Tackling Small-Scale Fluid Features in Large Domains for Computer Graphics

by

Xinxin Zhang

B. Information and Computation Science, Zhejiang University City College, 2009

M. Computer Science, New York University, 2012

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Doctor of Philosophy

in

THE FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES

(Computer Science)

The University of British Columbia

(Vancouver)

April 2017

Abstract

Turbulent gaseous phenomena, often visually characterized by their swirling nature, are mostly dominated by the evolution of vorticity. Small scale vortex structures are essential to the look of smoke, fire, and related effects, whether produced by vortex interactions, jumps in density across interfaces (baroclinity), or viscous boundary layers. Classic Eulerian fluid solvers do not cost-effectively capture these small-scale features in large domains. Lagrangian vortex methods show great promise from turbulence modelling, but face significant challenges in handling boundary conditions, making them less attractive for computer graphics applications. This thesis proposes several novel solutions for the efficient simulation of small scale vortex features both in Eulerian and Lagrangian frameworks, extending robust Eulerian simulations with new algorithms inspired by Lagrangian vortex methods.

Preface

This thesis incorporates three published papers as separate chapters:

- Chapter 2: X. Zhang, M. Li, and R. Bridson. 2016. *Resolving fluid boundary layers with particle strength exchange and weak adaptivity*. ACM Trans. Graph. 35,4(Proc. SIGGRAPH), Article 76.
- Chapter 3: X. Zhang, R. Bridson, and C. Greif. 2015. *Restoring the missing vorticity in advection-projection fluid solvers*, ACM Trans. Graph. 34, 4 (Proc. SIGGRAPH), Article 52.
- Chapter 4: X. Zhang and R. Bridson. 2014. *A PPPM fast summation method for fluids and beyond*. ACM Trans. Graph. 33, 6 (Proc. SIGGRAPH Asia), Article 206.

Apart from the usual supervision roles by Dr. Robert Bridson and Dr. Chen Greif, the single-scattering particle renderer used for images and animations was provided by Dr. Bridson, and help from Minchen Li in testing simulation examples for Chapter 2, all work is mine.

Table of Contents

List of Tables

List of Figures

Acknowledgments

I would like to thank my supervisors Dr. Chen Greif and Dr. Robert Bridson. They fed me with interesting questions and provided me helpful insights to my proposed solutions. I will also thank Dr. Mark Stock: his online summary of vortex methods literature has been a great resource for me when I was first exposed to this amazing numerical method. I also thank Yufeng Zhu, Essex Edwards, and Todd Keeler for being excellent group-mates, and for the discussions we've had to technical problems. I also would like to thank all the professors of the IMAGER lab at UBC, for making this lab a great place for computer graphics researches.

Thanks also go to my parents for their support; they haven't been to a college for their life and they don't know science, yet they supported me to pursue my interests. Thanks go to my friends, too: Yingsai Dong has been my close friend for many years since I started at UBC, we played sports and watched movies together, we celebrated to share happiness and back each other when short for money; he left Vancouver to work for facebook last year, and I miss him. Thanks go to my girlfriend, who hasn't been showing up in my life until recently: you are very very cute.

I would also like to thank NSERC (Natural Sciences and Engineering Research Council of Canada) for funding me to do the published research.

Chapter 1

Introduction

Be formless, shapeless, like water. — Bruce Lee.

In computer graphics, the animation of fluid such as smoke, fire and water is usually obtained by numerically advancing fluid quantities with the parameters provided by artists. To make the numerical techniques a useful tool for users with a minimal mathematical background, unconditionally stable methods [53], despite their numerical diffusion, have been widely adopted, trading off the quality of simulations.

Fluid motion is simply visually interesting: shear motions in a fluid generate eddies, these eddies merge and split, and move according to each other. These type of small-scale rotational structures, being highly noticeable in flow (see Fig.1.1), however, are often intractable to capture with the classic semi-Lagrangian fluid solvers due to numerical diffusion.

To combat the numerical diffusion problem, many methods have been proposed even just within computer graphics, such as higher order advection solvers [31, 50], momentum-conserving advection [34], and the non-diffusive Fluid Implicit Particle (FLIP) method [68].

As an alternative to these velocity-pressure schemes, researchers in and outside graphics field have been using vortex methods to track the turbulent motion of fluids. Vorticity is highly related to the swirling motion of flows; tracking the evolution of vorticity directly can often better generate and preserve detailed fluid fea-

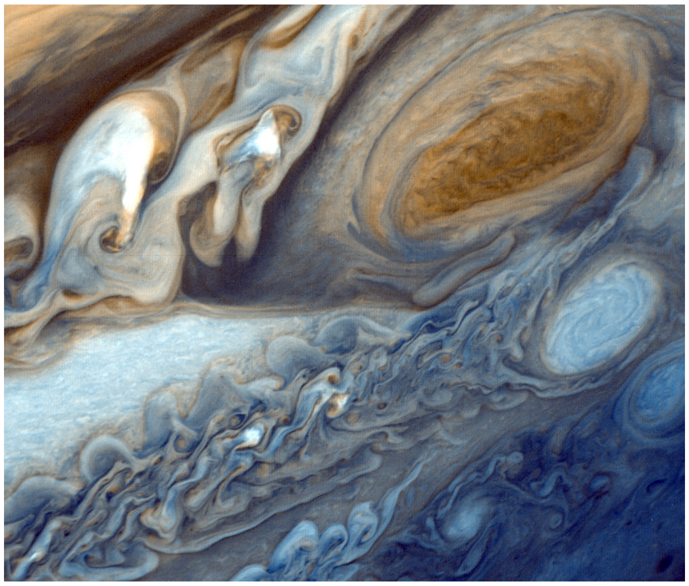


Figure 1.1: The chaotic motion of eddies produced by the storms in the atmosphere of Jupiter. Courtesy NASA/JPL-Caltech.

tures, with less computational elements than a typical computer graphics velocity-pressure fluid solver might require.

An early milestone in computer graphics used the vortex method and the Vortex-in-Cell (VIC) computation to reproduce the turbulent eddies on Jupiter [65]. Later, Park and Kim demonstrated that complex fluid motions can be simulated with only 8K vortex particles [44]. Vortex methods have also been used to study high Reynolds number flows [12, 55, 57].

However, many disadvantages of the vortex methods have also been observed. Calculating the flow velocity from vorticity requires $\mathscr{O}(N^2)$ computations for N vortex elements with a straightforward application of the Biot-Savart law. The number of vortex elements might grow exponentially during an entire simulation, making the computation intractable even when $\mathscr{O}(N)$ fast summation is used. Last but not least, boundary conditions in vortex methods require non-trivial treatment, and as a result, very limited boundary conditions have been applied in classical vortex method simulations.

We observe that the creation, evolution and interaction of vorticity elements is highly related to the turbulent, swirling feature of flow motions, but certain trade-offs have to be made to the vortex method for its efficient, flexible and robust application in computer graphics. This thesis discusses the success we have achieved in adapting the vortex method regarding the three aspects, and points the way forward to further improvements.

Chapter 2

Resolving Fluid Boundary Layers with Particle Strength Exchange and Weak Adaptivity

2.1 Introduction

In computer graphics, the incompressible Navier-Stokes equations are often used to produce realistic fluid animation. Storing fluid quantities on a Cartesian grid and performing pressure projection to handle incompressibility and boundary conditions, Eulerian approaches have proved their merit in scalably handling complex boundary shapes and maintaining stability with large time steps [7].

Despite the ease of use and implementation of Eulerian approaches, their ability to simulate high Reynolds flow remains a problem. In this type of flow, fluid motion is often strongly influenced by boundary layer dynamics. A boundary layer of vanishing thickness usually cannot be resolved at practical grid resolutions, making the no-stick or no-slip boundary conditions both diverge from physical predictability. Besides the possibility of producing inconsistent results under grid refinement, the flow motion is poorly or not at all controlled by changing the Reynolds number, as shown in Fig. 2.2: simulations at low resolution, which don't resolve the boundary layer, cannot hope to resemble the results produced by high

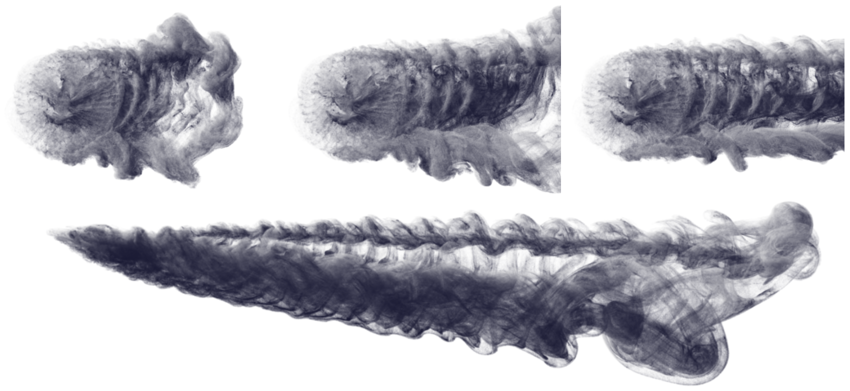


Figure 2.1: Simulation of a rotating fan in wind, $Re \approx 10^5$, visualized with smoke marker particles. Large scale motion and detailed boundary layer dynamics are cheaply and easily coupled using our method, producing high quality fluid animations at low cost. Top: top view of fan wake at frames 80, 120 and 200. Bottom: side view at frame 300.

resolution simulations.

Fluid-structure interactions in slightly viscous flow has been successfully modeled by vortex methods [9, 56]. Boundaries are viewed as generators of vorticity in vortex methods. However, solving the boundary integral equation for general geometry in 3D is non-trivial, and concerns also remain about how to reliably achieve stability with 3D vortex stretching [20, 66]: we have found it hard in practice to generally adopt vortex methods for computer graphics.

On the other hand, while the variational framework [6] with FLIP advection [68] is capable of simulating free-slip boundaries nicely, with insufficient grid resolution the momentum exchange near (more physical) no-slip boundaries is only poorly resolved.

To capture near-boundary flow accurately without incurring the same expense for the domain as a whole, adaptive methods have been proposed for graphics by many researchers, e.g. using adaptive grids [2, 37, 51] or domain decomposition [21]. While the former category has advantages in more smoothly varying the degree of refinement, there is extra overhead in mesh generation and memory indexing. On the other hand, domain decomposition solvers require extra iterations per time-step to couple the solutions between different domains.

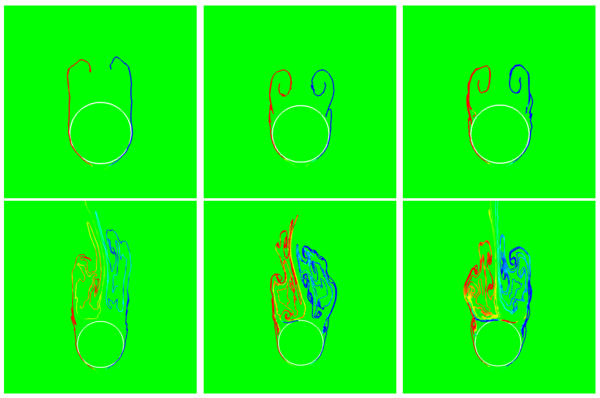


Figure 2.2: Vortex shedding from a cylinder in 2D at Re=15000, zoom-in view near the boundary. Left column: simulation results at low resolution (approximately 50×50 grid cells shown). Middle column: simulation obtained with $4\times$ resolution. Right column: simulation obtained with our method, with the same coarse grid but a $4\times$ refined grid overlaid just around the solid. Our method produces results visually consistent with the high resolution reference because only the boundary layer needs that resolving power.

We take inspiration from how, in FLIP, the particles carry flow velocities and move with the flow, while an Eulerian grid is used to adjust the particle velocities to a divergence-free state. In this paper, we extend this philosophy to arrive at an efficient, easy to implement, highly adaptive fluid solver:

- The classic FLIP scheme is augmented with a particle strength exchange (PSE) method [39] to solve the convection-diffusion part of the Navier-Stokes equations.
- By seeding extra ghost boundary particles at the boundary, our solver captures the boundary layer dynamics more accurately, producing results visually consistent with higher resolution simulations (cf. Fig. 2.2).
- Our time integration weakly couples the regionally refined solutions efficiently, alleviating the need for sophisticated global solvers.
- The regional refinements can be placed arbitrarily, even overlapping each

Table 2.1: Symbol abbreviations used throughout this paper.

ν	Kinematic viscosity of the fluid
$u_{p,i}$	Velocity stored on particle i
x_i	Position of particle i
$u_g(x)$	Velocity sampled on grid at position x
Ω	Fluid domain
$\Omega_{\text{sub},i}$	i^{th} Sub-domain with particle-mesh refinement
$W_h(x)$	Kernel used to spread a particle quantity to the grid
$H_\tau(x)$	Heat kernel for particle momentum exchange
h	Local grid cell spacing

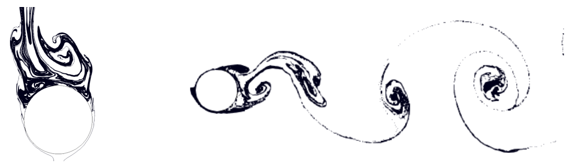


Figure 2.3: Our method captures the velocity field near the boundary efficiently and with high apparent fidelity. Left: zoom-in near the boundary flow. Right: the entire simulation, Re=15000.

other, without any geometric operations to merge them together, simplifying mesh generation for spatially adaptive solvers.

2.2 Related work

Boundary layers are often the source of turbulence in high Reynolds number flows. Pfaff et al. [45] attempted to capture this effect by seeding vortex particles from a precomputed artificial boundary layer to enhance a coarse simulation. However, their approach is limited to static boundaries and may be less physically plausible than desired as the vortex particles are seeded by chance. In contrast, our method aims to solve the viscous dynamics at the right scale, alleviating the limitation to static boundaries and improving direct physical control of the result.

When modelling fluid motion with vortex elements, vorticity can be seeded near the boundary by diffusing the boundary vortex sheet [44, 56]. In these approaches, the unknown vortex sheet strength is determined by solving boundary integral equations. Besides the expense of the GMRES solve for the vortex street strength, there are questions of solvability of the equations themselves for arbitrary topology under general motion, which is significant for computer graphics applications.

Particles, along with Eulerian velocity projections, have been widely used in computer graphics to capture fluid features which may fall between grid samples and be smoothed away by purely Eulerian solvers, e.g. FLIP [68], or derivative particles [52] and the closely related APIC [29]. We further extend this concept: in our method, particles are used to represent the change of flow momentum due to diffusion and external forces, while multiple-resolution Eulerian grids bounding different subregions are used to project particle momentum towards their divergence-free state (along with a global coarse grid).

Our scheme is related to the Iterated Orthogonal Projection (IOP) framework proposed by Molemaker et al. [41] but differs in two aspects: instead of re-projecting the field solution globally in consecutive iterations, our projection is only applied to a set of locally refined sub-domains; during each projection our pressure discretization respects the solid boundary conditions as per Batty et al. [6], while in IOP, boundary conditions are only imposed in a separate step of the iteration.

Chimera grids proposed by English et al. [15] are a promising tool for adaptive, large-scale fluid computations, but require nontrivial domain discretization to merge grids together, and a relatively expensive global solver. While the global solve may be critical for water simulations, in this paper we focus our attention on purely gaseous phenomena and demonstrate a faster and simpler weak coupling of grids is effective.

Domain decomposition schemes have been used as preconditioners for iterative linear solvers within pressure projection (e.g. [13]), or as a useful tool to decouple fluid features with different solution methods [21]. Our method also makes use of domain decomposition concepts: we solve the convection-diffusion and force calculation part on particles, and use multiple Eulerian grids as an auxiliary tool to perform velocity projection to obtain an adaptive refined and weakly coupled solution in the fluid domain.

Stock et al. [57] proposed an efficient one-way coupling to simulate rotor wake. The fluid state is first updated with a vortex particle time step, which provides the

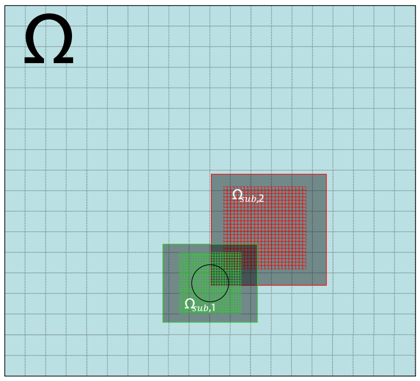


Figure 2.4: A typical domain construction used in our method. Ω indicates the global Eulerian domain where fluid motion is loosely captured. Eulerian subdomains with finer resolution can be placed anywhere to enhance the simulation quality locally, such as the green domain($\Omega_{\text{sub},1}$) for near-boundary turbulence and the red domain ($\Omega_{\text{sub},2}$) where the camera was placed. Particles are seeded to fill the space; within gray areas, particles are seeded at higher density to track finer details.

velocity boundary condition for the Eulerian sub-domain to advance itself. Vortex particles in the Eulerian sub-domain can then interpolate vorticity changes for the next time-step. In contrast, our space-filling Lagrangian particles carry fluid momentum instead of vorticity, to avoid the potential complexities associated with vortex stretching and viscous boundary conditions in general three-dimensional flows.

Multigrid solvers are becoming increasingly popular in graphics to accelerate the pressure solve (e.g. [18, 40], [3]) We likewise use multigrid for the various grids in our solver.

While Lentine et al. [33] used multi-level grids to obtain detailed fluid animations efficiently, their algorithm is limited to purely Eulerian schemes, whereas our solver is designed for hybrid particle-mesh methods like FLIP, which offer less numerical diffusion.

A common technique in industry is to inject additional detail with vorticity confinement [16, 54]) or post-process turbulence synthesis (e.g. [32, 48]). We did not use such methods, focusing on a more direct approach of better solving the

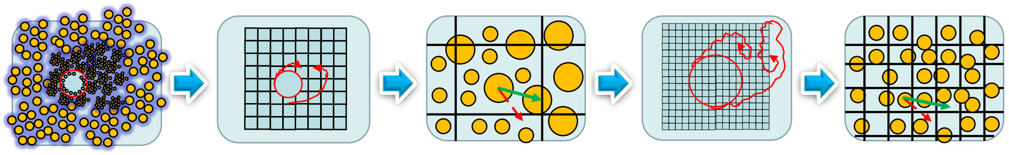


Figure 2.5: Overview of our algorithm. At the beginning of each time step, the velocity of each particle is known, the convection-diffusion part of Navier-Stokes equation is solved by a PSE method (§2.3.1). We then splat the particle velocity to the coarse grid, and make the resulting field divergence-free §2.3.2. The velocity change is interpolated to all particles for divergence correction. Then, for any subdomain $\Omega_{sub,i}$ in the field, the corrected particle momentum is splatted to the grid again and made locally divergence-free. Any particle within a subdomain $\Omega_{sub,i}$ is considered a small-scale particle and collects its momentum correction from the corresponding domain.

Navier-Stokes equations at higher Reynolds numbers, where the look can be controlled primarily with actual physical parameters. However, procedural techniques like these could be added as an additional layer to our method to produce even greater details.

2.3 VFLIP and weakly higher resolution regional projection (WHIRP).

The incompressible Navier-Stokes equations we approximately solve are:

$$\begin{aligned}\frac{\partial u}{\partial t} + (u \cdot \nabla) u &= -\frac{1}{\rho}\nabla p + \nu \Delta u + f, \\ \nabla \cdot u &= 0.\end{aligned} \tag{2.1}$$

We adopt the usual FLIP framework to handle the material derivative Du/Dt on the left hand side, storing velocity on particles which are moved in a grid-based velocity field. The pressure gradient and incompressibility condition (together with boundary conditions) are handled by a separate pressure projection step, augmented in this paper with multiple grids (see figure 2.4). We use particle strength exchange (PSE) for the viscous term, and integrate body forces f with an Euler step split from the rest of the time integration as usual.

An overview of our algorithm is given in Fig. 2.5, while each subroutine called in a time step is listed in Alg. 1. Technical details of each subroutine are given in the corresponding subsections, §2.3.1, §2.3.2,§2.3.3 and §2.3.4.

Algorithm 1 TimeStep(Δt)

```
1:  // Convection-diffusion §2.3.1
2:  For each particle $i$
3:      $u_{p,i}$ = PSE($\Delta t \nu$);
4:      $x_i$ = ForwardTrace($u_g$, $\Delta t$, $x_i$)
5:  // Hierarchical projection §2.3.2
6:  In fluid domain $\Omega$
7:      particles = Collect particles in $\Omega$
8:      DivergenceFreeUpdate(particles)
9:  For each sub-domain $\Omega_{sub,i}$ in $dx$ descending order
10:     particles = Collect particles in $\Omega_{sub,i}$
11:     DivergenceFreeUpdate(particles)
12: SeedAndDeleteParticles // §2.3.4
```

Algorithm 2 DivergenceFreeUpdate(particles)

```
1: $u_g^*$ = Splat particle velocities to grid cells
2: $u_g$ = Project($u_g^*$)
3: $\delta u = u_g - u_g^*$
4: For each particle $i$
5:     $u_{p,i} = u_{p,i} + \delta u(x_i)$
```

2.3.1 Solving the convection-diffusion equation with ghost particles

Given particles and their velocity at time step n, the convection-diffusion part of the Navier-Stokes equation,

$$\frac{Du}{Dt} = \nu \Delta u, \tag{2.2}$$

can be solved efficiently with a so-called Particle Strength Exchange method. To deal with boundary objects, at each time step we seed ghost particles randomly in a $2h$ band inside the boundary (where h is the spacing of the grid covering the solid, and with the same density as regular FLIP particle seeding), and set their velocity to that of the solid object. Given particle velocity $u_{p,i}$, the updated particle strength is then calculated using the stable formula

$$u_{p,i}^{n+1} = u_{p,i}^{n} + \frac{\sum_{j\in\eta}(u_{p,j} - u_{p,i})H_\tau(x_i - x_j)}{\sum_{j\in\eta} H_\tau(x_i - x_j)} \tag{2.3}$$

where η is the neighborhood in which we look for particles (a box of size $2h$), and $H_\tau(x)$ is the heat kernel with $\tau = \nu\delta t$, which reads

$$H_{\nu\delta t}(x) = \frac{1}{(4\pi\nu\delta t)^{d/2}} \exp\left(\frac{-\|x\|^2}{4\nu\delta t}\right) \tag{2.4}$$

The $(4\pi\nu\delta t)^{-d/2}$ factor can be elided due to the normalization in Equation 2.3.

Once the particle velocites have been diffused, particles are passively advected through the divergence-free flow field. We used Ralston's 3rd order Runge-Kutta time integrator [47], as the most efficient optimal RK scheme with a stability region overlapping the pure imaginary axis (for rotations):

$$\begin{aligned} u_i' &= \mathrm{SampleVelocity}(x_i) \\ u_i'' &= \mathrm{SampleVelocity}(x_i + 0.5\Delta t u_i') \\ u_i''' &= \mathrm{SampleVelocity}(x_i + 0.75\Delta t u_i'') \\ x_i^{n+1} &= x_i^n + \frac{2}{9}\Delta t u' + \frac{3}{9}\Delta t u'' + \frac{4}{9}\Delta t u''' \end{aligned} \tag{2.5}$$

When sampling velocity from the velocity buffer, we use trilinear interpolation. In the case where multiple overlapping grids are detected, we take the grid with the finest resolution. While this sounds overly simple, and potentially introduces discontinuities at the edges of grids, we have not observed any noticeable artifacts as typically the velocity jumps would be small, and it is only the integral in time of the velocity (the particle trajectories) which we observe. It would not be difficult to use a partition-of-unity blend to get a smoother velocity at grid edges if for some reason this was necessary. After advection, we further update the particle velocities with forcing terms such as buoyancy before pressure projection.

2.3.2 Regional projection for particle velocity correction

Once the post advection particle velocities are known, we use a regional projection method to correct the velocity field to an approximately divergence-free state. Following Batty et al. [6], finding the inviscid divergence-free projection of an input velocity field u^* is equivalent to a minimization problem:

$$\min_p \int_\Omega \frac{1}{2}\rho \| u^* - \frac{\Delta t}{\rho}\nabla p\|^2 - \int_\Omega \frac{1}{2}\rho \|u^*\|^2 + \int_S p\hat{n}\cdot \Delta t u_{pre}, \tag{2.6}$$

where u_{pre} is typically a prescribed velocity at solid or fluid domain boundaries S.

As outlined in Alg. 2, in our regional projection, we first splat all particle velocities to the Eulerian domain Ω with a wide kernel $W_h(x)$, as if we were computing the large eddy filtered version of the velocity field:

$$u_g(x) = \frac{\sum_j u_j W_h(x-x_j)}{\sum_j W_h(x-x_j)}. \tag{2.7}$$

Here x is a fixed grid cell position, and x_j and u_j are the the position and velocity of particle j respectively.

In our implementation, the usual bi-/trilinear hat function kernel was used:

$$\begin{aligned} W_h(x) &= \prod_{\theta=1}^{d} \phi\left(\frac{x_\theta}{h}\right), \text{with} \\ \phi(r) &= \begin{cases} 1-|r|, & r \in [-1,1), \\ 0, & \text{else}, \end{cases} \end{aligned} \tag{2.8}$$

where $d = 2,3$ is the spatial dimension, h is the kernel width which simply equals the grid spacing, and x_θ is the θ^{th} component of vector x.

After we splat the particle velocity to the grid, the velocity field on the grid is made divergence-free by pressure projection. Our projection uses a standard staggered grid and face-weighted pressure discretization similar to Batty et al. [6]. The velocity change is then interpolated to the particles to correct their velocities.

At this stage the velocity defined on the particles is (approximately) divergence-free at the coarse scale. To increase the flow fidelity in regions of interest (near boundaries or cameras, for example), further projections with higher resolution in each subdomain are performed with essentially the same procedure. We splat the coarse-corrected particle velocity to the corresponding refined domain using the kernel $W_h(x)$ with the smaller refined h, make the grid-based velocity field divergence-free, and interpolate the velocity change back to the particles in that region. In each subdomain projection, at the domain boundaries we restrict the normal component of velocity not to change (as established by splatting from the particles). We repeat this process until all the regional refined domains are processed.

During each projection, the Poisson equation for pressure is solved with a multi-grid preconditioned conjugate gradient method, whose implementation details are given in §2.3.3.

2.3.3 Our pressure solver

Similar to McAdams et al. [40] and Setaluri et al. [51], we use a multigrid preconditioned Krylov solver for our pressure Poisson equation. Our multigrid solver is constructed semi-algebraically: we construct $\mathscr{R}_L$ (the restriction matrix mapping the residual of level L to a coarser level) and $\mathscr{P}_L$ (the prolongation matrix used to add coarse level corrections back to the level L solution) using geometric information given by the fluid domain, but use matrix multiplications to construct the coefficient matrix A_{L+1} of each coarsening level $L+1$ (Alg. 3).

Algorithm 3 MultiGridCoarsening(A_L,$\mathscr{P}_L$,$\mathscr{R}_L$, A_{L+1})

```
1: for each cell in coarse level
2:   $i \leftarrow$ index of this cell
3:   for each sub cell in fine level
4:     $j \leftarrow$ index of this sub cell
5:     if $\mathrm{A}_L(j,j) \neq 0$
6:       $\mathscr{R}_L(i,j) = 1/2^d$ //d = 2,3 dimension of the problem
7:       $\mathscr{P}_L(j,i) = 1.0$
8: $\mathrm{A}_{L+1} \leftarrow 0.5\mathscr{R}_L \mathrm{A}_L \mathscr{P}_L$
```

As shown in Alg. 3, our restriction and prolongation matrices are transposes of each other, up to a scale factor, with restriction equivalent to simply averaging residuals across the fine level grid cells covered by a coarse grid cell, and prolongation equivalent to taking the nearest neighbour coarse value (i.e. ”parent” value);

this is a ”aggregation”-style multigrid with constant weights. For pure multigrid, this does not produce optimal convergence; we improve the scheme by applying a scaling of $\frac{1}{2}$ to the coarse level coefficient matrix, which, away from solid boundaries, gives exactly the same coefficient matrix as if we were to rediscretize the PDE on the coarse grid. For cases with solid boundaries, this coarsening strategy acts as a volume weighted rediscretization at the coarse level. Our coarsening strategy doesn't increase the size of the discrete Poisson stencil: a coarse cell has at most six neighbors with non-zero coefficients. This in turn allows for the usual red-black ordering to parallelize Gauss-Seidel sweeps, and in general makes smoothing far more efficient. For other implementation details of multigrid-preconditioned Krylov solvers, please refer to McAdams et al. [40].

In all our experiments, the multigrid-preconditioned conjugate gradient solver appeared to converge with a constant rate of approximately 0.1 per iteration, as shown in Table 2.2, though more complex schemes may be necessary for extremely complex solid geometry.

Table 2.2: The Semi-AMGPCG solver applied to a Poisson problem with single impulse, in a unit cube domain with a zero Neumann boundary condition along a sphere with diameter 0.2 in the center, and zero Dirichlet boundary conditions at the edges of the cube. The number of iterations required to reduce the residual by a factor of 10^{-10} is essentially independent of the grid size, providing one extra digit of accuracy per iteration.

Convergence rate of the AMGPCG solver					
Resolution	64^3	96^3	128^3	192^3	256^3
Convergence criterion	$10^{-10}\|\|r.h.s\|\|_\infty$				
# of iterations	10	11	11	11	11

2.3.4 Seeding and deleting particles

At the end of each time step, particles are seeded and deleted as necessary to maintain reasonable sampling of the fluid in each grid cell, according to the highest resolution grid nearby. Newly seeded particles take their velocity by interpolation from the finest available grid. We delete particles randomly from cells where the count exceeds a threshold (16 in our examples), and seed new particles where the

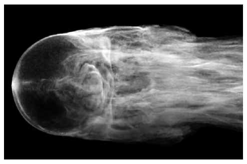


Figure 2.6: Impulsively started flow past a sphere at Re=15000 simulated using our solver

count is below another threshold (8 in our examples).

Referring to figure 2.4, the grey areas surrounding (but not inside) refined subdomains are treated with the finer grid spacing: we maintain a higher particle density in the grey areas as there is a chance that the fluid in the grey region will be advected into the refined subdomain during the course of the next time step. For our examples we used a bandwidth of $3h$, where h is the refined grid spacing, for this intermediate zone but it could of course be determined more dynamically and frugally based on the local velocity and the time step size.

As an alternative to avoid higher particle counts, we have experimented with only seeding extra particles in the refined regions, not the grey neighborhoods, but widening the support of the particle-to-grid kernel near the edge of the refined grid to avoid gaps in the data. This exchanges more work for less storage, which may be preferable in some cases, though the results are similar.

2.4 Results and discussion

We parallelized the proposed methods on a desktop machine with an Intel(R) Core(TM) i7-3930K CPU and 32 GB RAM. We compared our new solver to a typical inviscid FLIP solver as used in graphics, as well as a "naïve" viscous Navier-Stokes solver where viscosity is added with an explicit-in-time finite difference (since we are interested in very high Reynolds numbers, this explicit step was always stable). Simulations were visualized by judiciously injecting smoke marker particles which are passively advected with the grid velocity field. The accompanying video shows comparisons between the methods for different resolutions and different Reynolds numbers in a variety of examples.

In Fig. 2.6, we show a flow simulation frame obtained with our solver at

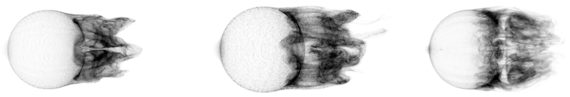


Figure 2.7: Vortex shedding in impulsively started flow around a 2m diameter sphere at Re=20000. Left: naïve grid-based viscosity model with $h = 0.0625$. Middle: naïve solver with $h = 0.03125$ (twice the resolution). Right: our model using a coarse grid of $h = 0.125$ and a refined grid around the sphere with $h = 0.03125$. The computation time for the left and right simulations are roughly equal.

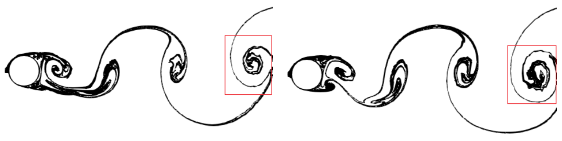


Figure 2.8: For a flow past cylinder simulation at $Re = 800$, inflow speed $U = 5m/s$, and cylinder diameter $D = 1.6m$, the experimental model predicts a Strouhal number of around 0.2 [35], hence a shedding frequency of around 0.625. The above pictures are from frame 754 and frame 920 of our simulation with $\Delta t = 0.01$, where vortices shed off the same side just reach the end of the domain; this gives a shedding frequency just over 0.6.

Reynolds number of 15000: the flow separates from the boundary at small angle, it remains laminar with structured vortices for about one diameter, and then becomes turbulent. This is similar to figure 55 on page 34 of An Album of Fluid Motion [61].

As shown in Fig. 2.7, with the naïve uniform-resolution grid-based viscosity model, a much higher resolution is necessary to capture important fluid motions such as vorticity detaching from the boundary layer. With our sub-grid particle-based viscosity model and regional refinement, the characteristic vortex shedding can be achieved at much lower cost.

In Fig. 2.8 we performed a 2D flow past cylinder simulation. The cylinder is only about 12 cells wide in the coarse grid; a 4× local refinement is applied to resolve the boundary layer. The shedding frequency produced by our solver

Table 2.3: Vortex shedding frequency statistics from numerical simulations

Vortex shedding frequency f_v of a flow-past-sphere simulation in 2D with various Strouhal numbers			
Reynolds number	400	1600	16,000
Strouhal number	0.188	0.195	0.197
Ground truth f_v	0.117	0.122	0.62
Numerical f_v	0.114	0.121	0.611

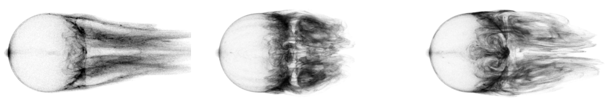


Figure 2.9: Our method on the same scenario as Fig. 2.7 with different resolutions. Left: coarse simulation (global $h = 0.25$, inner $h = 0.0625$). Middle: $2\times$ refinement for both (global $h = 0.125$, inner $h = 0.03125$). Right: only the inner grid is refined (global $h = 0.25$, inner $h = 0.03125$). Important fluid features can be cheaply captured by increasing just the inner grid resolution.

matches the experimental model. We also studied this type of flow with various Strouhal numbers: results are given in Table 2.3.

In Fig. 2.9 we take the same scenario (impulsively started flow around a sphere at $\mathrm{Re} = 20,000$) but look at changing global and inner grid resolution in our method. Vortex shedding is largely dominated by how well a solid boundary is resolved. Our method can resolve some sub-grid scale viscosity effects, but it fails to predict the near-boundary behavior when the refined grid is too coarse to resolve it. However, increasing the resolution of the local grid just around the boundary improves the simulation quality greatly without requiring refinement in the rest of the domain.

When the resolution of the coarse domain isn't enough to represent the solid object, our regional refinement may still be enough to capture the solid boundary well (Fig. 2.1, Fig. 2.10). Vortex shedding from solid objects is essential to fluid animation in these scenarios, which can be faithfully captured by our method.

Our method makes it much cheaper to adequately resolve the viscous boundary layer, locally and on-the-fly, alleviating the limitation to parametrize boundary

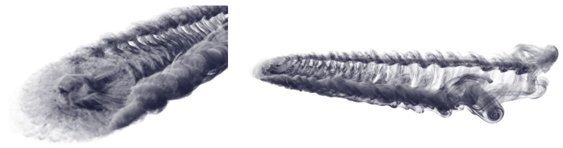


Figure 2.10: Rotating fan (each blade is $1.3m$ long, rotating at one cycle per second) in a constant wind of $2m/s$, with $\nu = 10^{-4}$, giving $Re \approx 10^5$. The refined grid has $h = 0.005m$. Left: zoom-in view at the fan blades. Right: the entire simulation.

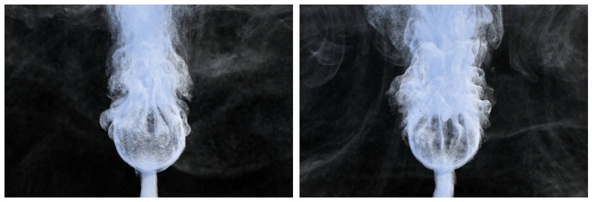


Figure 2.11: Smoke rising around a sphere of diameter 0.3m, with $\nu = 10^{-4}$ and the inner grid $h = 0.005m$, $Re \approx 10^3$. Left: simulation with a uniform FLIP solver. Right: simulation using our solver with a coarser global grid and a $4\times$ refined grid around the sphere. With approximately the same computation time, our solver captures structures in the boundary disturbances over the bottom of the sphere more sharply.

layer with far field flow motions as in Pfaff et al. [45]. It is thus suitable for more general simulation scenarios, e.g. buoyancy-driven plumes as shown in Fig. 2.11.

In Table 2.5, we notice that for the inner grid of 200^3, the simulation time is very close while the outer grid size varies from 100x50x50 to 150x75x75. This is due to the reason that in our experiments, as the resolution of inner grid increases, the particles assigned for the inner domain increases, as a result, the computation time became dominated by the viscous diffusion part, where each particle's velocity is adjusted by the kernel summation. In practice, diffusion only needs to be resolved up to a narrow band near the boundary, hence, the cost function can be quantified as

$$T = c_1 N_g + c_2 N_b \tag{2.9}$$

where, N_g is the total number of grid cells in the entire domain, including the coarse

Table 2.4: Timings of various simulations associated with this paper, measured on a desktop machine with Intel(R) Core(TM) i7-3930K CPU and 32 GB RAM.

Example	Method	Δt	Outer Grid Size	Inner Grid Size	Reynolds Number	Simulation Time (seconds per frame)
Flow past sphere	Naïve solver	0.03	0.0625	N/A	1,000	12.52
		0.03	0.03125	N/A	20,000	73.46
	Our method	0.03	0.125	0.0625	1,000	11.21
		0.03	0.0625	0.03125	20,000	70.76
		0.03	0.125	0.03125	15,000	56.07
Rotating fan in wind	Our method	0.03	0.041	0.02	$\approx 10^5$	120.40
Rising smoke plume		0.05	0.1	0.03	$\approx 10^3$	79.10
Flow past sphere	FLIP	0.03	0.0625	N/A	N/A	11.31
	FLIP + WHIRP	0.03	0.125	0.0625	N/A	4.06
	VFLIP	0.03	0.0625	N/A	N/A	15.41
	VFLIP + WHIRP	0.03	0.125	0.0625	N/A	4.20

Table 2.5: Timings of flow past sphere simulations with only differences in outer and inner grid sizes, measured on a desktop machine with Intel(R) Core(TM) i7-3930K CPU and 32 GB RAM.

Simulation Time		Outer Grid Size		
(seconds per frame)		100x50x50	150x75x75	200x100x100
Inner	66x66x66	12.19	19.71	34.36
Grid	100x100x100	28.99	35.46	50.52
Size	200x200x200	166.14	168.64	185.57

and subdomains, as the fluid advection and projection operations shall take similar cost for every grid cell. On the other hand, the diffusion scales linearly with the number of voxels in a narrow band near the boundary. We fitted our experimental data to get $c_1 = 0.0000126$ and $c_2 = 0.000120$. This shows that the particle diffusion time becomes dominant for highly refined simulations. We believe that rather than the current CPU parallelizeed code, a GPU parallelized code could be used to significantly reduce the computation time for the particle diffusion, as we did in our 2D examples.

2.5 Conclusions, limitations, and future work

We extended incompressible FLIP in two ways, with a particle-based viscosity model that can resolve momentum exchange at higher resolution than the grid, and with an extremely simple regional projection method (together with particle seeding/deletion rules) to adaptively refine in important areas, without needing more complex grid or mesh structures to globally couple different resolutions. Very little extra code beyond a standard FLIP simulator is required to implement this paper. Moreover we showed that matching the qualitative look of some high Reynolds numbers scenarios hinges just on resolving the viscous boundary layer, even while the rest of the domain can use a coarse grid.

The typical inviscid solver used in graphics doesn't take viscosity into account at all, and for example can only lead to vortex shedding through numerical errors related to grid size and time step. Both the naïve method and our method will fail to give characteristic results when the boundary layer isn't resolved, again leading to simulations whose look is more controlled by numerical errors related to grid size and time step than physical parameters. However, our method allows for resolving boundary layers much more efficiently, at which point physical parameters are at last a useful control and further refinement will reliably give similar but more detailed results, as artists generally hope for.

Our solver is still limited formally to first order accuracy in time, and the PSE approach is likely only first order in space: while we feel it's valuable for coarse grids and large time steps, it is not an efficient approach for convergence to a fully accurate solution. The PSE part is also inappropriate for highly viscous flows, where an implicit grid-based solver is almost certainly the better option.

The global time step is also a limitation of this approach to spatial-only adaptivity. The time step selected in graphics simulations, when multiple steps per frame are taken, is frequently determined by restricting the CFL number, i.e. limiting the number of grid cells a fluid particle may travel through in one time step. In refined subdomains with smaller grid cells, this is a much more stringent restriction than is needed for the coarse global grid, wasting some computational effort.

We have not considered fluid phenomena beyond simple smoke scenarios. Fire and explosions, where nonzero divergence may be prescribed by the pressure projection, would require some adjustment to our regional projection scheme. Free surface liquids, such as commonly used for water, would require even more thought if regional grids overlap the free surface.

There are several other avenues for further research. The convection-diffusion part we currently handle with PSE might be improved with other ideas from grid-free methods; we could also look at using an effective eddy viscosity to better model unresolved turbulence. Our regional projection is not limited to regular grids, but could also use unstructured tetrahedral meshes to conform more accurately to solid boundaries (while avoiding the need for a global, conforming, tetrahedral mesh, which could be a useful savings for simulations involving moving rigid bodies). Last but not least, it is possible to also track the vorticity on each particle to correct the missing angular momentum during particle advection, similar to Zhang et al. [67]; replacing the FLIP framework with APIC [29] might also allow for better tracking of vorticity.

Chapter 3

Restoring the Missing Vorticity in Advection Projection Fluid Solvers

Table 3.1: Algorithm abbreviations used through out this paper.

IVOCK	The computational routine (Alg.4) correcting vorticity for advection
SF	Classic Stable Fluids advection [53]
SF-IVOCK	IVOCK with SF advection
SL3	Semi-Lagrangian with RK3 path tracing and clamped cubic interpolation
BFECC	Kim et al.'s scheme [31], with extrema clamping([50])
BFECC-IVOCK	IVOCK with BFECC advection
MC	Selle et al.'s MacCormack method [50]
MC-IVOCK	IVOCK with MacCormack
FLIP	Zhu and Bridson's incompressible variant of FLIP [68]
FLIP-IVOCK	FLIP advection of velocity and density, SL3 for vorticity in IVOCK.

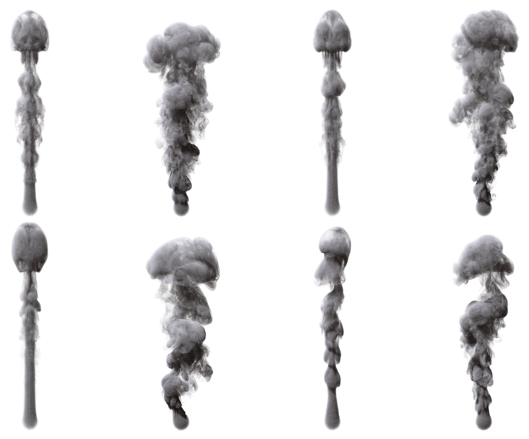


Figure 3.1: Rising smoke simulations with and without IVOCK (Integrated Vorticity of Convective Kinematics). From top left to bottom right,: Stable Fluids, Stable Fluids with IVOCK; BFECC, BFECC with IVOCK; MacCormack, MacCormack with IVOCK; FLIP, FLIP with IVOCK.

3.1 Introduction

In computer graphics, incompressible fluid dynamics are often solved with fluid variables stored on a fixed Cartesian grid, known as an Eulerian approach [53]. The advantages of pressure projection on a regular grid and the ease of treating smooth quantities undergoing large deformations help explain its success in a wide range of phenomena, such as liquids [19], smoke [16] and fire [43]. Bridson's text provides a good background [7].

In Eulerian simulations, the fluid state is often solved with a time splitting method: given the incompressible Euler equation,

$$\begin{gathered}\frac{\partial u}{\partial t}+(u\cdot\nabla)\,u=-\frac{1}{\rho}\nabla p+f\\ \nabla\cdot u=0\end{gathered}\tag{3.1}$$

fluid states are advanced by self-advection and incompressible projection. First

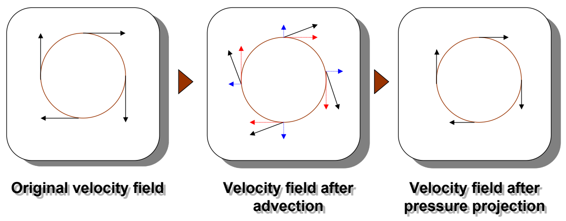

Original velocity field
Velocity field after advection
Velocity field after pressure projection

Figure 3.2: Self-advection maps the original velocity field into a rotational part and a divergent part, indicated by red and blue arrows respectively. Pressure projection removes the blue arrows, leaving the rotational part, and illustrating how angular momentum has already been lost.

one solves the advection equation

$$\frac{Du}{Dt} = 0 \tag{3.2}$$

to obtain an intermediate velocity field $\tilde{u}$, which is then projected to be divergence-free by subtracting ∇p, derived by a Poisson solve from $\tilde{u}$ itself. We formally denote this as $u^{n+1} = \text{Proj}\,(\tilde{u})$.

Self-advection of velocity, without continuous application of the pressure gradient as is typical of the first-order time-splitting schemes used in graphics, disregards the divergence-free constraint, allowing some of the kinetic energy of the flow to be transferred into divergent modes which are lost in pressure projection. We focus in particular on rotational motion: self-advection can sometimes cause a noticeable violation of conservation of angular momentum, as illustrated in Fig.3.2.

Despite many published solutions in improving the accuracy of advection scheme in an Eulerian framework (e.g. [34, 50]), or reducing the numerical diffusion with hybrid particle-in-cell solvers [68], even with higher-order integration [13], only a few papers have addressed this particular type of numerical dissipation.

Retaining the velocity-pressure approach but changing the time integration to a symplectic scheme, Mullen et al. [42] were able to discretely conserve energy when simulating fluids. However, the expense of their non-linear solver for Crank-Nicolson-style integration, and the potential for spurious oscillations arising from non-upwinded advection, raise questions about its practicality in general.

Another branch of fluid solvers take the vorticity-velocity form of the Navier-

Stokes equation, exploiting the fact that the velocity field induced from vorticity is always divergence free. These solvers advect the vorticity instead of the velocity of the fluid, preserving circulation during the simulation. Vorticity is usually tracked by Lagrangian elements such as vortex particles [44], vortex filaments [64] or vortex sheets [8]. Apart from the fact that the computational cost of these methods can be a lot more expensive per-time-step than a grid-based solver (three accurate Poisson solves are required instead of just one), there remain difficulties in handling solid boundaries and free surfaces (for liquids), and users may find it less intuitive working with vorticity rather than velocity in crafting controls for art direction. However, researchers have observed that vortex methods can better capture important visual details when running with the same time step and advection scheme [65].

In this paper, we combine our observation from vortex methods with Eulerian grid-based velocity-pressure solvers to arrive at a novel vorticity error compensation scheme. Our contributions include:

- A new scheme we dub IVOCK (Integrated Vorticity of Convective Kinematics), that tracks and approximately restores the dissipated vorticity during advection, independent of the advection method or other fluid dynamics in play (boundary conditions, forcing terms) being used.
- A set of novel techniques to store and advance vortex dynamics on a fixed spatial grid, maximizing the accuracy while minimizing the computational effort of IVOCK.
- Upgraded classic fluid solvers with IVOCK scheme, documenting the combination of IVOCK with different advection methods and different types of fluids.
- A new vortex stretching model in a non-divergence-free environment for compressible vortex flows occuring in volumetric combustion.

3.2 Related work

Stam's seminal Stable Fluids method [53] introduced a backward velocity tracing scheme to solve advection equation, making physically based fluid animation highly practical for computer graphics. With its unconditional stability, trading accuracy for larger time step, it provided a basis for most subsequent grid-based fluid solvers in graphics, for fluid phenomena such as smoke [16], water [19], thin flames [43], and volumetric combustion [17].

However, the first order accuracy in both time and space of Stable Fluids manifests in strong numerical diffusion/dissipation. Many researchers have taken it as a basic routine to build higher-order advection schemes. Kim et al. [31] proposed the BFECC scheme to achieve higher-order approximation by advecting the field back and forth, measuring and correcting errors. Selle et al. [50] eliminated the last advection step of BFECC to arrive at a cheaper MacCormack-type method, and also introduced extrema clamping in BFECC and MacCormack to attain unconditional stability, at the cost of discontinuities in velocity which can sometimes cause visual artifacts. In this paper, we use these advection schemes as examples to which we apply IVOCK, and show improvements with the IVOCK scheme for smoke and volumetric combustion simulations.

Hybrid particle-grid methods have been introduced to further reduce numerical diffusion in adection, notably Zhu and Bridson's adaptation of FLIP to incompressible flow [68]. Although FLIP is almost non-diffusive for advection, the velocity-pressure solver nevertheless may dissipate rotational motion as shown in figure 3.2, regardless of the accuracy of advection. FLIP cannot address this issue; in this paper we show our FLIP-IVOCK scheme outperforms FLIP in enhancing rotational motions for smoke.

Resolution plays an important role in fluid animations. McAdams et al. [40] proposed multigrid methods to efficiently solve the Poisson equation at a uniform high resolution. Losasso et al. [37] introduced a fluid solver running on an octree data structure to adaptively increase resolution where desired, while Setaluri et al. [51] adopted a sparse data approach. In our work, multigrid is employed for the vorticity-stream function solver while an octree code is used to impose domain-boundary values using Barnes-Hut summation.

Vortex dynamics has proved a powerful approach to simulating turbulence. Lagrangian vortex elements such as vortex particles [44], vortex filaments [64], or vortex sheets ([8], [46]) have been used to effectively model the underlying vorticity field. Recently, Zhang and Bridson [66] proposed a fast summation method

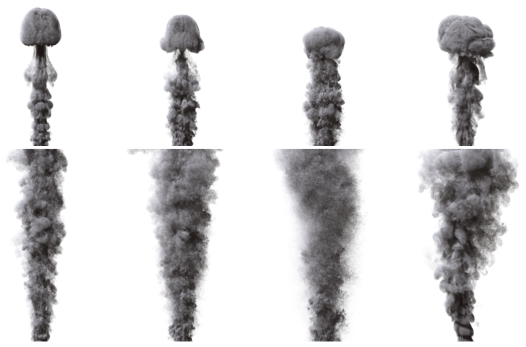


Figure 3.3: Vorticity confinement (VC) vs. IVOCK. Top row: frame 54 of a rising smoke scenario. From left to right, VC parameter $\varepsilon_1 = 0.125$, $\varepsilon_2 = 0.25$, $\varepsilon_3 = 0.5$, and SL3-IVOCK. Bottom row: frame 162. Vorticity confinement tends to create high-frequency noise everywhere, while IVOCK produces a natural transition from laminar to turbulent flow with realistic vortex structures along the way.

to reduce the computational burden for Biot-Savart summation, in an attempt to make these methods more practical for production. Unfortunately, these methods are less intuitive for artists (velocity can't be modified directly), tend to be more expensive (finding velocity from vorticity requires the equivalent of three Poisson solves), and continue to pose problems in formulating good boundary conditions. We were nevertheless inspired by the mechanism to induce velocity from vorticity, and constructed our IVOCK scheme as a way to bring some of the advantages of vortex methods to more practical velocity-pressure solvers.

To balance the trade-off between pure grid-based methods and pure Lagrangian vortex methods, researchers have incorporated vorticity-derived forces in Eulerian simulations. Foremost among these is the vorticity confinement approach [54], where a force field is derived from the current vorticity field to boost it. Selle at al. [49] refined this by tracing "vortex particles", source terms for vorticity confinement tracked with the flow which provides better artistic control over the added turbulence. Unfortunately, vorticity confinement relies on a non-physical parameter which must be tuned by the artist. As we can clearly see in figure 3.3, too small a parameter doesn't produce interesting motion for the early frames of a smoke animation, while still turning the smoke into incoherent noise in later frames. IVOCK on the other hand is built in a more principled way upon vortex dynamics, partially correcting the truncation error in velocity-pressure time-splitting, and produces natural swirling motions without any parameters to tune. In addition, it is orthogonal to vorticity confinement (viewed as an art direction tool) and can hence be used together (figure 3.13).

Turbulence can also be added to the flow as a post-simulation process, such as with wavelet turbulence [32]. In particular, the wavelet up-sampling scheme relies on a good original velocity field to produce visually pleasing animations; IVOCK is again orthogonal to this and could be adopted to enhance the basic simulation.

3.3 The IVOCK scheme

When solving Navier-Stokes, the fluid velocity is usually advanced to an intermediate state ignoring the incompressibility constraint:

$$\tilde{u} = \text{Advect}(u^n), \tag{3.3}$$

$$\tilde{\tilde{u}} = \tilde{u} + \Delta t f, \tag{3.4}$$

where u^n is the divergence-free velocity field from the previous time-step and f is a given force field which may include buoyancy, diffusion, vorticity confinement, and artistically controlled wind or motion objects.

From this intermediate velocity field $\tilde{\tilde{u}}$ one can construct the final divergence-free velocity u^{n+1} with pressure projection, which is usually the place where boundary conditions are also handled:

$$u^{n+1} = \text{Proj}\left(\tilde{\tilde{u}}\right). \tag{3.5}$$

We observe that in vortex dynamics, the intermediate velocity field $\tilde{u}$ of equation 3.3 is analogously solved using the velocity-vorticity ($u - \omega$) formula:

1. given ω^n, solve
$$\frac{D\omega}{Dt} = \omega \cdot \nabla u \tag{3.6}$$
to get $\tilde{\omega}$;

2. deduce the intermediate velocity field $\tilde{u}$ from $\tilde{\omega}$.

Equation 3.6 is the advection-stretching equation for vorticity in 3D, and step 2 of this $(u-\omega)$ formula is usually solved using a Biot-Savart summation, or equivalently by finding a streamfunction Ψ (vector-valued in 3D), which satisfies $\nabla^2\Psi = -\omega$, and readily obtaining the velocity $\tilde{u}$ from $\tilde{u} = \nabla \times \Psi$.

Equation 3.6 suggests the post-advection vorticity field $\omega^* = \nabla \times \tilde{u}$ should ideally equal the stretched and advected vorticity field $\tilde{\omega}$, but due to the simple nature of self-advection of velocity ignoring pressure, there will be an error related to the time step size.

We therefore define a vorticity correction $\delta\omega = \tilde{\omega} - \omega^*$, from which we deduce the IVOCK velocity correction δu and add this amount to the intermediate velocity $\tilde{u}$. Algorithm 4 provides an outline of the IVOCK computation before we discuss the details.

Algorithm 4 IVOCKAdvection(Δt,u^n, $\tilde{u}$)

```
1: $\omega^n \leftarrow \nabla \times u^n$
2: $\tilde{\omega} \leftarrow$ stretch($\omega^n$)
3: $\tilde{\omega} \leftarrow$ advect($\Delta t$, $\tilde{\omega}$)
4: $\tilde{u} \leftarrow$ advect($\Delta t$, $u^n$)
5: $\omega^* \leftarrow \nabla \times \tilde{u}$
6: $\delta\omega \leftarrow \tilde{\omega} - \omega^*$
7: $\delta u \leftarrow$ VelocityFromVorticity($\delta\omega$)
8: $\tilde{u} \leftarrow \tilde{u} + \delta u$
```

In essence, IVOCK upgrades the self-advection of velocity to match self-advection of vorticity, yet retains most of the efficiency and all of the flexibility of a velocity-pressure simulator. In particular, IVOCK doesn't change the pressure computation (as the divergence of the curl of the correcting streamfunction is identically zero, hence the right-hand-side of the pressure projection is unchanged by IVOCK), but simply improves the resolution of rotational motion for large time steps, which we earlier saw was hurt by velocity self-advection.

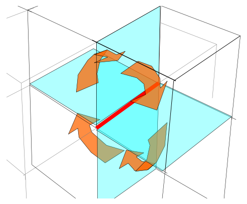


Figure 3.4: Vorticity and streamfunction components are stored in a staggered fashion on cell edges in 3D (red line), while velocity components are stored on face centers. This permits a natural curl finite difference stencil, as indicated by the orange arrows.

3.3.1 Vortex dynamics on grids

Data storage

Extending the classic MAC grid [25] widely adopted by computer graphics researchers for fluid simulation, we store vorticity and streamfunction components on cell edges in a staggered fashion, so that the curl operator can be implemented more naturally, as illustrated in figure 3.4. For example, if the grid cell size is h, the z-component of vorticity on the z-aligned edge centred at $(ih, jh, (k+\frac{1}{2})h)$ is approximated as

$$\omega_z(i,j,k+\frac{1}{2}) = \frac{1}{h}\left[\Big(v(i+0.5,j,k) - v(i-0.5,j,k)\Big) - \Big(u(i,j+0.5,k) - u(i,j-0.5,k)\Big)\right]. \quad (3.7)$$

Vortex stretching and advection on grid

In 3D flow, when a vortex element is advected by the velocity field, it is also stretched, changing the vorticity field.

We solve equation 3.6 on an Eulerian grid using splitting; we solve

$$\frac{\partial \omega}{\partial t} = \omega \cdot \nabla u, \quad (3.8)$$

to arrive at an intermediate vorticity field $\omega^{n+\frac{1}{2}}$, which is then advected by the

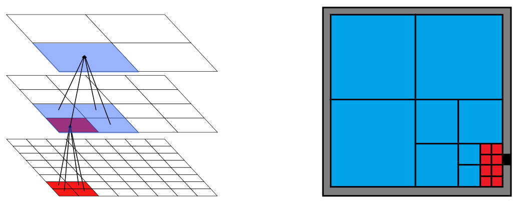


Figure 3.5: Barnes-Hut summation [5] for the boundary values demonstrated in 2D: we construct the monopoles of tree nodes from bottom-up (left), and then for an evaluation voxel (black) in the ghost boundary region (grey), we travel down the tree, accumulating the far-field influence by the monopoles (blue nodes), and do direct summation only with close enough red cells (right).

velocity field as

$$\frac{D\omega}{Dt} = 0. \tag{3.9}$$

When discretizing equation 3.8 on an Eulerian grid, a geometrically-based choice would be to compute the update with the Jacobian matrix of u:

$$\omega^{n+\frac{1}{2}} = \omega^n + \Delta t \mathscr{J}(u)\,\omega^n. \tag{3.10}$$

Constructing the Jacobian matrix and computing the matrix vector multiplication involves a lot of arithmetic, so we simplified this computation by using

$$\omega \cdot \nabla u = \frac{\partial u}{\partial \omega} = \lim_{\varepsilon \to 0} \frac{u(x + 0.5\varepsilon\omega) - u(x - 0.5\varepsilon\omega)}{\varepsilon}. \tag{3.11}$$

In our computation, with h the grid cell width, we choose $\varepsilon = \frac{h}{\|\omega\|_2}$, which can be seen as sampling the velocity at the two ends of a grid-cell-sized vortex segment and evaluating how this segment is stretched by the velocity field.

Once the vorticity field is stretched, it is advected by any chosen scheme introduced in §3.2 to get $\tilde{\omega}$.

Obtaining δu from $\delta\omega$

To deduce the velocity correction from the vorticity difference, we solve the Poisson equation for the stream function:

$$\nabla^2\Psi = -\delta\omega \tag{3.12}$$

We solve each component of the vector stream function separately: Let subscript x indicate the x-component of the vector function, we have

$$\nabla^2\Psi_x = -\delta\omega_x, \tag{3.13}$$

The equations for y and z components can be obtained in a similar fashion.

In vortex dynamics, it is important for this Poisson equation to be solved in open space to get natural motion, without artifacts from the edges of the finite grid. The open space solution of Poisson's equation can be found by a kernel summation with the corresponding fundamental solution. Consider the scalar Poisson problem,

$$\begin{aligned} \nabla^2\Psi_x &= -\delta\omega_x \\ \Psi_x(p) &= 0, p \to \infty, \end{aligned} \tag{3.14}$$

Its solution, in a discrete sense, is given by the N-body summation

$$\Psi_x(p_i) \approx \sum_{\text{all} j, j\neq i} \frac{\delta\omega_{xj} v_j}{4\pi r_{i,j}}, \tag{3.15}$$

where $\delta\omega_{xj}$, v_j are the vortex source strength (component-wise) and volume of each sample point (voxel), respectively, p_i is the evaluation position, and r_{ij} is the distance between the i^{th} and j^{th} sample positions. With M evaluation points (on the six faces of the domain boundary) and N source points (the number of voxels), the direct N-body summation has an $\mathcal{O}(MN)$ cost. However, once an evaluation position p_i is far from a cloud of s sources, the summation of the influence of each individual source can be accurately approximated by the influence of the monopole of those sources:

$$\sum_{j=1}^{s} \frac{\delta \omega_{x j} v_j}{4\pi \|p_i - p_j\|} \approx \frac{1}{4\pi \|p_i - p_c\|} \sum_{j=1}^{s} \delta \omega_{x j} v_j, \tag{3.16}$$

where p_c is the center of mass of the cloud, and $\sum_{j=1}^{s} \delta \omega_{x j} v_j$ is the so-called monopole. Defining an octree on the voxels, akin to multigrid, we construct from bottom-up the monopoles of clusters of voxels, and then for each evaluation position, we traverse the tree top-down as far as needed to accurately and efficiently approximate the N-body summation, as illustrated in Fig. 3.5. Similarly to Liu and Doorly [36], we apply this summation scheme at the $M \propto N^{\frac{2}{3}}$ boundary cells to approximate the open-space boundary values at an $\mathcal{O}(N^{\frac{2}{3}} \log(N))$ cost, which we then use as boundary conditions on a more conventional Poisson solve.

We discretize the Poisson equation with a standard 7-point finite difference stencil. Because the vorticity difference is small (one can view it as a truncation error proportional to the time step), we can get away with applying a single multigrid V-cycle to solve for each component of the stream function. Recall this is not the stream function for the complete velocity field of the flow, just the much smaller velocity correction to the intermediate self-advected velocity! One V-cycle provides adequate accuracy and global coupling across the grid at a computational cost of only about 20 basic red-black Gauss-Seidel iterations. This is cheaper than the pressure solve, which requires more than three V-cycles for the required accuracy, and is also substantially cheaper than a vorticity-velocity solver which requires three accurate Poisson solves.

Once the approximate streamfunction is determined, the velocity correction is computed as $\delta u = \nabla \times \Psi$.

3.3.2 Discussion

Vortex stretching

Before proceeding to the 3D applications of the IVOCK scheme for smoke §3.4.1, water §3.4.2 and combustion §3.4.3, we present a few examples illustrating the effect of vortex stretching.

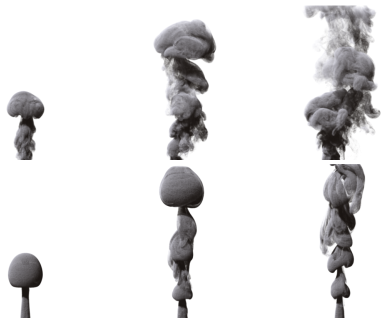


Figure 3.6: Two sequences from 3D rising smoke simulations. Top row: MC-IVOCK with vortex stretching. Bottom row: MC-IVOCK with vortex stretching switched off. With vortex stretching, vortex rings change their radius under the influence of other vortex rings, these process can easily perturb the shape of vortex rings, breaking them to form new vortex structures, which brings rich turbulence into the flow field.

In 2D flows, vortex stretching doesn't take place, hence one need only advect the vorticity field. Figure 3.7 compares SF and SF-IVOCK in 2D.

In 3D flows, vortex stretching plays an important role. Rising vortex rings leapfrogging each other is a physically unstable structure: the vortex rings break up under small perturbations and form new vortex structures. However, due to the added complexity of implementing the vortex stretching term, and earlier questions about it stability, practitioners are often tempted to simply drop it, despite this not being consistent with the fluid equations. Figure 3.6 illustrates that IVOCK without vortex stretching produces, at large time steps, visibly wrong results. In figure 3.8, we show that the ability of the IVOCK method to capture vortex stretching can bring more turbulence to the flow, such as wrinkles on the smoke front, again with relatively large time steps.

Additional forces

IVOCK is only a correction to the self-advection part of a standard velocity-pressure solver. As such, we do not need to take into account how additional force terms such as buoyancy, viscosity, and artist controls will interact with vorticity. These forces are incorporated into the velocity directly in a different step.

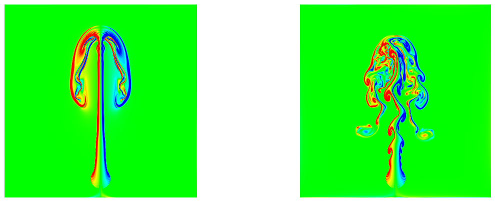


Figure 3.7: Left: 2D buoyancy flow simulated with SF. Right: the same with SF-IVOCK. The resolution and time step were the same; the IVOCK scheme produces a more richly detailed result.

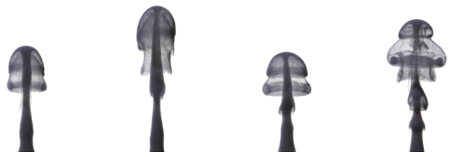


Figure 3.8: Vortex stretching enhances vortical motion, captured more accurately with IVOCK. Transparent renderings illustrate the internal structures. Left and middle-lfet: FLIP. Middle-right and right most: FLIP-IVOCK.

Accuracy vs. performance

While more multigrid V-cycles can be performed to improve the accuracy of the stream function solver, we observe the visual improvement is minor. In our experiments, more than 60% extra computation time was required for the solver to converge to a relative residual reduction of 10^{-6}, which is probably not worthwhile. Fig. 3.9 shows two simulations obtained with different convergence criteria.

Figure 3.9: Left: Frame 150 of an IVOCK simulation with only one multigrid V-cycle. Right: Frame 150 of an IVOCK simulation where multigrid V-cycles are taken reduce the residual by 10^{-6}.

Boundary conditions

While solving IVOCK dynamics, we don't take boundaries into account at all: we pose the vorticity-velocity equation in open space, and allow the subsequent pressure projection to handle boundaries. However, in some simulations with a strong shear layer along a viscous boundary, we found IVOCK could cause instability; this was cured easily by zeroing out $\delta\omega$ within a few grid cells of the boundary. We hypothesize the root cause of this problem is that the inviscid fluid equations we numerically solve predict free-slip boundary conditions at solids, allowing essentially an arbitrary $O(1)$ jump in tangential velocity across a solid boundary; evaluating vorticity with a stencil which also includes a voxel with solid velocity then results in an unphysical $O(1/h)$ singular spike, causing instability. Unless viscosity is taken into account *and* the viscous boundary layer is sufficiently resolved on the grid, evaluating vorticity near a solid is almost certainly numerically ill-advised.

It is worth noting that velocity-pressure solvers in general, and IVOCK in particular, rely on the numerical diffusion of the advection to produce vortex shedding along boundaries (which are otherwise not predicted by the fully inviscid equations). Figure 3.13 shows an example where rising smoke collides with an obstacle; the velocity boundary condition is handled by the pressure solver, and no special treatment is needed for the IVOCK streamfunction computation.

3.4 Applications and results

3.4.1 Smoke

The clearest application of IVOCK is in enriching smoke simulations, where vortex features are of crucial importance visually.

In 2D, we initialized a raising vortex pair from a heat source. With SF, this vortex pair all but vanishes after 100 time steps; SF-IVOCK preserves the vorticity much better, as shown in figure 3.10. We recorded the total vortex strength (enstrophy) at each time step, plotted in figure 3.11.

In 3D, we applied IVOCK to different advection schemes. Figure 3.1 illustrates the qualitative improvements to all of them. For large time steps, IVOCK

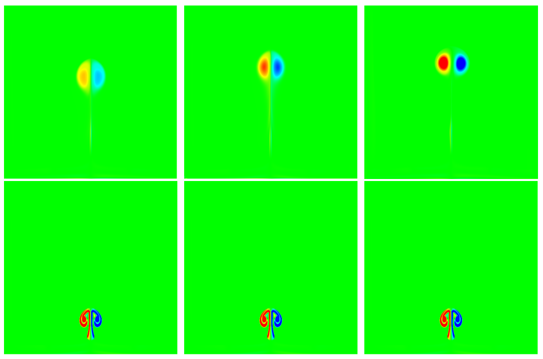


Figure 3.10: Rising vortex pair initialized by a heat source. Left column: SF with $\Delta t = 0.01$. Middle column: SF with $\Delta t = 0.0025$. Right column: SF-IVOCK with $\Delta t = 0.01$. Notice IVOCK preserves vorticity best and produces the highest final position among the three.

significantly enhances the rotational motion and structures before turbulence fully develops.

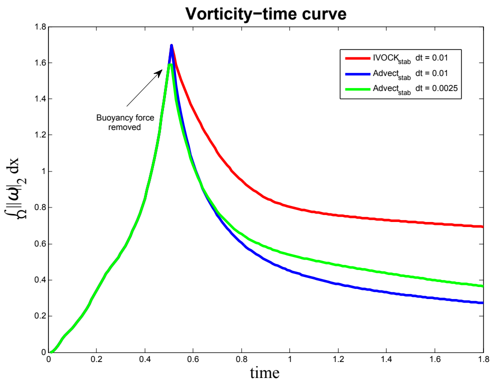


Figure 3.11: Vorticity vs. time curve of the 2D vortex pair simulation. While IVOCK does not conserve vorticity exactly due to approximations in advection and the streamfunction computation, it still preserves significantly more.

Computationally speaking, the overhead for IVOCK includes the stretching computation, vorticity evaluation, vorticity advection, and three V-cycles to approximate the vorticity streamfunction. These operations add about 10% to 25% extra runtime to the original method as a whole: see table 3.2 for sample per-

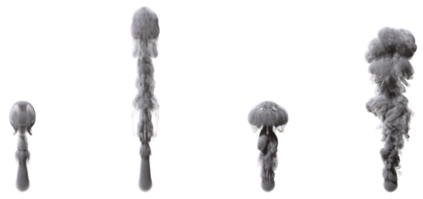


Figure 3.12: IVOCK can be cheaper and higher quality then taking small time steps. Left and mid-left: BFECC. Mid-right and right: BFECC-IVOCK with twice as large a time step, computed in much less time.

formance numbers. Alternatively put, the runtime overhead of IVOCK could be equivalent to improving the original method with about $0.94 \times \Delta x$ in spatial resolution, or taking about $0.83\times$ smaller time steps. Figure 3.12 illustrates that running IVOCK produces better results than even a simulation with half the time step size, despite running much faster.

Table 3.2: Performance comparison of IVOCK augmenting different schemes, for a smoke simulation at 128x256x128 grid resolution, running on an Intel(R) Core(TM) i7-3630QM 2.40GHz CPU.

Method	sec /time step	% overhead
SF	26.6	
SF-IVOCK	30.5	14%
SL3	27.7	
SL3-IVOCK	32.6	17%
BFECC	31.8	
BFECC-IVOCK	39.9	28%
MC	27.9	
MC-IVOCK	35.4	26%
FLIP	45.5	
FLIP-IVOCK	51.4	12%

3.4.2 Liquids

We also implemented IVOCK for liquids solved as free surface flow (see figure 3.14), with a solver based on Batty et al.'s [6]. In this case we only compute the vorticity correction $\delta\omega$ inside the liquid, sufficiently below the free surface so that

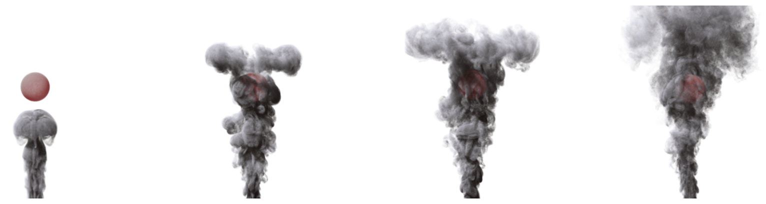


Figure 3.13: Rising smoke hitting a spherical obstacle: the velocity boundary condition is handled entirely by the pressure solve, and doesn't enter into the IVOCK scheme at all. This example includes a small additional amount of vorticity confinement to illustrate how the methods can be trivially combined.

extrapolated velocities are not involved in the vorticity stencil, as otherwise we found stability issues. We solve for the velocity correction with a global solve as before, disregarding boundaries.

Interestingly, we found IVOCK made little difference to the common water scenarios we tested, even ones where we made sure to generate visible vortex structures (e.g. trailing from a paddle pushing through water). We note first that typically the only visible part of a liquid simulation is the free surface itself, unlike smoke and fire, so interesting vortex motions under the surface are relatively unimportant. We also hypothesize that most visually interesting liquid phenomena has to do with largely ballistic motion (splashes) or irrotional motion (as in ocean waves), chiefly determined by gravity, so internal vorticity is of lesser importance. While potential flow can be modeled with a vortex sheet along the free surface, we leave capturing this stably in IVOCK to future work.

3.4.3 Fire

For combustion, the velocity field is not always divergence-free due to expansions from chemical reactions and intense heating. We subsequently modify the vortex stretching term, noting that ω/ρ is the appropriate quantity to track [58]:

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\omega}{\rho}\right) = \left(\frac{\omega}{\rho}\right) \cdot \nabla u. \tag{3.17}$$

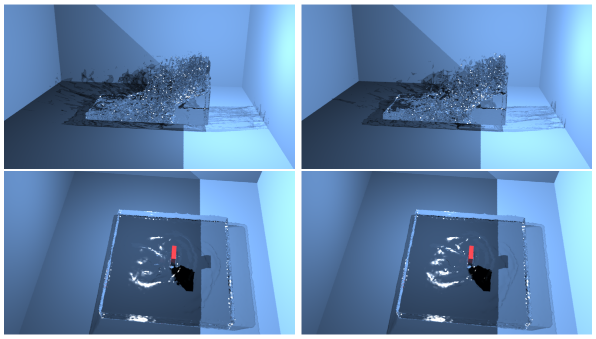


Figure 3.14: IVOCK applied to liquids. Top row: dam break simulations obtained with FLIP (left) and FLIP-IVOCK (right). Bottom row: moving paddle in a water tank, simulated with FLIP (left) and FLIP-IVOCK (right). In these and several other cases we tested, FLIP-IVOCK is not a significant improvement, presumably because interior vorticity is either not present (irrotational flow) or not visually important.

We discretize equation 3.17 with Forward Euler,

$$\frac{1}{\Delta t}\left(\frac{\omega^{n+1}}{\rho^{n+1}} - \frac{\omega^n}{\rho^n}\right) = \left(\frac{\omega^n}{\rho^n}\right)\cdot\nabla u^n \tag{3.18}$$

which implies that

$$\omega^{n+1} = \left(\frac{\rho^{n+1}}{\rho^n}\right)(\omega^n + \Delta t\,\omega^n\cdot\nabla u^n). \tag{3.19}$$

Therefore the new strength of a vortex element, after being stretched and advected, shall be scaled by $\frac{\rho^{n+1}}{\rho^n}$.

In compressible flow, mass conservation can be written as

$$\frac{\partial\rho}{\partial t} + \nabla\cdot(\rho u) = \frac{\mathrm{D}\rho}{\mathrm{D}t} + \rho\nabla\cdot u = 0. \tag{3.20}$$

Rewriting equation 3.20 gives

$$\frac{1}{\rho}\frac{\mathrm{D}\rho}{\mathrm{D}t} = \frac{\mathrm{D}\ln\rho}{\mathrm{D}t} = -\nabla\cdot u, \tag{3.21}$$

and by using Forward Euler again we get

$$\frac{\rho^{n+1}}{\rho^n} = \exp\left(-\Delta t \nabla \cdot u\right). \tag{3.22}$$

Plugging equation 3.22 into equation 3.17, we observe that in compressible flow, the vorticity field after being stretched and advected, should be scaled by a factor of $\exp\left(-\Delta t \nabla \cdot u\right)$. This scaling is the only change we make to IVOCK for compressible flow. We combined the modified IVOCK scheme with traditional volumetric combustion models described by Feldman et al. [17]; figure 3.15 shows example frames from such an animation.

3.5 Conclusion

We argue IVOCK is an interesting stand-alone method to cheaply enrich the highly flexible grid-based velocity-pressure framework with much better resolved vorticity at large time steps, and which can be applied to a variety of advection schemes and fluid phenomena. We believe it brings many of the advantages of vortex solvers to velocity-pressure schemes, but with only $10 \sim 25\%$ extra computation and without the complexity of handling boundary conditions etc. in a vorticity formulation. However, there are some limitations and areas for future work we would highlight.

Currently IVOCK is limited to fluid simulations on uniform grids. A version for adaptive grid or tetrahedral mesh solvers doesn't appear intrinsically difficult, as it could reuse the solver's Poisson infrastructure. Fully Lagrangian velocity-pressure particle methods, especially those that already require some form of vorticity confinement to combat intrinsic numerical smoothing, such as position-based fluids [38], may also benefit from the IVOCK idea.

The presented IVOCK scheme doesn't exactly recover total circulation (or energy) as Mullen et al.'s symplectic integrator does [42], nor does it reproduce turbulent flow quite as well as Lagrangian vortex methods. We also have at present little more than numerical evidence to argue for its stability, and we haven't thoroughly investigated the trade-offs of using just an approximate multigrid solution for the vorticity-velocity step. It could be worthwhile to enhance the stability by exploring Elcott et al.'s circulation preserving method [14].

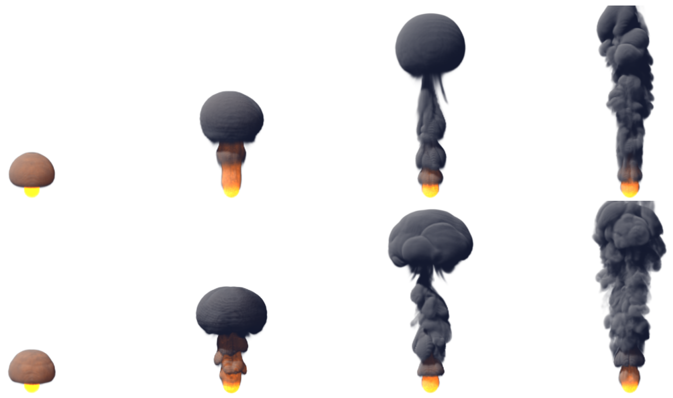


Figure 3.15: Fire simulations making use of our combustion model. Top row: BFECC. Bottom row: BFECC augmented with IVOCK using SL3 for vorticity and scalar fields. The temperature is visualized simply by colouring the smoke according to temperature.

The IVOCK scheme alone doesn't provide any insight into boundary layer models or unresolved subgrid details, hence one possible future project could be to combine it with subgrid turbulence models, for post-processing [32] or on-the-fly synthesis [45]. However, we did observe stability problems with IVOCK compensation active in strongly shearing boundary layers; until a better understanding of boundary layers is reached, we recommend only including the vorticity difference driving IVOCK at some distance away from boundary layers.

Chapter 4

A PPPM Fast Summation Method for Fluids and Beyond

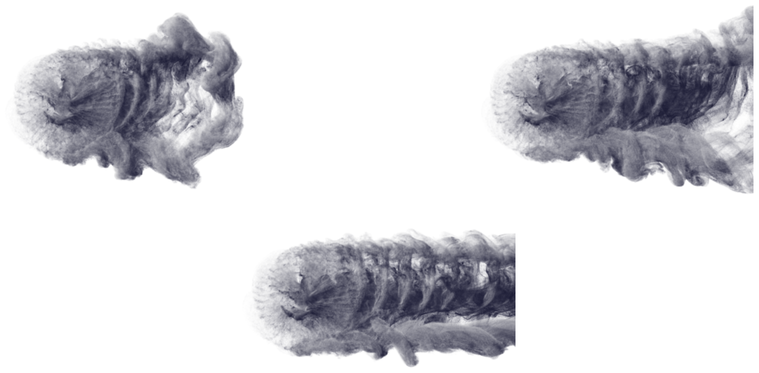


Figure 4.1: Vortices of raising smoke hitting a moving ball (ball not rendered).

4.1 Introduction

A common numerical problem in and outside of graphics is solving the Poisson equation $\nabla^2 f = -\rho$ whose exact solution (assuming zero boundary conditions at infinity for now) can be expressed as an integral with the fundamental solution kernel, in 3D:

Table 4.1: Common symbols used throughout the paper.

x_i	position of i^{th} Lagrangian element
X	position of a grid cell's center
$u(x)$	velocity evaluated for Lagrangian elements
$u(X)$	velocity evaluated at grid cells
u_{far}	the velocity introduced by far field vorticity
u_{near}	the velocity introduced by near field vorticity
u_ω	rotational part of the flow velocity
u_ϕ	irrotational, divergence free part of the flow velocity
u_∞	velocity at infinity, such as the speed at which the reference frame is moving
ω	3D vector valued vorticity strength
σ	scalar valued charge density per unit area
h	width of the grid cell
K	$K = 0, 1, 2, 3, ...$ the local correction range
Ψ	a vector valued stream function
Φ	scalar valued potential function
ω_L	the local vorticity field defined on nearby cells
Ψ_L	the local stream function with source term ω_L
u_L	velocity introduced by ω_L

$$f(x) = \int \rho(x') \frac{1}{4\pi \|x - x'\|} dx'. \tag{4.1}$$

(In other dimensions, the kernel is of course a little different).

In many physics applications, $\rho(x)$ is a sum of point sources at positions x_j and strengths ρ_j, or can be approximated as such: $\rho(x) = \sum_j \rho_j \delta(x - x_j)$ (e.g. vortex blobs [9]). In astrophysics, calculating the gravitational force by summing over all masses is essentially solving the Poisson problem with the density distribution as the right-hand side [27]. The same Poisson problem appears in electrostatics with a charge distribution on particles (or charge panels), with a recent interesting application in graphics [63]. A more thorough review regarding the Poisson problem and N-body dynamics can be found in Hockney and Eastwood's work [27]. In those simulations, the solution is usually obtained by

$$f(x) = \sum_{j: x \neq x_j} \frac{\rho_j v_j}{4\pi \|x - x_j\|}. \tag{4.2}$$

With N particles and M locations to evaluate, often at the N particles themselves, this takes $O(MN)$ time, which scales poorly.

The Poisson problem or its solution as a summation over particles has appeared in diverse graphics papers. For example, Kazhdan et al. [30] solve a Poisson problem to reconstruct a surface from point samples; Jacobson et al. [28] segment the inside from the outside of triangle soup by constructing a harmonic function from sources on triangle faces, again using a summation approach; many fluid solvers (e.g. [16, 53]) solve a Poisson problem every time step for pressure projection; and Lagrangian vorticity-based fluid solvers (e.g. [44]) evaluate the velocity field by summing over vortex sources with the Biot-Savart law, which is just the curl of equation (4.2).

While sometimes it is natural to discretize the Poisson equation on a grid with finite differences, and impose some boundary condition on the edges of the grid, often we want to solve the "free space" problem with boundaries only at infinity, or with sources or boundaries distributed arbitrarily through space: on particles, on contours, on the triangles of a mesh, etc. Here the integral or summation approach would shine but for the cost of directly evaluating the sum.

Alternative methods exist to accelerate the expensive summation. Foremost among these is the Fast Multipole Method (FMM) [23], which can reduce the cost from $O(MN)$ to $O(M+N)$. Probably due to implementation complexity and (typically) a large overhead in runtime, the FMM has not been widely adopted in graphics: parallelized direct summation can easily outperform FMM for N up to 100,000 in our experience.

Vortex-in-cell (VIC) methods [12] have also been adopted to efficiently albeit roughly approximate the velocity field. VIC splats the right-hand side of the Poisson problem to a fine grid, solves the PDE on the grid with finite differences or similar, and interpolates the solution back from the grid. The computational complexity is typically dominated by an efficient grid solve. However, VIC loses accuracy when the vorticity fields are under-resolved on the grid resolution; transferring values back and forth between grid and particles forms a major source of numerical diffusion.

A more promising approach, in our view, is the Particle-Particle Particle-Mesh (PPPM) method [1]. The Particle-Mesh component of PPPM approximates the smooth long-range interactions by splatting the particles on to a grid (mesh), using a fast solver for a finite-difference version of the Poisson problem on that grid, and then interpolating back to evaluation points, just like VIC. The Particle-Particle component greatly enhances the accuracy by including the kernel summation just for very close pairs of particles, whose interactions aren't properly captured with the grid. Assuming the distribution of particles is such that a moderate resolution grid exists with a small number of particles per cell, the particle operations in PPPM are $O(M+N)$ and the overall runtime is determined by the grid solver (which traditionally has been $O(N\log N)$ with an FFT solver). Moreover, operations such as splatting to or interpolating from a grid are easy to implement with little overhead. That said, prior PPPM methods have had trouble achieving linear runtime (due to the fast solver used) and full accuracy (due to boundary conditions at the edges of the grid, and errors associated with near-field interactions as computed on the grid).

This paper first extends PPPM with two significant algorithm improvements:

- a new "monopole" grid boundary condition to accurately simulate unbounded domains,
- and a new error correction for nearby particle contributions in the grid solve that is simple, fast, and accurate.

We also upgrade the traditional fast solver to linear-time multigrid, and parallelize it all on the GPU for exceptional performance.

In the second part we turn to applications of our new PPPM, in particular a new vortex particle smoke solver. Lagrangian vortex methods succinctly represent richly detailed flow, and suffer no numerical diffusion or projection-related dissipation as other solvers may. Their central step is reconstructing velocity from vorticity via the Biot-Savart law, essentially the curl of equation (4.2),

$$u(x) = \sum_{j:x\neq x_j} \nabla \frac{1}{4\pi\|x-x_j\|} \times \omega_j \tag{4.3}$$

or equivalently solving a Poisson problem for a potential Ψ and taking its curl to get the velocity. Inviscid solid boundary conditions can be implemented via a potental flow correction to this velocity field, which through a boundary integral discretization with an iterative linear solve likewise relies on an N-body summation / Poisson solve. We use our new PPPM method in each case to achieve a high performance, high quality smoke simulation. The greater accuracy of our PPPM approach gives visually superior results than simpler techniques such as Vortex-in-Cell and truncated kernels.

Apart from the challenge of efficient N-body summation, prior vortex particle methods in graphics have had difficulties with the vortex stretching term (necessary for conservation of angular momentum and the characteristic look of developing turbulence) and vortex shedding from solid boundaries. We propose solutions:

- an easy-to-implement vortex-segment-derived treatment of vortex stretching with automatic conservation properties,
- and a simple boundary layer treatment to shed vortices from solids, based on injecting vortex particles to boost the potential flow no-stick boundary condition to a no-slip boundary.

We also show that PPPM can be exploited for non-physics-related problems in graphics, such as Poisson surface reconstruction.

4.2 Related work

Vortex methods are popular for high Reynolds number computational fluid dynamics [11], very efficiently capturing turbulent details; as vorticity in particular is visually very important, many graphics researchers have adopted vortex approaches. Yaeger et al. [65] began the trend with a VIC solver to produce the realistic look of atmospheric flow on Jupiter. Angelidis and Neyret [4] used Lagrangian vortex primitives for their flexibility; Selle et al. [49] used vortex particles to augment a more traditional Eulerian velocity-pressure simulation. Pfaff et al. [45] used vortex particles from pre-processed boundary layers to synthesize turbulence. Weißmann and Pinkall [64] used Biot-Savart summation for vortex rings.

Brochu et al. [8] captured vorticity with a Lagrangian mesh, including its generation from buoyancy, and used FMM for linear-time Biot-Savart summation. Pfaff et al. [46] used the same vortex sheet representation to generate and evolve small scale details, but accelerated summation with a truncated kernel, relying on a coarse grid simulation to account for global effects. Pfaff et al.'s hybrid use of a grid and nearfield vortex summation is reminiscent of PPPM, but lacks the error correction stage we develop, and suffers from the numerical dissipation of the pressure projection on the grid. Both Brochu et al. and Pfaff et al. also pay a high cost in mesh operations to maintain the deforming vortex sheet, which we avoid by tracking density and calculating buoyancy with unconnected particles.

Despite their attraction for turbulence, vortex methods have not been widely aopted in the graphics industry, perhaps because of the difficulty in accelerating Biot-Savart summation, or boundary condition challenges, or the intricacies of vortex shedding. Addressing these problem motivates this paper.

Many graphics applications outside fluids also solve the Poisson problem. In Poisson surface reconstruction [30], the solution is important only around a narrow band of the input, but the associated Poisson problem is most naturally posed on an infinite domain, making an awkward fit for traditional voxel approaches.

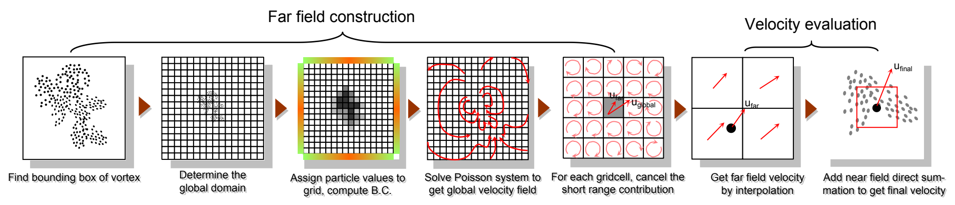
Gumerov and Duraiswami [24] demonstrated a high-performance GPU implement of FMM, reporting 1.4s to sum $N = 10^6$ particles on an NVIDIA GeForce 8800 GTX. Our PPPM code runs the same problem on an NVIDIA GeForce GT 650M (laptop) in 0.7s. More subjectively, our experience is that implementing a fast PPPM is vastly easier than even basic FMM.

We use multigrid as our fast grid solver as have many other graphics papers (e.g. [18, 40, 41]), though we should note that Henderson suggests on shared memory multiprocessors FFT still may be faster [26].

4.3 Particle-particle particle-mesh method(PPPM)

In this section we take Biot-Savart summation as an example to explain the PPPM algorithm in detail. See figure 4.2 for an overview.

Given the vorticity $\omega = \nabla \times u$ of an incompressible flow, and ignoring boundary conditions for now, we can reconstruct velocity (up to addition of the gradient of



Far field construction
Velocity evaluation
Find bounding box of vortex
Determine the global domain
Assign particle values to grid, compute B.C.
Solve Poisson system to get global velocity field
For each gridcell, cancel the short range contribution
u_far
u_global
Get far field velocity by interpolation
u_far
Add near field direct summation to get final velocity
u_final

Figure 4.2: An overview of our PPPM algorithm, which consists of a far-field construction step and a velocity evaluation step.

a harmonic scalar potential) by solving for a streamfunction Ψ whose curl is the velocity:

$$\begin{cases} \nabla^2 \Psi = -\omega \\ \Psi(x) = 0 \qquad x \to \infty \end{cases} \tag{4.4}$$

In the case where the right hand side of this Poisson system is given by a sum of n vortex elements, each carrying a vortex strength of ω_i, Ψ can be found by summation (c.f. equation 4.2). The velocity is just the curl of this sum (equation 4.3), further expressed as

$$u(x_i) = \frac{1}{4\pi} \sum_{j=1, j\neq i}^{n} \frac{\omega_j \times (x_i - x_j)}{\left|x_i - x_j\right|^3}, \tag{4.5}$$

which is known as Biot-Savart summation.

When evaluating at a specific point, we decompose this velocity as $u = u_{far} + u_{near}$, where u_{near} is the contribution from nearby vortices and u_{far} gathers influences from the rest.

Due to the singularity of the kernel, u_{near} varies rapidly in space and is related to the small scale motion (turbulence), while u_{far} is smooth and captures the large scale motion.

Realizing that multigrid Poisson solvers are excellent for smooth, large scale motions, while particle-particle direct summations are promising for the turbulent small scale motions, PPPM first estimates u_{far} for each grid cell using a particle-mesh solver (§4.3.1) and local-cancellation (§4.3.2). After that, PPPM interpolates

u_{far} for each vortex element and evaluates u_{near} using direct summation. Details of the computational steps in the PPPM Biot-Savart summation are outlined in Algorithm 5, where the subscript p indicates the quantity is carried by a particle, and the subscript g indicates the quantity is defined on a grid.

Algorithm 5 PPPMBiotSavartSummation(ω_p, v_p, x_p, N, u_p)

```
1: BB←GetBoundingBox($x_p$)
2: DetermineComputationDomain //3×BB
3: $\omega_g$ ←ParticleToMesh($\omega_p$)
4: ComputeMonopoleDirichletBoundaryCondition
5: $\Psi$ ←ParallelMultigridSolvePoisson
6: $u_g$ ←Curl($\Psi$)
7: for each grid cell in parallel
8:     Subtract near field estimate to get far field component
9: endfor
10: for each particle $i$ in parallel
11:     $u_{p,i} \leftarrow$ InterpolateFarField
12: endfor
13: for each particle $i$ in parallel
14:     Sum over particles in neighborhood $\eta(i)$ of $i$:
            (evaluating an accurate near field component)
15:     $\Delta u \leftarrow \sum_{j\in\eta(i)} \frac{\omega_{p,j}\times(x_{p,i}-x_{p,j})}{4\pi|x_{p,i}-x_{p,j}|^3}\left(1-\exp\left(\frac{|x-x_j|}{\alpha}\right)^3\right)$
16:     $u_{p,i} \leftarrow u_{p,i}+\Delta u$
17: endfor
```

4.3.1 Particle-mesh step in open space

In the particle-mesh step, we determine the large scale flow motion by solving a Poisson equation with Dirichlet condition(on domain boundary):

$$\begin{cases} \nabla^2\Psi = -\omega & \text{in } \Omega \\ \Psi = g & \text{on } \partial\Omega \end{cases} \tag{4.6}$$

The imposition of an artificial boundary at the edge of the grid is necessary for the solve, but makes approximation of the open space (infinite domain) problem tricky. We propose two special advances in our discretization to gain higher accuracy for the large scale motion in open space, compared to prior PPPM methods which use periodic boundaries or a homogenous Dirichlet condition. First, we use a computational domain that is three times as large as the bounding box of the vortex particles, giving us an effective padding region: the artificial boundary is distant from all sources in the problem. In open space the true solution Ψ at any evaluation point x_b on the boundary is exactly

$$\Psi\left(x_b\right) = \sum_{i=1}^{N} \frac{\omega\left(x_i\right) h^3}{4\pi \left|x_i - x_b\right|}. \tag{4.7}$$

Computing these values to be used as the Dirichlet boundary condition g would give, up to truncation error, the exact open space solution. However evaluating equation 4.7 directly is too costly: instead, we compute the cheap monopole approximation of the particle quantities, replacing equation 4.7 with

$$\Psi\left(x_b\right) = \sum_{i=1}^{N} \frac{\omega\left(x_i\right) h^3}{4\pi \left|x_i - x_b\right|} \approx \frac{\sum_{i=1}^{N} \omega\left(x_i\right) h^3}{4\pi \left|x_c - x_b\right|} = \frac{m\left(x_c\right)}{4\pi \left|x_c - x_b\right|} \tag{4.8}$$

where x_c is the center of all the vortex particles. The monopole is accurate when all the vortex particles are far away from the domain boundary, which we guarantee by our domain construction. We dub this the "monopole boundary condition" (line 4 in Algorithm 5), which experimentally we found doubles the accuracy of PPPM.

After we set up the computation domain and construct the boundary condition, the particle values are splatted to the grid using

$$\omega\left(X\right) = \frac{1}{h^3} \sum_i v_i \omega\left(x_i\right) \left(\prod_{\theta=1,2,3} w_\theta \left(\frac{X_\theta - x_{i,\theta}}{h} \right) \right) \tag{4.9}$$

where subscript $\theta = 1,2,3$ indicates the corresponding component of a 3D vector, h the size of the grid cell, v the volume of a vortex blob, i the index of a vortex blob, and w the splat kernel.

For the choice of w, we simply use nearest grid point (NGP) assignment, meaning each grid cell gathers the total strength of the vortex particles inside it:

$$w_\theta\left(r\right) = \begin{cases} 1, & r \in [-0.5, 0.5), \\ 0, & \text{else}, \end{cases} \tag{4.10}$$

In our implementation, we accelerate the process by parallelizing for each grid cell to gather vortex values around it, using spatial hashing [59] for efficient neighbour finding, following Green's implementation [22].

We then discretize the vector Poisson system using the seven-point second order finite difference scheme, arriving at three scalar Poisson systems, one for each component. We solve the resulting linear system for the streamfunction using multigrid, following Cohen et al.'s implementation [10].

Once we have found the streamfunction, we take its curl to get a velocity field u_{pm}. u_{pm} is then used to derive the far field approximation for each grid cell, using the procedure in the next section.

4.3.2 Cancelling local influences in the grid

To cancel the (relatively inaccurate) local contributions from nearby grid cells, we need to estimate a local stream function Ψ_L that solves

$$\nabla^2 \Psi_L = -\omega_L \tag{4.11}$$

in open space, where ω_L is the near-field vorticity only from the nearby grid cells.

Conceptually, we can achieve this by putting ω_L back in the global domain(grid), and solving the Poisson system again. (However, as we need a different estimate for each grid cell containing particles, in practice this would be far too expensive.)

More formally, putting ω_L back to the global domain is a prolongation, $\mathscr{P}\omega_L$. The global Poisson solve can be denoted as $\Psi = \mathscr{L}^{-1}\mathscr{P}\omega_L$. Reading back Ψ_L is a restriction of Ψ, $\mathscr{R}\left(\mathscr{L}^{-1}\mathscr{P}\omega_L\right)$. Finally, the local velocity field due to ω_L is readily $\mathscr{R}\left(\mathscr{L}^{-1}\mathscr{P}\omega_L\right)$.

The accuracy and efficiency of local cancellation depends on fast localized solutions for Ψ_L, say, expressed as a linear operator $\mathbb{A}$:

$$\Psi_L = \mathbb{A}h^2\omega_L, \tag{4.12}$$

with $\mathbb{A} \approx \frac{1}{h^2}\mathscr{R}\mathscr{L}^{-1}\mathscr{P}$.

Theuns [60] approximated $\mathbb{A}$ with an open space Green's function. however, this is quite different from the inverse of the grid-based operator, as the Green's function is singular at $r \to 0$ where the grid-based inverse is bounded; this renders it unable to evaluate the local contribution made by a grid value to itself. Walther [62] proposed a more accurate estimate at the cost of a pre-computation stage: with a local correction window of size K, the method stores a $\mathscr{O}\left(K^6\right)$ matrix $\mathbb{A}$ which solves $\Psi_L = \mathbb{A}\omega_L$. This computation is coupled to the grid resolution, restricting the simulation to static grids.

We instead make entries of $\mathbb{A}$ dimensionless quantities, allowing us to adapt the computation domain with more flexibility. We uncover a more accurate relationship between $\mathbb{A}$ and the open space Green's function. For instance, with the Green's function approach, one has

$$\Psi_L = \mathbb{G}h^2\omega_L \tag{4.13}$$

where $\mathbb{G}_{i,j} = \frac{h}{4\pi\left|X_i - X_j\right|}$ for $i \neq j$. (Observe that $\left|X_i - X_j\right|$ is of order h, making $\mathbb{G}_{i,j}$ dimensionless.)

Our observation is that the off-diagonal coefficients $\mathbb{A}_{i,j}$ are very close to $\mathbb{G}_{i,j}$, the open space Green's function evaluated at different grid cell centres. For diagonal terms, instead of being 0, we find $\mathbb{A}_{i,i} \to 0.25$ for large domains. This diagonal constant responds to the contribution made by a grid cell to itself.

Furthermore, if we replace the diagonal of $\mathbb{G}$ with this constant and compute its inverse, the seven-point finite difference stencil is essentially revealed. While we do not yet have a full derivation, we provide our evidence in figure 4.3. These numerical findings give us a new formulation to compute Ψ_L from ω_L. We outline this procedure in Algorithm 6.

After we obtain Ψ_L, we can proceed to compute u_L by taking the finite difference curl of Ψ_L

$$u_L = \nabla_h \times \Psi_L \tag{4.14}$$

Following this naively requires each grid cell to have a small buffer for the spatial varying function Ψ_L, making parallelizing impractical for GPU's memory limitation. Instead, we return to the construction of Ψ_L, first looking at off-diagonal contributions:

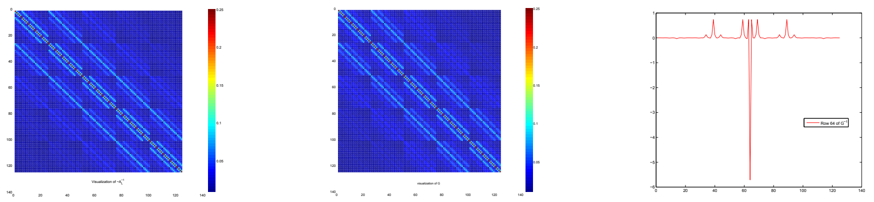


Figure 4.3: Relationship between inverse of finite difference operator and Green's function. Left: the local inverse matrix defined by $\mathbb{A}$. Middle: the local inverse matrix $\mathbb{G}$ constructed using Green's function and the diagonal terms from $\mathbb{A}$. Right: one row of the inverse of $\mathbb{G}$, almost revealed the 7-point stencil.

Algorithm 6 Local_inverse(ω_L)

```
1: $\Psi_L \leftarrow 0$
2: for $i,j,k \in L$ //$L = \{i_2, j_2, k_2 | i_2 \in [i-K, i+K], j_2 \in [j-K, j+K], k_2 \in [k-K, k+K]\}$
3:     $X_1 \leftarrow X_{i,j,k}$
4:     if $i_2, j_2, k_2 \in L$, $i_2, j_2, k_2 \neq i, j, k$
5:         $X_2 \leftarrow X_{i_2,j_2,k_2}$
6:         $\Psi_L(X_1) \leftarrow \Psi_L(X_1) + \frac{h^3 \omega_L(X_2)}{4\pi|X_1 - X_2|}$
           //using Green's function
7:
8:     else //$i_2, j_2, k_2 = i, j, k$
9:         $\Psi_L(X_1) \leftarrow \Psi_L(X_1) + 0.25h^2 \omega_L(X_2)$//notice $X_2 = X_1$ where, $0.25h^2$ is the diagonal constant
10:    endelse
11: endfor
```

$$\begin{aligned}\nabla_{X,h} \times \Psi_L = \nabla_{X,h} \times \sum_{X' \neq X} \frac{h^3 \omega_L(X')}{4\pi|X - X'|} \\ = \sum_{X \neq X'} h^3 \omega_L(X') \times \left(-\nabla_{X,h} \frac{1}{4\pi|X - X'|}\right) \qquad (4.15)\end{aligned}$$

where $\nabla_{X,h} \frac{1}{4\pi|X - X_j|}$ denotes a discrete gradient operator to the Green's function at X. Let e_1 be the unit vector $(1,0,0)$; we have

$$\left(\frac{\partial}{\partial x}\right)_{X,h}\frac{1}{4\pi|X-X'|} = \frac{1}{2h}\left(\frac{1}{4\pi|X+he_1-X'|}-\frac{1}{4\pi|X-he_1-X'|}\right). \tag{4.16}$$

When $X \pm he_k - X' = 0$, $\frac{1}{4\pi|X+he_k-X'|}$ is set to be 0.

We then compute a streamfunction buffer from the diagonal contributions,

$$\Psi_d = 0.25h^2\omega_g \tag{4.17}$$

We then take the discrete curl of Ψ_d to get u_d. The velocity introduced by the near field grid is consequently obtained via

$$u_L(X) = \sum_{j\neq i} h^3\omega_L(X') \times \left(-\nabla_{X,h}\frac{1}{4\pi|X-X'|}\right) + u_d \tag{4.18}$$

The new procedure overcomes the memory overflow issues with the naive approach, is a lot faster and more scalable, while giving exact same output (up to round-off) as the direct implementation.

4.3.3 Velocity evaluation

The far field influence at any given grid position X can be readily obtained via:

$$u_{far}(X) = u_{pm}(X) - u_L(X) \tag{4.19}$$

where u_L is determined by equation 4.18 in §4.3.2.

To evaluate the velocity for an arbitrary evaluation position x_i, we first interpolate from the far field buffer using trilinear interpolation,

$$u_{far}(x) \leftarrow \text{TriInterpolate}(u_{far}, x) \tag{4.20}$$

and then select all the nearby vortex elements within a cube C_L of size $(2K+1)h$ centred at x for near-field direct summation.

$$u(x) = u_{far} + \sum_{x_j \in C_L, x_j \neq x} \frac{\omega_j \times (x - x_j)}{4\pi |x - x_j|^3} \left(1 - \exp\left(\frac{|x - x_j|}{\alpha}\right)^3\right) \quad (4.21)$$

4.3.4 PPPM discussion

For analysis, we assume a reasonably uniform distribution of vortex particles in the computational domain. In the case where vortices were initialized on a surface sheet, the turbulent motion quickly breaks this sheet into many small blobs to make the vorticity distribution roughly uniform.

We divide the domain into equal sized grid cells so that each cell contains s particles on average, hence the number of cells is proportional to $\frac{N}{s}$. Transforming the particle quantities to the grid takes $\mathscr{O}(N)$ time; applying the monopole boundary condition takes $\mathscr{O}\left(N + \left(\frac{N}{s}\right)^{2/3}\right)$ operations; the multigrid Poisson solver with $\frac{N}{s}$ unknowns takes $\mathscr{O}\left(\frac{N}{s}\right)$ operations to get a solution; computing the discrete curl or gradient of the field requires also $\mathscr{O}\left(\frac{N}{s}\right)$ operations; interpolating from the grid quantity back to particles takes $\mathscr{O}(N)$ operations; and finally direct summation with nearby particles takes $\mathscr{O}(N)$ time.

To evaluate the velocity at some other m points, the runtime analysis is similar, only replacing the interpolating and local correction procedures in the aforementioned pipeline, hence we end up with $\mathscr{O}(m+n)$ complexity as claimed.

We tested the PPPM summation code for a random distribution of vortex particles, comparing the PPPM results with direct summation, and looked at the error in a weighted average manner (particles with a vanishing velocity get small weights),

As we can see from figure 4.5, the PPPM solution gets closer to direct summation when larger local correction (LC) windows are used. With a window size of $K = 3$, we obtained an error under 1%. Furthermore, since we bound the number of particles per cell, with 16384 vortex particles the cell size h is half the size used

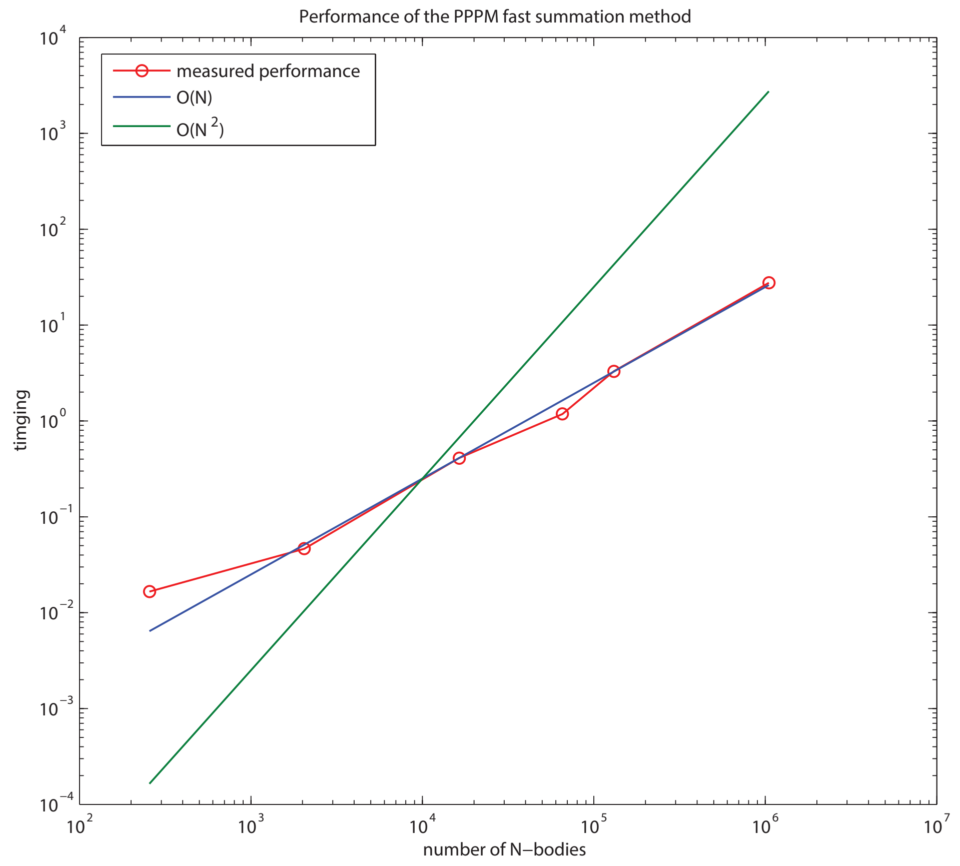


Figure 4.4: Performance of the PPPM fast summation. Computation time grows linearly with the number of computational elements.

for 2048 vortex particles. Therefore for the same K we have smaller physical LC radius in the 16384 case, but the accuracy doesn't decrease because one has better grid resolution to resolve the near field physics.

The performance of the PPPM summation is shown in figure 4.4. The computation time in our experiment grows linearly with respect to the number of vortex particles.

In table 4.2 we use direct summation as the reference solution to measure the accuracy of different approximations. We compare between our Monopole B.C. (MBC) and prior work's homogeneous B.C. We can see clearly that the MBC

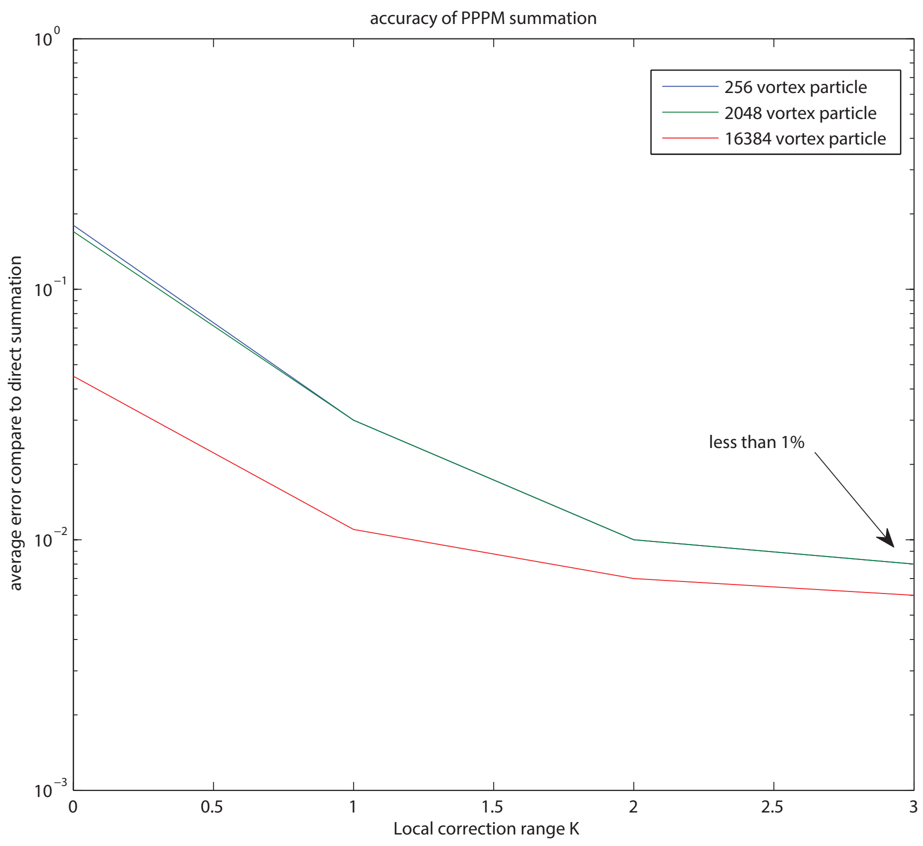


Figure 4.5: Accuracy statistics of the PPPM fast summation.

enhances the approximation in either case, while our PPPM method, being able to cancel the grid cell self-influence, gives approximation an order of magnitude higher than Theuns [60]. On the other hand, the particle-mesh method alone, even at higher cost, gives a poor approximation to direct summation.

Table 4.2: Accuracy of different method with and without the monopole B.C.

Method	K	w/ MBC	w/o MBC
PPPM 64^3	3	0.46%	1.13%
PM 128^3	N/A	6.19%	6.38%
PPPM 64^3 w/o diagonal cancellation [60]	3	2.78%	2.86%

4.4 Vortex-Particle Smoke

In vortex methods, given the vorticity distribution and boundary geometry, the velocity at any position x can be found as [11]

$$u(x) = u_\omega + u_\phi + u_\infty \tag{4.22}$$

which is a combination of a rotational flow u_ω from the vorticity, a potential flow u_ϕ determined by the solid objects §4.4.2, and a prescribed velocity field u_∞ defined by an artist (it is better for u_∞ to be divergence-free, otherwise, we suggest using $\nabla \times u_\infty$ as a force field).

Ignoring viscosity, the dynamics of vorticity can be modeled from the curl of the inviscid momentum equation,

$$\frac{D\omega}{Dt} = (\omega \cdot \nabla)u + \beta \nabla \rho \times g. \tag{4.23}$$

A vortex blob moves according to the combined velocity u of equation. 4.22, with vortex stretching($(\omega \cdot \nabla)u$) due to flow deformation in 3D and baroclinic vorticity generation $\beta \nabla \rho \times g$ concentrated on the density interface.

In the presence of solid objects, an irrotational divergence-free flow field u_ϕ can be found to cancel flow penetration at solid boundaries. Apart from modelling solid objects as boundary conditions in the vorticity solver, solid objects also can serve as a single layer source of “charge distribution” that introduces a harmonic potential whose gradient removes boundary penetration. More details about how to determine and use this "electrostatic" field can be found in §4.4.2.

We discretize the flow using a set of vortex blobs, with position x_i, volume v_i, density ρ_i, velocity u_i, and vortex density ω_i. At each time step, we compute u_ω

with Biot-Savart,

$$u_\omega(x) = \sum_{\text{all } j, j\neq i} \frac{v_j \omega_j \times (x - x_j)}{4\pi |x - x_j|^3} \left(1 - \exp\left(\frac{|x - x_j|}{\alpha}\right)^3\right), \quad (4.24)$$

using PPPM fast summation. Notice we mollify the Green's function kernel to desingularize the summation, a common necessity for vortex particle codes.

Subdividing the surface of solid objects into many small panels, the potential part of velocity, u_ϕ, can be determined by

$$u_\phi(x) = \sum_{all\, j, j\neq i} \frac{A_j \sigma_j (x_j - x)}{4\pi |x - x_j|^3} \left(1 - \exp\left(\frac{|x - x_j|}{\alpha}\right)^3\right) \quad (4.25)$$

where A_j is the area of j^{th} panel, σ_j is the potential source strength per unit area found on the single layer, and x_j is the centroid of the j^{th} panel.

At each time step, the strengths and positions of vortex blobs are updated by:

1. Initialize a vortex segment for each vortex blob based on the input vortex strength. §4.4.1
2. Update the position of each end of the vortex segment with the velocity determined by equation 4.22, hence, the vortex segment is stretched automatically. §4.4.1
3. Reduce instabilities introduced by vortex stretching.§4.4.1
4. Add baroclinic vorticity generation to the vorticity field.
5. Add vorticity due to vortex shedding.4.4.2

4.4.1 Vortex segment, vortex stretching and stability issue

This section describes the way to model vortex stretching using vortex segments. Vortex stretching introduces instability in some situations, which we address with

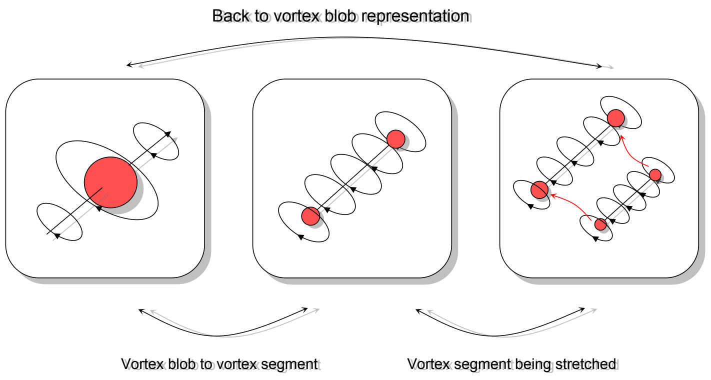

Back to vortex blob representation
Vortex blob to vortex segment
Vortex segment being stretched

Figure 4.6: We switch to a vortex segment representation of vortex particles at the beginning of each time step, move both ends of the vortex segment in the flow, then switch back to vortex blobs.

a geometry-inspired filtering approach.

Vortex segments

Vortex stretching is the rate-of-change of vorticity due to the deformation of fluid. This step is naturally handled by vortex ring or sheet representations: the stretching of the geometric elements automatically captures the vortex stretching term. This does not extent to vortex particles with no geometric extent. An obvious solution could be updating the vortex strength using

$$\omega^{n+1} = \omega^n + \omega^n \cdot \nabla u^n \tag{4.26}$$

It is unwise to take this approach because evaluating the gradient of velocity requires the second derivative of the Green's function. The singularity of this function introduces large numerical instabilities. Furthermore, computing the dot product is expensive.

We instead use a vortex-segment approach. A vortex segment is a small spinning cylinder whose central axis is aligned with the vorticity direction, with two ends x_a and x_b, length h, and constant circulation κ. A vortex blob with vorticity strength $v_i\omega_i$ can be translated to a vortex segment of length h_i, with unit direction $\hat{t}_i = \frac{\omega_i}{\sqrt{\|\omega_i\|^2}}$, and circulation $\kappa_i = \omega_i / (h_i \hat{t}_i)$. We translate our vortex blob into such

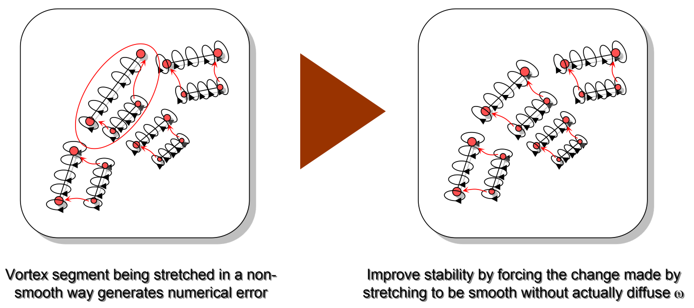

Vortex segment being stretched in a non-smooth way generates numerical error
Improve stability by forcing the change made by stretching to be smooth without actually diffuse ω

Figure 4.7: Sudden discontinuous motion of vortex segments introduces and amplifies numerical error, which is reduced then by smoothing the stretching terms.

vortex segments at the beginning of each time step, evaluate the velocity according to equation 4.22 for each end of the vortex segment, and move each end of the vortex segment according to

$$\begin{aligned} x_{i,a}^{n+1} &= x_{i,a}^{n} + \Delta t u_{i,a}^{n} \\ x_{i,b}^{n+1} &= x_{i,b}^{n} + \Delta t u_{i,b}^{n}. \end{aligned} \tag{4.27}$$

When both ends of the vortex segment are updated, the vortex segment is stretched and we transform it back to a vortex blob with vortex strength,

$$v_i \omega_i = \kappa_i \left(x_{i,b}^{n+1} - x_{i,a}^{n+1} \right). \tag{4.28}$$

Notice the circulation is conserved. This whole process is illustrated in figure 4.6.

Stability issues

Vortex stretching is potentially unstable. Without addressing this sensitivity, our simulation quickly diverges even with small time-steps. We noticed that in turbulent flow, even when the magnitude of the velocity is small, the gradient of the velocity field can be very large, which makes $\omega \cdot \nabla u$ large and drives the numerical instabilities. To address this problem, we realized that in the physical world, the rate of change of vorticity due to vortex stretching has some smoothness. We therefore compute the rate of change of vorticity due to stretching by

$$(\omega \cdot \nabla u)_i^n = \frac{\kappa_i \left(x_{i,b}^{n+1} - x_{i,a}^{n+1}\right) - \omega^n}{\Delta t} \tag{4.29}$$

and then apply a Gaussian filter to smooth $(\omega \cdot \nabla u)^n$ in the domain to get $(\omega \cdot \tilde{\nabla} u)_i^n$ for each particle. Vorticity is updated by

$$\omega_i^{n+1} = \omega_i^n + \Delta t (\omega \cdot \tilde{\nabla} u)_i^n. \tag{4.30}$$

This approach is effectively a geometric repairing that forces the deformation of a vortex segment to be consistent with nearby vortex segments, as illustrated in figure 4.7. With this approach, we stabilized the simulation and reliably achieved long simulations when using a constant large time step. Notice that in this scheme, we smooth the update instead of the quantity itself to preserve sharp features as much as possible, similar in spirit to FLIP [68].

4.4.2 Solid boundaries and vortex shedding

We incorporate boundary conditions by introducing an irrotational, divergence free flow field u_ϕ that cancels the normal velocity flux on the boundary made by $u_\omega + u_\infty$. We define u_ϕ as the gradient of a scalar potential function Φ, solved with boundary integral equations. After enforcing the no-through boundary condition, we compute a vortex sheet on the surface heuristically [9], and merge this surface vorticity into the vorticity field.

No-through boundary condition

The no-through boundary condition can be enforced by solving for a scalar potential field that satisfies Laplace's equation with Neumann boundary condition[8]

$$\begin{aligned} \Delta\Phi &= 0 \\ \frac{\partial \Phi}{\partial n} &= \left(u_{solid} - u_{fluid}\right) \cdot n \end{aligned} \tag{4.31}$$

We write this function as a single layer potential, with a continuous source distribution σ on the solid boundary,

$$\Phi(x) = \int\limits_{\partial\Omega} \frac{\sigma(y)}{4\pi|x-y|} dS(y). \tag{4.32}$$

Taking the normal derivative of Φ and substituting it into equation 4.31 gives a Fredholm equation of the second kind,

$$b(x) = -\frac{\sigma(x)}{2} + \int\limits_{\partial\Omega} \sigma(y) \frac{\partial}{\partial n_x} \frac{1}{4\pi|x-y|} dS(y) \tag{4.33}$$

where $f(x) = \left(u_{solid}(x) - u_{fluid}(x)\right) \cdot n(x)$.

We discretize this equation on a set of M boundary elements using mid-point rule to arrive at

$$b_i = -\frac{\sigma_i}{2} + \sum_{all\, j} \frac{\partial}{\partial n_i} \frac{\sigma_j A_j}{4\pi\left|x_i - x_j\right|} \tag{4.34}$$

where A_j is the area of j'th surface element, and x_i and x_j are the mid-points of corresponding boundary elements.

In practise, this equation gives a linear system for σ that is very well-conditioned: iterative solvers like BiCGSTAB converge in $\mathscr{O}(1)$ iterations. However, the $M \times M$ coefficient matrix is dense and evaluating the matrix vector product is M^2. To overcome this challenge, we reformulate equation 4.34 using the PPPM framework. Given a source distribution σ, with proper reordering, the off-diagonal part summation is exactly

$$b_i^{off-diag} = n_i \cdot \left(\sum_{all\, j \neq i} \frac{\sigma_j A_j (x_j - x_i)}{4\pi\left|x_i - x_j\right|^3} \right) \tag{4.35}$$

which takes the same form of evaluating a gravitational force based on mass particles, where, the “mass” of the particle here are defined as $\sigma_j A_j$. Hence our PPPM-accelerated evaluation of the matrix vector multiplication directly follows the routines described in Algorithm 7.

PPPM is used in two different places here. During the iteration, we use PPPM to compute the force based on the current estimate (line 7 of algorithm 7), and when the iteration is terminated we use PPPM to compute u_ϕ based on the single

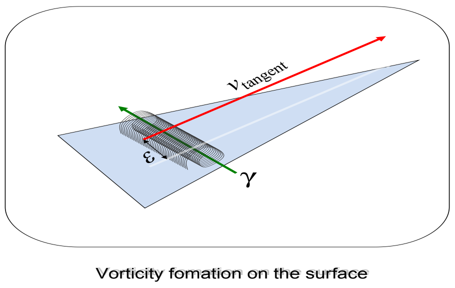


Figure 4.8: Vortex shedding process. In our approach, the vorticity strength is explicitly determined.

layer density.

Algorithm 7 PPPM_accelerated_Ax(b, σ, p, A, n, M)

```
1:  $f \leftarrow 0$
2:  $b \leftarrow 0$
3:  $m \leftarrow 0$
4:  for i=1:M in parallel
5:      $m_i = A_i * \sigma_i$
6:  endfor
7:  $f \leftarrow$ PPPM_Compute_Gravity($m$, $p$, $M$)
8:  for i=1:M in parallel
9:      $b_i \leftarrow n_i \cdot f_i - 0.5 * \sigma_i$
10: endfor
```

Vortex shedding

The inviscid assumption breaks down near boundaries, because it never introduces vorticity into the flow field. In reality fluid viscosity, no matter how small, generates vorticity concentrated along a thin boundary layer along solid surfaces. In high Reynolds number flows, this thin viscous boundary layer doesn't affect the validity of inviscid approximation being made elsewhere in the flow, but rather serves as a source emitting vorticity into the flow.

Chorin [9] modelled this process heuristically. in 2D. He assumed a constant vorticity strength on the boundary element and determined the vortex strength based on the tangential velocity slip. Extending this idea to 3D, we determine the

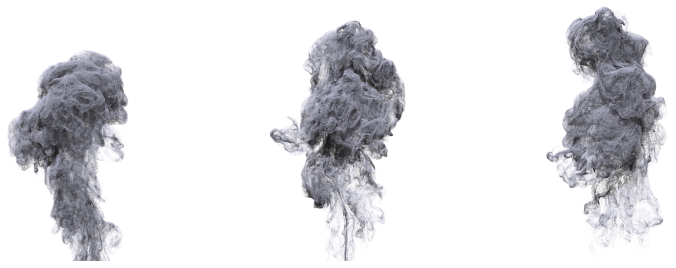


Figure 4.9: Rising smoke using different number of vortex particles. Left, 2049 particles: middle, 16384 particles; right, 130K particles.

surface vorticity direction based on the tangential velocity and the surface normal: vorticity is required to be perpendicular to these two vectors, its strength is determined so that at a position in the normal direction, ε away from the surface, the velocity introduced by this vorticity matches the tangential slip. In other words, if we put a vortex of strength $A_j\gamma_j$ at the position ε away from the surface, the velocity it introduces cancels the tangential slip at the boundary. This process is illustrated in figure 4.8.

Once the concentrated vorticity strength is determined, we release it to the flow field by adding some amount of its value to the nearest vortex blob around the surface or release them to a position at a small distance away from j'th boundary element. The amount to be released is determined as

$$\Delta\omega = \Delta t c A_j \gamma_j \tag{4.36}$$

4.5 Results and conclusions

4.5.1 PPPM for vortex flow

We implemented the PPPM algorithm on the GPU (nVidia GeForce GT 650m). As we can see in figure 4.4, the performance of PPPM scales linearly with the computational elements involved. On our machine, simulations are still running interactively with even 16K vortex particles on a single laptop GPU. We generate a preview simulation with a small amount of tracer particles (16K) at interactive

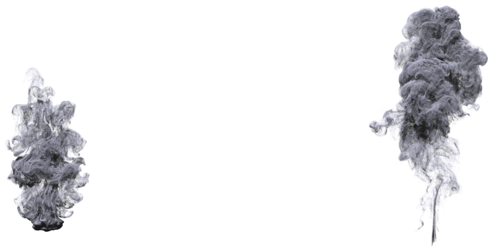


Figure 4.10: Without the far-field motion, direct summation using a cut-off kernel results in wrong animation. Left: simulation uses cut-off direct summation after 180 time steps. Right: simulation uses PPPM summation after 180 time steps.

Figure 4.11: Comparison of VIC and PPPM. Top row, sequence of 64^3 VIC simulation; bottom row: PPPM using the same resolution grid. Notice that the large scale motion of the two simulation matches before the point where turbulent motion becomes dominant.

rates, and then produce an enhanced result by just using more particle tracers (6M).

Figure 4.9 illustrates rich turbulent phenomena even with a small number of vortex particles.

To achieve our final results, we are not simply interpolating velocity to the tracer particles, but rather computing the velocity of each tracer particles with the full-influence Biot-Savart summation from vortex particles. It is only with fast summation that it is feasible to produce the results in figure 4.1, where 130K vortex particle and 6 million tracers were used. Each time step takes 100 sec to process on a laptop with nVidia's Geforce GT 650M graphics card.

We also observed that in our computation, local direct summation dominates the computation time. However, with only this near field direct summation (truncating the kernel to finite support), one obtains unrealistic animations. Figure 4.10, a cut-off direct summation uses the same cut-off range as the PPPM summation uses, takes almost same computation time as PPPM takes, but the smoke fails to rise.

On the other hand, the PPPM code without local correction is reduced to a VIC solver, which fails to produce small scale motions because of numerical smoothing. VIC running at higher resolution, to produce turbulent animations similar to PPPM, takes 64 times the memory cost and 10 times the simulation time every time step. Figure 4.11 shows representation frames.

Counter-intuitively solving for the tracer particle motion takes more time in a 256^3 VIC simulation, even though this is just interpolation, whereas in 64^3 PPPM there is a far-field interpolation followed by a more costly near field direct summation. We suspect this is because memory access with larger velocity buffer is less efficient.

Our no-stick boundary condition and vortex shedding model handles different boundary geometry robustly and produces visually plausible animations. We left the shedding coefficient as an artist controlled parameter. In the example shown in figure 4.12, we used a shedding coefficient $c = 0.6$, set the size of the moving object to one unit length, and let it move at a speed of 4 unit lengths/sec. The vortex shedding model is able to produce complex turbulent wake patterns.

4.5.2 Applying PPPM to Poisson surface reconstruction

Given a set of n sample points at position $\{x \,|\, x = x_i, i = 1,2,3\ldots n\}$, with normals $\hat{n}_i$ at x_i, the Poisson surface reconstruction [30] algorithm reconstructs the surface of the geometry in two steps:

1 Seek a scalar field ϕ whose gradient best matches the vector field V defined

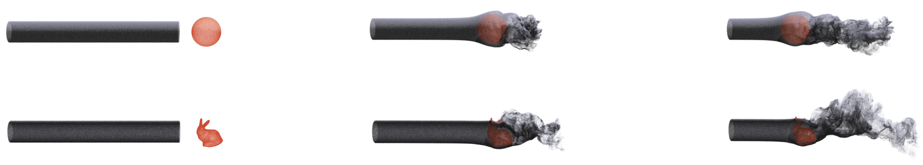


Figure 4.12: Moving objects in slightly viscous flow generate turbulent wakes. Top row: vortex shedding from a moving ball. Bottom row: a moving bunny.

by the input, i.e., find ϕ that solves

$$\nabla^2\phi = \nabla\cdot V \tag{4.37}$$

2 Determine a constant c s.t. the isosurface defined by $\phi(x) = c$ is a good match of the geometry to be reconstructed. Here, $c = \frac{1}{n}\sum_i \phi(x_i)$ (average of ϕ at input positions).

A detailed discussion of this algorithm is beyond the scope of this paper. Here we emphasize the computation. In the original paper, an adaptive Poisson solve on an octree was used. In our approach, we only need to define the right-hand-side on a narrow band of voxels near the input point clouds, and we can evaluate ϕ by summation.

More precisely, we obtain $\nabla\cdot V$ in the narrow band, then solve for ϕ with

$$\phi(x_i) = \sum_{j=1, j\neq i}^{n} \frac{h^3 f_j}{4\pi\|x_i - x_j\|} \tag{4.38}$$

here, $f_j = -(\nabla\cdot V)_j$ on voxel j, x_i and x_j the position of i'th and j'th voxel, respectively. Those voxels and f_j's are then viewed as particles with mass, allowing us to use PPPM to calculate the summation.

We tested PPPM surface reconstruction on a bunny and a dragon, shown in figure 4.13. While quality surface reconstruction is not the focus of this paper, we argue this shows the potential of PPPM for other branches of graphics. We are neither using super high resolution sparse voxels nor taking any effort in choosing

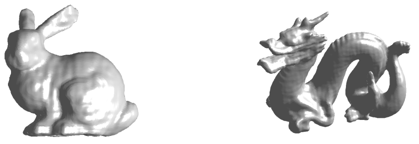


Figure 4.13: PPPM Poisson surface reconstruction of: left) a bunny, right) a dragon.

a good Gaussian kernel when splatting normals to construct the vector field V. For the dragon case we reconstructed, the voxel size is determined to be $\frac{1}{256}$ of the longest dimension. Since the computation involves only those sparse voxels, the summation approach became more suitable; it would be difficult for typical finite difference approaches to impose useful boundary conditions on the boundary cells of the narrow band.

4.6 Limitations

Unlike FMM, where one can get a desired accuracy by taking enough multipole expansions, in PPPM, further accuracy can not be achieved for local correction range greater than 7 grid cells. Not only is the local cancellation imperfect, but also, interpolating the far field has limits to its accuracy.

The proposed PPPM focuses only on the N-body problem with particles, or boundary elements that can be viewed as particles. Extending the PPPM summation framework to non-particles such as higher-order surface sheet or boundary elements should nonetheless be straightforward. One could switch to higher-order quadrature rules (or even exact integrals) for near field elements, or for each element generate samples based on quadrature rule, interpolate the value and scale it with the quadrature weights.

In FMM, one tracks adaptively distributed computational elements with adaptive data structure, whereas PPPM use uniform background grids and uniform space hash. This greatly simplifies the implementation and reduces computational overhead, but limits PPPM scalability for sparse data. Adaptive Poisson solvers [37] might address this problem.

4.7 Future work

Algorithmically, many improvements can be made to the proposed PPPM. higher order finite element solvers for the PM part and dipoles instead of monopole for ghost boundary would improve the accuracy further, potentially with a smaller grid making for even faster performance. For very large problems, with billions of particles, there may be interesting wrinkles in making a distributed version.

The PPPM philosophy could also be extended to higher-order boundary integrals, or diffusion kernels, potentially to accelerate applications in and outside fluids: preconditioning or posing sub-domain B.C. in large scale domain decomposition solvers, extending Eulerian simulation to open space, fast image and geometry processing techniques.

Chapter 5

Conclusion

Velocity-pressure and vorticity-velocity equations are different yet equivalent forms of the Navier-Stokes equations, and it is true that with adequately accurate numerical methods, no significant difference should be observed when solving either of the two forms. However, in practice we have observed that the vorticity-velocity approach is much more successful in *cheaply* resolving vorticity-rich phenomena, while the velocity-pressure formulation lends itself more easily to resolving boundary layers. Classifying and tracking physical phenomena with different characteristics using different formulations promises to be an efficient way to obtain solutions not possible by dealing with only one form.

5.1 Summary

In Chapter 2, a pure Eulerian solver was used to both resolve the boundary layer velocity field and the wake. In such flows what happens at the boundary dominates the dynamics off the boundary. The proposed method captures such flows accurately and cheaply.

In Chapter 3, we augment velocity-pressure solvers with a correction from the vorticity-velocity formulation. This easy-to-integrate procedure is able to enhance a wide variety of existing Eulerian solvers in computer graphics, enabling them to capture vortex structures with coarser grids and much larger time steps than possible with other known solvers.

In Chapter 4, a pure vorticity-velocity solver is demonstrated with a PPPM scheme to reduce the N-Body summation from $\mathcal{O}(N^2)$ to $\mathcal{O}(N)$. Besides the computational power of the PPPM method, its ease of implementation also stands out among existing sophisticated fast summation schemes.

5.2 Future work

The question remains if there are more convenient and efficient ways to bring the advantages of vorticity-velocity methods to the velocity-pressure schemes favoured in computer graphics. IVOCK demonstrates there is potential for such schemes, but was prone to stability problems near solid boundaries at high Reynolds number which weren't satisfactorily resolved.

One of the advantages of Lagrangian vortex methods is efficiently capturing flows in which vorticity is sparse, i.e. zero almost everywhere except on a lower dimensional set. This suggests that even for Eulerian velocity-pressure schemes, adaptive grids where vorticity drives grid refinement may be attractive.

A vorticity-velocity method could also be combined with a velocity-pressure scheme to resolve complex flows in a large scale, each handling the part of flow to which they are best suited. We have already achieved some results in this direction, as shown in Fig. 5.1. In this simulation, the vFLIP solver was used to handle complex flows near the object boundaries, which is cheaply and weakly coupled with a Vortex-In-Cell (VIC) solver for the off-boundary flows in open space. The VIC computation was constructed on sparse grid cells reflecting the sparse nature of vorticity, constructed as a collection of small fixed-sized cubic tiles of voxels, taken from a conceptually infinite grid but only instantiated where vorticity is present. This hybrid solver is by far the most efficient solver considered in the course of this thesis for capturing small-scale dynamics in a large domain, with minimum loss of detail.

Finally, while free surface boundary conditions are especially difficult to incorporate into vortex methods, compared to velocity-pressure schemes, there are many cases in water simulation where vorticity is concentrated only on the surface. Potential flow schemes in particular have been extremely successful for modeling moderate ocean waves, while falling short of being able to properly handle

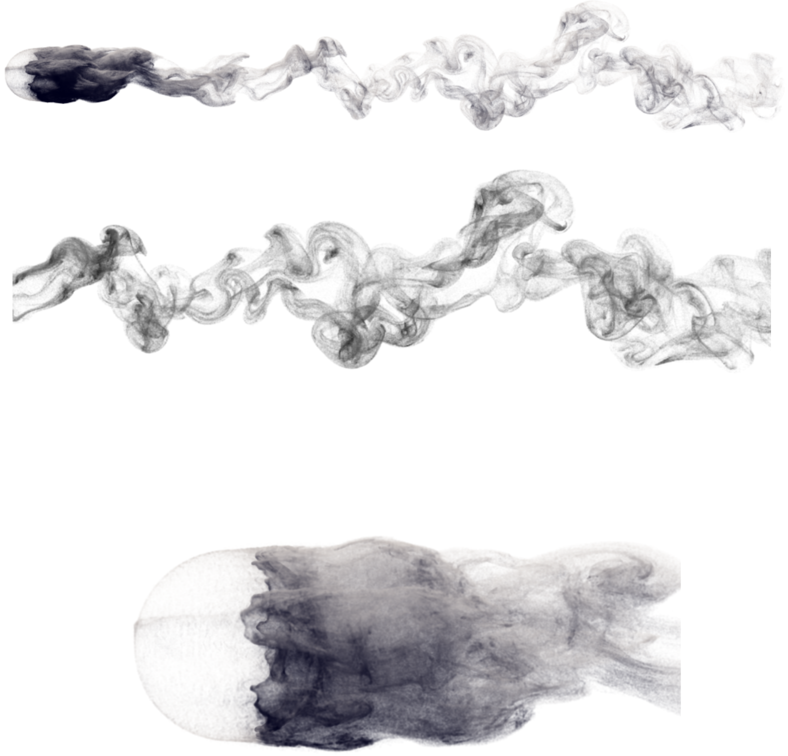


Figure 5.1: 3D simulation of flow past sphere at Re=8000. Top: one frame from the simulated result. Middle: zoom-in of the wake pattern. Bottom: near-boundary flow is captured accurately with the vFLIP solver and seamlessly coupled with the free-space solution.

overturning waves and more involved solid-interaction with splashing etc. We suspect hybrid solvers where a sparse vortex approach handles the deep water and a velocity-pressure method handles the free surface, may provide a solution.

Bibliography

[1] C. R. Anderson. A method of local corrections for computing the velocity field due to a distribution of vortex blobs. *J. Comp. Physics*, 62:111–123, 1986. → pages 47

[2] R. Ando, N. Thürey, and C. Wojtan. Highly adaptive liquid simulations on tetrahedral meshes. *ACM Trans. Graph. (Proc. SIGGRAPH 2013)*, July 2013. → pages 5

[3] R. Ando, N. Thürey, and C. Wojtan. A dimension-reduced pressure solver for liquid simulations. *EUROGRAPHICS 2015*, 2015. → pages 9

[4] A. Angelidis and F. Neyret. Simulation of smoke based on vortex filament primitives. In *Symposium on Computer Animation*, pages 87–96, 2005. → pages 48

[5] J. Barnes and P. Hut. A hierarchical O(N log N) force-calculation algorithm. *Nature*, 324:446–449, 1986. → pages x, 32

[6] C. Batty, F. Bertails, and R. Bridson. A fast variational framework for accurate solid-fluid coupling. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 26 (3):100, 2007. → pages 5, 8, 13, 39

[7] R. Bridson. *Fluid Simulation for Computer Graphics*. A K Peters / CRC Press, 2008. → pages 4, 24

[8] T. Brochu, T. Keeler, and R. Bridson. Linear-time smoke animation with vortex sheet meshes. In *Proc. ACM SIGGRAPH / Eurographics Symp. Comp. Animation*, pages 87–95, 2012. → pages 26, 27, 49, 64

[9] A. J. Chorin. Numerical study of slightly viscous flow. *Journal of Fluid Mechanics Digital Archive*, 57(04):785–796, 1973. → pages 5, 45, 64, 66

[10] J. M. Cohen, S. Tariq, and S. Green. Interactive fluid-particle simulation using translating Eulerian grids. In *ACM Symp. I3D*, pages 15–22, 2010. → pages 53

[11] G.-H. Cottet and P. Koumoutsakos. *Vortex methods - theory and practice*. Cambridge University Press, 2000. → pages 48, 60

[12] B. Couet, O. Buneman, and A. Leonard. Simulation of three-dimensional incompressible flows with a vortex-in-cell method. *Journal of Computational Physics*, 39(2):305 – 328, 1981. ISSN 0021-9991. → pages 2, 46

[13] E. Edwards and R. Bridson. Detailed water with coarse grids: combining surface meshes and adaptive discontinuous Galerkin. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 33(4):136:1–9, 2014. → pages 8, 25

[14] S. Elcott, Y. Tong, E. Kanso, P. Schröder, and M. Desbrun. Stable, circulation-preserving, simplicial fluids. *ACM Trans. Graph.*, 26(1), Jan. 2007. ISSN 0730-0301. → pages 42

[15] R. E. English, L. Qiu, Y. Yu, and R. Fedkiw. Chimera grids for water simulation. In *Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '13, pages 85–94, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2132-7. → pages 8

[16] R. Fedkiw, J. Stam, and H. W. Jensen. Visual simulation of smoke. In *Proc. ACM SIGGRAPH*, pages 15–22, 2001. → pages 9, 24, 27, 46

[17] B. E. Feldman, J. F. O'Brien, and O. Arikan. Animating suspended particle explosions. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 22(3):708–715, 2003. → pages 27, 42

[18] F. Ferstl, R. Westermann, and C. Dick. Large-scale liquid simulation on adaptive hexahedral grids. *Visualization and Computer Graphics, IEEE Transactions on*, PP(99):1–1, 2014. ISSN 1077-2626. → pages 9, 49

[19] N. Foster and R. Fedkiw. Practical animation of liquids. In *Proc. ACM SIGGRAPH*, pages 23–30, 2001. → pages 24, 27

[20] M. N. Gamito, P. F. Lopes, and M. R. Gomes. Two-dimensional simulation of gaseous phenomena using vortex particles. In *Computer Animation and Simulation '95*, Eurographics, pages 3–15. Springer Vienna, 1995. → pages 5

[21] A. Golas, R. Narain, J. Sewall, P. Krajcevski, P. Dubey, and M. Lin. Large-scale fluid simulation using velocity-vorticity domain decomposition. *ACM Transactions on Graphics (TOG)*, 31(6):148, 2012. → pages 5, 8

[22] S. Green. Cuda particles. *nVidia Whitepaper*, 2(3.2):1, 2008. → pages 53

[23] L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *J. Comp. Physics*, 73:325–348, 1987. → pages 46

[24] N. A. Gumerov and R. Duraiswami. Fast multipole methods on graphics processors. *Journal of Computational Physics*, 227:8290 – 8313, 2008/09/10/ 2008. → pages 49

[25] F. H. Harlow and J. E. Welch. Numerical calculation of time-dependent viscous incompressible flow of fluid with free surfaces. *Physics of fluids*, 8: 2182–2189, 1965. → pages 31

[26] R. D. Henderson. Scalable fluid simulation in linear time on shared memory multiprocessors. In *Proceedings of the Digital Production Symposium*, DigiPro '12, pages 43–52, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1649-1. → pages 49

[27] R. W. Hockney and J. W. Eastwood. *Computer Simulation Using Particles*. Taylor & Francis, Jan. 1989. ISBN 0852743920. → pages 45

[28] A. Jacobson, L. Kavan, , and O. Sorkine-Hornung. Robust inside-outside segmentation using generalized winding numbers. *ACM Trans. Graph.*, 32 (4):33:1–33:12, 2013. → pages 46

[29] C. Jiang, C. Schroeder, A. Selle, J. Teran, and A. Stomakhin. The affine particle-in-cell method. *ACM Trans. Graph.*, 34(4):51:1–51:10, July 2015. ISSN 0730-0301. → pages 8, 22

[30] M. M. Kazhdan, M. Bolitho, and H. Hoppe. Poisson surface reconstruction. In *Symp. Geometry Processing*, pages 61–70, 2006. → pages 46, 49, 69

[31] B. Kim, Y. Liu, I. Llamas, and J. Rossignac. FlowFixer: Using BFECC for fluid simulation. In *Proc. First Eurographics Conf. on Natural Phenomena*, NPH'05, pages 51–56, 2005. → pages 1, 23, 27

[32] T. Kim, N. Thurey, D. James, and M. H. Gross. Wavelet turbulence for fluid simulation. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 27(3):50, 2008. → pages 9, 29, 43

[33] M. Lentine, W. Zheng, and R. Fedkiw. A novel algorithm for incompressible flow using only a coarse grid projection. *ACM Trans. Graph.*, 29(4): 114:1–114:9, July 2010. ISSN 0730-0301. → pages 9

[34] M. Lentine, M. Aanjaneya, and R. Fedkiw. Mass and momentum conservation for fluid simulation. In *Proc. ACM SIGGRAPH / Eurographics Symp. Comp. Anim.*, pages 91–100, 2011. → pages 1, 25

[35] J. Lienhard. *Synopsis of Lift, Drag, and Vortex Frequency Data for Rigid Circular Cylinders*. Bulletin (Washington State University. College of Engineering. Research Division). Technical Extension Service, Washington State University, 1966. → pages ix, 17

[36] C. H. Liu and D. J. Doorly. Vortex particle-in-cell method for three-dimensional viscous unbounded flow computations. *International Journal for Numerical Methods in Fluids*, 32(1):23–42, 2000. ISSN 1097-0363. → pages 34

[37] F. Losasso, F. Gibou, and R. Fedkiw. Simulating water and smoke with an octree data structure. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 23(3): 457–462, 2004. → pages 5, 27, 71

[38] M. Macklin and M. Müller. Position based fluids. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 32(4):104, 2013. → pages 42

[39] S. Mas-Gallic. Particle approximation of a linear convection-diffusion problem with neumann boundary conditions. *SIAM J. Numer. Anal.*, 32(4): 1098–1125, Aug. 1995. ISSN 0036-1429. → pages 6

[40] A. McAdams, E. Sifakis, and J. Teran. A parallel multigrid poisson solver for fluids simulation on large grids. In *Proc. ACM SIGGRAPH / Eurographics Symp. Comp. Anim.*, pages 65–74, 2010. → pages 9, 14, 15, 27, 49

[41] J. Molemaker, J. M. Cohen, S. Patel, and J. Noh. Low viscosity flow simulations for animation. In *SCA '08: Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 9–18. Eurographics Association, 2008. → pages 8, 49

[42] P. Mullen, K. Crane, D. Pavlov, Y. Tong, and M. Desbrun. Energy-preserving integrators for fluid animation. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 28(3):38:1–38:8, 2009. → pages 25, 42

[43] D. Q. Nguyen, R. Fedkiw, and H. W. Jensen. Physically based modeling and animation of fire. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 21(3):721–728, 2002. → pages 24, 27

[44] S. I. Park and M.-J. Kim. Vortex fluid for gaseous phenomena. In *Proc. ACM SIGGRAPH / Eurographics Symp. Comp. Animation*, pages 261–270, 2005. → pages 2, 7, 26, 27, 46

[45] T. Pfaff, N. Thuerey, A. Selle, and M. Gross. Synthetic turbulence using artificial boundary layers. *ACM Trans. Graph.*, 28(5):121, 2009. → pages 7, 19, 43, 48

[46] T. Pfaff, N. Thuerey, and M. Gross. Lagrangian vortex sheets for animating fluids. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 31(4):112:1–8, 2012. → pages 27, 49

[47] A. Ralston. Runge-Kutta methods with minimum error bounds. *Mathematics of Computation*, 16(80):431–437, 1962. → pages 12

[48] H. Schechter and R. Bridson. Evolving sub-grid turbulence for smoke animation. In *Proceedings of the 2008 ACM/Eurographics Symposium on Computer Animation*, 2008. → pages 9

[49] A. Selle, N. Rasmussen, and R. Fedkiw. A vortex particle method for smoke, water and explosions. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 24 (3):910–914, 2005. → pages 28, 48

[50] A. Selle, R. Fedkiw, B. Kim, Y. Liu, and J. Rossignac. An Unconditionally Stable MacCormack Method. *J. Scientific Computing*, 35(2–3):350–371, 2008. → pages 1, 23, 25, 27

[51] R. Setaluri, M. Aanjaneya, S. Bauer, and E. Sifakis. SPGrid: A sparse paged grid structure applied to adaptive smoke simulation. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)*, 33(6):205:1–205:12, 2014. → pages 5, 14, 27

[52] O.-y. Song, D. Kim, and H.-S. Ko. Derivative particles for simulating detailed movements of fluids. *IEEE Trans. Vis. Comp. Graph.*, 13(4): 711–719, 2007. → pages 8

[53] J. Stam. Stable fluids. In *Proc. ACM SIGGRAPH*, pages 121–128, 1999. → pages 1, 23, 24, 26, 46

[54] J. Steinhoff and D. Underhill. Modification of the Euler equations for "vorticity confinement": Application to the computation of interacting vortex rings. *Physics of Fluids*, 6:2738–2744, 1994. → pages 9, 28

[55] M. J. Stock and A. Gharakhani. Toward efficient GPU-accelerated N-body simulations. *AIAA paper*, 608:7–10, 2008. → pages 2

[56] M. J. Stock and A. Gharakhani. A GPU-accelerated boundary element method and vortex particle method. In *AIAA 40th Fluid Dynamics Conference and Exhibit (July 2010)*, page 1, 2010. → pages 5, 7

[57] M. J. Stock, A. Gharakhani, and C. P. Stone. Modeling rotor wakes with a hybrid OVERFLOW-vortex method on a GPU cluster. In *28th AIAA Applied Aerodynamics Conference*, volume 20, 2010. → pages 2, 8

[58] E. G. Tabak. Vortex stretching in incompressible and compressible fluids. Courant Institute, Lecture Notes (Fluid Dynamics II), 2002. → pages 40

[59] M. Teschner, B. Heidelberger, M. Mueller, D. Pomeranets, and M. Gross. Optimized spatial hashing for collision detection of deformable objects. In *Proc. VMV*, pages 47–54, 2003. → pages 53

[60] T. Theuns. Parallel PPPM with exact calculation of short range forces. *Computer Physics Commun.*, 78(3):238 – 246, 1994. ISSN 0010-4655. → pages 53, 59, 60

[61] M. Van Dyke. *An album of fluid motion*. Parabolic Press, 1982. → pages 17

[62] J. H. Walther. An influence matrix particle-particle particle-mesh algorithm with exact particle-particle correction. *J. Comp. Physics*, 184:670–678, 2003. → pages 54

[63] H. Wang, K. A. Sidorov, P. Sandilands, and T. Komura. Harmonic parameterization by electrostatics. *ACM Trans. Graph.*, 32(5):155:1–155:12, Oct. 2013. ISSN 0730-0301. → pages 45

[64] S. Weißmann and U. Pinkall. Filament-based smoke with vortex shedding and variational reconnection. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 29 (4):115, 2010. → pages 26, 27, 48

[65] L. Yaeger, C. Upson, and R. Myers. Combining physical and visual simulation–creation of the planet Jupiter for the film "2010". *Proc. SIGGRAPH*, 20(4):85–93, Aug. 1986. ISSN 0097-8930. → pages 2, 26, 48

[66] X. Zhang and R. Bridson. A PPPM fast summation method for fluids and beyond. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)*, 33(6):206:1–11, 2014. → pages 5, 27

[67] X. Zhang, R. Bridson, and C. Greif. Restoring the missing vorticity in advection-projection fluid solvers. *ACM Trans. Graph.*, 34(4):52:1–52:8, July 2015. ISSN 0730-0301. → pages 22

[68] Y. Zhu and R. Bridson. Animating sand as a fluid. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 24(3):965–972, 2005. → pages 1, 5, 8, 23, 25, 27, 64