DV2520 — Assignment 3 — Clouds

Deadline	2011-10-28 17:00
Submission	tarball including source, Makefile and report (in pdf) uploaded to It's Learning

1 Description

In this assignment you will look at implementing a volumetric rendering of clouds from Perlin noise. One part of the problem is replacing the back-end of the fixed-functionality pipeline in OpenGL with your own ray-casting algorithm over a simple primitive, building up a volumetric image to avoid the need for marching rays or scene sorting. The second challenge is to fill the transparent voxels with Perlin noise to simulate a cloud pattern. For full marks this should be sampled from a 4D noise function to allow the cloud to animate. For best results pick a large volume size: the minimum acceptable is $64 \times 64 \times 64$ over a 64 frame animation. You will need to decide on a method to generate psuedo random numbers on the GPU as seed data for the noise function: high quality randomness is not required and you should consider a stateless method for maximum performance.

2 Grading

The grade that you receive is dependent on you demonstrating the following:

Grade 3:

- Implement ray casting, shooting one ray per pixel into a scene.
- Implement some form of voxel scene storage with a resolution of at least $64 \times 64 \times 64$ voxels.
- Accumulate the contribution of each voxel that a ray intersects to compute the RGB colour components of a ray.
- Implement some form of camera motion to show the scene in 3D.
- Profile your kernel while varying the number of rays and the number of voxels to produce an estimate of the time to check a single intersection, and how the performance scales across varying number of rays and voxels.
- Write a report (in grammatically correct English).
- Describe in your report the approach that you implemented (in enough detail to allow another engineer to implement an equivalent approach).

- Describe in your report how many FLOPs does your kernel use to compute an intersection test?
- Describe in your report how does the number of FLOPs scale with the number of voxels / rays?
- Describe in your report what is the CGMA ratio of your kernel in terms of the number of voxels and rays?

Grade 4, as with grade 3 plus:

- Implement a psuedo-random number generator using floating point arithmetic on the GPU. The quality of the randomness is not important, but the sequence should be repeatable and not show any obvious patterns over the range that you require below.
- Implement a 3D Perlin Noise sampling function, using your random number generator to create the vectors on each point in the coarse grid. You will need to pick how many voxels lie in between these coarse points to produce a visually pleasing effect (i.e not too smooth, and not too uniformly random).
- Generate the colour / greyscale of each voxel as a sample from the Perlin Noise function to produce a volumetric rendering of the noise. Your goal is to produce a reasonable volumetric rendering of clouds, you may want to consider thresholding the opacity of the noise and weighting the volumetric accumulation by distance. Ken Perlin's notes cover other tweaks to produce specific visual effects.
- Discuss how many FLOPs does your kernel use to compute a noise sample?
- Discuss which parameters affect this number of FLOPs, and how the work scales in terms of them? e.g. consider number of octaves, number of dimensions ...

Grade 5, as with 4 plus:

- Implement a 4D Perlin Noise function, use the frame counter as the fourth-dimension. Pick an appropriate granularity between your discrete frame counts and the grid resolution in this dimension.
- Loop your animation over an appropriate number of frames.
- The combined performance of the raycasting / intersection and the noise sampling.

- Discuss the number of FLOPs in the combined system and how it scales with the relevant parameters.
- Discuss the number of memory accesses in the combined system and how it scales in the relevant parameters.
- Discuss the scaling of the CGMA relative to these parameters, and what it tells you about the potential benefits of the optimisation techniques you have seen on the course. In particular discuss which parts of the implementation are computation-bound and which parts are memory-bound as you vary the parameters.

3 Submission

You must submit your source-code in a tarball, and report as a `.pdf`, through It's Learning.