

CIS 665: GPU Programming and Architecture

Gary Katz and Joseph Kider

February 18, 2008

1 Project Guidelines

The idea of this project is to take on a more intricate problem involving computations on the GPU, and demonstrate a plausible, efficient implementation of a particular problem.

The problem you select may be one of the choices listed in Section 3, or a problem of your own choosing. It must involve a nontrivial GPU computation at its core, with routines that require extensive use of shader programs or extensive use of CUDA kernels (more than 2 different ones, or do some non-trivial process). Students will work in groups of at most two (groups are strongly encouraged).

For each of the problems you choose, we will review background reading, relevant papers, current implementations etc. Contact us for more information. If you want to choose your own project (highly recommended), talk to us early on and we can point you to the appropriate literature.

The final project is worth a large portion of your final grade in the course (50%). Please allocate your time accordingly.

2 Project Requirements

1. Project choices must be made by Mar 17, 2007. At that time, you will submit a **detailed proposal** specifying the deliverables: problems that will be solved with working code (an example will be made available on blackboard).
2. Prior to Mar 17, 2008, each group will schedule at least **one meeting** with Joe or Gary to discuss their choice and review their proposal. You must have a proposal ready before this meeting: the purpose will be to discuss deliverables and make (possible) modifications in the proposal. This is especially important if you plan on choosing your own topic (be mindful of spring break (March 10-16) we will be away).
3. By April 14, 2008 you must meet with Joe and present an **project milestone report**. This should be a short writeup and an alpha version of your code. This version must show progress, and the goal should be specified in your specification proposal. The milestone report is designed to help you keep on-track and for us to provide feedback and advice. Describe your progress so far and your plan to finish. Keep it brief and to the point: enough for us understand what you are doing, assuming the material covered in class as background knowledge. Hopefully, you can reuse most of pieces the milestone report as a part of the final report. Small changes to the proposal can be reviewed at this time if there is a need.

4. **Presentations:** We will have a poster presentation during exam week (May 4-10 as late as we can make it for you). Each team should prepare a demo and be ready to give a short powerpoint presentation (15-20minutes) to the class. The Presentation session is a great opportunity for you to see other people's projects and get some last feedback before the final report. You will be expected to have your code ready to demo (or results if it is not a realtime algorithm) during the presentation, and be expected to answer reasonable questions on your methods by the class and instructors. (All students are expected to attend this presentation for your peers)
5. **Final report:** Final project reports should be emailed as PDF attachments to blackboard by noon May 10, 2008. Final code should be zipped and also sent via blackboard. (An example of a final report will appear on blackboard).
6. **Blog:** Every group is required to blog their progress at minimum once a week. You will open a blog on <http://www.blogger.com/home> and send Joe the URL so he can link your page to the main course website. This is a multiple week project and we expect that amount of work. We will read your blogs weekly and comment on your progress and answer questions we see with the work. As a side note, nothing goes according to plan. We know this. The blog will help facilitate problems and changes as you work on the project. This helps avoid major last minute problems. This part will be graded as part of your final report.

We plan to post all the final reports online so that you can read about other projects. Let us know if do not want your report to be posted (for example last year one group modified propriety software aGI).

We will evaluate your projects according to the following four criteria: Code: 10%, Report-15%, Presentation 15%, Proposal and milestone 10%.

In general we will look for the following:

Soundness: Are the claims technically correct and techniques and approaches reasonable for the problem?

Significance: Is the problem addressed important and/or interesting?

Novelty: Is there something new and interesting about the project (novel application, algorithm, analysis, evaluation)?

Clarity: Is the presentation clear and concise, but complete enough for someone familiar with a graphics background?

3 Possible Project Ideas

3.1 Computer Vision on the GPU

1. Real-Time, GPU-Based Foreground-Background Segmentation
2. A multi-view stereo evaluation has been proposed by Steve Seitz et al. The challenge involves recovering 3D reconstructions of complete objects from a large number of views. Among the reported techniques, two out of nine make an intensive usage of GPUs, both yielding large speedups: the work by Pons, Keriven and Labatut that took part in the original competition at CVPR06, and the work by Hornung and Kobbelt. Running times, accuracy and completeness of the methods are reported here. (Steve Seitz et al. A Comparison and Evaluation of Multi-View Stereo Reconstruction Algorithms, in IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), New York, 2006.)
3. Adding vision tools to OpenVIDIA Computer Vision on GPUs Project

3.2 Data or Image Compression on the GPU

Data/Image compression comes in general comes in two forms: lossy and lossless. This project would implement a compression scheme on the GPU. One example is as follows: Parallel fractal image compression using programmable graphics hardware. The main problem of fractal compression is the very high computing time needed to encode images. The implementation in the GPU exploits the SIMD architecture and inherent parallelism of recent GPUs to speed up the baseline approach of fractal encoding.

3.3 Rendering algorithms on the GPU

One of the following methods on the GPU:

1. Ray tracing is an image synthesis technique which simulates the interaction of light with surfaces. Most high-quality, photorealistic renderings are generated by global illumination techniques built on top of ray tracing. Real-time ray tracing has been a goal of the graphics community for many years. Unfortunately, ray tracing is a very expensive operation. VLSI technology has just reached the point where the computational capability of a single chip is sufficient for real-time ray tracing. Supercomputers and clusters of PCs have only recently been able to demonstrate interactive ray tracing and global illumination applications. Show how to improve Ray Tracing on the GPU in a stream program, and compare your methods to existing implementations
2. A real-time radiosity simulation running entirely on the GPU and comparison of your methods to existing implementations
3. Global illumination rendering methods can be implemented on GPU however do not follow the conventional local illumination model of DirectX/OpenGL pipelines, but require global

geometric or illumination information when shading a point. Implement Global illumination on the GPU or some major extension and compare your methods to existing implementations.

3.4 Graph algorithms on the GPU

There are many simple graph algorithms (connected components, breadth first-search) that run well in parallel and therefore might be implementable in the GPU. A toolkit of such algorithms would be very useful to have available for graph visualization. Note that many graph problems can also be formulated as matrix problems. Design GPU algorithms for one of the following problems: All-pairs shortest paths: given a graph and edge lengths, compute the length of the shortest path between each pair of vertices. Visualize the output as a matrix of values and visualize 1 and 2 connected components of a graph.

3.5 Infinite precision arithmetic

GPU channels are 32 bits only. For scientific computing (the space that makes most use of the GPU), this is woefully inadequate. However, there are currently no ways of performing extended precision computations on the GPU (for example, we cant specify doubles). Come up with an approach for doing extended precision computations on the GPU. You may do this by presenting modules that can perform basic arithmetic operations (addition/subtraction/multiplication/division) correctly extended precision, and then demonstrating the use of these modules in a problem (like matrix multiplication).

3.6 Other Ideas:

1. Volumetric Rendering in Real-Time
2. Fluid Flow Solver
3. Debugging on the GPU
4. Game AI on the GPU