Parallel Noise for Cloud Density Approximation

Eduardo Cuesta Córdova A00570171

Problem

Create a randomly generated tileable 3D texture that approximates how the density of cloud behaves intended for cloud rendering using parallelism in the GPU.

Worley noise

For this, the randomly generated image pattern produced by worley noise was chosen, which is conformed of web-like shapes (figure 1) that when inverted look like bubbles or spheres (figure 2) and can be made look like a cloud when multiple frequencies (called octaves) of worley noise are layered (figure 3). [1]

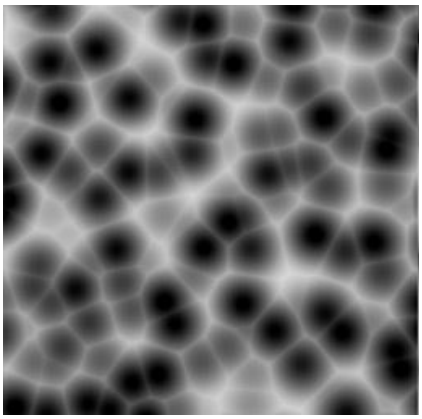

Figure 1: Worley noise in Unreal Engine [2]

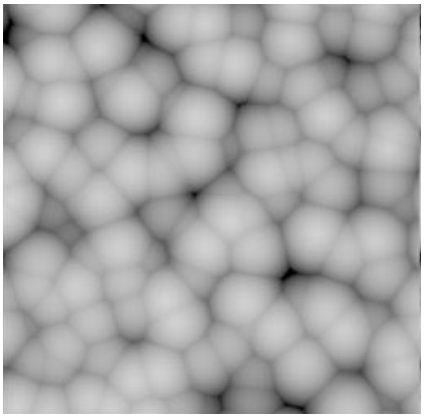

Figure 2: The same Worley noise in figure 1, but inverted [2]

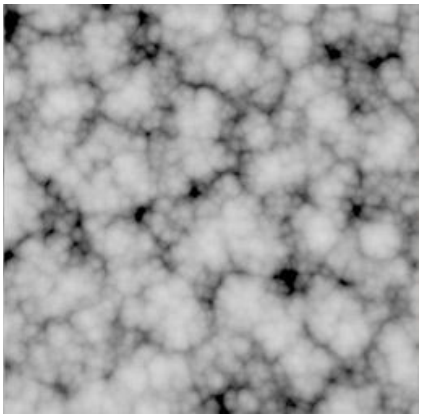

Figure 3: Multiple frequencies of the worley noise of figure 2
put together using different octaves [2]

Intuition for inverted worley noise generation

An easy way to visualize the worley noise generation process is as follows:

1. Create a 2D grid over the image. The quantity of cells in the grid will determine the size of the spherical shapes in the inverted noise.
   In figure 4, the black borders delimit each of the cells and the small squares inside them are pixels. The red square is one pixel.

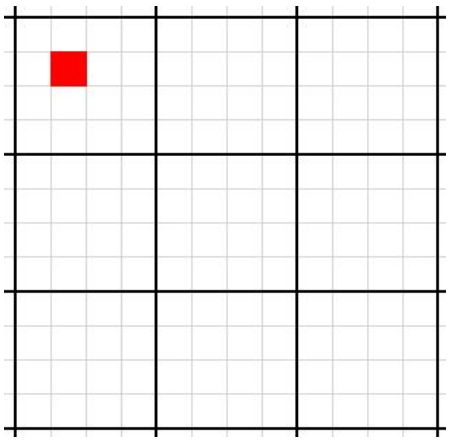

Figure 4, a representation of how the pixels and the grid work

2. Place a point in random coordinates inside each of the cells
3. For each pixel, retrieve the point that was placed in that pixel's cell and in each of the adjacent cells. Then get the smallest distance between the pixel and each of the points. That smallest distance will be the white color of the pixel.

In this way, the darkest parts of the image are those that are very near to where the points were placed because the distance between those pixels and the point will be a small value and thus have very little white [3].

To get inverted worley noise, the returned value for each pixel has to be 1 - distance instead of just the distance.

The same approach can be used to produce three dimensional results, in which a three dimensional grid is used with points that have three coordinates. As images are 2D, the z dimension is represented in slices. This means that a 32x32x32 volume will be represented by 32 slices of 32x32 pixels that will be put side by side making a 32x1024 image.

Why to overlay frequencies

Utilizing different octaves of noise to build the final noise makes the image look more natural by making the smallest frequencies the most important features and the biggest frequencies the least important ones.

An octave of noise is how big is the frequency and how small is the amplitude of the noise. An octave will have bigger frequency, but smaller amplitude than the previous octave.

This, for example, mimics what happens in terrain:

- The largest and highest level features are the mounts, plains and depressions
- There are smaller and less relevant for the overall terrain shape but more common features like deformations and caves in the those mounts, plains and depressions
- Inside those smaller features there are even smaller, even less relevant for the terrain and even more common features like dirt grains or rocks [5].

For overlaying the noise, the lowest frequency worley shapes were scaled by 0.625 to become the biggest feature of the image, the mid frequency was scaled by 0.25 to be the middle sized feature and and the biggest frequency was scaled by 0.125 to become the smallest feature of the final image [4].

Why this method was chosen

The method of utilizing worley based noise to approximate the way the density behaves in clouds was first used commercially in what was to become the Decima Engine for the Horizon Zero Dawn made by Guerrilla Games [1]. Their method included creating the worley noise, remapping it with another noise to make it more wispy and then using another smaller worley noise to erode it and get more detailed features. This same cloud density modelling method is also used in more recent Frostbite Engine [4] games and by Rockstar games in Red Dead Redemption 2 [6]. I chose to replicate the method because it has become very common in real time volumetric rendering in the videogames industry.

Usage of parallelism

It was chosen to make the worley noise texture creation in parallel because every pixel value can be calculated independently from other pixels and then written to the output texture. The idea of parallelism is to perform a big task by abstracting it like a small independent task that is repeated a lot of times. Then, these small tasks are run independently by many threads in massively parallel architectures like GPUs. As the noise images had lots of pixels, choosing a parallel approach was better than going for just concurrency or a small level of multithreading. The code was implemented in CUDA and designed to maximally use the threads in a block and use a number of blocks that depends on the quantity of pixels of the image for maximum scalability between different GPUs.

Hash based noise

For this project, the technique used was hash based noise, in which the point coordinates are generated by hashing the grid coordinates of each cell [4]. This improves on the previously mentioned technique since it uses way less memory because it doesn't store the locations of the points and has less memory accesses because it doesn't have to read the point positions from memory and instead just calculates them by just hashing the grid coordinates.
It also greatly simplifies the implementation.

Results

In figure 5 the worley noise can be seen clearly with the different octaves for incorporating smaller high frequency features. There are big predominant spheres and inside those there are smaller spheres, mimicking the way clouds look like.

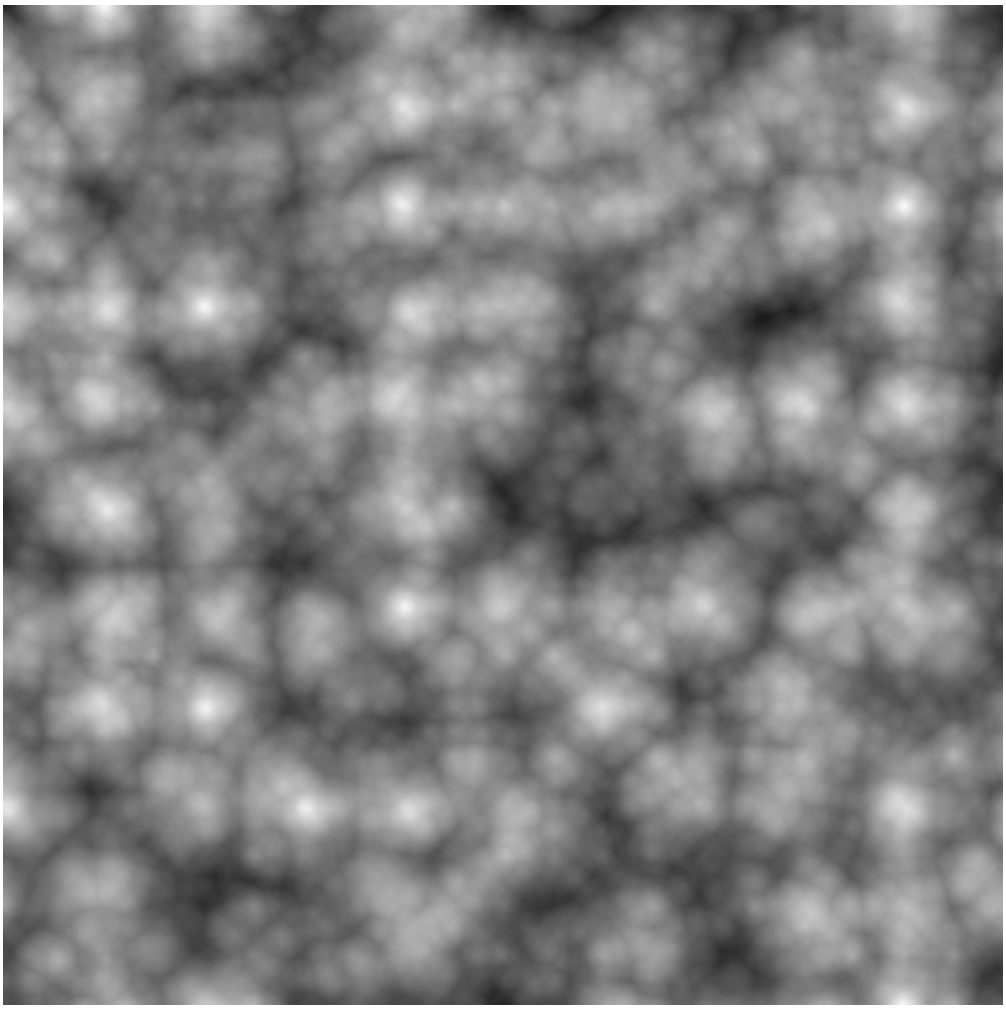

Figure 5: A single slice of 500x500 noise

Figure 6 shows three slices of subsequent depths of a 200x200x200 image of the same noise.

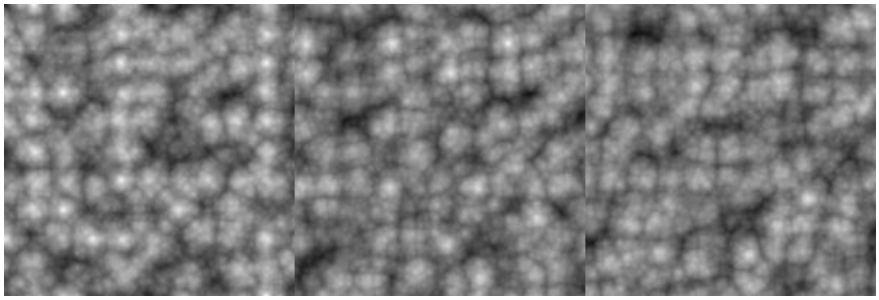

Figure 6: Three slices of 200x200 noise

Figure 7 is the intended output image of 32 slices of 32x32 images of noise.

Figure 7: 32 slices of 32x32 noise

Figure 8 shows a segment of the image in figure 7 to show that the continuity of the patterns is kept in nearby slices along the z dimension and that the different slices don't tile perfectly because they have slightly different depth.

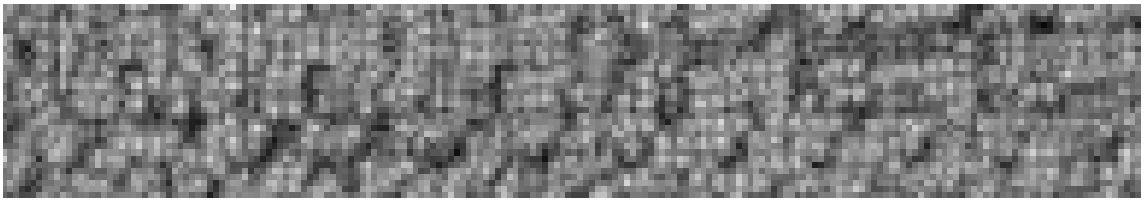

Figure 8: 6 slices of 32x32 noise

Instructions

To setup the program just make sure to have CUDA installed (https://developer.nvidia.com/cuda-downloads) and then:

1. Clone the repository
   ```
   git clone https://github.com/EduardoCuestaCordova/Tileable3DWorleyNoise.git
   ```
2. Get inside of the repository
3. Enter the directory NoiseGenerator/NoiseGenerator/
   ```
   \repo> cd NoiseGenerator/NoiseGenerator
   ```
4. Run the command:
   ```
   repo\NoiseGenerator\NoiseGenerator> nvcc kernel.cu
   ```
5. Then execute the file a.exe with the following parameters:
   a. Width of the resultant texture
   b. Height of the resultant texture
   c. Number of slices of the resultant texture (depth)
   d. Number of cells in the texture, equivalent to noise frequency

e. A png filename

```
\repo\NoiseGenerator\NoiseGenerator> ./a.exe 100 100 2 10 a100100210.png
```

This will generate a png image with the worley noise of dimensions width * depth x height.

References

1. Schneider, A. 2015. The Real-time Volumetric Cloudscapes of Horizon: Zero Dawn. SIGGRAPH advances in real time rendering course. http://advances.realtimerendering.com/s2015/The%20Real-time%20Volumetric%20Cloudscapes%20of%20Horizon%20-%20Zero%20Dawn%20-%20ARTR.pdf
2. Brucks, R. 2016. Getting the Most out of Noise in Unreal Engine 4. https://www.unrealengine.com/en-US/tech-blog/getting-the-most-out-of-noise-in-ue4
3. Lague, Sebastian. 2019. Coding Adventure: Clouds https://www.youtube.com/watch?v=4QOcCGI6xOU
4. Frostbite implementation linked in:
   Schneider, A. 2017. Code linked in Authoring Real Time Volumetric Cloudscape with the Decima Engine. SIGGRAPH advances in real time rendering course. http://advances.realtimerendering.com/s2017/Nubis%20-%20Authoring%20Realtime%20Volumetric%20Cloudscapes%20with%20the%20Decima%20Engine%20-%20Final%20.pdf
   Direct repo link: https://github.com/sebh/TileableVolumeNoise
5. Biagioli, A. 2014. Understanding Perlin Noise. Adrian's Soapbox. https://adrianb.io/2014/08/09/perlinnoise.html
6. Bauer, F. 2019. Creating the Atmospheric World of Red Dead Redemption 2: A Complete Integrated Solution. SIGGRAPH advances in real time rendering course. http://advances.realtimerendering.com/s2019/index.htm