Path tracer

Usage

Building on the department machines is likely to be challenging and I wouldn't recommended trying to manually set up the environment needed to build this project. A docker container can be found at https://hub.docker.com/repository/docker/greenblattryan/path-tracer This project requires clang-10 (using c++2a), cuda 10+, boost, and libstdc++-10. The path to cuda is hard coded as /usr/local/cuda/ because this is a requirement to use clang as a cuda compiler. I am not providing a Makefile (the build system is CMake), but the "scripts/build" script (uses python) can be used to build. The script has help accessible with –help. By default, builds will be done in the directories "release/" and "debug/". Make sure you use –release if release mode is desired. A binary is included. Unfortunately, it doesn't run on department machines with error "libomp.so" not found, but if this is installed, the binary should run on most linux platforms.

There are several binaries produced in this project. The "final" binary runs the project. This has various command line options for configuring what gets run. By default, the binary will run on the cpu, but the –gpu flag can be used to run on the gpu (nvidia only, reasonably recent driver and gpu required). I haven't tested on all platform, so I am not sure if this will work everywhere. To modify which compute architecture the project builds for see cmake/Modules/CMakeCUDAInformation.cmake and edit "cuda-gpu-arch=sm_75" to the arch of choice.

Build times for this project are very slow for a few reasons, so don't be surprised if this is the case. Also note that the gpu progress bar is less precise that the cpu progress bar. It may take a while to see the first change (for very high poly scenes)

Features

Images can be found in "outputs/". Some test scenes can be found in "scenes/".

Extra features: - Custom acceleration data structure (kdtree). I am not sure if this is faster than the stencil version, I wanted to run some comparison benchmarks, but I ran out of time.

- GPU support - running on the gpu causes a 10-40x speed up (testing with a 12 core cpu and an RTX 2070). Let me know if you want me to demonstrate, because it may be difficult to get this to run on older gpus.
- BRDF importance sampling (examples in outputs)
- Better termination probability - rather than using a fixed probability, it may be desirable to vary the probability. I did some testing, and determined that a good form for this is to compute some function of the current multiplier

(the total product of values which will be multiplied by the light value). After some testing, I determined a good function was something of the form: termination prob = max(abs((1-|0.57 * m/(1 + m)|$^{2)}$n), min_term_prob) where m is the multiplier, 0.57 is a normalization constant, m / (1 + m) is used to map the output from 0 to 1, |x|ˆ2 is squared norm, and n is a constant. This function is convex for n $>$ 1, and I determined a good value was 10 for n and 0.05 for min_term_prob. Again, example images are in the output directory. Note the README in the outputs/term_prob with testing information.