Simulating Water for Computer Graphics

Particle-in-Cell, Explicit Surfaces, and Discontinuous Galerkin

by

Essex Edwards

B.CIS, The University of the Fraser Valley, 2006
M.Sc, The University of British Columbia, 2010

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

Doctor of Philosophy

in

THE FACULTY OF GRADUATE AND POSTDOCTORAL
STUDIES

(Computer Science)

The University of British Columbia

(Vancouver)

December 2015

© Essex Edwards, 2015
Abstract

We propose several advances in the simulation of fluids for computer graphics. We concentrate on particle-in-cell methods and related sub-problems. We develop high-order accurate extensions to particle-in-cell methods demonstrated on a variety of equations, including constrained dynamics with implicit-explicit time integration. We track the liquid-air interface with an explicit mesh, which we show how to do in a provably exact fashion. To address the mismatched simulation and surface resolution, we solve the partial differential equations in each time step with a $p$-adaptive discontinuous Galerkin discretization. This allows us to use a coarse regular grid for the entire simulation. For solving the resulting linear system, we propose a novel mostly-algebraic domain decomposition preconditioner that automatically creates a coarse discontinuous Galerkin approximation of the problem.
Preface

The work presented in this thesis was supervised by Dr. Robert Bridson, as part of Imager Lab at the University of British Columbia. Generally, the projects were suggested by Dr. Bridson, while the research, design, implementation, and writing were largely performed by myself or other collaborators.


Chapter 5 and Appendix A discuss work that was published as “T. Brochu, E. Edwards, and R. Bridson. Efficient Geometrically Exact Continuous Collision Detection. ACM Transactions on Graphics, 31(4), 2012” [36]. Appendix A is a modified version of that published paper. My contribution to this project was the proof of correctness and the Root Parity Lemma, originally included as supplementary material to the published paper.

The work in Chapter 6 is available as a preprint on arXiv [69].
# Table of Contents

Abstract ................................................................. ii
Preface ................................................................. iii
Table of Contents ....................................................... iv
List of Tables ........................................................... viii
List of Figures ........................................................... ix
Glossary ................................................................. xi

1 Introduction .......................................................... 1
   1.1 Concepts ......................................................... 2
   1.2 Projects ........................................................ 3

2 Related Work ........................................................ 6
   2.1 Particle-in-Cell Methods ....................................... 6
   2.2 Discontinuous Galerkin ....................................... 10
   2.3 Matching Surface and Simulation Resolution ............... 12

3 DG Water ............................................................. 14
   3.1 Introduction ..................................................... 14
   3.2 Related Work .................................................... 16
   3.3 The Method ....................................................... 18
      3.3.1 Discretizing Projection ................................. 18
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.3.2</td>
<td>The Volume Mesh</td>
<td>19</td>
</tr>
<tr>
<td>3.3.3</td>
<td>The Approximation Space</td>
<td>21</td>
</tr>
<tr>
<td>3.3.4</td>
<td>The LDG Equations</td>
<td>22</td>
</tr>
<tr>
<td>3.3.5</td>
<td>Assembly and Linear Algebra</td>
<td>25</td>
</tr>
<tr>
<td>3.3.6</td>
<td>Interpolating to the Particles</td>
<td>27</td>
</tr>
<tr>
<td>3.3.7</td>
<td>Interpolating from the Particles</td>
<td>28</td>
</tr>
<tr>
<td>3.3.8</td>
<td>Particles and Advection</td>
<td>29</td>
</tr>
<tr>
<td>3.3.9</td>
<td>Surface Tracking</td>
<td>33</td>
</tr>
<tr>
<td>3.3.10</td>
<td>Viscosity</td>
<td>33</td>
</tr>
<tr>
<td>3.4</td>
<td>Results</td>
<td>34</td>
</tr>
<tr>
<td>3.5</td>
<td>Discussion</td>
<td>37</td>
</tr>
<tr>
<td>3.6</td>
<td>Conclusion</td>
<td>40</td>
</tr>
<tr>
<td>4</td>
<td>High-Order Accurate Particle-in-Cell Methods</td>
<td>41</td>
</tr>
<tr>
<td>4.1</td>
<td>Introduction</td>
<td>41</td>
</tr>
<tr>
<td>4.2</td>
<td>High-Order PIC</td>
<td>42</td>
</tr>
<tr>
<td>4.2.1</td>
<td>Deriving the Time Discretization</td>
<td>42</td>
</tr>
<tr>
<td>4.2.2</td>
<td>Moving the Particles</td>
<td>44</td>
</tr>
<tr>
<td>4.2.3</td>
<td>Stokes Solve</td>
<td>45</td>
</tr>
<tr>
<td>4.2.4</td>
<td>Interpolating from Grid to Particles</td>
<td>47</td>
</tr>
<tr>
<td>4.2.5</td>
<td>Reseeding</td>
<td>48</td>
</tr>
<tr>
<td>4.2.6</td>
<td>Startup</td>
<td>48</td>
</tr>
<tr>
<td>4.3</td>
<td>A Projection Method</td>
<td>48</td>
</tr>
<tr>
<td>4.4</td>
<td>Results</td>
<td>50</td>
</tr>
<tr>
<td>4.4.1</td>
<td>Taylor Green Flow</td>
<td>51</td>
</tr>
<tr>
<td>4.4.2</td>
<td>A Complex Rayleigh Taylor Instability</td>
<td>52</td>
</tr>
<tr>
<td>4.5</td>
<td>Discussion</td>
<td>54</td>
</tr>
<tr>
<td>4.6</td>
<td>Conclusion</td>
<td>54</td>
</tr>
<tr>
<td>5</td>
<td>Free-Surface Geometry</td>
<td>55</td>
</tr>
<tr>
<td>5.1</td>
<td>The Root Parity Lemma and Proof of Correctness</td>
<td>57</td>
</tr>
<tr>
<td>5.1.1</td>
<td>Outline</td>
<td>57</td>
</tr>
<tr>
<td>5.1.2</td>
<td>Piecewise Linear Functions</td>
<td>58</td>
</tr>
</tbody>
</table>
### List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 3.1</td>
<td>Equations for affine advection schemes</td>
<td>30</td>
</tr>
<tr>
<td>Table 3.2</td>
<td>Detailed timings of the algorithm’s components</td>
<td>38</td>
</tr>
<tr>
<td>Table 6.1</td>
<td>Common symbols for DD</td>
<td>71</td>
</tr>
<tr>
<td>Table 6.2</td>
<td>Iterations of CG with DDG</td>
<td>91</td>
</tr>
<tr>
<td>Table 6.3</td>
<td>Runtimes for DDG</td>
<td>92</td>
</tr>
<tr>
<td>Table 6.4</td>
<td>Normalized runtimes for DDG</td>
<td>93</td>
</tr>
</tbody>
</table>
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>A simulation generates thin splashes</td>
<td>15</td>
</tr>
<tr>
<td>3.2</td>
<td>Schematic diagram of detailed cut cells</td>
<td>20</td>
</tr>
<tr>
<td>3.3</td>
<td>A simulation entirely within one grid cell</td>
<td>21</td>
</tr>
<tr>
<td>3.4</td>
<td>An illustrative $p$-adaptive simulation</td>
<td>23</td>
</tr>
<tr>
<td>3.5</td>
<td>Comparison of advection methods</td>
<td>31</td>
</tr>
<tr>
<td>3.6</td>
<td>High-quality triangles from the explicit surface tracker</td>
<td>32</td>
</tr>
<tr>
<td>3.7</td>
<td>Sub-grid features properly captured by $p$-adaptivity</td>
<td>34</td>
</tr>
<tr>
<td>3.8</td>
<td>Examples with thin sheets</td>
<td>35</td>
</tr>
<tr>
<td>3.9</td>
<td>Comparison of our method to a commercial FLIP code</td>
<td>36</td>
</tr>
<tr>
<td>4.1</td>
<td>Procedure for a single time step of the implicit-explicit PIC</td>
<td>43</td>
</tr>
<tr>
<td>4.2</td>
<td>Error convergence of PIC on the Taylor Green vortex</td>
<td>51</td>
</tr>
<tr>
<td>4.3</td>
<td>Fluid density in a complex scenario</td>
<td>53</td>
</tr>
<tr>
<td>6.1</td>
<td>Error after smoothing with DD</td>
<td>72</td>
</tr>
<tr>
<td>6.2</td>
<td>Code to build DD restriction matrix</td>
<td>73</td>
</tr>
<tr>
<td>6.3</td>
<td>Regions $\Omega_{\text{cut}}$ and $\Omega_{\text{sub}}$ used for analysis</td>
<td>75</td>
</tr>
<tr>
<td>6.4</td>
<td>Meshes used in DDG examples</td>
<td>81</td>
</tr>
<tr>
<td>6.5</td>
<td>$p$-dependent convergence on a biharmonic problem</td>
<td>84</td>
</tr>
<tr>
<td>6.6</td>
<td>Optimal convergence when subdomain size is not fixed</td>
<td>85</td>
</tr>
<tr>
<td>A.1</td>
<td>Cloth folding over a spinning ball</td>
<td>117</td>
</tr>
<tr>
<td>A.2</td>
<td>Root parity test with no roots</td>
<td>124</td>
</tr>
<tr>
<td>A.3</td>
<td>Root parity test with roots</td>
<td>124</td>
</tr>
<tr>
<td>A.4</td>
<td>The ray-vs-bilinear-patch parity test</td>
<td>126</td>
</tr>
</tbody>
</table>
Glossary

BDD balancing domain decomposition
BDF backward differentiation formula
CCD continuous collision detection
CG conjugate gradient method
DAE differential algebraic equation
DD domain decomposition
DDG discretely-discontinuous Galerkin
DG discontinuous Galerkin
FD finite difference
FEM finite element method
FETI finite element tearing and interconnect, domain decomposition
FLIP fluid implicit particle
FVM finite volume method
GIMP generalized interpolation material-point method
LDG local discontinuous Galerkin
MAC the staggered marker and cell grid
MG multigrid
MLS moving least squares
MPM the material-point method
PCG preconditioned conjugate gradient
PDE partial differential equation
PIC particle-in-cell method
SPD symmetric positive definite matrix
SPH smoothed particle hydrodynamics
Chapter 1

Introduction

Numerical simulation of fluids has emerged as an important problem in computer graphics, and of course has long been a challenge in science and engineering. In computer graphics, these simulations typically treat water (or other liquids) as incompressible, uniform density, often inviscid, often with no surface tension, and with zero pressure at the free surface. This is described mathematically by the Navier-Stokes equations, with these simplifications applied.

Any algorithm for solving these equations has to tackle many problems, each of which have been the topic of many published works. The velocity field must be tracked somehow as it is advected by the fluid motion. Likewise, the liquid surface moves and must also be tracked, while allowing for topology changes. The incompressibility condition introduces a global coupling that almost invariably requires solving a large sparse linear system to find the fluid pressure.

Computer graphics, like any field, has its own set of requirements and error metrics that make some techniques more appropriate than others. Computer graphics emphasizes perceptual accuracy, human spatial and temporal scales (around one meter and one second), and the ability to plausibly and robustly handle very complex situations with relatively coarse grids and large time steps.

The bulk of this thesis is organized into chapters based on separate projects completed during my time at the University of British Columbia (UBC). The research chapters begin with Chapter 3, which describes a complete liquid simulation using modern and novel techniques. The remaining chapters investigate some of
those techniques in more detail, and develop and apply them in other ways. Historically speaking, most of the later chapters were done first, before being synthesized in Chapter 3, but we present Chapter 3 first to help establish a context for the other projects. Before the research chapters, Chapter 2 reviews the published literature in the field, further placing the thesis in context, and introducing the tools used in the research. The remainder of the introduction outlines the key concepts, the projects, and their connections.

1.1 Concepts

Throughout the thesis, several concepts and tools are used repeatedly. The core ones are briefly introduced below.

Particle-in-Cell. PIC methods simulate continuum mechanics problem in flows with high distortion. They work by representing the material state with Lagrangian particles, but solve all of the spatial equations by transferring information from the particles to a structured grid, solving the equations on the grid, and transferring results back to the particles.

PIC methods are used in Chapter 3 and Chapter 4.

High-Order Accuracy. High-order accurate methods and high-order convergence are used in every part of this thesis. For example, a function can be approximated as as piecewise polynomial with polynomials of degree $p$ inside of mesh elements of diameter $h$. If that function is sufficiently smooth, then the approximation error ($L_\infty$ norm) will be bounded by $O(h^{p+1})$. Other high-order schemes all have similar error bounds, although $h$ may refer to the time-step for time-dependent problems. When using high-order schemes, solution accuracy can be increased by reducing $h$ ($h$-refinement) or increasing $p$ ($p$-refinement). Which approach is most efficient is case-dependent. Refining both ($hp$-refinement) in an adaptive fashion is also possible.

Discontinuous Galerkin. DG is a flexible technique for solving partial differential equations (PDEs). Using DG, the problem domain is partitioned into elements and
the solution within each element is restricted to be a polynomial. Often DG methods are high-order accurate. The distinct feature of discontinuous Galerkin (DG), relative to finite element method (FEM), is that the functions are allowed to be discontinuous across element boundaries.

DG concepts are used in Chapter 3 and Chapter 6.

1.2 Projects

**DG Water.** In Chapter 3 many ideas come together to make a state-of-the-art liquid simulation. This chapter follows a trend in recent research of using explicit triangle meshes to track the liquid surface during simulation, while the physical forces are computed on an underlying grid. Using an explicit surface allows for very detailed and accurate surface tracking, but special care needs to be taken to couple the grid and the surface, or artifacts will develop.

The core problem is that the surface mesh resolution may be higher than the resolution of the grid. Consequently, there are deformations of the surface that cannot be resolved by the physics computed on the grid. Chapter 3 describes an adaptive method that still uses a simple coarse Cartesian grid. To capture high-resolution surface details we use exact detailed cut cells at boundaries, and adaptively use richer discrete models within grid cells near the surface, more specifically a $p$-adaptive DG method. This DG discretization of the spatial components is used in a particle-in-cell (PIC) discretization of the flow problem, necessitating some changes to the typical operations that couple the PIC particles to the grid.

**High-Order Accurate Particle-in-Cell Methods.** Particle-in-cell (PIC) methods, in particular fluid implicit particle (FLIP) and the material-point method (MPM), are very popular in computer graphics. However, they both suffer from problems of limited accuracy, and have been almost exclusively limited to first-order accuracy since the introduction of PIC in the 1950s. Chapter 4 introduces a high-order accurate PIC method.

This chapter concentrates on two key components: interpolation between the particles and the grid, and time-integration. This chapter is follow-up work to
my master’s thesis, which introduced a high-order accurate PIC method, but with limitations that made it impossible to apply to incompressible Navier-Stokes in the usual velocity-pressure form. In this thesis, we develop a semi-implicit time-integration scheme capable of handling the constrained dynamics necessary for Navier-Stokes in this form.

The PIC operations in Chapter 4 and Chapter 3 are quite similar. In both cases, particles are interpolated in a different way than most PIC methods, allowing for higher-order accuracy. Chapter 3 does not achieve global high-order convergence, primarily due to the use of low-order time integration, but Chapter 4 demonstrates how this can be done.

Free-Surface Geometry. Chapter 5 describes the first provably exact algorithm for detecting collisions between moving triangle mesh elements. This primitive operation is a fundamental building block for robust adaptive and dynamic meshes, which made possible the reliable surface tracking used in Chapter 3.

Algebraic DG Coarse Grid for Domain Decomposition. Many discretizations of elliptic PDEs lead to sparse symmetric positive definite (SPD) linear systems. For large 3D problems, iterative solvers such as preconditioned conjugate gradient (PCG) are usually necessary and the quality of the preconditioner becomes the crucial factor for efficiency and robustness. With an optimal preconditioner, the linear system can be solved in a time which scales linearly with the problem size. Domain decomposition (DD) is one common framework for optimal preconditioning [136, 153].

A popular framework for optimal DD solvers is the Schwarz method with coarse grid correction. First, the problem domain is partitioned into multiple subdomains, in which the local problem is solved by whatever technique is convenient. These solutions are missing the global modes, so the second component in DD is a coarse global discretizations of the PDE. Combined, the coarse global solution and the local solutions are potentially an optimal preconditioner.

In Chapter 6, we present a coarse discretization that can be constructed algebraically from the input matrix and the positions of the mesh nodes. This dis-
cretization is inspired by DG, as used in Chapter 3. The method is easy to use, applies to a broad variety of PDEs, and leverages the power of high-order convergence to outperform other simple algebraic schemes and converge optimally even on higher-order PDEs.
Chapter 2

Related Work

Each research chapter includes a focused review of the most relevant literature for that chapter. The following sections provide a broader view on the main tools and problems in the thesis.

2.1 Particle-in-Cell Methods

The PIC method is an approach to simulating continuum mechanics problems in flows with high distortion. PIC is a fundamentally Lagrangian scheme, in which all material properties are carried on Lagrangian particles that move with the flow. On each time step, some PDE must be solved to compute the forces (and other updates) applied to the particles. To compute these efficiently, all material properties are interpolated from the particles to a convenient computation grid. The PDE is then solved on the grid and updates are interpolated back to the particles.

The hybrid particle/mesh approach of PIC attempts to get the best of both Lagrangian and Eulerian approaches. The use of Lagrangian particles for tracking material properties can give very low numerical diffusion compared to Eulerian representations on relatively coarse grids. The use of a structured grid for the solution of the PDE enables the use of efficient and accurate finite difference (FD) methods, finite volume methods (FVM), or finite element methods (FEM) rather than the typically more complicated schemes, or less accurate approximations, used in fully mesh-free methods.
In the most general form of a modern PIC method for fluid and continuum mechanics problems, particles with positions $r$ and other attributes $q$ (momentum or velocity included in $q$, or derivable from it) are seeded throughout the domain as initial conditions. The attributes are transferred from the particles to a structured grid, so that $Q_i$ are the values on the grid. Some PDE discretization is solved on the grid to compute values at the end of the time step, $Q_{\text{new}}$. The changes over the course of the time step, i.e., $\Delta Q = Q_{\text{new}} - Q$, are interpolated from the grid back to the particles and used to update their values to end of the time step. Similarly, the particles are moved according to the velocity field interpolated from the grid.

The particle-in-cell method first appeared as a method for solving compressible flow problems in a series of Los Alamos National Lab technical reports starting in the 1950s [77, 90, 91], and in the academic literature shortly after [88, 92]. A good reference for this early work is the 1969 annotated bibliography by Harlow [89].

At this stage, PIC was interpreted as an essentially Eulerian methodology. The particles were only used when they moved from one grid cell to another: when a particle crossed cell boundaries, it would pick up some of the momentum, mass, and energy of its old cell, and deposit it into its new cell. This approach suffered from numerous problems, including excessive diffusion, noise, and instabilities.

The 1970s saw essentially no advances in PIC methods for fluid or continuum mechanics problems, because of its poor performance. In fact, Brackbill et al. wrote, “were it not for its usefulness in simulating plasmas, PIC would be mainly of historical interest” [25]. Fortunately, advances were being made with PIC in plasma simulations, and in the early 1980s, these advances were adopted from plasma dynamics back into fluid mechanics in the fluid implicit particle (FLIP) method [24, 25, 39].

In FLIP, the particles are the primary representation of the material and carry the entire material state, making it a fundamentally Lagrangian method. This change alone is mostly conceptual, but it paves the way for future improvements. When the time step is computed on the grid, only updates are interpolated back to the particles, rather than new values. For example, the particle velocities are updated with the acceleration computed on the grid, rather than directly interpolating the new particle velocities from the new grid velocities. These changes effectively removed all numerical diffusion from FLIP.
Through the 1990s, FLIP was extended to apply to history-dependent elasto-plastic materials, in the the material-point method (MPM) \cite{145, 146}. For MPM, the equations were derived from a weak formulation, and strong theoretical connections were made between FEM and PIC, with the particles interpreted as quadrature points. After its introduction, many papers were published on improvements, extensions, or applications of MPM. Recent applications of MPM include: brittle fracture \cite{107}, adhesive contact \cite{106}, explosive welding \cite{162}, non-local plasticity \cite{40}, and nearly-incompressible materials \cite{112}.

The generalized interpolation material-point method (GIMP) \cite{13} presents a unifying framework in which to analyze FLIP, MPM, and their relatives. In GIMP, both the particles and the grid nodes have shape functions, which may be unrelated to each other. The spatial PDEs are discretized in terms of the grid's degrees-of-freedom by a Petrov-Galerkin procedure that projects the particle fields onto the grid fields. The entries in the mass matrix, for example, are sums of integrals of the products of the grid shape functions and particle shape functions. The original MPM paper corresponds to the choice of linear “tent” functions for the grid shape functions, and Dirac-deltas for the particle shape functions.

With the framework established by MPM and GIMP, many papers investigated sources of error and potential improvements. Every method published to this point is inherently first-order accurate in space and usually first- or second-order accurate in time. Steffen et al. \cite{140} analyze the many spatial and temporal error sources within MPM, and describe a method to balance the error terms by choosing appropriate spatial and temporal resolutions.

Several papers explored implicit time integration schemes for MPM \cite{59, 86, 87, 144}. These typically focused on non-linear plasticity problems, and used iterative Newton-Krylov methods. The degrees of freedom being solved for are only the grid degrees of freedom, but each iteration needs to involve the particles because of their appearance in the mass-matrix and other terms. Wallstedt and Guilkey \cite{159} evaluated many explicit time integration schemes. Ultimately, they write that “while the choice of time integration scheme has a large impact on the overall accuracy of a simulation, the ultimate conclusion [...] is that, when the best of these choices is made, spatial error remains dominant.”

Spatial errors have received a lot of attention too. Steffen et al. \cite{139} write,
“much of the anomalous behavior exhibited by MPM can be attributed to the quadrature approximation properties of the method. In fact, many of the proposed improvements to MPM either explicitly or tacitly attempt to control and improve MPM’s quadrature behavior.” In particular, they shows that the grid-cell-crossing instability that plagues all PIC methods up to and including MPM is due to quadrature errors that result from using the particles as quadrature points on non-smooth integrands.

Improvements to the spatial terms have been made by adopting a Hermite-type scheme for interpolating the particle to the grid [158], by using smoother shape functions that reduce quadrature errors [139, 172], or by using higher-order elements for the grid shapes [5]. Moving away from the framework of GIMP, spatial errors are drastically reduced by applying least-squares approximation to interpolate from the particles to the grid [67, 160].

MPM, FLIP, and GIMP are almost exclusively applied to compressible material problems. In fact, MPM fails on incompressible problems due to locking [112]. However, in plasma dynamics, while the flows are not divergence-free, there are constraints on the divergence of the electric and magnetic fields. For a introduction to PIC’s use in plasma physics, we refer the reader to the review paper by Tskhakaya et al. [154]. Jacobs and Hesthaven [95, 96] describe a plasma PIC method that is high-order accurate in both space and time. They satisfy the divergence constraint either by projection or by weakening the constraint. Their approach is not immediately applicable to incompressible fluid dynamics, because of the different way that the divergence-constraint acts in the plasma equations, as well as significant differences between PIC’s application in plasma dynamics and fluid dynamics.

In computer graphics, PIC was introduced to fluid simulation [175] as a modified form of FLIP, adapted to incompressible flow. Rather than storing extensive properties (e.g., mass and momentum) and being concerned with conservation laws, the particles are interpreted as storing point-samples of the unknown continuous fields (e.g., density and velocity). The particle-to-grid interpolation problem is then a scattered-data interpolation/approximation problem. The grid discretization does not depend on any information about the particles, except through the right-hand-side “load” vector. The complete decoupling of the particles from the grid’s discretization is very flexible, but makes it unclear how to enforce conservation laws, if it is even possible. It is this flexible and decoupled form that we use in this
Another advantage of using the particles as massless point samples is that adding and removing particles is comparatively trivial; several interesting techniques have been proposed [6, 23]. In comparison, removing particles in MPM removes mass and momentum from the system, so more delicate techniques are required [105, 147].

Recently, several papers [141, 142] have been published at SIGGRAPH using MPM for elasto-plastic solids and melting. The high-quality results seem to have excited an interest in MPM for the graphics community, in addition to the ongoing success of FLIP.

Chapter 3 and Chapter 4 are both PIC methods with novel features.

2.2 Discontinuous Galerkin

Discontinuous Galerkin (DG) is a flexible technique for solving PDEs. Using DG, the problem domain is partitioned into elements (triangles, quadrilaterals, curvilinear simplicies, arbitrary polyhedra, etc.) and the solution within each element is restricted to be in some space of polynomials (or other approximation space). The distinct feature of DG, relative to conforming FEM, is that the functions are allowed to be discontinuous across element boundaries.

The standard Galerkin finite element method cannot be applied on such a function space, as it is not a subspace of the appropriate Sobolev space for typical PDEs. Naïvely applying FEM assembly routines to DG approximation spaces results in a singular system, with no coupling terms between adjacent cells. DG addresses this problem by applying the standard Galerkin approach within each element, and then inventing an inter-element flux term to handle the discontinuities. This flux term is not uniquely defined, and varies from one DG formulation to the next. Ideally, the flux should be designed so that the entire method is consistent, adjoint-consistent, conservative, stable (i.e., coercive), symmetric (for symmetric problems), and computationally efficient.

For the Poisson problem ($\nabla \cdot \nabla u = f$), which is all one needs to solve for simple fluids, Arnold et al. [9] presented a unified analysis of many existing DG formulations. The methods they identify as both stable and consistent are the interior-
penalty (IP) method [62], Brezzi et al. [29], non-symmetric interior penalty DG (NIPG) [126], Bassi et al. [15], and local DG (LDG) [47]. Of these, only Brezzi et al. [29] and LDG converge to optimal order independent of non-physical parameters. For certain simplifying parameter choices, LDG reduces to Brezzi’s method, so we concentrate on LDG.

LDG has seen a lot of development since its introduction for the convection-diffusion problem [51]. It has since been applied to: Stokes problem [52], the Oseen equations [53], quasi-Newtonian Stokes flow [42], incompressible Navier-Stokes [54], elastic solids [56], Maxwell’s equations [163], magnetohydrodynamics [83], Burger’s equation [132], and Hamilton-Jacobi equations [169]. For a more thorough overview of LDG, we refer the reader to the review papers by Cockburn and Castillo [46, 55].

Early on, LDG was analyzed for elliptic problems, and Castillo et al. [47] note that it “might be advantageous because of the ease with which the method handles hanging nodes, elements of general shapes, and local spaces of different types”. These are exactly the problems we face in Chapter 3. Yuan and Shu [171] took advantage of custom local spaces to show vastly improved performance when a well-adapted non-polynomial basis is used that matches the expected form of the solution. Bustinza [41] present a unified analysis of LDG for linear and non-linear PDEs.

Kanschat [100] have noted that LDG, appropriately applied, recovers the well-known staggered marker and cell (MAC) grid. It has also been shown that, subject to certain conditions, LDG recovers exactly divergence-free results even when the divergence-free condition is only applied weakly, and the basis is not divergence-conforming [57, 83]. Also, special multilevel preconditioners have been proposed for LDG [97–99].

There are more recent DG methods than LDG. One method of note is hybridizable DG (HDG) [44], for which the global system to solve is only in terms of a function defined on the boundary of the elements. The compact DG (CDG) method [121] also reduces the computational expense, relative to LDG, by making the matrix sparser. Another interesting trend is the development of fluxes based on fitting, rather than differential operators on the boundary, akin to how high-order fluxes are derived in finite volume methods [21, 110].
Chapter 3 uses DG primarily as a tool, and includes novel results for DG with complex time-dependent cut cells. Chapter 6 applies concepts from DG in a purely discrete setting to develop a novel preconditioner.

2.3 Matching Surface and Simulation Resolution

When an explicit surface mesh is used as the free-surface within a fluid simulation, something must be done to address the different resolution of the surface mesh and the volume mesh on which the spatial PDE is solved.

A typical fluid simulation in computer graphics uses a regular staggered grid for the discretization. There may be multiple surface vertices within a single grid cell forming a bump or other feature that cannot be resolved or reacted to by the simulation. In the best case, these high-frequency surface features simply persist in a non-physical fashion, but they may very well be unstable and grow in time. One must either regularize the surface, smoothing away any features below the grid resolution and lowering the effective surface resolution to that of the grid, or locally increase the simulation resolution near the surface.

Within the field of computer graphics, there have been several approaches to handling the problem of mismatched surface and grid resolution.

The simplest approach is to use a low-resolution mesh, or smooth/simplify the surface mesh until it has no features which are not captured by the simulation grid. As demonstrated by Müller [117], this has some advantages over implicit surfaces. However, to a large degree, this approach loses sight of the fact that we care much more about the surface detail than the bulk flow detail. We do not consider this approach any further, and always assume that the surface resolution is higher than the volume-mesh resolution.

One approach is to regularize the surface so that it is topologically equivalent to the topology seen by an underlying grid-based physics simulation. This gets the bulk physics correct, but without further regularization, fine scale details that are not representable on the simulation grid persist. For simulations of elastic or very viscous materials, this may be perfectly acceptable [165 166]. For liquids where fine surface details should not persist, Wojtan et al. [167] and Thürey et al. [152] apply surface-tension or mean-curvature flow to smooth them in a physically-
motivated way. The disadvantage of this approach is that any dynamic features of the surface below the grid resolution are essentially non-physical (unless the physical surface tension is sufficiently high). For situations with less surface tension, good results were achieved with a surface model resembling vortex sheet equations [20].

Another approach comes from Brochu et al. [35]. In this case, the surface is not modified to agree with the grid in any way [33]. They used an $h$-adaptive volume mesh that starts with a regular lattice and then inserts special mesh elements around the surface vertices. In this way, the surface is well represented on the grid. This approach does not suffer from the non-physical behaviour of the approach above, but using an unstructured $h$-adaptive volume mesh is more computationally expensive. Presumably, similar results could be achieved with other $h$-adaptive discretizations.

Another approach that uses an explicit triangle mesh for the surface works by maintaining a boundary conforming Lagrangian tetrahedralization of the entire spatial domain [113–115]. The surface triangles are the set of facets separating the water and air tetrahedra (or other materials). By using the tetrahedral mesh as the simulation mesh, this approach has a perfect match between simulation and surface degrees of freedom. Again, the use of an unstructured $h$-adaptive mesh is expensive.

The method in Chapter 3 approaches this problem in a new way, using $p$-adaptivity to enrich the approximation space within grid cells instead of adapting the simulation grid structure itself. This gives the grid enough degrees of freedom to resolve the motion of the surface mesh contained within, but retains much of the simplicity of regular Cartesian grids.
Chapter 3

DG Water

3.1 Introduction

Simulating liquids for visual effects demands a high resolution surface with detailed motion, but typically not the same high resolution in the entire volume: it is only the surface that we observe, and experiment and theory often indicate that most of the interesting dynamics (e.g. strong vorticity) are generated at the surface and remain strongest near the surface. We cannot avoid all volumetric computation, but much is gained by concentrating the bulk of the computation on the surface, as surface area scales quadratically with size while volume scales cubically. We are interested in techniques that take advantage of this opportunity. For liquid simulation, this encompasses methods for the surface tracker, the volumetric velocity/pressure solver, and their interactions.

Surface tracking may use an implicit method (e.g., level-set or volume-of-fluid), explicit method (e.g., marker particles or mesh), or a hybrid (e.g., particle level set). These approaches are already capable of spending computation and memory only at the surface. In this chapter we use an existing method to track the surface with an explicit triangle mesh, and concentrate on handling the dynamics. While the surface tracker may appear to be conceptually independent of the dynamics’ discretization, artifacts can easily arise if the surface tracker and dynamical model are poorly coupled.

Broadly speaking, there have been two approaches to simulating a high-resolution
surface without a correspondingly high-resolution mesh for the entire volume. The first category uses a simple coarse volumetric fluid model everywhere, and adds a secondary model for the missing high-resolution surface features. These methods are generally quite fast and attractively simple, but typically make simplifying assumptions that can lead to physically incorrect behavior. The second category of methods use an adaptive volumetric mesh that matches the high resolution at the surface but is low resolution in the interior of the liquid. The unified handling of all the dynamics is physically consistent and correct, but much more computationally expensive, in large part due to the complexity of using unstructured or semi-structured meshes.

We present an adaptive method that is physically consistent and correct, but still uses a simple coarse Cartesian grid. To capture high-resolution surface details
we use detailed cut cells at boundaries, and adaptively use richer discrete models within grid cells near the surface, more specifically a $p$-adaptive DG method.

In summary, our core technical contributions are:

- the first application of DG with exact cut cells to moving free-surface problems, and
- a novel particle advection scheme that requires fewer evaluations of the velocity field.

We also highlight some ideas that are new to fluid simulation in graphics:

- $p$-adaptive techniques, and
- embracing discontinuous approximations at all levels.

### 3.2 Related Work

Fluid simulation has a long history in computer graphics, and an even longer history in scientific computing, engineering, and physics. For an overview of the major techniques in graphics, we refer the reader to Bridson’s book [30].

This chapter uses the hybrid Lagrangian-Eulerian FLIP method [175]. FLIP is interpretable as an extension of the Eulerian velocity-pressure formulation with staggered time stepping [81]. Since its introduction to graphics, FLIP has seen several developments related to this work. Adaptive FLIP particle distributions have been introduced for use in an $h$-adaptive simulation [7] and for detailed tracking of thin sheets [6]. Other particle methods (e.g., smoothed particle hydrodynamics (SPH)) can also have adaptive particle distributions, including generally $h$-adaptive approaches [3] and special two-scale approaches [137]. Interpolation between FLIP particles and high-order grids has been seen outside of graphics with both smooth [67] and non-smooth [116] interpolants.

For tracking the liquid surface, marker particles [81] and level-set methods [119] or their combination in the particle level-set [73] have been most popular. Recently, explicit surface tracking with triangle meshes has gained interest; e.g. reviewed in a recent SIGGRAPH course [168]. In this work we use the El Topo explicit surface tracking library of Brochu et al. [33].
As mentioned above, some methods embed a high-resolution surface tracker in a low-resolution volumetric simulation and apply a second model to the surface to control sub-grid motion. Regularizing the surface so that its topology is representable in the grid-based physics simulation has been recognized as an important feature in these approaches: without regularization, fine scale details that are not captured in the simulation grid behave non-physically. For simulations of elastic or very viscous materials, simple persistence of fine details may be perfectly acceptable [165–167]. For liquids, fine surface details should not persist. Müller [117] handles this by remeshing the surface every time step, and restricting the geometry and topology within each cell to be very simple. Thürey et al. [152] applied surface-tension and mean-curvature flow to smooth small details in a physically-motivated way. For situations with less surface tension, good results were achieved with a surface model resembling vortex sheet equations [20].

A second class of approaches use $h$-adaptive volumetric simulations to put smaller diameter ($h$) volumetric elements near the surface, but large elements in the bulk flow, capturing surface dynamics in the same way as the volume dynamics. Classic examples of this are octrees [109] and unstructured tetrahedral meshes, including Eulerian approaches [7, 17, 49] and Lagrangian approaches [113–115]. Recognizing the simplicity and efficiency of regular grids, chimera grids combine multiple regular grids of different resolution [72] and tall-cell grids go adaptive only in the vertical direction [48, 94]. Special $h$-adaptive schemes designed for explicit surface trackers are also possible [35].

The techniques in this chapter are not $h$-adaptive, but rather $p$-adaptive, increasing the resolution in an area by increasing the approximation degree, $p$, in volumetric elements rather than by geometric refinement. To our knowledge, these techniques have never been applied in graphics, but have a long history in finite element methods (FEMs). When the approximation degree is allowed to be very large, one arrives at spectral [22] and spectral element methods [101]. While we use pure $p$-adaptivity, both $h$ and $p$ adaptivity can be combined into $hp$-adaptive methods [11, 129].

Our discretization of the Poisson-projection problem uses a discontinuous Galerkin (DG) approach, a close relative of the famous FEM. For a review of many DG methods for elliptic problems, we recommend the unified analysis by Arnold et al.
Of these, we make use of the local discontinuous Galerkin (LDG) method, which has been applied to many problems from Poisson to non-linear Navier-Stokes [46, 55]. DG has been used previously in computer graphics for elastic deformations [102], where its flexibility enabled the embedding of high-resolution solids inside low resolution volumetric meshes, similar to our use.

3.3 The Method
We work with the velocity-pressure ($u-p$) form of the Navier-Stokes equations, simplified by assuming incompressible, inviscid, and uniform-density fluid with free-slip conditions at solid-fluid boundaries and free-surface conditions with no surface-tension at the air-fluid boundaries. This common model is appropriate for simulating medium to large bodies of liquid.

We apply a staggered time stepping, with separate advection and projection steps. The fluid velocity field is stored with FLIP particles, and the boundary location is stored with an explicit triangle mesh (Figure 3.6). An outline of one time step is given in Algorithm 1 including references to the section where each step is described.

3.3.1 Discretizing Projection
The projection step of our solver takes an intermediate velocity field $\tilde{u}$ from advection and gravity, and applies a pressure gradient to make it divergence-free while respecting solid and air boundaries. The projected velocity field $u$ is found by

$$u + \nabla p = \tilde{u},$$

$$\nabla \cdot u = 0,$$

in the liquid domain $\Omega$, and subject to the boundary conditions.

Discretizing with DG follows the same outline as FEM. First the domain is partitioned into cells. Within each cell, some approximation space is specified for $u$ and $p$. Finally, a weak form of the PDE is satisfied within this space. Unlike typical FEM, DG allows each cell to have different approximation spaces regardless of discontinuities across cell boundaries or agreement with boundary conditions.
### Algorithm 1 Outline of a time step

**Input:** particle positions, velocities, and surface mesh at time $t_0$

**Output:** particle positions, velocities, and surface mesh at time $t_1$

1. **Projection Step:**
   2. Build cut-cell volume mesh
   3. Prepare basis functions
   4. Assemble Poisson matrix
   5. Interpolate velocity from particles to grid
   6. Solve linear system
   7. If viscosity $\neq 0$, apply viscous update
   8. Interpolate update from grid to particles

9. **Advection Step:**
   10. Add particles where necessary
   11. Advect FLIP particles, including surface
   12. Smooth surface
   13. Update surface tracker:
       - collision detection/response
       - topology changes
       - remeshing
   14. Remove particles where necessary
   15. Add body forces to FLIP velocities

Furthermore, the cells need not be simple shapes.

#### 3.3.2 The Volume Mesh

To construct our volume mesh, we begin with a regular Cartesian voxel grid with spacing $h$. In voxels that contain the boundary of the liquid, we intersect the liquid’s volume and voxel to get a surface mesh of the liquid in just that cell. This is the detailed cut cell that we use in our discretization, replacing the cubes of the Cartesian grid, but keeping the same regular structure (Figure 3.2). This volume mesh conforms to the liquid boundary, exactly capturing all sub-grid features. The only simplification comes later, by projecting the PDE into the approximation space.

Each triangle of the explicit surface is assigned either the free-surface or solid boundary condition. Triangles that are close to the solid mesh and have normal oriented opposite the solid’s normal are assigned solid-boundary conditions.

We need to avoid cut cells with small, skinny, or other otherwise poor geometry.
Figure 3.2: A schematic diagram of the detailed cut cells in the volumetric mesh (thick lines) constructed by intersecting the polyhedral liquid domain (blue fill) with the cubes of a regular grid (thin lines). Small cells are merged with neighbors (dashed lines). Note that (A) a grid cell may contain multiple connected components, and (B) a cut cell may have complex topology.

that can cause poor conditioning of the discretization. This issue has come up with elasticity [102] and 2D fluid problems [80, 124]. We follow a similar heuristic strategy of merging ‘poor’ cells with adjacent cells to create larger cells with better conditioning. It is always possible to avoid arbitrarily small volumes because our surface tracker does not produce arbitrarily thin features; see §3.3.9.

Deciding when and how to merge cells is a matter of heuristics, and our results were not sensitive to its details. When a cell’s volume is small ($V < h^3/100$), we merge it with the adjacent cell with which it has the largest shared face. For cells with intermediate volume ($V < h^3/4$), we merge them with a neighbour only if the area ($A$) of their shared face is sufficiently large ($A^{3/2} > V$). The condition on shared area prevents undesired merging of thin fluid sheets into a single large thin cell.
Figure 3.3: A simulation entirely within one grid cell. Note the detailed pressure variation within the thin sheet and detailed sub-grid velocities. This example uses degree 6 polynomials for pressure: 28 variables.

The location and size of the grid each frame is arbitrary, as it stores no data between time steps. In our 3D implementation, we perturb its location each time step to avoid degenerate cut-cell geometry.

3.3.3 The Approximation Space

DG requires an approximation space for \( u \) and \( p \) within each cell. For pressure, we use \( P_k \) – the polynomials of degree no more than \( k \) (i.e., the span of \( x^a y^b z^c \) where \( 0 \leq a, b, c \) and \( a + b + c \leq k \)). We could use the same space \( P_k \) for each component of velocity, but we use \( P_{k+1} \) instead because it provides a more accurate velocity for the same size pressure solve. Larger disparities, such as \( P_{k+2} \) for velocity, may be unstable. Figure 3.3 illustrates how these high-order approximation spaces can provide lots of sub-grid detail. If a cell has multiple connected components, each component uses a separate DG approximation space: we simply enrich the basis accordingly.

When applying high-order FEM on Cartesian grids, tensor product polynomials \( Q_k \) (like \( P_k \) but including powers up to \( \max(a, b, c) \leq k \)) are normally used. These make continuity simple while achieving \( k \)th order accuracy. However, \( Q^6 \) has di-
mension $k^3 + \mathcal{O}(k^2)$ whereas $P_k$ only has dimension $k^3/6 + \mathcal{O}(k^2)$ but achieves the same order of accuracy. Since DG allows discontinuities, we can use the $P_k$ spaces, so the pressure solve has only one sixth the number of variables of an equivalent continuous discretization.

The solution is independent of the choice of basis, but it has large effects on the conditioning of the linear system. We use a nodal basis for $P_k$ defined by points in a simplex, as commonly used for FEM. To adapt the basis to the irregularly-shaped cut cells, we approximately fit this simplex to each cell by fitting an oriented bounding box aligned with the cell’s inertia tensor and another axis-aligned bounding box, taking the smaller of these boxes, and using the largest tetrahedron that fits in the box. This provides anisotropically stretched basis functions that match the anisotropy of thin sheets or tendrils of liquid.

The cornerstone of $p$-adaptivity is using different approximation spaces in different cells. For cells touching the free surface we use high-degree polynomials (typically cubic or quartic pressure fields). All other cells use one degree less than their neighbor of highest degree, but no less than linear pressure, as shown in Figure 3.4. Using linear pressure as the minimum ensures that the hydrostatic case is solved exactly. This $p$-adaptive approach allows us to use a coarse regular grid for the whole domain, while still achieving fine-scale details where desired.

For smooth functions, $p$-refinement is more efficient than $h$-refinement. Error estimates generally bound the error by $\mathcal{O}(h^p)$, while the number of degrees of freedom is $\mathcal{O}((p/h)^3)$. Dividing those expressions, the error per degree of freedom decreases exponentially with $p$ but only geometrically with $h$. For non-smooth functions, this is false, but fortunately liquid simulations typically have smooth velocity and pressure, even when the surface geometry is non-smooth.

### 3.3.4 The LDG Equations

We use the LDG method, which is well-studied, flexible, and has simple parameters [55]. Using LDG, when the solution is exactly representable in the approximation space, LDG finds that solution exactly, independent of geometry. Consequently quiescent free-fall and hydrostatic liquids (linear $p$, constant $u$) are exactly reconstructed. In general, the boundary conditions, continuity, and the PDE are all
Figure 3.4: An illustrative $p$-adaptive simulation. This simulation uses quartic polynomials for pressure at the surface (red cells), and linear polynomials for pressure in the interior (light cells), with smooth grading of the polynomial degree in intermediate cells.
satisfied in a weak sense.

LDG for the Poisson problem was introduced by Castillo et al. [47]. This section briefly follows their construction of the equations. We refer the reader to their paper for full details and alternative forms of the equations.

We look first at the divergence-free condition. Multiplying by an arbitrary smooth test-function \( q \) and integrating over an arbitrary grid cell \( K \subseteq \Omega \) reveals

\[
\int_K q \nabla \cdot u = 0.
\]

Integrating by parts,

\[
\int_K \nabla q \cdot u \, dV = \int_{\partial K} q u \cdot n_K \, dA
\]  

(3.3)

where \( n_K \) is the unit outward normal on \( K \).

For continuous functions and conforming FEM, this is sufficient. However, \( u \) changes discontinuously across cell boundaries so it is not clear what value to use in the boundary integral. These equations must be modified to apply to discontinuous functions. DG methods introduce a numerical flux \( \hat{u} \) at discontinuities and modify Equation 3.3. The discrete solution finds a velocity field in the approximation space that satisfies

\[
\int_K \nabla q \cdot u \, dV = \int_{\partial K} q \hat{u} \cdot n_K \, dA
\]  

(3.4)

for all cells \( K \) in the mesh and all test functions \( q \) in the pressure approximation space.

Different DG methods define different numerical fluxes. Let \( K^+ \) and \( K^- \) be two cells with a shared face, with normals \( n^\pm \) and \( u = u^\pm \) in \( K^\pm \). Then, letting \( \theta = \frac{1}{2} - C_{12} \cdot n^+ \), LDG defines

\[
\hat{u} = \theta u^+ + (1 - \theta) u^- - C_{11}(n^+ p^+ + n^- p^-).
\]  

(3.5)

where \( C_{12} \) and \( C_{11} \) are parameters.

On the domain boundary, the flux is defined using the boundary conditions. At solid-boundaries, \( \hat{u} \) is taken to be the velocity of the solid. At the free-surface \( \hat{u} = u^+ - C_{11} n^+ p^+ \). Both of these are equivalent to a particular definition of \( u^- \), \( p^- \), \( C_{11} \), and \( C_{12} \) on the boundary.

We choose the parameters to be as simple as possible while achieving optimal convergence. The vector field \( C_{12} \) may be an arbitrary \( O(1) \) function. We bypass
defining it, and set \( \theta = \frac{1}{2} \) directly. The parameter \( C_{11} \) acts as a penalty parameter on discontinuities in the pressure. By using \( C_{11} = \mathcal{O}(1) \), LDG requires no mesh-dependent parameters, but using \( C_{11} = \mathcal{O}(h^{-1}) \) achieves a better order of accuracy. We use \( C_{11} = h^{-1} \).

Writing \( u \) and \( p \) in terms of coefficient vectors \( \mathbf{u} \) and \( \mathbf{p} \) for the velocity and pressure basis functions, and substituting into the integral equations (3.4), we arrive at an equivalent linear system \( \mathbf{G}^T\mathbf{u} - \mathbf{S}\mathbf{p} = \mathbf{0} \). The weak form of the momentum equation is discretized similarly and combined into the symmetric indefinite linear system

\[
\begin{bmatrix}
\mathbf{M} & \mathbf{G} \\
\mathbf{G}^T & -\mathbf{S}
\end{bmatrix}
\begin{bmatrix}
\mathbf{u} \\
\mathbf{p}
\end{bmatrix}
= 
\begin{bmatrix}
\mathbf{M}\tilde{\mathbf{u}} \\
\mathbf{0}
\end{bmatrix}
\]  

(3.6)

where \( \frac{1}{2}\mathbf{u}^T\mathbf{M}\mathbf{u} \) exactly equals the kinetic energy of the fluid. The right-hand-side \( \tilde{\mathbf{u}} \) is the intermediate velocity field after advection. It is found by interpolation from the FLIP particles (see §3.3.7). Any non-stationary solids would add another term on the right-hand-side.

### 3.3.5 Assembly and Linear Algebra

To assemble the system, we evaluate the integrals with exact high-order quadrature. In the interior of the domain, efficient Gaussian quadrature is available, and the grid structure allows a table of local integrals to be precomputed once, then added into the global matrix as needed. However, integration over cut cells must be performed every time step as their shapes change.

To integrate over a cut cell’s volume, we clip the water mesh to the Cartesian grid cell, triangulate its boundary, then inexpensively tetrahedralize the cell by connecting each face to an arbitrary central point. By taking sign into account for the final exact integral, we allow overlapping and inverted tetrahedra. The final volume integral is a sum of exact quadratures applied to each tetrahedron, after which the tetrahedra are discarded and used for nothing else. The boundary integrals are similarly computed by a summation of quadratures over faces.

The volume integrals in \( \mathbf{G} \) (\( \mathbf{S} \) has no volume integral) can be easily computed from \( \mathbf{M} \). Each grid cell contributes three identical diagonal blocks to \( \mathbf{M} \), one for...
each velocity component. Let \( \mathbf{M}_K \) be one of these blocks. Given two functions \( a \) and \( b \) (discretely represented as \( \mathbf{a} \) and \( \mathbf{b} \)) from \( P_{k+1} \) (i.e., one component of the velocity space), then \( \mathbf{M}_K \) satisfies

\[
\mathbf{a}^T \mathbf{M}_K \mathbf{b} = \int_K ab \, dV. \tag{3.7}
\]

In \( \mathbf{G} \) (see Equation 3.4), the volume integral component \( \mathbf{G}_K \) from a single cell satisfies

\[
\mathbf{u}^T \mathbf{G}_K \mathbf{q} = \int_K \nabla q \cdot \mathbf{u} \, dV \tag{3.8}
\]

with \( q \) from the pressure space and \( \mathbf{u} \) from the velocity space. Taking the partial derivative of \( q \) with respect to \( x \) is a linear operator whose action may be written as \( \mathbf{D}_x \), so that \( \mathbf{D}_x \mathbf{q} \) represents \( \frac{\partial q}{\partial x} \) in the same \( P_k \) basis as \( \mathbf{q} \). This matrix is dependent on the choice of basis (representative tetrahedron), so it may not be precomputed. Instead, on a reference tetrahedron, we precompute and store these derivative operators \( \mathbf{D}_x^{\text{ref}} \) (similarly for \( y \) and \( z \)). From those matrices, the operators for element \( K \) can be computed from the reference operators

\[
\mathbf{D}_x = J_{xx} \mathbf{D}_x^{\text{ref}} + J_{xy} \mathbf{D}_y^{\text{ref}} + J_{xz} \mathbf{D}_z^{\text{ref}}. \tag{3.9}
\]

The \( J \) terms come from the Jacobian of the transformation from the reference tet. to cell \( K \)’s representative tet. Thus, \( \mathbf{D}_x \) is simple to compute. Another matrix, \( \mathbf{B} \) can be precomputed, independent of cell shape, that converts from the \( P_k \) basis to the \( P_{k+1} \) basis. With these matrices, \( \mathbf{B} \mathbf{D}_x \mathbf{q} \) represents \( \frac{\partial q}{\partial x} \) in the same basis as velocity. Introducing one final matrix \( \mathbf{R}_x \) that extracts just the \( x \)-component of a velocity field, we arrive at

\[
\mathbf{u}^T \mathbf{G}_K \mathbf{q} = \int_K \nabla q \cdot \mathbf{u} \, dV \tag{3.10}
\]

\[
= \mathbf{u}^T \mathbf{R}_x^T \mathbf{M}_K \mathbf{BD}_x \mathbf{q} + \mathbf{u}^T \mathbf{R}_y^T \mathbf{M}_K \mathbf{BD}_y \mathbf{q} + \mathbf{u}^T \mathbf{R}_z^T \mathbf{M}_K \mathbf{BD}_z \mathbf{q} \tag{3.11}
\]

where the \( y \) and \( z \) subscripts are the natural generalizations. From this, we conclude

\[
\mathbf{G}_K = \mathbf{R}_x^T \mathbf{M}_K \mathbf{BD}_x + \mathbf{R}_y^T \mathbf{M}_K \mathbf{BD}_y + \mathbf{R}_z^T \mathbf{M}_K \mathbf{BD}_z.
\]
The implementation of this is simpler than the mathematics might make it first appear. For each polynomial degree, we store $B$, and the derivative matrices $D^\text{ref}$ on the reference tetrahedron. These are all small dense matrices. For each cell we compute $M_K$, another small dense matrix, by quadrature. Then we apply the Jacobian-term to get $D_x$ acting on the current cell’s representative tetrahedron. The action of $R$ amounts to storing the resulting matrix products $M_KBD$ into the correct spot in $G$. $R$ is never explicitly created at all.

The boundary integrals can be treated similarly. Again, a mass-like matrix must be computed by quadrature (or other means) for each face. From there, the necessary differential operators and dot products can be written in terms of the same precomputed matrices.

In the interior of the domain, the situation is even simpler. Because the cell shape is constant (a cube), all of the local matrices can be precomputed.

To solve system (3.6), we eliminate $u$ to get a symmetric positive definite system for the pressure only.

$$Lp = (S + G^T M^{-1} G)p = G^T \tilde{u}$$  \hspace{1cm} (3.12)

Because the basis functions in separate cells do not overlap, $M$ is block diagonal and easy to invert directly. We construct $L$ with several dense matrix operations per cell and solve this linear system with conjugate gradient, preconditioned with block-wise zero-fill incomplete Cholesky. Substitution after solving system (3.12) gives $u = \tilde{u} - M^{-1} G p$.

### 3.3.6 Interpolating to the Particles

Once $u$ and $\tilde{u}$ are known, we interpolate an update from the grid to the FLIP particles. The DG basis functions naturally define the interpolants $u(x)$ and $\tilde{u}(x)$, and a linear operator $P$ that evaluates these at all the particle locations. To update the
particle velocities $v_{\text{old}}$, we use Zhu and Bridson’s PIC/FLIP mixing strategy [175]:

\[
v_{\text{PIC}} = Pu \tag{3.13}
\]

\[
v_{\text{FLIP}} = v_{\text{old}} + Pu - \bar{u} \tag{3.14}
\]

\[
v_{\text{new}} = \theta v_{\text{FLIP}} + (1 - \theta)v_{\text{PIC}} \tag{3.15}
\]

We interpret this mixing of $v_{\text{PIC}}$ and $v_{\text{FLIP}}$ as the solution to a modified ODE that includes a decay of the (potentially) noisy particle velocities towards the noise-free grid velocities. For a decay with half-life $\lambda$, setting $\theta = 2^{-\Delta t/\lambda}$ achieves this decay in a stable fashion. Unlike directly setting $\theta$, this approach gives a time step independent amount of diffusion. We choose the half-life in terms of some characteristic time for the simulation, generally $\lambda = 0.5$ s (so, e.g., $\theta \approx 0.95$ when $dt = 1/30$ s).

This introduces a spatially-varying amount of diffusion, in agreement with the spatially-varying effective resolution of the grid. This is appropriate when interpreted as a regularization, but it should not be used as an approximate viscosity.

### 3.3.7 Interpolating from the Particles

In previous FLIP methods for computer graphics, the interpolation/approximation of the intermediate particle velocity $\bar{v}$ to the grid is done by evaluating a weighted-average at the nodal points of the grid. This is inappropriate in this discretization because the nodal points have no particular meaning: they may be oddly distributed in space, outside the fluid, or with another basis construction, nonexistent.

Instead, we find the coefficients $\bar{u}$ that minimize the error when interpolating back to the particles, $\min ||Pu - \bar{v}||$. In cut cells that contain few particles, many solutions have zero residual. To make the problem well-posed, we use a regularization term to select smooth velocity fields.

\[
\bar{u} = \arg \min_u \left( ||Pu - \bar{v}||^2_2 + \eta \sum_K \int_K ||\nabla u(x)||^2_2 \right) \tag{3.16}
\]

where $\eta = 0.1$ m$^{-1}$ in all our simulations. The regularization also adds robustness against small amounts of noise in the particle data.
This global optimization problem separates into a small independent least-squares problem in each cell, since we use discontinuous approximation spaces which are independent in each cell, and the smoothness term is designed for separability by ignoring discontinuities across cell faces.

### 3.3.8 Particles and Advection

The fluid state consists of the FLIP particles (positions $r$ and velocities $v$), as well as the surface triangle mesh (positions, velocities, and connectivity). All the surface vertices are also FLIP particles, which is vital for capturing thin sheets and threads where the fluid volume is so small that there may be no particles in the interior. Every time step, we add and remove particles as necessary so that each cell has approximately twice as many particles as it has basis functions for each velocity component, scaled down by volume-fraction in cut cells, ensuring that particles and grid represent a similar level of detail.

When the liquid surface is advected, vertices that were touching the solid boundary before being advected are projected back onto the solid surface at the end of the time step, consistent with the free-slip solid boundary conditions. To reduce collisions between the solid and liquid mesh, the liquid mesh is kept away from the solid by moving a small amount in the normal direction after this projection.

As in other FLIP solvers, the particles are advected using an ordinary differential equation (ODE) integrator through the static velocity field $u$ computed on the grid. The simplest classical integrator that has stable behavior around vortices is RK3. Unfortunately, this requires 3 evaluations of the velocity field per step, which are expensive high-order interpolations from the grid. Furthermore, these evaluation points may be outside the fluid volume, requiring some sort of extrapolation.

We present a new technique that uses only a single evaluation of the velocity field. With a single lookup into the grid, we build the linear Taylor approximation to $u$ at a particle’s location $r$, using $u(r)$ and $\nabla u(r)$. The velocity field in each cell is at least a quadratic polynomial, so $\nabla u$ is always available. (In conventional solvers, a local finite difference could be used to estimate the gradient instead, typically using values that are already in cache.) This affine velocity field is enough to describe
Table 3.1: Equations for affine advection schemes that calculate particle displacement $d$ given scaled velocity $b = \Delta t u(r)$ and velocity gradient $S = \Delta \nabla u(r)$ at the particle’s initial location $r$.

<table>
<thead>
<tr>
<th>Method</th>
<th>Equation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forward Euler</td>
<td>$d = b$</td>
</tr>
<tr>
<td>RK2</td>
<td>$d = (I + \frac{1}{2}S)b$</td>
</tr>
<tr>
<td>RK3</td>
<td>$d = (I + \frac{1}{3}S + \frac{1}{6}S^2)b$</td>
</tr>
<tr>
<td>RKN</td>
<td>$d = (I + \frac{1}{2}S + \cdots + \frac{1}{N!}S^{N-1})b$</td>
</tr>
<tr>
<td>Exact (for affine $u$)</td>
<td>$d = \begin{bmatrix} 1 &amp; 0 \ 0 &amp; 1 \end{bmatrix} \exp \left( \begin{bmatrix} S &amp; b \ 0 &amp; 0 \end{bmatrix} \right) \begin{bmatrix} 0 \ 1 \end{bmatrix}$</td>
</tr>
<tr>
<td>Trapezoidal/Midpoint</td>
<td>$d = (I - \frac{4}{3}S)^{-1}b$</td>
</tr>
</tbody>
</table>

vortices and shear flow, so we use RK3 to advect through this approximation, and achieve the same linear stability properties as RK3 applied directly to $u$.

To demonstrate the advection scheme, we uniformly distribute particles in a 2D box, and advect them through the divergence-free velocity field with stream-function $\sin(x) \sin(y)$. This velocity field is a single vortex with pure rotation at its center, and pure shear in the corners of the $[0, \pi]^2$ domain. Figure 3.5 shows the result of 100 time steps ($\Delta t = 1.5$) with RK2, RK3, our approximate RK3, and the correct solution. Each integrator uses a different number of substeps, such that all approaches require the same number of evaluations of $u$. We experimented with other advection schemes using integrators of various accuracy applied to velocity approximations of various order, and found this combination to be an appropriate trade-off between speed and accuracy. Table 3.1 lists algebraically simplified expressions for several integrators applied to the local affine approximation.

Tracing through the velocity field is usually also a significant expense for existing FLIP and semi-Lagrangian simulators, since each velocity evaluation requires an unstructured memory look-up and mixing of integer and floating-point pipelines: we believe our new technique should be of interest in accelerating other solvers too.
Figure 3.5: Comparison of advection methods. Uniformly-distributed tracer particles are advected in a stationary vortex. Each integrator takes substeps such that all approaches evaluate the velocity field 6 times per time step. After 100 time steps, RK2 and RK3 both show significant clumping and divergence of particles.
Figure 3.6: A still from the simulation in Figure 3.1, showing the complex splashing geometry, and high-quality triangles from the explicit surface tracker.
3.3.9 Surface Tracking

Surface tracking is handled by the El Topo library [33], which produces quality triangulations as shown in Figure 3.6. We made some small modifications to adapt El Topo to our use case.

Our discretization cannot handle arbitrarily thin sheets of liquid, so we prevented the surface tracker from creating them. We first tried deleting sheets of liquid that became too thin (e.g. < 0.02h), but this results in flicker at the tips of thin sheets as they are deleted. This is visible in our 2D videos, which use this strategy. We found better results in 3D by adding a repulsion force that tries to thicken sheets that are thinner than 0.05h. Thickening a thin sheet can cause an increase in fluid volume and momentum, but our results indicate it is an acceptable trade-off. Either choice helps ensure good conditioning and behavior of the DG problem.

Some high-frequency surface features are handled poorly by our discretization and the surface tracker, so we use a small amount of Laplacian smoothing on the surface every step. As with PIC/FLIP smoothing, we can interpret this as a decay of each vertex towards the average position of its neighbours, with half-life between 0.1s and 0.5s for different examples. Smoothing is not applied at sharp convex features, determined by dihedral angles between triangles.

3.3.10 Viscosity

While our focus is inviscid flow, we sometimes found adding a small amount of viscosity improved the stability and visual appeal of the results. Viscosity is applied as a separate time-splitting step, after projection, to the grid’s velocity field. We use an approximate implicit formulation of viscosity in which we apply the Galerkin discretization of the implicit backward-Euler viscous step in each cell, but solve each cell independently. Viscous fluxes between cells are ignored, so convergence to the viscous Navier-Stokes solution is not expected, but it serves as an effective regularization. This solve can increase the size of the discontinuities at cell boundaries, but not visibly so for the levels of viscosity in our examples. Figures 3.1 and 3.8 (bottom) used kinematic viscosity of $2 \cdot 10^{-2}$ and $2 \cdot 10^{-4}$ respectively, in non-dimensional units relative to $h$ (length) and 1 second (time). Other examples
Figure 3.7: Sub-grid features persist as non-physical bumps when using linear pressure, but are properly captured by physics when using quartic pressure, leaving only ripples in the surface. This view is approximately five grid cells wide.

use no viscosity.

3.4 Results

The discretization uses an extended model for pressure and velocity in a single cell. To experiment with the capabilities of this model, we ran a simulation in a single grid cell. This required no modifications to the method, just shorter edge lengths and more particles per cell than we typically use. Figure 3.3 shows a 1 × 1 simulation using sextic polynomials for pressure. Within the single cell, the method captures thin sheets, sloshing, and interactions with non-trivial solid boundaries. This example is, in essence, a very coarse spectral method, and emphasizes that our model is solving for full sub-grid physics, not applying a simple smoothing or more approximate fluid model to the surface.

The polynomial degree used in cut cells must be chosen in concert with mesh resolution. We examined the behavior as the polynomial degree changes by reproducing an experiment from [35]. We simulated a still pool, with a subgrid disturbance $h/2$ wide and $h/10$ high, with surface mesh edge lengths of $\approx h/4$. Figure 3.7 shows that with low-order polynomials there are persistent artifacts, but for quartic pressure and higher, the bump is captured by the physics and ripples across the domain. Furthermore, simulation quality increases gradually as the polynomial degree is increased through intermediate degrees. This example uses
no unphysical Laplacian smoothing in the normal direction.

Two controlled scenarios demonstrate the handling of thin sheets (Figure 3.8). Duplicating the experiment by Thürey et al. [152], we drop a ball of liquid onto a flat-topped pillar whence it expands into a thin sheet. As the sheet moves nearly ballistically through the air, small perturbations in its initial velocity amplify into a wavy shape. Second, we run a dam-break with a smooth obstacle less than $3h$ tall. The liquid forms a thin sheet as it spreads over and around the obstacle, following its curves and generating a detailed wake. In both scenarios, the thin sheets are still represented exactly in the discretization, due to the use of detailed cut cells.

To gain a sense of how practical the new approach is, we compared our code to a commercial FLIP solver, running the complex scenario from Figures 3.1 and 3.6. Figure 3.9 shows the results. Running on a 4-core 2.3GHz Intel Core i7 laptop with 4GB memory, the new DG solver took an average of 72 seconds per time step, with a $25^3$ grid and sheets as thin as $1/48^{th}$ of a grid cell. Timings for different
Figure 3.9: Comparison of our method on a $25^3$ grid (top), a commercial FLIP code on a $121^3$ grid (middle), and the same commercial code on a $49^3$ grid (bottom).
steps are given in Table 3.2. We ran the commercial FLIP solver (using a standard
discretization and sparse, tiled voxels for efficiency) at two different resolutions,
effectively $49^3$ and $121^3$. Both methods took up to three time steps per frame.

At $49^3$ the commercial FLIP solver used approximately the same number of
pressure variables as the DG code and was $14 \times$ faster at 5.25s per time step, but
of course gave far less detailed results: the solver could not resolve sheets thinner
than a grid cell. At $121^3$ the commercial FLIP solver took a similar compute
time, at 43s per time step, and had a qualitatively similar perceived level of detail.
However, the character of the detail is quite different. The DG simulation produces
smooth, structured, and very thin features, with sheets approximately $1/500^{th}$ of
the domain width in thickness – and even thinner in places. In contrast, the commercial FLIP code cannot reliably represent anything below a grid cell, $1/121^{th}$ of
the domain width. The commercial FLIP simulation produces rough and splashy results in which the sheets break up into droplets, and cannot produce the smooth and extremely thin sheets of the DG results without drastically higher grid resolution.

In contrast to our research prototype, the commercial code is thoroughly op-
timized and multithreaded. Given the scope for continued performance and par-
allelism improvements in the new code, the ease of $p$-adaptivity, and the fully
dynamic thin features that other solvers cannot capture nearly as efficiently, we
believe this represents a very practical way forward for liquid animation.

3.5 Discussion

Interpreting the Method  One simple interpretation of this method looks at it as
another spatially-adaptive alternative to octrees or unstructured meshes. Relative
to those methods, this approach has several advantages. First, it keeps the struc-
ture of the regular grid, significantly simplifying many parts of the algorithm. This
approach also naturally produces anisotropic elements around thin fluid ligaments
and sheets, providing significant reductions in problem size relative to the isotropic
$h$-adaptation that is nearly universal. Furthermore, increasing $p$ is more efficient
for problems with smooth solutions. However, it is difficult to achieve large reso-
lution differences between two areas of the simulation with just $p$-adaptivity, pre-
Table 3.2: Detailed timings of the algorithm’s components during the simulation from Figure 3.1, run on a laptop with an Intel i7-3610QM and 8GB of memory.

<table>
<thead>
<tr>
<th>Median Component Runtimes (seconds/time step)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Projection: build volume mesh and ( u-p ) basis 2</td>
</tr>
<tr>
<td>quadrature + assembly 15</td>
</tr>
<tr>
<td>linear solver 6</td>
</tr>
<tr>
<td>Advection: RK3, smoothing, collisions 2</td>
</tr>
<tr>
<td>Surface: collision handling 38</td>
</tr>
<tr>
<td>topology changes and remeshing 7</td>
</tr>
<tr>
<td>rebuilding data-structures 5</td>
</tr>
<tr>
<td>Entire Step: 82</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Median Variable Counts (number/time step)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grid: cut cells 1600</td>
</tr>
<tr>
<td>regular cells 2800</td>
</tr>
<tr>
<td>pressure variables 50800</td>
</tr>
<tr>
<td>Mesh: surface vertices 205000</td>
</tr>
</tbody>
</table>

cisely because of imposed grid structure. Our \( p \)-adaptive approach would also work on general \( h \)-adapted meshes, complementing the \( h \)-adaptive techniques, and indeed would simplify octree-like methods with T-junctions since continuity between elements is not required. We are interested in seeing their combination into an \( hp \)-adaptive method.

Another over-simplified way to look at this method is as an approach for regularizing sub-grid features that arise when embedding a high-resolution surface into a low-resolution grid. Unlike the two-model approaches in §3.2, this approach resolves the velocity-pressure field to sub-grid features in a unified way and a single solve. Nevertheless, it cannot handle arbitrary amounts of sub-grid detail without extremely expensive and high-order bases. Combining these techniques may improve the result quality. The small amount of Laplacian smoothing we apply to the mesh can be interpreted in this way.

Both of the above interpretations try to cast this method in terms of the familiar, and consequently miss its novel behaviors and particular strengths. This approach is adaptive in a new way. In a sense, the velocity field has no limit to how small a feature can be, just as the distance between two extrema in a polynomial can be
arbitrarily small. There is no free lunch, of course. The velocity field in a cell is limited to some finite dimensional space, and there is a limit on the total number and type of ‘features’ per grid cell. Still, the velocity field in a cell has the full approximating power of degree $k$ polynomials, regardless of how small the liquid volume is within that cell. As a result, sub-grid thin sheets (and other features) can stretch, curl, and generally behave physically.

Thin Sheets  This algorithm handles thin sheets remarkably well. We used sheets down to $0.02h$ thick, giving extremely anisotropic elements. We contrast our behavior here with the behavior of typical grid solvers. To represent the sheet using a level set on a standard MAC grid would require at least fifty times the grid resolution, i.e. greater than 2500 active degrees of freedom for a sheet that only uses 35 pressure variables in our approach. Methods using explicit surface tracking within a simple regular MAC grid could track the same geometry as our approach, but will still have difficulty producing an accurate velocity field in areas where sheets collide and produce complex topology within a single grid cell.

We chose a common physical model without surface tension, and without any effects from the ‘air’ outside the liquid domain. Physical experiments and theory identify that it is the velocity difference at the liquid-air interface that drives instability of sheets, and surface tension that drives retraction at their rims [71]. Without either of these effects, sheets and jets spread to be very thin without breaking up. This is in stark contrast to typical fluid simulations, in which discretization artifacts break up or delete fluid sheets, despite the fact that they have no physical model for that behavior. It’s also in stark contrast to our every-day experience with splashes at small scale, which atomize almost instantly. Capturing instability-driven break-up in a physical way remains an outstanding problem.

Discontinuities  This algorithm embraces discontinuities at every opportunity, granting more flexibility and efficiency to the algorithms. Advection, interpolation between the grid and the particles, and the volume integrals all involve purely local data. The relevant data can be stored contiguously in memory, allowing efficient computation on modern architectures. Similarly, large parts of the computation can
be expressed as dense matrix operations on data from a single cell (or pair of cells), and computed with highly-tuned dense BLAS and LAPACK routines.

**Limitations** Our method has some weaknesses when the velocity field is not very smooth. For example, a small collapsing air bubble requires a velocity field that quickly switches directions from one side of the bubble to the other, and these sometimes appear to have a ‘numerical pressure’ slowing their collapse. Given that collapsing bubbles are a non-physical artifact of the free surface model, slowing their collapse is not necessarily a bad thing. Another important case is when topological merges occur, introduce a nearly-discontinuous intermediate velocity, and cause large discontinuities in the discrete solution. The only time we saw visible discontinuities was in the first few frames after a droplet impacts a pool. They are imperceptible at regular playback speed. Despite these limitations, we still find the $p$-adaptive approach works well. These issues could, potentially, be resolved by using a different approximation space that includes velocity fields like these.

### 3.6 Conclusion

The combination of explicit surface tracking, detailed cut cells, and $p$-adaptive DG is an effective method for liquid simulation. They provide a qualitatively different set of performance characteristics than more common discretizations. In particular, they scale much better for smooth fields, and accurately capture thin sheets and features well below the grid scale. These techniques, applied together or independently, have lots of room for improvement and application in this and other areas.
Chapter 4

High-Order Accurate Particle-in-Cell Methods

4.1 Introduction

The particle-in-cell (PIC) method is an approach to simulating continuum mechanics problems featuring flows with high distortion. PIC is a fundamentally Lagrangian scheme, in which all material properties are carried on Lagrangian particles that move with the flow. On each time step, a PDE must be solved to compute the forces (and other changes) applied to the particles. To compute these efficiently, all material properties are transferred from the particles to a convenient computation grid where the PDE is solved. Updates are interpolated from the grid back to the particles.

As it is usually described, PIC is inherently low-order accurate in space and usually low-order accurate in time. We present a PIC method that is high-order accurate in both space and time. For concreteness, we concentrate on fourth-order accuracy, but in a way that is extensible to higher or lower orders. The proposed method maintains high-order accuracy even in the face of stiff or constrained dynamics (such as an incompressibility constraint), through an implicit-explicit time-integration strategy.

For a review of the PIC literature, see Section 2.1.
4.2 High-Order PIC

Our new approach is a direct successor to the high-order PIC method by Edwards and Bridson [67], with the intent to improve on its limitations. Their approach is unable to simulate incompressible Navier-Stokes in the natural velocity-pressure formulation

\[
\frac{Du}{Dt} = \nu \nabla \cdot \nabla u - \nabla p + f \\
\nabla \cdot u = 0
\]  

\((\partial/\partial t = \partial/\partial t + u \cdot \nabla)\) because it could only handle unconstrained dynamics. Furthermore, it was limited to explicit time steps, imposing severe time step restrictions for stiff problems.

The scheme in this chapter is semi-implicit in time, allowing for large time steps of stiff effects, such as viscosity. The implicit approach also naturally handles (infinitely-stiff) constraints, such as the divergence-free constraint (equation 4.2), allowing the simulation to work directly in the velocity-pressure form.

The remainder of this section derives and describes the discretization. The full algorithm is summarized in Figure 4.1

4.2.1 Deriving the Time Discretization

We begin with the incompressible Navier-Stokes equations (4.1 & 4.2) as well as the equation for a particle’s position \(r\) in the fluid:

\[dr/dt = u(r)\]  

Discretizing with a finite number of particles, we arrive at finite-dimensional vectors \(r, u, p\) for the particle positions, velocities, and pressure. Each particle stores the value of \(u\) at its location, e.g. \(u_i = u(r_i)\). This is unlike the original MPM and FLIP papers, or SPH, in which the particles are ‘blob’-like and store extensive quantities such as mass and momentum. We will see later that pressure need not be stored.

Along with the particles, PIC requires a background grid (mesh) and discretiza-
1. Input particle data:
   (a) $r_{n-1}$ positions
   (b) $u_{n-1} \ldots u_{n-k}$ velocity history
   (c) $p_{n-1} \ldots p_{n-k}$ pressure history (projection method only)

2. Reseed particles as necessary

3. Compute safe time step

4. Compute weights for the multistep methods

5. Move particles
   (a) Compute position update $\Delta r$ on particles
   (b) Interpolate update to the grid and back to the particles
   (c) Move the particles $r_n = r_{n-1} + \mathbf{I}^T \Delta r$

6. Compute known BDF derivative terms on particles

7. Compute pressure extrapolation on particles (projection method only)

8. Interpolate to grid

9. On the grid: solve Stokes problem, or projection problem

10. Interpolate update from grid to particles

**Figure 4.1:** Procedure for taking a single time step of the implicit-explicit PIC scheme.
To distinguish between the grid and particle representation of the same field, we use capital and lower-case letters. Arbitrary data \( q \) on the particles is transferred to the grid \( Q \) by \( Q \approx I^i(r)q \). The linear operator \( I^i(r) \) depends non-linearly on particle positions. Similarly, the grid’s representation can be interpolated to the particles using another linear operator \( q \approx I^i(r)Q \). In general, \( I^i \) and \( I^j \) need not satisfy any particular relationship. For accuracy, we assume that if \( q \) represents a smooth field, then \( I^i(r)q \) is a high-order accurate representation of that smooth field. Likewise, we require the same of \( I^i(r) \), the grid-to-particle interpolation. We slightly abuse notation here; rather than apply the scalar-field transfer operators component wise for the vector-fields, we simply write the same symbol applied to vector fields.

We do not interpret the particles to define any continuous field, except through an auxiliary grid. Spatial derivatives are defined only on the grid, and for simplicity of notation, we write \( \nabla \), \( \nabla \cdot \), \( \nabla^2 \) even for the discrete operators. How they are discretized is not important at this stage, as long as they are sufficiently accurate.

Substituting all of these relations back in to the Navier-Stokes equations, we arrive at a PIC-like spatial discretization:

\[
\frac{dr}{dt} = u \quad (4.4)
\]
\[
D\frac{Du}{Dt} = vI^i(r)\nabla^2I^i(r)u - I^i(r)\nabla I^i(r)p + f \quad (4.5)
\]
\[
I^i(r)\nabla \cdot I^i(r)u = 0 \quad (4.6)
\]

To discretize in time, we apply several multistep methods, making use of a history of velocity data \( u_{n-k} \ldots u_n \) from the last \( k \) time steps.

### 4.2.2 Moving the Particles

Equations 4.5 and 4.6 both have non-linear dependencies on \( r \), through the interpolation operators. In order to avoid solving a non-linear implicit problem, we tackle the advection equation (4.4) explicitly. Particle advection is not a stiff problem, so an explicit scheme causes no difficulty here.

To update the particle positions, we use the Adams-Bashforth method, which gives a formula for the change in position \( \Delta r \) from time \( t_{n-1} \) to time \( t_n \), using past
velocities.

\[ \Delta r = \sum_{i=1}^{k} \alpha_i u_{n-i} \quad (4.7) \]

The weights \( \alpha_i \) are such that if \( r(t) \) is a degree \( k \) polynomial in time, then \( \Delta r \) is exact.

This update could be applied directly to the particles (i.e. \( r_n = r_{n-1} + \Delta r \)). However, this would allow multi-streaming (intersecting particle trajectories) and makes the simulations less stable (see paragraph 4.4.2). To avoid this problem, \( \Delta r \) is interpolated to the grid and then back before being applied to the particles.

\[ r_n = r_{n-1} + I^u(r_{n-1})I^l(r_{n-1})\Delta r \quad (4.8) \]

This is consistent with the general approach in PIC, wherein all updates applied to the particles come from the grid, and the data on the particles alone is considered to be ‘noisy’ and unreliable until interpolated to the grid.

### 4.2.3 Stokes Solve

With \( r \) handled explicitly, the remainder of the PDE can take \( I^u \) and \( I^l \) as being time-dependent rather than \( r \)-dependent.

\[ du/dt - \nu I^l(t)\nabla^2 I^l(t)u + I^l(t)\nabla I^l(t)p = f \quad (4.9) \]

\[ I^l(t)\nabla \cdot I^l(t)u = 0 \quad (4.10) \]

Multiplying adjacent operators together, we arrive at the remaining equations to be discretized in time:

\[ (d/dt - A(t))u + B(t)p = f \quad (4.11) \]

\[ C(t)u = 0 \quad (4.12) \]

This is a linear differential algebraic equation (DAE), with time-varying linear operators. The particular form, with differential (\( u \)) and algebraic (\( p \)) variables separated in this way, puts this in Hessenberg size two form. In Navier-Stokes, \( CB \) is approximately the Laplace operator and should be invertible, which makes
this an index-2 DAE. These DAEs are solvable with the backward differentiation formula (BDF) method [10], which is stiffly stable up to 6\textsuperscript{th} order accuracy. Using BDF, the time derivative at time \( t_n \) is approximated by

\[
\frac{du}{dt}(t_n) \approx \sum_{i=0}^{k} \beta_i u_{n-i}
\] (4.13)

with weights \( \beta \) that make the approximation exact for low-order polynomials. Substituted into the first equation,

\[
\beta_0 u_n - vI^\dagger \nabla^2 I^\dagger u_n + I^\dagger \nabla I^\dagger p_n = \tilde{f}_n
\] (4.14)

where all matrices have implicit \( n \) subscripts, and \( \tilde{f} \) contains all the forcing terms and BDF history terms.

Solving this equation (along with the constraint equation) would be a complete BDF time step. However, this equation involves the particle-grid transfer operators, which is undesirable, as they are generally quite complex and make it impossible to directly apply efficient linear solvers developed for discretizations on the grid. So, we make some approximations to eliminate the transfer operators and solve a system entirely on the grid. Recalling that \( I^\dagger I^\dagger \) is approximately the identity operator for smooth functions, we interpolate all terms to the grid and back to the particles uniformly

\[
\beta_0 I^\dagger I^\dagger u_n - vI^\dagger \nabla^2 I^\dagger u_n + I^\dagger \nabla I^\dagger p_n = I^\dagger \tilde{f}_n
\] (4.15)

Typically, \( I^\dagger \) will be tall and full rank, implying that

\[
\beta_0 I^\dagger u_n - v\nabla^2 I^\dagger u_n + \nabla I^\dagger p_n = I^\dagger \tilde{f}_n
\] (4.16)

which, after re-labeling the \( I^\dagger q \)-like terms,

\[
\beta_0 U_n - v\nabla^2 U_n + \nabla P_n = \tilde{F}_n
\] (4.17)

is a set of equations entirely on the grid. Applying the same approach to the constraint equation, and combining them together, one arrives at an indefinite linear
system that must be solved on each time step:

\[
\begin{bmatrix}
(\beta_0 - \nu \nabla^2) & \nabla & 0 \\
\nabla & 0 & 0
\end{bmatrix}
\begin{bmatrix}
U_n \\
P_n
\end{bmatrix}
= \begin{bmatrix}
\tilde{F}_n \\
0
\end{bmatrix}
\tag{4.18}
\]

This is a well-studied problem that comes from solving time-dependent Stokes flow. One can discretize it in whatever way is convenient and accurate (e.g., FEM).

### 4.2.4 Interpolating from Grid to Particles

Following the derivation thus far, to find the new velocities on the particles from the values on the grid, one must solve the underdetermined linear system,

\[ U_n = I_n u_n. \tag{4.19} \]

A good approximate solution is found by interpolating the new velocity from the grid to the particles \( u_n^{\text{PIC}} = I_n U_n \). However, even if \( I_n \) is a pseudo-inverse of \( I \), the data in the nullspace must be included in the dynamics in some way, or they will accumulate noise. Consistent with other methods, we smooth this noise by regularization. Both ‘FLIP-style’ \( u^{\text{FLIP}} \) and ‘PIC-style’ \( u^{\text{PIC}} \) solutions are computed and combined:

\[ u^{\text{FLIP}} = u_{n-1} + I_n (U_n - U_{n-1}) \tag{4.20} \]

\[ u^{\text{PIC}} = I_n U_n \tag{4.21} \]

\[ u_n = \theta u^{\text{FLIP}} + (1 - \theta) u^{\text{PIC}} \tag{4.22} \]

\[ \theta = 2^{-\left(t_n - t_{n-1}\right)/\tau}. \tag{4.23} \]

The equation for \( \theta \) produces a stable and time step-independent decay of unresolved features and noise with a half-life \( \tau \) \cite{67}. We set \( \tau \) to a short, but natural timescale for the problem (e.g. 0.5 seconds for a simulation lasting several seconds).

Note that the pressure on the particles, \( p_n \), was not used, so it need not ever be stored.
4.2.5 Reseeding

Over time, initially uniform particle distributions become less uniform and require some reseeding. For efficiency, if there are many particles in a cell (e.g. > 15), we remove some particles from the simulation. For accuracy, when the number of particles in a grid cell becomes small (e.g. < 5), we add additional particles.

Newly created particles require a synthetic history of velocities (and possibly pressures §4.3), for use in the multistep method. We create this history by projecting the history of all the existing particles to the grid and interpolating those history values at the new particles’ locations. This results in \( k \)th order accurate approximate history values on new particles.

If a \( \Theta(1) \) fraction of the particles are added and/or removed each time step, then this error could add up to a \((k - 1)\)th order error term over the course of a simulation. In that case, \((k + 1)\)th order accurate history would need to be constructed during reseeding. However, we do not observe this loss of accuracy in our experiments. Reseeding typically improves the result.

This reseeding operation is quite expensive. The entire particle state needs to be projected to the grid, and then interpolated on to new particles. Alternative approaches would be an avenue of future research.

4.2.6 Startup

Multistep methods all have a problem with startup, because the high-order methods need a history over multiple time steps in order to work. We tackle this by using adaptive time step sizes and adaptive order. For a simulation with desired time steps of size \( \Delta t \), we begin with a step of size \( \Delta t^2 \) using BDF1. Each subsequent time step uses a higher-order BDF method until there is enough history for BDF4, then the step sizes are increased (by a factor of no more than 1.2\( \times \)) until the desired step sizes are reached.

4.3 A Projection Method

Solving SPD systems, in terms of just \( u \) or \( p \), is often preferable to solving the indefinite Stokes system (4.18). This sections derives a projection method with one SPD problem for implicit viscosity and another SPD problem for a pressure
correction.

Using a history of pressures from past time steps, stored on the particles, polynomial extrapolation gives a predicted pressure

\[ \tilde{p}_n = \sum_{i=1}^{k} \varepsilon_i p_{n-i} \]  

(4.24)

that we project to the grid

\[ \tilde{P}_n = I^i_n \tilde{p}_n. \]  

(4.25)

By substituting this for pressure in the viscous part of the Stokes equations (4.18), one arrives at a simpler SPD problem

\[ (\beta_0 - \nu \nabla^2) \tilde{U}_n = -\nabla \tilde{P}_n + \tilde{F}_n \]  

(4.26)

for an approximate velocity field \( \tilde{U}_n \). This solution can then be projected to satisfy the divergence-free constraint by finding \( \phi_n \) such that

\[ U_n + \nabla \phi_n = \tilde{U}_n \]  

(4.27)

\[ \nabla \cdot U_n = 0 \]  

(4.28)

which is a SPD Poisson problem \( \nabla^2 \phi_n = \nabla \cdot \tilde{U}_n \).

It remains to find the corrected pressure \( P_n \) from \( \tilde{P}_n \) and \( \phi_n \). Substituting this form of \( U_n \) into equation (4.17) and subtracting equation (4.26), one arrives at an equation for the updated pressure

\[ \nabla P_n = \nabla \tilde{P}_n + (\beta_0 - \nu \nabla^2) \nabla \phi_n \]  

(4.29)

In the continuous domain, \( \nabla^2 \nabla = \nabla \nabla^2 \), so we commute these operators, and formally integrate once, to compute the corrected pressure,

\[ P_n = \tilde{P}_n + (\beta_0 - \nu \nabla^2) \phi_n \]  

(4.30)

Once discretized, \( \nabla^2 \) and \( \nabla \) may not exactly commute, in which case this pres-
sure does not exactly satisfy the Stokes equations. However, it does satisfy the Stokes equations with a modified right hand side. Using discretized linear operators $G \approx \nabla$, $A_u \approx \beta_0 - \nu \nabla^2$ (for vector fields), and $A_p \approx \beta_0 - \nu \nabla^2$ (for scalar fields), the modified right hand side term is

$$\tilde{F}_n + (A_u G - GA_p) \phi_n.$$  

(4.31)

With $k$th order accurate discretizations for $A_*$ and $G$, the extra term $(A_u G - GA_p)$ is a $k$th order accurate approximation to 0. If it were identically zero, the projection method would exactly solve the Stokes system, independent of the extrapolated pressure. However, in general, an accurate pressure prediction is necessary.

Note that $\tilde{P}_n$ is a $k$th order accurate extrapolation, but local $(k + 1)$th order accuracy is required to achieve global $k$th order convergence. We expect that the updated $P_n$ after projection is $(k + 1)$th order accurate, as has been shown in similar contexts in other works [38, 84, 174].

If there are boundary conditions in terms of pressure (e.g. the free-surface $p = 0$ condition), then this projection method is cumbersome. These boundary conditions need to be enforced via $\phi$, but the conditions become more complicated because it is $(\beta_0 - \nu \nabla^2) \phi$ that gets added to the pressure, not $\phi$ directly. We do not address this issue, and simply prefer the coupled algorithm for this case.

The projection scheme is not exactly implementing BDF, so the stability requirements are different and unclear. The viscous and projection steps are stable operations, so we do not expect any significant additional instabilities from them.

This approach solves two simple linear systems instead of one more complex Stokes solve, but it is less accurate and requires a history to be stored for pressure. Whether or not this is the correct engineering trade-off will depend on the application. We prefer not to use this approach. While Stokes solvers are more complicated, they have seen a lot of research and effective solution strategies exist [135].

4.4 Results

In our 2D implementation, the PDEs are solved with a spectral discretization on a regular grid, with periodic boundary conditions. Interpolating from the particles to
the grid is done with moving least squares (MLS), while interpolating back is done with bi-cubic interpolation. These are similar to the choices in the explicit method [67]. In the spectral discretization, the projection and Stokes algorithms produce identical results, because the discrete gradient and Laplacian commute. Therefore we only show results using the Stokes approach.

### 4.4.1 Taylor Green Flow

The 2D Taylor-Green vortex is a closed form solution to incompressible Navier-Stokes

\[
\begin{align*}
    u(x, y) &= F(t) \sin(x) \cos(y) \\
    v(x, y) &= -F(t) \cos(x) \sin(y) \\
    F(t) &= \exp(-2\nu t)
\end{align*}
\]

in domain \( \Omega = [0, 2\pi]^2 \). This looks simple when written in the Eulerian coordinates, but is actually non-trivial behaviour on the particles as they move through the fluid. We simulate this problem to measure convergence of our method.

Error in velocity, on the grid, is measured at time \( t = 3.0 \). Figure 4.2 shows...
the convergence of the method, both with and without reseeding, and with ($\nu = 0.5$) and without ($\nu = 0$) viscosity. In every case, we observe approximately $4^{th}$ order convergence. In the inviscid case, reseeding causes a minor decrease in $L_\infty$ convergence at high resolutions.

### 4.4.2 A Complex Rayleigh Taylor Instability

To generate some more interesting dynamics, we add a Boussinesq buoyancy term to the dynamics. The PDE is the same as before, with forcing term $f = -\rho \nabla \Pi$ using fluid ‘density’ $\rho$ and gravitational potential energy $\Pi$. We evolve $\rho$ passively through the fluid, $\frac{D\rho}{Dt} = 0$, except for PIC/FLIP regularization as applied to the velocity. This regularization is necessary, since Rayleigh-Taylor instabilities tend to generate strong discontinuities in the density field, and our interpolation operators overshoot significantly around discontinuities.

Our setup uses periodic boundary conditions, with gravitational potential $\Pi = \cos(x) + \cos(y)$ in a domain $[0, 2\pi]^2$. The initial fluid density is

$$\rho = -2 \tanh(3 \cos(x) + 3 \cos(y) + 0.6 \cos(4x + 1) \cos(4x + 0.5) + 1.5)$$

which is designed to generate strong Rayleigh Taylor instabilities with complex behaviours and no symmetries. The simulation uses a 255 $\times$ 255 grid, $\Delta t = 1/40$, regularization half-life of $\tau = 0.2$ (for velocity and density), and was simulated from $t = 0$ to $t = 15$. Computation on a 4-core Intel i7 with 8GB of RAM took approximately 0.7 seconds per time step, of which 75% is spent interpolating from the particles to the grid. Figure 4.3 shows several frames of the resulting simulation.

**Multistreaming** In this experiment, we also tried allowing multistreaming by applying the $\Delta r$ update to particle positions directly, without interpolating to the grid and back. In this case, the multistreaming simulation develops noise at some points in the flow that subsequently cause extreme oscillations and velocities that pollute the entire simulation. We do not recommend allowing multistreaming.
Figure 4.3: Fluid density in a complex scenario, at time $t = 0, 1.5, 4.5, 15$. A spatially varying gravitational potential causes Rayleigh-Taylor instabilities and pulls the buoyant (white) fluid to the center.
4.5 Discussion

This project is a second exploration into globally high-order accurate PIC, following up on Edwards and Bridson [67]. As intended, this approach addresses several shortcomings on the previous work: it is able to handle the constrained dynamics of the velocity-pressure formulation of Navier-Stokes, and its semi-implicit nature allows it to take large time steps despite the presence of a stiff viscosity term.

There are numerous outstanding questions left by this work. First, we’ve neglected boundary conditions. We expect that adding boundary conditions to the grid solve will be sufficient, with little or no modification to the parts of the algorithm working with the particles. There are also still outstanding questions about the effect of reseeding. We would like a better theoretical or experimental understanding of whether or not reseeding needs to be done at a higher order of accuracy than the remainder of the algorithm, or ways to work around that requirement if necessary.

4.6 Conclusion

High-order accuracy is possible for PIC simulations of Navier-Stokes. The key is to treat the problem as a DAE and apply techniques for such problems. By using different multistep methods, the linear and nonlinear portions of the problem are separated.

Some of these ideas (using spatially high-order operators) have already been fruitfully applied in a graphics setting, as presented in Chapter 3.
Chapter 5

Free-Surface Geometry

This chapter proves the correctness of the continuous collision detection CCD algorithm by Brochu et al. [36]. The full paper, which presents the CCD algorithm and demonstrates its utility on several applications, is in Appendix A. The reader may prefer to read this appendix before the remainder of this chapter. It includes a more gentle introduction to the concepts, reasoning, and value of the approach.

In a simulation of water for computer animation, the free surface is the primary visual element. It is often the only place where the fluid motion is directly visible: an accurate representation of the surface is important. Using a Lagrangian triangle mesh that tracks the surface, as in Chapter 3, is one natural way to do this. A triangle mesh has good temporal coherence, has no problem representing and tracking thin and sharp features, and is a memory-efficient representation of the surface.

The surface of a fluid is never self-intersecting. This is a desirable invariant to maintain in the numerical model as well. In order to maintain this invariant for a surface represented by a moving triangle mesh, one ultimately needs methods for detecting collisions between moving mesh elements (point-vs-triangle and edge-vs-edge collisions). This is the continuous collision detection (CCD) problem.

The predicate form of CCD, which determines (true or false) whether a pair of elements collide, may have false-negatives (missed collisions) or false-positives (extraneous collisions) due to some form of approximation or simply floating-point rounding errors. Depending on the application, these errors may or may not be
acceptable, but they are certainly never desirable. We present the first exact CCD
algorithm for moving triangles.

The currently most popular CCD method [123] solves a cubic equation for the
time at which the elements are coplanar, then interpolates the geometry to that
time, and checks if the elements are intersecting (a simpler predicate). This ap-
proach writes the solution in terms of a series of intermediate constructions that
have irrational values that cannot be exactly expressed in floating point. This is es-
sentially impossible to implement without introducing rounding errors. Typically,
tolerances are added to every test to reduce false-negatives, but this introduces
false-positives and requires parameter tuning. All methods, prior to ours, suffer
from similar inexactness problems.

Our new approach is exact. It has no false-positives or negatives. Given
floating-point inputs, it always computes the correct result, as if computed with
exact arithmetic. Since the publication of this approach, several papers have pub-
lished other methods with similar exactness guarantees [150, 161].

Our approach transforms the CCD question into a root-counting problem. For
example, consider edge-edge collisions. Let the function \( f(\theta_1, \theta_2, t) \rightarrow \mathbb{R}^3 \) take the
barycentric coordinate \( \theta_1 \) of a point along the first edge and \( \theta_2 \) of a point along
the second edge, and return the vector from the first point to the second point at
time \( t \). If this function returns \( 0 \), then those points are coincident, indicating that a
collision is occurring.

There is a relationship between the number of roots of \( f \) in a domain \( \Omega \), and
whether or not the image of the domain boundary \( f(\partial \Omega) \) ‘contains’ the origin
point. More precisely, we characterize this with a lemma:

**Root Parity Lemma.** Suppose \( \Omega \subset \mathbb{R}^n \) is an \( n \)-polytope.

Suppose \( \vec{F} : \Omega \rightarrow \mathbb{R}^n \) is \( C^2 \), has \( p < \infty \) roots in \( \Omega \), has no roots on \( \Gamma = \partial \Omega \), and
has non-singular Jacobian at each root.

Suppose \( R \) is a ray from \( \vec{0} \) to infinity. Call any point \( \vec{x} \in \Gamma \) such that \( \vec{F}(\vec{x}) \in R \)
a crossing point, then the crossing number \( q \) is the number of crossing points.
Suppose there are no degenerate crossing points (defined later).

Then, \( p \equiv q \mod 2 \).

For many applications, element pairs with an even number of collisions can be
safely ignored. As a consequence, we need not count the number of roots directly. Instead, we measure the crossing number.

For the particular \( f \) that appears in the CCD problem, the surface \( (\Gamma) \) is composed of bilinear patches, for which we describe exact ray-patch intersection tests. From these tools, we assemble an exact CCD algorithm.

This work was published in SIGGRAPH 2012 [36]. My contribution to this work is the proof of correctness. It proves the root parity lemma, which is fundamental to the technique. The root parity lemma does not apply in all cases, so it also proves that the algorithm produces the correct output even in degenerate cases when that lemma does not apply. The proofs are in the next section of this chapter. Given my limited contribution to the remainder of the paper, it is included only as Appendix A.

The exact method described here was essential to achieve reliable results with the simulations in Chapter 3.

5.1 The Root Parity Lemma and Proof of Correctness

5.1.1 Outline

We present the root parity lemma to relate the roots of a function to the action of that function on the domain boundary. First, we prove the analog of the lemma for piecewise linear functions, then for more general \( C^2 \) functions.

For a linear function, on a simplicial domain, the root-party lemma is easy to show: either the simplex has one root and its image contains the origin, it has zero roots and its image does not contain the origin, or the function/geometry is degenerate.

For a piecewise linear function, defined on a simplicial mesh, each simplex can be analyzed as above. We then show that the crossing points contributed by each simplex in the mesh add up appropriately.

Finally, we use the good behavior on piecewise linear meshes to prove the result for \( C^2 \) meshes. We approximate the function with its piecewise linear interpolant on a mesh. When this mesh is sufficiently fine, there is a one-to-one mapping between the interpolant’s roots and the roots of the original function. We show
this by putting a small simplex around each root, and sufficiently small simplices elsewhere. Furthermore, by putting each crossing-point directly into the mesh, we guarantee a one-to-one correspondence between the crossing points of the original function and its interpolant.

The intersection function used in the CCD algorithm is a well-behaved $C^2$ functions, so we argue the correctness of our algorithm on top of the lemma.

5.1.2 Piecewise Linear Functions

Before proving the root parity lemma, we prove a related result for piecewise linear functions.

**Root Parity for Piecewise Linear Functions.** Suppose $\Omega \subset \mathbb{R}^n$ has a simplicial decomposition by mesh $M$.

Suppose $F : \Omega \mapsto \mathbb{R}^n$ is continuous, linear within each simplex of $M$, has $p < \infty$ roots in $\Omega$, has no roots on $\Gamma = \partial \Omega$, and is one-to-one in a small ball around each root.

Suppose $R$ is a ray from 0 to infinity. Call any point $x \in \Gamma$ such that $F(x) \in R$ a crossing point, then the crossing number $q$ is the number of crossing points. Suppose that the ray is not tangent to $F(\Gamma)$ at the image of any crossing points, and that $q < \infty$.

Then, $p \equiv q \mod 2$.

Suppose the hypotheses are true. Then the image under $F$ of any simplex in this mesh is also a simplex. Therefore the image of the entire mesh is also a simplicial mesh, $M'$, though it may be self-intersecting. The function $F$ is uniquely defined by the position of the vertices of $M'$, and roots of $F$ are given by simplices in $M'$ that contain the origin.

We may make various simplifying assumptions about $M'$. Such an assumption will be without loss of generality if, for any mesh $M'$, a perturbation exists that modifies it to satisfy these conditions without changing the parity of the number of roots or the parity of the crossing number for the ray.

Assume the origin does not lie on any facets of $M'$. This is possible because $F$ is invertible around each root. Therefore a small perturbation exists which will
push the origin from being on a facet to being in only one of the adjacent simplices; therefore not changing the number of roots.

Assume also that $M'$ has no degenerate simplices. That is, each simplex has volume. Simulation of Simplicity [65] addresses almost exactly this problem. A similar argument applies here. The only additional concern is that we do not modify the number of roots. Let $\varepsilon > 0$ be the distance from the origin to the nearest facet. Since the perturbation can be arbitrarily small, no vertex needs to be moved more than $\varepsilon$, so the number of roots does not change.

Now, consider a ray $R$ from $0$ to infinity satisfying the conditions of the root parity lemma. Such a ray exists because $F(\Gamma)$ is smooth almost everywhere and does not include $0$. Define a hit to be an intersection of the ray with the boundary of a simplex. If the ray intersects a facet shared by two simplices, then this is two hits. Each simplex can contribute hits.

First, consider a simplex from $M$ with no roots in it. The image of this simplex does not contain the origin. The ray crosses its boundary either zero or two times. This simplex contributes an even number of hits.

Second, consider a simplex from $M$ containing a root. The image of this simplex contains the origin in its interior. Consequently, the ray intersects its boundary once, contributing an odd number of hits.

Summing up all the hits, only simplices with roots contribute odd parity to the sum, so the parity of the number of hits equals the parity of the number of roots. Likewise, the parity of the number of hits equals the parity of the crossing number. This is because any intersection of the ray with an interior facet contributes two hits. Only the boundary facets, coincident with $\Gamma$, contribute odd parity. So the parity of the number of roots equals the parity of the number of crossings, as was to be shown.

5.1.3 The Root Parity Lemma

**Root Parity Lemma.** Suppose $\Omega \subset \mathbb{R}^n$ is an $n$-polytope.

Suppose $F : \Omega \rightarrow \mathbb{R}^n$ is $C^2$, has $p < \infty$ roots in $\Omega$, has no roots on $\Gamma = \partial \Omega$, and has non-singular Jacobian at each root.

Suppose $R$ is a ray from $0$ to infinity. Call any point $x \in \Gamma$ such that $F(x) \in R$
a crossing point, then the crossing number $q$ is the number of crossing points. Suppose that $F(\Gamma)$ is smooth at the image of any crossing points, that the ray is not tangent to $F(\Gamma)$ at any these points, and that $q < \infty$.

Then, $p \equiv q \mod 2$.

Suppose the hypotheses of the root parity lemma are true. Then, let the entire domain $\Omega$ be tessellated with a simplicial mesh $M$, as is possible for a polytope. Let each simplex have circumradius less than $\delta_{\text{out}}$, and let each root of $F$ be at the centroid of a regular simplex. We take the existence of such a mesh to be trivial. Let $\tilde{F}$ be the piecewise linear interpolant of $F$ on the mesh $M$. We argue below that there exists such a mesh for which $\tilde{F}$ and $F$ have the same number of roots, the same crossing number, and $F$ satisfies the hypotheses of the previous section. From this it will follow that the root parity lemma is true.

**Roots**

In this section, we show that if the mesh is sufficiently fine, then in any simplex, either $F$ and $\tilde{F}$ both have roots, or neither $F$ or $\tilde{F}$ have roots.

Let $x^*$ be an arbitrary root of $F$, and let $\Sigma$ be the simplex containing it. Since this simplex is regular, it has inradius $\delta_{\text{in}} = \kappa \delta_{\text{in}}$ with constant $\kappa$. In addition to the functions $F$ and $\tilde{F}$, we introduce $\hat{F} = J(x - x^*)$ which is the linear approximation to $F$ about the root (i.e. $J = \nabla F(x^*)$).

Clearly $\hat{F}(x^*) = 0$, so $\hat{F}$ has a root in $\Sigma$. Now, consider a point $q$ on the surface of $\Sigma$.

$$
\| \hat{F}(q) \|_2 = \| J(x - x^*) \|_2 \\
\geq \| J^{-1} \|_2^{-1} \| x - x^* \|_2 \\
\geq \delta_{\text{in}} \| J^{-1} \|_2^{-1}
$$

Therefore the image of $\Sigma$ under $\hat{F}$, which is also a simplex, has its surface at least $\delta_{\text{in}} \| J^{-1} \|_2^{-1}$ away from the origin.

By Taylor’s Theorem, we have,

$$
\| F(q) - \hat{F}(q) \| \leq c_1 \| q - q_i \|_2 \leq c_1 \delta_{\text{out}}^2
$$
where \( c_1 < \infty \) is a constant related to \( c_t \). Similarly, by the approximation quality of linear interpolants (for proof, see e.g. [134]), we have

\[
\|F(q) - \hat{F}(q)\| \leq c_2 \delta_{\text{out}}^2
\]

where \( c_2 < \infty \) is another constant related to \( c_t \). These can be combined to get the relationship,

\[
\|\hat{F}(q) - \bar{F}(q)\| \leq c_3 \delta_{\text{out}}^2.
\]

The origin is at least \( \delta_{\text{in}} \| J^{-1} \|_2^{-1} \) away from the surface of \( \hat{F}(\Sigma) \), and \( \hat{F}(\Sigma) \) is no more than \( c_3 \delta_{\text{out}}^2 \) away from \( \hat{F}(\Sigma) \). Since \( \hat{F}(\Sigma) \) is also a simplex, it follows that if

\[
\delta_{\text{in}} \| J^{-1} \|_2^{-1} > c_3 \delta_{\text{out}}^2 \iff 1/(\kappa c_3 \| J^{-1} \|_2) > \delta_{\text{out}},
\]

then \( \hat{F}(\Sigma) \) will also contain the origin, and \( \bar{F} \) will also have a root in \( \Sigma \). Since \( \kappa c_3 \| J^{-1} \|_2 \) is some constant bounded away from zero, we can choose such a \( \delta_{\text{out}} \). Consequently, both \( F \) and \( \bar{F} \) have a single root in \( \Sigma \), and \( \hat{F} \) is locally invertible there. Since there are finitely many roots, this constraint on \( \delta_{\text{out}} \) can be met at all roots by some constant \( \delta_{\text{out}} > 0 \).

Now we show that in the other simplices of the mesh, \( \bar{F} \) has no roots. Continuing with the point \( q \) on the surface of the simplex around a root, we find that

\[
\|F(q)\| \geq \|\hat{F}(q)\| - c_1 \|q - x\|^2 \\
\geq \|J^{-1}q - x\|^2 - c_1 \delta_{\text{out}}^2 \\
\geq \|J^{-1}\|^{-1} \|q - x\|^2 - c_1 \delta_{\text{out}}^2 \\
\geq \|J^{-1}\|^{-1} \delta_{\text{in}} - c_1 \delta_{\text{out}}^2 \\
\geq \|J^{-1}\|^{-1} \delta_{\text{in}} - c_1 \kappa^2 \delta_{\text{in}}^2 \\
\geq \|J^{-1}\|^{-1} \kappa^2 \delta_{\text{out}} - c_1 \delta_{\text{out}}^2 \\
\geq \alpha \delta_{\text{out}}^2 - c_1 \delta_{\text{out}}^2
\]

where \( \alpha > 0 \) is the minimum value over all roots of \( \|J^{-1}\|^{-1} \kappa^{-1} \).

Let \( S \) be the union of the simplices which contain roots of \( F \), as have already been addressed. Then, \( \Omega_0 = \bar{\Omega} \setminus \bar{S} \) is the remainder of the domain, including the
boundaries. In $\Omega_0$, $\|F\| > F_{\text{min}} > 0$. For sufficiently small simplices, the minimum value of $\|F\|$ will occur on the surface of one of the simplices surrounding a root. For simplicity, assume that this is the case. So, $F_{\text{min}} > \alpha \delta_{\text{out}} - c_1 \delta_{\text{out}}^2$ everywhere outside the simplices surrounding the roots. Then at all points $x$ outside the root-simplices $\|\tilde{F}(x) - F(x)\| \leq c_2 \delta_{\text{out}}^2$, and so

$$
\|\tilde{F}(x)\| \geq \|F(x)\| - c_2 \delta_{\text{out}}^2 \\
\geq F_{\text{min}} - c_2 \delta_{\text{out}}^2 \\
> \alpha \delta_{\text{out}} - c_1 \delta_{\text{out}}^2 - c_2 \delta_{\text{out}}^2 \\
> \alpha \delta_{\text{out}} - \delta_{\text{out}}^2 (c_1 + c_2)
$$

For sufficiently small $\delta_{\text{out}}$, we have $\|\tilde{F}(x)\| > 0$ because $\alpha > 0$. So, outside of the simplices surrounding the roots, $\tilde{F}$ has no roots.

It follows that $F$ and $\tilde{F}$ have the same number of roots.

**Crossing Points**

To show that $\tilde{F}$ also has the same crossing number as $F(\Gamma)$, we construct two new functions $H : \Gamma \mapsto \mathbb{R}$ and $G : \Gamma \mapsto \mathbb{R}^{n-1}$ by decomposing $F$ into its first component and the remainder: $F = [H, G]$. So $H(x) = F_1(x)$ is the first component of $F(x)$ and $G(x)$ is components 2 through $n$ of $F(x)$.

Without loss of generality, by a simple rotation, let the ray be the positive $x_1$ axis. In that case, $H$ measures the distance along the ray of the closest point to $F(x)$. Whereas, $G$ measures a vector-distance from $F(x)$ to the closest point on the ray. Consequently, a point $x$ is a crossing point of $R$ and $F(\Gamma)$ iff $H(x) > 0$ and $G(x) = 0$.

Now, consider the functions $\tilde{G}$ and $\tilde{H}$ defined as above, but using $\tilde{F}$ instead of $F$. As before, $x$ is a crossing point of $R$ and $\tilde{F}(\Gamma)$ iff $\tilde{H}(x) > 0$ and $\tilde{G}(x) = 0$. Notice also, $\tilde{H}$ and $\tilde{G}$ are the linear interpolants of $H$ and $G$ on the boundary elements of the mesh, which form an $n-1$ dimensional simplex mesh in $\Gamma$. We can use similar arguments as above to establish an exact correspondence between the crossing points of $R$ and $F(\Gamma)$ and the crossing points of $R$ and $\tilde{F}(\Gamma)$ by matching
the roots of $G$ and $\bar{G}$ and the signs of $H$ and $\bar{H}$ at those roots.

$G$ may not be locally invertible at all of its roots, so we take a different approach than above for $F$. Add all of the crossing points of $R$ and $F(\Gamma)$ as vertices to the mesh; by hypothesis there are a finite number of them. This does not contradict the earlier construction of a single simplex around each root of $F$, because $F$ has no roots on $\Gamma$. With these vertices in the mesh, $H(x) > 0$ and $G(x) = 0$ implies $\bar{G}(x) = 0$. Let any simplex containing a root of $G$ be small enough that it contains only one. By construction, it will be at a vertex, the root-vertex. All the other vertices of this simplex form an $(n-2)$-face, the far-face. By using anisotropic simplices around the roots, the far-face can always be distance $\delta > 0$ from the root, but fit in an arbitrarily small ball of radius $r$. Consequently, $\|G\| > \alpha > 0$ on the far-face. A described earlier, with a sufficiently fine mesh, $\bar{G}$ will have no roots on the far-face, and consequently no roots anywhere in the simplex, except at the root-vertex.

Let $\varepsilon$ be half the minimum magnitude of $H$ at any root of $G$. When $-\varepsilon < H < \varepsilon$, then $\|G\| > \delta > 0$. So, in a sufficiently fine mesh, $\bar{G} \neq 0$. This follows from the same bound used above to analyze $\bar{F}$. When $H < -\varepsilon$, in a sufficiently fine mesh we get $\bar{H} < 0$. Finally, when $H > \varepsilon$, a sufficiently fine mesh will have $\bar{H} > 0$ and, outside of the simplices constructed above, $\bar{G} \neq 0$. Combined with the results above, we conclude that $F$ and $\bar{F}$ have the same crossing points.

Combined with the earlier result, we have that $F$ and $\bar{F}$ have the same number of roots, and exactly the same crossing points. So we have shown that the root parity lemma is true.

### 5.1.4 Topological Degree

The root parity lemma is closely related to proofs and concepts from topological degree theory. In some sense, it is a specialization, and our proof is subsequently tailored for our application.

Topological degree defines a multidimensional generalization of the winding number. One definition of topological degree is the sum of the signs of the Jacobian determinants at all the roots of $F$. For non-singular roots, the parity of topological degree is the same as the parity of the number of roots, which is what we want to
measure. We refer the interested reader to the text by O’Regan et. al [118] for more details about topological degree.

While the definition above is in terms of the roots in the domain’s interior, an equivalent expression depends only on the boundary. In fact, it can be calculated by summing ±1 (depending on some property of \( F \) and its derivatives) at each intersection between a ray and the image of the boundary. Such an approach is described, for example, by Aberth in his text on numerical methods [2]. Because we only care about parity, our algorithm can simply count the number of intersections.

### 5.1.5 Proof of the Collision Algorithm

The function that the algorithm uses is \( C^2 \) with bounded curvature, defined in a polytope domain. When the input geometry to the collision detection algorithm produces a function \( F \) satisfying the hypotheses of the root parity lemma, then the correctness of the algorithm follows trivially.

However, \( F \) doesn’t always satisfy the hypotheses. It is always \( C^2 \) and it always has bounded second derivative. However, it may have roots on \( \Gamma \), it may have infinitely many roots, and it may have a singular Jacobian at any root. We argue here that the algorithm still does the right thing for collision detection in this case.

If \( F(x) = 0 \) on \( \Gamma \), then \( F(\Gamma) \) is at the origin. So, the ray’s vertex lies on the surface. The algorithm immediately identifies this as a collision without bothering with ray crossings.

If \( F \) has an infinite number of roots, then because of the multi-affine form of \( F \), it must have a root on \( \Gamma \), and this case is detected as above.

Otherwise, if \( \nabla F \) is not invertible at a root, then there exists a small perturbation to \( F \) strictly in the interior of the domain, such that it is not singular at any roots and is unchanged on \( \Gamma \). This perturbation does not affect the initial and final configuration of the mesh or the linear trajectories of the extremes of the mesh elements: it only causes an arbitrarily small adjustment to the interpolated trajectories. Thus, it will not change whether or not a significant collision has occurred, and the algorithm returns the correct result.

The root parity lemma also has conditions on the ray that the algorithm must meet. The only non-smooth regions of \( \Gamma \) are the edges. The algorithm traces a new
ray when an intersection with an edge is detected. If the crossing number is infinite, then the ray must hit the edges, which is detected as above. The requirement that the ray not be tangent is handled by the ray-patch parity algorithm, which returns the correct (even) parity in that case.

We note that while this proof contains several cases involving perturbing \( F \) or the ray, the algorithm does not need to do this. This is only a conceptual perturbation for use in the proof.

5.2 Discussion of the Odd-Parity Counting Argument

The claim made in the paper is that if a pair of mesh elements (a triangle and point, or two edges) whose vertices are moving on constant speed, linear trajectories collide an even number of times over the course of a time step, we can safely ignore the collisions (i.e. no collision resolution is required).

Although we do not have a rigorous proof for the correctness of this approach, we argue here that at the very least, a few desirable properties can be maintained. This model will clearly result in different behaviour than detecting and resolving every collision, however most collision processing systems already make simplifying assumptions such as not solving collisions in order of simulation time, resulting in an approximate yet plausible solution.

5.2.1 No Self-Intersections are Introduced

Suppose we are given a mesh that is intersection-free at \( t = 0 \), and has at least one edge-triangle intersection at \( t = 1 \). If collisions occur at the time boundaries (\( t = 0 \) or \( t = 1 \)) or element boundaries (e.g., at the endpoint of an edge, or the edge of a triangle) our algorithm always identifies this regardless of parity, and reports a collision. Otherwise, every collision changes the edge-triangle pair from an intersecting state to a non-intersecting state, or vice-versa. To switch from non-intersecting at \( t = 0 \) to intersecting at \( t = 1 \), there must have been an odd number of total collisions, which our algorithm detects via the root-parity argument, and reports a collision.
5.2.2 Disjoint Volumes Remain Disjoint

Suppose we have two closed meshes which define disjoint volumes at $t = 0$ (i.e. one mesh does not intersect or lie inside the other). Then at $t = 1$, if one mesh lies inside the other, our CCD algorithm would report a collision. A vertex outside a closed surface must collide with the surface an odd number of times to end up inside the surface. If there is an odd number of collisions between the vertex and the surface as a whole, then there must be at least one triangle with an odd number of collisions against the vertex.
Chapter 6

Discretely Discontinuous Galerkin Coarse Grid for Domain Decomposition

6.1 Introduction

Many discretizations of elliptic partial differential equations (PDEs) lead to sparse symmetric positive definite (SPD) linear systems of the form \( Au = f \). For large 3D problems, iterative solvers such as preconditioned conjugate gradient (PCG) are usually necessary. For a mesh with elements of diameter \( h \), the condition number often grows quickly as \( h \to 0 \): the quality of the preconditioner becomes the crucial factor for efficiency and robustness.

With an optimal preconditioner, the linear system can be solved to desired precision in a time which scales linearly with the problem size. Two popular and related frameworks for (potentially) optimal preconditioning are multigrid (MG) [32] and domain decomposition (DD) [136] [153]; we focus on the latter in this chapter. The key component we present is a coarse discretization of the PDE, using larger elements of size \( H \), providing the coarse grid correction to accelerate global convergence.

A critical factor in selecting a solver is the question of how much domain
knowledge the preconditioner requires. Geometric approaches require the practitioner to re-discretize the PDE at multiple scales, which for irregular domains and/or coefficients may be challenging. In contrast, algebraic approaches work almost entirely with the matrix $A$. While algebraic methods may be more difficult to develop, they can provide benefits in both ease of use and in handling irregular problems.

The method we propose here is essentially algebraic, but uses additional discrete information: we ask for a small set of generating vectors that span the space of degree $p$ polynomials. We construct a coarse basis by algebraically partitioning the domain into subdomains and using the restriction of each generating vector to each subdomain as its own basis function. The resulting coarse space functions are piecewise-smooth, with jumps at subdomain boundaries. From this basis, we construct a coarse problem by Galerkin projection.

We derive an error bound on the solutions to the coarse problem, and show that it is a high-order accurate convergent coarse grid approximation for a variety of PDEs and discretizations. Convergence requires a limited coarsening factor $[H/h]$ and sufficiently large $p$. Combined with DD in a Krylov method, we observe the number of required iterations decreases rapidly with $p$, and has reduced dependence on $[H/h]$, e.g., maintaining optimal scaling in the case $h = H^2$.

For any finite resolution of the fine problem, our coarse bases may or may not be interpreted as discontinuous. However, in the limit as $h \to 0$ with $H$ fixed, they are equivalent to the bases used in discontinuous Galerkin (DG) methods. We call our coarse basis functions discretely-discontinuous, giving rise to the name “Discretely-Discontinuous Galerkin” (DDG) for the approach.

We provide both theoretical and numerical evidence that discretely-discontinuous Galerkin (DDG) provides a convenient tool for easily constructing highly effective coarse grid corrections for a wide range of problems, varying over the type of discretization (e.g., classic finite elements or finite differences), the domain (from Cartesian grids to adaptive unstructured meshes), and the underlying PDE (e.g.,

---

While using mesh coordinates in the algorithm instead of just the matrix $A$ may arguably go beyond the strictest definition of an ‘algebraic’ method, we emphasize that no geometric operations are required, no mesh connectivity is involved, and no knowledge of the underlying continuum equations is needed. Other algebraic schemes which similarly use slightly more information than the matrix $A$ include Vaněk et al.’s smoothed aggregation method [155] and Brezina et al.’s AMGe [28].
vector-valued elasticity and fourth-order biharmonic problems).

6.2 Related Work

Our approach is closely related to the aggregation-based algebraic methods for constructing a coarse basis. For a more thorough review of aggregation techniques in the MG context, we refer the reader to review paper by Stüben [143]. We review the most closely related ideas and the DD setting.

The performance of non-smoothed aggregation, like ours, depends critically on \( \frac{H}{h} \). The simplest aggregation algorithm produces a piecewise constant coarse space. If \( \frac{H}{h} = O(1) \), then this preconditioner applied to the Poisson problem has condition number bounded independent of \( h \) [127, 128]. However, the condition number grows with \( \frac{H}{h} \).

For elasticity and higher-order PDEs, a piecewise constant basis is insufficient. Better aggregation techniques have been derived by requiring additional user input: the vectors that span the (near-)nullspace of the PDE [155], e.g., the rigid modes for elasticity and the linear polynomials for biharmonic problems.

These techniques are already optimal, in the sense that the number of iterations is bounded independent of problem size. Improvements to the iteration count can come in the form of constant factor reductions and reduced dependence on \( \frac{H}{h} \) or geometric dependencies such as the PDE, domain, coefficients, etc. Many works present modifications to aggregation-based techniques that improve their performance in these ways.

Despite the optimal scaling of non-smoothed aggregation, when \( \frac{H}{h} \) is large, the aggregation-based coarse solution is a poor approximation to the actual solution, and convergence is slow. Galerkin projection finds a solution which is optimal in energy norm, but the near-discontinuities at subdomain boundaries dominate the energy. One way to reduce this dependence on \( \frac{H}{h} \) is to keep the aggregation basis but apply a non-Galerkin projection, as in over-correction methods that apply a scaling to the Galerkin solution [19, 26]. In practice, this significantly improves the results.

Alternatively, one can work to change the basis. Sala et al. [128] show that the subdomains used for aggregation can be smaller than those used for the DD
smoothing step. Following this idea, the associated term in the bound on the condition number reduces geometrically with $H$. This requires some additional work to come up with the extra partitions, and enlarges the size of the coarse problem.

Another alternative is smoothed aggregation, which smooths the basis functions, thus reducing the steep jumps at subdomain boundaries. For the Poisson problem, this transforms an $\frac{H}{h}$ term in the condition number bound into an $\frac{H}{d}$ term, where $d$ is the smoothing diameter [127]. This keeps the size of the coarse problem the same as basic aggregation, but requires additional work to smooth the basis, and can increase the number of nonzeros in the coarse matrix.

Our method reduces to non-smoothed aggregation if the only generating vector is the constant vector, and our method inherits the upper bounds proven for non-smoothed aggregation. We increase the performance beyond non-smoothed aggregation by using a higher-order basis, which creates a high-order accurate rediscretization of the input PDE. By using a $p^{\text{th}}$ order coarse basis we reduce the energy at subdomain boundaries from $\frac{H}{h}$ to $\frac{H^{p+1}}{h}$. We find the added power of higher-order bases greatly reduces the required number of iterations.

Beyond non-smooth aggregation, discontinuous functions have appeared within DD algorithms before. For example, the restricted additive Schwarz method generates discontinuities at subdomain boundaries and has improved performance relative to the corresponding smooth method [43]. The improved performance is explained by Efstathiou and Gander: the restricted algorithm converges in the continuous limit while the additive overlapping algorithm does not [70]. When using DG discretizations, discontinuities are already present in the fine problem. Previous works have developed DD solvers specifically for DG [61, 63], or agglomerated fine DG problems to construct coarse ones [16]. While our approach uses a basis like that of DG methods, it does not require that the fine problem be discretized with DG, but can be interpreted as a rediscretization of the problem using a DG basis on elements of size $H$.

6.3 Preliminaries

The DDG algorithm requires no geometric interpretation or information for implementation (the generating vectors which typically would contain a basis for
polynomials in the nodal coordinates are treated as a black box). However, our understanding and analysis is intimately tied to a geometric interpretation, so we frequently refer to the geometric properties for simplicity. For reference, table 6.1 lists the major symbols used throughout this paper. They are also each defined at first use.

Table 6.1: Common symbols for DD used throughout this chapter.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Au</td>
<td>input, fine grid system</td>
</tr>
<tr>
<td>F</td>
<td>input, generating basis (tall matrix)</td>
</tr>
<tr>
<td>R0</td>
<td>restriction to coarse space</td>
</tr>
<tr>
<td>R_i</td>
<td>restriction to overlapping subdomain i &gt; 0</td>
</tr>
<tr>
<td>A_0</td>
<td>coarse grid matrix</td>
</tr>
<tr>
<td>A_0u_0</td>
<td>coarse grid system</td>
</tr>
<tr>
<td>h</td>
<td>element diameter in fine grid</td>
</tr>
<tr>
<td>H</td>
<td>element diameter in coarse grid</td>
</tr>
<tr>
<td>([H/h])</td>
<td>coarsening factor</td>
</tr>
<tr>
<td>p</td>
<td>polynomial degree used in coarse basis</td>
</tr>
<tr>
<td>(\delta)</td>
<td>subdomain overlap, in geometric distance</td>
</tr>
<tr>
<td>(\Delta)</td>
<td>subdomain overlap, in algebraic graph distance</td>
</tr>
<tr>
<td>d</td>
<td>spatial dimension</td>
</tr>
</tbody>
</table>

6.4 The Coarse Grid

An effective coarse grid needs to be able to approximate the error left by the one-level DD method. After a single pass of one-level DD, the error is extremely smooth in each subdomain, but not across subdomain boundaries (Figure 6.1). From this structure, we are motivated to use piecewise higher-order polynomials which have a similar piecewise-smooth structure.

Our coarse approximation space consists of piecewise polynomials which are smooth within each subdomain, but have arbitrary jumps at subdomain boundaries. The construction uses several user-supplied vectors, arranged in the columns of F, that span a degree \(p\) polynomial space. For many discretizations, F can be easily built using the nodal coordinates of the mesh and the constant vector.

To construct the coarse restriction and coarse system matrix, we follow the same Galerkin projection as used in previous aggregation methods (e.g. [155]);
Figure 6.1: Error after smoothing with DD for a simple Poisson problem with random right-hand-side. One-level DD produces a piecewise-smooth error after one iteration. Top row: partition, error after DD smoothing. Bottom row: first and second x-derivative of error.

the difference is in our choice of $F$. The subdomains are built by partitioning the discrete domain into non-overlapping subdomains $\Omega_i$ (i.e. subsets of indices) containing approximately $(H/h)^d$ nodes for problems in dimension $d$. Unless otherwise indicated, all of our examples were partitioned using a graph-based discrete algorithm from the SCOTCH library [120].

The coarse basis for subdomain $i$ is spanned by the columns of $\phi_i = R_i^T R_i F$, 

72
% F(i,j) = input basis j evaluated at node i.
% Node i is in subdomain partition(i) (zero-based ← index).

function R0 = basis(F,partition)

k = size(F,2);
[i,j,s] = find(F);
Rt = sparse(i, partition(i)*k+j,s);
[Rt."]= qr(Rt,0); % optional orthogonalization.
R0 = Rt';

Figure 6.2: MATLAB code to build the restriction matrix R₀ from input vectors F and a partition.

where Rᵢ is the restriction to the i-th partition domain (with no overlap). The final coarse restriction is made by concatenating these basis vectors together as rows in R₀. To help with conditioning, we orthogonalize the rows of R₀, which may be done independently for each subdomain as there is no overlap. Orthogonalization is optional, but the remainder of the paper assumes R₀ is orthogonal to simplify the analysis. Figure 6.2 shows simple (albeit inefficient) MATLAB code for this construction. A robust implementation must also detect when φᵢ is not full rank, and discard columns as necessary.

The coarse matrix A₀ is constructed by Galerkin projection, A₀ = R₀AR₀ᵀ, and the coarse approximation to u (of Au = f) is given by R₀ᵀu₀ where A₀u₀ = R₀f.

The size of A₀ increases with p, both in rank Θ(pᵈ) and in the number of non-zeros Θ(pᵈᵈ). However, the block sparsity pattern of A₀ is independent of p. It has the same sparsity pattern as the subdomain adjacency matrix, but with each non-zero replaced with a small dense block with size dependent on p. This structure allows for efficient numerical linear algebra using dense storage and operations. The optimal choice of p, in terms of total work to solve the problem, will depend on both the problem at hand and details of the implementation, but we generally found p = 3, cubic polynomials, is a good default.
6.5 Coarse Grid Analysis

We show that, under moderate assumptions, the error of the coarse solution is bounded by

$$\|R_0^T u_0 - u\|_A \leq c H^{1+p-q} (1 + [H/h]^{q-1/2}) |u|_{W^{2,q}(\Omega)}$$  \hspace{1cm} (6.1)

for PDEs of degree $2q$, where $c$ is independent of $h$ and $H$, and $u$ is the piecewise smooth interpretation of $u$ defined in the next section. When $[H/h] = O(1)$ and $1 + p > q$, this error converges with high-order in $H$. A convergent coarse grid approximation naturally allows the coarse grid correction to capture all components of the error not handled by fine grid smoothing, increasing the efficiency of the preconditioner.

We present two arguments for convergence of the coarse grid. First, we present an argument for FEM discretizations leveraging the extensive theory surrounding FEM and Sobolev norms. Second, we give an alternative argument that depends only on some discrete algebraic properties, which must be shown for each particular discretization.

6.5.1 Error Bound Using Geometric Properties

Here we restrict our attention to the finite element method.

Let the domain $\Omega$ be partitioned into subdomains $\Omega_i$. Let $u$ be in the Sobolev space $W^{q}_{\infty}(\Omega)$ (i.e. it should have bounded $q^{\text{th}}$-order weak derivatives) and suppose $u$ is $C^{\infty}$ in each subdomain. Let $u_h \in W^{q}_{\infty}$ be a FEM interpolant of this function on some mesh. We assume that both the subdomains and the mesh elements satisfy all the usual regularity assumptions for meshes with elements of diameter $H$ and $h$ respectively. We represent $u_h$ with a discrete vector $u$, assuming a nodal basis, so $u_i = u(x_i)$. Furthermore, we assume that the FEM interpolant satisfies $\|u\|_{W^{q}_{\infty}} \leq c \|u_h\|_{W^{q}_{\infty}}$, which is true when $\|u\|_{W^{q}_{\infty}} = \|u_h\|_{W^{q}_{\infty}} + O(h)$ and $h$ is smaller than some $h_0$.

Let the PDE be given as a symmetric elliptic bilinear form, $a(u, v) = \int_{\Omega} k[u] \cdot k[v] d\Omega$, with some linear functional $k[\cdot]$ involving up to $q^{\text{th}}$ order derivatives. For example, $k = \nabla$ for the Poisson problem or $k = \nabla^2$ for the biharmonic problem. We assume
Figure 6.3: Regions $\Omega_{\text{cut}}$ and $\Omega_{\text{sub}}$ used for analysis. A simple finite element mesh is divided into three DD subdomains. The elements separating the subdomain interiors from each other and from the boundary conditions are $\Omega_{\text{cut}}$. In $\Omega_{\text{sub}}$, the error is small because the approximation is good. The error is larger in $\Omega_{\text{cut}}$, but the total area is small enough that it does not hinder convergence.

continuity $a(u, u) \leq c||u||^2_{W_q^2}$, where $c$ denotes an arbitrary constant independent of $h$ and $H$. Discretized with the FEM, $u^T A u = a(u_h, u_h)$.

Each FEM nodal point lies within exactly one subdomain, and has an associated basis function. In some areas, basis functions from multiple subdomains overlap. Let the union of all mesh elements containing these overlapping areas, plus any elements touching the boundary of $\Omega$, be $\Omega_{\text{cut}}$, and let $\Omega_{\text{sub}} = \Omega \setminus \Omega_{\text{cut}}$ (Figure 6.3). For the purposes of the proof, we introduce additional bilinear forms $a_{\text{cut}}(u, v) = \int_{\Omega_{\text{cut}}} k[u] \cdot k[v] d\Omega$ and $a_{\text{sub}}$ defined analogously, along with their discretizations $A_{\text{cut}}$ and $A_{\text{sub}}$. Note that $a = a_{\text{cut}} + a_{\text{sub}}$ and $A = A_{\text{cut}} + A_{\text{sub}}$.

If the mesh and subdomains are sufficiently regular and the FEM basis functions have the usual compact support, then the ($d$-dimensional) volume $\mu(\Omega_{\text{cut}}) \leq c[|h/H|]$. It is linearly dependent on $h$ because $\Omega_{\text{cut}}$ is in a band of thickness $ch$ around the subdomain boundaries, and inversely dependent on $H$ because that is the rate at which the total subdomain surface area grows.

As in any Galerkin scheme, the coarse solution $R_{0}^{T} u_{0}$ is the minimum error
solution in the energy norm over the entire coarse space. Therefore the error is bounded by that of any particular coarse vector, including \( v = R_0^T R_0 u \). Using this and the splitting,

\[
\| R_0^T u_0 - u \|_A \leq \| v - u \|_A \quad (6.2)
\]

\[
= (\| v - u \|_{A_{sub}}^2 + \| v - u \|_{A_{cut}}^2)^{1/2} \quad (6.3)
\]

\[
\leq |v - u|_{A_{sub}} + |v - u|_{A_{cut}} \quad (6.4)
\]

Before tackling either of these terms, consider what \( v = R_0^T R_0 u \) is. Because \( R_0 \) is orthogonal, \( R_0^T R_0 u \) is the \( l_2 \) projection of \( u \) onto the coarse space. By construction of the coarse space, \( v \) has a continuous interpretation \( v \) that is a degree \( p \) polynomial in each subdomain \( \Omega_i \). We can find \( v \) directly from \( u \) by a per-subdomain least-squares approximation of \( u \) by a degree \( p \) polynomial, minimizing the sum of the squared error at each of the FEM nodal points. Barring pathological distributions of mesh nodes, \( v \) will be a high-order approximation to \( u \), satisfying the same error bounds commonly derived for FEM interpolants on a mesh with elements of size \( H \).

Now, we can bound the discrete error in terms of the geometric functions. We cite the appropriate theorems from Brenner et al. [27] for Sobolev and FEM-interpolant inequalities. Looking first in \( \Omega_{sub} \), we find that the coarse polynomials are a high-order approximation:

\[
|v - u|_{A_{sub}} = |v_h - u_h|_{A_{sub}} \quad \text{error interior to subdomains} \quad (6.5)
\]

\[
\leq c |v_h - u_h|_{W^q_2(\Omega_{sub})} \quad \text{discrete and FEM energy are equal} \quad (6.6)
\]

\[
\leq c |v - u|_{W^q_2(\Omega_{sub})} \quad \text{continuity assumption} \quad (6.7)
\]

\[
\leq c |v - u|_{\tilde{W}^q_2(\Omega_{sub})} \quad \text{convergent FEM for sufficiently small } h \quad (6.8)
\]

\[
= c |v - u|_{\tilde{W}^q_2(\Omega_{sub})} \quad \text{broken semi-norm defined below} \quad (6.9)
\]

\[
\leq c |v - u|_{\tilde{W}^q_2(\Omega)} \quad \text{increasing domain only increases the norm} \quad (6.10)
\]

\[
\leq c H^{1+p-q} |u|_{\tilde{W}^{1+p}_2(\Omega)} \quad \text{Theorem 4.4.20 [27]} \quad (6.11)
\]

\[
\leq c H^{1+p-q} |u|_{\tilde{W}^{1+p}_\infty(\Omega)} \quad \text{2-norm vs. } \infty\text{-norm} \quad (6.12)
\]
Here the broken semi-norm $\tilde{W}_b^p(Q)$ on domain $Q$ is defined as a sum over subdomains:

$$|u|_{\tilde{W}_b^p(Q)} = \left( \sum_i |u|^a_{W_b^p(\Omega_i \cap Q)} \right)^{1/a}$$  \hspace{1cm} (6.13)

This is naturally extended to a maximum over subdomains for $a = \infty$.

Turning to $\Omega_{cut}$, the coarse polynomials are not a high-order approximation in the energy norm, because $u$ is not smooth and $v$ is not even continuous in this region. However, $\Omega_{cut}$ is small enough that $L^\infty$ bounds are sufficient.

$$|v - u|_{A_{cut}} \leq c H^{1 + p - q} (1 + [H/h]^{q-1/2}) |u|_{\tilde{W}_b^{1+p}(\Omega)}$$  \hspace{1cm} (6.14)

$$\leq c H^{1 + p - q} (1 + [H/h]^{q-1/2}) |u|_{\tilde{W}_b^{1+p}(\Omega)}$$  \hspace{1cm} (6.15)

$$\leq c H^{1 + p - q} (1 + [H/h]^{q-1/2}) |u|_{\tilde{W}_b^{1+p}(\Omega)}$$  \hspace{1cm} (6.16)

$$\leq c H^{1 + p - q} (1 + [H/h]^{q-1/2}) |u|_{\tilde{W}_b^{1+p}(\Omega)}$$  \hspace{1cm} (6.17)

$$\leq c H^{1 + p - q} (1 + [H/h]^{q-1/2}) |u|_{\tilde{W}_b^{1+p}(\Omega)}$$  \hspace{1cm} (6.18)

$$\leq c H^{1 + p - q} (1 + [H/h]^{q-1/2}) |u|_{\tilde{W}_b^{1+p}(\Omega)}$$  \hspace{1cm} (6.19)

$$\leq c H^{1 + p - q} (1 + [H/h]^{q-1/2}) |u|_{\tilde{W}_b^{1+p}(\Omega)}$$  \hspace{1cm} (6.20)

$$\leq c H^{1 + p - q} (1 + [H/h]^{q-1/2}) |u|_{\tilde{W}_b^{1+p}(\Omega)}$$  \hspace{1cm} (6.21)

$$\leq c H^{1 + p - q} (1 + [H/h]^{q-1/2}) |u|_{\tilde{W}_b^{1+p}(\Omega)}$$  \hspace{1cm} (6.22)

$$\leq c H^{1 + p - q} (1 + [H/h]^{q-1/2}) |u|_{\tilde{W}_b^{1+p}(\Omega)}$$  \hspace{1cm} (6.23)

Combining the two bounds (6.12) and (6.23) completes the error bound

$$\| R_T^T u_0 - u \|_{A} \leq c H^{1 + p - q} (1 + [H/h]^{q-1/2}) |u|_{\tilde{W}_b^{1+p}(\Omega)}$$  \hspace{1cm} (6.24)

Numerical experiments suggest this bound has the correct asymptotic form, excluding the term $|u|_{\tilde{W}_b^{1+p}(\Omega)}$, which was not computed experimentally.

### 6.5.2 Error Bound Using Algebraic Properties

We can derive a similar bound based purely on algebraic components. As before, let $v = R_T^T R_0 u$ and $A = A_{cut} + A_{sub}$ be a splitting into symmetric positive semi-definite components. This splitting need not correspond to the FEM defini-
tion given earlier. However, we require that \( A_{\text{cut}} = \begin{bmatrix} B_{\text{cut}} & 0 \\ 0 & 0 \end{bmatrix} \) with \( B_{\text{cut}} \in \mathbb{R}^{m \times m} \) and 
\[ m \leq c^{[\beta/\alpha]} h^{-d}. \]
This is usually true and plays the role of \( \mu(\Omega_{\text{cut}}) \) from the geometric proof.

We assume that for all \( u \in V \),
\[
\| A_{\text{cut}} \|_2 \leq ch^{-2q}, \quad (6.25)
\]
\[
\| u \|_\infty \leq ch^{d/2} \| u \|_2, \quad (6.26)
\]
\[
| u - v |_{A_{\text{sub}}} \leq cH^{1+p-q} \| u \|_2, \quad (6.27)
\]
and
\[
| u - v |_\infty \leq cH^{1+p} \| u \|_\infty. \quad (6.28)
\]

The constants \( c \) include the roughness term \( | u |_{W_1^p(\Omega)} \) from the geometric proof.

Geometrically speaking, the subspace \( V \) must be restricted to functions with bounded roughness.

Showing that these assumptions are true for a particular discretization could exploit geometric properties as in the previous section.

From these assumptions, the convergence argument follows the exact same structure as in the geometric case and we do not repeat it. The final error bound is similar to the above:
\[
\| R_0^T u_0 - u \|_A \leq cH^{1+p-q}(1 + [H/h]^{q-1/2}) \| u \|_2. \quad (6.29)
\]

### 6.5.3 Practical Considerations

The proof and our use of the coarse grid in practice are not entirely consistent with each other. The coarse grid is used to approximate the error after applying one-level DD. For PDEs with smooth coefficients, as \( h \to 0 \) with \( H \) fixed, the error after one-level DD is piecewise \( C^\infty \) as in the proof. For any finite \( h \), it is only an approximation as accurate as the discretization.

In problems with discontinuous coefficients, even as \( h \to 0 \), the error after one-level DD is not \( C^\infty \) in each subdomain. It has kinks where the coefficients have discontinuities. We tried matching the partition boundaries to the discontinuities, or using a piecewise generating basis \( F \) that matches the discontinuities. We observed optimal scaling even without these strategies, but either strategy significantly ac-
celerated convergence with high-order polynomials.

When we use non-trivial overlap between subdomains, the boundaries of the smooth regions do not line up with the discontinuities in the coarse space. This is easy to resolve by adjusting the coarse subdomains, but this introduces more variation in the coarse subdomains’ size and shape. In numerical experiments, better results were obtained by ignoring this inconsistency with the proof and keeping the original subdomain shapes.

The roughness term in the error bound can increase with \( p \), suggesting that the error can actually increase with \( p \). However, because increasing \( p \) always grows the coarse vector space, and the Galerkin solve is optimal in that space, increasing \( p \) never increases the error.

6.6 Domain Decomposition

We incorporate the coarse grid within a standard multiplicative algebraic DD framework.

The DD subdomains begin with the same partition computed for the construction of the coarse basis. From the partition, overlapping subdomains are algebraically constructed by expanding the partition to include nodes within graph distance \( \Delta \) in the graph defined by \( A \). The expanded subdomains \( \tilde{\Omega}_i \) overlap in geometric bands of size \( \delta \). For simple meshes and discretizations, \( \delta = (1 + 2\Delta)/h \). Unless otherwise indicated, we use minimal overlap \( \Delta = 0 \).

Let \( \tilde{R}_i \) be the restriction matrix, such that \( u_i = \tilde{R}_i u \) is the vector of the elements of \( u \) from subdomain \( \tilde{\Omega}_i \), i.e. \( \tilde{R}_i \) is a subset of the rows of the identity matrix. For each subdomain, the local problem uses the matrix \( A_i = \tilde{R}_i A \tilde{R}_i^T \), and we solve these subdomain problems exactly. Given a current approximation \( u_k \), processing subdomain \( i \) updates the approximation to

\[
 u_{k+1} = u_k + \tilde{R}_i^T A_i^{-1} \tilde{R}_i (f - Au_k).
\]  

(6.30)

Iterating over all subdomains and updating the approximation to \( u \) after each, we arrive at the algorithm for one-level multiplicative overlapping Schwarz.

To build a two-level method, we multiplicatively combine one pass of one-level Schwarz as a pre-smoother, the coarse problem solution, and another pass through
the subdomains as a post-smoother. The post-smoother is done in reverse order, making the entire operation symmetric and usable with PCG.

We give some experimental results with a three-level method operating in a V-cycle. We construct the three-level problem by taking the two-level algorithm and applying it again to the coarse matrix $A_0$ to make an even coarser matrix $A_1$. To do this, we need a coarsened version of the generating vectors, which are simply $F_0 = R_0 F$. The coarsened coarse problem is equivalent to directly coarsening the original problem with larger subdomains. We keep the ratio between physical element sizes in adjacent levels the same (i.e. $h/H_0 = H_0/H_1$). The algebraic overlap $\Delta$ used for smoothing is also the same at all levels.

### 6.6.1 Condition Number

For several special cases, our approach reduces to previously published aggregation approaches. For the Poisson problem using $p = 0$, Sala [127] showed that the additive variant of the preconditioner has condition number bounded by $O(1 + [H/h])$.

For elasticity and biharmonic problems with $p = 1$, our approach is essentially a non-smoothed two-level variant of the multigrid method described by Vaněk et al. [155]. Their coarse grid uses the zero-energy modes, which are $p = 1$ for biharmonic and a subset of $p = 1$ for elasticity. They later prove optimal convergence, but only for the Poisson problem [156].

We do not have a condition number bound showing the dependence on $p$ and $q$. However, increasing $p$ beyond the low-order choices in the literature increases the dimension of the coarse space, which does not have a negative effect on convergence – it can only increase the rate of convergence.

### 6.7 Numerical Experiments

We demonstrate the performance of our coarse grid and DD as a preconditioner for PCG on a variety of PDEs and discretizations. Unless otherwise noted, all problems are solved to a $10^{-9}$ reduction in residual after the first application of the preconditioner. The right-hand-side vector $f$ is a random Gaussian-distributed vector, and the initial guess for $u$ is $0$. For the sake of easier reporting, we consider all problems with uniform meshes to be scaled such that rank$(A) = n = h^{-d}$, where $d$ is
the spatial dimension. Consequently, $H^{-d}$ is the number of subdomains.

The graph of the logarithm of the residual vs. the iteration count is typically very straight. Therefore we measure not just the integer iteration on which the residual is first smaller than the tolerance, but also the fractional iteration count at which the linear interpolation of this graph meets the tolerance. We found this reveals a lot of otherwise hidden detail and these fractional iterations are shown in Figure 6.6. When PCG takes more than 1000 iterations, we stop the solve and report iteration bounds based on condition number estimates derived from the Lanczos coefficients computed during PCG. For converged problems, these bounds agreed very well with actual iteration counts.

We use the following problems:

(A) Poisson 3D. $\nabla \cdot \nabla u = f$ discretized on an $m \times m \times m$ regular grid with a 7-point finite difference stencil. One face of the cube has a Dirichlet boundary condition and the remainder are Neumann. For this problem, we partition using recursive inertial partitioning [151] so that the matrix need never be explicitly constructed. The first partition uses a randomly-oriented plane to
ensure irregularly shaped subdomains.

(B) **Smooth Poisson.** \( \nabla \cdot S \nabla u = f \) discretized with piecewise linear finite elements on a 2D unstructured triangle mesh of a circular annulus with outer radius 5 times the inner radius. Both inner and outer boundaries use Dirichlet conditions. The scalar function \( S = \exp(1 + \sin(\pi (x+y))) \) is smooth. See Figure 6.4.

(C) **Non-Smooth Poisson.** \( \nabla \cdot D \nabla u = f \) discretized as above, but with discontinuous \( D = S + 100J \) where \( J = \mathcal{H}(0.25 + \cos(\pi x) \cos(2\pi y)) \) with Heaviside step function \( \mathcal{H} \). \( J \) is an indicator function for two ‘materials’ in the problem. We used algebraic partitions that do not conform to the material boundaries, but we use generating vectors \( F \) that are piecewise polynomial with respect to the material domains. This doubles the number of columns in \( F \), but only subdomains that include the material boundary end up with additional coarse basis functions, so \( A_0 \) is only marginally larger. In practice, we observe optimal scaling even without this extra work and it makes no difference with the piecewise constant basis. However, with the piecewise cubic basis, this material-aware \( F \) reduces the iteration count by nearly one half.

(D) **High-Order Poisson.** \( \nabla \cdot \nabla \) discretized on the same mesh as problems (B) and (C), but with continuous piecewise cubic finite elements.

(E) **Elasticity.** \( (\nabla \cdot \nabla + \nabla \nabla ^\top)u = f \) with Dirichlet boundary conditions, with vector \( u \). This is discretized on a spatially-adaptive unstructured 2D triangle mesh (Figure 6.4) with piecewise linear FEM. For the generating vectors \( F \), we take the degree \( p \) polynomials in each component of \( u \).

(F) **Biharmonic.** \( \nabla^4 u = f \) on a regular \( m \times m \) 2D grid discretized with a 13 point finite difference stencil. All boundaries have homogenous Dirichlet and Neumann conditions. This problem uses an algebraic overlap \( \Delta = 1 \), since performance is quite poor with \( \Delta = 0 \). Also this problem is solved only to \( 10^{-6} \) reduction in residual, as the fine discretizations are very poorly conditioned causing PCG to break down before reaching \( 10^{-9} \) as used in the above.
Table 6.2 summarizes the results for solving these problems at different $h$, $H$, and polynomial coarse spaces from piecewise constant $P_0$ to cubic $P_3$. With fixed $[H/h]$, we observe near constant iteration counts, independent of the problem size, when using the two-level algorithm. Part of the increase in iteration count can be attributed to degrading partition quality with the algebraic partitioner. Experiments (not shown) with more structured partitioning of the structured meshes showed less variation in iteration counts. The three-level V-cycle does not perform nearly as well, but still appears to be sub-logarithmic in $n$.

Table 6.3 shows the wall-clock time spent on setup (excluding partitioning) and solution of some problems from Table 6.2. The 2D problems were solved with a MATLAB implementation that solved each subdomain problem with the “back-slash” operator on each iteration, but stored a factorization of the coarse grid. The 3D problems were solved with a parallel C++ implementation that solved the subdomains with successive over-relaxation and solved the coarse grid with conjugate gradient, preconditioned with incomplete Cholesky. Both implementations ran on a 32-core Intel Xeon E5-2690 with 256GB of RAM. In all cases, the bulk of the runtime is spent on the subdomain solves, despite the use of poorly-scaling solvers for the coarse problem. Furthermore, the runtime scales approximately linearly with problem size, as desired. This is visible in Table 6.4, which displays runtime normalized by problem size. Perfect linear scaling would be shown as constant normalized runtime. The major exception to the approximately linear scaling is the 3D solver, which scales faster than linearly. Given that the number of iterations are constant, but the normalized runtime is improving on larger problems, this seems likely to be due to overhead that is being over-emphasized on small problems.

To directly explore the value in using higher-order coarse bases, we solve a 1000 $\times$ 1000 biharmonic problem with varying $p$, a large coarsening factor $[H/h] = 125$, and $\Delta = 2$. The large coarsening factor is desirable for efficient parallel implementations, but significantly reduces the accuracy of the low-order coarse grids. As shown in Figure 6.5, we observe the number of iterations decreases rapidly with increasing $p$.

Note that increasing $p$ increases the size of $A_0$, so there are diminishing returns with large $p$. Nonetheless, significant reductions in problem size are achieved for
Figure 6.5: $p$-dependent convergence on a biharmonic problem. The number of required iterations decreases rapidly with polynomial degree used in the coarse basis (parameters: $h^{-1} = 1000$, $[H/h] = 125$, $\Delta = 2$). For comparison, when using $p = 1$ but aggregating using smaller subdomains, the iteration count is much higher for the same number of coarse variables.

all $p$. The least reduction, with $p = 10$, is $\text{rank}(A_0) = (1/284) \cdot \text{rank}(A)$. A similar effect occurs in aggregation techniques when the aggregation subdomains are smaller than the subdomains used in smoothing. We compare to this approach by using $p = 1$ but aggregating on smaller subdomains. The high-$p$ basis significantly outperforms this approach (Figure 6.5).

Our error bound does not strictly require that $[H/h] = O(1)$ in order to produce a convergent coarse grid, and accompanying optimal preconditioner. For Poisson ($q = 1$) using a piecewise linear coarse basis ($p = 1$) and substituting the relation-
ship $h = H^2$, we arrive at the convergent bound

$$\| R_0^T u_0 - u \|_A \leq c H^{1/2} |u|_{\tilde W^2(\Omega)}. \quad (6.31)$$

Figure 6.6 shows the Poisson problem B with these parameters. Each subdomain has a number of nodes equal to the number of subdomains, so the coarse matrix and the subdomain matrices are a similar size, which is an interesting point in the design space. As in all overlapping DD methods, the smoother is very sensitive to $H/\delta$. To keep it approximately constant, we set $\Delta = \lceil H/4h \rceil$. The minor saw-tooth pattern in the graph comes directly from the remaining variation in $H/\delta$. With higher polynomial degrees, $h = H^s$ for higher powers of $s$ should be possible.

### 6.8 Weighting for a Better Coarse Grid

The error bound presented previously has a dependency on $[H/h]$ that a geometric coarse grid would not have. This section presents some incomplete work towards an $h$-independent coarse grid.

The $h$-dependence is not due to the choice of basis, but is an artifact of the Galerkin projection. Using the same basis, but with the $L_2$-optimal approximation
to the solution, the coarse approximation is given by,

$$u_{L^2} = R^T R A^{-1} f.$$  

This coarse approximation has an $L_2$ error bound that depends only on $H^{p+1}$, as it is simple piecewise polynomial approximation on subdomains of size $H$.

This works well, but is obviously computationally unreasonable: it requires computing $A^{-1} f$ which is already the solution to the high-resolution problem. However, we begin here to analyze some non-Galerkin projection schemes.

Let $S$ be the restriction onto the orthogonal complement of the coarse space (the rowspace of $R$). So that $[r^T s^T] \in \mathbb{R}^{n \times n}$ is an orthogonal matrix. We can write:

$$u_{L^2} = R^T R A^{-1} f$$

$$= R^T R I A^{-1} f$$

$$= R^T R \begin{bmatrix} R^T & S^T \end{bmatrix} \begin{bmatrix} R \ A^{-1} & R \end{bmatrix}^T \begin{bmatrix} R & S \end{bmatrix} f$$

$$= R^T R \begin{bmatrix} R^T \ A^{-1} & R \end{bmatrix} \begin{bmatrix} R \ A^{-1} & R \end{bmatrix}^T \begin{bmatrix} R \ S \end{bmatrix} f$$

$$= R^T \begin{bmatrix} I & 0 \end{bmatrix} \begin{bmatrix} R \ A^{-1} & R \end{bmatrix} \begin{bmatrix} R \ A^{-1} & R \end{bmatrix}^T \begin{bmatrix} R \ S \end{bmatrix} f$$

$$= R^T \begin{bmatrix} I & 0 \end{bmatrix} \begin{bmatrix} R A R^T \ A SAT^T \ S A T^T S A T^T \end{bmatrix} \begin{bmatrix} R \ S \end{bmatrix} f$$

$$= R^T \begin{bmatrix} I & 0 \end{bmatrix} \begin{bmatrix} A_1^{-1} & B \ B^T & A_2^{-1} \end{bmatrix} \begin{bmatrix} R \ S \end{bmatrix} f$$

$$= R^T A_1^{-1} R f + R^T B S f$$

$$= u_R + u_S.$$
In many cases, the matrix $A$ can be factorized as $A = G^T G$ in a meaningful way. For example, if $A$ is the Laplace operator $\nabla \cdot \nabla$, then $G$ can be a discretization of the gradient $\nabla$. More generally, when evaluating the finite element method by quadrature, each row of $G$ can correspond to a quadrature point, with a weighted discrete evaluation of $k[u]$. Taking this view, we expect $G$ to be taller than $A$, at least as sparse, and easy to obtain. Using this factorization,

$$A_1 = R G^T W G R^T$$

and

$$W = I - G S^T (S A S^T)^{-1} S G^T = I - P_{GST} = P_{GST}^\perp$$

which is the projection onto the orthogonal complement of the column-space of $G S^T$ (which is different from the column-space of $G R^T$). There is a similar closed form for $B$, but we will not use it.

The first term ($u_R = R^T A_1^{-1} R f$) has the form that we would like: it has the same dimensions as the Galerkin coarse matrix, and the entire operation is symmetric. The second term ($u_S$) is asymmetric and undesirably directly involves the large unknown matrix $S$. Can $u_S$ be safely dropped?

The $u_S$ term represents the coupling between high-frequency terms in the right-hand side and low frequency terms in the solution. If $f$ were smooth, then $||S f|| = O(H^{p+1})$, due to the approximation power of the coarse basis, and we could hope to ignore $u_S$ based on that argument. Unfortunately, there is no reason to expect $f$ to be smooth: one-level DD leaves a smooth error, but not a smooth residual. Independent of $f$, numerical experiments suggest the second term is relatively small: $||R^T A_1^{-1} R||_2 \approx 8 ||R^T B S||_2$ for a Poisson problem with $h^{-1} = 40$, $H^{-1} = 4$, $p = 1$. More experiments show this scales as $O(H^2)$ and I conjecture that the $H^2$ decay comes from the approximation power of $R$, so this term could be even smaller for higher-order bases. Because of these (incomplete) arguments, dropping $u_S$ seems reasonable.

The remaining difficulty is that $W$ is too computationally difficult to construct as given. Fortunately, there are many other matrices that will have the same exact
effect as $W$. Because $W$ only ever acts on the low-rank matrix $GR^T$, its action outside that subspace has no effect. To make this explicit, we can characterize the freedom we have with $W$. Take the singular value decomposition of $GR^T$:

$$GR^T = U \Sigma V^T = \begin{bmatrix} U_1 & U_2 \end{bmatrix} \begin{bmatrix} \Sigma_1 & 0 \\ 0 & 0 \end{bmatrix} V^T = U_1 \Sigma_1 V^T$$

Then, write $W$ in that basis,

$$W = UXU^T = \begin{bmatrix} U_1 & U_2 \end{bmatrix} \begin{bmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{bmatrix} \begin{bmatrix} U_1 & U_2 \end{bmatrix}^T$$

It follows that $RG^TWGR^T = V \Sigma_1 X_{11} \Sigma_1 V^T$, and we see that only $X_{11}$ matters. The blocks $X_{12}, X_{21},$ and $X_{22}$ are completely free.

This leads to the question, does there exist a matrix equivalent to $W$ that is easy to construct or approximate? Ideally, a diagonal approximation to $W$ would be possible, as this would give $A_1$ the same sparsity pattern as the Galerkin approach.

The connection between diagonal dominance of $W$ and diagonal dominance of $X$ is not obvious. As a start, we make $X$ as diagonally dominant as possible by choosing $X_{12} = X_{21}^T = 0$ and $X_{22} = I$, and call this choice $W_1$. This is also equivalent to

$$W_1 = I - P_{GR^T}P_{GS^T}P_{GR^T}$$

In experiments, this matrix has nearly unit diagonal. It is diagonally dominant on all rows which don’t correspond to crossings of subdomain boundaries. On rows that do cross subdomain boundaries, the sum of the magnitude of off the diagonals appear to be approximately proportional to the norm of the rows of $GR^T$.

The $u_R$ term is related to a non-Galerkin projection using a weighted norm:

$$u_{LS} = R^T A^{-1}_1 RG^TWb = \arg\min_{u \in \mathbb{R}^y} \|Gu - b\|_W^2.$$ 

So, we call $W$ the weight matrix. The difference between $u_{LS}$ and $u_R$ comes is the right-hand-side. For $u_R$ (and the Galerkin approximation) the right-hand-side is $f = RG^Tb$, but for $u_{LS}$ it is $f_{LS} = RG^TWb$. When using approximations to $W$, numerical experiments suggest that $u_{LS}$ is generally a better approximation to $u$. 

88
than $u_R$. Unfortunately, $b$ is not available when applying iterative Krylov methods, such as PCG.

**A Diagonal Weight Matrix**  Consider the simple first-order finite-difference discretization of the 2D Poisson problem on a regular grid, with $G$ measuring finite-difference derivatives between two points. Then, let $W_D$ have ones on the diagonal when the corresponding row of $G$ does not touch multiple subdomains, and have $ch/H$ on the diagonal if the row crosses subdomain boundaries, where $c = 8$ seems to work in practice.

Using $W_D$ in place of $W$, the coarse matrix is easily shown to be a first order approximation ($O(h)$ when $H$ is fixed) to the early DG method by Babuška and Zlámal [12]. Proofs of convergence of that approach require an $H^{-2}$ term on the diagonal in order to dominate consistency errors [9], but we found performance to be better when using $H^{-1}$.

We experimented with algebraic methods of deriving diagonal weights with similar properties, without prior knowledge of $h$ or $H$. Multiple schemes based on the row-norms of $GR^T$ seem to have the same asymptotic form. There is a large design space, with no obvious advantages to draw us to any corner.

This approach appears to generalize to other second order elliptic PDEs, but it does not continue to work for higher-order PDEs. On the biharmonic problem, this weighting generates the necessary penalty term on discontinuities at subdomain boundaries, but is missing the required penalty term for discontinuities in gradient across boundaries.

In the special case of piece-wise constant bases, this approach is equivalent to over-correction methods [19, 26]. However, for higher-order bases ($p > 0$), over-correction incorrectly re-weights the interior of the subdomains where the basis functions are smooth. Re-weighting only disrupts the correctly-functioning Galerkin projection here. The approach using $W_D$ fixes this defect.

My analysis based on $W$ stops here. We found no further way to make use of these observations in a numerical method. Research along these lines is left for the future.
6.9 Discussion

We have presented an algebraic coarse grid construction that produces a convergent rediscretization for a wide variety of PDEs. It works for both scalar- and vector-valued problems, both second- and fourth-order PDEs, and both smooth and discontinuous coefficients. The high-order DG-like coarse basis is easy to construct algebraically, and Galerkin projection generates a high-order convergent coarse rediscretization of the input problem. Combined with DD and PCG, we observe convergence in a number of iterations nearly independent of problem size, as expected from the existing theory for aggregation methods. Furthermore, increasing the polynomial degree rapidly reduces the number of required iterations: the fastest solves used high-degree polynomials.

The relationship between this approach and simple aggregation is similar to the relationship between high-order FEM and low-order FEM. High-order methods are appropriate when the solution to the problem is very smooth – which is exactly the case after subdomain smoothing. Similar to the situation in FEM, the high-order approach often drastically outperform the low-order schemes.

There are a number of outstanding questions raised by this work. We have shown a bound on the error of the coarse grid, dependent on \( H, h, p \) and the smoothness of the solution. Ideally, we would have a thorough understanding of the relationship between all of the parameters \((H, p, h, \text{ and } \delta)\) and the condition number or number of iterations to converge. We leave closing this gap in the analysis for future work.

On the more practical side, the generating vectors \( F \) are not difficult to supply, but it would be more convenient to construct similar high-order coarse grids directly from \( A \). This becomes even more important on more complex problems with spatially-varying coefficients, for which a well-adapted basis can significantly improve performance. Also, our approach still has an undesirable dependency on \([H/h]\) that is not present in geometric methods. Following the connection between the coarse basis and DG, we have done some preliminary work on algebraically constructing a DG-like discretization that is independent of \([H/h]\), but with mixed success.
Table 6.2: Iterations of CG with DDG to solve various problems in $n$ variables. $P_0$, i.e. non-smoothed aggregation, is not expected to achieve optimal scaling for $\nabla^4$; the $P_0$ column is included for comparison only. Problems marked “×” are too small to use the three-level solver - the coarsest partition would be a single subdomain.

<table>
<thead>
<tr>
<th>$n^{1/d}$</th>
<th>Two-Level $H/h=10$</th>
<th>Three-Level $H/h=10$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$P_0$ $P_1$ $P_2$ $P_3$</td>
<td>$P_0$ $P_1$ $P_2$ $P_3$</td>
</tr>
<tr>
<td>$\nabla \cdot \nabla^3$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>36 20 15 12</td>
<td>35 23 18 15</td>
</tr>
<tr>
<td>80</td>
<td>41 20 16 13</td>
<td>51 28 21 18</td>
</tr>
<tr>
<td>160</td>
<td>44 21 16 13</td>
<td>61 30 23 19</td>
</tr>
<tr>
<td>320</td>
<td>46 22 16 14</td>
<td>65 31 23 19</td>
</tr>
<tr>
<td>640</td>
<td>46 22 16 14</td>
<td>65 31 23 19</td>
</tr>
<tr>
<td>$\nabla \cdot \nabla^2$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>41 19 14 11</td>
<td>51 25 20 15</td>
</tr>
<tr>
<td>400</td>
<td>44 19 14 11</td>
<td>60 26 19 16</td>
</tr>
<tr>
<td>800</td>
<td>48 20 15 12</td>
<td>64 27 20 17</td>
</tr>
<tr>
<td>1600</td>
<td>52 20 15 12</td>
<td>68 28 22 17</td>
</tr>
<tr>
<td>3200</td>
<td>52 21 16 13</td>
<td>71 29 22 17</td>
</tr>
<tr>
<td>$\nabla \cdot \nabla^2$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>43 19 15 12</td>
<td>56 27 22 16</td>
</tr>
<tr>
<td>400</td>
<td>46 20 15 13</td>
<td>64 28 22 19</td>
</tr>
<tr>
<td>800</td>
<td>48 21 16 13</td>
<td>67 31 23 18</td>
</tr>
<tr>
<td>1600</td>
<td>52 21 17 14</td>
<td>69 33 23 20</td>
</tr>
<tr>
<td>3200</td>
<td>58 21 17 14</td>
<td>69 31 24 19</td>
</tr>
<tr>
<td>$\nabla^2 P_2 - \text{FEM}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>47 22 17 14</td>
<td>58 29 21 17</td>
</tr>
<tr>
<td>400</td>
<td>53 22 17 14</td>
<td>68 29 22 19</td>
</tr>
<tr>
<td>800</td>
<td>53 24 17 14</td>
<td>73 31 23 19</td>
</tr>
<tr>
<td>1600</td>
<td>56 23 19 15</td>
<td>75 32 25 19</td>
</tr>
<tr>
<td>3200</td>
<td>67 24 18 15</td>
<td>83 32 24 20</td>
</tr>
<tr>
<td>Elasticity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>72 29 21 19</td>
<td>83 40 32 25</td>
</tr>
<tr>
<td>400</td>
<td>81 29 24 18</td>
<td>96 41 30 25</td>
</tr>
<tr>
<td>800</td>
<td>82 33 23 20</td>
<td>104 42 31 26</td>
</tr>
<tr>
<td>1600</td>
<td>76 32 24 20</td>
<td>107 44 33 27</td>
</tr>
<tr>
<td>3200</td>
<td>76 33 27 23</td>
<td>108 47 36 29</td>
</tr>
<tr>
<td>$\nabla^4$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>698 62 20 12</td>
<td>600 154 44 24</td>
</tr>
<tr>
<td>400</td>
<td>2900 68 21 12</td>
<td>3500 186 53 27</td>
</tr>
<tr>
<td>800</td>
<td>5700 77 25 15</td>
<td>6900 184 55 32</td>
</tr>
<tr>
<td>1600</td>
<td>9500 99 31 15</td>
<td>11000 246 70 30</td>
</tr>
</tbody>
</table>
Table 6.3: Runtimes for the DDG results in the middle column of Table 6.2 (two-level scheme with $H/h=20$). Each entry shows the wall-clock time in seconds and the percent of total runtime spent on coarse grid setup and solution. Problems marked “×” did not converge within 1000 iterations.

<table>
<thead>
<tr>
<th>$n^{1/3}$</th>
<th>$P_0$</th>
<th>$P_1$</th>
<th>$P_2$</th>
<th>$P_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>79 (0%)</td>
<td>55 (0%)</td>
<td>42 (0%)</td>
<td>32 (1%)</td>
</tr>
<tr>
<td>80</td>
<td>152 (1%)</td>
<td>87 (1%)</td>
<td>64 (2%)</td>
<td>53 (3%)</td>
</tr>
<tr>
<td>160</td>
<td>481 (2%)</td>
<td>238 (4%)</td>
<td>186 (6%)</td>
<td>160 (11%)</td>
</tr>
<tr>
<td>320</td>
<td>1207 (3%)</td>
<td>614 (8%)</td>
<td>525 (15%)</td>
<td>497 (27%)</td>
</tr>
<tr>
<td>640</td>
<td>7053 (5%)</td>
<td>3785 (16%)</td>
<td>3140 (25%)</td>
<td>3563 (43%)</td>
</tr>
<tr>
<td>200</td>
<td>12 (2%)</td>
<td>5 (2%)</td>
<td>4 (3%)</td>
<td>3 (7%)</td>
</tr>
<tr>
<td>400</td>
<td>48 (1%)</td>
<td>22 (2%)</td>
<td>17 (4%)</td>
<td>15 (8%)</td>
</tr>
<tr>
<td>800</td>
<td>246 (1%)</td>
<td>108 (2%)</td>
<td>87 (4%)</td>
<td>72 (9%)</td>
</tr>
<tr>
<td>1600</td>
<td>1283 (1%)</td>
<td>572 (2%)</td>
<td>423 (4%)</td>
<td>358 (9%)</td>
</tr>
<tr>
<td>3200</td>
<td>6491 (0%)</td>
<td>2767 (2%)</td>
<td>2107 (4%)</td>
<td>1711 (9%)</td>
</tr>
<tr>
<td>200</td>
<td>11 (1%)</td>
<td>5 (2%)</td>
<td>4 (5%)</td>
<td>4 (11%)</td>
</tr>
<tr>
<td>400</td>
<td>51 (1%)</td>
<td>22 (2%)</td>
<td>18 (5%)</td>
<td>16 (11%)</td>
</tr>
<tr>
<td>800</td>
<td>227 (1%)</td>
<td>100 (2%)</td>
<td>84 (6%)</td>
<td>72 (9%)</td>
</tr>
<tr>
<td>1600</td>
<td>1051 (1%)</td>
<td>496 (2%)</td>
<td>362 (6%)</td>
<td>320 (12%)</td>
</tr>
<tr>
<td>3200</td>
<td>7170 (0%)</td>
<td>3006 (2%)</td>
<td>2254 (4%)</td>
<td>1896 (10%)</td>
</tr>
<tr>
<td>200</td>
<td>18 (1%)</td>
<td>9 (1%)</td>
<td>6 (4%)</td>
<td>6 (5%)</td>
</tr>
<tr>
<td>400</td>
<td>92 (0%)</td>
<td>43 (1%)</td>
<td>32 (3%)</td>
<td>27 (6%)</td>
</tr>
<tr>
<td>800</td>
<td>440 (0%)</td>
<td>190 (1%)</td>
<td>149 (3%)</td>
<td>121 (7%)</td>
</tr>
<tr>
<td>1600</td>
<td>1902 (0%)</td>
<td>853 (1%)</td>
<td>635 (4%)</td>
<td>498 (8%)</td>
</tr>
<tr>
<td>3200</td>
<td>10223 (0%)</td>
<td>4123 (1%)</td>
<td>3183 (3%)</td>
<td>2706 (7%)</td>
</tr>
<tr>
<td>200</td>
<td>30 (0%)</td>
<td>15 (1%)</td>
<td>11 (3%)</td>
<td>9 (6%)</td>
</tr>
<tr>
<td>400</td>
<td>147 (0%)</td>
<td>60 (1%)</td>
<td>46 (3%)</td>
<td>42 (7%)</td>
</tr>
<tr>
<td>800</td>
<td>644 (0%)</td>
<td>263 (1%)</td>
<td>207 (3%)</td>
<td>172 (7%)</td>
</tr>
<tr>
<td>1600</td>
<td>2778 (0%)</td>
<td>1160 (1%)</td>
<td>883 (4%)</td>
<td>727 (8%)</td>
</tr>
<tr>
<td>3200</td>
<td>14958 (0%)</td>
<td>6202 (1%)</td>
<td>5017 (3%)</td>
<td>4625 (6%)</td>
</tr>
<tr>
<td>200</td>
<td>198 (0%)</td>
<td>57 (1%)</td>
<td>17 (1%)</td>
<td>11 (3%)</td>
</tr>
<tr>
<td>400</td>
<td>× 321 (0%)</td>
<td>86 (1%)</td>
<td>45 (4%)</td>
<td></td>
</tr>
<tr>
<td>800</td>
<td>× 1565 (0%)</td>
<td>484 (2%)</td>
<td>225 (5%)</td>
<td></td>
</tr>
<tr>
<td>1600</td>
<td>× 7900 (1%)</td>
<td>2171 (2%)</td>
<td>1012 (5%)</td>
<td></td>
</tr>
</tbody>
</table>
Table 6.4: Normalized runtimes for the DDG results in the middle column of Table 6.2 (two-level scheme with $H/h=20$). Each entry shows the wall-clock time (as in Table 6.4), normalized by problem size. The exact number shown is $1000000 \cdot \text{runtime}/n$. The percent of total runtime spent on coarse grid setup and solution is shown in parentheses. Problems marked “×” did not converge within 1000 iterations.

<table>
<thead>
<tr>
<th>$n^{1/3}$</th>
<th>$P_0$</th>
<th>$P_1$</th>
<th>$P_2$</th>
<th>$P_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>1234 (0%)</td>
<td>860 (0%)</td>
<td>656 (0%)</td>
<td>500 (1%)</td>
</tr>
<tr>
<td>80</td>
<td>297 (1%)</td>
<td>170 (1%)</td>
<td>125 (2%)</td>
<td>104 (3%)</td>
</tr>
<tr>
<td>160</td>
<td>117 (2%)</td>
<td>60 (4%)</td>
<td>45 (6%)</td>
<td>40 (11%)</td>
</tr>
<tr>
<td>320</td>
<td>37 (3%)</td>
<td>19 (8%)</td>
<td>16 (15%)</td>
<td>15 (27%)</td>
</tr>
<tr>
<td>640</td>
<td>27 (5%)</td>
<td>15 (16%)</td>
<td>12 (25%)</td>
<td>14 (43%)</td>
</tr>
<tr>
<td>200</td>
<td>292 (2%)</td>
<td>131 (2%)</td>
<td>102 (3%)</td>
<td>84 (7%)</td>
</tr>
<tr>
<td>400</td>
<td>297 (1%)</td>
<td>138 (2%)</td>
<td>106 (4%)</td>
<td>93 (8%)</td>
</tr>
<tr>
<td>800</td>
<td>384 (1%)</td>
<td>169 (2%)</td>
<td>136 (4%)</td>
<td>113 (9%)</td>
</tr>
<tr>
<td>1600</td>
<td>501 (1%)</td>
<td>223 (2%)</td>
<td>165 (4%)</td>
<td>140 (9%)</td>
</tr>
<tr>
<td>3200</td>
<td>634 (0%)</td>
<td>270 (2%)</td>
<td>206 (4%)</td>
<td>167 (9%)</td>
</tr>
<tr>
<td>200</td>
<td>272 (1%)</td>
<td>132 (2%)</td>
<td>108 (5%)</td>
<td>91 (11%)</td>
</tr>
<tr>
<td>400</td>
<td>320 (1%)</td>
<td>138 (2%)</td>
<td>112 (5%)</td>
<td>97 (11%)</td>
</tr>
<tr>
<td>800</td>
<td>354 (1%)</td>
<td>157 (2%)</td>
<td>131 (6%)</td>
<td>105 (13%)</td>
</tr>
<tr>
<td>1600</td>
<td>411 (1%)</td>
<td>194 (2%)</td>
<td>142 (6%)</td>
<td>125 (12%)</td>
</tr>
<tr>
<td>3200</td>
<td>700 (0%)</td>
<td>294 (2%)</td>
<td>220 (4%)</td>
<td>185 (10%)</td>
</tr>
<tr>
<td>200</td>
<td>426 (1%)</td>
<td>213 (1%)</td>
<td>156 (4%)</td>
<td>143 (5%)</td>
</tr>
<tr>
<td>400</td>
<td>570 (0%)</td>
<td>270 (1%)</td>
<td>197 (3%)</td>
<td>165 (6%)</td>
</tr>
<tr>
<td>800</td>
<td>682 (0%)</td>
<td>294 (1%)</td>
<td>231 (3%)</td>
<td>188 (7%)</td>
</tr>
<tr>
<td>1600</td>
<td>742 (0%)</td>
<td>333 (1%)</td>
<td>248 (4%)</td>
<td>194 (8%)</td>
</tr>
<tr>
<td>3200</td>
<td>996 (0%)</td>
<td>402 (1%)</td>
<td>310 (3%)</td>
<td>264 (7%)</td>
</tr>
<tr>
<td>200</td>
<td>750 (0%)</td>
<td>374 (1%)</td>
<td>272 (3%)</td>
<td>239 (6%)</td>
</tr>
<tr>
<td>400</td>
<td>916 (0%)</td>
<td>373 (1%)</td>
<td>286 (3%)</td>
<td>262 (7%)</td>
</tr>
<tr>
<td>800</td>
<td>1008 (0%)</td>
<td>411 (1%)</td>
<td>324 (3%)</td>
<td>269 (7%)</td>
</tr>
<tr>
<td>1600</td>
<td>1110 (0%)</td>
<td>464 (1%)</td>
<td>353 (4%)</td>
<td>290 (8%)</td>
</tr>
<tr>
<td>3200</td>
<td>1466 (0%)</td>
<td>608 (1%)</td>
<td>492 (3%)</td>
<td>453 (6%)</td>
</tr>
<tr>
<td>200</td>
<td>4941 (0%)</td>
<td>1437 (1%)</td>
<td>427 (1%)</td>
<td>273 (3%)</td>
</tr>
<tr>
<td>400</td>
<td>2008 (0%)</td>
<td>535 (1%)</td>
<td>283 (4%)</td>
<td></td>
</tr>
<tr>
<td>800</td>
<td>2445 (0%)</td>
<td>757 (2%)</td>
<td>351 (5%)</td>
<td></td>
</tr>
<tr>
<td>1600</td>
<td>3086 (1%)</td>
<td>848 (2%)</td>
<td>395 (5%)</td>
<td></td>
</tr>
</tbody>
</table>
Chapter 7

Conclusion

Fluid simulation in computer graphics depends on a many different mathematical tools. In this thesis, I developed new techniques for several important components of a modern liquid solver. The research chapters each include their own conclusion, discussing limitations and potential future work. Here, I extend that discussion to include the impact of the projects and other new research, and discuss the future of the general approaches used in the thesis projects.

Detailed Water with DG. This project was presented at SIGGRAPH 2014. People seemed generally excited about the results, but confused by the unfamiliar and intricate use of DG. As this was just presented last year, it is unclear what sort of impact it will have.

With this project, we set out to solve the resolution-mismatch problem with embedding high-resolution meshes in low-resolution grids. We did not completely eliminate these problems, and still had to include some non-physical surface regularization. In hindsight, using an approximation space that is not adapted to the surface shape may never be strong enough to completely solve the problem in an efficient manner. Some new work by Ando et al. [8] has started to investigate surface-adapted bases. While this project was in development, the need to completely solve this embedding problem in a physical way was slightly reduced by the publication of more physically-realistic regularization approach based on vortex sheets [20]. These approaches are completely complementary.
The power and flexibility of \( p \)-adaptivity went beyond my expectations and are applicable to far more than just surface resolution matching. There is a lot of room for future work based on this project, most obviously extending this to \( hp \)-adaptivity, which promises even more performance and flexibility.

**High-Order PIC.** This project achieved the goal of addressing some specific shortcomings of the high-order accurate approach by Edwards and Bridson [67]. However, it is difficult to assess its overall value and it seems unlikely that the resulting scheme will ever be used as-is.

I hope this work will inspire and inform future projects using more accurate formulations of PIC, FLIP, or MPM. In the engineering field, where MPM is most used, that seems unlikely to happen in the near future, because the scheme is so heavily modified relative to traditional MPM. The loss of discrete conservation properties could be the strongest impediment to adoption.

In the field of computer graphics, our approach is not so foreign. It is similar to the version of FLIP already used for computer graphics. Furthermore, conservation laws are not given as much importance by computer graphics practitioners. The MPM methods published in graphics venues so far [141, 142] have used the engineering/conservation form, but I wouldn’t be surprised to see the adoption of some ideas from Chapter 4. As a whole, PIC schemes are treated very flexibly in computer graphics, with lots of extensions and modifications from the standard engineering approach.

**Continuous Collision Detection.** The CCD paper (Chapter 5) seems to have brought the problem of inexact CCD algorithms to people’s attention. Since its publication, Wang [161] has shown how to prevent false-negatives while minimizing false-positives in the standard (constructive) CCD approach, by carefully tracking floating-point errors and introducing several other predicates to use when the error-bound shows the point-triangle test to be inaccurate. Tang et al. [150] derive an alternative exact algorithm, similar to the traditional approach, but with an exact reformulation. Their approach runs faster than that proposed in Chapter 5 and is conceptually more familiar to users of the standard approach. So far, there have
been no follow-up approaches using the root-parity reformulation, but given its
flexibility it would not surprise me if it sees future use as well.

Discretely Discontinuous Galerkin Coarse Grid. This project presents an alge-
braic coarse grid that works for a wide variety of problems, including high-order
PDEs. The approach is closely related to existing aggregation approaches, but has
significantly more flexibility due to its use of high-order convergence to combat
other sources of error.

While it has strictly better asymptotic error bounds than basic aggregation ap-
proaches, this project did not entirely accomplish the goals I wanted to achieve.
The convergence rate of the algorithm in Chapter 6 has a dependency on $[H/h]$, while
geometric DD has convergence rate independent of fine resolution $h$ and
coarse resolution $H$. The dependency on $[H/h]$ is common to many algebraic DD
schemes, including both aggregation and non-aggregation schemes (e.g. BDD [111]
and FETI [78]). Nonetheless, I hoped to avoid this undesirable property, and I’m
optimistic that similar strategies can achieve this goal.

This approach occupies an interesting niche in the design space between ge-
ometric methods and ‘fully’ algebraic methods, which work only with the input
linear system. Whether or not it will be used as is, I hope it can serve as a stepping-
stone towards even better techniques.

High-Order Methods. A common trend through many of the projects in this thesis
is the use of high-order accurate techniques.

High-order methods have a number of advantages over low order techniques.
The most obvious is the high-order convergence rate from which they get their
name. This often leads to increased accuracy for a fixed discretization resolution,
which hopefully translates in to reduces time and/or memory to achieve the same
quality of result. Furthermore, high-order methods often do more computation per
variable than low-order schemes, which works especially well on modern com-
puter architectures that have memory access times far slower than floating-point
operations [4, 45].

The character of the errors made by high-order methods can also be differ-
ent from that of low order methods. For example, in fluid problems, numerical diffusion/viscosity is a common numerical error in low order schemes. A high-order scheme may completely eliminate the usual diffusion term, resulting only in a ‘hyper-diffusion’ related to higher-order derivatives. In fact, many high-order methods achieve exact results in the case that the solution is a low-order polynomial. This may not sound interesting at first glance, but it includes many common scenarios; for example: parabolic ballistic trajectories, linear pressure in hydrostatic fluids, linear velocity in rigidly-moving bodies. Solving these base-cases incorrectly can sometimes result in obvious visual artifacts.

In exchange for these benefits, high-order techniques have several disadvantages. They are often not as stable, or the stability analysis is much more difficult. The algorithms are a lot more complicated and difficult to implement. At very low resolution, high-order methods are a lot more computationally expensive and may produce low-accuracy results. Finally, high-order schemes are not as flexible. It is difficult to write a modular system in which components can be added and removed without disrupting the high-order convergence. The weakest link in the chain ruins the global convergence rate.

High-order methods are far from ubiquitous in computer graphics, but they are making in-roads in some sub-problems. Second-order accurate boundary conditions in fluid problems have become common (e.g. [7, 18, 74]), because they show drastically improved results (reduced artifacts) relative to first-order techniques. In solving the advection problem, higher-order schemes (e.g. BFECC [64] and MacCormack [130]) have also gained a lot of traction, as have higher-order interpolation (e.g. cubic [79] and CIP [103]) schemes during advection. These techniques significantly reduce the numerical diffusion during advection, relative to the simplest first-order semi-Lagrangian technique. The use of high-order integration techniques for ODEs (e.g. Runge-Kutta methods) and spatial integration (e.g. Gaussian quadrature) are common as well.

The future of high-order methods in computer graphics remains to be seen. Given their potential advantages, and the trends in hardware architectures, it seems plausible that they will see increasing adoption.
Bibliography


[50] B. Cockburn and J. Gopalakrishnan. The derivation of hybridizable discontinuous Galerkin methods for Stokes flow. *SIAM Journal on*


[100] G. Kanschat. Divergence-free discontinuous Galerkin schemes for the
Stokes equations and the MAC scheme. *International Journal for
→ pages [11]


[102] P. Kaufmann, S. Martin, M. Botsch, and M. Gross. Flexible simulation of
deformable models using discontinuous Galerkin FEM. *Graphical Models*,
2008. → pages [18, 20]

without dimensional splitting. In *Computer Graphics Forum*, volume 27,
doi:10.1111/j.1467-8659.2008.01144.x → pages [97]

[104] B. Lambov. Interval arithmetic using SSE-2. In P. Hertling, C. M.
Hoffmann, W. Luther, and N. Revol, editors, *Reliable Implementation of
Real Number Algorithms: Theory and Practice*, volume 5045 of *Lecture

[105] G. Lapenta and J. Brackbill. Control of the number of particles in fluid and
MHD particle in cell methods. *Computer Physics Communications*, 87(1):

[106] F. Li, J. Pan, and C. Sinka. Modelling adhesive contact between fine

using material point method. *International Journal of Impact Engineering*,

*Proc. 28th Australasian Conference on Computer Science - Volume 38*,
Australian Computer Society, Inc. ISBN 1-920-68220-1. URL
http://portal.acm.org/citation.cfm?id=1082161.1082173 → pages [131]


111


Appendix A

Efficient Geometrically Exact Continuous Collision Detection

This appendix is a copy of Brochu et al. [36], with minor reformatting for inclusion here. This is the full paper for the approach outlined in Chapter 5. The content of Chapter 5 is the proof of correctness referred to in this paper.

A.1 Introduction

We consider continuous collision detection (CCD) as the process of detecting if a mesh moving between initial and final configurations comes into contact with itself at any point in time. CCD is a critical element of many algorithms for physical simulation, when a non-intersecting mesh invariant must be maintained, and for path planning, when the feasibility of a path must be guaranteed. Beyond efficiency, a good CCD algorithm should therefore be safe: no false negatives, i.e. missed collisions, can be tolerated. It should also be accurate in the sense of minimizing the number of false positives, i.e. non-collisions being flagged as collisions, for the effectiveness and efficiency of algorithms using CCD. This paper provides (1) the first CCD algorithm to guarantee safety and accuracy despite using rounded floating-point arithmetic, under the paradigm of Exact Geometric Computation described by Yap [170]: we compute the correct Boolean answer (collision or not) as if exact arithmetic were used. As part of our CCD algorithm we also present (2)
Figure A.1: Four layers of cloth folding over a spinning ball, with on-the-fly adaptive remeshing driven by curvature, provide an example where geometrically exact continuous collision detection has advantages over previous techniques.

We also introduce a new, efficient, geometrically exact ray vs. bilinear patch intersection parity test, which can be used to precisely determine if a point is inside a quad-mesh-bounded volume or not. We also introduce a new adaptive cloth simulation method which maintains intersection-free meshes despite remeshing, as an example of the practical advantage of geometrically exact CCD, and show there is no performance penalty for using geometrically exact CCD.

We restrict our attention to triangle meshes in 3D, with an intersection-free initial configuration, so CCD can be reduced to two primitive tests: does a moving point hit a moving triangle, or does a moving edge hit another moving edge? Note that we ignore the connectedness of the mesh: multiple meshes are treated as lumped together, so there is no difference between inter-object collision and self-collision. We further assume that vertices move with constant velocity during the time step and that triangles are linearly interpolated between their vertices at intermediate times. If a collision does happen, we do not require its precise time and
A.2 Related Work

A.2.1 The Cubic Solver Approach

The most popular current method for continuous collision detection for triangle meshes was introduced by Provot [123]. First a cubic equation is solved to determine coplanarity times, then the interpolated geometry is checked for overlap at these times to determine if a collision actually occurs. Bridson et al. [31] significantly reduced the number of false negatives due to floating-point error by introducing error tolerances in the root-finding algorithm used to solve the cubic equation and using a static distance query at the coplanarity times: collisions are reported if the mesh elements are within some small distance of each other.

However, the minimum error tolerances required for safe CCD are difficult to predict in advance. Especially in cases where the primitives remain nearly coplanar for the entire step, such as hair segments [131] sliding against each other on skin, cancellation error in simply computing the coefficients of the cubic can eliminate almost all precision in the rest of the calculation. (Of course, in constantly coplanar cases, the method breaks down entirely.) Even if the cubic is represented exactly, its roots are in general irrational and must be rounded to floating-point numbers. Completing the error analysis with further bounds on the construction of the intermediate geometry at the rounded coplanarity time, bounds on the calculated barycentric coordinates of closest points, and then bounds on the distance appears intractable. In practice, a usable tolerance can typically be found by trial and error for a large class of similar simulations (e.g. cloth animations), but different applications such as adaptive cloth, hair, or liquid surface tracking can require enervating per-simulation adjustment, which makes writing a general purpose library especially tricky.

The cubic approach naturally gives false positives if the tolerance is high enough to work. If the tolerance is too high (a definite possibility if restricted to single precision arithmetic, for example) this can seriously slow down or even completely...
stymie collision resolution: tuning the tolerance for a new simulation isn’t always easy.

A fully symbolic implementation could in principle resolve the above problems, apart from the degenerate constantly coplanar case, but the computational overhead would be drastic. In this paper we show a different approach to CCD can be fully safe and accurate without need for tuning, yet run just as fast.

A.2.2 Other Work in Continuous Collision Detection

Stam [138] extended the cubic solver approach to explicitly test if two mesh elements approach closer than a given distance during the time step, resulting in a sixth degree polynomial to solve for potential collision times. This helps to resolve the coplanar-motion degeneracy mentioned above, but poses an even less tractable rounding error analysis problem for safe CCD, suffers from the same false-positive issues, and is a heavier burden computationally.

Alternative methods for computing the time of possible collisions, such as conservative local advancement [148] offer potential speed-ups over the cubic solver approach, but don’t robustly deal with rounding error, relying on user-set tolerances to account for slight non-planarities in intersection/proximity testing.

The constant vertex velocity model underlying this paper and the cubic solver approach is perhaps the most natural for general deformable motions. However, for rigid bodies, constant linear and angular velocity of the entire model makes more sense — though the helical trajectories of vertices are somewhat more difficult to handle. Zhang et al. [173] demonstrate significant acceleration of conservative advancement using the Taylor model generalization of interval arithmetic, but again rely on user-set tolerances to cope with the inexact solve and rounding error.

Brochu & Bridson [33, 34] suggest using a simplicial space-time mesh to model the motion of the mesh, reducing CCD to simplex intersection tests in four dimensions. While these tests could be computed exactly with known determinant-based predicates, this approximation leads to an unintuitive model for the mesh geometry at intermediate times: mesh edges develop kinks and triangles develop folds; normals do not vary continuously over time. This precludes the use of CCD culling techniques which assume the geometry is linearly interpolated at intermedi-
ate times [149]. More importantly, the unusual model of motion causes unintuitive and undesired collisions. For example, two close but parallel triangles can move together with no collisions in the standard model, but their non-standard model can create the triangles in an inconsistent way, causing a hard-to-resolve collision.

Raytracing can be seen as a special case of CCD, generally easier since the geometry is static relative to the “motion” of the light ray. We highlight Ramsey et al.’s ray-bilinear patch test [125] as particularly relevant, as our 3D CCD test in fact relies on ray-bilinear patch intersection parity tests. However, our new approach is geometrically exact and fully robust, unlike Ramsey et al.’s constructive approach which is vulnerable to rounding error; on the other hand our test only provides the parity of the number of intersections, not their location.

The related problem of culling collision tests is very well studied in computer graphics. Several approaches using bounding volume hierarchies have been proposed, as well as culling using bounding boxes with regular grids, sweep-and-prune testing, and sweeping-plane testing: see Ericson’s book for example [75]. We observe that except for axis-aligned methods which only use comparisons (no arithmetic), the culling literature generally does not worry about verifiably handling rounding error. We briefly address this issue later, but emphasize our focus is the correctness of the core element vs. element test, not the efficiency of broader culling methods.

A.2.3 Exact Geometric Computation

The cubic solver approach is an example of a constructive geometric algorithm, in that intermediate geometric quantities are computed (such as planarity times and interpolated positions) and used in a sequence of calculations. In this, as in many other geometric tests, the necessary rounding analysis to get a provably correct algorithm (accounting for the errors in all intermediate quantities) is intractable while the symbolic or exact arithmetic version would be too slow (necessitating radicals in this case).

An alternative approach is to decompose a geometric test into a set of simpler predicates, providing discrete answers such as “does a point lie to the left, to the right or on a line?” rather than continuous values. Approximate continuous values
may be computed alongside, of course, but the discrete correctness of the algorithm as a whole relies only on the correctness of the discrete answers from the predicates. Several approaches to defining and implementing correct predicates exist; the most successful is the paradigm of Exact Geometric Computation (EGC). We recommend Yap’s article as an excellent review of the topic [170], and the CGAL project for examples of applications and ongoing research [1]. In brief, a geometrically exact predicate must return the same discrete answer as if computed with exact arithmetic (from the floating point input) even if under the hood it takes a faster approach. Our method is the first geometrically exact CCD test for general CCD, but exact predicates for other problems have long been used in graphics and elsewhere.

Building on previous work by Dekker [60] and Priest [122], Shewchuk presented practical systems for exactly evaluating a number of geometric predicates needed for Delaunay mesh generation [133]. These sign-of-determinant predicates are equally useful for detecting self-intersections for triangle meshes. When higher precision than provided by floating point hardware is required, the system uses floating-point expansions (the sum of a sequence of floats) leveraging fast floating-point hardware even for exact arithmetic.

In this paper we decompose CCD into a set of simplex intersection tests, based on the same standard sign-of-determinant tests, together with the evaluation of the sign of a simple polynomial function. No radicals or even divisions are required, making it straightforward to implement exactly using expansion arithmetic like Priest and Shewchuk. Furthermore, through the use of fast interval arithmetic filters, we can rapidly find the provably correct signs without need for high precision expansions in all but the most extreme cases, leading to highly efficient execution on average.

A.3 Continuous Collision Detection in 3D

In 3D CCD, there are two fundamental collisions tests: point-triangle and segment-segment. The input to each are the location of the vertices at the beginning and end of the time step. We denote the location in space of vertex \( i \) at the beginning of the time step as \( \mathbf{x}_i \), and its location at the end of the time step as \( \mathbf{\hat{x}}_i \). Then for each test,
we are given 8 points: $x_0, x_1, x_2, x_3, \hat{x}_0, \hat{x}_1, \hat{x}_2, \text{and } \hat{x}_3$. For convenience, we will normalize the time step so that $t \in [0, 1]$. The constant velocity model of motion gives the location of a vertex at an intermediate time as $x_i(t) = (1 - t)x_i + t\hat{x}_i$.

First consider the point-triangle test. In the following we will index the moving vertex with 0, and the triangle vertices as 1, 2, and 3. Any point on the triangle can be written as: $x(u, v) = (1 - u - v)x_1 + ux_2 + vx_3$, where $x_1$, $x_2$, and $x_3$ are the triangle corners, and $u, v \in [0, 1]$ with $u + v \leq 1$. The vector between a point on the moving triangle defined by the coordinates $(u, v)$ and the other vertex, at time $t$, can then be written as:

$$F(t, u, v) = x_0(t) - [(1 - u - v)x_1(t) + ux_2(t) + vx_3(t)]$$

$$= (1 - t)x_0 + t\hat{x}_0$$

$$- (1 - u - v)((1 - t)x_1 + t\hat{x}_1)$$

$$- u((1 - t)x_2 + t\hat{x}_2)$$

$$- v((1 - t)x_3 + t\hat{x}_3).$$

This is a tri-affine function which is zero precisely when the vertex lies on the triangle. The domain for the point-triangle test is therefore $\Omega = [0, 1] \times \{u, v \geq 0 | u + v \leq 1\}$, a triangular prism.

A similar tri-affine function can be defined for the segment-segment collision test, the vector between the point at fraction $u \in [0, 1]$ along one segment and the point at fraction $v \in [0, 1]$ along the other, at time $t \in [0, 1]$. The domain is then $[0, 1]^3$, the unit cube.

CCD then amounts to discovering if such a function has a root in the domain, a point in $\Omega$ which $F$ maps to 0. We make an important simplification: we report a collision if there is any zero on the domain of the boundary (i.e. at the initial or final time, or at any edge or endpoint of the geometry) or if there is an odd number of roots in the interior. We justify ignoring the case of a nonzero even number of interior roots by noting that an edge-triangle intersection cannot be introduced in the mesh if the total number of collisions between the edge and triangle has even parity. Likewise a vertex cannot enter or exit a closed mesh without either colliding with an edge or colliding with the triangles an odd number of times, and
therefore with at least one triangle an odd number of times, so our method cannot miss essential collisions in this category either; see the supplemental material for more discussion.

A.3.1 Determining Root Parity

Write the image of $\Omega$ under $F$ as $\Omega = \{ y \mid y = F(x), x \in \Omega \}$, and similarly $\Gamma = \{ y \mid y = F(x), x \in \Gamma \}$ for the image of the domain boundary $\Gamma = \partial \Omega$.

If $F$ were smooth and one-to-one, determining if $0 \in F(\Omega)$ could be done by counting the number of crossing of a ray to infinity from $0$ through the boundary image $F(\Gamma)$: an odd number of crossings indicates a root by the usual Jordan-Brouwer Theorem argument. However, our $F$ may not be one-to-one: the image of $\Omega$ can “fold over itself”. The more general Brouwer topological degree theory [118] can be applied in this case. It is the parity of the ray crossings with $F(\Gamma)$ that gives us the parity of the number of roots: the sum of an odd number of intersections for each separate root leads to an odd total if and only if there is an odd number of roots — with the proviso that if $0 \in F(\Gamma)$, i.e. we have a root on the domain boundary, we always report a collision. More formally:

Root Parity Lemma. Suppose $\Omega \subset \mathbb{R}^n$ is an $n$-polytope.

Suppose $F : \Omega \mapsto \mathbb{R}^n$ is $C^2$, has $p < \infty$ roots in $\Omega$, has no roots on $\Gamma = \partial \Omega$, and has non-singular Jacobian at each root.

Suppose $R$ is a ray from $0$ to infinity. Call any point $x \in \Gamma$ such that $F(x) \in R$ a crossing point, then the crossing number $q$ is the number of crossing points. Suppose that $F(\Gamma)$ is smooth at the image of any crossing points, that the ray is not tangent to $F(\Gamma)$ at any these points, and that $q < \infty$.

Then, $p \equiv q \mod 2$.

We offer a sketch of a proof of the lemma, and that it applies to the particular functions we need for CCD, in the supplemental material.

This lemma also describes our algorithm. We report a collision if the image of the boundary $F(\Gamma)$ passes through $0$, and otherwise cast a ray from $0$ in an arbitrary direction, and then count the number of crossings of the ray though $F(\Gamma)$, choosing a different direction and trying again if any crossings are tangent or lie on corners.
Figure A.2: Root parity test. In this case there are no roots in the domain, so the origin is outside of $F(\Omega)$.

Figure A.3: Root parity test with one and two roots in the domain. A ray cast from the origin will have odd and even parity, respectively.

Figures A.2 and A.3 illustrate this approach for the 2D case, showing cases where we have zero, one, and two roots in the domain.

We transform the boundary of these domains (cube or triangular prism) by the corresponding function $F$ to get a *generalized* hexahedron or prism, and test for ray crossings on each of their faces. The hexahedron has potentially non-planar bilinear patches for faces (the restriction of the tri-affine function to a face of the domain is bi-affine), and the prism is composed of three bilinear patches and two
triangles. Computing ray-triangle crossings can be done with exact arithmetic — however, we know of no prior practical method for quickly and exactly computing the crossings of a ray through a bilinear patches, or even the parity of the number of crossings which is all we need. We thus introduce an efficient method for exactly computing this parity.

A.3.2 Ray-Bilinear-Patch Crossing Parity Testing

We first define a continuous scalar function $\phi(x)$ which is positive if $x$ is on one side of the patch and negative on the other side, and to permit exact evaluation with floating-point expansions define it using only multiplication, addition, and subtraction — see Section A.7 for the derivation.

Next consider the tetrahedron spanned by the four corners of the bilinear patch. It is composed of two pairs of triangles, one pair corresponding to each side of the bilinear patch. For the “positive” triangle pair, any point $x$ on either triangle has the property that $\phi(x) \geq 0$, and vice versa for the “negative” pair of triangles. For the test, we consider two cases depending on whether the ray origin $0$ lies inside the tetrahedron or not — which can be determined directly from standard sign-of-determinant “orientation” predicates with the tetrahedron’s triangular faces.

If the ray origin $0$ lies inside the tetrahedron, we can determine the sign of $\phi(0)$ and replace the ray-patch test with two ray-triangle tests, using the triangles corresponding to the opposite sign of $\phi(0)$. Ray-triangle intersection can also be broken down into determinant predicates [85]. If there is an intersection between the ray and either triangle, then the ray must also pass once through the bilinear patch.

If instead the ray origin lies outside of the tetrahedron, we can use either set of triangles as an equivalent proxy for the bilinear patch. The parity of the number of intersections between the ray and the triangle pair matches the parity of the number of intersections between the ray and the bilinear patch.

Pseudocode for the test is given in algorithm 2; Figure A.4 illustrates the 2D analog. Since it relies only on determinants and the evaluation of $\phi$, i.e. just multiplication and addition/subtraction, the test can be evaluated using floating-point expansions to give the geometrically exact result.
Algorithm 2 Ray-bilinear-patch crossing parity

Given: Ray origin $0$, direction $R$, and a bilinear patch.

Form the tetrahedron from the bilinear patch corner vertices.

Let $F_1^+$, $F_2^+$ be the tetrahedron faces where $\phi \geq 0$.

Let $F_1^-$, $F_2^-$ be the tetrahedron faces where $\phi \leq 0$.

if 0 is inside the tetrahedron then

if $\phi(0) > 0$ then

return $\text{intersect}(0, R, F_1^-) \lor \text{intersect}(0, R, F_2^-)$

else

return $\text{intersect}(0, R, F_1^+) \lor \text{intersect}(0, R, F_2^+)$

else

▷ Use either pair of triangles return $\text{intersect}(0, R, F_1^+) \text{ XOR } \text{intersect}(0, R, F_2^+)$

\[ \phi(x) > 0 \]
\[ \phi(x) < 0 \]

Figure A.4: A 2D analog of the ray-vs-bilinear-patch parity test. Rays A and B have origins on the “negative” side of the patch, and so we test against the proxy geometry on the “positive” side. Ray A intersects both the patch and the proxy geometry, while B intersects neither. Rays C and D have origins outside the bounding simplex, and so can be tested with either proxy geometry.

A.3.3 Putting it Together

We now have the tools we need for determining the intersection parity of a ray versus a set of bilinear patches and triangles. For segment-segment CCD, our algorithm runs 6 ray-vs-patch tests for the faces of the hexahedron, and for point-triangle CCD, we run 3 ray-vs-patch and 2 ray-vs-triangle tests for the faces of the
triangular prism, and determine the parity of the total number of intersections. If we have an odd parity, we know there is an odd number of roots in the domain, and so we must flag this as a collision. Algorithm 3 shows the point-vs-triangle test (the segment-vs-segment test is analogous).

There are a few special cases we must watch for. If the origin lies exactly on a patch or a triangle (i.e. there is a root on the boundary of the domain), then we report the collision and skip the ray testing. This includes, for example, the fully degenerate cases of exactly planar motion (such as two edges sliding into each other along a flat surface) that entirely defeats the cubic solver. In our simulations this type of collision is vanishingly rare unless artificially induced.

We must also take care if the ray hits an edge shared between two patches, between two triangles (acting as proxy geometry in the ray-vs-patch test), or between a patch and a triangle. If this occurs, we will see two positive ray intersection tests. In the context of inside-outside testing, this may or may not be correct (see Figure A.5). Fortunately, since we are using exact arithmetic, we can precisely detect these degenerate cases (one barycentric coordinate will be exactly zero). In such a case, we simply choose a new ray direction at random and run the test again. Again, in our testing this happens only extraordinarily rarely.

---

**Algorithm 3** Point-triangle collision test

Given: corner vertices of the domain, \( X \)
Create ray \((0, R)\) with arbitrary direction \( R \)
\( S \leftarrow 0 \)

for \( i = 1 \rightarrow 3 \) do
  Form bilinear patch \( i \) with appropriate \( F(X) \)
  \( p_i \leftarrow \text{intersection parity of ray } (0, R) \text{ vs bilinear patch } i \)
  \( S \leftarrow S + p_i \)

for \( j = 1 \rightarrow 2 \) do
  Form triangle \( j \) with appropriate \( F(X) \)
  if Ray \((0, R)\) intersects triangle \( j \) then
    \( S \leftarrow S + 1 \)
  return \( S \equiv 1 \pmod{2} \)
Figure A.5: Degenerate ray intersections. Two rays, each hitting two segments at their common endpoint. If we are testing each segment individually, then in both cases the parity of ray intersections is even. Here the parity of intersection count cannot determine whether points A and B are inside the quadrilateral. Perturbing the rays slightly would produce the correct results in both cases.

A.4 Implementation

Fast implementation of exact intersection testing is crucial for making our approach practical. Computing intersections with expansion arithmetic is expensive, so we use a filter: we evaluate the determinants and $\phi$ first with interval arithmetic, only switching to exact expansions when the sign is indeterminate (the final interval contains zero). See Brönnimann et al. for the case of determinants [37].

To avoid repeatedly switching the rounding mode during interval arithmetic, we use the standard “opposite trick”, storing the negative of the lower bound and defining new operations that rely on one rounding direction only. We have also experimented with using SIMD vectors to store the intervals and using vector intrinsics for arithmetic operations [82, 104], but found that our implementation of this strategy was not significantly faster in practice than simply operating on two doubles.
Collision test culling is also critical for efficiency. We compute the axis-aligned bounding box (AABB) of each moving edge, triangle and vertex and only run collision detection when AABBs overlap, accelerating this test with a regular background grid. We further cull tests by checking if, for any of several non-axis normal directions, there is a plane separating the origin from the transformed hexahedron or prism. Our implementation of this plane test uses interval arithmetic for robustness: only when all vertices are definitely on the negative or positive side of a plane (no interval contains zero), do we consider the plane to be a separating plane. This relatively inexpensive plane-based testing eliminates 99% of the tests, considerably improving performance.

A.4.1 Resolving Collisions

We implemented our new algorithm using interval filtered floating-point expansion arithmetic, providing practical, provably robust CCD code for deforming meshes without any user-tuned tolerances. However, at this point we can only guarantee collision detection: this says nothing about resolving these collisions, i.e. finding a physically consistent adjustment to the final configuration of the mesh that eliminates all collisions. Indeed, taking into account that the final positions are quantized to a finite number of bits of precision, provably robust but ideally physical collision resolution may involve the solution of a rather daunting large-scale integer programming problem. As an example of the complications involved, even just transformation of an intersection-free mesh with a rotation matrix can potentially create self-intersection once the results are rounded again.

We use the velocity filtering approach initiated by Provot [123] and extended by Bridson et al. [31] and Harmon et al. [93]. First repulsion forces are applied to proximal mesh elements, followed by several sweeps of CCD, applying individual impulses when collisions are detected. If there remain unresolved collisions after these sweeps, we gather overlapping collisions into “impact zones” and solve for the set of impulses which will resolve all collisions in the zone simultaneously. If this system is degenerate, we compute a single rigid motion for the vertices of the offending impact zone, ensuring no collision (modulo the quantization issue mentioned above). This was referred to as “rigid impact zones” in Bridson’s original
paper [31].

While those previous works used the normal at the time of collision (typically from the triangle or from the cross-product of edges), we have found this is not a crucial choice. Interpolating the geometry at $t = 0.5$ and computing the normal from the vector between the closest points on the two mesh elements has worked equally well in our experiments, and is more computationally efficient.

### A.5 Examples

We tested our new CCD routines in a standard mass-spring cloth simulator with an initially curved sheet of cloth of resolution of 40×400 vertices, dropped on a solid ground plane. As shown in Figure A.6, this results in a large number of collisions as the cloth stacks up on itself.

![Figure A.6: CCD stress test.](image)

Although mass-spring systems are popular due to their simplicity and ease of implementation, implementers of cloth animation systems have been turning to increasingly sophisticated models in recent years. For example, the Finite Element
Method (FEM) can achieve accurate results and is less dependent on the mesh structure than using edge-based springs [76].

In the related sub-field of simulating volumetric elastic solids for graphics, it is becoming increasingly popular to perform on-the-fly optimization of the volumetric simulation mesh [14, 164, 165]. The benefits of remeshing include reducing error for highly-sheared elements, and concentrating computational effort where it is needed to resolve small-scale details. For cloth, an additional important benefit of remeshing is to increase vertex density in regions of high curvature, so that curved regions can be accurately represented without having to globally refine the surface mesh.

A few authors have suggested refining the simulation elements for cloth [108, 157], however, the idea has not been as popular for cloth as it has for solid elasticity. One reason for the lack of uptake is the difficulty in dealing with collisions. For example, adding and removing vertices without introducing self-intersections is a major concern if continuous collision detection assumes that the mesh is intersection-free at the beginning of each time step. Adding and removing vertices and altering the triangulation at discrete times compounds the difficulty in choosing suitable collision and intersection error tolerances.

Armed with our parameterless collision detection system, we demonstrate a complete FEM simulator with on-the-fly, adaptive remeshing. We use linear elasticity with rotated finite elements, as described by Etzmuß et al. [76], and simple edge crossover springs for bending forces. We choose simple edge splitting, flipping, and collapsing as our mesh optimization operations, and make them all collision safe, using CCD on “pseudo-trajectories”, as described by Brochu & Bridson [34]. To increase the vertex density in high-curvature areas, we scale the measured edge lengths by local curvature estimates when deciding to collapse or split edges. (We note that these operations are perhaps not as suitable for cloth simulation as a regular subdivision scheme, but it is a reasonable proxy for examining the challenges faced when maintaining intersection-free adaptive surfaces.) Figure A.7 shows a frame from a simulation with a single piece of cloth, and the underlying rest-state mesh. We also show a more challenging CCD scenario, with several layers of cloth draped over a solid sphere (Figure A.1).

All of our tests were performed on a single core of a 2.7 GHz Intel i5 processor
Figure A.7: Cloth with an adaptive simulation mesh

with 4GB of RAM. We integrated our new CCD algorithm into the open-source El Topo surface tracking library [33], as it provides an intersection-free remeshing framework, suitable for adaptive cloth simulation, and provides an implementation of the cubic-solver based CCD approach for comparison. To test our new CCD, we simply substituted El Topo’s inexact CCD and intersection testing functions with our new implementation.

The average time spent per call to CCD, not including culling based on AABB comparisons, but including plane-based culling, was approximately 614 nanoseconds for segment-segment testing, and 439 ns for point-triangle testing. By comparison, the average time in El Topo’s cubic solver CCD implementation (again not including AABB culling) was 649 ns and 659 ns.

Counting only tests which were positive (exercising the entire path of our code), the average time per call was 19 microseconds for segment-segment, and 15 µs for point-triangle, compared to 1.2 µs for both tests with the cubic solver CCD. This indicates that without culling, our new algorithm is more expensive than the cubic-solver version, as expected, but also that our culling is very effective.
A.6 Conclusions

We have presented a novel approach to continuous collision detection, constructed from a set of predicates which can be evaluated exactly. We have shown that the collision detection problem can be rephrased as determining whether a function has an odd number of roots in a given domain. We then showed how this problem can be reduced to testing a ray from the origin against the image of the domain boundary and counting the parity of the crossings. This in turn reduces to a set of ray-vs-triangle and ray-vs-bilinear-patch tests, built from determinant and \( \phi \) sign evaluations.

Our implementation uses a floating-point filter approach for efficiency — first determining if the correct Boolean result can be determined using interval arithmetic, and only using floating-point expansions for exact evaluation if required. We demonstrated the utility of our approach with a challenging test case: simulation of cloth undergoing on-the-fly remeshing with a large amount of contact. To our knowledge, this is the first time an adaptive cloth simulation scheme has been presented which explicitly deals with the challenges of continuous collision detection.

There are several avenues of future work. While irrelevant for the simulations we presented, being able to distinguish zero from a positive even number of roots could be critical for other applications. Our approach should extend naturally from multi-affine functions to testing intersections with higher-degree polynomial patches. Rigid body motion is more naturally expressed in terms of screw motions; though the intermediate positions involve trigonometric functions of time, reparametrization similar to the NURBS approach to conic sections would lead to a multivariate polynomial problem amenable to this attack.

A.7 Implicit Function for a Bilinear Patch

We define the following multivariate polynomial \( \phi(x) \) where the indices 0 to 3 refer to the patch corner vertices:

\[
\phi(x) = h_{12}(x) - h_{03}(x).
\]
The two $h$-functions are designed to be zero on the straight edges of the patch via products of plane $g$-functions for the various subsets of three vertices:

\[
\begin{align*}
    h_{12}(x) &= g_{012}(x) g_{132}(x) \\
    h_{03}(x) &= g_{013}(x) g_{032}(x) \\
    g_{pq}(x) &= (x - x_p) \cdot (x_q - x_p) \times (x_r - x_p).
\end{align*}
\]

We claim that the zero level set of $\phi(x)$ contains the bilinear patch. To confirm this is indeed the function we seek, take an arbitrary point $x$ with barycentric coordinates $\alpha, \beta, \gamma,$ and $\delta$ w.r.t. the corners of the patch:

\[
x = \alpha x_0 + \beta x_1 + \gamma x_2 + \delta x_3.
\]

Recall that the barycentric coordinates of a point with respect to a tetrahedron are proportional to the signed volumes of the tetrahedra formed by the point and each of the triangular faces. Letting $V$ be six times the signed volume of the tetrahedron spanning the corners of the patch, observe that:

\[
\begin{align*}
    g_{132}(x) &= \alpha V \\
    g_{032}(x) &= \beta V \\
    g_{013}(x) &= \gamma V \\
    g_{012}(x) &= \delta V
\end{align*}
\]

Therefore our function evaluates to:

\[
\phi(x) = \delta \alpha V^2 - \gamma \beta V^2.
\]

Assuming $V \neq 0$, this is zero if and only if $\alpha \delta = \beta \gamma$, which occurs precisely for the parameterized bilinear surface:

\[
\begin{align*}
    \alpha &= (1 - s)(1 - t) \\
    \beta &= (1 - s)t \\
    \gamma &= s(1 - t) \\
    \delta &= st.
\end{align*}
\]

Moreover, it is clear that $\phi(x)$ changes sign across the zero level set, dividing space into positive and negative regions separated by the conic containing the bilinear patch.
This construction breaks down if $V = 0$. However this is the case when the patch is perfectly flat, where we can simply replace the entire ray-patch intersection test with two ray-triangle tests.

### A.8 Acknowledgments

This work was supported in part by a grant from the Natural Sciences and Engineering Research Council of Canada. We would also like to thank Eitan Grinspun and our anonymous reviewers for helpful discussion and comments.