Julia Tucher - jjt4

Final Project Proposal

In my final project, I am planning to explore and optimize an algorithm for ray tracing volumetric data as developed by Marc Levoy. Also known as volumetric ray casting, this algorithm produces a two dimensional image output based on 3D data, incorporating color, opacity, and light sources to create a 2D model of a 3D scene[1]. This method is an alternative to rasterization of 3D images which utilizes a 3D geometric structure of complex scenes and has applications in medical imaging, architecture, and other graphics processing.

There are four main parts of the volumetric ray tracing algorithm: ray casting, sampling, shading, and compositing. Given a viewpoint on a 3D scene (call it a square dimensional image NxN), every pixel in the square image casts a ray into the volume space (which can be thought of as a cuboid space NxNxW). Then, a subray of each possible length along the cast ray is sampled, where sampling points may be aligned with or between voxels. A final sample is calculated by triangularly interpolating from nearby voxels. Because each sample point has a certain amount of illumination, there is an RBGA value at each point and gradient of illumination can be calculated, and the final value of each ray is shaded based on the orientation of the image's light source. Then, each pixel is calculated based on the composite value of all of the rays in that line of sight through a rendering equation.

I plan to implement the algorithm as a GPU program using CUDA. In the original papers discussing Levoy's ray tracing algorithm, the serialization of these processes occur results in very slow processing of complex scenes[2]. Using a GPU allows for individual computation of pixel values, which occur independently, to happen concurrently. Thus, using the GPU to parallelize such that each thread corresponds to a pixel and calculated each ray along that line of sight (executing each of the four steps as described above for a pixel). This method of parallelization uses the SIMD architecture of the GPU to optimize instead of reducing computation through retaining values in CPU memory (as done by other versions of the algorithm optimized for CPU).

For non-parallel optimizations, I will explore shared memory in CUDA, which should provide an actual decrease in runtime because of the way that sample points along a ray share voxel data with 7 other nearby rays. Thus, I plan to use at least four different block configurations to see how maximizing threads sent to each streaming multiprocessor impacts runtime, as with the last GPU assignment, but also because implementing shared memory may lead to various improvements in runtime depending on how many pixels can share data with neighboring pixels that may share corresponding nearby voxel data. Another non-parallel optimization I will examine is whether or not early termination of a ray improves runtime. Essentially, as a ray progresses, it may pass through voxels with high enough opacity that the rest of the ray is irrelevant. However, checking an opacity condition adds conditional branching that may affect runtime.

---

[1] "Volume Ray Casting." *Wikipedia*, Wikimedia Foundation, 4 Nov. 2019, en.wikipedia.org/wiki/Volume_ray_casting.

[2] Levoy, Marc. "Efficient Ray Tracing of Volume Data." *ACM Transactions on Graphics*, vol. 9, no. 3, 1990, pp. 245–261., doi:10.1145/78964.78965.