Eurographics/ ACM SIGGRAPH Symposium on Computer Animation (2010)
M. Otaduy and Z. Popovic (Editors)

A parallel multigrid Poisson solver for fluids simulation on large grids

A. McAdams[1,2], E. Sifakis[1,3], and J. Teran[1,3]

[1]University of California, Los Angeles
[2]Weta Digital [3]Walt Disney Animation Studios

Abstract

We present a highly efficient numerical solver for the Poisson equation on irregular voxelized domains supporting an arbitrary mix of Neumann and Dirichlet boundary conditions. Our approach employs a multigrid cycle as a preconditioner for the conjugate gradient method, which enables the use of a lightweight, purely geometric multigrid scheme while drastically improving convergence and robustness on irregular domains. Our method is designed for parallel execution on shared-memory platforms and poses modest requirements in terms of bandwidth and memory footprint. Our solver will accommodate as many as $768^2\times1152$ *voxels with a memory footprint less than* 16*GB, while a full smoke simulation at this resolution fits in* 32*GB of RAM. Our preconditioned conjugate gradient solver typically reduces the residual by one order of magnitude every* 2 *iterations, while each PCG iteration requires approximately* 6.1*sec on a* 16*-core SMP at* 768^3 *resolution. We demonstrate the efficacy of our method on animations of smoke flow past solid objects and free surface water animations using Poisson pressure projection at unprecedented resolutions.*

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation; G.1.8 [Numerical Analysis]: Finite difference methods—Multigrid and multilevel methods

1. Introduction

Incompressible flows have proven very effective at generating many compelling effects in computer graphics including free surface, multiphase and viscoelastic fluids, smoke and fire. One of the most popular techniques for enforcing the incompressibility constraint is MAC grid based projection as pioneered by [HW65]. They showed that for simulations on a regular Cartesian lattice, staggering velocity components on cell faces and pressures at cell centers gives rise to a natural means for projecting such a staggered velocity field to a discretely divergence free counterpart. This projection is ultimately accomplished by solving a Poisson equation for the cell centered pressures. Categorization of the Cartesian domain into air, fluid or solid cells gives rise to what [BBB07] describe as the "voxelized Poisson equation" with unknown pressures and equations at each fluid cell, Dirichlet boundary conditions in each air cell and Neumann boundary conditions at faces separating fluid and solid cells. This technique for enforcing incompressibility via voxelized Poisson was originally popularized in computer graphics by [FM96] who used Successive Over Relaxation (SOR) to solve the system. [FF01] later showed that incomplete Cholesky preconditioned conjugate gradient (ICPCG) was more efficient. This has since become a very widely adopted approach for voxelized Poisson and has been used for smoke [SRF05, MTPS04, FSJ01] and fire [HG09] as well as continuum models for solids such as sand [ZB05] and hair [MSW*09]. The voxelized pressure Poisson equation is also popular in free surface flows due to its efficiency and relative simplicity [CMT04, FF01, GBO04, HK05, HLYK08, KC07, MTPS04, NNSM07, WMT05, ZB05]. There are alternatives to voxelized Poisson, e.g., [BBB07] used a variational approach that retains the same sparsity of discretization while improving behavior on coarse grids for both smoke and free surface. Also, many non-MAC grid based approaches have been used to enforce incompressibility, typically for smoke rather than free surface simulations. For ex-

DOI: 10.2312/SCA/SCA10/065-073

ample, [MCP*09, KFCO06, FOK05, ETK*07] use conforming tetrahedralizations to accurately enforce boundary conditions, [LGF04] uses adaptive octree-based discretization, and [CFL*07] makes use of tetrahedralized volumes for free surface flow.

Incomplete Cholesky preconditioned CG remains the state of the art for voxelized Poisson. Although ICPCG certainly offers an acceleration over unpreconditioned CG, it is still impractical at high resolutions since the required number of iterations increases drastically with the size of the problem, and the cost of updating and explicitly storing the preconditioner is very high for large domains. Furthermore, since ICPCG involves sparse back-substitution, it cannot be parallelized without further modification [HGS*07]. Whereas multigrid methods have the potential for constant asymptotic convergence rates, our purely geometric multigrid preconditioned conjugate gradient algorithm is the *first* in the graphics literature to demonstrate these convergence rates on the highly irregular voxelized domains common to free surface simulation.

Our main contributions

We handle an *arbitrary mix of Dirichlet and Neumann boundary conditions* on *irregular voxelized domains* common to free surface flow simulations. Previous works proposing geometric multigrid methods for rectangular domains [TO94, AF96, BWRB05, BWKS06, KH08], as well as irregular Neumann boundaries (via a projection method) [MCPN08], cannot handle this general case.

We use a *purely geometric, matrix-free formulation* and *maintain constant convergence rates*, regardless of boundary geometry. [BWRB05] presents a lightweight, matrix-free geometric multigrid method for uniform rectangular domains; however, extension to voxelized domains has proved challenging. In fact, [HMBVR05] chose a cut-cell embedded boundary discretization, citing non-convergence on the voxelized Poisson discretization due to instabilities at irregular boundaries. Even higher order embeddings do not ensure constant convergence as noted by [CLB*09], where they solve the Poisson equation on a surface mesh embedded in a background lattice: they noticed convergence deterioration when the coarse grid could not resolve the mesh. Furthermore, a number of works have opted to use the more general algebraic multigrid, despite its significantly greater set-up costs and difficult to parallelize nature [TOS01], due to the challenges of finding a convergent geometric solution [CFL*07, CGR*04, PM04].

In comparison to widely-used ICPCG solvers, our method *requires less storage, is more convergent, and more parallelizable*. Unlike ICPCG, no preconditioning matrices are explicitly stored. This allows us to accommodate as many as $768^2 \times 1152$ voxels with less than 16GB of memory. Whereas ICPCG requires sparse backsubstition and cannot be parallelized without significant modifications, this is not the case for our solver. Finally, we demonstrate that our solver is significantly faster than ICPCG on moderately sized problems, and our convergence rates are independent of grid size, so this performance comparison improves with resolution. As a result, we can solve (with residual reduction by a factor of 10^4) at 128^3 resolution in less than 0.75 seconds and 768^3 resolution in about a minute.

2. Related Work

Multigrid methods have proven effective for a number of applications in graphics including image processing [KH08, BWKS06, RC03], smoke simulation [MCPN08], mesh deformation [SYBF06], thin shells [GTS02], cloth simulation [ONW08], fluid simulation [CFL*07, CTG10], geometry processing [NGH04], deformable solids simulation [OGRG07], rendering [Sta95, HMBVR05], and have been efficiently ported to the GPU [BFGS03, GWL*03, GSMY*08]. While the majority of works in this area either focused on rectangular domains or algebraic multigrid formulations on irregular geometries, there are some notable geometric multigrid solutions for irregular domains. [HMBVR05] and [CLB*09] embed their irregular geometry into regular lattices, producing irregular stencils. [RC03, ONW08] use a coarse-to-fine geometric hierarchy which ensures mesh alignment across levels, but these methods require the geometry to be fully resolved at the coarsest level and thus are impractical for resolving fine fluid features. [SAB*99] show that using multigrid as a preconditioner to CG rather rather than a stand-alone solver can alleviate instability issues on some problems. Multigrid preconditioned CG for the Poisson equation on rectangular grids can be found in [Tat93] and the algorithm is parallelized in [TO94] and later [AF96]. [KH08] introduced a higher-order parallel multigrid solver for large rectangular images.

3. Method overview

We wish to solve the Poisson boundary value problem:

$$\begin{gathered} \Delta p = f \ \text{ in } \ \Omega \subset \mathbf{R}^3 \\ p(\mathbf{x}) = \alpha(\mathbf{x}) \ \text{ on } \ \Gamma_D, \qquad p_n(\mathbf{x}) = \beta(\mathbf{x}) \ \text{ on } \ \Gamma_N \end{gathered} \tag{1}$$

where the boundary of the domain $\partial\Omega = \Gamma_D \cap \Gamma_N$ is partitioned into the disjoint subsets Γ_D and Γ_N where Dirichlet and Neumann conditions are imposed, respectively. For fluids simulations, Ω corresponds to the body of water, while Dirichlet boundary conditions are imposed on the air-water interface and Neumann conditions at the surfaces of contact between the fluid and immersed objects (or the walls of a container). Without loss of generality we will assume that the Neumann condition is zero ($\beta(\mathbf{x})$=0) since non-zero conditions can be expressed by modifying the right hand side.

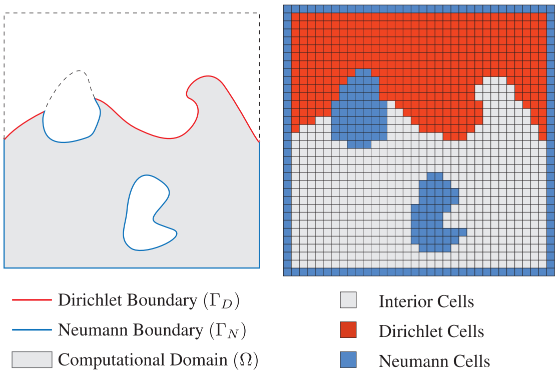

Dirichlet Boundary (Γ_D)
Neumann Boundary (Γ_N)
Computational Domain (Ω)
Interior Cells
Dirichlet Cells
Neumann Cells

Figure 1: *Left: Example of the geometry for a Poisson problem. Right: Our voxelized representation of this computational domain.*

3.1. Discretization

We discretize the Poisson equation on a uniform Cartesian lattice, and store the unknown pressure variables p_{ijk} at the cell centers of this lattice. In our approach, both the computational domain and boundary conditions are defined at the resolution of this background lattice. In particular, the computational domain is represented as a collection of grid cells (or voxels), which we refer to as *interior cells* (colored gray in the example of Figure 1, right). We express boundary conditions in a volumetric, voxel-based fashion. Specifically, we define Dirichlet conditions on a voxelized volumetric region, instead of imposing them strictly along $\partial\Omega$. We label the cells in this region as *Dirichlet cells* (colored red in Figure 1, right). We similarly define regions of *Neumann cells* (shown in blue in Figure 1) by rasterizing solid objects and container walls onto the grid. Since pressures are cell-centered, Neumann conditions are naturally imposed at the cell faces between interior and Neumann cells. For simplicity of implementation, as detailed in Section 4, we always assume a "ghost" layer of Neumann cells surrounding our computational domain, as illustrated in Figure 1. This assumption does not hinder generality since any configuration (e.g., a box with all Dirichlet boundaries) can be trivially padded with an extra layer of Neumann cells without affecting the solution on Ω. For every interior cell we construct a discrete Poisson equation as:

$$\sum_{(i',j',k')\in\mathcal{N}^*_{ijk}} \frac{p_{i'j'k'}-p_{ijk}}{h^2} = f_{ijk} \qquad (2)$$

$$\mathcal{N}_{ijk} = \{(i\pm1,j,k),(i,j\pm1,k),(i,j,k\pm1)\}$$

$$\mathcal{N}^*_{ijk} = \{(i',j',k')\in\mathcal{N}_{ijk} : \text{cell}(i',j',k') \text{ is } \textit{not} \text{ Neumann}\}$$

In this definition, $\mathcal{N}_{ijk}$ are the 6 neighboring cells across the faces of the interior cell (i,j,k), $\mathcal{N}^*_{ijk}$ is the subset of $\mathcal{N}_{ijk}$ that excludes any Neumann neighbor cells, and h is the grid step size. Equation (2) is derived from the standard 7-point Poisson finite difference stencil using the zero-Neumann boundary condition $(p_{i'j'k'}-p_{ijk})/h=0$ to eliminate pressure values at Neumann cells from the discrete equations. Away from the boundary of Ω this yields the standard 7-point stencil. Note that this formulation is identical to [FM96, FF01].

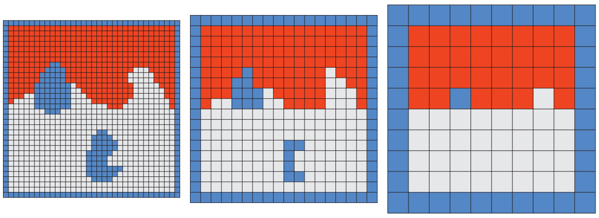


Figure 2: *Illustration of our coarsening strategy. Note the geometric and topological discrepancy between different resolution levels.*

3.2. Multigrid cycle

We describe a geometric multigrid method for the Poisson problem defined in (2). Our approach uses a V-Cycle of the Multigrid Correction Scheme [TOS01] and the pseudocode for each V-Cycle iteration is given in Algorithm 1. The V-Cycle procedure requires a discretization of the Poisson problem at $L+1$ levels of resolution, denoted $\mathcal{L}^{(0)},\mathcal{L}^{(1)},\dots,\mathcal{L}^{(L)}$, where level 0 is at the finest resolution and level L is the coarsest. We also define a smoother at every level of the hierarchy and restriction/prolongation operators for transferring data across resolutions.

Algorithm 1 V-Cycle of the Multigrid Correction Scheme. *Setting the initial guess to zero (line 2) is only necessary when the V-Cycle is used as a preconditioner. When the V-Cycle is used as an iterative solver line 2 should be omitted.*

```
1:  procedure V-CYCLE($\mathbf{u}^{(0)}$,$\mathbf{b}^{(0)}$)    ▷ total of $L+1$ levels
2:      $\mathbf{u}^{(0)} \leftarrow \mathbf{0}$
3:      for $l = 0$ to $L-1$ do
4:          Smooth($\mathcal{L}^{(l)}$,$\mathbf{u}^{(l)}$,$\mathbf{b}^{(l)}$)
5:          $\mathbf{r}^{(l)} \leftarrow \mathbf{b}^{(l)} - \mathcal{L}^{(l)}\mathbf{u}^{(l)}$
6:          $\mathbf{b}^{(l+1)}$ ←Restrict($\mathbf{r}^{(l)}$), $\mathbf{u}^{(l+1)} \leftarrow 0$
7:      end for
8:      Solve $\mathbf{u}^{(L)} \leftarrow (\mathcal{L}^{(L)})^{-1}\mathbf{b}^{(L)}$
9:      for $l = L-1$ down to 0 do
10:         $\mathbf{u}^{(l)} \leftarrow \mathbf{u}^{(l)}$+Prolongate($\mathbf{u}^{(l+1)}$)
11:         Smooth($\mathcal{L}^{(l)}$,$\mathbf{u}^{(l)}$,$\mathbf{b}^{(l)}$)
12:     end for
13: end procedure
```

Our approach is purely geometric: at every level of the multigrid hierarchy we construct a voxelized description of our domain, classifying each cell as interior, Dirichlet or Neumann. The procedure of Section 3.1 is then followed to construct a discrete Poisson operator $\mathcal{L}$ from this voxelized representation. The background lattice for each coarser level of the hierarchy is constructed following a standard 8-to-1 cell coarsening procedure, generating a grid with twice the step size. Our hierarchy is deep: the coarsest level is typically $8\times 8\times 8$. The coarse grid is positioned in such a way that the bounding box of the domain, excluding the ghost layer of Neumann cells, aligns with cell boundaries at both resolutions (see Figure 2). A coarse cell will be labeled Dirichlet, if any of its eight fine children is a Dirichlet cell. If none of the eight children is a Dirichlet cell, but at least one is interior, then the coarse cell will be labeled as interior. Otherwise, the coarse cell is labeled Neumann. As seen in Figure 2 this coarsening strategy can create significant geometric discrepancies when fine grid features are not resolved on the coarse, and even change the topology of the domain, e.g., Neumann bubbles are eventually absorbed into the interior region, and thin interior features are absorbed into the Dirichlet region. The impact of this discrepancy is addressed later in this section.

We construct the restriction operator as the tensor product stencil $\mathcal{R}=\mathcal{B}\otimes\mathcal{B}\otimes\mathcal{B}$, where $\mathcal{B}$ is a 1D stencil with 4 points given by:

$$(\mathcal{B}\mathbf{u}^h)(x) = \tfrac{1}{8}u^h(x-\tfrac{3h}{2}) + \tfrac{3}{8}u^h(x-\tfrac{h}{2}) + \\ + \tfrac{3}{8}u^h(x+\tfrac{h}{2}) + \tfrac{1}{8}u^h(x+\tfrac{3h}{2}) = u^{2h}(x)$$

Figure 3 (left) illustrates the 2D analogue of this tensor product stencil, spanning 16 points (compared to 64, in 3D). Note that since our variables are cell centered, the coarse grid variables will not coincide with a subset of the fine variables. The prolongation operator is defined by $\mathcal{P}^T=8\mathcal{B}\otimes\mathcal{B}\otimes\mathcal{B}$, i.e., a scaled transposition of the restriction operator. We can verify that the prolongation is exactly a trilinear interpolation operator from the coarse to the fine grid variables. We limit the output of the prolongation and restriction operators to interior cell variables only; therefore, we only restrict into interior coarse cells, and only prolongate into interior fine variables. Conversely, if a restriction/prolongation stencil uses a non-interior cell variable as input, we substitute a zero value instead.

Our smoother of choice is the damped Jacobi method with parameter $\omega = {}^2\!/_3$ (see Algorithm 2), which is known to be stable for the Poisson equation and facilitates parallelism better than Gauss-Seidel methods. For problems with mixed boundary conditions and irregular domain geometries it is common practice to perform additional smoothing near the boundary of the computational domain. Consequently, we define a *boundary region* consisting of interior cells within a certain distance of Dirichlet or Neumann cells. Figure 3 (right) illustrates the boundary region (shaded black) within a radius of 2 of non-interior cells (measured by L_1-distance). In our approach, we perform a number of Gauss-Seidel iterations (see Algorithm 2, bottom) on the boundary region in addition to the damped Jacobi smoother applied on the entire domain. Our complete smoothing procedure will start with M Gauss-Seidel sweeps on the boundary region, continue with a damped Jacobi sweep over the entire domain, and finish with another N Gauss-Seidel boundary sweeps. Finally, we can approximate the exact solution for the coarsest level in the hierarchy by simply iterating the smoothing procedure a large number of times.

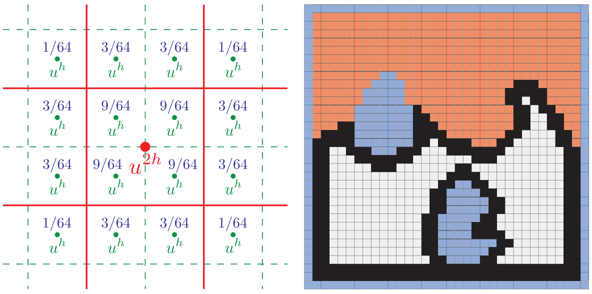


Figure 3: *Left: 2D illustration of the restriction operator. Right: Illustration of the boundary region. A band width of 2 is pictured.*

Algorithm 2 Damped Jacobi $(\omega = 2/3)$ and Gauss-Seidel Smoothers

```
procedure DampedJacobiSmooth($\mathcal{L}, \mathbf{u}, \mathbf{b}, \mathcal{I}$)
    foreach $I=(i,j,k)\in\mathcal{I}$ do ▷ $\mathcal{I}$ is set of cell indices
        $\delta_I \leftarrow (b_I - \mathcal{L}_I\mathbf{u})/\mathcal{L}_{II}$▷ $\mathcal{L}_I$ is the equation in cell $I$
    foreach $I=(i,j,k)\in\mathcal{I}$ do
        $u_I \mathrel{+}= \frac{2}{3}\delta_I$ ▷ $\boldsymbol{\delta}$ is an auxiliary variable
end procedure
procedure GaussSeidelSmooth($\mathcal{L}, \mathbf{u}, \mathbf{b}, \mathcal{I}$)
    foreach $I=(i,j,k)\in\mathcal{I}$ do
        $u_I \mathrel{+}= (b_I - \mathcal{L}_I\mathbf{u})/\mathcal{L}_{II}$
end procedure
```

The multigrid scheme we just described is particularly simplistic, using an elementary coarsening strategy, a very basic smoother and without specialized transfer operators near the boundary. Not surprisingly, attempting to use this V-Cycle as an iterative solver for the Poisson problem reveals the following shortcomings:

Instability: When using multigrid as the solver instead of a preconditioner, the iterates produced by the sequence of V-Cycles can become highly oscillatory or divergent unless a substantial smoothing effort is spent near the boundary. Specifically, we found it necessary to perform at least 30-40 iterations of boundary smoothing (15-20 iterations before the interior sweep, and 15-20 after) on a boundary band at least 3 cells wide, in order to make the V-Cycle iteration stable for highly irregular domains. In contrast, a regular rectangular box domain with all Dirichlet boundary conditions required only 2-3 boundary iterations on a boundary band just a single cell wide to remain perfectly stable with a 0.46 convergence rate. This sensitivity to the regularity of the domain is justified given the geometric discrepancies between different levels for irregular domains, and the imprecision of the transfer operators near the boundary, as discussed in [TOS01]. We will show that this sensitivity to the boundary treatment is removed when the V-Cycle is used as a preconditioner.

Stagnation: Even when the V-Cycle iteration is stable, on certain irregular domains the residual reduction rate quickly degrades towards 1, as opposed to converging to a value less than 0.5 as typically expected of functional multigrid schemes. This complication appears on irregular domains where successive levels of the multigrid hierarchy may exhibit significant topological differences, as coarse modes may not be accurately represented. In these cases, costly eigenanalysis or recombined iterants may prove useful [TOS01]; however, the highly irregular and changing domains common to fluid simulations make these methods impractical.

3.3. Multigrid-preconditioned conjugate gradient

The instability and slow convergence issues described in the previous section can be efficiently ameliorated by using the multigrid V-Cycle as a preconditioner for an appropriate Krylov method. Our experiments indicated that the elementary multigrid scheme in section 3.2 can be an extremely efficient preconditioner, achieving convergence rates comparable or superior to the ideal performance of the V-Cycle on regular domains, even if the smoothing effort vested in the multigrid V-Cycle is significantly less than what is necessary to make it a convergent solver on its own.

We will use the V-Cycle described in Section 3.2 as a preconditioner for the conjugate gradient method. Algorithm 3 provides the pseudocode for our PCG solver. Our algorithm is numerically equivalent to the traditional definition of PCG, as stated for example in [GvL89], but certain variables are being reused for economy of storage, and operations have been rearranged to facilitate optimized execution, as discussed in Section 4. Preconditioning operates by constructing a symmetric, positive definite matrix $\mathcal{M}$ which is easier to invert than $\mathcal{L}$, and such that $\mathcal{M}^{-1}\mathcal{L}$ is significantly better conditioned than $\mathcal{L}$.

In our case, we will use the multigrid V-Cycle as the preconditioner $\mathcal{M}^{-1}$ as described in [Tat93]. In particular, if we define $\mathbf{u} := \mathcal{M}^{-1}\mathbf{b}$, then $\mathbf{u}$ is the result of one iteration of the multigrid V-Cycle for the problem $\mathcal{L}\mathbf{u} = \mathbf{b}$ *with zero initial guess, and zero boundary conditions*. We can easily verify that under these conditions, the action of the V-Cycle indeed corresponds to a linear operator; the requirement that $\mathcal{M}$ be symmetric and positive definite, however, is less trivial. We refer to [Tat93] for a detailed analysis of the conditions for symmetry and definiteness, and only present here a set of sufficient conditions instead. The V-Cycle of Algorithm 1 will produce a symmetric and definite preconditioner if:

- The restriction/prolongation operators are the transpose of one another (up to scaling). This is common practice, and also the case for the V-Cycle we described.

- The smoother used in the *upstroke* of the V-Cycle (Algorithm 1, line 11) performs the operations of the smoother used for the *downstroke* (line 4) in reverse order. E.g., if Gauss-Seidel iteration is used, opposite traversal orders should be used when descending or ascending the V-Cycle, respectively. For Jacobi smoothers, no reversal is necessary as the result is independent of the traversal order.

- The solve at the coarsest level (Algorithm 1, line 8) needs to either be exact, or the inverse of $\mathcal{L}^{(L)}$ should be approximated with a symmetric and definite matrix. If the smoother is iterated to approximate the solution, a number of iterations should be performed with a given traversal order, followed by an equal number of iterations with the order reversed.

Algorithm 3 Multigrid-preconditioned conjugate gradient. *Red-colored steps in the algorithm are applicable when the Poisson problem has a nullspace (i.e., all Neumann boundary conditions), and should be omitted when Dirichlet boundaries are present.*
(†) $\mathbf{u} \leftarrow \mathcal{M}^{-1}\mathbf{b}$ is implemented by calling V-Cycle($\mathbf{u},\mathbf{b}$)

```
procedure MGPCG($\mathbf{r},\mathbf{x}$)
    $\mathbf{r} \leftarrow \mathbf{r} - \mathcal{L}\mathbf{x}$, $\mu \leftarrow \bar{\mathbf{r}}$, $\nu \leftarrow \|\mathbf{r} - \mu\|_\infty$
    if ($\nu < \nu_{\max}$) then return
    $\mathbf{r} \leftarrow \mathbf{r} - \mu$, $\mathbf{p} \leftarrow \mathcal{M}^{-1}\mathbf{r}^{(\dagger)}$, $\rho \leftarrow \mathbf{p}^T\mathbf{r}$
    for $k = 0$ to $k_{max}$ do
        $\mathbf{z} \leftarrow \mathcal{L}\mathbf{p}$, $\sigma \leftarrow \mathbf{p}^T\mathbf{z}$
        $\alpha \leftarrow \rho/\sigma$
        $\mathbf{r} \leftarrow \mathbf{r} - \alpha\mathbf{z}$, $\mu \leftarrow \bar{\mathbf{r}}$, $\nu \leftarrow \|\mathbf{r} - \mu\|_\infty$
        if ($\nu < \nu_{\max}$ or $k = k_{\max}$) then
            $\mathbf{x} \leftarrow \mathbf{x} + \alpha\mathbf{p}$
            return
        end if
        $\mathbf{r} \leftarrow \mathbf{r} - \mu$, $\mathbf{z} \leftarrow \mathcal{M}^{-1}\mathbf{r}^{(\dagger)}$, $\rho^{\text{new}} \leftarrow \mathbf{z}^T\mathbf{r}$
        $\beta \leftarrow \rho^{\text{new}}/\rho$
        $\rho \leftarrow \rho^{\text{new}}$
        $\mathbf{x} \leftarrow \mathbf{x} + \alpha\mathbf{p}$, $\mathbf{p} \leftarrow \mathbf{z} + \beta\mathbf{p}$
    end for
end procedure
```

Finally, Poisson problems with all-Neumann boundary conditions (e.g., simulations of smoke flow past objects) are known to possess a nullspace, as the solution is only known up to a constant. The algorithm of Algorithm 3 is modified in this case to project out this nullspace, and the modifications are highlighted in red in the pseudocode. No modification to the V-Cycle preconditioner is needed.

4. Implementation and Optimizations

Minimizing boundary cost: The cost of performing 30-40 smoothing sweeps on the boundary to stabilize the V-Cycle as a solver cannot be neglected, even though the interior smoothing effort would asymptotically dominate. In practice, this amount of boundary smoothing would be the most costly operation of the V-Cycle for a 256^3 grid or smaller, assuming serial execution. For parallel execution, boundary smoothing would be the clear bottleneck for even higher resolutions (512^3 or more in practice) since the boundary smoother has inferior scalability to the interior smoother due to the irregularity of the boundary band. Since in our approach the V-Cycle will only be used as a CG preconditioner, we do not need to spend that much effort smoothing the boundary. Our experiments indicated that even with no extra smoothing near the boundary the PCG algorithm would be stable and convergent, but only with a moderate acceleration over the unpreconditioned CG method. However, the following carefully designed boundary smoothing led to very rapid convergence.

The extent of our boundary region depends on our prolongation operator as well as our standard 8-to-1 coarsening scheme. Define $D(\mathcal{P},I^{(l)})$ as the set of coarse cells in the prolongation stencil of an interior cell $I^{(l)}$. For each $I^{(l+1)} \in D(\mathcal{P},I^{(l)})$, if any of its 8 fine children are not interior cells, $I^{(l)}$ is included in the boundary band. This heuristic produces a boundary region with a width up to 3 cells (near irregular interfaces) or as low as 1 cell (near walls and boundaries resolved by all grids). Additionally, we only perform boundary smoothing *after* the interior sweep on the downstroke of the V-Cycle and *before* the interior sweep on the upstroke. We found that using 2 Gauss-Seidel iterations on the boundary band at the finest level struck the best balance of computational cost and PCG efficiency. Since the size of the boundary band at every coarser level is reduced by a factor of four, we increase the number of boundary sweeps by a factor of 2 every level (i.e., performing 2^{l+1} sweeps at level l), which incurs only a minimal overhead but further accelerates convergence.

Compact representation of $\mathcal{L}$: The discrete operator $\mathcal{L}$ contains the standard 7-point stencil in the interior of the domain, with a coefficient of $-6/h^2$ on the center cell and $1/h^2$ on the 6 face neighbors. We can eliminate the $1/h^2$ factor by observing that the preconditioning ability of $\mathcal{M}^{-1}$ remains unchanged if the entire matrix is scaled. Thus, if we consider the V-Cycle to be a solver for $h^2\Delta$ instead of the unscaled Poisson operator, the operator coefficients at the interior of the finest level simply become -6 and 1. If we modify the prolongation operator as $\mathcal{P}^T = 2\mathcal{B}\otimes\mathcal{B}\otimes\mathcal{B}$, then the integer-valued scaled-Poisson stencils (with entries -6 and 1) can be used at all levels of the hierarchy. This stencil will only assume a different form near Neumann boundaries: the scaled 7-point stencil will have a coefficient of 1 for any non-Neumann neighbors, a value of zero for Neumann neighbors, and the center coefficient will be the negative count of non-Neumann neighbors. Instead of storing these zero Neumann coefficients, we enforce the condition that any variables where the discrete operator $\mathcal{L}$ will be applied to, shall have zero values on any non-interior cells. As a consequence, we only need to store a negative integer (the center coefficient) to represent the Poisson operator near the boundary.

Using zero initial guess and boundary conditions: When the multigrid V-Cycle is used as a preconditioner, it is solving a Poisson problem with zero boundary conditions, and uses a zero initial guess as discussed in Section 3.3. Also, every subsequent level of the multigrid correction scheme will also employ zero initial guess and boundary conditions. This fact allows us to simplify and combine steps 2,4, and 5 of the algorithm in Algorithm 1, which are the most costly components of the V-Cycle algorithm. For simplicity, consider an all-Dirichlet problem, where $\mathcal{L}$ has the same 7-point stencil (with coefficients -6 and 1 as discussed before) everywhere in the domain. Using the fact that $\mathbf{u}^{(0)} = \mathbf{0}$, line 3 of the damped Jacobi smoother in Algorithm 2 produces $\boldsymbol{\delta} = -\frac{1}{6}\mathbf{b}$ and the result of the relaxation will be $\mathbf{u} = -\frac{1}{9}\mathbf{b}$. The residual after the relaxation will be $\mathbf{r} = \mathbf{b} - \mathcal{L}(-\frac{1}{9}\mathbf{b}) = \frac{1}{9}(\mathcal{L}+9\mathbf{I})\mathbf{b}$. Thus, both the smoothed solution $\mathbf{u}$ as well as the residual $\mathbf{r}$ resulting from this smoother can be computed simultaneously from $\mathbf{b}$, in a single streaming pass over these 3 variables (δ is just temporary and need not be computed). Of course, this procedure will be invalid near Neumann boundaries, where $\mathcal{L}$ has a modified stencil. For this reason, we mask the output on $\mathbf{u}$ to exclude cells with non-standard stencils, setting those values to zero instead. These cells will be smoothed in the boundary sweep immediately afterwards. Lastly, the Gauss-Seidel boundary sweep will invalidate the residuals computed near the boundary band. We correct this by updating the residuals in the 1-ring of the boundary band at the end of the smoothing procedure.

Pipelining operations: Streaming operations in the PCG algorithm were often combined in a single pass through memory. The pseudocode in Algorithm 3 has been written to indicate such combinable operations on the same line, e.g., lines 2, 4, 6, 8, 13, and 16.

Multithreading: Most of the operations in our solver are trivially parallelizable, as they are order independent (excluding the boundary smoother). In order to parallelize the boundary Gauss-Seidel smoother, we tiled the domain with blocks of size 4 × 4 × 4 which were subsequently red-black colored. Subsequently, we parallelize the Gauss-Seidel sweep over the red blocks, distributing a balanced number of blocks to each processor, without fine-grain locking. Black blocks are processed in a separate parallel sweep. Iterating the Gauss-Seidel smoother as many as 10 times within each block costs less than twice the cost of a single iteration (due to cache effects) which further improved the efficiency of our preconditioner.

Variable reuse and storage footprint: Our methodology reuses vector variables to minimize the memory footprint. We only require (and reuse) the following 5 vector variables, defined over the entire domain: $\mathbf{x}$ (for initial guess and solution), $\mathbf{r}$ (for initial right hand side, subsequent residuals and also the right-hand side $\mathbf{b}^{(0)}$ at the top level of the V-Cycle),

$\mathbf{p}, \mathbf{z}$ (reused throughout CG, also used as the solution $\mathbf{u}^{(0)}$ at the finest V-Cycle level) and $\boldsymbol{\delta}$ (reused to hold the residual $\mathbf{r}$ within the V-Cycle). Vector variables for the coarser levels are smaller by a factor of 8. We also represent the extent of the interior/boundary regions using bitmaps for a minor storage overhead (e.g., for the Jacobi smoother or transfer operators which are restricted to interior cells). Quantities such as the diagonal element of $\mathcal{L}$ (and its inverse) are only stored for the boundary region. Ultimately, the entire PCG solver for a $768^2 \times 1152$ grid has a memory footprint slightly under 16GB (out of which, 12.8GB used for $\mathbf{x}, \mathbf{r}, \mathbf{p}, \mathbf{z}$ and $\boldsymbol{\delta}$) and an entire smoke solver at this resolution fits easily in 32GB of memory. We note that our method maintains minimal metadata, and the memory for all data variables can be reused outside of the solver. For example, we have reused $\mathbf{p}, \mathbf{z}$ and $\boldsymbol{\delta}$ in the advection step to store the pre-projection velocities.

5. Examples

We demonstrate the efficacy of our solver with high resolution smoke and free surface simulations. We use simple semi-Lagrangian advection [Sta99] for the density and level sets. Velocity extrapolation for air velocities was done as in [ZB05] and fast sweeping [Zha04] was used for level set reinitialization. Semi-Lagrangian advection is plagued by substantial numerical viscosity at lower resolutions; however, the efficiency and lightweight nature of our voxelized Poisson solver allows for sufficiently high resolution to produce detailed incompressible flows with these comparably simple methods. Figure 6 demonstrates the effect of the available resolution. We found a tolerance of $\|r\|_\infty < 10^{-3}$ for smoke simulations and and 10^{-5} for free surface produced visually pleasing results. Table 2 gives runtimes for the voxelized Poisson solves in our simulations. Although we used resolutions substantially higher than typical, the Poisson solves represent between 25% and 50% of total simulation time. These examples demonstrate the solver's ability to handle rapidly changing domain geometries and to converge with just a few PCG iterations in practical settings.

Figure 4 compares the convergence of our method with CG and incomplete Cholesky PCG in terms of rate per iteration. As incomplete Cholesky PCG involves sparse backsubstitution and cannot be parallelized without significant effort (if at all), we compare serial runtimes for an incomplete Cholesky PCG implementation, a "black-box" conjugate gradient solver, our streamlined conjugate gradient solver, and our multigrid preconditioned solver to a given tolerance on the smoke past sphere domain in Table 1. Even without the aid of parallelization, we see a 4.5× speed-up at 128^3 over ICPCG, a 10× speed-up at 256^3, and a 20× speed-up at 512^3 resolution. Our ICPCG and "black-box" CG solvers are generic implementations with explicit compressed sparse row matrix representations.

"Black Box"(†) and Pipelined(*) CG (serial)

Resolution	64^3	96^3	128^3	192^3	256^3	384^3	512^3
Initialization Time (s)	0.0585	0.0617	0.1403	0.3931	1.0628	2.9837	6.9207
Iterations to r=1e-4	110	165	221	332	445	667	965
Iterations to r=1e-8	181	309	367	623	822	1261	1538
Time to r=1e-4 (s)†	2.32	8.73	42.45	149.94	503.97	2558.44	9010.28
Time to r=1e-4 (s)*	1.39	7.80	21.24	113.86	340.24	1999.10	5540.92
Time to r=1e-8 (s)†	3.81	16.35	70.49	281.37	930.92	4836.87	14360.43
Time to r=1e-8 (s)*	2.09	12.70	35.10	212.68	663.44	3524.00	8831.03

Incomplete Cholesky PCG (serial)

Resolution	64^3	96^3	128^3	192^3	256^3	384^3	512^3
Initialization Time (s)	0.20	0.55	1.24	4.01	9.61	32.23	76.47
Iterations to r=1e-4	36	52	72	107	138	213	278
Iterations to r=1e-8	57	78	104	149	194	295	395
Time to r=1e-4 (s)	0.94	5.04	17.00	88.22	283.10	1526.29	4679.32
Time to r=1e-8 (s)	1.50	7.56	24.55	122.85	397.98	2113.88	6648.68

Multigrid PCG (serial)

Resolution	64^3	96^3	128^3	192^3	256^3	384^3	512^3
Initialization Time (s)	0.08	0.10	0.24	0.63	1.63	5.01	12.21
Iterations to r=1e-4	9	9	11	10	12	12	13
Iterations to r=1e-8	15	16	17	18	19	20	21
Time to r=1e-4 (s)	0.57	2.82	3.70	10.74	25.92	89.71	211.88
Time to r=1e-8 (s)	0.88	4.80	5.57	18.50	39.54	144.47	332.71

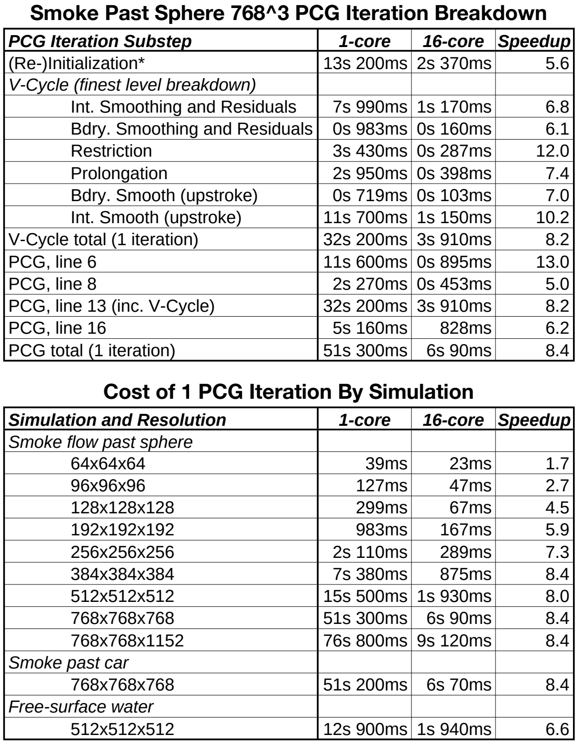
Table 1: *Number of iterations and* ***serial*** *runtimes for our modified conjugate gradient(*) and a "black box"(†) solver (top), incomplete Cholesky PCG (middle), and our method (bottom). Initialization time includes building any matrices and computing incomplete Cholesky factorization for ICPCG or building multigrid hierarchy for MGPCG.*

6. Limitations and future work

Our performance at high resolutions is largely due to parallelization. While the convergence rate remains excellent, parallel scalability is inferior at lower resolutions. Our method was tuned for shared-memory multiprocessors since we targeted problem sizes that would not fit on GPU memory. However, our method would benefit from the bandwidth of GPU platforms and we will investigate GPU ports of our method as future work. Methods that approximate the domain with sub-cell accuracy could be expected to produce improved results for the same base resolution. The results of [SW05] suggest methods such as [BBB07] would likely be accelerated with a small adjustment to our method. The resolution enabled by our method could be combined with techniques such as wavelet turbulence [KTJG08] or vorticity confinement [FSJ01] to add even more detail. In the future, an adaptive multigrid formulation such as semicoarsening [TOS01] could also be used to precondition an adaptive method such as [LGF04], although obtaining the same level of scalability would be challenging in this context.

Acknowledgments

We wish to thank P. Dubey and the Intel Throughput Computing Lab for valuable feedback and support. Special thanks to Andrew Selle for his help with our audio-visual materials. A.M., E.S. and J.T. were supported in part by DOE 09-LR-04-116741-BERA, NSF DMS-0652427, NSF CCF-0830554, ONR N000140310071. We wish to acknowledge the Stanford Graphics Laboratory and XYZrgb Inc. for the dragon model.



Smoke Past Sphere 768^3 PCG Iteration Breakdown

PCG Iteration Substep	*1-core*	*16-core*	*Speedup*
(Re-)Initialization*	13s 200ms	2s 370ms	5.6
V-Cycle (finest level breakdown)			
Int. Smoothing and Residuals	7s 990ms	1s 170ms	6.8
Bdry. Smoothing and Residuals	0s 983ms	0s 160ms	6.1
Restriction	3s 430ms	0s 287ms	12.0
Prolongation	2s 950ms	0s 398ms	7.4
Bdry. Smooth (upstroke)	0s 719ms	0s 103ms	7.0
Int. Smooth (upstroke)	11s 700ms	1s 150ms	10.2
V-Cycle total (1 iteration)	32s 200ms	3s 910ms	8.2
PCG, line 6	11s 600ms	0s 895ms	13.0
PCG, line 8	2s 270ms	0s 453ms	5.0
PCG, line 13 (inc. V-Cycle)	32s 200ms	3s 910ms	8.2
PCG, line 16	5s 160ms	828ms	6.2
PCG total (1 iteration)	51s 300ms	6s 90ms	8.4

Cost of 1 PCG Iteration By Simulation

Simulation and Resolution	*1-core*	*16-core*	*Speedup*
Smoke flow past sphere			
64x64x64	39ms	23ms	1.7
96x96x96	127ms	47ms	2.7
128x128x128	299ms	67ms	4.5
192x192x192	983ms	167ms	5.9
256x256x256	2s 110ms	289ms	7.3
384x384x384	7s 380ms	875ms	8.4
512x512x512	15s 500ms	1s 930ms	8.0
768x768x768	51s 300ms	6s 90ms	8.4
768x768x1152	76s 800ms	9s 120ms	8.4
Smoke past car			
768x768x768	51s 200ms	6s 70ms	8.4
Free-surface water			
512x512x512	12s 900ms	1s 940ms	6.6

Table 2: *Execution cost and parallel speedup for our method. All parallel computations were carried out on a 16-core Intel Xeon X7350 server with 32GB of RAM. (*)Initialization refers to multigrid hierarchy initialization.*

References

[AF96] Ashby S., Falgout R.: A parallel multigrid preconditioned conjugate gradient algorithm for groundwater flow simulations. *Nuclear Science and Engineering 124*, 1 (1996), 145–159.

[BBB07] Batty C., Bertails F., Bridson R.: A fast variational framework for accurate solid-fluid coupling. *ACM Trans. Graph. 26*, 3 (2007), 100.

[BFGS03] Bolz J., Farmer I., Grinspun E., Schröoder P.: Sparse matrix solvers on the GPU: conjugate gradients and multigrid. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers* (New York, NY, USA, 2003), ACM, pp. 917–924.

[BWKS06] Bruhn A., Weickert J., Kohlberger T., Schnörr C.: A multigrid platform for real-time motion computation with discontinuity-preserving variational methods. *Int. J. Comput. Vision 70*, 3 (2006), 257–277.

[BWRB05] Baranoski G. V. G., Wan J., Rokne J. G., Bell I.: Simulating the dynamics of auroral phenomena. *ACM Trans. Graph. 24*, 1 (2005), 37–59.

[CFL*07] Chentanez N., Feldman B. E., Labelle F., O'Brien J. F., Shewchuk J. R.: Liquid simulation on lattice-based tetrahedral meshes. In *SCA '07: Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation* (Aire-la-Ville, Switzerland, Switzerland, 2007), Eurographics Association, pp. 219–228.

[CGR*04] Clarenz U., Griebel M., Rumpf M., Schweitzer M., Telea A.: Feature sensitive multiscale editing on surfaces. *The Visual Computer 20*, 5 (2004), 329–343.

[CLB*09] Chuang M., Luo L., Brown B. J., Rusinkiewicz S., Kazhdan M.: Estimating the Laplace-Beltrami operator by restricting 3D functions. In *Eurographics Symposium on Geometry Processing* (2009), Eurographics Assocation.

[CMT04] Carlson M., Mucha P., Turk G.: Rigid fluid: animating the interplay between rigid bodies and fluid. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers* (New York, NY, USA, 2004), ACM, pp. 377–384.

[CTG10] Cohen J. M., Tariq S., Green S.: Interactive fluid-particle simulation using translating eulerian grids. In *I3D '10: Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2010), ACM, pp. 15–22.

[ETK*07] Elcott S., Tong Y., Kanso E., Schröder P., Desbrun M.: Stable, circulation-preserving, simplicial fluids. *ACM Trans. Graph. 26*, 1 (2007), 4.

[FF01] Foster N., Fedkiw R.: Practical animation of liquids. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 2001), ACM, pp. 23–30.

[FM96] Foster N., Metaxas D.: Realistic animation of liquids. *Graph. Models Image Process. 58*, 5 (1996), 471–483.

[FOK05] Feldman B., O'Brien J., Klingner B.: Animating gases with hybrid meshes. *ACM Trans. Graph. 24*, 3 (2005), 904–909.

[FSJ01] Fedkiw R., Stam J., Jensen H.: Visual simulation of smoke. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 2001), ACM, pp. 15–22.

[GBO04] Goktekin T., Bargteil A., O'Brien J.: A method for animating viscoelastic fluids. *ACM Trans. Graph. 23*, 3 (2004), 463–468.

[GSMY*08] Göddeke D., Strzodka R., Mohd-Yusof J., McCormick P., Wobker H., Becker C., Turek S.: Using GPUs to improve multigrid solver performance on a cluster. *International Journal of Computational Science and Engineering 4*, 1 (2008), 36–55. doi: 10.1504/IJCSE.2008.021111.

[GTS02] Green S., Turkiyyah G., Storti D.: Subdivision-based multilevel methods for large scale engineering simulation of thin shells. In *Proceedings of the seventh ACM symposium on Solid modeling and applications* (2002), ACM New York, NY, USA, pp. 265–272.

[GvL89] Golub G., van Loan C.: *Matrix Computations*. The John Hopkins University Press, 1989.

[GWL*03] Goodnight N., Woolley C., Lewin G., Luebke D., Humphreys G.: A multigrid solver for boundary value problems using programmable graphics hardware. In *Proceedings of the ACM SIGGRAPH/Eurographics Conf. on Graphics Hardware* (2003), pp. 102–111.

[HG09] Horvath C., Geiger W.: Directable, high-resolution simulation of fire on the GPU. *ACM Trans. Graph. 28*, 3 (2009), 1–8.

[HGS*07] Hughes C., Grzeszczuk R., Sifakis E., Kim D., Kumar S., Selle A., Chhugani J., Holliman M., Chen Y.-K.: Physical simulation for animation and visual effects: Parallelization and characterization for chip multiprocessors. In *Intl. Symp. on Comput. Architecture* (2007).

[HK05] Hong J., Kim C.: Discontinuous fluids. *ACM Trans. Graph. 24*, 3 (2005), 915–920.

[HLYK08] Hong J., Lee H., Yoon J., Kim C.: Bubbles alive. *ACM Trans. Graph. 27*, 3 (2008), 1–4.

[HMBVR05] Haber T., Mertens T., Bekaert P., Van Reeth F.: A computational approach to simulate subsurface light diffusion in arbitrarily shaped objects. In *GI '05: Proceedings of Graphics Interface 2005* (School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 2005), Canadian Human-Computer Communications Society, pp. 79–86.

[HW65] Harlow F., Welch J.: Numerical calculation of time-dependent viscous incompressible flow of fluid with a free surface. *Phys. Fl. 8* (1965), 2812–2189.

[KC07] Kim T., Carlson M.: A simple boiling module. In *SCA '07: Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation* (Aire-la-Ville, Switzerland, Switzerland, 2007), Eurographics Association, pp. 27–34.

[KFCO06] Klingner B., Feldman B., Chentanez N., O'Brien J.: Fluid animation with dynamic meshes. *ACM Trans. Graph. 25*, 3 (2006), 820–825.

[KH08] Kazhdan M., Hoppe H.: Streaming multigrid for gradient-domain operations on large images. *ACM Trans. Graph. 27*, 3 (2008), 1–10.

[KTJG08] Kim T., Thürey N., James D., Gross M.: Wavelet turbulence for fluid simulation. *ACM Trans. Graph. 27*, 3 (2008), 1–6.

[LGF04] Losasso F., Gibou F., Fedkiw R.: Simulating water and smoke with an octree data structure. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers* (New York, NY, USA, 2004), ACM, pp. 457–462.

[MCP*09] Mullen P., Crane K., Pavlov D., Tong Y., Desbrun M.: Energy-preserving integrators for fluid animation. *ACM Trans. Graph. 28*, 3 (2009), 1–8.

[MCPN08] Molemaker J., Cohen J., Patel S., Noh J.: Low viscosity flow simulations for animation. In *Proceedings of ACM SIGGRAPH Symposium on Computer Animation* (2008), Gross M., James D., (Eds.), Eurographics / ACM SIGGRAPH, pp. 15–22.

[MSW*09] McAdams A., Selle A., Ward K., Sifakis E., Teran J.: Detail preserving continuum simulation of straight hair. *ACM Trans. Graph. 28*, 3 (2009), 1–6.

[MTPS04] McNamara A., Treuille A., Popović Z., Stam J.: Fluid control using the adjoint method. *ACM Trans. Graph. 23*, 3 (2004), 449–456.

[NGH04] Ni X., Garland M., Hart J.: Fair Morse functions for extracting the topological structure of a surface mesh. *ACM Transactions on Graphics (TOG) 23*, 3 (2004), 613–622.

[NNSM07] Nielsen M., Nilsson O., Söderström A., Museth K.: Out-of-core and compressed level set methods. *ACM Trans. Graph. 26*, 4 (2007), 16.

[OGRG07] Otaduy M. A., Germann D., Redon S., Gross M.: Adaptive deformations with fast tight bounds. In *SCA '07: Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation* (Aire-la-Ville, Switzerland, Switzerland, 2007), Eurographics Association, pp. 181–190.

[ONW08] Oh S., Noh J., Wohn K.: A physically faithful multigrid method for fast cloth simulation. *Computer Animation and Virtual Worlds 19*, 3-4 (2008), 479–492.

[PM04] Papandreou G., Maragos P.: A fast multigrid implicit algorithm for the evolution of geodesic active contours. *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on 2* (2004), 689–694.

[RC03] Richard F. J. P., Cohen L. D.: A new image registration technique with free boundary constraints: application to mammography. *Comput. Vis. Image Underst. 89*, 2-3 (2003), 166–196.

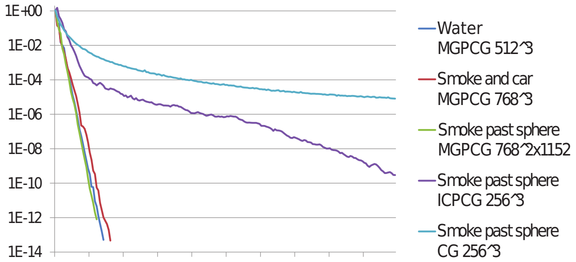


Figure 4: *Comparison of multigrid-preconditioned CG (MGPCG), incomplete Cholesky PCG (ICPCG) and unpreconditioned CG (CG). The horizontal axis corresponds to iterations, the vertical indicates the residual reduction factor* $|r_k|/|r_0|$ *after k iterations.*

[SAB*99] Sussman M., Almgren A. S., Bell J. B., Colella P., Howell L. H., Welcome M. L.: An adaptive level set approach for incompressible two-phase flows. *J. Comput. Phys. 148*, 1 (1999), 81–124.

[SRF05] Selle A., Rasmussen N., Fedkiw R.: A vortex particle method for smoke, water and explosions. *ACM Trans. Graph. 24*, 3 (2005), 910–914.

[Sta95] Stam J.: Multiple scattering as a diffusion process. In *In Eurographics Rendering Workshop* (1995), pp. 41–50.

[Sta99] Stam J.: Stable fluids. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques* (1999), pp. 121–128.

[SW05] Singh K., Williams J.: A parallel fictitious domain multigrid preconditioner for the solution of Poisson's equation in complex geometries. *Comput. Meth. Appl. Mech. Eng. 194* (2005), 4845–4860.

[SYBF06] Shi L., Yu Y., Bell N., Feng W.: A fast multigrid algorithm for mesh deformation. *ACM Trans. Graph. 25*, 3 (2006), 1108–1117.

[Tat93] Tatebe O.: The multigrid preconditioned conjugate gradient method. In *Proceedings of the Sixth Copper Mountain Conference on Multigrid Methods* (1993), NASA Conference Publication 3224, pp. 621–634.

[TO94] Tatebe O., Oyanagi Y.: Efficient implementation of the multigrid preconditioned conjugate gradient method on distributed memory machines. In *Supercomputing '94: Proceedings of the 1994 conference on Supercomputing* (Los Alamitos, CA, USA, 1994), IEEE Computer Society Press, pp. 194–203.

[TOS01] Trottenberg U., Oosterlee C., Schuller A.: *Multigrid*. San Diego: Academic Press, 2001.

[WMT05] Wang H., Mucha P., Turk G.: Water drops on surfaces. *ACM Trans. Graph. 24*, 3 (2005), 921–929.

[ZB05] Zhu Y., Bridson R.: Animating sand as a fluid. *ACM Trans. Graph. 24*, 3 (2005), 965–972.

[Zha04] Zhao H.: A fast sweeping method for the eikonal equation. *Math. Comp. 74*, 250 (2004), 603–627.