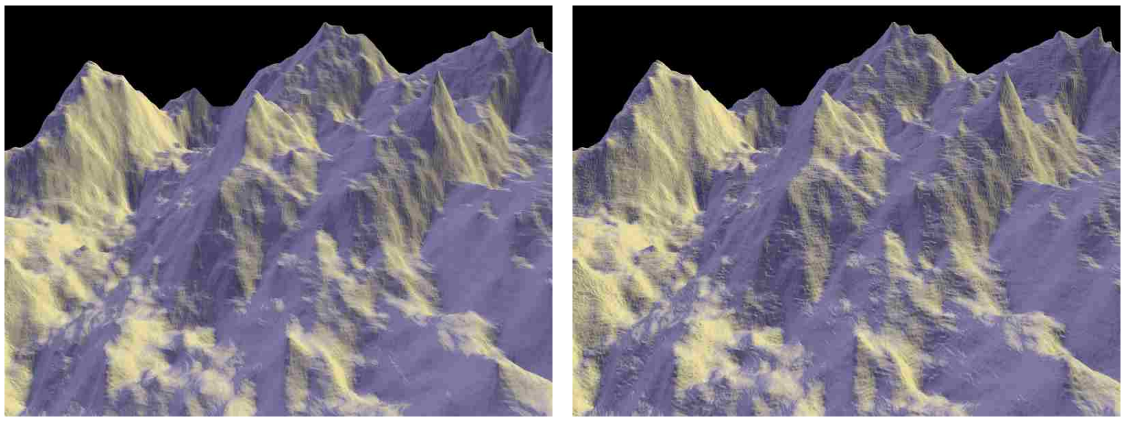


Figure 3.12: Terrain generated with the regular insertion method (left) and with the optimized quick fill method after 7% (right). The optimized terrain is slightly rougher than the regular terrain but maintains its basic shape and is generated much more quickly.

3.2.6 Edge Trimming

Unfortunately, the modified midpoint displacement algorithm does introduce some artifacting near the edges of the terrain (see figure 3.13). The elevation of the extreme edge points is pulled towards the elevation of the corner points. We have found that a simple solution is to initially generate the terrain slightly larger than desired—generally 5% larger in each direction—then trim the edges from the resulting mesh.

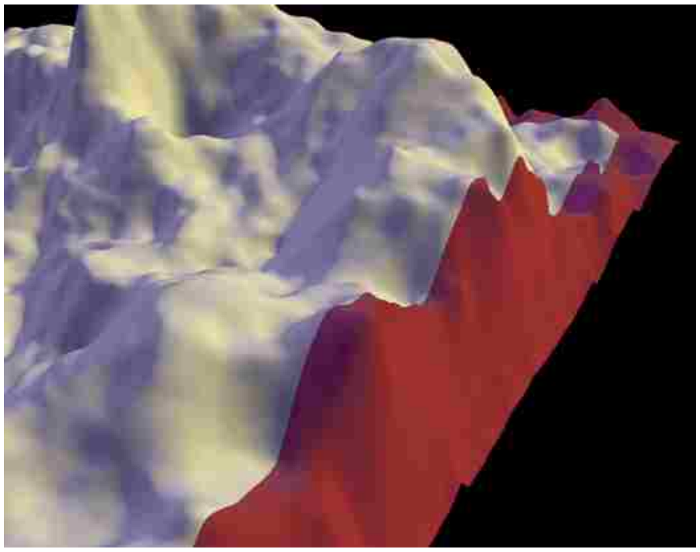


Figure 3.13: Edge artifacting and trimmed region. Heights at terrain edges pull towards the corner elevations. We therefore remove a small percentage of the terrain near the edge (removed portion shown in red).

3.2.7 Summary

We control the formation of the fractal terrain by placing vertices into the terrain mesh at specific locations according to the user's sketch. A list of all potential points is used to store information about points that are added later in the process. Feature interaction is handled primarily by assigning a creation order priority to each feature type and by allowing created features to claim potential points, preventing later features from affecting the area. All random values used in the midpoint displacement process are stored between regenerations to minimize changes as the user edits the terrain sketch. Significant speed optimizations for terrain previews are possible because the first vertices placed in the terrain affect the appearance more than vertices placed later.

Chapter 4

Results

4.1 Modified Midpoint Displacement Algorithm

The terrain produced by the arbitrary vertex insertion order midpoint displacement algorithm has similar qualities to terrain produced by the traditional square-diamond midpoint displacement algorithm, with some notable exceptions. The vertex insertion order of the traditional algorithm produces ridges that nearly always line up with major axes. Ridges in our modified algorithm may line up in any direction and can even curve naturally. Terrain generated by our algorithm is also better able to handle regions with different fractal parameters.

4.2 Interactivity

We are able to produce the terrain previews in 5 - 15 seconds depending on the sketch complexity. Fully generated meshes containing one million faces can be produced in under 1 minute using the quick fill optimization with very little loss in terrain quality. As shown in figure 3.2, the information stored between terrain generations allows the user to add or remove feature strokes with minimal changes to other areas of the terrain. The placement of the features in the terrain always precisely matches the layout sketch (see figure 4.1).

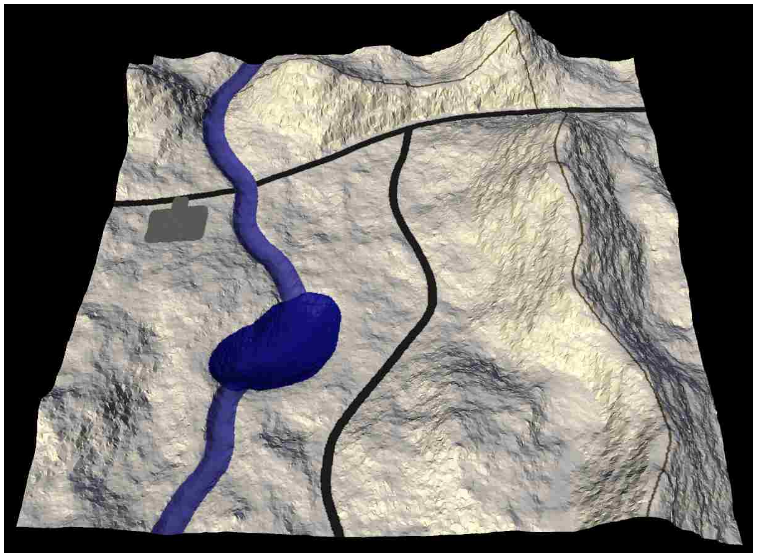


Figure 4.1: Terrain that has been textured with its own sketch to demonstrate feature placement accuracy.

4.3 User Control

Using procedural terrain to fill in unconstrained regions of the terrain allows the artist to concentrate only on important areas of the layout. Figure 4.2 shows how a very simple sketch can specify a windy road through a mountain pass. At the same time, because of the straightforward interaction rules, the system can easily handle complex feature interactions, such as those in figure 4.3, without resulting in implausible terrain. The user can specify as much or as little of the terrain layout as is desired.

4.4 Future Work

Applying terrain erosion algorithms to the output of our terrain synthesis could greatly improve the visual realism. The sketch interface of our system would be an appropriate place for the user to specify erosion parameters at the same time as he or she is controlling the fractal parameters and feature layout.

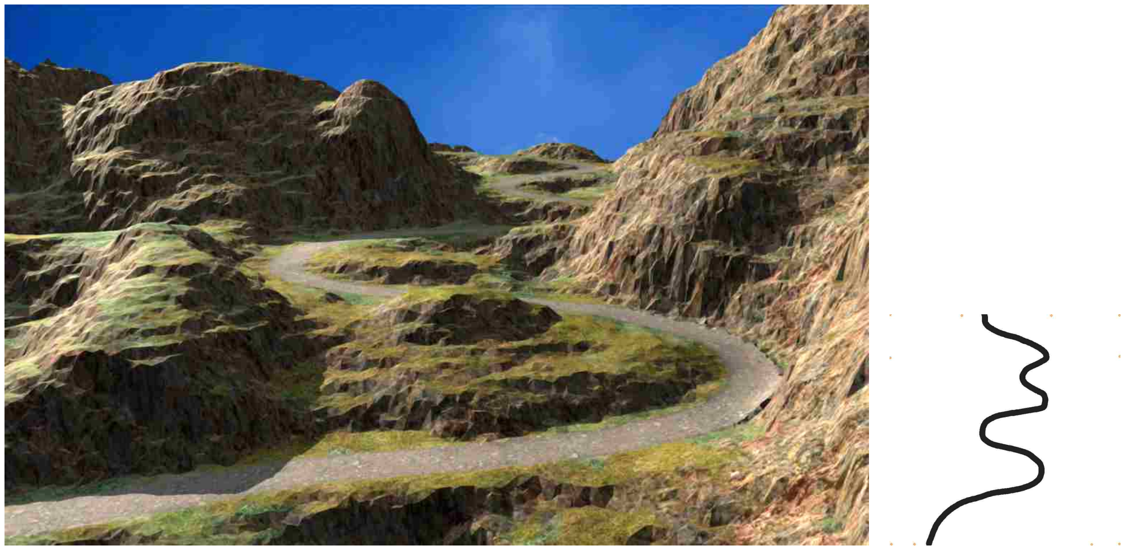


Figure 4.2: Mountain pass created with 10 fixed points and a single road feature stroke.

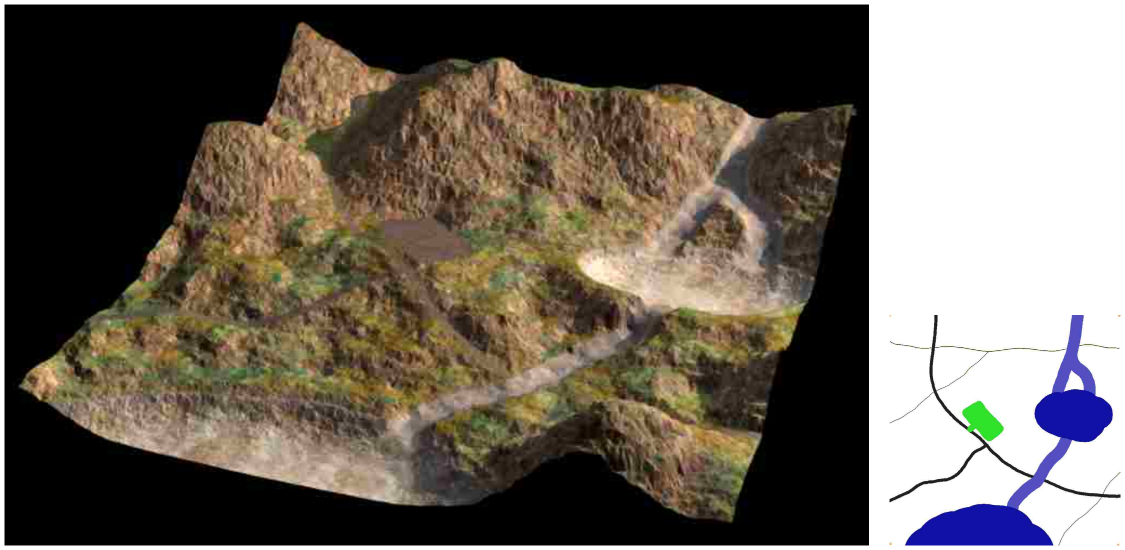


Figure 4.3: Generated terrain with complex interactions of multiple feature types.

This paper discusses methods for procedurally generating the shapes of multiple terrain feature types. In nature there are often many variations to each of these basic feature types, such as secondary ridges on mountains, or deep cuts in the bed of winding rivers. We wish to experiment with methods for describing these variations through the user sketch or the brush parameters.

In some situations, procedural terrain is used to generate new terrain without any user intervention. For example, video games sometimes use procedural terrain to create new areas of the game for players. We feel that by finding a method to procedurally generate the sketches used by our system, instead of having them manually created, we could add an extra layer of abstraction to the process. Rather than specifying just variability and elevations, programmers could directly specify feature layout rules. This would simplify the terrain generation rules and result in terrain that better follows the programmers' expectations.

In the traditional midpoint displacement algorithm, larger expanses of terrain can be generated by tiling multiple terrain regions. As they are generated, edge points are shared between the adjacent tiles. Unfortunately, tiling regions in our system is not so simple. It may be possible to implement tiling as follows. A separate Delaunay triangulation would be maintained for each tile. It would be determined which neighboring tiles, if any, are affected by a newly added vertex. The vertex would be added to each tile it affects (and only those tiles it affects). The first few vertices would be added to many or all of the tiles, but this could be done quickly because of the low vertex count in each tile. Later vertices would be added to only a small number of tiles, likely just one, with each tile still having a much lower vertex count than the full terrain mesh normally would. We believe that this may reduce the overall expected generation time and allow for larger terrain meshes.

4.5 Conclusions

We have presented new methods for generating procedural terrain from user sketches. The high level feature-based sketch interface gives artists tools to define the terrain layout more intuitively than point constraint methods. Our arbitrary vertex insertion order midpoint displacement algorithm provides excellent flexibility for constraining terrain to fit the user sketches. This has resulted in a wider variety of sketched terrain feature types than previously allowed by other systems. The methods introduced in this paper efficiently and correctly handle the interactions between multiple terrain features.

We have also described techniques for improving the interactivity of terrain synthesis. Our system can rapidly generate the synthesized terrain, giving the user the opportunity to make corrections as needed. Despite the random nature of procedural terrain, users are able to change portions of the sketch without impacting other regions of the terrain layout. Together, the techniques we have presented reduce the time needed to create terrain models and improve the directability of the terrain synthesis process.

References

[BA05] Belhadj F., Audibert P.: Modeling landscapes with ridges and rivers: bottom up approach. *GRAPHITE '05: Proceedings of the 3rd international conference on Computer graphics and interactive techniques in Australasia and South East Asia* (New York, NY, USA, 2005), ACM, pp. 447–450.

[Bel07] Belhadj F.: Terrain modeling: a constrained fractal model. *AFRIGRAPH '07: Proceedings of the 5th international conference on Computer graphics, virtual reality, visualisation and interaction in Africa* (New York, NY, USA, 2007), ACM, pp. 197–204.

[FFC82] Fournier A., Fussell D., Carpenter L.: Computer rendering of stochastic models. *Communications of the ACM 25*, 6 (1982), pp. 371–384.

[GS85] Guibas L., Stolfi J.: Primitives for the manipulation of general subdivisions and the computation of voronoi. *ACM Transactions on Graphics 4*, 2 (1985), pp. 74–123.

[Lis94] Lischinski D.: Incremental delaunay triangulation. *Graphics gems IV* (San Diego, CA, USA, 1994), Academic Press Professional, Inc., pp. 47–59.

[Mil86] Miller G. S. P.: The definition and rendering of terrain maps. *SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1986), ACM, pp. 39–48.

[MKM89] Musgrave F. K., Kolb C. E., Mace R. S.: The synthesis and rendering of eroded fractal terrains. *SIGGRAPH '89: Proceedings of the 16th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1989), ACM, pp. 41–50.

[PH93] Prusinkiewicz P., Hammel M.: A fractal model of mountains with rivers. *Proceedings of Graphics Interface '93* (1993), pp. 174–180.

[SS05] STACHNIAK S., STUERZLINGER W.: An algorithm for automated fractal terrain deformation. *Proceedings of Computer Graphics and Artificial Intelligence* (2005), pp. 64–76.

[ST89] SZELISKI R., TERZOPOULOS D.: From splines to fractals. *SIGGRAPH '89: Proceedings of the 16th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1989), ACM, pp. 51–60.

[VM96] VEMURI B. C., MANDAL C.: A fast gibbs sampler for synthesizing constrained fractals. *VIS '96: Proceedings of the 7th conference on Visualization '96* (Los Alamitos, CA, USA, 1996), IEEE Computer Society Press, pp. 29–35.

[ZSTR07] ZHOU H., SUN J., TURK G., REHG J. M.: Terrain synthesis from digital elevation models. *IEEE Transactions on Visualization and Computer Graphics 13*, 4 (July/August 2007), pp. 834–848.