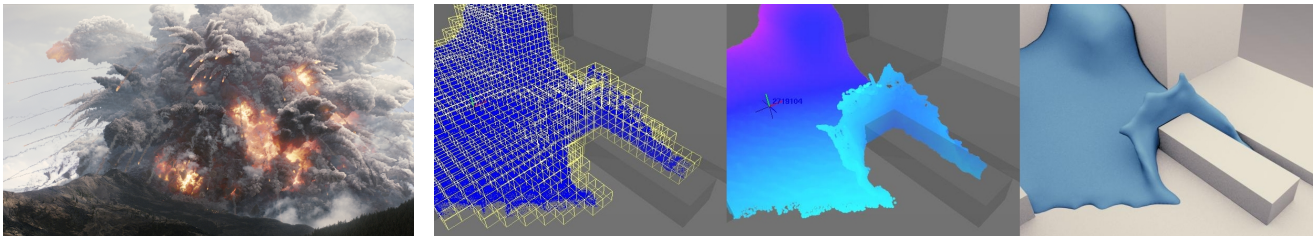
GPU Fluids in Production: Accelerating the Pressure Projection

Dan Bailey*
Double Negative

Ian Masters†
Double Negative



From left to right: *Smoke simulations from 2012 © 2009 Columbia Pictures Industries, Inc., Liquid simulation using CUDA-based linear solver - GPU blocks shown in yellow containing 256 cells each, FLIP particles previewed as points, meshed and rendered fluid surface*

Abstract

Effects such as billowing smoke, raging fire or turbulent water have been increasingly represented in recent movies. While fluid simulation techniques have benefited from research advances, they remain computationally expensive. We introduced GPU acceleration to our proprietary fluid solver, Squirt[1] as the first step towards solving complex production scenes in reasonable simulation time.

1 Introduction

The pressure projection is typically the bottleneck in a fluid simulation as it requires a slow, iterative process to enforce the incompressibility condition of the Navier Stokes equations. Our linear solver uses the preconditioned conjugate gradient method to achieve this. However, choosing an effective preconditioner which also performs well in parallel is notoriously hard.

To address this, we developed a strategy for parallel preconditioning specifically designed for use in fluid simulations. By avoiding generic sparse matrix representations and exploiting the strengths of current NVidia GPU architecture, we deployed a CUDA-based linear solver for use in production that exhibits significant increases in speed over the best available alternatives on the CPU.

2 Our Approach

Our preconditioning uses a variation of the outer-product form of the AINV preconditioner [Benzi & Tuma], to construct A-conjugate sets of vectors from the standard basis. Our major contribution is to rely on consistent offsets to access data within padded 3D grids on the GPU instead of depending on a drop tolerance to denote the sparsity in a generic 2D sparse matrix representation.

In addition, we developed a block-dropping strategy which drops any blocks not containing fluid. Aside from memory allocation, the time for performing the linear solve therefore becomes purely dependent on the region of simulated fluid rather than on the size of the fluid domain.

In memory, the fluid cells are arranged in contiguous blocks of cells, their size matching the number of threads to be used per block on the GPU with the grid resolution padded to meet the edges of the blocks. As most of the calculations for a cell involve using values from neighbouring cells, the whole block can be pulled into shared memory at once using coalesced reads. This reduces the overhead of global memory accesses. Note that the indices of the cell blocks that contain fluid are stored in a cache at the start of the solve.

The AINV algorithm builds a factorised approximate inverse:

$$M = ZD^{-1}Z^T \approx A^{-1} \tag{1}$$

A being the matrix to invert, M its approximate inverse, Z an upper triangular matrix and D a diagonal matrix containing the pivots.

During the A-orthogonalization process, there is a direct trade-off between reducing memory and forming M such that it is closer to the inverse of A. While a traditional approach is to drop elements below a defined threshold to enforce sparsity, we propose to calculate the pivots and six off-diagonal values (i, j, k, ij, ik, jk), regardless of whether they exceed this threshold, storing them in separate 3D grids. These values are then accessed directly on the GPU when applying the preconditioner using offsets that have been calculated from the domain size. While the resulting preconditioner is slightly less effective than when using a threshold that would produce similar sparsity, the cost of calculating a few extra iterations is vastly outweighed by being able to take advantage of faster GPU memory regions.

Dense Smoke	Res	Its	P-time	Its-time	T-time
CPU PCG(MIC)	100^3	48	0.328	2.51	2.84
GPU PCG(AINV)	100^3	186	0.350	0.803	1.15
CPU PCG(MIC)	200^3	92	1.25	39.9	41.2
GPU PCG(AINV)	200^3	369	3.62	11.5	15.1
Sparse Liquid	**Res**	**Its**	**P-time**	**Its-time**	**T-time**
CPU PCG(MIC)	100^3	16	0.194	0.635	0.635
GPU PCG(AINV)	100^3	49	0.139	0.0432	0.182
CPU PCG(MIC)	200^3	27	0.327	8.16	8.16
GPU PCG(AINV)	200^3	98	0.200	0.441	0.641

Comparison of our GPU linear solver with a threaded CPU solver preconditioned with Modified Incomplete Cholesky, showing the formation time (P-time), iteration time (Its-time) and total time (T-time) for dense smoke and sparse liquid simulations.

References

M. BENZI, J., AND TUMA, M. 2000. Robust approximate inverse preconditioning for the conjugate gradient method. *J. Sci. Comput. 22*, 4, 1318–1332.

*e-mail: drb@dneg.com
†e-mail: iim@dneg.com
[1] originally co-authored by Marcus Nordenstam and Robert Bridson

SIGGRAPH 2010, Los Angeles, California, July 25 – 29, 2010.
ISBN 978-1-4503-0210-4/10/0007