1 Progress

This week progress. First of all, I extended my method to include an arbitrary number of directions. This is because for certain models it may be necessary to add more cameras in order to render all the possible points (e.g.) the dragon. So instead of using a cube map now I am using a texture array. The concept is very similar, and in addition it simplified the shaders. However, we can still see that some areas are not covered at all (next picture). To solve this problem, the solution would be either to implement a more advanced camera placement algorithm or using a A-buffer instead of multiple cameras.

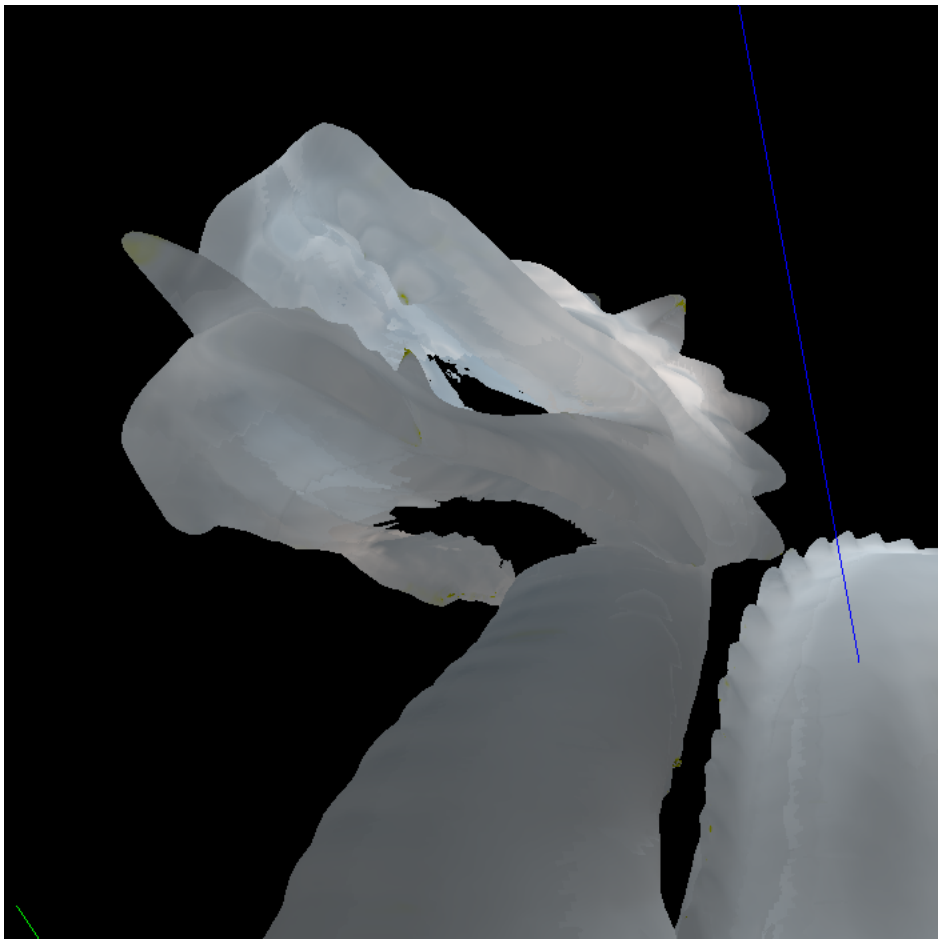


Figure 1: Some areas in the rendering are not covered.

Secondly, I added some randomness on the GPU, randomizing the rotation of the patter for each pixel. In this way, the line artifacts are removed, at the price of a lot of noise. As we can see, even 1000 samples cannot remove the noise completely (see next picture).

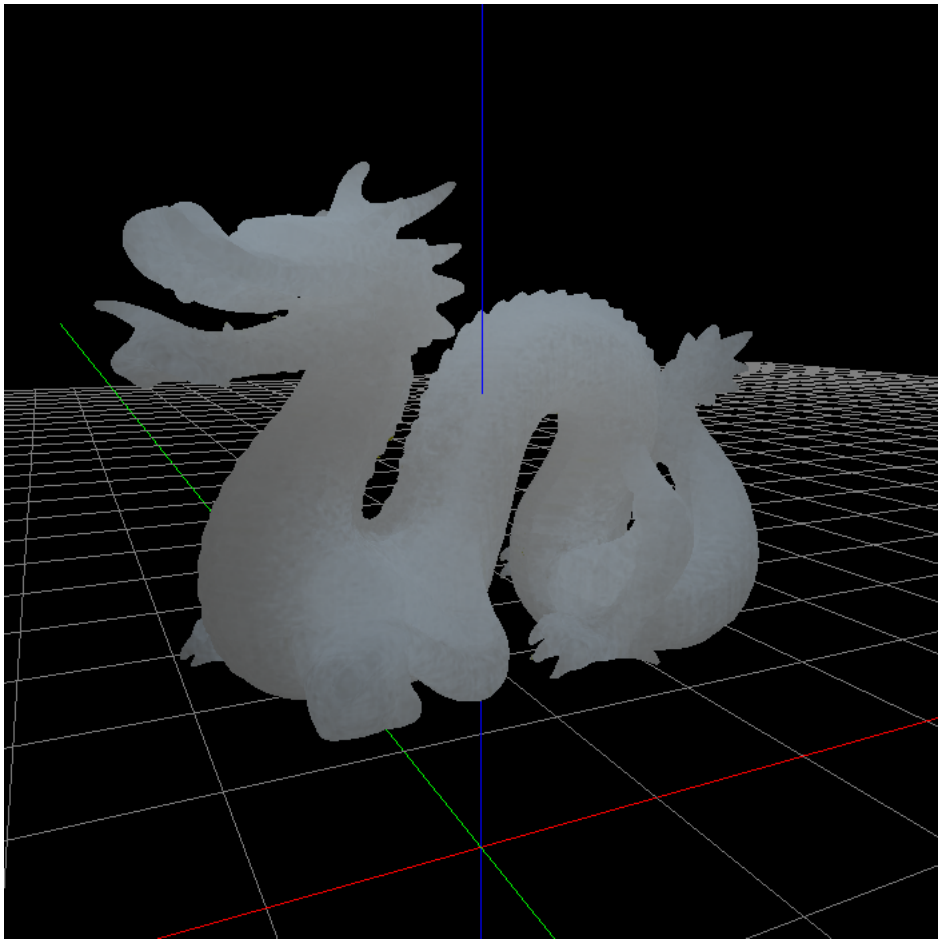


Figure 2: Randomized rotation of pattern. Jensen's Dipole, 1000 samples.

Finally, I added a time evolution of the pattern. A limited number of points is used, but for each frame the points are rotated progressively around. In addition, a small perturbation of the radius is added, in order to avoid artifacts. The schema I have adopted is illustrated in the following picture:

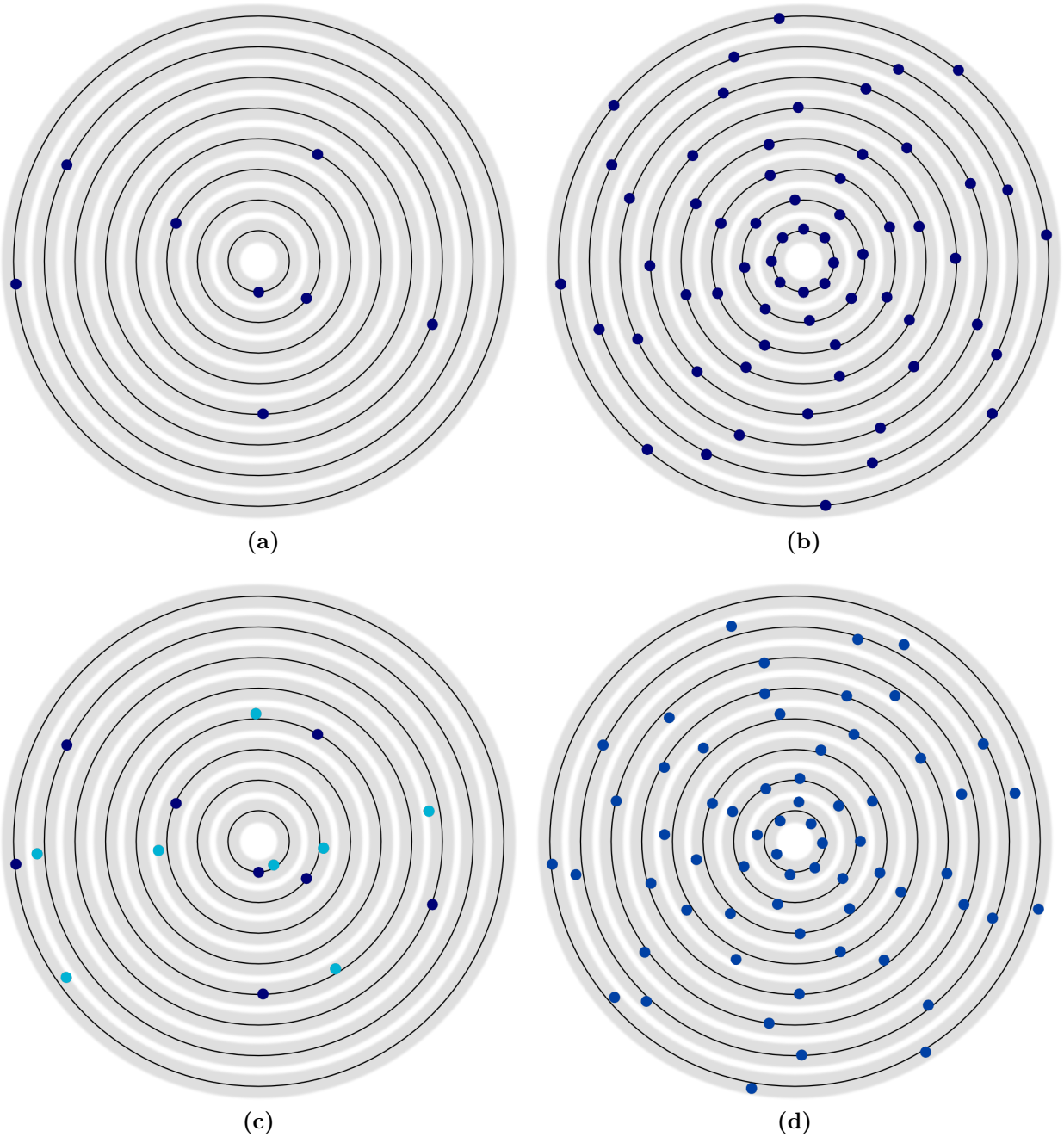
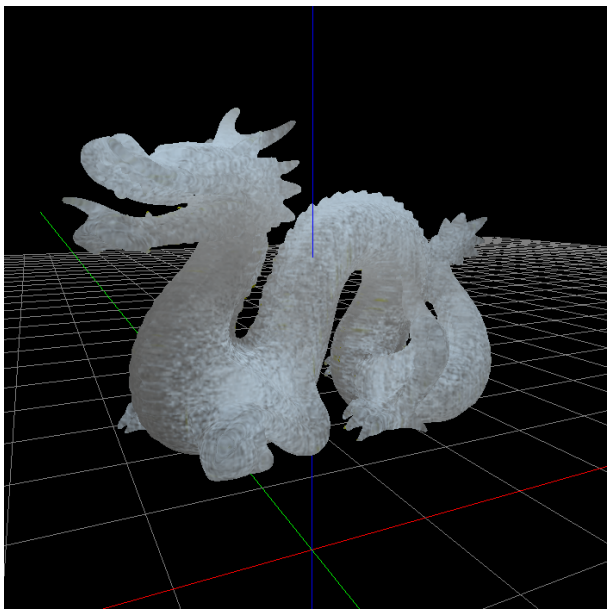
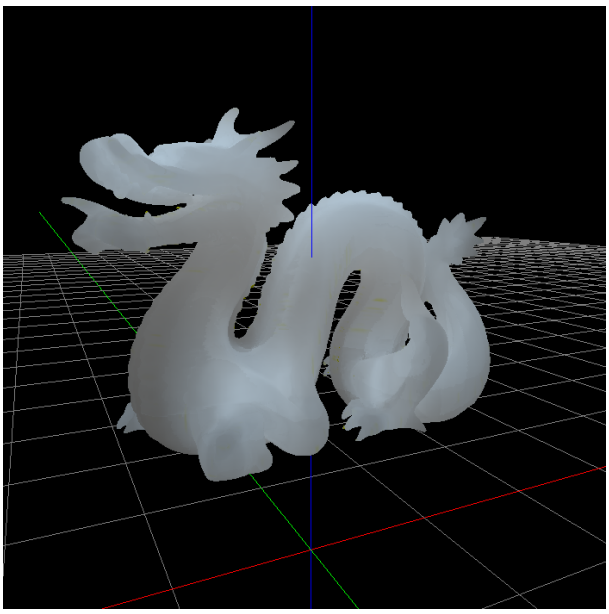


Figure 3: Creation of the proposed pattern. In Figure 3a, A pattern generated by the Hammersmith algorithm (8 samples). In Figure 3b, the pattern obtained by rotating 8 times the pattern around the center. In Figure 3c, our solution, in which we slightly displace each sample once we rotate it. The resulting pattern is shown in Figure 3d

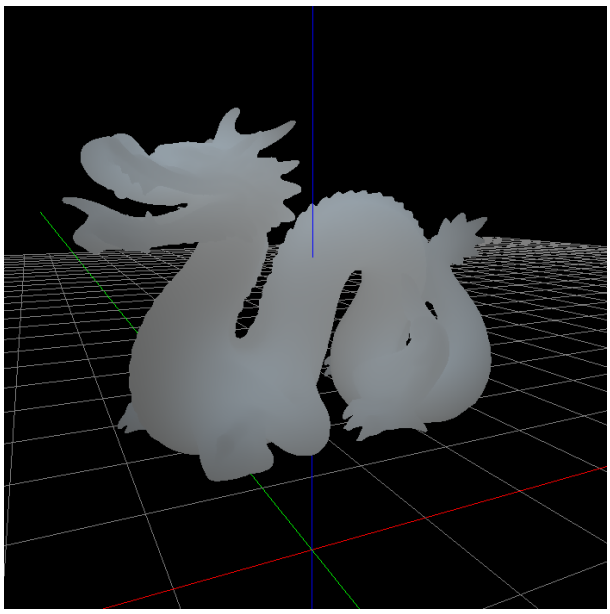
This, at convergence (100 frames, 120 samples per frame), gives me the following results. Each frame during the evolution of the system takes about 50ms with the directional dipole and 40ms with Jensen's dipole.



(a) One step, 120 samples



(b) 100 steps, 120 samples/step



(c) Reference

(d) 10 times the difference between 4b and 4c

Figure 4: This week results.

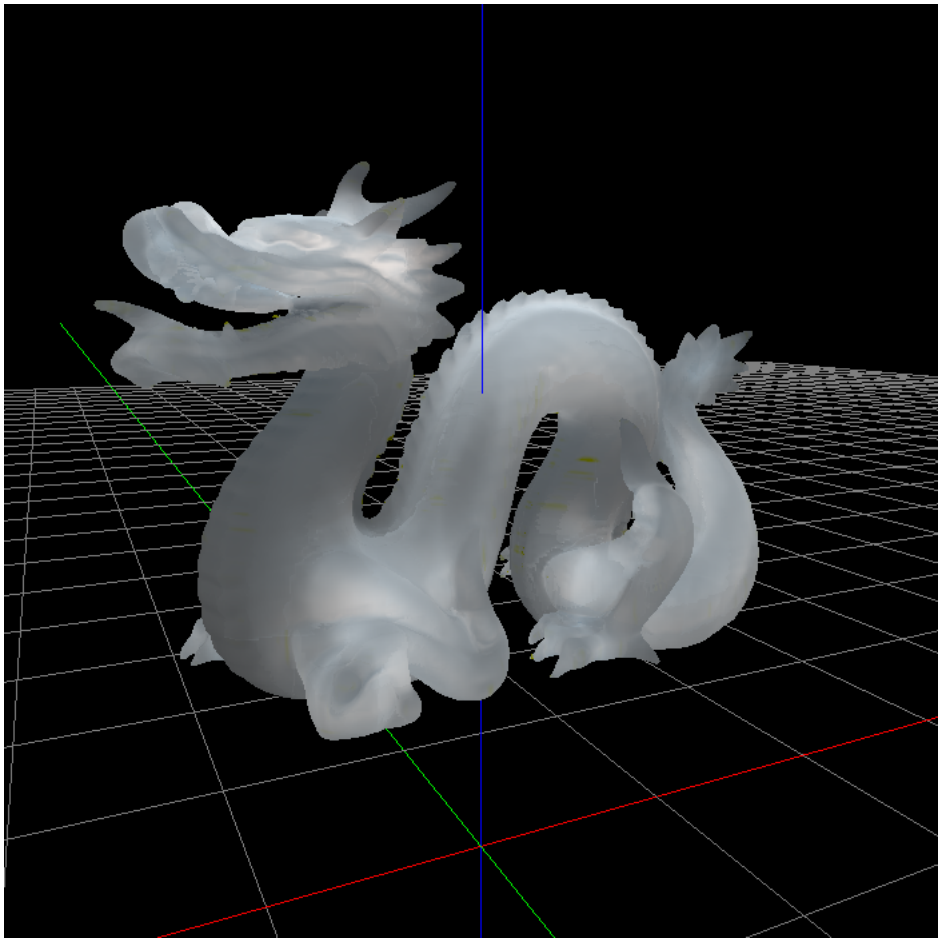


Figure 5: Directional dipole.

A possible solution for solving the noisiness of the first stages would be to use mipmaps to get a basic filtering of the image. In fact, mipmaps act as a simple box-filter, that eliminates high frequency noise - what we want. I tried to apply it, but the problem is that the black color outside the model starts bleeding inside, ruining the result. I need to implement then my own mipmap generation if I want to use this technique.

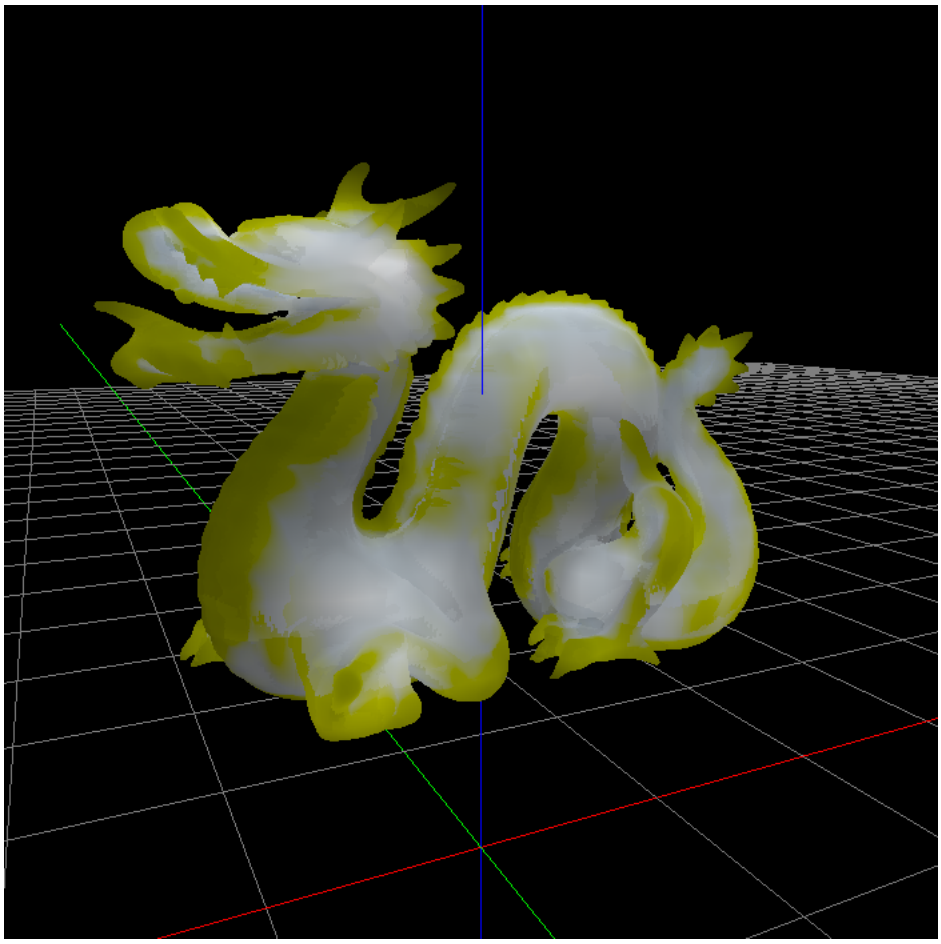


Figure 6: Color bleeding happens when we tried to use the low order mipmaps.

2 Future work

Next steps that I would like to take in the domain of *quality*:

- Number of cameras and their placement: where should they be? (Difficulty: Hard)
- Using an A-buffer with OpenGL 4.2 capabilities: to test and see, it would eliminate the problem of the cameras. (Difficulty: Medium)
- Investigate time and sampling patterns that may help a faster convergence. (Difficulty: ?)
- Manual generation of mipmaps, in order to include a border to limit the bleeding effect. (Difficulty: Medium)
- Multiple lights and introduction of point lights (Difficulty: Easy)

Next steps that I would like to take in the domain of *performance*:

- General optimization in shaders (mostly numerical)
- Random numbers using texture instead of deterministic GPU generation.
- Optimize the rendering to texture array/cubemap to be in only one pass.