A PPPM Fast Summation Method for Fluids and Beyond

Xinxin Zhang*
UBC Computer Science

Robert Bridson†
UBC Computer Science, Autodesk Canada

Figure 1: *Vortices of raising smoke hitting a moving ball (ball not rendered).*

Abstract

Solving the N-body problem, i.e. the Poisson problem with point sources, is a common task in graphics and simulation. The naive direct summation of the kernel function over all particles scales quadratically, rendering it too slow for large problems, while the optimal Fast Multipole Method has drastic implementation complexity and can sometimes carry too high an overhead to be practical. We present a new Particle-Particle Particle-Mesh (PPPM) algorithm which is fast, accurate, and easy to implement even in parallel on a GPU. We capture long-range interactions with a fast multigrid solver on a background grid with a novel boundary condition, while short-range interactions are calculated directly with a new error compensation to avoid error from the background grid. We demonstrate the power of PPPM with a new vortex particle smoke solver, which features a vortex segment-approach to the stretching term, potential flow to enforce no-stick solid boundaries on arbitrary moving solid boundaries, and a new mechanism for vortex shedding from boundary layers. Comparison against a simpler Vortex-in-Cell approach shows PPPM can produce significantly more detailed results with less computation. In addition, we use our PPPM solver for a Poisson surface reconstruction problem to show its potential as a general-purpose Poisson solver.

CR Categories: G.1.9 [Mathematics of Computing]: Numerical Analysis—Integral Equations; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation

Keywords: Poisson, PPPM, vortex particles

Links: DL PDF

*e-mail:zhxx@cs.ubc.ca
†e-mail:rbridson@cs.ubc.ca

Table 1: *Common symbols used throughout the paper.*

$\boldsymbol{x}_i$	position of i^{th} Lagrangian element
$\boldsymbol{X}$	position of a grid cell's center
$\boldsymbol{u}(\boldsymbol{x})$	velocity evaluated for Lagrangian elements
$\boldsymbol{u}(\boldsymbol{X})$	velocity evaluated at grid cells
$\boldsymbol{u}_{far}$	the velocity introduced by far field vorticity
$\boldsymbol{u}_{near}$	the velocity introduced by near field vorticity
$\boldsymbol{u}_\omega$	rotational part of the flow velocity
$\boldsymbol{u}_\phi$	irrotational, divergence free part of the flow velocity
$\boldsymbol{u}_\infty$	velocity at infinity, such as the speed at which the reference frame is moving
$\boldsymbol{\omega}$	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
$\boldsymbol{\Psi}$	a vector valued stream function
Φ	scalar valued potential function
$\boldsymbol{\omega}_L$	the local vorticity field defined on nearby cells
$\boldsymbol{\Psi}_L$	the local stream function with source term $\boldsymbol{\omega}_L$
$\boldsymbol{u}_L$	velocity introduced by $\boldsymbol{\omega}_L$

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:

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

(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 [Chorin 1973]). 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 [Hockney and Eastwood 1989]. The same Poisson problem appears in electrostatics with a charge distribution on particles (or charge panels), with a recent interesting application in graphics [Wang et al. 2013]. A more thorough review regarding the Poisson problem and N-body dynamics can be found

in Hockney and Eastwood's work [1989]. In those simulations, the solution is usually obtained by

$$f(x) = \sum_{j:\, x \neq x_j} \frac{\rho_j v_j}{4\pi \|x - x_j\|}. \tag{2}$$

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. [2006] solve a Poisson problem to reconstruct a surface from point samples; Jacobson et al. [2013] 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. [Stam 1999; Fedkiw et al. 2001]) solve a Poisson problem every time step for pressure projection; and Lagrangian vorticity-based fluid solvers (e.g. [Park and Kim 2005]) evaluate the velocity field by summing over vortex sources with the Biot-Savart law, which is just the curl of equation (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) [Greengard and Rokhlin 1987], 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 [Couet et al. 1981] 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 [Anderson 1986]. 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 (2),

$$\boldsymbol{u}(\boldsymbol{x}) = \sum_{j:\, \boldsymbol{x} \neq \boldsymbol{x}_j} \nabla \frac{1}{4\pi \|\boldsymbol{x} - \boldsymbol{x}_j\|} \times \boldsymbol{\omega}_j \tag{3}$$

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.

2 Related work

Vortex methods are popular for high Reynolds number computational fluid dynamics [Cottet and Koumoutsakos 2000], very efficiently capturing turbulent details; as vorticity in particular is visually very important, many graphics researchers have adopted vortex approaches. Yaeger et al. [1986] began the trend with a VIC solver to produce the realistic look of atmospheric flow on Jupiter. Angelidis and Neyret [2005] used Lagrangian vortex primitives for their flexibility; Selle et al. [2005] used vortex particles to augment a more traditional Eulerian velocity-pressure simulation. Pfaff et al. [2009] used vortex particles from pre-processed boundary layers to synthesize turbulence. Weißmann and Pinkall [2010] used Biot-Savart summation for vortex rings.

Brochu et al. [2012] captured vorticity with a Lagrangian mesh, including its generation from buoyancy, and used FMM for linear-time Biot-Savart summation. Pfaff et al. [2012] 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 [Kazhdan et al. 2006], 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 [2008] 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. [Molemaker et al. 2008; Ferstl et al. 2014; McAdams et al. 2010]), though we should note that Henderson suggests on shared memory multiprocessors FFT still may be faster [2012].

3 Particle-Particle Particle-Mesh method

In this section we take Biot-Savart summation as an example to explain the PPPM algorithm in detail. See figure 2 for an overview.

Given the vorticity $\boldsymbol{\omega} = \nabla \times \boldsymbol{u}$ of an incompressible flow, and ignoring boundary conditions for now, we can reconstruct velocity (up to addition of the gradient of a harmonic scalar potential) by solving for a streamfunction $\boldsymbol{\Psi}$ whose curl is the velocity:

$$\begin{cases} \nabla^2 \boldsymbol{\Psi} = -\boldsymbol{\omega} \\ \boldsymbol{\Psi}(\boldsymbol{x}) = 0 \qquad \boldsymbol{x} \to \infty \end{cases} \tag{4}$$

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 $\boldsymbol{\omega}_i$, $\boldsymbol{\Psi}$ can be found by summation (c.f. equation 2). The velocity is just the curl of this sum (equation 3), further expressed as

$$\boldsymbol{u}(\boldsymbol{x}_i) = \frac{1}{4\pi} \sum_{j=1, j\neq i}^{n} \frac{\boldsymbol{\omega}_j \times (\boldsymbol{x}_i - \boldsymbol{x}_j)}{|\boldsymbol{x}_i - \boldsymbol{x}_j|^3}, \tag{5}$$

which is known as Biot-Savart summation.

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

Due to the singularity of the kernel, $\boldsymbol{u}_{near}$ varies rapidly in space and is related to the small scale motion (turbulence), while $\boldsymbol{u}_{far}$ 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 $\boldsymbol{u}_{far}$ for each grid cell using a particle-mesh solver (§3.1) and local-cancellation (§3.2). After that, PPPM interpolates $\boldsymbol{u}_{far}$ for each vortex element and evaluates $\boldsymbol{u}_{near}$ using direct summation. Details of the computational steps in the PPPM Biot-Savart summation are outlined in Algorithm 1, 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 1 PPPMBiotSavartSummation($\boldsymbol{\omega}_p$, v_p, $\boldsymbol{x}_p$, N, $\boldsymbol{u}_p$)

BB←GetBoundingBox($\boldsymbol{x}_p$)
DetermineComputationDomain //3×BB
$\boldsymbol{\omega}_g$ ←ParticleToMesh($\boldsymbol{\omega}_p$)
ComputeMonopoleDirichletBoundaryCondition
$\boldsymbol{\Psi}$ ←ParallelMultigridSolvePoisson
$\boldsymbol{u}_g$ ←Curl($\boldsymbol{\Psi}$)
for each grid cell in parallel
  Subtract near field estimate to get far field component
endfor
for each particle i in parallel
  $\boldsymbol{u}_{p,i} \leftarrow$ InterpolateFarField
endfor
for each particle i in parallel
  Sum over particles in neighborhood $\eta(i)$ of i:
    (evaluating an accurate near field component)
  $\Delta\boldsymbol{u} \leftarrow \sum_{j\in\eta(i)} \frac{\boldsymbol{\omega}_{p,j} \times (\boldsymbol{x}_{p,i} - \boldsymbol{x}_{p,j})}{4\pi |\boldsymbol{x}_{p,i} - \boldsymbol{x}_{p,j}|^3} \left(1 - \exp\left(\left(\frac{|\boldsymbol{x} - \boldsymbol{x}_j|}{\alpha}\right)^3\right)\right)$
  $\boldsymbol{u}_{p,i} \leftarrow \boldsymbol{u}_{p,i} + \Delta\boldsymbol{u}$
endfor

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):

$$\begin{cases} \nabla^2 \boldsymbol{\Psi} = -\boldsymbol{\omega} & \text{in } \Omega \\ \boldsymbol{\Psi} = g & \text{on } \partial\Omega \end{cases} \tag{6}$$

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 $\boldsymbol{x}_b$ on the boundary is exactly

$$\Psi(\boldsymbol{x}_b) = \sum_{i=1}^{N} \frac{\boldsymbol{\omega}(\boldsymbol{x}_i)\, h^3}{4\pi\, |\boldsymbol{x}_i - \boldsymbol{x}_b|}. \tag{7}$$

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 7 directly is too costly: instead, we compute the cheap monopole approximation of the parti-

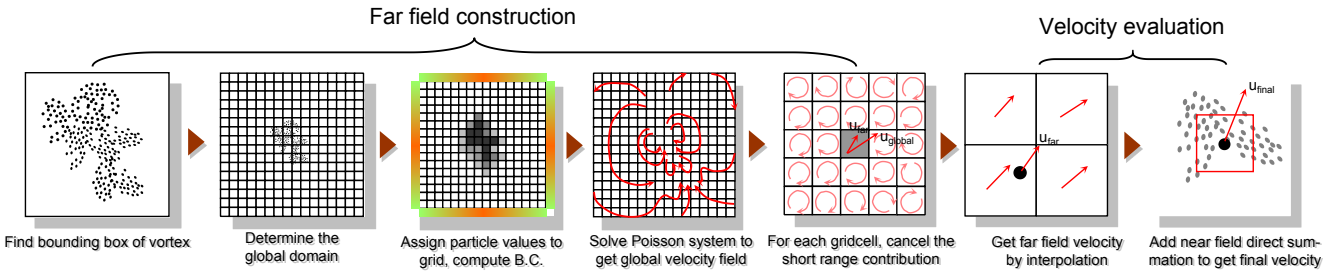

Figure 2: *An overview of our PPPM algorithm, which consists of a far-field construction step and a velocity evaluation step.*

cle quantities, replacing equation 7 with

$$\Psi(\boldsymbol{x}_b) = \sum_{i=1}^{N} \frac{\boldsymbol{\omega}(\boldsymbol{x}_i) h^3}{4\pi |\boldsymbol{x}_i - \boldsymbol{x}_b|} \approx \frac{\sum_{i=1}^{N} \boldsymbol{\omega}(\boldsymbol{x}_i) h^3}{4\pi |\boldsymbol{x}_c - \boldsymbol{x}_b|} = \frac{m(\boldsymbol{x}_c)}{4\pi |\boldsymbol{x}_c - \boldsymbol{x}_b|} \tag{8}$$

where $\boldsymbol{x}_c$ 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 1), 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

$$\boldsymbol{\omega}(\boldsymbol{X}) = \frac{1}{h^3} \sum_i v_i \boldsymbol{\omega}(\boldsymbol{x}_i) \left(\prod_{\theta=1,2,3} w_\theta \left(\frac{\boldsymbol{X}_\theta - \boldsymbol{x}_{i,\theta}}{h} \right) \right) \tag{9}$$

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:

$$w_\theta(r) = \begin{cases} 1, & r \in [-0.5, 0.5), \\ 0, & \text{else}, \end{cases} \tag{10}$$

In our implementation, we accelerate the process by parallelizing for each grid cell to gather vortex values around it, using spatial hashing [Teschner et al. 2003] for efficient neighbour finding, following Green's implementation [2008].

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 [2010].

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

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

$$\nabla^2 \Psi_L = -\boldsymbol{\omega}_L \tag{11}$$

in open space, where $\boldsymbol{\omega}_L$ is the near-field vorticity only from the nearby grid cells.

Conceptually, we can achieve this by putting $\boldsymbol{\omega}_L$ 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 $\boldsymbol{\omega}_L$ back to the global domain is a prolongation, $\mathcal{P}\boldsymbol{\omega}_L$. The global Poisson solve can be denoted as $\Psi = \mathcal{L}^{-1}\mathcal{P}\boldsymbol{\omega}_L$. Reading back Ψ_L is a restriction of Ψ, $\mathcal{R}\left(\mathcal{L}^{-1}\mathcal{P}\boldsymbol{\omega}_L\right)$. Finally, the local velocity field due to $\boldsymbol{\omega}_L$ is readily $\mathcal{R}\left(\mathcal{L}^{-1}\mathcal{P}\boldsymbol{\omega}_L\right)$.

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

$$\Psi_L = \mathbb{A} h^2 \boldsymbol{\omega}_L, \tag{12}$$

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

Theuns [1994] 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 [2003] 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 $\mathcal{O}\left(K^6\right)$ matrix $\mathbb{A}$ which solves $\Psi_L = \mathbb{A}\boldsymbol{\omega}_L$. 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

$$\Psi_L = \mathbb{G} h^2 \boldsymbol{\omega}_L \tag{13}$$

where $\mathbb{G}_{i,j} = \frac{h}{4\pi |X_i - X_j|}$ for $i \neq j$. (Observe that $|X_i - X_j|$ 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 3. These numerical findings give us a new formulation to compute Ψ_L from $\boldsymbol{\omega}_L$. We outline this procedure in Algorithm 2.

Algorithm 2 Local_inverse($\boldsymbol{\omega}_L$)

$\boldsymbol{\Psi}_L \leftarrow 0$
for $i, j, k \in L$ //$L = \{i_2, j_2, k_2 | i_2 \in [i-K, i+K], j_2 \in [j-K, j+K], k_2 \in [k-K, k+K]\}$
  $\boldsymbol{X}_1 \leftarrow \boldsymbol{X}_{i,j,k}$
  if $i_2, j_2, k_2 \in L, i_2, j_2, k_2 \neq i, j, k$
    $\boldsymbol{X}_2 \leftarrow \boldsymbol{X}_{i_2,j_2,k_2}$
    $\boldsymbol{\Psi}_L(\boldsymbol{X}_1) \leftarrow \boldsymbol{\Psi}_L(\boldsymbol{X}_1) + \frac{h^3 \boldsymbol{\omega}_L(\boldsymbol{X}_2)}{4\pi |\boldsymbol{X}_1 - \boldsymbol{X}_2|}$
    //using Green's function
  else //$i_2, j_2, k_2 = i, j, k$
    $\boldsymbol{\Psi}_L(\boldsymbol{X}_1) \leftarrow \boldsymbol{\Psi}_L(\boldsymbol{X}_1) + 0.25h^2 \boldsymbol{\omega}_L(X_2)$//notice $X_2 = X_1$ where, $0.25h^2$ is the diagonal constant
  endif
endfor

After we obtain Ψ_L, we can proceed to compute $\boldsymbol{u}_L$ by taking the finite difference curl of Ψ_L

$$\boldsymbol{u}_L = \nabla_h \times \Psi_L \tag{14}$$

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:

$$\begin{aligned}\nabla_{\boldsymbol{X},h} \times \Psi_L &= \nabla_{\boldsymbol{X},h} \times \sum_{\boldsymbol{X'} \neq \boldsymbol{X}} \frac{h^3 \boldsymbol{\omega}_L(\boldsymbol{X'})}{4\pi |\boldsymbol{X} - \boldsymbol{X'}|} \\ &= \sum_{\boldsymbol{X} \neq \boldsymbol{X'}} h^3 \boldsymbol{\omega}_L(\boldsymbol{X'}) \times \left(-\nabla_{\boldsymbol{X},h} \frac{1}{4\pi |\boldsymbol{X} - \boldsymbol{X'}|} \right)\end{aligned} \tag{15}$$

where $\nabla_{\boldsymbol{X},h} \frac{1}{4\pi |\boldsymbol{X} - \boldsymbol{X}_j|}$ denotes a discrete gradient operator to the Green's function at $\boldsymbol{X}$. Let $\boldsymbol{e}_1$ be the unit vector $(1, 0, 0)$; we have

$$\begin{aligned}&\left(\frac{\partial}{\partial x}\right)_{\boldsymbol{X},h} \frac{1}{4\pi |\boldsymbol{X} - \boldsymbol{X'}|} \\ &= \frac{1}{2h} \left(\frac{1}{4\pi |\boldsymbol{X} + h\boldsymbol{e}_1 - \boldsymbol{X'}|} - \frac{1}{4\pi |\boldsymbol{X} - h\boldsymbol{e}_1 - \boldsymbol{X'}|} \right).\end{aligned} \tag{16}$$

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

We then compute a streamfunction buffer from the diagonal contributions,

$$\Psi_d = 0.25h^2 \boldsymbol{\omega}_g \tag{17}$$

We then take the discrete curl of Ψ_d to get $\boldsymbol{u}_d$. The velocity introduced by the near field grid is consequently obtained via

$$\begin{aligned}&\boldsymbol{u}_L(\boldsymbol{X}) \\ &= \sum_{j \neq i} h^3 \boldsymbol{\omega}_L(\boldsymbol{X'}) \times \left(-\nabla_{\boldsymbol{X},h} \frac{1}{4\pi |\boldsymbol{X} - \boldsymbol{X'}|} \right) + \boldsymbol{u}_d\end{aligned} \tag{18}$$

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.

3.3 Velocity evaluation

The far field influence at any given grid position $\boldsymbol{X}$ can be readily obtained via:

$$\boldsymbol{u}_{far}(\boldsymbol{X}) = \boldsymbol{u}_{pm}(\boldsymbol{X}) - \boldsymbol{u}_L(\boldsymbol{X}) \tag{19}$$

where $\boldsymbol{u}_L$ is determined by equation 18 in §3.2.

To evaluate the velocity for an arbitrary evaluation position $\boldsymbol{x}_i$, we first interpolate from the far field buffer using trilinear interpolation,

$$\boldsymbol{u}_{far}(\boldsymbol{x}) \leftarrow \text{TriInterpolate}(\boldsymbol{u}_{far,} \boldsymbol{x}) \tag{20}$$

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

$$\begin{aligned}&\boldsymbol{u}(\boldsymbol{x}) = \boldsymbol{u}_{far} \\ &+ \sum_{\boldsymbol{x}_j \in C_L, \boldsymbol{x}_j \neq \boldsymbol{x}} \frac{\boldsymbol{\omega}_j \times (\boldsymbol{x} - \boldsymbol{x}_j)}{4\pi |\boldsymbol{x} - \boldsymbol{x}_j|^3} \left(1 - \exp\left(\frac{|\boldsymbol{x} - \boldsymbol{x}_j|}{\alpha} \right)^3 \right)\end{aligned} \tag{21}$$

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 $\mathcal{O}(N)$ time; applying the monopole boundary condition takes $\mathcal{O}\left(N + \left(\frac{N}{s}\right)^{2/3}\right)$ operations; the multigrid Poisson solver with $\frac{N}{s}$ unknowns takes $\mathcal{O}\left(\frac{N}{s}\right)$ operations to get a solution; computing the discrete curl or gradient of the field requires also $\mathcal{O}\left(\frac{N}{s}\right)$ operations; interpolating from the grid quantity back to particles takes $\mathcal{O}(N)$ operations; and finally direct summation with nearby particles takes $\mathcal{O}(N)$ 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 $\mathcal{O}(m+n)$ complexity as claimed.

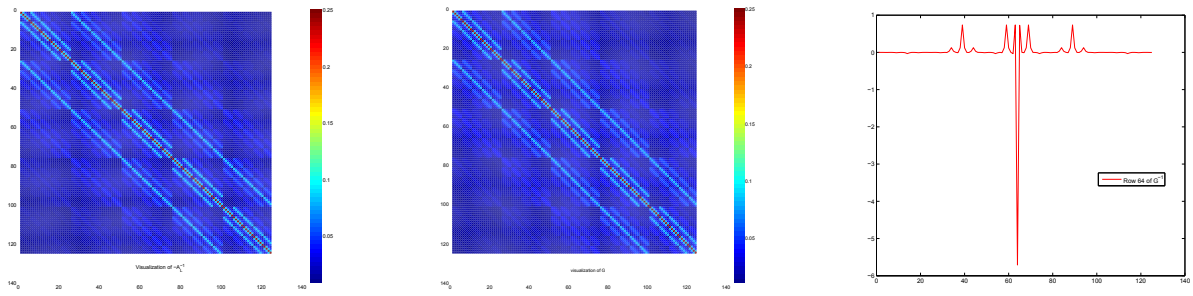

Figure 3: *Relationship between inverse of finite difference operator and Green's function. Left: the local inverse matrix defined by $\mathbb{A}$. Middle: the local inverse matrix $\mathbb{G}$ constructed using Green's function and the diagonal terms from $\mathbb{A}$. Right: one row of the inverse of $\mathbb{G}$, almost revealed the 7-point stencil.*

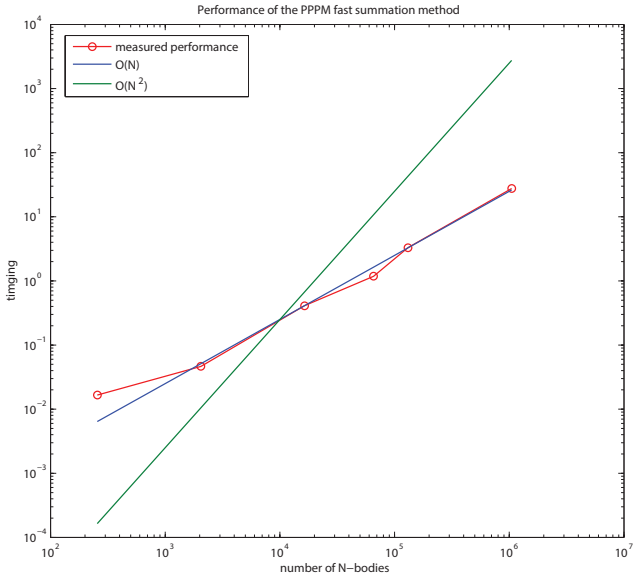

Figure 4: *Performance of the PPPM fast summation. Computation time grows linearly with the number of computational elements.*

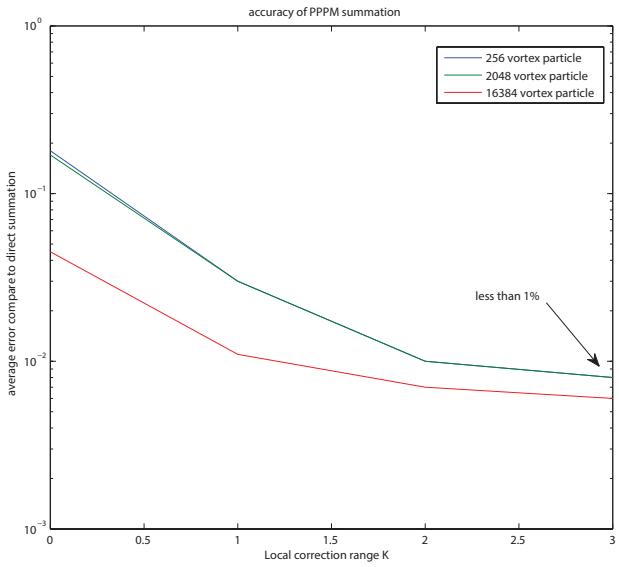

Figure 5: *Accuracy statistics of the PPPM fast summation.*

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 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 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. The computation time in our experiment grows linearly with respect to the number of vortex particles.

In table 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 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 [1994]. On the other hand, the particle-mesh method alone, even at higher cost, gives a poor approximation to direct summation.

4 Vortex-Particle Smoke

In vortex methods, given the vorticity distribution and boundary geometry, the velocity at any position $\boldsymbol{x}$ can be found as [Cottet and Koumoutsakos 2000]

$$\boldsymbol{u}(\boldsymbol{x}) = \boldsymbol{u}_\omega + \boldsymbol{u}_\phi + \boldsymbol{u}_\infty \tag{22}$$

which is a combination of a rotational flow $\boldsymbol{u}_\omega$ from the vorticity, a potential flow $\boldsymbol{u}_\phi$ determined by the solid objects §4.2, and a prescribed velocity field $\boldsymbol{u}_\infty$ defined by an artist (it is better for $\boldsymbol{u}_\infty$ to be divergence-free, otherwise, we suggest using $\nabla \times \boldsymbol{u}_\infty$ as a force field).

Ignoring viscosity, the dynamics of vorticity can be modeled from

Table 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 [Theuns 1994]	3	2.78%	2.86%

the curl of the inviscid momentum equation,

$$\frac{D\boldsymbol{\omega}}{Dt} = (\boldsymbol{\omega}\cdot\nabla)\,\boldsymbol{u} + \beta\nabla\rho\times\boldsymbol{g}. \tag{23}$$

A vortex blob moves according to the combined velocity $\boldsymbol{u}$ of equation. 22, with vortex stretching($(\boldsymbol{\omega}\cdot\nabla)\,\boldsymbol{u}$) due to flow deformation in 3D and baroclinic vorticity generation $\beta\nabla\rho\times\boldsymbol{g}$ concentrated on the density interface.

In the presence of solid objects, an irrotational divergence-free flow field $\boldsymbol{u}_\phi$ 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 §4.2.

We discretize the flow using a set of vortex blobs, with position $\boldsymbol{x}_i$, volume v_i, density ρ_i, velocity $\boldsymbol{u}_i$, and vortex density $\boldsymbol{\omega}_i$. At each time step, we compute $\boldsymbol{u}_\omega$ with Biot-Savart,

$$\boldsymbol{u}_\omega(\boldsymbol{x}) = \sum_{\text{all } j,j\neq i} \frac{v_j\boldsymbol{\omega}_j\times(\boldsymbol{x}-\boldsymbol{x}_j)}{4\pi\left|\boldsymbol{x}-\boldsymbol{x}_j\right|^3}\left(1-\exp\left(\frac{\left|\boldsymbol{x}-\boldsymbol{x}_j\right|}{\alpha}\right)^3\right), \tag{24}$$

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, $\boldsymbol{u}_\phi$, can be determined by

$$\boldsymbol{u}_\phi(\boldsymbol{x}) = \sum_{all\ j,j\neq i} \frac{A_j\sigma_j(\boldsymbol{x}_j-\boldsymbol{x})}{4\pi\left|\boldsymbol{x}-\boldsymbol{x}_j\right|^3}\left(1-\exp\left(\frac{\left|\boldsymbol{x}-\boldsymbol{x}_j\right|}{\alpha}\right)^3\right) \tag{25}$$

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 $\boldsymbol{x}_j$ 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. §4.1
2. Update the position of each end of the vortex segment with the velocity determined by equation 22, hence, the vortex segment is stretched automatically. §4.1
3. Reduce instabilities introduced by vortex stretching.§4.1
4. Add baroclinic vorticity generation to the vorticity field.
5. Add vorticity due to vortex shedding.4.2

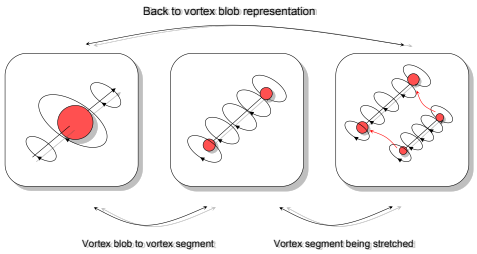

Figure 6: *We switch to a vortex segment representation of vortex particles at the beginning of each time step, move both ends of the vortex segment in the flow, then switch back to vortex blobs.*

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 a geometry-inspired filtering approach.

4.1.1 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

$$\boldsymbol{\omega}^{n+1} = \boldsymbol{\omega}^n + \boldsymbol{\omega}^n\cdot\nabla\boldsymbol{u}^n \tag{26}$$

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 $\boldsymbol{x}_a$ and $\boldsymbol{x}_b$, length h, and constant circulation κ. A vortex blob with vorticity strength $v_i\boldsymbol{\omega}_i$ can be translated to a vortex segment of length h_i, with unit direction $\hat{t}_i = \frac{\boldsymbol{\omega}_i}{\sqrt{\|\boldsymbol{\omega}_i\|^2}}$, and circulation $\kappa_i = \boldsymbol{\omega}_i/\left(h_i\hat{t}_i\right)$. We translate our vortex blob into such vortex segments at the beginning of each time step, evaluate the velocity according to equation 22 for each end of the vortex segment, and move each end of the vortex segment according to

$$\begin{aligned}\boldsymbol{x}_{i,a}^{n+1} &= \boldsymbol{x}_{i,a}^n + \Delta t\boldsymbol{u}_{i,a}^n\\ \boldsymbol{x}_{i,b}^{n+1} &= \boldsymbol{x}_{i,b}^n + \Delta t\boldsymbol{u}_{i,b}^n.\end{aligned} \tag{27}$$

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,

$$v_i\boldsymbol{\omega}_i = \kappa_i\left(\boldsymbol{x}_{i,b}^{n+1} - \boldsymbol{x}_{i,a}^{n+1}\right). \tag{28}$$

Notice the circulation is conserved. This whole process is illustrated in figure 6.

4.1.2 Stability issues

Vortex stretching is potentially unstable. Without addressing this sensitivity, our simulation quickly diverges even with small timesteps. We noticed that in turbulent flow, even when the magnitude

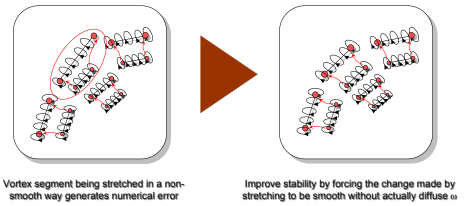

Figure 7: *Sudden discontinuous motion of vortex segments introduces and amplifies numerical error, which is reduced then by smoothing the stretching terms.*

of the velocity is small, the gradient of the velocity field can be very large, which makes $\boldsymbol{\omega} \cdot \nabla \boldsymbol{u}$ 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

$$(\boldsymbol{\omega} \cdot \nabla \boldsymbol{u})_i^n = \frac{\kappa_i \left(\boldsymbol{x}_{i,b}^{n+1} - \boldsymbol{x}_{i,a}^{n+1} \right) - \boldsymbol{\omega}^n}{\Delta t} \tag{29}$$

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

$$\boldsymbol{\omega}_i^{n+1} = \boldsymbol{\omega}_i^n + \Delta t (\boldsymbol{\omega} \cdot \tilde{\nabla} \boldsymbol{u})_i^n. \tag{30}$$

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 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 [Zhu and Bridson 2005].

4.2 Solid boundaries and vortex shedding

We incorporate boundary conditions by introducing an irrotational, divergence free flow field $\boldsymbol{u}_\phi$ that cancels the normal velocity flux on the boundary made by $\boldsymbol{u}_\omega + \boldsymbol{u}_\infty$. We define $\boldsymbol{u}_\phi$ 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 [Chorin 1973], and merge this surface vorticity into the vorticity field.

4.2.1 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[Brochu et al. 2012]

$$\begin{aligned} \Delta \Phi &= 0 \\ \frac{\partial \Phi}{\partial \boldsymbol{n}} &= (\boldsymbol{u}_{solid} - \boldsymbol{u}_{fluid}) \cdot \boldsymbol{n} \end{aligned} \tag{31}$$

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

$$\Phi(\boldsymbol{x}) = \int_{\partial\Omega} \frac{\sigma(\boldsymbol{y})}{4\pi |\boldsymbol{x} - \boldsymbol{y}|} dS(\boldsymbol{y}). \tag{32}$$

Taking the normal derivative of Φ and substituting it into equation 31 gives a Fredholm equation of the second kind,

$$b(\boldsymbol{x}) = -\frac{\sigma(\boldsymbol{x})}{2} + \int_{\partial\Omega} \sigma(\boldsymbol{y}) \frac{\partial}{\partial \boldsymbol{n}_x} \frac{1}{4\pi |\boldsymbol{x} - \boldsymbol{y}|} dS(\boldsymbol{y}) \tag{33}$$

where $f(\boldsymbol{x}) = (\boldsymbol{u}_{solid}(\boldsymbol{x}) - \boldsymbol{u}_{fluid}(\boldsymbol{x})) \cdot \boldsymbol{n}(\boldsymbol{x})$.

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

$$b_i = -\frac{\sigma_i}{2} + \sum_{all\ j} \frac{\partial}{\partial \boldsymbol{n}_i} \frac{\sigma_j A_j}{4\pi |\boldsymbol{x}_i - \boldsymbol{x}_j|} \tag{34}$$

where A_j is the area of j'th surface element, and $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ 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 $\mathcal{O}(1)$ 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 34 using the PPPM framework. Given a source distribution σ, with proper reordering, the off-diagonal part summation is exactly

$$b_i^{off-diag} = \boldsymbol{n}_i \cdot \left(\sum_{all\ j \neq i} \frac{\sigma_j A_j (\boldsymbol{x}_j - \boldsymbol{x}_i)}{4\pi |\boldsymbol{x}_i - \boldsymbol{x}_j|^3} \right) \tag{35}$$

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 3.

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 3), and when the iteration is terminated we use PPPM to compute $\boldsymbol{u}_\phi$ based on the single layer density.

Algorithm 3 PPPM_accelerated_Ax($\boldsymbol{b}$, σ, $\boldsymbol{p}$, A, $\boldsymbol{n}$, M)

1: $\boldsymbol{f} \leftarrow \boldsymbol{0}$
2: $\boldsymbol{b} \leftarrow \boldsymbol{0}$
3: $\boldsymbol{m} \leftarrow \boldsymbol{0}$
4: **for** i=1:M in parallel
5: $\quad \boldsymbol{m}_i = A_i * \sigma_i$
6: **endfor**
7: $\boldsymbol{f} \leftarrow$**PPPM_Compute_Gravity**($\boldsymbol{m}$, $\boldsymbol{p}$, M)
8: **for** i=1:M in parallel
9: $\quad b_i \leftarrow \boldsymbol{n}_i \cdot \boldsymbol{f}_i - 0.5 * \sigma_i$
10: **endfor**

4.2.2 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 [1973] 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.

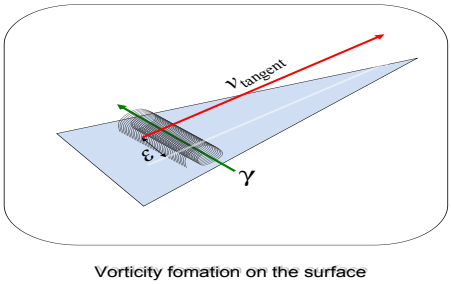

Figure 8: *Vortex shedding process. In our approach, the vorticity strength is explicitly determined.*

Figure 9: *Rising smoke using different number of vortex particles. Left, 2049 particles: middle, 16384 particles; right, 130K particles.*

Extending this idea to 3D, we determine the 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 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

$$\Delta\boldsymbol{\omega} = \Delta t c A_j \boldsymbol{\gamma}_j \tag{36}$$

5 Results and Conclusions

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

Figure 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 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.*

Figure 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 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 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 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 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.

Figure 12: *Moving objects in slightly viscous flow generate turbulent wakes. Top row: vortex shedding from a moving ball. Bottom row: a moving bunny.*

Figure 13: *PPPM Poisson surface reconstruction of: left) a bunny, right) a dragon.*

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 [Kazhdan et al. 2006] 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 by the input, i.e., find ϕ that solves

$$\nabla^2 \phi = \nabla \cdot V \tag{37}$$

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

$$\phi(x_i) = \sum_{j=1, j\neq i}^{n} \frac{h^3 f_j}{4\pi \|x_i - x_j\|} \tag{38}$$

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 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 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.

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 its PPPM scalability for sparse data. Adaptive Poisson solvers [Losasso et al. 2004] might address this problem.

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 subdomain B.C. in large scale domain decomposition solvers, extending Eulerian simulation to open space, fast image and geometry processing techniques.

Acknowledgments

This research was supported by a grant from the Natural Sciences and Engineering Research Council of Canada.

References

ANDERSON, C. R. 1983. *Vortex Methods for Flows of Variable Density*. PhD thesis, University of California, Berkeley.

ANDERSON, C. R. 1986. A method of local corrections for computing the velocity field due to a distribution of vortex blobs. *J. Comp. Physics 62*, 111–123.

ANGELIDIS, A., AND NEYRET, F. 2005. Simulation of smoke based on vortex filament primitives. In *Symposium on Computer Animation*, 87–96.

BROCHU, T., KEELER, T., AND BRIDSON, R. 2012. Linear-time smoke animation with vortex sheet meshes. In *Proc. Symp. Comp. Animation*, 87–95.

BROWN, G. L., AND ROSHKO, A. 1974. On density effects and large structure in turbulent mixing layers. *J. Fluid Mechanics 64*, 775–816.

CHORIN, A. J. 1973. Numerical study of slightly viscous flow. *Journal of Fluid Mechanics Digital Archive 57*, 04, 785–796.

COHEN, J. M., TARIQ, S., AND GREEN, S. 2010. Interactive fluid-particle simulation using translating Eulerian grids. In *ACM Symp. I3D*, 15–22.

COTTET, G.-H., AND KOUMOUTSAKOS, P. 2000. *Vortex methods - theory and practice*. Cambridge University Press.

COUET, B., BUNEMAN, O., AND LEONARD, A. 1981. Simulation of three-dimensional incompressible flows with a vortex-in-cell method. *Journal of Computational Physics 39*, 2, 305 – 328.

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

FERSTL, F., WESTERMANN, R., AND DICK, C. 2014. Large-scale liquid simulation on adaptive hexahedral grids. *Visualization and Computer Graphics, IEEE Transactions on PP*, 99, 1–1.

GREEN, S. 2008. Cuda particles. *nVidia Whitepaper 2*, 3.2, 1.

GREENGARD, L., AND ROKHLIN, V. 1987. A fast algorithm for particle simulations. *J. Comp. Physics 73*, 325–348.

GUMEROV, N. A., AND DURAISWAMI, R. 2008. Fast multipole methods on graphics processors. *Journal of Computational Physics 227* (2008/09/10/), 8290 – 8313.

HENDERSON, R. D. 2012. Scalable fluid simulation in linear time on shared memory multiprocessors. In *Proceedings of the Digital Production Symposium*, ACM, New York, NY, USA, DigiPro '12, 43–52.

HERRMANN, M. 2005. An Eulerian level set/vortex sheet method for two-phase interface dynamics. *J. Comput. Phys. 203*, 2 (Mar.), 539–571.

HOCKNEY, R. W., AND EASTWOOD, J. W. 1989. *Computer Simulation Using Particles*. Taylor & Francis, Jan.

JACOBSON, A., KAVAN, L., , AND SORKINE-HORNUNG, O. 2013. Robust inside-outside segmentation using generalized winding numbers. *ACM Trans. Graph. 32*, 4, 33:1–33:12.

KAZHDAN, M. M., BOLITHO, M., AND HOPPE, H. 2006. Poisson surface reconstruction. In *Symp. Geometry Processing*, 61–70.

KOUMOUTSAKOS, P., COTTET, G.-H., AND ROSSINELLI, D. 2008. Flow simulations using particles: bridging computer graphics and CFD. In *ACM SIGGRAPH 2008 Classes*, 25:1–25:73.

LOSASSO, F., GIBOU, F., AND FEDKIW, R. 2004. Simulating water and smoke with an octree data structure. *ACM Transactions on Graphics 23*, 457–462.

MCADAMS, A., SIFAKIS, E., AND TERAN, J. 2010. A parallel multigrid poisson solver for fluids simulation on large grids. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, SCA '10, 65–74.

MOLEMAKER, J., COHEN, J. M., PATEL, S., AND NOH, J. 2008. Low viscosity flow simulations for animation. In *SCA '08: Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, Eurographics Association, 9–18.

MUSETH, K., LAIT, J., JOHANSON, J., BUDSBERG, J., HENDERSON, R., ALDEN, M., CUCKA, P., HILL, D., AND PEARCE, A. 2013. Openvdb: An open-source data structure and toolkit for high-resolution volumes. In *ACM SIGGRAPH 2013 Courses*, ACM, New York, NY, USA, SIGGRAPH '13, 19:1–19:1.

PARK, S. I., AND KIM, M.-J. 2005. Vortex fluid for gaseous phenomena. In *Proc. Symp. Comp. Animation*, 261–270.

PFAFF, T., THUEREY, N., SELLE, A., AND GROSS, M. 2009. Synthetic turbulence using artificial boundary layers. *ACM Trans. Graph. 28*.

PFAFF, T., THUEREY, N., AND GROSS, M. 2012. Lagrangian vortex sheets for animating fluids. *ACM Trans. Graph. 31*, 4, 112:1–8.

SELLE, A., RASMUSSEN, N., AND FEDKIW, R. 2005. A vortex particle method for smoke, water and explosions. *ACM Transactions on Graphics 24*, 910–914.

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

TESCHNER, M., HEIDELBERGER, B., MUELLER, M., POMERANETS, D., AND GROSS, M. 2003. Optimized spatial hashing for collision detection of deformable objects. In *Proc. VMV*, 47–54.

THEUNS, T. 1994. Parallel PPPM with exact calculation of short range forces. *Computer Physics Commun. 78*, 3, 238 – 246.

WALTHER, J. H. 2003. An influence matrix particle-particle particle-mesh algorithm with exact particle-particle correction. *J. Comp. Physics 184*, 670–678.

WANG, H., SIDOROV, K. A., SANDILANDS, P., AND KOMURA, T. 2013. Harmonic parameterization by electrostatics. *ACM Trans. Graph. 32*, 5 (Oct.), 155:1–155:12.

WEISSMANN, S., AND PINKALL, U. 2010. Filament-based smoke with vortex shedding and variational reconnection. *ACM Trans. Graph. 29*.

WU, J.-Z. 1995. A theory of three-dimensional interfacial vorticity dynamics. *Physics of Fluids 7*, 2375–2395.

YAEGER, L., UPSON, C., AND MYERS, R. 1986. Combining physical and visual simulation–creation of the planet Jupiter for the film "2010". *Proc. SIGGRAPH 20*, 4 (Aug.), 85–93.

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