Lab #2 for CPE 458 & CPE 473

Triangle intersections and anti-aliasing

Due Thursday, May 5th, 2011

1. Orientation

This is a team based lab. Please continue with your established teams of a mix of 458 and 473 students. You are expected to help one another with understanding ray-tracing and CUDA. You will be required to hand in your lab and demonstrate specific profiling reports to be specified later.

1. Introduction

We will continue with building a ray tracer on both the CPU and GPU. You need to add code to your program to intersect rays with triangles and you will need to add the ability to turn on an off anti-aliasing (super-sampling). We will continue to profile your code (the pure CPU version versus the version with intersections done on the GPU).

2. Write ray triangle intersection for the CPU

To best integrate the team's knowledge, in a pair programming style, the 458 member(s) of the team should write the triangle intersection routine for the CPU.

3. Profile running on large triangle files

1. For example the bunny_jumbo_tris.pov file found here: http://users.csc.calpoly.edu/~zwood/teaching/csc473/index.html

4. Outsourcing triangle intersection tests to the GPU

We suggest writing a separate triangle intersection kernel (and not adding branching to your sphere intersection kernel) and again defining a separate C-style struct, `cuda_triangle_t` -- including the 3 vertices of the triangle. You will have to handle files that have both a mix of triangles and spheres, but start by just getting a GPU version with just triangles working. You have a choice about how you'd like to handle the indexing of objects. You can either use the same ordering from lab #1 (where the CUDA index will match the index of the object on the CPU, implying that on the CPU the lists of triangles and spheres are separate) or a version that keeps a mixed list of geometric objects on the CPU and then copies the triangles and spheres separately into the C-style arrays and copies an 'index' value into the array, where the index value represents the CPU index of the object.

To best integrate the team's knowledge, in a pair programming style, make sure that the 473 team member writes the GPU version of the triangle intersection code. Follow similar steps as those specified in Lab #1.

5. Measuring your speedup

We suggest that you re-build your program using gprof and look at where the program is now spending most of its time.

6. Add anti-aliasing

Now, write a CPU version of the ray tracer that allows for the addition 4 stratified super-sampled rays per pixel that are jittered within the pixel extent to the ray tracer to allow for anti-aliasing. Reconstruct the pixel color using a box filter (i.e. weight each ray ¼ of the color contribution). Test and profile your code with both the CPU only version of your code. Now augment your existing GPU code to include these super-sampled rays in your intersection tests. You will be asked to provide specific profiling results for the purely CPU anti-aliased results and the version using CUDA for intersections.