raytracing with CUDA [closed]

Asked 16 years, 3 months ago Modified 9 years, 4 months ago

Viewed 18k times

11

Closed. This question needs to be more focused. It is not currently accepting answers.

Want to improve this question? Update the question so it focuses on one problem only by editing this post.

Closed 2 years ago.

The community reviewed whether to reopen this question 2 years ago and left it closed:

> Original close reason(s) were not resolved

Improve this question

I'm currently implementing a raytracer. Since raytracing is extremely computation heavy and since I am going to be looking into CUDA programming anyway, I was wondering if anyone has any experience with combining the two. I can't really tell if the computational models match and I would like to know what to expect. I get the impression that it's not exactly a match made in heaven, but a decent speed increasy would be better than nothing.

cuda raytracing

Share

Improve this question

Follow

asked Sep 2, 2008 at 13:12

Morten Christiansen

19.5k ● 23 ● 73 ● 96

Just a pointer to my open-source, portable (Windows/Linux) GPL implementation of a CUDA raytracer. – ttsiodras Sep 7, 2011 at 9:17

2 Answers

Sorted by: Highest score (default)

24

One thing to be very wary of in CUDA is that divergent control flow in your kernel code absolutely KILLS performance, due to the structure of the underlying GPU hardware. GPUs typically have massively data-parallel workloads with highly-coherent control flow (i.e. you have a couple million pixels, each of which (or at least large swaths of which) will be operated on by the **exact** same shader program, even taking the same direction through all the branches. This enables them to make some hardware optimizations, like only having a single instruction cache, fetch unit, and decode logic for each group of 32 threads. In the ideal case, which is common in graphics, they can broadcast the same instruction to all 32 sets of execution units in the same cycle (this is known as SIMD, or Single-Instruction Multiple-Data). They can **emulate** MIMD (Multiple-Instruction) and SPMD

(Single-Program), but when threads within a Streaming Multiprocessor (SM) diverge (take different code paths out of a branch), the issue logic actually switches between each code path on a cycle-by-cycle basis. You can imagine that, in the worst case, where all threads are on separate paths, your hardware utilization just went down by a factor of 32, effectively killing any benefit you would've had by running on a GPU over a CPU, particularly considering the overhead associated with marshalling the dataset from the CPU, over PCIe, to the GPU.

That said, ray-tracing, while data-parallel in some sense, has widely-diverging control flow for even modestly-complex scenes. Even if you manage to map a bunch of tightly-spaced rays that you cast out right next to each other onto the same SM, the data and instruction locality you have for the initial bounce won't hold for very long. For instance, imagine all 32 highly-coherent rays bouncing off a sphere. They will all go in fairly different directions after this bounce, and will probably hit objects made out of different materials, with different lighting conditions, and so forth. Every material and set of lighting, occlusion, etc. conditions has its own instruction stream associated with it (to compute refraction, reflection, absorption, etc.), and so it becomes quite difficult to run the same instruction stream on even a significant fraction of the threads in an SM. This problem, with the current state of the art in ray-tracing code, reduces your GPU utilization by a factor of 16-32, which may make performance unacceptable for your

application, especially if it's real-time (e.g. a game). It still might be superior to a CPU for e.g. a render farm.

There is an emerging class of MIMD or SPMD accelerators being looked at now in the research community. I would look at these as logical platforms for software, real-time raytracing.

If you're interested in the algorithms involved and mapping them to code, check out POVRay. Also look into photon mapping, it's an interesting technique that even goes one step closer to representing physical reality than raytracing.

Share Improve this answer Follow

edited Oct 20, 2008 at 6:12

community wiki
2 revs
Matt J

maybe time to update this. it was fun reading 10 years later and with nvidia doing all gpu raytracing – user-2147482637 Dec 20, 2019 at 3:58

9

It can certainly be done, has been done, and is a hot topic currently among the raytracing and Cuda gurus. I'd start by perusing
http://www.nvidia.com/object/cuda_home.html

But it's basically a research problem. People who are doing it well are getting peer-reviewed research papers out of it. But *well* at this point still means that the best GPU/Cuda results are approximately competitive with best-of-class solutions on CPU/multi-core/SSE. So I think that it's a little early to assume that using Cuda is going to accelerate a ray tracer. The problem is that although ray tracing is "embarrassingly parallel" (as they say), it is not the kind of "fixed input and output size" problem that maps straightforwardly to GPUs -- you want trees, stacks, dynamic data structures, etc. It can be done with Cuda/GPU, but it's tricky.

Your question wasn't clear about your experience level or the goals of your project. If this is your first ray tracer and you're just trying to learn, I'd avoid Cuda -- it'll take you 10x longer to develop and you probably won't get good speed. If you're a moderately experienced Cuda programmer and are looking for a challenging project and ray tracing is just a fun thing to learn, by all means, try to do it in Cuda. If you're making a commercial app and you're looking to get a competitive speed edge -- well, it's probably a crap shoot at this point... you might get a performance edge, but at the expense of more difficult development and dependence on particular hardware.

Check back in a year, the answer may be different after another generation or two of GPU speed, Cuda compiler development, and research community experience.

 answered Sep 2, 2008 at 13:54

Follow

Larry Gritz
13.6k ● 7 ● 43 ● 42

I have a small project building my first raytracer and have never done any CUDA work so I'm in a poor position to make anything great, but over the next year I'm working with GPGPU technology. This leads me to get familiar with CUDA and I was wondering to which extend I can use this knowledge. – Morten Christiansen Sep 19, 2008 at 15:42

1 Are you certain it is an embarrassingly parallel problem? Solving to find the next object of reflection and variations in material handling (as pointed out by Matt J) seem like they might break parallelism significantly. But please correct me if I am wrong. – user334911 Aug 27, 2013 at 22:08