UC Riverside

UC Riverside Electronic Theses and Dissertations

Title

Coupling, Conservation, and Performance in Numerical Simulations

Permalink

https://escholarship.org/uc/item/66r9r1vm

Author

Ding, Ounan

Publication Date

2019

Peer reviewed|Thesis/dissertation

eScholarship.org Powered by the California Digital Library
University of California

UNIVERSITY OF CALIFORNIA
RIVERSIDE

Coupling, Conservation, and Performance in Numerical Simulations

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

by

Ounan Ding

December 2019

Dissertation Committee:

Dr. Craig Schroeder, Chairperson
Dr. Tao Jiang
Dr. Christian Shelton
Dr. Tamar Shinar

Copyright by
Ounan Ding
2019

The Dissertation of Ounan Ding is approved:

Committee Chairperson

University of California, Riverside

Acknowledgments

The text of this thesis, in part or in full, is a reprint of the material as it appears in [47]. The co-author Craig Schroeder listed in that publication directed and supervised the research which forms the basis for Chapter 2 of this dissertation.

The text of this thesis, in part or in full, is a reprint of the material as it will appear in Journal of Computational Physics. The co-author Craig Schroeder listed in that publication directed and supervised the research which forms the basis for Chapter 3 of this dissertation. The co-author Tamar Shinar helped work out with the MAC transfers and show that they worked.

I would like to thank my advisor Craig Schroeder for his advice and help in problem solving. He also has taught me how to plan realistic goals and approach my goals gradually.

I would like to thank Tamar Shinar for her ideas in one of my paper, and the introduction to physically based simulation. Her class CS 210 Scientific Computing is extremely useful as a starting point in my research.

I would like to thank the rest of my dissertation committee, Tao Jiang and Christian Shelton, for the time and effort in serving my defense and candidacy.

I would like to thank Muzaffer Akbay for his help in onboarding me at the beginning. He also provided many useful suggestions for the milestones along my PhD journey. During my last paper I had many discussions with Brian Crites and Heran Bhakta. I would like to thank them for the suggestions to formulate the ideas of test cases in that project. I would also like to thank Jonathan Kaneshiro for helping with rendering in the penalty force project.

I would like also to thank the University of California for the Graduate Division Fellowship, which covered part of the 2017-2018 academic year.

To my wife Fei Zhao, for her understanding and support during my PhD journey.

ABSTRACT OF THE DISSERTATION

Coupling, Conservation, and Performance in Numerical Simulations

by

Ounan Ding

Doctor of Philosophy, Graduate Program in Computer Science
University of California, Riverside, December 2019
Dr. Craig Schroeder, Chairperson

This thesis considers three aspects of the numerical simulations, which are coupling, conservation, and performance. We conduct a project and address one challenge from each of these aspects.

We propose a novel penalty force to enforce contacts with accurate Coulomb friction. The force is compatible with fully-implicit time integration and the use of optimization-based integration. In addition to processing collisions between deformable objects, the force can be used to couple rigid bodies to deformable objects or the material point method. The force naturally leads to stable stacking without drift over time, even when solvers are not run to convergence. The force leads to an asymmetrical system, and we provide a practical solution for handling these.

Next we present a new technique for transferring momentum and velocity between particles and MAC grids based on the Affine-Particle-In-Cell (APIC) framework [93, 94] previously developed for co-located grids. We extend the original APIC paper [93] and show

that the proposed transfers preserve linear and angular momentum and also satisfy all of the original APIC properties.

Early indications in [93] suggested that APIC might be suitable for simulating high Reynolds fluids due to favorable retention of vortices, but these properties were not studied further. We use two dimensional Fourier analysis to investigate dissipation in the limit $\Delta t = 0$. We investigate dissipation and vortex retention numerically to quantify the effectiveness of APIC compared with other transfer algorithms.

Finally we present an efficient solver for problems typically seen in microfluidic applications. Microfluidic "lab on a chip" devices are small devices that operate on small length scales on small volumes of fluid. Designs for microfluidic chips are generally composed of standardized and often repeated components connected by long, thin, straight fluid channels. We propose a novel discretization algorithm for simulating the Stokes equations on geometry with these features, which produces sparse linear systems with many repeated matrix blocks. The discretization is formally third order accurate for velocity and second order accurate for pressure in the L^∞ norm. We also propose a novel linear system solver based on cyclic reduction, reordered sparse Gaussian elimination, and operation caching that is designed to efficiently solve systems with repeated matrix blocks.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Numerical simulations have found applications in many fields, such as film making, engineering, and computer-aided design (CAD). In these applications, we are especially interested in three important aspects of the numerical simulation, which are coupling, conservation, and performance. The interaction between objects with the same or different materials plays an essential role in the physically based animation; without the coupled behavior, the animation would be dull. The conservation laws determine how dynamics work in the real world. Numerical simulations should respect it. Finally, we want to run simulations as fast as possible, which is especially desired in CAD applications; with faster simulations, designers can get rapid feedback on their designs and revise them responsively. In this thesis, three projects will be presented, and each of them will focus on one aspect of numerical simulation.

We consider the coupling between objects in Chapter 2. One major problem we solve is to enforce the Coulomb friction accurately, which is important to provide visually

plausible simulations. One typical example is the stable stack (see Figure 2.12), where friction force is vital in supporting this stack. Failure to compute the friction force accurately would lead to small but unrecoverable drift at each time step, and make the stack collapsed eventually. However, this problem is difficult due to the non-linearity of the friction force. In Chapter 2 we propose a contact model based on penalty force that enforces Coulomb friction accurately. This proposed force model is general enough to be used to couple rigid bodies to deformable objects or the material point method. We also provide an implicit Newton-based solver to solve the system equations that results from backward Euler. This chapter is based on published work [47].

Another key factor in numerical simulations is conservation. In Chapter 3 we will discuss this topic in the background of the hybrid method, which uses both particles and Eulerian grid in the simulation. Because there are two different representations in the simulation, we will need a transfer algorithm to convert between them. Due to the mismatch of the number of particle and grid degrees of freedom, this transfer is not generally exact. In the traditional Particle-In-Cell (PIC) method [79], directly interpolating the velocities between particle and grids introduces excessive numerical dissipation. On the other hand, in Fluid Implicit Particle (FLIP) [26, 25], letting the particle velocity bypass the transfer and only transferring the velocity changes from the grid to particle reduces the numerical dissipation, but introduces noise. In Chapter 3 we propose a new transfer algorithm, which is based on the Affine-Particle-In-Cell (APIC) framework [93, 94] was originally developed for co-located grids. Our proposed algorithm transfers velocity and momentum between particles and MAC grids. It preserves angular momentum and also satisfies the original

properties of co-located APIC. We also propose a tool based on 2D Fourier Transform to analyze the numerical dissipations in our transfer algorithm. Based on this tool, we study the dissipation level of our method and compare it with other existing transfer algorithms.

The last project is about the performance of the numerical simulation. It comes from a CAD background. Nowadays, engineers design their products using computers and verify their functions with numerical simulations before actual manufacturing. The simulation speeds up the workflow a lot. In Chapter 4 we consider a specific application in the design of the microfluidics Lab-on-Chip (LoC) system, where the common laboratory functions of chemistry and biology are integrated on a small chip. This LoC system typically consists of many long and thin pipes connected by joints. Researchers layout these pipes and joints so that the chip satisfies some desired properties, such as a linear velocity gradient should show up in a place, or two reagents are mixed sufficiently. Many general purpose tools or software packages exist for simulating this kind of flow. However, microfluidics chip designers still have to spend several hours in simulation with these numerical simulators. We observe that the unique geometry in the microfluidics LoC system leads to many duplicate computations, and this observations provides us a large room for improvement in performance. In Chapter 4 we take a closer look at the domain and propose an efficient solver by exploiting the duplicate and independent computations. The proposed method can solve the problem with 1 million degrees of freedom in 1 second using 16 threads on a typical workstation.

Chapter 2

Penalty force for coupling materials with Coulomb friction

Accurately computing the friction force between contacting objects is difficult. One reason is that friction force is nonlinear; the type of the Coulomb friction has to be determined first, and then the magnitude of the friction can be computed. Another reason is that friction force at different collision positions works together to achieve a balance. For example, in the stable stack test (see Figure 2.12), the friction force resulted from three contacts collaborates to support the top object in the stack. The simulation should be aware of this nonlinear and coupled nature of friction force. In this chapter, we will present a method that couples materials with Coulomb friction accurately.

2.1 Introduction

Collisions between objects are a very important visual phenomenon, and the problem of resolving them has been explored extensively within computer graphics. Methods for resolving collisions can be classified broadly into three categories: Constraint-based approaches, penalty methods, and impulse-based methods.

Constraint-based approaches formulate contacts as algebraic constraints. These methods come in a wide variety, based on how the constraints are formulated, how friction is treated, the type of numerical problem that results, and how that problem is solved. The simplest formulations ignore friction during contact resolution, treating collisions as simple constraints [15, 60] and applying friction as a post-process. These methods are attractive because they yield problems that can be solved efficiently. Coevoet et al. [37] used a frictionless formulation with linearized finite element forces to achieve real time simulations of soft robots. Including friction in the formulation complicates the formulation significantly, resulting in linear or nonlinear complementarity problems (LCP or NCP) [13, 38, 158, 8]. This approach is particularly popular for rigid bodies, which are well-suited to this formulation due to the long-range effects of contacts and collisions in systems and the difficulty of resolving them simultaneously, but formulations with deformable objects are also possible [136]. Methods based on LCP or NCP formulations tend to scale poorly with the number of constraints n, scaling as $O(n^2)$ or worse [49]. The difficulty arises because the problem of resolving contact constraints is combinatorial in nature. Indeed, the general problem of computing contact forces to satisfy normal and frictional constraints is NP-hard [13]. The problem may also have no solution, necessitating the use of impulses to guarantee a solution.

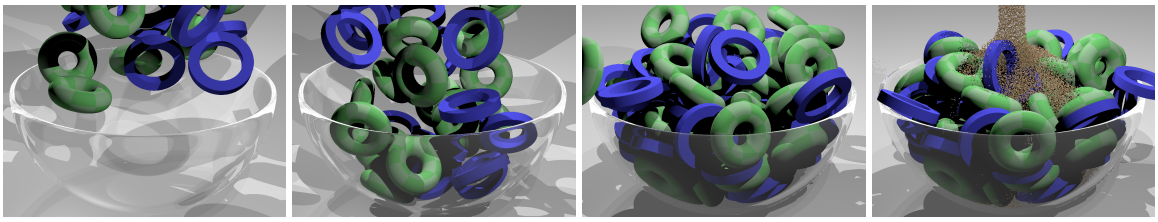


Figure 2.1: Rigid bodies are dropped into a bowl, and sand is poured on top. ©2019 IEEE

When collisions are decoupled from internal forces for deformable objects, implicit forces can be computed independently per object and in parallel, which can help offset the scaling with respect to the number of constraints [39].

Penalty formulations instead treat contacts as elastic forces, which are integrated alongside other forces. Due to their simplicity, penalty formulations are quite old [167, 125, 127, 15]. These methods are attractive because the computational cost of applying penalty forces scales linearly with the number of constraints, making them a popular choice for haptics [80]. A major limitation is the need to tune penalty stiffnesses, which represent a trade-off between performance and accuracy. Penalty methods normally allow some degree of collision to exist, though a stiff barrier potential is sometimes used to prevent this at additional computational cost. In this paper, we will use a penalty formulation because of its flexibility and because it allows us to naturally couple contact forces with elastic forces.

Impulse-based formulations represent a third approach to resolving contact. These methods work by resolving contacts one by one, iterating until some convergence criterion is achieved. This strategy is able to include friction and coefficient of restitution in a simple way. Due to its simplicity, this approach has been widely used, such as for cloth

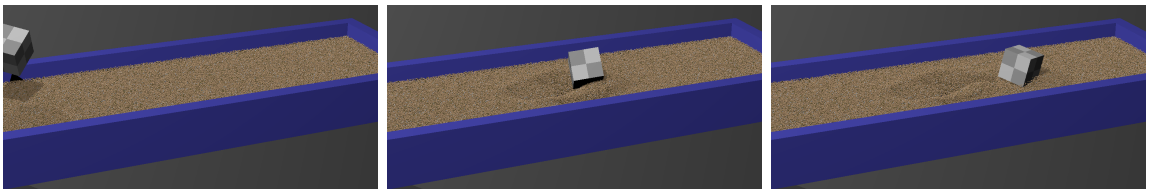


Figure 2.2: In this simulation, we throw a cube over a sand box. The cube tumbles through the sand and comes to rest. ©2019 IEEE

[140, 30] or rigid bodies [76]. Due to the iterative nature, these methods tend to be biased to some degree on the order in which collisions are processed, though many strategies exist to limit or avoid this. Impulse-based contacts are generally processed separately from elastic forces.

2.1.1 Penalty methods

Penalty forces compute contact forces as functions of some measure of penetration depth or close approach. [90] propose a penalty force model for contact between a robot and a ground based on linear damper springs. [121] proposed a penalty formulation based on a nonlinear damper spring, which they construct to avoid force discontinuities when contacts are created or lost. [181] adapt this nonlinear damping spring to the general contact problem between polygonal objects. They propose a penalty force model for contacts that uses damper springs connected to attachment points to enforce frictional contact. Their attachment points move when dynamic friction is applied, a strategy we also employ. Their penalty force is designed with the advantages and limitations of explicit time integration in mind. The characteristics of explicit time integration dictate many of the core design decisions, including a careful treatment to avoid applying too much dynamic friction, which could nonphysically flip the tangential sliding direction. The method of [179] adapts the

penalty force model of [181] to the problem of robustly simulating problems involving large, time-varying contact areas. They use semi-implicit integration, including contact forces in their implicit integration but applying frictional forces explicitly.

Implicit friction The explicit friction formulations used by these methods are quite simple. It would seem tempting to include these models in the nonlinear force computations, as indeed [179] suggests as a possibility. In practice, this is a more difficult proposition than it may seem. In the case of [179], their implicit formulation solves a linearization of the problem (one step of Newton), followed by explicit (and very nonlinear) friction. The non-smooth nature of the static-dynamic transition is incompatible with the linearized nature of such a solver; the choice of static vs dynamic would need to be made *before* finding out whether the friction cone would actually be violated, which could result in non-physical strong sticking, tangential velocity reversal, or energy gain. This problem could be overcome by using multiple Newton iterations, which would give the solver a chance to correctly decide between static and dynamic friction. This solution in turn leads to Newton convergence problems (friction is non-smooth), line searches (or other stabilization techniques), and the need to gracefully handle configurations far from the current one (these occur occasionally when the trial Newton step is poor, during line searches, etc.). Solving these impediments to implicit friction is a major contribution of this paper.

An alternative strategy is to formulate the penalty force using an estimate of the volume of contact [80, 49]. This formulation emerged as an alternative to methods that computed penalty forces only for the deepest penetrating point, since this leads to significant artifacts such a chattering. This problem is largely avoided by applying forces to many or

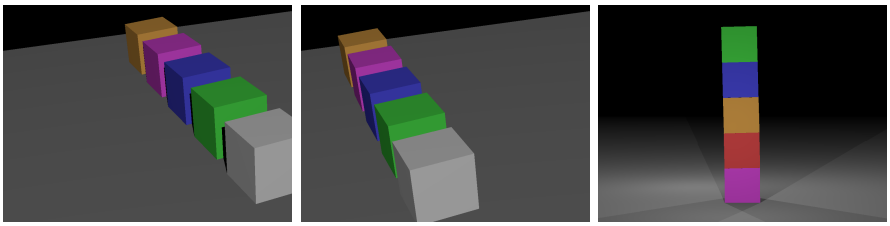


Figure 2.3: (Left and middle) Analytic friction tests showing MPM/level set (orange), MPM/rigid body (magenta), rigid/rigid (blue), and rigid/deformable (green) contacts sliding to the left while obeying the analytic solution (silver). (Right) A stack of mixed objects remains standing indefinitely (rigid: magenta, blue, green; deformable: red, orange). ©2019 IEEE

all penetrating particles, though a degree of resolution-dependence remains. Due to the complications involved with harmonizing attachment points and contact volume, we pursue a depth-based formulation and apply forces to all penetrating particles.

The computation of depth for deformable objects is nontrivial and has received significant attention [56]. Of particular relevance is [166], which used continuous collision detection to compute colliding point-face and edge-edge pairs. Their algorithm is explicit and includes Coulomb friction. Once computed, forces are computed along a chosen normal direction between pairs and persist until the contact is resolved.

2.1.2 Coupling

Part of our motivation for pursuing a penalty force formulation is its ability to couple different types of solid materials. In this paper, we consider coupling of rigid bodies to deformable bodies or to the material point method (MPM).

Many methods have been considered for coupling rigid bodies to deformable bodies [14, 134, 92, 106] to varying degrees. For example, [155, 99] allow rigid bodies and deformable bodies to exchanges forces and be embedded but do not consider contacts between them.

The method of [154] treats collisions using a combination of impulse-based methods but also includes some contact forces in their implicit solves. More recent method [182, 99] achieve interactive or real-time rates through model simplifications.

Coupling between rigid bodies and MPM has received much less attention. MPM was original developed by Sulsky [161, 162] as an extension of hybrid PIC/FLIP fluid schemes to viscoelastic materials. Since its introduction to graphics by [160] as a way to simulate snow, MPM has become an increasingly popular simulation choice for complex materials, including sand [100, 40]. The method of [40] was the first to couple MPM with rigid bodies. Their method couples MPM-based sand to rigid bodies by unifying the internal frictional contact forces from sand with frictional contact forces with rigid bodies. (Sand was coupled to rigid bodies originally in [129], but they did not simulate sand with MPM.) Because the coupling of [40] relies on the sand constitutive model, it is not a general coupling method and cannot be applied to other MPM-based materials.

2.1.3 Tight coupling

In addition to coupling between materials, we are interested in tight coupling between elastic and contact forces. For some phenomena, such as stacking objects in a pile (See Figure 2.12, where a rigid body is stacked on three deformable (Lagrangian or MPM) objects), the coupling between elastic forces, normal contact forces, and friction plays a critical role in the resulting dynamics and long-term stack stability. If the bottom objects move out, then the top object will move down; this downward motion is progress towards collapse which is physically irreversible due to conservation of energy. A method that treated friction as a post-process would allow the pile to fall slightly (the correct behavior in the absence of

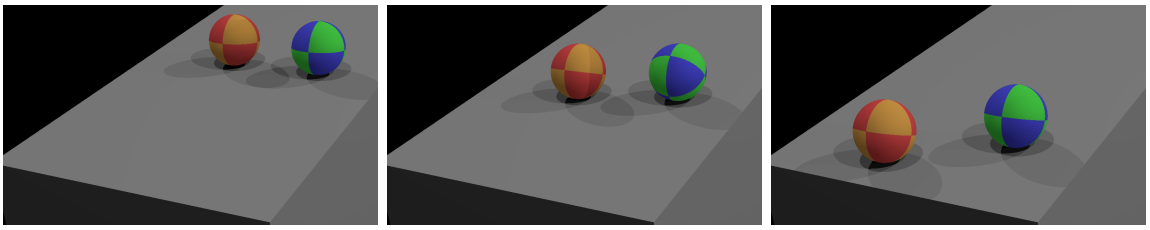


Figure 2.4: Our frictional contact force handles rolling friction automatically. The blue/green sphere rolls due to friction, while the red/orange sphere has a very low friction coefficient and slides. Note that the orange sphere falls down the incline faster, as would be expected. ©2019 IEEE

friction) converting the gravitational potential into kinetic energy, which is then dissipated as friction during the post-process. An iterative method (impulse-based, partitioned) can leave small residual errors, which will accumulate over time until collapse. Even many analytic methods for systems with only rigid bodies struggle with long-term pile stability [97]. A similar problem occurs with wedging (see Figure 2.5).

2.1.4 Our contribution

We propose a novel penalty force with the following properties.

- The force accurately enforces Coulomb friction, including both static and dynamic friction as well as transitions between them. Where this cannot be done, the force falls back gracefully.
- The force is compatible with use with fully-implicit time integration, including the special requirements of line-search-based methods (unlike existing penalty methods).
- The force can be used to couple rigid bodies to deformable objects or to the material point method.

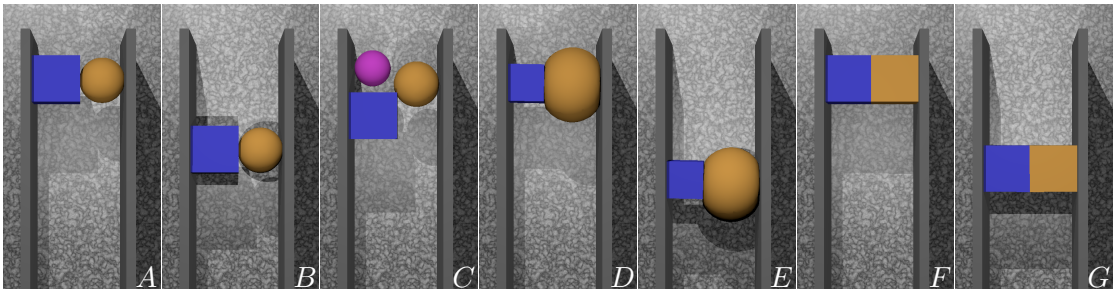


Figure 2.5: Our forces pass wedge tests, where objects are prevented from falling by virtue of being wedged between two parallel plates with friction. The first three figures shows a wedged deformable/rigid configuration (A); if friction is reduced, the objects slide (B). The wedged objects may also be dislodged by other collisions (C). We also demonstrate wedging with MPM/rigid (D and E) and rigid/rigid (F and G), first in a wedged configuration and then sliding when friction is reduced. The blue blocks and magenta spheres are rigid. ©2019 IEEE

- The force naturally produces stable stacking with zero drift over time, even if implicit integration is solved only approximately (unlike impulse-based methods or most existing constraint-based methods).

- We demonstrate a practical implicit stabilized Newton-based solver capable of solving the system of equations that results from backward Euler. The equations are especially challenging due to the inclusion of friction and are non-smooth with asymmetrical derivatives. The solver is also capable of stabilizing elastoplastic simulations, such as those that arise from the Drucker-Prager constitutive model for sand.

2.2 Penalty force

We propose a penalty friction force to model collisions and contacts between *particles* and *objects*. The flexibility of the proposed force comes from the variety of representations that can be treated as particles or objects.

Particles. The particles can be almost anything Lagrangian. In our examples, we use (1) degrees of freedom from a deformable object simulation, (2) particles from a material point method (MPM) simulation, and (3) vertices from the surface mesh of a rigid body. Note that in cases (2) and (3), the *particles* are distinct from the actual degrees of freedom on which the integration of forces will occur (grid nodes for MPM, velocity and angular velocity for rigid bodies). It is sufficient for the velocities (and positions) of these *particles* to be computed from the degrees of freedom. Following the principle of virtual work, we use the transpose of this mapping to apply forces.

Objects. Suitable choices for the objects are a bit more restrictive, since the formulation of the penalty force must be tailored to properties of the object. We formulate versions of the penalty force for level sets and triangular surface meshes. Using these, we demonstrate collisions against fixed objects and boundaries, rigid bodies, and deformable objects. We use these options to demonstrate a range of capabilities, including deformable object self-collisions, rigid-rigid collisions and contacts, rigid-deformable coupling, rigid-MPM coupling, and handling of collisions with fixed objects for all simulation types.

Spring force. We formulate our force as a spring force connecting the *particle* (at location $\mathbf{Z}$) with an *attachment point* on the object (at location $\mathbf{X}$). The attachment point is able to slide along the surface of the object to a potentially new location $\mathbf{Y}$, a process we will refer to as *relaxation*. For the force itself we simply use a zero-length spring, where the force applied to the particle $\mathbf{Z}$ is $\mathbf{f} = k(\mathbf{Y} - \mathbf{Z})$. If the object is dynamic, an equal and opposite force is applied at the relaxed attachment location $\mathbf{Y}$. The constant k is the stiffness of the

penalty force, which we choose as a compromise between the depth of penetration and the amount of stiffness in the resulting system.

Enforcing friction. The attachment point may be thought of as the place that the particle *should* be, and the spring applies a force on the particle to pull it there. The attachment point behaves as a massless point that collides with the object by exchanging normal and tangential forces subject to Coulomb friction. This massless point is not solved for as a degree of freedom; rather, its position is computed by *relaxing* from its original location $\mathbf{X}$ to a new location $\mathbf{Y}$ which satisfies the Coulomb friction cone. This relaxation step is a nonlinear projection operation (at least approximately), which we will denote as $\mathbf{Y} = \mathbf{P}(\mathbf{X}, \mathbf{Z})$. Eliminating $\mathbf{Y}$, we get the penalty force $\mathbf{f} = k(\mathbf{P}(\mathbf{X}, \mathbf{Z}) - \mathbf{Z})$, where $\mathbf{X}$ is fixed and k is a constant. Note that the projection operator $\mathbf{P}$ will depend on the degrees of freedom of the object, which may itself be dynamic. At the end of the time step, we update the attachment point ($\mathbf{X}^n \rightarrow \mathbf{X}^{n+1}$) using $\mathbf{X}^{n+1} = \mathbf{Y} = \mathbf{P}(\mathbf{X}^n, \mathbf{Z}^{n+1})$. What remains is to formulate this projection, which takes different forms for different types of objects. We provide pseudocode in a separate technical document for the force and relaxation routines, as well as their derivatives. To simplify notation, we will omit superscripts on $\mathbf{X}$, $\mathbf{Y}$, and $\mathbf{Z}$. The original attachment location $\mathbf{X}$ is fixed, the particle position $\mathbf{Z}$ is being computed by a newton solve, and the relaxed attachment location $\mathbf{Y}$ is computed when needed from $\mathbf{X}$ and $\mathbf{Z}$.

Representation of attachments. In the case of moving objects, the attachment point must be fixed to the object in a meaningful way. For rigid bodies, we store the attachment point in the object space of the rigid body. For deformable objects, we represent

the attachment by its barycentric coordinates in the triangle that the particle is colliding with. In this way, attachment points naturally move with the object; when static friction is being applied, the representations of attachment points are untouched.

Mechanism of stable stacking. The means by which stable stacking is achieved is worth emphasizing. In constraint-based or impulse-based formulations, small movements of colliding points across a surface (due to, among other things, convergence error) may lead to long-term drift of the colliding point over the surface. This may in turn cause piles or stacks to collapse. Methods that use penalty forces with attachment points behave quite differently. The colliding particles are still able to move around (in both normal and tangential directions), but these particles are tethered to an attachment point. As long as the attachment point is prevented from moving relative to the object (which we accomplish by storing the attachment *in the reference space of the object*), no long-term progress towards collapse is possible. The particle can only move around in a small region near its attachment point.

2.2.1 Relaxation - properties

Let $\mathbf{Y} = \mathbf{P}(\mathbf{X}, \mathbf{Z})$ be the relaxed (projected) attachment point and $\mathbf{f} = k(\mathbf{Y} - \mathbf{Z})$ the resulting force. If $\mathbf{n} = \mathbf{n}(\mathbf{Y})$ is the local normal direction of the object at the projected attachment location, then our point should satisfy the following properties.

- $\mathbf{Y}$ is on the surface on the object.
- The projection is idempotent: $\mathbf{Y} = \mathbf{P}(\mathbf{Y}, \mathbf{Z})$.
- The friction cone is satisfied: $\|\mathbf{f} - (\mathbf{f} \cdot \mathbf{n})\mathbf{n}\| \leq \mu \mathbf{f} \cdot \mathbf{n}$.

- The attachment point should not be relaxed further along the surface than necessary to satisfy the friction cone.

Note that the last requirement is a slightly weakened form of the usual complementary condition, which is necessary to accommodate a non-smooth object surface. Since the force is linear in positions, the friction constraint is equivalent to the more convenient geometrical constraint

$$\|(\mathbf{Y}-\mathbf{Z})-((\mathbf{Y}-\mathbf{Z})\cdot\mathbf{n})\mathbf{n}\| \leq \mu(\mathbf{Y}-\mathbf{Z})\cdot\mathbf{n}. \tag{2.1}$$

The choice $\mathbf{n}=\mathbf{n}(\mathbf{Y})$ was not the only option available for computing a normal direction; Figure 2.6 motivates the appropriateness of this choice.

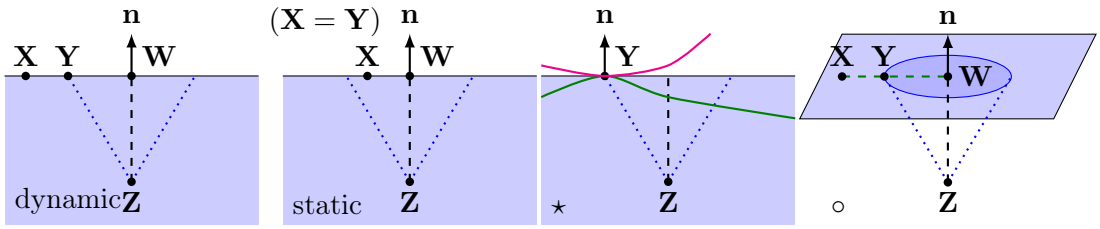


Figure 2.6: A particle $\mathbf{Z}$ as pulled to the surface by an attachment point $\mathbf{X}$. In dynamic friction, the attachment point $\mathbf{X}$ is outside the friction cone (dotted) and relaxes along the surface to the friction cone $\mathbf{Y}$, resulting in a penalty force that satisfies Coulomb friction. In static friction, the attachment point $\mathbf{X}$ is already inside the friction cone and does not move ($\mathbf{X}=\mathbf{Y}$). In the non-planar case ($\star$), there is not a single constant normal direction. Intuitively, the attachment at $\mathbf{Y}$ should lie on the friction cone for the planar surface as well as the green and red curved surfaces. That is, friction is a local property that depends on the surface at only one location. Viewed in 3D ($\circ$), the attachment $\mathbf{X}$ is relaxed towards the closest point on the plane $\mathbf{W}$ until it reaches the cone at $\mathbf{Y}$, since the projection of a force from $\mathbf{X}$ to $\mathbf{Z}$ onto the surface points in this direction. ©2019 IEEE

Relaxation with a plane

We begin with the simple case where the object is a plane (See Figure 2.6). If (2.1) is satisfied with $\mathbf{Y} = \mathbf{X}$, then the attachment experiences static friction and does not move. Otherwise, the attachment $\mathbf{X}$ should be moved along the surface by the penalty force $\mathbf{f}$ until (2.1) is satisfied. Let $\mathbf{W} = \mathbf{Z} - \phi(\mathbf{Z})\mathbf{n}$ be the closest point on the plane to $\mathbf{Z}$, where ϕ is the level set function for the planar object. This force pulls $\mathbf{X}$ along the surface towards $\mathbf{W}$ to the point $\mathbf{Y} = \mathbf{W} + s(\mathbf{X} - \mathbf{W})$, where s is the largest scalar $0 \leq s \leq 1$ satisfying (2.1). Solving this equation yields

$$s = \mu \frac{\|\mathbf{Z} - \mathbf{W}\|}{\|\mathbf{X} - \mathbf{W}\|}. \tag{2.2}$$

Relaxation with level set

If we are colliding a particle with a level set, and that level set represents a plane, then we have a simple algorithm to compute $\mathbf{Y}$ directly. More generally, the level set will not be flat. Relaxing the attachment based on the local force would amount to solving an ODE, which is unnecessarily complicated. Instead, we relax the attachment point towards the closest point $\mathbf{W} = \mathbf{Z} - \phi(\mathbf{Z})\mathbf{n}(\mathbf{Z})$ on the surface of the object as we did in the plane case. Unlike the plane case, a point $\mathbf{K} = \mathbf{W} + s(\mathbf{X} - \mathbf{W})$ along the segment connecting two points $\mathbf{X}$ and $\mathbf{W}$ on the surface need not be on the surface. We project this point to the surface to compute the desired relaxed point $\mathbf{Y} = \mathbf{K} - \phi(\mathbf{K})\mathbf{n}(\mathbf{K})$.

The attachment point $\mathbf{Y}(s)$ is a nonlinear function of a single scalar $0 \leq s \leq 1$. Let

$$g(s) = ((\mathbf{Y} - \mathbf{Z}) \cdot \mathbf{n}(\mathbf{K}))^2 - \bar{\mu}\|\mathbf{Y} - \mathbf{Z}\|^2 \qquad \bar{\mu} = \frac{1}{\mu^2 + 1}.$$

Then $g(s) \geq 0$ is equivalent to (2.1). If $g(1) \geq 0$ then $\mathbf{Y}(1) = \mathbf{K} = \mathbf{X}$ satisfies (2.1); the attachment experiences static friction and does not move. Otherwise, $g(1) < 0$. Observe that $\mathbf{Y}(0) = \mathbf{W}$ and $\mathbf{n}(\mathbf{W}) = \mathbf{n}(\mathbf{Z})$ so that $g(0) = (1 - \bar{\mu})\phi(\mathbf{Z})^2 \geq 0$. If $g(s)$ is continuous, a suitable s must exist, and a method such as bisection may be used to compute it. In practice, $g(s)$ need not be continuous (e.g., at sonic points of the level set), and bisection may terminate at a discontinuity. In this case, the friction cone is only weakly satisfied. While differentiating this procedure is manageable, doing so robustly is quite difficult.

Instead, we observe that we are typically only moving points a small distance during a time step. Locally, the level set should look like a plane. Motivated by this, we simply compute s using (2.2) rather than by bisection. This approximation gives up the projection property and means that Coulomb friction is only approximately satisfied, but it seems to be a good comprise in practice.

Initial attachment location. When a collision first occurs, an attachment location $\mathbf{X}$ must be chosen. Ideally, one would chose $\mathbf{X}$ to be the location on the surface of the level set where the particle crossed it. To avoid complications for rigid bodies, we simply use $\mathbf{X} = \mathbf{Z} - \phi(\mathbf{Z})\mathbf{n}(\mathbf{Z})$; this contrasts with the CCD case, where a good initial attachment location is readily available. While this is less accurate than using the point of entry, this only introduces an error in the time step in which the collision is first encountered. In sub-

sequent time steps, the attachment location $\mathbf{X}$ will not move relative to the object (static friction) or will relax across the surface (dynamic friction) according to Coulomb friction.

Note that no friction will be applied when the particle is at the location where the collision was first detected (or more generally whenever the particle happens to be below the attachment point). The lack of force under these conditions does not mean the particle is not being tethered to this attachment location. The particle will feel strong frictional forces if it ever attempts to leave the vicinity of the attachment point. This is not particularly surprising; a stationary box on a level surface also experiences zero frictional force. The box will only experience non-zero frictional forces if one tries to move it.

Limitations of the level set formulation. The primary limitation of the level set formulation is the discontinuities caused by sonic points. The formulation works well when the surface has low curvature. This keeps sonic points away from the surface and also makes the approximate computation of s more accurate.

Relaxation with surface mesh

When colliding against a level set, we can immediately determine whether a collision is occurring and project to the closest point on the surface. When we are colliding with a triangulated surface mesh, these operations are more expensive.

Detection. We use continuous collision detection (CCD) to detect when a particle has penetrated a triangulated surface. This conveniently gives us the barycentric coordinates for the point of entry as well, which we use to set the initial attachment $\mathbf{X}$. CCD can normally only be employed to detect new collisions, and CCD algorithms expend great effort to ensure that a collision-free state is maintained. This is because CCD will not detect a colliding

particle if it was already colliding. In our case, a collision-free state is not maintained; since our force is a penalty force, forces are only applied if a small amount of penetration is retained. This is not a problem for us, since CCD will correctly flag new collisions without a collision-free state. We do not need CCD to tell us about existing collisions, since we keep a record of these interactions anyway (we must store attachment locations for them). Depending on the implementation, CCD may also detect when a colliding particle exits the surface; these can be easily detected by checking the triangle normal and may be safely ignored.

Intuition for relaxation. As intuition for the relaxation process we formulate, imagine that the attachment point is a block sitting on the surface and connected to the colliding point $\mathbf{Z}$ with an actual spring. The block experiences Coulomb friction against the surface mesh. Ignoring inertial effects, the attachment point should slide along the surface of the triangle mesh along the direction it is being pulled by the spring force. This process terminates when the friction cone is satisfied. At this point, we check to see if $\mathbf{Z}$ is pulling $\mathbf{Y}$ into the surface or away from the surface. If the latter, we flag the attachment as inactive and do not apply forces to it.

Convexity consideration. In the case of non-convex objects, one has to make a choice about when the separation test should be performed, since an attachment may be pulled away from the object during relaxation, even though the colliding point is in fact inside the object. One option would be for the attachment to be able to separate from the surface during relaxation (and possibly re-collide with it). Another option is to delay the separation test until relaxation has completed. In practice, we found the latter option to

be simpler to implement, more efficient, and more effective. See the technical document for details of the separation tests.

Finite state machine.

We implement the resulting relaxation algorithm as a finite state machine (FSM) with four states: (1) the attachment point is in the interior of a triangle, (2) the attachment point is on an edge of a triangle, (3) attachment point is on an edge and moves along it, and (4) the attachment point is at a vertex and tries to leave it via a neighboring primitive. At each step, we start with an initial attachment $\mathbf{Y}^{(k)}$ on the current primitive and compute a new attachment $\mathbf{Y}^{(k+1)}$ location. The algorithm starts in state (1) with $\mathbf{Y}^{(0)} = \mathbf{X}$. When the algorithm terminates at step n, we set $\mathbf{Y} = \mathbf{Y}^{(n)}$. These states are illustrated in Figure 2.8 and Figure 2.9 and described in detail below. The transitions are illustrated in Figure 2.7.

Triangle cases.

Cases (1) and (2) are nearly identical. In both cases, a relaxation $\mathbf{Y}^{(k)} \to \hat{\mathbf{X}}$ is performed along the plane of the triangle in the same way as for a planar level set. If the computed attachment point $\hat{\mathbf{Y}}$ lies inside the triangle, then the algorithm terminates with

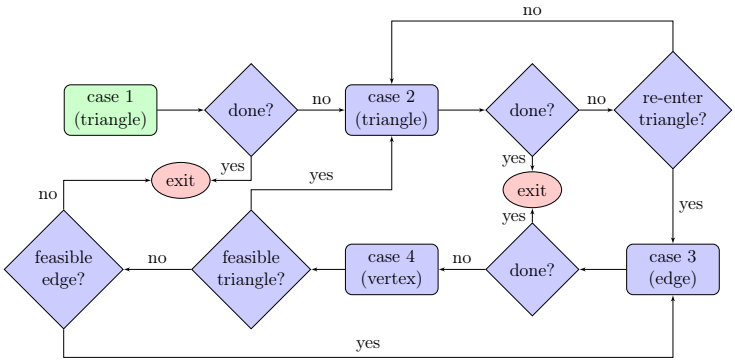


Figure 2.7: This figure shows the case transitions for the mesh-based relaxation algorithm. The algorithm begins at the green node and ends when reaching a red node. ©2019 IEEE

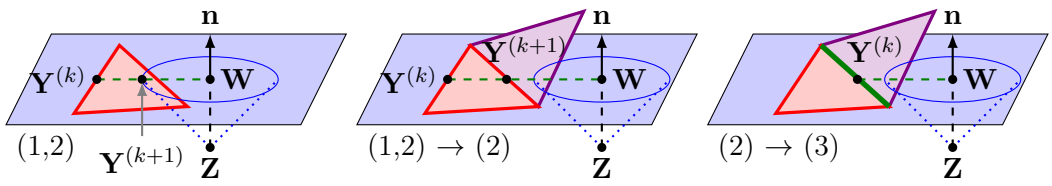


Figure 2.8: In the triangle cases (1,2), the attachment starts at $\mathbf{Y}^{(k)}$, which is in the interior (case (1)) or on the edge (case (2)). The attachment is relaxed within the plane of the triangle; if the friction cone is reached within the triangle (left), the relaxation terminates. Otherwise, the attachment relaxes to the location where it leaves the triangle (middle); the algorithm enters case (2) with the neighboring violet triangle. In case (2), the attachment starts on an edge; if it is drawn outside the triangle, then the algorithm transitions to case (3) to relax along the green edge (right). ©2019 IEEE

$\mathbf{Y} = \mathbf{Y}^{(k+1)} = \hat{\mathbf{Y}}$. Otherwise, we draw a segment from $\mathbf{Y}^{(k)}$ to $\hat{\mathbf{Y}}$ and intersect it with the triangle's boundary. The intersection location is $\mathbf{Y}^{(k+1)}$. If we are in case (1) or the edge intersected is not the same one we entered on (so that $\mathbf{Y}^{(k)} \neq \mathbf{Y}^{(k+1)}$), then we transition to state (2). Otherwise, we are stuck on an edge and unable to make progress along either adjacent triangle. We may still be able to make progress along the edge, and we transition to state (3). See Figure 2.8.

Edge case. The edge case (3) is similar to case (1), except that the surface point $\mathbf{W}$ is computed as the closest point on the line containing the edge. If the computed attachment point $\hat{\mathbf{Y}}$ lies inside the segment, then the algorithm terminates with $\mathbf{Y} = \mathbf{Y}^{(k+1)} = \hat{\mathbf{Y}}$. Otherwise, we transition to case (4). See Figure 2.9.

Vertex case. The vertex case (4) is quite different from the other cases, since the attachment point does not move. Rather, the goal of this state is merely to find a way to leave it. First, we check the triangles adjacent to the vertex. If the spring force would pull the attachment point into the triangle, then we transition to this triangle in state (2) with

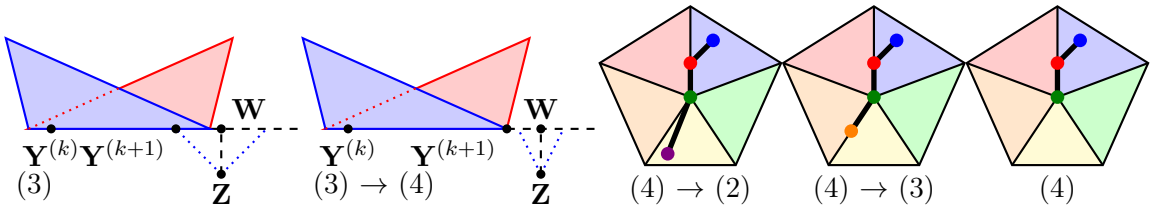


Figure 2.9: In case (3), an attachment starts on an edge $\mathbf{Y}^{(k)}$ and relaxes towards the point $\mathbf{W}$ along the edge closest to the particle $\mathbf{Z}$. If the friction cone is reached along the edge, then the relaxation terminates (left). Otherwise, the attachment relaxes to a vertex and the algorithm resumes in case (4) by trying to leave the vertex (left middle). Case (4) handles relaxation through a vertex. This case is reached when an attachment (blue) gets stuck on an edge (red) and relaxes along it to a vertex (green). Case (4) transitions to case (2) if progress can be made (violet) in one of the neighboring triangles (middle). Otherwise case (4) transitions to case (3) if progress can be made (orange) along one of the neighboring edges (right middle). If no progress can be made, relaxation terminates (right). ©2019 IEEE

$\mathbf{Y}^{(k+1)} = \mathbf{Y}^{(k)}$. Note that it is not necessary to compute the actual attachment location in this triangle. If no progress can be made in a triangle, then we must check the adjacent edges. If the spring force would pull the attachment point along an edge, then we transition to this edge in state (3) with $\mathbf{Y}^{(k+1)} = \mathbf{Y}^{(k)}$. Otherwise, the attachment point would not be pulled from the vertex, and the algorithm terminates with $\mathbf{Y} = \mathbf{Y}^{(k)}$. See Figure 2.9.

Termination considerations. Observe that if the attachment moves ($\mathbf{Y}^{(k)} \neq \mathbf{Y}^{(k+1)}$) then $\|\mathbf{Y}^{(k+1)} - \mathbf{Z}\| < \|\mathbf{Y}^{(k)} - \mathbf{Z}\|$. That is, the attachment point is being drawn closer to $\mathbf{Z}$. As as long as the attachment can be prevented from stalling in one spot, one can prove that primitives are visited at most a finite number of times (see the technical document for a proof). Assuming the triangle mesh is not degenerate, this can only occur when the attachment point is at a vertex. This can happen in practice due to round-off error. This can be easily checked, since the barycentric coordinates of the embedding are computed during case (2). We make two modifications to the algorithm to address this

possibility: transition to case (4) if a barycentric weight is larger than $1 - 128\epsilon$ where ϵ is the floating point epsilon, and only transition away from case (4) if the test for progress is significantly larger than round-off error.

2.2.2 Collision detection

For level-set-based forces we detect collisions using the level set. We accelerate the search by rasterizing particles and level sets to regular sparse grids. For CCD-based forces, collisions are detected using CCD without a collision-free state. We use bounding box hierarchies to accelerate the search. In either case, we maintain hash tables of known pairs, which we use to avoid repeatedly registering the same pairs. In the case of CCD-based forces, we perform the hash table check on bounding box candidates before solving the cubic, since the hash table lookup is significantly cheaper. Once registered, each collision pair is retained along with its current attachment point until the collision becomes inactive. At the end of each time step, inactive collision pairs are pruned and attachment points for the remaining pairs are updated to their newly relaxed locations. Note that this postprocessing step is merely making permanent the decisions already made during the Newton solver. Collision pairs that are pruned at the end of the time step were inactive at the end of the Newton solve and thus were applying no force.

2.3 Stabilized Newton solver

We discretize our equations of motion using backward Euler, which leads to the nonlinear problem

$$\mathbf{M}\frac{\mathbf{v}^{n+1}-\mathbf{v}^n}{\Delta t} = \mathbf{f}(\mathbf{x}^{n+1}) \qquad\qquad \mathbf{x}^{n+1} = \mathbf{x}^n + \Delta t \mathbf{v}^{n+1},$$

where $\mathbf{M}$ is the lumped mass matrix and $\mathbf{f}(\mathbf{x})$ are our forces (gravity, internal, and collision forces). Using $\Delta\mathbf{v} = \mathbf{v}^{n+1} - \mathbf{v}^n$ as our degrees of freedom, we have $\mathbf{g}(\Delta\mathbf{v}) = \mathbf{M}\Delta\mathbf{v} - \Delta t\mathbf{f}(\mathbf{x}^n + \Delta t\mathbf{v}^n + \Delta t\Delta\mathbf{v}) = \mathbf{0}$. For simplicity, we refer to the solution vector as $\mathbf{z}$, where $\mathbf{z} = \Delta\mathbf{v}$. We refer to the accompanying technical document for details on the computation of $\mathbf{g}$ and its derivative.

Our contact formulation produces nonlinear systems that result in asymmetrical linear systems. We found that stabilizing our Newton solver greatly improved solver reliability. Stabilizing the solver had the added benefit of facilitating debugging, since it allows convergence problems to be tracked down to a definite source.

When using Newton's method to solve $\mathbf{g}(\mathbf{z}) = \mathbf{0}$, where $\mathbf{H} = \nabla\mathbf{g}$, we must repeatedly solve $\Delta\mathbf{z} = -\mathbf{H}^{-1}\mathbf{g}$ to obtain a series of corrections. For our forces, it is easy to see that $\mathbf{H}$ is not symmetric. This implies that $\mathbf{g} = \mathbf{0}$ does not correspond to minimizing anything, since otherwise $\mathbf{H}$ would be a Hessian of this objective and thus symmetric. This immediately rules out methods such as [60].

One known alternative is to minimize $E = \frac{1}{2}\|\mathbf{g}\|^2$ instead [132]. If $\mathbf{g} = \mathbf{0}$ has a solution, then clearly this will be a global minimum for E, at which point $E = 0$. One could

optimize E directly, but this would involve computing the Hessian of E, which would in turn require us to compute second derivatives of forces.

Instead, we note that one can actually minimize E while continuing to use the line search direction $\Delta\mathbf{z} = -\mathbf{H}^{-1}\mathbf{g}$, as was done in [18]. This direction is usually very desirable; it is the direction we would use if we could not stabilize the solver. To be effective, we need this direction $\Delta\mathbf{z}$ to be a downhill direction for our new objective E. Consider that a step will be taken in some direction $\mathbf{u}$. The directional derivative of E in this direction is $\nabla E \cdot \mathbf{u} = \mathbf{g}^T\mathbf{H}\mathbf{u}$. Thus, a direction $\mathbf{u}$ is a downhill direction if $\mathbf{g}^T\mathbf{H}\mathbf{u} < 0$. We can immediately see that the Newton direction is *always* such a direction, since $\mathbf{g}^T\mathbf{H}\Delta\mathbf{z} = -\mathbf{g}^T\mathbf{H}\mathbf{H}^{-1}\mathbf{g} = -\mathbf{g}^T\mathbf{g} < 0$ (unless $\mathbf{g} = \mathbf{0}$, in which case we are done). This is not true of the standard objective, where this guarantee only holds where $\mathbf{H}$ is positive definite. In practice, computing the Newton direction exactly would be too expensive, and it need not actually exist (for example if $\mathbf{H}$ is singular). Instead, we test to see if it is downhill directly and choose a suitable fallback if it is not (see its accompanying technical document). Finally, we perform Wolfe condition line searches on E in the usual way. Our termination condition for Newton's method is that $\min(\|\mathbf{g}\|, \|\mathbf{H}^T\mathbf{g}\|) < \tau$, where τ is our Newton tolerance. Checking $\|\mathbf{g}\| < \tau$ ensures that we have an accurate solution to our problem (solve $\mathbf{g} = \mathbf{0}$). The test $\|\mathbf{H}^T\mathbf{g}\| < \tau$ ensures that we also have a good solution to the problem of minimizing E, the problem that the line search is trying to solve. Pseudocode for our solver is provided in the accompanying technical document.

While this method is only guaranteed to converge to a local optimum, in practice we have never observed convergence to anything other than an optimal solution. The sonic

points of level sets cause force discontinuities, which in turn lead to discontinuities along the line search. This may cause convergence to fail for thin or sharp objects when using level sets. For this reason, we favor the CCD-based formulation for such objects.

2.4 Results

In our tests deformable objects in tetrahedral volume adopt fixed co-rotated constitutive model [159], with Young's modulus $E = 10^6$ and Poisson's ratio $\nu = 0.45$. Deformable objects in MPM material adopts the same model but with Young's modulus $E = 10^3$ and Poisson's ratio $\nu = 0.3$. Cloth uses mass-spring model [29], with linear stiffness $1000/(1+\sqrt{2})$ and bending stiffness $200/(1+\sqrt{2})$. The sand uses the Drucker-Prager model from [100] with Young's modulus $E = 35.37 \times 10^6$ and Poisson's ratio $\nu = 0.3$. We set tolerance of Newton's Method to be 1 by default, and use smaller tolerance in some tests to avoid visual artifacts. Detail configurations are shown in Table 2.1. The maximum number of Newton iterations is not limited (we set it to 1000). Based on our measurement the average number of iterations is 3.1 per time step.

Analytic friction tests. The heart of our MPM-rigid coupling method method is our penalty force for applying contact and friction. We show that our friction is accurate by comparing our force against the classical analytic solution for a point mass sliding along an inclined plane. In Figure 2.3 we slide a block down an inclined plane with a variety of different object types (MPM, rigid bodies, and deformable objects), shown next to a proxy for the analytic solution. This demonstrates that we achieve accurate friction regardless of object representation. In Figure 2.10, we show quantitative agreement with the analytic

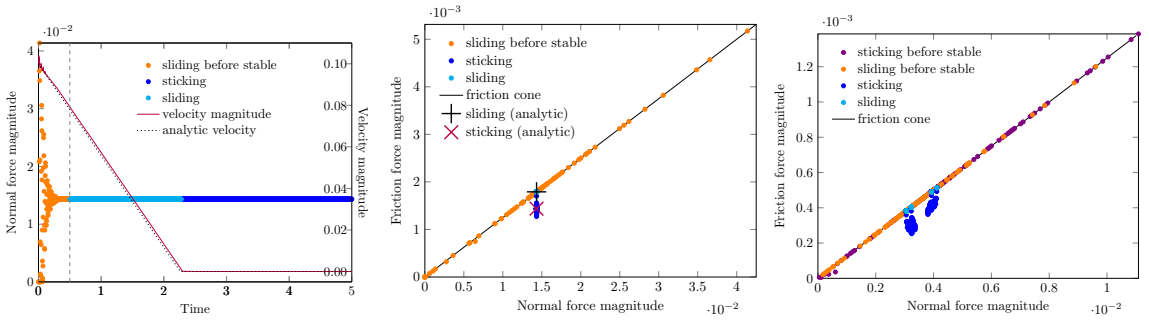


Figure 2.10: In this figure, we compare our simulation of a deformable cube on an inclined plane (the green cube in Figure 2.3, but with Newton tolerance 10^{-3}) with the analytic solution. (Left) In this plot, we show computed total normal contact force for the cube over time. At the beginning of the simulation the forces (•) are variable due to the bouncing of the cube as it falls from its initial stress-free configuration onto the inclined plane. After the cube has settled, the cube slides (•) until coming to rest (•). Superimposed are the analytic (·····) and computed (—) velocity magnitudes for the cube, showing close agreement between computed and analytic solution. Note that the abrupt change in the velocity magnitude slope corresponds with the switch from dynamic to static friction. (Middle) Here, we plot total normal and total tangential contact forces for the cube. During the settling phase (•), dynamic friction forces are variable but always on the friction cone (—). Afterwards, the computed dynamic (•) and static (•) friction forces closely match the analytic solution (+ and ×) and obey the Coulomb friction cone. (Right) Here, we plot total normal and total tangential contact forces per simulation vertex. During the settling phase (static •, dynamic •), friction closely tracks but never violates the friction cone (—). Once settled, the cube experiences dynamic (•) followed by static (•) friction. As expected, dynamic friction lies on the friction cone, and static friction lies below it. Note that the different vertices of the cube carry different amounts of the cube's mass, so the per-vertex contact force magnitudes cluster around different values. Our penalty contact force never violates the Coulomb friction cone. ©2019 IEEE

solution. This figure also demonstrates an important property of our method, that normal and frictional forces always obey the Coulomb friction cone. We have run this comparison with a tighter Newton tolerance to reduce deviations from the analytic solution caused by solver accuracy. In all of our inclined plane tests, the block is given an initial velocity $0.1m/s$ along the inclined plane. Depending on the friction used (see Table 2.1), the block continues sliding ($\mu = 0.1$) or comes to rest ($\mu = 0.125$). Our method naturally causes rigid bodies to roll (see Figure 2.4).

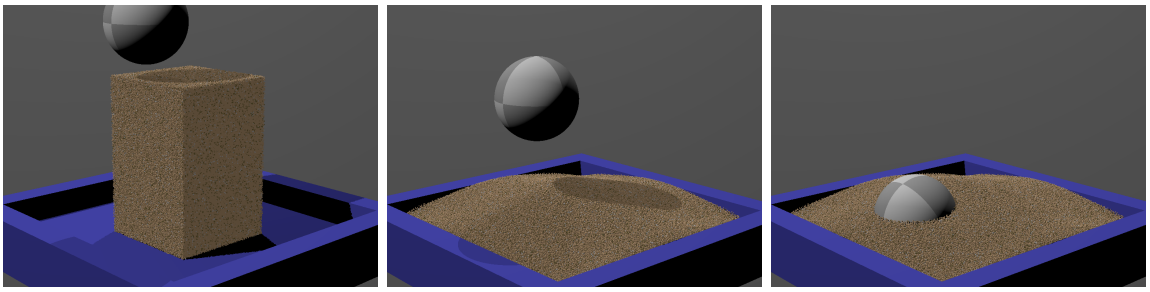


Figure 2.11: Sand is dropped into a pile, and then a heavy sphere is dropped on time, penetrating into the sand. ©2019 IEEE

Stacking. One of the effects that is difficult to handle correctly when elastic forces, friction, and contacts are not performed together is stacking. Our method produces stable stacking (see Figure 2.3). Here we stack five blocks (a mixture of rigid and deformable objects), which stands stably and never falls (it remained standing for 100,000 frames, by which time it had come to rest). As long as contacts are in the sticking state, the attachments will never move. Because of this, the contacts will never drift, even over long periods of time. In Figure 2.12, we form a pyramid by stacking a rigid ball on deformable or MPM objects. The stacks are stable and do not drift over time. This is a difficult test for many methods to pass. If elastic forces are evolved without friction, the objects at the base will slide out slightly, and the sphere will make some progress downward. This progress will not be corrected when friction is applied to the base objects. If the sphere is able to make downward progress or the base objects are able to make outward progress, the pyramid will eventually collapse. We tested the pyramid stack using the method of [154] and verified that it does indeed creep to collapse (see video).

Wedging tests. Another test that is difficult to pass without coupling between elastic forces, contact, and friction is the wedging test. In this test, two objects are squeezed

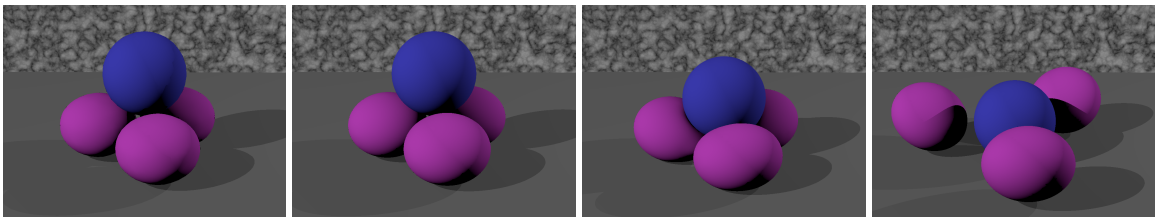


Figure 2.12: Our method maintains stable stacks in configurations that rely on force balance between elastic forces, contact, and friction. Stable stacking is shown with rigid/deformable (left) and MPM/rigid (left middle). If friction is reduced, the stack collapses as expected (right two). ©2019 IEEE

by two fixed parallel vertical walls (see Figure 2.5), and gravity is applied. Initially the objects overlap with the walls a little bit. This small penetration generates penalty forces. With adequate friction, the objects should be able to jam in place and never slide down the wall. As with the pyramid stack, the key to long term stability in this case is preventing the objects from making downward progress once they have become jammed. This jamming depends on the interaction of all three types of forces. We demonstrate jamming at higher friction and that sliding is recovered if friction is lowered.

Coupling with complex materials. Like [40], we are able to couple rigid bodies with sand. In Figure 2.2, we throw a cube over a sand box. In Figure 2.11, we roll a sphere down a sand pile. In Figure 2.16, demonstrate that pouring sand on a sphere causes that sphere to roll, thereby demonstrating the frictional forces between them. In Figure 2.13, demonstrate that we are able to couple to complex MPM materials other than sand. In Figure 2.1, we demonstrate that we can scale to large numbers of objects.

General tests. Our method handles dynamic scenes, non-convex objects, and interactions between different materials. In Figure 2.14, we demonstrate compatibility with

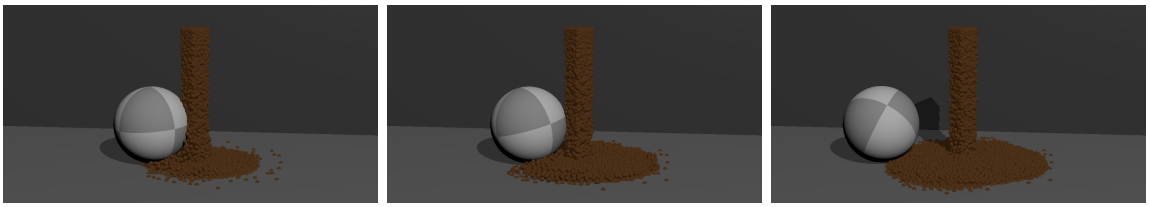


Figure 2.13: A complex MPM material interacts with a rigid sphere. ©2019 IEEE

cloth. In Figure 2.15, we demonstrate that deformable-deformable and deformable-rigid contacts also work correctly with non-convex objects.

To judge the relative cost of our collision handling in comparison to other parts of the algorithm, we computed a breakdown of the runtime cost for three representative examples: MPM "sand_on_sphere" (Figure 2.16), non-convex rigid/deformable "torus_dr" (Figure 2.15), and rigid/deformable/cloth "cloth" (Figure 2.14). For the MPM example, collision-related steps added only minor cost (4.0% for CCD, 0.6% for computing collision forces). The significant majority of the time was spend computing and applying internal sand forces (90.3%), with the remaining steps taking about 5% of the total time. The low cost of collisions in this example is due to three factors: the large number of particles involved in the sand, the relatively low number of particles close enough to collision objects to be involved in collision processing, and the cost of particle/grid transfers. For the non-MPM tests, continuous collision detection is a major part of the total cost (torus: 43.5%, cloth: 54.0%). In both cases, calculating collision forces, computing and applying collision force derivatives, and performing relaxation are minor (less than 2%). Computing and applying internal forces and other solver-related steps contribute most of the rest (torus: 55.4%, cloth: 44.0%).

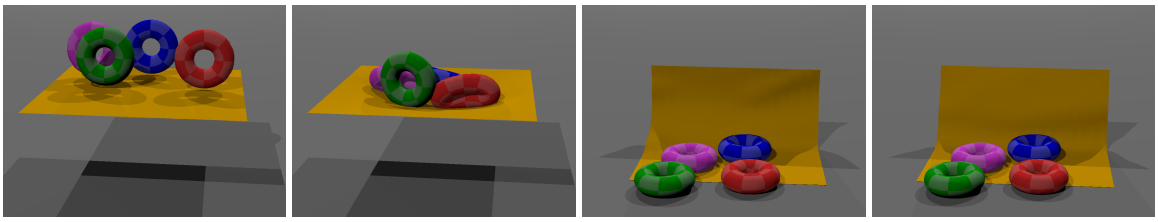


Figure 2.14: Two deformable and two rigid bodies are dropped onto a piece of cloth. ©2019 IEEE

To evaluate the convergence of the relaxation procedure, we consider the breakdown of states encountered during the non-convex example Figure 2.15. The relaxation nearly always terminates at the starting state (approximately 99.77% of the time). Thus for performance purposes, the algorithm completes in one step, and the history that must be stored for derivatives is of constant size (see the technical document for details). For robustness, however, the remaining 0.23% cases must still be handled reliably. An single extra step (case $1 \to 2$) was taken in nearly all of the remaining cases (0.21%). The other case transitions that were observed in this example were (in order from most to least frequent): $1 \to 2 \to 3$, $1 \to 2 \to 2$, $1 \to 2 \to 2 \to 2$, and finally $1 \to 2 \to 3 \to 4 \to 2$ (which occurred three times in this simulation). No relaxation terminated in more than 5 steps for this example. This example illustrates that it is possible for the attachment to relax across multiple triangles in a single time step (up to four triangles in this example).

The constant k determines the stiffness of the penalty forces and is an important parameter in any penalty force model. Reducing k results in deep penetration, which produces observable artifacts and must be avoided. Increasing k produces linear systems with poorer conditioning, which makes solvers less accurate and slower to converge. Reduced accuracy from the linear solver is not a major concern in our case, since these errors will be

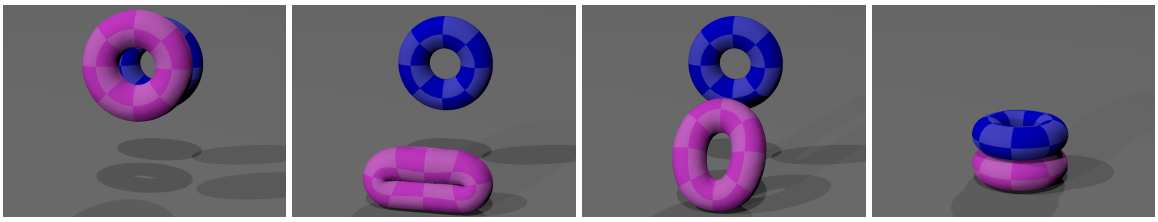


Figure 2.15: Two tori (magenta is deformable, blue is rigid) are dropped to the ground, demonstrating the ability to handle non-convex objects, including self-collisions. ©2019 IEEE

corrected in the next Newton iteration. However, the slower convergence is a major concern, and careful tuning of parameters (both penalty stiffnesses and other solver parameters) can improve solver performance significantly. We observed that acceptable results can be obtained with only very crude parameter tuning. Indeed, most of our tunable parameters are simply powers of ten (See Table 2.1).

We run the test "torus_dr" with a range of coefficients of friction (0.1, 0.2, 0.5, 0.6, 1.0 and 10) and a range of stiffness (10, 10^2, 10^3 and 10^4). The method is able to accurately simulate dynamics at any coefficient of friction. The performance is not sensitive to the coefficient of friction. The simulations are stable and convergent in all cases, though convergence problems are observed for the highest stiffness ($k = 10^4$). Performance is strongly dependent on the stiffness, and a reasonable choice of stiffness is important for practical applications.

To understand the performance of our method compared to other penalty-based methods, we test our contact algorithm against that of [181]. Direct and fair comparison is complicated by the fact that [181] is a fully explicit method (Runge Kutta and a contact/friction postprocess). We have implemented the contact and friction forces of [181]

Name	dt	$\frac{\text{Time}}{\text{frame}}(s)$	CCD?	μ	k	grid	#p†	Tol
stack	0.01	0.12	Y	0.3	10^6		432	1
prmd_rd	0.01	0.10	Y	0.3	10^3		507	0.1
prmd_rm	0.01	4.28	Y	0.3	10^2	32^3	11k	1
prmd_rm slip	0.01	1.02	Y	0.05	10^2	32^3	11k	1
prmd_dd	0.002	0.53	Y	0.2	10^3		676	0.1
prmd_dd_lag	0.002	0.56	Y	0.2	10^3		676	0.1
prmd_dd_lagfr	0.002	0.58	Y	0.2	10^3		676	0.1
plane_r slide	0.001	0.20	N	0.1	10^3			1
plane_r stick	0.001	0.07	N	0.125	10^3			1
plane_d slide	0.001	0.05	N	0.1	10^2		8	1
plane_d stick	0.001	0.01	N	0.125	10^2		8	0.1
iplane_m slide	0.005	6.30	N	0.1	1	32^3	1.3k	1
iplane_m stick	0.005	1.34	N	0.125	1	32^3	1.3k	1
plane_m slide	0.005	1.44	N	0.1	1	32^3	1.3k	1
plane_m stick	0.005	2.16	N	0.125	1	32^3	1.3k	1
wdg_rm	0.01	0.30	N	0.25	10^2	32^3	1.5k	1
wdg_rm fall	0.01	0.18	N	0.1	10^2	32^3	1.5k	1
wdg_dr	0.002	0.62	N	0.05	10^3		169	0.1
wdg_dr fall	0.002	8.41	N	0.002	10^3		169	0.1
wdg_rr	0.01	0.05	N	0.08	10			1
wdg_rr fall	0.01	0.03	N	0.03	10			1
wdg_dr_r	0.01	1.26	N	0.03	10^3		169	0.1
plane_sph roll	0.005	0.05	Y	0.3	10			1
plane_sph slip	0.005	0.04	Y	0.01	10			1
torus_dr	0.002	2.17	Y	0.6	10^2		2.6k	0.01
cloth	0.005	6.73	Y	0.3	500		3.8k	1
sandbox_cube	0.001	242^{+}	Y	0.3	10^5	64^3	0.4m	1
sandbox_sph	0.001	7095^{+}	N	0.9	25000	128^3	0.9m	0.1
sand_on_sph	0.001	597	Y	0.9	10^4	96^3	64k	1
goo_on_sph	0.001	147^{+}	Y	0.3	10^4	96^3	120k	1
bowl	0.001	632	Y	0.3	10^5	96^3	31k	0.01

Table 2.1: Simulation Parameters and Timings for All of Our Simulations. † Number of particles used in a test. $^{+}$ The timing information is measured using 8 threads. ©2019 IEEE

using our implicit integration and CCD-based collision detection (which conveniently gives us robust normals automatically). We implemented two variants of their algorithm: (1) explicit contact and friction ("prmd_dd_lag") and (2) implicit normal contact followed by explicit friction ("prmd_dd_lagfr"). The discontinuous nature of the dynamic/static friction transition prevents us from implementing a fully-implicit version of their friction force. We

compare with our own with implicit contact and friction ("prmd_dd"). We use a pyramid stack of four deformable objects (like Figure 2.12 but with all four objects deformable). From this test, we observed that neither (1) nor (2) leads to stable stacks; both eventually collapse (see the supplementary video). This is not particularly surprising; indeed, our method will also not get stable stacks if friction is not treated implicitly. In particular, it is not sufficient for a penalty method to use attachments to achieve stable stacking.

We also examined the runtime of the two approaches, noting that we are running [181] under conditions for which it was never intended (implicit integration) and with a collision detection scheme that is likely less efficient than the non-CCD library that the original method used. Neither version was optimized. We observe that version (2) is slightly slower than our version. Since (2) uses implicit normal contact, CCD is performed in each Newton iteration as it is with our method. Since the actual force computations are negligible for both methods, we would expect the two to run at about the same pace, and indeed the performance is very close (see Table 2.1). There is a slight extra cost to (2), however, and it is caused by degradation in the performance of the Newton solver. In our simulation, our stack settles down, which allows the Newton solver to converge somewhat quicker. Lagging contact and/or friction means contact and elasticity are always fighting, which slows down the Newton solver. The comparison of (1) with our method is somewhat more surprising. In this case, contact and friction are both explicit, and CCD is performed only once per time step. This reduces the cost of CCD (from 32% for our method to 5% for (1)). This savings, however, comes at a cost. The fighting between elastic and contact forces is now much worse (before elastic forces only fought with friction). The resulting slower convergence

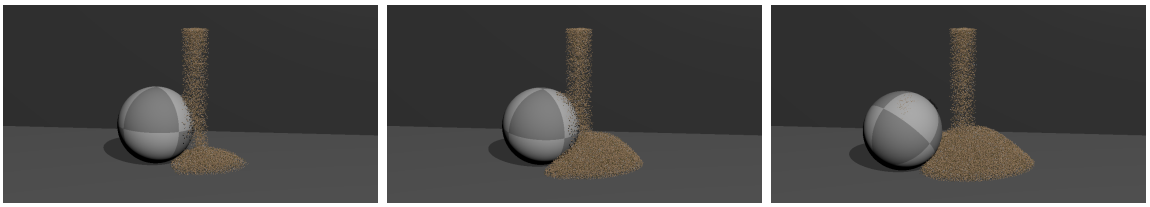


Figure 2.16: We pour sand over a sphere; friction between the sand and the sphere causes the sphere to be pulled into the sand column. ©2019 IEEE

in Newton's method is greater than the savings from CCD, and (1) ends up also running slower than our scheme. We note, however, that the performance differences are very slight, and which version is faster will depend strongly on the example setup, collision detection scheme chosen, level of code optimization, and a large number of other factors. Our contact algorithm is not intended as a way to make contact faster; rather, this comparison merely shows that it is competitive with existing methods and not too slow to be practical.

2.5 Conclusions and Future Work

We have demonstrated that we are able to simultaneously apply general elastic forces and frictional contact between different types of objects through the use of a novel frictional contact penalty force. We have demonstrated that this force allows us to achieve MPM-rigid and deformable-rigid coupling. Our force enables us to enforce frictional boundary conditions against the particles of an MPM simulation rather than being limited to processing these contacts on grid nodes, which prevents particle drift or bunching at collision objects.

Although the method is quite versatile, it has several important limitations. Because contact is enforced with a penalty force, it adds extra stiffness to the system and does

not completely prevent penetration between objects. As is normally the case for penalty forces, our penalty force does not model restitution. One promising avenue for extending our method to include a coefficient of restitution is through the use of a nonlinear damped spring [121]. Our implementation assumes isotropic friction; we see no reason that anisotropic friction could not be implemented by looking up local friction parameters (at the attachment location) and then replacing the circular cone with an elliptical cone in the yield criterion.

When colliding triangle meshes, we only process point-triangle pairs. In particular, we do not consider edge-edge pairs. We made this compromise for practical reasons. Using an edge in place of a particle introduces many complications. For example, would the edge be connected to an attachment point or an attachment edge? Is the attachment at the original barycentric collision location, or can the spring force slide along the edge? If it slides to the edge, does it become a point collision again?

The linear systems that result are asymmetrical, and we must solve them with GMRES. Although the memory requirements and computational load of GMRES scale quadratically with the number of iterations, our stabilization of the Newton solve safely allows us to limit the number of iterations (we use 20). If our Newton step was not computed accurately, we can rely on the line search.

Another limitation is that we can only hope to converge to a local minimum of our objective E. Unlike more standard optimization formulations, a local minimum for our objective does not lead to a solution to Newton's second law, though we have never observed it to converge to a non-global minimum. Our line searches are also more sensitive to the force derivatives than normal, since the derivatives used by the Wolfe line search

involve the force derivatives; only forces are involved in the standard formulation. The level set formulation is limited to smooth objects due to the presence of sonic points; the CCD formulation is smoother and does not have sonic points. The CCD formulation presented is not free from kinks and discontinuities, however, and we have observed Newton's method to fail to converge for the CCD formulation. This normally occurs when the system is nearly converged (so that the objective slope is low) and when the penalty stiffness is high (so that any kinks or discontinuities that may be present are amplified). Since the system is usually near convergence anyway, we simply continue the simulation with the incompletely converged result. We leave the problem of developing a more kink-resistant solver formulation for future work.

Our method is not as efficient as less strongly coupled formulations (such as ones that lag friction) due to the extra stiffness and asymmetric problem. Timing results and parameters for all of our simulations are given in Table 2.1.

Chapter 3

Affine particle in cell method for MAC grids and fluid simulation

In this chapter, we switch our topic to incompressible fluid simulation. A typical source of artifacts in fluid simulation is dissipation, which violates the conservation laws. The simulated fluid would appear excessive viscous and behave like glue or honey even we intend to simulate water. We will discuss this conservation topic under the context of hybrid simulation in this chapter.

3.1 Introduction

Hybrid particle/grid methods have been used for decades to simulate many different physical phenomena, including compressible flow, incompressible flow, plasma physics, computational solids, granular materials, and many more [72]. The original hybrid scheme was Particle In Cell (PIC) [79], which was originally devised for fluids. PIC worked by map-

ping particle state to a fixed Eulerian grid, on which forces are computed. The updated grid state is then mapped back to particles. In the original method, these mapping steps were done using linear interpolation and nearest-point interpolation. These low-order interpolation strategies were critical to the original method, since using smoother interpolation for both transfers produces excessive dissipation. However, smoother interpolation strategies are important for eliminating cell-crossing instabilities [16] and avoiding discontinuities in the flow map derivatives [157]. As a result, the original PIC method was not widely adopted.

A major improvement came with the introduction of Fluid Implicit Particle (FLIP) [26, 25], which mapped *changes* in velocities from the grid to the particles. This broke the cycle of repeated velocity interpolation, allowing smoother interpolation kernels to be used while avoiding excessive dissipation. This has led to a number of new interpolation kernels, including GIMP [16, 128], CPDI [149, 131]. B-spline interpolation has also been shown to work well [157].

This also greatly improved the angular momentum conservation properties of the particle/grid transfers [31, 27]. Indeed, FLIP transfers can be used to implement schemes that conserve momentum, angular momentum, and total energy [116, 115]. Another major advance in hybrid methods came with the introduction of the Material Point Method (MPM), which extended hybrid methods to handle viscoelastic solids [161, 162].

Although most hybrid methods today are based on FLIP transfers, such schemes are known to suffer from noise caused by numerical instabilities. While all hybrid particle-grid approaches suffer to some degree from the finite grid instability [104, 135] (or the ringing instability [24, 75]), these errors are quite prominent when FLIP transfers are used. This

is particularly true when using MPM [161, 162] for simulating history dependent materials. The finite grid instability may be understood as a mismatch between the modes that can be represented on particles and the modes that can be represented on the mesh.

Some explanation and intuition for the causes of these instabilities, as well as ideas for reducing them, may also be gleaned by examining the way transfers interact with grid forces. In a PIC-like scheme, particle velocities are transferred (interpolated) to the grid, then transferred (interpolated) back to particles at the end of the time step. The effect of this repeated interpolation is significant dissipation. If noise is added to the particle velocities, the noise is filtered out during the interpolations and also to some degree by the physics and grid-based numerical scheme. This makes PIC-like schemes very stable. FLIP transfers back velocity differences instead, which avoids the dissipation. If no changes are made to the grid velocities, then the particle velocities are unmodified. However, this also makes the scheme respond differently to particle noise. If the particle-to-grid transfer operator has a nullspace, then any noise in the nullspace would not be transferred to the grid. Since this noise component will not be damped on the grid, no corresponding correction will be transferred back to the particles. The noise component is not damped. Typically there are many more particles than grid nodes, which means the transfer operator must have a large nullspace in which particle noise may persist without damping. Failing to efficiently damp errors introduced during integration leads to numerical instability.

When using FLIP transfers, the particle velocities are not used to move the particle positions; rather, particle positions are directly interpolated from the grid. This is equivalent to using an interpolated, PIC velocity for position updates. This greatly limits the negative

	XPIC(r)	XPIC(1)
1	$m_i^n = \sum_p w_{ip}^n m_p$	$m_i^n = \sum_p w_{ip}^n m_p$
2	$m_i^n \mathbf{v}_i^n = \sum_p w_{ip}^n m_p \mathbf{v}_p^n$	$m_i^n \mathbf{v}_i^n = \sum_p w_{ip}^n m_p \mathbf{v}_p^n$
3	Grid evolution: $\mathbf{v}_i^n \rightarrow \tilde{\mathbf{v}}_i^{n+1}$	Grid evolution: $\mathbf{v}_i^n \rightarrow \tilde{\mathbf{v}}_i^{n+1}$
4a	$\mathbf{v}_i^{n,1} = r\mathbf{v}_i^n$	
4b	$\mathbf{v}_i^{n,k} = \frac{r-k+1}{k} \sum_p \sum_j \frac{m_p w_{ip}^n w_{jp}^n}{m_i^n} \mathbf{v}_j^{n,k-1}$	
4c	$\mathbf{v}_i^{n*} = \sum_{k=1}^{r} (-1)^{k+1} \mathbf{v}_i^{n,k}$	
4d	$\mathbf{v}_p^{n+1} = \sum_i w_{ip}^n (\mathbf{v}_i^{n*} + \tilde{\mathbf{v}}_i^{n+1} - \mathbf{v}_i^n)$	$\mathbf{v}_p^{n+1} = \sum_i w_{ip}^n \tilde{\mathbf{v}}_i^{n+1}$
5	$\tilde{\mathbf{x}}_i^{n+1} = \mathbf{x}_i^n + (\mathbf{v}_i^n + \tilde{\mathbf{v}}_i^{n+1}) \frac{\Delta t}{2}$	$\tilde{\mathbf{x}}_i^{n+1} = \mathbf{x}_i^n + (\mathbf{v}_i^n + \tilde{\mathbf{v}}_i^{n+1}) \frac{\Delta t}{2}$
6	$\mathbf{x}_p^{n+1} = \hat{\mathbf{x}}_p^{n+1} + \sum_i w_{ip}^n (\mathbf{v}_i^{n*} - \mathbf{v}_p^n) \frac{\Delta t}{2}$	$\mathbf{x}_p^{n+1} = \hat{\mathbf{x}}_p^{n+1} + \sum_i w_{ip}^n (\mathbf{v}_i^n - \mathbf{v}_p^n) \frac{\Delta t}{2}$
	$\hat{\mathbf{x}}_p^{n+1} = \sum_i w_{ip}^n \tilde{\mathbf{x}}_i^{n+1}$	$\hat{\mathbf{x}}_p^{n+1} = \sum_i w_{ip}^n \tilde{\mathbf{x}}_i^{n+1}$

Figure 3.1: Time integration scheme for XPIC of order r. XPIC(1) is equivalent to PIC except for the more accurate grid and particle position update. Step 4b is repeated for $2 \leq k \leq r$.

impact of the spurious particle velocities. This also means it is possible for a simulation to come to rest with nonzero velocities (See Figure 3.19). As long as the final velocity field is in the transfer operator's nullspace, the grid velocity will be zero and the particles will not move. Despite these issues, FLIP transfers are still most commonly used, particularly for MPM. Blends between PIC and FLIP transfers are also a viable alternative [186, 28, 160], using a small amount of the PIC solution to dissipate noise that might otherwise accumulate in the FLIP solution.

Recently, a new transfer called Affine Particle In Cell (APIC) was developed as a PIC-like alternative to FLIP [93, 94]. These transfers interpolate information from particles to grid and also from grid to particles, as in the original PIC (See Figure 3.2). To reduce

dissipation, the rowspace of the transfer operator is enriched by storing velocities and a measure of velocity gradients on particles. (It is worth pointing out that [169] did something similar in the context of FLIP transfers, but the benefit in this case was noticeable but relatively modest. FLIP is already not very dissipative, and the modifications to the transfers do not reduce the finite grid instability.) In doing so, fewer velocity modes are filtered out by the transfers, dramatically reducing dissipation. In particular, it is possible to develop APIC schemes that conserve both linear and angular momentum in the case of co-located grids [94].

Since the development of APIC, another scheme called XPIC [78] was developed as another compromise between the dissipation of PIC and the noise of FLIP. XPIC is a family of schemes that is in many ways a blend of PIC and FLIP (and includes PIC as a member). XPIC uses PIC transfers to filter out noise from the FLIP solution, unlike a simple blend which merely damps it. XPIC significantly reduces FLIP-style noise, but unlike APIC does not eliminate it. In the special case of XPIC(1), the transfer of velocity from grid to particle is equivalent to PIC method, but XPIC(1) differs in the position update (See Figure 3.1). XPIC(1) may be considered as an improved PIC. Other regularization strategies have also been employed to mitigate the noise caused by FLIP [53].

Another transfer strategy is moving least squares (MLS) [53], which computes a polynomial best fit to transfer velocity information to particles. MLS is capable of high order accuracy but is rather expensive due to the need to solve a system of equations per particle for the transfers. Indeed, some variants of APIC may be formulated as a PIC-style MLS with polynomial degree one [57, 88].

Although the focus on APIC transfers has mostly been in the context of MPM, [93] also explored a limited extension to MAC grids. *In this work, we show that an APIC scheme can be developed for MAC grids that also conserves both linear and angular momentum.*

Being a new method, there are many questions about the behavior and utility of APIC that have not been explored. In the context of fluids, one important concern is the suitability of PIC for high Reynolds number flows. *Shedding light on this question in the primary goal of this work.*

We show how a two dimensional Fourier transform can be used to study the dissipation of transfers for MAC-grid-based incompressible fluids, in a similar way to how one dimensional Fourier transforms are currently being used for one dimensional MPM. We also show that two dimensional Fourier transform can be used to study the interaction between transfers and pressure projection. This makes it possible to compare APIC, PIC, FLIP and XPIC in terms of dissipation of two dimensional incompressible flows.

3.2 Numerical method

3.2.1 Notation

In this document, we use notation to give hints as to the meaning of symbols. As a general rule, bold lowercase symbols ($\mathbf{x}_p^n$, $\mathbf{p}^{P,n}$, $\mathbf{e}_a$) are vectors, bold uppercase symbols ($\mathbf{D}_{pa}^n$, $\mathbf{I}$) are matrices, and non-bold symbols (w_{ipa}^n, m_p, Δt, $\tilde{v}_{ia}^{n+1}$) are scalars. We follow the convention that all vector quantities are considered to be column vectors unless explicitly transposed. Thus, quantities like ∇p will be treated as column vectors.

Many symbols use a combination of subscripts and superscripts. Subscripts are used to index grid nodes (i, j), particles (p), and spatial dimensions (a, b). We index MAC faces by treating each axis direction as a regular grid, which is indexed with ia. The axis direction is denoted as $\mathbf{e}_a$. Quantities associated with both grid and particle indices have both indices (w^n_{ipa}). A superscript of n indicates a quantity near the beginning of the time step (before forces are applied), and a superscript of $n+1$ indicates a quantity computed later in the time step. Other adornments are used to distinguish quantities that would otherwise get the same name ($\tilde{\mathbf{x}}^{n+1}_{ia}$ vs $\mathbf{x}^{n+1}_{ia}$) or to denote intermediates (v^*_{ia}). The superscripts P and G indicate global particle-based or grid-based quantities (^{P,n}, ^{G,n}). To avoid confusion, we will never use the summation convention in this document; all summation is specified explicitly.

Note that the indices can be used to unambiguously distinguish quantities on MAC grids (w^n_{ipa}) from those on co-located grids (w^n_{ip}). Such quantities have the same meaning and differ only in the grid layout chosen.

3.2.2 Weights

Hybrid schemes are notable for requiring information to be transferred between particles and a grid. These transfers are defined using an interpolating kernel, which is assumed to satisfy the partition of unity and interpolation properties

$$\sum_i N(\mathbf{x}^n_p - \mathbf{x}^n_i) = 1 \qquad\qquad \sum_i \mathbf{x}^n_i N(\mathbf{x} - \mathbf{x}^n_i) = \mathbf{x} \tag{3.1}$$

for any $\mathbf{x}$. The kernel $N(\mathbf{x})$ is used to define interpolation weights and weight gradients as $w_{ip}^n = N(\mathbf{x}_p^n - \mathbf{x}_i^n)$ and $\nabla w_{ip}^n = \nabla N(\mathbf{x}_p^n - \mathbf{x}_i^n)$. The properties of $N(\mathbf{x})$ lead to properties for w_{ip}^n and ∇w_{ip}^n:

$$\sum_i w_{ip}^n = 1 \qquad \sum_i w_{ip}^n \mathbf{x}_i^n = \mathbf{x}_p^n \qquad \sum_i w_{ip}^n (\mathbf{x}_i^n - \mathbf{x}_p^n) = \mathbf{0} \qquad \sum_i \mathbf{x}_i^n (\nabla w_{ip}^n)^T = \mathbf{I}. \tag{3.2}$$

In the case of MAC grids and the proposed time integration, weights are defined independently for each axis with $\bar{w}_{ipa}^n = N(\mathbf{x}_p^n - \mathbf{x}_{ia}^n)$ and $w_{ipa}^n = N(\mathbf{x}_p^{n+1} - \mathbf{x}_{ia}^n)$. In our discretization we advance positions *before* transferring particle information to the grid, and the algorithm needs two sets of weights when using FLIP or XPIC. We use $\bar{w}_{ipa}^n$ to denote weights before moving particles and w_{ipa}^n to denote weights after. It is not actually necessary to compute two sets of weights, since $w_{ipa}^n = \bar{w}_{ipa}^{n+1}$.

The x, y, and z faces form regular Cartesian grids that are offset from one another. The same properties hold independently per axis:

$$\sum_i w_{ipa}^n = 1 \quad \sum_i w_{ipa}^n \mathbf{x}_{ia}^n = \mathbf{x}_p^{n+1} \quad \sum_i w_{ipa}^n (\mathbf{x}_{ia}^n - \mathbf{x}_p^{n+1}) = \mathbf{0} \quad \sum_i \mathbf{x}_{ia}^n (\nabla w_{ipa}^n)^T = \mathbf{I}. \tag{3.3}$$

For completeness, the linear, quadratic, and cubic splines we investigate for the kernel $\hat{N}(x)$ are:

linear	quadratic	cubic
$\begin{cases} 1-\lvert x\rvert & 0 \le \lvert x\rvert < 1 \\ 0 & 1 \le \lvert x\rvert \end{cases}$	$\begin{cases} \frac{3}{4} - x^2 & 0 \le \lvert x\rvert < \frac{1}{2} \\ \frac{1}{2}(\frac{3}{2} - \lvert x\rvert)^2 & \frac{1}{2} \le \lvert x\rvert < \frac{3}{2} \\ 0 & \frac{3}{2} \le \lvert x\rvert \end{cases}$	$\begin{cases} \frac{2}{3} - \frac{1}{2}x^2(2-\lvert x\rvert) & 0 \le \lvert x\rvert < 1 \\ \frac{1}{6}(2-\lvert x\rvert)^3 & 1 \le \lvert x\rvert < 2 \\ 0 & 2 \le \lvert x\rvert \end{cases}$

From this, $N(\mathbf{x}) = \hat{N}(x)\hat{N}(y)$ in 2D and $N(\mathbf{x}) = \hat{N}(x)\hat{N}(y)\hat{N}(z)$ in 3D.

3.2.3 Overview of co-located transfers

General outlines for a sample algorithm for PIC, FLIP, and APIC with a co-located grid are provided in Figure 3.2. A corresponding outline for XPIC is provided in Figure 3.1. In each case, the time step begins by transferring particle mass m_p and momentum $m_p\mathbf{v}_p^n$ to the grid to produce mass m_i^n and momentum $m_i^n\mathbf{v}_i^n$ on the grid (steps 1-2). Velocity is computed by dividing the mass from the momentum. This velocity is updated on the grid in some way, such as by applying MPM finite element forces (step 3). This results in an updated grid velocity $\tilde{\mathbf{v}}_i^{n+1}$, which is then transferred back to particles, resulting in the new particle velocity $\mathbf{v}_p^{n+1}$ (step 4). Finally, particles are updated to new locations $\mathbf{x}_p^{n+1}$ by interpolating them from moving grid positions $\tilde{\mathbf{x}}_i^{n+1}$ (steps 5-6). Additional steps may be required depending on the specifics of the grid evolution algorithm, such as maintaining a deformation gradient ($\mathbf{F}_p^n \to \mathbf{F}_p^{n+1}$).

This simple outline is flexible, and many variations have been considered. For example, steps 4-6 may be moved to the beginning of the time step; in this case, the transfers may be interpreted as a type of semi-Lagrangian advection for an Eulerian algorithm [23].

	PIC	FLIP	APIC
1	$m_i^n = \sum_p w_{ip}^n m_p$	$m_i^n = \sum_p w_{ip}^n m_p$	$m_i^n = \sum_p w_{ip}^n m_p$
2a			$\mathbf{D}_p^n = \sum_i w_{ip}^n \mathbf{r}_{ip}^n (\mathbf{r}_{ip}^n)^T$
			$\mathbf{r}_{ip}^n = \mathbf{x}_i^n - \mathbf{x}_p^n$
2b			$\mathbf{C}_p^n = \mathbf{B}_p^n (\mathbf{D}_p^n)^{-1}$
2c	$m_i^n \mathbf{v}_i^n = \sum_p w_{ip}^n m_p \mathbf{v}_p^n$	$m_i^n \mathbf{v}_i^n = \sum_p w_{ip}^n m_p \mathbf{v}_p^n$	$m_i^n \mathbf{v}_i^n = \sum_p w_{ip}^n m_p \mathbf{u}_{ip}^n$
			$\mathbf{u}_{ip}^n = \mathbf{v}_p^n + \mathbf{C}_p^n \mathbf{r}_{ip}^n$
3	Grid evolution: $\mathbf{v}_i^n \to \tilde{\mathbf{v}}_i^{n+1}$	Grid evolution: $\mathbf{v}_i^n \to \tilde{\mathbf{v}}_i^{n+1}$	Grid evolution: $\mathbf{v}_i^n \to \tilde{\mathbf{v}}_i^{n+1}$
4a	$\mathbf{v}_p^{n+1} = \sum_i w_{ip}^n \tilde{\mathbf{v}}_i^{n+1}$	$\mathbf{v}_p^{n+1} = \mathbf{v}_p^n + \sum_i w_{ip}^n \hat{\mathbf{v}}_i^{n+1}$	$\mathbf{v}_p^{n+1} = \sum_i w_{ip}^n \tilde{\mathbf{v}}_i^{n+1}$
		$\hat{\mathbf{v}}_i^{n+1} = \tilde{\mathbf{v}}_i^{n+1} - \mathbf{v}_i^n$	
4b			$\mathbf{B}_p^{n+1} = \sum_i w_{ip}^n \tilde{\mathbf{v}}_i^{n+1} (\mathbf{r}_{ip}^n)^T$
5	$\tilde{\mathbf{x}}_i^{n+1} = \mathbf{x}_i^n + \Delta t \tilde{\mathbf{v}}_i^{n+1}$	$\tilde{\mathbf{x}}_i^{n+1} = \mathbf{x}_i^n + \Delta t \tilde{\mathbf{v}}_i^{n+1}$	$\tilde{\mathbf{x}}_i^{n+1} = \mathbf{x}_i^n + \Delta t \tilde{\mathbf{v}}_i^{n+1}$
6	$\mathbf{x}_p^{n+1} = \sum_i w_{ip}^n \tilde{\mathbf{x}}_i^{n+1}$	$\mathbf{x}_p^{n+1} = \sum_i w_{ip}^n \tilde{\mathbf{x}}_i^{n+1}$	$\mathbf{x}_p^{n+1} = \sum_i w_{ip}^n \tilde{\mathbf{x}}_i^{n+1}$

Figure 3.2: Representative time integration schemes for PIC, FLIP, and APIC. Details of the grid evolution are identical for each. Note that the transfers are very similar, though APIC includes a few extra steps related to the additional particle state. Subscripts indicate where quantities like ($\mathbf{v}_i$ is velocity on the grid; $\mathbf{v}_p$ is velocity on particles).

PIC and FLIP differ only in step 4a, and a PIC/FLIP blend may be constructed by interpolating these two velocity updates [186]. A more general form for steps 4b and 6 in APIC is considered in [94], which allows them to do midpoint rule for their time integration and achieve conservation of angular momentum. Viewed as an advection scheme, APIC may be compared to [180], which also stores derivative information to reduce diffusion.

3.2.4 APIC for MAC grids

We begin our treatment of the MAC case by laying out the full time integration scheme. Since we will be using our MAC grid for fluids, we discretized the Navier-Stokes equations. We implement a projection method, as is typically done with FLIP [186, 23].

Advection

In a standard standard MPM discretization, particle positions are updated at the end of the time step. If this is done with Chorin splitting, a pressure projection will be used to make the fluid velocity divergence-free, after which this velocity will be transferred to particles and moved. The resulting particle velocities are not divergence free, and we have observed convergence problems in the resulting scheme.

Instead, we note that [186] moves particles at the beginning of the time step. This is in line with Chorin splitting, which projects the advected velocity field, resulting in a divergence-free velocity field at the end of the time step. To implement this, we delay the position update until the beginning of the next time step but otherwise compute the update in exactly the same way.

$$\mathbf{x}_p^{n+1} = \sum_{ia} \bar{w}_{ipa}^n \mathbf{e}_a \mathbf{e}_a^T \tilde{\mathbf{x}}_{ia}^n = \mathbf{x}_p^n + \Delta t \mathbf{v}_p^n. \tag{3.4}$$

Here we use $\bar{w}_{ipa}^n = N(\mathbf{x}_p^n - \mathbf{x}_{ia}^n)$ to denote the weights *before* moving particles, reserving $w_{ipa}^n = N(\mathbf{x}_p^{n+1} - \mathbf{x}_{ia}^n)$ for the weights *after* moving particles. $\mathbf{e}_a$ are axis directions. Note that $\tilde{\mathbf{x}}_{ia}^n$ is the quantity $\tilde{\mathbf{x}}_{ia}^{n+1}$ from the previous time step. In the case of APIC (or PIC), the second equality holds, and the particles can be moved using particle velocities without referencing $\bar{w}_{ipa}^n$. Although the XPIC position update is more accurate than the PIC update, its use would prevent our APIC transfers from satisfying the APIC properties. Generalized APIC transfers [94] could likely be adapted to MAC grids to achieve similar benefits. Since these are more complex than the original transfers, we base our transfers on the original APIC transfers and use PIC position updates.

In the case of FLIP and XPIC, we store $\tilde{v}_{ia}^{n+1}$ from the previous time step, which allows us to compute new positions using the summation. Since this information in not available for the first frame, we perform a particle-to-grid transfer to obtain initial velocities for $\tilde{v}_{ia}^{n+1}$. The PIC/APIC update could be used for the first time step instead.

Particle to grid

The next step is transferring particle mass m_p, velocity $\mathbf{v}_p^n$, and information about velocity derivatives $\mathbf{b}_{pa}^n$ from particles to the grid.

$$m_{ia}^n = \sum_p w_{ipa}^n m_p \tag{3.5}$$

$$\mathbf{D}_{pa}^n = \sum_i w_{ipa}^n (\mathbf{x}_{ia}^n - \mathbf{x}_p^{n+1})(\mathbf{x}_{ia}^n - \mathbf{x}_p^{n+1})^T \tag{3.6}$$

$$m_{ia}^n v_{ia}^n = \sum_p w_{ipa}^n m_p \mathbf{e}_a^T \mathbf{v}_p^n + \sum_p w_{ipa}^n m_p (\mathbf{b}_{pa}^n)^T (\mathbf{D}_{pa}^n)^{-1} (\mathbf{x}_{ia}^n - \mathbf{x}_p^{n+1}) \tag{3.7}$$

This looks very similar to the co-located case, but there are some subtle differences. There is one matrix $\mathbf{D}_{pa}^n$ defined per axis, and $\mathbf{b}_{pa}^n$ is a vector per axis. These vectors may be interpreted as the columns of the matrix $\mathbf{B}_p^n$ that is used in the co-located case. Being a MAC layout, velocities are staggered, and the scalar components of velocity v_{ia}^n are stored at separate locations. Recall that w_{ipa}^n is computed using the new particle positions, which explains the unexpected use of $\mathbf{x}_p^{n+1}$ in these equations.

Grid evolution

We split the grid evolution step into two parts: gravity and pressure. Gravity is applied explicitly: $v_{ia}^* = v_{ia}^n + \Delta t g_a$. We use finite differences to discretize the Poisson equation and perform a velocity projection on a MAC grid layout to obtain an incompressible velocity field $v_{ia}^* \rightarrow \tilde{\mathbf{v}}_{ia}^{n+1}$. We assume a constant density ρ for the pressure discretization. These steps are performed in exactly the same way as for an Eulerian MAC discretization. We assume inviscid Euler, so we do not apply viscosity. Although we have masses on the grid, we do not use them for the pressure projection, since doing so causes *boiling* in the fluid. That is, an initially stationary pool of water would develop currents in it.

3.2.5 Grid to particle

After updating grid velocities, we update our final particle data following essentially the same algorithm as in the co-located case.

$$\mathbf{v}_p^{n+1} = \sum_{ia} w_{ipa}^n \tilde{v}_{ia}^{n+1} \mathbf{e}_a \tag{3.8}$$

$$\mathbf{b}_{pa}^{n+1} = \sum_{i} w_{ipa}^n \tilde{v}_{ia}^{n+1} (\mathbf{x}_{ia}^n - \mathbf{x}_p^{n+1}) \tag{3.9}$$

$$\tilde{\mathbf{x}}_{ia}^{n+1} = \mathbf{x}_{ia}^n + \Delta t \tilde{v}_{ia}^{n+1} \mathbf{e}_a \tag{3.10}$$

Unlike the co-located algorithm, we do not advance positions; we delay this step until the beginning of the next time step. In Chapter A we show that these transfers satisfy the original APIC properties.

3.3 Fourier analysis of transfers

We use Fourier analysis to characterize the dissipation of APIC transfers compared to PIC, FLIP, and XPIC transfers. There are two main settings in which this can be explored. The first is by considering a round trip from grid to particle and then back to grid. In between these transfers, particles are moved, and this step must be ignored to retain linearity. This approach is relatively simple, but it is unable to analyze methods like FLIP, which retain information on particles between time steps. The second approach is to consider transfers from particles to grid and then back to particles. In this case, other grid steps (pressure projection in our case) lie between these two transfers. Since these steps are in general linear, they can be included in the analysis. This analysis approach is compatible with FLIP and XPIC, and we use it to draw a strong contrast between FLIP and APIC.

To facilitate Fourier analysis, we must add two assumptions: (a) the domain is periodic and (b) all cells have the same particle distribution. Note that (b) does *not* imply that the particle distribution is regular. Particles may be positioned quite irregularly within a cell, but that irregular layout must be the same for all cells. Because of (a), it is convenient to treat grid indices as periodic.

3.3.1 Grid to particle to grid

Linear transfer matrix While the transfers themselves are linear (as functions of velocities), the advection step is nonlinear due to the movement of the particles and corresponding changes in interpolation weights. For the purposes of analysis, we can eliminate the nonlinearity by considering the limit $\Delta t \to 0$. This corresponds to transferring from grid to

particles and immediately back to the grid and approximates the dissipation that results when small time steps are taken. We can then express the grid-to-particle-to-grid transfer as a matrix $v_{ia}^{n+1} = \sum_{jb} M_{ia,jb} \tilde{v}_{jb}^{n+1}$.

Axis-independence Inspecting the definition of the transfers carefully, we see that the axis components decouple. We can instead write $v_{ia}^{n+1} = \sum_j M_{ij}^a \tilde{v}_{ja}^{n+1}$, where $M_{ia,jb} = M_{ij}^a \delta_{ab}$. That is, each face axis has its own separate transfer matrix, but faces in different direction do not affect each other. This is not surprising; an object translating in the x direction should not begin moving in the y direction. Since each dimension is independent, we can focus on one axis arbitrarily and drop the axis indices. The grid layout is now just a regular grid. As such, the analysis in this section applies equally to transfers with a MAC layout and transfers with a collocated layout. We can now write $v_i^{n+1} = \sum_j M_{ij} \tilde{v}_j^{n+1}$, where M_{ij} is a matrix with as many rows and columns as grid nodes.

Analyzing dissipation with eigenvalues When we analyze the dissipation of the transfers, it is helpful to isolate the dissipation caused by transfers from the dissipation caused by other parts of the evolution, such as the pressure projection. For this reason, we consider the consequences of repeating these transfers. The eigenvalues λ of M_{ij} tell us how dissipative the transfers are. Eigenvectors corresponding to $\lambda = 1$ are preserved across the transfer without dissipation. Velocity eigenvectors with $0 < \lambda < 1$ decay due to dissipation. Eigenvectors with $\lambda = 0$ are eliminated entirely. $|\lambda| > 1$ would indicate instability. Within this periodic setup, we observe that M_{ij} is symmetric (each cell affects its left neighbor by the same amount as its right neighbor), which leads to real eigenvalues. In practice, we also

observe that $0 \leq \lambda \leq 1$ for all of the schemes we consider. $\lambda \approx 1$ is ideal. A velocity mode corresponding an eigenvector with eigenvalue λ decays by factor λ^k, where the exponent k is the number of grid-to-particle-to-grid transfers.

Fourier analysis

Circulant With the periodicity assumptions, the transfer matrix M_{ij} will be *tensor product* circulant. That is, if $i = (r, s)$ and $j = (u, v)$ are grid indices in 2D, then $M_{(r,s),(u,v)} = M_{(r+k,s+m),(u+k,v+m)}$ for any k and m, where we make use of the convention of treating the indices as periodic. This just says that the transfer operator appears the same for all cells, which is expected since all cells are indistinguishable. From this, we conclude that M_{ij} has the special structure $M_{(r,s),(u,v)} = c_{r-u,s-v}$, where c_i is the 0^{th} column of M_{ij}. This is the multidimensional analog of a circulant matrix.

Eigenvalues from Fourier transform A tensor product circulant matrix is diagonalized by a multidimensional Fourier transform in exactly the same way that a circulant matrix is diagonalized by a Fourier transform. The eigenvalues of M_{ij} can be computed as the multidimensional Fourier transform of its column c_i. The Fourier transform conveniently identifies one eigenvalue λ_i with each grid node i, which provides a convenient visual representation of the eigenvalues in 2D. This is the basis for Figure 3.3. Fourier analysis has been used to analyze hybrid schemes before in 1D (e.g., in [78]), but as far as we are aware we are the first to adopt it as a tool in higher dimensions. This generalization is important, since incompressible flow is trivial in 1D and vortices do not exist in 1D.

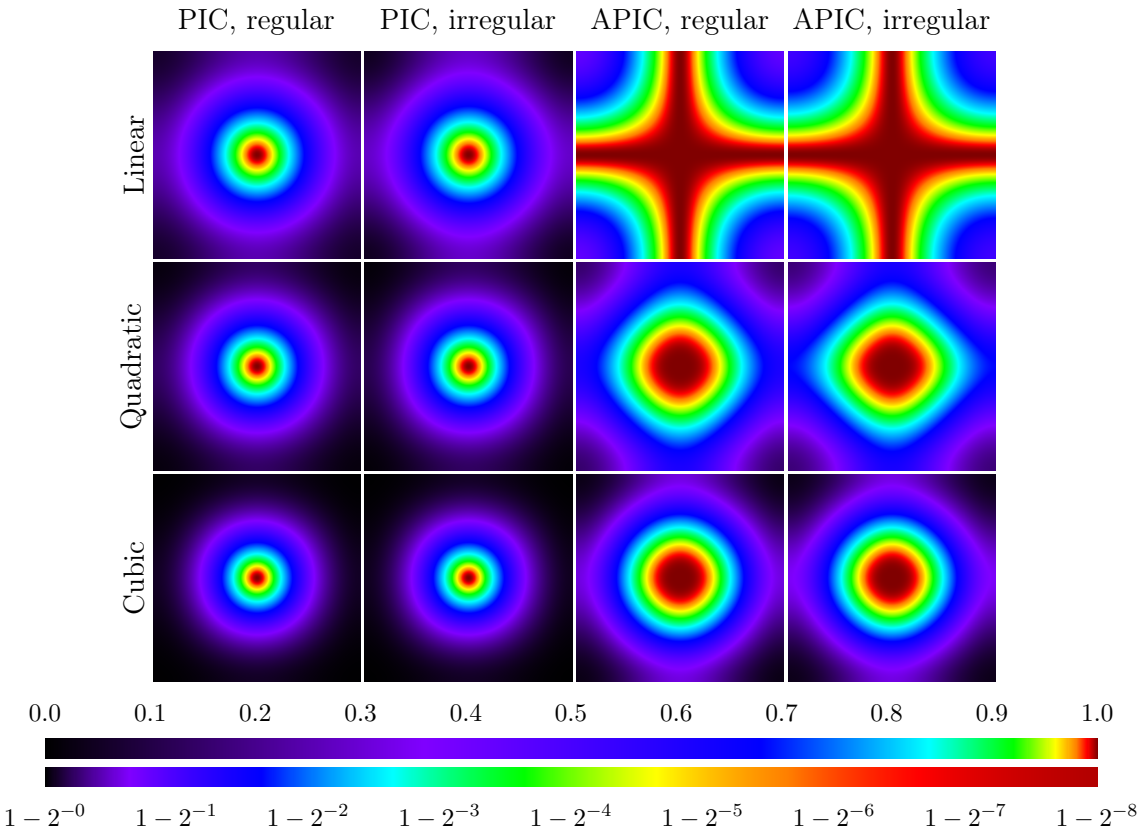


Figure 3.3: Eigenvalue images for PIC and APIC using 4 particles per cell and linear, quadratic, or cubic splines.

Computing c_i While it is possible to work out c_i and M_{ij} analytically (and we have done so in a few cases), the results are not enlightening. Instead, we compute c_i numerically by performing the transfers on a velocity field containing a single nonzero entry. The multidimensional Fourier transform of the result gives us the eigenvalues.

Eigenvalue images

Sparsity of c_i The next useful observation is that c_i is very sparse; the nonzero entries in c_i are at most a few cells away from the nonzero entry in the velocity field. That is, $c_i = c_{rs} = 0$ for $|r| > w$ or $|s| > w$, where w here is the width of the stencil. Note that

we are using $c_{rs} = c_{(r,s)}$ as a convenient shorthand. $w \leq 3$ for all of our splines (linear, quadratic, cubic) and transfers (APIC, FLIP, PIC, XPIC).

Resolution-independence of c_i Next, we observe that the nonzero entries of c_i do not depend on the resolution provided the grid size $m \times n$ is large enough ($m, n \geq 2w+1$). The transfers are local, so the nonzero entries of c_i cannot depend on the *number* of grid cells. Since Δx has units of length but the entries of c_i are dimensionless, the entries of c_i must be independent of Δx. This resolution-independence means that high-resolution images are cheap to compute, since they only require a moderate number of very low resolution transfers to deduce c_i followed by a full-resolution multidimensional Fourier transform.

Explicit form of eigenvalues The eigenvalues $\lambda_i = \lambda_{rs}$ are given by the Fourier transform

$$\lambda_{rs} = \sum_{u=0}^{m} \sum_{v=0}^{n} c_{uv} e^{\frac{2\pi i r u}{m}} e^{\frac{2\pi i s v}{n}} = \sum_{u=-w}^{w} \sum_{v=-w}^{w} c_{uv} e^{\frac{2\pi i r u}{m}} e^{\frac{2\pi i s v}{n}} \tag{3.11}$$

This gives us the eigenvalues for *any* size grid. If we index $\lambda(x, y)$ instead with rational numbers in the range $-\frac{1}{2} \leq x, y < \frac{1}{2}$, where $x = \frac{r}{m}$ and $y = \frac{s}{n}$.

$$\lambda(x, y) = \sum_{u=-w}^{w} \sum_{v=-w}^{w} c_{uv} e^{2\pi i x u} e^{2\pi i y v} \tag{3.12}$$

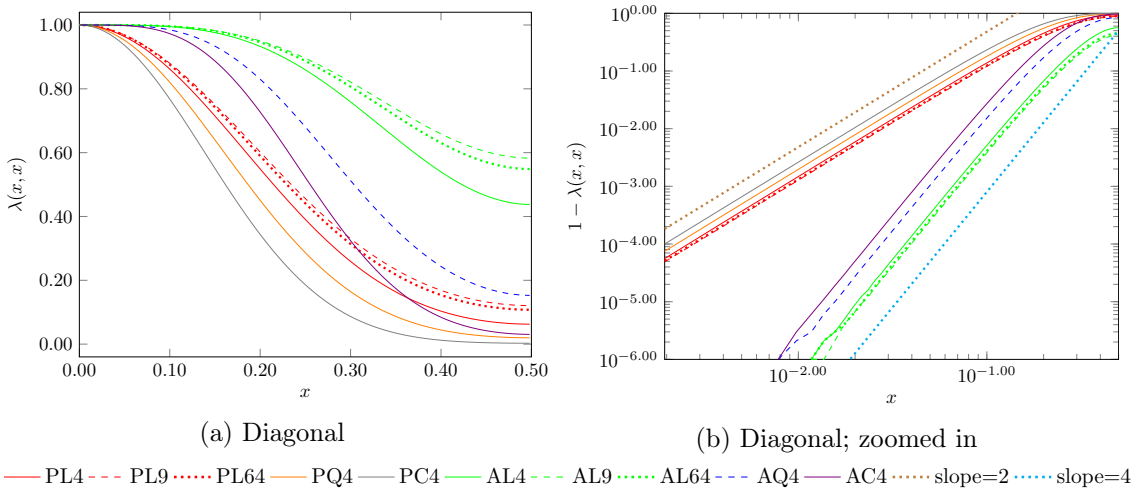
Observe that this does not depend on the resolution $m \times n$ of the grid. Indeed, we can treat this as a continuous function for eigenvalues. The eigenvalues for any finite resolution are obtained by sampling the appropriate location within the continuous map. The map is sym-

metric, with $\lambda(x, y) = \lambda(-x, y) = \lambda(x, -y) = \lambda(-x, -y)$. The constant mode corresponds to $\lambda(0, 0)$ and lies in the middle of the image. This is how the images shown in Figures 3.3 are constructed.

Numerical study - regular seeding

In the first study, we compute eigenvalue images corresponding to a range of parameters. We test PIC and APIC transfers using linear, quadratic, and cubic splines (Figure 3.3). In each ease, we seed with 4 particles per cell using either regular seeding (particles seeding in a regular grid pattern) or irregular seeding (4 particles in a blue noise pattern, with all cells having the same arrangement of particles). In each case, low-frequency modes lie in the center of the image. Dark red modes are near 1, and black modes are near zero, with colors assigned on a logarithmic scale according to the color bars shown with the images. (All of the eigenvalue images are shown with the same color scale, which is repeated for convenience.) Images with larger red regions near the middle are less dissipative. From the images, we see that APIC is dramatically less dissipative than PIC. Comparing across splines, we see that dissipation increases with spline order. This is not particularly surprising, since higher order splines interpolate over a larger range. The difference between cubic and quadratic splines is quite modest, and we do not see a strong motivation to prefer one over the other on grounds of dissipation.

To make the differences between the methods easier to see, 1D cross sections from these eigenvalue images are shown in Figures 3.4. The location of the cross sections are illustrated in Figure 3.6a. For reference, the results are shown for linear splines with 4, 9, and 64 particles per cell to illustrate the sampling dependence for linear splines. Quadratic



(a) Diagonal

(b) Diagonal; zoomed in

Figure 3.4: Eigenvalues for PIC (P) and APIC (A), with 4, 9, or 64 particles per cell and linear (L), quadratic (Q), and cubic (C) interpolation splines. Only the 4 particles per cell case is shown for quadratic and cubic splines; the others are indistinguishable. The most important part of the plot is the top left near $(0, 1)$, which corresponds to scale factor for larger scale vortices. For reference, $\lambda(0.10, 0.10)$ is the decay factor for a vortex 5 grid cells in diameter. $\lambda(0.05, 0.05)$ is the decay factor for a vortex 10 grid cells in diameter. Values closer to one are dissipated less. In the right plot, the region near $(x, \lambda) = (0, 1)$ is magnified with and $(x, 1 - \lambda)$ plotted using logarithmic scales. In this plot, the order of falloff in dissipation as a function of vortex size manifests as the slope of the curve. For reference, lines with slopes 2 and 4 are included in the plot. Note that the orders are 2 for PIC and 4 for APIC, consistent with the analysis in Section 3.3.1.

and cubic splines are not sensitive to sampling density. The curves for these splines at higher sampling density have been omitted since they overlap the corresponding curve at 4 particles per cell. Note that $\lambda(x, 0) = \lambda(0, x)$ due to symmetry of the particle distribution. The difference in damping is especially visible in the zoomed-in versions of these plots. Observe that APIC remains very close to 1 for much larger x than PIC due to the extra zero derivatives at the origin.

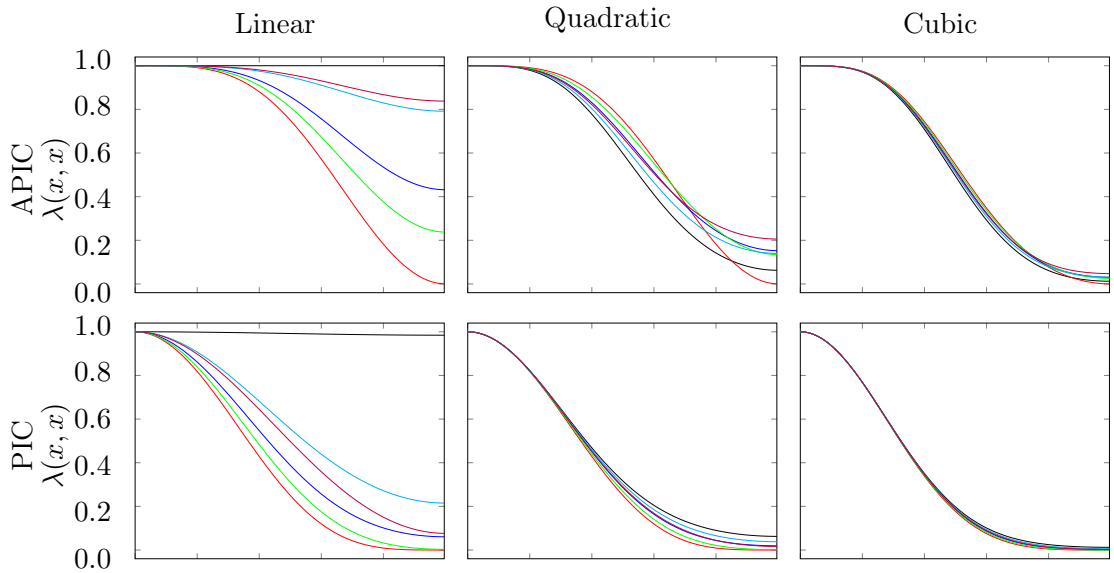
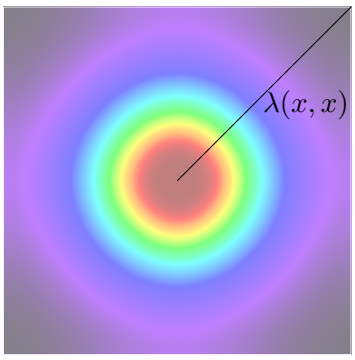


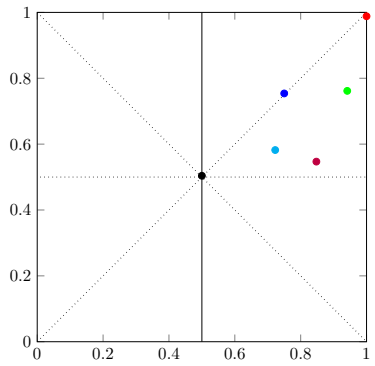
Figure 3.5: Eigenvalues for APIC and PIC, with linear, quadratic, and cubic splines and one particle per cell. The curve colors indicate the location of the particle in a cell, which are shown in Figure 3.6b. The particle position dependence decreases as the interpolation order increases, and this dependence is somewhat more pronounced for APIC.

Numerical study - irregular seeding

Our second eigenvalue image study shows the sensitivity of transfers to particle positioning within a cell. In these tests, we use one particle per cell placed in one of the positions shown in Figure 3.6b. The center (black) point corresponds to putting the particle at the grid degree of freedom location. The red location corresponds to offsetting half a grid cell diagonally away from the degrees of freedom. Because of symmetry, we can restrict our samples to half of one quadrant. Each cell has one particle in the same location. Results are shown in Figure 3.5. As one might expect, one generally observes that particle position dependence decreases as interpolation order increases. Consistent with the above analysis, eigenvalues are larger for APIC than PIC, reflecting its reduced dissipation. One may regard these curves as representing (approximately) the *range* of possible eigenvalues that



(a) Location of cross sections.



(b) Single particle seed locations.

Figure 3.6: Guides for Figures 3.4 and 3.5. The particle seed locations indicate where the particles were located for Figure 3.5. The center black dot corresponds to placing the particle at the location of the grid degree of freedom.

may result from a particular type of transfer (PIC, APIC) with a particular interpolation kernel (linear, quadratic, cubic). The eigenvalue plots for multiple particles are an average of the plots for each particle individually. Thus, it is possible to place bounds on how much dissipation is possible *independent of the particle distribution*. Of course, the distribution is still the same in each cell, so this should be taken as a guide rather than as a hard bound.

This approach to analyzing the effects irregular seeding, though informative, is still quite limited. Every cell has the same number of particles in the same locations, so the overall particle distribution is still very regular. Indeed, Fourier analysis is only possible because of this overall regularity. A globally irregular seeding may still yield eigenvalues (and thus produce dissipation) that is different from what is seen in this analysis. Tiling the particle distribution over a larger block size (for example, having a fixed irregular particle distribution within each 2×2 block of grid cells) would provide a tradeoff between sampling a more irregular particle distribution and the ability to apply Fourier analysis. This would

produce 8 eigenvalue images with half the resolution of the original since Fourier analysis must now be performed over blocks. We do not purse this strategy here.

Taylor-Green vortex

In this analysis, we are particularly interested in understanding the tendency for transfers to damp out vorticity. Our model for a vortex is the Taylor-Green vortex, which has a convenient representation in terms of Fourier basis modes. They are also an analytic solution to the Navier-Stokes equations. This makes it an ideal model for studying dissipation.

Let the physical dimensions of our domain be $[-\pi, \pi] \times [-\pi, \pi]$ and assume that the resolution is square, with $m = n$. The Taylor-Green vortex is given by

$$\mathbf{v}(x, y) = \langle -\sin(ax)\cos(ay), \cos(ax)\sin(ay) \rangle, \tag{3.13}$$

where a is an integer that determines the scale of the vortex. This represents a $2a \times 2a$ checkerboard pattern of vortices which alternate between rotating clockwise and counterclockwise. An example of a Taylor-Green vortex is shown in Figure 3.7a. Larger a correspond to larger numbers of smaller vortices. Stretched vortices may also be considered,

$$\mathbf{v}(x, y) = \langle -\sin(ax)\cos(by), \cos(ax)\sin(by) \rangle, \tag{3.14}$$

with different scales on x and y, but these are not solutions to the Navier-Stokes equations. They are, however, conveniently expressed in Fourier modes.

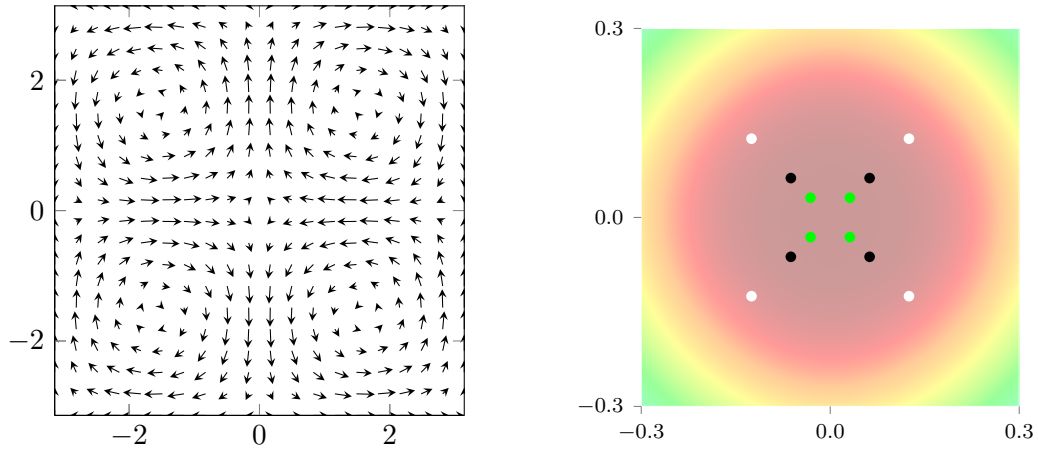
Location of Taylor-Green eigenvalues in image The Fourier mode with wavenumber (k_1, k_2) is $e^{2\pi i x k_1} e^{2\pi i y k_2}$. Observe that the Taylor-Green vortex $\mathbf{v}(x, y)$ is a linear combination of the four modes $(\pm a, \pm a)$. The corresponding eigenvalues are $\lambda(x, y) = \lambda(\pm\frac{a}{m}, \pm\frac{a}{n})$. The location of Taylor-Green modes is shown in Figure 3.7b for a few sample values of a and resolutions.

Scaling of Taylor-Green Due to symmetry, these four eigenvalues are all equal. Thus, the Taylor-Green vortex is transferred into a scaled copy of itself. The factor by which it is reduced is $\lambda(\frac{a}{m}, \frac{a}{n})$. Note that as the resolution $m \times n$ is increased, $(\frac{a}{m}, \frac{a}{n}) \to (0, 0)$. The key to understanding the behavior of vortices under refinement is thus understood by examining the behavior of $\lambda(x, y)$ near $(0, 0)$. We will use the differentiability of $\lambda(x, y)$ to characterize $\lambda(x, y)$ near $(0, 0)$ in Section 3.3.1.

On the other hand, a Taylor-Green vortex of a fixed resolution (e.g., 8 grid cells across) corresponds to a fixed place in the eigenvalue image ($\lambda(\pm\frac{1}{16}, \pm\frac{1}{16})$ in this case). The eigenvalue image thus gives a direct indication of the number of pixels required to resolve a Taylor-Green vortex with a specified amount of dissipation. This is not surprising, since the local dissipation of a vortex should not depend on how large the overall computational domain is. Some Fourier modes corresponding to Taylor-Green vortices are shown in Figure 3.7b.

Dissipation under refinement

In this section, we analyze how the dissipation of Taylor-Green vortices (and stretched vortices) changes under refinement. Fix an initial resolution $m \times n$ and a stretched



(a) An example of Taylor-Green vortex on $[-\pi, \pi]^2$ with $a = 1$ and $b = 1$

(b) Fourier modes of Taylor-Green vortices overlaid on transfer eigenvalues.

Figure 3.7: Taylor-Green vortex and its Fourier modes. In the right part, the location and color of the dots indicate the size (8 pixels: white, 16 pixels: black, 32 pixels: green) of Taylor-Green vortex.

Taylor-Green vortex (a, b). Each round trip transfer scales this vortex by $\lambda(\frac{a}{m}, \frac{b}{n})$. Next, lets scale the resolution to $qm \times qn$. As noted in Section 3.3.1, for $\lambda(\frac{a}{qn}, \frac{a}{qn}) \to \lambda(0, 0)$ as $q \to \infty$. Since all of the transfers under consideration preserve constant velocity fields, $\lambda(0, 0) = 1$. Approximating $\lambda(x, y)$ by a Taylor series and noting $0 \leq \lambda \leq 1$, we have $\lambda(\frac{a}{qn}, \frac{a}{qn}) \approx 1 - cq^{-\gamma}$. This approximation corresponds to the first $\gamma - 1$ mixed partial derivatives of $\lambda(x, y)$ vanishing at $(0, 0)$. It can be shown that $\gamma = 2$ for PIC and $3 \leq \gamma \leq 4$ for APIC. APIC achieves $\gamma = 4$ for quadratic and cubic splines due to the special properties of those splines. The constant c depends on many things, including the layout of the particles.

Under spatial refinement (as $q \to \infty$), we must take more time steps. Assume that $\Delta t = c_2 \Delta x^{\kappa}$, where likely values include $\kappa \in \{1, \frac{3}{2}, 2\}$. Thus, even though dissipation is less at higher resolution, the transfers must be repeated more often. The net dissipation λ is

thus

$$\lambda \approx (1 - cq^{-\gamma})^{Tq^{\kappa}} \approx 1 - c_3 q^{\kappa-\gamma} = 1 - c_4 \Delta x^{\gamma-\kappa}. \tag{3.15}$$

Thus, for first order convergence, we must have $\gamma \geq \kappa + 1$, since otherwise dissipation alone creates an error greater than first order. For PIC, this limits us to $\kappa = 1$. This rules out explicit surface tension ($\kappa = \frac{3}{2}$). On the other other extreme, APIC with quadratic or cubic splines gives $4-\kappa$, which is compatible with third order accuracy with $\Delta t = O(\Delta x)$ and with second order accuracy with $\Delta t = O(\Delta x^2)$. An interesting consequence of this appears to be that APIC cannot be the basis of a method that is higher than third order accurate in space. The predicted dissipation orders of 2 for PIC and 4 for APIC are observed numerically in Figure 3.4.

3.3.2 Particle to grid to particle

Limitations of grid-particle-grid The grid-particle-grid view of transfers is convenient since its results are concisely described by a single image. This image tells us how dissipative transfers are in the limit $\Delta t \to 0$. Unfortunately, the grid-particle-grid path is not linear for FLIP, since new particle velocities ultimately depend on old particle velocities.

Particle-grid-particle An alternative way to examine dissipation is to start with information on particles and simulate the steps that occur until we transfer velocities back to particles. Since none of the methods store state on the grid, this avoids the problem with the grid-particle-grid view for FLIP. The particle-grid-particle view, however, is quite a bit

more complicated. This path includes the pressure projection step, which mixes velocity components in different directions. When each cell gets the same particle distribution, all cells are indistinguishable; the particles within a cell will all be distinguishable unless the particle distribution within a cell is highly symmetrical. In the case of APIC, the $\mathbf{B}_p^n$ matrices must also be considered as degrees of freedom. In 2D with p particles per cell, each cell will contribute $d = 2p$ degrees of freedom for PIC an FLIP but $d = 6p$ for APIC. Rather than a transfer operator M_{ij} that is a scalar per grid node, we must consider our operator to be $M_{(ia)(jb)}$, where a and b run over the d degrees of freedom per cell.

Pressure step is required If we ignore the complications introduced by pressure projection and just do the transfers as we did before, then we run into a different problem. If we make no changes to the grid velocity before transferring back to particles, then FLIP will map back a zero difference. The resulting particle-grid-particle map is an identity map, which does not tell us anything interesting about the transfers. To get any useful insight into FLIP, we must include pressures.

Including pressure We note that pressure projection is linear and is also conveniently diagonalized by the Fourier transform. These properties make it compatible with our analysis. (Viscosity also shares these properties, and one could include it in the analysis. Since our interest is in the inviscid case, we do not do this.) The pressure projection introduces another complication: the particle-grid-particle map is no longer sparse. This simply means the transfers and Poisson solve must be done at the resolution of the final image, which is not a significant problem. Although pressure projection is very convenient to perform

directly in Fourier space, we apply pressure using the same central difference discretization we use for simulation.

Analysis procedure

Block tensor product circulant In the grid-particle-grid case, we were able to reduce the problem to one degree of freedom per cell. The resulting transfer M_{ij} was tensor product circulant and could be diagonalized with a multidimensional Fourier transform. The situation is now more complicated, since each cell has d degrees of freedom associated with it. The matrix $M_{(ia)(jb)}$ is still *block tensor product circulant* in the indices i, j, but not in a, b. That is, if $i = (r, s)$ and $j = (u, v)$ are grid indices in 2D, then $M_{(r,s,a),(u,v,b)} = M_{(r+k,s+m,a),(u+k,v+m,b)}$ for any k and m. Rather than diagonalizing the matrix, the Fourier transform will bring the matrix into a block-diagonal structure.

Constructing and representing the operator The first column of this operator no longer suffices to represent the entire operator, but the first d columns do. That is, $M_{(r,s,a),(u,v,b)} = c_{r-u,s-v,a,b}$. The columns c_{iab} are obtained by initializing all particle velocity degrees of freedom to zero, except particle velocity degree of freedom b in the first grid cell. This is repeated for each of the d degrees of freedom b in the first cell.

Diagonalization procedure Although the matrix $M_{(ia)(jb)}$ is no longer fully diagonalizable by Fourier transforms, Fourier transforms can still be used to render it block diagonal

with $d \times d$ blocks. By computing d^2 multidimensional Fourier transforms, we have

$$\beta_{rsab} = \sum_{u=0}^{m} \sum_{v=0}^{n} c_{uvab} e^{\frac{2\pi i r u}{m}} e^{\frac{2\pi i s v}{n}} \tag{3.16}$$

The β_{rsab} are the diagonal blocks. It associates to every grid cell $i = (r, s)$ a matrix $N_{ab} = \beta_{rsab}$. The eigenvalues of the $d \times d$ matrices N_{ab} are the eigenvalues of $M_{(ia)(jb)}$. Since d is relatively small, we simply compute the eigenvalues of these blocks directly. This procedure computes the $m \times n \times d$ eigenvalues of $M_{(ia)(jb)}$ and conveniently associates d eigenvalues to every cell based on frequency.

Visualization In the grid-particle-grid case, we had a single eigenvalue per grid cell, which we were able to plot conveniently as an image. In the particle-grid-particle case, we now have d eigenvalues with no particular ordering. We sort the d eigenvalues in each cell by magnitude and construct d images. The smallest eigenvalue in each cell corresponds to the first image, the second smallest eigenvalues correspond to the second image, and so on. The visualizations of the results for all methods are shown in Figure 3.8.

Analysis results - PIC and APIC

The results that are obtained for PIC and APIC are quite simple and easily understood. The image formed from the largest eigenvalue is the same image that is constructed in the grid-particle-grid case. The image formed from the second-largest eigenvalue is 1 in the center and zero elsewhere. All of the remaining images are zero.

To see why this is the case, lets consider the three steps involved in the construction of the operator $M = APB$, where B is the transfer from particle to grid, P is the pressure projection, and A is the transfer from grid to particle. We can transform this into Fourier space, in which case we can write $\hat{M} = \hat{A}\hat{P}\hat{B}$, where hats represent the individual operators in Fourier space. Observe that each of these operators is block-diagonal, with one block per grid cell. Each block $\hat{A}$ is $d \times 2$, $\hat{P}$ is 2×2, and $\hat{B}$ is $2 \times d$. Thus, M has rank at most 2. This explains why only two images are nonzero; the $d - 2$ zero images correspond to nullspace modes of the particle to grid transfer.

Next, we turn to the first two images. Observe that $\hat{M} = \hat{A}\hat{P}\hat{B}$ and $\hat{N} = \hat{B}\hat{A}\hat{P}$ have the same nonzero eigenvalues with the same multiplicity. (Indeed, if u is a vector with nonzero eigenvalue λ, then $\lambda u = \hat{M}u = \hat{A}\hat{P}\hat{B}$. $v = \hat{B}u$ is an eigenvector of $\hat{N}$, since $\hat{N}v = \hat{B}\hat{A}\hat{P}\hat{B}u = \lambda\hat{B}u = \lambda v$. Observe that $\lambda \neq 0$ implies $v \neq 0$. It is also easy to see that distinct eigenvectors of $\hat{M}$ for the same nonzero eigenvalue map to distinct eigenvectors of $\hat{N}$.)

The operator $\hat{N} = \hat{B}\hat{A}\hat{P}$ is composed of two pieces. Recall that BA is the grid-particle-grid transfer operator. The blocks of $\hat{B}\hat{A}$, which I denote using subscripts as $(\hat{B}\hat{A})_i = \hat{B}_i\hat{A}_i$ must be of the form $\lambda_i\mathbf{I}$, where λ_i was the eigenvalue computed for that cell in the grid-particle-grid case. Thus, $\hat{N}_i = \lambda_i\hat{P}_i$.

For $i = (0, 0)$, $\hat{P}_i = \hat{P}_{(0,0)} = \mathbf{I}$, since the constant translation modes are divergence free. Since constant translation is preserved by PIC and APIC, $\lambda_{(0,0)} = 1$. This explains why the middle pixel of both images corresponds to eigenvalue 1.

For any other $i \neq (0,0)$, $\hat{P}_i = \mathbf{I} - \frac{ii^T}{i^T i}$ is a rank-one projection operator with eigenvalues 0 and 1. The eigenvalues of $\hat{N}_i$ are 0 and λ_i. Since the largest eigenvalue is always λ_i, the image of maximum eigenvalues is the same as the grid-particle-grid case. The second eigenvalue of $\hat{P}_i$ being 1 for the middle and 0 elsewhere explains the image for the second-to-largest eigenvalue. Although the projection was actually performed with the finite difference stencil, the images look the same.

Analysis results - FLIP

The results for FLIP are very different. In this case, all images but one are identically 1. The only nontrivial image is filled with the smallest eigenvalues and is shown in the last row of Figure 3.8.

The explanation for the images with eigenvalue 1 everywhere is similar to the reason for the trivial images for PIC and APIC. One of the trivial images corresponds to the modes that are transferred to the grid, resulting in nonzero divergence-free velocity fields that are unaffected by the pressure projection. Since no grid velocity change occurs, FLIP behaves as the identity map on these modes. The other trivial images correspond to null modes of the particle to grid transfer. These produce a zero grid velocity, which is unchanged by pressure. In the case of FLIP, however, the grid velocity difference (zero) is mapped back, resulting in no change. The operator behaves as the identity map on these modes.

That leaves the nontrivial mode. This appears qualitatively similar to the nontrivial modes observe for PIC, except that it is inverted and has a single pixel corresponding to eigenvalue 1 in the middle. (Compare with Figure 3.5.) This is the mode corresponding to

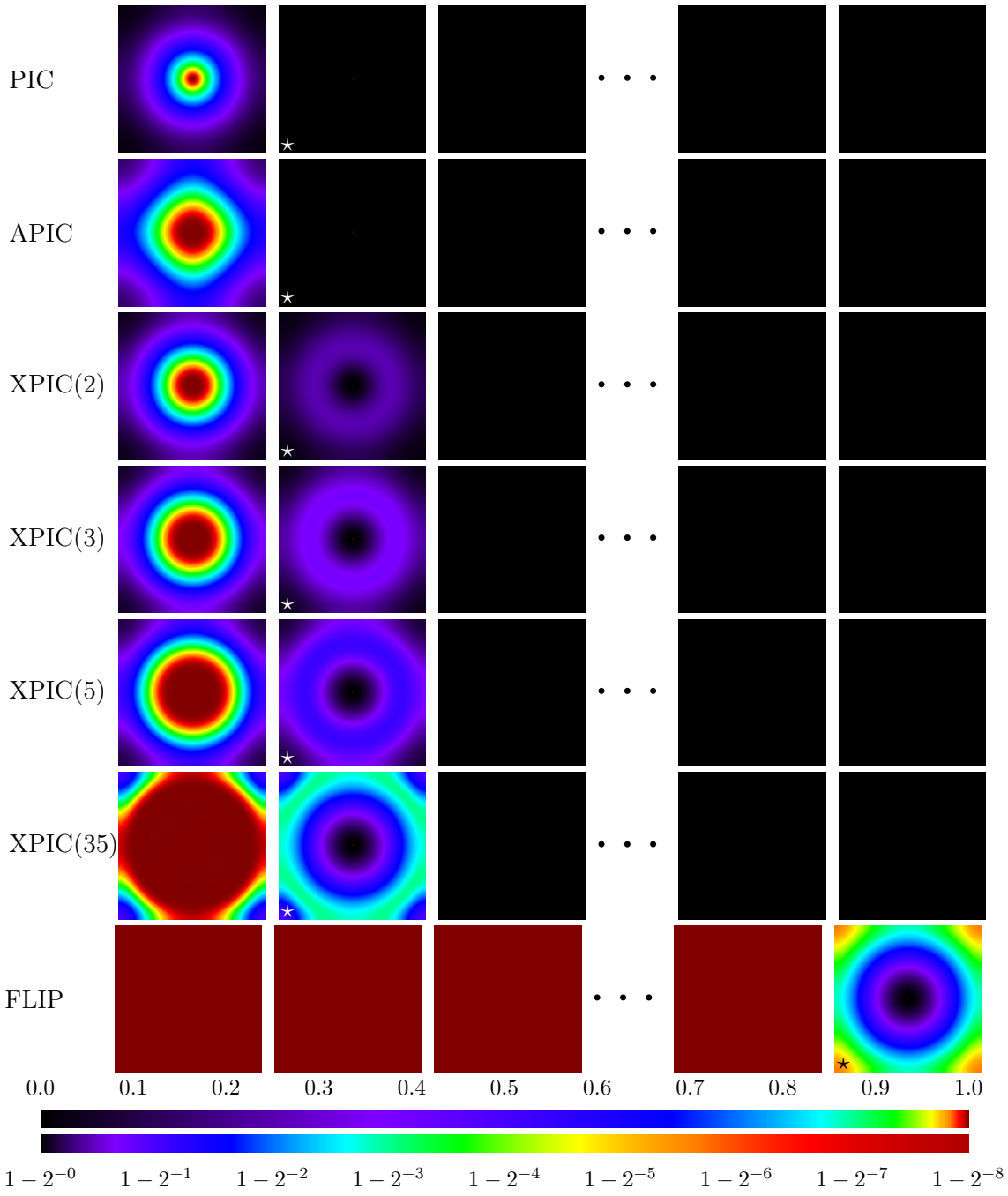


Figure 3.8: Eigenvalues for particle-grid-particle transfers, with **4 particles per cell** and **quadratic** splines. Some trivial eigenvalue images are omitted. Images with a "$\star$" have a red dot in the center of the image (constant velocity mode). The dissipation of APIC is approximately between XPIC(2) and XPIC(3). The dissipation of XPIC(m) improves significantly with order. In exchange, XPIC(m) lets through some undesirable modes (second column), which is minimal for low orders but grows steadily with m. The general behavior of FLIP is radically different from XPIC(m) or APIC; it lets through almost everything.

the pressure projection. The eigenvalue 1 in the middle corresponds to the constant velocity being divergence free. In the case of PIC and APIC, the rest of the image would be black, since all other divergent modes should be projected out. In the case of FLIP, however, this is not so. For low-frequency modes, the image appears black, and these divergent modes are projected well. Higher frequency modes, however, do not readily survive the transfer to the grid. Since only a relatively small portion of the high-frequency velocity modes survive to the grid, only that small amount of velocity can be projected out by the pressure solve. Since only a small grid change was made, only a small change is made to the particle. The result is that high-frequency divergent velocity modes on particles are not efficiently projected. This goes some way towards explaining why updating FLIP particle positions with particle velocities produces very bad results; in addition to the particle velocities being noisy, the velocity field on the particles is not even divergence free.

3.3.3 Analysis results - XPIC

In XPIC, all but two eigenvalues are all zero; this is related to the nullspace of XPIC transfer. This is an immediate improvement from FLIP, since the amount of noise that can survive on particles is already drastically reduced. In this way, XPIC is far more like PIC or APIC than it is like FLIP. The largest of the nontrivial eigenvalues is similar to APIC, and the eigenvalue improves as the XPIC order increases. XPIC(1) matches PIC, since the schemes are the same. The dissipation of APIC lies somewhere between XPIC(2) and XPIC(3). For higher orders, XPIC is significantly less dissipative than APIC. In practice, XPIC is typically run at XPIC(2) or XPIC(5), with the latter being significantly less dissipative than APIC.

When we start looking at the second largest eigenvalue, we see that the reduced dissipation of XPIC comes at a minor cost in the effectiveness of projection. Because XPIC performs particle-to-grid and grid-to-particle transfers repeatedly without projecting divergent modes, these divergent modes can be present in the particle velocity. These modes are not removed by pressure projection and their frequency response appear as "halo" in the other nontrivial eigenvalue (see the XPIC rows in Figure 3.8). Even by XPIC(3), some of these halo eigenvalue are already larger than 0.4. Practically, this means that particle divergence may persist for several time steps, but it cannot accumulate over time. It should be noted that XPIC was not constructed as a transfer scheme for incompressible fluids, so the interplay between transfers and projection was not a consideration during its development.

3.4 Implementation notes

3.4.1 Extrapolation

When a boundary is not periodic, transfers will require information from outside the fluid domain. We handle this by extrapolating information into the ghost region. For all transfer algorithms, extrapolation happens before the grid-to-particle transfer and after the particle-to-grid transfer.

Reflected particles Consider two physical scenarios. In the first we have particles that are moving towards a solid wall, which causes them to stop moving normal to the wall and slide tangentially along it. In the second scenario, the wall is missing but instead a mirror image of the particles is added on the other side of the wall. The particles will move

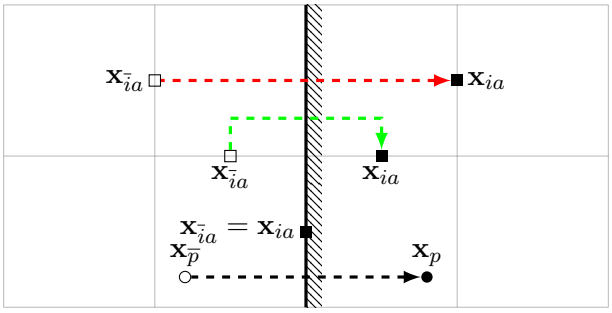


Figure 3.9: Grid or particle attributes (solid) and their reflected counterparts (hollow). Faces on the boundary are their own reflections. The hatched sides indicate the inside domain.

towards the plane where the wall was and collide with each other, resulting in them sliding tangentially along where the wall was. The results is these two cases are the same. We can therefore mimic the effects of colliding with a wall by simply mirroring the particles on the other side. We do not actually create extra particles outside; we simply perform a *touch up* after doing particle-to-grid transfers so that they behave as if there were reflected particles. The corrections depend on the type of boundary condition and are simple modifications that are applied to the grid in a thin layer near the boundary. These take the form of adding data that was transferred into the ghost region to the reflected location inside, possibly with a change of sign.

We use a bar over a quantity to denote its corresponding reflected counterpart across the grid boundary. For example, $\mathbf{x}_{\bar{i}a}$ and $\mathbf{x}_{\bar{p}}$ are the locations of reflected grid faces and particles. For contrast, $\overline{\mathbf{x}_{ia}}$ and $\overline{\mathbf{x}_p}$ are the reflected locations of a grid face and a particle. Since we only perform this reflection across grid faces, reflecting the location of one face always results in the location of another face. We are thus justified in defining $\mathbf{x}_{\bar{i}a} = \overline{\mathbf{x}_{ia}}$ and $\mathbf{x}_{\bar{p}} = \overline{\mathbf{x}_p}$. Note that the particle represented by $\bar{p}$ is only conceptual and for derivation purposes; we do not actually construct these particles.

They are shown in Figure 3.9. Since the basis functions are invariant to reflections, the weight generated from reflected particle to the inside domain index, $w_{i\overline{i}pa} = N(\mathbf{x}_{ia} - \mathbf{x}_{\overline{p}}) = N(\overline{\mathbf{x}_{\overline{i}a} - \mathbf{x}_p}) = N(\mathbf{x}_{\overline{i}a} - \mathbf{x}_p) = w_{\overline{i}pa}$. Similarly we also have $w_{\overline{i}\overline{p}a} = w_{ipa}$. Consider a general grid attribute q_{ia} (which could be mass or a component of velocity, momentum, or force) inside the domain consists of contributions from internal particles and reflected particles. We want the reflected value to be $q_{\overline{p}} = bq_p + c_p$, where $b = \pm 1$ and c_p depend on the type of attribute and boundary conditions (See Figure 3.10). Assuming for simplicity a PIC transfer,

$$q_{ia} = \sum_p w_{ipa} q_p + \sum_{\overline{p}} w_{i\overline{p}a} q_{\overline{p}} = \sum_p w_{ipa} q_p + \sum_p w_{\overline{i}pa}(bq_p + c_p)$$

$$q_{\overline{i}a} = \sum_p w_{\overline{i}pa} q_p + \sum_{\overline{p}} w_{\overline{i}\overline{p}a} q_{\overline{p}} = \sum_p w_{\overline{i}pa} q_p + \sum_p w_{ipa}(bq_p + c_p)$$

$$bq_{\overline{i}a} = \sum_p w_{\overline{i}pa} bq_p + \sum_p w_{ipa}(q_p + bc_p) = q_{ia} - \sum_p w_{\overline{i}pa} c_p + b \sum_p w_{ipa} c_p$$

Compare these with the grid values $\hat{q}_{ia}$ that would be if no reflected particles were used.

$$\hat{q}_{ia} = \sum_p w_{ipa} q_p \qquad\qquad \hat{q}_{\overline{i}a} = \sum_p w_{\overline{i}pa} q_p$$

In the case $b = 1$ and $c_p = 0$, these rules amount to $q_{ia} = q_{\overline{i}a} = \hat{q}_{ia} + \hat{q}_{\overline{i}a}$. That is, simply copy the ghost values into the interior, and then copy these values back into the ghost region. In the case $b = -1$ and $c_p = 0$, $q_{ia} = -q_{\overline{i}a} = \hat{q}_{ia} - \hat{q}_{\overline{i}a}$. This is implemented by subtracting ghost values from the inside values, then copying the inside data to the ghost region with

a sign flip. Notice that these rules are very simple grid-based fixes that only need to be applied near the boundary.

The case $c_p \neq 0$ is only needed for inhomogeneous boundary conditions, which are only relevant for velocities. In this case, we would be enforcing $v = v_{bc}$ at the boundary. However, we actually transfer momentum from particles to grid, not velocity directly. Thus, we must reflect about the desired value of momentum for the particles, $c_p = 2v_{bc}m_p$. Applied to the transfer rules, we see

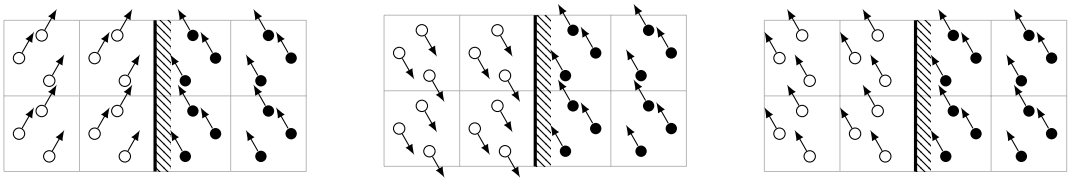
$$q_{ia} = \sum_p w_{ipa} q_p + \sum_p w_{\bar{i}pa}(bq_p + c_p) = \hat{q}_{ia} + b\hat{q}_{\bar{i}a} + \sum_p w_{\bar{i}pa} 2v_{bc} m_p = \hat{q}_{ia} + b\hat{q}_{\bar{i}a} + 2m_{\bar{i}a} v_{bc}$$

$$bq_{\bar{i}a} = q_{ia} - \sum_p w_{\bar{i}pa} 2v_{bc} m_p + b\sum_p 2w_{ipa} v_{bc} m_p = q_{ia} + 2(b\hat{m}_{ia} - \hat{m}_{\bar{i}a})v_{bc},$$

where $\hat{m}_{ia}$ also refers to velocity before the boundary condition treatment has been applied. The same correction rules can be obtained for APIC transfers by defining $\mathbf{b}_{\overline{p}a}$ appropriately.

We use $b = 1$ and $c_p = 0$ for (1) masses, (2) free surface momentum transfers, and (3) the tangential components of momentum transfers for slip boundary conditions. We use $b = -1$ and $c_p = 0$ for (1) no-slip boundary conditions and (2) normal components of momentum transfers for slip boundary conditions. See Figure 3.10. We use $c_p \neq 0$ for inhomogeneous velocity boundary conditions. We perform this kind of extrapolation for mass and momentum after the particle-to-grid transfer.

We also perform extrapolation after forces are applied. Ideally, velocities should continue to satisfy appropriate boundary conditions after forces have been applied. This is accomplished by extrapolating forces using the homogeneous version of the same boundary



(a) Slip (b) No-Slip (c) Free surface

Figure 3.10: Reflecting velocities across the boundary in different ways mimics different types of boundary conditions. The hatched side indicates the inside domain. The hollow circles are reflected particles.

condition applied to velocities. For our analytic body force, no modifications should be made to the for inside region, since these forces are analytic and thus correct. We do, however, extrapolate the inside values to the ghost region.

There is another extrapolation performed after grid evolution, since the grid-to-particle transfers will read from the ghost region. As with analytical forces, we leave the interior values alone (the pressure projection boundary conditions ensure that the interior values already satisfying the boundary conditions) but extrapolate the interior values to the ghost regions according to the boundary condition type. This step is especially important with FLIP transfers, since FLIP will compute the velocity difference based on current and previous velocity, which has been already extrapolated.

We also note that there will be overlapped access at the corners of ghost region. The extrapolation rules for each direction are compatible with each other, this is just a matter of correct implementation. Accumulate ghost data into the interior first, then extrapolate it back to the ghost region as a second pass.

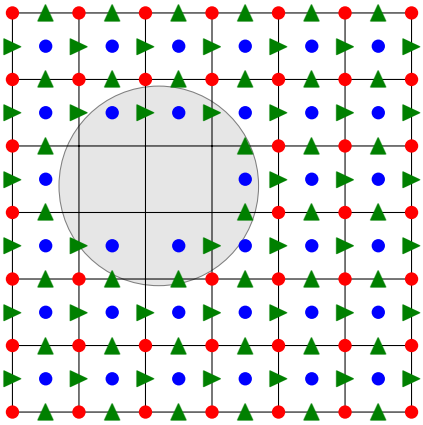


Figure 3.11: Cut-cell layout for our degrees of freedom. Red nodes are inside the fluid. Blue cell centers contain pressure degrees of freedom during the pressure projection. The green triangles are MAC faces where fluid velocities are being projected.

3.4.2 Cut-cell formulation

Some of our tests include irregular objects. For these tests we use the more accurate pressure projection formulation of [130]. The objects are represented by a level set on grid nodes. Nodes not in an object are in fluid. MAC faces hold valid velocity degrees of freedom if any of their nodes are in fluid. MAC cells hold valid pressures if any of their nodes are in fluid. Note that if a MAC face is valid then both neighboring cells are also valid. See Figure 3.11. This discretization is just the regular central differencing stencil away from objects. Since our objects do not touch the domain walls, we handle domain wall boundary conditions as in the finite differencing discretization.

Three concerns must be addressed for this layout to be valid. (1) Valid velocities must be available at all MAC faces being projected. Each valid face has at least one valid node. Provided the geometry is adequately resolved on the grid, this valid node will have a cell that is entirely inside fluid; reseeding will guarantee that this cell has particles. All

of these particles will contribute to the velocity on the face being considered provided the transfer stencil is at least as wide as the quadratic stencil. (2) Valid faces must have valid pressures on either side (except at the domain walls) so that the pressure gradient can be applied. Our discretization has this property as noted above. (3) We must have enough valid grid velocities to transfer back to particles. This is not normally true for us, and the way we handle this determines the type of boundary condition we enforce. If the object has no-slip boundary conditions, then we set the invalid velocities inside the object with the object's velocity. If the object has no-slip boundary conditions, then we do moving least squares transfers as described below.

3.4.3 Moving least squares

APIC transfers may be formulated as Moving Least Squares (MLS) [57, 88] by expressing the transfers as a least squares optimization. If there is only one particle p and we were doing co-located APIC transfers, then a grid-to-particle-to-grid ($\tilde{\mathbf{v}}_i^{n+1} \rightarrow \mathbf{v}_i^{n+1}$) with $\Delta t = 0$ (so that $w_{ip}^{n+1} = w_{ip}^n$ and $\mathbf{x}_p^{n+1} = \mathbf{x}_p^n$) can be expressed as

$$\mathbf{v}_p^{n+1} = \sum_i w_{ip}^n \tilde{\mathbf{v}}_i^{n+1} \qquad \mathbf{B}_p^{n+1} = \sum_i w_{ip}^n \tilde{\mathbf{v}}_i^{n+1} (\mathbf{x}_i^n - \mathbf{x}_p^n)^T$$

$$\mathbf{D}_p^{n+1} = \sum_i w_{ip}^n (\mathbf{x}_i^n - \mathbf{x}_p^n)(\mathbf{x}_i^n - \mathbf{x}_p^n)^T \qquad m_i^{n+1} = w_{ip}^n m_p$$

$$\mathbf{C}_p^{n+1} = \mathbf{B}_p^{n+1} (\mathbf{D}_p^{n+1})^{-1} \qquad m_i^{n+1} \mathbf{v}_i^{n+1} = w_{ip}^n m_p (\mathbf{v}_p^{n+1} + \mathbf{C}_p^{n+1} (\mathbf{x}_i^n - \mathbf{x}_p^n))$$

The same $\mathbf{v}_p^{n+1}$ and $\mathbf{C}_p^{n+1}$ are obtained by choosing $\mathbf{v}_p^{n+1}$ and $\mathbf{C}_p^{n+1}$ to minimize the objective

$$E = \sum_i w_{ip}^n \left\| \mathbf{v}_i^{n+1} - \tilde{\mathbf{v}}_i^{n+1} \right\|^2 = \sum_i w_{ip}^n \left\| \mathbf{v}_p^{n+1} + \mathbf{C}_p^{n+1}(\mathbf{x}_i^n - \mathbf{x}_p^n) - \tilde{\mathbf{v}}_i^{n+1} \right\|^2.$$

That is, the particle data is chosen so that the round trip transfer preserves the grid velocity field in a weighted least squares sense. The nice part about this formulation is that it makes sense when some data is invalid. Let $s_i = 1$ for valid nodes and $s_i = 0$ for invalid nodes. Then, $\mathbf{v}_p^{n+1}$ and $\mathbf{C}_p^{n+1}$ can be chosen to minimize

$$E = \sum_i s_i w_{ip}^n \left\| \mathbf{v}_p^{n+1} + \mathbf{C}_p^{n+1}(\mathbf{x}_i^n - \mathbf{x}_p^n) - \tilde{\mathbf{v}}_i^{n+1} \right\|^2.$$

If $\mathbf{B}_p^{n+1}$ is used for state, then it can be computed with

$$\mathbf{D}_p^{n+1} = \sum_i s_i w_{ip}^n (\mathbf{x}_i^n - \mathbf{x}_p^n)(\mathbf{x}_i^n - \mathbf{x}_p^n)^T \qquad \mathbf{B}_p^{n+1} = \mathbf{C}_p^{n+1}\mathbf{D}_p^{n+1}$$

For MAC transfers, a similar minimization problem can be formulated as

$$E = \sum_{ia} s_{ia} w_{ipa}^n \left(\mathbf{e}_a^T \mathbf{v}_p^{n+1} + (\mathbf{c}_{pa}^{n+1})^T (\mathbf{x}_{ia}^n - \mathbf{x}_p^n) - \tilde{v}_{ia}^{n+1} \right)^2,$$

This leads to the following algorithm. First, compute the intermediates

$$q_{pa} = \sum_i s_{ia} w_{ipa}^n \quad \mathbf{g}_{pa} = \sum_i s_{ia} w_{ipa}^n (\mathbf{x}_{ia}^n - \mathbf{x}_p^n) \quad \mathbf{D}_{pa}^{n+1} = \sum_i s_{ia} w_{ipa}^n (\mathbf{x}_{ia}^n - \mathbf{x}_p^n)(\mathbf{x}_{ia}^n - \mathbf{x}_p^n)^T$$

$$u_{pa} = \sum_{ia} s_{ia} w_{ipa}^n \tilde{v}_{ia}^{n+1} \qquad \mathbf{h}_{pa} = \sum_{ia} s_{ia} w_{ipa}^n (\mathbf{x}_{ia}^n - \mathbf{x}_p^n) \tilde{v}_{ia}^{n+1}$$

Then, solve a linear system and compute $\mathbf{b}_{pa}^{n+1}$ and $\mathbf{v}_p^{n+1}$.

$$\begin{pmatrix} q_{pa} & \mathbf{g}_{pa}^T \\ \mathbf{g}_{pa} & \mathbf{D}_{pa}^{n+1} \end{pmatrix} \begin{pmatrix} y_{pa} \\ \mathbf{c}_{pa}^{n+1} \end{pmatrix} = \begin{pmatrix} u_{pa} \\ \mathbf{h}_{pa} \end{pmatrix} \qquad \mathbf{b}_{pa}^{n+1} = \mathbf{D}_{pa}^{n+1}\mathbf{c}_{pa}^{n+1} \qquad \mathbf{v}_p^{n+1} = \sum_a y_{pa}\mathbf{e}_a.$$

This linear system is symmetric positive semidefinite but may be singular if insufficient valid data is available; it should be solved in the minimum norm least squares sense. The linear system is only 4×4 in 3D, so this algorithm is quite efficient. One may see that this algorithm decays into the usual transfers when all velocities are valid ($s_{ia} = 1$). In this case, $q_{pa} = 1$, $\mathbf{g} = \mathbf{0}$, $u_{pa} = y_{pa} = \mathbf{e}_a^T\mathbf{v}_p^{n+1}$, and $\mathbf{h}_{pa} = \mathbf{b}_{pa}^{n+1}$.

3.4.4 Reseeding

To correct the particle coverage, we reseed the particles in a similar way to [23]. For each cell the number of particles are counted and if it is below 2 we randomly sample new particles in the cell so that the number of particles meet the minimum requirement. The velocities for the new particles are interpolated from the grid. For a cell which is partially occupied by an object, we reject the sample positions that are inside the object. If the number of particles in a cell exceeds 2^{d+1} where d is the dimension, we randomly select particles for removal. We perform a particle-to-grid transfer if any new particles are added to ensure consistency between particles and grid. This extra transfer is only necessary for XPIC, but we perform it for all transfers to keep the evolution as similar as possible.

3.4.5 Grid forces

We use the body force in the Navier-Stokes equations as a forcing term to make chosen velocity and pressure fields into analytic solutions. This is done by substituting the velocity and pressure (assuming zero viscosity) into the equations:

$$\mathbf{f} = \frac{\partial \mathbf{v}}{\partial t} + \nabla \mathbf{v} \cdot \mathbf{v} + \frac{1}{\rho} \nabla p.$$

3.5 Numerical examples

3.5.1 Convergence

To demonstrate the correctness of the APIC transfers, as well as the correctness of the PIC and FLIP versions of the scheme used for comparison, we perform some simple convergence tests. All velocity errors are reported using the grid velocity at the end of the time step $\tilde{v}_{ia}^{n+1}$ as well as using the particle velocities at the end of the time step $\mathbf{v}_p^{n+1}$. For each test, we report the L^∞ and L^2 errors calculated according to the formulas

$$L_G^\infty = \max_{ia} \left| \tilde{v}_{ia}^{n+1} - \mathbf{v}(\tilde{\mathbf{x}}_{ia}^{n+1}, t^{n+1}) \cdot \mathbf{e}_a \right| \qquad L_G^2 = \sqrt{\frac{1}{N_G} \sum_{ia} \left(\tilde{v}_{ia}^{n+1} - \mathbf{v}(\tilde{\mathbf{x}}_{ia}^{n+1}, t^{n+1}) \cdot \mathbf{e}_a \right)^2}$$

$$L_P^\infty = \max_{p} \left\| \mathbf{v}_p^{n+1} - \mathbf{v}(\mathbf{x}_p^{n+1}, t^{n+1}) \right\|_\infty \qquad L_P^2 = \sqrt{\frac{1}{N_P} \sum_{p} \left\| \mathbf{v}_p^{n+1} - \mathbf{v}(\mathbf{x}_p^{n+1}, t^{n+1}) \right\|_2^2}$$

where $N_G = \sum_{ia} 1$, $N_P = \sum_{p} 1$ are the numbers of simulated grid faces and particles and $\mathbf{v}(\mathbf{x}, t)$ is the analytic velocity field.

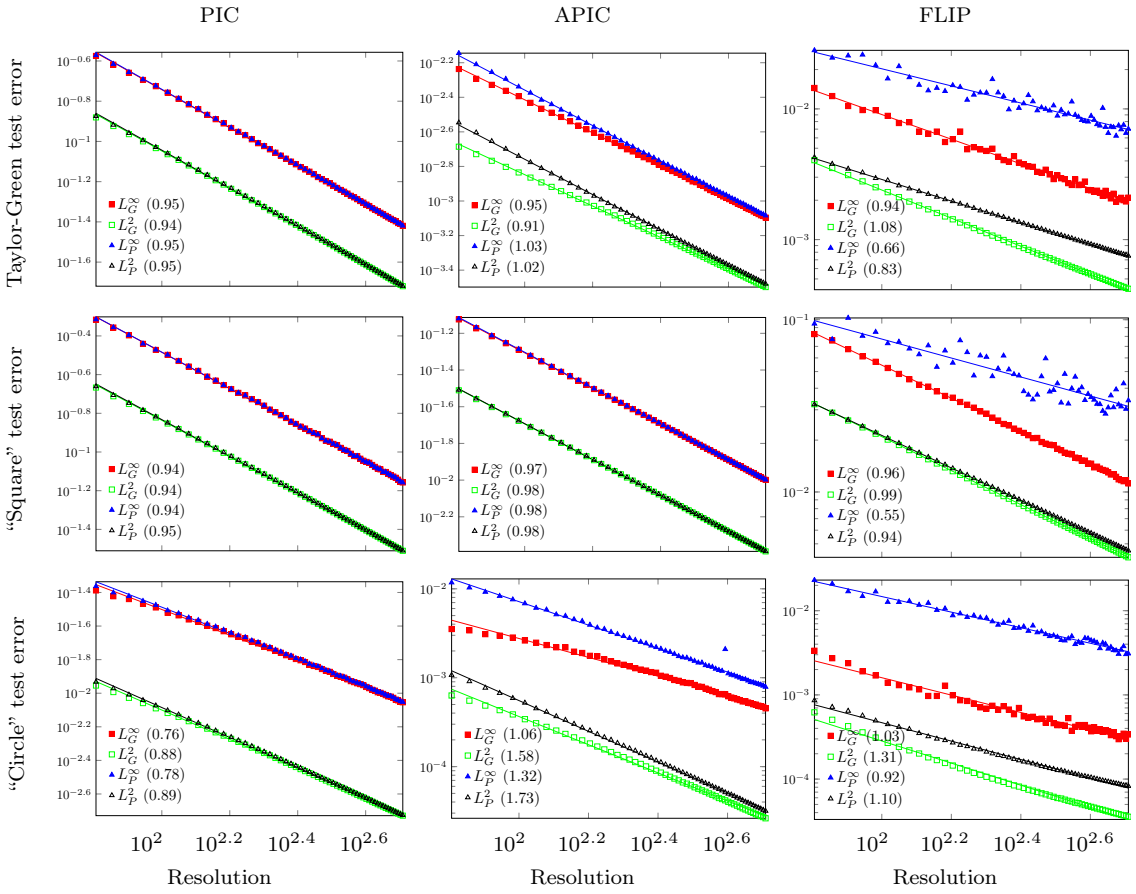


Figure 3.12: Convergence tests with three different transfers for each of the four error measures. The markers are the actual error computed, and the lines are least squares regression lines used to calculate the convergence order. Convergence orders are listed in the legends. Resolution is the number of cells in each direction of the grid.

Taylor Green

For this test, we verify the convergence of a Taylor-Green vortex. We use a $[-\pi, \pi]^2$ domain with initial velocity field $\mathbf{v}_0 = \langle -\sin(ax)\cos(ay), \cos(ax)\sin(ay)\rangle$. The fluid has physical properties $\rho = 3$ and $\mu = 0$. The analytic solution is $\mathbf{v}(\mathbf{x}, t) = \mathbf{v}_0(\mathbf{x})e^{-2a^2\nu t}$, where $\nu = \frac{\mu}{\rho}$. All of the tests were run with $\Delta x = \frac{2\pi}{N}$ and $\Delta t = \frac{1}{N}$ to a final time of $T = 1$, where N is the number of grid cells in each direction. The same test was run with PIC, APIC, and FLIP transfers. The same particle distribution is used for each test, which was computed as

blue noise by Poisson disk sampling. In particular, the particle distribution is never the same per cell (and thus the transfer matrices are not circulant). Convergence plots are shown in the first row of Figure 3.12. A few observations stand out immediately from the convergence plots. PIC converges convincingly at first order with close agreement between particle and grid states. APIC converges cleanly at first order, but here the particles have some error relative to grids; this is because the extra velocity contribution on particles (due to $\mathbf{b}_{pa}^{n+1}$) was not taken into account during the error analysis; the significance of this difference diminishes under refinement. The most striking observation is with respect to FLIP. When errors are measured on the grid, first order convergence is observed. On particles, however, the convergence is weaker than first order. The reason for the reduced convergence order on FLIP is unknown, but it is suspected to be related to the accumulation of error in transfer null modes on particles (indeed, this error does not seem to affect grid convergence, though one may wonder for how long this situation can persist).

"Square"

For this test, we verify the convergence of an analytic velocity field with slip boundary conditions. First we make two stream functions $\phi_1(x,y) = f(x)f(y)$ and $\phi_2(x,y) = g(x)f(y)$, where $f(x) = x(1-x)(x^2-x-1)$ and $g(x) = x(1-x)(x+1)(3x^2-7)$. Then the analytic velocity field constructed from these stream functions as $\mathbf{v} = \langle -\frac{\partial \phi_1}{\partial y}, \frac{\partial \phi_1}{\partial x} \rangle + t\langle -\frac{\partial \phi_2}{\partial y}, \frac{\partial \phi_2}{\partial x} \rangle$ on domain $[0,1]^2$. This velocity field is divergence free. The stream functions are chosen such that the normal directional derivative of tangential velocity vanishes on the boundary. Thus it is compatible with the slip boundary conditions. The analytic pressure is chosen as $p(x,y,t) = xy(1-x)(1-y)(x-xy+y^2+t)$, so that $p = 0$ at the boundaries. The fluid has

physical properties $\rho = 3$ and $\mu = 0$. All of the tests were run with $\Delta x = \frac{1}{N}$ and $\Delta t = \frac{1}{4N}$ to a final time of $T = 1$. The chosen analytic velocity field has a larger peak speed, so we use a smaller Δt for this simulation, and the convection would not affect out results. Convergence plots are shown in the second row of Figure 3.12. With the help of extrapolation, we get linear order convergence except L_P^∞ error of FLIP. This is due to the same reason as in the Taylor-Green convergence test.

"Circle"

For this test, we verify the convergence of a velocity field with object boundaries that are not aligned with the grid. We use a $[-2, 2]^2$ domain but exclude a circle of radius 1 at the origin. All boundaries are slip. We construct a divergence-free velocity from the stream function S defined by

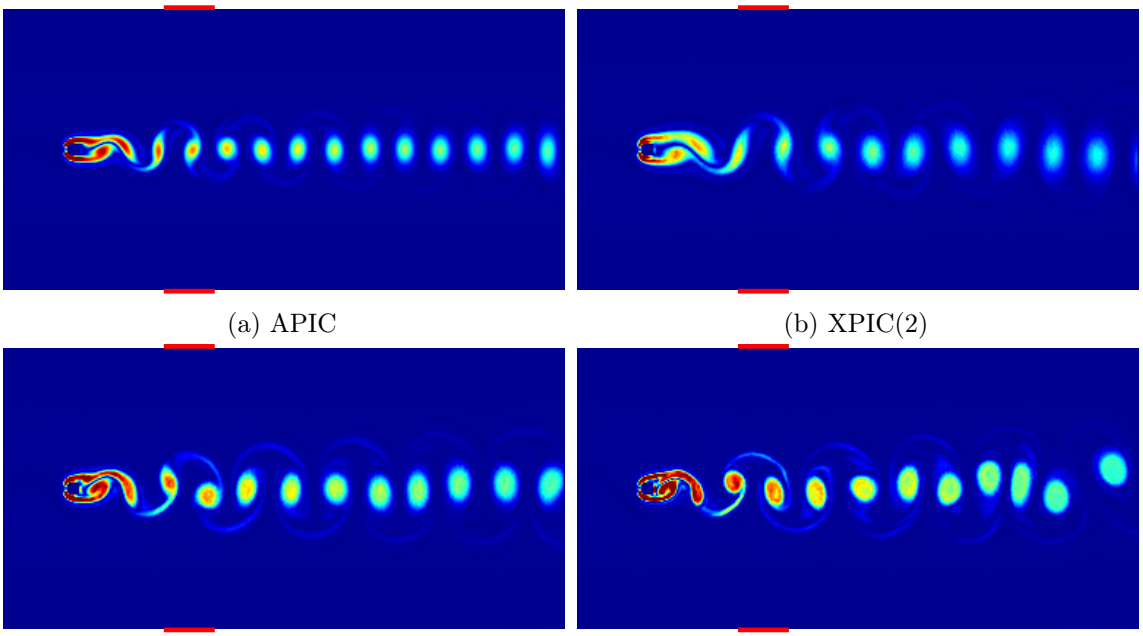
$$\begin{aligned} S = {} & \frac{1}{177500000}(y-2)(y+2)(x-2)(x+2)(x^2+y^2-1) \\ & \times (1775x^6y^4 + 1775x^4y^6 - 14585x^6y^2 - 31229x^4y^4 - 14585x^2y^6 + 36100x^6 + 158486x^4y^2 \\ & + 158486x^2y^4 + 36100y^6 - 248104x^4 - 538625x^2y^2 - 248104y^4 \\ & + 384500x^2 + 384500y^2 - 213392) \end{aligned}$$

This field is chosen so that the normal velocities and the normal derivative of tangential velocity are 0 at all boundaries. The analytic pressure is chosen as $p(x, y, t) = x - xy + y^2 + t$. The fluid has physical properties $\rho = 3$ and $\mu = 0$. All of the tests were run on an $N \times N$ grid with $\Delta x = \frac{4}{N}$ and $\Delta t = \frac{4}{N}$ to a final time of $T = 1$. Cut-cell discretization is used to handle the curved circle boundary, moving least squares are used for transfers, and reseeding is used

to maintain particle coverage. Convergence plots are shown in the last row of Figure 3.12. In all tests the particle and grid error measured in L^2 norm reach the first order convergence. The convergence of L_P^∞ error also approximately reach the first order but some outliers are observed in APIC. This occurred because particles were seeded so close to the boundary edge that some of their weights were near roundoff error. Since a quadratic spline is used, only one row of velocities is available, which is insufficient to reconstruct the full velocity. The MLS system in this situation is singular, which leads to an inaccurate transfer. We note that when transferred back to the grid, this particle will accurately interpolate velocity to the faces that are well-supported. The velocity interpolated to the nearly unsupported faces is inaccurate, but the weights are so small that this is irrelevant. This is why the grid velocity is clean. This problem could be avoided in a number of ways, including simply preventing particles from being so close to the domain boundaries.

Vortex shedding

In this test we simulate a constant velocity field $\langle 1, 0\rangle$ passing a circle. We use a $[-2, 14] \times [-4, 4]$ domain, with slip boundary conditions on its $y = \pm 2$ sides, inflow ($u = 1$) at $x = -2$, and free surface on the $x = 14$ side. The circle is centered at $\langle 0, 0\rangle$ with radius 0.25. The fluid has physical properties $\rho = 1$ and $\mu = 0$. Initially particles are sampled by blue noise in the domain. New particles are created to fill the vacancy as existing particles move in the $+x$ direction. Cut-cell discretization and reseeding are used for this test. The same test was run with APIC and XPIC(2,3,5) transfers. We run the tests to a cyclical state and then examine the frequency of vortex shedding. To extract the vortex signal, we



(a) APIC (b) XPIC(2)

(c) XPIC(3) (d) XPIC(5)

Figure 3.13: Vortex shedding simulation with APIC and XPIC transfers. The red bars above and below the images indicate the range where vorticity is computed to extract vortex signal.

compute vorticity inside a chosen window. This total vorticity ranges from positive (when a vortex with positive vorticity is centered in the window) to negative (when a vortex spinning the opposite way passes). This curve is oscillatory, and we compute its frequency using a Fourier transform. We show the simulation results in Figure 3.13. APIC sheds at frequency 0.49 Hz, and XPIC family sheds at frequency 0.39 Hz (XPIC(2)), 0.42 Hz (XPIC(3)), and 0.41 Hz (XPIC(5)). The corresponding Strouhal numbers are between 0.19 and 0.24. This is consistent with fluid flow over a range of higher Reynolds numbers ($300 - 10^6$).

3.5.2 Dissipation and noise

Taylor Green vortex

Measuring vorticity

One of the main objectives of this work is to analyze how well vorticity is preserved under different transfers. To do this, we need useful measures of how well vorticity is preserved. We use two scalar measures

$$E_{vel} = \int_\Omega \|\mathbf{u}\|^2 \, dV \qquad\qquad E_{vort} = \int_\Omega \|\nabla \times \mathbf{u}\|^2 \, dV,$$

which measure the kinetic energy and magnitude of vorticity. We omit density and constants from the measures for convenience. We compute these on the grid from $\tilde{v}_{ia}^{n+1}$ and discretize them as

$$E_{vel} = \frac{1}{N_G} \sum_{ia} (\tilde{v}_{ia}^{n+1})^2 \qquad E_{vort} = \frac{1}{N_C} \sum_{c\alpha\beta} \left(\left(\frac{\partial u_\alpha}{\partial x_\beta} \right)_c - \left(\frac{\partial u_\beta}{\partial x_\alpha} \right)_c \right)^2 \qquad N_C = \sum_c 1,$$

where N_C is the number of cells; the index c runs over all cells that have sufficient neighboring information to compute the vorticity measure. The indices α, β run over the spatial dimensions. We normalize the discretized measures so that they do not depend on the

resolution. The partial derivatives are approximated with central differences:

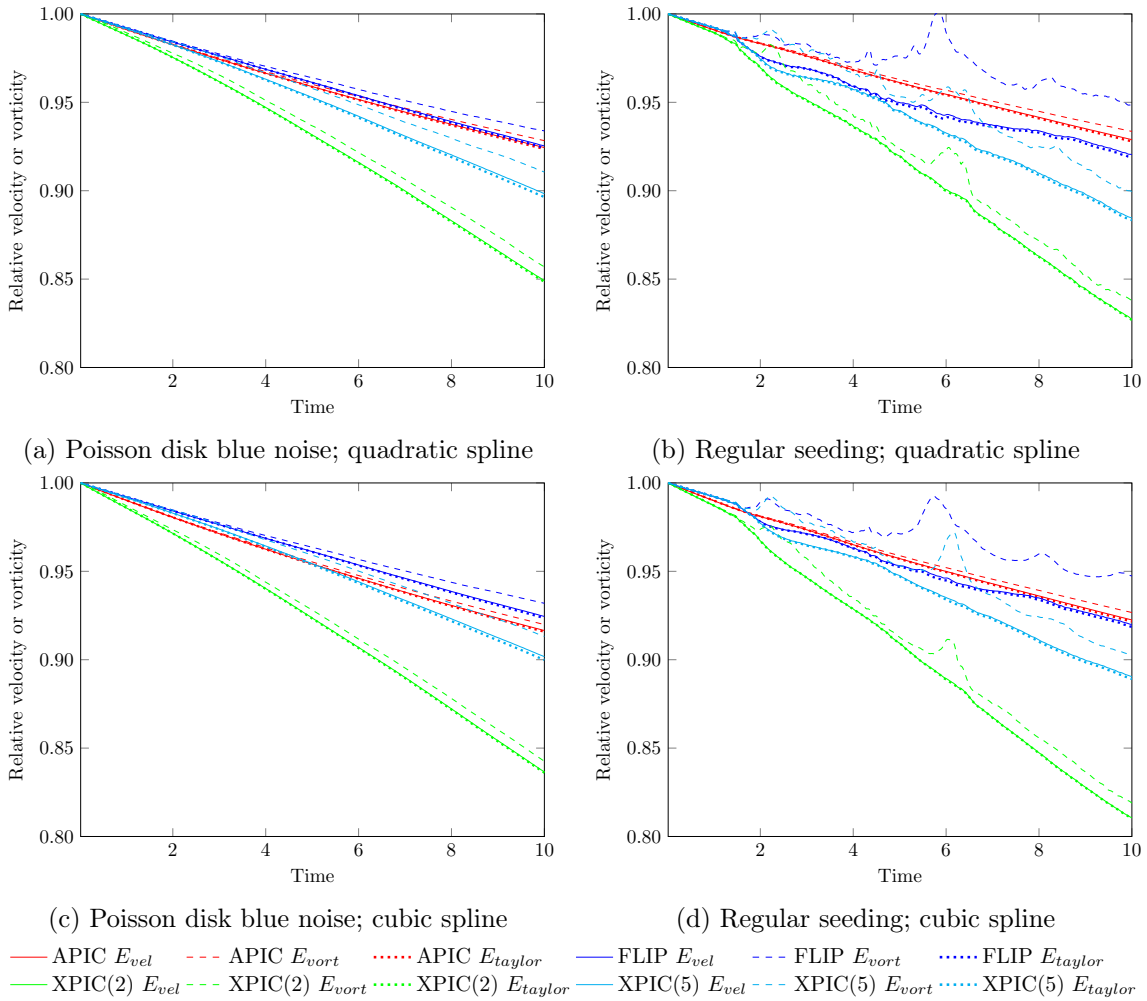
$$\left(\frac{\partial u_1}{\partial x_2}\right)_{(i,j)} = \frac{1}{2}\left(\frac{\tilde{v}^{n+1}_{i-\frac{1}{2},j+1} - \tilde{v}^{n+1}_{i-\frac{1}{2},j-1}}{2\Delta y} + \frac{\tilde{v}^{n+1}_{i+\frac{1}{2},j+1} - \tilde{v}^{n+1}_{i+\frac{1}{2},j-1}}{2\Delta y}\right)$$
$$\left(\frac{\partial u_2}{\partial x_1}\right)_{(i,j)} = \frac{1}{2}\left(\frac{\tilde{v}^{n+1}_{i+1,j-\frac{1}{2}} - \tilde{v}^{n+1}_{i-1,j-\frac{1}{2}}}{2\Delta x} + \frac{\tilde{v}^{n+1}_{i+1,j+\frac{1}{2}} - \tilde{v}^{n+1}_{i-1,j+\frac{1}{2}}}{2\Delta x}\right)$$

The same four-point central-differenced and central-averaged stencil is also used in 3D.

For this example, we begin with the basic setup from Section 3.5.1. For this section, we fix $\mu = 0$ and $N = 64$. We also use a later final time $T = 10$ to observe the longer-term behavior. In this test, we are interested in studying (a) dissipation of energy, (b) transfer of energy into incorrect velocity modes, (c) loss of vorticity, (d) the effects of particle seeding, and (e) the effects of spline choice (quadratic or cubic).

We test APIC, FLIP and XPIC (order 2 and 5) using Poisson disk seeding (4 particles per cell on average) and regular seeding (2×2 particles per grid cell). PIC is omitted from this test since it dissipates energy too rapidly to draw an interesting comparison. The results are shown in Figure 3.14. In the figure, the measures E_{vel}, E_{vort}, and E_{taylor} are normalized by their values after the first transfer from particles to grid. The measure E_{taylor} is like E_{vel}, except only contributions from the Taylor-Green Fourier modes are included.

There are a few interesting observations to be made from the results. FLIP transfers are not affected much by the choice of spline order, but it is sensitive to the particle distribution. APIC is relatively insensitive to the spline and seeding, though the higher-order spline and irregular seeding both increase dissipation very slightly. APIC and FLIP



(a) Poisson disk blue noise; quadratic spline

(b) Regular seeding; quadratic spline

(c) Poisson disk blue noise; cubic spline

(d) Regular seeding; cubic spline

Figure 3.14: Loss in energy (solid lines) and vorticity (dashed lines) for a Taylor-Green vortex over time using FLIP, APIC and XPIC transfers. The dotted lines show the amount of energy in the Fourier modes corresponding to the Taylor-Green vortex. All curves are normalized relative to the values obtained after transferring from the particles to the grid in the first time step.

have similar levels of dissipation. XPIC is also sensitive to the particle distribution as FLIP, and the XPIC transfer with lower order shows relatively larger dissipation.

Noting the analysis of the prior sections, we perform an FFT on the velocity field and report the magnitudes of the velocity modes as a colored image as we did for the transfer modes. In Figure 3.15, we look at the bleeding of the vortex into other Fourier

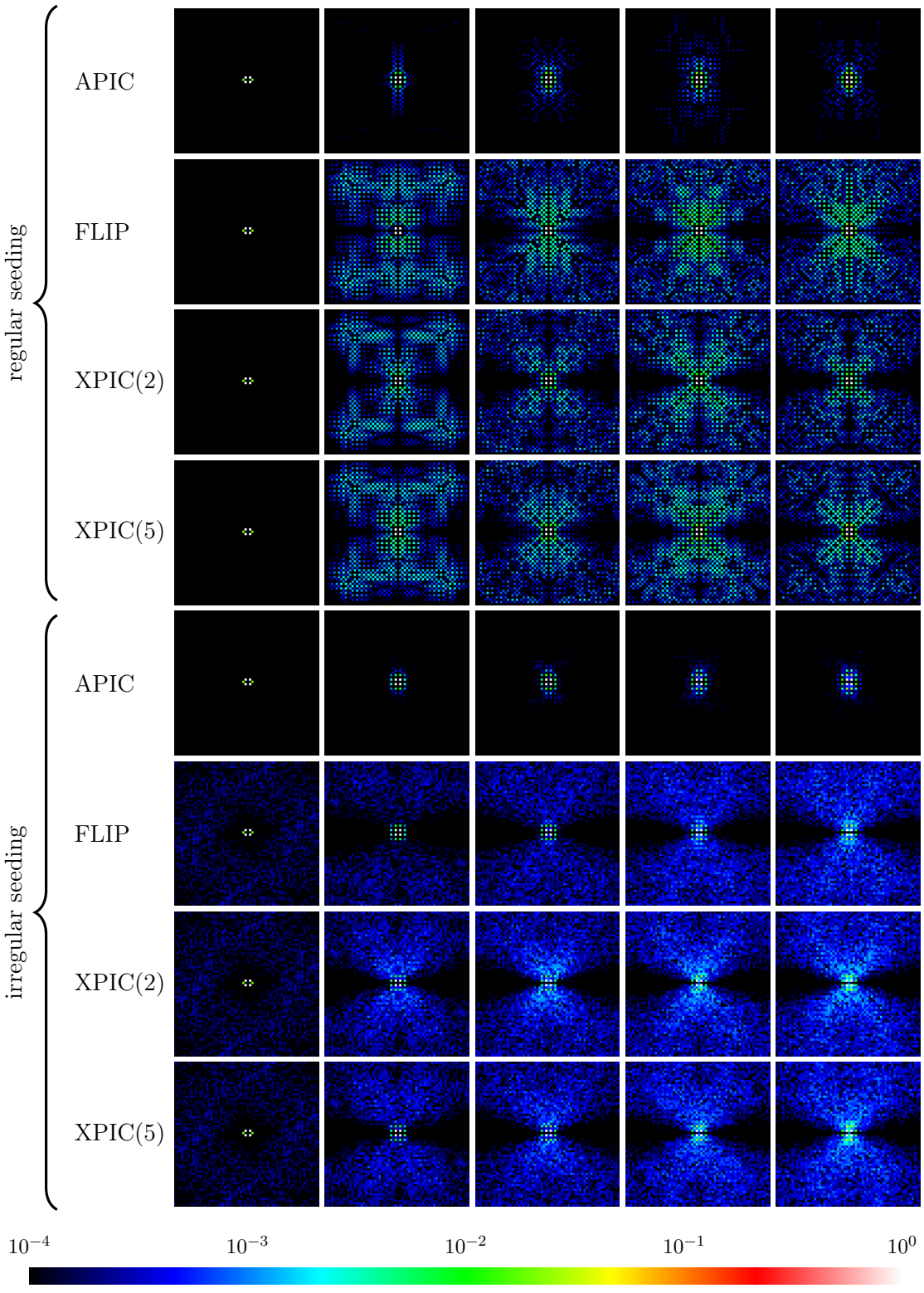


Figure 3.15: Energy leakage into other Fourier modes, at times $t = 0, 2, 4, 6, 8$, with 4 particles per cell. The first image is immediately after the initial particle to grid transfer.

modes. From this we can see that APIC is bleeding mostly into nearby low-frequency velocity modes (near the center of the image), wheres FLIP and XPIC transfer energy into higher frequency modes. The error visible after the very first transfer for FLIP and XPIC with irregular seeding is caused by trying to represent the velocity field on particles; if the APIC particles are initialized with $\mathbf{b}_{pa}^0 = \mathbf{0}$, the same errors are observed.

In Figure 3.15, significant portion of the velocity field can be found in incorrect high-frequency Fourier modes (modes on the order of a few percent). The quantity $E_{vel} - E_{taylor}$ for Figure 3.14 reflects the amount of kinetic energy that has been transferred to incorrect Fourier modes, and here the difference appears negligible. Because kinetic energy measures the squares of velocity, modest velocity errors (e.g., on the order of 5%) in modes that should be zero make only a very small difference in energy (around 0.25%). Since positions are updated using velocities (not their square), these errors are still significant.

We observe that regular seeding and irregular seeding produce noticeably different results on this test. Regular seeding introduces leakage that is several times higher than for irregular seeding. (Green pixels are observed well away from the middle of the image when regular seeding is used, indicating energy leakage into high-frequency Fourier modes. For irregular seeding, only shades of blue are observed away from the low-frequency modes in the middle.) A highly regular particle distribution appears to exacerbate this bleeding.

Inlet

In this test we use a $[0, 1]^d$ domain, where $d = 2$ for 2D and $d = 3$ for 3D. All boundaries are slip except some portions of the $y = 0$ boundary, where we place sources (fixed inflow velocity $v = 0.2$) and sinks (outflow, $p = 0$). The layouts are shown in

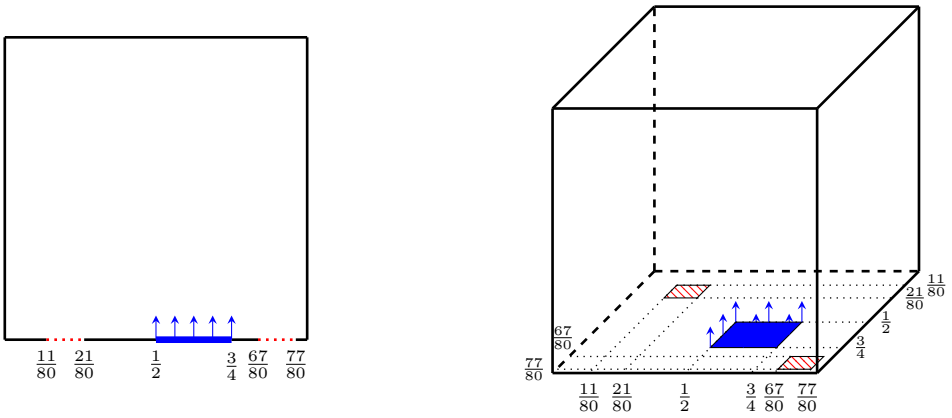


Figure 3.16: Inlet test configurations for 2D and 3D. The simulation domain is $[0,1]^d$ where $d = 2$ or $d = 3$. The red dotted or hatched areas are free surfaces, the blue areas are sources with a normal velocity $v = 0.2$, and all other boundaries are slip and have 0 normal velocity.

Figure 3.16. At time $T = 80\,s$ we turn off the source and sinks (enforcing slip boundary conditions everywhere) and observe how energy is dissipated until time $T = 200\,s$.

We compute kinetic energy for particles and grid. The kinetic for grid is discretized as

$$\mathcal{KE}^G = \sum_{ia} \frac{1}{2} m_{ia} (\tilde{v}_{ia}^{n+1})^2$$

where the indices run through all internal faces with non-zero mass. The kinetic energy for particles are discretized as

$$\mathcal{KE}^P = \sum_{p} \frac{1}{2} m_p \|\mathbf{v}_p^n\|^2 + \sum_{pa} \frac{1}{2} m_p (\mathbf{b}_{pa}^n \cdot (\mathbf{D}_{pa}^n)^{-1} \mathbf{b}_{pa}^n)$$

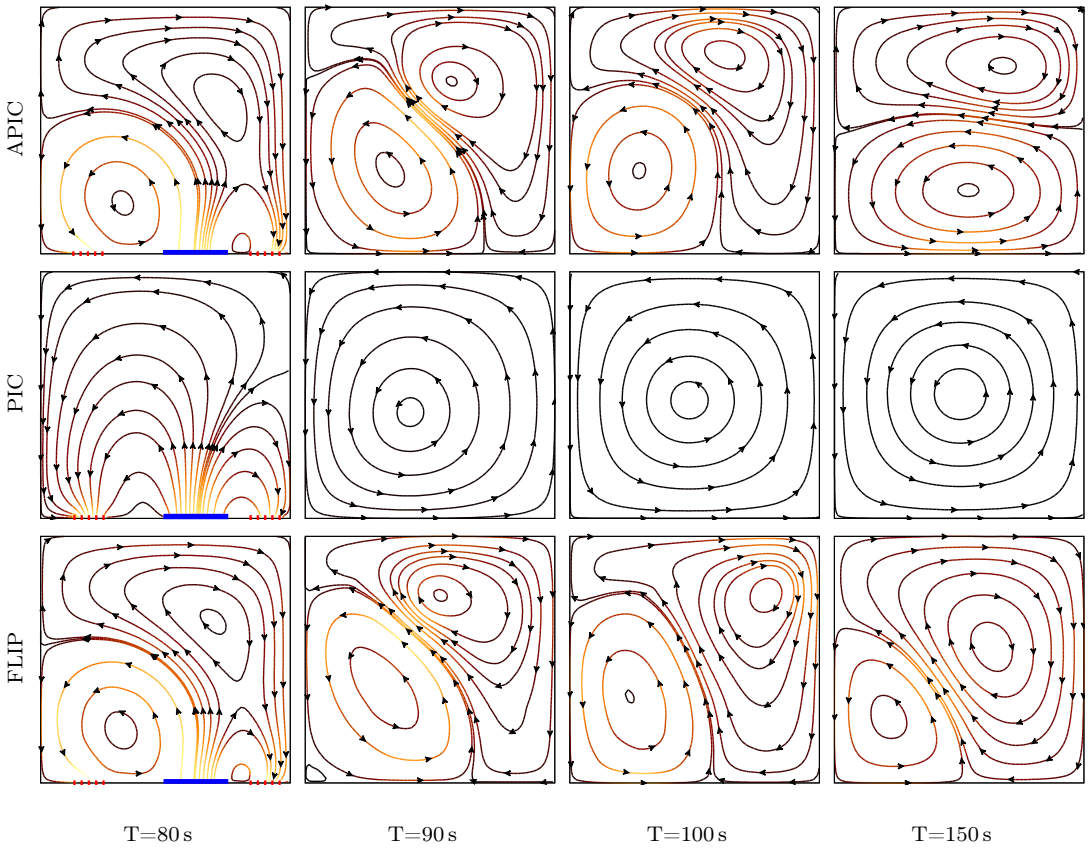


Figure 3.17: Snapshots of inlet tests. Streamline colors indicate fluid velocity magnitude, with black indicating slow fluid and yellow representing the fastest flow. The first column of frames ($T = 80\,s$) captures the last moment before we seal the boundary. The source and sinks are marked by solid blue and dotted red as same as Figure 3.16. After that vortices evolve without any input.

The extra contribution to kinetic energy is an estimate to the affine contributions; it is omitted for non-APIC transfers. The vorticity energy is computed as

$$\mathcal{V} = \sum_p \frac{1}{2} m_p \left\| \sum_{ia} \nabla w_{ip}^n \times v_{ia}^n \mathbf{e}_a \right\|^2 .$$

Snapshots of 2D simulations are shown in Figure 3.17. After the sources are turned off ($T > 80$), two vortices are formed in APIC and FLIP. In Figures 3.18 and 3.19 we plot

the vorticity and kinetic energy as a function of time. Observe that the particle and grid energy closely track each other in 2D for all versions as well as in 3D for PIC and APIC. In the 3D FLIP simulation, we can observe a significant difference between the grid and particle kinetic energy. The particle energy grows while the inlet is open even though the grid energy remains stable. Although grid velocities decay for all 3D simulations, the particle energy and particle-based vorticity does not decay to zero for the 3D FLIP simulation. Even when the particles come effectively to rest, the particles carry non-negligible velocities.

3.6 Conclusions

We have presented a new MAC-grid-based APIC transfer that preserves linear and angular momentum and also satisfies the original APIC properties. The full scheme is not conservative, since we perform the pressure solve using constant density as a compromise to avoid *boiling* artefacts.

We used 2D Fourier transforms to understand the numerical properties of the transfer. Compared with the 1D Fourier transform currently being used to analyze transforms, Fourier transforms in 2D give us some advantages. The first advantage is that we are able to analyze the Taylor-Green vortex, which gives us a way of studying dissipation of vortices. The second advantage is that it allows us to include pressure projection in the analysis, which extends the analysis meaningfully to include FLIP transfers. Studying in two dimensions also lets us studying the transfers' sensitivity to different particle distributions rather than merely irregularities in particle spacing.

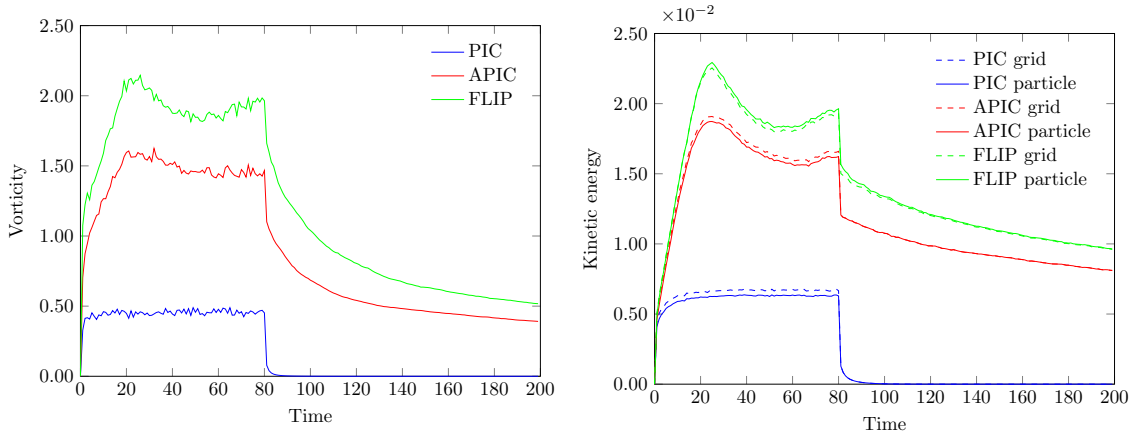


Figure 3.18: Vorticity and kinetic energy of 2D inlet test. Fluid is pumped through the domain until time 80 s, at which point the domain boundary is sealed and the fluid continues to circulate. For PIC, little energy is accumulated, and the circulation decays rapidly. For both FLIP and APIC, the circulation drops off quickly but then levels off. The grid and particle kinetic energy track each other closely for all three methods, which suggests that FLIP is quite stable on this example. FLIP retains more energy throughout the simulation.

Compared with direct computation of the eigenvalues of the transfer matrix, the 2D Fourier transform lets us efficiently compute and intuitively understand the eigenvalues of transfers. It arranges the eigenvalues by giving us a meaningful image rather than a long list of eigenvalues. From these images we can tell how a vortex of a spatial scale dissipates for example. Finally the Fourier transform in 2D provides us images for various transfers so we can compare them conveniently and visually.

In terms of dissipation, the comparison between APIC and PIC is not a surprise; PIC is very dissipative. The comparison with XPIC is instructive, as this is the first direct comparison between the two transfers as far as we are aware. The level of dissipation in APIC lies between XPIC with order 2 and 3. XPIC becomes less dissipative with higher order. In this spectrum, FLIP has zero dissipation. The opposite side of dissipation is noise, where modes survive on particles but should not. On this side, APIC and PIC are effectively

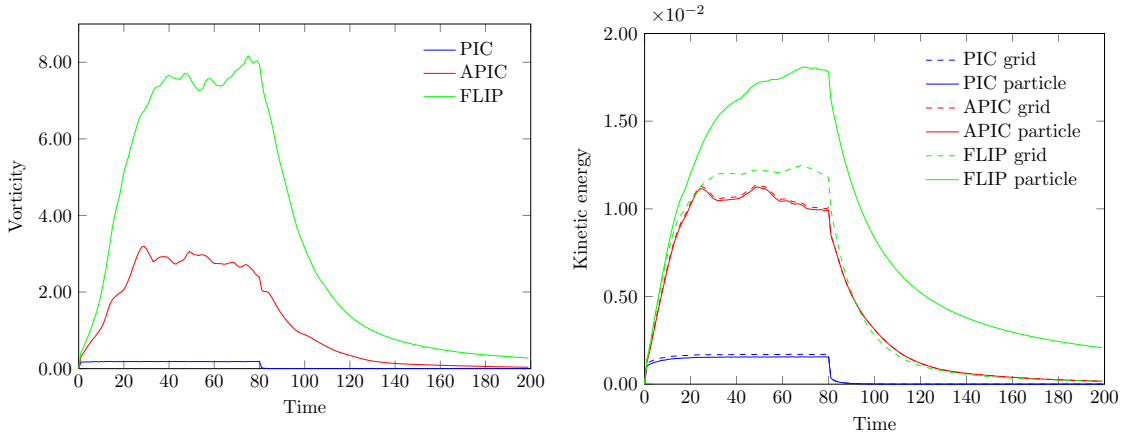


Figure 3.19: Vorticity and kinetic energy of 3D inlet test. Fluid is pumped through the domain until time 80 s, at which point the domain boundary is sealed and the fluid continues to circulate. The grid-based kinetic energy for all three methods decay to zero. For PIC and APIC, particle energy tracks the grid energy, and the particle-based vorticity decays to zero. The behavior of FLIP is very different. Particle energy is significantly greater than grid energy throughout the simulation. At its peak, about 1/3 of the particle energy does not transfer to the grid. During the pumped phase, energy accumulates on particles but not on the grid. As the grid energy decays away to zero, a significant amount of particle energy remains. The particle-based vorticity measure decays more than the energy, but it also stops short of zero. Since the vorticity measure is most sensitive to changes on the length scale of one grid cell, this suggests that the particles end with velocity modes whose wavelength is significantly less than the cell size.

perfect. XPIC is not too bad at lower orders, but it does tend to retain divergent velocity modes on particles over short time scales (See Figure 3.8). Like FLIP, it also tends to transfer low-frequency velocity modes into higher-frequency velocity modes (See Figure 3.15).

3.6.1 Limitations

The analysis presented in this paper provides intuition for how transfers behave; it does not fully characterize the transfers. The use of Fourier analysis limits the analysis to using a globally regular particle arrangement, which may skew the analysis. Truly irregular particle configurations may behave somewhat differently, and they may be more or less

dissipative than the tiled case. Nevertheless, the analysis presented provides useful insight into the methods involved.

The APIC transfers introduced are linear and angular momentum conserving, but the overall algorithm is not. This is because we found it necessary to use a constant-density pressure projection to avoid boiling artefacts in the simulations, where areas with thinner particle coverage appear less dense and rise under gravity. We postpone the problem of achieving full conservation of linear and angular momentum for future work. For related reasons, we postpone consideration of free surface flows for future work.

The desirable properties of the APIC transfers are tied to position update employed. As with co-located transfers, a generalized version of the transfers [94] may significantly broaden the range of positional updates over which good transfer properties may be obtained. In particular, a version of the transfers compatible with the XPIC position update (likely a MAC version of [94]) would be desirable.

The algorithm presented is not observed to work well with free surfaces. There are two reasons for this. The first is that our pressure projection is performed with constant density. Isolated particles command a larger volume on the grid than particles in the bulk, which causes problems when escaped particles land on the fluid surface. Particle-deficient pockets are created when fluid regions merge. This is a well-known problem with MPM, and some approaches have been proposed to address it [165, 59].

Chapter 4

Cached Gaussian elimination for simulating Stokes flow on domains with repetitive geometry

In this chapter, we still consider the fluid simulation, but this time we focus on the performance. The major part of the run time spent in a simulation is usually from linear solving, and the structure of the linear system dramatically affects the performance. With careful discretization, we find that the simulation domains with thin and long channels lead us to a sparse linear system with many duplicate block matrices, which in turn produce many reusable and independent computations if we solve the system by elimination. There will be an enormous gain of performance if we cache the reusable results and perform the independent computations in parallel. The discretization method, and also the linear solver are present in this chapter.

4.1 Introduction

Fluid simulation plays an important role in engineering. These applications vary greatly in the type of fluid being considered, the shape and size of the fluid domain, the number of phases, and in many other ways. This has lead to the development of a wide variety of methods that try to be as flexible and general as possible, maximizing their applicability to a wide range of applications. This flexibility comes at a cost, as such methods are unable to take advantage of specific application-specific properties. In this work, we develop a method for simulating fluids through channel-based microfluidic devices. The fluid domains for these devices generally consist of simple components connected by long, thin pipes. In this paper, we specifically develop a discretization algorithm that converts the Stokes flow problem into a linear algebra problem that is of a form that can be efficiently solved, and we propose an algorithm to solve these linear algebra problems very efficiently. The discretization is formally third order accurate in L^∞ for velocity and second order accurate in L^∞ for pressure. We demonstrate that the proposed algorithm achieves significant speedups on such problems at practical resolutions.

Existing methods for microfluidics simulation Numerical simulation of microfluidic devices presents few fundamental problems for existing methods, and software packages suitable for microfluidics applications are readily available. Indeed, numerical studies are typically carried out using an off-the-shelf software package such as OpenFOAM [84], COMSOL [117, 185], CFD-ACE+ [34], or Fluent [35] (See [177, 67] for an overview of existing tools). Although some of these software packages often have support specifically for mi-

crofluidic applications, the operate using general-purpose numerical methods and do not take advantage of the special properties of these devices. The computational cost of these methods has led to significant interest in application-specific numerical methods. A particularly popular model is the one-dimensional analysis model, which approximates the full fluid equations based on an analogy between flow of fluid through tubes and the flow of current through wires [73, 171, 68, 69]. See also [74] for a thorough introduction to these techniques. We take a different approach to obtaining faster simulation results. Rather than relying on properties of these devices to approximate the physics, we instead use these properties to accelerate the solution of the full fluid equations.

Properties of microfluidic devices Microfluidic devices are devices that operate on fluids on small (microliter or nanoliter) scales to perform a variety of tasks, such as common laboratory tests. Microfluidic chips are generally constructed by laying out components that perform specific operations on fluid volumes. Examples of common microfluidic operations are merging (combining different reagents together), mixing (forcing fluids through a serpentine flow to encourage the fluid to mix through molecular diffusion), delaying (holding fluid for a designated period of time to allow chemical reactions to complete), or forking (dividing a fluid flow among multiple directions for separate uses). These components are then connected with thin fluid channels to route fluid from one component to the next. A natural result of the way these devices are designed and constructed is that the geometry contains many duplicated copies of a relatively small number of distinct components. A relatively large fraction of the fluid domain consists of thin, straight (or occasionally circular) fluid channels. The global topology of the device is typically quite simple, usually planar

and sometimes even lacking loops. Although the proposed algorithm is a general-purpose algorithm for single-phase Stokes flow, it is specifically designed and optimized around the particular features of the geometry of the fluid domain. Though developed specifically for Stokes flow, it can be readily adapted to a variety of PDEs, including Navier-Stokes, the Poisson equation, and the heat equation.

4.1.1 Meshing strategies

In this paper, we discretize the Stoke equations using the finite element method with tetrahedral (triangular) elements. Constructing finite element meshes has been studied extensively. Most commonly used triangular/tetrahedral methods fall into one of three broad categories: mesh-based, Delaunay, and advancing front. *Mesh-based* methods begin by laying down a regular grid and using this grid to define a regular pattern of well-shaped elements in the interior; the region between the boundary and the regular portion is filled in some other way. Cartesian grids (or octree for local refinement) [183, 151] and BCC grids [126, 102] are both popular. *Delaunay-based* methods are based on the optimal triangle quality provided by the Delaunay triangulation (and tetrahedral analogue) [105, 172, 12, 63]. The Delaunay triangulation only defines a triangle mesh given vertices, which must be placed by another algorithm. Popular approaches for vertex placement are initial regular seeding or by point insertion [173, 152, 143, 22]. Additional mesh modifications may be required to force the Delaunay triangulation to conform to the boundary [63, 87, 95]. *Advancing front* methods construct elements along an advancing front, which is initially the boundary. New locations are selected (or existing vertices reused) based on element quality and lack of self intersections [114, 113, 112, 111]. Advancing front methods may also be combined with the

Delaunay criterion [120]. The problem of generating quadrilateral or hexahedral meshes has also received significant attention; since we do not generate such meshes, we instead refer the reader to an excellent review on the topic [137].

The effectiveness of the proposed algorithm relies on our meshes having special properties. The meshing of repeated components needs to be identical, and the mesh within pipes needs to be highly repetitive. We also require a few simple additional properties of the mesh. These are fairly unusual properties to request from a general-purpose meshing algorithm, and we are aware of no algorithms that satisfy them. For these reasons, we construct our own simple application-specific meshing algorithm. Our input geometry is assumed to be broken into components of known types. This allows us to naturally follow a decomposition/template-matching approach [164, 33, 133].

4.1.2 Existing sparse linear system solvers

The most significant contribution of the proposed method is the special structure of the linear algebra problem and our algorithm for solving it. The linear algebra problem is symmetric, indefinite, and sparse. Methods for solving these problems fall generally into direct and indirect methods.

Direct solvers

Direct methods for solving sparse linear systems of equations have been extensively studied [45]. These methods are mostly variations on Gaussian elimination and related factorizations: LU, Cholesky, LDL^T. Simply applying the corresponding direct method for dense systems to sparse systems tends to quickly result in large amounts of fill-in. Effective

direct solvers for sparse systems seek to strike a balance between reducing fill-in, utilizing available computational resources (SIMD, parallelism), and controlling memory usage. Our algorithm is a block-elimination algorithm that is designed around the specific properties of our fluid domains. It is designed to exploit commodity manycore hardware with significant SIMD processing resources. Our elimination algorithm is divided into four distinct stages and draws on ideas borrowed from a variety of other direct methods.

Elimination ordering Fill-in can be reduced by choosing a suitable elimination ordering [110], and many ordering strategies have been evaluated. Of these, two strategies are most relevant to our method. The first of these is the COLAMD algorithm [44], which is an approximation of the minimum degree ordering [62, 66]. Variations on the minimum degree ordering have been popular throughout the history of the development of sparse direct solvers, and we use the COLAMD ordering in the final stage of our elimination algorithm. Nested dissection [61, 178, 108] has also received significant attention. Nested dissection is a recursive divide-and-conquer strategy where the domain is first divided in half by inserting a *separators*; this divides the domain into two independent problems, which may be solved in parallel. In a final step, the separators are eliminated. Separators play a similar role in our algorithm, where we use them for isolation, to expose parallelism, and to expose redundancy. Unlike with more general problems, where eliminating the final separator is often the most expensive step in the entire algorithm, our special domain-specific geometry means that separators are generally very small and can be eliminated relatively efficiently.

Multifrontal methods Permuting the rows and columns of the matrix before performing factorization suffices to reduce fill-in, but the straightforward algorithm is not able to effectively utilize SIMD performance. This lead to the development of frontal methods [91, 86, 109], which perform the elimination steps on a dense *frontal matrix*, which allows dense linear algebra (and efficient BLAS routines) to be used. These methods were replaced with multifrontal methods, which are based on the observation that elimination dependencies take the form of an *elimination tree*, and a new independent elimination front can be started from each leaf of the tree [52]. Since these fronts are independent, they may be eliminated in parallel [50, 51, 7, 5]. A process of amalgamation (also called supernodes) is used to eliminate multiple rows with similar sparsity patterns at the same time to exploit more efficient level-3 BLAS operations [52, 10, 36]. Publicly available libraries implementing the multifrontal method are readily available, including MUMPS [5, 6, 4] and UMFPACK [43, 41, 42]. We compare the performance of the proposed method against both libraries in Section 4.6.6.

Cyclic reduction The proposed algorithm utilizes the idea of cyclic reduction, a variation on Gaussian elimination for tridiagonal systems where all odd rows are eliminated in parallel to expose opportunities for parallelism [85, 32]. This reduces the problem size by approximately half; it produces another tridiagonal system so the process can be repeated. As a serial algorithm, cyclic reduction requires about 2.7 times as many operations as the usual Gaussian elimination [89, 9]. The benefit of this method is that it exposes large numbers of operations that can be performed efficiently in parallel on a variety of architectures [70], especially on GPUs [184]. Cyclic reduction may also be formulated as a divide-and-

conquer algorithm, with separate subproblems for even and odd variables [58]. Although our domains may have complex topology and do not lead to tridiagonal systems, many of the decisions that we make during discretization are designed to produce a tridiagonal block structure over significant portions of the matrix. This allows us to take advantage of cyclic reduction during a portion of our elimination phase.

Domain decomposition Domain decomposition methods divide the domain into regions which can be inverted in parallel. This reduces the problem to a smaller Schur compliment system, which contains only degrees of freedom along the interfaces between domains [141, 142]. The independent domains expose ample opportunities for parallelism [145, 55, 2]. Due to the size and density of the Schur compliment, it is common to invert this system using an iterative solver like PCG [98]. (As noted below, domain decomposition is also a popular *preconditioner* for PCG.) The mortar method divides the domain into separate regions, which are stitched together with Lagrange multipliers; as with other domain decomposition methods, the independent domains may be inverted independently [119, 176].

Relation to the proposed method Although our method is neither a multifrontal method nor cyclic reduction, it has many similarities to these methods. Our block-elimination may be considered as an amalgamation strategy to increase opportunities for level 3 BLAS use. We plan out our computations during a planning stage, and we also make critical use of the ability to begin elimination from many blocks in parallel. Since our blocks are large enough to make effective use of vector resources, we do not assemble fronts in the proposed method. This effectively breaks up large frontal calculations into similarly-sized pieces as

was done in [6]. As in cyclic reduction, we have a tridiagonal block structure for significant portions of our matrix, and we eliminate them using a recursive even-odd strategy to maximize parallelism and (when possible) caching opportunities.

4.1.3 Iterative methods

Some of the most efficient algorithms known for solving large sparse linear systems are iterative. Of these, the Krylov methods are perhaps the most popular with the conjugate gradient (CG) algorithm being the earliest, best known, and most understood [83, 153]. CG assumes that the matrix is sparse and symmetric positive definite, but other Krylov schemes such as MINRES [138] or GMRES [148] may be used instead when the system is symmetric but indefinite. The convergence of Krylov methods depends on the conditioning of the system [96, 168], and a preconditioner is often required for rapid convergence [71]. One important class of efficient preconditioners is based on domain decomposition [156, 48], which splits the domain into subdomains. The smaller (and cheaper) sub-problems provide rapid local convergence, and a coarsened problem is solved to improve global convergence. The most efficient preconditioners, however, are multigrid methods, which also use a coarsened problem to improve low-frequency convergence but use a smoother instead for high-frequency convergence; this coarsening process is repeated in a hierarchy for optimal $O(n)$ convergence.

Multigrid methods have become the standard for efficient large-scale preconditioners, especially for the Poisson equation [139, 81, 174], though they can also be applied to the Stokes [175], Navier-Stokes [124, 65, 123], Euler equations [122], and fluid-structure interaction [118]. Multigrid parallelizes well and is well-suited to GPU implementation [21, 77] and heterogeneous environments [107]. Although multigrid is asymptotically optimal for large

problems (even scaling to billions of degrees of freedom [101]), it is generally not the most efficient choice at medium or low resolutions, especially in domains with the small features typical of microfluidics designs. This work thus fills two important roles. (1) *The proposed algorithm allows the Stokes equations to be solved more efficiently on microfluidics problems at small and medium resolutions.* (2) At high resolution, multigrid methods require a separate solver to solve the system at the coarsest resolution; *the proposed algorithm is ideally suited for this purpose.*

Contributions and novelty In this paper, we make the following novel contributions.

- We propose a special solver for sparse symmetric indefinite systems of linear equations that have many repeated matrix blocks. This solver uses a combination of caching, cyclic reduction, and general sparse solver techniques to solve these linear systems very rapidly.

- We propose a discretization algorithm for the Stokes equations that produces linear systems in a form suitable for our new rapid solver. The discretization is formally third order accurate in velocity and second order accurate in pressure in the L^∞ norm.

- We evaluate the algorithm across resolutions, cores, and with existing solvers. The full Stokes algorithm is significantly faster than existing solvers at medium resolutions (around 1M degrees of freedom) on the types of geometry that typically occur in designs for microfluidic devices. The algorithm scales well to many cores.

4.2 Overview of algorithm

4.2.1 Stokes equation

Microfluidic devices operate at small length scales (feature width $< 0.1\,mm$) on small volumes of fluid ($< 1\,\mu L$) traveling at slow speeds ($< 1\,cm\,s^{-1}$). At these scales, the Reynolds number is low ($\ll 1$), and Stokes flow becomes a good approximation for the fluid flow (though not always [46]). Within the Stokes regime, the dynamics are dominated by incompressibility and a balance of viscous and pressure forces. The momentum equations reduce to

$$\nabla \cdot \boldsymbol{\sigma} = 0, \qquad \nabla \cdot \mathbf{u} = 0, \qquad \boldsymbol{\sigma} = \mu(\nabla \mathbf{u} + \nabla \mathbf{u}^T) - p\mathbf{I},$$

where $\boldsymbol{\sigma}$ is the fluid stress, $\mathbf{u}$ is the fluid velocity, μ is the dynamic viscosity, and p is the pressure. We consider a mixture of velocity ($\mathbf{u} = \mathbf{a}$) or traction ($\boldsymbol{\sigma}\mathbf{n} = \mathbf{b}$) boundary conditions, where $\mathbf{n}$ is the normal direction. The resulting discretization is a symmetric and indefinite sparse linear system of equations.

Although we will limit our discussion and discretization to the Stokes equations, most of the ideas are not specific to the Stokes equations. In particular, the proposed algorithm can be readily adapted to solve the Poisson equation, heat equation, and Navier-Stokes equations.

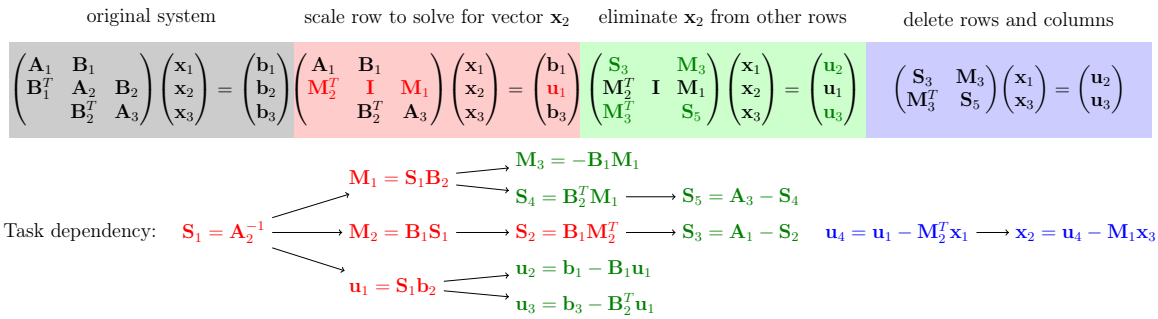


Figure 4.1: Any row of a system can be eliminated, provided the diagonal block can be inverted. Elimination preserves system symmetry; the matrices $\mathbf{A}_i$ and $\mathbf{S}_i$ are symmetric. Each step of the elimination process requires a primitive linear algebra operation, which may be considered as a task. Even on this very small example, opportunities for computing tasks in parallel emerge rapidly.

4.2.2 Properties of microfluidic devices

Although fluid domains may be very irregular and complex, this is not the case in some important domains. For example the plumbing in a typical building is composed almost entirely from pipes with standardized diameters, which meet at a relatively small number of standardized junctions (tee, elbow, cross, reducer, plug, valve, etc.). The advantages of designing the plumbing in buildings in this way are obvious; the standardized parts can be cheaply mass produced and are readily available in hardware stores. As long as these pipes and standardized junctions are discretized in exactly the same way each time they occur, they will produce identical matrix blocks in the final system.

Although microfluidic devices are fabricated entirely differently (typically by a process like CNC milling), in practice the designs of these devices tend to closely resemble plumbing. These designs are dominated by standardized components (joints, mixers, delays) connected by straight (or less commonly circular) fixed-width channels. A typical chip is designed by first determining which components are required to perform the desired fluidic

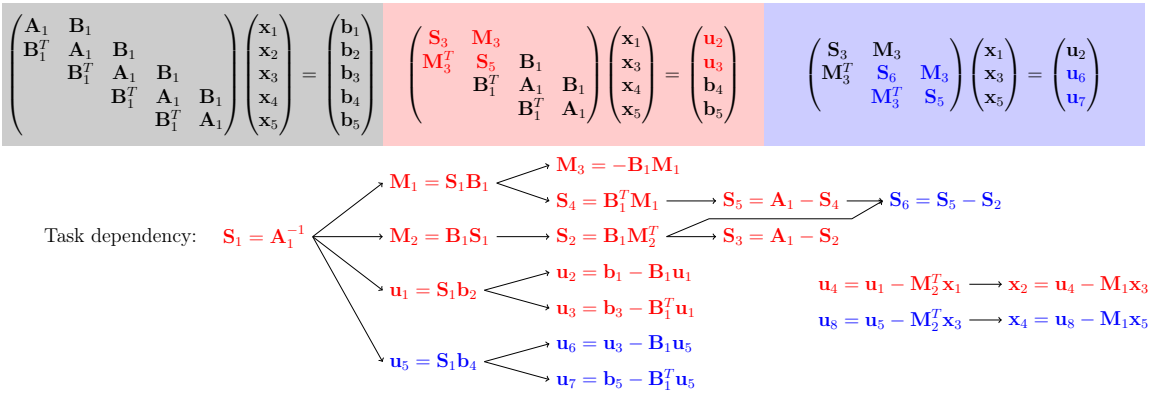


Figure 4.2: Eliminating similar but independent rows benefits greatly from caching. In this case, the first elimination (red) generates 13 tasks, of which 6 are $O(n^3)$ matrix operations. The second elimination (blue) generates only 6 tasks, all of which are much cheaper $O(n^2)$ operations. Additional parallelism is introduced, including in the backsolve phase.

operations (combine two input fluids, mix them together thoroughly, let them react for a specified amount of time, etc.). Then, the components are connected by channels to route fluids from component to component in the proper sequence. The result is that, as with the plumbing example, one may discretize the fluid domain so that the final matrix contains many identical matrix blocks.

4.2.3 Elimination

Gaussian elimination classically precedes by eliminating rows from a matrix one by one in a serial algorithm. One may begin by eliminating any row or block of rows (ignoring stability concerns) as shown in Figure 4.1. This effectively modifies neighboring rows of the matrix based on the sparsity pattern. The eliminated row may be removed from the system, though its entries will be required during the backsolve phase. This is equivalent to forming the Schur complement. If the original matrix is symmetric and the diagonal block is chosen as the pivot, the new matrix will also be symmetric.

Observe that only the row being eliminated and its neighboring rows (based on the matrix sparsity pattern) are modified; rows that are not neighbors can thus be eliminated independently and in parallel. These are key observation that underlie the success of multifrontal methods [52]. If the two independent rows being eliminated have identical matrix blocks, many of the calculations required to eliminate one of the rows can be reused when eliminating the other row (See Figure 4.2).

These observations suggest that significant performance improvements may be possible if one is able to create duplicated matrix blocks in the system matrix. Under general circumstances of irregular problem domains and irregular meshing, one would not expect duplicated matrix blocks to occur. The ability to benefit from caching relies on repeated geometry and discretization that takes advantage of it. As noted earlier, the geometry of microfluidics devices tends to be redundant; we just need to be careful to discretize these redundancies consistently.

4.2.4 Pipes

Long and thin channels (which we will generally refer to as pipes) are common in microfluidics devices; indeed, a significant fraction of the fluid domain may consist of pipes. Pipes are special for our purposes because they are very efficient to eliminate. Consider a long thin pipe, which is broken up into fixed-width slices. Each slice has identical geometry and is discretized identically. The resulting system matrix will be block tridiagonal. All of the blocks along the diagonal are identical, and all of the off-diagonal blocks are identical (up to transpose). The matrix follows the same pattern as in Figure 4.2.

Observe that all odd rows may be eliminated independently for nearly the same cost as eliminating just one of the rows. The only calculations that cannot be reused are the (much less expensive) vector operations that occur as part of the forward and backward triangular solves. Further, most of the matrices that are left behind after eliminating all of the odd rows are again identical (they all follow the $(\mathbf{M}_3^T, \mathbf{S}_6, \mathbf{M}_3)$ pattern observed in the middle row at end end of Figure 4.2). Thus, the process can be repeated. This recursive even-odd elimination pattern is just cyclic reduction [58]. Ignoring vector operations, each recursive step requires a constant number of matrix operations. Since the number of recursive steps is logarithmic in the number of slices in the pipe, long pipes can be eliminated very efficiently. Moreover, caching is possible between pipes even when they have different lengths, as long as the pipe diameters and slice widths are the same. In practice, there are additional complications relating to scaling and orientation; these will be addressed in Section 4.3.9.

4.2.5 Cross sections as blocks

The process of discretizing our geometry into our linear system begins with a geometric definition of a block. These blocks divide the fluid domain into small regions whose discretizations will eventually become matrix blocks. Blocks should be redundant where possible to facilitate the formation of repeated matrix blocks; the choice of blocks will have significant performance implications.

We have seen that tridiagonal block matrix structure can be eliminated very efficiently and without fill-in using cyclic reduction. This suggests that the geometry should be sliced into cross sections that have only two neighboring cross sections, as we do for the pipe. This definition works for geometry that is topologically a pipe. For more irregular

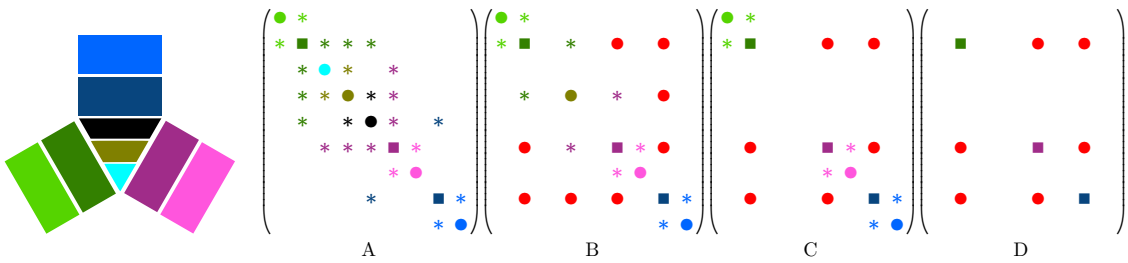


Figure 4.3: Eliminating components will not fill-in past separators. On the left we show an example domain, which consists of four components (three arms and one joint) and is split into blocks marked by different colors. In (A) we show its corresponding system, where block matrices for separators are marked by squares (■ ■ ■), and other block matrices on the diagonal are marked by circles (● ● ● ● ● ●). Non-zero off-diagonal block matrices for connections are marked by *, colored by the connection's parent block. In (B) we eliminate the bottom and top block (● ●) inside the joint. This introduces fill-ins (red circles ●) but they are all confined in the separators. In (C), we eliminate the remaining middle block (●) inside the joint. Finally we eliminate all non-separators (● ● ●) to reach a system in (D). Note that before (D), eliminating a component does not fill-in any other components.

geometry like a tee junction, some blocks must have more than two neighbors, and some degree of fill-in is unavoidable. Instead, we seek to limit the propagation of fill-in through the matrix. We do this by inserting *separators* around irregular components. We eliminate the separators after all other blocks, effectively dividing the system into isolated matrices. Fill-in from any component is localized to the component itself and the separator blocks that bound it (See Figure 4.3). We can then define a (non-separator) block to be a cross section of geometry that has at most two neighboring non-separator blocks. In this way, we can use cyclic reduction to efficiently eliminate the blocks within components, which comprise the significant majority of blocks. At this point, only a relatively small number of separator blocks remain. They are eliminated last; fill-in during this stage may be significant, but it is limited by both the small number of blocks involved and the planar connectivity typically found in microfluidic devices.

4.2.6 Reusable component

Separators isolate components from each other, allowing them to be discretized independently. Duplicated components need only be divided into blocks and discretized once. In addition to saving time and space, this also ensures that duplicated components lead to duplicated blocks and duplicated block matrices. Reuse of computations occurs at the level of blocks, not components per se. For example, pipes of different lengths should be divided into blocks that are the same width so that calculations may be reused.

Transforms can change the block matrices of a component, preventing immediate reuse. We can nevertheless reuse components by discretizing them in a canonical coordinate system and then assembling matrix blocks in the local coordinate system. This can be accomplished through row an column scaling on the final system, as we show in Section 4.3.9.

4.2.7 Algorithm steps

We close this overview with the algorithmic tasks that must be completed for the proposed algorithm along with forward references to the discussion of each step.

1. Identify canonical components 4.3.3
2. Construct geometry blocks and identify canonical blocks 4.3.4
3. Construct canonical block meshes 4.3.5
4. Assign degrees of freedom to canonical blocks 4.3.6
5. Assemble canonical matrices 4.3.6
6. Assign global degrees of freedom 4.3.7

7. Assemble the system matrix blocks4.3.8

8. Transform the right hand side 4.3.9

9. Plan block elimination 4.4.1

10. Execute jobs 4.4.2

11. Transform the solution 4.3.9

4.3 Discretization

We are interested in discretizing the Stokes equations over thin and repetitive geometry. We adopt a standard finite element treatment and basis pair for the Stokes equations, which we summarize here for completeness.

4.3.1 Finite element formulation

Our finite element discretization follows [11, 150]. We start directly with $\nabla \cdot \boldsymbol{\sigma} + \mathbf{f} = \mathbf{0}$, where $\boldsymbol{\sigma} = \mu(\nabla \mathbf{u} + \nabla \mathbf{u}^T) - p\mathbf{I}$, rather than simplifying to $\mu \nabla^2 \mathbf{u} + \mathbf{f} = \nabla p$ using the incompressibility condition. While this reduces the sparseness of our system, it simplifies the treatment of traction boundary conditions $\boldsymbol{\sigma}\mathbf{n} = \mathbf{b}$. We assume a single fluid phase, so that μ is constant.

Let $\mathbf{w}$ be a test function chosen from the same function space as velocity $\mathbf{u}$. Then, the weak form of the momentum equation may be written as

$$\begin{aligned} 0 = \int_\Omega \mathbf{w} \cdot (\nabla \cdot \boldsymbol{\sigma} + \mathbf{f})\, dV &= \int_\Omega \nabla \cdot (\mathbf{w} \cdot \boldsymbol{\sigma}) - \nabla \mathbf{w} : \boldsymbol{\sigma} + \mathbf{w} \cdot \mathbf{f}\, dV \\ &= \int_{\partial\Omega} \mathbf{w} \cdot \boldsymbol{\sigma}\mathbf{n}\, dA - \int_\Omega \nabla \mathbf{w} : \boldsymbol{\sigma}\, dV + \int_\Omega \mathbf{w} \cdot \mathbf{f}\, dV \end{aligned}$$

$$\begin{aligned} -\int_{\partial\Omega} \mathbf{w} \cdot \boldsymbol{\sigma}\mathbf{n}\, dA + \int_\Omega \nabla \mathbf{w} : \boldsymbol{\sigma}\, dV &= \int_\Omega \mathbf{w} \cdot \mathbf{f}\, dV \\ -\int_{\partial\Omega} \mathbf{w} \cdot \boldsymbol{\sigma}\mathbf{n}\, dA + \int_\Omega \nabla \mathbf{w} : \left(\mu(\nabla \mathbf{u} + \nabla \mathbf{u}^T) - p\mathbf{I}\right) dV &= \int_\Omega \mathbf{w} \cdot \mathbf{f}\, dV, \end{aligned}$$

where $\mathbf{f}$ is the external force. Letting N_i and P_i be bases for velocity and pressure,

$$\mathbf{u} = \sum_i N_i \mathbf{u}_i \qquad \mathbf{w} = \sum_i N_i \mathbf{w}_i \qquad p = \sum_i P_i p_i \qquad \phi = \sum_i P_i q_i.$$

Then the integrals can be written as

$$\int_\Omega \mu \nabla \mathbf{w} : (\nabla \mathbf{u} + \nabla \mathbf{u}^T)\, dV = \sum_{ij} \mathbf{w}_i^T \mathbf{u}_j \underbrace{\mu \int_\Omega \left(\frac{\partial N_i}{\partial \mathbf{x}}\right)^T \frac{N_j}{\partial \mathbf{x}}\, dV}_{\mathrm{tr}(\mathbf{D}_{ij})} \tag{4.1}$$

$$+ \sum_{ij} \mathbf{w}_i^T \underbrace{\left(\mu \int_\Omega \frac{\partial N_i}{\partial \mathbf{x}} \left(\frac{\partial N_j}{\partial \mathbf{x}}\right)^T dV\right)}_{\mathbf{D}_{ij}} \mathbf{u}_j \tag{4.2}$$

$$= \sum_{ij} \mathbf{w}_i^T \underbrace{(\mathrm{tr}(\mathbf{D}_{ij})\mathbf{I} + \mathbf{D}_{ij})}_{\mathbf{A}_{ij}} \mathbf{u}_j = \sum_{ij} \mathbf{w}_i^T \mathbf{A}_{ij} \mathbf{u}_j \tag{4.3}$$

$$\int_\Omega \nabla \mathbf{w} : p\mathbf{I}\, dV = \int_\Omega \nabla \cdot \mathbf{w} p\, dV = \sum_{ij} \mathbf{w}_i^T p_j \underbrace{\int_\Omega \frac{\partial N_i}{\partial \mathbf{x}} P_j\, dV}_{-\mathbf{g}_{ij}} = -\sum_{ij} \mathbf{w}_i^T \mathbf{g}_{ij} p_j \tag{4.4}$$

$$\int_\Omega \mathbf{w}\cdot\mathbf{f}\,dV = \sum_i \mathbf{w}_i^T\mathbf{f}_j \underbrace{\int_\Omega N_iN_j\,dV}_{k_{ij}} \tag{4.5}$$

$$\int_{\partial\Omega} \mathbf{w}\cdot\boldsymbol{\sigma}\mathbf{n}\,dA = \int_{\partial\Omega_n} \mathbf{w}\cdot\mathbf{b}\,dA = \sum_i \mathbf{w}_i^T\mathbf{b}_j \underbrace{\int_{\partial\Omega_n} N_iN_j\,dA}_{m_{ij}}. \tag{4.6}$$

For the incompressibility equation, we begin with $\nabla\cdot\mathbf{u} = -l$, where l is a source term that we include to simplify analytic testing. Physically, $l = 0$, but we allow $l \neq 0$ to simplify testing (See Section 4.6.2). Let ϕ be a test function chosen from the same function spaces as pressure p. Then, the weak form of the incompressibility equation is just

$$-\int_\Omega \phi\nabla\cdot\mathbf{u}\,dV = \int_\Omega \phi l\,dV.$$

Substituting in our basis produces the integrals

$$\int_\Omega \phi\nabla\cdot\mathbf{u}\,dV = \sum_{ij} q_i \underbrace{\left(\int_\Omega P_i\frac{\partial N_j}{\partial\mathbf{x}}\,dV\right)}_{-\mathbf{g}_{ji}^T}\mathbf{u}_j$$

$$\int_\Omega \phi l\,dV = \sum_i q_i \underbrace{\int_\Omega P_i l\,dV}_{c_i}.$$

Let $\mathbf{A}$, $\mathbf{G}$, $\mathbf{K}$, and $\mathbf{M}$ denote block matrices whose blocks are given by $\mathbf{A}_{ij}$, $\mathbf{g}_{ij}$, k_{ij}, and m_{ij}. Similarly, let $\mathbf{u}$, $\mathbf{p}$, $\mathbf{b}$, $\mathbf{c}$, and $\mathbf{f}$ be block vectors whose blocks are given by $\mathbf{u}_i$, p_i, $\mathbf{b}_i$, c_i, and $\mathbf{f}_i$. Then, we can express the full system as

$$\begin{pmatrix} \mathbf{A} & \mathbf{G} \\ \mathbf{G}^T & \mathbf{0} \end{pmatrix}\begin{pmatrix} \mathbf{u} \\ \mathbf{p} \end{pmatrix} = \begin{pmatrix} \mathbf{Mb}+\mathbf{Kf}+\mathbf{d} \\ \mathbf{c} \end{pmatrix}, \tag{4.7}$$

Figure 4.4: $(\mathcal{P}_2, \mathcal{P}_1)$ Taylor-Hood elements for 2D and 3D. The filled circles (•) are velocity degrees of freedom and hollowed circles (∘) are pressure degrees of freedom.

where $\mathbf{d}$ is a vector of velocity boundary conditions. This vector is obtained by eliminating velocity dofs associated with velocity boundary conditions from the system and moving them to the right hand side.

4.3.2 Taylor-Hood element

The choice of basis functions N_i and P_i play an important role in the numerical stability of a discretization; the Stokes equations require a stable finite element pair to avoid serious numerical problems [19]. We adopt the $(\mathcal{P}_2, \mathcal{P}_1)$ Taylor-Hood element (See Figure 4.4), which is known to be a stable pair for the Stokes equations [54]. This choice is not essential, and other elements may be preferred [20]. We have found this choice to provide a favorable tradeoff between discretization accuracy, implementation complexity, and numerical stability.

4.3.3 Component construction

For the purposes of this work, we assume that the input geometry is already broken into labeled components. That is, we know what portions of the fluid domain are pipes, joints of various connectivities (bends, tees, crosses), mixers, etc. This design decision greatly

simplifies the implementation of the algorithm, and it reflects the way in which these devices are constructed in practice.

The first step in our discretization process is to transform each component into a canonical coordinate system (with respect to translations, rotations, and optionally scale). We refer to these as *canonical components*. This facilitates the identification of reused components; components are equivalent when their canonical components are the same. Only unique canonical components are represented. Components are represented as a pointer to a canonical component, a transformation, and connectivity information. The transformations will be used for assembling matrix blocks and transforming the right hand side and solution vectors, as described in Section 4.3.9. Only canonical components are passed forward to the later stages of the discretization process (block construction, meshing, and integration). Connectivity information will be used to assemble the final matrix blocks and global system.

The connectivity between components is important for matrix construction, since connections correspond to off-diagonal matrix blocks in the final system. We do this in terms of *connections*. Components have *sockets*, which are places along their boundary where they connect to neighboring components. A pipe has a socket at each end; a tee-junction has three sockets. Connections have a well-defined *cross-sectional shape*, which may differ from connection to connection; these are also canonicalized. Connections consist of (a) the two components that are being connected, (b) which socket of each component is involved, and (c) the canonicalized cross section shape.

We will use the canonical cross section shapes later to ensure consistent mesh discretization between components an blocks. In our implementation, we assume rectangular

cross sections between components with fixed depth but potentially different widths. This is consistent with how many microfluidic devices are manufactured, but other cross section shapes may be more appropriate in other contexts (e.g., circular for plumbing). The algorithm is not sensitive to the shapes of cross sections; our use of rectangular cross sections is purely for convenience.

4.3.4 Block construction

The block construction phase has three primary goals: (a) divide canonical components into *geometry blocks*, (b) identify duplicated geometry blocks, and (c) construct a block-level connectivity graph from the component-level connectivity graph. Geometry blocks are small geometric regions of the fluid domain that will be discretized and will correspond to block matrices in the final global system. Geometry blocks are the level of granularity at which meshing and finite element integration are performed. Ideally, the domain will be divided in to large numbers of small blocks, most of which are identical and have few neighbors.

In our implementation, we used simple rules to divide components into blocks. Pipes are divided into cross sections (geometry blocks) of a fixed characteristic width h (the triangle edge length). Since pipes may have any length, the last geometry block in a pipe may have an irregular, which we limit to the range $[\frac{h}{2}, \frac{3h}{2}]$. This simple strategy ensures that all but one geometry block within each pipe will be identical, and these blocks will also be identical to the geometry blocks of other pipes with the same cross section. We divide irregular canonical components into strips of width approximately equal to h; strips may run parallel or perpendicular to the pipe direction. Geometry blocks are constructed

in a canonical frame to identify duplicates. As with canonical components, we perform all per-block operations on these canonical blocks. Each physical block stores a transform and a pointer to its canonical block.

Once canonical components and divide into geometry blocks, we must update our connectivity graph. Nodes of the graph are blocks, which store a transform and point to a canonical block. Edges of the graph represent connections between blocks. These connections store the same information as their component-wise counterparts: the blocks being connected, the socket of each block being connected, and the canonical cross section. Note that connections may involve many blocks.

The interiors of geometry blocks are disjoint (they do not overlap). When the boundaries of two geometry blocks intersect, we call the blocks neighbors. Based on this, we divide geometry blocks into three types: *regular blocks*, *irregular blocks*, and *separator blocks*. Separator blocks occur at the boundaries of components; one of blocks adjacent to each connection is designated as a separator block. In practice, one of these components will be a pipe (or at least pipe-like); we designate the outermost blocks of these pipes as the separator blocks. As many of the remaining blocks are classified as regular as possible, subject to the rule that regular blocks may have at most two neighboring regular blocks. The remaining blocks are classified as irregular blocks. We illustrate different types of geometry blocks in Figure 4.5. The three types of geometry blocks will be treated differently during the elimination stage of the algorithm.

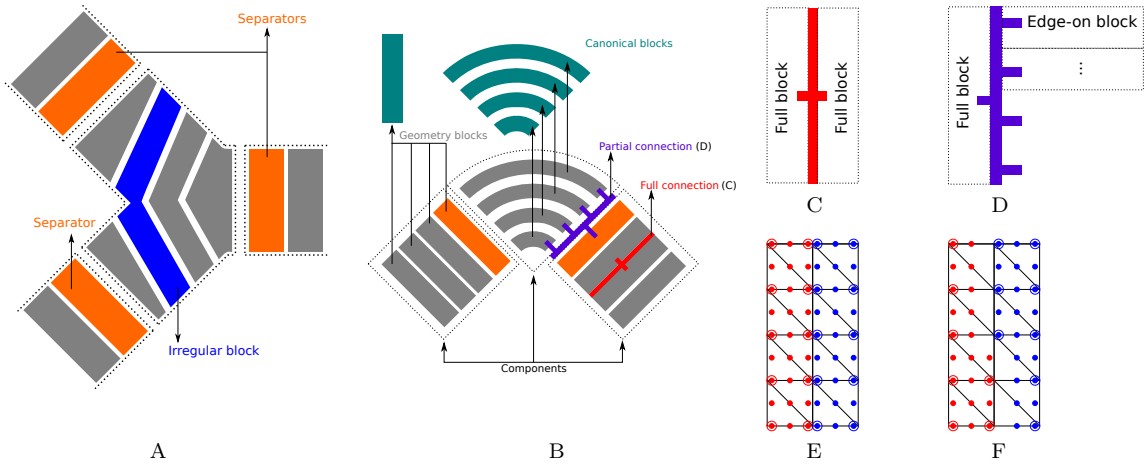


Figure 4.5: Illustration of terminology. In A and B we show two example domains, where components are enclosed in dotted lines. Components are divided into smaller pieces called geometry blocks; most steps of the algorithm function at this level of granularity. Geometry blocks are classified by their connectivity. Geometry blocks that are on the end of a pipe and touch another component are designated as separator blocks, or separators (■). Non-separator geometry blocks with at most two non-separator neighbor blocks are called regular blocks (■). All remaining geometry blocks have three or more non-separator neighbor blocks and are called irregular blocks (■). Between geometry blocks we might have full (■) or partial (■) connections, shown in B and illustrated separately in C and D. We call the block adjacent to a connection either a full block or an edge-on block based on the connection type, as shown in C and D. In B, we identify blocks with unique shapes as canonical blocks (■). Then we triangulate the canonical blocks and assign the degrees of freedom. When two geometry blocks are next to each other, their canonical degrees of freedom will be duplicated on their boundary, as shown in E. We resolve these to get the global degrees of freedom in F.

4.3.5 Canonical mesh construction

Once we have divided our geometry into geometry blocks, we need to construct meshes on those blocks. We use tetrahedral meshes (triangle meshes in 2D). We divide the algorithm into two stages.

Canonical cross section meshing The first stage is to mesh the connections between blocks. We independently mesh each canonical cross section. Although the meshing of these

cross sections can be performed arbitrarily, special considerations are needed to make sure that the final meshes will be consistent. We achieve this by choosing an interface mesh that is reversible. That is to say that the interface mesh looks the same when viewed from either side.

Canonical block meshing Meshing the interfaces between blocks first based on canonical cross sections gives us a number of important benefits. The interface between blocks is fixed, so we can construct meshes for each block independently. The mesh for the geometry block must conform to the interface mesh, but its generation is otherwise flexible. Since canonical blocks share the same geometry and the same canonical cross sections (and thus the same interface meshes), we can also give them the same mesh. This allows us to construct meshes independently per canonical block. Since the number of canonical blocks is typically much less than the number of geometry blocks, we typically only need to construct and store a mesh for a small fraction of the total fluid domain. We refer to the meshes constructed for canonical blocks as *canonical meshes*.

Meshing restrictions Although the meshing strategies are generally flexible, we do impose a few extra requirements. We require that each element must have at least one edge that is not on the boundary. Note that an edge that lies in the interior of a cross section between two blocks is not considered to be a boundary edge. That is, it is on the boundary of the block but not on the boundary of the full fluid domain mesh. This topology restriction is needed to prevent a numerical nullspace in our final discretization [17]. The second

requirement that we impose on block meshes is a connectivity requirement based on the assignment of cross section degrees of freedom to blocks; we address this in Section 4.3.7.

4.3.6 Canonical block matrix assembly

Once we have constructed our canonical meshes, we can begin the process of matrix assembly. The first step of our matrix assembly process is to compute the finite element integrals within each canonical mesh. We allocate degrees of freedom according to our Taylor-Hood finite element basis (See Section 4.3.2). We have a pressure degree of freedom at each vertex and co-located velocity degrees of freedom at each vertex and each edge of the mesh (See Figure 4.4). The Stokes equations are assembled into a symmetric indefinite linear system following the formulation in Section 4.3.1. We refer to these matrices as *canonical block matrices*.

Canonical block matrix assembly may be performed independently per canonical mesh (i.e., per canonical block). Since many blocks often share the same canonical block, matrix assembly is typically only performed for a subset of blocks. Matrix assembly occurs in the configuration of the canonical blocks, not in the configuration of the actual blocks. We represent our canonical block matrices as dense matrices; the number of degrees of freedom within each block should be kept small. See Section 4.5.2 for a discussion of block size and the use of dense matrix blocks. In practice, we delay canonical block matrix assembly until the execution stage. Matrices are assembled when they are first required to improve memory and cache usage.

4.3.7 Global degrees of freedom assignment

At this stage of the algorithm, we have a notion of canonical degrees of freedom, which are defined from the canonical mesh that we have computed for each canonical block. The canonical block matrices that we have assembled are indexed in terms of the canonical degrees of freedom. Canonical degrees of freedom do not correspond to physical degrees of freedom per se; canonical blocks are assembled in a reference coordinate system, and a single canonical block may correspond to many different geometry blocks within the fluid domain.

Geometry blocks naturally inherit degrees of freedom from their canonical block; we refer to these degrees of freedom as *geometry blocks degrees of freedom*. Geometry blocks degrees of freedom do correspond to physical degrees of freedom, but a single physical degree of freedom may belong to more than one block. This occurs for all degrees of freedom which occur along the connections between blocks, as shown in Figure 4.5. Our task is to assign each physical degree of freedom to one of geometry blocks that contains it. When doing so, we must be careful to avoid numerical problems later in the algorithm. We will call these *global block degrees of freedom*; they exist in one-to-one correspondence with physical degrees of freedom.

The degree of freedom mapping must be performed on geometry blocks and not canonical blocks. It is sometimes not possible to assign ownership of degrees of freedom to all instances of a canonical block in the same way. Blocks that are not indexed the same way do not produce duplicated matrix blocks in the final system, so it is desirable for the mapping to be done the same way whenever possible.

Boundary conditions Boundary conditions affect the assignment of global block degrees of freedom. Velocity degrees of freedom are not allocated where velocity boundary conditions are being enforced; these velocity samples are instead moved to the right hand side. Pressure degrees of freedom are allocated where velocity boundary conditions are being enforced.

Stability restrictions In order to avoid breakdowns during the elimination process, we divide the degrees of freedom along each connection between two blocks evenly between the two blocks. The reasons for this are discussed in detail in Section 4.5.1.

Connection types When assigning parent/child at each connection, it is helpful to distinguish between connections with one block on each side (*full connection*) and connections where a single block on one side of the connection touches multiple blocks on the other (*partial connection*). When a block occupies one entire side of a connection, we call the block a *full block*. Otherwise, the block is considered an *edge-on block*. Full connections have two full blocks. Partial connections have one full block and many edge-on blocks. We illustrate these concepts in Figure 4.5.

Ownership convention A simple convention in 2D to resolve ownership of degrees of freedom on connections is to walk around the perimeter of a geometry block in counterclockwise order. When you encounter a connection at which you are a full block, the half of the connection that you encounter first is the half owned by that block. The degrees of freedom on the other half are owned by the block or blocks on the other side of the connection. With this convention, both full blocks at a full connection agree on which half of the degrees of freedom are owned by which block. The only ambiguity is the degree of freedom in the

middle (which may be at a vertex or an edge). We must establish a globally-consistent rule for the ownership of this middle degree of freedom. We refer to the block that owns the middle degree of freedom as the *parent* and the block that does not own it as the *child*. Note that a block may (and usually is) a child at one connection and a parent at another. We employ two rules:

1. At partial connections, the full block is always the parent.

2. Blocks that are full blocks with respect to exactly connections are the parent of one connection and the child of the other.

When the two rules come into conflict, the first rule wins. The purpose of these rules is to avoid creating matrix dependencies between non-neighbor blocks (See Section 4.3.7). The assignment is otherwise arbitrary.

Triple junctions At partial connections (P), there are degrees of freedom shared by three blocks. One of these blocks (A) is the full block with respect to the partial connection. The other two are edge-on blocks (B, C) with respect to this connection; they always connect to each other through a full connection (Q). The partial connection P takes precedence; block A owns the same degrees of freedom that it would if P were a full connection. If the degree of freedom is not owned by block A, we decide whether the degree of freedom belongs to block B or C by looking at connection Q.

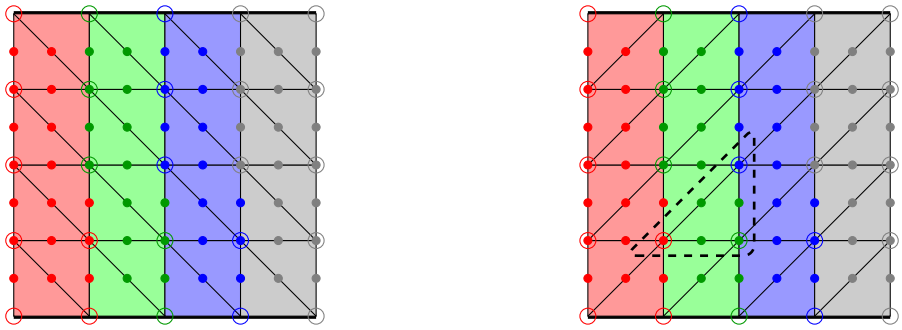


Figure 4.6: Block meshing with different diagonal edge directions. The background colors indicate the territory of blocks. The colors on edges and vertices indicate which block owns the degrees of freedom. The filled and hollowed circles are velocities and pressures respectively. The triangulation on the left is able to separate non-adjacent blocks. However the diagonal edge in alternate direction will allow the non-adjacent blocks (blue and red blocks shown on the right) interact. The dashed triangle shows the element containing the blue and red vertices that would introduce a non-zero matrix entry.

Spurious connectivity

When we constructed geometry blocks, we did so in a way that ensured that most blocks only touch two neighbors. Geometry blocks were considered neighbors only if their boundaries intersected. With respect to degrees of freedom, however, the notion of connectivity is somewhat different. Two degrees of freedom are connected if they share an element; pairs of degrees of freedom belonging to the same element correspond to nonzero matrix entries. We want to make sure that the matrix notion of connectivity corresponds to the geometry notion. As shown in Figure 4.6, it may be possible for degrees of freedom of non-neighboring blocks to be connected if care is not taken. This occurs whenever an element of a geometry block has vertices belonging to two different blocks. (This also occurs at triple junctions, but in this case the blocks involved are already neighbors.) In the case of our simple triangulation strategy, this problem is avoided by (a) choosing the diagonal

directions carefully and (b) preventing the block from being the child on both connections. In rare cases, spurious connectivity is still not eliminated; we resolve this by merging blocks.

4.3.8 System assembly

During canonical block matrix assembly, we performed finite element integration on each canonical block to compute canonical block matrices. This gave us a matrix and right hand side for each canonical block as in (4.7). We will denote the KKT matrix for block a as $\mathbf{B}_a$ an the right hand side as $\mathbf{b}_a$. These quantities are indexed by canonical block degrees of freedom. Observe that $\mathbf{B}_a$ is a symmetric matrix. We will ignore transformations in this section; we show how to include them Section 4.3.9.

The global system that we must solve has matrix blocks that are indexed with with global degrees of freedom. Global indices are unique (each degree of freedom belongs to exactly one global matrix block), while a single degree of freedom may exist within multiple canonical blocks. This means that integral contributions may have been calculated for a particular degree of freedom within multiple canonical blocks. These contributions must be added up while calculating global matrix blocks.

We introduce index mapping matrices $\mathbf{P}_{ab}$ to denote the correspondences between degrees of freedom in global blocks and canonical blocks. We define $(\mathbf{P}_{ab})_{ij} = 1$ if the *global* degree of freedom i within block a corresponds to the same degree of freedom as the *canonical* degree of freedom j within block b. $(\mathbf{P}_{ab})_{ij} = 0$ otherwise. Note that $\mathbf{P}_{aa}$ is just the canonical-to-global index map for block a. Since all dofs in a global block exist inside the corresponding canonical block, $\mathbf{P}_{aa}\mathbf{P}_{aa}^T = \mathbf{I}$.

Let $\mathbf{E}_{ab}$ be the global matrix block corresponding to block-row a and block-column b. Let the corresponding right hand side blocks be denoted as $\mathbf{h}_a$. These blocks can be computed from canonical matrix blocks as

$$\mathbf{E}_{ab} = \sum_c \mathbf{P}_{ac}\mathbf{B}_c\mathbf{P}_{bc}^T \qquad\qquad \mathbf{h}_a = \sum_c \mathbf{P}_{ac}\mathbf{b}_c,$$

where c runs over adjacent blocks. If a and c are not neighboring blocks (they do not share any degrees of freedom), then $\mathbf{P}_{ac} = \mathbf{0}$.

4.3.9 Transforms

To introduce transforms into our matrix blocks, we must first determine how integrals transform over individual elements. We assume that all transforms are affine (per block).

Elementwise transforms Consider a single element. A world space coordinate $\mathbf{x}$ can be transformed from its canonical space version $\hat{\mathbf{x}}$ by $\mathbf{x} = \mathbf{F}\hat{\mathbf{x}}+\mathbf{c}$, where $\mathbf{F}$ is a transform matrix and $\mathbf{c}$ is a constant displacement. We also denote $J = \det \mathbf{F}$. Then the basis functions in world space (without a hat) and in canonical space (with a hat) and their derivatives are related by

$$N_i(\mathbf{x}) = \hat{N}_i(\hat{\mathbf{x}}) = \hat{N}_i(\mathbf{F}^{-1}(\mathbf{x}-\mathbf{c})) \qquad P_i(\mathbf{x}) = \hat{P}_i(\hat{\mathbf{x}}) = \hat{P}_i(\mathbf{F}^{-1}(\mathbf{x}-\mathbf{c}))$$

$$\frac{\partial N_i}{\partial \mathbf{x}}(\mathbf{x}) = \mathbf{F}^{-T}\frac{\partial \hat{N}_i}{\partial \hat{\mathbf{x}}}(\mathbf{F}^{-1}(\mathbf{x}-\mathbf{c})) \qquad \frac{\partial P_i}{\partial \mathbf{x}}(\mathbf{x}) = \mathbf{F}^{-T}\frac{\partial \hat{P}_i}{\partial \hat{\mathbf{x}}}(\mathbf{F}^{-1}(\mathbf{x}-\mathbf{c})).$$

Our canonical matrix blocks $\mathbf{B}_a$ consist of viscosity blocks $\mathbf{A}$ and gradient blocks $\mathbf{G}$, as in (4.7). The blocks $\mathbf{A}$ are comprised of per-element blocks $\mathbf{A}_{ij} = \operatorname{tr}(\mathbf{D}_{ij})\mathbf{I} + \mathbf{D}_{ij}$, which are defined in (4.2) and (4.3). The blocks $\mathbf{G}$ are comprised of per-element vectors $\mathbf{g}_{ij}$, which are defined in (4.4). These transform as

$$\mathbf{g}_{ij} = J\mathbf{F}^{-T}\hat{\mathbf{g}}_{ij} \qquad\qquad \mathbf{D}_{ij} = J\mathbf{F}^{-T}\hat{\mathbf{D}}_{ij}\mathbf{F}^{-1}.$$

The matrix $\mathbf{A}_{ij}$, however, does not generally transform in a simple way, unless $\mathbf{F}^{-T}\mathbf{F}^{-1} = \mathbf{I}\operatorname{tr}(\mathbf{F}^{-T}\mathbf{F}^{-1})$. This is true if our transform is comprised of a combination of rotation, uniform scale, and translation. This restriction is why we were limited to transformations of this type when computing canonical blocks. With this assumption, we also have

$$\mathbf{A}_{ij} = J\mathbf{F}^{-T}\hat{\mathbf{A}}_{ij}\mathbf{F}^{-1}.$$

Blockwise transforms Blocks are composed by combining elementwise $\mathbf{A}_{ij}$ and $\mathbf{g}_{ij}$ into block-wise versions $\mathbf{A}$ and $\mathbf{G}$. We can express these in world space and canonical space matrices (for block a) as

$$\mathbf{B}_a = \begin{pmatrix} \mathbf{A} & \mathbf{G} \\ \mathbf{G}^T & \mathbf{0} \end{pmatrix} \qquad \hat{\mathbf{B}}_a = \begin{pmatrix} \hat{\mathbf{A}} & \hat{\mathbf{G}} \\ \hat{\mathbf{G}}^T & \mathbf{0} \end{pmatrix} \qquad \mathbf{B}_a = \mathbf{H}_a^T\hat{\mathbf{B}}_a\mathbf{H}_a,$$

where $\mathbf{H}_a$ is defined as

$$\mathbf{H}_a = \sqrt{J}\begin{pmatrix} \mathbf{F}^{-1} & & & & & \\ & \ddots & & & & \\ & & \mathbf{F}^{-1} & & & \\ & & & 1 & & \\ & & & & \ddots & \\ & & & & & 1 \end{pmatrix}.$$

Observe that $\mathbf{H}_a$ applies the matrix transform $\mathbf{F}^{-1}$ to each co-located velocity degree of freedom and only scales pressure degrees of freedom. Since both pressure and viscosity blocks are scaled by J, this scaling is split between the row and column scaling. This allows us to use the same $\mathbf{H}_a$ on both sides. Note that $\mathbf{H}_a$ is indexed using canonical indexing. Note also that the index a refers to the block a of the fluid domain, not a canonical block. Canonical blocks do not have transformations associated with them. Rather, each block stores a pointer to a canonical block and the transformation of the block relative to its canonical block. The matrix $\mathbf{H}_a$ is used to transform from the canonical coordinate system into world space. Although $\mathbf{H}_a$ is defined using canonical indexing, we can define a globally-indexed version by $\overline{\mathbf{H}}_a = \mathbf{P}_{aa}\mathbf{H}_a\mathbf{P}_{aa}^T$. This matrix has the same form as $\mathbf{H}_a$, but it contains a different number of dofs.

Global matrix blocks in canonical coordinates We can define a canonical-space version $\hat{\mathbf{E}}_{aa}$ of the world space block $\mathbf{E}_{aa}$, which is naturally defined according to $\mathbf{E}_{aa} =$

$\overline{\mathbf{H}}_a^T \hat{\mathbf{E}}_{aa} \overline{\mathbf{H}}_a$ or $\hat{\mathbf{E}}_{aa} = \overline{\mathbf{H}}_a{}^{-T} \mathbf{E}_{aa} \overline{\mathbf{H}}_a{}^{-1}$. We can extend this to off-diagonal blocks as

$$\hat{\mathbf{E}}_{ab} = \overline{\mathbf{H}}_a{}^{-T} \mathbf{E}_{ab} \overline{\mathbf{H}}_b{}^{-1}.$$

Then,

$$\begin{aligned}
\mathbf{E}_{ab} &= \sum_c \mathbf{P}_{ac} \mathbf{B}_c \mathbf{P}_{bc}^T \\
\hat{\mathbf{E}}_{ab} &= \sum_c \overline{\mathbf{H}}_a{}^{-T} \mathbf{P}_{ac} \mathbf{B}_c \mathbf{P}_{bc}^T \overline{\mathbf{H}}_b{}^{-1} \\
&= \sum_c \overline{\mathbf{H}}_a{}^{-T} \mathbf{P}_{ac} \mathbf{H}_c^T \hat{\mathbf{B}}_c \mathbf{H}_c \mathbf{P}_{bc}^T \overline{\mathbf{H}}_b{}^{-1} \\
&= \sum_c \overline{\mathbf{P}}_{ac} \hat{\mathbf{B}}_c \overline{\mathbf{P}}_{bc}^T \\
\overline{\mathbf{P}}_{ac} &= \overline{\mathbf{H}}_a{}^{-T} \mathbf{P}_{ac} \mathbf{H}_c^T.
\end{aligned}$$

This directly relates global matrix blocks in canonical coordinates with the canonical block matrices.

Transformation invariance The matrix $\overline{\mathbf{P}}_{ac}$ maps degrees of freedom (as $\mathbf{P}_{ac}$ does) but also applies a transformation along the way. This transformation is $\sqrt{J_a^{-1} J_c} \mathbf{F}_a^T \mathbf{F}_c^{-T}$ for co-located velocity degrees of freedom and $\sqrt{J_a^{-1} J_c}$ for pressure degrees of freedom. If two blocks are connected, then the relative orientation between the two blocks is fixed. If one block is rotated, scaled, or translated, then the connected block must be rotated, scaled, and translated by the same amount in order to remain connected. This would replace

$\mathbf{F}_a \to \mathbf{R}\mathbf{F}_a$ and $\mathbf{F}_c \to \mathbf{R}\mathbf{F}_c$, so that $\mathbf{F}_a^T\mathbf{F}_c^{-T} \to \mathbf{F}_a^T\mathbf{R}^T\mathbf{R}^{-T}\mathbf{F}_c^{-T} = \mathbf{F}_a^T\mathbf{F}_c^{-T}$ is unchanged. Similarly, $J_a^{-1}J_c = \det(\mathbf{F}_a^T\mathbf{F}_c^{-T})^{-1}$ must remain unchanged.

Duplicated matrix blocks Noting that $\overline{\mathbf{P}}_{ac}$ does not depend on block orientation suggests that $\hat{\mathbf{E}}_{ab}$ may be computed once for each canonical block, but this is not the case. We will have $\hat{\mathbf{E}}_{ab} = \hat{\mathbf{E}}_{cd}$ if (a) all of the blocks involved in the sum correspond to the same canonical blocks, (b) are connected through the same sockets, and (c) are indexed the same way in global indexing. For example, $\hat{\mathbf{E}}_{aa} \neq \hat{\mathbf{E}}_{cc}$ if blocks a and c are connected to different types of blocks, even though a and c have the same canonical block. Requirement (c) is actually somewhat stronger than is required, since not all indices of the blocks involved may participate in the computation of $\hat{\mathbf{E}}_{ab}$. Nevertheless, we use rule (c) since it is easy to check during global degree of freedom assignment. Identifying copies of $\hat{\mathbf{E}}_{ab}$ that are the same is critical, as this the only form of redundancy that will be passed to the final linear system that must be solved.

Transformed system Consider a simple geometry consisting of three blocks (1, 2, and 3) connected in sequence. The world-space global system that must be solved looks like

$$\begin{pmatrix} \mathbf{E}_{11} & \mathbf{E}_{12} & \\ \mathbf{E}_{12}^T & \mathbf{E}_{22} & \mathbf{E}_{23} \\ & \mathbf{E}_{23}^T & \mathbf{E}_{33} \end{pmatrix} \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \mathbf{x}_3 \end{pmatrix} = \begin{pmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \mathbf{b}_3 \end{pmatrix},$$

Here, $\mathbf{x}_a$ are the degrees of freedom assigned to block a (including velocity and pressure). In general, the blocks $\mathbf{E}_{ab}$ will vary with the orientations of the blocks. We can replace these

with canonical-space versions

$$\begin{pmatrix} \overline{\mathbf{H}}_1^T \hat{\mathbf{E}}_{11} \overline{\mathbf{H}}_1 & \overline{\mathbf{H}}_1^T \hat{\mathbf{E}}_{12} \overline{\mathbf{H}}_2 & \\ \overline{\mathbf{H}}_2^T \hat{\mathbf{E}}_{12}^T \overline{\mathbf{H}}_1 & \overline{\mathbf{H}}_2^T \hat{\mathbf{E}}_{22} \overline{\mathbf{H}}_2 & \overline{\mathbf{H}}_2^T \hat{\mathbf{E}}_{23} \overline{\mathbf{H}}_3 \\ & \overline{\mathbf{H}}_3^T \hat{\mathbf{E}}_{23}^T \overline{\mathbf{H}}_2 & \overline{\mathbf{H}}_3^T \hat{\mathbf{E}}_{33} \overline{\mathbf{H}}_3 \end{pmatrix} \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \mathbf{x}_3 \end{pmatrix} = \begin{pmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \mathbf{b}_3 \end{pmatrix}$$

$$\begin{pmatrix} \hat{\mathbf{E}}_{11} & \hat{\mathbf{E}}_{12} & \\ \hat{\mathbf{E}}_{12}^T & \hat{\mathbf{E}}_{22} & \hat{\mathbf{E}}_{23} \\ & \hat{\mathbf{E}}_{23}^T & \hat{\mathbf{E}}_{33} \end{pmatrix} \begin{pmatrix} \overline{\mathbf{H}}_1 \mathbf{x}_1 \\ \overline{\mathbf{H}}_2 \mathbf{x}_2 \\ \overline{\mathbf{H}}_3 \mathbf{x}_3 \end{pmatrix} = \begin{pmatrix} \overline{\mathbf{H}}_1{}^{-T} \mathbf{b}_1 \\ \overline{\mathbf{H}}_2{}^{-T} \mathbf{b}_2 \\ \overline{\mathbf{H}}_3{}^{-T} \mathbf{b}_3 \end{pmatrix},$$

This amounts to solving for transformed degrees of freedom $\mathbf{y}_a = \overline{\mathbf{H}}_a \mathbf{x}_a$ with a canonical-space matrix and a transformed right hand side. The solution is easily transformed back into world space with $\mathbf{x}_a = \overline{\mathbf{H}}_a{}^{-1} \mathbf{y}_a$. Solving this canonical-space version of the problem is preferable since it will normally contain many more duplicate matrices than the global space version of the problem.

As with canonical block matrix assembly, we delay the assembly of the global system matrix blocks until the task execution phase. These matrix blocks are assembled from canonical block matrices as they are needed.

4.4 Elimination algorithm

With our global system fully assembled, our next task is to solve the resulting linear system. The system is block sparse. Many of the blocks in the system are duplicates of other matrix blocks, possibly with transpose. The elimination algorithm proceeds in four phases.

Regular blocks Regular blocks, by definition, have at most two neighboring regular block. If all other blocks are removed, the global system decomposes into separate components (each geometrical component corresponds to a connected component in the resulting matrix). Each of these connected components is tridiagonal and can be eliminated efficiently with cyclic reduction. We use this cyclic reduction order to eliminate all regular blocks from the global system. Note that regular blocks may have more than two neighbors in the global system due to the presence of irregular blocks and separators. Irregular blocks and separators result in fill-in during elimination, but they also bound this fill-in by preventing it from spreading outside of a component. The use of cyclic reduction exposes a large number of tasks that can be executed in parallel. In the presence of duplicate matrix blocks (especially pipes), it is also very effective at exposing caching opportunities. The vast majority of blocks are regular, so relatively few blocks remain after this stage of elimination.

Irregular blocks After regular blocks are eliminated, only irregular blocks and separators remain. Irregular blocks are eliminated next in arbitrary order. Eliminating irregular blocks creates fill-in, but separators bound the fill-in by preventing it from spreading to other components. As long as care is taken to ensure that the order of elimination is the same every time the blocks in a component are eliminated (start cyclic reduction from the same side, and eliminate irregular blocks in the same order), *nearly all* of the computations involved in eliminating one copy of a component will coincide exactly with the computations needed to eliminate another copy.

Separators I Separators are eliminated in two phases. During the first phase, separators with at most two neighbors are eliminated in arbitrary order. These blocks are easy to detect, and their elimination never produces fill-in.

Separators II The remaining separators are eliminated from the system. We use CO-LAMD order [44] to reduce fill-in. This is the only elimination stage where the amount of fill-in produced is not readily bounded. This is compensated by the fact that only a small fraction of the original number of blocks remain in the system. (The planar topology if typical microfluidic devices also tends to limit fill-in.) After this stage, all rows of the system have been eliminated.

4.4.1 Planning and optimization

Each phase of elimination proceeds by repeated application of block-row elimination. Each row operation consists of a sequence of basic linear algebra operations (See Figures 4.1 and 4.2). The elimination stages are treated as planning stages; rather than performing the operations required, we instead treat each operation as a tasks. The dependency relationships between the tasks form a directed acyclic graph.

Forward and backward substitution The row elimination operation also emits tasks for both forward and backward substitution. Note that the operations that will be required for backward substitution are known during elimination stage, even though these tasks would not be able to execute until after the forward phase.

Name	Resolution	Total dofs	Blocks	Tasks	Avg dofs/block
grid20	16	6.0 M	43080	174.0 K	140.2
rgrid0	16	2.0 M	14334	35.5 K	138.0
voronoi-s4	16	1.3 M	9798	34.7 K	136.2
grid20-3d	6	5.8 M	15470	108.0 K	373.3
rgrid0-3d	6	1.9 M	5281	14.9 K	358.5
voronoi-s4-3d	6	1.3 M	3582	13.8 K	354.5

Table 4.1: Statistics of scaling with parallelism tests

Matrix and vector IDs To facilitate handling duplicates, we store a sparse matrix of matrix IDs. Each *essentially unique* block matrix is assigned a unique ID. Two matrices are not considered essentially unique if they differ only by negation or transpose. We reserve a bit for negation and a bit for transpose to represent matrices that are essentially the same as another matrix. We reserve two special IDs to indicate a zero matrix and an identity matrix; since the block row and block column in which the matrix is stored uniquely identifies its dimensions, it is unnecessary to distinguish special matrices of different sizes. A similar ID scheme is applied to vectors.

Caching Each task consists of a simple linear algebra operation and produces an intermediate matrix or vector as output. Each of these intermediate quantities is assigned an ID. A simple hashing scheme is used to detect that an intermediate quantity is being computed twice. The hashing is aware of negation, transpose, associativity, and (for addition) commutativity. We do not include distributivity, as this would make the problem very difficult. This simple hashing scheme allows us to detect and eliminate duplicate calculations and simply reuse the results of the earlier computation. This simple idea is the basis for the majority of the performance benefits observed from the proposed algorithm.

Operation simplification One benefit of representing identity and zero objects with special indices is that we are able to simplify or eliminate many operations during the planning stage. For example, during an elimination step a matrix-vector multiply by a zero vector simply results in the ID for the zero vector; no task is produced. Similarly, when a zero matrix is added to another matrix, the ID for the second matrix is returned without generating a task. These simplifications can be quite dramatic. For many problems, nearly all initial right hand side vectors are zero, and the vast majority of vector operations that occur during forward elimination will be optimized away. When it does not prevent caching opportunities, we merge tasks to correspond to BLAS operations. This allows us, for example, to merge some sequences of operations ($A = B + C$, $B = -D$, $C = EF$, $E = G^T$) into a single BLAS operation ($A = -D + G^T F$). This also allows us to use the same memory location for A and D. We use the Intel MKL-BLAS for our basic linear algebra and LAPACK for our matrix inverses (and final-block pseudo-inverse when required).

Stability and pseudo-inverse for the last block Our stability considerations ensure that our elimination procedure does not break down during elimination. That is, we will never be required to invert a matrix block that is singular. The one exception to this is the very last block. If the whole system contains a constant pressure nullspace due to the absence of traction boundary conditions, the last block to be inverted will be singular. We perform a pseudo-inverse on this singular block. This is done at most once and does not affect performance.

4.4.2 Task execution

During the task execution phase, we perform all of the actual computations required for elimination. In addition, we also assemble the matrices for the global system matrix as they are required by elimination calculations (See Sections 4.3.6 and 4.3.8). These tasks are computationally or memory intensive and benefit from the parallelism, load balancing, and memory management that we perform during task execution. In addition to a core computation, tasks are also responsible for allocating and assembling finite element matrices (if required and not already available), allocating space for their output (unless it shares space with an input), and freeing memory for intermediates that are no longer required.

We assign a priority to each task equal to the length of the longest dependency chain starting at the task [1, 64, 3]. The time required to complete the most expensive dependency chain places a lower bound on the time required to complete a set of tasks, even if unlimited processors are available to complete them. These priorities tend to encourage long dependency chains to be executed quickly. Indeed, favorable scaling to 16 cores is observed with the proposed method (See Section 4.6.4).

4.5 Analysis

4.5.1 Stability

Being equivalent to un-pivoted Gaussian elimination on a permuted system [103], breakdowns (zeros on the diagonal) and entry growth are in general possible [9]. Cyclic reduction has been extensively studied for the solution of the Poisson equation, which is

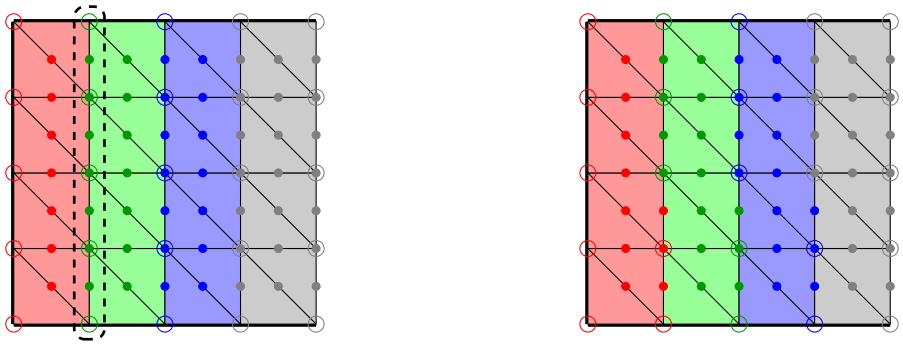


Figure 4.7: Here we show how different placements of degrees of freedom would affect the elimination. The background colors indicate the territory of blocks. The colors on edges and vertices indicate which block owns the degrees of freedom. The filled and hollowed circles are velocities and pressures respectively. The left, bottom, and top sides are with velocity boundary conditions. On the left eliminating the red block would fail. This is because the degrees of freedom on the edge (marked by dashed rectangle) are all owned by the green block, and the elimination of the red block can be regarded as solving a smaller system with solely velocity boundary conditions (three from the original domain, and right side being the effective one). The constant pressure nullspace makes the system singular. We solve this by dividing the degrees of freedom between the adjacent blocks (shown on the right).

symmetric and positive definite. It has been shown that cyclic reduction is stable for diagonal dominant and symmetric positive definite systems [58], where off-diagonal entries even tend to become smaller in subsequent iterations [85, 82]. Since our systems are symmetric indefinite, we must take care to avoid these problems.

We place a restriction on block meshing (See Section 4.3.4) to avoid numerical nullspaces and elimination breakdowns. Assignment of global degrees of freedom also plays an important role in preventing breakdowns. Consider the elimination of an individual block. As the first step we invert its diagonal block matrix in the system. This block is just a Stokes flow discretization of the corresponding geometry block with some effective boundary conditions. When the effective boundary conditions correspond to velocity boundary conditions, this discretization has the constant-pressure nullspace. This situation occurs when we assign all degrees of freedom to one of neighbor blocks. We illustrate one example

in Figure 4.7. Our solution to this problem is to split the degrees of freedom between the neighbor blocks.

4.5.2 Scaling

One of the limitations of the proposed method is its scaling with resolution. In the absence of caching opportunities, we will have n geometry blocks, each with m degrees of freedom. Let s be the number of separators (similar to the number of components).

Memory usage Each block is dense and requires $O(m^2)$ storage, for $O(m^2 n)$ overall storage. The first three phases of elimination create at most a constant amount of fill in, so total memory use through the end of these phases is $O(m^2 n)$. This leaves us with $s \ll n$ separators. During this phase, fill-in is possible. Based on the planar topology, the number of operations should grow as $O(s^{1.5})$, which leads to $O(m^2 s^{1.5})$ additional storage. Under refinement by a factor of k, $n \to kn$, $m \to km$ ($m \to k^2 m$ in 3D), and $s \to s$. This leads to $O(k^3)$ (2D) or $O(k^5)$ (3D) scaling in memory usage. In practice, actual memory requirement is significantly better than these predictions, since many blocks are duplicates and need not be stored. Nevertheless, this is noticeably worse than the optimal scaling of $O(k^2)$ and $O(k^3)$ respectively. Indeed, memory is the limiting factor of our method in 3D, where we start reaching our memory limitations at around 10M degrees of freedom on realistic geometry.

Computational cost Computational cost closely follows memory usage, except that operations on our blocks scale as $O(m^3)$. This gives us $O(m^3 n + m^3 s^{1.5})$ computational cost. This scales with resolution k as $O(k^4)$ in 2D and $O(k^7)$ in 3D, which compares poorly with optimal at $O(k^2)$ and $O(k^3)$. In practice, this scaling is not observed as long as channel is

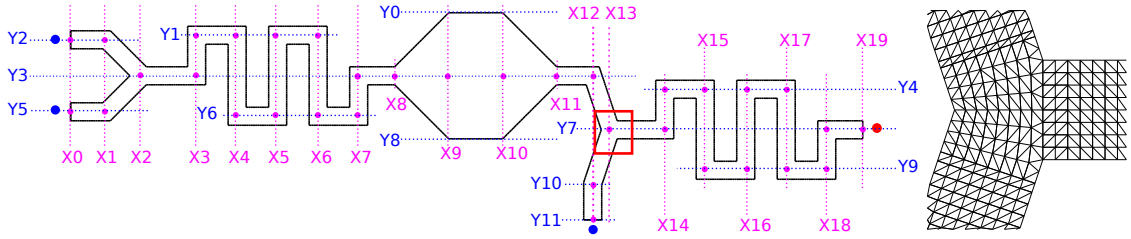
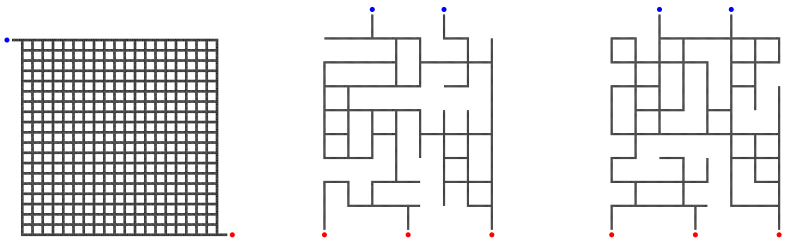


Figure 4.8: Domain of the test case "wide". The blue dots (•) indicate inflow ports, and the red dots (•) indicate free surface outflow. On the right we show the mesh inside the red box at resolution 8. The x and y coordinates of nodes are labeled with "X#" and "Y#" respectively. Their values can be read from Table 4.4.

not too wide. Over the relevant range of resolutions, our tests suggest 2D scaling of $O(k^a)$ with a between 2.1 and 2.6 on suitable geometry and 3D scaling with a between 5.2 and 5.4. (See Section 4.6.5 for details.) The proposed method is performance competitive with state-of-the-art methods even at around 1M degrees of freedom in both 2D and 3D. (See Section 4.6.6 for timing comparisons.)

Impacts of cross section size The proposed method scales poorly with block size. This suggests that blocks should have as few degrees of freedom as possible while satisfying connectivity requirements. The detrimental effects of large cross sections may be observed in the "wide" test case (in 2D and 3D), where a very wide cross section in a portion of the fluid domain causes global performance deterioration. (See Section 4.6.6.) The isolating effects of separator blocks and the independence of components means that components with large cross sections may be eliminated using an alternative sparse direct solver (such as MUMPS); the sparse LU or LDL^T factorization may then be used in lieu of dense blocks for the component. This would allow the method to overcome the effects of such components while retaining the benefits in other regions. Observe that the caching benefits are retained;



"grid20" "rgrid0" "rgrid1"

Figure 4.9: Domains of grid-shaped tests. The blue dots (•) indicate inflow ports, and the red dots (•) indicate free surface outflow. For simplicity, we draw each pipe as a filled stroke, by connecting the central vertices at the ends of the pipe. In "grid20" The coordinates for the bottom left and top right vertices are $(0, 0)$ and $(0.95, 0.95)$ respectively. In tests "rgrid0" and "rgrid1" The coordinates of vertices are contained in a box with bottom left corner $(0, 0)$ and top right corner $(1.575, 2.025)$. In all of these tests a uniform cross section of 0.0125 is used.

if the same large component is repeated in the device's design, it need only be eliminated once.

4.6 Numerical results

4.6.1 Sample device geometries

We use six different geometry templates for our numerical tests.

- **"wide"** is an example of a relatively simple microfluidic device. The device and its precise geometry are shown in Figure 4.8. This geometry includes a component with very large cross sections to illustrate the performance degradation that occurs in this case.

- **"grid20"** is a large regular grid of pipes (See Figure 4.9). This example benefits heavily from the regularity of the geometry, resulting in lots of caching opportunities;

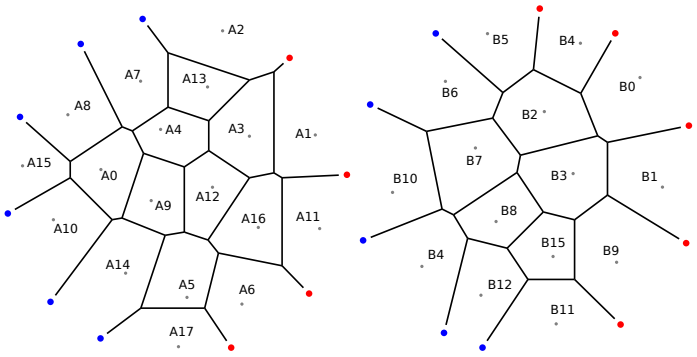


Figure 4.10: Domains of tests "voronoi-s4" (left) and "voronoi-s15" (right). The blue dots (●) indicate inflow ports, and the red dots (●) indicate free surface outflow. The cell centers are labeled by "A#" and "B#" in "voronoi-s4" and "voronoi-s15" respectively. The coordinates of cell centers are listed in Table 4.5.

this tends to accelerate the earlier elimination stages. On the other hand, it has a large number of separator blocks, which makes the final stage of elimination more expensive. Because of the large number of pipes, this example also has the highest degree of freedom count relative to the resolution of the pipes.

- **"rgrid0"** and **"rgrid1"** were selected from a large database of grid-like automatically generated microfluidic devices [170]. In that work, this database of devices was very expensive to compute using a commercial software package. The geometry for these devices is shown in Figure 4.9.

- **"voronoi-s4"** and **"voronoi-s15"** are randomly generated from Voronoi diagrams clipped to the circle centered at the origin with radius 0.5 (See Figure 4.10). In these tests all pipes are joined at unique angles; this prevents any blocks (other than the pipes) from being cached.

Name	Resolution	Total dofs	Blocks	Tasks	Avg dofs/block
grid20	18	7.6 M	48602	187.0 K	157.2
grid20	36	30.9 M	98300	305.7 K	314.4
rgrid0	18	2.5 M	16134	42.8 K	154.9
rgrid0	36	10.1 M	32495	78.3 K	311.7
voronoi-s4	18	1.7 M	11041	38.6 K	153.1
voronoi-s4	36	6.8 M	22226	75.9 K	307.7
grid20-3d	4	1.5 M	9948	94.9 K	154.1
grid20-3d	10	29.7 M	26514	134.3 K	1118.4
rgrid0-3d	4	0.5 M	3438	10.1 K	145.2
rgrid0-3d	10	9.7 M	8855	23.3 K	1089.9
voronoi-s4-3d	4	0.3 M	2340	9.8 K	143.3
voronoi-s4-3d	10	6.5 M	6066	22.1 K	1066.4

Table 4.2: Statistics of scaling with refinement tests. Only the smallest and largest resolutions are shown.

In each example, a fixed channel width w is used for all pipes. In 3D tests, we extrude the domain along the z direction by w, which produces a square cross section for pipes. We use characteristic block width $h = \frac{w}{r}$, where r is the resolution. At all inflow regions, we enforce velocity boundary conditions with a quadratic Poiseuille flow velocity profile in 2D. In 3D, the input velocity profile is quadratic in both the horizontal and vertical directions; the velocity is zero at the walls and greatest in the middle. We use flow rates of $0.005m^2s^{-1}$ (2D) or $0.005m^3s^{-1}$ (3D) at all inflows on all non-analytic tests.

4.6.2 Analytic convergence tests

We begin by performing convergence tests on all of our devices. Since analytic solutions to the Stokes equations are known only for simple geometry setups, we instead use the method of manufactured solutions to perform our analytic tests [144, 147, 146]. This allows us to choose arbitrary velocity and pressure fields. We choose velocity and pressure fields that are combinations of trigonometry and polynomials and have oscillations on the length scale of about 0.4, which is small enough to be well-resolved at all resolutions and yet high enough to exhibit significant nonlinearity. In all tests, we used the analytic fields

below.

Field	$\mathbf{u}(\mathbf{x})$	$p(\mathbf{x})$
2D	$\begin{pmatrix} \sin(12x)y + \cos(15y) + xy \\ \cos(14x)\cos(13y) + \sin(16y)x + x^2 - 1 \end{pmatrix}$	$\sin(15x + 10y + 1)$
3D	$\begin{pmatrix} \sin(14x)y + \cos(15y)z + xy \\ \cos(14x)\cos(16y) + \sin(15y)x + x^2 + yz - 1 \\ \sin(17z)y + \cos(15x)z \end{pmatrix}$	$\sin(15x + 14y + 1) + \cos(16z)$

Note that the velocity fields are not divergence free, do not follow the domain geometry, and do not satisfy the Stokes equations. Instead, we use a right hand side term for divergence (See Section 4.3.1), enforce velocity boundary conditions at all inflows and pipe walls, enforce traction boundary conditions at outflows, and use a forcing term to make the analytic solution satisfy the Stokes equations. This allows us to do very precise refinement studies even with our very irregular geometry. We use a viscosity of $\mu = 1$.

We compute L^∞ and L^2 errors of computed pressures p and velocities $\mathbf{u}$ using

$$L_p^\infty = \max_i |p(\mathbf{x}_i) - p(\mathbf{x}_i)| \qquad L_p^2 = \sqrt{\frac{1}{N_p}\sum_i \left(p(\mathbf{x}_i) - p(\mathbf{x}_i)\right)^2}$$

$$L_{\mathbf{u}}^\infty = \max_j \|\mathbf{u}(\mathbf{x}_j) - \mathbf{u}(\mathbf{x}_j)\|_\infty \qquad L_{\mathbf{u}}^2 = \sqrt{\frac{1}{N_{\mathbf{u}}}\sum_j \|\mathbf{u}(\mathbf{x}_j) - \mathbf{u}(\mathbf{x}_j)\|_2^2},$$

where N_p is the total number of pressure degrees of freedom, and $N_{\mathbf{u}}$ is the total number of vertices and edges with velocity degrees of freedom. We conduct the refinement study by changing the resolution r, which is the number of elements along the cross section of a pipe.

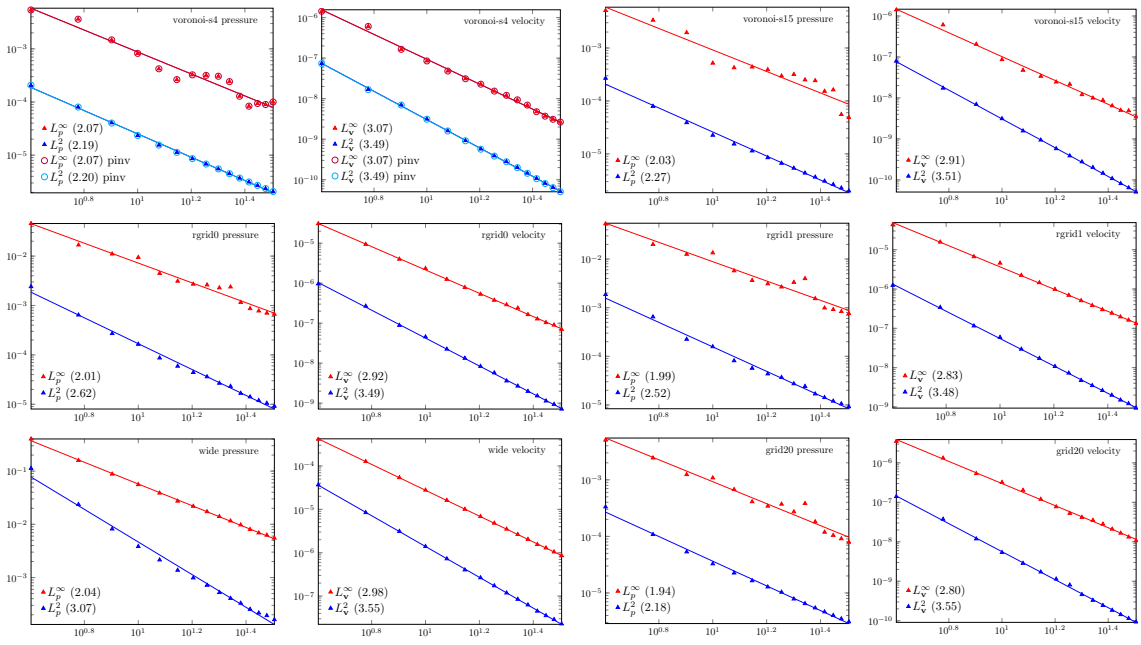


Figure 4.11: Analytic convergence tests for L^∞ and L^2 error measures in 2D. The markers indicate the computed errors. The solid lines are least square regression lines used to compute the convergence rates. The convergence rates are shown in the legends. The resolution is the number of edges in a regular channel. We also run tests on a modified version of "voronoi-s4", which contains velocity boundary conditions only. In that case, pseudo-inverse is used to eliminate the last block. We show L^∞ and L^2 errors using circles (○ and ○ respectively). These tests are indicated with "pinv."

The results of the refinement tests are shown in Figure 4.11 for 2D and Figure 4.12 for 3D. In all cases, we observe second order convergence in pressure and third order convergence in velocity in both L^∞ and L^2. This is the optimal convergence order for the Taylor-Hood elements that we use in our discretization.

In 3D, memory usage restricts the resolutions to $r = 10$. At this resolution, simulations contain on the order of 10M degrees of freedom. (See Table 4.2 for precise statistics.)

Nullspace We repeat test "voronoi-s4" in both 2D and 3D with all traction boundary conditions replaced by velocity boundary conditions. These boundary conditions result in

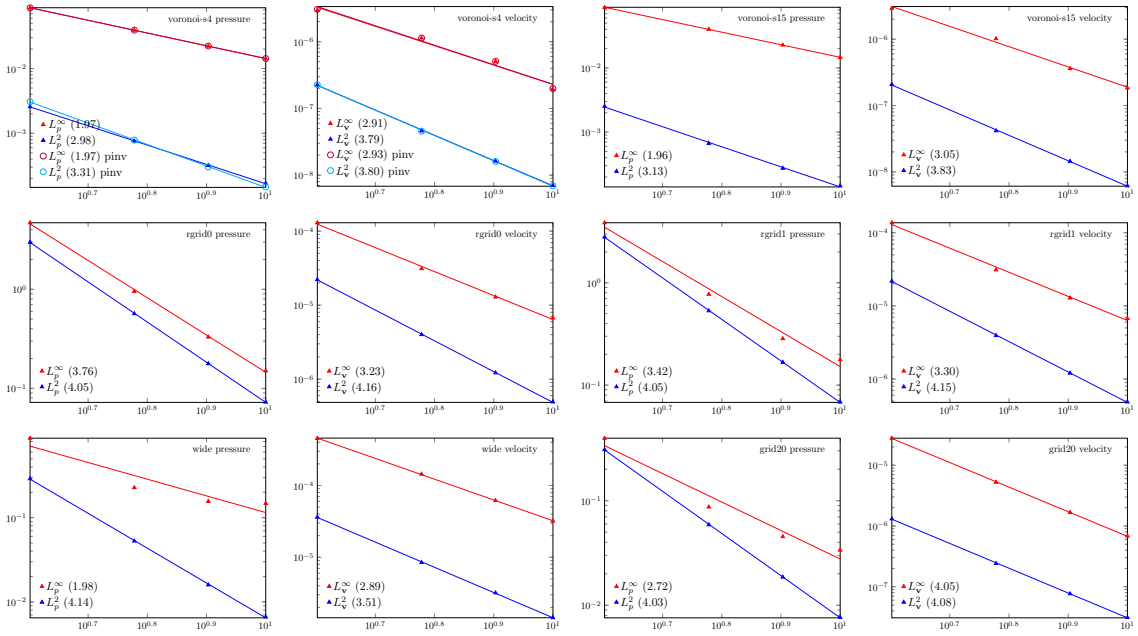


Figure 4.12: Analytic convergence tests for L^∞ and L^2 error measures in 3D. The markers indicate the computed errors. The solid lines are least square regression lines used to compute the convergence rates. The convergence rates are shown in the legends. The resolution is the number of edges in a regular channel. We also run tests on a modified version of "voronoi-s4", which contains velocity boundary conditions only. In that case, pseudo-inverse is used to eliminate the last block. We show L^∞ and L^2 errors using circles (○ and ○ respectively). These tests are indicated with "pinv."

a constant pressure nullspace. This nullspace is handled as described in Section 4.4.1. The convergence results are shown alongside the original boundary conditions in Figures 4.11 and 4.12 and are indicated by "pinv" in the legend. The pressure nullspace has no significant effect on the accuracy, convergence, or performance of the method.

4.6.3 Convergence tests using real boundary conditions

In the analytic convergence tests, we considered test cases with a smooth velocity and pressure profiles everywhere. Real flows around sharp corners, however, may have high velocity gradients in these regions. High velocity gradients are also observed at corners

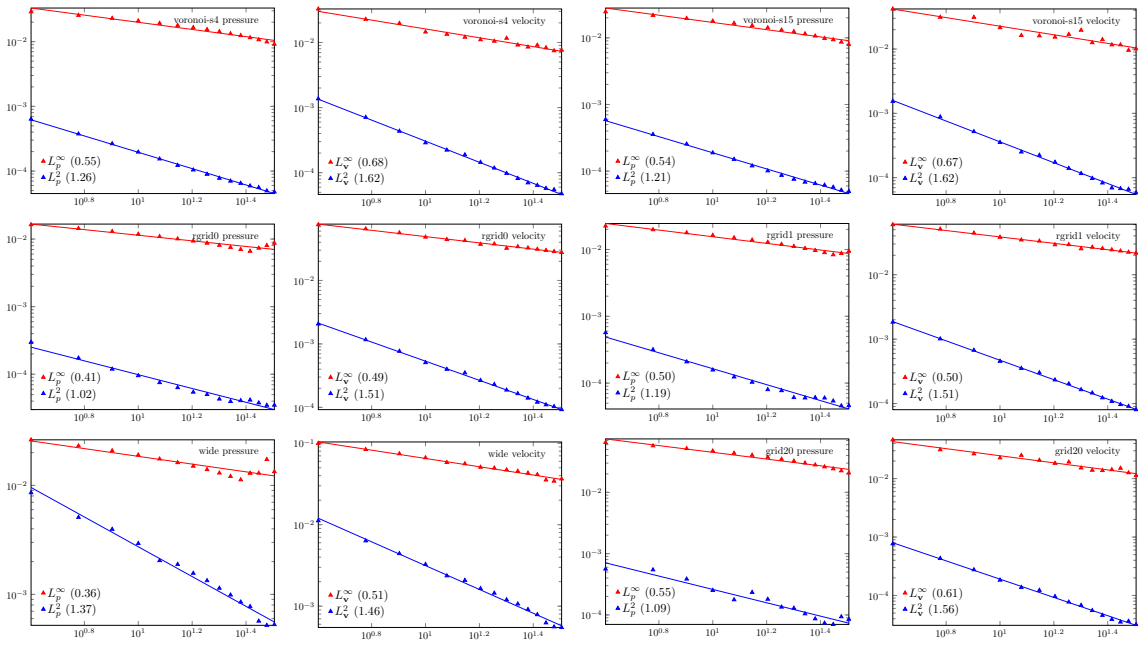


Figure 4.13: Convergence tests with real boundary conditions for L^∞ and L^2 error measures in 2D. The markers indicate the computed errors. The solid lines are least square regression lines used to compute the convergence rates. The convergence rates are shown in the legends. The resolution is the number of edges in a regular channel.

where a Poiseuille flow profile must conform to a traction-free outflow boundary condition. These gradients reduce the convergence order in L^∞, especially with the regular element sizes used in this study. In all tests, inflow ports have a flow rate of 0.005. The viscosity is 8.9×10^{-4}. We use the solution at a fine resolution ($r = 60$ for 2D and $r = 16$) as our reference to compute the error. The error is normalized by the maximum magnitude seen in the reference solution. The results of convergence tests for 2D and 3D are shown in Figure 4.13 and Figure 4.14. We show the distribution of errors and solution gradients in Figure 4.16.

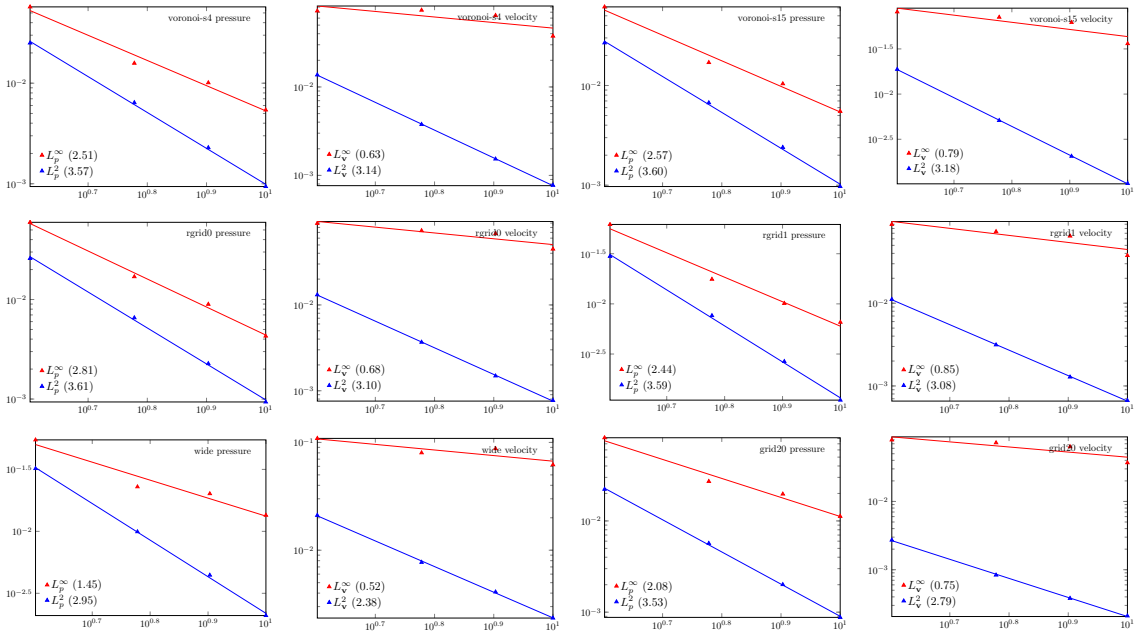


Figure 4.14: Convergence tests with real boundary conditions for L^∞ and L^2 error measures in 3D. The markers indicate the computed errors. The solid lines are least square regression lines used to compute the convergence rates. The convergence rates are shown in the legends. The resolution is the number of edges in a regular channel.

4.6.4 Scaling with parallelism

In this section we run the tests "grid20", "rgrid0" and "voronoi-s4" at fixed resolution ($r = 16$ in 2D, $r = 6$ in 3D) with different numbers of cores (1-16) to evaluate how well the method scales with available cores. The physical properties are the same as in Section 4.6.3. Statistics of the tests are shown in Table 4.1, and results are shown in Figure 4.15. A speedup of 9-13 times is observed in 3D when increasing cores from 1 to 16. In 2D, a more modest factor of 5-8 is observed instead. The reduced scaling in 2D is due to the lower computational cost of tasks in 2D (and thus scheduling overhead is relatively more expensive). Test "grid20" is more expensive since it is a larger test with more degrees of

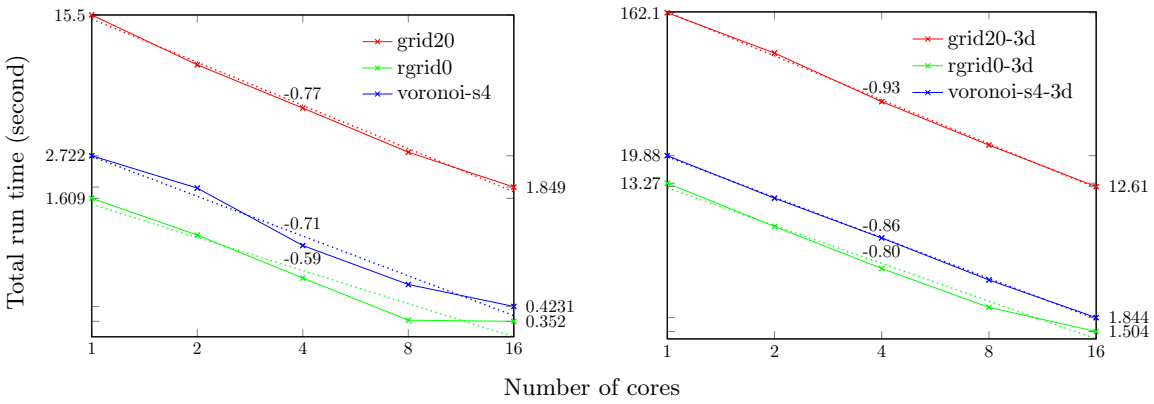


Figure 4.15: The solving time for various number of threads in logarithm scale. We measure the total run time including meshing, system construction, elimination, and back solve. The dotted lines are the regression lines of the data. The regression order is also shown above each of them. The run time for 1 and 16 threads are shown by the left and right vertical axis.

freedom. The numbers included in the plots are fit line slopes; a slope of s indicates runtime scaling as $O(c^s)$, where c is the number of cores. $s = -1$ is ideal.

4.6.5 Scaling with refinement

In this section, we evaluate the scaling of the method with resolution. We again run tests "grid20", "rgrid0" and "voronoi-s4." This time, we fix the core count at 16 and instead vary the resolution. Test statistics are shown in Table 4.2, and results are shown in Figure 4.18. The numbers in the plots represent the slope of the fit line. A slope of s indicates runtime $O(r^s)$, where r is the resolution. In 2D, observed scaling is $O(r^{2.1}) - O(r^{2.6})$; this compares very favorable with ideal $O(r^2)$ and is much better than the predicted $O(r^5)$. In 3D, observed scaling is $O(r^{5.2}) - O(r^{5.4})$; this is significantly worse than the optimal $O(r^3)$ but also much better than the predicted $O(r^7)$. In each case, observed scaling is far better

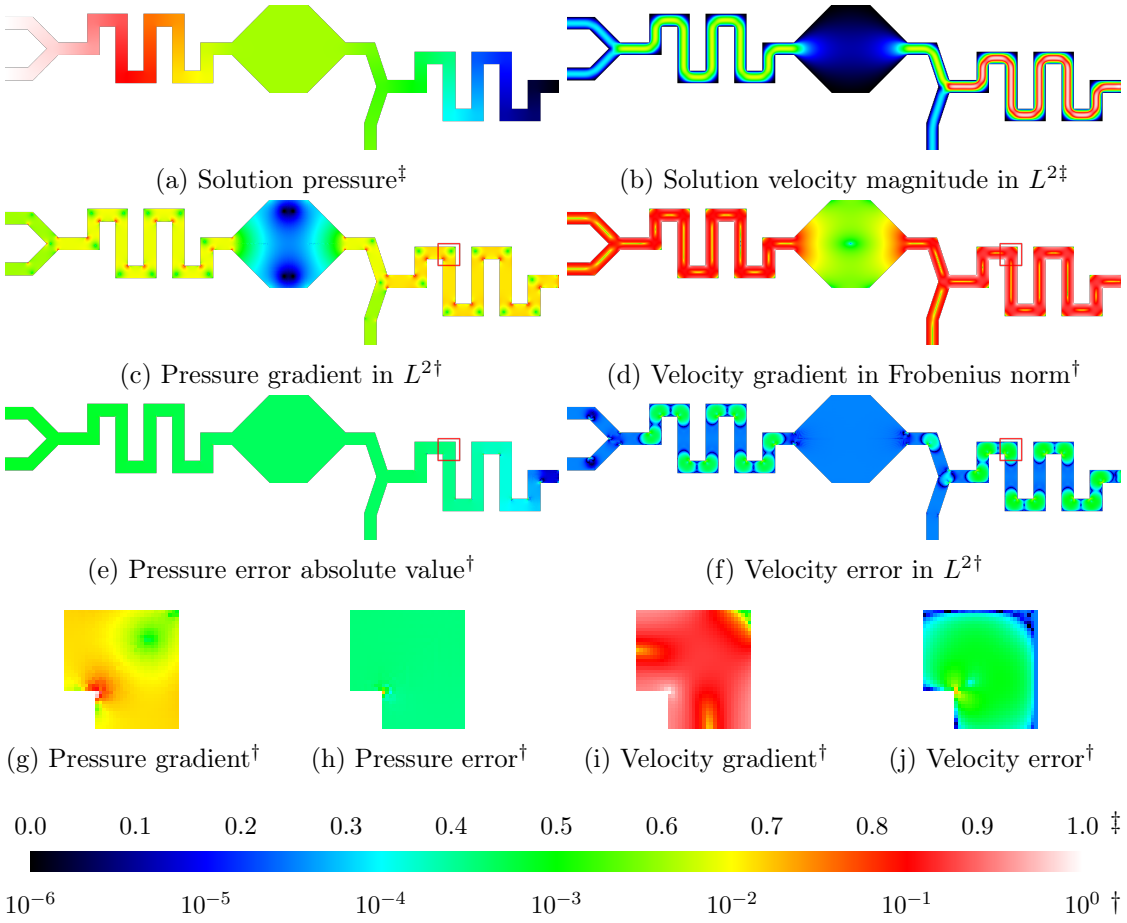


Figure 4.16: Solutions (first row), gradients (second row) and errors (third row) of test "wide" at resolution 16. The solutions are in linear scale (‡), and the gradients and errors are in logarithm scale (†). The solutions and gradients are normalized by the maximum values that are evaluated at the center of each element. To compute the errors, we compare the results with the solution at resolution 32. The errors are also normalized by the maximum velocity or pressure magnitude. The gradients and errors enclosed in the red rectangle are shown in the fourth row.

than one would predict based on the analysis of the algorithm in Section 4.5.2. Caching is certainly a major factor in the improved scaling, but this alone can only explain a factor of $O(\frac{r}{\ln r})$ improvement. Improved exploitation of parallelism (SIMD, threading) with larger problem sizes may also contribute to the improved performance. The near perfect scaling in 2D is quite surprising but very noticeable. The rather poor scaling in 3D is also quite

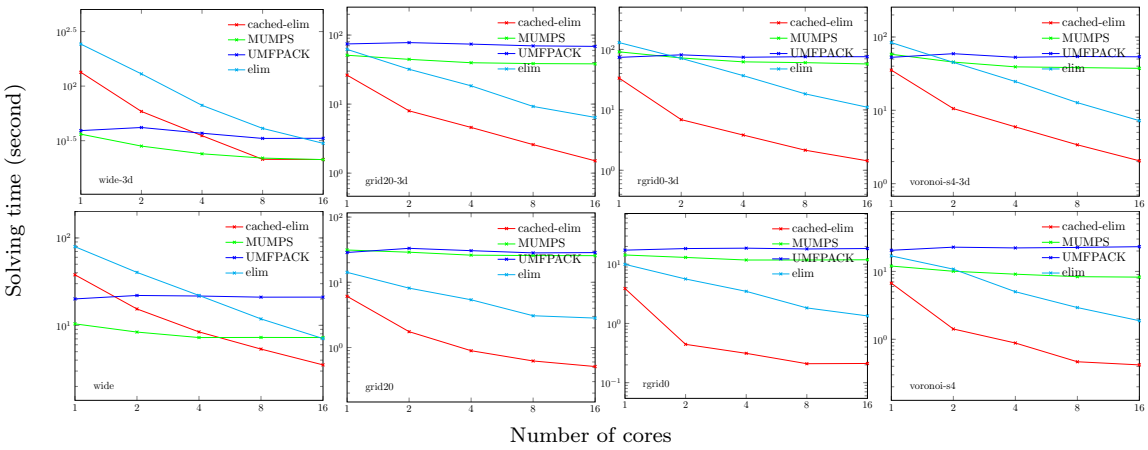


Figure 4.17: Solving time of our method ("cached-elim"), our method without caching ("elim"), MUMPS, and UMFPACK with different number of threads.

clear. The resolutions for which 3D is manageable are adequate to produce results accurate to 1-3 decimal places (depending on the variable evaluated and the norm chosen), which is likely adequate for prototyping purposes. At higher resolutions, the method is best used used as a coarse-grid solver for multigrid. The proposed method effectively performs an LU factorization; subsequent iterations require only the forward/backward substitution.

4.6.6 Comparison with general direct sparse solvers

In this section we compare our method with two direct solvers used for general sparse system: MUMPS[6, 4] and UMFPACK[41]. We use test cases "wide", "grid20", "rgrid0" and "voronoi-s4" in both 2D and 3D. We choose a resolution for each test so that there are around one million degrees of freedom. The statistics are listed in Table 4.3. We compare the methods based on runtime and also scaling with threads.

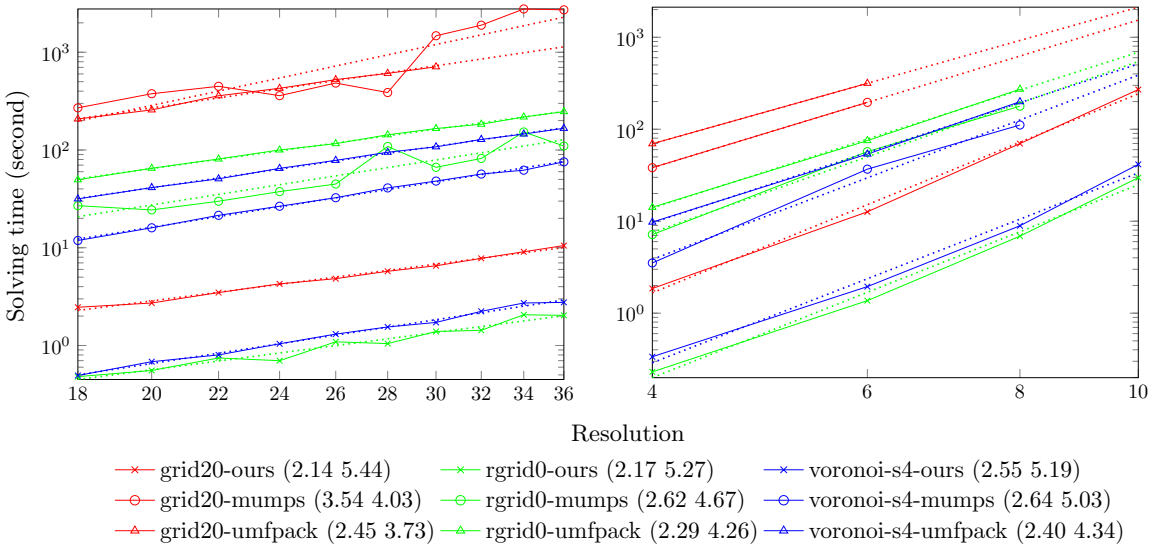


Figure 4.18: Scaling with refinement using our method (×), MUMPS (○), and UMFPACK (△). Different colors indicate the test cases ("grid20", "rgrid0", or "voronoi-s4"). We run these tests in both 2D (left) and 3D (right). Some data points are missing for MUMPS and UMFPACK because we run out of memory for those resolutions. The dotted lines are least square regression lines used to compute the increasing order. The orders for 2D (first entry) and 3D (second entry) are shown along with the corresponding legend items. An order of s indicates a complexity of $O(n^s)$ as discussed in Section 4.5.2.

MUMPS (version `5.2.1`) uses MPI for parallelism. Each instance calls sequential LAPACK routines. We call MUMPS so that it is aware of that our system is symmetric indefinite. (MUMPS also supports the shared memory parallelism through OpenMP. We tried this setup with one MPI instance and let the LAPACK implementation spawn threads. This was not as efficient as the MPI approach, so we do not show these results here.)

UMFPACK (version `5.7.8`) supports threading through a parallel LAPACK implementation. In the setup of UMFPACK, we specify the OpenMP threads number. The default control switches are used for the UMFPACK solver.

For both MUMPS and UMFPACK, the total solving time is measured for symbolic analysis, numeric factorization, and final numeric solve steps. For both solvers, we solve the

Name	Resolution	Total dofs	Blocks	Tasks	Avg dofs/block
wide	32	1.0 M	3036	16.0 K	314.4
grid20	8	1.5 M	20992	121.3 K	71.9
rgrid0	12	1.1 M	10702	27.2 K	103.9
voronoi-s4	16	1.3 M	9798	34.7 K	136.2
wide-3d	8	0.6 M	714	4.1 K	795.0
grid20-3d	4	1.5 M	9948	94.9 K	154.1
rgrid0-3d	6	1.9 M	5281	14.9 K	358.5
voronoi-s4-3d	6	1.3 M	3582	13.8 K	354.5

Table 4.3: Statistics for comparison tests with MUMPS, UMFPACK, and Krylov solvers. The resolutions are chosen so that the total number of dofs is around one million.

world space system, thus avoiding the extra transform passes on the solution and right hand side required for our method. The time required to set up the systems is excluded in all cases; only linear system solve time is being compared. Results are shown in Figure 4.17. We were surprised to observe that MUMPS and UMFPACK did not scale well with increasing core count. Our method is significantly faster on all tests except "wide." These improvements are generally the result of caching. The test case "wide" demonstrates a limitation of our method (See Section 4.7), though even on this example we eventually catch with increasing numbers of cores.

4.6.7 Comparison with iterative solver

The family of Krylov subspace based solvers is also commonly used for solving general sparse linear systems. In the case of symmetric indefinite matrices, MINRES is frequently used. The cost of Krylov solvers varies considerably, with system conditioning and the preconditioner effectiveness, and preconditioner cost being major factors. Rather than compare the cost of solving the system with particular choices of preconditioner, we instead compare the cost of solving the systems with our algorithm with the cost of performing *one iteration of unpreconditioned MINRES*. The cost of a MINRES iteration was estimated by

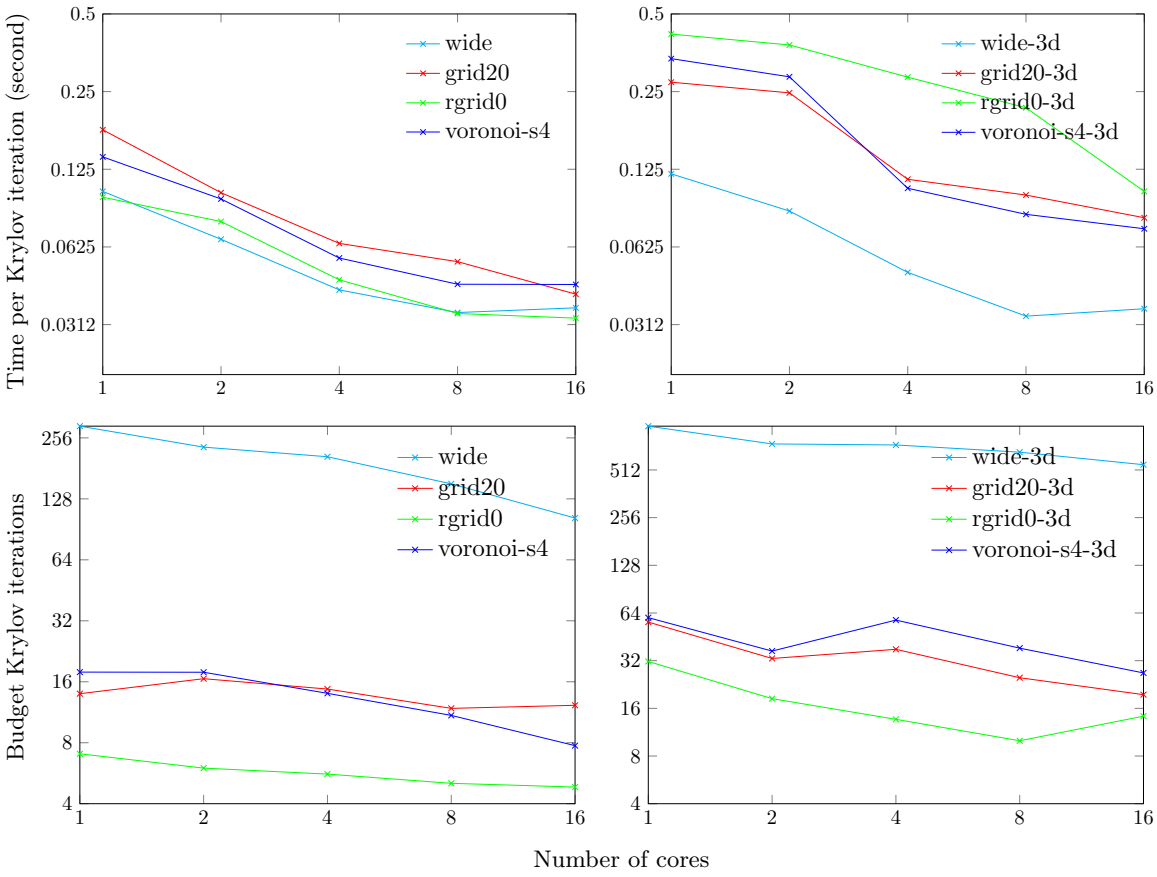


Figure 4.19: Comparison with Krylov solver. The solving time of our method is converted to number of Krylov iterations based on the time per iteration observed in our reference MINRES solver.

running 10 iterations of MINRES on the linear system and taking the average. Our MINRES implementation uses MKL-BLAS for the vector operations and MKL's sparse matrix-vector routines for the matrix-vector multiply. All of the MINRES linear algebra operations are threaded. We use the same set of tests and setup as in Section 4.6.6.

The test results are shown in Figure 4.19. With the expect of the "wide" test, our method converges for the price of about 20 (in 2D) or 60 (in 3D) unpreconditioned Krylov iterations for a Stokes systems with approximately 1M degrees of freedom. On the "wide" test, our cost is equivalent to a bit less than 300 (in 2D) or 1000 (in 3D) unpreconditioned

X0 = 0.000	X1 = 0.050	X2 = 0.100	X3 = 0.175
X4 = 0.231	X5 = 0.288	X6 = 0.344	X7 = 0.400
X8 = 0.450	X9 = 0.525	X10 = 0.600	X11 = 0.675
X12 = 0.725	X13 = 0.750	X14 = 0.825	X15 = 0.881
X16 = 0.938	X17 = 0.994	X18 = 1.050	X19 = 1.100
Y0 = 0.088	Y1 = 0.056	Y2 = 0.050	Y3 = 0.000
Y4 = -0.019	Y5 = -0.050	Y6 = -0.056	Y7 = -0.075
Y8 = -0.088	Y9 = -0.131	Y10 = -0.150	Y11 = -0.200

Table 4.4: Coordinates for the test case "wide".

Krylov iterations. Unpreconditioned MINRES would make little progress on these problems in 60 iterations and does not even converge in 1000; a preconditioner should always be used on these problems. Preconditioners compete for time with the rest of the Krylov iteration. While effective preconditioners exist which can converge in fewer than 60 iterations, doing so at the *cost* of 60 iterations would be quite difficult.

4.7 Limitations

Although the proposed method can in principle be applied to arbitrary fluid problems, in practice it is only efficient for fluid domains that have special geometrical properties. Irregularities are well-tolerated, provided they are local and do not lead to blocks with excessive numbers of degrees of freedom. Geometry without repetitions (either in the form of repeated components or straight pipes) produces no caching opportunities and thus no speedup over existing solvers. Though sensitive to the geometry, the method is relatively flexible with respect to PDE (Navier-Stokes, Poisson, heat equation) and other discretization choices (finite volume, finite difference; triangles vs. quads).

Due to the need for repetitions in geometry to be passed on to the linear solver, some amount of special-purpose discretization is required. Pipes must be broken into geometry blocks directly, and canonical components and blocks must still be identified. One is still free to use existing meshing libraries for non-pipe components and for geometry blocks. If a component was meshed with a general purpose library, a simple breadth-first-search traversal of the mesh elements could be used to automatically generate geometry blocks and canonical meshes. The meshing algorithm employed uses uniformly-sized elements, which do not accurately capture the large velocity and pressure gradients that occur in isolated parts of the fluid domain. There is no fundamental reason that an adaptive discretization could not be employed, and we leave this for future work.

Although our algorithm remains quite efficient at interesting resolutions, the algorithm scales poorly with block matrix size. For high enough resolutions, the algorithm will eventually become slower than many competing methods, especially iterative methods. The algorithm, however, only scales poorly with respect to feature width (channel width). It scales perfectly with respect to channel length. In particular, the algorithm would be very efficient on enormous devices (one could efficiently simulate an immensely complex chip), provided the features that comprise the device are all small.

4.8 Conclusions

We have demonstrated a method for discretizing thin and repetitive geometry into a linear algebra problem with repeated matrix blocks. We have also constructed an algorithm to efficiently solve matrices with this structure. The algorithm is very efficient

A0 = (-0.231,0.050)	A1 = (0.427,0.156)	A2 = (0.142,0.476)	A3 = (0.225,0.152)
A4 = (-0.048,0.173)	A5 = (0.033,-0.342)	A6 = (0.202,-0.363)	A7 = (-0.109,0.321)
A8 = (-0.332,0.218)	A9 = (-0.077,-0.045)	A10 = (-0.377,-0.104)	A11 = (0.440,-0.131)
A12 = (0.111,-0.004)	A13 = (0.096,0.303)	A14 = (-0.155,-0.268)	A15 = (-0.471,0.062)
A16 = (0.252,-0.127)	A17 = (0.007,-0.493)		
B0 = (0.349,0.314)	B1 = (0.417,-0.019)	B2 = (0.058,0.210)	B3 = (0.146,0.022)
B4 = (0.168,0.418)	B5 = (-0.115,0.447)	B6 = (-0.243,0.302)	B7 = (-0.151,0.100)
B8 = (-0.089,-0.124)	B9 = (0.278,-0.248)	B10 = (-0.403,-0.035)	B11 = (0.094,-0.435)
B12 = (-0.135,-0.348)	B13 = (-0.314,-0.260)	B14 = (0.085,-0.230)	

Table 4.5: Coordinates for the test cases "voronoi-s4" and "voronoi-s15".

up to moderate resolutions and is competitive with existing methods over this range of resolutions. The proposed method is the most efficient algorithm we are aware of for solving the Stokes flow equations on microfluidic chips. Discretizations of around 1M degrees of freedom can be solved in about one second on a workstation.

Future work There are many promising avenues for extending and improving the proposed method. The method may be coupled with multigrid to scale to higher resolutions or with other direct solvers to handle wide components more efficiently. Our implementation of the algorithm is quite simple and does not extend naturally to other problem domains besides our specific application, even though other domains exhibit similar geometrical features (e.g., plumbing). Our extension from 2D to 3D assumes that geometry is simply extruded, but we do not exploit this, such as by applying FFTs in this direction as is done in the FACR algorithm [163]. Such a method would improve the 3D scaling almost to the level of 2D scaling both in terms of memory and computational complexity.

Bibliography

[1] Thomas L Adam, K. Mani Chandy, and JR Dickson. A comparison of list schedules for parallel processing systems. *Communications of the ACM*, 17(12):685–690, 1974.

[2] H Adeli and O Kamal. Concurrent analysis of large structures–i. algorithms. *Computers & structures*, 42(3):413–424, 1992.

[3] Ishfaq Ahmad, Yu-Kwong Kwok, and Min-You Wu. Analysis, evaluation, and comparison of algorithms for scheduling task graphs on parallel processors. In *Proceedings Second International Symposium on Parallel Architectures, Algorithms, and Networks (I-SPAN'96)*, pages 207–213. IEEE, 1996.

[4] P. R. Amestoy, A. Guermouche, J.-Y. L'Excellent, and S. Pralet. Hybrid scheduling for the parallel solution of linear systems. *Parallel Computing*, 32(2):136–156, 2006.

[5] Patrick R Amestoy, Iain S Duff, and J-Y L'excellent. Multifrontal parallel distributed symmetric and unsymmetric solvers. *Computer methods in applied mechanics and engineering*, 184(2-4):501–520, 2000.

[6] Patrick R Amestoy, Iain S Duff, Jean-Yves L'Excellent, and Jacko Koster. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM Journal on Matrix Analysis and Applications*, 23(1):15–41, 2001.

[7] Patrick R Amestoy and Lain S Duff. Vectorization of a multiprocessor multifrontal code. *The International Journal of Supercomputing Applications*, 3(3):41–59, 1989.

[8] Mihai Anitescu and Florian A Potra. Formulating dynamic multi-rigid-body contact problems with friction as solvable linear complementarity problems. *Nonlinear Dynamics*, 14(3):231–247, 1997.

[9] Peter Arbenz, Markus Hegland, et al. The stable parallel solution of general narrow banded linear systems. 1997.

[10] Cleve Ashcraft and Roger Grimes. The influence of relaxed supernode partitions on the multifrontal method. *ACM Transactions on Mathematical Software (TOMS)*, 15(4):291–309, 1989.

[11] Diego C. Assêncio and Joseph M. Teran. A second order virtual node algorithm for stokes flow problems with interfacial forces, discontinuous material parameters and irregular domains. *J. Comput. Phys.*, 250(1):77–105, 2013.

[12] Timothy J Baker. Automatic mesh generation for complex three-dimensional regions using a constrained delaunay triangulation. *Engineering with Computers*, 5(3-4):161–175, 1989.

[13] David Baraff. Coping with friction for non-penetrating rigid body simulation. *ACM SIGGRAPH computer graphics*, 25(4):31–41, 1991.

[14] David Baraff and Andrew Witkin. Partitioned dynamics. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA ROBOTICS INST, 1997.

[15] David Baraff and Andrew Witkin. Large steps in cloth simulation. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '98, pages 43–54, New York, NY, USA, 1998. ACM.

[16] SG. Bardenhagen and EM. Kober. The generalized interpolation material point method. *Comp Mod in Eng and Sci*, 5(6):477–496, 2004.

[17] Michel Bercovier and Olivier Pironneau. Error estimates for finite element method solution of the stokes problem in the primitive variables. *Numerische Mathematik*, 33(2):211–224, 1979.

[18] Florence Bertails-Descoubes, Florent Cadoux, Gilles Daviet, and Vincent Acary. A nonsmooth newton solver for capturing exact coulomb friction in fiber assemblies. *ACM Transactions on Graphics (TOG)*, 30(1):6, 2011.

[19] Daniele Boffi, Franco Brezzi, Michel Fortin, et al. *Mixed finite element methods and applications*, volume 44. Springer, 2013.

[20] Daniele Boffi, Nicola Cavallini, Francesca Gardini, and Lucia Gastaldi. Local mass conservation of stokes finite elements. *Journal of scientific computing*, 52(2):383–400, 2012.

[21] Jeff Bolz, Ian Farmer, Eitan Grinspun, and Peter Schröoder. Sparse matrix solvers on the gpu: conjugate gradients and multigrid. In *ACM transactions on graphics (TOG)*, volume 22, pages 917–924. ACM, 2003.

[22] H Borouchaki, F Hecht, E Saltel, and PL George. Reasonably efficient delaunay based mesh generator in 3 dimensions. In *Proceedings 4th International Meshing Roundtable*, pages 3–14, 1995.

[23] L. Boyd and R. Bridson. Multiflip for energetic two-phase fluid simulation. *ACM Transactions on Graphics (TOG)*, 31(2):16, 2012.

[24] J. Brackbill. The ringing instability in particle-in-cell calculations of low-speed flow. *J Comp Phys*, 75(2):469–492, 1988.

[25] J. Brackbill, D. Kothe, and H. Ruppel. FLIP: A low-dissipation, PIC method for fluid flow. *Comp Phys Comm*, 48:25–38, 1988.

[26] J. Brackbill and H. Ruppel. FLIP: A method for adaptively zoned, particle-in-cell calculations of fluid flows in two dimensions. *J Comp Phys*, 65:314–343, 1986.

[27] J.U. Brackbill. On modelling angular momentum and vorticity in compressible fluid flow. *Comp Phys Comm*, 47(1):1–16, 1987.

[28] R. Bridson. *Fluid simulation for computer graphics*. Taylor & Francis, 2008.

[29] R. Bridson, S. Marino, and R. Fedkiw. Simulation of clothing with folds and wrinkles. In *Proc ACM SIGGRAPH/Eurograph Symp Comp Anim*, SCA '03, pages 28–36, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[30] Robert Bridson, Ronald Fedkiw, and John Anderson. Robust treatment of collisions, contact and friction for cloth animation. *ACM Transactions on Graphics (ToG)*, 21(3):594–603, 2002.

[31] D. Burgess, D. Sulsky, and J. Brackbill. Mass matrix formulation of the FLIP particle-in-cell method. *J Comp Phys*, 103:1–15, 1992.

[32] Billy L Buzbee, Gene H Golub, and Clair W Nielson. On direct methods for solving poisson's equations. *SIAM Journal on Numerical analysis*, 7(4):627–656, 1970.

[33] Soo-Won Chae and Jung-Hwan Jeong. Unstructured surface meshing using operators. In *Proceedings, 6th International Meshing Roundtable*, pages 281–291, 1997.

[34] CK Chang, CC Lai, and Chen-Kuei Chung. Numerical analysis and experiments of capillarity-driven microfluid chip. In *2011 6th IEEE International Conference on Nano/Micro Engineered and Molecular Systems*, pages 1032–1035. IEEE, 2011.

[35] Salvatore Cito, Jordi Pallares, Alexandre Fabregat, and Ioanis Katakis. Numerical simulation of wall mass transfer rates in capillary-driven flow in microchannels. *International Communications in Heat and Mass Transfer*, 39(8):1066–1072, 2012.

[36] C Cleveland Ashcraft, Roger G Grimes, John G Lewis, Barry W Peyton, Horst D Simon, and Petter E Bjørstad. Progress in sparse matrix methods for large linear systems on vector supercomputers. *The International Journal of Supercomputing Applications*, 1(4):10–30, 1987.

[37] Eulalie Coevoet, Adrien Escande, and Christian Duriez. Optimization-based inverse model of soft robots with contact handling. *IEEE Robotics and Automation Letters*, 2(3):1413–1419, 2017.

[38] Richard W Cottle, Jong-Shi Pang, and Richard E Stone. *The Linear Complementarity Problem*. Academic Press, 1992.

[39] Hadrien Courtecuisse, Jérémie Allard, Pierre Kerfriden, Stéphane PA Bordas, Stéphane Cotin, and Christian Duriez. Real-time simulation of contact and cutting of heterogeneous soft-tissues. *Medical image analysis*, 18(2):394–410, 2014.

[40] G. Daviet and F. Bertails-Descoubes. A semi-implicit material point method for the continuum simulation of granular materials. *ACM Trans Graph*, 35(4), jul 2016.

[41] Timothy A Davis. Algorithm 832: Umfpack v4. 3—an unsymmetric-pattern multifrontal method. *ACM Transactions on Mathematical Software (TOMS)*, 30(2):196–199, 2004.

[42] Timothy A Davis. A column pre-ordering strategy for the unsymmetric-pattern multifrontal method. *ACM Transactions on Mathematical Software (TOMS)*, 30(2):165–195, 2004.

[43] Timothy A Davis and Iain S Duff. An unsymmetric-pattern multifrontal method for sparse lu factorization. *SIAM Journal on Matrix Analysis and Applications*, 18(1):140–158, 1997.

[44] Timothy A Davis, John R Gilbert, Stefan I Larimore, and Esmond G Ng. Algorithm 836: Colamd, a column approximate minimum degree ordering algorithm. *ACM Transactions on Mathematical Software (TOMS)*, 30(3):377–380, 2004.

[45] Timothy A Davis, Sivasankaran Rajamanickam, and Wissam M Sid-Lakhdar. A survey of direct methods for sparse linear systems. *Acta Numerica*, 25:383–566, 2016.

[46] Dino Di Carlo. Inertial microfluidics. *Lab on a Chip*, 9(21):3038–3046, 2009.

[47] Ounan Ding and Schroeder Craig. Penalty force for coupling materials with coulomb friction. *IEEE transactions on visualization and computer graphics*, 2019.

[48] Victorita Dolean, Pierre Jolivet, and Frâdâric Nataf. *An introduction to domain decomposition methods: algorithms, theory, and parallel implementation*, volume 144. SIAM, 2015.

[49] Evan Drumwright. A fast and stable penalty method for rigid body simulation. *IEEE transactions on visualization and computer graphics*, 14(1):231–240, 2008.

[50] Iain S Duff. Parallel implementation of multifrontal schemes. *Parallel computing*, 3(3):193–204, 1986.

[51] Iain S Duff. The parallel solution of sparse linear equations. In *International Conference on Parallel Processing*, pages 18–24. Springer, 1986.

[52] Iain S Duff and John K Reid. The multifrontal solution of indefinite sparse symmetric linear. *ACM Transactions on Mathematical Software (TOMS)*, 9(3):302–325, 1983.

[53] E. Edwards and R. Bridson. A high-order accurate particle-in-cell method. *Int J Numer Meth Eng*, 90:1073–1088, 2012.

[54] Alexandre Ern and Jean-Luc Guermond. *Theory and practice of finite elements*, volume 159. Springer Science & Business Media, 2013.

[55] Charbel Farhat, Edward Wilson, and Graham Powell. Solution of finite element systems on concurrent processing computers. *Engineering with Computers*, 2(3):157–165, 1987.

[56] Susan Fisher and Ming C Lin. Fast penetration depth estimation for elastic bodies using deformed distance fields. In *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, volume 1, pages 330–336. IEEE, 2001.

[57] Chuyuan Fu, Qi Guo, Theodore Gast, Chenfanfu Jiang, and Joseph Teran. A polynomial particle-in-cell method. *ACM Transactions on Graphics (TOG)*, 36(6):222, 2017.

[58] Walter Gander and Gene H Golub. Cyclic reduction–history and applications. *Scientific computing (Hong Kong, 1997)*, pages 73–85, 1997.

[59] Ming Gao, Andre Pradhana, Xuchen Han, Qi Guo, Grant Kot, Eftychios Sifakis, and Chenfanfu Jiang. Animating fluid sediment mixture in particle-laden flows. *ACM Transactions on Graphics (TOG)*, 37(4):149, 2018.

[60] Theodore F Gast, Craig Schroeder, Alexey Stomakhin, Chenfanfu Jiang, and Joseph M Teran. Optimization integrator for large time steps. *IEEE transactions on visualization and computer graphics*, 21(10):1103–1115, 2015.

[61] Alan George. Nested dissection of a regular finite element mesh. *SIAM Journal on Numerical Analysis*, 10(2):345–363, 1973.

[62] Alan George and Joseph WH Liu. The evolution of the minimum degree ordering algorithm. *Siam review*, 31(1):1–19, 1989.

[63] Paul Louis George, Frédéric Hecht, and Éric Saltel. Automatic mesh generator with specified boundary. *Computer methods in applied mechanics and engineering*, 92(3):269–288, 1991.

[64] Apostolos Gerasoulis and Tao Yang. A comparison of clustering heuristics for scheduling directed acyclic graphs on multiprocessors. *Journal of Parallel and Distributed Computing*, 16(4):276–291, 1992.

[65] UKNG Ghia, Kirti N Ghia, and CT Shin. High-re solutions for incompressible flow using the navier-stokes equations and a multigrid method. *Journal of computational physics*, 48(3):387–411, 1982.

[66] John R Gilbert, Cleve Moler, and Robert Schreiber. Sparse matrices in matlab: Design and implementation. *SIAM Journal on Matrix Analysis and Applications*, 13(1):333–356, 1992.

[67] Thomas Glatzel, Christian Litterst, Claudio Cupelli, Timo Lindemann, Christian Moosmann, Remigius Niekrawietz, Wolfgang Streule, Roland Zengerle, and Peter Koltay. Computational fluid dynamics (cfd) software tools for microfluidic applications–a case study. *Computers & Fluids*, 37(3):218–235, 2008.

[68] Nils Gleichmann, Daniéll Malsch, Peter Horbert, and Thomas Henkel. Toward microfluidic design automation: a new system simulation toolkit for the in silico evaluation of droplet-based lab-on-a-chip systems. *Microfluidics and Nanofluidics*, 18(5-6):1095–1105, 2015.

[69] Nils Gleichmann, Daniell Malsch, Mark Kielpinski, Wilhelm Rossak, Günter Mayer, and Thomas Henkel. Toolkit for computational fluidic simulation and interactive parametrization of segmented flow based fluidic networks. *Chemical Engineering Journal*, 135:S210–S218, 2008.

[70] Gene H Golub and James M Ortega. *Scientific computing: an introduction with parallel computing*. Elsevier, 2014.

[71] Gene H Golub and Charles F Van Loan. *Matrix computations*, volume 3. JHU Press, 2012.

[72] Y. Grigoryev, V. Vshivkov, and M. Fedoruk. *Numerical Particle-In-Cell Methods: Theory and Applications*. Walter de Gruyter, 2002.

[73] Andreas Grimmer, Medina Hamidović, Werner Haselmayr, and Robert Wille. Advanced simulation of droplet microfluidics. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 15(3):26, 2019.

[74] Andreas Grimmer and Robert Wille. *Designing Droplet Microfluidic Networks: A Toolbox for Designers*. Springer, 2019.

[75] C. Gritton. *Ringing Instabilities in Particle Methods*. PhD thesis, The University of Utah, 2014.

[76] Eran Guendelman, Robert Bridson, and Ronald Fedkiw. Nonconvex rigid bodies with stacking. In *ACM Transactions on Graphics (TOG)*, volume 22, pages 871–878. ACM, 2003.

[77] Gundolf Haase, Manfred Liebmann, Craig C Douglas, and Gernot Plank. A parallel algebraic multigrid solver on graphics processing units. In *High performance computing and applications*, pages 38–47. Springer, 2010.

[78] Chad C Hammerquist and John A Nairn. A new method for material point method particle updates that reduces noise and enhances stability. *Computer Methods in Applied Mechanics and Engineering*, 318:724–738, 2017.

[79] F. Harlow. The particle-in-cell method for numerical solution of problems in fluid dynamics. *Meth Comp Phys*, 3:319–343, 1964.

[80] Shoichi Hasegawa, Nobuaki Fujii, Katsuhito Akahane, Yasuharu Koike, and Makoto Sato. Real-time rigid body simulation for haptic interactions based on contact volume of polygonal objects. *Transactions of the Society of Instrument and Control Engineers*, 40(2):122–131, 2004.

[81] Michael T Heath. *Scientific computing*. McGraw-Hill New York, 2002.

[82] Don Heller. Some aspects of the cyclic reduction algorithm for block tridiagonal linear systems. *SIAM Journal on Numerical Analysis*, 13(4):484–496, 1976.

[83] Magnus Rudolph Hestenes and Eduard Stiefel. *Methods of conjugate gradients for solving linear systems*, volume 49. NBS Washington, DC, 1952.

[84] Duong A Hoang, Volkert van Steijn, Luis M Portela, Michiel T Kreutzer, and Chris R Kleijn. Benchmark numerical simulations of segmented two-phase flows in microchannels using the volume of fluid method. *Computers & Fluids*, 86:28–36, 2013.

[85] Roger W Hockney. A fast direct solution of poisson's equation using fourier analysis. *Journal of the ACM (JACM)*, 12(1):95–113, 1965.

[86] Paul Hood. Frontal solution program for unsymmetric matrices. *International Journal for Numerical Methods in Engineering*, 10(2):379–399, 1976.

[87] EN Houstis and LR Rice. Three-dimensional boundary-constrained triangulations. *Artificial Intelligence, Expert Systems & Symbolic Computing*, page 215, 1992.

[88] Yuanming Hu, Yu Fang, Ziheng Ge, Ziyin Qu, Yixin Zhu, Andre Pradhana, and Chenfanfu Jiang. A moving least squares material point method with displacement discontinuity and two-way rigid body coupling. *ACM Transactions on Graphics (TOG)*, 37(4):150, 2018.

[89] Kai Hwang. Advanced computer architecture: Parallelism, scalability, programmability:, mcgraw-hill book co. international edition. 1993.

[90] Yoonkwon Hwang, Eiichi Inohira, Atsushi Konno, and Masaru Uchiyama. An order n dynamic simulator for a humanoid robot with a virtual spring-damper contact model. In *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*, volume 1, pages 31–36. IEEE, 2003.

[91] Bruce M Irons. A frontal solution program for finite element analysis. *International Journal for Numerical Methods in Engineering*, 2(1):5–32, 1970.

[92] Johan Jansson and Joris SM Vergeest. Combining deformable-and rigid-body mechanics simulation. *The Visual Computer*, 19(5):280–290, 2003.

[93] 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, 2015.

[94] C. Jiang, C. Schroeder, and J. Teran. An angular momentum conserving affine-particle-in-cell method. *J Comp Phys*, 338:137–164, 2017.

[95] Barry Joe. GeompackâĂŤa software package for the generation of meshes using geometric algorithms. *Advances in Engineering Software and Workstations*, 13(5-6):325–331, 1991.

[96] Shmuel Kaniel. Estimates for some computational techniques in linear algebra. *Mathematics of Computation*, 20(95):369–378, 1966.

[97] Danny M Kaufman, Shinjiro Sueda, Doug L James, and Dinesh K Pai. Staggered projections for frictional contact in multibody systems. *ACM Transactions on Graphics (TOG)*, 27(5):164, 2008.

[98] AI Khan and BHV Topping. Parallel finite element analysis using jacobi-conditioned conjugate gradient algorithm. *Advances in Engineering Software*, 25(2-3):309–319, 1996.

[99] Junggon Kim and Nancy S Pollard. Fast simulation of skeleton-driven deformable body characters. *ACM Transactions on Graphics (TOG)*, 30(5):121, 2011.

[100] G. Klár, T. Gast, A. Pradhana, C. Fu, C. Schroeder, C. Jiang, and T. Teran. Drucker-prager elastoplasticity for sand animation. *ACM Transactions on Graphics (SIGGRAPH 2016).*, 2016.

[101] Martin Kronbichler, Ababacar Diagne, and Hanna Holmgren. A fast massively parallel two-phase flow solver for microfluidic chip simulation. *The International Journal of High Performance Computing Applications*, 32(2):266–287, 2018.

[102] François Labelle and Jonathan Richard Shewchuk. Isosurface stuffing: fast tetrahedral meshes with good dihedral angles. In *ACM Transactions on Graphics (TOG)*, volume 26, page 57. ACM, 2007.

[103] Jules J Lambiotte Jr and Robert G Voigt. The solution of tridiagonal linear systems on the cdc star 100 computer. *ACM Transactions on Mathematical Software (TOMS)*, 1(4):308–329, 1975.

[104] A. Langdon. Effects of spatial grid simulation in plasmas. *J Comp Phys*, 6(2):247–267, 1970.

[105] Charles L Lawson. Software for c1 surface interpolation. In *Mathematical software*, pages 161–194. Elsevier, 1977.

[106] Julien Lenoir and Sylvere Fonteneau. Mixing deformable and rigid-body mechanics simulation. In *Computer Graphics International, 2004. Proceedings*, pages 327–334. IEEE, 2004.

[107] Haixiang Liu, Nathan Mitchell, Mridul Aanjaneya, and Eftychios Sifakis. A scalable schur-complement fluids solver for heterogeneous compute platforms. *ACM Transactions on Graphics (TOG)*, 35(6):201, 2016.

[108] Joseph WH Liu. The minimum degree ordering with constraints. *SIAM Journal on Scientific and Statistical Computing*, 10(6):1136–1145, 1989.

[109] Joseph WH Liu. The multifrontal method for sparse matrix solution: Theory and practice. *SIAM review*, 34(1):82–109, 1992.

[110] JW Liu and A George. Computer solution of large sparse positive definite systems. *Prentice-Hall, Inc. Englewood Cliffs. NJ*, 7632:1981, 1981.

[111] Sai Huen Lo. Volume discretization into tetrahedra–ii. 3d triangulation by advancing front approach. *Computers & Structures*, 39(5):501–511, 1991.

[112] SH Lo. Volume discretization into tetrahedra–i. verification and orientation of boundary surfaces. *Computers & Structures*, 39(5):493–500, 1991.

[113] Rainald Löhner. Progress in grid generation via the advancing front technique. *Engineering with computers*, 12(3-4):186–210, 1996.

[114] Rainald Lohner, Paresh Parikh, and Clyde Gumbert. Interactive generation of unstructured grids for three dimensional problems. 1988.

[115] E. Love and D. Sulsky. An unconditionally stable, energy-momentum consistent implementation of the the material point method. *Comp Meth App Mech Eng*, 195:3903–3925, 2006.

[116] E. Love and D. L. Sulsky. An energy-consistent material-point method for dynamic finite deformation plasticity. *Int J Numer Meth Eng*, 65(10):1608–1638, 2006.

[117] Wan Shi Low, Nahrizul Adib Kadri, Wan Abas, et al. Computational fluid dynamics modelling of microfluidic channel for dielectrophoretic biomems application. *The Scientific World Journal*, 2014, 2014.

[118] Xin Lv, Yong Zhao, XY Huang, GH Xia, and XH Su. A matrix-free implicit unstructured multigrid finite volume method for simulating structural dynamics and fluid–structure interaction. *Journal of Computational Physics*, 225(1):120–144, 2007.

[119] Yvon Maday, Cathy Mavriplis, and Anthony T Patera. Nonconforming mortar element methods-application to spectral discretizations. In *Domain Decomposition Methods*, pages 392–418, 1989.

[120] David L Marcum and Nigel P Weatherill. Unstructured grid generation using iterative point insertion and local reconnection. *AIAA journal*, 33(9):1619–1625, 1995.

[121] Duane W Marhefka and David E Orin. Simulation of contact using a nonlinear damping model. In *Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on*, volume 2, pages 1662–1668. IEEE, 1996.

[122] Dimitri J Mavriplis. Multigrid solution of the two-dimensional euler equations on unstructured triangular meshes. *AIAA journal*, 26(7):824–831, 1988.

[123] Dimitri J Mavriplis and Antony Jameson. Multigrid solution of the navier-stokes equations on triangular meshes. *AIAA journal*, 28(8):1415–1425, 1990.

[124] Aleka McAdams, Eftychios Sifakis, and Joseph Teran. A parallel multigrid poisson solver for fluids simulation on large grids. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 65–74. Eurographics Association, 2010.

[125] Michael McKenna and David Zeltzer. Dynamic simulation of autonomous legged locomotion. In *ACM SIGGRAPH Computer Graphics*, volume 24, pages 29–38. ACM, 1990.

[126] Neil Molino, Robert Bridson, Joseph Teran, and Ronald Fedkiw. A crystalline, red green strategy for meshing highly deformable objects with tetrahedra. In *International Meshing Roundtable*, pages 103–114. Citeseer, 2003.

[127] Matthew Moore and Jane Wilhelms. Collision detection and response for computer animation. In *ACM Siggraph Computer Graphics*, volume 22, pages 289–298. ACM, 1988.

[128] John A Nairn. Numerical simulation of orthogonal cutting using the material point method. *Engineering Fracture Mechanics*, 149:262–275, 2015.

[129] R. Narain, A. Golas, and M. Lin. Free-flowing granular materials with two-way solid coupling. *ACM Trans Graph*, 29(6):173:1–173:10, 2010.

[130] Yen Ting Ng, Chohong Min, and Frédéric Gibou. An efficient fluid–solid coupling algorithm for single-phase flows. *Journal of Computational Physics*, 228(23):8807–8829, 2009.

[131] Vinh Phu Nguyen, Chi Thanh Nguyen, Timon Rabczuk, and Sundararajan Natarajan. On a family of convected particle domain interpolations in the material point method. *Finite Elements in Analysis and Design*, 126:50–64, 2017.

[132] J. Nocedal and S.J. Wright. *Numerical Optimization*. Springer series in operations research and financial engineering. Springer, 2006.

[133] Dietrich Nowottny. Quadrilateral mesh generation via geometrically optimized domain decomposition. In *Proceedings, 6th International Meshing Roundtable*, pages 309–320, 1997.

[134] James F O'Brien, Victor B Zordan, and Jessica K Hodgins. Combining active and passive simulations for secondary motion. *IEEE Computer Graphics and Applications*, 20(4):86–96, 2000.

[135] H. Okuda. Nonphysical noises and instabilities in plasma simulation due to a spatial grid. *J Comp Phys*, 10(3):475–486, 1972.

[136] Miguel A Otaduy, Rasmus Tamstorf, Denis Steinemann, and Markus Gross. Implicit contact handling for deformable objects. In *Computer Graphics Forum*, volume 28, pages 559–568. Wiley Online Library, 2009.

[137] Steven J Owen. A survey of unstructured mesh generation technology. In *IMR*, pages 239–267, 1998.

[138] Christopher C Paige and Michael A Saunders. Solution of sparse indefinite systems of linear equations. *SIAM journal on numerical analysis*, 12(4):617–629, 1975.

[139] William H Press. *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press, 2007.

[140] Xavier Provot. Collision and self-collision handling in cloth model dedicated to design garments. In *Computer Animation and Simulation '97*, pages 177–189. Springer, 1997.

[141] Janusz S Przemieniecki. Matrix structural analysis of substructures. *AIAA Journal*, 1(1):138–147, 1963.

[142] Alfio Quarteroni and Alberto Valli. *Domain decomposition methods for partial differential equations*. Number BOOK. Oxford University Press, 1999.

[143] Stefano Rebay. Efficient unstructured mesh generation by means of delaunay triangulation and bowyer-watson algorithm. *Journal of computational physics*, 106(1):125–138, 1993.

[144] Patrick J Roache. Code verification by the method of manufactured solutions. *Journal of fluids engineering*, 124(1):4–10, 2002.

[145] JL ROGERS JR and J SOBIESZCZANSKI-SOBIESKI. Exploiting parallel computing with limited program changes using a network of microcomputers. *Advances in engineering software*, 9(1):29–33, 1987.

[146] Christopher J Roy. Review of code and solution verification procedures for computational simulation. *Journal of Computational Physics*, 205(1):131–156, 2005.

[147] Christopher John Roy, CC Nelson, TM Smith, and CC Ober. Verification of euler/navier–stokes codes using the method of manufactured solutions. *International Journal for Numerical Methods in Fluids*, 44(6):599–620, 2004.

[148] Youcef Saad and Martin H Schultz. Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on scientific and statistical computing*, 7(3):856–869, 1986.

[149] A Sadeghirad, Rebecca M Brannon, and J Burghardt. A convected particle domain interpolation technique to extend applicability of the material point method for problems involving massive deformations. *International Journal for numerical methods in Engineering*, 86(12):1435–1456, 2011.

[150] C. Schroeder, A. Stomakhin, R. Howes, and J. Teran. A second order virtual node algorithm for navier-stokes flow problems with interfacial forces and discontinuous material properties. *J. Comput. Phys.*, 265:221–245, 2014.

[151] Mark S Shephard and Marcel K Georges. Automatic three-dimensional mesh generation by the finite octree technique. *International Journal for Numerical methods in engineering*, 32(4):709–749, 1991.

[152] Jonathan Richard Shewchuk. Triangle: Engineering a 2d quality mesh generator and delaunay triangulator. In *Workshop on Applied Computational Geometry*, pages 203–222. Springer, 1996.

[153] Jonathan Richard Shewchuk et al. An introduction to the conjugate gradient method without the agonizing pain, 1994.

[154] T. Shinar, C. Schroeder, and R. Fedkiw. Two-way coupling of rigid and deformable bodies. In *SCA '08: Proceedings of the 2008 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 95–103, 2008.

[155] E. Sifakis, T. Shinar, G. Irving, and R. Fedkiw. Hybrid simulation of deformable solids. In *Proc ACM SIGGRAPH/Eurograph Symp Comp Anim*, pages 81–90, 2007.

[156] Barry Smith, Petter Bjorstad, William D Gropp, and William Gropp. *Domain decomposition: parallel multilevel methods for elliptic partial differential equations*. Cambridge university press, 2004.

[157] Michael Steffen, Robert M Kirby, and Martin Berzins. Analysis and reduction of quadrature errors in the material point method (MPM). *Int J Numer Meth Eng*, 76(6):922–948, 2008.

[158] David E Stewart and Jeffrey C Trinkle. An implicit time-stepping scheme for rigid body dynamics with inelastic collisions and coulomb friction. *International Journal for Numerical Methods in Engineering*, 39(15):2673–2691, 1996.

[159] A. Stomakhin, R. Howes, C. Schroeder, and J.M. Teran. Energetically consistent invertible elasticity. In *Proc. Symp. Comp. Anim.*, pages 25–32, 2012.

[160] A. Stomakhin, C. Schroeder, L. Chai, J. Teran, and A. Selle. A material point method for snow simulation. In *ACM Transactions on Graphics (SIGGRAPH 2013)*, pages 102:1–10, 2013.

[161] D. Sulsky, Z. Chen, and H. Schreyer. A particle method for history-dependent materials. *Comp Meth in App Mech Eng*, 118(1):179–196, 1994.

[162] D. Sulsky, S. Zhou, and H. Schreyer. Application of a particle-in-cell method to solid mechanics. *Comp Phys Comm*, 87(1):236–252, 1995.

[163] Paul N Swarztrauber. The methods of cyclic reduction, fourier analysis and the facr algorithm for the discrete solution of poisson's equation on a rectangle. *Siam Review*, 19(3):490–501, 1977.

[164] Jeffrey A Talbert and Alan R Parkinson. Development of an automatic, two-dimensional finite element mesh generator using quadrilateral elements and bezier

curve boundary definition. *International Journal for Numerical Methods in Engineering*, 29(7):1551–1567, 1990.

[165] Andre Pradhana Tampubolon, Theodore Gast, Gergely Klár, Chuyuan Fu, Joseph Teran, Chenfanfu Jiang, and Ken Museth. Multi-species simulation of porous sand and water mixtures. *ACM Transactions on Graphics (SIGGRAPH 2017)*, 36(4):105, 2017.

[166] Min Tang, Dinesh Manocha, Miguel A Otaduy, and Ruofeng Tong. Continuous penalty forces. *ACM Trans. Graph.*, 31(4):107–1, 2012.

[167] Demetri Terzopoulos, John Platt, Alan Barr, and Kurt Fleischer. Elastically deformable models. *ACM Siggraph Computer Graphics*, 21(4):205–214, 1987.

[168] Abraham Van der Sluis and Henk A van der Vorst. The rate of convergence of conjugate gradients. *Numerische Mathematik*, 48(5):543–560, 1986.

[169] P. Wallstedt and J. Guilkey. Improved velocity projection for the material point method. *Comp Mod in Eng and Sci*, 19(3):223, 2007.

[170] Junchao Wang, Philip Brisk, and William H Grover. Random design of microfluidics. *Lab Chip*, 16(21):4212–4219, 10 2016.

[171] Junchao Wang, Victor GJ Rodgers, Philip Brisk, and William H Grover. Instantaneous simulation of fluids and particles in complex microfluidic devices. *PloS one*, 12(12):e0189429, 2017.

[172] David F Watson. Computing the n-dimensional delaunay tessellation with application to voronoi polytopes. *The computer journal*, 24(2):167–172, 1981.

[173] Nigel P Weatherill and Oubay Hassan. Efficient three-dimensional delaunay triangulation with automatic point creation and imposed boundary constraints. *International Journal for Numerical Methods in Engineering*, 37(12):2005–2039, 1994.

[174] Pieter Wesseling. Introduction to multigrid methods. Technical report, Institute for Computer Applications in Science and Engineering Hampton VA, 1995.

[175] Gabriel Wittum. Multi-grid methods for stokes and navier-stokes equations. *Numerische Mathematik*, 54(5):543–563, 1989.

[176] Barbara I Wohlmuth. A mortar finite element method using dual spaces for the lagrange multiplier. *SIAM journal on numerical analysis*, 38(3):989–1012, 2000.

[177] Martin Wörner. Numerical modeling of multiphase flows in microfluidics and micro process engineering: a review of methods and applications. *Microfluidics and nanofluidics*, 12(6):841–886, 2012.

[178] Jianlin Xia, Shivkumar Chandrasekaran, Ming Gu, and Xiaoye S Li. Superfast multifrontal method for large structured linear systems of equations. *SIAM Journal on Matrix Analysis and Applications*, 31(3):1382–1411, 2009.

[179] Hongyi Xu, Yili Zhao, and Jernej Barbič. Implicit multibody penalty-based distributed contact. *IEEE transactions on visualization and computer graphics*, 20(9):1266–1279, 2014.

[180] T. Yabe, F. Xiao, and T. Utsumi. The constrained interpolation profile method for multiphase analysis. *J Comp Phys*, 169(2):556–593, 2001.

[181] Katsu Yamane and Yoshihiko Nakamura. Stable penalty-based model of frictional contacts. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 1904–1909. IEEE, 2006.

[182] Yin Yang, Guodong Rong, Luis Torres, and Xiaohu Guo. Real-time hybrid solid simulation: spectral unification of deformable and rigid materials. *Computer Animation and Virtual Worlds*, 21(3-4):151–159, 2010.

[183] Mark A Yerry and Mark S Shephard. Automatic three-dimensional mesh generation by the modified-octree technique. *International Journal for Numerical Methods in Engineering*, 20(11):1965–1990, 1984.

[184] Yao Zhang, Jonathan Cohen, and John D Owens. Fast tridiagonal solvers on the gpu. *ACM Sigplan Notices*, 45(5):127–136, 2010.

[185] Teng Zhou, Tong Liu, Yongbo Deng, Limin Chen, Shizhi Qian, and Zhenyu Liu. Design of microfluidic channel networks with specified output flow rates using the cfd-based optimization method. *Microfluidics and Nanofluidics*, 21(1):11, 2017.

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

Appendix A

APIC properties

The properties of our MAC APIC transfers are similar to those of the co-located method [94]. The analysis of the method as presented is notationally complicated compared to the original since the position update is being delayed until the beginning of the next time step.

In this section, we demonstrate that the proposed APIC transfers satisfy the same properties as the original co-located APIC transfers: conservation of linear and angular momentum, preservation of affine velocity fields, and single particle stability.

The proposed scheme conserves momentum and angular momentum in a local sense. Total momentum is conserved, and momentum is only transferred to nearby neighbors (limited by the interpolation stencil size). Due to this interpolation, the transfers will exhibit a degree of momentum diffusion. Since the proposed transfers are not flux-based, it is unlikely that they could be used to track shocks. This is not a problem for incompressible flow (which never exhibits shocks), but this may be a limitation in some applications.

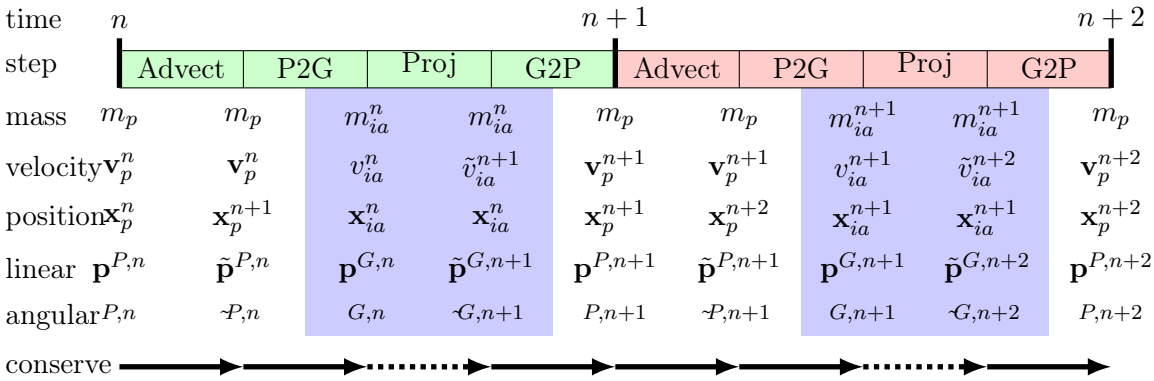


Figure A.1: The proposed time integration scheme can be divided into four stages: advection, particle-to-grid transfer, pressure projection, and grid-to-particle transfer. Two full time steps are shown (first step is green, second step is red). After each stage, the state (mass, velocity, and position) are represented differently. Corresponding to each state is a corresponding measure of linear momentum and angular momentum. The transitions shown with solid arrows are conserved, as proved in Chapter A. The transitions shown with dotted arrows are conserved under a *different* definition of momentum and angular momentum.

Linear momentum

We can define particle-based and grid-based measures of momentum in the usual way.

$$\mathbf{p}^{P,n} = \tilde{\mathbf{p}}^{P,n} = \sum_p m_p \mathbf{v}_p^n \qquad \mathbf{p}^{G,n} = \sum_{ia} m_{ia}^n v_{ia}^n \mathbf{e}_a \qquad \tilde{\mathbf{p}}^{G,n+1} = \sum_{ia} m_{ia}^n \tilde{v}_{ia}^{n+1} \mathbf{e}_a \tag{A.1}$$

With these definitions, momentum is conserved across a particle-to-grid transfer, since

$$\begin{aligned}
\mathbf{p}^{G,n} &= \sum_{ia} m_{ia}^n v_{ia}^n \mathbf{e}_a = \sum_{ia} \left(\sum_p w_{ipa}^n m_p \mathbf{e}_a^T \mathbf{v}_p^n + \sum_p w_{ipa}^n m_p (\mathbf{b}_{pa}^n)^T (\mathbf{D}_{pa}^n)^{-1} (\mathbf{x}_{ia}^n - \mathbf{x}_p^{n+1}) \right) \mathbf{e}_a \\
&= \sum_p m_p \mathbf{v}_p^n \sum_a \mathbf{e}_a \mathbf{e}_a^T \sum_i w_{ipa}^n + \sum_{ap} m_p (\mathbf{b}_{pa}^n)^T (\mathbf{D}_{pa}^n)^{-1} \mathbf{e}_a \sum_i w_{ipa}^n (\mathbf{x}_{ia}^n - \mathbf{x}_p^{n+1}) \\
&= \sum_p m_p \mathbf{v}_p^n = \mathbf{p}^{P,n}
\end{aligned}$$

The transfer from the grid back to the particle also conserves momentum, since

$$\begin{aligned}\mathbf{p}^{P,n+1} &= \sum_p m_p \mathbf{v}_p^{n+1} = \sum_p m_p \sum_{ia} w_{ipa}^n \tilde{v}_{ia}^{n+1} \mathbf{e}_a = \sum_{ia} \left(\sum_p m_p w_{ipa}^n \right) \tilde{v}_{ia}^{n+1} \mathbf{e}_a \\ &= \sum_{ia} m_{ia}^n \tilde{v}_{ia}^{n+1} \mathbf{e}_a = \tilde{\mathbf{p}}^{G,n+1}\end{aligned}$$

Angular momentum

We can define the following particle-based and grid-based measures of angular momentum.

$$^{P,n} = \sum_p \mathbf{x}_p^n \times m_p \mathbf{v}_p^n + \sum_{ap} m_p \mathbf{b}_{pa}^n \times \mathbf{e}_a \qquad ^{G,n} = \sum_{ia} \mathbf{x}_{ia}^n \times m_{ia}^n v_{ia}^n \mathbf{e}_a \tag{A.2}$$

$$^{\tilde{P},n} = \sum_p \mathbf{x}_p^{n+1} \times m_p \mathbf{v}_p^n + \sum_{ap} m_p \mathbf{b}_{pa}^n \times \mathbf{e}_a \qquad ^{\tilde{G},n+1} = \sum_{ia} \tilde{\mathbf{x}}_{ia}^{n+1} \times m_{ia}^n \tilde{v}_{ia}^{n+1} \mathbf{e}_a \tag{A.3}$$

With these definitions, angular momentum is conserved across a particle-to-grid transfer, since

$$
\begin{aligned}
{}^{G,n} &= \sum_{ia} \mathbf{x}_{ia}^n \times m_{ia}^n v_{ia}^n \mathbf{e}_a \\
&= \sum_{ia} \mathbf{x}_{ia}^n \times \mathbf{e}_a \left(\sum_p w_{ipa}^n m_p \mathbf{e}_a^T \mathbf{v}_p^n + \sum_p w_{ipa}^n m_p (\mathbf{b}_{pa}^n)^T (\mathbf{D}_{pa}^n)^{-1} (\mathbf{x}_{ia}^n - \mathbf{x}_p^{n+1}) \right) \\
&= \sum_{ia} \mathbf{x}_{ia}^n \times \mathbf{e}_a \sum_p w_{ipa}^n m_p \mathbf{e}_a^T \mathbf{v}_p^n + \sum_{ia} \mathbf{x}_{ia}^n \times \mathbf{e}_a \sum_p w_{ipa}^n m_p (\mathbf{b}_{pa}^n)^T (\mathbf{D}_{pa}^n)^{-1} (\mathbf{x}_{ia}^n - \mathbf{x}_p^{n+1}) \\
&= \sum_{pa} \left(\sum_i w_{ipa}^n \mathbf{x}_{ia}^n \right) \times \mathbf{e}_a m_p \mathbf{e}_a^T \mathbf{v}_p^n + \sum_{pa} m_p \mathbf{e}_a^{*T} \left(\sum_i \mathbf{x}_{ia}^n w_{ipa}^n (\mathbf{x}_{ia}^n - \mathbf{x}_p^{n+1})^T \right) (\mathbf{D}_{pa}^n)^{-1} \mathbf{b}_{pa}^n \\
&= \sum_{pa} \mathbf{x}_p^{n+1} \times \mathbf{e}_a m_p \mathbf{e}_a^T \mathbf{v}_p^n + \sum_{pa} m_p \mathbf{e}_a^{*T} \mathbf{D}_{pa}^n (\mathbf{D}_{pa}^n)^{-1} \mathbf{b}_{pa}^n \\
&= \sum_p m_p (\mathbf{x}_p^{n+1})^* \left(\sum_a \mathbf{e}_a \mathbf{e}_a^T \right) \mathbf{v}_p^n + \sum_{pa} m_p \mathbf{e}_a^{*T} \mathbf{b}_{pa}^n \\
&= \sum_p \mathbf{x}_p^{n+1} \times m_p \mathbf{v}_p^n + \sum_{ap} m_p \mathbf{b}_{pa}^n \times \mathbf{e}_a = {}^{\mathcal{P},n}
\end{aligned}
$$

The transfer from the grid back to the particle also conserves angular momentum, since

$$
\begin{aligned}
{}^{P,n+1} &= \sum_p \mathbf{x}_p^{n+1} \times m_p \mathbf{v}_p^{n+1} + \sum_{ap} m_p \mathbf{b}_{pa}^{n+1} \times \mathbf{e}_a \\
&= \sum_p \mathbf{x}_p^{n+1} \times m_p \sum_{ia} w_{ipa}^n \tilde{v}_{ia}^{n+1} \mathbf{e}_a + \sum_{ap} m_p \sum_i w_{ipa}^n \tilde{v}_{ia}^{n+1} (\mathbf{x}_{ia}^n - \mathbf{x}_p^{n+1}) \times \mathbf{e}_a \\
&= \sum_{ipa} m_p w_{ipa}^n \tilde{v}_{ia}^{n+1} \big(\mathbf{x}_p^{n+1} + (\mathbf{x}_{ia}^n - \mathbf{x}_p^{n+1}) \big) \times \mathbf{e}_a \\
&= \sum_{ia} \left(\sum_p m_p w_{ipa}^n \right) \tilde{v}_{ia}^{n+1} \mathbf{x}_{ia}^n \times \mathbf{e}_a \\
&= \sum_{ia} \mathbf{x}_{ia}^n \times m_{ia}^n \tilde{v}_{ia}^{n+1} \mathbf{e}_a \\
&= \sum_{ia} (\tilde{\mathbf{x}}_{ia}^{n+1} - \Delta t \tilde{v}_{ia}^{n+1} \mathbf{e}_a) \times m_{ia}^n \tilde{v}_{ia}^{n+1} \mathbf{e}_a \\
&= {}^{G,n+1} - \Delta t \sum_{ia} \tilde{v}_{ia}^{n+1} \mathbf{e}_a \times m_{ia}^n \tilde{v}_{ia}^{n+1} \mathbf{e}_a = {}^{G,n+1}
\end{aligned}
$$

Affine

The APIC affine property is that an affine velocity field is preserved across transfers between particles and grid when particles are not moved ($\Delta t = 0$). Since $\Delta t = 0$, we have $\mathbf{x}_{ia} = \tilde{\mathbf{x}}_{ia}$ (since nothing is moving). We can also ignore superscripts, since time does not matter. Let $f(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{b}$ define an arbitrary affine function. We begin by assuming that the grid velocity is described by this function ($\tilde{v}_{ia} = \mathbf{e}_a^T(\mathbf{A}\mathbf{x}_{ia} + \mathbf{b})$) and transferring to

particles ($\tilde{v}_{ia} \to \{\mathbf{v}_p, \mathbf{b}_{pa}\}$).

$$\mathbf{v}_p = \sum_{ia} w_{ipa}\tilde{v}_{ia}\mathbf{e}_a = \sum_{ia} w_{ipa}\mathbf{e}_a\mathbf{e}_a^T(\mathbf{A}\mathbf{x}_{ia} + \mathbf{b}) = \left(\sum_a \mathbf{e}_a\mathbf{e}_a^T\right)(\mathbf{A}\mathbf{x}_p + \mathbf{b}) = \mathbf{A}\mathbf{x}_p + \mathbf{b}$$

$$\begin{aligned}\mathbf{b}_{pa} &= \sum_i w_{ipa}(\mathbf{x}_{ia} - \mathbf{x}_p)\tilde{v}_{ia} \\ &= \left(\sum_i w_{ipa}(\mathbf{x}_{ia} - \mathbf{x}_p)(\mathbf{x}_{ia})^T\right)\mathbf{A}^T\mathbf{e}_a + \left(\sum_i w_{ipa}(\mathbf{x}_{ia} - \mathbf{x}_p)\right)\mathbf{e}_a^T\mathbf{b} = \mathbf{D}_p\mathbf{A}^T\mathbf{e}_a\end{aligned}$$

From this we can see the corresponding representation of this affine velocity field on particles. Using these particle values and transferring back to the grid ($\{\mathbf{v}_p, \mathbf{b}_{pa}\} \to v_{ia}$) yields.

$$\begin{aligned}m_{ia}v_{ia} &= \sum_p w_{ipa}m_p\mathbf{e}_a^T\mathbf{v}_p + \sum_p w_{ipa}m_p(\mathbf{b}_{pa})^T(\mathbf{D}_{pa})^{-1}(\mathbf{x}_{ia} - \mathbf{x}_p) \\ &= \sum_p w_{ipa}m_p\mathbf{e}_a^T(\mathbf{A}\mathbf{x}_p + \mathbf{b}) + \sum_p w_{ipa}m_p(\mathbf{D}_p\mathbf{A}^T\mathbf{e}_a)^T(\mathbf{D}_{pa})^{-1}(\mathbf{x}_{ia} - \mathbf{x}_p) \\ &= \sum_p w_{ipa}m_p\mathbf{e}_a^T(\mathbf{A}\mathbf{x}_p + \mathbf{b}) + \sum_p w_{ipa}m_p\mathbf{e}_a^T\mathbf{A}(\mathbf{x}_{ia} - \mathbf{x}_p) \\ &= \sum_p w_{ipa}m_p\mathbf{e}_a^T(\mathbf{A}\mathbf{x}_p + \mathbf{b} + \mathbf{A}\mathbf{x}_{ia} - \mathbf{A}\mathbf{x}_p) \\ &= \left(\sum_p w_{ipa}m_p\right)\mathbf{e}_a^T(\mathbf{A}\mathbf{x}_{ia} + \mathbf{b}) = m_{ia}\mathbf{e}_a^T(\mathbf{A}\mathbf{x}_{ia} + \mathbf{b}) = m_{ia}\tilde{v}_{ia}\end{aligned}$$

Since $v_{ia} = \tilde{v}_{ia}$, the velocity field obtained by transferring from grid to particles and back to grid matches the original velocity field.

Stability

The final APIC property is the stability criterion, which requires that a single particle translating in the absence of forces ($\tilde{v}_{ia}^{n+1} = v_{ia}^n$) should translate with no change to

velocity or affine momentum state ($\mathbf{v}_p^{n+1} = \mathbf{v}_p^n$, $\mathbf{b}_{pa}^{n+1} = \mathbf{b}_{pa}^n$, and $\mathbf{x}_p^{n+1} = \mathbf{x}_p^n + \Delta t \mathbf{v}_p^n$). The translation requirement is trivially satisfied. Starting from particles, we first compute the grid-based quantities. Note that summation on particles is omitted since there is only one particle.

$$
\begin{aligned}
\mathbf{x}_p^{n+1} &= \mathbf{x}_p^n + \Delta t \mathbf{v}_p^n \\
m_{ia}^n &= w_{ipa}^n m_p \\
m_{ia}^n v_{ia}^n &= w_{ipa}^n m_p \mathbf{e}_a^T \mathbf{v}_p^n + w_{ipa}^n m_p (\mathbf{b}_{pa}^n)^T (\mathbf{D}_{pa}^n)^{-1} (\mathbf{x}_{ia}^n - \mathbf{x}_p^{n+1}) \\
v_{ia}^n &= \mathbf{e}_a^T \mathbf{v}_p^n + (\mathbf{b}_{pa}^n)^T (\mathbf{D}_{pa}^n)^{-1} (\mathbf{x}_{ia}^n - \mathbf{x}_p^{n+1})
\end{aligned}
$$

We can see that velocities are unchanged since

$$
\begin{aligned}
\sum_{ia} w_{ipa}^n v_{ia}^n \mathbf{e}_a &= \sum_{ia} w_{ipa}^n \mathbf{e}_a \big(\mathbf{e}_a^T \mathbf{v}_p^n + (\mathbf{b}_{pa}^n)^T (\mathbf{D}_{pa}^n)^{-1} (\mathbf{x}_{ia}^n - \mathbf{x}_p^{n+1}) \big) \\
&= \left(\sum_a \mathbf{e}_a \mathbf{e}_a^T \right) \mathbf{v}_p^n + \sum_a \mathbf{e}_a (\mathbf{b}_{pa}^n)^T (\mathbf{D}_{pa}^n)^{-1} \sum_i w_{ipa}^n (\mathbf{x}_{ia}^n - \mathbf{x}_p^{n+1}) = \mathbf{v}_p^n \\
\mathbf{v}_p^{n+1} &= \sum_{ia} w_{ipa}^n \tilde{v}_{ia}^{n+1} \mathbf{e}_a = \sum_{ia} w_{ipa}^n v_{ia}^n \mathbf{e}_a = \mathbf{v}_p^n
\end{aligned}
$$

Finally, affine momentum is unchanged since

$$\begin{aligned}
\mathbf{b}_{pa}^{n+1} &= \sum_i w_{ipa}^n \tilde{v}_{ia}^{n+1}(\mathbf{x}_{ia}^n - \mathbf{x}_p^{n+1}) = \sum_i w_{ipa}^n v_{ia}^n(\mathbf{x}_{ia}^n - \mathbf{x}_p^{n+1}) \\
&= \sum_i w_{ipa}^n(\mathbf{x}_{ia}^n - \mathbf{x}_p^{n+1})\big(\mathbf{e}_a^T\mathbf{v}_p^n + (\mathbf{b}_{pa}^n)^T(\mathbf{D}_{pa}^n)^{-1}(\mathbf{x}_{ia}^n - \mathbf{x}_p^{n+1})\big) \\
&= \left(\sum_i w_{ipa}^n(\mathbf{x}_{ia}^n - \mathbf{x}_p^{n+1})\right)\mathbf{e}_a^T\mathbf{v}_p^n + \sum_i w_{ipa}^n(\mathbf{x}_{ia}^n - \mathbf{x}_p^{n+1})(\mathbf{b}_{pa}^n)^T(\mathbf{D}_{pa}^n)^{-1}(\mathbf{x}_{ia}^n - \mathbf{x}_p^{n+1}) \\
&= \left(\sum_i w_{ipa}^n(\mathbf{x}_{ia}^n - \mathbf{x}_p^{n+1})(\mathbf{x}_{ia}^n - \mathbf{x}_p^{n+1})^T\right)(\mathbf{D}_{pa}^n)^{-1}\mathbf{b}_{pa}^n = \mathbf{D}_{pa}^n(\mathbf{D}_{pa}^n)^{-1}\mathbf{b}_{pa}^n = \mathbf{b}_{pa}^n
\end{aligned}$$

This establishes the one-particle stability criterion.