Realtime Procedural Terrain Generation

Realtime Synthesis of Eroded Fractal Terrain for Use in Computer Games

Jacob Olsen, xenorg@imada.sdu.dk
Department of Mathematics And Computer Science (IMADA)
University of Southern Denmark

October 31, 2004

Abstract

The main goal of this paper is to provide an overview of a variety of methods for synthesis of eroded terrain for use in computer games, VR worlds and the like. Traditionally, such software uses either predefined terrains or runtime generated data based on simple fractal noise techniques.

In recent years, the advances in processing power of average home computers have made it possible to simulate erosion processes near-realtime by putting emphasis on speed at the expense of physical correctness. This paper presents a fast method to synthesize natural looking fractal terrain and then proceeds to evaluate and suggest optimizations for two of the most commonly used erosion algorithms [1, 2]. With some criteria for applicability in computer games in mind, a new and much faster algorithm is then proposed. Finally, a few issues regarding terrain modifications for maximum playability are discussed.

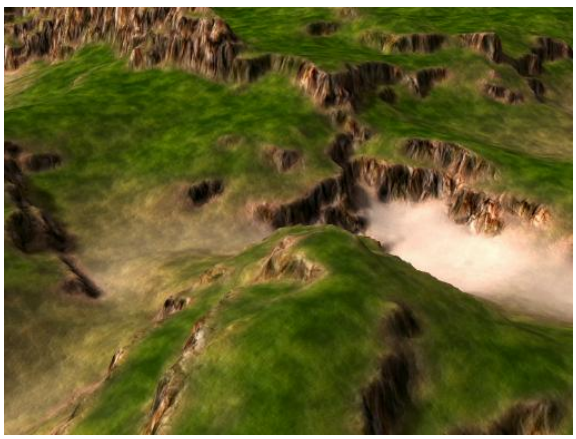

Figure 1: *A rendered view of a synthesized, eroded terrain created with the techniques discussed in this paper.*

Definitions

Data representation

In the algorithms described in this paper, terrain will be represented by two-dimensional height maps using floating point values between 0 and 1. Unless otherwise stated, all examples use square maps with side length $N = 2^9 = 512$, giving a total of $N^2 = 2^{18} = 262144$ cells, each cell containing a height value.

The height map is denoted H and the individual cells are addressed as $h_{i,j}$, where i and j are coordinates ranging from 0 to 511. Some calculations will address cells outside this range; in this case, modulo is used to wrap the coordinates around so that the right neighbour of a right-most cell will be the left-most cell in the same row etc.

All implementations were done i Java, and all calculation times are from tests executed on a fairly standard 2.4 GHz Pentium 4 PC.

Defining erosion

The effects of erosion are difficult to describe mathematically: The term erosion covers many naturally occurring phenomena, and different terrain types and climates will produce many different kinds of changes to a landscape. For simplicity, a set of desirable traits (from a computer game development perspective) that will be used to measure how eroded a height map is, is defined. Overall, most types of erosion dissolve material from steep slopes, transport it downhill and then deposit the material at lower inclinations. This tends to make steep slopes even steeper, and flatten out low-altitude terrain when the transported material is deposited. To aid in the analysis of the changes in inclination, the slope map S is defined

such that

$$s_{i,j} = \max(|h_{i,j} - h_{i-1,j}|, \\ |h_{i,j} - h_{i+1,j}|, \\ |h_{i,j} - h_{i,j-1}|, \\ |h_{i,j} - h_{i,j+1}|)$$

in other words, the greatest of the height differences between the cell and its four neighbours in a Von Neumann neighbourhood.

This paper focuses on the synthesis of eroded terrain for use in computer games; therefore, the ideal for eroded terrain must suit this application. Physical correctness and visual appearance are secondary, what matters is applicability. In most computer games and VR environments using large-scale outdoor terrain, persons or vehicles move around on the terrain, and various structures are placed on the terrain. Movement and structure placing is often restricted to low inclinations, which means that a low average value of a height map's corresponding slope map is desirable. This rule alone would make a perfectly flat height map ideal, which is why a second rule is added saying the greater the standard deviation of the slope map, the better. The ideal for eroded terrain is therefore a height map whose corresponding slope map has a low mean value (reflecting the overall flattening of the terrain due to material deposition) and a high standard deviation (material is dissolved from steep areas making them even steeper, and deposition flattens the flat areas further). The slope map mean value, $\bar{s}$, and standard deviation, σ_s, are defined on the slope map S as follows:

$$\bar{s} = \frac{1}{N^2} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} s_{i,j}$$

$$\sigma_s = \sqrt{\frac{1}{N^2} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} (s_{i,j} - \bar{s})^2}$$

Using these, an overall "erosion score", ε, is defined as

$$\varepsilon = \frac{\sigma_s}{\bar{s}}$$

(on the assumption that $\bar{s} \neq 0$)

Generation of base terrain

A technique often used for fast terrain generation is simulating $1/f$ noise (also known as "pink noise") which is characterized by the spectral energy density being proportional to the reciprocal of the frequency, i.e.

$$P(f) = \frac{1}{f^a}$$

where $P(f)$ is the power function of the frequency and a is close to 1. This kind of noise approximates real-world uneroded mountainous terrain well and has been used widely in computer graphics for the past decades. Two methods for generating $1/f$-like noise, spectral synthesis and midpoint displacement, are discussed below.

In generating a terrain base for the erosion algorithms to work on, it is worth noting that the closer the terrain base is to the desired result, the less work is required by the (often calculation heavy) erosion algorithm itself. To help create a terrain base with better characteristics of eroded terrain, the use of Voronoi diagrams and perturbation filtering are introduced below.

Spectral synthesis

Spectral synthesis simulates $1/f$ noise by adding several octaves (layers) together, each octave consisting of noise with all its spectral energy concentrated on a single frequency. For each octave, the noise frequency is doubled and the amplitude A is calculated by

$$A = p^i$$

where i is the octave number starting with 0 at the lowest frequency and p is called the persistence. Letting $p = 0.5$ will approximate $1/f$ noise because each time the frequency is doubled in the next octave, the amplitude will be halved.

The octaves themselves are created by filling in evenly spaced pseudo random numbers corresponding to the octaves's frequency, and then calculate the remaining values by interpolation - see Figure 2 for a visual comparison of interpolation methods. While cubic interpolation gives the best results, the slightly visible vertical and horizontal artifacts caused by linear interpolation are an acceptable trade-off for a computation time reduced to roughly one fifth.

Midpoint displacement

Another approach at simulating $1/f$ noise is by a midpoint displacement method, in this case the diamond-square algorithm [3, 4, 5]. Instead of calculating every cell in several octaves (up to 9 octaves with $N = 2^9$) and then adding together the octaves, the value of each cell need only be calculated once.

The midpoint displacement method works by recursively calculating the missing values halfway

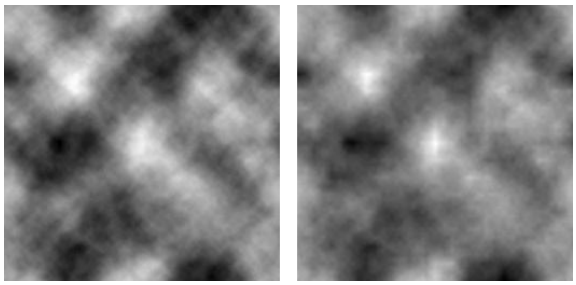

Figure 2: *Cubic interpolation (left) versus linear interpolation (right) for the spectral synthesis algorithm.*

between already known values and then randomly offset the new values inside a range determined by the current depth of the recursion. With a persistence of 0.5, this range is halved with each recursive step, and an approximation of $1/f$ noise is created. Ideally, the random offsets should have a gaussian distribution inside the offset range, but for the purpose of synthesizing terrain, uniformly distributed values are acceptable (and much faster to calculate).
The implementation done for this paper is the square-diamond algorithm, named after the order in which midpoint values are determined (see Figure 3).

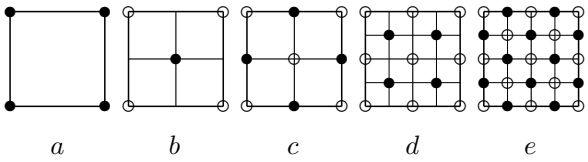

Figure 3: *Two iterations of the diamond-square algorithm. Pseudo random number are used for initial values in step a. In step b (the "diamond" step) a new value is found by offsetting the average of the four values of step a. Step c (the "square" step) fills in the rest of the midpoint values also by offsetting the average of the four neighbours of each new point. Steps d and e show the next iteration.*

Figure 4 shows a visual comparison of the two ways of distributing values inside the random offset ranges. Although uniform distribution produces a more jagged terrain, this can be compensated for by lowering the persistence. Since the version using gaussian distribution takes 4 times longer to generate, uniform distribution is to be preferred.
The midpoint displacement method also allows for individual adjustments of the random offset ranges depending on coordinates or altitude, which can be used to give the terrain a more eroded look by multiplying the size of the offset range with the height average when calculating new values. This causes low altitude areas to become smoother, thereby simulating deposition of eroded material. This method is referred to as smoothed midpoint displacement.

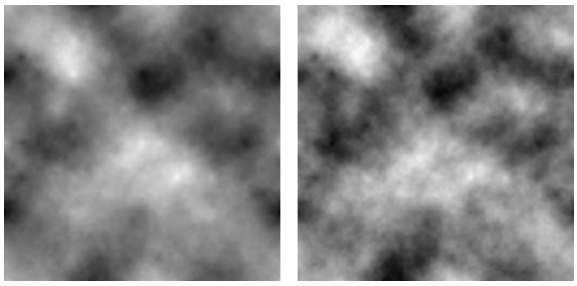

Figure 4: *Gaussian (left) versus uniform (right) distribution of random offsets for the midpoint displacement algorithm.*

Voronoi diagrams

The problem with using $1/f$ noise for simulating real world terrain is that it is statistically homogeneous and isotropic - properties that real terrain does not share. One way to break the monotony and control the major characteristics of the landscape are Voronoi diagrams whose use in procedural texture generation has been described by Steven Worley [6]. Voronoi diagrams can be used for a variety of effects when creating procedural textures - most variants resemble some sort of cell-like structures that can be used to simulate tissue, sponge, scales, pebbles, flagstones, or in this case, entire mountains.
The implementation used in this paper works by dividing the map into regions and then randomly place a number of "feature points" in each region. For each cell in the map, a set of values d_n, $n = 1, 2, 3, \ldots$ are calculated according to a defined distance metric so that d_1 is the distance to the nearest feature point, d_2 is the distance to the next nearest distance point etc. Linear combinations of the form

$$h = c_1 d_1 + c_2 d_2 + c_3 d_3 + \cdots + c_n d_n$$

with coefficients $c_1 \ldots c_n$ will then produce the cellular structures - see Figure 5 for examples. For creating mountainous features, the coefficients $c_1 = -1$ and $c_2 = 1$ (with the rest being zeroes) are used as it can add distinct ridge lines and connected riverbeds to the terrain. These values also give the Voronoi diagrams another useful property which will be

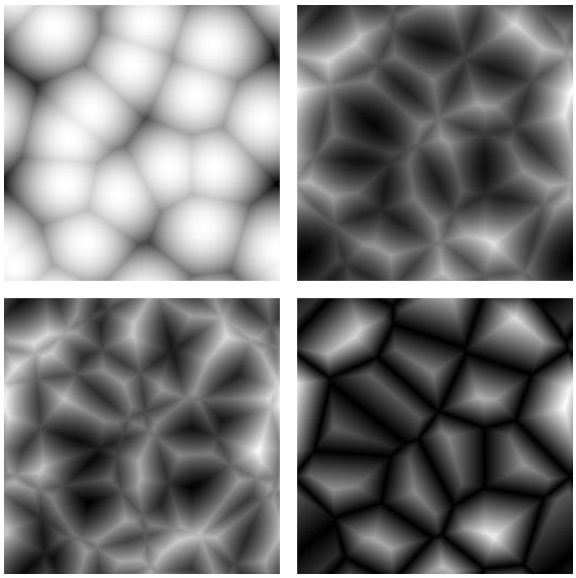

Figure 5: *Examples of Voronoi diagrams with coefficients $c_1 = -1$ (upper left), $c_2 = 1$ (upper right), $c_3 = 1$ (bottom left), $c_1 = -1$ and $c_2 = 1$ (bottom right).*

covered in the section regarding playability issues.

Normally, distances are determined by the Euclidean distance metric

$$d = \sqrt{dx^2 + dy^2}$$

which is quite slow because of the square root. Changing the distance metric to

$$d = dx^2 + dy^2$$

produces a large speedup. As Figure 6 shows, the difference in the resulting height map is insignificant. This optimization together with a reduction in search radius when finding nearest feature points (which occasionally produces minor errors) reduces calculation time to one third.

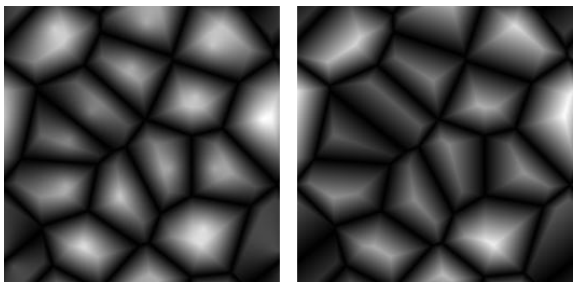

Figure 6: *Euclidean distance metric (left) versus the faster distance metric (right) for Voronoi diagrams.*

Combination and perturbation

Although Voronoi diagrams have some useful properties that $1/f$ noise lacks, they are no substitute for the noise functions. The best results are achieved with some combination of both; in this case two thirds smoothed diamond-square method noise and one third Voronoi diagram with coefficients $c_1 = -1$ and $c_2 = 1$ will be used. This combination is referred to as the combined height map.

To crumple the straight lines of the Voronoi diagram, a perturbation filter as described in [6] pages 90-91 is applied. This filter works by using a noise function (similar to the ones described above) to calculate a displacement with random distance and direction for each cell. The combined height map before and after perturbation can be seen in Figure 7. The magnitude of the perturbation filtering is set to 0.25, meaning that a given point in the height map cannot be displaced more than $\frac{N}{4}$ cells.

The perturbation filtering itself also increases the erosion score because some areas are stretched and some are compressed, which increases σ_s. Figure 8 shows the average relationship between perturbation magnitude and erosion score for at large number of test runs on the combined height map generated from different random seed numbers. Erosion score rises to a maximum at a perturbation magnitude of 0.25 and then slowly declines.

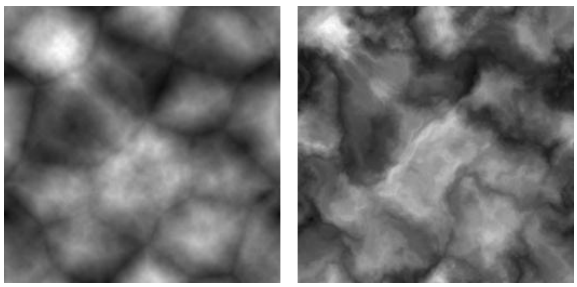

Figure 7: *The combined height map before perturbation (left) and after (right).*

The final base terrain is shown in Figure 9. For visual comparison, all image examples of various erosion algorithms in the following sections use this terrain as a starting point. Figure 10 shows a rendered view of this height map.

Analysis

Average calculation times and erosion scores for the methods discussed in this section can be seen in Table 1. As can be seen, the implementations of spectral synthesis and midpoint displacement

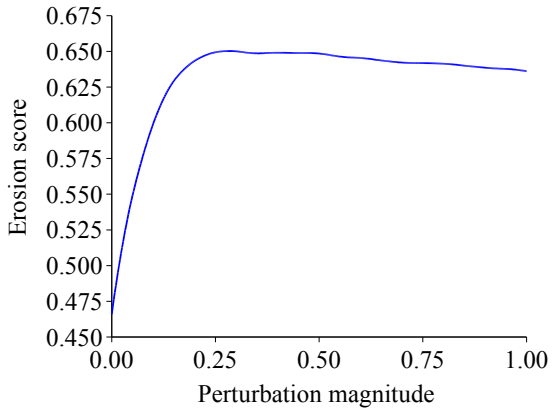

Figure 8: *The relationship between perturbation magnitude and erosion score for the combined height map.*

all achieve nearly the same erosion score, but the midpoint displacement method with uniform random offset distribution is by far the fastest. The smoothed version is only marginally slower, but manages to achieve a higher erosion score.
The Voronoi diagrams in themselves do not score as much as the noise functions, but the faster metric seems to be better suited for the coefficients used. When combined with the modified version of the midpoint displacement method, the erosion score almost reaches the level of the modified midpoint displacement method alone. As shown in Figure 8, the perturbation filter improves the erosion score even further.
With $N = 512$ the base terrain can be synthesized in less than 1 second. Even with $N = 1024$, the synthesis of the base terrain is done in less than 3 seconds.

Erosion algorithms

Two types of erosion algorithms are examined in this section, namely thermal erosion (sometimes referred to as thermal weathering) and hydraulic erosion. These were first described by Ken Musgrave *et al* in 1989 [1], and have since established themselves as a base from which various improvements (mostly in terms of physical correctness) have been suggested [2, 7, 8, 9, 10].
A reference implementation of each type is compared to speed optimized version that will still deliver comparable results. For thermal erosion, the original method suggested in [1] is used, while a version of hydraulic erosion suggested in [2] is used because of its speed.
Both methods are iterated cellular automata meaning that calculations in each iteration are

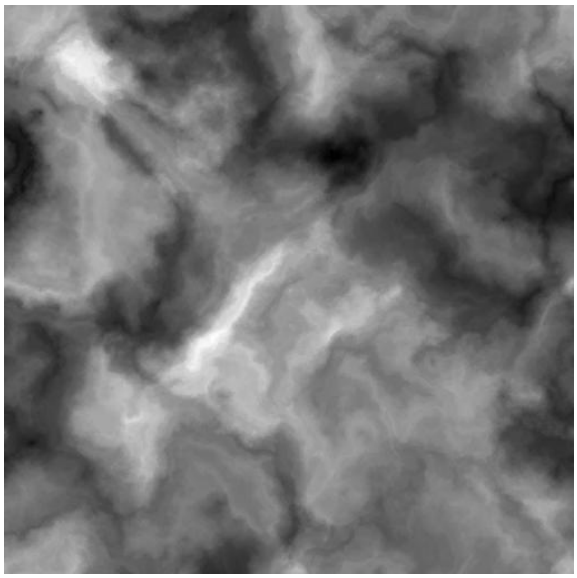

Figure 9: *The base terrain used in image examples of the erosion algorithms.*

done by examining each cell and its neighbourhood in turn. Two different types of neighbourhoods are used: The Moore neighbourhood which includes all 8 neighbours of a cell, and the Von Neumann neighbourhood which only includes 4 of the neighbouring cells (see Figure 11). With the currently examined cell having value h and its neighbours being named h_i, the height difference to each neighbour, d_i, is defined as

$$d_i = h - h_i$$

meaning that lower neighbours produce positive height differences. For maximum correctness, the Moore neighbourhood was used in both reference implementations.

Thermal erosion

Overview

Thermal erosion simulates material braking loose and sliding down slopes to pile up at the bottom. The reference implementation works as follows: A percentage of the material at the top of a slope whose inclination is above a threshold value - the talus angle T - will be moved down the slope until the inclination reaches T:

$$h_i = \begin{cases} d_i > T: & h_i + c(d_i - T) \\ d_i \leq T: & h_i \end{cases}$$

This is illustrated in Figure 12: At the first timestep, $d_1 = T$ and $d_2 > T$, which means that material will be moved from h to h_2. With $c = 1$, the amount of moved material results in $d_2 = T$

Type	N	Calc. time	Erosion score
Spectral synthesis, cubic interpolation	512	0.783 s	0.425
Spectral synthesis, linear interpolation	512	0.157 s	0.417
Midpoint displacement, Gaussian distribution	512	0.439 s	0.438
Midpoint displacement, uniform distribution	512	0.108 s	0.401
Midpoint displacement, uniform distribution, smoothed	512	0.144 s	0.478
Voronoi diagram, Euclidean metric, long search range	512	1.322 s	0.323
Voronoi diagram, fast metric, short search range	512	0.468 s	0.347
Noise and Voronoi combination	512	0.709 s	0.460
Noise and Voronoi combination, perturbed	512	0.831 s	0.657
Noise and Voronoi combination, perturbed	1024	2.738 s	0.673

Table 1: *Calculation times and erosion scores for the methods discussed in the first section. Calculation times for combinations include time to calculate the noise and Voronoi maps. All numbers are averages from a large number of test runs.*

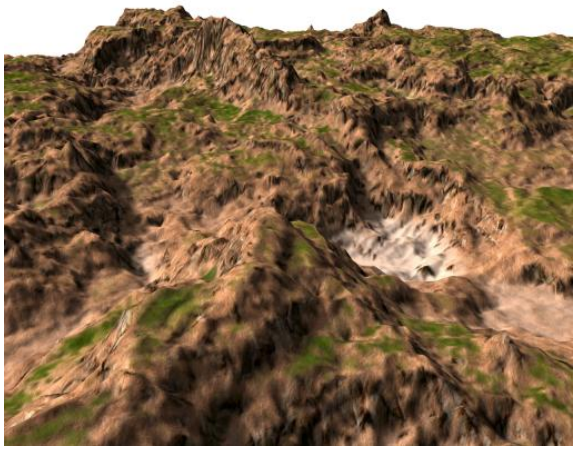

Figure 10: *A rendered view of the base terrain used in image examples of the erosion algorithms. For easy comparison, all rendered views use the same camera position and direction.*

and $d_1 < T$. However, this is a simplified example where material from h only needs to be moved to one neighbour cell. In the case where several neighbours whose height difference is above the talus angle exist, the moved material must be distributed after the form

$$h_i = h_i + c(d_{\max} - T) \times \frac{d_i}{d_{\text{total}}}$$

where $d_{\max}$ is the greatest of the d_i and d_{total} is the sum of the d_i greater than T.

A reasonable value for c is 0.5; higher values may cause oscillation when the changes to the height map are applied only after completion of an entire iteration, and lower values will simply cause slopes steeper than T a slower asymptotical approach to the talus angle. For the talus threshold, a value of $T = \frac{4}{N}$ was chosen.

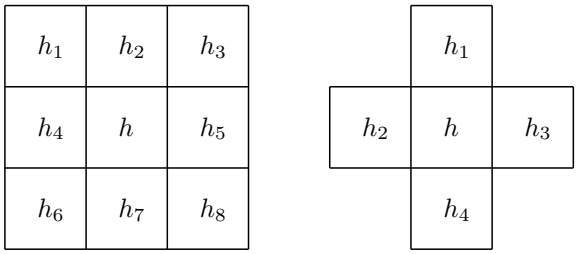

Figure 11: *Moore (left) and Von Neumann (right) neighbourhoods for cellular automata.*

Optimizations

To produce a speed optimized version of the reference implementation, four changes were made:

1. A Von Neumann neighbourhood was used instead of the Moore neighbourhood.
2. Material distribution was changed so that material is only distributed to the lowest neighbour instead of all lower neighbours.
3. Material distribution was changed to allow more material to be moved per iteration.
4. The calculations for each cell changes the height map immediately instead of being written to a difference map that is applied after an entire iteration is completed.

The reference implementation maintains values for $d_{\max}$ and d_{total}, which means that for every neighbour h_i of a cell h, the following must be done:

$d_i = h - h_i$
if $(d_i >$ talus$)$:
    $d_{\text{total}} = d_{\text{total}} + d_i$
    if $(d_i > d_{\max})$:
        $d_{\max} = d_i$

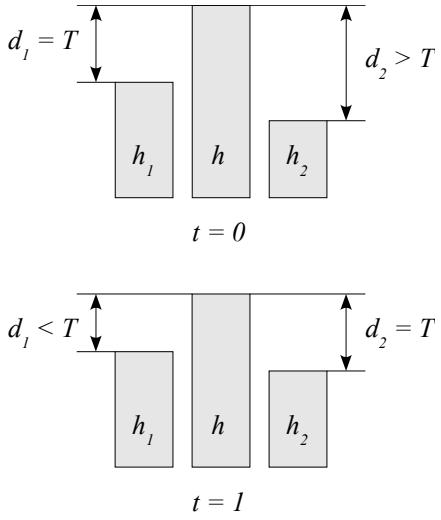

Figure 12: *A simplified example of thermal erosion: d_2 is greater than the talus angle, so material is moved from h to h_2 until d_2 equals the talus angle.*

Switching from the Moore neighbourhood to the Von Neumann neighbourhood will halve the number of times these conditional checks have to be done. Since the amount of moved material per cell is proportional to $d_{\max}$, using the Von Neumann neighbourhood will move the same amount of material 50% of the time. Even if the $d_{\max}$ of the Moore neighbourhood is outside of the Von Neumann neighbourhood, the $d_{\max}$ of the Von Neumann neighbourhood still tends to be close to its value.

When the amount of material to be moved has been calculated, each neighbour h_i whose $d_i > T$ receives a fraction proportional to $\frac{d_i}{d_{\text{total}}}$ (where d_{total} is the sum of the d_i greater than T). This can be simplified a lot by distributing material to the lowest neighbour only as this renders the calculation of d_{total} and the fractions to be distributed superfluous. A drawback is that less material can be moved per cell, which can partly be compensated for by moving as much material Δh as possible:

$$\Delta h = \frac{d_{\max}}{2}$$

This causes h to be levelled with its lowest neighbour if their height difference is greater than T.

In the reference implementation, only a percentage c of the maximum amount of material to be moved was transfered. This was done to avoid oscillation, and the same problem applies here: Four large height values surrounding a deep hole may not only fill up the hole, but create a tall spike instead. Oscillations like this occur because the height map remains unchanged until all cells have been processed. One way to solve this is to change the height map immediately when moving material - in the above example this would mean that the hole would not receive any more material once it was raised to a level where the height differences to the surrounding cells were below the talus threshold. Another advantage of direct changes to the height map is a slight increase in calculation speed.

When experimenting with different kinds of neighbourhoods, it was also noted that a "rotated" version of the Von Neumann neighbourhood (see Figure 13) gave slightly better results both in terms of higher erosion score and less difference between the two versions.

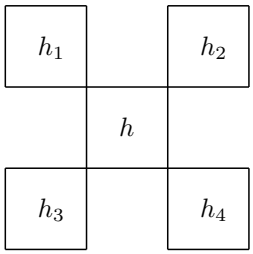

Figure 13: *The modified Von Neumann neighbourhood used in the speed optimized version of thermal erosion.*

Analysis

Calculation time averages for the first 500 iterations of the two implementations can be seen in Figure 14. Time required per iteration remains constant for both implementations, but the reference implementation takes 6 times longer; 500 iterations are calculated in 60 seconds, while the speed optimized version does it in 10 seconds.

Figure 15 shows erosion score averages of the first 500 iterations. The reference implementation scores 5% better after 500 iterations, but the speed optimized version seems to be more effective during the first 80 iterations. To explore this further, a new graph was created, showing the height map difference after every 10 iterations (see Figure 16). The optimized version seems to stabilize faster, meaning that most of the change is done during the first 50 iterations. The reference implementation does more change overall during the 500 iterations, but does not seem to be stabilizing until after 150 iterations.

For practical or visual applications, no more than 50 iterations of any of the versions are needed to change the shape of the terrain to show the distinct effects of thermal erosion, namely the con-

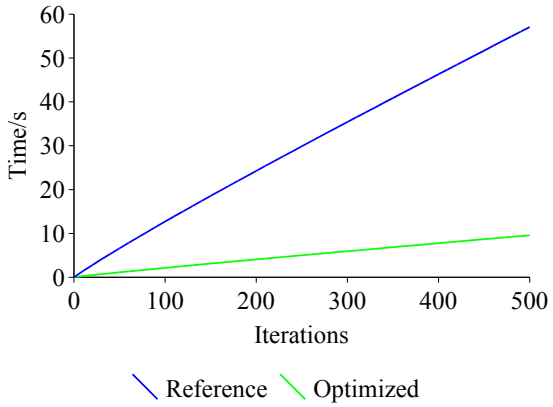

Figure 14: *Calculation times of the first 500 iterations of the reference and optimized implementation of thermal erosion.*

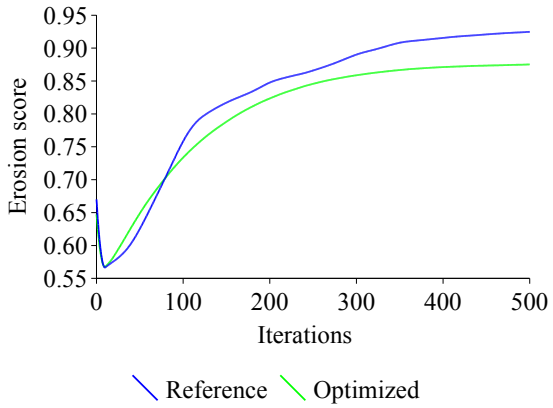

Figure 15: *Erosion scores of the first 500 iterations of the reference and optimized implementation of thermal erosion.*

stant angled slopes. Figure 17 compares height maps produced by the two versions after 50 iterations, and Figure 18 shows rendered views of these height maps.

The difference between the outputs of the two implementations after 50 iterations is 2% on average, where a difference of 100% corresponds to a height map with all height values at 0 versus a height map with all height values at 1.

While thermal erosion manages to increase the erosion score by lowering the angle of most slopes, terrains produced by this kind of erosion does not resemble the ideal defined earlier since the constant angled slopes leave very little completely flat area.

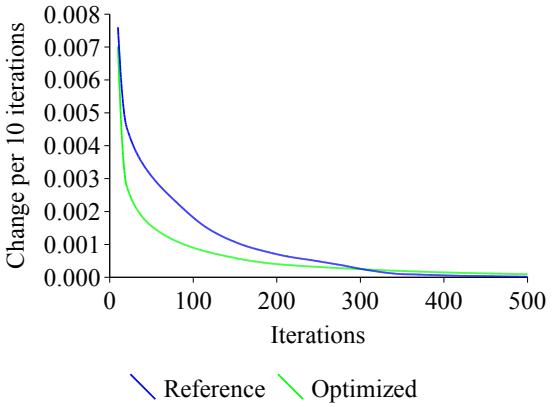

Figure 16: *Change per 10 iterations of the first 500 iterations of the reference and optimized implementation of thermal erosion.*

Hydraulic erosion

Overview

Hydraulic erosion simulates changes to the terrain caused by flowing water dissolving material, transporting it and depositing it elsewhere.
For the reference implementation of hydraulic erosion, a slightly altered version of Beneš and Forsbach's method [2] has been used. The algorithm is split up into four independent steps:

1. Appearance of new water.
2. Water eroding the underlying terrain and capturing the dissolved material.
3. Transportation of water and sediment.
4. Evaporation of water and deposition of sediment.

Apart from the height map, hydraulic erosion also maintains a water map W and a sediment map M for keeping track of the flow of water and dissolved material.
In step 1, a constant amount of water K_r is added to each cell every iteration to simulate rain:

$$w_{i,j} = w_{i,j} + K_r$$

In step 2, an amount of the height value proportional to the amount of water present in the same cell is converted to sediment:

$$h_{i,j} = h_{i,j} - K_s \times w_{i,j}$$
$$m_{i,j} = m_{i,j} + K_s \times w_{i,j}$$

where K_s is the solubility constant of the terrain.
In step 3, the water and its contents of dissolved material are transported following a set of rules

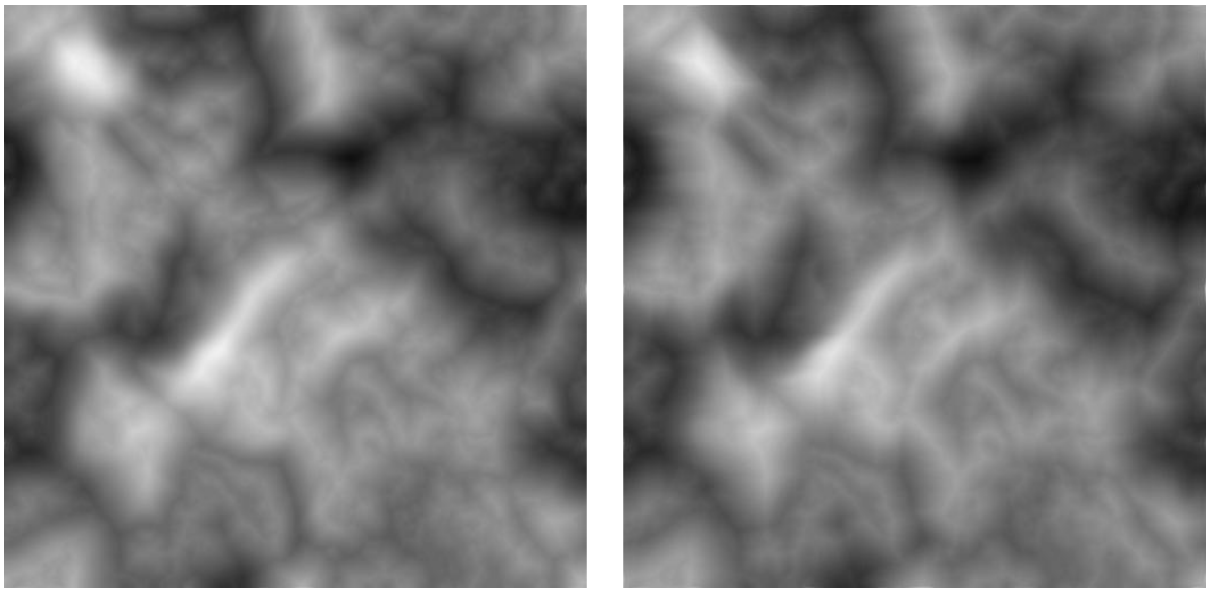

Figure 17: *Comparison between the reference implementation of thermal erosion (left) and the speed optimized version (right). Images show the height map after 50 iterations.*

similar to the material distribution in thermal erosion. Only now, water is being distributed to the neighbour cells instead and the distribution seeks to level out any height differences of the total altitude $a = h + w$ so that

$$a = a_i$$

for each neighbour i in the cell's neighbourhood whose total height is less than that of the currently examined cell.
The amount of water moved to the ith neighbour, Δw_i, is calculated by

$$\Delta w_i = \min(w,\ \Delta a) \times \frac{d_i}{d_{\text{total}}}$$

where $\Delta a = a - \bar{a}$ is the total height of the current cell minus the average total height of the cells involved in the distribution, $d_i = a - a_i$ and d_{total} is the sum of all positive d_i.
If the total amount of water to be moved away from the currently examined cell is less than the amount of water present in this cell, an amount equalling the difference between the total height a of the cell and the average total height $\bar{a}$ after water distribution, is moved. If more water needs to be moved away than the cell holds, whatever amount of water present will be distributed among the lower neighbours. These two cases are illustrated in Figure 19.
Once the water distribution has been calculated, sediment distribution follows quite easily. An assumption is made that all dissolved material m is uniformly distributed within the water w of the cell. The amount of sediment Δm_i to be transported to a neighbouring cell i is therefore

$$\Delta m_i = m \times \frac{\Delta w_i}{w}$$

In step 4, a percentage of the water w determined by the evaporation coefficient K_e evaporates again:

$$w = w \times (1 - K_e)$$

The maximum amount of sediment m_{max} that can be carried by the water w is determined by the sediment capacity coefficient K_c such that

$$m_{max} = K_c \times w$$

Once part of the water has evaporated, the amount of sediment exceeding the maximum capacity (if any), Δm, is transfered back to h:

$$\begin{aligned} \Delta m &= \max(0,\ m - m_{\max}) \\ m &= m - \Delta m \\ h &= h + \Delta m \end{aligned}$$

The various constants and coefficients used in the reference implementation are:

$$\begin{aligned} K_r &= 0.01 \\ K_s &= 0.01 \\ K_e &= 0.5 \\ K_c &= 0.01 \end{aligned}$$

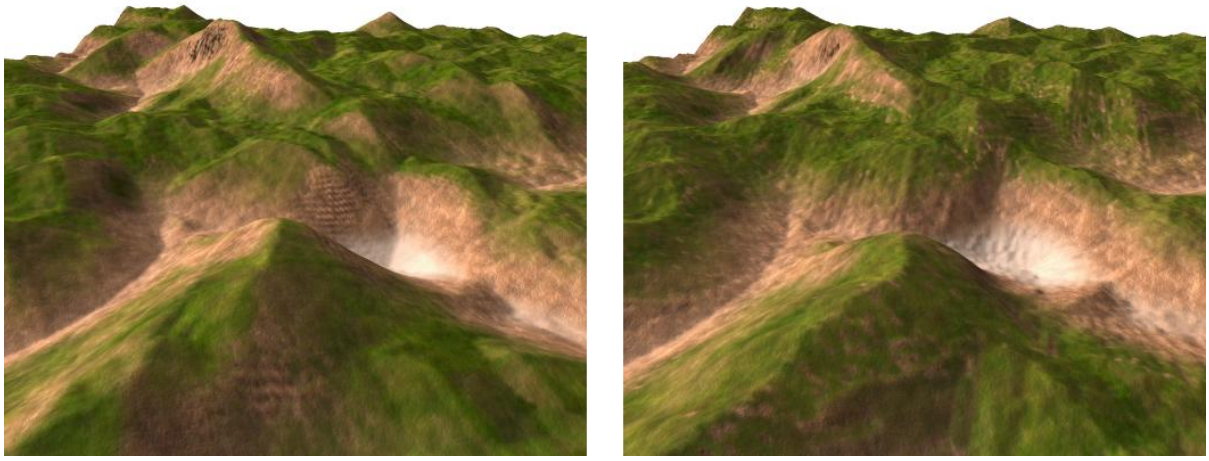

Figure 18: *Renderings from the two height maps shown in Figure 17 - the left image shows the reference implementation after 50 iterations, and the right image shows the speed optimized version after 50 iterations.*

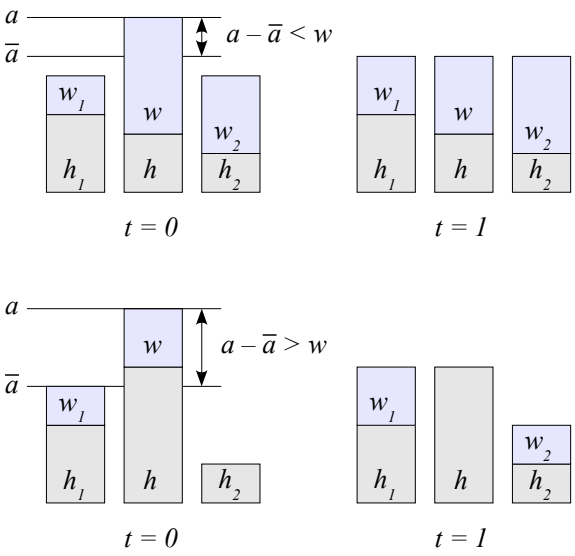

Figure 19: *Two cases for water distribution in hydraulic erosion. In the first case, only a fraction of the water w is moved to level out the total heights, while in the second case all of the available water is distributed to the neighbours without reaching level.*

Optimizations

To speed optimize the reference implementation, four changes were made:

1. A Von Neumann neighbourhood was used instead of the Moore neighbourhood.

2. Water distribution was changed so that water is only distributed to the lowest neighbour instead of all lower neighbours.

3. Instead of maintaining a sediment map, all water is assumed to contain an equal amount of dissolved material.

4. The calculations for each cell change the height and water maps immediately instead of being written to difference maps that are applied after an entire iteration is completed.

As with the speed optimizations of thermal erosion, the change in neighbourhood is done to halve the number of neighbour cells for which the following needs to be done for every cell in each iteration:

$$
\begin{aligned}
&a_i = h_i + m_i \\
&d_i = a - a_i \\
&\text{if } (d_i > 0): \\
&\qquad d_{\text{total}} = d_{\text{total}} + d_i \\
&\qquad a_{\text{total}} = a_{\text{total}} + a_i \\
&\qquad \text{cells} = \text{cells} + 1
\end{aligned}
$$

where a_{total} and cells are used to calculate $\bar{a}$. The change of neighbourhood stills results in roughly the same amount of water being transported; only the distribution differs. This is because the amounts of water being transported are so small compared to the height differences between the cells that the second case in Figure 19 where $w < a - \bar{a}$ by far is the most frequently occurring.
This allows for a good approximation of the water distribution by simply moving water to the lowest neighbour only, thereby saving the need for calculating the fractions received by each lower neighbour.
In the reference implementation, the choice of coefficients such that $K_s = K_c$ means that the

amount of material dissolved per water equals the transport capacity of the water. This causes all the water to always be at maximum sediment saturation: Any water evaporation is immediately followed by sedimentation of the material that exceeds the transport capacity. Since $m = K_c \times w$ for every cell, there is no need to maintain a separate sediment map. Even with $K_c < K_s$ this is a good approximation, since w drops exponentially which means that maximum saturation is reached very quickly.
To limit reads and writes to the height and water maps further, all changes are written directly instead of using difference maps that are applied only after the entire iteration has been completed.

Analysis

Calculation time averages for the first 500 iterations of the two implementations can be seen in Figure 20. Time required per iteration remains very close to constant for both implementations, but the reference implementation takes 5.5 times longer; 500 iterations are calculated in 148 seconds on average, while the speed optimized version does it in 27 seconds.

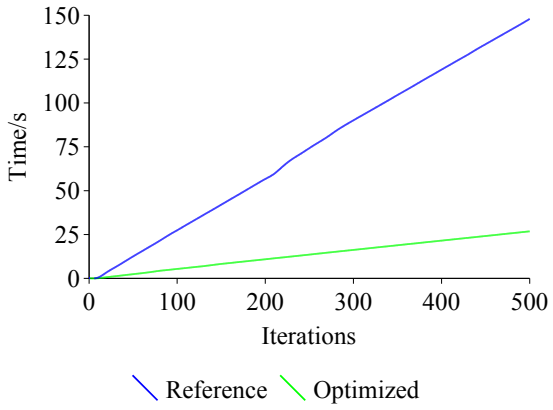

Figure 20: *Calculation times of the first 500 iterations of the reference and optimized implementation of hydraulic erosion.*

Figure 21 shows erosion score averages of the first 500 iterations. Apart from the first 100 iterations, both versions apparently resemble linear functions with the optimized implementation climbing twice as fast as the reference. Due to the realtime requirements, results from more than 100 iterations are irrelevant, but to see how erosion scores eventually develop, a graph showing scores for the first 10,000 iterations can be seen in Figure 22. Here, both curves appear more like logarithmic functions, but where the reference implementation climbs to an erosion score of 2.8 during the 10,000 iterations, the optimized version reaches a score of 7.8. However, at this point none of the height maps resemble the original at all; the optimized implementation reaches its high score by changing the entire height map to a few almost completely levelled terraces divided by very steep slopes.
After 100 iterations, the optimized version scores 20% better than the reference and has a much higher erosion score growth rate during the first 25 iterations where the reference implementation produces almost no increase in erosion score at all.

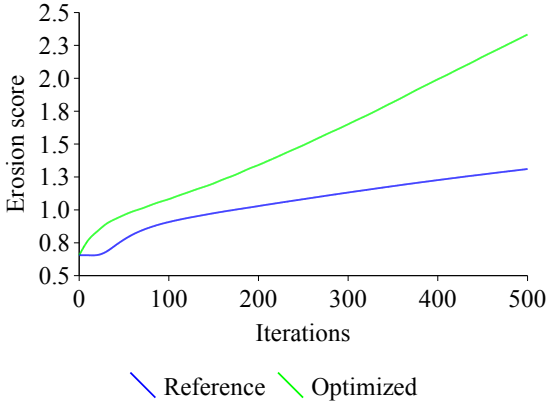

Figure 21: *Erosion scores of the first 500 iterations of the reference and optimized implementation of hydraulic erosion.*

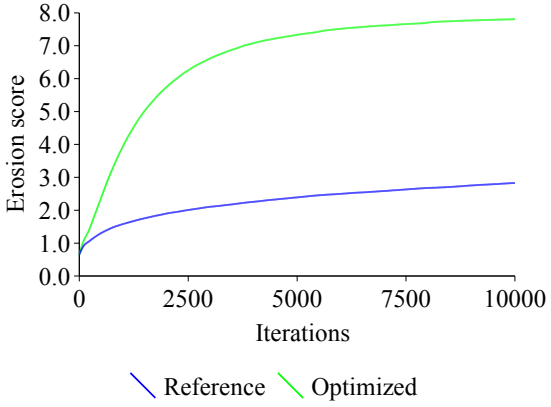

Figure 22: *Erosion scores of the first 10,000 iterations of the reference and optimized implementation of hydraulic erosion.*

Rate of change is shown on the chart in Figure 23 where change per 10 iterations is plotted. As opposed to thermal erosion (see Figure 16), both curves start out at very low values and quickly climb to a maximum about one fourth the rate of

change in thermal erosion. From here, the reference implementation seems to stabilize much faster than the optimized version: After 500 iteration, the rate of change is 3 times higher for the optimized version.
The difference between the outputs of the two implementations after 100 iterations is 1% on average, where a difference of 100% corresponds to a height map with all height values at 0 versus a height map with all height values at 1.

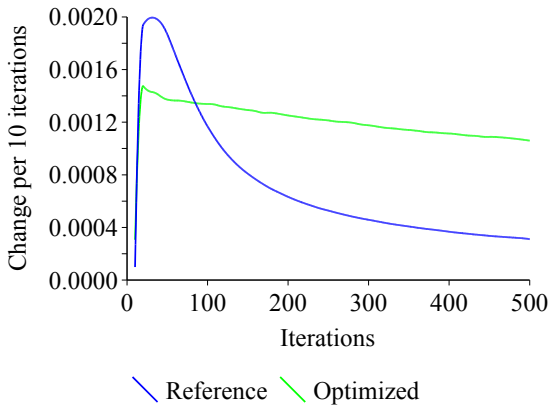

Figure 23: *Change per 10 iterations of the first 500 iterations of the reference and optimized implementation of hydraulic erosion.*

For practical or visual applications, around 100 iterations of both versions are needed to change the terrain to achieve the desired shape. Figure 24 compares height maps produced by the two versions after 100 iterations, and Figure 25 shows rendered views of these height maps.

A new proposed algorithm

With the overall goal of reaching an erosion score as high as possible in as short time as possible, the optimization and analysis of the thermal and hydraulic erosion algorithms have shown two things:

1. Because of its low erosion score, thermal erosion is unsuited for use in computer games where flatness of the terrain is important, but the speed optimized version is fast enough to be used for runtime synthesis of terrain.

2. The effects of hydraulic erosion are much better suited, but even the speed optimized algorithm is too slow to be used for runtime synthesis of terrain.

To create an algorithm that combines the speed of thermal erosion with the high erosion score of hydraulic erosion, it has to be as simple as the speed optimized version of thermal erosion and at the same time emulate what causes the hydraulic erosion to reach a high score. This was achieved by modifying the speed optimized version of thermal erosion as described below.

Overview

The high erosion score of the speed optimized version of hydraulic erosion is caused by two factors:

1. Nearly flat areas tend to be levelled out when the water flowing downhill is dispersed over a wider area and evaporates, leaving its carried sediment to fill up any irregularities. The flattening lowers $\bar{s}$ which in turn increases the erosion score.

2. While the overall height of the terrain remains largely unaffected, the levelling out leaves less area for the remaining slopes, making them steeper. This increases σ_s which in turn increases the erosion score.

Thermal erosion seems to do the opposite of this: Instead of levelling out flat areas and making slopes steeper, it distributes material from steep slopes across flat areas until every $s_{i,j} \leq T$. The new proposed algorithm is simply the same as the speed optimized version of thermal erosion, but with the conditional check

$$if\ (d_{\max} > T)$$

inverted to

$$if\ (d_{\max} \leq T)$$

As shown in the example in Figure 26, this causes slopes steeper than the talus threshold to remain unaffected while flatter areas are levelled out.
Below is a piece of pseudocode describing the central part of the algorithm which is run on every cell each iteration:

$d_{\max} = 0$
for every i in neighbourhood :
    $d_i = h - h_i$
    if $(d_i > d_{\max})$:
        $d_{\max} = d_i$
        $l = i$
if $(0 < d_{\max} \leq T)$:
    $\Delta h = \frac{1}{2} d_{\max}$
    $h = h - \Delta h$
    $h_l = h_l + \Delta h$

The algorithm was found to work well with a Von Neumann neighbourhood and values of T between $\frac{8}{N}$ and $\frac{16}{N}$.

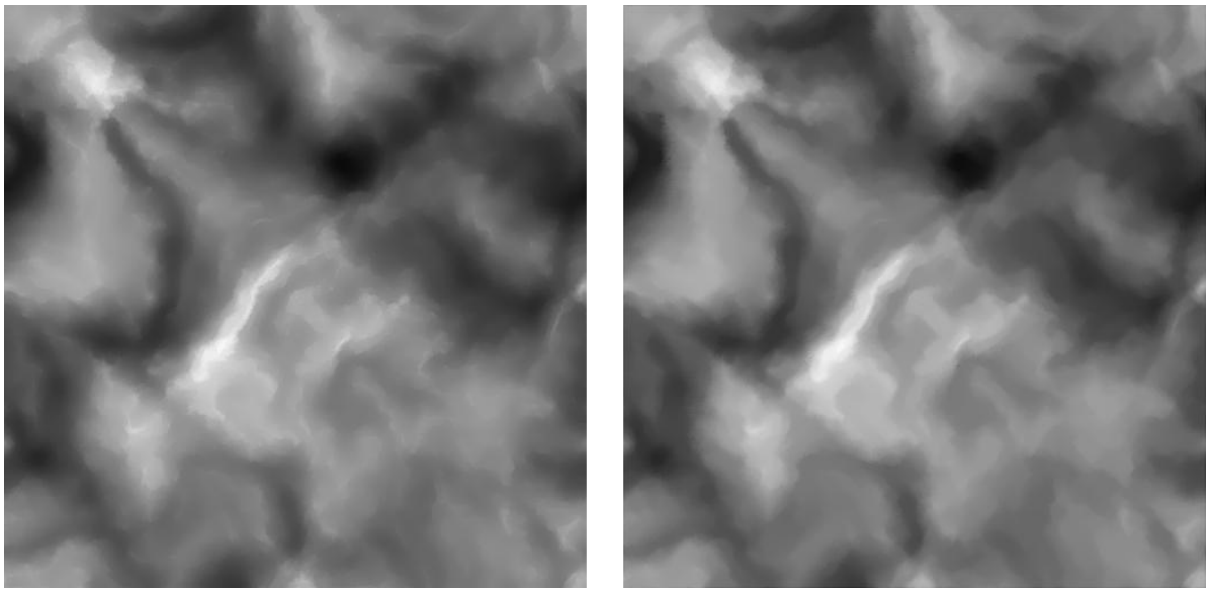

Figure 24: *Comparison between the reference implementation of hydraulic erosion (left) and the speed optimized version (right). Images show the height map after 100 iterations.*

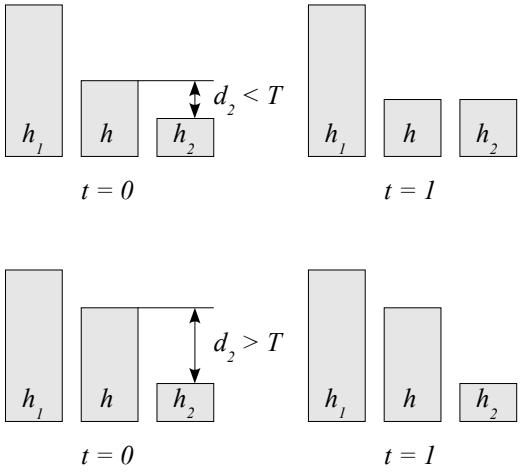

Figure 26: *Two examples of the new proposed erosion algorithm. In the first case, $d_2 < T$ so h and h_2 are levelled out, while in the second case, $d_2 > T$ which leaves both h and h_2 unaffected. In both cases $h_1 > h$ and is therefore ignored.*

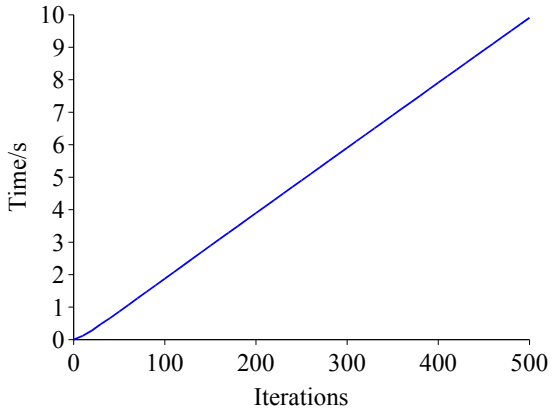

Figure 27: *Calculation times of the first 500 iterations of the new proposed algorithm.*

Analysis

Calculation time averages for the first 500 iterations can be seen in Figure 27. Time required per iteration remains constant and the 500 iterations are done in approximately 10 seconds, matching the speed optimized version of thermal erosion.

Figure 28 depicts erosion score averages for different values of T during the first 500 iterations. For the three sampled values, a proportional relationship between T and erosion score seems to exist. Erosion scores rise quickly: After 50 iterations they reach between 70-80% of the level at 500 iterations.

With $T = \frac{16}{N}$ an erosion score of 2.15 is reached in 50 iterations, corresponding to 1 second of calculation time. For comparison, the speed optimized version of hydraulic erosion does not reach this score until after 450 iterations, corresponding to 24 seconds of calculation time.

The change per 10 iterations graph shown in Figure 29 confirms that most of the change happens during the first 50 iterations after which the algorithm seems to stabilize quickly. Not surprisingly, the rate of change matches that of thermal erosion whereas hydraulic erosion only changes at one fourth the speed during the first 50 it-

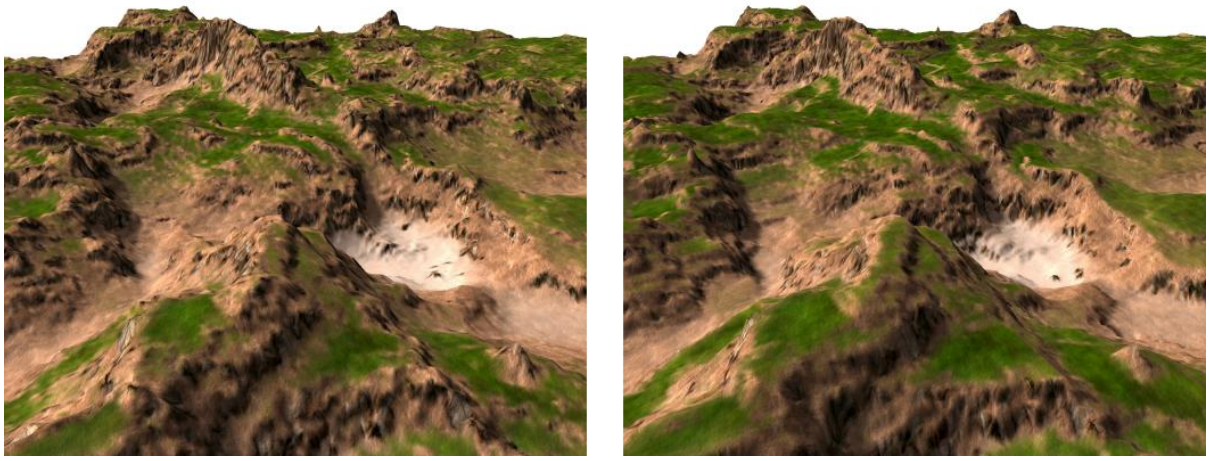

Figure 25: *Renderings from the two height maps shown in Figure 24 - the left image shows the reference implementation after 100 iterations, and the right image shows the speed optimized version after 100 iterations.*

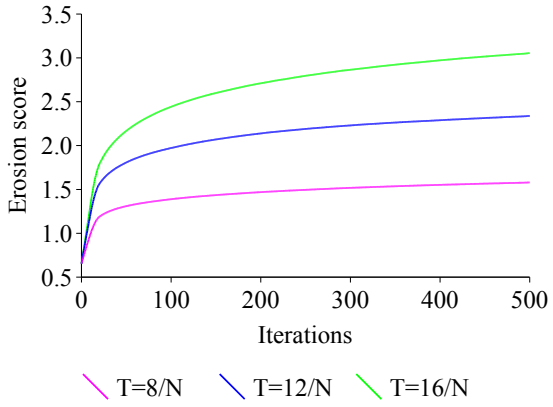

Figure 28: *Erosion scores of the first 500 iterations of the new proposed algorithm.*

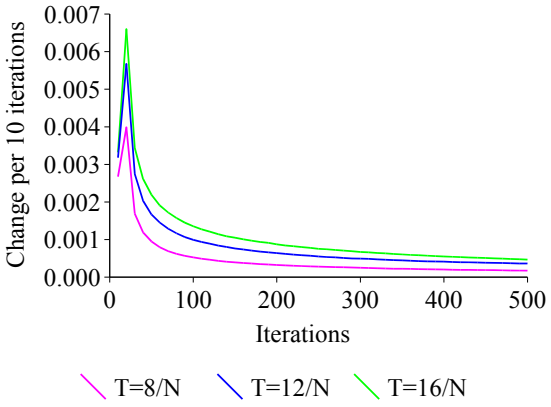

Figure 29: *Change per 10 iterations of the first 500 iterations of the new proposed algorithm.*

erations. Together with the quick initial rise in erosion score, this suggests that the optimal number of iterations for producing profound erosion effects as quickly as possible is around 50.

Height maps after 50 iterations with $T = \frac{8}{N}$ and $T = \frac{16}{N}$ as well as rendered scenes of these, can be seen of Figure 30 and Figure 31 respectively. Larger values of T creates more levelling of the terrain and leaves less but steeper slopes.

With 50 iterations of this algorithm, an eroded terrain meeting the criteria defined at the beginning of this paper can be synthesized from scratch in less than 2 seconds for $N = 512$ (about one quarter of a million height values) and in less than 7 seconds for $N = 1024$ (about one million height values). This actually makes it possible to use these techniques for runtime terrain generation in modern computer games, where loading times up to 30 seconds are not uncommon.

Playability issues

To explore the use of procedurally generated terrain in computer games a bit further, a few playability issues for games in the realtime strategy genre are discussed here. This type of game makes more demands to the terrain shape than just visual appearance, as player units must be able to move around in the terrain, and there must exist a decent number of flat spots of a certain minimum size for various structures to be placed upon.

Main criteria

The above considerations regarding unit movement and building placement can be formalized

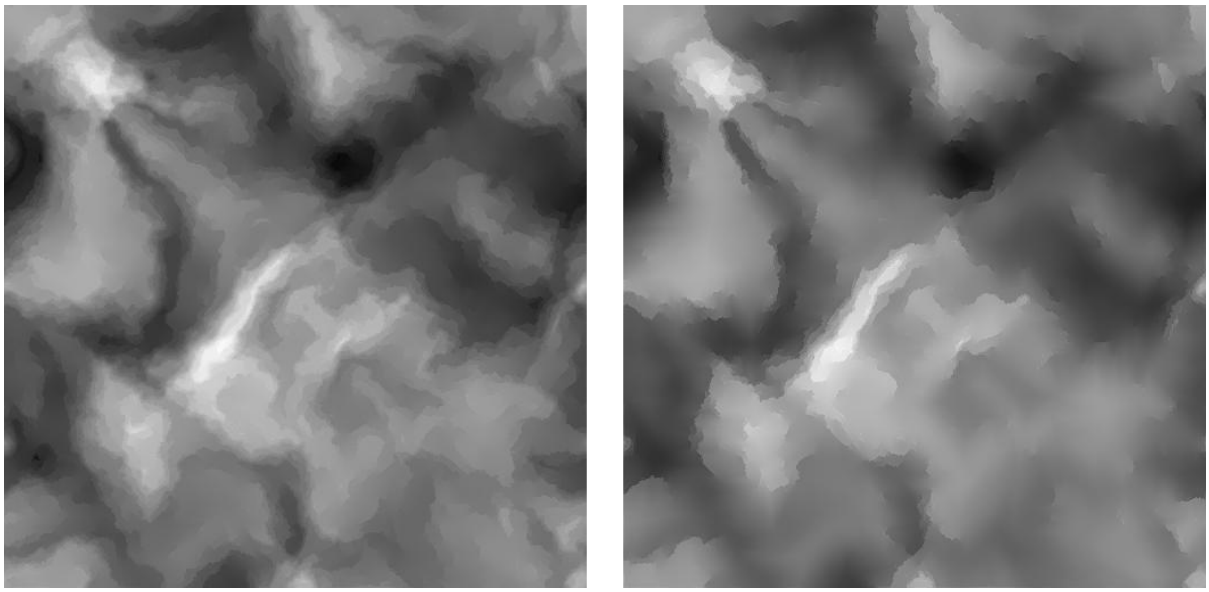

Figure 30: *Comparison between $T = \frac{8}{N}$ (left) and $T = \frac{16}{N}$ (right) for the new proposed algorithm. Images show the height map after 50 iterations.*

in two criteria:

- Cells with an inclination below a certain threshold T_u allowing unit movement should be connected in an area as large as possible.
- Cells with an inclination below a certain threshold $T_b < T_u$ allowing building placement should cover an area as large as possible (although not necessarily a connected area). Only cells included by the first criterion are considered (i.e. a spot suitable for building placement is worthless if it is isolated from the rest of the map).

To analyse height maps according to these criteria, the following binary maps (maps containing either the value 0 or 1) of the same size as the height map in question are defined:

- A, the accessibility map, is a binary map representing the cells $h_{i,j}$ of the height map whose corresponding slope values $s_{i,j} < T_u$ such that $s_{i,j} < T_u \Rightarrow a_{i,j} = 1$.
- U, the unit map, is a binary map containing the largest connected area of cells from the A map whose value is 1.
- F, the flatness map, is a binary map representing the cells $h_{i,j}$ of the height map whose corresponding slope values $s_{i,j} < T_b$ such that $s_{i,j} < T_b \Rightarrow f_{i,j} = 1$.
- B, the building map, is the binary map consisting of all $b_{i,j} = u_{i,j} \times f_{i,j}$.

Since both units and buildings may take up more space than one cell, the variables N_u and N_b are used to denote the side length of the square representing the space occupied by units and buildings respectively. For example, $N_u = 1$ means that a unit only occupies one cell, while $N_b = 5$ means that a building occupies a square of 5×5 cells.
Taking unit and building size into consideration means modifying A before calculating U and B, such that only the cells of value 1 that can form squares of sizes $N_u \times N_u$ remain 1. When calculating U from A, N_u must also be taken into consideration when determining whether two areas are connected, e.g. a unit taking up 3×3 cells cannot pass through a narrow connection only 1 cell wide. An illustrated example of how U is calculated can be seen in Figure 32.
Likewise, B must be modified such that only the cells of value 1 that can form squares of sizes $N_b \times N_b$ remain 1.
A "unit score", υ, and a "building score", β, are then defined as the fraction of the cells of U and B whose value are 1 (the same as the average value of the maps).
Because a completely flat height map (which is undesirable) would maximize υ and β, a final overall "game suitability score" γ is defined as

$$\gamma = \varepsilon \times \upsilon \times \beta$$

i.e. the product of the erosion score, unit score and building score. The terrain must now achieve high unit and building scores while retaining a high erosion score.

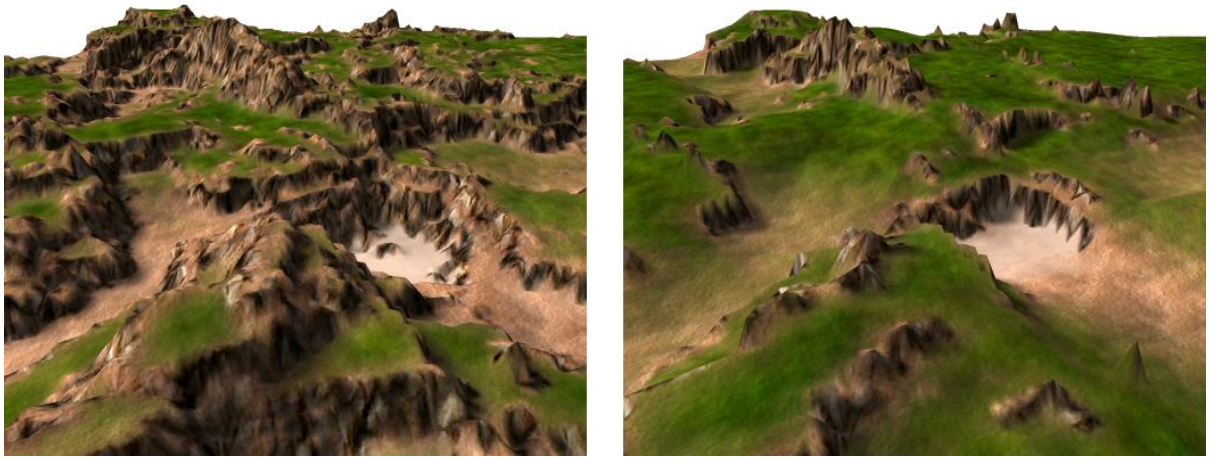

Figure 31: *Rendered views of the two height maps shown in Figure 30. A much larger part of the terrain is levelled out when using* $T = \frac{16}{N}$.

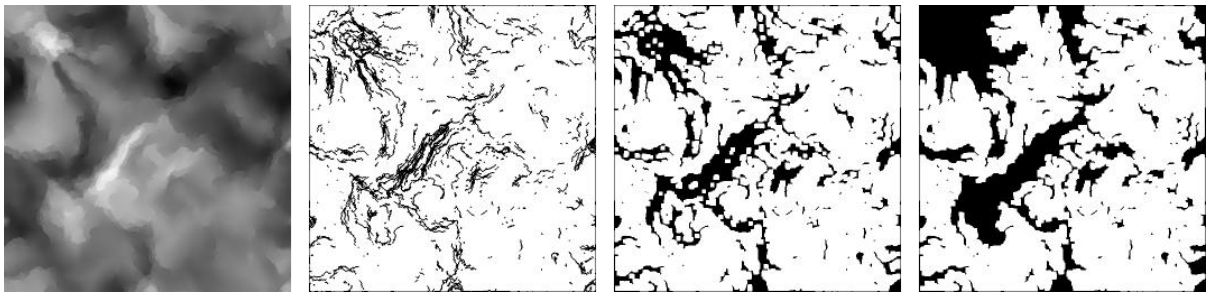

Figure 32: *The construction of the unit map* U. *Step 1 shows an eroded height map* H *and step 2 its corresponding accessibility map* A *based on the threshold value* $T_u = \frac{8}{N}$. *In step 3,* A *is reduced to those cells that can contain squares of size* $N_u \times N_u$ *(where* $N_u = 7$ *in this example). In step 4, the map is finally reduced to* U *which is the largest unit-accessible area taking the unit size* N_u *into consideration.*

To guarantee that every height map generated by random seeds meet certain playability criteria (a unit score $v > 0.75$ for instance) is impossible without some sort of iterated feedback system where the height map is analyzed and changed repeatedly until the desired criterion is met. This however requires far too much work to to be done runtime, so instead the methods for synthesizing terrains must be adjusted until it can be shown statistically that the criteria are met in, say, 99% of the cases.

These adjustments can be done in many ways; one way is to introduce more flat areas during synthesis of the base terrain as described below.

Base terrain modifications

Terrain flatness plays an important role in the evaluation of game suitability, and one very effective way to control the major features of the terrain is through the Voronoi diagram used for the base terrain.

It was noted earlier that the use of the coefficients $c_1 = -1$ and $c_2 = 1$ gave the Voronoi diagram a useful property. The usefulness lies in the fact that it allows for individual control of each of the "hills" of the Voronoi diagram by using a related *domain map*.

A domain map is created by assigning a value to each of the feature points of the Voronoi diagram. When the Voronoi diagram is rendered, the domain map is calculated at the same time by giving each cell the same value as the closest feature point. This creates a number of patches of one colour, whose borders represent the coordinates where $d_1 = d_2$.

The coefficients of the corresponding Voronoi diagram ensure that the values located at the domain borders are always 0, since

$$d_1 = d_2 \Rightarrow -1 \times d_1 + 1 \times d_2 = 0$$

This means that each of the hills are exactly covered by a patch from the domain map; by multiplying the Voronoi diagram with the domain map,

the hills can then be scaled individually without creating sudden jumps in the height values. Figures 33 and 34 show some examples of how this can be done.

The fact that these Voronoi diagrams always have a value of 0 midway between two feature points also makes it possible to widen the passages between the hills. This can easily be done by subtracting a constant from all values and clipping negative values to 0 - see Figure 35 for an example.

Any of these modifications only add a about 10 milliseconds to the time required for the synthesis of the base terrain for $N = 512$ and 40 milliseconds for $N = 1024$.

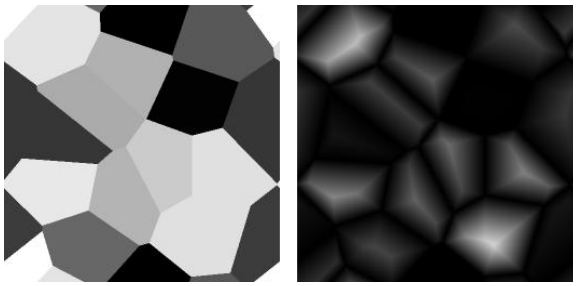

Figure 33: *Multiplying a domain map with random values (left) with the Voronoi diagram, produces hills of varying height (right).*

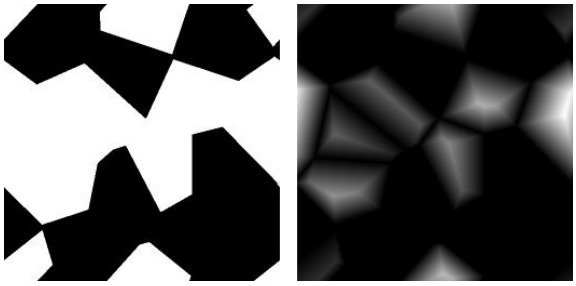

Figure 34: *Multiplying a domain map with randomly assigned values of either 0 or 1 (left) with the Voronoi diagram, produces a mixture of flats and hills (right).*

Analysis

A large number to of test runs were made to determine average game suitability scores for the different kinds of erosion algorithms described in this paper. The constants used were similar to those found in the project described in the

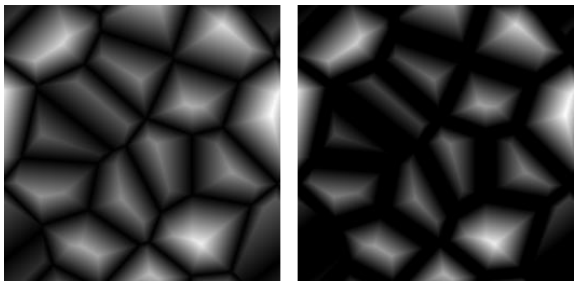

Figure 35: *Subtracting a small amount of the height values (and clipping negative values to 0) of the original Voronoi diagram (left) widens the narrow passages between the hills (right).*

Examples of use section:

$$T_u = \frac{8}{N}$$
$$T_b = \frac{2}{N}$$
$$N_u = 1$$
$$N_b = 9$$

With a height scale of $\frac{N}{16}$, T_u equals an inclination of 26.6 degrees and T_b equals an inclination of 7.1 degrees. A unit takes up only one cell, while a building covers 9×9 cells.

Table 2 show average scores for terrains created by different erosion methods.

Although most of the base terrain is accessible by units, almost no flat areas of sufficient size for building placement exist, resulting in a minimal game suitability score.

Both versions of thermal erosion manage to flatten 98% of the terrain to an inclination below T_u, but the problem with this method is that most of the terrain will attain *exactly* the same inclination as the algorithm's talus threshold $T = \frac{4}{N}$, leaving no areas with an inclination below $T_b = \frac{2}{N}$, which again results in a minimal game suitability score.

The two versions of hydraulic erosion produce slightly better building scores, but the overall game suitability scores are still too low for them to be of any practical use, especially when taking realtime generation into account. More iterations would have given them higher scores, but the calculation work would become too demanding. An interesting observation is that while the reference and speed optimized version have similar erosion and unit scores, the optimized version seems to produce more almost completely flat areas.

The new algorithm is very sensitive to the value of T. With $T = \frac{8}{N}$, it produces so many slopes with an inclination above T_u that the terrains often are broken up into several large areas inaccessible from each other, causing a very low unit score.

Terrain type	ε	υ	β	γ
Base terrain	0.665	0.713	0.002	0.001
Reference thermal erosion, 50 iterations	0.614	0.980	0.002	0.001
Optimized thermal erosion, 50 iterations	0.626	0.983	0.002	0.001
Reference hydraulic erosion, 100 iterations	0.916	0.834	0.050	0.039
Optimized hydraulic erosion, 100 iterations	1.077	0.828	0.157	0.140
New algorithm, 50 iterations, $T = 8/N$	1.339	0.303	0.173	0.070
New algorithm, 50 iterations, $T = 12/N$	1.850	0.752	0.393	0.580
New algorithm, 50 iterations, $T = 16/N$	2.204	0.905	0.434	0.877
New algorithm, 50 iterations, $T = 20/N$	2.340	0.949	0.429	0.953

Table 2: *Erosion, unit, building and game suitability scores for terrains created by different erosion methods.*

Test runs with $T = \frac{12}{N}$ and $T = \frac{16}{N}$ produce much better results: Between 0.75 and 0.90 on unit scores and around 0.4 on building scores.

Individual scores approximate normal distributions pretty well, so with standard deviations of 0.23 and 0.20 repsectively, it is safe to assume that for $T = \frac{12}{N}$, 97.5% of all terrains reach a game suitability score of at least 0.120 and for $T = \frac{16}{N}$, 97.5% of all terrains reach a game suitability score of at least 0.477.

Although $T = \frac{20}{N}$ produce even better overall scores (97.5% of all terrains reach a game suitability score of at least 0.585), $T = \frac{16}{N}$ was chosen as the optimum value because it scores better in the critical building score and because the terrains themselves have a much better *visual* appearance.

Table 3 shows scores for various changes to the Voronoi diagram used in the synthesis of the base terrain as described above. All examples use 50 iterations of the new erosion algorithm using $T = \frac{16}{N}$.

Both tested methods (removing a percentage of the hills as shown in Figure 34 or reducing the hills by subtracting a constant value from the Voronoi diagrams and clipping negative values to 0 as shown in Figure 35) increase overall game suitability scores as the effects are increased. The greatest impact is seen on the building scores, since both methods cause more completely flat areas in the Voronoi diagram.

Overall, changing the Voronoi diagram component of the synthesized base terrain makes it possible to boost the game suitability score a further 25% without removing the hills added by the Voronoi diagram completely.

Applying these modifications will most likely deduct from the visual appearance of the terrain as it will become more flat and featureless, but taken together with adjustments of the threshold value T and the number of iterations of the new proposed algorithm, the terrains can easily be fine-tuned to meet defined criteria for unit movement and building placement.

Examples of use

The methods shown in this paper for creating game-suitable natural looking eroded terrain in realtime are used in the realtime strategy game Tribal Trouble[1] by Oddlabs[2]. Tribal Trouble is the first game of its kind to feature a nearly unlimited amount of different terrains by creating them runtime from a random seed number and a few adjustable parameters (among these the amount of Voronoi hills). Figure 36 shows a screenshot of how these terrains look when shaped into an island and having some vegetation added.

The game uses height maps of varying size up to 512×512, meaning that terrain base synthesis and erosion is done in 2 seconds on a standard PC. For the terrain creation, procedural techniques are also used to synthesize 7 different surface textures and apply these to the terrain according to its features such as height, slope and relative height (height values compared to the average of their surrounding neighbours within a certain radius). The game uses the same values of N_u, N_b, T_u and T_b as those used in the analysis of playability issues. U and B maps as described in this analysis are used for unit pathfinding and placing of buildings - see Figures 37 and 38.

Summary

This paper took a new angle on the synthesis of eroded fractal terrain, namely from a computer game development perspective. With computer games of the realtime strategy genre and similar types in mind, a set of desirable traits were condensed into an erosion score that, together with

[1] http://tribaltrouble.com
[2] http://oddlabs.com

Terrain type	ε	υ	β	γ
75% hill coverage	2.280	0.916	0.487	1.017
50% hill coverage	2.337	0.923	0.526	1.134
25% hill coverage	2.371	0.929	0.556	1.225
Vertical offset -0.1 with clipping	2.256	0.910	0.468	0.962
Vertical offset -0.2 with clipping	2.312	0.918	0.511	1.086
Vertical offset -0.3 with clipping	2.346	0.922	0.539	1.166
Vertical offset -0.4 with clipping	2.361	0.926	0.557	1.219
Vertical offset -0.5 with clipping	2.372	0.928	0.567	1.249

Table 3: *Erosion, unit, building and game suitability scores for terrains using different modifications to the Voronoi diagram used in the synthesis of the base terrain.*

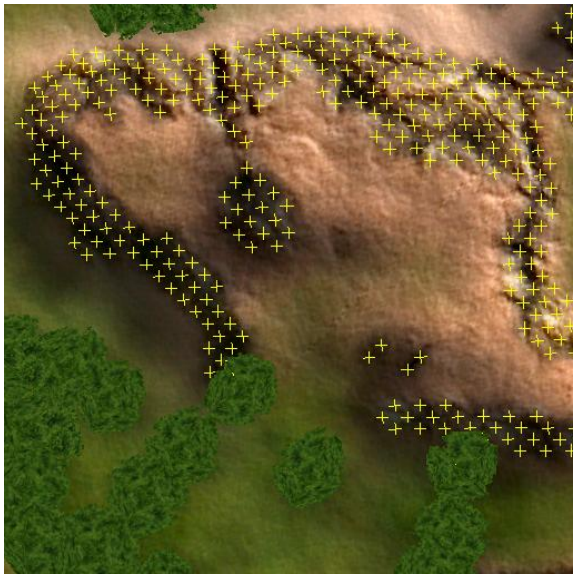

Figure 37: *Screenshot from Tribal Trouble showing the use of the unit map U. The yellow crosses indicate inaccessible cells of the height map, i.e. cells whose inclination are above T_u.*

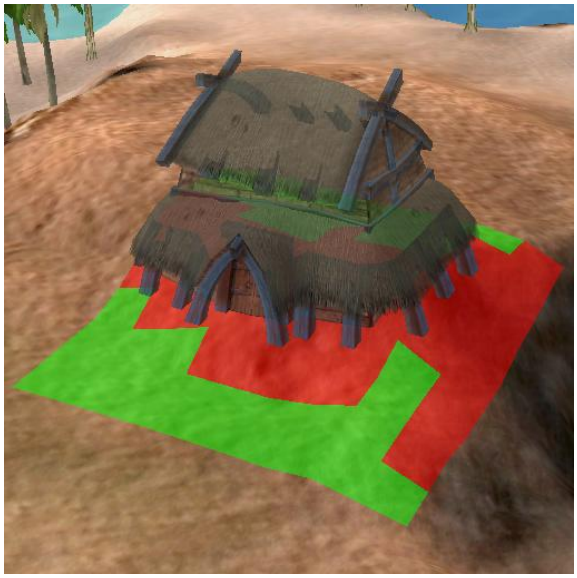

Figure 38: *Screenshot from Tribal Trouble showing the use of the building map B. When placing a building, all cells must be green, i.e. none of the cells covered can have an inclination above T_b.*

the strict realtime requirement, was used to evaluate methods of terrain synthesis and implementations of two of the classical erosion algorithms. For base terrain synthesis, it was shown that more natural-looking terrains than just plain $1/f$ noise could be synthesized in less than 1 second for 512×512 height maps, and in less than 3 seconds for 1024×1024 height maps on a standard PC.

Not even with speed optimizations (at the cost of physical correctness) that reduced calculation times to between one fifth and one sixth, did any of the classical erosion algorithms meet the realtime requirement while producing the desired output: The speed optimized version of thermal erosion was found to be fast enough, but scored much too low in evaluation. The output of hydraulic erosion scored a lot better, but even the speed optimized version was too slow.

A new algorithm using the same simple method as the speed optimized version of thermal erosion, but modified to change the terrain in another way, was introduced. When evaluated, the algorithm was able to triple the terrain's erosion score in 1 second for 512×512 height maps and 4 seconds for 1024×1024 height maps, making it much more suitable for games.

Finally, two important demands to the terrain used in realtime strategy games were discussed. Methods for evaluating terrains according to these criteria were introduced, and it was shown that the methods for evaluation together with modifications to the synthesis of the base terrain could be used to fine-tune the terrains to meet playability criteria.

Figure 36: *Screenshot from Tribal Trouble, a realtime strategy game using the methods described in this paper for fast runtime generation of terrains.*

References

[1] F. Kenton Musgrave, Craig E. Kolb and Robert S. Mace: The Synthesis and Rendering of Eroded Fractal Terrains. *Computer Graphics*, Volume 23, Number 3, July 1989, pages 41-50.

[2] Bedřich Beneš and Rafael Forsbach: Visual Simulation of Hydraulic Erosion. *Department of Computer Science, ITESM, Campus Ciudad de México.*

[3] Eng-Kiat Koh and D. D. Hearn: Fast Generation and Surface Structuring Methods for Terrain and Other Natural Phenomena. *Eurographics*, Volume 11 (1992), Number 3.

[4] Alan Fournier, Don Fussell and Loren Carpenter: Computer Rendering of Stochastic Models. *Communications of the ACM*, Volume 25, Issue 6 (June 1982).

[5] Nicoletta Sala, Silvia Metzeltin and Massimo Sala: Applications of Mathematics in The Real World: Territory And Landscape. *University of Italian Switzerland*, 2002.

[6] David S. Ebert, F. Kenton Musgrave, Darwyn Peachey, Ken Perlin and Steven Worley: Texturing and Modeling: A Procedural Approach (Third Edition). *Morgan Kaufmann Publishers*, 2003.

[7] N. Chiba, K. Muraoka and K. Fujita: An Erosion Model Based on Velocity Fields for the Visual Simulation of Mountain Scenery. *The Journal of Visualization and Computer Animation*, Volume 9, 1998, pages 185-194.

[8] Kenji Nagashima: Computer generation of eroded valley and mountain terrains. *The Visual Computer*, 1997, pages 13:456-464.

[9] D. D'Ambrosio, S. Di Gregorio, S. Gabriele and R. Gaudio: A Cellular Automata Model for Soil Erosion by Water. *Department of Mathematics, University of Calabria, Italy.*

[10] Bedřich Beneš and Rafael Forsbach: Layered Data Representation for Visual Simulation of Terrain Erosion. *Department of Computer Science, ITESM, Campus Ciudad de México.*

References [1]-[5] and [7]-[10] as well as this paper can be downloaded from `http://oddlabs.com/jo/terrain/`