Rendering Clouds using OpenGL Fragment Shaders

CS184 - Spring 2021

William Dai
Callam Ingram
Roman Taylor
Zachary Young

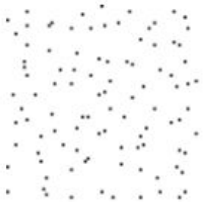
APRIL 20, 2021/ R/ EDIT

2D Worley Noise

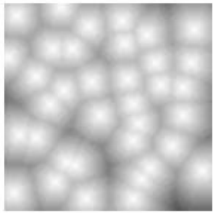
First task in generating realistic clouds was to pick pick an implementation approach. We settled on **Worley Noise** technique. The idea is to scatter random points all over the screen and find the distance between every pixel and it's closest point. Then, we would use that distance to shade every pixel with a value between 0 and 255 scaled by the distance.

To test our 2D Worley Noise generation algorithm, we used a simple **LodePNG** library in order to generate .png files.

First we shaded the background in white color and randomly generated n points. At this stage there is an optimization worth mentioning: evenly subdivide the screen into rectangles and place only one random point per rectangle. This way, every pixel in the scene has to be checked only against a rectangle it belongs to in addition to neighboring rectangles.



Next we found a closest distance between each random point and every pixel in the scene. Normalized by max distance and multiplied by 255 – we now able to shade our scene. The figure displayed on your right is the inverted result [255 – normalized_distance].



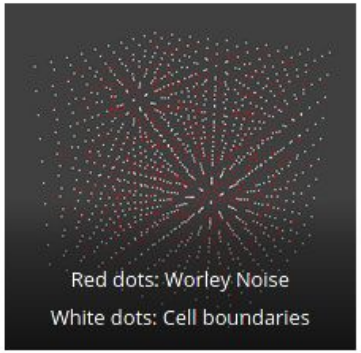
Posted in **Tasks**

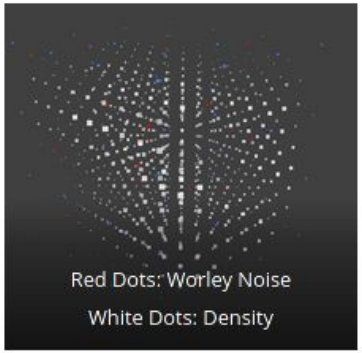
APRIL 22, 2021 / Z / EDIT

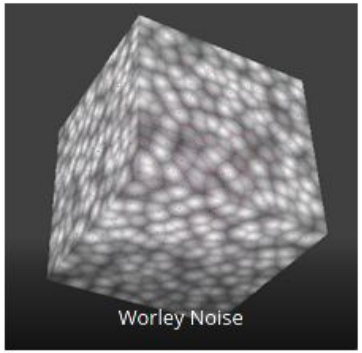
3D Worley Noise

After testing our Worley Noise algorithm in 2D, we extended it to work in 3D space. The computational demand of our algorithm was really high because we had to calculate the **max_distance(every_pixel, every_random_worley_point)**.

In order to minimize the computations, we split up the cube into sub-sections and placed one random point in each sub-section. Now, every pixel has to check it's distance against random worley noise points located in 26 adjacent sub-sections in addition it's own section (total of 27 checks per pixel).







The biggest challenge was in blending cube edges as we replicated the Worley Noise. In short, for the sub-sections touching the outter-planes of the cube, we had to offset our worley noise points to the correct location in order to compute correct distances.

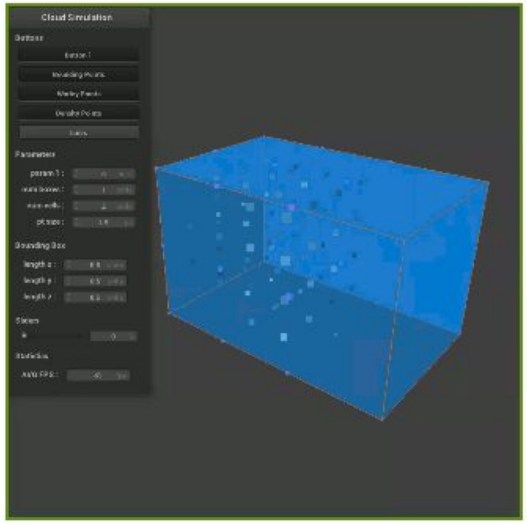
APRIL 26, 2021/R/EDIT

Prior to Ray Tracing

Prior to ray tracing, we generated a bounding box structure using two 3-dimensional points.

The two points represent rear-leftmost-lower point & front-rightmost-upper corners of the bounding box. These points enabled us to shift x, y, and z coordinates to obtain the remaining 6 corners (total of 8).

Given the 8 corners, we were able to generate 12 lines outlining our volume using GL_LINES primitive (please see thin orange outline). Then, from the same points, we generated 12 triangles representing our bounding volume planes (please see light blue & 50% transparent walls). These were done using GL_TRIANGLES primitive.



Lastly we have added three sliders to our GUI in order to manipulate the bounding box: length x, length y, and length z. These simple parameters allow us to easily resize the bounding box volume in three dimensions.