15-418 Project Proposal: Parallel Terrain Generation

By Petros Emmanouilidis and Victor Zayakov

Title

Parallel Perlin Noise Terrain Generation using CUDA

Website URL

https://vzayakov.github.io/418FinalProject/

Summary

We will implement 2 different data-parallel approaches for generating Perlin noise maps on an Nvidia GPU with CUDA and analyze their performance. One approach divides the problem spatially while the other divides computation across iterations of the algorithm. The generated map will be interpreted as 3D voxel terrain and rendered through the open-source software Blender for visualization.

Background

Perlin noise is a procedural texture primitive often used for generating terrain that looks natural. Compared to other noise textures, Perlin noise is known for featuring smooth transitions that mimic natural topographies. The base algorithm for generating a 2D Perline noise map is the following:

1. Divide the map into a grid of pixels and assign a random unit-length gradient vector to each intersection in the grid
2. For each pixel, calculate dot products between the gradient vectors at each of its corners and an offset vector from the pixel center to each corner
3. For each pixel, interpolate between the 4 dot products and increment the pixel's value by the result
4. Repeat the previous 3 steps for a fixed number of iterations

In addition, if we have time we may also implement and parallelize Fractal Brownian Motion (FBM). FBM adds fine detail to the terrain and makes it look even more realistic. This algorithm involves combining multiple iterations of the Perlin noise function, to create

terrain that is self-similar. This means that as we zoom into the terrain, it looks the same as in the zoomed-out view.

We are also looking into the Voronoi noise algorithm for adding different biomes to our terrain if we have the time. This algorithm creates cells or regions of a pseudorandom size and shape that cover the entirety of the terrain, which can then be assigned to different "biomes". Within each biome we can have different gradients and/or number of iterations of the Perlin noise algorithm, to produce differences in the terrain.

The Challenge

To generate Perlin noise maps with CUDA, we will explore and compare 2 distinct approaches, each with its tradeoffs.

The Perlin Noise computation is embarrassingly parallel in theory. For each pixel, we calculate the dot products between the gradient vectors at each of the 4 corners of its grid square and then interpolate. However, in practice, this involves excessive memory reuse and has a low arithmetic intensity. As a result, these characteristics require effective and cautious use of shared memory and work allocation strategies to optimally parallelize this problem.

First, we are considering a Spatial Partitioning approach. This is the most intuitive method and involves partitioning the map into equal square chunks and assigning each chunk to a CUDA block. Within each block, each thread is responsible for computing a single pixel (or a small group of pixels) over all iterations of the algorithm. Intersection gradient vectors are shared variables that are renewed in each iteration. This approach introduces a memory bottleneck as gradients at the edge of each block are shared between adjacent blocks. This would require that such edge gradients hold the same values in multiple blocks, requiring synchronization, a requirement that becomes tricky to deal with over multiple iterations.

Second, we are considering a Temporal Partitioning approach. Unlike its spatial counterpart, instead of assigning CUDA blocks to map chunks, we assign blocks to entire iterations of the algorithm. In other words, each block is responsible for computations over the entire Perlin map for a single iteration. Since each iteration of the algorithm is independent of previous iterations, we can compute each step in parallel and then reduce the results from each iteration to a single value for each pixel in the map. This method avoids the memory pitfalls of the spatial approach but introduces an additional strain of sum reduction at the end of the computation, which could potentially bottleneck performance.

Lastly, we could introduce a hybrid approach that borrows from both of the formerly described methods in search of a sweet performance middle ground.

Furthermore, we may implement Fractal Brownian Motion (FBM) and biome generation if time permits. This would involve performing multiple iterations of Perlin noise to create FBM. Furthermore, we may utilize different gradient parameters and/or the number of iterations based on the biome. This would introduce additional challenges, such as synchronization over iterations of the Perlin noise algorithm, memory consistency issues, and also uneven runtime between pixels (if more iterations are performed on some pixels than on others).

Resources

We will not be using any starter code for our project. However, we are consulting a relevant research paper for reference. The paper is cited below:

H. Li, Xianguo Tuo, Y. Liu, and X. Jiang, "A Parallel Algorithm Using Perlin Noise Superposition Method for Terrain Generation Based on CUDA architecture," Jan. 2015, doi: https://doi.org/10.2991/meita-15.2015.183. Accessed 13 Nov. 2024.

In terms of hardware, we will be using an NVIDIA RTX series GPU for the Perlin noise terrain generation. We plan to use the GPUs on the Gates cluster machines since those already have CUDA installed and are readily available for us to use. We also plan to use Blender on a personal computer to render our generated terrain, to be displayed as part of the demo.

Goals and Deliverables

Goals

Plan to achieve (at minimum):

- 2 distinct data-parallel CUDA implementations for producing 2D Perlin Noise maps (see Challenge section)
    - One algorithm utilizing a spatial partitioning approach
    - One algorithm utilizing a temporal partitioning approach
- Scripts for converting 2D noise maps to 3D voxel height maps, as well as functionality for exporting to and rendering with Blender, for visualization
- An analytic comparison between the 2 approaches, focusing on both speedup and memory usage.

Hope to achieve:

- Create a parallel algorithm for Fractal Brownian Motion
- Add biomes to our terrain, using a parallel version of Voronoi noise generation

Far-reach goal:

- Minecraft terrain!

Demo Plan

- We will present the rendered outputs of our terrain generation algorithm, which should look very neat, especially if we get FBM and/or biomes working
- We will present speedup graphs for the 2 different CUDA implementations of Perlin noise, and compare the two. We may also compare the GPU memory usage of the two implementations.
- If possible, feature a real-time interactive terrain generation demo (this is a far-reach goal)

Platform Choice

The hardware platform for our project is an NVIDIA GeForce RTX 2080 GPU on the GHC cluster machines, and the algorithm will be implemented in CUDA, to be run on the GPU. The Perlin Noise algorithm involves separate computations for each pixel of an image, hence a data-parallel approach is most suitable for achieving the best performance. GPUs are best suited for data-parallel programming, and we already have prior experience with CUDA.

Furthermore, the main practical use of terrain generation algorithms is for video games and simulations. Such software mainly utilizes GPUs for its graphics workload, so we also wanted our algorithm to be tailored for a GPU.

Schedule

Week 1, Nov 11 - Nov 17:	● Compile a list of relevant resources ● Begin Perlin Noise approaches
Week 2, Nov 18 - Nov 24:	● Continue with Perlin Noise implementations ● Have at least one of the two implementations done by the end of the week
Week 3, Nov 25 - Dec 1:	● Conclude on how to convert noise maps to 3D terrain and how to visualize results (Blender) ● Debug and finalize both Perlin Noise implementations ● Look into biome generation with FBM and Voronoi maps, and begin implementing
Week 4, Dec 2 - Dec 8:	● Continue work on FBM and Voronoi ● Set up analysis of Perlin approaches
Week 5, Dec 9 - Dec 15:	● Finalize any remaining work on FBM and Voronoi ● Finalize any output/visualization scripts ● Prepare for demo

80% Milestone:
Plain Perlin Noise, with 3D voxels. Basic Blender rendering. 2 distinct parallel implementations, which we can compare.

100% Milestone:
Perlin Noise working, with modifiable parameters (scale, octaves, persistence, lacunarity). Basic biome generation using Voronoi noise. 2 distinct parallel implementations which we can compare, while also varying the 4 parameters.

120% Milestone:
Complex biome generation, visualized 3-dimensional biomes generated through a combination of Voronoi and Perlin noise. 2 distinct parallel implementations which we can compare, while also varying the 4 parameters.