Hacking the GPU for Scientific Computing
A Journal of Notes and Ideas

Peter Schröder
Caltech

August 14, 2002

1 Algorithms

What algorithms are of interest? What algorithms are reasonable to want to run on the GPU?

1.1 Conjugate Gradient Solvers

What primitives are needed for this? In what generality do we want to implement these. For the skinny on CG see [12].

In order to demonstrate the utility of this algorithm we'll want to show a number of different exmaples of it in action. Some may require CG others Bi-CG yet others will likely benefit from online recomputation of the matrix A. Potential applications:

- **Mesh Smoothing:** This could make for a good interactive demonstration in which we will both solve and display interactively the result. Probably the best algorithm for this can be found in Desbrun *et al.* [5].
- **Parameterization:** There are multiple flavors here which all come down to basically the same matrix but slightly different ways to compute its entries. For the nicest functionals we should use Desbrun *et al.* [4] (both their conformal and authalic). Of interest here are the different boundary conditions. The easy one is Dirichlet. The nice one is Neumann, though it requires coupling of the two systems for the u and v coordinates. Can this somehow be broken out? Another nice functional is that of Floater [7] as it has all positive coefficients.
- **Optimization:** I am less clear on this, but perhaps we can use some variational optimization example as an application from geometric modeling.
- **What else?** Other examples?

1.1.1 Preconditioning

Many of these problems are very ill-conditioned when the problem size grows. We should include at least some preconditioners. Certainly diagonal. What others are reasonable? I killer one would be hierarchical preconditioning. But this requires building some kind of hierarchy on the residual vector. However, maybe the CPU can build the hierarchy and then send it down in some fancy manner. Other methods are partial LU and Cholesky. I have no idea what's involved in making those work.

1.2 Subdivision Surfaces

Everyone will look for this. It's a clear and obvious winner. It will also make for a nice architectural comparison with the CPU paper

1.2.1 Table Driven Evaluation

Along the lines of Bolz and Schröder [1]. Issues:

- How do we pack tables into textures? What does this mean for texture offset computations? The tradeoff here is that we can only pass so many texture coordinates into the fragment program. If each table had its own texture we'd need only one interpolated texture coordinate, but then we'd need to have all the necessary handles. We likely don't have enough of those. How does one describe this trade-off and what kind of optimization problem does this result in. Is this come kind of nasty vertex coloring problem? Register allocation under serious constraints?
- How to deal with different subdivision depths? One set of precomputed tables or multiple passes?
- Render to vertex buffer? How does this work?
- Avoiding bit errors on the patch boundaries. This will require careful enforcement of all symmetries and exact zeros (were applicable) in the basis function tables in a pre-process. What kinds of symmetries do we want to exploit? What will this do to texture coordinate setup? In particular we need to enforce a global order on the basis functions which needs to be reflected in the local compositing order to enforce bit accurate results on the boundary.

1.2.2 Recursive Evaluation

This is much more challenging. How to do this in a streaming context? We don't want to just split each quad into four quads and keep going. This would imply very large numbers of recomputations and have very low arithmetic intensity. Should we follow some of the ideas of Pulli and Segal [9]? If so, what would that look like here? Or do we bite the bullet and figure out how to do expanding topological datastructures?

1.3 PDEs

What would make for interesting examples?

1.3.1 Stable/Simple Fluids

It looks like the Stam technique [13] is straightforward to implement (we need a solver, but the CG solver may just fit the bill...). Another possibility would be to use the Fourier technique he describes [14] if we have a decent Fourier (Cosine? Sine?) transform.

1.3.2 Thinshells

Cloth, paper, alumninum foil, metal cans, etc. come to mind. Of course one of the tricks will be to have collision detection. How to do that?

Subdivision style [8] This would be great for arithmetic intensity. Could be a fair amount of effort though with not clearly defined pay-off. In any case, it is probably valuable to at least think this through just to learn about the issues.

Discrete differential operator style Energies defined in terms of discrete curvature operators. Probably fairly good arithmetic intensity, hopefully not too hard to implement, and would further add whizbang factor to the work of Eitan and Anil.

1.4 Molecular Dynamics

These typically have very high arithmetic intensity. Need to hear more details on this from Steve Mayo though.

1.5 Fourier Transforms

An obvious candidate for a standard tool. Might need to read up on how this was done on the CM or how this is done in processor networks to understand optimal flow of this.

1.6 Wavelet Transforms

Another standard tool that comes to mind. The kicker would be incorporation of normal frame parameterized details. How would one do a decoder on the GPU? Implement the FSM? That sounds hard.

1.7 Sorting

Another standard tool. May need this at times to get around scatter limitation.

1.8 Reductions and Scans

Definitely need this to evaluate norms of vectors. Thinking about some general way to set this up is probably a good idea as this will be useful for all manner of algorithms. One should also consider partial sum reductions and not just "sum," but a number of more complex functions. In fact, maybe one can write this down in a way that it becomes possible to pass in a function.

1.9 Collision Detection

I don't even know where to begin here. Can we have some hierarchical datastructure in memory for this? If yes, how do we dynamically update it without scatter? Maybe the recent ray tracing papers offer some insight here [10, 2] or maybe some older ones [6, 11, 3].

1.10 Volume Rendering

Is texture compositing still the right idea here? Or should we really compute 1D line integrals through the volume?

Acknowledgements This work was supported in part by NSF (DMS-0138458, DMS-0220905, ACI-9982273) and the DOE (W-7405-ENG-48/B341492). Other support was provided by nVidia, Intel, Alias|Wavefront, Pixar, Microsoft, and the Packard Foundation. Special thanks to Matt Papakipos, Henry Moreton, Michael Cox, Ian Buck, and Pat Hanrahan.

References

[1] BOLZ, J., AND SCHRÖDER, P. Rapid Evaluation of Catmull-Clark Subdivision Surfaces. In *Proceedings of Web3D*, 11–17, 2002.

[2] CARR, N. A., HALL, J. D., AND HART, J. C. The Ray Engine. In *Proceedings Graphics Hardware*, 2002.

[3] DELANY, H. C. Ray Tracing on a Connection Machine. In *Proceedings of the 2nd International Conference on Supercomputing*, 659–667, 1988.

[4] DESBRUN, M., MEYER, M., AND ALLIEZ, P. Intrinsic Parameterizations of Surface Meshes. In *Proceedings of Eurographics*, 2002.

[5] DESBRUN, M., MEYER, M., SCHRÖDER, P., AND BARR, A. Implicit Fairing of Irregular Meshes using Diffusion and Curvature Flow. In *Proceedings of SIGGRAPH*, 317–324, 1999.

[6] DRUCKER, S. M., AND SCHRÖDER, P. Fast Radiosity Using a Data Parallel Architecture. In *Third Eurographics Workshop on Rendering*, 247–258, 1992.

[7] FLOATER, M. Mean Value Coordinates. Tech. rep., SINTEF, Oslo, 2002. Available as http://www.oslo.sintef.no/am/Geom/Homepages/mif/pubs/mean_value.ps.

[8] GRINSPUN, E., KRYSL, P., AND SCHRÖDER, P. CHARMS: A Simple Framework for Adaptive Simulation. *ACM Transactions on Graphics 21*, 3 (2002), 281–290.

[9] PULLI, K., AND SEGAL, M. Fast Rendering of Subdivision Surfaces. In *Rendering Techniques '96*, 61–70, 1996.

[10] PURCELL, T. J., BUCK, I., MARK, W. R., AND HANRAHAN, P. Ray Tracing on Programmable Graphics Hardware. *ACM TOG 21*, 3 (2002), 703–712.

[11] SCHRÖDER, P., AND DRUCKER, S. M. A Data Parallel Algorithm for Raytracing of Heterogeneous Databases. In *Proceedings of Graphics Interface*, 167–175, 1992.

[12] SHEWCHUK, J. R. An Introduction to the Conjugate Gradient Method Without the Agonizing Pain. Available at http://www-2.cs.cmu.edu/~jrg/jrspapers.html#cg, 1994.

[13] STAM, J. Stable Fluids. In *Proceedings of SIGGRAPH*, 121–128, 1999.

[14] STAM, J. A Simple Fluid Solver based on the FFT. *Journal of Graphics Tools 6*, 2 (2001), 43–52.