A Novel Algorithm for Incompressible Flow Using Only a Coarse Grid Projection

Michael Lentine*
Stanford University
Industrial Light + Magic

Wen Zheng
Stanford University

Ronald Fedkiw
Stanford University
Industrial Light + Magic

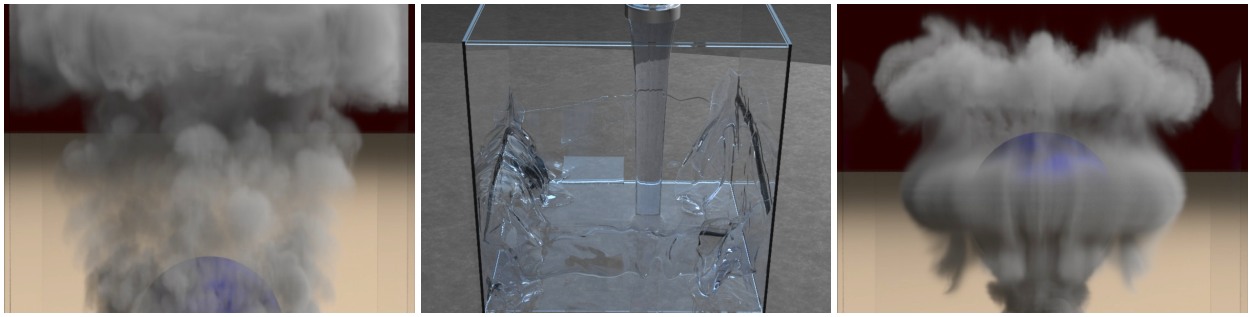

Figure 1: *High resolution smoke and water. (Left) Smoke flowing above a sphere on a* $512 \times 1024 \times 512$ *grid. (Middle) Water pouring into a box on a* $512 \times 512 \times 512$ *grid. (Right) Smoke flowing around a sphere on a* $512 \times 1024 \times 512$ *grid.*

Abstract

Large scale fluid simulation can be difficult using existing techniques due to the high computational cost of using large grids. We present a novel technique for simulating detailed fluids quickly. Our technique coarsens the Eulerian fluid grid during the pressure solve, allowing for a fast implicit update but still maintaining the resolution obtained with a large grid. This allows our simulations to run at a fraction of the cost of existing techniques while still providing the fine scale structure and details obtained with a full projection. Our algorithm scales well to very large grids and large numbers of processors, allowing for high fidelity simulations that would otherwise be intractable.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Physically Based Modeling;

Keywords: simulation, incompressible flow, smoke, water

1 Introduction

Physical simulation of fluids is one of the most interesting and challenging problems because of the amount of small scale details that realistic fluids exhibit. Although many authors such as [Foster and Metaxas 1997; Stam 1999; Fedkiw et al. 2001] have used grid-based techniques to produce visually compelling results, the size of the grids that these techniques can use is limited by the amount of computational power available.

As a consequence many authors have developed techniques that add details to these simulations with noise. For example, Kolmolgorov noise (see [Stam and Fiume 1993; Lamorlette and Foster 2002; Rasmussen et al. 2003]) and curl noise (see [Bridson et al. 2007]) can be used to enhance the visual fidelity of fluid simulations by coupling the noise to the incompressible Navier-Stokes equations producing a more detailed flow. Alternatively, [Kim et al. 2008] and [Narain et al. 2008] determine where to add noise using information from the existing simulation and then add it as a postprocess which allows them to add noise where it is best suited. Other techniques such as [Schechter and Bridson 2008] both determine where to add noise and couple the noise to the Navier-Stokes equations. All of these techniques are successful at adding details but are nonphysical and can produce significantly less realistic results than simply simulating with a higher resolution grid.

Another approach is to improve the baseline simulation on the existing grid. This can be done by using higher order methods in space, such as BFECC, QUICK, and MacCormack methods (see [Dupont and Liu 2003; Kim et al. 2005; Selle et al. 2008; Molemaker et al. 2008]), or in time, such as Runge Kutta. One could also work to maintain certain invariants such as energy (see [Mullen et al. 2009]). Although these methods increase the accuracy and fidelity of the resulting simulation, they are more expensive than traditional fluid simulation and are still limited by the Nyquist frequency of the grid. To increase the grid resolution while keeping the increase in cost to a minimum, adaptive grid techniques were introduced such as AMR [Berger and Oliger 1984] and octrees [Losasso et al. 2004]. These techniques are effective at reducing the computational cost in cells where there is not much detailed motion while maintaining details where needed. However, the increased complexity of using these complicated structures both increases computational cost and hinders the ability to design robust numerical methods.

In contrast to grid-based approaches, particle-based methods (see e.g. [Reeves 1983; Desbrun and Cani 1996; Müller et al. 2003]) are not limited by the resolution of the grid. However, these approaches do not store the connectivity of the surface and require additional computational costs to keep track of the surface and to re-mesh. Furthermore, these methods increase in cost as they approach the incompressible limit, and thus many authors use weakly compressible equation-of-state formulations. There has also been work on combining particle and grid-based approaches. [Gao et al. 2009] uses a combination of grid and particle-based approaches to produce realistic simulations at variable speeds. [Selle et al. 2005] introduced a particle-based method to add vorticity to grid-based simulations using particles that are able to accurately track the vor-

*e-mail: mlentine,zhw,fedkiw@cs.stanford.edu

ticity of the simulation and [Pfaff et al. 2009] extended it to work well with objects. Although these techniques do add details at little cost, they are still limited by the base resolution of the simulation.

In order to handle very high resolution grids, [Wicke et al. 2009] introduces a reduced order model that can handle large grids at a small cost. However, the use of basis functions lacks the physical realism and details that are achieved through traditional grid-based techniques. [Rasmussen et al. 2003; Horvath and Geiger 2009] introduce methods that can run large scale two-dimensional algorithms and then can extend the results to three dimensions. Although these do produce visually compelling results in some instances, the two-dimensional simulations are not a very good approximation of the three-dimensional behavior for the more general case.

Recently, there has been a large interest in methods for creating a higher resolution result from a lower resolution simulation. [Nielsen et al. 2009] introduced a method for increasing the resolution of a simulation to make it more directable, but this also increases the cost. In liquids, authors have worked to track the surface of the liquid on a higher resolution grid than the underlying fluid simulation (see e.g. [Kim et al. 2009]). [Yoon et al. 2009] uses the vortex particle method in order to increase the resolution of a simulation without the cost of running a simulation on a higher resolution grid. They demonstrated that the vortex particle method of [Selle et al. 2005] inherently contains additional computational detail which is lost when mapping it onto the underlying grid, and preserve more of it by mapping it on to the higher resolution grid.

In contrast to these methods which rely on a low resolution core simulation, we propose a method that runs on a high resolution grid but creates a divergence-free velocity field using a coarser grid to speed up the calculations. This allows us to run simulations using higher resolutions while significantly reducing the computational cost, and possibly more importantly, run simulations that scale more linearly on large computational frameworks.

2 Performance Analysis

Simulation performance remains a large obstruction to using very large grids. To alleviate this issue we must develop techniques that are not only more efficient but can take advantage of the computational power available which can be on one or across many machines. Our method improves both the performance on one core and the scalability across many cores.

2.1 Scaling

Traditional grid-based algorithms have a cost

$$C = C_l + C_p, \tag{1}$$

where C_p is the cost of the projection and C_l is the cost of the remainder of the algorithm. When we increase the resolution of the grid in three dimensions by a factor of k this becomes roughly

$$C_k \approx k^4 C_l + k^4 C_p. \tag{2}$$

This is because as we double the size of the grid we have eight times as many cells and each cell is half as wide, requiring twice as many time steps (when assuming a CFL condition). Note that $k^4 C_p$ is only an approximation of the real cost since the matrix inversion does not generally scale linearly with the size of the matrix. One could use multi-grid methods, limit the number of conjugate gradient iterations, etc. to make this scale more linearly, but C_p will still scale no better than k^4.

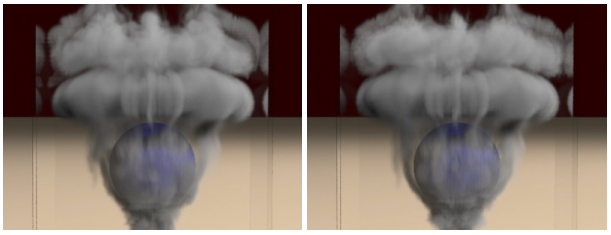

Figure 2: *Smoke flowing around a moving sphere with on a* $512 \times 1024 \times 512$ *grid. (Left) uses a CFL number of 0.9 and takes time steps half as large as those generally taken on a* $256 \times 512 \times 256$ *grid. (Right) uses a CFL number of 2.2 making the time steps around 2.5 times as large.*

Our goal is to reduce the cost of this system without losing the fine scale details of a higher resolution simulation. To achieve this, we break up the projection into a single projection on a coarser grid, which takes time C_{cp}, and a number of smaller projections that run fully decoupled on individual coarse grid cells, costing C_{fp}. This makes the cost for our algorithm

$$C = C_l + C_{cp} + C_{fp}. \tag{3}$$

If we refine our simulation grid, but not our coarse grid, C_{cp} scales only with the number of time steps, and the entire algorithm scales roughly as

$$C_k \approx k^4 C_l + k C_{cp} + k^4 C_{fp}. \tag{4}$$

Even if we assume that $C_{cp} + C_{fp} > C_p$, for some k, $kC_{cp} + k^4 C_{fp} < k^4 C_p$. However, we found that $C_{cp} + C_{fp} > C_p$ only for very small resolutions (less than $16 \times 16 \times 16$) meaning that for reasonably sized examples or large examples our method runs significantly faster than traditional grid-based methods. We note that in our experience, about 90% of the time spent in a one-phase smoke simulation is spent during the projection meaning that C_p easily dominates C_l. From a complexity point of view this makes sense since the advection in C_l requires touching every grid point and its neighbors approximately once whereas a conjugate gradient algorithm would require touching every grid point and its neighbors once for every iteration of the conjugate gradient solve.

In spite of the analysis above, in general it seems that one of the k's can be removed from (4) instead getting $C_k \approx k^3 C_l + C_{cp} + k^3 C_{fp}$ by keeping the same timestep on the finer grid that was used on the coarser grid. Semi-Lagrangian advection (see [Stam 1999]) makes this possible. Although we do not expect this to work for an indefinite number of refinements, for one or two, we have achieved good results without halving the time step (see for example Figure 2).

2.2 Multi-core and Multi-processor Machines

As chip technology continues to advance, the use of multi-core and multi-processor machines and GPUs becomes more and more important, and algorithms such as [Bolz et al. 2003] that can take advantage of this with little cost are able to run much more quickly. Simulating one-phase smoke typically consists of two primary steps, the explicit advection step and the implicit projection step. Looking at (1), C_l is the advection step and C_p is the projection step. The advection step easily scales to multiple cores and multiple processors as one can simply break up the grid into several distinct parts and run advection on each part individually. The only issue then becomes how to deal with the boundaries. If the memory is shared, one can devise various strategies such as keeping separate copies of the information to alleviate issues with writing to data that needs to be read. There are also other strategies such as red/black domain decomposition schemes, locking, etc. Using

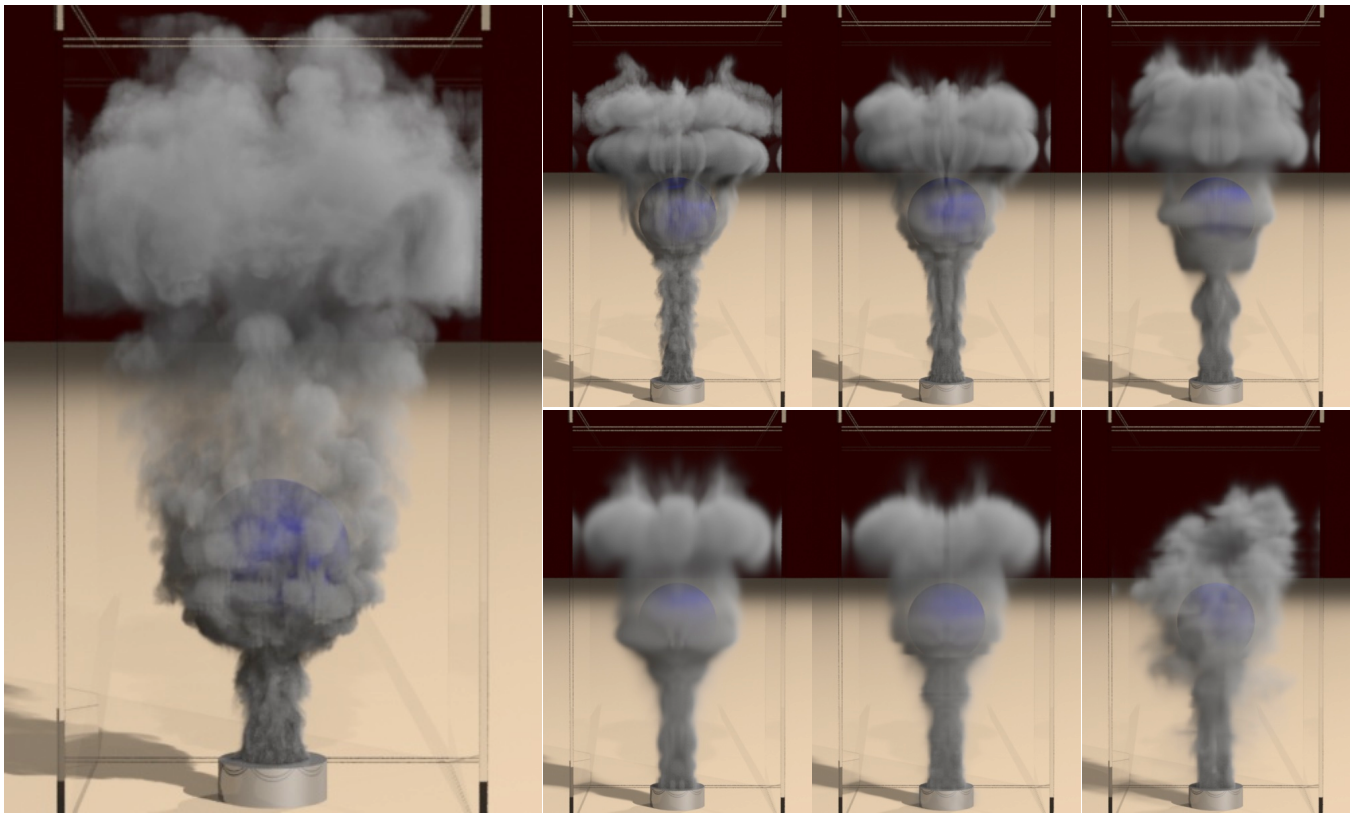

Figure 3: *An example with smoke flowing around a moving sphere. The Large figure is a* $512 \times 1024 \times 512$ *simulation with a* $64 \times 128 \times 64$ *coarse grid. The smaller figures are comparisons of a simulation using different grid resolutions. The resolutions are starting from the left to right, top to bottom: a* $512 \times 1024 \times 512$ *simulation with a* $64 \times 128 \times 64$ *coarse grid, a* $256 \times 512 \times 256$ *simulation with a* $64 \times 128 \times 64$ *coarse grid, a* $128 \times 256 \times 128$ *simulation with a* $64 \times 128 \times 64$ *coarse grid, a* $64 \times 128 \times 64$ *base simulation, a* $64 \times 128 \times 64$ *simulation with a* $32 \times 64 \times 32$ *coarse grid, and a* $64 \times 128 \times 64$ *simulation with a* $16 \times 32 \times 16$ *coarse grid with Kolmolgorov noise.*

these strategies, one can usually expect to diminish the computational overhead of the overlapping boundaries. However, for large problems, where many computational nodes are desired, one would need a huge many core shared memory machine with a very large amount of memory – which is quite expensive and can lead to issues with cooling and energy. If algorithms that run on large computers become less tightly coupled, they require less synchronization and thus less shared memory, allowing cluster cores to be physically further apart (even on separate machines).

Thus, a more typical scenario is to switch to a non-shared memory model where memory might be shared by clusters of cores but is not shared between different clusters. Instead, each cluster has its own RAM. Although much more practical, this large non-shared memory infrastructure leads to a significant computational overhead as it requires data communication. That same overhead is also present when trying to utilize graphics cards. For the advection step C_l, boundary data only needs to be copied between clusters of cores once for each time step. Note that to do this a CFL condition is assumed to bound the amount of cells that need to be copied. Projection, on the other hand, requires at least one copy for each iteration of the solver. For a typical conjugate gradient solver this means that we have to copy or exchange data on the boundaries before every iteration. If we were to use a multi-grid method instead, we would still need many communications for each step in a V-cycle and several V-cycles are typically required for convergence.

Practical experience has shown that the computational bottleneck of exchanging boundary information leads to the projection scaling relativity poorly and dominating the cost of the simulation even more so than on a single core. Our algorithm on the other hand uses a far less expensive global solve, C_{cp}, on a coarse grid, which increases its computational demands much slower than usual as the simulation grid is refined (its coefficient is k, not k^4, in (4)). An extremely important aspect of this algorithm is that the computational cost of increasing the resolution of the grid is mostly contained in C_l and C_{fp} as can be seen in (4), and *both* C_l and C_{fp} scale extremely well with the number of processors (almost perfectly linearly). The reason for this is that they are comprised of a large number of independent tasks. During the computations of C_{fp}, no communication of boundary information is required once the boundary conditions are initially set. The result is that on a large number of processors our algorithm is dominated by C_{cp}, which is significantly less expensive than C_p.

3 Making a Divergence-Free Flow

In order to use a coarser projection step we must determine how to make a coarser grid from our simulation grid, solve on coarser grid, and then map the results to our simulation grid in a way that maintains the divergence-free property. Our algorithm for one-phase smoke proceeds as in [Fedkiw et al. 2001], except instead of solving the Poisson equation on the fine grid we do the following:

1. Map the velocity field to a coarse uniform grid
2. Make the resulting field divergence free on that coarse grid
3. Map these velocities back to our fine simulation grid
4. Make the resulting field divergence free on the fine grid

3.1 Navier-Stokes Equations

On the fine simulation grid, we solve the inviscid, incompressible Navier-Stokes equations for the conservation of mass and momentum, given by

$$\vec{u}_t + \vec{u} \cdot \nabla \vec{u} = -\frac{1}{\rho}\nabla p + \vec{f} \tag{5}$$

$$\nabla \cdot \vec{u} = 0, \tag{6}$$

where $\vec{u}$ is the velocity field of the fluid, ρ is the density of the fluid, $\vec{f}$ are any external forces (such as gravity), and p is the fluid pressure. We solve these equations by first calculating an intermediate velocity field $\vec{u}^\star$ using

$$\frac{\vec{u}^\star - \vec{u}^n}{\Delta t} + \vec{u}^n \cdot \nabla \vec{u}^n = \vec{f} \tag{7}$$

and subsequently adding in the pressure forces via

$$\frac{\vec{u}^{n+1} - \vec{u}^\star}{\Delta t} = -\frac{1}{\rho}\nabla p, \tag{8}$$

where the pressure is calculated by solving a Poisson equation of the form

$$\nabla \cdot \frac{1}{\rho}\nabla \hat{p} = \nabla \cdot \vec{u}^\star \tag{9}$$

and $\hat{p} = p\Delta t$.

(7) is solved on the fine simulation grid and represents C_l. The resulting velocities $\vec{u}^\star$ are then mapped to coarse grid (step 1). (8) and (9) representing C_{cp} are then solved on the coarse grid (step 2). After determining $\vec{u}^{n+1}$ on the coarse grid, we then determine $\vec{u}^{n+1}$ on the fine grid, as represented by C_{fp} (steps 3-4).

3.2 Mapping to the Coarse Grid

Conceptually, our scheme was designed by taking a fine simulation grid and performing binary coarsening up to some level as is done with octrees. This results in both a fine and coarse grid that we could use in our algorithm. However, our framework is actually much more flexible and resembles that proposed in [Losasso et al. 2006]. They started with a base level grid and place an octree in each cell so that the resolution could be increased locally. This was done to lower the computational cost of accessing octree nodes by removing the top of the tree which replaces some number of levels with uniform grid access. Their octrees could be any size including no refinement at all, see figures 4(a) and 4(b). The deepest levels of their octrees would represent the simulation grid in our framework whose degrees of freedom we would like to upgrade and their highest level represents the uniform coarse grid in which we will carry out the projection C_{cp}. However, our method for C_{fp} is general enough to not only support different levels of octree refinement as shown in figures 4(a) and 4(b) but to also support uniform grids inserted into each coarse grid cell as shown in Figure 4(c) or even multiple levels of uniform grids as shown in Figure 4(d). This flexibility comes from our method for solving each coarse cell individually. However, we note that for most of our examples we chose to use uniform grids within each coarse cell as in Figure 4(c).

From the degree of freedom standpoint, every face on the finest resolution of all grids represents one degree of freedom for velocity whereas each face of the base coarse grid represents a degree of freedom for a much coarser set of velocities. We map the fine grid velocities to the coarser grid by using an area-weighted average of all the fine grid velocities which is given by,

$$\vec{u}_c^\star = \frac{1}{n}\sum_{f \in faces} A_f \vec{u}_f^\star \tag{10}$$

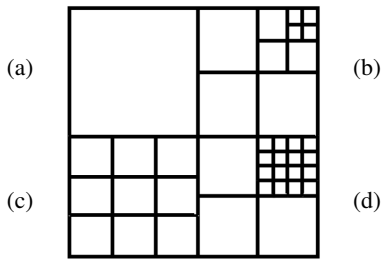

Figure 4: *A refined grid with four coarse cells. (a) A coarse cell with no refinement. (b) A coarse cell with an octree inside (c) A coarse cell with a uniform grid inside. (d) A coarse cell with multiple levels of uniform grids inside.*

where $faces$ is the set of fine scale faces that overlap with the coarse face c, A_f is the area of the fine face, and n is the number of elements in $faces$. One could liken this to methods that map particles to a background grid such as the PIC/FLIP method proposed in [Zhu and Bridson 2005], the vortex particles proposed in [Selle et al. 2005], or the SPH method proposed in [Losasso et al. 2008]. All these authors had proposed methods for mapping from the velocity degrees of freedom defined on particles to a background coarse grid, performing computation on the coarse grid, and then mapping information back. Contrary to these techniques in which the typical particle to grid mapping aims to have every particle influence the coarse grid, we do not map velocity degrees of freedom that are not incident on the coarse faces, meaning that they have no influence on the coarse grid. We admit that it is desirable to map more degrees of freedom; however, it turns out that our mapping allows for a more efficient handling of boundary conditions and is one of the key ideas that allows C_{fp} to scale linearly with the number of processors.

3.3 Coarse Grid Projection

Although we can use the standard Navier-Stokes equations for the coarse projection in the absence of objects, we must modify these equations when dealing with objects or octrees. This is because our objects are rasterized on the fine simulation grid, and a coarse grid face can thus contain a fraction of an object. Following [Losasso et al. 2004], we used the volume-weighted Poisson equation given by

$$V_{cell}\nabla \cdot \left(\vec{u}^\star - \frac{1}{\rho}\nabla \hat{p}\right) = 0. \tag{11}$$

We can then write

$$V_{cell}\nabla \cdot \vec{u}^\star = A_{face}\sum_{f \in faces}\left(\vec{u}_f^\star \cdot \vec{n}_f\right)\beta_f + \left(\vec{u}_s \cdot \vec{n}_f\right)(1-\beta_f) \tag{12}$$

and

$$V_{cell}\nabla \cdot \left(\frac{1}{\rho}\nabla \hat{p}\right) = A_{face}\sum_{f \in faces}\left(\left(\frac{1}{\rho}\nabla \hat{p}\right)_f \cdot \vec{n}_f\right)\beta_f \tag{13}$$

where $\vec{u}_s$ is the velocity of the solid and β_f is the fraction of the face that is not covered by a solid. Using (11) with definitions (12) and (13) instead of (9) on the coarse grid allows for high fidelity modeling of stationary and moving solids as shown in figures 3 and 5. Note that in (12) and (13) we have factored A_{face} out in front of the sum as opposed to [Losasso et al. 2004]. This is because on coarse grid all of our faces have the same size. As long as this is true, one can omit A_{face} from the equations and still retain symmetry. Otherwise it is more appropriate to include A_{face} inside the summation.

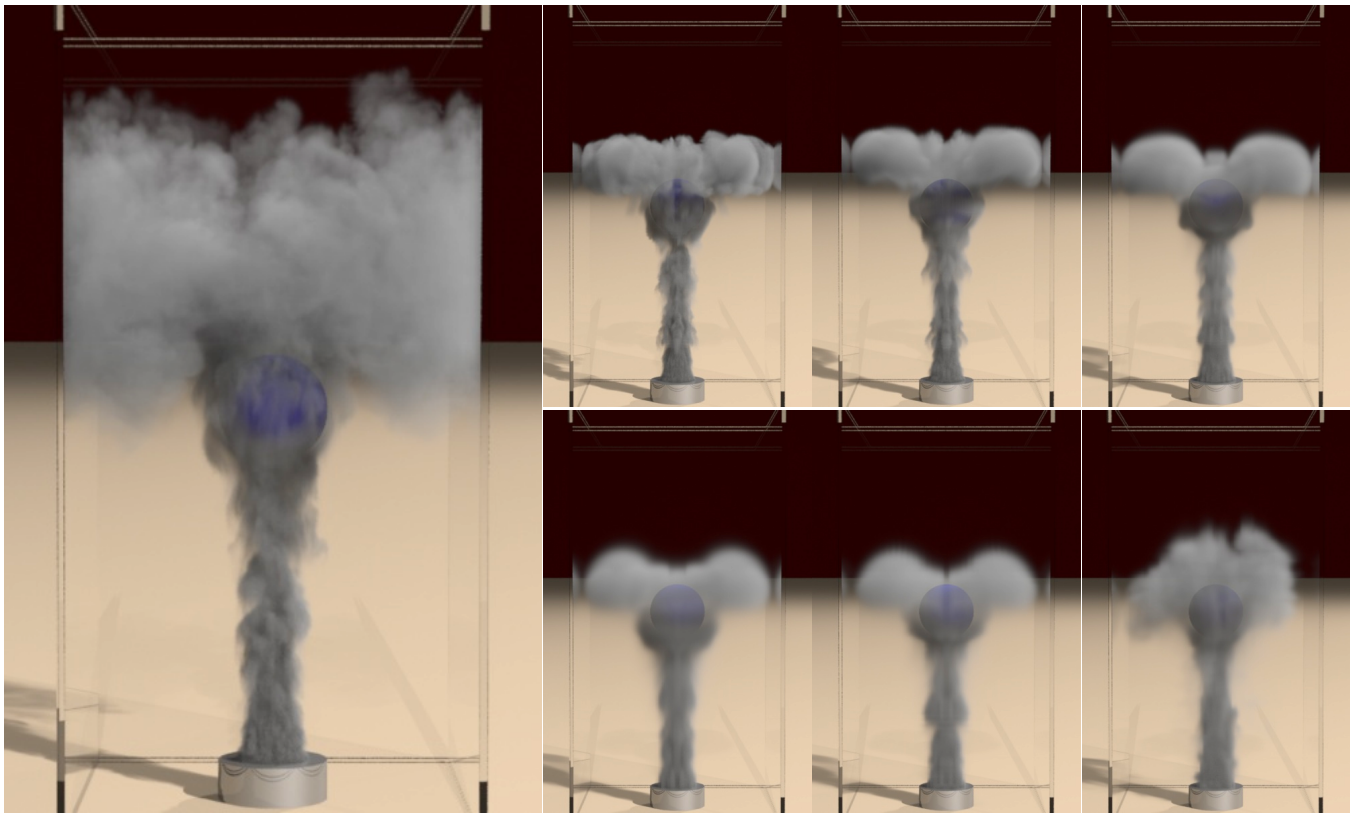

Figure 5: *An example with smoke flowing around a static sphere. The large figure is a* $512 \times 1024 \times 512$ *simulation with a* $64 \times 128 \times 64$ *coarse grid. The smaller figures are comparisons of a simulation using different grid resolutions. The resolutions are starting from the left to right, top to bottom: a* $512 \times 1024 \times 512$ *simulation with a* $64 \times 128 \times 64$ *coarse grid, a* $256 \times 512 \times 256$ *simulation with a* $64 \times 128 \times 64$ *coarse grid, a* $128 \times 256 \times 128$ *simulation with a* $64 \times 128 \times 64$ *coarse grid, a* $64 \times 128 \times 64$ *base simulation, a* $64 \times 128 \times 64$ *simulation with a* $32 \times 64 \times 32$ *coarse grid, and a* $64 \times 128 \times 64$ *simulation with a* $16 \times 32 \times 16$ *coarse grid with Kolmolgorov noise.*

3.4 Mapping to the Fine Grid

After solving for a divergence-free field on the coarse grid, we then need to map results of (8) and (9) back to the fine simulation grid. As mentioned earlier, many particle methods do this; however, whereas particle-based methods require every particle degree of freedom to receive information from the coarse grid, we instead only map to the fine grid faces that are incident upon the coarse grid faces which we once again stress is a key to our algorithm. As is widely done in PIC/FLIP type methods, one can map either velocities or changes in velocities back to the fine degrees of freedom (see e.g. [Zhu and Bridson 2005]). In other words, our new velocities for a given fine simulation grid face are:

$$\vec{u}_f^{n+1} = \alpha \vec{u}_c^{n+1} + (1-\alpha)(\vec{u}_f^{\star} + \vec{u}_c^{n+1} - \vec{u}_c^{\star}) \qquad (14)$$

where α is a constant between 0 and 1, $\vec{u}_c$ are the velocities on our coarsened uniform grid and $\vec{u}_f$ are the velocities on the fine simulation grid. Note that in the case with α equal to 1, every fine grid face is set to have the same velocity as the coarse grid face containing it. In this case, the total flux of material into or out of any coarse grid cell is defined by the local fluxes of the fine grid and is still divergence free, although using α equal to 1 severely dampens the flow field. When α equals 0 the fine grid velocities $\vec{u}_f^{\star}$ simply acquire a change in velocity equal to the change that the coarse grid experienced. Because we created the coarse grid velocity $\vec{u}_c^{\star}$ with an area-weighted average of the $\vec{u}_f^{\star}$ incident upon the face, adding this change to every fine grid cell incident on the face creates a net flux of 0 through the cell and thus a divergence-free flow field as defined by the fine grid faces. This is much less dissipative because it uses a constant velocity difference to update the fine grid cells as opposed to a constant velocity. Note that when α is not 1 or 0, since the divergence operator is linear, one still obtains a divergence-free field on each coarse cell as defined by the fine grid velocities. Moreover, one can blend a degree of dissipation into the numerical method as is typical of a PIC/FLIP scheme. In our examples, we found that lower values for an α create more detailed and dynamic results and thus use a value of 0. Of course if one desires the flow to be more damped a higher value of α could easily be substituted. However, in our examples, we found that additional damping was not needed. After the mapping, we must determine the fine grid velocities interior to each coarse grid cell to make a divergence-free velocity field.

3.5 Fine Grid Local Projections

The method we used for mapping to the coarse grid, carrying out the coarse grid projection, and mapping back both gives continuity across each coarse grid cell as defined by the fine grid velocity degrees of freedom as well as a divergence-free coarse velocity field. In order to create a divergence-free field on the fine grid, we first consider each coarse grid cell to be its own unique computational domain. $\vec{u}^{\star}$ is given for every interior degree of freedom as was computed previously by (7). We then solve (8) and (9) for those interior degrees of freedom with fixed velocity boundary conditions for every fine grid velocity incident on the boundaries of this coarse grid cell.

If the coarse grid cell is not refined as in Figure 4(a), nothing needs

(a) (b) (c) (d) (e)

Figure 6: *A 2D smoke simulation run with a* 128×128 *base grid. (a) is a simulation on a* 128×128 *fine grid and a* 64×64 *coarse grid using interpolation. (b) is our method using a* 128×128 *grid and a* 64×64 *grid for projection. (c) is a base simulation on a* 128×128 *grid. (d) is a simulation on a* 256×256 *fine grid and a* 128×128 *coarse grid with interpolation. (e) is our method using a* 256×256 *grid with a* 128×128 *grid for projection.*

to be done. If the coarse grid cell contains a uniform grid as in Figure 4(c), one simply solves (8) and (9) on that uniform grid. If the coarse grid cell contains an octree as in Figure 4(b), one can simply use the octree method of [Losasso et al. 2004] in order to find a divergence-free set of velocities for the interior degrees of freedom. We also propose a new method that can be used based on hierarchical subdivision. Consider Figure 4(b), looking only at the first level of octree refinement. In this case, four new degrees of freedom are added, two vertical velocities and two horizontal velocities. For this example, (8) has four unknowns at the center of each of the four computational cells. However, because there are Neumann boundary conditions, the system has a rank 1 degeneracy and one unknown pressure can be removed. Thus we can set up a 3×3 system of equations and solve with a fast direct method. For faces between cells that are not refined any further, that represents the final velocity degree of freedom. This is true for the bottom and left velocities in Figure 4(b). For cells that are refined further, e.g. the top and right velocities in that figure, one can use (14) to map either the velocities or change in velocity to those faces. One would then consider each subcell such as the upper right hand corner of Figure 4(b) and repeat the process by subdividing once and solving the 3×3 system.

In two spatial dimensions, these 3×3 matrices are very quick to invert and this leads to a very quick hierarchical solver that can be efficiently implemented for example on a GPU. In three dimensions there are eight pressures and seven degrees of freedom, leading to a 7×7 matrix which can also be solved quickly. One might also imagine using the hierarchical subdivision only to do the first level and using the octree solver to solve the upper right hand corner of Figure 4(b). Similarly, consider Figure 4(d), one can use the first level of the hierarchical approach to get the first four velocities and then apply a uniform grid solver on the upper right hand corner. Note that when solving these small matrices, one can use a direct method, such as Cholesky factorization, since these matrices are symmetric positive definite after the extra degree of freedom is eliminated. Because these systems are fairly quick to solve and because there are many independent systems to solve, one can imagine using threads to solve these systems simultaneously. This problem also lends itself well to specialized architectures such as the GPU that can very quickly execute many low cost processes. We note that although we implemented and tested all of the above methods for the fine local projections, for larger subgrids such as $8 \times 8 \times 8$ we primarily used one level of refinement with PCG as we found that this was fastest.

4 Discussion

In our examples section, we choose various examples of both smoke and water and ran them using a standard method on a reasonable base grid. We then analyzed two different strategies for using our method. One was to carry out grid refinements and achieve scaling as exemplified by (4) where the base grid resolution was used for our projection in all finer grid simulations. This was done to illustrate that very high resolution simulations can be run using our method without suffering the curse of dimensionality on C_{cp}. As was proposed in [Fedkiw et al. 2001], the vorticity confinement used in the base simulation was reduced linearly proportional to the size of the grid for the finer grid simulations. Even with less vorticity confinement we achieved significantly more computational detail. We also consider whether our base simulation could be accelerated using an even coarser grid for the projection step. As these grids became coarser and coarser, for example sixteen grid cells in a dimension, we did notice artifacts resulting from using our method. However, we stress that a fluid simulation on a very low resolution grid would be enormously simplified as shown in Figure 7. To alleviate these artifacts on the very coarse grid, we verified that feasible but not spectacular results could be obtained by using a Kolmolgorov spectrum as done in [Rasmussen et al. 2003] in combination with a $16 \times 32 \times 16$ grid but stress that it was only used in this one simulation to demonstrate that even at resolutions we would not recommend, one could obtain a plausible result.

Figure 6 illustrates that not all algorithms that scale according to (4) produce good results. Figure 6(c) is a baseline simulation on a 128×128 grid to give a sense one would expect to obtain with a standard method. Figure 6(b) shows our method on a 128×128

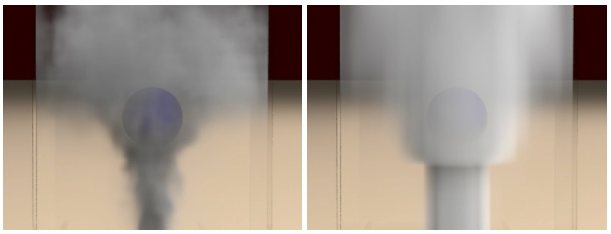

Figure 7: *A comparison between (Left) a simulation using our method with a fine resolution of* $64 \times 128 \times 64$ *and a coarse resolution of* $16 \times 32 \times 16$ *and (Right) without using our method on a* $16 \times 32 \times 16$ *grid.*

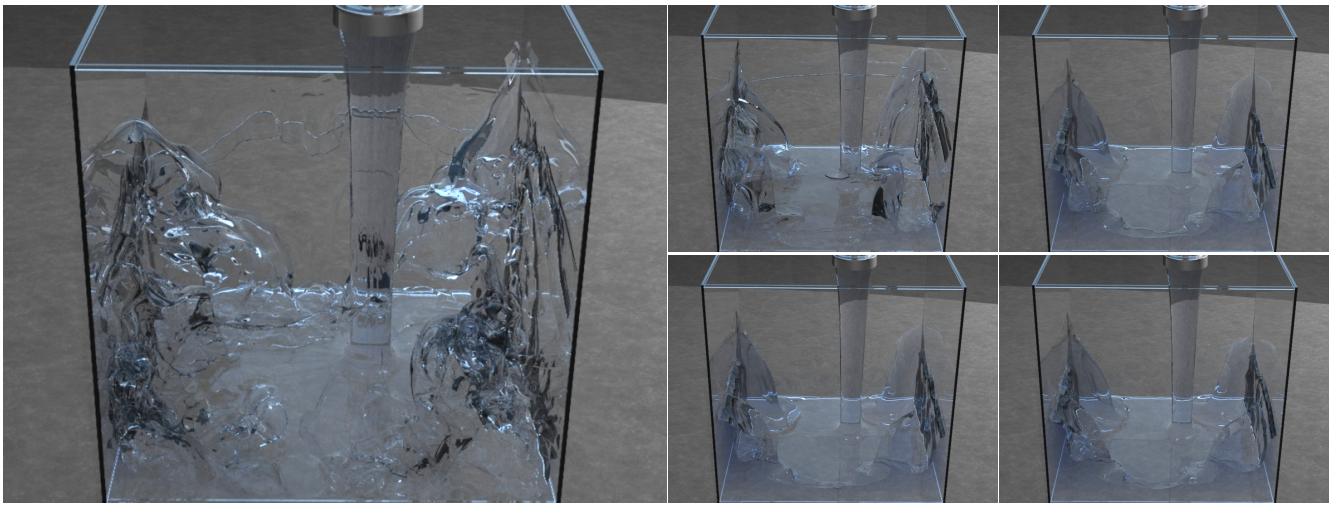

Figure 8: *An example with water pouring into a box. This is a comparison of a simulation using different grid resolutions. The resolutions are starting from the left to right, top to bottom: a* $512 \times 512 \times 512$ *simulation with a* $128 \times 128 \times 128$ *coarse grid, a* $256 \times 256 \times 256$ *simulation with a* $128 \times 128 \times 128$ *coarse grid, a* $128 \times 128 \times 128$ *base simulation, a* $128 \times 128 \times 128$ *simulation with a* $64 \times 64 \times 64$ *coarse grid, and a* $128 \times 128 \times 128$ *simulation with a* $32 \times 32 \times 32$ *coarse grid.*

grid using a grid which was coarsened to 64×64 for the projection step. Note that figures 6(b) and (c) are quite similar in detail and structure. Figure 6(a) uses our algorithm except that the fine grid projections represented by C_{fp} are replaced with simple interpolation from the coarse grid. This method would also scale as (4) but produces inferior results. Not only is detail lost, but more importantly, the structure of the smoke is lost due to added viscosity. This exemplifies the need for the fine grid projections represented by C_{fp} in order to obtain detail and structure that would be present on a finer grid. However, we note that there are other techniques such as [Kim et al. 2008] and [Yoon et al. 2009] that do not need to run a large simulation. Instead they run a coarse simulation and add details as a postprocess at the fine scale.

We carried out this same comparison for the case of using a finer simulation resolution but using the same resolution for the projection as the base simulation shown in Figure 6(c). Figure 6(e) shows the results obtained using our method. Note the highly increased structure and detail whereas Figure 6(d), which is obtained using interpolation, has no obvious added benefits over Figure 6(c). In summary, simply using our coarsening methodology with linear interpolation loses large amounts of detail in coarsening (Figure 6(c) becomes Figure 6(a)) and gains no detail when refining (Figure 6(c) becomes Figure 6(d)). In contrast, our method preserves detail when coarsening (Figure 6(c) becomes Figure 6(b)), and gains significantly more detail when refining (Figure 6(c) becomes Figure 6(e)). Although our results do achieve similar details to the fine grid base simulation, our results do have numerical differences. Figure 9 demonstrates the quantitative differences between our method and a standard projection on the fine grid.

While not our original intent, we also explored using our technique for liquids. As for smoke, refinement worked rather well. An added complication with water is in dealing with the free surface where constant pressure Dirichlet boundary conditions are imposed. Since the coarse grid does not contain the pressure degrees of freedom to represent the boundary conditions on the fine grid, one has to either over or under estimate mixed air/water regions. Failing to put constant pressure Dirichlet boundary conditions on the coarse grid cell means that that cell will be solved in a divergence-free manner on the coarse grid, which is preserved by our mapping to the fine grid. This means that any air pockets in that cell would not collapse properly. Moreover, wave type forces that are created by different height columns of water would be lost. If instead, the coarse cell constant pressure boundary condition is set, this treats the whole cell as if it was air and may allow too much or too little flow inside as this cell is not divergence free. This can result in underwater air pockets not only collapsing but losing mass. We tried both of these methods and observed both phenomena to some degree.

Because water waves are generated based on pressure differences one would expect that water would be more sensitive to the coarse grid approximation than smoke. However, as shown in Figure 8, we found that refining a reasonable base simulation added detail without noticeable issues until the surface started to become flat and at rest, at which point some minor artifacts could be seen. Under coarsening, however, these artifacts were harder to ignore. Therefore, we decided to use the coarse grid approximation to obtain the flow field only for cells interior to the water. We then collected all fine scale cells near the free surface (i.e. fine grid cells inside a coarse cell that contains a mixture of fluid and air) for a second Poisson solve that used the velocities from the fine grid projections as Neumann boundary conditions. Performing this larger solve removed the artifacts albeit adding to the computation cost. However, since the cells near the surface only represent a lower dimensional set, essentially two spatial dimensions out of three, this additional cost disappears under grid refinement, although on the grids we used there was some overhead. We also note that this overhead tends to be fairly minimal because the surface solve is only needed for relatively calm fluid, making the solution quicker to obtain.

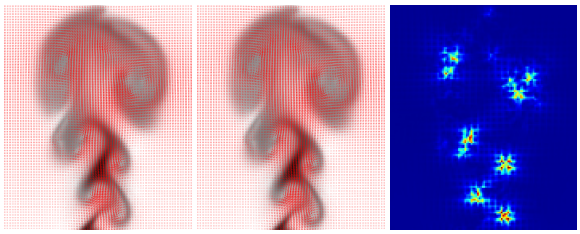

Figure 9: *A quantitative comparison of our technique and a standard fluid simulation. (Left) Our technique with a* 128×256 *fine grid and a* 32×64 *coarse gird. (Center) The base simulation with a* 128×256 *grid. (Right) A comparison of the velocities between the two techniques. The warmer colors illustrate bigger differences. The maximum velocity error is about 1%.*

Example	Fine	Coarse	Fine 1 proc	Coarse 1 proc	Ours 1 proc	Fine 64 procs	Coarse 64 procs	Ours 64 procs
Static Sphere	$64 \times 128 \times 64$	$16 \times 32 \times 16$	$5.6m/34s$	$1.2s/0.4s$	$20s/2s$	$2.2m/13s$	$12s/4s$	$30s/3s$
Static Sphere	$64 \times 128 \times 64$	$32 \times 64 \times 32$	$5.6m/34s$	$4.5s/0.9s$	$20s/2s$	$2.2m/13s$	$30s/6s$	$1m/6s$
Static Sphere	$128 \times 256 \times 128$	$64 \times 128 \times 64$	$5.8h/7m$	$5.6m/34s$	$8.3m/25s$	$16m/19s$	$2.2m/13s$	$2.7m/8s$
Static Sphere	$256 \times 512 \times 256$	$64 \times 128 \times 64$	$50h/1h$	$5.6m/34s$	$1.6h/2m$	$2h/2.5m$	$2.2m/13s$	$9m/11s$
Static Sphere	$512 \times 1024 \times 512$	$64 \times 128 \times 64$	-	$5.6m/34s$	-	$47h/28m$	$2.2m/13s$	$43m/26s$
Moving Sphere	$64 \times 128 \times 64$	$16 \times 32 \times 16$	$8m/32s$	$0.5s/0.1s$	$20s/2s$	$2m/12s$	$15s/5s$	$40s/4s$
Moving Sphere	$64 \times 128 \times 64$	$32 \times 64 \times 32$	$8m/32s$	$3s/0.6s$	$20s/2s$	$2m/12s$	$45s/9s$	$1.1m/7s$
Moving Sphere	$128 \times 256 \times 128$	$64 \times 128 \times 64$	$9h/7m$	$8m/32s$	$16m/31s$	$10m/20s$	$2m/12s$	$6m/12s$
Moving Sphere	$256 \times 512 \times 256$	$64 \times 128 \times 64$	$75h/1h$	$8m/32s$	$2.5h/2m$	$3h/2.5m$	$2m/12s$	$17m/15s$
Moving Sphere	$512 \times 1024 \times 512$	$64 \times 128 \times 64$	-	$8m/32s$	-	$70h/28m$	$2m/12s$	$70m/28s$
Water	$128 \times 128 \times 128$	$32 \times 32 \times 32$	$10m/15s$	$20s/2s$	$5.3m/8s$	$2m/3s$	$25s/2.5s$	$1m/1.5s$
Water	$128 \times 128 \times 128$	$64 \times 64 \times 64$	$10m/15s$	$2.3m/7s$	$6.7m/10s$	$2m/3s$	$1m/3s$	$1m/1.5s$
Water	$256 \times 256 \times 256$	$128 \times 128 \times 128$	$3.7h/2.2m$	$10m/15s$	$2h/74s$	$43m/17s$	$2m/3s$	$11m/4.5s$
Water	$512 \times 512 \times 512$	$128 \times 128 \times 128$	-	$10m/15s$	-	$16h/3.2m$	$2m/3s$	$65m/13s$

Table 1: *Timing information for our examples as well as base simulations on the fine and coarse grid using both 1 processor and 64 processors. Some large resolution simulations (noted by -) could not be run on a single processor due to RAM restrictions. All of our timings are given in time per frame/time per time step. Note that all examples were run at 24 frames per second and with a CFL number of 0.9.*

5 Examples

We demonstrate the effectiveness of our algorithm on a number of smoke and water simulations. All our smoke simulations contained a density source at the bottom of the domain, and our water simulations contained a water source at the top of the domain. Figure 6 shows two-dimensional smoke examples for illustration. We then demonstrate our algorithm for three dimensional examples of smoke and water. For each of these examples, we first ran a base simulation without using our method. We then used our method to coarsen the projection resolution, which improves the performance while maintaining similar looking results. We also used our method to refine the base simulation, showing that we can achieve very detailed results in a reasonable amount of time.

5.1 Smoke

We ran a number of three-dimensional smoke simulations as shown in figures 3 and 5. We would like to point out the stark differences in detail resulting from the different structures of the smoke that can be achieved by using our method to refine the grid resolution. For example, when comparing the smoke examples of Figure 3, we achieve two distinct vortex rings in the high resolution example but only achieve one with the lower resolutions. Also note the fine scale vorticies around the sphere as shown in Figure 5.

5.2 Water

For water, we ran a number of simulations as shown in Figure 8. For these examples, we used the surface solve towards the end of the simulations when the fluid starts to settle but did not when there is highly turbulent flow near the beginning of the simulation. As with smoke, we can achieve a large amount of additional detail by using our method to increase the resolution of the water.

5.3 Timing

The timings for our simulations are shown in Table 1. We compare our results with those obtained by running a base simulation on the coarse grid, and a base simulation on the refined grid. Note, for example, that our method runs approximately 67 times faster on the high resolution static sphere example, shown on line four of the table above. This makes previously infeasible simulations tractable. An important detail to note is that a base high resolution simulation quickly runs up against hard memory limits on a machine. This is partially alleviated with our method, as there is significantly less memory access during the projection step.

When comparing our method to base simulations on the coarse grid, our method runs slower when using a single processor. This is because a large amount of time is spent in C_{fp}. However, our method scales well with a large number of processors, and we achieve similar timings to the coarse grid base simulations for each time step. This is because the cost of C_{fp} scales linearly and as a result, both algorithms are dominated by the global projection (C_{cp} or C_p). We note that in some cases our method actually runs more quickly per time step than the base simulation because we take smaller time steps for a larger resolution grid.

6 Conclusions and Future Work

We have introduced a novel algorithm that improves the performance of existing fluid simulations and can achieve realistic results using large fluid grids. Our algorithm effectively reduces the amount of time required for the Poisson solve by using a coarse grid projection and then small projections within each coarse grid cell.

Although our method was used to solve the Poisson equation for incompressible flow, (8) updates $u^\star$ to u^{n+1} by subtracting $\frac{1}{\rho}\nabla\hat{p}$. This means that at every face of our grid we define $\nabla\hat{p}$, the derivatives of p, on the entire grid. We can think of this as solving a general Poisson equation $\nabla \cdot \left(\frac{1}{\rho}\nabla p\right) = f$ for some f. Our method provides a technique for quickly and efficiently finding an approximation for ∇p on the fine grid. Because many other applications make use of Poisson equations and/or its derivatives, it would be interesting to explore the use of our methodology in those areas. However, one should be cautious of the fact that we are not doing a formal Hodge decomposition, meaning that the divergence-free vector field that we get is not the unique field that decomposes $u^\star$ into a divergence-free field plus the gradient of a scalar field. That being said, we have found that this approximation works very well for our applications and is likely to do so for others.

One interesting avenue of future work would be to integrate our method with a multi-grid solver. Standard multi-grid solvers often need special treatment for problems involving free surfaces, object boundaries, special exterior domain boundary conditions, etc. Otherwise, their convergence rate can be a bit slow. The difficulty lies in that multi-grid solutions do not obtain a divergence-free flow field until they are fully converged (this is also true for cg and all traditional methods). Our method provides an interesting twist in that it can be used as a prolongation operator for multi-grid. One use of this is that instead of running a multi-grid solver fully to convergence, the solver can be short-circuited and our method can be used as a final prolongation producing a divergence-free flow field.

Acknowledgements

Research supported in part by ONR N0014-06-1-0393, ONR N00014-06-1-0505, ONR N00014-05-1-0479 for a computing cluster, NIH U54-GM072970, NSF ACI-0323866, and King Abdullah University of Science and Technology (KAUST) 42959. M.L. was supported in part by an Intel Ph.D. Fellowship. We would like to thank Christos Kozyrakis for additional computing resources and Jacob Leverich for helping us use those resources.

References

BERGER, M., AND OLIGER, J. 1984. Adaptive mesh refinement for hyperbolic partial differential equations. *J. Comput. Phys. 53*, 484–512.

BOLZ, J., FARMER, I., GRINSPUN, E., AND SCHRODER, P. 2003. Sparse matrix solvers on the gpu: Conjugate gradients and multigrid. *ACM Trans. Graph. (SIGGRAPH Proc.) 22*, 3, 917–924.

BRIDSON, R., HOURIHAM, J., AND NORDENSTAM, M. 2007. Curl-noise for procedural fluid flow. *ACM Trans. Graph. 26*, 3, 46.

DESBRUN, M., AND CANI, M.-P. 1996. Smoothed particles: A new paradigm for animating highly deformable bodies. In *Comput. Anim. and Sim. '96 (Proc. of EG Wrkshp. on Anim. and Sim.)*, Springer-Verlag, R. Boulic and G. Hegron, Eds., 61–76.

DUPONT, T., AND LIU, Y. 2003. Back and forth error compensation and correction methods for removing errors induced by uneven gradients of the level set function. *J. Comput. Phys. 190/1*, 311–324.

FEDKIW, R., STAM, J., AND JENSEN, H. 2001. Visual simulation of smoke. In *Proc. of ACM SIGGRAPH 2001*, 15–22.

FOSTER, N., AND METAXAS, D. 1997. Controlling fluid animation. In *Comput. Graph. Int.*, 178–188.

GAO, Y., LI, C.-F., HU, S.-M., AND BARSKY, B. A. 2009. Simulating gaseous fluids with low and high speeds. *Comput. Graph. Forum 28*, 7, 1845–1852.

HORVATH, C., AND GEIGER, W. 2009. Directable, high-resolution simulation of fire on the gpu. *ACM Trans. Graph. 28*, 3, 1–8.

KIM, B.-M., LIU, Y., LLAMAS, I., AND ROSSIGNAC, J. 2005. Using BFECC for fluid simulation. In *Eurographics Workshop on Natural Phenomena 2005*.

KIM, T., THÜREY, N., JAMES, D., AND GROSS, M. 2008. Wavelet turbulence for fluid simulation. In *SIGGRAPH '08: ACM SIGGRAPH 2008 papers*, 1–6.

KIM, D., SONG, O.-Y., AND KO, H.-S. 2009. Stretching and wiggling liquids. In *SIGGRAPH Asia '09: ACM SIGGRAPH Asia 2009 papers*, ACM, New York, NY, USA, 1–7.

LAMORLETTE, A., AND FOSTER, N. 2002. Structural modeling of flames for a production environment. *ACM Trans. Graph. (SIGGRAPH Proc.) 21*, 3, 729–735.

LOSASSO, F., GIBOU, F., AND FEDKIW, R. 2004. Simulating water and smoke with an octree data structure. *ACM Trans. Graph. (SIGGRAPH Proc.) 23*, 457–462.

LOSASSO, F., FEDKIW, R., AND OSHER, S. 2006. Spatially adaptive techniques for level set methods and incompressible flow. *Computers and Fluids 35*, 995–1010.

LOSASSO, F., TALTON, J., KWATRA, N., AND FEDKIW, R. 2008. Two-way coupled sph and particle level set fluid simulation. *IEEE Trans. on Vis. and Comput. Graph. 14*, 4, 797–804.

MOLEMAKER, J., COHEN, J., PATEL, S., AND NOH, J. 2008. Low viscosity flow simulations for animation. In *SCA '08: Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, Eurographics Association, 9–18.

MULLEN, P., CRANE, K., PAVLOV, D., TONG, Y., AND DESBRUN, M. 2009. Energy-preserving integrators for fluid animation. In *SIGGRAPH '09: ACM SIGGRAPH 2009 papers*, 1–8.

MÜLLER, M., CHARYPAR, D., AND GROSS, M. 2003. Particle-based fluid simulation for interactive applications. In *Proc. of the 2003 ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.*, 154–159.

NARAIN, R., SEWALL, J., CARLSON, M., AND LIN, M. C. 2008. Fast animation of turbulence using energy transport and procedural synthesis. In *SIGGRAPH Asia '08: ACM SIGGRAPH Asia 2008 papers*, ACM, New York, NY, USA, 1–8.

NIELSEN, M. B., CHRISTENSEN, B. B., ZAFAR, N. B., ROBLE, D., AND MUSETH, K. 2009. Guiding of smoke animations through variational coupling of simulations at different resolutions. In *SCA '09: Proc. of the 2009 ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.*, 217–226.

PFAFF, T., THUEREY, N., SELLE, A., AND GROSS, M. 2009. Synthetic turbulence using artificial boundary layers. In *SIGGRAPH Asia '09: ACM SIGGRAPH Asia 2009 papers*, 1–10.

RASMUSSEN, N., NGUYEN, D., GEIGER, W., AND FEDKIW, R. 2003. Smoke simulation for large scale phenomena. *ACM Trans. Graph. (SIGGRAPH Proc.) 22*, 703–707.

REEVES, W. 1983. Particle systems - a technique for modeling a class of fuzzy objects. In *Comput. Graph. (Proc. of SIGGRAPH 83)*, vol. 17, 359–376.

SCHECHTER, H., AND BRIDSON, R. 2008. Evolving sub-grid turbulence for smoke animation. In *SCA '08: Proc. of the 2008 ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.*, 1–7.

SELLE, A., RASMUSSEN, N., AND FEDKIW, R. 2005. A vortex particle method for smoke, water and explosions. *ACM Trans. Graph. (SIGGRAPH Proc.) 24*, 3, 910–914.

SELLE, A., FEDKIW, R., KIM, B., LIU, Y., AND ROSSIGNAC, J. 2008. An unconditionally stable MacCormack method. *Journal of Scientific Computing 35*, 2, 350–371.

STAM, J., AND FIUME, E. 1993. Turbulent wind fields for gaseous phenomena. In *Proc. of SIGGRAPH 1993*, 369–376.

STAM, J. 1999. Stable fluids. In *Proc. of SIGGRAPH 99*, 121–128.

WICKE, M., STANTON, M., AND TREUILLE, A. 2009. Modular bases for fluid dynamics. In *SIGGRAPH '09: ACM SIGGRAPH 2009 papers*, ACM, New York, NY, USA, 1–8.

YOON, J.-C., KAM, H. R., HONG, J.-M., KANG, S.-J., AND KIM, C.-H. 2009. Procedural synthesis using vortex particle method for fluid simulation. *Comput. Graph. Forum 28*, 7, 1853–1859.

ZHU, Y., AND BRIDSON, R. 2005. Animating sand as a fluid. *ACM Trans. Graph. (SIGGRAPH Proc.) 24*, 3, 965–972.