Real-time Guided Procedural Terrain Generation

Sima Vlad Grigore
Technical University of Cluj-Napoca
Str. G. Bariţiu 28, 400027, Cluj-Napoca, România
simavlad_10@yahoo.com

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

ABSTRACT
This paper presents an application that allows real-time large terrain generation using crude user input regarding the terrain's height features. Our technique uses a preprocessing phase to generate plausible terrain out of crude bitmaps and achieves real-time performances by splitting the large terrain into chunks and processing and displaying only the chunks surrounding the user, while maintaining the height features suggested in the original crude input.

Author Keywords
Terrain synthesis; Real-time; 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
Creating and modeling realist environments, has become over the last decades one of the most disputed and discussed aspect in computer graphics. Given the fast evolution that technology is facing these days, and the significant hardware progress in graphics, more and more detailed terrains are developed using a procedural approach. Since 1982 when it found its first commercial use in the well-known Star Track series by generating alien planets, the concept of creating content based on algorithms or rules, has faced a rapid and impressive growth, being used in large and diverse areas nowadays.

Procedural content generation (PCG) represents the process of creating content with limited or absent implication from the user. This doesn't always mean that the user won't be able to affect in anyway the result of the generation part, but that this process will mainly relay on the randomness of the algorithms involved. Based on the stage in which it is created, we can distinguish two different types of procedural content generation. The first one is offline PCG, and it's called this way because all the content is generated during the development stage and the result comes with all the features already implemented. This is a method that is very safe regarding faults, and has a lower demand for the algorithm because, in both cases, all the errors or the results that don't fit a particular expected model can be corrected by hand.

The second type of PCG is the online version, which as the name suggests, involves creating terrain at startup, also referred to as real-time generation of content. Despite the fact that in this scenario the algorithms are far more complex, it can offer various opportunities for generating infinite terrain, creating an unforgettable experience for the users.

A common agreement in the field of PCG is that neither the procedural nor the model base approach fulfill the needs of the user in what concerns a realistic representation of the environment. On one hand, using solely a procedural approach will leave the terrain with uncharacteristic features and, on the other hand, by employing the model-based approach, terrain lacks realism. In order to solved this drawback, a compromise is made between these two possible approaches, leaving many windows of opportunities for creating suitable content.

The same approach is utilized in this paper, relaying on both the procedural method and the model based one in order to obtain realistic output that follows the user's input. In the context of our application, using only the user's input would provide a terrain that lacks detail and, as there are no pre-conditions for the level of detail for the initial bitmap, there might be unrealistic transitions between adjacent zones that is not normal in a natural environment. Using the other approach, a purely procedural one, would result in a terrain which might not meet the user's expectations and also might developed uncharacteristic features.

RELATED WORKS
Olsen [1] created real time terrain, but the paper was not fully focused on performance and how to obtain it, but rather it was focused on presenting ways of using and improving algorithms for erosion. His real-time generation has a loading time, which can be an acceptable way of developing a real-time application, as many games nowadays have an allocated loading time, in which they generate some pre-required data or just fetch them from memory. Our application behaves in a similar way, in order to generate the features points. In what concerns the creation of terrain Olsen used a *1/n* noise. Then he combined this with Voronoi diagrams, which is similar to

Worley noise which we used as a base algorithm to develop a suitable one for our needs. Lastly, he used improved thermal erosion, modified to simulate the properties of hydraulic erosion.

Génevaux et al [2] generated content using the hydrological erosion, and using a hydrology-based method in order to represent it. The construction of the algorithm starts by using the input provided by the user. The input is composed of the following elements: terrain outline, river mouths and other parts of the river. The aim of the algorithm is to form a complete river network, kept under the form of a graph, starting from some basic elements and sketch provided by the user. When the algorithm is completed, blocks such as junctions, springs, deltas, and river trajectories are created, and later used to help rendering the final terrain.

Mangra et al [3] present 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. They 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.

Zhou et al [4] present an example-based system for terrain synthesis. In their approach, patches from a sample terrain (represented by a height field) are used to generate a new terrain. The synthesis is guided by a user-sketched feature map that specifies where terrain features occur in the resulting synthetic terrain. Both the example height field and user's sketch map are analyzed using a technique from the field of geomorphology. The system finds patches from the example data that match the features found in the user's sketch. Patches are joined together using graph cuts and Poisson editing. The order in which patches are placed in the synthesized terrain is determined by breadth-first traversal of a feature tree and this generates improved results over standard raster-scan placement orders. Their technique supports user-controlled terrain synthesis in a wide variety of styles, based upon the visual richness of real-world terrain data.

Schneider et al [5] interactively synthesize artificial terrains using procedural descriptions. They present a new GPU method for real-time editing, synthesis, and rendering of infinite landscapes exhibiting a wide range of geological structures, building upon the concept of projected grids to achieve near-optimal sampling of the landscape. They describe the integration of procedural shaders for multifractals into this approach, and propose intuitive options to edit the shape of the resulting terrain. The method is multi-scale and adaptive in nature, and it has been extended towards infinite and spherical domains. In combination with geo-typical textures that automatically adapt to the shape being synthesized, a powerful method for the creation and rendering of realistic landscapes is presented.

REAL-TIME GUIDED TERRAIN GENERATION

While purely procedural terrains have a tremendous amount of detail over terrains generated based on models, they still lack the control the user can have over the final result of the process. In order to overcome this issue, this paper proposes a simple but efficient algorithm that would help the user control the major elements from the terrain he wants to create, by being able to offer as input a crude grayscale bitmap where he can mark the main elements from the landscape.

Regarding our application, we can divide it into two main parts that are going to be discuss later on, in this paper. First, we focus on the procedural part that will contain three important steps: edge smoothing, adding detail using a type of noise and then the process of smoothing the noise created in the previous step. The second part of this paper focuses on generating terrain in real time by creating new chunks of terrain as the user moves thought the scene. This part will apply the steps described in the first one, and as the user moves, new chucks of terrain will be generated, while the last ones will be delete from the scene, all of these steps being controlled by a predefined range or radius.

Procedural Terrain Generation

We are going to explain step by step each of the aforementioned steps, starting with edge smoothing. This step is required because, as specified earlier, the user input has not restrictions in what concerns the level of detail it must provide. So, there is quite a chance that the inexperienced user might provide a black image with a white square in the middle. The problem here is that the transition between the plain represented by the black color and the mountains represented by the white color, would be a 100% height transition and it would look extremely unnatural. In order to avoid this, we use Worley Noise [6] as a source of inspiration, an algorithm designed to help smooth the boundaries (edges) of the input bitmap. The idea of the algorithm is to generate uniformly distributed random features points all over the map. In order to ensure this will happen, the initial bitmap was split into equal grids and, in each grid, a particular number of seed points was generated. This way we ensure that they are distributed uniformly on the surface. Then, in order to compute the actual height value of the current pixel, we compute a weighted sum of values from the closest N features points. Thus, each of the feature points will contribute with a percent directly proportional with the distance between it and the point whose value is being computed.

The second phase is represented by the addition of detail to the equalized bitmap. This step is absolutely necessary

because the user's input might not provide enough details for the map to be relevant and look like a real environment. As presented in the last step, if the user decides to introduce a crude bitmap with only two colors and some random geometrical figure, then the terrain will look unnaturally flat. To overcome this limitation, a type of coherent noise must be used in order to obtain suitable results that also respect the users input. For this task, we picked Perlin Noise [7], a type of coherent noise capable of adding new and significant details. Perlin Noise uses a gradient approach, by computing gradient vectors in each corner of the square, if we refer to the 2D version, or cube, if we refer to the 3D version, then uses linear interpolation between these gradient vectors in order to compute the height value of the current point.

Last, but not least, the step that we are going to discuss is the smoothing we will apply to the height map that was generated so far, by using two different types of digital filters, the mean and median filters [8][9]. This step is extremely important, as Perlin Noise might have made some unwanted changes with a particular point, meaning it could have assigned it a greater value than expected or a lower one. In this case the value of the point will be recomputed based on his neighbors' heights by using these two types of filters. For the mean filter, we use a kernel of size 3x3, computing the mean of all surrounding neighbors' height values as the current height. The other type of digital filter used is the median filter, which sorts neighbors and then picks the value that lays at the center of the interval. After this step is completed the height map is ready to be rendered, and this happens by applying it to multiple terrain game object.

Real time guided generation

The second step of our method is represented by the real-time generation, which involves the creation of new terrain as the user moves though the environment. The algorithm works as follows: it places the user in the scene's origin and it crops 9 chunks from the input provided by the user, transforming them into terrain later. The main idea is to use a tile base approach, by utilizing 3x3 matrix of chunks, placing the user in the middle and updating it as the user moves thought the scene. The initial position in the input provided by the user is in the center of the crude bitmap. In order to achieve this functionality, we use the center of the image as a pointer, that will be later on updated based on users' decisions.

Then, on the chunks that were just taken from the bitmap, we apply the first step of the algorithm in order to create a viable height map that still respects the input that the user provided. At this point we have a height map that represents part of the terrain that the user wanted. Now, as this application offers the possibility for the user to experiment real time generation, if the user decides to move in a random direction on the X and Z axes, the application will intercept this. It uses a function that checks in each frame the current position of the user and, if the user travels a certain distance from the starting point, the pointer that indicates the user's position in the bitmap will be updated using a predefined value and taking into account the direction in which the user chose to walk. As mentioned in the previous sections, the application is not meant to generate infinite terrain, the size of the result being directly influenced by the size of the bitmap given by the user. When the user reaches the end of the environment, it means that the user has reached the edge of the input he provided so the application won't generate any more terrain and the user will have to either move in another direction or get stuck as a collision detection will be implemented to prevent the user from falling,

EXPERIMENTS

Validating results

Initial testing was done to prove that the algorithm does indeed successfully provide a detailed model of plausible terrain. Our real-time generation algorithm was applied both using a pre-refined heightmap and a crude user created bitmap to guide it.

Figure 1 shows the result when applying our algorithm using a pre-refined heightmap as a source. The red square represents the area of terrain that is generated around the user's position and the screen captures illustrate what the user sees when pointing north, south and west. As the user moves around the scene, the generated landscape will change accordingly, but still preserve all the height characteristics in the heightmap.

Guided terrain generation

Since one key aspect of this application is the possibility to guide the algorithm using crude user generated bitmaps, the second test case was built around such input. Figure 2 shows the resulting terrain when applying our technique using a crude bitmap as a source. Again, the red square represents the area of terrain that is generated around the user's position and the screen captures illustrate what the user sees when pointing north, south and west. As the user moves around the scene, the generated landscape will change accordingly, but still preserve all the height characteristics suggested in the original crude bitmap source.

Performance considerations

Performance is directly connected with the size of the image, because the feature points algorithm divides the image into smaller grids of size 4x4 and generates a fixed number of random samples that are later used to compute the height values for each pixel. The experiments made were with a 128x128 size pixel image, followed by 256x256 and we even used a 512x512 image to test the output data. Regarding processing time, when using a 128x128 image there is no distinguishable delay introduced by the process. When using a bigger size as the input, there

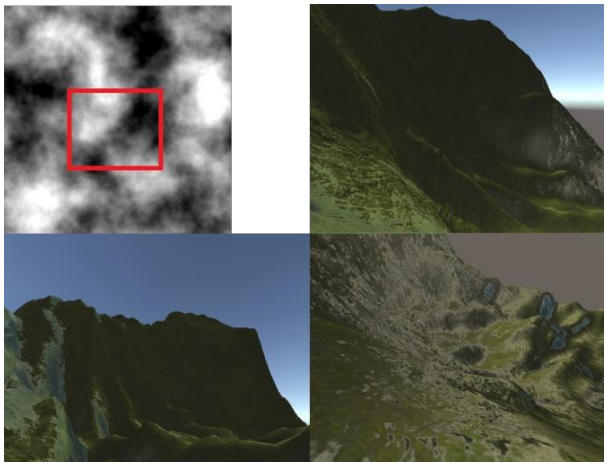

Figure 1. Terrain generation using a pre-refined heightmap as a source

is a slightly delay when employing a 256x256 image size, while using a 512x512 image size would bring a delay equal to 10-15 seconds at most. The results are acceptable as this part represents a pre-processing stage for the real-time generating method.

The second aspect when assessing the application's performances is to experiment various sizes of a chunk. In the current version of the application there is a 64x64 dimension size, so each time user reaches a certain distance, a new chunk of this size is generated in the direction of the user's movement. In this current implementation, it runs smoothly and without a visible delay when generating new chunks of terrain. Growing the dimension of the kernel used will definitely affect the run time of the algorithm. A size of 128x128 or 256x256 would not bring catastrophic results but thinking of a 1024x1024 or even bigger size would definitely introduce a delay that would affect the user's experience.

CONCLUSION

As mentioned before, the purpose of this paper was to describe the functionality of an application that aims to create procedural terrain in real-time, being guided by the user's input. The user's input has no restriction in what concerns the level of detail, so a preprocessing algorithm had to be applied in order to obtain a viable height map used to create the mesh that would be rendered later. The real-time generation algorithm works by splitting the mesh into chunks and processing and displaying only those around the user's position. Future developments include porting the real-time computation to the GPU in order to improve performance.

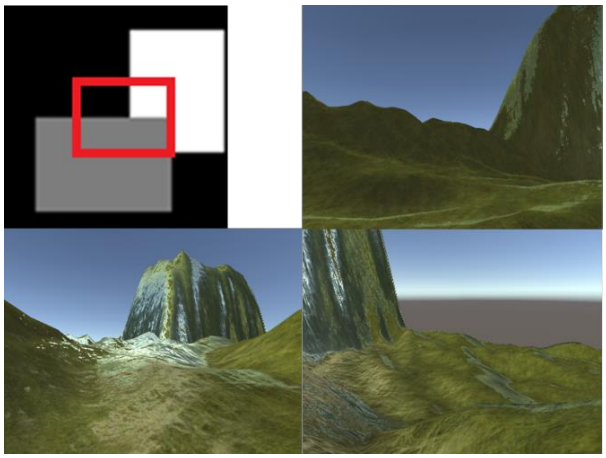

Figure 2. Terrain generation using a crude bitmap as a source

REFERENCES

1. Jacob Olsen. Realtime procedural terrain generation - realtime synthesis of eroded fractal terrain for use in computer games, (2004).
2. Jean-David Génevaux, Éric Galin, Eric Guérin, Adrien Peytavie, and Bedrich Benes. 2013. Terrain generation using procedural models based on hydrology. ACM Trans. Graph. 32, 4, Article 143 (July 2013), 13 pages.
3. Alexandre Philippe Mangra, Adrian Sabou, Dorian Gorgan. Terrain Synthesis from Crude Heightmaps. *RoCHI 2016,* 113-118
4. H. Zhou, J. Sun, G. Turk and J. M. Rehg. Terrain Synthesis from Digital Elevation Models. *IEEE Transactions on Visualization and Computer Graphics 13*, 4 (2007), 834-848.
5. J. Schneider, T. Boldte, R. Westermann. Real-Time Editing, Synthesis, and Rendering of Infinite Landscapes on GPUs. *Conference on Vision, Modeling, and Visualization (VMV)*, (2006)
6. S. Worley. A Cellular Texture Basis Function. *SIGGRAPH '96 Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, (1996)
7. K. Perlin. Making Noise. (2002). www.noisemachine.com/talk1/
8. R. Fisher, S. Perkins, A. Walker and E. Wolfart. Spatial filters - Median filter. (2003) http://homepages.inf.ed.ac.uk/rbf/HIPR2/median.htm
9. R. Fisher, S. Perkins, A. Walker and E. Wolfart. Spatial filters - Mean filter. (2003) http://homepages.inf.ed.ac.uk/rbf/HIPR2/mean.htm