Fall 2011

RiverLand 2.0: Blending of Multiple User-defined Slopes in a Procedurally Modeled Terrain

Jeffrey Jensen
San Jose State University

RiverLand 2.0: Blending of Multiple User-defined Slopes in a Procedurally Modeled Terrain

A Writing Project

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Jeffrey Jensen

Under the Advisement of Dr. Soon Tee Teoh

August 2011

SAN JOSÉ STATE UNIVERSITY

The Undersigned Project Committee Approves the Project Titled

RiverLand 2.0: Blending of Multiple User-defined Slopes in a Procedurally Modeled Terrain

by

Jeffrey Jensen

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

Dr. Soon Tee Teoh	Department of Computer Science	Date
Dr. David Taylor	Department of Computer Science	Date
Dr. Chris Pollett	Department of Computer Science	Date

ABSTRACT

BLENDING OF MULTIPLE USER-DEFINED SLOPES IN A PROCEDURALLY MODELED TERRAIN

By Jeffrey Jensen

This writing project attempts to improve on and add features to the current program called RiverLand originally designed and implemented by Dr. Soon Tee Teoh. I discuss the original methods used by RiverLand to create procedurally generated terrain. I then explore the weaknesses of the original RiverLand which include having only linear ridges and undesirable medial axis cells. I then tackle the problem of recurring patterns when texturizing a surface with very few textures. I propose how to solve these problems and explain the methods used to accomplish this. I discuss the user interfaces that were designed to accommodate the added features to RiverLand. I also discuss the open problems with the updated RiverLand.

Table of Contents

List of Figures

Figures

1 Introduction

1.1 Background Information

1.1.1 Procedural Modeling

Procedural modeling is the method of generating 3D objects and environments automatically by following a set of rules and parameters. Procedural modeling can be very useful because it allows developers to generate large quantities of 3D objects without having to explicitly define each object's properties. Instead, parameters are given for the general characteristics of the object or environment, and random numbers are used to create something unique. This kind of modeling can be used to create such things as terrain, buildings, cities, trees, and many other objects. The challenge of procedural modeling is to develop an algorithm that produces realistic results while remaining efficient.

The content created from these procedural modeling algorithms can then be input into computer games, movies, or they might even be used as a starting point for users or developers to edit the content further to more closely meet their needs. Using procedural modeling techniques can greatly reduce the amount of time and effort needed to create complicated environments. For example, if a developer wanted to create a city for his 3D computer game, he or she could simply input which areas of the city are residential, commercial, and industrial as well as other important structures, and the city would be automatically generated using the given parameters as seen in Dr. Teoh's Autopolis [8].

An example where procedural modeling is used extensively is in a computer game called Minecraft. In this game, players explore a 3D world that is automatically generated as needed. The world can grow as long as there is enough memory to contain the world [7]. Figure 1 shows a player's currently explored world in Minecraft. Procedural modeling allows for expansive worlds that could never be produced explicitly by developers.

Figure 1: A game world procedurally generated in Minecraft

1.2 Original RiverLand

1.2.1 Overview

RiverLand was a program originally designed and implemented by Dr. Soon Tee Teoh. RiverLand generates an island with meandering rivers and calculates ridges based on those rivers [9]. The user is allowed to define the shape of the island by painting on a 2D canvas, as well as specify ridges lines which rivers cannot cross (as seen in Figure 2).

Ridges are also generated by searching outwards from the river edge's to find the furthest distance from all nearby rivers.

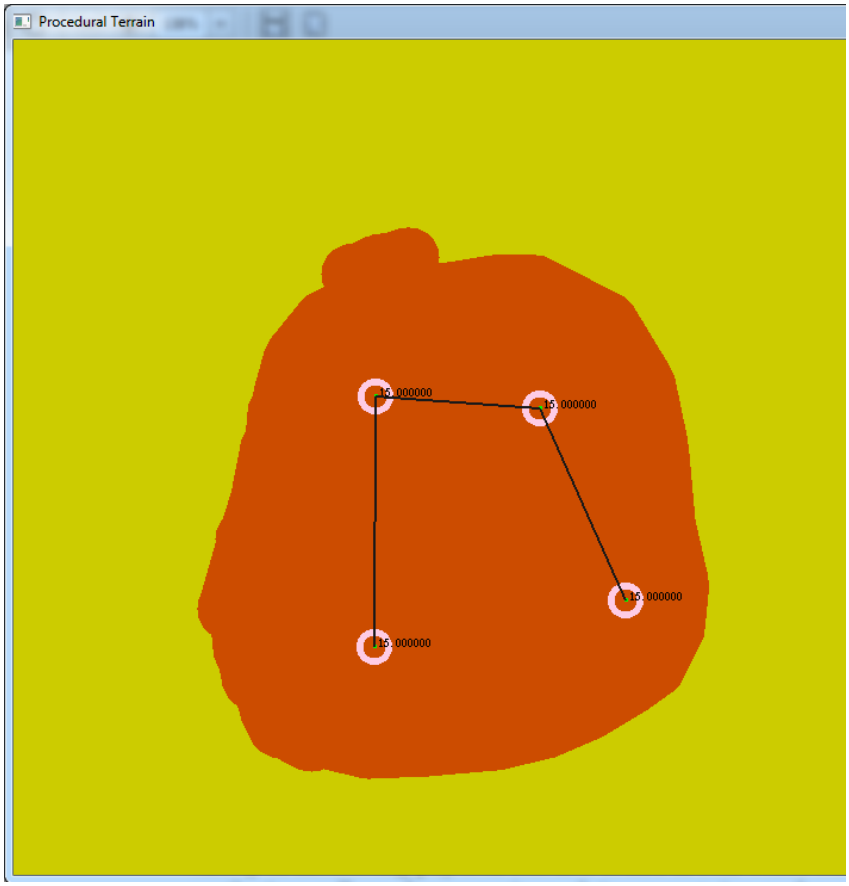

Figure 2: A user drawing an island in the canvas and creating ridge lines in RiverLand

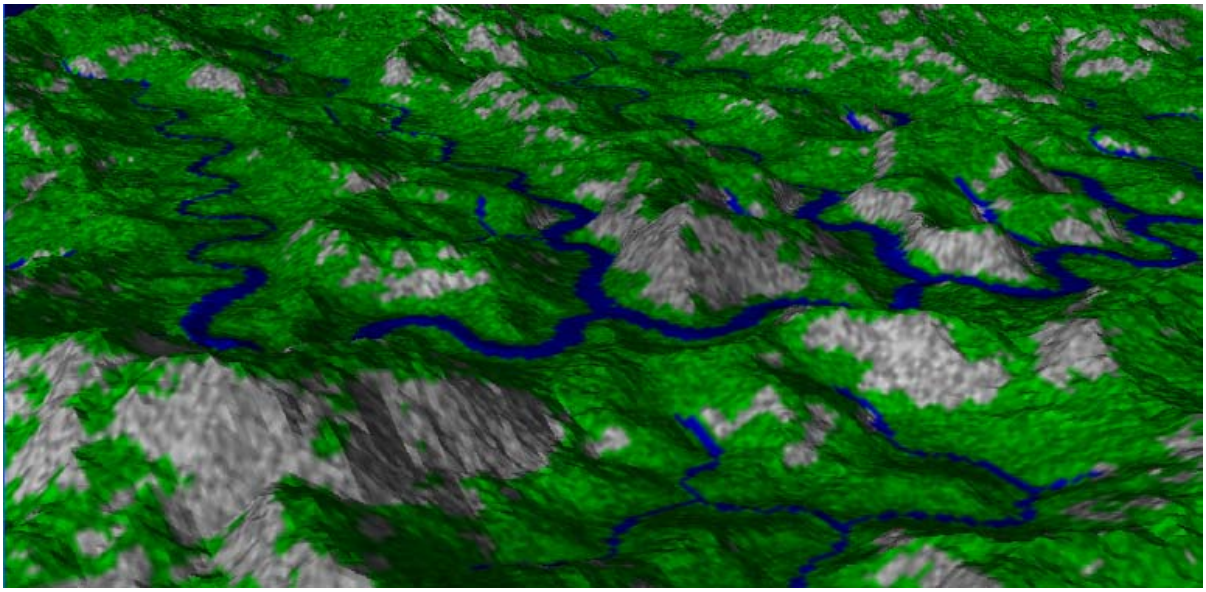

Figure 3: Original RiverLand rivers and ridges [2]

1.2.2 Generating Rivers

RiverLand generates rivers by first creating a number of randomly located river mouths, or seeds, along the island's coast. From each of these rivers mouths, a river is grown by going inland, perpendicular to the coast, for a certain distance. At this distance, the river direction is shifted by a random angle between 90 and -90 degrees. This process is repeated for a certain number of times to create a connected string of *SegmentPoints*. By adjusting the segment length and the skew of the river direction at each *SegmentPoint*, the user can create many varying river types, from long straight rivers to small winding rivers.

After these *SegmentPoints* have been generated, a meandering river is fit through the points. Between each *SegmentPoint*, the amount of river curvature is randomly created, alternating with each *SegmentPoint*, to create a river that continuously travels left and right. This will create more realistic rivers as most rivers in reality do not travel in straight lines.

To make the rivers seem even more realistic, tributaries are spawned from the rivers. These tributaries are grown outwards from randomly generated seed points on the concave banks of the rivers. They grow in the same way as the rivers, meandering through *SegmentPoints*. These tributaries may also have tributaries of their own, allowing for more interesting landscapes. As rivers and tributaries spawn tributaries of their own, their length and movement is more restricted.

After all the river networks have been generated, the height of these river cells must be determined. The river mouth is set to be zero, and for each river cell further inward, the height increases according to *AverageSlope* with some random offset. As the river goes further inland still, the river is allowed to have a steeper slope. This simulates the steep mountains that can sometimes exist near river sources.

1.2.3 Generating Terrain Height

To create the height of all the land cells, all medial axis cells must be found. Medial axis cells are cells that are the maximum distance from the nearby river and coast cells. An example of these cells is shown in Figure 4. To accomplish this, all the river and coast cells are put into a list which all have a distance *d* equal to 0, since they are the river/coast cells. For each cell in this list, all adjacent cells are found and are set to have $d_{adjacent} = d_{cell} + slope \times dist$, where *dist* is the horizontal distance between the cells and *slope* is usually 1.0. *slope* can be changed by the user to create ridges where each side of the ridge has a different slope. The cell is then removed from the list, while the adjacent cells are added to the list. A pointer is also added to these adjacent cells that specifies which cell sent them to the list. This process is repeated so that each land cell is set with its distance from the river.

When all the distances have been set, the cells which did not have a pointer to an adjacent cell are then considered to be the medial axis cells.

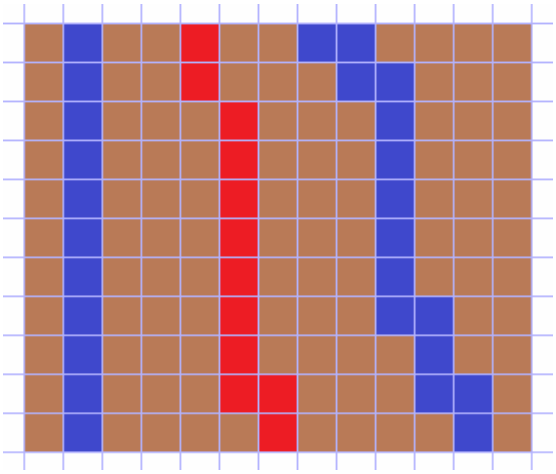

Figure 4: Example of medial axis cells; medial axis cells in red, river cells in blue, land cells in brown.

Once all medial axis cells have been found, the height of each medial axis is calculated by taking the height of the river cell that the medial axis cell came from (more precisely, the river cell that had the influence on *d*, from above) and adding *slope* × *d*. The height of all remaining cells can then be found by linearly interpolating along the path from each river cell to the medial axis cell.

1.2.4 Problems

1.2.4.1 Linear Slopes

The original RiverLand linearly interpolated along the path from the river cells to the medial axis cells to find the height of the cells in between. This type of interpolation creates smooth slopes but does not allow for varying types of ridges. This does not allow for things like rough canyons, valleys, and plateaus. The terrains that are generated with RiverLand appear very uniform. In the real world, most geological features do not have exactly linear slopes. Even when using a fractal overlay to give the slope some randomness, the prevalent shape of the slopes is still linear.

In Figure 5 below, the right side of the mountain does not have a linear slope but rather a more parabolic shape. This is one of the kinds of slopes we are trying to achieve in this project.

Figure 5: Mountain with non-linear slope [5]

1.2.4.2 Undesirable Medial Axis Cells

Another problem of the original RiverLand was that the terrain would be littered with medial axis cells that did not accurately approximate the ridges the user would like to have based on the rivers. These medial axis cells make it extremely difficult to integrate non-linear slopes into the terrain. The resulting ridges are short and run perpendicular to the main ridges.

As can be seen in Figure 6, there are many medial axis cells that are highly undesirable. Because of meandering rivers, many rows of medial axis cells are created that extend out all the way to the rivers. The desirable medial axis cells are the ones which form rows that are parallel to the rivers.

This problem is not relevant when all slopes in the terrain are linear, but when trying to alter the slopes to resemble something more realistic, we run into problems because we are targeting specific ridge slopes. When the user wants to pick a ridge to alter, it can be difficult to get the desired slope to appear when there are small surrounding ridges that still have linear slopes.

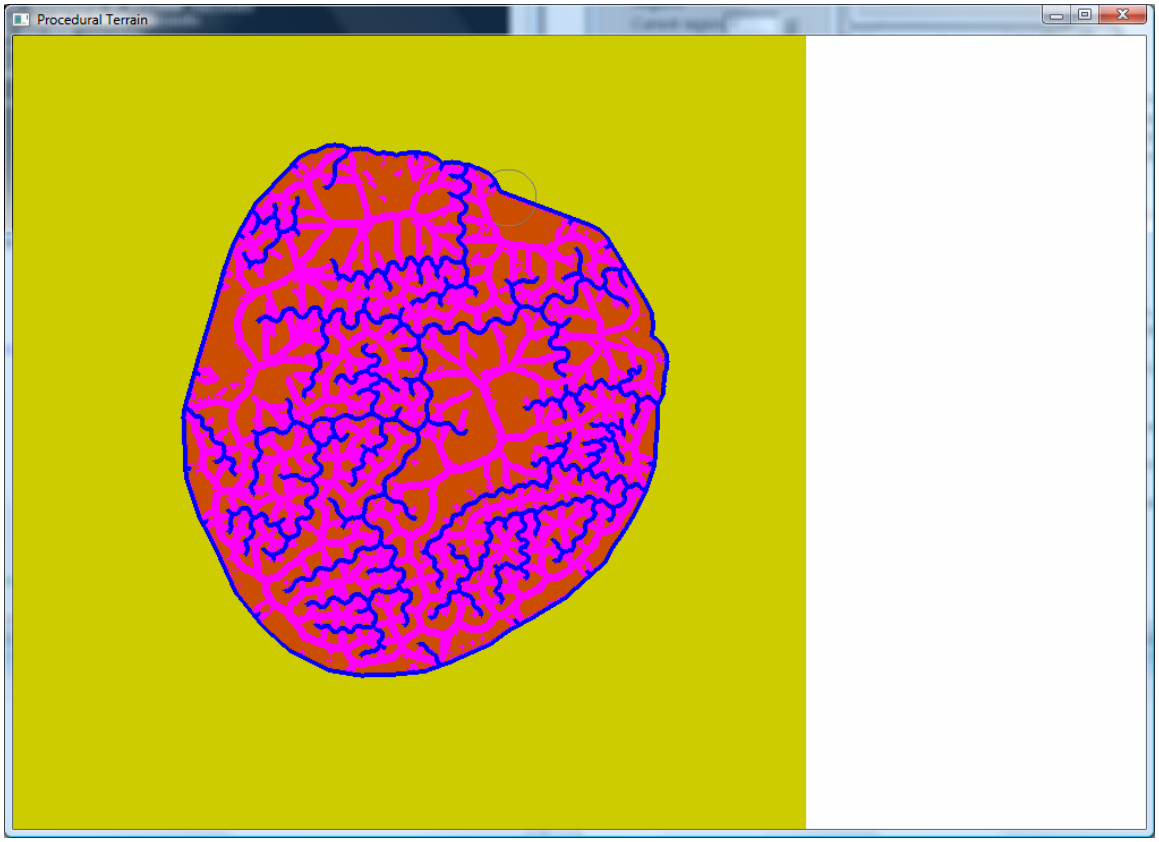

Figure 6: 2D view of terrain showing all medial axis cells in pink (Original RiverLand)

2 Integrating Multiple User Defined Slopes

2.1 Overview

In order to solve the problem of only having linearly interpolated slopes, the updated RiverLand features the ability of the user to define his or her own type of slope for a particular side of a ridge after the initial slopes have been generated. The user can also specify the amount of influence for where the slope will affect the terrain. The user defined slopes are then blended into the slope to create a seamless terrain.

2.2 User Interfaces Designed

Several interfaces were designed and implemented in order to accommodate the added feature of letting users define their own slopes. These interfaces are designed to give the user an accurate way to create the desired slope they want for the ridge.

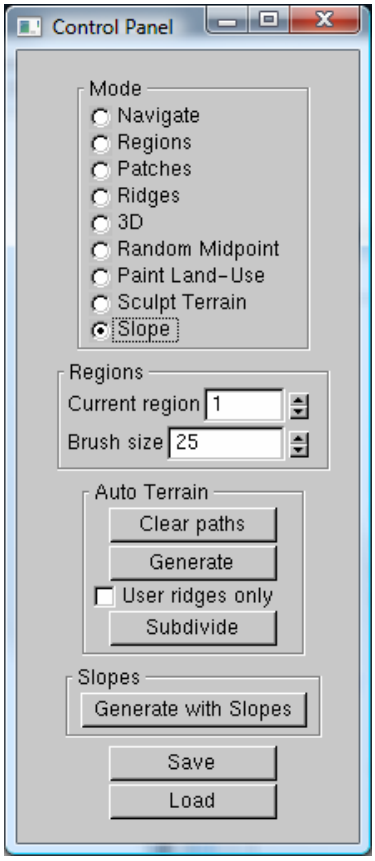

Figure 7: “Slope” mode added to “Control Panel”

Figure 7 shows the updated control panel for RiverLand which includes a new slope mode. To create a new user defined slope on the current terrain, the user must select “Slope” mode. The user must then select a point on the terrain which specifies the area on the terrain they would like to alter. The user can check the boxes “Show maximal cells”, “Show height map”, and “Show distance to river” in the 2D View Options interface to get a better idea of where to place the slope. These options let the user see the ridges and their slopes more easily.

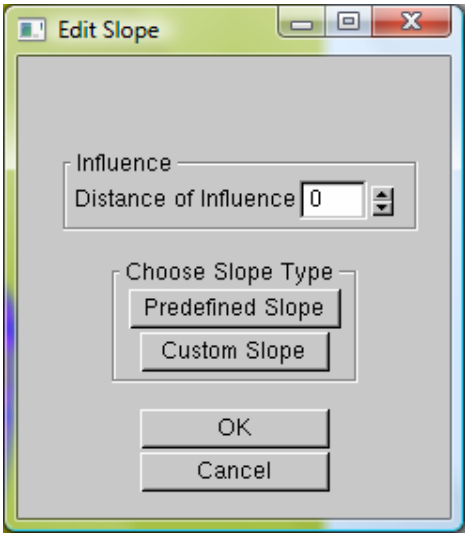

Figure 8: “Edit Slope” Interface

After selecting the area on the terrain where the user wishes to adjust the slope, the interface in Figure 8 is shown. This interface allows the user to set the distance of influence for the slope. The user can then choose either to have a predefined slope, which consists of a set of predefined functions, or a custom slope which the user can draw out.

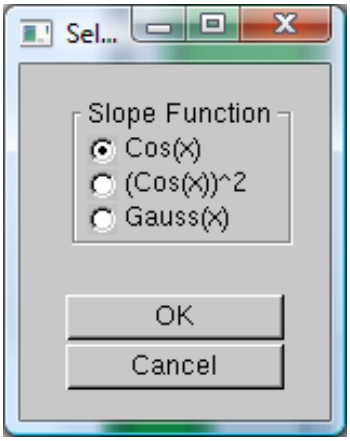

Figure 9: “Select Predefined Slope” Interface

If the user decides to have a predefined slope, they are presented with the interface in Figure 9. Currently the program offers three predefined functions: cosine(x), (cosine(x))^2, and Gauss(x) which is the slope of a standard normal distribution curve between 0 and 3.1.

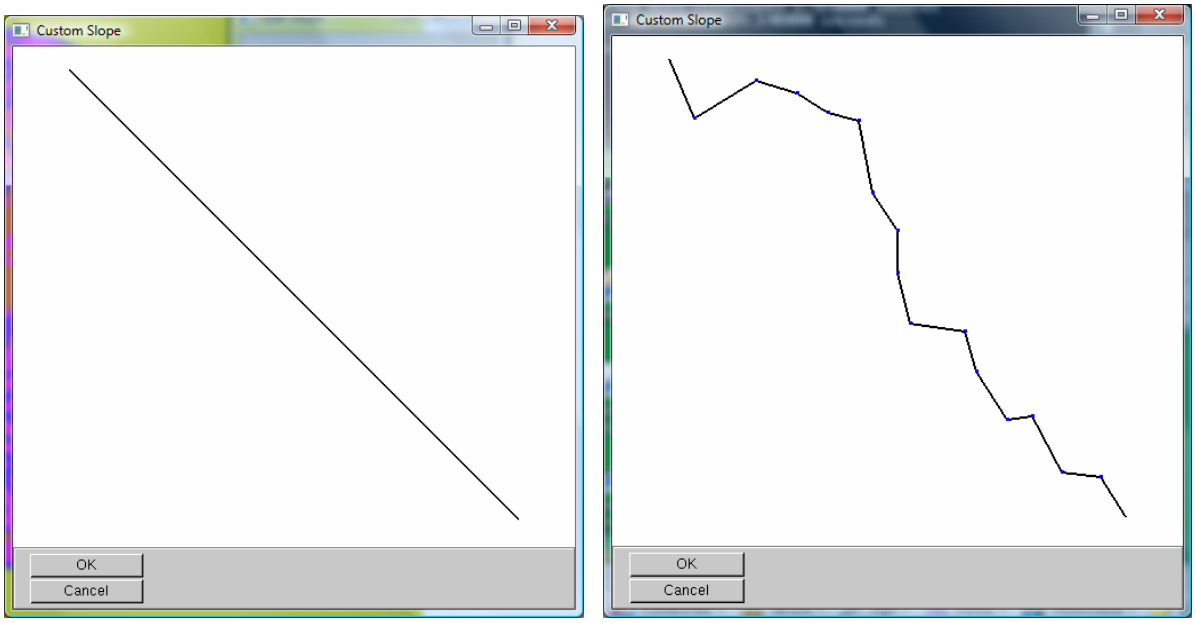

Figure 10: “Create Custom Slope” Interface

If the user finds that the predefined slopes do not meet their needs, they may choose to create a custom slope. The interface for this can be seen in Figure 10. This interface allows the user to manipulate the slope of a line segment with a 1:1 slope which represents the side of the ridge. This does not necessarily mean that the ridge selected had a 1:1 slope. The lower end of the line segment represents the edge of the river while the upper end represents the top of the ridge. The slope created will be stretched or compressed, horizontally and/or vertically, to fit onto the ridge. The picture on the left shows the interface initially with a linear slope. The picture on the right shows the interface after the user has created the slope with a rocky terrain. The user defines the custom slope by adding points to the slope by clicking the left mouse button. The user can then further adjust the slope by clicking on any point with the right mouse button. The clicked point can then moved, so long as it stays between the two adjacent points horizontally. The number of points in the custom slope is limited to 64.

After the user has selected the kind of slope they want, they can check the “Show user slopes” box in the “2D View Options” interface to see the path of cells going from the river to the medial axis cell which contains the cell that the user clicked on. This path contains the cells that will have the chosen slope with greatest influence (100%). This can be seen in Figure 11. The medial axis cells are shown in pink, while the user selected slope is defined to have maximum influence on the cells shown in green.

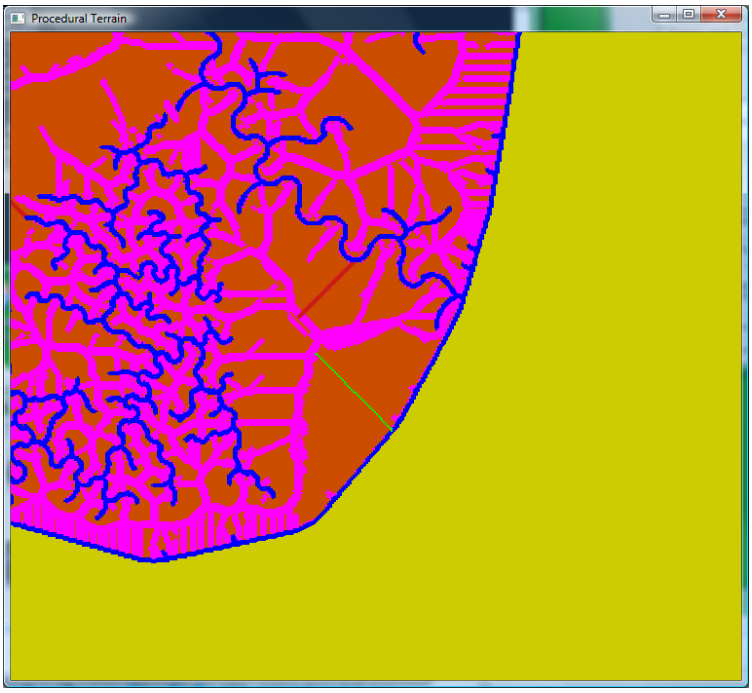

Figure 11: Center of user selected slope shown in green

2.3 Slope Influence

In this section, I will discuss what slope influence means and how it is calculated. When the user initially sets the slope influence in the interface shown in Figure 8, they are defining the absolute distance away from the path shown in Figure 11 where the slope of the cells will be influence by that slope. Beyond this distance, the slope will have no influence, so cells will have linear slopes unless they are influenced by another user-defined slope.

To determine the amount of influence a cell will receive from a user defined slope, a flood fill algorithm is used starting from the path connecting the top of the ridge to the river. The distance of influence (the same number specified by the user earlier) is recorded in each of the cells in this path. They are then added to a list of cells to be processed. The algorithm proceeds by first removing a cell from the list and exploring each of its adjacent neighbors. Each neighbor cell is recorded with a distance of influence 1 less than the current cell unless the neighbor cell has already been recorded with a higher distance of influence by another iteration of the algorithm. The neighbor cells that had their influence updated are then added to the end of the list to be processed. A neighbor cell is not added to the list if it is a medial axis cell, river cell, or coastal cell. This prevents a user defined slope from influencing the opposite side of the ridge. This would cause undesirable results.

Multiple slopes can affect the same cells when their influences overlap, so instead of associated each cell with a slope, each cell has a linked list of slopes associated with them in the terrain. When recording the amount of influence a cell should receive from a slope, the algorithm searches the list of slopes the cell has, if any, to see if the current slope exists. If it does, then the abovementioned check is performed to see if the influence already recorded is larger than the current influence. If the slope does not exist, the slope is added onto the linked list of cell.

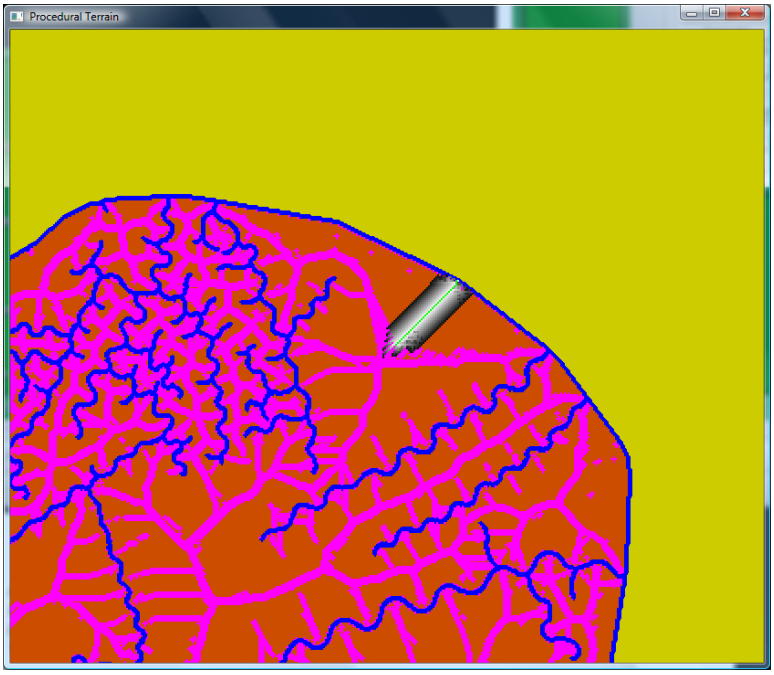

Figure 12: 2D view of slope's influence

Figure 12 shows the slope's influence decreasing as it gets farther away from the targeted path. It does not pass over medial axis cells (pink) or coastal cells (blue). The scale is from black (very little influence) to white (full influence).

2.4 Blending User Slopes with Existing Terrain

2.4.1 Overview

I will now discuss the techniques used to integrate the user slopes into the existing terrain. We start by explaining the method to blend a single user defined slope with the existing terrain which has all linear slopes. We will then discuss how the method changes when multiple user slopes are being considered. In this case, multiple defined slopes might overlap. We propose a solution that takes into account a slope's influence along with its shape to generate a new slope that seamlessly blends them together.

2.4.2 Blending with Linear Slopes

When there is only a single user defined slope, the slope must be blended with the existing terrain which has a linearly interpolated slope. To accomplish this, each cell which receives influence from the user defined slope calculates its height by dividing the maximum influence of the slope by the current influence the cell has. This result will yield a percentage, which is the weight the user defined slope has on the cell. For example, a user has defined a slope with a maximum influence of 120. The current cell is 96 units away from the defined slope, which gives it an influence value of 24. The height for this cell is calculated by using a weighted average of the user defined slope (24/120 = 20%) and a linear slope (1 – (24/120) = 80%). If the height would have been 100 with a linear slope and 50 with the user defined slope, the resulting height would be 100*0.80 + 50*0.20 = 90.

$$\text{Height} = (\text{SlopeInfluence}_{\text{current}} / \text{SlopeInfluence}_{\text{max}})*(\text{Height}_{\text{custom}}) + (1- (\text{SlopeInfluence}_{\text{current}} / \text{SlopeInfluence}_{\text{max}}))*(\text{Height}_{\text{linear}})$$

2.4.3 Blending With Multiple User Slopes

There will be times when a cell is under the influence of two or more user defined slopes. If this is the case, the height of the cell will be calculated as a weighted average of all the slopes that influence it as well as the underlying linear slope. For example, a cell is influenced by slopes S_1, S_2, S_3, and S_{Linear} where the maximum influences are 50, 70, 30, and 100 respectively, and the cell has influence values for these slopes of 25, 35, 10, and 100 respectively. The maximum influence and the current influence for the linear slope must always match because the linear slope expands throughout the entire terrain. To calculate the height, we find the weight for each slope and calculate a weighted average of all the slopes.

We first add up all the percentages as such: 25/50 + 35/70 + 10/30 +100/100 = 2.33. We then divide each percentage by this total, so weight of S_1 = 0.50/2.33 = 21.5%, weight of S_2 = 0.50/2.33 = 21.5%, weight of S_3 = 0.33/2.33 = 14.3%, and weight S_{Linear} = 1.0/2.33 = 42.9%.

$$TotalWeight = \Sigma \left((SlopeInfluence_{current}) / (SlopeInfluence_{max})\right)_i$$

$$IndividualWeight_i = ((SlopeInfluence_{current} / SlopeInfluence_{max})_i) / TotalWeight$$

$$Height = \Sigma\ IndividualWeight_i * (Height_{custom})_i$$

2.4.4 Results

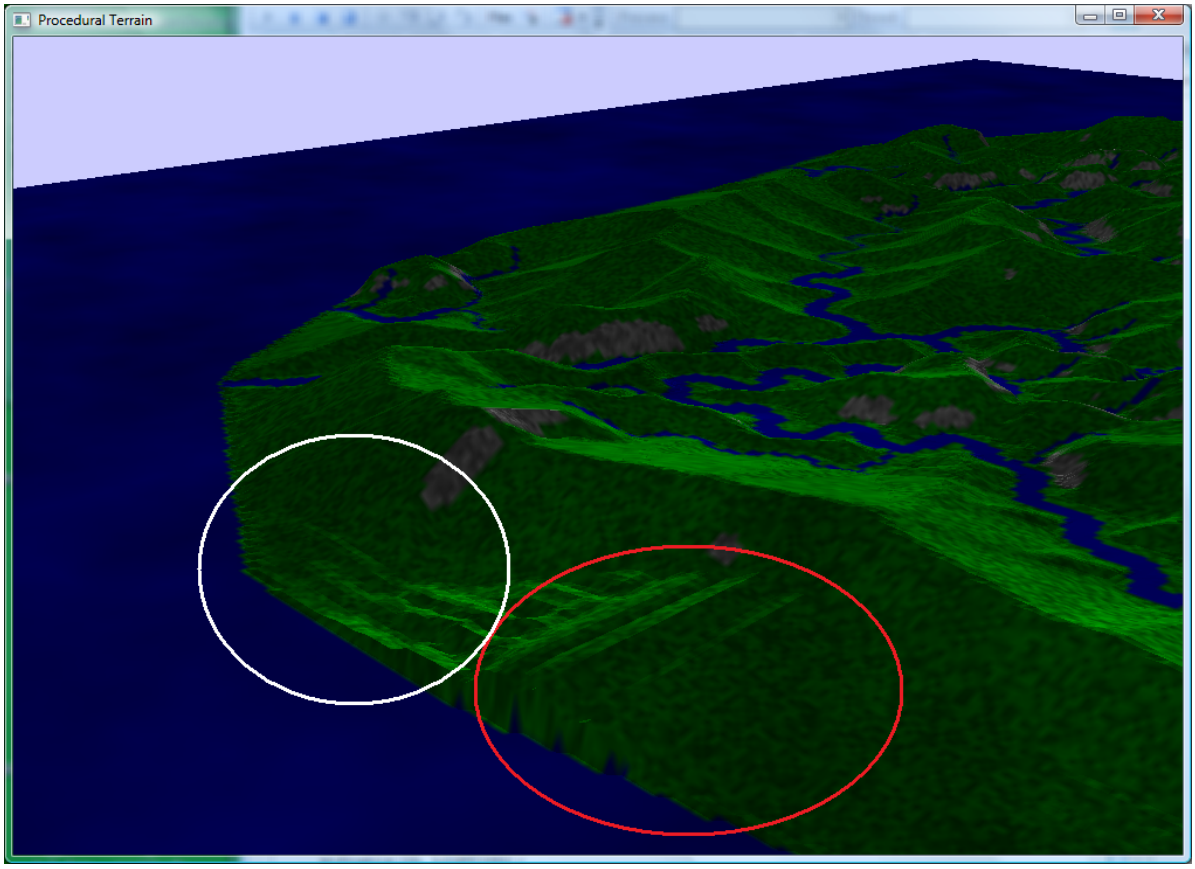

Figure 13. Blending of multiple user-defined slopes with existing terrain.

An example of the technique used is shown in Figure 13. The fractal overlay has been disabled to see the slopes more clearly. The area shown in the white circle has been defined with a slope which starts out flat and becomes steeper near the center of the ridge. The area shown in the red circle has been defined with a slope that rises quickly near the ocean and levels out into a plateau afterwards. It can be seen that both slopes have been blended into the existing terrain while smoothly transitioning into one another. All other ridges where no slope has been defined continue to have a linear slope.

This way of blending multiple user defined slopes will ensure that all the slopes will be smoothed together. There are no sudden changes into a different type of slope. This is important to the believability of the terrain. Users can easily spot even slight changes in the terrain if they are not done smoothly.

3 Pruning Undesirable Medial Axis Cells

3.1 Overview

As discussed before, there are many medial axis cells which cause problems when attempting to integrate user-defined slopes into the terrain. I will now discuss the method that was used to prune these medial axis cells. This method is experimental and does not completely remove all undesirable medial axis cells.

This method targets those medial axis cells that run perpendicular to the rivers and are close to them as well. These kinds of cells are shown in Figure 13. The circle on the far left shows an area that is full of medial axis cells.

This area would not be able to change its linear slope at all because there is no clear side to any one ridge. The circle on the right shows a single ridge that should also be eliminated. These kinds of ridges are not large enough to allow proper slope manipulation. They also interfere with the manipulation of slopes from larger ridges.

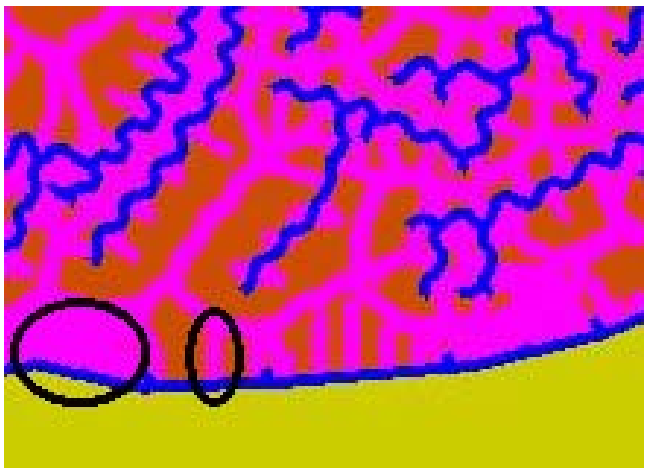

Figure 14: Medial axis cells targeted for removal

Each medial axis cell has a pointer that points back to a river cell in came from. This information is useful because this method attempts to remove a medial axis cell if all of its adjacent medial axis cells point back to a river cell that is close to the river cell that the candidate for pruning points back to. Figure 15 demonstrates this idea. The candidate for removal is marked with an 'X'. The arrows indicate which river cell the medial axis cells point back to. It can be seen that for these kinds of medial axis cells which are close, and sometimes even touching the rivers, all point back to nearby river cells. The distances measured in the diagram must all be above a relatively small threshold to keep the medial axis cell.

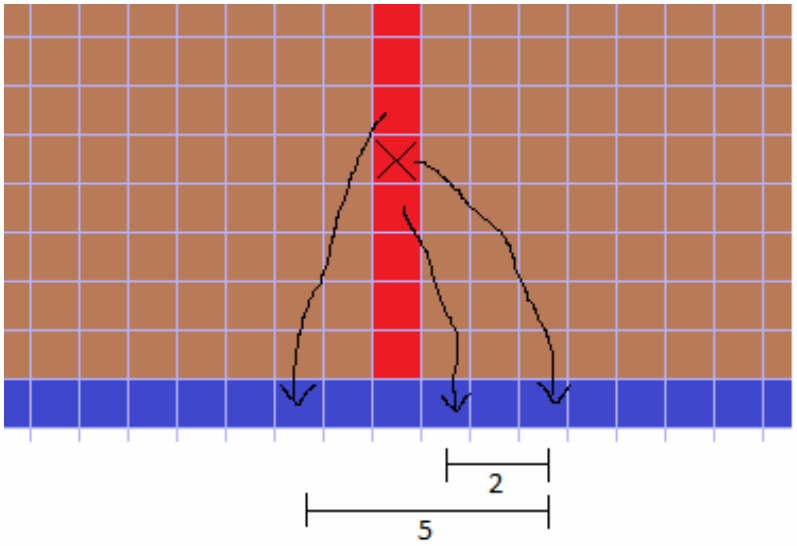

Figure 15: Pruning Method Clarification

3.2 Pruning Method

To accomplish this, we first create a 2-dimensional array of integers where each river cell is compared against every other river cell. The array is implemented as a list of lists to create a sparse matrix. A sparse matrix is needed as the number of river cells can go into the tens of thousands for a height map of 700 by 700 units. Only a small amount of table will have entries.

For each river cell, we traverse the river in both directions as well as following any tributaries in a depth-first search fashion. Once the algorithm has traversed the river for a suitable length, we make an entry into the sparse matrix for each river cell that was traversed, recording the distance that was traversed to reach the cell. This information will be useful for easy lookup when deciding to remove a medial axis cell.

Once this array has been constructed, we can then search for medial axis cells to prune. All the medial axis cells are put into a list that is sorted by their shortest distance to a river. They are sorted in this fashion so that medial axis cells do not become isolated. In Figure 15, the cells closest to the river would each be removed in order. For each of the medial axis cells in the list, we look at the river cells that the adjacent cells point to. As mentioned above, if all these river cells are nearby the river cell the medial axis cell points to, we can remove it. We can look up these distances using the table that was constructed in the previous step. If the entry in the table is less than a certain threshold, the medial axis cell can be pruned. We use a threshold of 20 in this project. The medial axis cell must then be converted to a normal cell. If this is not true for any of the neighbor cells, then the medial axis is not pruned.

3.3 Converting Medial Axis Cells

There are many things to consider when deciding to prune a medial axis cell. Since non-medial axis cells must point to a neighboring cell that leads towards a medial axis cell, the pruned cell will be made to point to a neighboring medial axis cell. It must also be given another pointer that points to this neighbor because all non-medial axis cells have a pointer to a medial axis cell.

The final step is to fix all the pointers for non-medial axis, which includes the recently pruned cells. For each non-medial axis cell in the terrain, if the medial axis cell it points to has been pruned, the pruned medial axis cell must now be pointing to another medial axis cell (although this cell could also have been pruned). The algorithm follows this path of pointers until it finds a medial axis cell that has not been pruned and sets this cell as the end of the path.

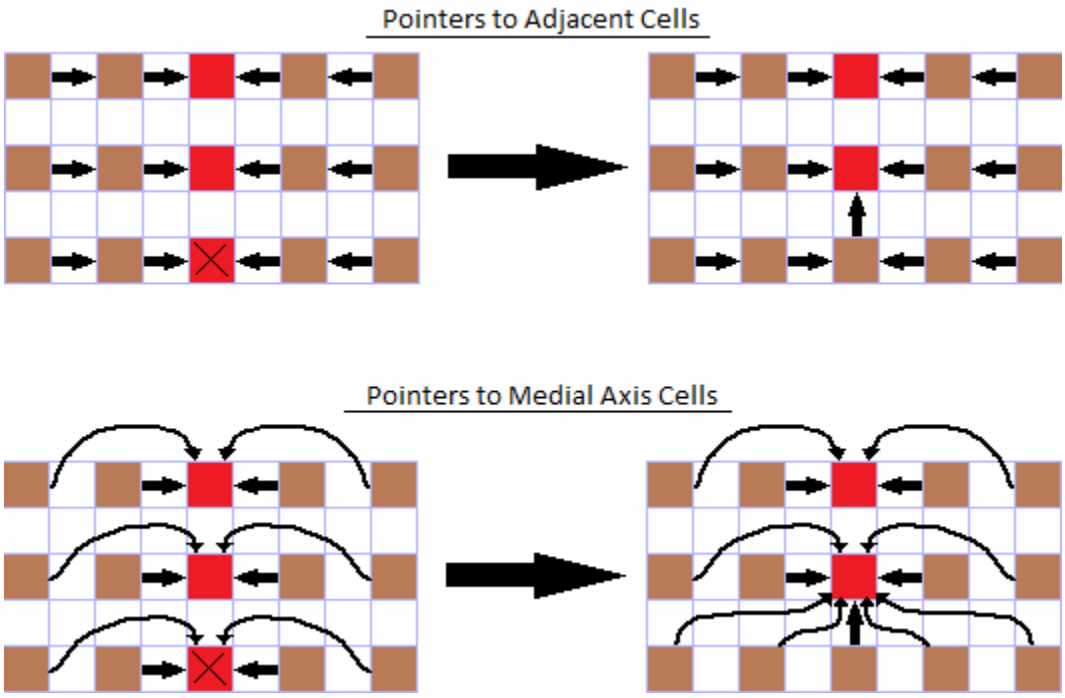

Figure 16: Changes in Pointers when converting medial axis cells to normal land cells. Brown cells represent normal land cells. Red cells represent medial axis cells, with the 'X' representing the medial axis cell that will be removed. Space has been left between cells for clarity.

Figure 16 illustrates the changes that are made to cells in the terrain. In the upper image, the pointers that make a path to the medial axis cells are shown. The medial axis cell that has been removed has been given a new pointer to its neighboring medial axis cell. In the lower image, the pointers directly to the medial axis cells are shown. When the marked medial axis cell is removed, the land cells that were pointing to it are changed to point to the neighboring medial axis cell.

3.4 Results

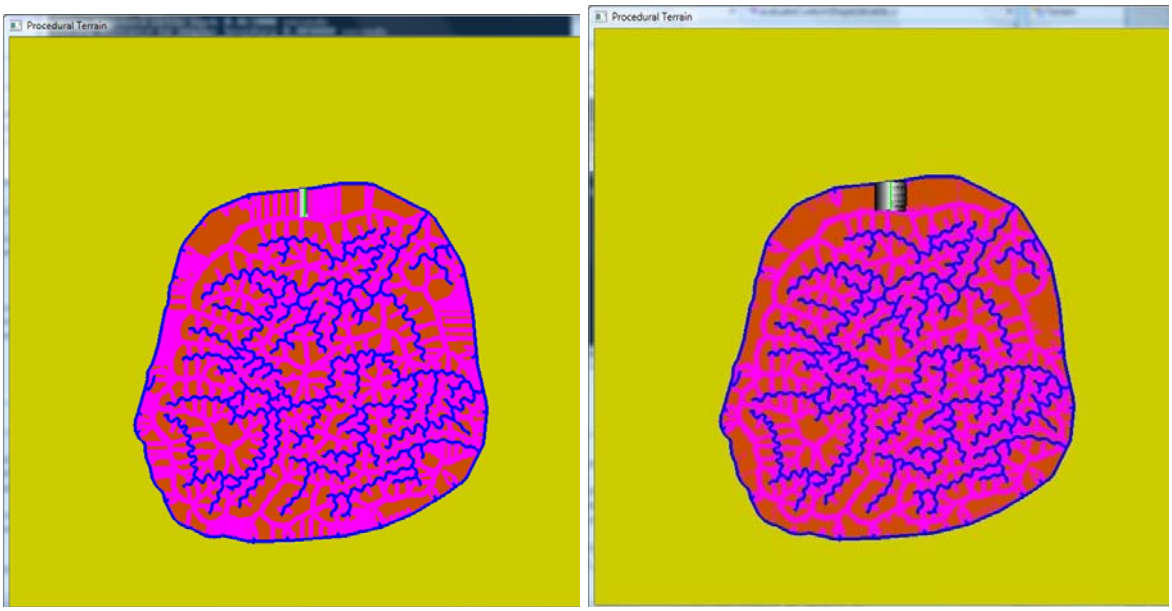

Figure 17: Comparison of No Pruning (Right) vs. Pruning (Left) Showing Influence of User-defined Slope

It was found that the method used was effective at removing many of the problematic medial axis cells. Figure 17 shows the comparison between the terrain without pruning, on the left, and the terrain with pruning, on the right. The influence of the user defined slope can be seen near the top of the terrain. It is clearly visible that when pruning is done, the slope's influence affects more of the cells that the user intended to alter. The influence can become "boxed in" by medial axis cells when no pruning is done as seen in the left image. Many medial axis cells were pruned, but many still remain. More specialized pruning techniques would be necessary to eliminate these cells.

3.3 Performance Concerns

The method used here was effective and efficient at removing medial axis cells. The amount of time to prune is O(n) as the number of river cells increases. This is true because even though the dimensional size of the sparse matrix for river cells increases by n^2, each river cell is only compared against a limited number of cells, not every other river cell.

	Run #1 (seconds)	Run #2 (seconds)	Run #2 (seconds)	Avg. Runtime (seconds)
350 x 350 height map	1.203s	1.194s	1.218s	1.205s
700 x 700 height map	3.822s	3.955s	3.875s	3.844s

Figure 18: Runtime of pruning on different size height maps.

Figure 18 shows the results of running RiverLand 2.0 with pruning on different size height maps. When using a height map of 350 by 350 cells, pruning took an average of 1.205 seconds to complete. When using a height map of 700 by 700 cells, which is 4 times as many cells, pruning took an average of 3.844 seconds to complete. When increasing the number of cells by 4, the runtime increased by a factor 3.19. This confirms that pruning medial axis cells has a runtime of no more than O(n).

4 Non-periodic Texturization

4.1 Introduction

In order to bring 3D objects and environments closer to the real word, they must have realistic looking textures. Textures can take something that looks plain and boring and bring it to life, giving it much needed depth and detail. For objects that are relatively small, unique textures can be made for each surface without spending a great deal of time. For larger objects however, textures are made to repeat over surfaces. These texture patterns are not so apparent when used on things like buildings, but they are easily noticeable when used on a terrain because repeated patterns are not always completely continuous. When users explore the terrain, they can forget that they are just looking at a virtual world.

But when they begin to see patterns in the textures, they immediately become aware of the fact that what they are looking at is just a simulation. The goal is to create a method that uses a few textures to create a continuous landscape without any repeating patterns.

4.2 Previous Work by Cani and Neyret

M. Cani and F. Neyret have proposed a method of applying textures as equilateral triangles instead of square patches. They create the textures in such a way that they can be randomly arranged while still being continuous. The edges of these textures are made so that there are only a few different edges. Figure 19 shows an image from their work depicting a single continuous edge and the four resulting textures that would need to be created. When applying the textures to an object, the algorithm randomly picks a texture from the ones that satisfy the boundaries conditions.

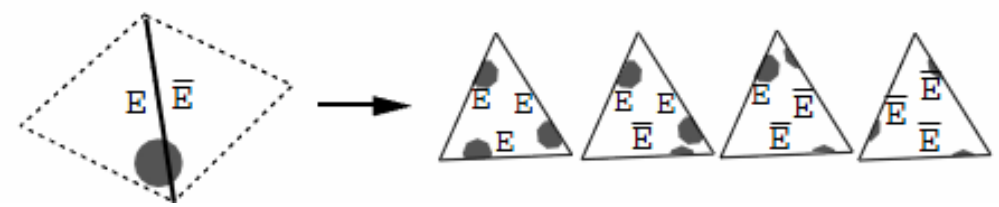

Figure 19: An oriented edge, and the set of four texture samples that need to be created to fit the different boundary conditions it produces [1].

This idea solves the problem nicely because it allows for random placement of textures while keeping the entire surface continuous. The difference here is that terrains usually involve height maps in a 2D array. Because of this, textures are applied as right triangles instead of equilateral triangles. This does not allow for the same kind of flexibility for arranging the tiles.

4.3 Texturization Method

In order to apply textures randomly to the terrain using right triangles, we need to develop a new set of textures that can be used to fit different boundary conditions. We propose to have an oriented edge pair for each side of a right triangle instead of having a single edge pair for all sides. Figure 20 illustrates the oriented edges created for each sides where A, B, and C match with A', B', and C' respectively.

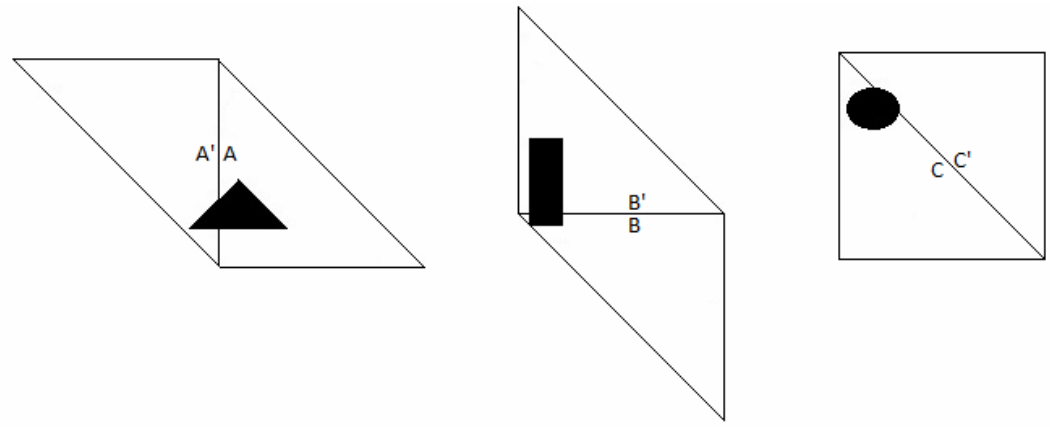

Figure 20: Oriented edges for each side of the right triangle

In this demonstration, we have 2 possible edges for each side and 3 sides, so we will need 2*2*2 = 8 different textures triangles to satisfy all possible boundary conditions. Figure 21 illustrates the 8 texture triangles needed when using the edge pairs (A, A'), (B, B'), and (C, C').

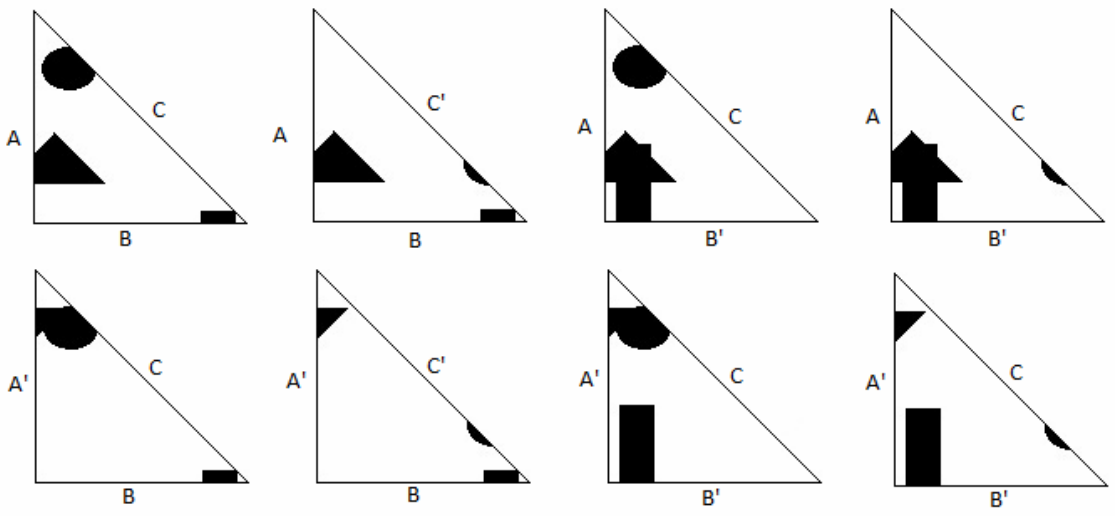

Figure 21: The set of 8 textures that is needed to satisfy all boundary conditions when using a one edge per side.

To apply these textures to our terrain, we proceed through each side of every triangle to be textured and decide which edge to assign to it. If the neighboring triangle has assigned an edge to the shared side, that side will be assigned the corresponding edge (B' for B, for example) for that triangle. If no edge has been assigned, an edge is chosen at random where each edge has a 50% chance of being chosen. Once every side has been assigned an edge, a texture can be chosen that fits with the three edges.

4.4 Results

We have applied this method of texturizing to RiverLand to create a landscape with a continuous surface while maintaining a non-period pattern, as can be seen in Figure 22. Although the result does not resemble an actual terrain, it does illustrate the results of implementing random texture placement.

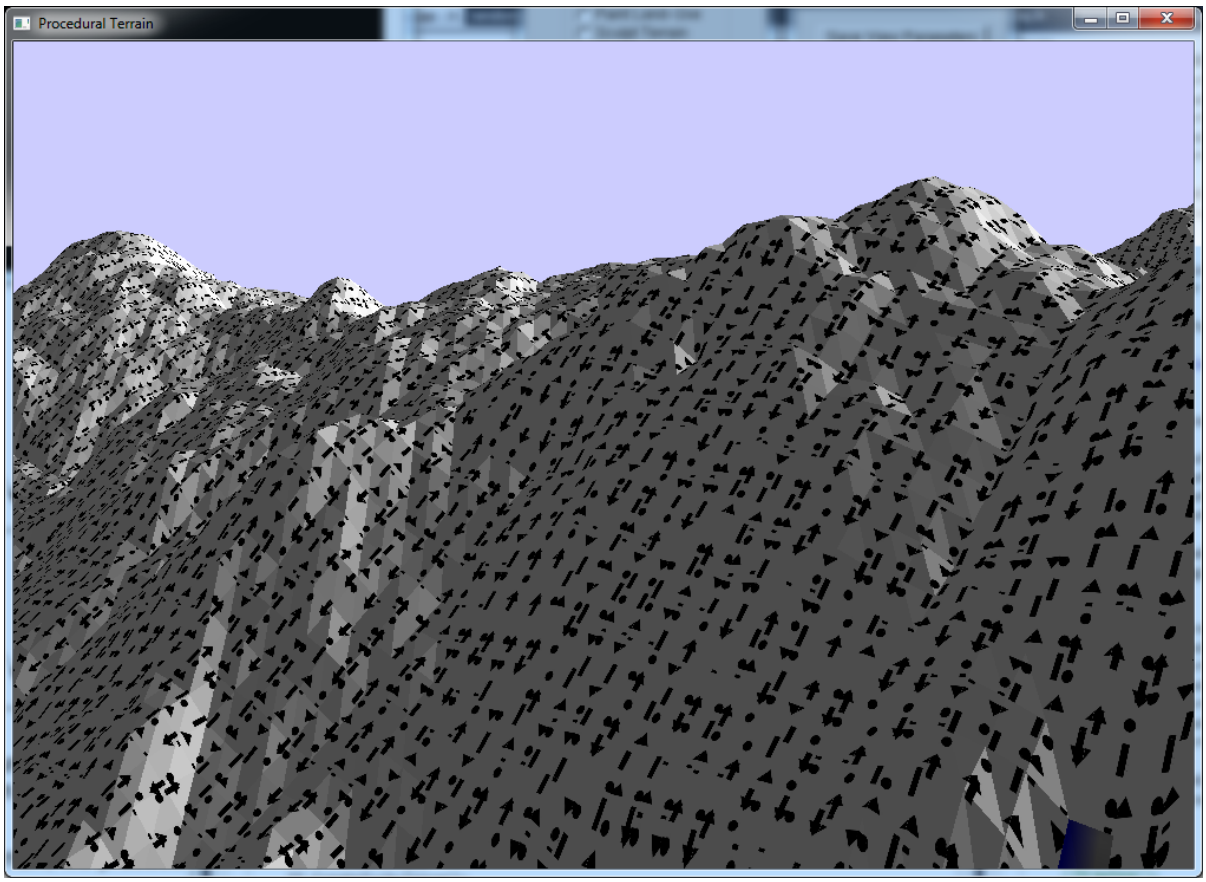

Figure 22: Non-period texturization of RiverLand using the 8 texture triangles mentioned.

5 Open Problems and Future Work

In this project, we used a method for flooding a user defined slope across the terrain starting with a line from the river to the ridge line. In the future we would like to be able to specify any piecewise linear curve along the terrain and have the user defined slope flood away from this curve. The algorithms used in the original program allowed the slopes to be easily manipulated from the ridge to the river. In order to manipulate slopes across any direction on the terrain, the original algorithms would need to be revised.

The method used in this paper to prune medial axis cells was efficient on large terrains, but it did not remove all the undesirable medial axis cells from the terrain. In the future, multiple techniques will need to be used to remove more of these cells.

RiverLand 2.0's non-periodic texturization is still lacking in the fact that the right triangles are all facing the same way which would allow users to spot patterns more easily. If the triangles were allowed to placed in a different orientation, it would allow the texturization to be even less periodic as the triangles would be randomly oriented as well as randomly textured. Figure 23 is an illustration of this idea. This way of positioning the triangles greatly increases the number of textures that will need to be made in order to deal with all the restrictions on which triangles can fit into the mesh.

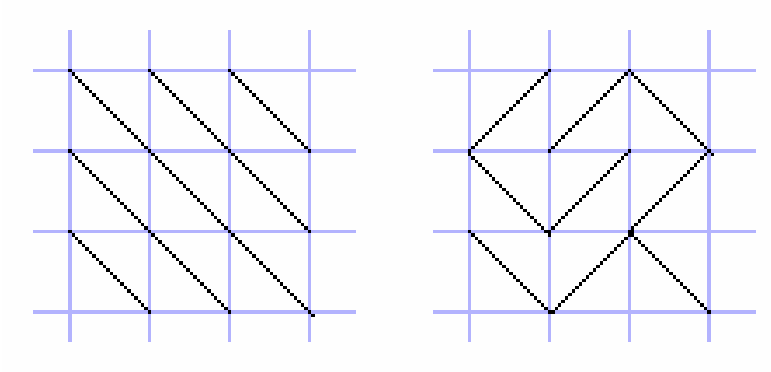

Figure 23: Right: Standard tiling of height map; Left: Same height map with random orientation of triangles

The texturization of the terrain of RiverLand could not done as described Cani and Neyret because of the nature of the data set that was constructed by the original algorithm. If the original algorithm of RiverLand had created a height map in an equilateral or even hexagonal fashion, the terrain could be textured using equilateral triangles. In this way, no extra work would have to be done to apply Cani and Neyret's method to RiverLand.

6 Conclusion

In this project, we have presented a method of allowing users to gain more control over the terrain using RiverLand. While the original RiverLand offered users the ability to create their own island and specify ridges, RiverLand 2.0 lets users draw their own slopes to shape the ridges to their liking. Users are able to determine if they want the slope to consume the entire ridge or if they would rather have several smaller localized slope types.

RiverLand 2.0 eliminated many of the undesirable medial axis cells so that users could use the new slope feature to a further extent. This greatly increased the usability of the slope feature as it removed many of the “fake” ridges which were restricting the user-defined slopes from taking hold in the landscape. Although many were eliminated, unwanted medial axis cells still remain which can still cause problems when trying to refine the terrain. RiverLand 2.0 also dealt with texturing the landscape with a few simple textures to get a texturization that did not have a definite pattern. The method provided by Cani and Neyret worked nicely for equilateral triangles but provided more restrictions when adapting it to height maps which for the most part use right triangles because of the grid nature of the raw data.

References

1. Audiber, P. and Belhadj, F. Modeling Landscapes with Ridges and Rivers. Proceedings of the ACM Symposium on Virtual Reality Software and Technology (VSRT 2005). 2005.

2. Cani, M. and Neyret, F. Pattern-Based Texturing Revisited. Proceedings of the 26th annual conference on Computer graphics and interactive techniques (SIGGRAPH '99), 1999. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.42.6343&rep=rep1&type=pdf

3. Kamal, K. R. and Uddin, Y. S. Parametrically controlled terrain generation. Proceedings of the 5th international conference on Computer graphics and interactive techniques in Austrail and Southeast Asia (GRAPHITE '07), 2007. doi: 10.1145/1321261.1321264

4. Kilgard, M. J. The OpenGL Utility ToolKit (GLUT) Programming Interface API Version 3. URL: http://www.opengl.org/resources/libraries/glut/spec3/spec3.html

5. Mountain Picture, URL: http://www.tripadvisor.com/LocationPhotos-g303883-Nashik_Maharashtra.html

6. OpenGL 2.1 Reference Pages. URL: http://www.opengl.org/sdk/docs/man/

7. Persson, M. 2011, March. Terrain Generation, Part 1. URL: http://notch.tumblr.com/post/3746989361/terrain-generation-part-1

8. Teoh, S. T., 2007. Autopolis: Allowing User Influences in the Automatic Creation of Realistic Cities. Advances in Visual Computing: Proc. of the 3rd International Symposium on Visual Computing (ISVC'07), Nov 2007. URL : http://cs.sjsu.edu/~teoh/research/papers/isvc07.pdf

9. Teoh, S. T., 2009. RiverLand: An Efficient Procedural Modeling System for Creating Realistic-Looking Terrains. URL : http://cs.sjsu.edu/~teoh/research/presentations/isvc2009.ppt