Eurographics/ ACM SIGGRAPH Symposium on Computer Animation (2008)
M. Gross and D. James (Editors)

Low Viscosity Flow Simulations for Animation

Jeroen Molemaker[1,2], Jonathan M. Cohen[1,3], Sanjit Patel[1], Jonyong Noh[1,4]

[1]Rhythm and Hues, USA
[2]Institute of Geophysics and Planetary Physics, UCLA, USA
[3]NVIDIA, USA
[4]Graduate School of Culture Technology, KAIST, Korea

Abstract

We present a combination of techniques to simulate turbulent fluid flows in 3D. Flow in a complex domain is modeled using a regular rectilinear grid with a finite-difference solution to the incompressible Navier-Stokes equations. We propose the use of the QUICK advection algorithm over a globally high resolution grid. To calculate pressure over the grid, we introduce the Iterated Orthogonal Projection (IOP) framework. In IOP a series of orthogonal projections ensures that multiple conditions such as non-divergence and boundary conditions arising through complex domains shapes or moving objects will be satisfied simultaneously to specified accuracy. This framework allows us to use a simple and highly efficient multigrid method to enforce non-divergence in combination with complex domain boundary conditions. IOP is amenable to GPU implementation, resulting in over an order of magnitude improvement over a CPU-based solver. We analyze the impact of these algorithms on the turbulent energy cascade in simulated fluid flows and the resulting visual quality.

1. Introduction

Simulation of fluid phenomena is an important part of visual simulation of the natural world. Since the paper of Stam [Sta99], numerical simulation of fluids has become prevalent in computer graphics. However, many popular fluid simulation methods suffer from excessive numerical dissipation. Visually, this leads to viscous, sluggish looking flows which are inadequate for highly energetic and turbulent phenomena such as shown in Figure 1.

We propose two strategies for combating excessive numerical dissipation. The first is to use a *low viscosity advection algorithm*. Semi-Lagrangian advection algorithms [Sta99] allow for large numerical time-steps that are not limited by a CFL condition, but come with the price of high numerical dissipation. This dissipative effect increases with the length of a time step. Higher-order semi-Lagrangian schemes [FSJ01, Str99, KLLR05] reduce the amount of numerical viscosity but still carry a relatively steep numerical dissipative penalty that increases as the ratio between actual time step and maximum CFL time step increases. We suggest the use of QUICK, an explicit third order advection algorithm that minimizes numerical dissipation while maintaining numerical stability. We advocate QUICK for computer graphics applications because we believe it provides an excellent balance among efficiency, ease of implementation, and visual appearance. QUICK was introduced by [Leo79] and is widely used in numerical atmospheric and ocean models where the need for minimal numerical dissipation is paramount [SM98]. Other explicit schemes have been proposed for computer graphics applications, for example [FM97]. However, we are not aware of QUICK's use previously, although [WHLP94] used the related QUICKEST algorithm to calculate stream surface functions for flow visualization.

In nonlinear turbulent flows, the momentum advection operator gives rise to *triad* interactions. Triad interactions involve three wave numbers and transfer energy from lower to higher spatial frequencies. An energy flux towards smaller scales is an inherent property of 3-dimensional turbulence. In fluids such as water, molecular viscosity acts as a sink of kinetic energy at scales of a few millimeters. In numerical models, it is essential that a form of dissipation is present at the smallest scales that are resolved by the grid (wave number $k = 2\pi/\Delta x$, where Δx is the grid spacing). Without such dissipation, energy will pile up unbounded at this wave number, leading to numerical instability [TL76]. Therefore any numerical code needs a form of dissipation, whether it is ex-

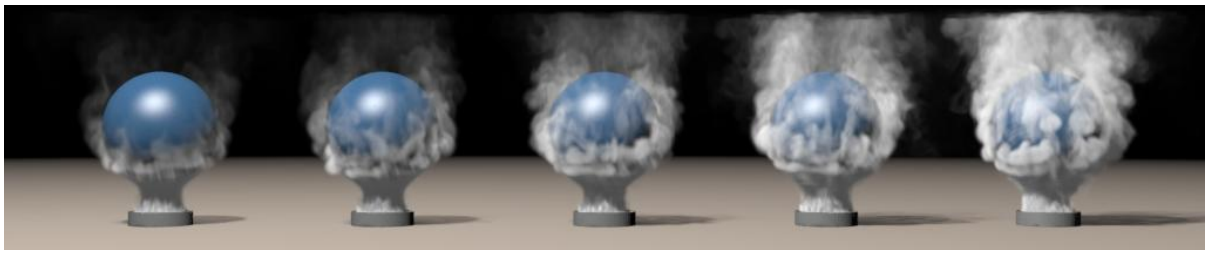


Figure 1: *Warm smoke rising around a sphere. There are no force terms in addition to thermal buoyancy – all turbulence arises naturally from instabilities inherent in the Navier-Stokes equations. The intensity of the heat source increases from left to right.*

plicitly modeled as a Laplace or a hyperviscosity operator, or in some other form such as numerical dissipation resulting from advection.

The turbulent kinetic energy spectrum is characterized by an energy spectrum that falls off with the wave number as $E_k = k^{-5/3}$ [TL76]. Therefore, as Δx shrinks, the highest resolved wave number increases, and the amount of kinetic energy that must be removed to prevent energy pile-up decreases. A scale selective advection operator such as QUICK removes only energy at the smallest scales required to keep the simulation stable for the grid resolution. Semi-Lagrangian advection, in contrast, is dissipative across a much wider range of the energy spectrum than is required. So while Semi-Lagrangian advection is dissipative enough that no further viscosity is needed for stability, it is more dissipative than is required for numerical stability, leading to loss of visual quality when simulating turbulent fluids. Figure 8 compares QUICK's spectral damping against that of several other popular advection algorithms.

Our second strategy for combating excessive dissipation is to use *globally high resolution grids*. The requirement for the flow to be relatively smooth at the smallest resolved scales leads to an upper bound of the Reynolds number ($Re = UL/\nu$, where U, L are velocity and length scale of the problem and ν the viscosity) that can be simulated by a particular grid. The number of required grid points in each direction scales as $Re^{3/4}$ [FP96]. Therefore, for a turbulent flow with high Reynolds number, we *must* use globally high resolution, not just locally high resolution as with an octree method. We emphasize that this argument does not depend on any particular discretization or algorithm, but *is inherent in any numerical approach that is limited to a range of resolved spatial scales*. Higher resolution grids do not per se result in lower numerical viscosity. Rather, a given grid resolution will allow solutions with a certain minimal amount of viscosity. In order to lower this amount, it is *necessary* to globally increase the grid resolution (but not sufficient). Thus a high resolution grid combined with QUICK's scale selective dissipation is able to produce very turbulent flows with guaranteed stability. Figure 2(c) shows how this results in more turbulent motion at higher resolutions.

In the incompressible Navier-Stokes equations, the pressure field is determined completely by the incompressibility condition and the flow boundary conditions. Solving for the pressure field requires the solution of a Poisson equation, which is an elliptical equation and therefore must be solved simultaneously over the entire grid. Most solvers used in the graphics community employ a preconditioned conjugate gradient (PCG) method to solve for pressure because PCG is easy to code and allows for complex domains and boundary conditions. PCG's drawback is that for large grids, which are required by our desire for globally high resolution, the running time scales poorly and quickly becomes the CPU bottleneck. Multigrid methods are also used to solve the Poisson equation, and have the advantage of scaling linearly in the number of unknowns. However, they can be complex to implement for internal boundary conditions.

We introduce a new iterative method for integrating a wide range of boundary conditions into a fast multigrid-based Poisson solver, which we term *Iterated Orthogonal Projections* (IOP). The IOP algorithm recasts the enforcement of non-divergence and boundary conditions as a series of orthogonal projections from the space of all discrete vector fields onto affine subspaces in which non-divergence and boundary conditions are satisfied. Furthermore, IOP is well-suited for implementation on a GPU, resulting in over an order of magnitude speedup over a CPU-based implementation. The range of boundary conditions applicable to IOP includes any condition that can be expressed as a linear constraint on the resulting fluid velocity field. This includes walls, moving objects, inflow, outflow, and continuative (open) boundaries. An important exception is free surface boundary conditions, because Dirichelet conditions on the pressure field do not translate into linear constraints on the fluid velocity field. Therefore our method only applies to single-phase fluids.

We propose combining the QUICK advection scheme with globally high resolutions grids, which are possible because we use a fast IOP-based Poisson solver. The two major contributions of our paper are:

- description QUICK and its analysis relative to other commonly used advection algorithms in graphics, and
- introduction of the IOP algorithm.

As we demonstrate in this paper, combining these techniques results in flows which exhibit very low numerical dissipation while remaining computationally efficient.

2. Related work

Finite difference solutions of the 3D Navier-Stokes equations have been popular in computer graphics since the work of [Sta99, FSJ01]. The "stable fluids" approach developed in these papers has been extended to other natural phenomena such as free surface flows [EMF02], explosions [FOA03], and fire [NFJ02]. Methods for improving the accuracy of gaseous fluid simulations have generally attempted to either improve the accuracy of flow boundary conditions, or improve the conservation of flow quantities such as vorticity, circulation, and momentum. [LGF04] developed an octree-based discretization capable of capturing highly detailed boundary conditions. An alternate approach [KFCO06] has focused on using an irregular grid for the computational mesh to exactly match complex boundary geometries. [KLLR05], [SFK*07], and [ZB05] also propose less dissipative advection algorithms to reduce numerical dissipation. Section 4 provides a direct comparison between QUICK and a number of alternate algorithms, including the BFECC method in [KLLR05]. We hope to make the case that our method is a viable alternative for turbulent flows. [ETK*07] has a goal very similar to our own, in that they seek specifically to minimize the amount of numerical dissipation. However, their approach is quite different – focussing on the conservation of vorticity, they recast the Navier-Stokes equations in the vortex formulation, and directly discretize and solve this equation over a simplicial (tetrahedral) mesh using discrete differential forms. This discretization discretely conserves circulation along closed paths through the domain.

An alternate method for simulating highly turbulent flows, proposed in [FSJ01, SRF05], is to use artificial forces such as *vorticity confinement* that seek to restore vorticity that has been lost due to excessive viscosity. These approaches have produced some of the most visually complex fluid simulations to date. However, while vorticity confinement reinjects vorticity (and kinetic energy) that is lost due to dissipation, it cannot fully compensate for excess numerical dissipation. For example, in a (non-simulated) low viscosity flow, inertia is well-conserved. This can be visible as coherent flow features that traverse the flow for a significant distance, without slowing down or breaking up. By using low viscosity methods, we are able to capture these features, as shown in Figures 7 and 3 and seen on the accompanying video.

Figure 3: *A "daemon death" from The Golden Compass. Low viscosity allows for correct conservation of momentum which is visible as slow sustained but turbulent flow around obstacles. Our method properly handles complex geometries such as the fallen soldier's body and open boundaries at the edge of the finite simulation domain.*

3. Algorithms

Algorithm 1 Fluid Solver Overview

1: $\frac{\partial \mathbf{u}^n}{\partial t} \leftarrow -\frac{\partial (u\mathbf{u})^n}{\partial x} - \frac{\partial (v\mathbf{u})^n}{\partial y} - \frac{\partial (w\mathbf{u})^n}{\partial z}$ (using QUICK)
2: $\frac{\partial \mathbf{u}^n}{\partial t} \leftarrow \frac{\partial \mathbf{u}^n}{\partial t}$ +[any other forces or terms]
3: Calculate $\mathbf{u}^{n+1}$ using Equation 6
4: Perform accelerated IOP until required accuracy.

The order of operations in our fluid solver is listed in Algorithm 1. We begin by describing the QUICK scheme in Section 3.1 followed by IOP in Section 3.2.

3.1. QUICK, a low dissipation advection scheme

To reduce numerical dissipation we use an upwind weighted advection scheme (QUICK) that was introduced by [Leo79]. The QUICK scheme approximates the advective fluxes of the cell boundaries with 3rd order accuracy. The numerical dissipation of the QUICK scheme is highly scale selective, providing damping only at the highest wave numbers that are resolved, and thus automatically decreases numerical viscosity at higher grid resolutions. Therefore, unlike fully non-dissipative schemes such as 2nd order central differencing, QUICK needs no additional viscosity for numerical stability.

As in [IGLF06], we use the flux form of the advection equation because it discretely conserves mass and momentum. The advection equation for momentum for a non-divergent flow can be written in conservative form as:

$$\frac{\partial \mathbf{u}}{\partial t} + \frac{\partial (u\mathbf{u})}{\partial x} + \frac{\partial (v\mathbf{u})}{\partial y} + \frac{\partial (w\mathbf{u})}{\partial z} = 0 \qquad (1)$$

(a) Semi-Lagrangian advection (b) QUICK advection (c) QUICK advection at higher resolution

Figure 2: *Smoke stirred by a paddle. Panel (a) is produced using semi-Lagrangian advection on a* $64 \times 32 \times 32$ *grid. The large amount of numerical viscosity suppresses the naturally occurring unstable motions in the flow, leading to a smoke field that is highly filamented but without smaller scales in the flow. Panels (b) and (c) are produced using QUICK advection on grids of the same and higher resolution (*$128 \times 64 \times 64$*) respectively. Especially in panel (c), natural instabilities are no longer suppressed and the flow field exhibits a naturally turbulent field containing many scales.*

Where $\mathbf{u}$ is the velocity vector with components (u,v,w) in the x, y and z directions respectively.

Assume an equidistant Cartesian grid labeled with (i,j,k) with node spacing $(\Delta x, \Delta y, \Delta z)$ in x, y, and z. Using a staggered velocity representation, the advective term can be discretized as the finite difference of the fluxes at the cell faces. The fluxes across the cell faces are approximated with a upwind weighted 3rd-order formula. For example, the advection of v in the x-direction is discretized as

$$\frac{\partial uv}{\partial x}_{i,j+\frac{1}{2},k} \approx \frac{(uv)_{i+\frac{1}{2},j+\frac{1}{2},k} - (uv)_{i-\frac{1}{2},j+\frac{1}{2},k}}{\Delta x} \tag{2}$$

Here, $i \pm \frac{1}{2}$ refers to the cell face in the positive/negative x-direction. Since the velocity field is staggered, u is already available at the cell faces in the x-direction but we have to interpolate these values to obtain a estimates at the staggered y-positions,

$$u_{i+\frac{1}{2},j+\frac{1}{2},k} = 0.5\left(u_{i+\frac{1}{2},j,k} + u_{i+\frac{1}{2},j+1,k}\right). \tag{3}$$

The value of $v_{i+\frac{1}{2},j+\frac{1}{2},k}$ is discretized using an upwind weighted, 3rd order approximation given by

$$v_{i+\frac{1}{2},j+\frac{1}{2},k} = \frac{1}{8}v_{i-1,j+\frac{1}{2},k} + \frac{2}{8}v_{i,j+\frac{1}{2},k} + \frac{5}{8}v_{i+1,j+\frac{1}{2},k} \tag{4}$$

if $u_{i+\frac{1}{2},j+\frac{1}{2},k} > 0$ and

$$v_{i+\frac{1}{2},j+\frac{1}{2},k} = \frac{5}{8}v_{i,j+\frac{1}{2},k} + \frac{2}{8}v_{i+1,j+\frac{1}{2},k} + \frac{1}{8}v_{i+2,j+\frac{1}{2},k} \tag{5}$$

if $u_{i+\frac{1}{2},j+\frac{1}{2},k} < 0$. We refer the reader to [Leo79] for further details and analysis.

The commonly used first-order Euler forward time stepping scheme is unstable when used with a minimally dissipative scheme such as QUICK [CHQZ88]. Therefore, it is necessary to use a time discretization that has a less restrictive region of stability. We have chosen to combine the QUICK advection scheme with a 2nd order Adams-Bashforth (AB2) time stepping scheme [FP96]. For a non-constant time step size Δt that changes sufficiently smoothly in time, the AB2 scheme is:

$$\mathbf{u}^{n+1} = \mathbf{u}^n + \frac{\Delta t^n}{\Delta t^{n-1}}\left[\left(\frac{1}{2}\Delta t^n + \Delta t^{n-1}\right)\frac{\partial \mathbf{u}}{\partial t}^n - \frac{1}{2}\Delta t^n \frac{\partial \mathbf{u}}{\partial t}^{n-1}\right] \tag{6}$$

For the first time step, AB2 is "primed" with a single Euler step. In order to ensure that Δt is sufficiently smooth, we always subdivide the remaining frame time into even parts so that we are not left with one very short "catch-up" step to synchronize our time stepping with 1/24 second frames required for film.

Typically the time step for semi-Lagrangian schemes is limited to no more than 5 times the maximum time step allowed by the CFL condition. By comparison, QUICK advection is stable only when the time step is below the CFL condition. In practice, therefore, the difference in computational cost between a semi-Lagrangian method and QUICK is a constant number which does not depend on grid resolution. In exchange for this higher constant, one obtains dramatically less unnecessary artificial dissipation. QUICK is slightly more expensive than first-order upwind. However, because QUICK uses a regular fixed memory access pattern, it is actually faster per time step than semi-Lagrangian schemes that may have irregular random memory accesses.

Similar discretizations apply to advected scalar fields that are defined at cell centers such as temperature or density. For temperature, the flux formulation guarantees that total heat content is exactly preserved. The smoke fields visible in Figures 1 and 7 were deformed using QUICK.

3.2. Iterated Orthogonal Projections

3.2.1. Multigrid Poisson solvers

Typically, the computational bottleneck of an incompressible fluid solver is the calculation of pressure at every time step, which requires solving a Poisson equation. The most widely used method in the graphics community to solve the Poisson equation is the preconditioned conjugate gradient (PCG) method. There are many advantages to the PCG method. It is a robust method that is relatively easy to implement and can handle complex domains shapes. Its major disadvantage is that at higher resolutions the condition number of the Poisson matrix becomes increasingly poor, and the computational cost to obtain a specified accuracy in the solution of the Poisson equation (and corresponding accuracy in the enforcement of the incompressibility condition) becomes larger. The computational cost of the PCG method with an incomplete Cholesky (IC) preconditioner [GvL96] scales as $O(N^{1.33})$ where N is the total number of nodes in the grid.

In contrast, a multigrid-based Poisson solver scales multigrid methods will be faster than any PCG implementation for large enough grid sizes. The first column of Table 1 shows the speed of obtaining a Poisson solution for a fluid simulation run on a single $2.2mHz$ dual Opteron workstation using the PCG method with an IC preconditioner, while the second column shows the speed for a multigrid solver based on [Yav96]. A multigrid solver at 256^3 resolution is entirely affordable on a single CPU, whereas a PCG solver quickly becomes unrealistically expensive for practical purposes. Except where noted, all pressure solvers in this paper have been run in double precision until the maximum divergence is below 10^{-8}. Multigrid is especially amenable

	PCG		Multigrid	
Grid	time	time/N	time	time/ N
32^3	0.47 s	1.4e-5 s	0.06 s	1.8e-6 s
64^3	8.63 s	3.3e-5 s	0.50 s	1.9e-6 s
128^3	137.9 s	6.6e-5 s	3.61 s	1.7e-6 s
256^3	N/A	N/A	30.3 s	1.8e-6 s

Table 1: *Average computation time for the projection to a non-divergent flow for a fluid in a simple domain for PCG and multigrid methods on 2.2 mHz dual Opteron. Convergence is to less than 10^{-8} divergence per grid cell.*

to GPU implementation, as demonstrated by [BFGS03] and [GWL*03]. We have implemented [Yav96] using the CUDA language [NVI07]. As Table 2 shows, multigrid is extremely efficient on a GPU, almost 55 times faster than the CPU at the highest resolution. The left column shows timings for an optimized PCG solver running on an NVIDIA 8800GTX. The right column shows the same solution obtained with a multigrid solver. Note that at 128^3 and 256^3 resolutions, the PCG version does not converge due to the loss of stability of the Poisson matrix as a result of the GPU's limited floating point precision. Multigrid, however, is more robust to numerical precision issues and converges in all cases.

	PCG		Multigrid	
Grid	time	time/N	time	time/ N
32^3	0.05 s	1.5e-6 s	0.04 s	1.1e-6 s
64^3	0.46 s	1.7e-6 s	0.06 s	2.2e-7 s
128^3	N/A	N/A	0.13 s	6.0e-8 s
256^3	N/A	N/A	0.56 s	3.3e-8 s

Table 2: *Average computation time for the projection to a non-divergent flow for a fluid in a simple domain for PCG and multigrid methods on an NVIDIA 8800GTX. Because of the limited floating point precision, convergence is to less than 10^{-5} divergence per grid cell.*

The multigrid solver of [Yav96] has excellent convergence properties and is straightforward to implement, but it can only deal with simple domains that do not contain internal obstacles or other complex geometries. There are alternatives to a standard multigrid method such as black-box [Den82], algebraic [Bra86] or geometrical [WL04] multigrid methods. All three methods are viable alternatives that for large enough grids will be superior to any PCG method. Unfortunately, these methods are not easy to implement and in certain situations the ideal, linear scaling of computational cost with number of unknowns may be lost [Den82, WL04]. Our goal with IOP is to create a fast Poisson solver than can handle complex boundary conditions and is easy to implement and tune.

By limiting the number of iterations in PCG to a fixed amount, PCG can also be made to scale as $O(N)$. However, without sufficient PCG iterations the resulting velocity field will have significant divergence, which leads to visibly noticeable artifacts such as clumped marker particles. FFT-based methods, which scale as $O(N \log N)$ can also be used to solve the Poisson equation. However, they are difficult to implement for a variety of boundary conditions and can only be used for simple problem geometries without complex domain boundaries.

3.2.2. Boundary conditions as orthogonal projections

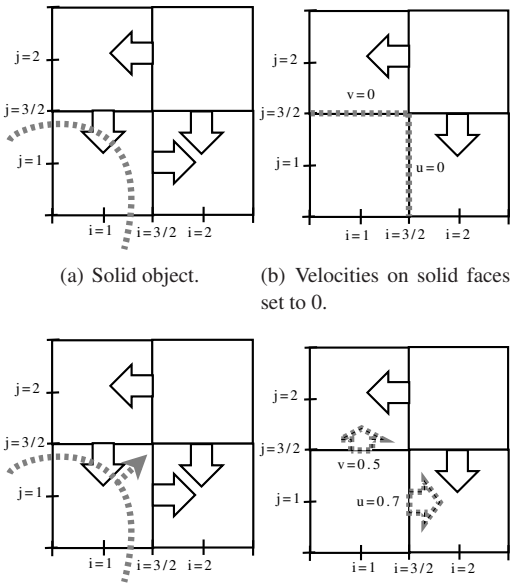
As mentioned in the introduction, removing the divergent part of the flow can be viewed as an orthogonal projection [Cho68]. An orthogonal projection can be written as a matrix-vector multiplication,

$$\mathbf{x}_{ndiv} = P_{ndiv}\mathbf{x} \tag{7}$$

where $\mathbf{x}$ is a discrete velocity field and P_{ndiv} is the projection matrix that removes the divergent part of $\mathbf{x}$. Eigenvalues of the matrix P_{ndiv} will be either 0 or 1, with Eigenvector corresponding to the null space and range of P_{ndiv} respectively.

The range of P_{ndiv} is exactly the space of all discrete vector fields with zero divergence, which we refer to as $\mathbf{X}_{ndiv}$. We do not actually implement the projection via a matrix multiplication as in Equation 7, but rather by solving a Poisson equation. However, for analysis purposes it is helpful to think of the process of solving for pressure and subtracting the pressure gradient as applying the linear operator P_{ndiv} to a vector.

By modifying the right-hand side of the Poisson equation as in [FOA03], we could just as easily project to a certain fixed divergence. This still corresponds to an orthogonal projection, except that the space $\mathbf{X}_{ndiv}$ is now affine rather than linear. Geometrically, a linear projection can be thought of as projecting a point onto a plane that contains the origin, while an affine projection projects onto a plane that might not.



(a) Solid object.

(b) Velocities on solid faces set to 0.

(c) Object moving with velocity $(0.7, 0.5)$.

(d) u and v set to x and y components of object's velocity.

Figure 4: *The velocity field in (a) is updated in (b) by setting the faces of the staggered grid to 0, corresponding to a discrete approximation to a solid object in cell (1,1) that is not moving. In (c), the object is moving with velocity* $(0.5, 0.7)$*, and in (d) the u an v velocities on the faces are set correspondingly. Closed or inflow/outflow conditions are identical, with the values set on the exterior faces of the domain. These procedures can all be viewed as trivial orthogonal projections.*

The enforcement of many types of velocity boundary conditions also can be viewed as an orthogonal projection. For example, Figure 4 shows how solid object boundary conditions may be enforced trivially by simply setting the corresponding u, v, and w values at the faces near a solid object equal to the x,y, and z components of that object's velocity. This has the effect, for example, of enforcing that the fluid's normal velocity is zero at the boundary of a non-moving object. We implement this procedure in the trivial way, by directly setting the grid values as shown in Figure 4(b).

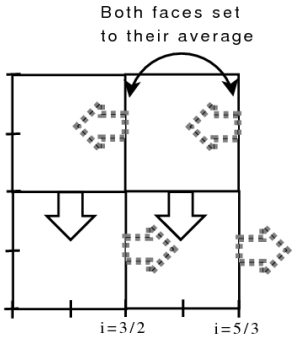


Figure 5: *To enforce open boundaries on the positive x face of the domain, we set u values on the* $i = 3/2$ *and* $i = 5/3$ *faces to the average of the two values. This is an orthogonal projection*

Setting these boundary conditions can be thought of as corresponding to orthogonal projections over the space of discrete vector fields. For instance, the requirement that the normal velocity is zero at a particular face can be written as a matrix multiplication with a diagonal unity matrix that has zeros on the diagonal at those positions where a boundary velocity is enforced. As with the zero-divergence criteria, the space of discrete vector fields that satisfy these boundary conditions is a linear subspace. If the enforced velocity is non-zero as in Figure 4(d), the projection is onto an affine rather than a linear subspace, but the situation is otherwise identical.

If we think of this "boundary condition enforcement" procedure as a linear operator, we can write:

$$\mathbf{x}_{bound} = P_{bound}\mathbf{x} \tag{8}$$

where P_{bound} represents the orthogonal projection of a general discrete vector field into the (linear or affine) subspace of discrete velocity fields that satisfy the boundary conditions. We stress that although we have written this process as a matrix-vector multiplication, it is implemented by simply setting the grid values as desired.

The orthogonal projection viewpoint is a quite general: for example, Figure 5 shows how we enforce continuative (open) boundary conditions at the edge of the simulation domain. In fact, any linear or affine constraint on the vector field can be satisfied via an orthogonal projection. The practical calculation of any given projection may not be as simple as enforcing open, closed, or moving object conditions, but the analysis in this section still applies.

Let $\mathbf{X}_{int} = \mathbf{X}_{ndiv} \cap \mathbf{X}_{bound}$ be the intersection of the ranges of both projection matrices. $\mathbf{X}_{int}$ is the subspace of vector fields that satisfy both the non-divergence condition *and* all specified boundary conditions simultaneously. If $\mathbf{X}_{bound}$ and $\mathbf{X}_{ndiv}$ are vector spaces, $\mathbf{X}_{int}$ will always be non-empty. If one or more of the spaces are affine, it is required that $\mathbf{X}_{int} \neq \emptyset$, or else there is no way to simultaneously satisfy all constraints. P_{int} is the abstract projection operator that projects orthogonally onto $\mathbf{X}_{int}$. Enforcing both the boundary conditions and the non-divergence condition is equivalent to applying P_{int}. In the PCG approach, for example, boundary conditions are enforced directly, and then the Poisson equation is modified such that subtracting the pressure gradient maintains these boundary conditions. This approach is equivalent to evaluating $P_{int}\mathbf{x}$ in a single step.

The following is a property of projection operators that is generally true.

$$P_{int} = \lim_{n\to\infty} (P_{ndiv}P_{bounds})^n . \tag{9}$$

To prove this, we introduce the product matrix of both projections P_{iter}:

$$P_{iter} = P_{ndiv}P_{bound} \tag{10}$$

Since the Eigenvalues of P_{ndiv} and P_{bound} are either 0 or 1, the Eigenvalues of P_{iter} will be between 0 and 1 inclusive. This property guarantees convergence of an iteration that consists of a simple repeated multiplication with P_{iter}. Eigenvalues of P_{iter} will be small when the subspaces of P_{ndiv} and P_{bound} are nearly orthogonal and will be closer to 1 when the individual subspaces are more parallel. The only Eigenvectors not killed by repeated application of P_{iter} will be those for which the corresponding Eigenvalues in P_{ndiv} and P_{bc} are both 1. Therefore iterated projections by P_{iter} is guaranteed to converge towards P_{int} thus proving Equation 9, with the rate of convergence determined by the Eigenvalues of P_{iter}.

Equation 9 translates to an algorithm for applying P_{int} without explicitly constructing it: iteratively apply P_{bounds} and P_{ndiv} by successively directly enforcing the boundary conditions, then using a fast multigrid solver [Yav96] to project out the divergent part. Because the multigrid solver does not respect the boundary conditions, they will no longer be properly enforced at the end of an iteration. However, the error in boundary conditions is reduced after each iteration, and can be driven as low as desired by repeated IOP iterations. In other words, the IOP framework independently satisfies both constraints, and converges to $\mathbf{x}_{int}$ in an iterative manner.

Within each IOP iteration, we choose to first enforce boundary conditions then enforce zero divergence. This order leads to a solution that is fully non-divergent, but may allow some errors in the boundary conditions. Similarly to limiting the number of PCG iterations to a small number regardless of convergence, iterating IOP a fixed number of time guarantees $O(N)$ scaling in the number of unknowns. Unlike unconverged PCG, however, unconverged IOP will *only* have error in enforcement of the boundary conditions, not in the enforcement of non-divergence.

The convergence rate of IOP will be slow if the matrix P_{iter} has Eigenvalues close to 1. To accelerate convergence in cases that need a more accurate solution, we have modified

Iterations	$max(\varepsilon_{bound})$	$max(\nabla \cdot \mathbf{u})$
1	0.13e-1	1.0e-8
3	0.17e-3	1.0e-8
5	0.21e-5	1.0e-8

Table 3: *Maximum error in enforcement of the boundary conditions and non-divergence on the moving object for the paddle test using semi-Lagrangian advection on a* $64 \times 32 \times 32$ *grid for different number of iterations at frame 40.*

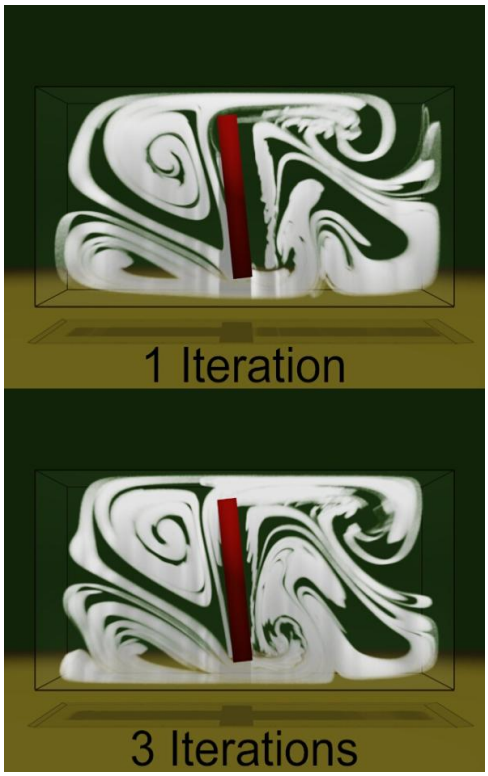


Figure 6: *Smoke stirred by a paddle using semi-Lagrangian advection on a* $64 \times 32 \times 32$ *grid. Upper panel: 1 IOP iteration, lower panel: 3 IOP iterations. Although additional iterations of IOP lead to more accurate enforcement of boundary conditions, the flow remains qualitatively identical. Semi-Lagrangian advection was used because it generates less turbulence and hence the errors in boundary condition enforcement are more visible.*

the basic iteration cycle in the spirit of a Krylov subspace method [GvL96], making more optimal use of each direction by which the solution vector is changed during a single iteration. The accelerated form of IOP is shown in Algorithm 2.

Table 3 shows the maximum of the error in the enforcement of the boundary conditions the paddle simulation in Figure 6. While rapid convergence cannot be guaranteed for all situations, we have found that in practice, a limited number of iterations suffices for a satisfactory visual result. We believe this is because small errors in the boundary conditions being met will still produce the boundary layers and accompanying velocity shears that are characteristic features of fluid flow. In fact, for all other images in this paper and on the accompanying video, only a single IOP iteration is used. A surprising result is that a single IOP iteration is often enough for visually demanding applications such as feature film visual effects. This is the key benefit of IOP – by sacrificing some error in meeting boundary conditions, we can obtain a significant overall speedup, often without a noticeable impact on visual quality.

In our experience, the multigrid method of [Yav96] and directly enforcing boundary conditions are so fast that for a small number of iterations (e.g. five or fewer), IOP is actually faster than simultaneously satisfying all constraints in a single step either with PCG or a black-box multi grid solver based on [Den82]. We expect this to be even more true for a GPU implementation, although to date we have only verified this claim on a CPU. We note an additional disadvantage of PCG (although not black-box multigrid) is that certain types of boundary conditions (e.g. open boundaries) result in a Poisson matrix that is asymmetric and hence more difficult to solve using iterative method like PCG.

Algorithm 2 Accelerated IOP

1: it = 1
2: **while** desired convergence not achieved achieved **do**
3: $\mathbf{u}^{it} \leftarrow P_{bound}\mathbf{u}^{it-1}$ (Directly set velocities to enforce boundary conditions)
4: $\mathbf{u}^{it} \leftarrow P_{ndiv}\mathbf{u}^{it}$ (Project onto non-divergent flow)
5: $\varepsilon^{it} = error(\mathbf{u}_{it})$ (L_∞ error in boundary conditions)
6: **if** $it > 1$ **then**
7: $\mathbf{u}_{dif} = \mathbf{u}^{it} - \mathbf{u}^{it-1}$ (correction vector)
8: $\alpha = \varepsilon^{it} / \varepsilon^{it-1}$
9: $\mathbf{u}_{tmp} = \mathbf{u}^{it} + \alpha\, \mathbf{u}_{dif}$
10: $\varepsilon_{tmp} = error(\mathbf{u}_{tmp})$
11: **if** $\varepsilon_{tmp} < \varepsilon_{it}$ **then**
12: $\mathbf{u}^{it} = \mathbf{u}_{tmp}$
13: **end if**
14: **end if**
15: it ++
16: **end while**

Figure 7: *Blocks falling over produce an area of propagating turbulence. Minimal viscosity allows for sustained propagation and the emergence of small scale motion.*

4. Results

Figure 1 shows a warm plume rising around a sphere. Simulations are shown for a constant heat source that is increasing in temperature from left to right with a thermodynamics model similar to [FSJ01] (Equations 6-8 with $\alpha = 0$). The flow is visualized using a passive scalar density field that is rendered as smoke. The density field is advected by the flow and is dissipated back to a zero over a set time scale. Larger values for the heat source at the bottom provide more potential energy to drive the flow, producing plumes that are increasingly turbulent. For larger heat sources the flow around and above the sphere becomes naturally chaotic and the flow visually contains many spatial scales. The simulations of the plume were computed on a $192 \times 128 \times 192$ numerical grid running on a 2.2 mHz dual Opteron workstation, with a computational cost of between 15 s and 200 s per frame, depending on the number of substeps required by the CFL condition. We expect this would be faster with our the GPU-based Poisson solver, although we have not integrated it into our full Navier Stokes solver yet.

Figure 2 depicts a moving paddle mixing a smoke field in a rectangular domain. This figure is included to demonstrate the differences that arise from different levels of numerical viscosity that are present during a numerical simulation. The simulations are rendered using massless marker particles that are advected by the flow. Figure 2(a) shows semi-Lagrangian advection at a grid resolution of $64 \times 32 \times 32$. There is a large amount of folding and stretching leading to thin filaments in the smoke field, but the high amount of numerical viscosity suppresses the naturally occurring unstable motions. Figure 2(b) shows the same set-up simulated using the QUICK advection scheme. The flow is much

more energetic and exhibits many additional small scale features. These small scales arise from natural instabilities driven by gradients in the flow field (shear instabilities) and are no longer fully suppressed by artificial viscosity. Figure 2(c) shows the additional effect of a finer grid resolution ($128 \times 64 \times 64$), reducing the amount of numerical viscosity even further because of QUICK's scale selective damping. A natural turbulent cascade towards small scales is now possible. This allows for efficient mixing of smoke, completely destroying the physically incorrect stretched filaments that were evident in Panel (a).

To quantify the effect of different advection schemes and their different levels of numerical dissipation we computed kinetic energy spectra for simulations using several different advection schemes. Figure 8 shows spectra of kinetic energy for the paddle simulation at a resolution of $64 \times 32 \times 32$ at frame 40. A spectrum of kinetic energy represents the energy that is present at any particular wave number, with higher wave numbers representing smaller spatial scales. The spectra are obtained by performing a 3 dimensional Fourier transform of the velocity field. Energy is integrated over the 3 wave number components to obtain an energy spectrum for absolute wave number $k = \sqrt{k_x^2 + k_y^2 + k_z^2}$.

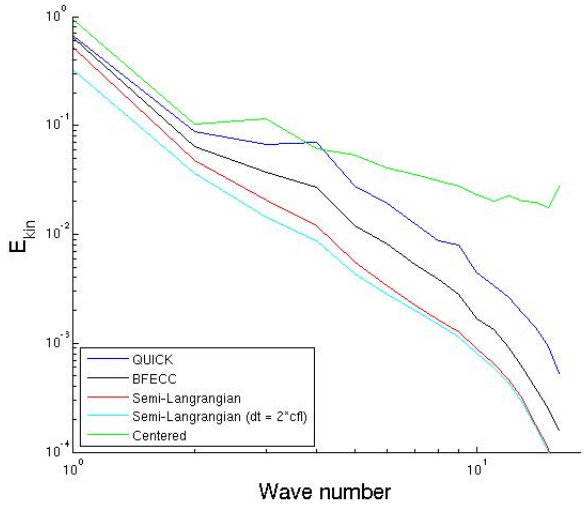


Figure 8: *Spectra of kinetic energy for the paddle simulation on a $64 \times 32 \times 32$ grid using different advection algorithms: QUICK, BFECC, semi-Lagrangian with the same time step size as the other methods, semi-Lagrangian using a time-step that is twice as large, and non-dissipative 2nd-order centered differencing. Centered differencing suffers from an accumulation of energy at the highest resolved wave number resulting in numerical instability. Of all other methods, the QUICK algorithm retains most of the kinetic energy at all wave numbers while maintaining numerical stability.*

Inspection of Figure 8 shows that centered differencing, a purely non-dissipative scheme, retains the largest amount of kinetic energy. However, note the pile-up of energy at the highest wave numbers (smallest resolved scales). This transfer of energy is the effect of triad interactions and will lead to numerical instability. Of the other algorithms investigated, the semi-Lagrangian scheme using the larger time step size is the most dissipative, leading to the least amount of kinetic energy at all scales. The difference in kinetic energy with the semi-Lagrangian scheme using a smaller time step is most pronounced at larger scales. The BFECC scheme [KLLR05] does much better, leading to an approximately three-fold increase in energy for the higher wave numbers. QUICK performs best, as the kinetic energy spectrum falls off more gradually towards higher wave numbers, producing kinetic energy levels at higher wave number that are an order of magnitude larger then those obtained using semi-Lagrangian advection. The difference in spectral slope is a result of QUICK's scale selective numerical dissipation. In effect, dissipation is only significant at the highest wave numbers where it is a necessity for numerical stability.

Figure 3 shows an effect from the feature film "The Golden Compass." This simulation involves a creature transforming into golden dust and flowing around a dead soldier. It is a highly turbulent simulation with complex geometry, closed boundaries at the bottom of the domain, and open boundaries on the sides. Even though this simulation had to meet the visually demanding requirements of feature film, a single IOP iteration was enough.

Figure 7 shows a set of blocks falling onto a flat surface. The flow that is generated by the blocks' motion produces an area of rolling turbulence that propagates away from the blocks. The near absence of viscosity allows the prolonged propagation of the turbulent field as evidenced by the layer of smoke that is being mixed up in the clear air above. This simulation was computed on a $128 \times 64 \times 64$ numerical grid. The horizontal extent of the simulated domain is indicated by the extent of the layer of smoke. Open boundary conditions are used at the sides of the computational domain.

5. Conclusion

As we have demonstrated, the combination of QUICK with IOP is a robust approach for simulating highly turbulent fluid flows. We have chosen these methods because when used together, they accurately model the energy cascade present in real turbulence which is important for visual modeling of many phenomena. In fact, these methods are so turbulent that they rapidly lead to chaotic motion, which can present a challenge for artistic control. On the other hand, they are efficient, robust, and have few parameters to tune, which allows an artist to achieve a desired effect by rapidly trying several different versions of a simulation.

Acknowlegdments

Many thanks go to Markus Kurtz and Jerry Tessendorf for their help in producing the graphical results for this paper.

References

[BFGS03] BOLZ J., FARMER I., GRINSPUN E., SCHRODER P.: Sparse matrix solver on the GPU: Conjugate gradients and multigrid. In *Proceedings of ACM SIGGRAPH 2003* (2003), vol. 22 of *ACM Transaction on Graphics*, pp. 917–924.

[Bra86] BRANDT A.: Algebraic multicgrid theory: The symmetric case. *Appl. Math. Comput. 19* (1986), 23–56.

[Cho68] CHORIN A. J.: Numerical solution of the Navier-Stokes equations. *Math. Comput. 1968 22* (1968), 745–762.

[CHQZ88] CANUTO C., HUSSAINI M. Y., QUARTERONI A., ZANG T. A.: *Spectral Methods in Fluid Dynamics*. Springer-Verlag, New York, 1988.

[Den82] DENDY J. E.: Black box multigrid. *J Comp Phys. 48* (1982), 366–396.

[EMF02] ENRIGHT D. P., MARSCHNER S. R., FEDKIW R. P.: Animation and rendering of complex water surfaces. *ACM Transactions on Graphics 21*, 3 (July 2002), 736–744.

[ETK*07] ELCOTT S., TONG Y., KANSO E., SCHRÖDER P., DESBRUN M.: Stable, circulation-preserving, simplicial fluids. *ACM Transactions on Graphics 26*, 4 (2007).

[FM97] FOSTER N., METAXAS D.: Controlling fluid animation. In *Proceedings Graphics Interfaces 1997* (1997), pp. 178–188.

[FOA03] FELDMAN B. E., O'BRIEN J. F., ARIKAN O.: Animating suspended particle explosions. *ACM Transactions on Graphics 22*, 3 (July 2003), 708–715.

[FP96] FERZIGER J. H., PERIC M.: *Computational Methods for Fluid Dynamics*. Springer-Verlag, New York, 1996.

[FSJ01] FEDKIW R., STAM J., JENSEN H. W.: Visual simulation of smoke. In *SIGGRAPH 2001 Conference Proceedings* (August 2001), pp. 15–22.

[GvL96] GOLUB G., VAN LOAN C.: *Matrix Computations*. John Hopkins Universtiy Press, 1996.

[GWL*03] GOODNIGHT N., WOOLLEY C., LEWIN G., LUEBKE D., HUMPHREYS G.: A multigrid solver for boundary value problems using programmable graphics hardware. *Graphics Hardware* (2003), 102–135.

[IGLF06] IRVING G., GUENDELMAN E., LOSASSO F., FEDKIW R.: Efficient simulation of large bodies of water by coupling two and three dimensional techniques. *ACM Transactions on Graphics 25*, 3 (July 2006), 805–811.

[KFCO06] KLINGNER B. M., FELDMAN B. E., CHENTANEZ N., O'BRIEN J. F.: Fluid animation with dynamic meshes. *ACM Transactions on Graphics 25*, 3 (July 2006), 820–825.

[KLLR05] KIM B., LIU Y., LLAMAS I., ROSSIGNAC J.: Flowfixer: Using bfecc for fluid simulation. In *Eurographics Workshop on Natural Phenomena* (2005).

[Leo79] LEONARD B. P.: A stable and accurate convective modelling procedure based on quadratic upstream interpolation. *Comput. Meth. Appl. Mech. Eng. 19* (1979), 59–98.

[LGF04] LOSASSO F., GIBOU F., FEDKIW R.: Simulating water and smoke with an octree data structure. *ACM Transactions on Graphics 23*, 3 (Aug. 2004), 457–462.

[NFJ02] NGUYEN D. Q., FEDKIW R., JENSEN H. W.: Physically based modeling and animation of fire. In *SIGGRAPH 2002 Conference Proceedings* (August 2002), pp. 721–728.

[NVI07] NVIDIA: *NVIDIA CUDA Compute Unified Device Architecture*, 1.1 ed., 2007.

[SFK*07] SELLE A., FEDKIW R., KIM B., LIU Y., ROSSIGNAC J.: An unconditionally stable MacCormack method. *Journal of Scientific Computing* (2007).

[SM98] SHCHEPETKIN A., MCWILLIAMS J.: Quasi-monotone advection schemes based on explicit locally adaptive dissipation. *Monthly Weather Review 126* (1998), 1541–1580.

[SRF05] SELLE A., RASMUSSEN N., FEDKIW R.: A vortex particle method for smoke, water and explosions. *ACM Transactions on Graphics 24*, 3 (Aug. 2005), 910–914.

[Sta99] STAM J.: Stable fluids. In *Proceedings of SIGGRAPH 99* (Aug. 1999), Computer Graphics Proceedings, Annual Conference Series, pp. 121–128.

[Str99] STRAIN J.: Semi-lagrangian methods for level set equations. *J. Comp. Phys.*, 151 (1999), 498–533.

[TL76] TENNEKES H., LUMLEY J.: *A first course in Turbulence*. MIT press, 1976.

[WHLP94] WIJK J. J. V., HIN A. J. S., LEEUW W. C. D., POST F. H.: Three ways to show 3d fluid flow. *IEEE Comput. Graph. Appl. 14*, 5 (1994), 33–39.

[WL04] WAN J. W., LIU X.-D.: A boundary condition-capturing multigrid approach to irregular boundary problems. *SIAM J. Sci. Comput. 25* (2004), 1982–2003.

[Yav96] YAVNEH I.: On red-black SOR smoothing in multigrid. *SIAM J. Sci. Comput. 17*, 1 (1996), 180–192.

[ZB05] ZHU Y., BRIDSON R.: Animating sand as a fluid. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers* (New York, NY, USA, 2005), ACM Press, pp. 965–972.