Terra: A 3D Terrain Generator and Visualizer

Richard Christiensen E. Aluning
Institute of Computer Science
College of Arts and Sciences
University of the Philippines Los Baños
College 4031, Laguna, Philippines
rcaluning@yahoo.com

Joseph Anthony C. Hermocilla
Institute of Computer Science
College of Arts and Sciences
University of the Philippines Los Baños
College 4031, Laguna, Philippines
jachermocilla@uplb.edu.ph

ABSTRACT
Artificial terrains are widely used in animations, computer games, and simulations. This study investigates the characteristics of terrains generated by three algorithms namely Fault Formation, Midpoint Displacement, and Perlin Noise. Implementation of these algorithms in Terra allows users to visualize and navigate over the generated terrain in 3D.

1. INTRODUCTION
Terrain is an area of land. Examples are rocky mountains, grass plains, rolling hills, all combining to form a beautiful landscape. Terrains are used in diverse fields, not just because of their aesthetic value, but also for scientific and entertainment applications.

Modern computer games, such as flight simulators and role-playing type games, involve players navigating over mountains and islands. These types of terrains add realism to the gaming experience of players.

Scientists studying hydrological processes use terrains to test flow routing algorithms. Different types of terrain can affect the behaviour of flows. Flood simulations use terrains to determine areas where floods are most likely to occur.

Terrain data can be obtained by manual surveying of actual land formations or remote sensing. However, this approach is tedious and not cost-effective. An alternative is to use computer algorithms to generate artificial terrains for use in applications described above.

This study investigates the characteristics of terrains generated by Midpoint Displacement, Fault Formation, and Perlin Noise. These algorithms are implemented in Terra. Users can generate terrains by selecting a desired algorithm. Generated terrains can be visualized and navigated in 3D.

The succeeding sections discuss the details of the algorithms and the actual result of the implementation in Terra. Sample terrains for each algorithm were generated and evaluated.

2. THEORETICAL FRAMEWORK
2.1 Heightmaps
A heightmap is used to store elevation data. It is normally represented as a grayscale image file or a two-dimensional array of values. Algorithms for generating artificial terrains populate a heightmap with height or elevation values.

2.2 Midpoint Displacement
Midpoint Displacement[2], also known as Plasma Fractal or Diamond Square Algorithm, is a fractal technique which exhibits self-similarity and is recursive. A heightmap's corners are named points A, B, C and D. The midpoint of segments AB(1), BD(2), DC(3) and CA(4) are first calculated. The midpoints (1,3) and (2,4) are then connected to obtain the intersection point E. To calculate the height of E, the height values of A, B, C, and D are averaged and a $2^{-roughness}$ coefficient is added. Each quadrant is then processed recursively using the same principle. Figure 1 illustrates how Midpoint Displacement works.

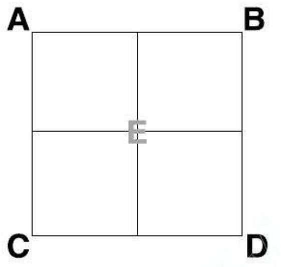


Figure 1: Initial configuration of a heightmap when using Midpoint Displacement.

2.3 Fault Formation
Fault Formation[7] generates faults in a terrain. A random line is added to an empty heightmap then a random height is assigned to a side. The process is repeated until the desired terrain is achieved. The random height is linearly decreased and must eventually reach zero. Figure 2 shows the first step of Fault Formation on an initially empty heightmap.

2.4 Perlin Noise
Perlin Noise adds up noise functions at a range of different scales. It is calculated using n dimension. A noise function

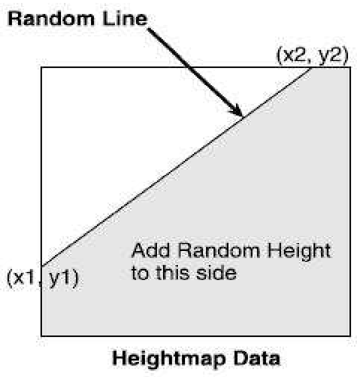


Figure 2: Initial configuration of a heightmap when using Fault Formation.

and an interpolation function are needed to generate the height values. The noise function generates random values and the interpolation function makes the values closer to each other. The algorithm is derived from the idea of using waves to generate a terrain. Figure 3 is an example wave function used in Perlin Noise.

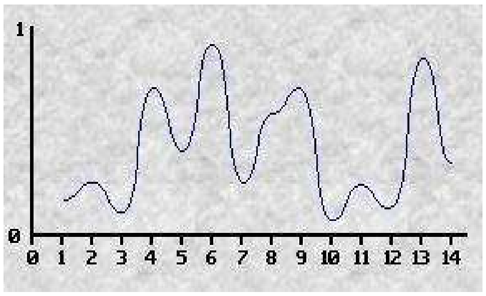


Figure 3: A wave function used in Perlin Noise.

3. SOFTWARE DEPENDENCIES

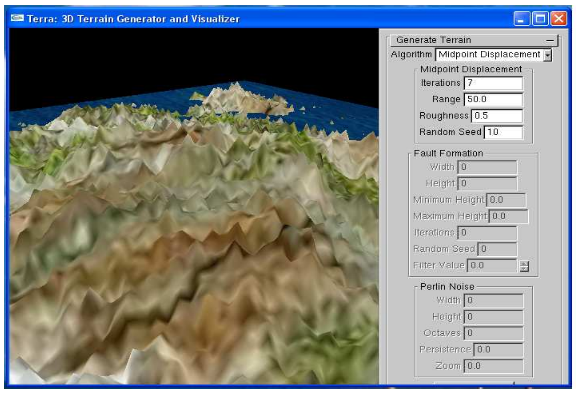
The following are the third-party libraries used in implementing the algorithms in Terra:

- Open Graphics Library (OpenGL)[9][1]
- Graphics Library Utility Toolkit(GLUT)
- Graphics Library Utility Interface (GLUI)
- Simple DirectMedia Layer (SDL)[6]
- Microsoft Visual C++ 2008 Express Edition

4. RESULTS AND DISCUSSION

Figure 4 shows the user interface of Terra. The user can choose the algorithm to generate the artificial terrain. Each algorithm has several parameters that users can supply. These parameters will affect the visual quality of the terrain. Desired terrain can be generated by assigning appropriate values to the selected algorithm's parameters.

Midpoint Displacement parameters include *number of iterations, height range, terrain roughness* and *random seed.*



Algorithm: Midpoint Displacement

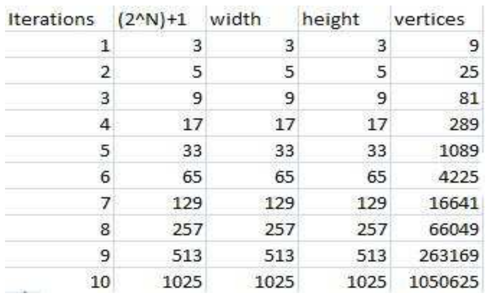
Midpoint Displacement
Iterations: 7
Range: 50.0
Roughness: 0.5
Random Seed: 10

Fault Formation
Width: 0
Height: 0
Minimum Height: 0.0
Maximum Height: 0.0
Iterations: 0
Random Seed: 0
Filter Value: 0.0

Perlin Noise
Width: 0
Height: 0
Octaves: 0
Persistence: 0.0
Zoom: 0.0

Figure 4: User interface of Terra.

The number of iterations computes the number of vertices of the terrain. It can be computed as $(2^N) + 1$ where N is the number of iterations. Because of this property, Midpoint Displacement always has the same width and height. Figure 5 shows the growth of the number of vertices per iteration. The height range parameter displaces the terrain but is decreased per iteration. The random seed configures the random number generator's output.



Iterations	(2^N)+1	width	height	vertices
1	3	3	3	9
2	5	5	5	25
3	9	9	9	81
4	17	17	17	289
5	33	33	33	1089
6	65	65	65	4225
7	129	129	129	16641
8	257	257	257	66049
9	513	513	513	263169
10	1025	1025	1025	1050625

Figure 5: Vertex count per iteration increase in Midpoint Displacement.

Fault Formation parameters include *width, height, minimum displacement, maximum displacement, number of times of displacement, random seed* and *filter value.* The width and height parameters configure the size of the grid. The minimum and maximum displacement are used to calculate the depression and elevation of the terrain respectively. Every time the terrain is raised, it must be decreased linearly. The formula for displacement is given below.

$$disp = maxDisp + (\frac{iterationsDone}{itMinDisp})(minDisp - maxDisp)$$

maxDisp and *minDisp* refers to the maximum and minimum displacement, *iterationsDone* is the current iteration and *itMinDisp* is the minimum iterations to displace the terrain. If *itMinDisp* is equal to 100, then *displacement* will be equal to the minimum displacement.

The random seed configures the random number generator's output. All the parameters except for filter value are in the standard implementation of Fault Formation. In this study, the *Finite Impulse Response (FIR)* filter was used to simulate erosion. FIR filter erodes the sides of the elevated regions to smoothen the heightmap.

Perlin Noise parameters are *width, height, octaves, persistence and zoom.* Width and height represents the dimension of the terrain. Octaves refer to the number of generated noise to be added. Persistence is the terrain roughness and zoom is the size of the terrain features. The octaves parameter sums up the generated noise and takes its average. Persistence is the amplitude of the terrain defined as $amplitude = persistence^i$ where i is the ith function to be added and computes the elevated areas in the terrain. Zoom is the size of the features of the terrain. A high value of zoom generates a terrain with large features.

4.1 Terrain Visualization

The terrain generation algorithm creates a heightmap which serves as input to the visualization routine. Each column and row of the heightmap becomes the x and y coordinates and the height value the z coordinate, forming a single vertex (x, y, z) in 3D. OpenGL is used in this stage to render the terrain. The primitive GL_QUADS is used to render a tile which consists of four vertices. After rendering all individual tiles the terrain mesh or wire frame is formed as shown in the Figure 6.

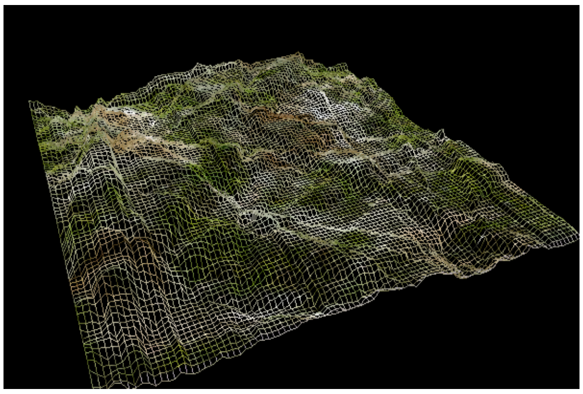


Figure 6: Terrain mesh created from a heightmap.

4.2 Texture Mapping

To add realism to the visualization of the terrain, the wire frame model is texture-mapped. SDL is used to assign the texture images to the tiles. A texture-mapped view of Figure 6 is shown in Figure 7.

4.3 Water Level

Terra also provides an option for rendering water over the terrain, especially when visualizing islands. Water level information however is not included when the heightmap is saved. It is only a parameter used in the visualization that can be set when viewing the terrain.

4.4 Saving Generated Terrain

The generated terrain can be saved as grayscale image file, specifically a .BMP file. The intensity of the pixels represent the height values. Terra can also load heightmaps created from other programs as long as they are in the .BMP format.

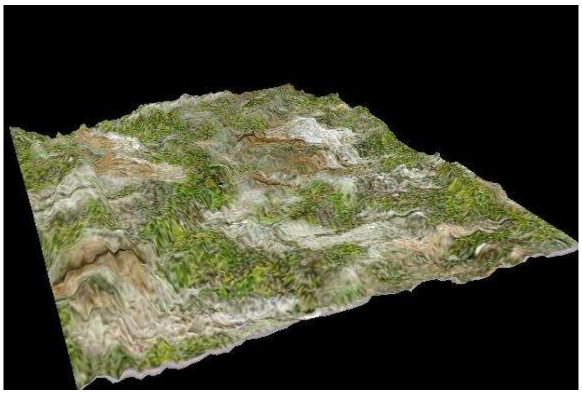


Figure 7: Texture-mapped terrain of Figure 6.

4.5 Evaluation of the Algorithms

The evaluation of the effectiveness of the three algorithms in generating artificial terrains is based on the visual inspection of the generated terrain. Appropriate parameters of an algorithm are supplied to approximate the appearance of a target terrain. In this study, the target terrains are plains, mountains, and islands. These terrains were chosen because they show the extreme cases on which a certain algorithm can simulate. Plains must be realistic enough such that that it is not purely flat but with bumps and low elevations. Mountains, on the other hand, should appear to be realistic in terms of height and roughness. Lastly, islands should simulate a chunk or chunks of a coastal terrain.

4.5.1 Mountains

Figures 8-10 shows the texture-mapped generated terrains using the three algorithms to approximate a mountain. Midpoint displacement has a peak and has a rough surface. Fault Formation literally generates faults that looks like stairs. Perlin Noise rendered a smooth yet steep mountain.

4.5.2 Plains

Figures 11-13 shows the texture-mapped generated terrains using the three algorithms to approximate a plain. The results of the plains generated are interesting. Midpoint Displacement generated a plain that suggests a rocky surface. Fault Formation shows cracks that looks like eroded soil. Perlin Noise failed to simulate a plain effectively.

4.5.3 Islands

Figures 14-16 shows the texture-mapped generated terrains using the three algorithms to approximate an island. The islands are mountains rendered with water. Midpoint Displacement shows a rocky shore. Fault Formation shows cliffs. Perlin Noise shows mountains in an island.

4.6 Viewer Feedback

To further evaluate effectiveness of the algorithms, a survey was conducted to 97 respondents. They were asked to rank (1 to 3) the terrains (Figures 8-10,11-13,14-16) based on the degree of resemblance to a certain type of terrain. A rank of 1 exhibits the most resemblance based on the user's evaluation. The respondents were not informed of the algorithms used to generate the terrain. The ranks for each terrain type are summed. Table 1 summarizes the results.

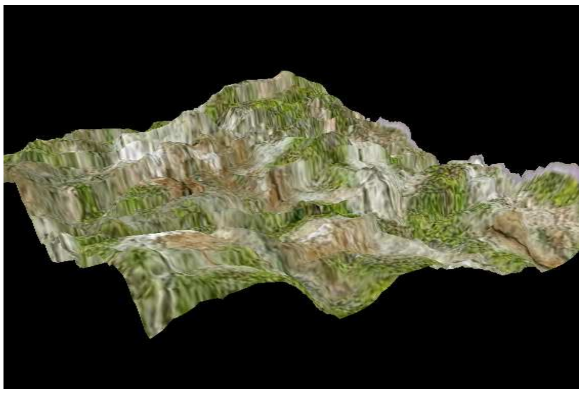


Figure 8: Mountain generated using Fault Formation.

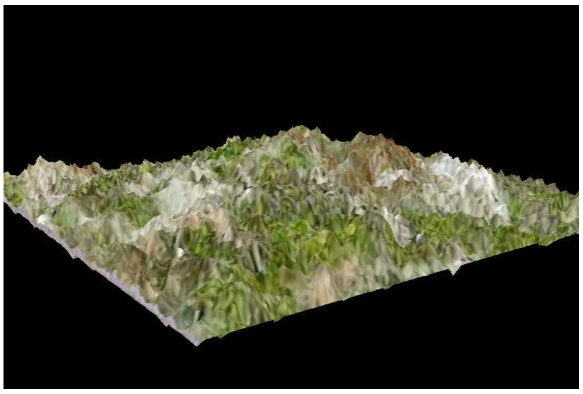


Figure 9: Mountain generated using Midpoint Displacement.

For mountains, the terrain generated by Fault Formation was ranked highest by the respondents. On the other hand, terrains generated by Midpoint Displacement for plains and islands were ranked highest. Perlin Noise generated terrains ranked low in all the types.

5. CONCLUSION

In general, the three algorithms can generate approximations of the target terrains. Visual quality, however, is still the basis of how good a particular terrain is. Midpoint Displacement is effective in rendering rough mountains. It is the only algorithm, compared to the other two, that can make mountain peaks. Smoothness is difficult to achieve using Midpoint Displacement. Fault Formation generates visually appealing terrains. Without filtering, stair-like terrains result. Perlin Noise is good in producing wave-like terrains. Increasing the zoom parameter renders a smooth mountain. It performs poorly when generating plains. Results of viewer feedback showed that Fault Formation is best for generating mountains, and Midpoint Displacement for plains and islands.

6. RELATED WORK

There have been several works in the literature about artificial terrain generation. Majority of which focus on developing new methods or improving existing algorithms. Others

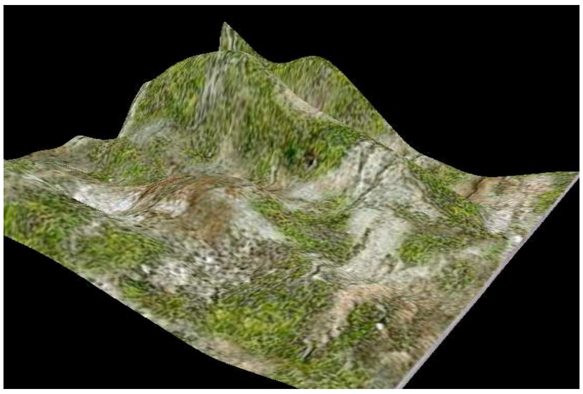


Figure 10: Mountain generated using Perlin Noise.

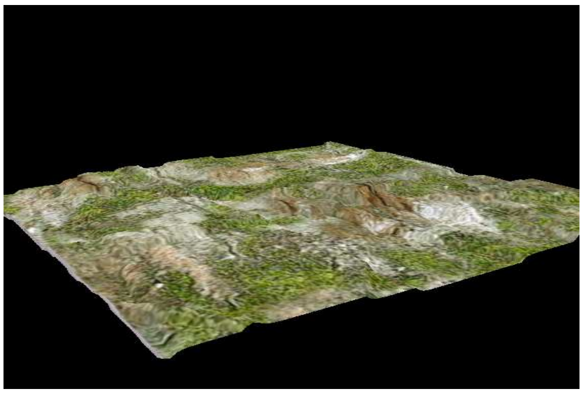


Figure 11: Plain generated using Fault Formation.

work towards applications of artificial terrains in real-world projects.

Using Midpoint Displacement, Olsen [5] focused on the effects of erosion in terrain. He presented ways to erode a terrain by thermal and hydraulic erosion.

Koh [4] also used heightmaps implemented cellular automata in his thesis. He was interested in stream erosion and its effects in the rendered terrain. He considered three types of landscape phenomena is his work which are stream erosion on firm terrain, desert dune formation and transport and accumulation of fallen snow.

Saunders [8] developed a system for integrating several height fields to make new terrain. He used Genetic Algorithms to combine different terrain features. This makes the final terrain look more realistic because of the diversity in land formations.

There are also existing applications for three dimensional terrain generation. Terragen Classic[3], from Planetside, is a scenery generator, created with the goal of generating photo-realistic landscape images and animations. Terragen 2 is the improved version of Terragen 1 with the goal of creating entire worlds from imagination or importing real world terrain datasets to have the most realistic visualizations possible. User's can control weather, landscape, rivers,

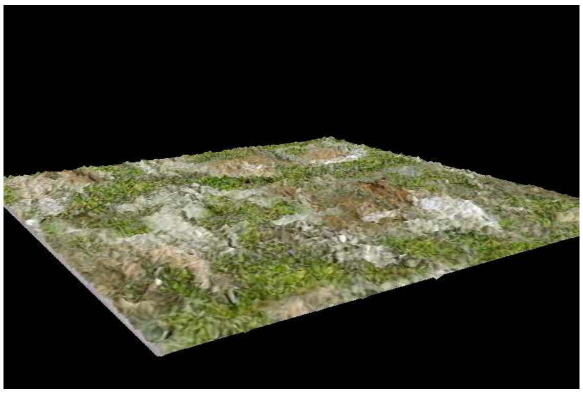


Figure 12: Plain generated using Midpoint Displacement.

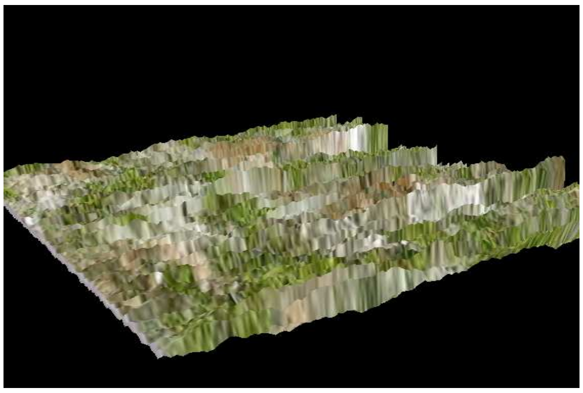


Figure 13: Plain generated using Perlin Noise.

lakes and oceans, suns, moons, and stars.

7. REFERENCES

[1] D. Astle and K. Hawkins. *Beginning OpenGL Game Programming*. Premier Press, 2004.

[2] A. Fournier, D. Fussell, and L. Carpenter. Computer rendering of stochastic models. *Commun. ACM*, 25(6):371–384, 1982.

[3] F. Inc. Planetside - home. `http://www.planetside.co.uk/`, November 2009.

[4] A. Koh. Dramatic landscapes: Cellular automata modeling of landscape phenomena. Technical report, School of Computer Science and Software Engineering, Monash University, 2004.

[5] J. Olsen. Real-time procedural terrain generation. Technical report, Department of Mathematics and Computer Science, University of South Denmark, October 2004.

[6] E. Pazera. *Focus on SDL*. Premier Press, 2002.

[7] T. Polack and W. H. d. Boer. *Focus on 3D Terrain Programming*. Premier Press, 2002.

[8] R. L. Saunders. Terrainosaurus: Realistic terrain synthesis using genetic algorithm. Technical report, Office of the Graduate, Studies of Texas AM University, 2006.

[9] R. S. Wright and B. Lipchak. *OpenGL SuperBible (3rd Edition)*. Sams, Indianapolis, IN, USA, 2004.

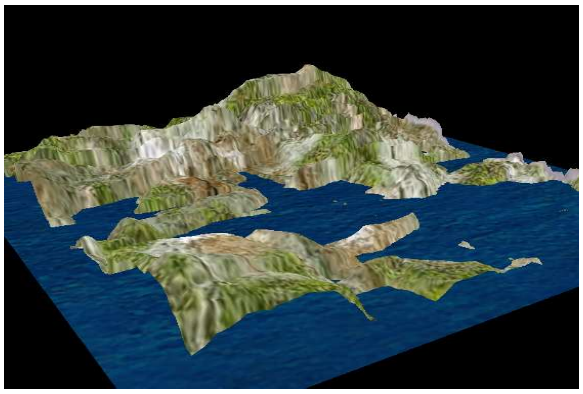


Figure 14: Island generated using Fault Formation.

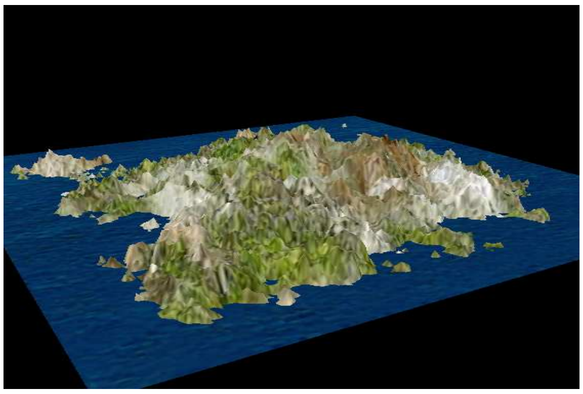


Figure 15: Island generated using Midpoint Displacement.

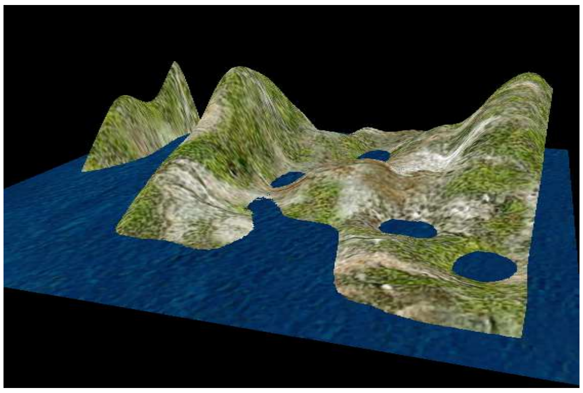


Figure 16: Island genereated using Perlin Noise.

Table 1: Survey result.

Algorithm/Terrain	Mountain	Plain	Island
Fault Formation	**156**	161	184
Midpoint Displacement	222	**136**	**136**
Perlin Noise	204	285	262