Generative Technology: Multi-Layering Perlin Noise Textures in Terrain Generation

CS4991 Capstone Report, 2023

John Zoscak
Computer Science
The University of Virginia
School of Engineering and Applied Science
Charlottesville Virginia, USA
jmz9sad@virginia.edu

ABSTRACT

Modern terrain generation algorithms have limited use in artwork and games because of their lack of realism and inability to produce photorealism in a creative way at the same level a human designer can. A possible solution in this space involves the layering of deterministic noise generators in a way that is more creative, artistic, and gives users greater levels of control. I used Golang (for learning and speed reasons) and an OpenGL bindings library for generating a GUI. The resulting terrains were more complex and visually appealing. Furthermore, the nature of the program enables layers of customizability. In the future, the program can be translated to use C++ with DirectX, and the object structure can be reorganized to facilitate custom configurable texture objects that can be added to structures for aggregation into terrains in an intuitive way. This change will increase the user accessibility of the code, will increase speed by lowering the overhead, and will adhere to current trends in programming for the gaming industry.

1. INTRODUCTION

Automatic terrain generation is a computer software tool that has its uses mainly in the generation of scenery and landscapes in various entertainment mediums. It has been used to generate landscapes and backdrops in movies and in open-world video games. It has also entered the space of landscape generation for the purpose of simulated engineering testing. Furthermore, the field of deterministic noise generation has been used in some less apparent fields, such as animation effects and dynamic flows on application widgets.

The goal of automatic terrain generation is to leverage deterministic random noise tools, as well as other deterministically modellable phenomena in real-world terrains. Different methods will ultimately produce different results in terms of smoothness, realism, and consistency in the final terrain produced by the software. When it comes to different applications, each method of generation typically has different use cases because of the characteristics of the generated results.

In my work, I explore the solution of generating terrains by a vector sum of 2-D textures. By utilizing a simple terrain generation methodology, I was able to see that linearly generated progressive terrains can be processed in parallel during progressive generation phases of terrain generating software. By utilizing matrix algebra, the terrains can be treated as textures with variable contribution to final terrains. Furthermore, I was able to demonstrate that this is a viable solution for giving users greater control over the final terrain.

2. RELATED WORKS

Studying Perlin noise generation first sparked my interest in doing research on terrain generation. As a reference for evaluating my hypothesis, I utilized technical descriptions of the algorithm. These sources gave algorithmic and programmatic descriptions of the underlying Perlin noise algorithms, therefore supporting my evaluations and conclusions drawn from the project (Perlin Noise, 2023).

One of the most relevant advancements in terrain generation includes the use of erosion simulation. By leveraging the behavior of real phenomena in the algorithms, the results have been shown to carve realistic terrains surrounding water features. (Lim, et al., 2022; Génevaux, et al., 2013) Despite this, there are unimplemented elements of creativity, control, and entropy that could make software of this kind more artistic and interesting in certain cases.

Regarding the combinational use of terrain generation algorithms, I referenced very similar work that approached terrain generation with multi-layer erosion simulations. This well-documented project used a multi-layer approach by having many underlying erosion simulations and various erosion processes stacked on top of one another (McDonald, 2022). My work draws upon similar ideas but emphasizes the isolation of layers more than this work.

3. PROJECT DESIGN

My project design is simple: GUI related functions exist in separate code spaces as generating software. The program can generate simple Perlin noise terrains, and what I deem "bipartite" Perlin noise terrains.

3.1 Review of System Architecture

The terrain generation code lies inside of the generate.go file. This contains the code that defines the single and multilayer structures, as well as all the vertex buffer objects (VBOs) that describe the shape of the triangles to be rendered. Furthermore, code for progressively generating terrain according to requested movements lies in generate.go. The main. go file handles unpacking serialized generation parameters, interpreting the terrain types, then initializing the essential structures from generate.go and setting up the user interface (UI). The serialized terrains are stored as json files.

During the "initialization" phase, the terrain is first generated, the UI, and geometry to be rendered are set up. In the "progressive generation" phase, the user is free to explore the terrain by changing the window of the terrain that is generated and loaded into the rendered point VBO in real-time.

3.2 Requirements

The goal of the project was to visualize terrains, progressively generate them in real-time according to requested displacements, and to experiment with the effects of vector summation of various textures.

3.3. System Limitations

Because all arithmetic in graphics is done with float32, some terrain parameters should be set to multiples of 2 to avoid problems with lossy division that may cause incorrect VBO indexing when preparing the triangles that are rendered.

3.4 Specifications

The current implementation can generate simple terrains and "bipartite" terrains. Simple terrains include a Perlin noise texture, whereas bipartite terrains include two Perlin noise textures that are vector summed. In bipartite terrains, textures have coefficients which define their level of contribution to the final terrain. They also have gradient widths and heights which are important for changing the granularity of the Perlin noise generated. Granularity specification is not necessary in traditional Perlin noise generation, but because there is more than one texture that contributes to the terrain, the granularity of each texture is important in relation to the other.

3.5 Challenges

Determining the best methods for optimization proved to be difficult. It is computationally infeasible to store multiple textures and have a separate vector summation. This results in redundancy where already rendered spaces are recalculated during progressive generation phases of the application's lifecycle.

3.6 Solutions

Redundancy is solved by keeping only the minimum number of VBOs required. This includes the point VBO which describes the terrain shape, and the index VBO which describes how triangles are to be rendered between the points. The point VBO stores only the final aggregated terrain. During progressive generation phases of processing, this is optimized because it eliminates recomputing points that already existed in the rendered window. Similarly, during the progressive generation phase, when processing the textures and computing the aggregated terrain, each layer is computed in-line and in-place. The solution reduces the memory space that is used and helps reduce the redundancy of the program.

4. RESULTS

The final program can render terrains very quickly. My laptop initialized and rendered the terrain in less than four seconds. Additionally, my hypothesis that more complex and visually appealing terrains could be produced by a vector sum of linearly-generated progressive terrains was correct. Shown below, as Figures 1 and 2, are two simple Perlin noise terrains. Below them is a bipartite terrain, Figure 3, generated by my software, which treats them as textures to be aggregated into one terrain.

This demonstrates that well-tuned parameters for textures can be subsequently aggregated in a way that produces a more visually appealing and custom terrain result. Furthermore, by processing the texture layers in-line with each other, the final terrain is initialized and progressively generated with the same big-oh runtime and theoretical memory use as the simpler terrains. This indicates that the solution of generation is viable and scalable for applications where performance matters.

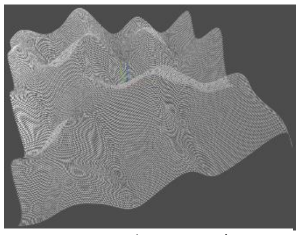
Figure 1: Simple Terrain (macro granularity)

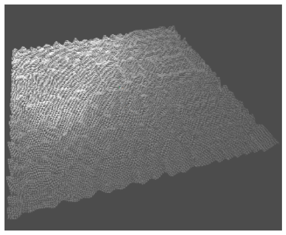
Figure 2: Simple Terrain (micro granularity)

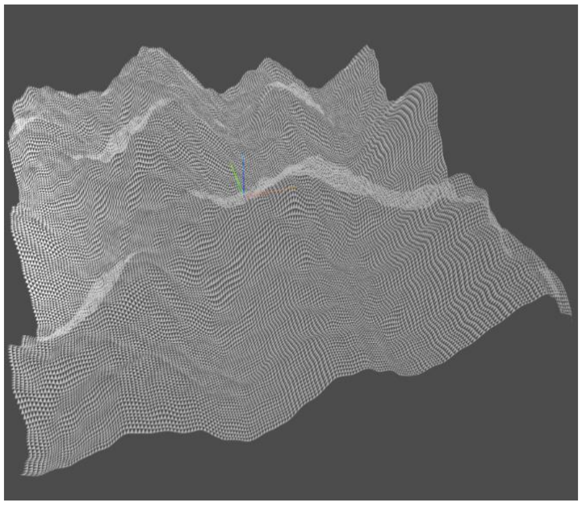
Figure 3: Bipartite Terrain (Vector sum of macro and micro terrains as textures)

5. CONCLUSION

Terrain generation is relevant in a wide range of fields for a variety of problem sets. The solutions that my program puts forth are designed to emphasize greater control in work done for those fields. Specifically, my work is targeted towards the gaming industry and artwork. However, it does present a solution that can be extended to use humanmade custom textures, which can be used for simulations and testing in many fields.

The research effectively demonstrates a solution for improved terrain complexity, separation of macro and micro textures, and presents combinational use of independent algorithms as viable in final terrain generation. These solutions can help to enable better procedural terrains, and can also be utilized for purposeful customization, while still emphasizing real-time computer-generation as a tool.

Doing research in this field and on this technology has enhanced my understanding of computer graphics. With a more technical understanding of computer graphics, I feel that I can provide insight into various modelling software optimizations that I could not have done before.

6. **FUTURE WORK**

The technical solution that I have provided in my research is not viable for distribution for a couple of reasons: It does not provide a framework for adding external textures, it is written in Golang, (which is not optimal for exporting it to gaming systems) it uses OpenGL (whereas DirectX is more common), and it renders terrains using geometries instead of more relevant technologies in terrain generation like triangle strips. If these upgrades are implemented, then my code could be more easily used in external software and could be directly interacted with more easily.

The most important future upgrade for the system would be the framework for adding external textures. This change would involve changing the generate.go file to handle inline calculations by iteratively calculating texture computations from a data structure that holds the generation parameters of each texture added to the terrain. This functionality would result in the big-oh runtime of generation to be a function of the number of textures. Furthermore, there would need to be changes that enable the selection of multiple serialized texture parameters.

The second most important change would be the use of triangle strips instead of geometries. This would improve the speed of the render by only requiring triangles within a certain distance of the camera source to be rendered. As an extension of this, the real-time generation can be further optimized by increasing its size beyond that seen in the triangle strip render. Seamless chunk rendering can be integrated with this feature. By separating the window rendered from what is calculated, there is not always a need to shift the point VBO when there is a change in the requested window during progressive generation. This would improve real-time generation efficiency by reducing the amount of memory manipulation.

REFERENCES

Perlin noise. (2023). In *Wikipedia*. https://en.wikipedia.org/w/index.php?title=Perlin_noise&oldid=1148235423

Lim, F. Y., Tan, Y. W., & Bhojan, A. (2022). Visually Improved Erosion Algorithm for the Procedural Generation of Tile-based Terrain. *Proceedings of the 17th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*, 49–59. https://doi.org/10.5220/0010799700003124

McDonald, Nick. (2022). An Efficient Data Structure for 3D Multi-Layer Terrain and Erosion Simulation – *Nick's Blog*. (n.d.). Retrieved April 27, 2023, from https://nickmcd.me/2022/04/15/soilmachine/

Génevaux, J.-D., Galin, É., Guérin, E., Peytavie, A., & Benes, B. (2013). Terrain generation using procedural models based on hydrology. *ACM Transactions on Graphics*, *32*(4), 1–13. https://doi.org/10.1145/2461912.2461996