Realistic Image-Based Terrain Heightfield Generation

Joshua Barczak*  Nicholas A. Ferris*  Katherine A. Hirsch*
Computer Science and Electrical Engineering
University of Maryland Baltimore County

Figure 1: Image-based terrain from a firefighting simulation (left) and corresponding wireframe (right).

Abstract

Generating realistic 3D terrain heightfields from 2D image maps can be a non-trivial task. Simple image maps (for instance, those with few differing RGB values) typically produce terrains with jagged slopes. More complex image maps (those with many different RGB values), while producing smoother heightfields, take longer to produce.

We present an approach to image-based heightfield generation which uses Gaussian filtering and a random perturbation algorithm to create heightfields from simple image maps. We also discuss the use of procedural texturing on these heightfields to produce a realistic simulation of an outdoor terrain that includes grasslands, bodies of water, hills, and mountains.

Keywords: Terrain generation, heightfields, Gaussian filtering, procedural texturing.

1 Introduction

Randomized terrain generation has become a popular tool used by interactive game developers to create a variety of environments while saving development time and media storage space. In theory, a well-designed random terrain generation algorithm could produce a limitless number of interactive gameplay environments based on the same level design scheme.

There are a number of methods for generating these random terrains. In most cases, the realism of a generated terrain is closely correlated to the detail of the input used in the generation. If, for example, a 2D, RGB image map is the basis for a 3D landscape, increasing the number of different colors in the map will result in a heightfield with a larger degree of height variations across the terrain. However, producing image maps with high color counts is more labor-intensive than producing maps with only a few colors. Thus, the problem of using image maps with low color counts to produce terrains with realistic heightfields presents itself.

We present a solution to this problem which applies several techniques to a simple image map to produce a realistic, smooth heightfield. Our solution involves generating a basic heightfield from an image map of few colors, applying a Gaussian filtering function to the heightfield to create smooth slopes on edges, randomly perturbing the heightfield (with the amount of perturbation varying with the terrain type), applying a second Gaussian filter to smooth hills, and finally applying procedurally-generated textures to the final terrain.

2 Related Work

Terrain generation using heightfields has been used in interactive simulations for years [LoDH]. More recent work has developed methods for improving these heightfields, such as applying an averaging function to smooth elevations [GF], using binary triangle trees and quadtrees to render the heightfields [TeDH], and generating the heightfields from image maps [BMP].
With more and more emphasis being placed on the generation of real-time interactive environments, heightfield algorithm research has shifted more toward visibility determination [DLG] and level-of-detail rendering [UW, B3D].

It should be noted that most previous work with image-based terrain generation requires complex image maps to generate realistic terrains. Typically, these image maps will be based on topographic photographs of real geographic areas or, in cases where such photographs are not readily available (for instance, when the terrain to be rendered is representative of another planet), painstakingly designed artistic images. Our work focuses on allowing for cases when photographs and well-designed maps are not practical, such as when a large variety of image-based terrains must be generated. In such a case, simple image maps would suffice, but only if they would produce terrains approaching the realism of those generated from more complex images. The heightfield generation technique described here combines elements from several existing methods to solve the problem of producing realistic, natural heightfields from simple image maps.

3 Implementation Details

The following sections detail the implementation of heightfield generation algorithms built on top of the terrain generation system included in David Eberly’s Wild Magic 3D game engine [WM]. Figure 2 is a screenshot of a terrain generated in Wild Magic without textures or any of the improvements we will suggest. Note that the terrain has jagged hills and mountains. This is due to the fact that the image map used to generate the heightfield is limited to a few colors.

3.1 Primary Gaussian Filtering

To smooth the most obvious jagged slopes, we first apply a 5×5 integer-valued convolution kernel that approximates an isotropic Gaussian distribution of the form

\[ G(x, y) = \left( \frac{1}{2\pi\sigma^2} \right) e^{-\frac{x^2+y^2}{2\sigma^2}} \]

When \( \sigma = 1.0 \), this kernel is as appears in Figure 3. Note that each value is scaled by 273 [GS].

\[
\begin{array}{ccccc}
1 & 4 & 7 & 4 & 1 \\
4 & 16 & 26 & 16 & 4 \\
7 & 26 & 41 & 26 & 7 \\
4 & 16 & 26 & 16 & 4 \\
1 & 4 & 7 & 4 & 1 \\
\end{array}
\]

Figure 3: Primary Gaussian kernel.

Using an isotropic Gaussian filter instead of a standard averaging filter has the advantage of being rotationally symmetric. Such a characteristic is especially useful with heightfields generated from image maps when the direction of smoothing is arbitrary.

At this point in the terrain generation process, the transitions between terrain types (represented in the image map by color transitions) have been smoothed. However, the terrains are unrealistically smooth except for at terrain type transitions. Figure 4a shows a bitmap used to generate the terrain shown in Figure 4b (with the Gaussian filtering applied).

Figure 4: Example image map.

Figure 4: Gaussian-filtered terrain.
3.2 Random Perturbations

To create realistic variations within each terrain type following the application of the 5x5 Gaussian filter, we next add random perturbations across the heightfield. The amount of perturbation varies with each terrain type. The following algorithm demonstrates this method.

```plaintext
for each (x, y) in the heightfield
    h = heightfield.GetHeight(x, y)
    delta_max = terrain.PerturbAmount(Type(x, y))
    delta = ( rand() % 2*delta_max )
    - delta_max
    h += delta
    if (h < 0) h = 0
    heightfield.SetHeight(x, y, h)
```

The terrain's PerturbAmount() function returns an integer value associated with the particular terrain type at (x, y). Smoother terrain types (such as shallow water and grasslands) are associated with lower perturbation amounts while terrain types with larger point-to-point height variations (such as mountains) receive a higher degree of perturbation.

Once the random perturbations have been applied, we are left with an unrealistically jagged terrain (Figure 5a). While one possible solution to this problem would be to decrease the earlier amount of perturbation (Figure 5b), this still leaves jagged hills.

3.3 Secondary Gaussian Filtering

Instead of reducing the amount of perturbation, we can apply a second Gaussian filter to smooth out the hills. Unlike the 5x5 kernel we applied earlier, we use a 3x3 now (Figure 6) because another 5x5 kernel would destroy the hills created by the random perturbations.

```
1 2 1
2 4 2
1 2 1
```

The result of this filter on our example terrain is shown in Figure 6. Notice the numerous grass hills in the background not present just after the primary Gaussian filter is applied. Also notice that the secondary Gaussian filter leaves more hills intact than simply decreasing the amount of perturbation as in Figure 5b.

3.4 Procedural Texturing

At this point, the heightfield values are in their final state. While not directly related to the generation of a realistic heightfield, we can use our heightfield to texture the terrain.
In our example, we have five terrain types: deep water, shallow water, grasslands, hills, and mountains. Obviously, each terrain type must have its own representative texture. However, we must also consider areas of our terrain that are in transition from one terrain type to another. For example, where shallow water and grasslands meet, we would like to have a dirt or sand texture.

One possible method to achieve this terrain transition would be to edit the original image map to add another color to represent the transitional terrain type. This would also require us to create an additional parameter in the random perturbation function to handle the heightfield convolutions for the new terrain type. However, if each terrain type must have a transitional type for every other terrain type, an \( n \) -color image map will actually need \( 1 + \ldots + n \) colors to handle all possible transitions. This counters our goal of keeping image maps simple.

One alternative would be to procedurally generate both the base textures for non-transitional terrain areas and use the Type() function called during the random perturbation process to detect transitional areas of the terrain and place the appropriate transitional textures over those areas.

Figure 7 shows the use of procedural texturing in a transitional area of the terrain.

After deciding to use image maps to generate the terrain heightfields, we developed the series of techniques described earlier to enable us to create realistic terrains from relatively simple image maps. As a result, a new and interesting simulation level can be designed in any bitmap editor in a matter of seconds.

Figure 8 shows some of the bitmaps created for the firefighting simulation and their corresponding 3D terrains (complete with burning trees and smoke).

**Figure 7: Procedural texturing on transitional terrain.**

**Figure 8: Image bitmaps and corresponding 3D terrains.**

## 4 Interactive Applications

The example terrain maps we have shown were developed for use in an interactive firefighting simulation. The algorithms and techniques presented here were especially useful when developing the simulation because we required the ability to create a variety of different terrains. Purely randomly generated terrains could not be used because some of the generated terrains were not well suited to interesting gameplay. For example, a line of impassable mountains could block the player from reaching the area of the map on fire.

Figure 8 shows some of the bitmaps created for the firefighting simulation and their corresponding 3D terrains (complete with burning trees and smoke).

## 5 Results

The heightfield convolution algorithms, even when used in parallel, operate on a terrain generated from a 128×128 image bitmap in a trivial amount of time. Figure 9 shows the time for the firefighting simulation to render the terrain with none, some, or all of the heightfield convolution algorithms enabled.

<table>
<thead>
<tr>
<th>Heightfield convolution techniques enabled</th>
<th>Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>5.17</td>
</tr>
<tr>
<td>Primary Gaussian filtering</td>
<td>5.18</td>
</tr>
<tr>
<td>Above and random perturbations</td>
<td>5.20</td>
</tr>
<tr>
<td>Above and secondary Gaussian filtering</td>
<td>5.22</td>
</tr>
</tbody>
</table>

Figure 9: Heightfield convolution effects on rendering time.

Similarly, none of the techniques had any significant impact on frame rates. The frame rates in Figure 10 (measured in frames per second) are the average rates over entire program execution of the firefighting
simulation with full tree burning and smoke effects enabled.

<table>
<thead>
<tr>
<th>Heightfield convolution techniques enabled</th>
<th>Frame rate (fps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>39.7</td>
</tr>
<tr>
<td>Primary Gaussian filtering</td>
<td>39.1</td>
</tr>
<tr>
<td>Above and random perturbations</td>
<td>38.8</td>
</tr>
<tr>
<td>Above and secondary Gaussian filtering</td>
<td>38.4</td>
</tr>
</tbody>
</table>

Figure 10: Heightfield convolution effects on frame rate.

All benchmarks were calculated on a system with a Pentium M 1.5GHz processor, 512MB RAM, and an ATI Radeon Mobility 9600 64MB graphics card.

6 Conclusions and Future Work

The three-fold heightfield convolution technique described here provides a method for generating terrains based on image maps without relying on the complexity of the image to produce realistic heightfields. These heightfields also lend themselves to simple procedural texturing that even takes transitional terrain types into account.

Future work on this topic may include exploring the use of these methods in real-time continuous level of detail terrain systems. The low impact of these techniques on rendering time and frame rate already suggests that such an extension would be possible.

The techniques themselves could also be further refined to take advantage of other convolution algorithms, including Bayesian and Monte Carlo filtering.

Acknowledgements

We would like to thank Marc Olano from the University of Maryland Baltimore County and David Eberly, President of Magic Software Inc., for their assistance with this project.

References


