Noise Modeler: An Interactive Editor and Library for Procedural Terrains via Continuous Generation and Compilation of GPU Shaders

Johan K. Helsing, Anne C. Elster

Noise Modeler: An Interactive Editor and Library for Procedural Terrains via Continuous Generation and Compilation of GPU Shaders

Johan K. Helsing and Anne C. Elster

Norwegian University of Science and Technology (NTNU), Trondheim, Norway
johanhelsing@gmail.com, and elster@ntnu.no

Abstract. In online procedural generation, content is generated as the game is running on the consumers computer. Our GPU-based Noise Modeler composites noise and other functions through a flow-graph editor similar to the ones used by procedural shader editors and offline terrain generators. Our framework enables non-programmers to edit models for procedural terrain while observing the effect of changes immediately in a real-time preview. Each time a change is made to the model, a corresponding GLSL shader function is automatically generated. The shader is then compiled, and used to render a real-time terrain preview.

Keywords: Online terrain generation, noise synthesis, real-time procedural content generation, stochastic implicit surface modeling

1 Introduction and background

In many recent games, an extremely vast and explorable world is one of the main features. Some recent games and their terrain models are shown below:

Table 1: Terrain models in some recent game engines.
Static/dynamic indicates whether terrains are editable during run-time.

Engine	Released	Heightmaps	Displacement	Smooth voxel	Languages
Upvoid Engine	2014	No	No	Yes, dynamic	C#
Unity 4.3	2014	Dynamic	No	With plug-ins	JavaScript, C#, Boo
Unreal Engine 4	2014	Static	No	No	C++, UnrealScript
CryENGINE 3	2009	Dynamic	No	Prior to 3.5.3	C++
Torque Game Engine	2007	Static	No	No	C++, TorqueScript
Source Engine	2004	Static	Static	No	C++, Lua, Python, ...
Panda 3D	2002	Dynamic	No	No	C++, Python

Idealy, the shipped game executable should be able to – without any involvement of humans – generate worlds that are virtually endless. Such terrains may be generated by combining and transforming noise functions [2, 4], and maybe running erosion algorithms on the generated terrain afterwards.

Issues with current approaches:
Compositing noise is typically done by utilizing a noise generation library, such as libnoise or Accidental Noise Library. However, there are two problems with this approach: 1) these libraries are commonly written for the CPU and generating large amounts of terrain can be expensive and may limit the LOD. 2) tuning the terrain generation algorithm can be difficult. Whenever the terrain generation algorithm has changed, the game typically has to be rebuilt and relaunched in order to see if the change achieved the desired effect.

A game developer could improve the generation time by writing terrain generating code as a GPU shader [5, 7]. The issue with having a long feedback loop in the design process still remains. In the event of height-map-based terrain being used, a solution could be to use a node-based procedural shader editor to assist in the development of the terrain shader. Such shader editors commonly let the terrain designer work with a graph-based editor to create a two-dimensional texture that can be previewed in real-time.

However, two issues arise from the fact that shader editors normally edit textures, not height maps, limiting previews are to diffuse maps, normal maps, bump maps and specular maps. Previews for height displacement maps are uncommon and diffuse and normal maps are a poor substitute. Also, these editors are commonly tied tightly to a specific rendering engine. This can be cumbersome when trying to integrate the terrain with other parts of the game engine, such as the AI and path finding.

Houdini, World Machine and Lithosphere, are three very useful graph-based noise synthesis tools that are commonly used to empower a traditional content creation process with procedural techniques. These tools, however, rely on creating terrain height map textures at, or before, build-time. They are hence not usable when creating an endless world, or a replayable game as discussed earlier.

This paper describes our Noise Modeler, a terrain generation framework designed to unify the flexibility and power of graph-based noise synthesis tools, such as Houdini, World Machine or Lithosphere, with in-game generation.

2 Our Approach

Current noise-synthesis tools offer real-time previews , but none of them are designed with in-game generation in mind. Due to their proprietary license and unavailable source code, there is little to be done with World Machine or Houdini. Lithosphere on the other hand, is AGPL3-licensed, so one way to remedy this would have been to extend or modify Lithosphere to also function as a library.

In ANL, libnoise, Lithosphere and many procedural shader editors, each node in the graph-model represents a 2D function. $f(x, y)$ which outputs a single value. Each node can have "sources", other nodes it depends on to calculate the result of $f(x, y)$. When terrain is generated, an output node is queried for its height at a given position. The graph is then traversed recursively to generate the output. Each node is responsible for querying its sources for the values needed to compute its output.

Our Noise Modeler, however, use nodes to represent function *calls*. I.e. the nodes are not using sources to answer a query, but model the the input position is inserted in one end of the graph and then the corresponding function calls are then executed, and the output is then read from the other end of the graph. This model is quite similar to the one used by Houdini.

The advantage of modeling the graph this way, is that the inputs and outputs of the graph are much more flexible with regard to supported terrain representations, i.e. it can easily be used to model voxel terrains, vector displacement terrains or layered terrains.

Our library has been carefully implemented without dependencies on the GUI application. The follwing are the main three modules:

model provides an object-oriented representation of function graphs and their relationships. The interface provides ways to modify and create new graphs.
serialization serializes and parses the model graphs to and from JSON.
code generation generates GLSL functions equivalent to the function graphs.

The generated code is stand-alone, i.e. it does not rely on any textures or other buffers to compute the function values. It is thus callable from a wide range of shader stages, including fragment, vertex, tessellation and compute shaders. Its limited set of GLSL functionality is also easily portable to most platforms.

2.1 Noise and stochasticc terrains

Stochastic interpolation may be used to generates terrains by evaluating a large batch of noise values at once. Mandelbrot [8] uses a two-dimensional **fractional Brownian motion (fBm)**, as an approximation of terrain altitudes. Fournier et al. [3] rendered terrains using **stochastic interpolation**, an approximation of **fBm** by recursively interpolating values with a pseudo-random offset proportional to the distance between the data points interpolated.

A **procedural noise function**, may be used to approximate fBm. Our Noise Modeler uses an unpredictable continuous function with range approximately $[-1, 1]$ and an approximate frequency of 1 Hz for this work. Ideally, the noise should also be isotropic, meaning that regardless of how you rotate the noise, it will look similar. A formal definition of procedural noise can be found in [7].

On its own, noise is not a very close approximation to fBm, but by combining the function with itself scaled to different frequencies and amplitudes, it is possible to get an approximation satisfactory for terrain generation. This implementation of fBm is a common choice among applications that need simple procedurally generated terrains. Babington [1] has for example used this implementation to generate terrains for the NTNU HPC-Lab snow simulator. Babington used the algorithm with Perlin noise as the noise function. Nordahl [10] enhanced this implementation by providing a GUI that could be used to adjust the inputs to the fBm function while the simulator was running. Musgrave [9] also contains a comprehensive guide on other ways noise can be used as a building block to create a wide range of natural structures

2.2 Generating shaders

Generating the shaders consists of the following four steps:

1. *Generate a function declaration for the graph based on the external inputs and outputs.* In the case of height map terrains, this is usually a function that takes a single two-dimensional vector as an input parameter and returns a single float as a value parameter.
2. *Generate unique variable declarations for all connected outputs.* For each node, look at which outputs have edges to other nodes. For each of those outputs, generate a unique id and use it to generate a declaration for a GLSL variable.
3. *Sort the nodes topologically* so that nodes come before the nodes their outputs are connected to.
4. *Generate code for each node.* For each node, generate declarations and assignments to default values for all of its inputs. For each of the connected inputs, generate an assignment to the unique variable of the corresponding output. Then generate code specific to the node type. i.e. for a multiplication node, perform a multiplication or for a noise node, generate a call to the noise generating function. At last, generate assignments to unique variables for any of the node's outputs that are connected.

2.3 Using the shaders

The generated shaders functions may be used in the follwoing two ways:

1. *To generate a height map* If the world's position coordinates are specified as shader attributes, the shader function can be used to render a patch of height values onto a two-dimensional frame buffer which can be transferred back to the CPU. In this format the height map is typically usable by many game engines, and it may be used as if it was a regular height map loaded from disk.
2. *Directly offsetting a vertex in a vertex shader.* Given the length and width coordinates of a vertex on flat surface, the shader function can be used to move the vertex along the length axis to the appropriate height. This technique works well with a wide variety of level-of-detail and tessellation algorithms. The terrain preview in our Noise Modeler is implemented using this approach.

3 Results

The developed software achieved what it was designed to do: It is an implementation of a graph-based editor for implicit procedural terrain that does not require explicit geometry to be generated before build-time to be usable by game engines. Thus, it is a tool usable by games that are designed to have endless worlds and feature unique content every playthrough.

One such example usage of the library, is the terrain editor itself. i.e. it uses GLSL code generated by the library to show a terrain preview. Another example, is a simple command line application (nmcli) that simply generates a GLSL function given a file containing a serialized implicit terrain surface. At last a benchmark application was developed which demonstrates how explicit geometry (height maps) can be generated and transferred to CPU memory.

Our library demonstrates that our approach makes it possible to generate explicit geometry for implicit terrains at interactive rates. This has been demonstrated in two ways: Firstly, by running the editor itself, it can be observed that a terrain preview is rerendered at interactive rates as the model is edited.

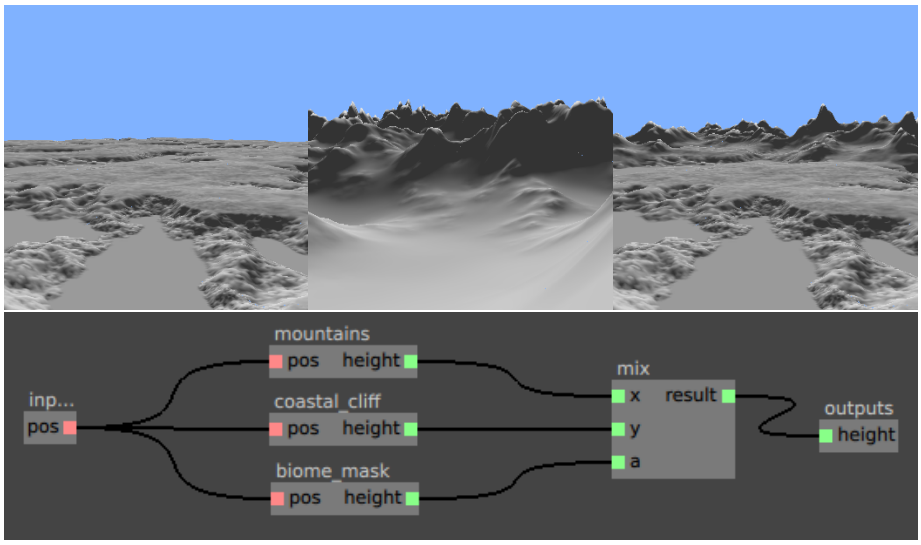

Fig. 1: Different biomes can be combined using a high-level mask, such as fractional-brownian motion (fbm) with a low frequency and few octaves.

Secondly, a more thorough benchmark was performed by Helsing [6]. In this benchmark, batch evaluation of large patches of a height map terrain was performed and compared with similar CPU-based noise libraries.

4 Conclusion

Most procedural terrain editors focus solely on generating height maps at the game developers computer, ignoring many powerful capabilities of procedural generation, the most important being replayability and vastness. Our framework shows that it is possible to model procedural terrains interactively in a user-friendly application, while at the same time retaining the ability to integrate with a game engine and generate explicit terrain geometry during run-time. The

efficient computation of height values allowed an interactive, high-quality terrain preview to be updated with a vertex count comparable to state-of-the-art video games.

Our framework may be used to model height map terrains, and supports several popular algorithms for stochastic implicit terrains. Our framework may also model and generate GLSL code for — although not preview — other types of terrains, including voxel terrains and vector displacement terrains. It may thus integrate well with a variety of game engines with different approaches to terrain modeling.

Future work includes adding support for previewing additional types of terrain representations such as voxel terrain and vector displacement terrain. In addition an effort to port the library to JavaScript has been started[1] with the intention of making the format usable by web applications using WebGL.

References

[1] Kjetil Babington. "Terrain Rendering Techniques for the HPC-Lab Snow Simulator". Master's Thesis. Norwegian University of Science and Technology, 2012.

[2] David S Ebert et al. *Texturing & Modeling: A Procedural Approach*. Morgan Kaufmann, 2003.

[3] Alain Fournier, Don Fussell, and Loren Carpenter. "Computer Rendering of Stochastic Models". In: *Communications of the ACM* 25.6 (1982), pp. 371–384.

[4] Manuel Gamito. "Techniques for Stochastic Implicit Surface Modelling and Rendering". PhD thesis. England: University of Sheffield, 2009.

[5] Ryan Geiss. "Generating Complex Procedural Terrains Using the GPU". In: *GPU Gems 3*. Addison-Wesley Professional, 2007, pp. 7–37.

[6] Johan K. Helsing. "Framework for Real-Time Editing of Endless procedural Terrains". Master's Thesis. Norwegian University of Science and Technology, 2014.

[7] Ares Lagae et al. "State of the Art in Procedural Noise Functions". In: *Eurographics 2010-State of the Art Reports* (2010).

[8] Bernard Mandelbrot. *The Fractal Geometry of Nature*. CA: Freeman, 1982.

[9] F Kenton Musgrave. "Procedural Fractal Terrains". In: *Texturing & Modeling: A Procedural Approach*. Morgan Kaufmann, 2003.

[10] Andreas Nordahl. "Enhancing the HPC-Lab Snow Simulator with More Realistic Terrains and Other Interactive Features". Master's Thesis. Norwegian University of Science and Technology, 2013.

[1] `https://github.com/johanhelsing/noisemodelerjs`