Terrain Synthesis from Crude Heightmaps

Alexandre Philippe Mangra
Technical University of Cluj-Napoca
Str. G. Bariţiu 28, 400027, Cluj-Napoca, România
alexandre.mangra@gmail.com

Adrian Sabou
Technical University of Cluj-Napoca
Str. G. Bariţiu 28, 400027, Cluj-Napoca, România
adrian.sabou@cs.utcluj.ro

Dorian Gorgan
Technical University of Cluj-Napoca
Str. G. Bariţiu 28, 400027, Cluj-Napoca, România
dorian.gorgan@cs.utcluj.ro

ABSTRACT
This paper presents an approach to terrain synthesis from minimal-detail user-provided heightmaps. There is no assumption regarding the level of detail provided, in order to allow users without access to powerful heightmap tools and/or resources to generate useable terrain based on a self-provided crude feature plan. We present the issues stemming from a lack of detail in user input, notably sharp altitude increases and oversimplified feature edges, and proceed to elaborate on using the terrain synthesis algorithm to solve the issues and create a level of detail that more closely resembles realistic terrain models. The algorithm pipeline is presented and parametrized to show how the user can influence the resulting model.

Author Keywords
Terrain synthesis; Heightmap; Worley noise; Perlin noise; Filters.

ACM Classification Keywords
H.5.m. Information interfaces and presentation (e.g., HCI): Miscellaneous.

General Terms
Algorithms; Terrain models.

INTRODUCTION

Over the past decades, computational power has become less expensive and more powerful thanks to technological advances. Alongside it, computer generated imagery (CGI) increased in availability and potential. CGI is one of the mainstays of technology, used in various fields like video games, movies, art, simulation software and anywhere else image generation is beneficial. One of the reasons for its popularity is the artistic freedom it entails, coupled with the potential to mimic something that does not exist in the real world.

When compared to physical props and background, computer generated imagery becomes evidently advantageous. There is a large number of images unable to be accurately reproduced without the use of computers, be it spaceships, hellish creatures, otherworldly plants or simply vast, expanding landscapes. Creating a quality physical replica would incur costs unreasonable for any budget, not to mention unfeasible if we're considering the entire landscape of an alien planet. A virtual reproduction's costs can easily be quantified in the artist's and/or programmer's work and the required hardware.

Thus, the industry's needs have fueled the development of an array of algorithms and generation software tailored specifically at creating this kind of models. Among them, terrain generation is one of the most used fields due to it contributing significantly at reducing production costs of backgrounds.

The terrain model can be created procedurally (using a set of rules) or based on a set of given input data. The latter is usually combined further with algorithms to refine the given data and produce something usable. Purely procedural terrain suffers from restricting the user control over the final location of terrain features like mountains, hills, plains, rivers or islands. On the other side of the spectrum, some synthesis algorithms working with input data such as heightmaps, feature graphs or guides require at least part of the data to be highly specific. This can prove inconvenient to the casual user, forcing him to spend increasing amounts of time researching the software used and finding ways of creating the necessary input.

The casual user, then, raises new issues when trying to create terrain with specific features. He will, in most cases, be unable to properly form input data relevant for the application that would lead up to the desired terrain model. It can become tedious and time-consuming to master a new skill or software in order to obtain decent results.

One issue will be the sudden altitude increases caused by the user creating the input heightmap by hand. Painting the heightmap with only a handful of colors or grayscale values leads to the creation of a layered terrain which does not conform to reality nor has any kind of transition between layers, hence the sharp, perfectly vertical, altitude changes.

Another issue is the lack of detail on such models. The layman will have neither the time nor experience to paint "rough" edges, as seen in nature at the delimitation of two differently elevated areas. There is a high probability of encountering very uniform edges, if not downright straight, thus breaking the illusion of natural, chaotic, form.

This paper elaborates on a simple algorithm which tries to solve these issues by detailing very crude input to a point where it becomes usable, either as the final terrain model or as a more precise input heightmap for more complex algorithms.

RELATED WORKS

A variety of techniques for procedural or user-guided generation already exists, due to the high demand of terrain generation in the movies and video games industries. As information becomes increasingly available, more and more people try to expand the horizons by either improving existing methods or finding new ones. Terrain synthesis is one of those expanding domains, with entire companies being built around terrain generation software and a growing number of research papers detailing new algorithms.

One such example is World Machine Software, LLC and their sole product: World Machine 1. An immensely powerful terrain generation software which allows users to create terrain from scratch. This is done by layering algorithms and directing data through a pipeline formed by the user. It also supports user-guided generation, by allowing the input for algorithms to be provided through external files. At first glance, however, it does not offer ways to process low-detail input. Elevation discrepancies remain sorely visible throughout the processing pipeline. A person trying to control the features will be unable to do so unless he or she invests enough time in learning how to use this complex software. If such procedures exist, they are unintuitive at best.

A case should be made for Gaia 2, procedural terrain software created by Procedural Worlds, which, among other capabilities, allows users to define where they want certain features to be placed by inserting specialized markers called “stamps”. This greatly alleviates the input issues but restricts the user to the set of available stamps (currently over 150) as an advantageous tradeoff between control and power. The only downside is its reliance to the Unity game engine since it is provided as a Unity “asset”, a plug-in of sorts.

Aside from terrain synthesis software, the number of papers detailing new and experimental algorithms for synthesis is on the rise. The focus is on giving as much power as possible to the user, creating new ways of synthesizing terrain from different input data-sets. Somewhat unsurprisingly, the tendency is to reach for improved reproduction quality. To give the end-user the power to remake relief forms based on certain patterns and to do so at the best quality level possible.

For example, one of the more well-known papers on the topic is the work of Zhou et al. 3, describing an algorithm to map a relief style onto a simplistic user-provided sketch. As long as the user finds a heightmap describing the desired relief shape and pattern, he can utilize the algorithm to great results. This only partially solves the issue of low-detail user guidance. Firstly because only one type of pattern can be applied at a time, preventing, for instance, both a mountain range and a river or lake to be mapped in the same map instance. Secondly, because obtaining such patterns may or may not prove difficult, depending on what the user intends to obtain and the available patterns on the internet. These problems arise, of course, because of the high specialization of the algorithm and are perfectly acceptable in the context of the goal set for this procedure.

Another such work is that of Cruz et al. 4, with an objective similar to that of Zhou et al.: user-guided terrain synthesis. This paper focuses on having an input graph besides the simplistic sketch, called “guide” here. They try to create geomorphically correct terrain from a collection of real-world data. Very similar in both scope and surfacing issues to the previously presented work: it requires information the layman may not immediately have available and it becomes hard to model several terrain features at once.

In the quest for improving the obtained terrain, most researchers specialize their work, leaving the inexperienced user dead in the water. Even when a software product implements something with general availability, the learning curve is almost never shallow. Large amounts of time must be invested for the average user to obtain usable results from most of today’s software implementations.

There are two main types of generation algorithms: guided and unguided. Unguided algorithms are easy to use and are controlled by input parameters, but it is hard to restrict the output to certain desired features (i.e. placement of mountains/ rivers/ plateaus). Guided algorithms exist and most of them try to use either existing real-world data 3, 4 or place markers on the spots where certain features are to be created 2. All these methods demand auxiliary input from the user, which may be hard to obtain.

USER-GUIDED TERRAIN SYNTHESIS

While procedurally generating terrain has plenty of advantages, such as speed of generation, variety and realistic detailing, the main drawback is the lack of user involvement in the placement of terrain features. This makes it hard to create something specific and which conforms to the user’s requirements.

As described in the previous section, this gave birth to a series of algorithms and software which do exactly that: create terrain based on a set of specific user input data. They give more freedom to affect the end product and allow one to model the shape of the terrain based on their own wishes. Artists and designers gain tremendous power by being able to create terrain in drawing that is then converted to a highly-realistic 3D model. Researchers and other technical-oriented people gain an equal amount of power by being able to convert data obtained from the real world to create incredible virtual replicas.

For the hobbyist, however, or any other inexperienced user the challenge becomes much greater. One has no use for powerful tools if they require large time investments to master. Furthermore, there is a definite possibility that said user is not interested in highly-detailed or geomorphically correct terrain. The main interest point is the creation of a terrain model simulacrum that abides by the user’s requirements. Most of the people interested in terrain synthesis will not be artists, capable of creating detailed heightmaps to provide to the software nor

experienced enough to find the other resources needed as input, such as real-world data, formatted in a way which the software expects.

Following is the algorithm proposed to solve this problem by interpreting low-detail heightmaps and synthesizing a terrain model that, while not necessarily accurate from a realistic point of view, meets the requirements set by the user through the input heightmap and places the terrain features where they are expected. It is assumed that an input is provided in the form of a crudely-drawn heightmap, lacking detail. To generate realistic terrain, at this stage of development, plus the issue of needing a collection of patches, explained earlier in this paper, may be hard to obtain by the inexperienced user.

Edge smoothing

The first step is to solve two issues in the same pass. The issues being:

Sharp elevation level transitions

The neophyte user will provide a heightmap where one elevation level ends and another beings with a drastic difference in value/height. Best example would be the user wanting a mountain surrounded by sea and drawing with a high value in an area of very low values. This will cause a vertical drop (value change of 100%) between the value level represented by the "mountain" (white) and the one of the "sea" (black), as can be seen in Figure 1. Going straight from perfect gray to white or black is also not a good use-case, since the value switches by 50% of the total. A lesser but still perfectly vertical drop.

Figure 1. Sharp transition example.

Simplistic edge definitions

The layman will not have the art skills or appropriate resources to paint better edges. He or she will resort to basic straight or curved lines as shape delimiters, also exemplified on Figure 1.

Both of these problems can be solved by a single, well-chosen algorithm which creates a transitory area between levels and breaks the edges up in a rougher contour. In this case it is a custom-made algorithm inspired by Worley noise 5.

Solution

The first step consists of scattering seed points randomly but evenly across the surface of the heightmap, taking the equivalent height values from the input. I. e. if point X's location is above a black pixel, its value will be 0.0. If it's above a white pixel, its value will be 1.0. The number of seed points is proportional to the number of pixels in the heightmap and can be adjusted for different end results.

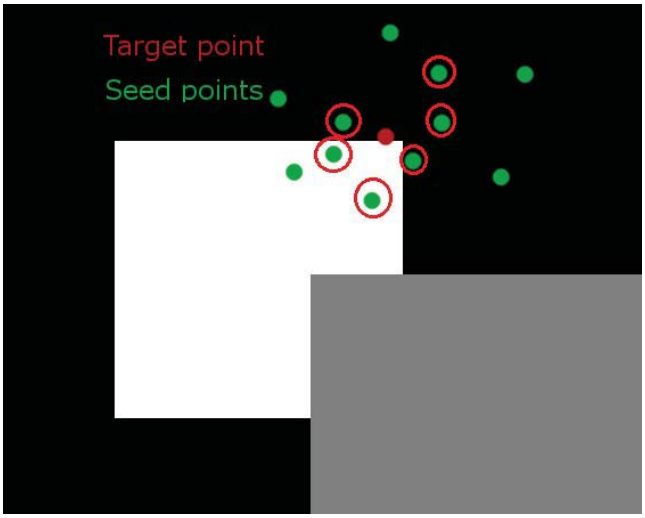

Figure 2. Seed points around a fixed point.

The second step is parsing the entire mesh and adjusting the heights of points based on the nearest N seed points as a linear interpolation of their assigned height values using the distance between the affected point and the seed as a weight. This differs from classical Worley noise, where only the N-th closest point is considered in the rendering function. Increasing N enlarges the area which affects the mesh point, meaning the transitional area between altitudes becomes wider. Exemplified in Figure 2. Circled are the seed points used in computing the new point's height when N = 6. The target point will be affected by 4 perfectly black seeds and 2 perfectly white ones, meaning it will end up closer to, but not perfectly, black.

The randomness of the seed point location creates the rough, natural edges that users expect to see and permits infinite variations on the exact contour when randomly redistributing the seed points again: at one time, the mesh point is affected by 6 low-height points and 3 great-height points then, during another run, the same point is affected by only 2 low-height points and 7 great-height points because in the new seed distribution, the positions change and, hence, distances are altered.

The issue of sharp elevation transitions is solved by the linear interpolation between neighboring seed points by creating transition areas which are equally random in appearance, even if this detail is less noticeable.

Adding detail

After creating transitory areas at the edges, the model is left with large expanses of flat terrain. This happens because all throughout the same level of value, encompassing seed points will all have the same value, hence linear interpolation produces that value repeatedly.

At this point, another procedural generation algorithm can be used and overlaid on top of the model in order to create rough detail across the flat areas. We have chosen

Perlin noise 6 for this purpose, which is a homogenous noise implementation created by Ken Perlin in the early '80s. The fact that this noise is homogenous means that any two neighboring points have close values and create a pleasing, flowing aspect, unlike true random noise.

The noise will be added as a small increase in height across the entire terrain model. This means it will affect both the large areas of flat terrain and the previously created transitory ones. This will improve the aspect of the terrain and make it more palatable for the human eye. As a side-effect, it adds randomness throughout the model, increasing reusability and the diversity of potential outputs. However, since the detailing is small compared to the overall scale of the terrain, this remark is not of such great importance.

We should add that this step may be replaced by another way of imprinting a more realistic texture to the terrain. The caveat is that one should take care not to add complexity to the user interaction, like needing a secondary input, such as a realistic texture for imprinting upon the model or an extended number of added parameters.

Detail smoothing

After applying the Perlin noise as a means for detailing the terrain model, the shape of the model needs to be smoothened to eliminate any kind of sharp peaks that may occur near the edges. This step also helps make the terrain more pleasing to the eye.

Odd peaks and shapes

This phenomenon may happen because as Perlin is applied uniformly across the mesh, it also affects the slopes previously created. The points on these slopes will be displaced and sometimes the displacement goes against the desired shape, i.e. a point will increase in height whilst it would be aesthetically pleasing to remain fixed or decrease in height.

Digital filters – image processing

The chosen solution to the previous problem is to run the whole model through a digital noise reduction filter. This will effectively remove any "noise" which, in this case, is represented by those seemingly random shapes.

A median or mean filter 7, 8 with a 3x3 kernel is perfectly reasonable to solve this issue and any other oddities the terrain model may show. It is applied to the entire model. One run through should suffice, since over-applying a filter will reduce the level of detail, counter to the initial purpose of this algorithm. Likewise, care should be exercised with more powerful filters, some of which will strip too much detail even with a single pass.

After the completion of this step, one should be left with a reasonably detailed terrain model which respects the initial feature placement requirements provided by the user through a crudely-drawn heightmap. Needless to say, this algorithm will work just as well with more complex input, meaning it is suitable for the entire range of possible heightmap detail.

IMPLEMENTATION

The Unity game engine

For implementing and testing, the Unity 9 game engine was chosen because of its existing rendering engine and ease of programming using self-contained scripts. All steps have been converted into C# scripts and linked together.

The algorithm is implemented using operations on a float value matrix representing the terrain model then said matrix is applied onto the heights of a mesh, effectively rendering the result onscreen.

The following testing section has been fully realized using the Unity implementation. Due to mesh restrictions, the size of the samples has been reduced to under or at 128x128 pixels.

TESTING

For the purposes of testing, only the aesthetics of the final terrain model have been taken into consideration. Completely ignoring the performance aspect, since it is reliant on implementation, the testing focuses on confirming that the before-stated issues are solved and that the final model is at least partially resembling a natural form of terrain. The three heightmaps used for testing are: an overly simplistic, a slightly detailed, and a very detailed one. The first two were made by hand, the third one is sampled from the Internet 10; presented in Figure 3.

Figure 3. Left: Simple Heightmap; Right: Detailed Heightmap; Bottom: Complex Heightmap.

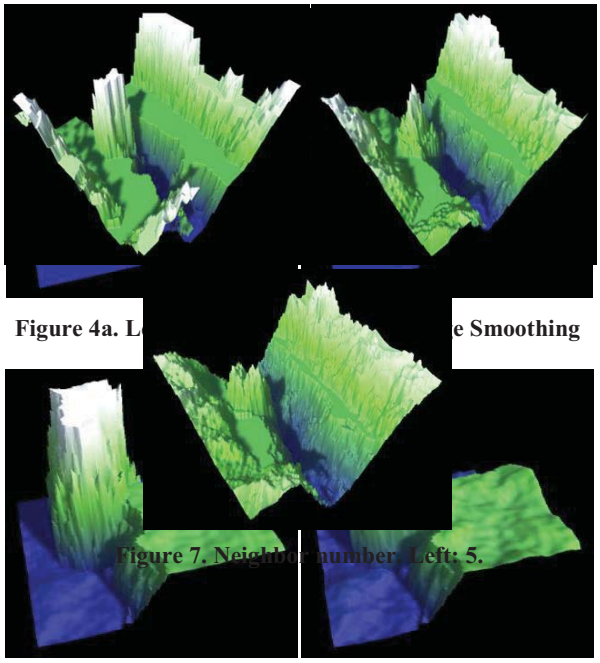

Figure 4a. L e Smoothing

Figure 7. Neighbor number. Left: 5.

Figure 4b. Left: Perlin Noise; Right: Filtering.

Validating result

Initial testing was done to prove the algorithm does indeed end with a detailed model of plausible terrain. It bears mentioning again that the end goal was not realistic terrain. Instead, it was to create a level of detail that more closely resembles realistic terrain models. Any sufficiently detailed model which may pass for terrain is good enough for confirmation. Figures 4a and 4b show how the model advances from its initial state to the final, more detailed output. Figure 4b also showcases visual artefacts (sharp edges, noticeable on the "mountain" edge) remaining from previous steps and how they are eliminated through filtering.

Figures 5 shows the effect on more detailed heightmaps. The result is proof that when confronted with *too much* detail, the algorithm overrides part of it with its own edge smoothing.

Varying parameters for edge smoothing

These tests have been done to empirically find reasonable value ranges for the number of nearest seed points and the total number of seed points by altering one of them and keeping the other constant.

Number of seed points

This number dictates how many seed points in total are scattered throughout the plane. The number is related to the total number of pixels available (height x width). We shall call this number TotalPx. As we grow the number of seeds, the area of influence for each point in the model decreases, as more seeds are found in its direct vicinity.

This preserves more of the initial detail of the image, counter to what edge smoothing is supposed to do. On the other hand, too few seeds mean that the terrain will no longer respect all the details provided. There may be

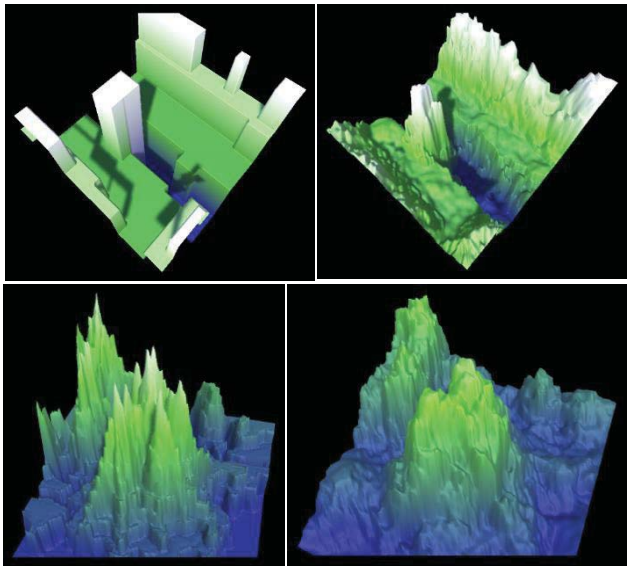

Figure 5. Original and final model. Top: Detailed and Bottom: Complex Heightmap.

entire areas uncovered by seeds and, thus, initial detail is not preserved enough. Figure 6 presents a succession of models, showcasing this effect.

Number of nearest seed points

Testing the number of nearest N points taken into consideration is also a worthwhile experiment, to showcase how different values affect the outcome. For this test, the number of seed points is TotalPx/25. Results empirically show that as the number of neighbors decreases, detail fidelity increases, up to a point where the desired smoothing effect is cancelled. When there are too many neighbors taken into consideration, the smoothing is too strong and the entire map becomes flattened. Figure 7 shows this effect.

Conclusion

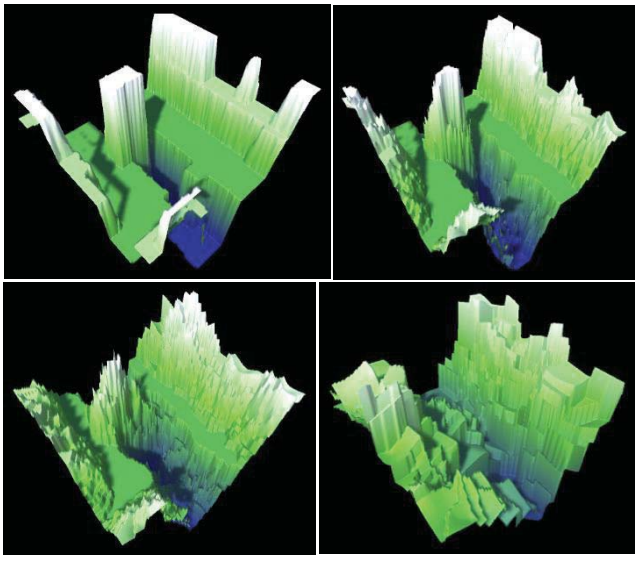

Figure 6. Seed Point Number. Top-left: TotalPx; Top-right: TotalPx/20; Bottom-left: TotalPx/50; Bottom-right: TotalPx/250.

While the number of seed points and neighboring seeds affect each other too, empirical results point to the area centered in TotalPx/20 -> TotalPx/25 seed points total and considering around 10-15 neighbors. This should provide acceptable results for most use-cases. Of course, this does not prevent one to experiment and find proper values depending on the given input heightmap.

CONCLUSIONS

In the world of procedural generation, terrain synthesis is one of the most common uses, allowing for inexpensive yet complex backgrounds in movies, video games, simulation software and other possible areas of interest. The rising demand for such algorithms has given birth to a vast array of advances in this field, ranging from pure optimization to hybrid algorithms and brand-new ones designed to bring a wealth of detail into the final model.

While specialized software is constantly trying to simplify the interface and make procedural terrain generation available to the layman, it must always make compromises regarding input detail versus output detail.

Trying to detail incomplete or crude heightmaps is something few people are trying to elaborate on since the focus is on the end product – a realistic terrain model – and all inputs are usually simply mirroring the demands for the algorithm instead of the other way around.

This paper presented a procedural generation algorithm that is supposed to work with minimal input detail. It outputs something aesthetically close to real terrain models, even if it lacks any kind of groundbreaking detailing. A case can be made for using this algorithm as a preliminary for other systems, detailing crude heightmaps to a level acceptable for more advanced synthesis software and / or algorithms.

While not overly complex, this algorithm proved that there is hope for terrain synthesis from input of any detailing level and that one may still discover new techniques for allowing inexperienced users to generate beautiful scenery with minimal effort.

Future work may involve adding more steps towards creating true geomorphically correct terrain. Possibly erosion algorithms or storing a knowledge base of real-world heightmap information. Testing other initial steps, replacing the randomly seeded linear interpolation and experimenting with the results is another direction of future study worthy for consideration.

REFERENCES

1. Discover, World Machine Software, LLC, [Online]. Available: www.world-machine.com/about.php?page= features.
2. Introducing Gaia, Procedural Worlds, 2015. [Online]. Available: www.procedural-worlds.com/gaia/.
3. Zhou, A. H., Sun, J., Turk, G. and Rehg, J. M. Terrain synthesis from digital elevation models, IEEE Transactions on Visualization and Computer Graphics, 2007.
4. Cruz, L., Ganacim, F., Lucio, D., Velho, L. and de Figueiredo, L. H. *Exemplar-based Terrain Synthesis*, 2013.
5. Worley, S. A Cellular Texture Basis Function. *In SIGGRAPH '96 Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, 1996.
6. Perlin, K. Making Noise, 2002. [Online]. Available: www.noisemachine.com/ talk1/.
7. Fisher, R., Perkins, S., Walker, A. and Wolfart, E. Spatial filters - Median filter, 2003. [Online]. Available: homepages.inf.ed.ac.uk/rbf/HIPR2/median.htm.
8. Fisher, R., Perkins, S., Walker, A. and Wolfart, E. Spatial filters - Mean filter, 2003. [Online]. Available: homepages.inf.ed.ac.uk/rbf/HIPR2/mean.htm.
9. Unity - Game Engine, Unity Technologies, [Online]. Available: Unity Technologies.
10. Bisen, From Heightmap to Worldspace in Skyrim, Hoddminir, [Online]. Available: hoddminir.blogspot.ro/ 2012/02/from-heightmap-to-worldspace-in-skyrim.html.