

CIS 700: Project

Suresh Venkatasubramanian

1 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 2, or a problem of your own choosing. It must involve a nontrivial GPU computation at its core, with routines that require extensive use of fragment/vertex programs. Students will work in groups of **at most two**.

For each of the problems listed in Section 2, I will provide background reading, relevant papers, etc. Contact me for more information. If you want to choose your own project, talk to me and I can point you to the appropriate literature.

1.1 Requirements

1. Project choices must be made by **Mar 3, 2005**. At that time, you will submit a detailed proposal specifying the *deliverables*: problems that will be solved with working code.
2. Prior to Mar 3, each group will schedule *at least* one meeting with me 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.
3. The deliverables, along with working code, will include a report that describes in detail your implementation, strategies used/discarded, and analysis of code behaviour in terms of running time (frames/sec, seconds/computation etc) and quality of output.

You will also be expected to make a 30 minute presentation of your work, along with a short demo.

4. You will be evaluated on how well your final demo matches the deliverables in your proposal, the overall efficiency of your implementation (in a design rather than horse-race sense), and the quality of your report and presentation.

2 Proposals

This is a list of example proposal outlines. You are free to choose one of these, or come up with your own ideas. The topics are meant to be suggestions that open up an area of study: you are not required to follow these ideas to the letter; your proposal will specify what you plan to implement.

2.1 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.

2.1.1 Project idea:

Design GPU algorithms for one of the following problems:

- 1- (and 2-) connected components of a graph (a k -connected component of a graph is a subset of vertices and edges that cannot be disconnected by the removal of $k - 1$ edges). Visualize the output by drawing the graph and coloring components in distinct colors.
- 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 (i, j) where (i, j) is the distance between i and j

2.2 General Matrix operations

All kinds of basic matrix operations have been implemented, but none of the advanced operators, like matrix inversion and various decomposition techniques (Cholesky, QR, SV). Many problems in data stream analysis make use of general matrix operators and a GPU-based toolkit might be quite useful.

2.2.1 Project Idea:

Design an algorithm that implements some subset of the following: Singular Value Decomposition, Cholesky decomposition, QR decomposition, Eigenvector calculations, matrix inversion.

2.3 Superfast Matrix multiplication

There are now many different schemes for matrix multiplication that work in different settings, for different kinds of inputs. However, we have no idea which scheme works best, and which schemes work better than the CPU implementations in BLAS etc.

2.3.1 Project Idea:

Implement the best matrix multiplication scheme possible. Fine tune all parameters, compare and contrast different methods. The program should be able to take two large matrices, decide what parts of the computation are best done on the CPU and which are best done on the GPU.

2.4 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 can't specify doubles).

2.4.1 Project Idea:

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 in extended precision, and then demonstrating the use of these modules in a problem (like matrix multiplication).

2.5 Image processing on the GPU

Image processing and vision is another area where GPU algorithms can be of great use. A basic image processing toolkit accelerated by the GPU would be very useful in real-time image processing.

2.6 Project Idea:

Choose a few basic image processing primitives (edge detection, convolution, image segmentation) and implement them on the GPU. A good demo would be one that could take frames from a video stream and analyze them on the fly.

2.7 Pseudorandomness

For many applications, the best algorithms make use of a source of random bits (computer security is a good example). The GPU currently has no capability to generate reasonably high quality pseudorandomness.

2.7.1 Project Idea:

Implement a scheme for generating pseudorandomness on the GPU. you can use linear congruential generators or another suitable algorithm. The code should have srand and drand like functionality for use by an application. For example, I'd want to be able to call drand from a Cg fragment or vertex program.

2.8 Volumetric Rendering

While volumetric geometry such as clouds and explosions have reached an impressive level of believability in film visual effects, their real time counterparts suffer from the many hacks and cheats used to create them cheaply. With the advent of programmable GPU's however, it is possible to implement the same shading algorithm used by the high-end fx studios, in real time (or close to it).

2.8.1 Project Idea:

Design GPU implementations of the following algorithms:

- Ray Marching on Sphere primitives
- 3D density functions (Fractional Brownian Motion, Turbulence, Perlin Noise, etc.)
- Shading acceleration using 3D textures.

For reference, see "Advanced Renderman", pages 314-319 and 417-431

2.9 Measured BRDF Shading Models

Specular, Diffuse, Ward, and other common shading models used in graphics are only approximations of how real world materials interact with light. Their true appearance can be more precisely characterized using a BRDF (Bi-Directional Reflectance Distribution Function). A BRDF specifies the light energy reflected for every combination of incident and reflected light directions. BRDF's can be measured for real world surfaces using a gonioreflectometer, and tables of such data are freely available from many sources. Several vfx companies and research groups have implemented shaders which read this data, but to this date, none have been optimized for real-time rendering on the GPU.

2.9.1 Project Idea:

Design data-driven BRDF shading model for the GPU, so that the data can be stored efficiently on the GPU and can be queried efficiently at render time.

2.10 View Dependent Displacement Mapping

The holy grail of texture based modeling is a technique known as displacement mapping (a texture which actually moves surface points). Until recently, the only known technique for displacement on a GPU was excessively tessellating the geometry and moving the vertices. In SIGGRAPH 2003, however, researchers from Microsoft developed a technique known as "View Dependent Displacement Mapping" which precomputes displacement from a series of views and creates displacement "silhouettes" by modulating opacity in the fragment program.

2.10.1 Project Idea:

Implement a system for creating view dependent displacement maps (hint: maya plugin-in, Renderman Shader, RenderTexture, etc.). Create a system to render displaced geometry using view dependent displacement maps on a GPU.

2.11 Soft Shadows

Lighting and shadows continue to be some of the most difficult visual effects to simulate in real time. Current technology handles hard-edged shadows very well, but soft shadows with depth dependent blur remain a complicated problem. In SIGGRAPH 2003, researchers from Chalmers University of Technology, Sweden, developed a "Geometry-based Soft Shadow Volume Algorithm using Graphics Hardware", which addresses this issue and achieves real time results.

2.11.1 Project Idea:

Implement Geometry-based Soft Shadow Volumes.

2.12 Shader Optimization (Compositing Layered Shading Models)

It has long been a goal of the computer graphics industry to make shader writing more art directed rather than technology oriented. However the most beautiful shaders are often the slowest to run. When writing shaders to reproduce a material, most designers think best in terms of layers of materials (e.g., the facade of an old sign could be thought of as dust on top of chipped paint on top of rust on top of metal). However, such shaders involve the computation of lighting terms for each layer, which can bring frame rates to crawl. Studios are currently investigating methods to preserve the "look" of such shading models while optimizing their performance (the best of both worlds).

2.12.1 Project Idea:

Research and implement a scheme to optimize layered shading models for the GPU.

- Create a system for layered shading on a GPU and write some example shaders.
- Create a system for optimizing these shading models and compare both frame rates and appearance (hint: the most difficult effect to optimize are the view dependent lighting terms like specular and rim).