2017

Extensible Terrain Generator

William Frye
Grand Valley State University

Extensible Terrain Generator

By
William Frye
April, 2017

Extensible Terrain Generator

By
William Frye

A project submitted in partial fulfillment of the requirements for the degree of

Master of Science in

Computer Information Systems

at

Grand Valley State University

April, 2017

_____

Dr. Robert Adams **Date**

Table of Contents

Abstract

The goal of the Extensible Terrain Generator is to provide a flexible framework for experimentation with procedural terrain generation. This framework supports easy addition of new terrain generation algorithms, automatically generates UI elements to control their parameters, and even allows swapping out the entire underlying data structure with minimal effort. Terrain algorithms are decoupled from the actual terrain data structure, and as such either side can be changed with no need to modify the other.

A grid-based heightmap terrain data structure and a plate tectonics simulation algorithm are currently implemented, with an icosphere-based terrain data structure waiting to be swapped in—this will require no changes to the plate tectonics algorithm. Planned future additions include erosion simulation, meteor impacts, and volcanism.

Output is configurable per terrain data structure type, with the grid-based terrain outputting a grayscale heightmap suitable for import into a variety of applications.

Introduction

Procedural terrain generation is both an interesting problem and a very difficult one for which to define a "correct" solution. While there are portions of the subject that can be measured—performance, stability, specific features—the primary concern is largely a subjective one: does the output look good? Is it believable as natural terrain? If not, is it at least interesting? The motivation for this project was to create a simple but flexible framework to support the experimentation and rapid iteration necessary to address those questions, with the goal of producing believable terrain generated by numerous independent algorithms each simulating natural processes.

Background and Related Work

Existing commercial terrain generators, such as Terragen and L3DT, use noise algorithms almost exclusively. While they can produce enormous terrains with incredible efficiency, they're just noise—under close inspection, the results are often bland, repetitive, or just don't make sense from a geological standpoint without manual intervention. Both of the cited examples rely heavily on manual intervention from artists to produce quality results. These applications aren't open for alterations, and if they were, would likely not be structurally suited to being extended in the ways this project requires—their focus is on the output, while this project is more about simulation.

Several journal articles were reviewed before beginning this project, but none approached the problem in the desired manner. *Parametrically controlled terrain generation* (Kamal 2007) describes a number of different algorithms and techniques for manipulating a heightmap, but always at a very limited scale, e.g. a

single mountain. *A lazy, lightweight algorithm for generating very large navigable terrains* (Johnson 2011) simply described a process for generating terrains from existing noise textures; useful as a second step, perhaps, but the heightmap texture needs to be generated first.

Program Requirements

The Extensible Terrain Generator's requirements are as follows:

- Must support arbitrarily sized and shaped terrains
- Must support the addition of new terrain types with minimal maintenance
- Must support multiple simulation algorithms to mutate said terrain
- Must support easy addition of new algorithms with minimal maintenance
- Must support multiple types of output for generated terrains

Extensibility is the primary concern, and as a result, the focus of the project has been directed primarily at the architecture of the underlying framework, along with proof-of-concept implementations of the various components.

Implementation

The core logic was written in C#/.NET due to prior familiarity and useful features, while the GUI was a learning experience with WPF.

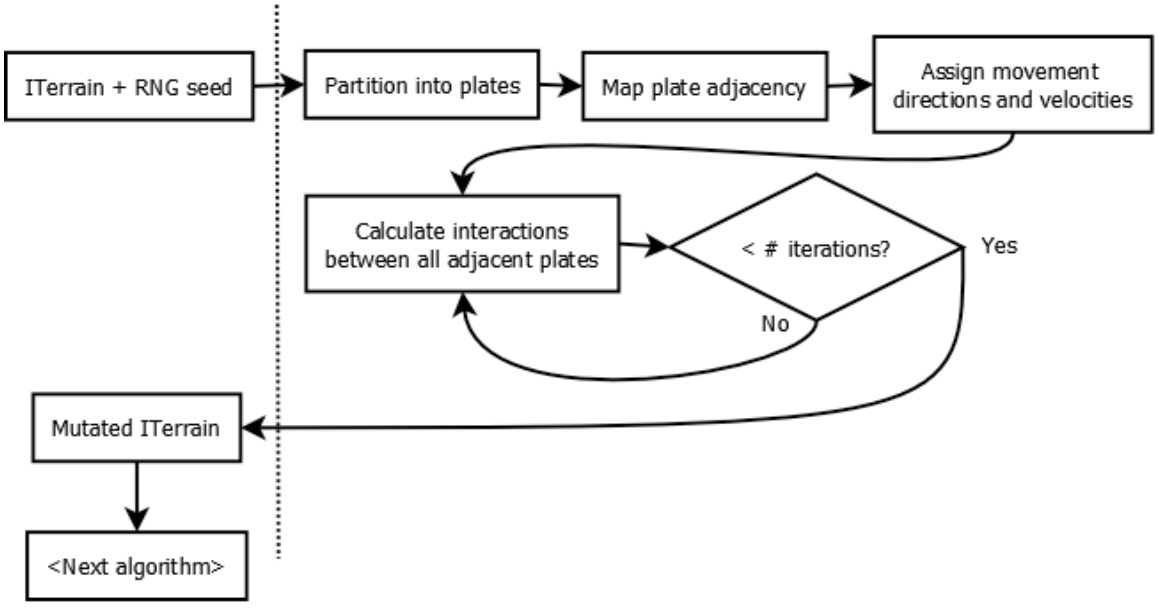
The project uses a basic MVC architecture in which the Model is the core logic, which is encapsulated within its own project file and has no UI. The View is little more than a container which holds whatever controls the Controller generates—this is done using reflection. The major breakthrough in the project's structure came with the discovery of how to dynamically build controls and databind them to objects generated at runtime by gathering the important components via reflection based on interfaces—this allowed the UI to be effectively zero maintenance regardless of the number of terrain algorithms implemented. Obviously, this was a huge timesaver, and it effectively decouples the model from the view. The GUI can be swapped out for a different one with minimal effort, or even removed completely—the Controller could easily be hooked up to function with a command line application.

Interfaces and reflection are the most important pieces of the project. Every critical component is abstracted behind an interface so as to be replaceable with minimal effort; determining what these critical components were and how to effectively decouple them from each other was the single most time-consuming portion of the project, but also the most beneficial long-term. C#'s extensive reflection capabilities are used to pull the pieces together and present them as options to the user via the WPF GUI. Public properties representing parameters on interface implementers become controls based on their type, e.g. an integer parameter results

in a textbox which only accepts integer input being added to the GUI. Only the Controller needs to know what these interfaces are, and only to find them.

The Plate Tectonics algorithm utilizes Microsoft's System.Numerics Vector library, which greatly simplified the code by providing efficient vector math capabilities. As each terrain node is fundamentally a 3D vector with some additional metadata, usage will likely spread to other areas over time. Once the nodes are partitioned into plates, each plate is assigned a direction and velocity. Each plate is then tested against every plate neighboring it, utilizing vectors based on their directions and velocities to detect and resolve any collisions between them in a proportional manner.

Terrain algorithms are applied to the Terrain data structure in an iterative manner, mutating its nodes with each iteration. Each ITerrainAlgorithm requires only an ITerrain and an RNG seed, at minimum, with additional parameters at their individual discretion. The figure below depicts an ITerrain being passed in to the Plate Tectonics algorithm, which partitions the ITerrain's nodes into contiguous plates and calculates interactions between them. Assuming this is the first algorithm to be run on the ITerrain, all nodes will enter with default elevations, and upon completion, the ITerrain's nodes will have been updated with the elevations calculated by the algorithm. Additional algorithms may then be applied as desired.



Results, Evaluation, and Reflection

The Extensible Terrain Generator, using the sample implementations of the terrain data structure and plate tectonics algorithm, effectively generates a heightmap representing a terrain fractured into plates. There is certainly room for improvement here, but it's largely aesthetic: replace the linear interpolation of collision results with something less regular, add more randomization to plate boundaries, etc. Performance when

generating large terrains is less than perfect, particularly in the logic that partitions the terrain into plates. Handling of extremely large datasets in general is an unresolved issue, and some means of streaming data to and from disk during generation is going to be necessary.

While the application isn't quite as feature-rich as originally intended, the underlying architecture meets the requirements:

- Must support arbitrarily sized and shaped terrains - Yes
- Must support the addition of new terrain types with minimal maintenance - Yes
- Must support multiple simulation algorithms to mutate said terrain - Yes
- Must support easy addition of new algorithms with minimal maintenance - Yes
- Must support multiple types of output for generated terrains - Yes

The framework is as extensible as planned: different terrain types can be supported by simply adding a new implementation of the ITerrain interface, and additional types of simulation can be added by implementing additional ITerrainAlgorithms. Every component has at least a proof-of-concept implementation: GridTerrain for the ITerrain interface, PlateTectonics for ITerrainAlgorithm interface, and a greyscale heightmap output. Numerous additions can be made as time allows, and most of the really difficult problems are solved. The main focus of the project was supporting experimentation, and it was successful in that regard.

Originally, an icosphere-based terrain and two to three additional terrain algorithms were planned. Due to a consistent stream of emergencies at work and a fairly sudden move, time became a serious issue and many of the planned features had to be cut. Focus shifted instead to just what was necessary to demonstrate the capabilities of the framework. It is tempting to claim that this shows an ability to adapt to external forces in an attempt to salvage the project, but it's probably more realistically interpreted as a failure of planning in the early stages. Both initial estimates of available time and estimates of the difficulty of several aspects of the project proved to be serious weaknesses.

Conclusions and Future Work

The framework created by this project opens the door for an enormous amount of future work. Finishing the partially-implemented icosphere terrain data structure is high on the list; performance enhancements will necessarily follow ("necessarily" because of the sheer size—calculations documented on the project's GitHub wiki place the number of data points at around 600,000,000 for a resolution of one point per kilometer on an earth-sized sphere). Of course, additional terrain algorithms are the most important thing on the list; the framework was designed with this in mind. In addition to refinement of the existing plate tectonics algorithm, implementations of such natural processes as erosion, volcanism, and meteorite impacts are planned. Direct control over the order in which algorithms are applied is also a planned feature.

In a slightly more distant future, extra metadata can be added to the individual terrain nodes to support simulations of things other than simple elevation changes—weather, for instance.

Bibliography

Belhadj, F., & Audibert, P. (2005). Modeling landscapes with ridges and rivers. *Proceedings of the ACM symposium on Virtual reality software and technology - VRST '05,* 151-154. doi:10.1145/1101616.1101648

Beneš, B. (2006). Physically-based hydraulic erosion. *Proceedings of the 22nd Spring Conference on Computer Graphics - SCCG '06,* 17-21. doi:10.1145/2602161.2602163

Driemel, A., Har-Peled, S., & Raichel, B. (2012). On the expected complexity of voronoi diagrams on terrains. *Proceedings of the 2012 symposuim on Computational Geometry - SoCG '12,* 101-110. doi:10.1145/2261250.2261266

Johnson, B. (2011). A lazy, lightweight algorithm for generating very large navigable terrains. *Proceedings of the 49th Annual Southeast Regional Conference on - ACM-SE '11,* 281-286. doi:10.1145/2016039.2016112

Kamal, K. R., & Uddin, Y. S. (2007). Parametrically controlled terrain generation. *Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia - GRAPHITE '07,* 17-23. doi:10.1145/1321261.1321264

Mcguire, M., & Sibley, P. G. (2004). A heightfield on an isometric grid. *ACM SIGGRAPH 2004 Sketches on - SIGGRAPH '04,* 141-141. doi:10.1145/1186223.1186399

Tools:

- www.bibme.org – Bibliography generation from journal article titles
- Microsoft Visual Studio 2015 – coding
- Git/GitHub – source control

Appendices

GitHub repository: https://github.com/fryew/terrain-generator