Procedural Planet Generation

Champlain Valley Union High School

Milo Cress
Advisor: Anna Couperthwait

February 20, 2019

Abstract

In this article, we present a system for the online rendering of realistic terrain at varying spatial and temporal scales using concurrent functional programming, and leveraging modern multicore computers.

1 Maps

Foundational to the concept of terrain generation is its representation in 3-dimensional space. In cartography, maps provide information about an area of land by translating a given 2-dimensional point to a value. Heightmaps translate latitudes and longitudes to locations in 3-dimensional space by supplying a z coordinate to a given (x, y) coordinate pair. In climatology, "wetmaps" track rainfall over given areas, and predict, given an (x, y) coordinate pair, the inches or centimeters of rain that land on or near than point in a given amount of time. In more complex simulations, vector fields which simulate wind and weather are used by meteorologists to predict the spread of weather events across an area. By encapsulating a tract of land and the forces that act on it as a spatial function, the large-scale calculation of terrain becomes trivial.

1.1 TODO Functional programming

When speaking about maps, it's easy to imagine them as they are presented in most cartographical texts – a layering of lines and colors on top of a page. However, attempting to use this model for procedural generation presents two immediate challenges:

1. We don't know how these maps were created. All we're given is a data dump and assurance of its accuracy.
2. We are working in a limited resolution, and therefore must trade scale for level of detail. Since this representation of a map forces us to know the value of every point at once, we can either know the values of a small range of points with a great degree of detail, or a vast range of points with a limited degree of detail.

1.1.1 Monoids

1.1.2 Functors

1.1.3 Endofunctors

1.1.4 Monads

Monads are just monoids in the category of endofunctors!

1.1.5 Typeclasses

2 TODO Engine

2.1 Raytracing

2.1.1 Distance estimators

2.1.2 Hacking procedural generation

2.2 Shadows

2.2.1 Hard shadows

2.2.2 Soft shadows

2.3 Reflections

2.3.1 Specular

2.3.2 Diffuse

2.3.3 Caustics

2.4 Ambient occlusion (SSAO vs true ray-casting)

3 TODO Simulation

3.1 Continental Drift

3.1.1 Force-Mapping over time (developing vector fields over heightmaps)

3.2 Erosion

3.2.1 Gradient-descent force-maps

4 TODO Optimization

4.1 Functional programming (reprise)

4.1.1 Deforestation (fusion)

The Haskell Community was clear in its recommendation of the Glasgow Haskell Compiler for optimization of functional code [2].

4.1.2 Graph reduction

4.1.3 Parallelism

4.1.4 Aptitude for compilers and static analysis (DSL)

4.1.5 Automatic differentiation

1. Reworking code to parameterize types

4.1.6 GPU - parallelism

"GPUs are capable of massively parallel computation" [1]

4.1.7 Haskell Parallel DSL

4.1.8 Reworking code for parameterized types

4.1.9 Cuda library

Mark Engelhardt, a systems engineer and geospatial programmer, stressed thte importance of low-level optimization, even in high-level languages such as Haskell.

5 TODO To do

5.1 Material system

5.1.1 Node-system (blender)

5.1.2 Generative adversarial networks for textures

5.2 Typeclasses

5.2.1 UV mapping typeclass

5.2.2 Random sampling typeclass for soft shadows and diffuse shading

6 DONE Works Cited

References

[1] John Cohn. *Massively Parallel Computations*. Jan. 2019.

[2] Haskell Community. *Triggering Stream-Fusion in Haskell*. Jan. 2018.