Peer Reviewed

Title:
Virtual Node Algorithms for Simulating and Cutting Deformable Solids

Author:
Wang, Yuting

Acceptance Date:
2014

Series:
UCLA Electronic Theses and Dissertations

Degree:
Ph.D., Mathematics 0540UCLA

Advisor(s):
Teran, Joseph M

Committee:
Osher, Stanley, Anderson, Chris, Terzopoulos, Demetri

Permalink:
http://escholarship.org/uc/item/4753d717

Abstract:

Copyright Information:
All rights reserved unless otherwise indicated. Contact the author or original publisher for any necessary permissions. eScholarship is not the copyright owner for deposited works. Learn more at http://www.escholarship.org/help_copyright.html#reuse
Virtual Node Algorithms for Simulating and Cutting Deformable Solids

A dissertation submitted in partial satisfaction
of the requirements for the degree
Doctor of Philosophy in Mathematics

by

Yuting Wang
Physics based simulations often require solving differential equations on domains with irregular geometry. A popular approach to their numerical approximation is the finite element method with unstructured meshes that conform to the domain geometries. However, in many applications such as shape optimization for elastic material and virtual surgery, the simulation domains frequently change shape. Under these cases, continuous remeshing can be a daunting task, and often produces ill-conditioned elements, which will reduce the accuracy and stability of the simulations. Virtual node algorithms (VNA) were invented to overcome this difficulty by embedding the simulation domains on Cartesian grids or structured meshes that do not conform to their geometry. Solutions are then computed on the embedding domains to take advantage of their regularity. The term ‘virtual nodes” refers to the degrees of freedom that lie outside the actual domains, reflecting the fact that there is no material associated with them. In this dissertation we present two novel virtual node algorithms.

In Part I, we present a cut cell method for enforcing Dirichlet and Neumann boundary conditions with nearly incompressible linear elastic materials in irregular domains. Virtual nodes on cut uniform grid cells are used to provide geometric flexibility in the domain boundary shape without sacrificing accuracy. We use a mixed formulation utilizing a MAC-type staggered grid with piecewise bilinear displacements centered at cell faces and piecewise constant pressures at cell centers. These discretization choices provide the necessary stability
in the incompressible limit and the necessary accuracy in cut cells. Numerical experiments suggest second order accuracy in $L^\infty$. We target high-resolution problems and present a class of geometric multigrid methods for solving the discrete equations for displacements and pressures that achieves nearly optimal convergence rates independent of grid resolution.

In Part II, we present a novel VNA for changing tetrahedron mesh topology to represent arbitrary cutting triangulated surfaces. Our approach addresses a number of shortcomings in the original VNA of [69]. First, we generalize the VNA so that cuts can pass through tetrahedron mesh vertices and lie on mesh edges and faces. The original algorithm did not make sense for these cases and required often ambiguous perturbation of the cutting surface to avoid them. Second, we adaptively embed a material mesh in the simulation mesh to resolve the cutting geometry. In the original VNA, cutting surfaces are resolved at the resolution of the simulation mesh, this does not only limit the geometric details of each cut, but also places a constraint on the number of times an element can be cut. Our adaptive approach removes both of these limitations. Finally, we present a novel, provably-robust floating point mesh intersection routine that registers triangulated surface cuts against the background tetrahedron mesh without the need for exact arithmetic.
The dissertation of Yuting Wang is approved.

Demetri Terzopoulos
Stanley Osher
Chris Anderson
Joseph Teran, Committee Chair

University of California, Los Angeles
2014
To my family and friends
# Table of Contents

I  A Second-Order Virtual Node Algorithm for Nearly Incompressible Linear Elasticity in Irregular Domains 1

1  Introduction ......................................................... 2

2  Existing methods ................................................... 5

3  Mixed finite element formulation .................................. 8
   3.1 Discretization .................................................... 9
   3.2 Implementation details .......................................... 13
   3.3 Discrete geometric representation and cut cell integration ...... 16

4  Dirichlet boundary conditions ...................................... 19
   4.1 Discretizing the Dirichlet problem ................................. 19
   4.2 Constructing the null-space for the Dirichlet constraints .......... 23

5  Multigrid .......................................................... 27
   5.1 Discretization hierarchy ........................................... 28
   5.2 Relaxation ........................................................ 29
      5.2.1 Approximated distributive relaxation ........................ 31
      5.2.2 Higher order defect correction ............................... 33
   5.3 Boundary relaxation .............................................. 34
   5.4 Boundary relaxation for the reduced system in Dirichlet boundary condition case ............................................. 35
   5.5 Coarsening ....................................................... 37
6 Numerical examples ........................................ 39
   6.1 Discretization error .................................. 42
   6.2 Multigrid efficiency .................................. 42

7 Conclusion and future work ................................. 50

II An Adaptive Virtual Node Algorithm with Robust Mesh Cutting 51

8 Introduction ................................................. 52

9 Related work ................................................. 54

10 Adaptive virtual node cutting algorithm .................. 56
   10.1 Overview .............................................. 56
   10.2 Step1: Split .......................................... 58
   10.3 Step 2: Merge ......................................... 59
   10.4 Step 3: Subdivide ...................................... 61

11 Robust intersection computation .......................... 63
   11.1 Overview .............................................. 63
   11.2 Floating point error ................................... 64
   11.3 Tolerance model ....................................... 68
   11.4 2D ..................................................... 69
       11.4.1 Tolerance Computation .......................... 69
       11.4.2 Vertex-Vertex .................................... 70
       11.4.3 Edge-Vertex ...................................... 71
11.4.4 Edge-Edge ................................................. 73
11.4.5 Triangle-Vertex .......................................... 75
11.5 3D ............................................................. 76
  11.5.1 Tolerance Computation ................................. 76
  11.5.2 Vertex-Vertex ........................................... 77
  11.5.3 Edge-Vertex .............................................. 80
  11.5.4 Edge-Edge ................................................ 82
  11.5.5 Triangle-Vertex ......................................... 88
  11.5.6 Triangle-Edge ............................................ 91
  11.5.7 Tetrahedron-Vertex ..................................... 94

12 Examples .......................................................... 97

13 Discussions ....................................................... 101

References .......................................................... 102
List of Figures

3.1 Staggered grid finite element quadrangulation and embedded domain boundary. 10

3.2 Left: one interior pressure cell and the variables corresponding to the 13 degrees of freedom of the elemental stiffness matrix by taking integral over the pressure cell. Right: the four integral subcells of the pressure cell. Center: the variables that an integral over subcell $\omega_1$ contributes to. 16

3.3 Global stiffness matrix stencils centered at an interior $x$ variable (left), $y$ variable (middle) and $p$ variable (right). 16

3.4 A zoom-in view of Figure 3.1(a). A levelset function is sampled on a doubly refined grid (left); a segmented curve $\partial \Omega_h$ is generated to approximate the boundary of the geometric domain (right). 17

4.1 Cell aggregations. a) $x$ component boundary cells and the first two representative nodes and the incident $y$—cells of each representative node; b) all representative nodes for $y$ component cells and their incident cells, orphan cells will be attached to their nearest neighbor cells; c) and d) final cell aggregations together with their representative nodes for $x$ and $y$ grids. 26

5.1 Boundary band and distributive region. 30

5.2 Coarsening of grid variables 37

5.3 Restriction operator stencils. 38

6.1 Left: undeformed keyhole domain; right: deformed keyhole domain 40

6.2 Left: undeformed flower domain; right: deformed flower domain 41

6.3 Left: undeformed spiral domain, right: deformed spiral domain 41
6.4 Order of accuracy for keyhole domain; top: Neumann boundary condition; bottom: Dirichlet boundary condition; left: Poisson’s ratio=0.3; right: Poisson’s ratio=0.49; square marker: $x$ component; circle marker: $y$ component.

6.5 Order of accuracy for flower domain. Top: Neumann boundary condition, bottom: Dirichlet boundary condition; left: Poisson’s ratio=0.3; right: Poisson’s ratio=0.49; square marker: $x$ component; circle marker: $y$ component.

6.6 Order of accuracy for spiral domain. Top: Neumann boundary condition; bottom: Dirichlet boundary condition; left: Poisson’s ratio=0.3; right: Poisson’s ratio=0.49; square marker: $x$ component; circle marker: $y$ component.

6.7 Multigrid V-(1,1) cycle convergence under refinement (Poisson’s ratio = 0.49, periodic boundary conditions, finite element distribution)

6.8 Multigrid V-(1,1) cycle convergence rate at resolutions from 32 to 1024 (Poisson’s ratio = 0.49). Left: Dirichlet boundary condition; right: Neumann boundary condition.

6.9 Multigrid V-(1,1) cycle convergence under refinement (Poisson’s ratio = 0.49). Left: Dirichlet boundary condition; right: Neumann boundary condition.

10.1 Schematic overview of our cutting algorithm. In the middle row, the embedding elements are omitted due to space limit except for the left one. The embedding relationship is shown more completely in the second cut, while the first cut shows the splitting and merging of material in detail.

10.2 Left: The maximally split configuration for the original virtual node cutting algorithm in 2D. The cuts (red) can not pass through nodes or overlap with edges. Right: The element maximally splits into 3 pieces, materials are embedded in the duplicated copies of the original element. Nodes on the dotted segments are the “virtual nodes”.

x
10.3 Our maximally split configuration in 2D contains three types of cut faces (top row). The red flag is set if intersections are registered at the green location and one of the blue locations on the triangle. In the maximally split configuration in 3D (top right), each color represents a different type of cut face, which are shown individually on the bottom row. The correspondence is, from bottom left to bottom right: green, red, yellow, and blue. For each of the four 3D cases (bottom), the shaded flag is set if intersections are registered at the red location, one of the green locations, and one of the blue locations on the tetrahedron.

11.1 The intersection test for two edges. Left: To test $P, Q$ are on different sides of $AB$, we require $\text{sgn}(\overrightarrow{PA} \times \overrightarrow{PB}) \neq \text{sgn}(\overrightarrow{QA} \times \overrightarrow{QB})$. Right: To test $A, B$ are on different sides of $PQ$, we require $\text{sgn}(\overrightarrow{AP} \times \overrightarrow{AQ}) \neq \text{sgn}(\overrightarrow{BP} \times \overrightarrow{BQ})$.

11.2 Left: two points within the vertex-vertex tolerance and are registered as intersecting (red). Middle: the two points separate, but the cutting segment remains within the edge-vertex tolerance and the edge-vertex pair is registered as intersecting (green). Right: the cutting segment is far enough from degeneracies that face-vertex (yellow) and edge-edge (orange) intersections are registered.

11.3 Tolerances for 2D.

11.4 Proof illustrations for vertex edge in 2D

11.5 Proof illustrations for edge edge in 2D.

11.6 Proof illustrations for triangle vertex in 2D.

11.7 Tolerances for 3D.

11.8 Proof illustrations for edge edge in 3D.

11.9 Proof illustrations for triangle vertex in 3D.

11.10 Proof illustrations for triangle edge in 3D.
11.11 Proof illustrations for tetrahedron vertex in 3D. .................................................. 96

12.1 An armadillo mesh with 380K tetrahedra is progressively cut with an “S” and then cut into three pieces. ................................................................. 98

12.2 An apple is peeled demonstrating the ability to do incremental cuts and produce thin geometry, shown from the front (left) and the side (right). The skin thickness is $1/60$ of the diameter of the ball. ................................................................. 98

12.3 We cut letters out of stretched elastic sheet, demonstrating the ability to mix cutting, re-cutting, and simulation. ................................................................. 99

12.4 We demonstrate the ability to construct tetrahedralized volumes from triangulated surfaces using our cutting algorithm. We cut a box by a cow surface and then cut it again with a bunny surface, resulting in two separate volumes. Details are accurately resolved in both volumes, and the corners are sharp. ................................................................. 100
6.1 Periodic boundary condition multigrid asymptotic convergence rates (Poisson’s ratio=0.49, resolution=128 × 128). For optional prolongations, \( P_{hi} \) is bilinear interpolation and \( P_{lo} \) is for \( P = 4R^T \). For the optional distributions, FE is the distribution matrix discretized with the bilinear finite element method, and FD is using defect correction by employing finite difference distributive relaxation. ......................................................... 43
ACKNOWLEDGMENTS

First and foremost I would like to thank my advisor Professor Joseph Teran for his generous support, help and guidance during the past five years. I would also like to thank Professor Chris Anderson, Professor Stanley Osher and Professor Demetri Terzopoulos for their time and interest to review my work.

The first part of this work was done in collaboration with Yongning Zhu, Jeffrey Hellrung, Alejandro Cantarero, Eftychios Sifakis and Joseph Teran. The published article “A Second-Order Virtual Node Algorithm for Nearly Incompressible Linear Elasticity in Irregular Domains”, Journal of Computational Physics, 231(21), 7092-7117, 2012, contains much of the material presented here.

The second part of this dissertation is based on another research project done in collaboration with Chenfanfu Jiang, Craig Schroeder and Joseph Teran, the paper “An Adaptive Virtual Node Algorithm with Robust Mesh Cutting” has been accepted for publication at Symposium on Computer Animation 2014. Especially, I would like to thank Dr. Craig Schroeder for his mentorship.
VITA

2005-2009  B.S. in Mathematics, Zhejiang University, China.

2009-2011  M.A. in Mathematics, University of California at Los Angeles.

2009-2014  Research and teaching assistant, University of California at Los Angeles.
Part I

A Second-Order Virtual Node Algorithm for Nearly Incompressible Linear Elasticity in Irregular Domains
CHAPTER 1

Introduction

We focus on the equilibrium equations of linear elasticity in an arbitrary domain $\Omega$

\[-\nabla \cdot \sigma(u) = f \quad \text{in} \ \Omega \quad (1.1)\]

\[u|_{\Gamma_d} = u_0 \quad \text{on} \ \Gamma_d \quad (1.2)\]

\[(\sigma(u) \cdot n)|_{\Gamma_n} = \tau \quad \text{on} \ \Gamma_n. \quad (1.3)\]

We use $u$ to denote the displacement mapping of the particles of material, $\sigma$ is the Cauchy stress tensor, $f$ is the external force per unit area, $u_0$ is the prescribed Dirichlet boundary displacements (over the Dirichlet portion of $\partial \Omega$: $\Gamma_d$) and $\tau$ is the prescribed external surface traction (over the Neumann portion of $\partial \Omega$: $\Gamma_n$). In linear elasticity, the stress $\sigma(u)$ is linearly dependent on Cauchy strain $\vec{\epsilon}(u)$:

\[\vec{\epsilon}(u) = \frac{\nabla u + (\nabla u)^T}{2} \quad (1.4)\]

\[\sigma(u) = 2\mu \vec{\epsilon}(u) + \lambda \text{tr} \vec{\epsilon}(u) I \quad (1.5)\]

\[= \mu (\nabla u + (\nabla u)^T) + \lambda (\nabla \cdot u) I \quad (1.6)\]

Therefore, the equations of linear elastic equilibrium can be equivalently written as

\[-(\mu \Delta I + (\lambda + \mu)\nabla \nabla^T)u = f \quad \text{in} \ \Omega \quad (1.7)\]

\[u|_{\Gamma_d} = u_0 \quad \text{on} \ \Gamma_d \quad (1.8)\]

\[\mu(u_n + \nabla(u \cdot n)) + \lambda(\nabla \cdot u)n|_{\Gamma_n} = \tau \quad \text{on} \ \Gamma_n. \quad (1.9)\]

When supporting irregular domain geometries, a natural choice for the numerical approximation of these equations is the finite element method (FEM) with unstructured meshes.
that conform to the geometry of $\partial \Omega$. However, meshing complex geometries can prove difficult and time-consuming when the boundary frequently changes shape. This task is even more difficult when using the more elaborate element types seen in mixed FEM formulations. Mixed formulations are typically necessary for stability with the nearly incompressible materials we consider here. In many applications, such as shape optimization for elastic materials or crack propagation, it is necessary to change the geometry of the domain at each iteration of a simulation. In such cases, frequent unstructured remeshing can be prohibitively costly (especially in 3D). Also, many numerical methods, such as finite differences, do not naturally apply to unstructured meshes. These concerns motivated the development of “embedded” (or “immersed”) methods that approximate solutions to (1.7) on Cartesian grids or structured meshes that do not conform to the boundary. Retention of higher order accuracy in $L^\infty$ with such embedding strategies is an ongoing area of research. Typically, the difficulty is to determine numerical stencils near the domain boundary that retain the accuracy achieved in the interior, away from the irregular features. While the accuracy of the discretization is an issue, the structure of the regular grid can be leveraged to provide very efficient solution methods. For high resolution discretization, particularly in the case of nearly incompressible materials, efficient solution of the discrete systems can be challenging. In these cases, direct methods become too slow and memory intensive. Geometric multigrid methods and domain decomposition approaches have been shown to provide very favorable performance in this setting, however their application to embedded discretizations of irregular domains is not straightforward. Ultimately, special attention must be paid near the boundary for both discretization accuracy and efficient numerical linear algebra.

With these concerns in mind, we introduce a second order virtual node method for approximating the equations of linear elastic equilibrium with irregular embedded Neumann and Dirichlet boundaries on a uniform Cartesian grid. We use a regular grid because it simplifies the implementation, permits straightforward Lagrange multiplier spaces for Dirichlet constraints and naturally allows for higher order accuracy in $L^\infty$. Furthermore, the approach has excellent potential for efficient parallel implementation as indicated by the
similarities to the first order method investigated in [114]. The method is most suited for problems like level set-based shape optimization where the geometry of the domain is frequently changing and constant remeshing is a clearly inferior alternative to embedding (see e.g [102, 1, 81, 31, 40, 91]). Also, we allow for nearly incompressible materials by introducing pressure as an additional unknown in a mixed variational formulation. Our discretization of this variational formulation is then based on a MAC-type staggering of $x$ and $y$-component displacements with pressures at cell centers. Our approach combines piecewise bilinear interpolation of displacement components with the addition of “virtual” nodes on cut cells that accurately account for the irregular shape of the geometry boundary. The variational nature of the method naturally enables symmetric numerical stencils at the boundary. We use Lagrange multipliers to enforce embedded Dirichlet conditions weakly. In the general case, our choice of Lagrange multiplier space admits an efficient means for smoothing boundary equations in our geometric multigrid method for solving the discretized equations. Numerical experiments indicate second order accuracy in $L^\infty$ independent of Poisson’s ratio, domain geometry or boundary condition type.

The remainder of the paper proceeds as follows: in Chapter 2 we discuss results from existing methods; in Chapter 3 we discuss the mixed variational formulation we use to accurately handle the nearly incompressible case; in Section 3.1 we detail our use of embedding over MAC-type staggered grids to discretize the mixed formulation; in Chapter 4 develop a novel approach to enforcing Dirichlet boundary conditions as a constraint on the Neumann problem and lastly we discuss a novel geometric multigrid method for the solution of our discretized equations in Chapter 5.
CHAPTER 2

Existing methods

Our approach is second-order accurate in $L^\infty$ with both embedded Neumann and embedded
Dirichlet boundary conditions over irregular domains and our discrete systems are developed
to facilitate a class of multigrid methods that achieve nearly optimal convergence rates
independent of grid resolution. Furthermore, our method retains these properties with nearly
incompressible materials. While there are many existing approaches for linear elasticity,
particularly in the case of unstructured mesh based approaches, ours is the first embedded
method to support this feature set. In our discussion of existing approaches, we will focus
only on methods that avoid unstructured meshing when addressing irregular boundaries
for incompressible materials. Also, we will discuss methods addressing the difficulties that
arise when ensuring scalability to high resolution problems, particularly in the case of nearly
incompressible materials.

Embedded techniques use a computational domain that simply encompasses rather than
generically adheres to the irregular domain (see Figure 3.1). This avoids the difficulties
associated with unstructured mesh generation, but can complicate the enforcement of
boundary conditions. A good review of these issues is given by Lew et al in [64]. They point
out that these techniques actually originated with the papers of Harlow and Welch [54] and
Charles Peskin [86], at least in the context of incompressible materials. Some of the first
embedded methods were fictitious domain methods by Hyman [57] and Saul’ev [89]. The
fictitious domain approach has been used with incompressible materials in a number of works
[49, 50, 48, 84, 83, 41, 97, 19, 87]. These approaches embed the irregular geometry in a more
simplistic domain for which fast solvers exist (e.g. Fast Fourier Transforms). The calcula-
tions include fictitious material in the complement of the domain of interest. A forcing term (often from a Lagrange multiplier) is used to maintain boundary conditions at the irregular geometry. Although these techniques naturally allow for efficient solution procedures, they depend on a smooth solution across the embedded domain geometry for optimal accuracy, which is not typically possible.

The extended finite element method (XFEM) and related approaches in the finite element literature also make use of geometry embedded in regular elements. Although originally developed for crack-based field discontinuities in elasticity problems, these techniques are also used with embedded problems in irregular domains. Daux et al first showed that these techniques can naturally capture embedded Neumann boundary conditions [38, 95]. These approaches are equivalent to the variational cut cell method of Almgren et al in [2]. Enforcement of Dirichlet constraints is more difficult with variational cut cell approaches [68, 64] and typically involves a Lagrange multiplier or stabilization. Dolbow and Devan recently investigated the convergence of such approaches with incompressible materials and point out that much analysis in this context remains to be completed [39]. Despite the lack of thorough analysis, such XFEM approaches appear to be very accurate and have been used in many applications involving incompressible materials in irregular domains [100, 12, 36, 34, 45].

There are also many finite difference (FDM) and finite volume methods (FVM) that utilize cut uniform grid cells. Many of these methods have been developed in the context of incompressible flow. For example, Almgren et al use cut uniform bilinear cells to solve the Poisson equation for pressures in incompressible flow calculations [2]. Marellle et al use collocated grids and define define sub cell interface and boundary geometry in cut cells via level sets [66]. Ng et al also use level set descriptions of the irregular domain and achieve second order accuracy in $L^\infty$ for incompressible flows [74]. The approach of Batty and Bridson is similar, but not as accurate [11]. Although not technically a cut cell approach, the immersed interface method has been used to improve accuracy for incompressible flow calculations in irregular domains [104, 87, 113, 32, 111, 88]. Cut cell FDM and FVM have also been developed for incompressible and nearly incompressible elastic materials. Bijelonja
et al use cut cell FVM to enforce incompressibility more accurately than is typically seen with FEM [18]. Beirao da Veiga et al use polygonal FVM cells to avoid remeshing with irregular domains [14]. Barton et al [10] and Hill et al [56] use cut cells with Eulerian elastic/plastic flows.
CHAPTER 3

Mixed finite element formulation

In order to accurately handle linear elastic materials near the incompressible limit, we use an augmented form of the equilibrium equations. By introducing a pressure variable as an unknown, we can achieve a stable numerical discretization independent of the degree of incompressibility. We will use the weak form of this augmented system to derive a mixed finite element formulation. The augmented form of our equations arises by introducing $p = -\lambda/\mu \nabla \cdot u$. With this definition, $\sigma(u) = \mu (\nabla u + (\nabla u)^T) - \mu p \mathbf{I}$ and the derived PDE

$$\begin{align*}
-\mu(\Delta \mathbf{I} + \nabla \nabla^T)u + \mu \nabla p &= f \quad \text{in } \Omega \\
-\mu \nabla \cdot u - \frac{\mu^2}{\lambda} p &= 0 \quad \text{in } \Omega \\
\mathbf{u}|_{\Gamma_d} &= \mathbf{u}_0 \quad \text{on } \Gamma_d \\
\mu(u_n + \nabla (u \cdot n)) - \mu p n|_{\Gamma_n} &= g \quad \text{on } \Gamma_n
\end{align*}$$

is then equivalent to the original PDE.

We use this augmented form of the equations to derive an equivalent variational form of the linear elastic equilibrium equations. A weak form can be derived by taking the inner product of the strong form with an arbitrary vector valued function $v \in V_0 = (H^1_0(\Omega))^d$ and by enforcing the equation $p = -\lambda/\mu \nabla \cdot u$ weakly:

Find $(\mathbf{u}, p) \in H^1(\Omega) \times H^1(\Omega) \times L^2(\Omega)$, $\mathbf{u}|_{\Gamma_d} = \mathbf{u}_0$, such that

$$\begin{align*}
\int_\Omega 2\mu \left( \frac{\nabla \mathbf{u} + \nabla \mathbf{u}^T}{2} \right) : \left( \frac{\nabla \mathbf{v} + \nabla \mathbf{v}^T}{2} \right) - \mu p (\nabla \cdot \mathbf{v}) d\mathbf{x} &= \int_\Omega f \cdot \mathbf{v} d\mathbf{x} + \int_{\partial \Omega} g \cdot \mathbf{v} ds \quad \forall \mathbf{v} \in H^1_0(\Omega) \times H^1_0(\Omega) \\
\int_\Omega \left( -\mu q \nabla \cdot \mathbf{u} - \frac{\mu^2}{\lambda} p q \right) d\mathbf{x} &= 0 \quad \forall q \in L^2(\Omega)
\end{align*}$$
3.1 Discretization

We discretize this variational formulation using a mixed finite element method defined on a MAC-type staggered grid. Han et. al. demonstrated the stability and optimal convergence of this formulation applied to the Stokes equations on a square domain \([52]\). We generalize this approach to the case of nearly incompressible linear elasticity in embedded domains. We approximate the Sobolev space \(V\) with a finite element subspace \(V_h\), where each displacement component of a function in \(V_h\) is represented as a piecewise bilinear scalar function defined on a staggered quadrilateral grid (see Figure 3.1). To be more specific, consider the staggered grids:

\[
\mathcal{G}_h^x = \{(ih, (j - 1/2)h) : (i, j) \in I_x \subset \mathbb{Z}^2\}
\]

\[
\mathcal{G}_h^y = \{((i - 1/2)h, jh) : (i, j) \in I_y \subset \mathbb{Z}^2\}.
\]

Here, \(h\) is the discrete spacing between grid points. Furthermore, we use the following notation to denote quadrilaterals defined by these grids:

\[
T_{ij}^x = \{(x, y) : ih < x < (i+1)h, (j - 1/2)h < y < (j + 1/2)h\}
\]

\[
T_{ij}^y = \{(x, y) : (i - 1/2)h < x < (i+1/2)h, jh < y < (j + 1/2)h\}.
\]

The sets \(I_x\) and \(I_y\) used in the definition of grids \(G^x_h\) and \(G^y_h\) are defined as the collection of vertices incident on some quadrilateral \(T_{ij}^x\) or \(T_{ij}^y\), respectively, whose intersection with the domain \(\Omega\) is non-empty. In other words, \(I_x\) and \(I_y\) are the sets of vertices in the staggered lattices that are at most a distance of \(h\) away from \(\Omega\). Henceforth, we will use

\[
\mathcal{T}_h^x = \{T_{ij}^x : T_{ij}^x \cap \Omega \neq \emptyset\}
\]

and

\[
\mathcal{T}_h^y = \{T_{ij}^y : T_{ij}^y \cap \Omega \neq \emptyset\}
\]

to denote the collection of \(x\) and \(y\) grid quadrilaterals that intersect (or embed) the domain \(\Omega\). We construct two subspaces of \(H^1(\Omega)\) based on these quadrangulations respectively:

\[
V^h_x = \{v_h \in C^0(\Omega) : v_h|_{T_{ij}^x} \in Q_1(T_{ij}^x) \quad \forall T_{ij}^x \in \mathcal{T}_h^x \quad \text{s.t.} \quad T_{ij}^x \cap \Omega \neq \emptyset\}
\]
\[ V^h_x = \{ v_h \in C^{(0)}(\Omega) : v_h|_{T^y_{ij}} \in Q_1(T^y_{ij}) \ \forall T^y_{ij} \in T^y_h \text{ s.t. } T^y_{ij} \cap \Omega \neq \emptyset \} \]

For simplicity of notation in subsequent equations we will also use mappings \( \eta_1 : I_1 = \{1, 2, \ldots, N_x\} \rightarrow I_x \) and \( \eta_2 : I_2 = \{1, 2, \ldots, N_y\} \rightarrow I_y \) to associate each \( x \) and \( y \) grid vertex with a unique integer between \( 1 \) and \( N_x \) and \( 1 \) and \( N_y \) respectively. With this convention, any approximated solution \( \tilde{u} \in V_x^h \times V_y^h \) can be expressed as

\[
\tilde{u}(\vec{x}) = \left( \begin{array}{c}
\sum_{k_1 \in I_1} u_{1k_1} N_{1k_1}(\vec{x}) \\
\sum_{k_2 \in I_2} u_{2k_2} N_{2k_2}(\vec{x})
\end{array} \right)
\]

where \( N_{1k_1} \) and \( N_{2k_2} \) are the commonly used piecewise bilinear interpolating functions associated with nodes \( k_1 \) and \( k_2 \) respectively in \( T_x^h \) and \( T_y^h \). Our discrete equations for the approximate solution \( \tilde{u} \) can thus be seen to be over \( N_x + N_y \) unknowns.

Figure 3.1: Staggered grid finite element quadrangulation and embedded domain boundary.

We additionally approximate the space for pressure \( V_p = L^2(\Omega) \) with a piecewise constant finite element space \( V_p^h \) defined on a quadrangulation \( T^p_h \) over the primary grid (or henceforth, the pressure grid) \( G_p^h \):

\[
G_p^h = \{ ((i + \frac{1}{2})h, (j + \frac{1}{2})h) : (i, j) \in I_p \subset \mathbb{Z}^2 \}
\]

\[
T^p_{ij} = \{ (x, y) : ih < x < (i + 1)h, jh < y < (j + 1)h \} \]
\[ T_h^p = \{ T_{ij}^p : T_{ij}^p \cap \Omega \neq \emptyset \} \]
\[ V_p^h = \{ p_h \in L^2(\Omega) : p_h|_{T_{ij}^p} \in P_0(T_{ij}^p) \quad \forall T_{ij}^p \in T_h^p \} \]

The grid \( G_h^p \) is a cell-centered grid (as opposed to a node-centered grid, such as \( G_x^h \) and \( G_y^h \)). That is, we assume that pressure variables live at the cell centers of this grid. In other words, there is one pressure variable located in each \( T_{ij}^p \). The set \( I_p \) is defined similarly to \( I_x \) and \( I_y \), however here it refers to the collection of cell centered indices in the grid \( G_h^p \) whose associated quadrilaterals \( T_{ij}^p \) have a non-zero intersection with \( \Omega \). For the sake of simplicity in subsequent equations, we again use a mapping \( \eta_3 : I_3 = \{1, 2, \ldots, N_p\} \to I_p \) to associate each cell in the pressure grid with a unique integer between 1 and \( N_p \). Thus, any approximated solution of the pressure has the following representation:

\[ p(\vec{x}) = \sum_{k\in I_3} p_{k_p} \chi_{T_{k_p}^p}(\vec{x}) \]  

(3.10)

where \( \chi_{T_{k_p}^p}(\vec{x}) \) is the characteristic function for the quadrilateral \( T_{k_p}^p \). That is,

\[ \chi_{T_{k_p}^p}(\vec{x}) = \begin{cases} 
1, & \vec{x} \in T_{k_p}^p \\
0, & \vec{x} \notin T_{k_p}^p.
\end{cases} \]

We choose test functions \( \vec{v}(\vec{x}) = N_{m_{k_m}}(\vec{x}) \vec{e}_m, \ m = 1, 2 \) and substitute the finite element discretization (3.9), (3.10) into each term in the mixed variational form (3.5),

\[ 2\mu \int_{\Omega} \nabla \vec{u} + \left( \nabla \vec{u} \right)^T : \nabla \vec{v} + \left( \nabla \vec{v} \right)^T \ d\vec{x} \]

\[ = \mu \int_{\Omega} \nabla \vec{u} : \left( \nabla \vec{v} + \left( \nabla \vec{v} \right)^T \right) d\vec{x} \]

\[ = \mu \sum_{i,j\in\{1,2\}} \int_{\Omega} u_{i,j} (v_{i,j} + v_{j,i}) d\vec{x} = \mu \sum_{i,j\in\{1,2\}} \int_{\Omega} u_{i,j} (N_{m_{k_m,j}}(\vec{x})\delta_{mi} + N_{m_{k_m,i}}(\vec{x})\delta_{mj}) d\vec{x} \]

\[ = \mu \sum_{i\in\{1,2\}} \int_{\Omega} \left( \sum_{j\in\{1,2\}} u_{i,j} N_{m_{k_m,j}}(\vec{x})\delta_{mi} + u_{i,m} N_{m_{k_m,i}}(\vec{x}) \right) d\vec{x} \]

\[ = \mu \sum_{i\in\{1,2\}} \sum_{k_i \in I_i} \int_{\Omega} u_{ik_i} \left( \sum_{j\in\{1,2\}} N_{ik_i,j}(\vec{x}) N_{m_{k_m,j}}(\vec{x})\delta_{mi} + N_{ik_i,m}(\vec{x}) N_{m_{k_m,i}}(\vec{x}) \right) d\vec{x} \]

\[ = \mu \sum_{i\in\{1,2\}} \sum_{k_i \in I_i} u_{ik_i} \left( \delta_{mi} \int_{\Omega} \sum_{j\in\{1,2\}} N_{m_{k_i,j}}(\vec{x}) N_{m_{k_m,j}}(\vec{x}) d\vec{x} + \int_{\Omega} N_{ik_i,m}(\vec{x}) N_{m_{k_m,i}}(\vec{x}) d\vec{x} \right) \]
\[-\mu \int_{\Omega} p \nabla \cdot \vec{v} d\vec{x} = -\mu \sum_{k_p \in I_p} \int_{T_p \cap \Omega} N_{km,m}(\vec{x}) d\vec{x}\]
\[\int_{\Omega} \vec{f} \cdot \vec{v} d\vec{x} = \int_{\Omega} f_m N_{km,m}(\vec{x}) d\vec{x}\]
\[\int_{\Gamma_n} \vec{g} \cdot \vec{v} d\vec{s} = \int_{\Gamma_n} g_m N_{km,m}(\vec{x}) d\vec{s}.\]

We can also choose \(v = 0\) and \(q(\vec{x}) = \chi T_{kp}(\vec{x})\) to derive the pressure equations:

\[-\mu \sum_{k_i \in I_i} u_{ik_i} \int_{T_p \cap \Omega} N_{i,i,k}(\vec{x}) d\vec{x} - \frac{\mu}{\lambda} \sum_{k_p \in I_p} \int_{T_p \cap \Omega} 1 d\vec{x} = 0.\]

Since the variational form is derived from an energy minimization problem, the discretized linear system can trivially be seen to be symmetric. Specifically, if we take the convention that \(\vec{u}^h \in \mathbb{R}^{N_x + N_y}\) is our vector of displacement unknowns (where we assume that \(x\) degrees of freedom are ordered first and \(y\) second) and \(\vec{p}^h \in \mathbb{R}^{N_p}\) is the vector of pressure unknowns, then our system over the vector \(\vec{u}^h\) of \(N = N_x + N_y + N_p\) degrees of freedom is of the form:

\[
\begin{pmatrix}
L^h_u & G^h T \\
G^h & D^h_p
\end{pmatrix}
\begin{pmatrix}
\vec{u}^h \\
\vec{p}^h
\end{pmatrix}
= \begin{pmatrix}
\vec{f}^h \\
\vec{0}
\end{pmatrix}

\text{or}

\hat{L}^h \vec{u}^h = \hat{f}^h \tag{3.11}

where \(\vec{u}^h = (\vec{u}^h, p^h)\) and \(\vec{f}^h = (\vec{f}^h, 0)\). Furthermore, our use of regular grids gives the discrete equations a finite difference interpretation. If we scale the system by \(-\frac{1}{h^2}\), each block in the discrete system approximates the corresponding differential operator in (3.1), i.e. \(3.11\) discretizes the following equation:

\[
\mu^2 \begin{pmatrix}
-\mu(\Delta + \nabla \nabla^T) & \mu \nabla \\
-\mu \nabla^T & -\mu^2/\chi
\end{pmatrix}
\begin{pmatrix}
\vec{u} \\
p
\end{pmatrix}
= \begin{pmatrix}
-\vec{f}^h h^2 \\
0
\end{pmatrix}

\tag{3.12}

The linear system is the Hessian matrix of a saddle point problem, therefore the discretized system is symmetric but indefinite. In fact, the first diagonal block \(L_u^h\) is negative definite, and the other diagonal block is positive definite.
3.2 Implementation details

For ease of implementation, we perform the integrations involved in the discrete equations in an element-by-element fashion. Each area integral is represented as a sum of integrals over spatially disjoint elements whose union is the embedded domain. Specifically, we individually address the integration over the intersection of each quadrilateral of the pressure grid with the embedded domain $T_{kp} \cap \Omega$:

$$
\int_{\Omega} N_{mk,i,j}(\vec{x}) N_{mk,m,j}(\vec{x}) \, d\vec{x} = \sum_{kp \in I_p} \int_{T_{kp} \cap \Omega} N_{mk,i,j}(\vec{x}) N_{mk,m,j}(\vec{x}) \, d\vec{x}
$$

$$
\int_{\Omega} N_{ik,m}(\vec{x}) N_{mk,m,i}(\vec{x}) \, d\vec{x} = \sum_{kp \in I_p} \int_{T_{kp} \cap \Omega} N_{ik,m}(\vec{x}) N_{mk,m,i}(\vec{x}) \, d\vec{x}
$$

$$
\int_{\Omega} N_{mk,m}(\vec{x}) \, d\vec{x} = \sum_{kp \in I_p} \int_{T_{kp} \cap \Omega} N_{mk,m}(\vec{x}) \, d\vec{x}
$$

$$
\int_{\Gamma_n} N_{mk,m}(\vec{x}) \, ds = \sum_{kp \in I_p} \int_{T_{kp} \cap \Gamma_n} N_{mk,m}(\vec{x}) \, ds.
$$

In the interior, this simply amounts to evaluating the same integrals over each full quadrilateral $T_{kp}$. However, at the boundary, care must be taken to respect the material region alone when the intersection between the pressure cells and the embedded domain is non-trivial. In both the boundary and interior cases there will be 13 degrees of freedom involved in the integration over such a pressure cell. This is because the staggering of variables leads to 13 interpolating functions supported over a given pressure cell (6 $x$-components, 6 $y$-components and one pressure). In other words, we express the matrix (or stiffness matrix) in our discrete system as a sum of $13 \times 13$ element stiffness matrices $A_{kp}$. Furthermore, we break the integrals involved in a given element $T_{kp}$ up into four subintegrals over the subquadrants $(\omega_1, \omega_2, \omega_3, \omega_4)$ of $T_{kp}$ (see Figure 3.2). This is because the integrands are all smooth over these regions. Notably, they are quadratic and we simply preform these integrations analytically. The elemental stiffness matrix is accumulated from the following sub-stiffness-matrices $A_{\omega_i} = A_{\omega_1}^{kp} + A_{\omega_2}^{kp} + A_{\omega_3}^{kp} + A_{\omega_4}^{kp}$. For example, $A_{\omega_1}^{kp}$ involves $x_1$, $x_2$, $x_3$, $x_4$, $y_7$, $y_8$, $y_{10}$ and $y_{11}$ as demonstrated in Figure 3.2 right, therefore, it only has non-zero values on rows and
columns involving these degrees of freedom. If we order the 13 nodes with indices shown in Figure 3.2 left, then on the interior of the domain, where $T_{kp}^p \cap \Omega = T_{kp}^p$, the sum of these four subintegrals is always the same:

\[
A_{kp}^\mu = \begin{pmatrix}
\frac{1}{4} & 0 & 0 & -\frac{1}{4} & 0 & 0 & 9/64-3/32-3/64 & 3/64-1/32-1/64 \\
0 & \frac{1}{4} & -1/4 & 0 & 0 & 0 & 3/64 & 3/32-9/64 & 1/64 & 1/32-3/64 \\
0 & 0 & 0 & -1/4 & 1/4 & 0 & -3/64 & 1/32 & 1/64-9/64 & 3/32 & 3/64 \\
0 & 0 & -1/4 & 0 & 0 & 1/4-1/64-1/32 & 3/64-3/64-3/32 & 9/64 \\
-3/64-9/64 & 1/32 & 3/32 & 1/64 & 3/32 & 0 & 0 & 1/4 & 0 & -1/4 & 0 \\
3/64 & 1/64 & 3/32 & 1/32-9/64-3/64 & 0 & -1/4 & 0 & 1/4 & 0 & 0 \\
\end{pmatrix}
\]

The global stiffness matrix generated from $A_{kp}^\mu$ has a stencil shown in Figure 3.3.

However for boundary cells where $T_{kp}^p \cap \Omega \neq T_{kp}^p$, we have to perform the integrations involved in each of $A_{kp}^\mu_i$ carefully, taking into account the boundary geometry. We discuss this in the next section. The process of constructing the global stiffness matrix $A$ from each of the $13 \times 13$ element stiffness matrices $A_{kp}^\mu$ is explained in Algorithm 1.
Algorithm 1 Construction of global stiffness matrix $A$ from elemental $A^{k_p}$

1: $A \leftarrow \vec{0}$

2: for $k_p = 1$ to $N_p$ do

3:   if $T^{p}_{k_p} \cap \Omega = T^{p}_{k_p}$ then  

4:      Use $A^{k_p}$ from equation (3.13)  

5:   else  

6:      Perform integration over each subquadrant $\omega_i$ to compute $A^{k_p}_{\omega_i}$  

7:      $A^{k_p} = A^{k_p}_{\omega_1} + A^{k_p}_{\omega_2} + A^{k_p}_{\omega_3} + A^{k_p}_{\omega_4}$

8:     for $i^p = 1$ to 13 do

9:        $i = \text{mesh}(k_p, i^p)$  

10:       The mesh maps the 13 degrees of freedom involved in $A^{k_p}$ to their position in a global array

11:     end for

12:     for $j^p = 1$ to 13 do

13:        $j = \text{mesh}(k_p, j^p)$

14:        $A_{ij^p} = A^{k_p}_{ij^p}$

15:     end for

16: end for
3.3 Discrete geometric representation and cut cell integration

We discretize the domain $\Omega$ by embedding it in a regular grid. Specifically, we use a signed distance level set function defined over a doubly refined subgrid:

$$
\mathcal{G}^\phi = \{(ih/2, jh/2)\}.
$$

This doubly refined subgrid is thus a superset of all grid nodes in the $x$, $y$ and $p$ grids. The signed distance values at the nodes of the doubly refined grid $\mathcal{G}^\phi$ are used to determine the points of intersection between the zero isocontour and the coordinate axes aligned edges of
Figure 3.4: A zoom-in view of Figure 3.1(a). A levelset function is sampled on a doubly refined grid (left); a segmented curve $\partial \Omega_h$ is generated to approximate the boundary of the geometric domain (right).

$G^\phi$. The boundary of $\Omega$ is then approximated by a segmented curve $\partial \Omega^h$ connecting these intersection points. The geometric domain is approximated within the region enclosed by $\partial \Omega^h$ (see Figure 3.4). Near the boundary, the domain within each subgrid cell is approximated by a polygon determined from the boundary edges of the subgrid cell and by straight lines that connect boundary intersection points as demonstrated in Figure 3.1. Thus we can think of our discrete domain as a union of doubly refined uncut quadrilaterals on the interior and cut polygonal regions contained in doubly refined quadrilaterals on the boundary.

This partitioning of the domain into doubly refined quadrilaterals naturally supports our integration conventions needed for the matrices $A_{\omega_i}^{k^p}$ discussed in the previous section. The integrals needed for these matrices are evaluated trivially when $\omega_i$ is not cut. However when $\omega_i$ is cut by the boundary, we can still perform the integrations analytically following ideas from the recent cut cell approach in [13]. The integrands of each term in the matrices $A_{\omega_i}^{k^p}$ are polynomials in $x$ and $y$ of degree 2. That is, each integral is of the form:

$$\int_{\omega_i \cap \Omega} ax^2 + bxy + cy^2 + dx + ey + f d\vec{x}$$
Our level set based representation of the geometry means that the domain of integration \( \omega_i \cap \Omega \) is polygonal. In other words, we need to evaluate a second order polynomial over a polygonal domain. This task can be done trivially by noting that

\[
\int_{\omega_i \cap \Omega} ax^2 + bxy + cy^2 + dx + ey + f \, d\vec{x} = \int_{\omega_i \cap \Omega} \nabla \cdot \left( \begin{array}{c}
\frac{ax^3}{3} + \frac{bx^2y}{2} + cxy^2 + \frac{dx^2}{2} + exy + fx \\
0
\end{array} \right) \, d\vec{x}.
\]

That is, because \( \omega_i \cap \Omega \) is polygonal and our integrand can be expressed in terms of the divergence of a cubic function, application of the divergence theorem yields the easily evaluated formula:

\[
\int_{\omega_i \cap \Omega} \nabla \cdot \left( \begin{array}{c}
\frac{ax^3}{3} + \frac{bx^2y}{2} + cxy^2 + \frac{dx^2}{2} + exy + fx \\
0
\end{array} \right) \, d\vec{x} = \sum_{s=1}^{N_{\partial(\omega_i \cap \Omega)}} n_1(s) \int_{\partial(\omega_i \cap \Omega)_s} \left( \hat{a}(s)t^3 + \hat{b}(s)t^2 + \hat{c}(s)t + \hat{d}(s) \right) \, dt
\]

Here, the \( N_{\partial(\omega_i \cap \Omega)} \) is the number of line segments in the boundary of the polygonal domain, \( \partial(\omega_i \cap \Omega)_s \) is the \( s \)-th segment in the polygonal boundary, \( n_1(s) \) is the \( x \) component of the outward normal to the \( s \)-th segment and \( \hat{a}(s), \hat{b}(s), \hat{c}(s), \hat{d}(s) \) are the cubic coefficients arising in the boundary integrals over each segment. Again, each term in the sum can be evaluated analytically. This careful treatment of the integrals arising in each \( A_{\omega_i}^{k_p} \) is the key to obtaining second order accuracy in \( L^\infty \).
CHAPTER 4

Dirichlet boundary conditions

We have thus far assumed that our solution satisfies the Dirichlet boundary conditions and that our test functions vanish on the Dirichlet boundary. However, because we use a regular grid that does not conform to the actual domain, it is not convenient to directly define a finite element space with a specific value at the irregular boundary. Instead, we can enforce these conditions weakly using the following variational problem:

\[
\begin{align*}
\int_{\Omega} 2\mu \left( \frac{\nabla \bar{u} + \nabla \bar{u}^T}{2} \right) : \left( \frac{\nabla \bar{\nu} + \nabla \bar{\nu}^T}{2} \right) + \mu p (\nabla \cdot \bar{\nu}) \, d\bar{x} \\
= -\int_{\Omega} \bar{f} \cdot \bar{\nu} \, d\bar{x} + \int_{\partial \Omega} \bar{g} \cdot \bar{\nu} \, ds & \quad \forall \bar{\nu} \in H^1_{0,\Gamma_d}(\Omega) \times H^1_{0,\Gamma_d}(\Omega) \\
\int_{\Omega} \left( -\mu q \nabla \cdot \bar{u} - \frac{\mu^2}{2} \nabla p \right) \, d\bar{x} = 0 & \quad \forall q \in L^2(\Omega) \\
\int_{\Gamma_d} \bar{u} \cdot \bar{w} \, ds = \int_{\Gamma_d} \bar{u}_0 \cdot \bar{w} \, ds & \quad \forall \bar{w} \in (H^{-1/2}(\Gamma_d))^2.
\end{align*}
\]

Here, we introduce the Dirichlet condition as a constraint. Specifically, we require that the \(L^2\) inner product of the solution and an arbitrary function \(\bar{w} \in (H^{-1/2}(\Gamma_d))^2\) is the same as the inner product of the Dirichlet data \(\bar{u}_0\) with \(\bar{w}\). This makes the problem a constrained minimization.

4.1 Discretizing the Dirichlet problem

In order to discretize the Dirichlet condition in the weak formulation, we approximate \((H^{-1/2}(\Gamma_d))^2\) using a subspace \(A^h_x \times A^h_y = P_0(T^x \cap \Gamma_d^h) \times P_0(T^y \cap \Gamma_d^h)\), which is composed of piecewise constant functions over \(x\) and \(y\) component grid cells that intersect the Dirichlet
boundary. Here we use $\Gamma_d^h$ to denote the portion of $\partial \Omega^h$ over which the Dirichlet constraint is being enforced. We call any $x$ or $y$ cell $T^i$ with $T^i \cap \partial \Omega^h \neq \emptyset$ a boundary cell. The superscript $i$ is used to denote whether the cell is in the $x$ or $y$ grids with $i = 1$ signifying an $x$ cell and $i = 2$ signifying a $y$ cell. We use $\vec{w}_{T^i} = \chi_{T^i}(\vec{x})\vec{e}_i$ as the basis functions for $\Lambda_x^h \times \Lambda_y^h = P_0(T^x \cap \Gamma_d^h) \times P_0(T^y \cap \Gamma_d^h)$. Here, $\chi_{T^i}(\vec{x})$ is the characteristic function of the cell $T^i$:

$$\chi_{T^i}(x) = \begin{cases} 1, & \vec{x} \in T^i \\ 0, & \vec{x} \notin T^i. \end{cases}$$

Note that we have one basis function per boundary $x$ or $y$ cell. If we use $N_{xd}$ and $N_{yd}$ to denote the number of $x$ and $y$ boundary cells respectively, we can see that the dimension of the space $\Lambda_x^h \times \Lambda_y^h$ is $N_{xd} + N_{yd}$.

With this approximation, the Dirichlet boundary condition constraint can be expressed as a linear system $B^h \vec{u}^h = \vec{u}^h_0$ ($B^h \in \mathbb{R}^{(N_{xd}+N_{yd}) \times (N_x+N_y)}$, $\vec{u}^h_0 \in \mathbb{R}^{(N_{xd}+N_{yd})}$) where each equation enforces an integral constraint over the intersection of the discrete boundary $\Gamma_d^h$ with some $x$ or $y$ boundary cell $T^i$:

$$\sum_{k_i \in I_i} u_{ik_i} \int_{T^i \cap \Gamma_d^h} N_{ik_i}(\vec{x}) \, ds = \int_{T^i \cap \Gamma_d^h} u_{0i}(\vec{x}) \, ds$$

where $\vec{u}_0 = (u_{01}, u_{02})$. In practice, we evaluate the integral for a given boundary cell $T^i$ over the portion of the Dirichlet boundary curve $T^i \cap \Gamma_d^h$ from the four subquadrilaterals of $T^i$ arising from the doubly refined grid (as discussed in Section 3.3). This is simple because in each of these subquadrilaterals $\Gamma_d^h$ is just a single line segment. We use the following approximation for the right hand side terms in the constraint system:

$$\int_{T^i \cap \Gamma_d^h} u_{0i}(\vec{x}) \, ds = \sum_{j=1}^{4} \int_{\omega_j^{i} \cap \Gamma_d^h} u_{0i}(\vec{x}) \, ds \approx \sum_{j=1}^{4} u_{0i}(c(\omega_j^{i} \cap \Gamma_d^h)) \int_{\omega_j^{i} \cap \Gamma_d^h} 1 \, ds$$

where $\omega_j^{i}$ is one of the four subquadrilaterals of the cell $T^i$ and $c(\omega_j^{i} \cap \Gamma_d^h)$ is the midpoint of the segment $\omega_j^{i} \cap \Gamma_d^h$. We use the same treatment for the entries in the matrix on the left hand side:

$$\int_{T^i \cap \Gamma_d^h} N_{ik_i}(\vec{x}) \, ds = \sum_{j=1}^{4} \int_{\omega_j^{i} \cap \Gamma_d^h} N_{ik_i}(\vec{x}) \, ds.$$
We note that the integrand here is simply an $x$ or $y$ bilinear interpolating function. Therefore, the four terms in the sum can be evaluated analytically because they are simply quadratics in any linear parameterization of a given boundary segment $\omega^*_j \cap \Gamma^h_d$.

The discrete constrained minimization problem can be solved using a Lagrange multiplier method resulting in the following KKT system:

$$
\begin{pmatrix}
L^h_u & G^h & B^h \vphantom{G}^T \\
G^h & D_p^h & 0 \\
B^h & 0 & 0
\end{pmatrix}
\begin{pmatrix}
\vec{u}^h \\
\vec{\rho}^h \\
\vec{\lambda}^h
\end{pmatrix}
= 
\begin{pmatrix}
\vec{f}^h \\
\vec{0} \\
\vec{u}_0^h
\end{pmatrix}
$$

There is one Lagrange multiplier degree of freedom per Dirichlet constraint. In other words, $\vec{\lambda}^h$ is in $\mathbb{R}^{(N_{xd} + N_{yd})}$. When we consider boundary equations in the sections that follow, we temporarily eliminate pressure variables $\vec{\rho}^h$ with the following substitution $L^h = L^h_u - G^h (D_p^h)^{-1} G^h$:

$$
\begin{pmatrix}
L^h & B^h \vphantom{G}^T \\
B^h & 0
\end{pmatrix}
\begin{pmatrix}
\vec{u}^h \\
\vec{\lambda}^h
\end{pmatrix}
= 
\begin{pmatrix}
\vec{f}^h \\
\vec{u}_0^h
\end{pmatrix}
$$

This system is extremely ill-conditioned for nearly incompressible materials, however it will simplify the exposition of the forthcoming discussion of Dirichlet boundary condition treatment. Furthermore, when performing equation relaxation in our multigrid solver, we temporarily perform this elimination when treating equations near the boundary of the domain.

Before discussing our geometric multigrid solution approach for these systems of equations, we would like to first discuss some important aspects of the Dirichlet system. Our treatment of the Dirichlet condition is somewhat nonstandard and we list here a few important details related to the constraint matrix $B^h$.

1. $B^h$ consists of two decoupled blocks. There is one block for the $x$ boundary equations and one for the $y$ equations. The only non-zero columns of $B^h$ are associated with nodes that are incident on an $x$ or $y$ boundary cell. Therefore, for sufficiently interior degrees of freedom, the KKT system is exactly the same as (3.12) or (5.7). That is,
although the constraint matrix is in $\mathbb{R}^{(N_xd+N_yd)\times(N_x+N_y)}$, it really only acts on a small subset of the $N_x + N_y$ displacement degrees of freedom.

2. $B^h \in \mathbb{R}^{(N_xd+N_yd)\times(N_x+N_y)}$, where $(N_xd + N_yd) < (N_x + N_y)$. In fact,

$$B^h = \begin{pmatrix} B^x & 0 \\ 0 & B^y \end{pmatrix},$$

where $B^x \in \mathbb{R}^{N_xd \times N_x}$ and $B^y \in \mathbb{R}^{N_yd \times N_y}$. However, it can be shown that $B^h$ has full row rank. We refer the reader to the work [13] for a more detailed discussion of why this is so.

3. Our numerical linear algebra approach to the problem is based on the construction of a matrix $Z^h \in \mathbb{R}^{(N_x+N_y)\times((N_x+N_y)-(N_xd+N_yd))}$ (of full column rank) whose columns span the kernel space of $B^h$. Since $B^h$ has full row rank, there exists a column permutation $\mathcal{P}$, such that $B^h \mathcal{P} = [B_m|B_{n-m}]$, where $B_m \in \mathbb{R}^{(N_xd+N_yd)\times(N_xd+N_yd)}$ is non-singular and $B_{m-n} \in \mathbb{R}^{(N_xd+N_yd)\times((N_x+N_y)-(N_xd+N_yd))}$. With this permutation, we can construct a so-called fundamental basis for the null-space of $B^h$:

$$Z^h = \begin{bmatrix} -B_m^{-1}B_{n-m} \\ I \end{bmatrix}. \quad (4.6)$$

4. The vector

$$\bar{c}^h = \begin{bmatrix} B_m^{-1}\bar{u}_0^h \\ 0 \end{bmatrix} \quad (4.7)$$

satisfies $B^h \mathcal{P} \bar{c}^h = \bar{u}_0^h$. Therefore, all solutions $\bar{u}^h$ can be expressed as $\bar{u}^h = \mathcal{P}(\bar{c}^h + Z^h \bar{v}^h)$ with $\bar{v}^h \in \mathbb{R}^{((N_x+N_y)-(N_xd+N_yd))}.$

5. Our construction of a null-space for the Dirichlet constraint allows us to eliminate the Lagrange multipliers. Substituting $\bar{u}^h = \mathcal{P}(\bar{c}^h + Z^h \bar{v}^h)$ in (4.5), we have

$$L^h \mathcal{P}(\bar{c}^h + Z^h \bar{v}^h) + B^h \bar{\lambda}^h = \bar{f}^h$$
Left multiplying this equation with \((\mathbf{PZ}^h)^T\), and applying the property \(\mathbf{B}^h\mathbf{PZ}^h = 0\), we have

\[
(\mathbf{PZ}^h)^T \mathbf{L}^h \mathbf{PZ}^h \bar{v}^h + (\mathbf{B}^h \mathbf{PZ}^h)^T \bar{\lambda}^h = (\mathbf{PZ}^h)^T (\bar{f}^h - \mathbf{L}^h \mathbf{Pc}^h)
\]

\[
(\mathbf{Z}^h)^T \mathbf{P}^T \mathbf{L}^h \mathbf{PZ}^h \bar{v}^h = (\mathbf{PZ}^h)^T (\bar{f}^h - \mathbf{L}^h \mathbf{Pc}^h). \tag{4.8}
\]

That is, if we can solve \(\bar{v}^h\) from the reduced system (4.8), then the KKT system solution can be reconstructed with \(\bar{u}^h = \mathbf{P}(\bar{c}^h + \mathbf{Z}^h \bar{v}^h)\). Without loss of generality, we assume the variables to be reordered such that \(\mathbf{P} = \mathbf{I}\). We will make use of this property in the smoother for our geometric multigrid method.

### 4.2 Constructing the null-space for the Dirichlet constraints

Our treatment of the Dirichlet conditions is based on our ability to construct a null-space \(\mathbf{Z}^h\) satisfying \(\mathbf{B}^h\mathbf{Z}^h = 0\) and a special solution \(\bar{c}^h\) satisfying \(\mathbf{B}^h\bar{c}^h = \bar{u}^h_0\). The main issue we will discuss now is how to find a fundamental basis \(\mathbf{Z}^h\) that produces a numerically well-conditioned system of equations. This topic was originally discussed in the work of Bedrossian et al [13]. However, we found that in our case of nearly incompressible materials, an even more aggressive approach is needed. Bedrossian et al first suggested an ordering for the boundary integral equations and incident boundary nodes that led to a readily inverted upper triangular \(\mathbf{B}_m\). However, they showed that although this construction was straightforward, the conditioning of this \(\mathbf{B}_m\) deteriorated exponentially in the discretization resolution and was thus not practical. They then showed that it is possible to derive a diagonal \(\mathbf{B}_m\) with a slight modification to the definition of the constraints. Specifically, they showed that an aggregation scheme where cells of a “double-wide” grid were used to define the extent of the line integral constraints led to a diagonal \(\mathbf{B}_m\). This was done by choosing the center node of the 9 nodes incident on the four original cells in the “double-wide” cell as the representative node in the permutation of columns in \(\mathbf{B}^h\). In other words, the center node of the four aggregated cells was given the same index as the row associated with the constraint over those cells. This aggregation of cells is equivalent to replacing a collection of rows in the
original “single-wide” $B^h$ with the sum of the collection of rows. The choice of which rows to sum together is determined by the “double-wide” cell they belong to. Since no aggregated rows have a non-zero entry associated with the center node of any other aggregate, the $B_m$ is diagonal and thus $Z^h$ is trivially constructed. This aggregation can equivalently be seen as using a subspace $\Lambda_x^{2h} \times \Lambda_y^{2h} \cap H^{-1/2}(\Gamma_d)$ that is based on a coarsened grid. Remarkably, this process does not affect the $L^\infty$ convergence behavior of the scheme. Unfortunately, while the reduced system in [13] had a satisfactory condition number for the Poisson equation with an incomplete Cholesky preconditioned conjugate gradient solver, we found that this was not the case for nearly incompressible linear elasticity and geometric multigrid. Specifically, the conditioning of the equations at the boundary was poor enough to make the treatment of boundary regions prohibitively costly when performing the smoothing operations used in geometric multigrid.

We propose a new boundary integral constraint aggregation that allows us to generate a diagonal $B_m$ and a better-conditioned reduced system. We note that the elements of the original $B^h$ are scaled by a segment length; therefore, a very small segment (associated with a node that is far from the boundary) will generate very small diagonal elements for a $B_m$ arising from aggregation. We found this to be a source of the suboptimal conditioning. We propose a modification to the aggregation process that is designed to provide larger diagonal entries in subsequent $B_m$. We first define the weight of each node to be the sum of the column in the original “single-wide” $B^h$. This weight is the diagonal entry in $B_m$ that would arise from an aggregation of the four cells centered around the node. To increase the diagonal entries in the final $B_m$, we simply select four cell aggregates such that the associated representative nodes will be chosen in descending order of the total weight of each node. After we choose a node to define a four cell aggregate, the nine nodes incident on the four cells are eliminated from consideration. This process of prioritizing the aggregation rather than inheriting it from the coarse grid has a drawback. There may be some rows that are never added to an aggregate region. For these rows, it may be impossible to choose a representative node. We resolve this by aggregating such a row into a spatially adjacent...
aggregate. In practice, we also found that limiting representative nodes in the aggregation to ghost nodes (i.e. those geometrically outside the domain \( \Omega \)) gave even better conditioning. We briefly summarize this process in Algorithm 2.

**Algorithm 2 Aggregation Selection**

1: **procedure** \textsc{AggregationSelection}(\( B^x, B^y \))

2:   **for** \( v \) in \{1, 2\} **do**

3:      aggregation cells list \( \text{Aggr}^v = \{ \} \)

4:      aggregation representative list \( \text{Rep}^v = \{ \} \)

5:      set all active node\(^v\) variables and all integral cells\(^v\) active

6:      row weight sum \( w_i \leftarrow \sum_j B_{ij} \)

7:      Sort \( w \) in a decreasing order

8:      **for** \( w_j \) in \( w \) **do**

9:         if node\(^j\) is active **then**

10:            \( \text{Rep}^v \leftarrow \text{Rep}^v \cup \{ j \} \)

11:            \( \text{Aggr}^v \leftarrow \text{Aggr}^v \cup \{ c \text{ for all cell}_c \text{ adjacent to node}^j \} \)

12:            deactivate all cells adjacent to node\(^j\)

13:            deactivate all nodes adjacent to node\(^j\)

14:       **end for** \( \triangleright \) Now every ghost node belongs to at least one aggregation

15:   **for** all boundary cell\(^i\) **do**

16:      if cell\(^i\) is inactive **then**

17:         **Find** the closest aggregation of cell\(^i\) \( \text{Aggr}_k^i \)

18:         \( \text{Aggr}_k^i \leftarrow \text{Aggr}_k^i \cup \{ i \} \)

19:      **end for** \( \triangleright \) Pickup orphans

20: **end for**

21: **end procedure**
Figure 4.1: Cell aggregations. a) $x$ component boundary cells and the first two representative nodes and the incident $y$– cells of each representative node; b) all representative nodes for $y$ component cells and their incident cells, orphan cells will be attached to their nearest neighbor cells; c) and d) final cell aggregations together with their representative nodes for $x$ and $y$ grids.
CHAPTER 5

Multigrid

We develop an efficient multigrid solver for the discrete systems produced by our method. Our method is purely geometric, and based on the Multigrid Correction Scheme (see Algorithm 3). The framework admits a simple implementation, however special care is needed to retain near-textbook multigrid convergence rates, especially in the presence of highly irregular domains or nearly incompressible materials. The sections that follow will detail the key components of our multigrid solver: a hierarchy of discretizations, a smoothing procedure, and appropriate transfer operators (i.e. restriction and prolongation) between levels of the hierarchy. Although our design decisions include certain common practices, these components have been significantly customized to fit the needs of the specific discretization being followed, and facilitate both convergence and computational efficiency even near the incompressible limit.

Algorithm 3 Multigrid defect correction

1: procedure V-Cycle($\hat{\mathbf{L}}^h, \hat{\mathbf{u}}^h, \hat{\mathbf{f}}^h$
2: if problem at low resolution and easy to solve then
3: $\hat{\mathbf{u}}^h \leftarrow (\hat{\mathbf{L}}^h)^{-1}\hat{\mathbf{f}}^h$ and return;
4: PreRelaxation($\hat{\mathbf{L}}^h, \hat{\mathbf{u}}^h, \hat{\mathbf{f}}^h$
5: Restriction: $\hat{\mathbf{f}}^{2h} \leftarrow \mathbf{R}(\hat{\mathbf{f}}^h - \hat{\mathbf{L}}^h\hat{\mathbf{u}}^h)$
6: V-Cycle($\hat{\mathbf{L}}^{2h}, \hat{\mathbf{u}}^{2h}, \hat{\mathbf{f}}^{2h}$
7: Prolongation: $\hat{\mathbf{u}}^h \leftarrow \hat{\mathbf{u}}^h + \mathbf{P}\hat{\mathbf{u}}^{2h}$
8: PostRelaxation($\hat{\mathbf{L}}^h, \hat{\mathbf{u}}^h, \hat{\mathbf{f}}^h$
9: end procedure
5.1 Discretization hierarchy

We consider a hierarchy of resolutions, each corresponding to a discretization on a progressively larger grid size. In particular, we employ a grid step of $h$ on the finest level of the hierarchy (numbered as level zero), followed by discretizations with grid step sizes of $2h, 4h, \ldots, 2^L h$, for a total of $L + 1$ hierarchy levels. In detail, the hierarchy is constructed as follows:

- At every level of the hierarchy, say the $l$-th one, we define the background grids $G^x_{2^l h}, G^y_{2^l h}, G^p_{2^l h}$ corresponding to the $x$, $y$, and $p$-variables respectively.

- A level set function is computed over the respective doubly-refined subgrids $G^\phi_{h}, G^\phi_{2h}, G^\phi_{4h}, \ldots$ for each level. Obviously, coarser levels may fail to resolve certain high-frequency features of the domain geometry, leading to possible discrepancies between the discrete systems at various levels, which will be further addressed in our discussion of the smoother and transfer operators.

- Using the level set values associated with a given grid, we generate the discrete domains $T^x_{2^l h}, T^y_{2^l h}, T^p_{2^l h}$, and allocate the unknown arrays $\vec{u}^{2^l h}$ and $\vec{p}^{2^l h}$ as well as the right-hand sides $\vec{f}^{2^l h}$ and $\vec{f}^{2^l p}$ of the respective equations. Note that, although at the finest level of the hierarchy we have used $\vec{f}^h_p = 0$ by virtue of our discretization, the right hand side $\vec{f}^{2^l p}_p$ for coarser levels ($l \geq 1$) will generally be nonzero in the Multigrid Correction Scheme (see Algorithm 3).

The multigrid method is based on a hierarchical discretization. On each level $l$, we first rasterize our domain into regular cells with $h = 2^{-l}$, and define a staggered discretization on $G^x_h, G^y_h, G^p_h$. Based on the levelset function values sampled on the grid node points of a doubly refined grid, a geometric discretization of the domain is defined (see Figure 3.1 top right). The active degrees of freedom and the corresponding right hand side values are thus defined on the nodes of $x$, $y$ and $p$- cells that overlap with the discretized domain $\Omega^h$.
i.e. the nodes of the dashed line cells in Figure 3.1 bottom and the nodes of the thicker line cells in Figure 3.1 top left.

When there is a Dirichlet boundary condition, a constraint matrix $B^h$ is defined for each level, enforcing the integral of $x$ or $y$ displacement components along the discretized domain boundary $\partial \Omega^h$ of the current level and within each cell aggregation (each cell group with the same color in Figure 4.1) precomputed on the same level, to be the same as that of the Dirichlet values for the corresponding displacement component. Each of these constraints (or cell aggregations) corresponds to one Lagrange multiplier. In the fundamental basis method, we eliminate these multipliers by solving for the fundamental basis coefficients $\vec{v}^h$ in $c^h + Z^h \vec{v}^h$. By definition of $c^h$ (4.7) and $Z^h$ (4.6), there is a one-to-one mapping between $v^h$ components and active $x$ and $y$ degrees of freedom that are not aggregation representatives. Thus, the reduced system (4.8) is defined on these degrees of freedom only. Due to the fact that the reconstructed $u^h = c^h + Z^h \vec{v}^h$ satisfies the constraints automatically, we do not need to restrict any residual for the constraint system, i.e. on coarser levels, the Dirichlet boundary constraint values are always zero. Also, $\vec{\lambda}^h$ do not need to be solved, therefore, we do not need to record their values, nor prolongate their corrections. When we restrict the residuals of the governing equation, i.e. $\vec{r}^h = f^h - L^h \vec{u}^h - B^h \vec{\lambda}^h$, we restrict zero for all equations that will need a $\vec{\lambda}^h$ value. In other words, we restrict zero residuals from equations involving boundary nodes, i.e. the nodes in Figure 4.1 cells. Thus, a single level discretization is defined for all levels with powers of 2 resolutions.

### 5.2 Relaxation

The interior equations are uniform and have the same properties, while near the boundary, the equations have very different stencils. We distinguish the interior equations and boundary equations, and apply different relaxations on them. First, we recognize cells that are outside any $5 \times 5$ box centered at certain exterior cell, illustrated in Figure 5.1, right. For equations that are incident to these cells, a distributive interior relaxation is applied. Then,
we recognize cells that are active cells and within a $9 \times 9$ box centered at some exterior cells, illustrated in Figure 5.1, left. We apply a different boundary relaxation within this boundary band. In each single level relaxation, we first sweep over the boundary band, and apply a few iterations of boundary relaxations, then apply one iteration of interior relaxation followed by another few iterations of boundary relaxations.

The efficiency of a multigrid method is closely related to the smoothing efficiency of a single level relaxation. With Poisson’s equation, simple Jacobi or Gauss-Seidel will typically suffice as an efficient smoother. These techniques efficiently reduce the high-frequency component of the error and make it possible for a coarse grid to provide a meaningful correction to a finer grid. This property is fundamentally important for the efficiency of the geometrically hierarchical approach to solving the equations. Unfortunately, the equations of nearly incompressible linear elasticity with augmented pressure require more care than the comparably simplistic discrete Poisson equation. Although our system is not symmetric positive definite, we can modify the equations to a more convenient form as in [114] to design a proper geometric multigrid smoother. We confirm that a change of variables leads to an approximate block triangularization of the discrete system with each diagonal block being a
symmetric semi-definite discretization of Poisson. Our smoother is then constructed to be an emulation of the Gauss-Seidel relaxation applied on each block.

5.2.1 Approximated distributive relaxation

We follow the idea in [114] and develop a distributive relaxation. First, we apply a change of variable:

\[
\begin{pmatrix}
\tilde{u} \\
p
\end{pmatrix} =
\begin{pmatrix}
I & -\nabla \\
\nabla^T & -2\Delta
\end{pmatrix}
\begin{pmatrix}
\tilde{v} \\
q
\end{pmatrix}
\quad \text{or} \quad \tilde{u} = \tilde{M}\tilde{v} \tag{5.1}
\]

and substitute into (1.7) to achieve a new system

\[
\begin{pmatrix}
\mu\Delta I & 0 \\
\mu(1 + \frac{\nu}{\chi})\nabla^T & -\mu(1 + \frac{2\nu}{\chi})\Delta
\end{pmatrix}
\begin{pmatrix}
\tilde{v} \\
q
\end{pmatrix} =
\begin{pmatrix}
\tilde{f} \\
0
\end{pmatrix}
\quad \text{or} \quad \tilde{L}\tilde{M}\tilde{v} = \tilde{f} \tag{5.2}
\]

for some auxiliary variable \(\tilde{v} = (\tilde{v}, q)\). The derived PDE system is a block lower triangular system, and can be solved in a forward substitution process, i.e. first solve the \(\tilde{v}\) equations, and then freeze the \(\tilde{v}\) variables in the second equation and solve the \(q\) equation. Moreover, with certain choice of discretizations, the same triangulation can be realized on the discretized system, i.e. \(\tilde{L}_h\tilde{M}_h\) is also a block lower triangular linear system [114].

Due to the fact that each of the diagonal blocks of the discrete auxiliary system is a discretization of the Laplacian operator, we can relax the whole system using a Gauss-Seidel relaxation on each component of the \(\tilde{v}\) variables followed by another Gauss-Seidel relaxation on the \(p\) variables and achieve the same smoothing efficiency as that of the Gauss-Seidel relaxation applied on Poisson’s equation. At any approximation of \(\tilde{v}\) and \(q\), the approximate solutions to the augmented system can be reconstructed using (5.1).

In practice, we do not need to explicitly construct \(\tilde{v}\). In a Gauss-Seidel relaxation applied on \(\tilde{v}\), we iteratively solve for local corrections \(\hat{v}_i \leftarrow \hat{v}_i + \delta e_i\), such that the local residual \((\hat{f}_i - \tilde{L}\tilde{M}\hat{v})_i\) is zeroed out. Therefore, \(\delta = (\tilde{L}\tilde{M})^{-1}\hat{r}_i\). Such corrections invoke local corrections to \(\hat{u}\) in a distributive pattern, i.e. \(\hat{u}_i \leftarrow \hat{u}_i + \delta\hat{M}e_i\), thus defines the distributive relaxation scheme in Algorithm 4

31
Algorithm 4 Distributive Smoothing

1: procedure DistributiveSmoothing($\hat{L}^h, \hat{M}^h, \hat{\mu}^h, \hat{f}^h$) 

2: for $v$ in \{u_1, u_2, p\} do 
   \hspace{1em} $\triangleright$ Must iterate on $u_1$ and $u_2$ before $p$

3: for $i$ in Lattice[$v$] do 
   \hspace{1em} $\triangleright$ $i$ is an equation index

4: $r \leftarrow \hat{f}_i^h - \hat{L}_i^h \cdot \hat{\mu}^h$ 
   \hspace{1em} $\triangleright \hat{L}_i^h$ is the $i$-th row of $\hat{L}^h$

5: $\delta \leftarrow r / (\hat{L}^h \hat{M}^h)_{ii}$ 
   \hspace{1em} $(\hat{L}^h \hat{M}^h)_{ii}$ is a precomputed constant for each component

6: $\hat{\mu}^h += \delta m_i^T$ 
   \hspace{1em} $\triangleright m_i$ is the $i$-th row of $\hat{M}^h$

7: end for

8: end for

9: end procedure

For a staggered finite difference discretization, the triangularization of the discretized system can be achieved by discretizing the change of variable operator using centered differences for the gradient and divergence operators and a five point stencil for the Laplacian operator as shown in [114]. However, when we use a finite element discretization, there is no discrete change of variables with same sparsity that leads to an exact triangularization. Instead, we discretize the gradient operator in (5.1) using the stencils derived in a finite element method, i.e. $\nabla^h = \frac{1}{\mu h^2} G^h T = \begin{pmatrix} D_x^h \\ D_y^h \end{pmatrix}$ mapping from $p$ variables to $x$ and $y$ variables with the locations illustrated in Figure 3.3-right and the stencils being:

$$
D_x^h = \frac{1}{h} \begin{bmatrix} -1/8 & 1/8 \\ -3/4 & 3/4 \\ -1/8 & 1/8 \end{bmatrix}, \quad D_y^h = \frac{1}{h} \begin{bmatrix} 1/8 & 3/4 & 1/8 \\ -1/8 & -3/4 & -1/8 \end{bmatrix}. \quad (5.3)
$$

Similarly, the Laplacian operator in (5.1) is discretized from a standard piecewise bi-linear finite element discretization.

$$
M_p^h = \frac{1}{h^2} \begin{bmatrix} 1/3 & 1/3 & 1/3 \\ 1/3 & -8/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \end{bmatrix}. \quad (5.4)
$$
Although the linear system $\hat{L}^h \hat{M}^h$ is not block triangular, our numerical results show that the derived distributive relaxation is able to reduce the high-frequency error components efficiently. Results of using this relaxation scheme will be shown in Chapter 6.

5.2.2 Higher order defect correction

We also adopt the idea of higher order defect correction, which will allow us to develop a less expensive distributive relaxation. In a defect correction scheme for an arbitrary linear system $L\tilde{u} = \tilde{f}$, we solve for the correction $\delta\tilde{u} = \tilde{u}_{\text{exact}} - \tilde{u}$, which satisfies an equation $L\delta\tilde{u} = \tilde{f} - L\tilde{u}$.

In practice, we approximate the equation with another system $L_{\text{approx}}\delta\tilde{u} = \tilde{f} - L\tilde{u}$ that is easier to solve. For example, in a multigrid correction scheme, a coarse grid system is used as the approximated equation for solving the correction at the fine grid resolution, i.e. $L = L_{\text{fine}}$, $L_{\text{approx}} = L_{\text{coarse}}$. In the high order defect correction scheme, a lower order discretization is employed as the approximated system for solving a higher order discretization correction.

In our case, the finite difference discretization is a lower order system, and the finite element discretization is a high order system, i.e. $L = L_{\text{fem}}$, $L_{\text{approx}} = L_{\text{fdm}}$. In other words, we consider solving the following correction equation:

$$
\hat{L}^{f,d,h} \delta\tilde{u} = \tilde{f} - \hat{L}^{f,e,h} \tilde{u}.
$$

One of the benefits we obtain from such an approximation is that we can use the existing distributive relaxation with the exact triangulation of the discretized system (5.2) introduced in [80].

To be specific, let us rewrite the finite difference system as

$$
\hat{L}^{f,d,h} \tilde{u} = \left( \begin{array}{ccc}
L^{f,d,h}_{u} & G^{f,d,h} & \tilde{u}^{f,d,h} \\
G^{f,d,h} & D^{f,d,h}_{p} & \tilde{p}^{f,d,h} \\
\end{array} \right) \left( \begin{array}{ccc}
\tilde{u}^{f,d,h} \\
\tilde{p}^{f,d,h} \\
\end{array} \right) = \left( \begin{array}{ccc}
\tilde{f}^{f,d,h} \\
0 \\
\end{array} \right)
$$

and rewrite the finite element system scaled by $-1/h^2$ to match the scaling of the differential equation

$$
-\frac{1}{h^2} \hat{L}^{f,e,h} \tilde{u} = -\frac{1}{h^2} \left( \begin{array}{ccc}
L^{f,e,h}_{u} & G^{f,e,h} & \tilde{u}^{f,e,h} \\
G^{f,e,h} & D^{f,e,h}_{p} & \tilde{p}^{f,e,h} \\
\end{array} \right) \left( \begin{array}{ccc}
\tilde{u}^{f,e,h} \\
\tilde{p}^{f,e,h} \\
\end{array} \right) = -\frac{1}{h^2} \left( \begin{array}{ccc}
\tilde{f}^{f,e,h} \\
0 \\
\end{array} \right)
$$

33
In a high order defect correction scheme employing a finite difference discretization, we solve for a correction $\delta \tilde{u} = \tilde{M}^{fd,h} \delta \tilde{v}$ to locally satisfy

$$\hat{L}^{fd,h} \hat{M}^{fd,h} \delta \tilde{v} = -\frac{1}{h^2} (\tilde{f}^{fe,h} - \hat{L}^{fe,h} \tilde{u}^{current})$$

This derives a sparser distributive relaxation, shown in Algorithm 5.

**Algorithm 5** High Order Defect Correction Distributive Smoothing

1: procedure HighOrderDefectCorrectionDistributiveSmoothing($\hat{L}^{fd}, \hat{M}^{fd}, \hat{L}^{fe}, \hat{M}^{fe}, \tilde{u}, \tilde{f}^{fe,h}$)
2:     for $v$ in $\{u_1, u_2, p\}$ do
3:         for $i$ in Lattice[$v$] do
4:             $r \leftarrow \tilde{f}^{fe,h} - \hat{L}_i^{fe} \cdot \tilde{u}$
5:                 $\delta \leftarrow r / (\hat{L}^{fd} \hat{M}^{fd})_{ii}$
6:                 $\tilde{u} += \delta m_i^T$
7:     end for
8: end for
9: end procedure

The previous two types of distributive relaxation are not applicable for variables near the domain boundary. In fact, near the boundary, some of the variables in the distribution stencil may not exist. We follow the idea in [114], and temporarily build an unaugmented system in the boundary band (see Figure 5.1, left).

### 5.3 Boundary relaxation

Near the domain boundary, the previous distributive relaxation is not well defined; a special relaxation is required. In the Neumann boundary condition case, we eliminate $p$ from the
augmented system (3.11) by left multiplying the equation with
\[
\hat{U} = \begin{pmatrix} I & -GD_p^{-1} \\ 0 & I \end{pmatrix}.
\]

Therefore,
\[
\hat{U}L\tilde{u} = \begin{pmatrix} L_u - GD_p^{-1}G^T & 0 \\ G^T & D_p \end{pmatrix} \begin{pmatrix} \tilde{u} \\ p \end{pmatrix} = \hat{U}\tilde{f}.
\tag{5.7}
\]

In the first equation for \( \tilde{u} \), the equation is symmetric and positive definite, and hence can be solved again using Gauss-Seidel relaxation. This unaugmented system is a consistent discretization to the original PDE (1.7). Although Gauss-Seidel relaxation is not an efficient smoother for the unaugmented system if defined everywhere, for the purpose of boundary relaxation, we only build the temporary unaugmented system and relax it within a very narrow boundary band as demonstrated in Figure 5.1, and temporarily freeze the interior variables. The solution is strongly restricted by nearby interior values, therefore the Gauss-Seidel relaxation is still efficient and stable. Typically, with about 5 to 10 sweeps of boundary relaxation before and after each interior relaxation sweep, the boundary residual is reduced to as small as the interior residual. Once we relaxed \( \tilde{u} \) well enough, we freeze \( \tilde{u} \) and substitute into the second equation in (5.7) to resolve pressure variables.

### 5.4 Boundary relaxation for the reduced system in Dirichlet boundary condition case

In the Dirichlet boundary condition case, the boundary system (4.5) is a KKT system, which is indefinite, and cannot be resolved using Gauss-Seidel relaxations. Alternative approaches such as Kaczmarz relaxation or box relaxation may be efficient smoothers, however, their computational cost are much more expensive. Instead, we follow the fundamental basis method, to solve \( \tilde{v}^h \) from the reduced system (4.8), and reconstruct the solution of the KKT system (4.5) using \( \tilde{v}^h = \tilde{v} + Z^h\tilde{v}^h \). Since \( L^h \) is symmetric positive definite, \( Z^{hT}L^hZ^h \) is also symmetric positive definite, hence can be solved using Gauss-Seidel relaxation (see Algorithm
In practice, a Gauss-Seidel iteration on (4.8) iteratively solves for a correction on each single degree of freedom by solving the following scalar equation

\[
\ell_{ii}^T \ell_r^h (\ell_{ih} + \delta \ell_i) = \ell_{ii}^T \ell_r^h (f_r^h - \ell_r^h \ell_{ih}) ,
\]

where \( \ell_r^h = \ell_h^T \ell_r \ell_h \) i.e.

\[
(\ell_r^h)_{ii} \delta = \ell_{ii}^T \ell_h^T (f_r^h - \ell_r^h \ell_{ih} - \ell_r^h \ell_h \ell_{ih} \ell_{ih}^T) = \ell_{ii}^T \ell_r^T (f_r^h - \ell_r^h \ell_{ih})
\]

and then applies the correction: \( \ell_i^h \leftarrow \ell_i^h + \delta \ell_i \). Equivalently, \( \ell_i^h \) is updated as \( \ell_i^h \leftarrow \ell_i^h + \delta \ell_i \). Therefore we can equivalently solve for a correction on \( \ell_i^h \) to emulate the Gauss-Seidel iteration on \( \ell_i^h \) (see Algorithm 7).

**Algorithm 6** Dirichlet boundary relaxation - \( \ell_i^h \)

1: \( \ell_i^h \leftarrow \hat{0} \)
2: **for** \( i = 1 \) to \( m \) **do**
3: \( \delta \leftarrow \ell_{ii}^T \ell_h^T (f_r^h - \ell_r^h \ell_{ih} - \ell_r^h \ell_h \ell_{ih} \ell_{ih}^T) / (\ell_h^h)_{ii} \)
4: \( \ell_i^h \leftarrow \ell_i^h + \delta \ell_i \)
5: **end for**

**Algorithm 7** Dirichlet boundary relaxation - \( \ell_i^h \)

1: \( \ell_i^h \leftarrow \hat{c} \)
2: **for** \( i = 1 \) to \( m \) **do**
3: \( \delta \leftarrow \ell_{ii}^T \ell_h^T (f_r^h - \ell_r^h \ell_{ih}) / (\ell_h^h)_{ii} \)
4: \( \ell_i^h \leftarrow \ell_i^h + \delta \ell_i \)
5: **end for**

During the Dirichlet boundary relaxation applied on \( \ell_i^h \), a numerical error may be introduced driving \( \ell_i^h \) away from the linear space \( \ell_{ih} + \ell_{ih} \ell_h \). Also, when we apply coarse grid correction in Algorithm 3, the corrected solution \( \ell_i^h + P \ell_i^{2h} \) may not be in the solution space. Therefore, a projection onto the solution space needs to be applied after the prolongation.
step. First of all, since the projected solution is $\bar{\mathbf{u}}_p^h = \bar{\mathbf{c}}^h + \mathbf{Z}^h \bar{\mathbf{v}}^h$, for some $\bar{\mathbf{v}}^h$, and according to the definition of $\mathbf{Z}^h$ and $\bar{\mathbf{c}}^h$ in (4.6) and (4.7), we have $\bar{\mathbf{v}}^h = \mathbf{Q} \bar{\mathbf{u}}^h$, where $\mathbf{Q}$ projects a solution vector to a sub-vector by eliminating the degrees of freedom that correspond to aggregation representative nodes. Therefore, the projected solution is $\bar{\mathbf{u}}_p^h = \bar{\mathbf{c}}^h + \mathbf{Z}^h \mathbf{Q} \bar{\mathbf{u}}^h$.

### 5.5 Coarsening

In a geometric multigrid method, we define a discretization on each level. On the interior of the region, we restrict residuals from the fine grid to coarse grid by applying a restriction operator $\mathbf{R}$ for each component defined on the staggered grids with stencils illustrated in Figure 5.3. We consider two types of prolongation stencils. First, we consider prolongation $\mathbf{P}_{lo} = 2^d \mathbf{R}$. Second, we consider piecewise bi-linear interpolation for $\bar{\mathbf{u}}$ in combination with the same pressure prolongation as in $\mathbf{P}_{lo}$, which we donate as $\mathbf{P}_{hi}$.

![Figure 5.2: Coarsening of grid variables](image)

(a) Fine grid active cells  
(b) Coarse grid active cells, overlaid with the fine active cells demonstrated using dashed lines

However, on the boundary, there is no guarantee that all dependencies of the coarse grid variable restriction stencils are active fine grid degrees of freedom. Therefore, we truncate
our restriction stencils to the active degrees of freedom, which is equivalent to restricting zero residuals from inactive regions. Also, when a Dirichlet boundary condition presents, we can not compute the residuals $\vec{r}^h = \vec{f}^h - L^h \vec{u}^h - B^h T^h \lambda^h$ when a $\lambda$ value is involved. In this case, we apply a boundary relaxation strong enough such that the boundary residual is smaller than interior relaxations, and restrict zero boundary residual for these equations.

The coarse grid constraint system right hand side should have been computed from the restriction of the fine grid constraint system residual. However, due to the fact that our solutions $\vec{u}^h_p = \vec{c}^h + Z^h \vec{v}^h$ always satisfy the boundary constraints exactly, the coarse grid Dirichlet boundary condition is always zero.

Also, prolongation is implemented in a distributive way, i.e. we iterate over the active coarse grid corrections, and distribute their values to all active fine level degrees of freedom. Near the domain boundary this is equivalent to prolongating a zero correction from exterior coarse grid locations, which is reasonable. We notice that this prolongation may lead to a solution away from the fundamental basis solution. Therefore, a projection process is applied immediately after prolongation $\vec{u}^h \leftarrow \vec{c}^h + Z^h Q \vec{u}^h$.

So far, we discussed the components of a multigrid method, and we continue by discussing the numerical results.
We investigate two aspects of our algorithm: discretization error and multigrid efficiency. In this chapter, we apply our method on various domains with Neumann or Dirichlet boundary conditions and with a wide range of Poisson’s ratios. We considered three deformations defined on three geometric domains:

1. **Keyhole domain** A Keyhole domain is enclosed by a smooth curve connecting 8 tangential circles with centers

   \[ \vec{c}_1 = (0.25, 0.25); \quad \vec{c}_2 = (0.75, 0.25); \]
   \[ \vec{c}_3 = (0.25, 0.75); \quad \vec{c}_4 = (0.75, 0.75); \]
   \[ \vec{s}_1 = (0.5, 0.6875); \quad \vec{s}_2 = (0.5, 0.3125); \]
   \[ \vec{s}_3 = (0.3125, 0.5); \quad \vec{s}_4 = (0.6875, 0.5); \]

   and radius 0.2 for the first 4 circles and \( r_s = \frac{\sqrt{17}}{4} - 0.2 \) for the last 4 circles. The radius \( r_s \) is chosen such that the circle curves are tangential and hence generate a smooth boundary.

   The keyhole domain can also be represented by the zero levelset of the following function:

   \[ \varphi(\vec{x}) = \max(\min(\text{dist}(\vec{x}, \vec{c}_i, 0.2), \text{dist}(\vec{x}, \vec{0}, r_0)), -\min(\text{dist}(\vec{x}, \vec{s}_i, r_s))) \]

   where \( \text{dist}(\vec{x}, \vec{x}_0, r) = |\vec{x} - \vec{x}_0| - r \), and \( r_0 = \left| \frac{0.2}{\sqrt{17}}(4, 1) - (0.25, 0.25) \right| \).

   A constant divergence deformation is considered, giving the exact boundary conditions
and the exact solution for the purpose of error computation.

\[
\phi_1(x, y) = 2x + \frac{1}{2} \cos \pi x \sin \pi y \quad (6.1)
\]

\[
\phi_2(x, y) = 2y - \frac{1}{2} \sin \pi x \cos \pi y \quad (6.2)
\]

![Figure 6.1: Left: undeformed keyhole domain; right: deformed keyhole domain](image)

2. **Flower domain** A flower-shaped domain with inner radius 0.2, outer radius 0.4 is considered with a levelset function:

\[
\varphi(\vec{x}) = \text{dist}(\vec{x}, \vec{0}, 0.5, 0.3 + 0.1 \cos 5\theta)
\]

where \(\theta\) is the argument of \((x, y)\). A deformation with spatially varying divergence is considered as an exact solution.

\[
\phi_1(x, y) = \frac{2x}{\sqrt{\pi}} \cos \frac{\pi}{2} y \quad (6.3)
\]

\[
\phi_2(x, y) = \frac{2x}{\sqrt{\pi}} \sin \frac{\pi}{2} y \quad (6.4)
\]

3. **Spiral domain** We consider a spiral shaped domain, defined by the zero levelset of

\[
\varphi(\vec{x}) = r(\vec{y}) - (0.33 + 0.08 \cos 5\theta(\vec{y})^{\frac{1}{2}})
\]
where $\vec{y}$ is $\vec{x} - (0.5, 0.5)$ rotated around $(0.5, 0.5)$ by $\theta = 14(2r(\vec{x}))^{\frac{1}{2}}$ and the deformation:

\[
\phi_1(x, y) = \left(\frac{1}{2}x + \frac{1}{2}\right) \cos\left(\frac{\pi}{6} + \frac{2}{3}\pi y\right) \tag{6.5}
\]

\[
\phi_2(x, y) = \left(\frac{1}{2}x + \frac{1}{2}\right) \sin\left(\frac{\pi}{6} + \frac{2}{3}\pi y\right) \tag{6.6}
\]
6.1 Discretization error

All our testing domains are embedded in a $[0, 1]^2$ domain, and we discretize this square domain with a regular grid of different resolutions ranging from 32 to 1024 in each direction. We plotted $\log_2 |\tilde{u}_{\text{exact}} - \tilde{u}|_\infty$ versus $\log_2$ resolution and estimated the solution accuracy order by fitting the data with a linear function. We remove the Neumann boundary condition null space by enforcing a non-embedded Dirichlet condition on all degrees of freedom within the domain $[7/16, 9/16]^2$. From the plotted error convergence behavior, we observe a second-order convergence for all three types of domains (see Figure 6.4, Figure 6.5, Figure 6.6); for both Neumann and Dirichlet boundary conditions and for a wide range of material parameters including near-incompressible materials. We notice that the order of accuracy is slightly smaller for domains with complicated boundaries. An important source of the inaccuracy is introduced by the inconsistent domain discretization at different resolutions.

6.2 Multigrid efficiency

We also investigated the efficiency of multigrid methods. First of all, we consider a periodic boundary condition problem defined on $[0, 1]^2$ and with the exact solution being

$$\phi_1(x, y) = \sin 2\pi x + \cos 2\pi y$$
$$\phi_2(x, y) = \cos 2\pi x + \sin 2\pi y.$$ 

Although periodic boundary conditions will not appear in practical elasticity problems, we consider the periodic boundary condition problem to evaluate the multigrid solver while avoiding issues that may arise with boundary relaxation. We first fix the problem resolution to $128 \times 128$ and apply finite element distributive relaxation and the distributive relaxation for the finite difference defect correction problem as the interior relaxations. We also apply the bilinear prolongation and a prolongation with $P = 4R^T$. While low incompressibility problems generate convergence rates no larger than 0.3 for a multigrid V-(1,1) cycle with all different prolongation and distribution options, we focus on the the harder high-
incompressible case with Poisson’s ratio being 0.49, and investigate both V-(1,1) cycle and W-(1,1) cycle convergence. As shown in Table 6.1, both finite element distributive relaxation and finite difference defect correction scheme generate convergence rates less than 0.5 with a multigrid V-(1,1) cycle. Although finite difference distributive relaxation generates slower convergence than finite element distributive relaxation for the V-(1,1) cycle, with the help of a bilinear interpolation or W-(1,1) cycle, we are able to generate as good convergence rate of 0.23.

<table>
<thead>
<tr>
<th>boundary condition</th>
<th>distribution</th>
<th>multigrid cycle</th>
<th>$P_{hi}$</th>
<th>$P_{lo}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Periodic</td>
<td>FD</td>
<td>V-(1,1)</td>
<td>0.24</td>
<td>0.42</td>
</tr>
<tr>
<td></td>
<td>FD</td>
<td>W-(1,1)</td>
<td>0.23</td>
<td>0.25</td>
</tr>
<tr>
<td></td>
<td>FE</td>
<td>V-(1,1)</td>
<td>0.13</td>
<td>0.24</td>
</tr>
<tr>
<td></td>
<td>FE</td>
<td>W-(1,1)</td>
<td>0.13</td>
<td>0.30</td>
</tr>
<tr>
<td>Dirichlet</td>
<td>FD</td>
<td>V-(1,1)</td>
<td>0.72</td>
<td>0.72</td>
</tr>
<tr>
<td></td>
<td>FD</td>
<td>W-(1,1)</td>
<td>0.72</td>
<td>0.72</td>
</tr>
<tr>
<td></td>
<td>FE</td>
<td>V-(1,1)</td>
<td>0.37</td>
<td>0.36</td>
</tr>
<tr>
<td></td>
<td>FE</td>
<td>W-(1,1)</td>
<td>0.42</td>
<td>0.42</td>
</tr>
<tr>
<td>Neumann</td>
<td>FD</td>
<td>V-(1,1)</td>
<td>0.70</td>
<td>0.70</td>
</tr>
<tr>
<td></td>
<td>FD</td>
<td>W-(1,1)</td>
<td>0.68</td>
<td>0.68</td>
</tr>
<tr>
<td></td>
<td>FE</td>
<td>V-(1,1)</td>
<td>0.50</td>
<td>0.50</td>
</tr>
<tr>
<td></td>
<td>FE</td>
<td>W-(1,1)</td>
<td>0.35</td>
<td>0.35</td>
</tr>
</tbody>
</table>

Table 6.1: Periodic boundary condition multigrid asymptotic convergence rates (Poisson’s ratio=0.49, resolution=128 × 128). For optional prolongations, $P_{hi}$ is bilinear interpolation and $P_{lo}$ is for $P = 4R^T$. For the optional distributions, FE is the distribution matrix discretized with the bilinear finite element method, and FD is using defect correction by employing finite difference distributive relaxation.

We further investigate the multigrid convergence rate for various resolutions by sticking to one scheme which uses finite element distribution, low order prolongation and a V-(1,1)
cycle, and plot the convergence rate for problems discretized with resolutions from 32 to 1024 (see Figure 6.7). A consistent multigrid convergence rate is observed under refinement.

While all schemes give a nice convergence rate in periodic cases, the convergence rate with Neumann boundary conditions and Dirichlet boundary conditions varies. In non-trivial boundary condition cases, the convergence rate is mainly restricted by the efficiency of the boundary relaxation. Therefore, the convergence rate of a V-(1,1) cycle and a W-(1,1) cycle are very similar. Also different prolongation schemes generate very similar convergence rates (see Table 6.1 for the convergence rate of all algorithm options for the flower domain problem at a fixed resolution of $128 \times 128$). The difference between using a finite element distributive relaxation and a finite difference defect correction distributive relaxation reflects the efficiency of the whole smoother in combination with the boundary relaxations.

We further investigate the convergence rate under refinement, due to the fact that different prolongation schemes and multigrid cycles generate similar convergence rates. We only plot the asymptotic convergence rate of a V-(1,1) cycle with $P = 4R^T$ and using finite element distributive relaxation in the interior. For both the Dirichlet and Neumann boundary condition problem, we plot the asymptotic convergence rate for resolutions from 32 to 1024 and the residual reduction at each iteration for representative resolution numbers that are powers of 2. We observed consistent convergence rate at all resolutions (see Figure 6.8 and Figure 6.9).
Figure 6.4: Order of accuracy for keyhole domain; top: Neumann boundary condition; bottom: Dirichlet boundary condition; left: Poisson’s ratio=0.3; right: Poisson’s ratio=0.49; square marker: $x$ component; circle marker: $y$ component.
Figure 6.5: Order of accuracy for flower domain. Top: Neumann boundary condition, bottom: Dirichlet boundary condition; left: Poisson’s ratio=0.3; right: Poisson’s ratio=0.49; square marker: $x$ component; circle marker: $y$ component.
Figure 6.6: Order of accuracy for spiral domain. Top: Neumann boundary condition; bottom: Dirichlet boundary condition; left: Poisson’s ratio=0.3; right: Poisson’s ratio=0.49; square marker: $x$ component; circle marker: $y$ component.
Figure 6.7: Multigrid V-(1,1) cycle convergence under refinement (Poisson’s ratio = 0.49, periodic boundary conditions, finite element distribution)

Figure 6.8: Multigrid V-(1,1) cycle convergence rate at resolutions from 32 to 1024 (Poisson’s ratio = 0.49). Left: Dirichlet boundary condition; right: Neumann boundary condition.
Figure 6.9: Multigrid V-(1,1) cycle convergence under refinement (Poisson’s ratio = 0.49).

Left: Dirichlet boundary condition; right: Neumann boundary condition.
CHAPTER 7

Conclusion and future work

We developed a second-order mixed finite element discretization for linear elasticity of all material parameters from compressible to highly incompressible. We developed a multigrid method for the linear system induced by the discretization. By applying approximated distributive relaxation, we can achieve a fast and parameter-independent convergence rate when no boundary conditions present. With specified boundary conditions defined on a variety of domains, we also demonstrated that the multigrid method can maintain a good convergence rate with only a small numbers of boundary relaxations. However, the optimum convergence demonstrated in the periodic boundary condition case cannot be achieved. We are interested in investigating a more efficient boundary smoother that avoids unaugmentation and a continuous extension to Stokes problem.
Part II

An Adaptive Virtual Node Algorithm with Robust Mesh Cutting
CHAPTER 8

Introduction

Cutting tetrahedralized deformable objects with arbitrary triangulated surfaces is a natural requirement in applications like virtual surgery and crack propagation. The virtual node algorithm (VNA) was developed to model topological changes defined by cuts that do not lie on mesh facet boundaries [69]. Although splitting a tetrahedron mesh along element faces is the simplest means of changing mesh topology, it limits the paths of cutting surfaces to be aligned with the facets of the original grid. The VNA was designed to generalize this approach to cuts that follow more arbitrary geometric paths. Avoidance of expensive tetrahedral re-meshing approaches that rebuild the mesh to respect propagating cuts was the primary motivation. With the VNA, topological changes are achieved by duplicating mesh elements that intersect the cut. Duplicated copies of mesh elements then contain only a portion of the material being modeled (often referred to as embedded material), but all duplicate mesh elements are copies of original elements and thus have predictable (and ideally high quality) conditioning (Figure 10.2).

However, the original algorithm does have some obvious limitations. First, it is often desirable to allow a cut to overlap with nodes, edges or faces of mesh elements, but the original VNA approach does not handle these cases so the cutting surfaces must be perturbed to avoid them. Furthermore, an element cannot resolve further cuts once it’s split, because each edge can only be cut at one location (Figure 10.2). We provide a new approach that removes both of these limitations. First, we redevelop the original VNA in a manner that allows cuts to pass through all mesh facets including vertices, edges and faces. Second, we develop an adaptive approach to the embedding and duplication processes that allows
an element to be cut arbitrary number of times. This adaptivity is only used to refine
the material mesh for cutting, so ideal conditioning of the original simulation elements are
maintained.

Lastly, we provide a novel mesh intersection algorithm for robustly defining the intersec-
tions of the cutting triangulated surface against the tetrahedron mesh, which eliminates the
possibility of missing intersections and leading to incorrect cutting results.
Related work

Simulation of topological change in Lagrangian meshes was introduced to computer graphics in the pioneering work of [98]. [27] and [110] provide detailed surveys of applications of such mesh cutting.

Early approaches typically made use of simple separation along mesh element boundaries. [77] models fracture by breaking bonds in a cubic lattice, splines are used to smooth the breakage edge, but it’s hard to get rid of the isolated shards. [67] models objects as connected voxels and animates explosion by breaking their connections. Tetrahedral meshes are used in [93, 72, 76], splits are made along the tetrahedron faces. While most of these works focus on fracture modeling, [76] simulates surgical incisions with explicit triangulated surfaces. Element deletion is also used to represent topology changes [44, 37, 60, 43].

The available geometric detail in this type of approach was increased somewhat by subdivision of elements in the mesh prior to splitting [70, 17]; however, this tended to introduce elements with poor aspect ratios. More geometrically rich cutting surfaces were generated by allowing separation along more arbitrary paths (albeit with the expense of re-meshing). [73] models massive fracture under blast waves. [78, 79] subdivide the simulation mesh continuously to simulate brittle and ductile fractures. A large number of potentially ill-conditioned simulation elements can be produced under this framework. Recently, such approaches have been used to create some very compelling results for a variety of materials [103, 35, 109, 108, 51, 9].

Embedded methods have been developed to minimize the complexity of re-meshing by embedding material surfaces into the existing mesh [71, 69, 8, 92, 47, 82, 55]. The common
idea of these works is to run simulations on well conditioned embedding meshes, while use embedded material meshes for visualization, thus take care of both simulation efficiency and geometric flexibility. Although these works generalized the approach to fracture, the embedding idea goes back at least to free form deformations [90, 42, 30, 96].

Particle-based methods can also provide flexibility for topology change [85]. Other interesting models for cut patterns were developed in [58, 59, 73].
CHAPTER 10

Adaptive virtual node cutting algorithm

10.1 Overview

There are two tetrahedral meshes in our framework, a simulation mesh on which the simulation algorithm is run, and a material mesh whose elements are embedded in those of the simulation mesh to accurately represent the geometry of the object (middle left of Figure 10.1). The material mesh is also used to resolve cuts and determine the mesh topology. It can have much higher resolution than the simulation mesh to accurately resolve the cutting geometry, and there are no restrictions on the conditioning of its elements. Our cutting algorithm runs within seconds on high resolution meshes, such as cutting a tetrahedralized volume with 3 million tetrahedra by a 20K triangle mesh with delicate geometric details (Figure 12.4); while simulation is usually the bottleneck of computation, so it’s often desirable to have a low resolution simulation mesh in interactive applications such as virtual surgery, and its elements should be well conditioned to avoid stringent timesteps.

In our algorithm (Figure 10.1), the cutting surface is resolved at the resolution of the material mesh as intersections registered on the nodes, edges, faces and interior of its elements. The intersections are computed in a novel way that ensures robustness. We then split each element based on a “maximally split” configuration as shown in Figure 10.3, and merge the elements that are connected by the embedded material. Our ability to adaptively subdivide the material mesh removes the limitation of how many times an original material element can be cut. In this chapter, we first describe the split and merge algorithms followed by subdivision, then give a detailed treatment of our robust intersection computation.
Figure 10.1: Schematic overview of our cutting algorithm. In the middle row, the embedding elements are omitted due to space limit except for the left one. The embedding relationship is shown more completely in the second cut, while the first cut shows the splitting and merging of material in detail.
Figure 10.2: Left: The maximally split configuration for the original virtual node cutting algorithm in 2D. The cuts (red) can not pass through nodes or overlap with edges. Right: The element maximally splits into 3 pieces, materials are embedded in the duplicated copies of the original element. Nodes on the dotted segments are the “virtual nodes”.

10.2 Step1: Split

The original VNA allows for a finite number of embedded cuts in a given element. Specifically, individual tetrahedra in the mesh can only be split into at most four embedded subregions (one associated with each node). This implies a “maximally split” configuration of the mesh in which all possible cuts have been made. In the original algorithm, each disjoint piece in this configuration consists of the elements in the one-ring of the nodes in the original mesh. Every other node in the piece is then a duplicated copy of the original node (Figure 10.2). Although not originally described this way, the algorithm can be conceived of via manipulations in the material connections of these pieces in the maximally split configuration as in [92]. We will provide this material connection description of our modified algorithm here.

We modify the available splits of a material element from the original four node-associated regions to 24 sub-elements as shown in the top right of Figure 10.3, the 2D version is shown on its left. The four nodes of each sub-element lie on the node, edge, face and interior of the material element respectively, and their locations are determined by the intersections generated from the cutting surface. This modification of the maximally split configuration is designed to allow cuts to overlap with nodes, edges and faces of the material elements thus removing a major limitation of the original VNA.
We refer to faces of the sub-elements as cut faces since they will ultimately represent the embedded cut surface geometry. With this view, the first step in the splitting algorithm is to define which cut faces are active, which means the sub-elements are separated along them. There are 60 cut faces in each material element: 24 lying on element faces (6 on each face) that defines the splitting between material elements, 24 cutting through element faces (6 for each face), 12 cutting through element edges (2 for each edge), and we can simply store a bit flag for each of them. Due to the simplicity of our splitting scheme, turning on a cut face only amounts to checking whether the corresponding intersections are registered on the element by a cutting triangle, as shown in Figure 10.3. This approach greatly simplifies the polygonization process in [92] without losing much flexibility, especially with our adaptive refinement of the material mesh introduced in Section 10.4.

After turning on the cut faces, we compute connected components of sub-elements in each material element (top right of 10.1). Afterwards, a copy of the element is made for each connected component. Each copy has four of its own nodes distinct from any of the other copies, and keeps a copy of the active cut faces to remember the previous cuts when cuts are introduced incrementally. We simply use 24 flags (6 in 2D) to keep track of the sub-elements contained in each copy. The embedding simulation element is also duplicated and attached on each copy of its embedded material element. In practice, we only need to duplicate the material elements that are split and the simulation elements any of whose embedded elements are duplicated, rendering the splitting algorithm local to the cut regions.

10.3 Step 2: Merge

The duplication of each cut element generates excessive splits to the mesh, we need to merge the elements to recover the correct mesh topology. In this step, tetrahedron copies of an original material element created in the first phase are compared to copies of its face adjacent neighbor before splitting. If any of these copies share a material connection through their neighboring face, the three pairs of nodes on the copies of the face are condensed to three,
Figure 10.3: Our maximally split configuration in 2D contains three types of cut faces (top row). The red flag is set if intersections are registered at the green location and one of the blue locations on the triangle. In the maximally split configuration in 3D (top right), each color represents a different type of cut face, which are shown individually on the bottom row. The correspondence is, from bottom left to bottom right: green, red, yellow, and blue. For each of the four 3D cases (bottom), the shaded flag is set if intersections are registered at the red location, one of the green locations, and one of the blue locations on the tetrahedron.
thus sewing the elements back together along the face (from the middle right of Figure 10.1 to its left). This process only requires checking whether the shared face contains an inactive cut face that both elements share as sub-element faces, which is again a benefit from our simple splitting scheme. When two material elements are merged, the nodes of their embedding simulation elements are also condensed if they are duplicated from the same original node, thus we obtain the correct topology for both the material and simulation meshes. Similar to the splitting step, all these operations are local to the cut region.

Neither original VNA nor [92] allows cutting triangles to overlap with faces of the material elements, which requires error-prone perturbation of cut surfaces to prevent these cases. In our maximally split configuration however, each sub-element has a cut face laying on the faces of the material element, which naturally allows for splitting neighboring elements. In fact, we show in Chapter 11 that the ability to register degeneracies allows intersections to be computed robustly.

10.4 Step 3: Subdivide

The inherent maximally split configuration of the VNA places limitations on the number of times a material element can split, thus the number of cuts it can resolve. This limitation was a large motivation for the work in [92]. However, the generality of the available cutting surfaces in [92] comes at the cost of significant algorithmic complexity. Even with the extra complexity, that approach is still unable to treat cuts that pass through nodes, edges or faces.

After performing a cutting step, we perform a local subdivision on the material mesh without changing the embedding simulation mesh. We subdivide each duplicated material element into its sub-elements, and replace it with these new elements in the material mesh (the second figure in the middle row of Figure 10.1). When a new cut is introduced, the new elements can be split into 24 pieces just like the original material elements, thus extends the splitting capability of the original mesh. While this subdivision could clearly have been
performed while creating fewer elements, this simple strategy worked well for our needs.

To maintain the validity of the mesh, subdividing the cut elements may require their uncut neighbors to be refined as well. However, subdividing the neighbors will not produce new nodes, which stops the subdivision from propagating to the entire mesh and producing exponentially more elements. For example, in the second and third figures in the second row of Figure 10.1, the element on the bottom right is subdivided by connecting the intersection on an edge to its opposite node, which doesn’t produce any new subdivision points. This strategy can be easily extended to 3D.

Note that we perform the subdivision step after the cutting step. This forces each cut to be resolved at the resolution of the material mesh and avoids creating an unnecessarily refined mesh to resolve the details of a complex cutting surface. Rather, the subdivision step afterwards allows cuts to be cut again without introducing large deviations from the cutting surface where cuts meet; deviations are instead limited by the sizes of the tolerances used by the robust intersection routines, which are a small fraction of the size of an element.

When performing incremental cutting, we delay the subdivision step until the entire cut has been made. When an element is split due to the incremental cut, each copy receives a copy of the cutting flags. This allows cutting of the element to resume as more of the incremental cut becomes available while avoiding excessive subdivision on incremental cuts that consist of large numbers of small additions to the cutting surface.
CHAPTER 11

Robust intersection computation

11.1 Overview

As described above, our cutting algorithm completely relies on the intersections registered on the material mesh, however, intersection computations can be unreliable under floating point arithmetic. Consider the pair of edges $AB$ and $PQ$ in Figure 11.1, the intersection computation requires testing the signs of the signed areas of triangle $ABP$, $ABQ$, $APQ$ and $BPQ$. For example, the quantity $a = \overrightarrow{PA} \times \overrightarrow{PB}$ is twice the signed area of $ABP$, so it can be used for the sign test. When the point $P$ is close to or lies on $AB$, the floating point error accumulated in computing $a$ can exceed its magnitude and lead to an opposite sign. This kind of errors exist in other primitive pairs (point-point, point-edge, point-triangle, edge-triangle and point-tetrahedron) too, which make the intersection results unpredictable. Sometimes an element may fail to split when intersections are mis-registered, thus leads to an incorrect topology of the cut mesh.

Our strategy for robustly handling intersections is to register primitive pairs that are close to each other as degeneracies, and avoid computing the intersection of higher level primitive pairs that contain them. In the example above, we will register $P$ as lying on $AB$ if they are close under certain tolerance, thus no further intersection test between $AB$ and $PQ$ is necessary. In general, we register intersections in a hierarchical manner from the lowest level primitive pairs (vertex-vertex) to highest (edge-edge and point-triangle in 2D (Figure 11.2), edge-triangle and point-tetrahedron in 3D). With degeneracies registered, the signed areas or volumes in intersection tests will be far enough from zero so their signs
can be checked robustly. The main difficulty is to find the proper tolerances such that no
degeneracy or intersection is missed and robustness is guaranteed. This relies on bounding
the floating point errors in computing the signed areas and volumes.

In the following sections, we first describe our strategy for floating point error estimation,
then present the complete algorithms for intersection computations and proof of their
correctness.

11.2 Floating point error

As mentioned above, the key quantities in intersection computations are signed areas, signed
volumes which involves cross products and dot products of vectors. This section lists how
floating point errors accumulate in the basic arithmetic operations, thus the overall error

![Figure 11.1: The intersection test for two edges. Left: To test $P, Q$ are on different sides of $AB$, we require $\text{sgn}(\overrightarrow{PA} \times \overrightarrow{PB}) \neq \text{sgn}(\overrightarrow{QA} \times \overrightarrow{QB})$. Right: To test $A, B$ are on different sides of $PQ$, we require $\text{sgn}(\overrightarrow{AP} \times \overrightarrow{AQ}) \neq \text{sgn}(\overrightarrow{BP} \times \overrightarrow{BQ})$.](image-url)
bounds in these quantitates can be derived. We then instrumented our code to automatically track conservative floating point error bounds through the computations. This produces floating point error guarantees that places constraints on the values that may be chosen for the tolerances.

Let $\mathcal{F}[f(x)]$ indicate that the computation of $f(x)$ is being carried out under floating point computation, where $x$ is the input coordinates. In general, we will be deriving bounds for floating point error $E[f(x)] = |\mathcal{F}[f(x)] - f(x)|$ or relative error $R[f(x)]$, where $E[f(x)] = |f(x)|R[f(x)]$ for various operations. In the case of vector quantities, $R[u] = \max(R[u_x], R[u_y])$ for 2D and $R[u] = \max(R[u_x], R[u_y], R[u_z])$ for 3D. All coordinates are taken to be exact. Let $\epsilon < 10^{-5}$ be the machine epsilon.

For the purposes of computing floating point error we store for each quantity

- $s$: A string representing how the quantity is computed
- $c, p$: The maximum magnitude the quantity could have, in the form $cL^p$, where $L$ is the maximum edge length of the bounding box surrounding each primitive pair.

![Figure 11.2:](image) Left: two points within the vertex-vertex tolerance and are registered as intersecting (red). Middle: the two points separate, but the cutting segment remains within the edge-vertex tolerance and the edge-vertex pair is registered as intersecting (green). Right: the cutting segment is far enough from degeneracies that face-vertex (yellow) and edge-edge (orange) intersections are registered.
• \( n \): A flag indicating that the quantity is guaranteed to be nonnegative

• \( a, b \): relative floating point error, of the form \( 1 + a\epsilon + b\epsilon^2 \).

• \( r \): A flag indicating that the floating point error is relative to the quantity being computed. Otherwise, it is relative to \( cL^p \)

• \( v \): A flag indicating that the floating point error estimate is valid. This is used in cases where nothing useful can be said about the floating point error, such as when dividing by a quantity that cannot be bounded from zero.

The state is updated using a set of rules for each operation that is used. A result is invalid if any of its inputs are, and \( s \) is updated. If all inputs are valid and a rule from below applies, the result is valid. If more than one rule applies, the more restricted rule is used. The rules are applied automatically by instrumenting the code. Note that no upper bound can, in general, be derived for the case of division since no lower bound for the denominator is available. Since we only use division for normalizing vectors, the choice \( c = 1 \) will always be appropriate. This special case is noted in the table with a “*”.

We also assume there is no underflow or overflow in all operations. Underflow can be avoided by shifting the meshes when computing the intersections so their coordinates are all sufficiently bigger than zero. No overflow is a reasonable assumption for simulations that do not blow up.
<table>
<thead>
<tr>
<th>ID</th>
<th>Rule</th>
<th>Restrictions</th>
<th>$n$</th>
<th>$r$</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>$x = -x_1$</td>
<td>none</td>
<td>0</td>
<td>$r_1$</td>
</tr>
<tr>
<td>B</td>
<td>$x = x_1 + x_2$</td>
<td>$p_1 = p_2$</td>
<td>$n_1$ and $n_2$</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>$x = x_1 + x_2$</td>
<td>$n_1$ and $n_2$ and $r_1$ and $r_2$ and $p_1 = p_2$</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>D</td>
<td>$x = x_1 - x_2$</td>
<td>$p_1 = p_2$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>E</td>
<td>$x = x_1 - x_2$</td>
<td>$p_1 = p_2$ and $a_1 = b_1 = a_2 = b_2 = 0$</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>F</td>
<td>$x = x_1x_2$</td>
<td>none</td>
<td>$s_1 = s_2$ or $(n_1$ and $n_2)$</td>
<td>$r_1$ and $r_2$</td>
</tr>
<tr>
<td>G</td>
<td>$x = x_1/x_2$</td>
<td>$r_1$ and $r_2$</td>
<td>$n_1$ and $n_2$</td>
<td>1</td>
</tr>
<tr>
<td>H</td>
<td>$x = \sqrt{x_1}$</td>
<td>$n_1$ and $r_1$</td>
<td>1</td>
<td>$r_1$</td>
</tr>
<tr>
<td>I</td>
<td>$x =</td>
<td>x_1</td>
<td>$</td>
<td>none</td>
</tr>
<tr>
<td>J</td>
<td>$x = #$</td>
<td>none</td>
<td>$# \geq 0$</td>
<td>1</td>
</tr>
<tr>
<td>K</td>
<td>$x = \text{coord}$</td>
<td>none</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>Rule</th>
<th>$c$</th>
<th>$p$</th>
<th>$a$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>$x = -x_1$</td>
<td>$c_1$</td>
<td>$p_1$</td>
<td>$a_1$</td>
<td>$b_1$</td>
</tr>
<tr>
<td>B</td>
<td>$x = x_1 + x_2$</td>
<td>$c_1 + c_2$</td>
<td>$p_1$</td>
<td>$1 + \frac{c_1a_1 + c_2a_2}{c_1 + c_2}$</td>
<td>$1 + \frac{c_1(a_1 + b_1) + c_2(a_2 + b_2)}{c_1 + c_2}$</td>
</tr>
<tr>
<td>C</td>
<td>$x = x_1 + x_2$</td>
<td>$c_1 + c_2$</td>
<td>$p_1$</td>
<td>$1 + \max(a_1, a_2)$</td>
<td>$1 + \max(a_1 + b_1, a_2 + b_2)$</td>
</tr>
<tr>
<td>D</td>
<td>$x = x_1 - x_2$</td>
<td>$c_1 + c_2$</td>
<td>$p_1$</td>
<td>$1 + \frac{c_1a_1 + c_2a_2}{c_1 + c_2}$</td>
<td>$1 + \frac{c_1(a_1 + b_1) + c_2(a_2 + b_2)}{c_1 + c_2}$</td>
</tr>
<tr>
<td>E</td>
<td>$x = x_1 - x_2$</td>
<td>$c_1 + c_2$</td>
<td>$p_1$</td>
<td>$1$</td>
<td>0</td>
</tr>
<tr>
<td>F</td>
<td>$x = x_1x_2$</td>
<td>$c_1c_2$</td>
<td>$p_1 + p_2$</td>
<td>$1 + a_1 + a_2$</td>
<td>$(1 + a_1)(1 + a_2) + b_1 + b_2$</td>
</tr>
<tr>
<td>G</td>
<td>$x = x_1/x_2$</td>
<td>$1^*$</td>
<td>$p_1 - p_2$</td>
<td>$1 + a_1 + a_2$</td>
<td>$(1 + a_1)(1 + a_2) + b_1 + b_2 + a_2^2$</td>
</tr>
<tr>
<td>H</td>
<td>$x = \sqrt{x_1}$</td>
<td>$\sqrt{c_1}$</td>
<td>$p_1/2$</td>
<td>$1 + a_1/2$</td>
<td>$1 + b_1/2 +</td>
</tr>
<tr>
<td>I</td>
<td>$x =</td>
<td>x_1</td>
<td>$</td>
<td>$c_1$</td>
<td>$p_1$</td>
</tr>
<tr>
<td>J</td>
<td>$x = #$</td>
<td>$</td>
<td>#$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>K</td>
<td>$x = \text{coord}$</td>
<td>$1/2$</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
11.3 Tolerance model

Once we know the error that can occur in a quantity computed under floating point arithmetic, its comparison with tolerances can give us information about its bound under exact arithmetic, which can in turn be used to prove the robustness of our intersection computations.

The comparisons that we do take the general form $g(\tau) - f(x) \geq 0$, where $x$ represent the input coordinates and $f$ and $g$ are functions, with $g$ strictly monotonically increasing over a suitable range surrounding $\tau$. Since this check will be performed in code under floating point arithmetic, we need a way of determining what we can guarantee based on the outcome of this check. We do this by introducing two more tolerances, $\tau_{-}$ and $\tau_{+}$ such that $\tau_{-} < \tau < \tau_{+}$ and

$$g(\tau_{-}) - f(x) \geq 0 \implies F[g(\tau) - f(x)] \geq 0 \implies g(\tau_{+}) - f(x) \geq 0. \quad (11.1)$$

Note that the tolerance comparisons against $\tau_{-}$ and $\tau_{+}$ are in exact arithmetic. These comparisons do not occur in code but rather are used for proving correctness. Only the middle comparison occurs in floating point. This model allows us to isolate the uncertainties of floating point from other aspects of the analysis. For each such tolerance $\tau$, the bounding tolerances $\tau_{-}$ and $\tau_{+}$ must be chosen so that (11.1) holds.

Note that if $E[g(\tau) - f(x)] < |g(\tau) - f(x)|$, then the sign of $F[g(\tau) - f(x)]$ can be determined unambiguously and agrees with the corresponding exact comparison. In this case, (11.1) holds trivially. Consider instead the choices $x$ for which $E[g(\tau) - f(x)] \geq |g(\tau) - f(x)|$. For these $x$, we require the stricter condition

$$g(\tau_{-}) - f(x) \leq F[g(\tau) - f(x)] \leq g(\tau_{+}) - f(x), \quad (11.2)$$

from which $\tau_{-}$ and $\tau_{+}$ can be computed given a bound on $E[g(\tau) - f(x)]$. Note that (11.2) implies (11.1).

The other type of comparison that we require is a simple comparison against zero, $f(x) > 0$. To have any chance at robustness, we must guarantee that we always get these comparisons
right. The way we do that depends on the extent to which we can bound $f(x)$ from zero and the way in which $f(x) > 0$ is used. The possibilities that arise are

- $|f(x)| > \tau_-$ is guaranteed. We ensure $E[f(x)] < \tau_-$ and perform the test with $F[f(x)] > 0$.

- If an intersection exists, then $|f(x)| > \tau_+$. In this case we ensure $E[f(x)] < \tau_-$ and perform the tests $F[\tau - |f(x)|] \geq 0$ and $F[f(x)] > 0$. The test $F[\tau - |f(x)|] \geq 0$ implies no intersection, since $F[\tau - |f(x)|] \geq 0 \implies |f(x)| \leq \tau_+$. On the other hand, $F[\tau - |f(x)|] < 0 \implies |f(x)| > \tau_-$, so that $f(x) > 0$ is computed correctly.

### 11.4 2D

Before beginning our treatment of the individual cases, we list each tolerances ($T$) with their corresponding bounds ($T_-$ and $T_+$) in Figure 11.3 for reference.

#### 11.4.1 Tolerance Computation

We compute the maximum bounding box edge length for each primitive in the simulation mesh $L_0$ and for each primitive on the cutting mesh $L_1$. We will process any two primitives if their bounding boxes are within distance $\sigma_+$ of one another. Note that $\sigma_+$ is the maximum of the tolerances. The maximum coordinate difference that can be obtained (in exact arithmetic) is $\hat{L} = L_0 + L_1 + \sigma_+$. Using $\hat{L}$ as the upper bound, $\sigma_+ = 7\sqrt{\epsilon L}$. Thus, we can say

<table>
<thead>
<tr>
<th>Name</th>
<th>$T_-$</th>
<th>$T$</th>
<th>$T_+$</th>
<th>$R[T]$</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma$</td>
<td>$6.0\sqrt{\epsilon L}$</td>
<td>$6.5\sqrt{\epsilon L}$</td>
<td>$7.0\sqrt{\epsilon L}$</td>
<td>$4\epsilon$</td>
<td>vertex-vertex</td>
</tr>
<tr>
<td>$\tau$</td>
<td>$4.0\sqrt{\epsilon L}$</td>
<td>$4.5\sqrt{\epsilon L}$</td>
<td>$5.0\sqrt{\epsilon L}$</td>
<td>$4\epsilon$</td>
<td>edge-vertex</td>
</tr>
<tr>
<td>$\delta$</td>
<td>$5.0\sqrt{\epsilon L}$</td>
<td>$5.5\sqrt{\epsilon L}$</td>
<td>$6.0\sqrt{\epsilon L}$</td>
<td>$4\epsilon$</td>
<td>vertex-vertex degeneracy rejection</td>
</tr>
<tr>
<td>$\kappa$</td>
<td>$10\epsilon L^2$</td>
<td>$21\epsilon L^2$</td>
<td>$32\epsilon L^2$</td>
<td>$4\epsilon$</td>
<td>edge-edge area bound</td>
</tr>
</tbody>
</table>

Figure 11.3: Tolerances for 2D.
\[
\hat{L} = L_0 + L_1 + 7\sqrt{\epsilon\hat{L}} \text{ or } \hat{L} = \frac{L_0 + L_1}{1 - 7\sqrt{\epsilon}}.
\]
Further, we know that \(\mathcal{F}[\hat{L}] > (1 - 5\epsilon)\hat{L}\). Let \(k = 1 + 5\epsilon\).
To account for floating point error, we will instead define \(L = k\frac{L_0 + L_1}{1 - 7\sqrt{\epsilon}}\) (which corresponds to the definition \(\sigma_+ = 7\sqrt{\epsilon}L\) we will eventually use). With this definition, \(\mathcal{F}[L] > \hat{L}\). Then, \(L\), even though it is computed under floating point, is an upper bound on the exact difference between coordinates. The relative errors in the tolerances are shown in Figure 11.3.
Pseudocode for computing tolerances is in Algorithm 8.

\begin{algorithm}
\begin{algorithmic}
\Function{Compute_Tolerances}{A, B}
\State \(L_a\) \Comment{maximum bound box edge length of mesh A}
\State \(L_b\) \Comment{maximum bound box edge length of mesh B}
\State \(s \leftarrow \sqrt{\epsilon}\)
\State \(L \leftarrow \frac{1+5\epsilon}{1-7\sqrt{\epsilon}}(L_a + L_b)\)
\State \(t = sL\)
\State \(\sigma = 6.5t\)
\State \(\tau = 4.5t\)
\State \(\hat{\sigma} = 5.5t\)
\State \(\kappa = 21\epsilon L^2\)
\EndFunction
\end{algorithmic}
\end{algorithm}

11.4.2 Vertex-Vertex

Let \(A\) and \(B\) be two vertex locations. If

\[
d^2 = \|A - B\|^2 \leq \sigma^2
\]

we register a vertex-vertex intersection. In the cases that follow, we may now assume that \(\|A - B\| > \sigma_-\) in exact arithmetic. Pseudocode for this case is given in Algorithm 9.

Floating point: \(R[d^2] \leq 5\epsilon. \ E[d^2] \leq 9\epsilon L^2.\)
Algorithm 9 Vertex Vertex 2D
1: function VERTEX_VERTEX(A, B)
2:     if $\|A - B\|^2 \leq \sigma^2$ then
3:         return TRUE
4:     else
5:         return FALSE
6: end function

Tolerance constraints:

- $\min(\sigma_+ - \sigma, \sigma - \sigma_-) > 8\epsilon\sigma$.

11.4.3 Edge-Vertex

Let $AB$ and $P$ be an edge and a vertex. The check we need to perform is

$$d = \frac{\| (A - B) \times (P - A) \|}{\|A - B\|} \leq \tau \quad 0 < a < 1,$$

(11.4)

where $a$ is the interpolation weight. Let

$$\hat{a} = A - B \quad m = \| \hat{a} \| \quad u = \frac{\hat{u}}{m} \quad w = P - A \quad \hat{a} = u \cdot w \quad \hat{a} = m - \hat{a} \quad a = \frac{\hat{a}}{m} \quad d = |u \times w|.$$

(11.5)

Let us require that $\tau_+ < \frac{\sqrt{3}}{2}\sigma_-$. Treating vertex-vertex degeneracies guarantees $\|P - A\| > \sigma_-$ and $\|P - B\| > \sigma_-$. If in addition $m \leq \hat{\sigma}_+ = \sigma_-$, we conclude $d > \tau_+$ (See Figure 11.4(a)). Thus, step 7 of Algorithm 10 will not reject a valid intersection. This additional check allows us to assume $m > \hat{\sigma}_-$ in later stages, which in turn allows us to protect the division in step 10. Noting that $\hat{a}$ and $\hat{a}$ are signed distances to $A$ and $B$ along the segment $AB$, we also have $|\hat{a}| > \sqrt{\sigma^2_+ - \tau^2_+}$ and $|\hat{a}| > \sqrt{\sigma^2_+ - \tau^2_+}$ (See Figure 11.4(b)). We use these inequalities to protect the comparisons in step 17 of Algorithm 10.

Floating point: $R[m] \leq 4\epsilon$. $E[m] \leq 5\epsilon L$. $E[d] \leq 19\epsilon L$. $E[\hat{a}] \leq 19\epsilon L$. $E[\hat{a}] \leq 26\epsilon L$. 

71
Algorithm 10 Edge Vertex 2D

1: function Edge,Vertex(A, B, P)
2:   if Degenerate intersection then
3:     return (FALSE, 0)
4:   \( \hat{u} \leftarrow A - B \)
5:   \( m \leftarrow \|\hat{u}\| \)
6:   if \( m \leq \sigma \) then
7:     return (FALSE, 0)
8:   \( u \leftarrow \frac{\hat{u}}{m} \)
9:   \( w \leftarrow P - A \)
10:  if not \(|u \times w| \leq \tau\) then
11:     return (FALSE, 0)
12:   \( \hat{a} \leftarrow u \cdot w \)
13:   \( \bar{a} \leftarrow m - \hat{a} \)
14:  if \( \hat{a} < 0 \) or \( \bar{a} < 0 \) then
15:     return (FALSE, 0)
16:  return (TRUE, \( \frac{\hat{a}}{m} \))
17: end function

Figure 11.4: Proof illustrations for vertex edge in 2D

(a) Bound on \( m \).
(b) Bound on \( \hat{a} \) and \( \bar{a} \).
Tolerance constraints:

- $m > 0$, enforced by $m \geq \hat{\sigma}$, with $E[m] \leq 5\epsilon L$, so $\hat{\sigma} > 5\epsilon L$.
- $a > 0$, enforced by $a > \sqrt{\sigma^2 - \tau^2}$, with $E[a] \leq 26\epsilon L$, so $\sqrt{\sigma^2 - \tau^2} > 26\epsilon L$.
- $\dot{a} > 0$, enforced by $\dot{a} > \sqrt{\sigma^2 - \tau^2}$, with $E[\dot{a}] \leq 19\epsilon L$, so $\sqrt{\sigma^2 - \tau^2} > 19\epsilon L$.
- $\tau_+ < \frac{\sqrt{3}}{2} \sigma_-$.
- $\sigma_- = \hat{\sigma}_+$.
- $\min(\hat{\sigma}_+ - \hat{\sigma}, \hat{\sigma} - \hat{\sigma}_-) > 8\epsilon L$.
- $\min(\tau_+ - \tau, \tau - \tau_-) > 20\epsilon L$.

11.4.4 Edge-Edge

Let $AB$ and $PQ$ be two edges and let

$$a_A = 2 \text{area}(APQ) \quad a_B = 2 \text{area}(BPQ) \quad a_P = 2 \text{area}(ABP) \quad a_Q = 2 \text{area}(ABQ).$$

The edges $AB$ and $PQ$ intersect if $a_A$ and $a_B$ differ in sign and $a_P$ and $a_Q$ differ in sign.

Assuming no degeneracy has been registered and an edge-edge intersection exists, $\min(|a_A|, |a_B|, |a_P|, |a_Q|) \geq \kappa_+$, where $\kappa_+ = 2\tau^2$. Since these are symmetrical, it suffices to consider $a_P$. Let $\ell(AB)$ represent the length of segment $AB$. Figure 11.5(a) and Figure 11.5(a) show the major cases. In Figure 11.5(a), the point $R$ on line $AB$ closest to $P$ lies between $A$ and $B$, so that the edge-vertex degeneracy assumption implies $\ell(PS) \geq \ell(PR) > \tau_-$. Figure 11.5(b) shows a case where $R$ does not lie between $A$ and $B$. In this case, $\ell(PR) \geq \ell(RU) \geq \ell(BT) > \tau_-$. As before, $\ell(PS) \geq \ell(PR) > \tau_-$. Repeating the logic from points $A$ and $B$ gives the equivalent bounds $\ell(AS) > \tau_-$ and $\ell(BS) > \tau_-$. Finally, $|a_P| = \ell(AB)\ell(PR) = (\ell(AS) + \ell(BS))\ell(PR) > 2\tau^2$. 

73
Assuming the signs differ as required above, the interpolation weights are

\[ \alpha_{PQ} = \frac{a_P}{a_P - a_Q}, \quad \alpha_{AB} = \frac{a_A}{a_A - a_B}, \]  

which can always be computed robustly in floating point. The algorithm is shown in Algorithm 11.

**Algorithm 11** Edge Edge 2D

1: `function Edge_Edge(A, B, P)`
2:     `if` Degenerate intersection `then`
3:         `return` (FALSE, 0, 0)
4:     `a_P` ← `(P - A) × (B - A)`
5:     `a_Q` ← `(Q - A) × (B - A)`
6:     `if` \(|a_P| ≤ \kappa \) or \(|a_Q| ≤ \kappa \) or \(\text{sgn}(a_P) = \text{sgn}(a_Q)\) `then`
7:         `return` (FALSE, 0, 0)
8:     `a_A` ← `(A - P) × (Q - P)`
9:     `a_B` ← `(B - P) × (Q - P)`
10:    `if` \(|a_A| ≤ \kappa \) or \(|a_B| ≤ \kappa \) or \(\text{sgn}(a_A) = \text{sgn}(a_B)\) `then`
11:       `return` (FALSE, 0, 0)
12:    `return` (TRUE, \(\frac{a_A}{a_A - a_B}, \frac{a_P}{a_P - a_Q}\))
13: `end function`

**Floating point:** \(\mathcal{E}[a_P] ≤ 9\epsilon L^2\). \(\mathcal{E}[a_Q] ≤ 9\epsilon L^2\). \(\mathcal{E}[a_A] ≤ 9\epsilon L^2\). \(\mathcal{E}[a_B] ≤ 9\epsilon L^2\).

**Tolerance constraints:**

- \(\kappa_+ = 2\tau_-^2\).

- \(|a_P| > 0\), enforced by guaranteeing \(|a_P| > \kappa_-\) and requiring \(\kappa_- > \mathcal{E}[a_P] ≥ 9\epsilon L^2\).

- \(\min(\kappa_+ - \kappa, \kappa - \kappa_-) > 10\epsilon L^2\).
11.4.5 Triangle-Vertex

Let $ABC$ and $P$ be a triangle and a vertex. Let

$$a_A = 2 \text{area}(PBC) \quad a_B = 2 \text{area}(APC) \quad a_C = 2 \text{area}(ABP).$$

(11.8)

An intersection occurs if $\text{sgn}(a_A) = \text{sgn}(a_B) = \text{sgn}(a_C)$.

Assuming no degeneracy has been registered and an triangle-vertex intersection exists, $\min(|a_A|, |a_B|, |a_C|) \geq \kappa_+$, where $\kappa_+ = 2\tau_-^2$. Since these are symmetrical, it suffices to consider $a_C$. Since $|a_P| = \frac{1}{2}\ell(AB)\ell(PR)$, we need to bound $\ell(AB)$ and $\ell(PR)$. If $R$ is on segment $AB$, then $\ell(PR) > \tau_-$ by edge-vertex degeneracy as shown in Figure 11.6(a).

Otherwise, we have the case shown in Figure 11.6(a), in which case $\ell(PR) \geq \ell(PW) \geq \ell(PS) \geq \tau_-$. Similar bounds $\ell(PT) > \tau_-$ and $\ell(PS) > \tau_-$ are obtained when considering edges $AC$ and $BC$. In figure Figure 11.6(a) we see that $\ell(AB) \geq \ell(VU) \geq \ell(PT) + \ell(PS) \geq 2\tau_-$. Finally, $|a_P| = \frac{1}{2}\ell(AB)\ell(PR) > 2\tau_-^2$.

Since these areas all have the same sign, the barycentric weights are robustly computed in floating point as

$$\gamma_A = \frac{a_A}{a_A + a_B + a_C} \quad \gamma_B = \frac{a_B}{a_A + a_B + a_C} \quad \gamma_C = \frac{a_C}{a_A + a_B + a_C}.$$ 

(11.9)
The final algorithm is shown in Algorithm 12.

Algorithm 12 Triangle Vertex 2D

1: function TRIANGLE_VERTEX(A, B, P)
2:     if Degenerate intersection then
3:         return (FALSE, 0, 0, 0)
4:     a_A ← (B – P) × (C – P)
5:     a_B ← (P – A) × (C – A)
6:     a_C ← (B – A) × (P – A)
7:     if |a_A| ≤ κ or |a_B| ≤ κ or |a_C| ≤ κ or sgn(a_A) ≠ sgn(a_B) or sgn(a_B) ≠ sgn(a_C)
        then
8:         return (FALSE, 0, 0, 0)
9:     return (TRUE, \frac{a_A}{a_A + a_B + a_C}, \frac{a_B}{a_A + a_B + a_C}, \frac{a_C}{a_A + a_B + a_C})
10: end function

Floating point: \mathcal{E}[a_A] ≤ 9\epsilon L^2. \mathcal{E}[a_B] ≤ 9\epsilon L^2. \mathcal{E}[a_C] ≤ 9\epsilon L^2.

Tolerance constraints:

- \kappa_+ = 2\tau^-.
- a_A > 0, enforced by guaranteeing |a_A| > \kappa_- and requiring \kappa_+ > \mathcal{E}[a_A] ≥ 9\epsilon L^2.
- \min(\kappa_+ - \kappa, \kappa - \kappa_-) > 10\epsilon L^2.

11.5 3D

11.5.1 Tolerance Computation

The situation with L in 3D is similar to 2D, except that tolerances will now be of the form
7\epsilon^2 \hat{L}, so \hat{L} = \frac{L_0 + L_1}{1 – 7\epsilon^2}. Further, we know that \mathcal{F}[\hat{L}] > (1 – 5\epsilon)\hat{L}. Let k = 1 + 5\epsilon. To account
for floating point error, we will instead define $L = k \frac{L_0 + L_1}{1 - 7\epsilon^2}$. With this definition, $F[L] > \hat{L}$. Then, $L$, even though it is computed under floating point, is an upper bound on the difference that could be computed between two coordinates. Since each tolerance $\zeta$ is computed as $\zeta = c_\zeta \sqrt{\epsilon} L$ for some $c_\zeta$, $R[\zeta] < 4\epsilon$. Also, $R[\zeta^2] < 9\epsilon$ and $R[\sqrt{\zeta^2 - \kappa^2}] < 5\epsilon$. Pseudocode for computing tolerances is in Algorithm 13.

11.5.2 Vertex-Vertex

Let $A$ and $B$ be two vertex locations. If

$$d^2 = \|A - B\|^2 \leq \sigma^2$$

(11.10)

we register a vertex-vertex intersection. In the cases that follow, we may now assume that $\|A - B\| > \sigma_-$ in exact arithmetic. The algorithm is shown in Algorithm 14.

Floating point: $R[d^2] \leq 6\epsilon$. $E[d^2] \leq 16\epsilon L^2$.

Tolerance constraints:

- $\min(\sigma_+ - \sigma, \sigma - \sigma_-) > 8\epsilon \sigma$. 

Figure 11.6: Proof illustrations for triangle vertex in 2D.
Algorithm 13  Computing tolerances in 3D

1: function Compute_Tolerances(A, B)

2: \[ L_a \leftarrow \text{maximum bound box edge length of mesh } A \]

3: \[ L_b \leftarrow \text{maximum bound box edge length of mesh } B \]

4: \[ s \leftarrow \sqrt{\epsilon} \]

5: \[ a \leftarrow s^2 \]

6: \[ b \leftarrow sa \]

7: \[ L \leftarrow \frac{1+\frac{5\epsilon}{1+\epsilon}}{L_a + L_b} \]

8: \[ c = aL \]

9: \[ d = L^2 \]

10: \[ e = bLd \]

11: \[ f = \epsilon d^2 \]

12: \[ \sigma = 6.5c \]

13: \[ \tau = 4.5c \]

14: \[ \delta = 2.25c \]

15: \[ \gamma = 2.25c \]

16: \[ \hat{\sigma} = 5.5c \]

17: \[ \mu = 24e \]

18: \[ \rho = 56e \]

19: \[ \xi = 56e \]

20: \[ \lambda = 1215f \]

21: \[ \phi = 470f \]

22: \[ \nu = 6844.5f \]

23: \[ \zeta = 1317f \]

24: end function
<table>
<thead>
<tr>
<th>Name</th>
<th>$T_-$</th>
<th>$T$</th>
<th>$T_+$</th>
<th>$\mathcal{R}[T]$</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma$</td>
<td>$6.0\epsilon^3 L$</td>
<td>$6.5\epsilon^3 L$</td>
<td>$7.0\epsilon^3 L$</td>
<td>$4\epsilon^3 L$</td>
<td>vertex-vertex</td>
</tr>
<tr>
<td>$\tau$</td>
<td>$4.0\epsilon^3 L$</td>
<td>$4.5\epsilon^3 L$</td>
<td>$5.0\epsilon^3 L$</td>
<td>$4\epsilon^3 L$</td>
<td>edge-vertex</td>
</tr>
<tr>
<td>$\delta$</td>
<td>$2.0\epsilon^3 L$</td>
<td>$2.25\epsilon^3 L$</td>
<td>$2.5\epsilon^3 L$</td>
<td>$4\epsilon^3 L$</td>
<td>face-vertex</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>$2.0\epsilon^3 L$</td>
<td>$2.25\epsilon^3 L$</td>
<td>$2.5\epsilon^3 L$</td>
<td>$4\epsilon^3 L$</td>
<td>edge-edge</td>
</tr>
<tr>
<td>$\hat{\sigma}$</td>
<td>$5.0\epsilon^3 L$</td>
<td>$5.5\epsilon^3 L$</td>
<td>$6.0\epsilon^3 L$</td>
<td>$4\epsilon^3 L$</td>
<td>vertex-vertex degeneracy rejection</td>
</tr>
<tr>
<td>$\lambda$</td>
<td>$910\epsilon L^4$</td>
<td>$1215\epsilon L^4$</td>
<td>$1521\epsilon L^4$</td>
<td>$4\epsilon L^4$</td>
<td>edge-edge area bound</td>
</tr>
<tr>
<td>$\phi$</td>
<td>$150\epsilon L^4$</td>
<td>$470\epsilon L^4$</td>
<td>$760.5\epsilon L^4$</td>
<td>$4\epsilon L^4$</td>
<td>edge-edge volume bound</td>
</tr>
<tr>
<td>$\nu$</td>
<td>$3422.25\epsilon L^4$</td>
<td>$6844.5\epsilon L^4$</td>
<td>$10266.75\epsilon L^4$</td>
<td>$4\epsilon L^4$</td>
<td>triangle-vertex area bound</td>
</tr>
<tr>
<td>$\zeta$</td>
<td>$658\epsilon L^4$</td>
<td>$1317\epsilon L^4$</td>
<td>$\approx 1976\epsilon L^4$</td>
<td>$4\epsilon L^4$</td>
<td>triangle-vertex volume bound</td>
</tr>
<tr>
<td>$\mu$</td>
<td>$16\epsilon^3 L^3$</td>
<td>$24\epsilon^3 L^3$</td>
<td>$32\epsilon^3 L^3$</td>
<td>$4\epsilon^3 L^3$</td>
<td>triangle-edge volume bound</td>
</tr>
<tr>
<td>$\xi$</td>
<td>$28\epsilon^3 L^3$</td>
<td>$56\epsilon^3 L^3$</td>
<td>$\approx 83\epsilon^3 L^3$</td>
<td>$4\epsilon^3 L^3$</td>
<td>triangle-edge volume bound</td>
</tr>
<tr>
<td>$\rho$</td>
<td>$28\epsilon^3 L^3$</td>
<td>$56\epsilon^3 L^3$</td>
<td>$\approx 83\epsilon^3 L^3$</td>
<td>$4\epsilon^3 L^3$</td>
<td>tetrahedron-vertex volume bound</td>
</tr>
</tbody>
</table>

Figure 11.7: Tolerances for 3D.

**Algorithm 14** Vertex Vertex 3D

1: function VERTEX_VERTEX($A, B$)

2: if $\|A - B\|^2 \leq \sigma^2$ then

3: return TRUE

4: else

5: return FALSE

6: end function
11.5.3 Edge-Vertex

Let $AB$ and $P$ be an edge and a vertex. The check we need to perform is

$$d = \frac{\|(A - B) \times (P - A)\|}{\|A - B\|} \leq \tau \quad 0 < a < 1,$$

where $a$ is the interpolation weight. Let

$$\hat{u} = A - B \quad m = \|\hat{u}\| \quad u = \frac{\hat{u}}{m} \quad w = P - A \quad \hat{a} = u \cdot w \quad \bar{a} = m - \hat{a} \quad a = \frac{\hat{a}}{m} \quad d^2 = \|u \times w\|^2.$$  \hfill (11.12)

Let us require that $\tau_+ < \frac{\sqrt{3}}{2} \sigma_\minus$. The details of 3D are identical to 2D, except that $d$ now involves a square root rather than an absolute values. Since our crude approach to bounding roundoff error will not work for $\mathcal{E}[d]$, we instead check $d^2$ against $\tau^2$. The bounds $|\hat{a}| > \sqrt{\sigma_\minus^2 - \tau_+^2}$ and $|\bar{a}| > \sqrt{\sigma_\minus^2 - \tau_\plus^2}$ and the correctness of the algorithm (see Algorithm 15) are obtained in the same way as in 2D.

**Floating point:** $\mathcal{R}[m] \leq 4\epsilon$. $\mathcal{E}[m] \leq 7\epsilon L$. $\mathcal{E}[d^2] \leq 261\epsilon L^2$. $\mathcal{E}[\hat{a}] \leq 31\epsilon L$. $\mathcal{E}[\bar{a}] \leq 42\epsilon L$.

**Tolerance constraints:**

- $m > 0$, enforced by $m \geq \hat{\sigma}$, with $\mathcal{E}[m] \leq 7\epsilon L$, so $\hat{\sigma}_\minus > 7\epsilon L$.
- $\bar{a} > 0$, enforced by $\bar{a} > \sqrt{\sigma_\minus^2 - \tau_\plus^2}$, with $\mathcal{E}[\bar{a}] \leq 42\epsilon L$, so $\sqrt{\sigma_\minus^2 - \tau_\plus^2} > 42\epsilon L$.
- $\hat{a} > 0$, enforced by $\hat{a} > \sqrt{\sigma_\minus^2 - \tau_\plus^2}$, with $\mathcal{E}[\hat{a}] \leq 31\epsilon L$, so $\sqrt{\sigma_\minus^2 - \tau_\plus^2} > 31\epsilon L$.
- $\tau_+ < \frac{\sqrt{3}}{2} \sigma_\minus$.
- $\sigma_\minus = \hat{\sigma}_\plus$.
- $\min(\hat{\sigma}_\plus - \hat{\sigma}, \hat{\sigma} - \hat{\sigma}_\minus) > 9\epsilon L$.
- $\min(\tau_+ - \tau, \tau - \tau_\minus) > \frac{131\epsilon L^2}{\tau}$.
\begin{algorithm}
\caption{Edge Vertex 3D}
\begin{algorithmic}[1]
\Function{Edge\_Vertex}{A, B, P}
\If{Degenerate intersection}
\State \Return (FALSE, 0)
\EndIf
\State \( \hat{u} \leftarrow A - B \)
\State \( m \leftarrow \|\hat{u}\| \)
\If{\( m \leq \sigma \)}
\State \Return (FALSE, 0)
\EndIf
\State \( u \leftarrow \frac{\hat{u}}{m} \)
\State \( w \leftarrow P - A \)
\If{\( \|u \times w\|^2 \leq \tau^2 \)}
\State \Return (FALSE, 0)
\EndIf
\State \( \hat{a} \leftarrow u \cdot w \)
\State \( \tilde{a} \leftarrow m - \hat{a} \)
\If{\( \hat{a} < 0 \) or \( \tilde{a} < 0 \)}
\State \Return (FALSE, 0)
\EndIf
\State \Return (TRUE, \( \frac{\hat{a}}{m} \))
\EndFunction
\end{algorithmic}
\end{algorithm}
11.5.4 Edge-Edge

Let $AB$ and $PQ$ be two edges. Let

$$u = B - A \quad v = Q - P \quad w = P - A \quad r = u \times v \quad m^2 = \|r\|^2 \quad n = r \times w$$ \hspace{1cm} (11.13)

The distance $d$ and interpolation weights $a$ (from $A$ to $B$) and $b$ (from $P$ to $Q$) are

$$\hat{d} = r \cdot w \quad \hat{a} = n \cdot v \quad \hat{b} = n \cdot u \quad d = \frac{|\hat{d}|}{m} \quad a = \frac{\hat{a}}{m^2} \quad b = \frac{\hat{b}}{m^2} \quad \bar{a} = m^2 - \hat{a} \quad \bar{b} = m^2 - \hat{b}$$ \hspace{1cm} (11.14)

and the intersection should be registered if $\hat{a} > 0$, $\bar{a} > 0$, $\hat{b} > 0$, $\bar{b} > 0$, and $\hat{d}^2 < \gamma^2 m^2$. In the case that an intersection has occurred but no degenerate intersection has been registered, we can bound both $m$, $\bar{a}$, $\hat{a}$, $\bar{b}$, and $\hat{b}$ from zero.

11.5.4.1 Bound on $m$

Let $C$ be the point on $AB$ closest to $PQ$, and $R$ is the point on $PQ$ closest to $AB$. Assume WLOG $\beta = \ell(AC) = \min(\ell(AC), \ell(BC), \ell(PR), \ell(QR))$. Let $A'$ and $B'$ be the points $A$ and $B$ projected down to the plane $Z$ containing $PQ$ and parallel to $AB$ as shown in Figure 11.8(a). Assume also that the angle $\theta = A'RP$ is the smaller of the two angles made between $A'B'$ and $PQ$. Let $M$ be the point on $PQ$ closest to $A'$. Let $\alpha = \ell(A'M)$ and note that $\beta = \ell(AC) = \ell(A'R)$. $\sin \theta = \frac{a}{\beta}$. Note that the definition of $\beta$ implies that $M$ will lie on
Algorithm 16 Edge Edge 3D

1: function Edge_Edge(A, B, P, Q)
2:     if Degenerate intersection then
3:         return (FALSE, 0, 0)
4:     u ← B - A
5:     v ← Q - P
6:     r ← u × v
7:     m² ← ||r||²
8:     if m² ≤ λ then
9:         return (FALSE, 0, 0)
10:    w ← P - A
11:    d ← r · w
12:    if not d² ≤ γ²m² then
13:        return (FALSE, 0, 0)
14:    n ← r × w
15:    ā = n · v
16:    ō = n · u
17:    ā = m² - ā
18:    ō = m² - ō
19:    if ā ≤ φ or ō ≤ φ or ā ≤ φ or ō ≤ φ then
20:        return (FALSE, 0, 0)
21:    return (TRUE, ā/m², ō/m²)
22: end function
segment \( PQ \). Since \( \ell(AM) \) is an edge-vertex pair, \( \ell(AM) > \tau_\alpha \). \( \ell(AA') \) is the distance from \( AB \) to the plane of \( A'B' \), so \( \ell(AA') = d \leq \gamma_+ \). \( \beta \geq \alpha = \ell(A'M) = \sqrt{\ell(AM)^2 - \ell(AA')^2} > \sqrt{\tau_\alpha^2 - \gamma_+^2} \). Using the definition of \( \beta \), \( \|u\| = \ell(AB) = \ell(AC) + \ell(BC) \geq 2\beta \) and \( \|v\| = \ell(PQ) = \ell(PR) + \ell(QR) \geq 2\beta \). Finally, \( m = \|r\| = \|u \times v\| = \|u\|\|v\| \sin \theta \geq (2\beta)^2 \frac{d}{3} = 4\alpha \beta > 4(\tau_\alpha^2 - \gamma_+^2).

11.5.4.3 Algorithm

Pseudocode for the edge edge case is shown in Figure 16. The main difficulty in proving correctness is the form of the comparison on line 14. We must be able to bound \( E[m] \ll m \) for reasonable bounds to be found for \( \gamma_\alpha \) and \( \gamma_+ \). There are two difficulties with doing this. The first is that no bounds on \( E[m] \) can be derived using the simple automated framework in Section 11.2, and the second complication is that \( E[m^2] \) can be derived but not \( R[m^2] \). Using the test \( \hat{d}^2 \leq \gamma^2 m^2 \) rather than \( |\hat{d}| \leq \gamma m \) avoids the need to compute \( m \) entirely.

Deriving \( R[m^2] \) from \( E[m^2] \) can be achieved by bounding \( m^2 \) from below. Unfortunately, the bound on \( m^2 \) only holds provided the intersection exists, but the test on line 14 must
be reliable even when no intersection exists since it might otherwise allow an edge edge pair that is too distant to be counted as an intersection. The solution to this is to bound $m^2$ first using the test on line 9, where $\lambda_+ = 16(\tau_2 - \gamma_+^2)^2$. If the test on line 9 causes the algorithm to terminate, then $\mathcal{F}[\lambda - m^2] \geq 0$, in which case $m^2 \leq \lambda_+ = 16(\tau_2 - \gamma_+^2)^2$ implies no intersection. If the algorithm continues to line 12, then $m^2 \geq \lambda_-$. Correctness of line 14 depends on suitably choosing $\gamma_-$ and $\gamma_+$. 

What remains is to check the signs of $\hat{a}$, $\hat{a}$, $\hat{b}$, and $\bar{b}$. Following the established logic, we would do two checks for each, namely $|\hat{a}| \leq \phi$ and $\hat{a} < 0$, where $\phi_+ = 8(\tau_2 - \gamma_+^2)^2$. If either test succeeds, there is no intersection. Both tests together are equivalent (even under floating point) to the test $\hat{a} \leq \phi$. The logic is the same for the other three sign checks, so line 22 performs the remaining checks correctly. Note that the way $\bar{a}$ was computed and checked for sign implies that $\frac{\hat{a}}{m^2}$ will be computed between 0 and 1 in floating point. The case of $\bar{b}$ is similar, so that my returned results are robust (there may be some accuracy loss, however).

11.5.4.4 Addressing $\gamma^2 m^2 - \hat{d}^2$

We want $\mathcal{F}[\gamma^2 m^2 - \hat{d}^2] \geq 0$ to imply $\gamma_+^2 m^2 - \hat{d}^2 \geq 0$, and $\mathcal{F}[\gamma^2 m^2 - \hat{d}^2] < 0$ to imply $\gamma_-^2 m^2 - \hat{d}^2 < 0$, and we prove that the tolerances given in Figure 11.7 will guarantee this.

Since $m^2 > \lambda_- = 910\varepsilon L^4$ and $\mathcal{E}[m^2] \leq 129\varepsilon L^4$, $\mathcal{R}[m^2] \leq \frac{129}{990}$. Using $\mathcal{R}[\gamma^2] < 10\varepsilon$ we have $\mathcal{E}[\gamma^2 m^2] < (1 + 12\varepsilon)\mathcal{R}[m^2] + 12\varepsilon < \frac{1}{7}$ and then $\mathcal{E}[\gamma^2 m^2] < \frac{1}{7} \gamma^2 m^2$.

Note that if $\hat{d}^2 - \mathcal{E}[\hat{d}^2] > \gamma^2 m^2 + \mathcal{E}[\gamma^2 m^2]$, we will have $\mathcal{F}[\gamma^2 m^2 - \hat{d}^2] < 0$, so that the sign can be computed unambiguously. Thus, we can restrict ourselves to the case $\hat{d}^2 - \mathcal{E}[\hat{d}^2] \leq \gamma^2 m^2 + \mathcal{E}[\gamma^2 m^2]$ or $\hat{d}^2 \leq \bar{\phi} \gamma^2 m^2 + \mathcal{E}[\hat{d}^2]$. We also have $\mathcal{F}[\hat{d}^2] \leq \left( \hat{d} + \mathcal{E}[\hat{d}] \right)^2 (1 + \varepsilon) = \hat{d}^2 + \hat{d}^2 \varepsilon + 2\hat{d} \mathcal{E}[\hat{d}](1 + \varepsilon) + \mathcal{E}[\hat{d}]^2 (1 + \varepsilon)$ and $\mathcal{F}[\hat{d}^2] \geq \left( \hat{d} - \mathcal{E}[\hat{d}] \right)^2 (1 - \varepsilon) = \hat{d}^2 - \hat{d}^2 \varepsilon - 2\hat{d} \mathcal{E}[\hat{d}](1 - \varepsilon) + \mathcal{E}[\hat{d}]^2 (1 - \varepsilon)$, so $\mathcal{E}[\hat{d}^2] \leq \hat{d}^2 \varepsilon + 2 \hat{d} \mathcal{E}[\hat{d}](1 + \varepsilon) + \mathcal{E}[\hat{d}]^2 (1 + \varepsilon) \leq (\bar{\phi} \gamma^2 m^2 + \mathcal{E}[\hat{d}^2]) \varepsilon + 2 \sqrt{\bar{\phi} \gamma^2 m^2 + \mathcal{E}[\hat{d}^2] \mathcal{E}[\hat{d}](1 + \varepsilon) + \mathcal{E}[\hat{d}]^2 (1 + \varepsilon)}$. 

85
Let \( g(x) = (\frac{8}{7}\gamma^2 m^2 + x)e + 2\sqrt{\frac{8}{7}\gamma^2 m^2 + x}\mathcal{E}[\tilde{d}](1 + \epsilon) + \mathcal{E}[\tilde{d}]^2(1 + \epsilon) - x \). Note that
\[
\begin{align*}
g\left(\frac{2}{35}\gamma^2 m^2\right) &= \left(\frac{8}{7}\gamma^2 m^2 + \frac{2}{35}\gamma^2 m^2\right)e + 2\sqrt{\frac{8}{7}\gamma^2 m^2 + \frac{2}{35}\gamma^2 m^2}\mathcal{E}[\tilde{d}](1 + \epsilon) + \mathcal{E}[\tilde{d}]^2(1 + \epsilon) - \frac{2}{35}\gamma^2 m^2 \\
&= \gamma^2 m^2\left(\frac{6}{5}e - \frac{2}{35}\right) + 2\sqrt{\frac{6}{5}\gamma m\mathcal{E}[\tilde{d}](1 + \epsilon) + \mathcal{E}[\tilde{d}]^2(1 + \epsilon)} \\
&\leq \gamma^2 m^2\left(\frac{6}{5}e - \frac{2}{35}\right) + 2\sqrt{\frac{6}{5}\gamma m(47\epsilon L^3)(1 + \epsilon) + (47\epsilon L^3)^2(1 + \epsilon)} \\
&= \left(\gamma m\left(\frac{6}{5}e - \frac{2}{35}\right) + 94\sqrt{\frac{6}{5}\epsilon L^3(1 + \epsilon)}\right)\gamma m + (47\epsilon L^3)^2(1 + \epsilon) \\
&\leq \left((2.25\epsilon^{\frac{2}{3}} L)(4(\tau_2^2 - \gamma_2^2))\left(\frac{6}{5}e - \frac{2}{35}\right) + 94\sqrt{\frac{6}{5}\epsilon L^3(1 + \epsilon)}\right)\gamma m + (47\epsilon L^3)^2(1 + \epsilon) \\
&= \left(\frac{351}{4}\epsilon^{\frac{5}{3}} L^3\left(\frac{6}{5}e - \frac{2}{35}\right) + 94\sqrt{\frac{6}{5}\epsilon L^3(1 + \epsilon)}\right)\epsilon^{\frac{5}{3}} L^3\gamma m + (47\epsilon L^3)^2(1 + \epsilon) \\
&\leq -5\epsilon^{\frac{2}{3}} L^3 \gamma m + (47\epsilon L^3)^2(1 + \epsilon) \\
&\leq -5\epsilon^{\frac{2}{3}} L^3(2.25\epsilon^{\frac{2}{3}} L)(4(\tau_2^2 - \gamma_2^2)) + (47\epsilon L^3)^2(1 + \epsilon) \\
&\leq -\frac{1755}{4}\epsilon^{\frac{2}{3}} L^6 + (47\epsilon L^3)^2(1 + \epsilon) \\
&= \left(-\frac{1755}{4} + 47^2\epsilon^{\frac{2}{3}} (1 + \epsilon)\right)\epsilon^{\frac{2}{3}} L^6 \\
&\leq -400\epsilon^{\frac{2}{3}} L^6 \\
&< 0
\end{align*}
\]

86
Further, if $x \geq \frac{2}{35} \gamma^2 m^2$ then

$$g'(x) = \epsilon + \frac{\mathcal{E}[\hat{d}](1 + \epsilon)}{\sqrt{\gamma^2 m^2 + x}} - 1$$

$$\leq \epsilon + \frac{\mathcal{E}[\hat{d}](1 + \epsilon)}{\sqrt{\gamma^2 m^2}} - 1$$

$$= \epsilon + \frac{\mathcal{E}[\hat{d}](1 + \epsilon)}{\gamma m} - 1$$

$$\leq \epsilon + \frac{47\epsilon L^3(1 + \epsilon)}{(2.25\epsilon^4 L)(4(\tau^2 - \gamma^2_+))} - 1$$

$$= \epsilon + \frac{188 \epsilon^4 (1 + \epsilon)}{351} - 1$$

$$\leq \epsilon + \frac{188 \epsilon^4 (1 + \epsilon)}{351} - 1$$

$$< -0.9$$

Thus, I conclude that $g(x) < 0$ for all $x \geq \frac{2}{35} \gamma^2 m^2$. Since $g(\mathcal{E}[\hat{d}^2]) \geq 0$, we must have $\mathcal{E}[\hat{d}^2] \leq \frac{2}{35} \gamma^2 m^2$.

We know $\mathcal{F}[(\gamma^2 m^2 - \hat{d}^2)] < 0$ implies $\mathcal{F}[\gamma^2 m^2] - \mathcal{F}[\hat{d}^2] < 0$, which leads to $0 > \mathcal{F}[\gamma^2 m^2] - \mathcal{F}[\hat{d}^2] > \gamma^2 m^2 - \mathcal{E}[\gamma^2 m^2] - \hat{d}^2 - \mathcal{E}[\hat{d}^2] > \gamma^2 m^2 - \frac{1}{7} \gamma^2 m^2 - \hat{d}^2 - \frac{2}{35} \gamma^2 m^2 \geq \frac{4}{5} \gamma^2 m^2 - \hat{d}^2 > \gamma^2 m^2 - \hat{d}^2$.

Similarly $\mathcal{F}[\gamma^2 m^2 - \hat{d}^2] \geq 0$ implies $\mathcal{F}[\gamma^2 m^2] - \mathcal{F}[\hat{d}^2] \leq \gamma^2 m^2 + \mathcal{E}[\gamma^2 m^2] - \hat{d}^2 + \mathcal{E}[\hat{d}^2] \leq \gamma^2 m^2 + \frac{1}{7} \gamma^2 m^2 - \hat{d}^2 + \frac{2}{35} \gamma^2 m^2 = \frac{9}{5} \gamma^2 m^2 - \hat{d}^2 \leq \gamma^2 m^2 - \hat{d}^2$.

**Floating point:** $\mathcal{E}[m^2] \leq 129 \epsilon L^4$. $\mathcal{E}[\hat{d}] \leq 47 \epsilon L^3$. $\mathcal{E}[\hat{d}^2] \leq 589 \epsilon L^6$. $\mathcal{E}[\hat{a}] \leq 129 \epsilon L^4$. $\mathcal{E}[\hat{b}] \leq 129 \epsilon L^4$. $\mathcal{E}[\hat{a}] \leq 281 \epsilon L^4$. $\mathcal{E}[\hat{b}] \leq 281 \epsilon L^4$. $m^2 \leq 12 L^4$.

**Tolerance constraints:**

- $\lambda_+ = 16(\tau^2 - \gamma^2_+)^2$

- $\phi_+ = 8(\tau^2 - \gamma^2_+)^2$

- $m^2 \leq \lambda$ with $\mathcal{E}[m^2] \leq 129 \epsilon L^4$ leads to $\min(\lambda_+ - \lambda, \lambda - \lambda_+) > 130 \epsilon L^4$
11.5.5 Triangle-Vertex

Let $ABC$ and $P$ be a triangle and a vertex. Then, Let

$$u = B - A \quad v = C - A \quad w = P - A \quad r = u \times v \quad m^2 = ||r||^2 \quad n = r \times w \quad (11.15)$$

Now, the signed distance $d$ and barycentric weights $a$, $b$, $c$ can be computed from

$$\hat{d} = r \cdot w \quad \hat{c} = -n \cdot u \quad \hat{b} = n \cdot v \quad \hat{a} = m^2 - \hat{b} - \hat{c} \quad d = \frac{\hat{d}}{m} \quad c = \frac{\hat{c}}{m^2} \quad b = \frac{\hat{b}}{m^2} \quad a = \frac{\hat{a}}{m^2}. \quad (11.16)$$

The intersection criterion is now $\hat{d}^2 \leq \delta^2 m^2$, $\hat{a} > 0$, $\hat{b} > 0$, and $\hat{c} > 0$. If satisfied and no degenerate intersection was registered, then we can bound $m$, $\hat{a}$, $\hat{b}$, and $\hat{c}$ from zero.

11.5.5.1 Bound on $m$

Let $P'$ be $P$ projected into the plane $ABC$ as shown in Figure 11.9(a). $P$ projects to lines $AB$, $AC$, and $BC$ at $T$, $S$, and $R$ respectively. If $R$ lies between $B$ and $C$, then $\ell(PR) > \tau_-$. 

Figure 11.9: Proof illustrations for triangle vertex in 3D.

- $\hat{a} \leq \phi$ with $\mathcal{E}[\hat{a}] \leq 129\epsilon L^4$ leads to $\min(\phi_+ - \phi, \phi - \phi_-) > 130\epsilon L^4$.
- $\bar{a} \leq \phi$ with $\mathcal{E}[\bar{a}] \leq 281\epsilon L^4$ leads to $\min(\phi_+ - \phi, \phi - \phi_-) > 282\epsilon L^4$.
- $\hat{d}^2 \leq \gamma^2 m^2$, addressed in Section 11.5.4.4.
Algorithm 17 Triangle Vertex 3D

1: function TRIANGLE_VERTEX(A, B, C, P)

2: if Degenerate intersection then

3: return (FALSE, 0, 0, 0)

4: $u \leftarrow B - A$

5: $v \leftarrow C - A$

6: $r \leftarrow u \times v$

7: $m^2 \leftarrow \|r\|^2$

8: if $m^2 \leq v$ then

9: return (FALSE, 0, 0, 0)

10: $w \leftarrow P - A$

11: $\hat{d} \leftarrow r \cdot w$

12: if not $\hat{d}^2 \leq \delta^2 m^2$ then

13: return (FALSE, 0, 0, 0)

14: $n \leftarrow r \times w$

15: $\hat{b} = n \cdot v$

16: $\hat{c} = -n \cdot u$

17: $\hat{a} = m^2 - \hat{b} - \hat{c}$

18: if $\hat{a} \leq \zeta$ or $\hat{b} \leq \zeta$ or $\hat{c} \leq \zeta$ then

19: return (FALSE, 0, 0, 0)

20: return (TRUE, $\frac{\hat{a}}{m^2}$, $\frac{\hat{b}}{m^2}$, $\frac{\hat{c}}{m^2}$)

21: end function
Since $\ell(P'P) = d \leq \delta_+$, we have $\ell(P'R) > \sqrt{\tau_+^2 - \delta_+^2}$, with similar bounds for $\ell(P'S)$ and $\ell(P'T)$. If $R$ projects beyond $B$ and $C$, the same bound can be obtained from $\ell(P'S)$ or $\ell(P'T)$ using the same logic as in the 2D case. Note that the inradius $r$ of $ABC$ must satisfy $r > \sqrt{\tau_+^2 - \delta_+^2}$. Noting also that the ratio of the area of the incircle to the area of the triangle must satisfy $\frac{\pi r^2}{\text{area}(ABC)} \leq \frac{\pi}{3\sqrt{3}}$, with equality for an equilateral triangle, we have $m = 2 \text{area}(ABC) \geq 6\sqrt{3}r^2 > 6\sqrt{3}(\tau_+^2 - \delta_+^2)$.

### 11.5.5.2 Bound on $\hat{a}$, $\hat{b}$, and $\hat{c}$

Let $s = P' - P$ and $V = |\text{vol}(ABP'P)| = \frac{1}{6}||((P' - A) \times (P - A)) \cdot (B - A)|| = \frac{1}{6}||(P' - P) \times (P - A)) \cdot (B - A)|| = \frac{1}{6}||(s \times w) \cdot u||$. Since $s = \pm \frac{d}{m}r$, $|\hat{c}| = | - n \cdot u| = |(r \times w) \cdot u| = \frac{m}{d}||(s \times w) \cdot u|| = \frac{6m}{d}V = \frac{2m\ell(PP')}{\text{area}(ABP')} = 2m \text{area}(ABP')$. The bound $\text{area}(ABP') > \tau_+^2 - \delta_+^2$ is obtained from $\min(\ell(P'R), \ell(P'S), \ell(P'T)) > \sqrt{\tau_+^2 - \delta_+^2}$ as in the 2D case. Finally, $|\hat{c}| > 2m(\tau_+^2 - \delta_+^2) > 12\sqrt{3}(\tau_+^2 - \delta_+^2)^2$.

### 11.5.5.3 Algorithm

Pseudocode for the face-vertex case is shown in Figure 17. This algorithm and its correctness follows the edge-edge case rather closely. Line 9 bounds $m^2$ from zero using the fact that $m^2 > \nu_+ = (6\sqrt{3}(\tau_+^2 - \delta_+^2))^2 = 108(\tau_+^2 - \delta_+^2)^2$ whenever an intersection should be registered. In particular, if the comparison on line 9 succeeds and the algorithm terminates, then we know that no intersection should have been registered. This test allows the test on line 14 to be reliable. The correctness of the test on line 14 will follow from choosing $\delta_-$ and $\delta_+$ properly.

What remains is the check on the signs of $\hat{a}$, $\hat{b}$, and $\hat{c}$ on line 21. Following the logic of the edge-edge case, we use tests of the form $\hat{a} \leq \zeta$, where $\zeta_+ = 12\sqrt{3}(\tau_+^2 - \delta_+^2)^2$, since this covers both the sign and magnitude checks on $\hat{a}$ that would otherwise be required. The roundoff error analysis for $\hat{d}^2 \leq \delta^2 m^2$ is identical to the edge-edge case and is omitted here.
Floating point: \( \mathcal{E}[m^2] \leq 129\epsilon L^4 \). \( \mathcal{E}[\hat{d}] \leq 47\epsilon L^3 \). \( \mathcal{E}[\hat{a}] \leq 445\epsilon L^4 \). \( \mathcal{E}[\hat{b}] \leq 129\epsilon L^4 \). \( \mathcal{E}[\hat{c}] \leq 129\epsilon L^4 \).

Tolerance constraints:

- \( \nu_+ = 108(\tau_+^2 - \delta_+^2)^2 \)
- \( \zeta_+ = 12\sqrt{3}(\tau_-^2 - \delta_-^2)^2 \)
- \( m^2 \leq \nu \) with \( \mathcal{E}[m^2] \leq 129\epsilon L^4 \) leads to \( \min(\nu_+ - \nu, \nu - \nu_-) > 130\epsilon L^4 \)
- \( \hat{a} \leq \zeta \) with \( \mathcal{E}[\hat{a}] \leq 445\epsilon L^4 \) leads to \( \min(\zeta_+ - \zeta, \zeta - \zeta_-) > 446\epsilon L^4 \).
- \( \hat{b} \leq \zeta \) with \( \mathcal{E}[\hat{b}] \leq 129\epsilon L^4 \) leads to \( \min(\zeta_+ - \zeta, \zeta - \zeta_-) > 130\epsilon L^4 \).
- \( \hat{c} \leq \zeta \) with \( \mathcal{E}[\hat{c}] \leq 129\epsilon L^4 \) leads to \( \min(\zeta_+ - \zeta, \zeta - \zeta_-) > 130\epsilon L^4 \).
- \( \hat{d}^2 \leq \delta^2 m^2 \), identical to the case addressed in Section 11.5.4.4, since \( \gamma_- = \delta_- \), \( \gamma = \delta \), \( \gamma_+ = \delta_+ \), and the values \( \mathcal{E}[m^2] \) and \( \mathcal{E}[\hat{d}] \) are the same.

11.5.6 Triangle-Edge

Let \( ABC \) and \( PQ \) be a triangle and an edge and

\[
\begin{align*}
v_A &= 6 \text{vol}(BCPQ) \quad v_B = 6 \text{vol}(CAPQ) \quad v_C = 6 \text{vol}(ABPQ) \quad (11.17) \\
v_P &= 6 \text{vol}(ABCP) \quad v_Q = 6 \text{vol}(ABCQ). \quad (11.18)
\end{align*}
\]

The triangle intersection barycentric coordinates are given by

\[
\begin{align*}
\gamma_A &= \frac{v_A}{v_A + v_B + v_C} \quad \gamma_B &= \frac{v_B}{v_A + v_B + v_C} \quad \gamma_C &= \frac{v_C}{v_A + v_B + v_C} \\
\alpha_P &= \frac{v_P}{v_P - v_Q} \quad \alpha_Q &= \frac{v_Q}{v_P - v_Q}. \quad (11.19)
\end{align*}
\]

The intersection criteria are that \( \alpha_P \) and \( \alpha_Q \) differ in sign but that \( \gamma_A \), \( \gamma_B \), and \( \gamma_C \) agree in sign. If these conditions are met, we show the above computations to be robust.
(a) Area bound for triangle-edge.

Figure 11.10: Proof illustrations for triangle edge in 3D.

Be begin with a bound on $|v_p|$ and $|v_Q|$. Consider the case shown in Figure 11.10(a), where $P$ projects to plane $ABC$ at a point $P$ in triangle $ABC$ and $Q$ projects to a point $Q'$ outside triangle $ABC$. Then, $\ell(PP') > \gamma_-$ and $\ell(QQ') \geq \ell(SR) > \gamma_-$. Thus, we see that, in general, $\ell(PP') > \min(\gamma_-, \delta_-)$. $|v_p| = 6 \left| \text{vol}(ABC) \right| = 2 \text{area}(ABC) \ell(PP')$.

Note that $\min(\ell(DT), \ell(DU), \ell(DV)) > \gamma_-$, so that the case of bounding $\text{area}(ABC)$ is analogous to that considered in Section 11.5.5.1 and leads to $2 \text{area}(ABC) > 6\sqrt{3}\gamma_-^2$. Finally, $|v_p| = 2 \text{area}(ABC) \ell(PP') > 6\sqrt{3}\gamma_-^2 \min(\gamma_-, \delta_-) = \xi_+$.

Next, we consider a bound on $|v_A|$, $|v_B|$, and $|v_C|$. Here, $|v_A| = 6 \left| \text{vol}(BCPQ) \right| = 6 \left| \text{vol}(BCPD) \right| + 6 \left| \text{vol}(BCDQ) \right| = 2 \text{area}(DBC) (\ell(PP') + \ell(QQ'))$. Bounding $\text{area}(DBC)$ is analogous to the proof in Section 11.4.5 using $\min(\ell(DT), \ell(DU), \ell(DV)) > \gamma_-$, leading to $2 \text{area}(DBC) > 2\gamma_-^2$. Finally, $|v_A| = 2 \text{area}(DBC)(\ell(PP') + \ell(QQ')) > 4\gamma_-^2 \min(\gamma_-, \delta_-) = \mu_+$.

The logic for Algorithm 18 is very similar to that of Algorithm 11 and Algorithm 12, where we perform separate sign and magnitude comparisons.

**Floating point:** $\mathcal{E}[v_A] \leq 47\epsilon L^3$. $\mathcal{E}[v_B] \leq 47\epsilon L^3$. $\mathcal{E}[v_C] \leq 47\epsilon L^3$. $\mathcal{E}[v_P] \leq 47\epsilon L^3$. $\mathcal{E}[v_Q] \leq 47\epsilon L^3$. 

92
Algorithm 18 Triangle Edge 3D

1: function TRIANGLE_EDGE(A, B, C, P, Q)
2:     if Degenerate intersection then
3:         return (FALSE, 0, 0, 0, 0)
4:     end if
5:     \( v_A \leftarrow ((B - Q) \times (C - Q)) \cdot (P - Q) \)
6:     \( v_B \leftarrow ((C - Q) \times (A - Q)) \cdot (P - Q) \)
7:     \( v_C \leftarrow ((A - Q) \times (B - Q)) \cdot (P - Q) \)
8:     if |v_A| \leq \mu \text{ or } |v_B| \leq \mu \text{ or } |v_C| \leq \mu \text{ or } \text{sgn}(v_A) \neq \text{sgn}(v_B) \text{ or } \text{sgn}(v_B) \neq \text{sgn}(v_C) \)
9:         return (FALSE, 0, 0, 0, 0)
10:    end if
11:     \( v_P \leftarrow ((A - P) \times (B - P)) \cdot (C - P) \)
12:     \( v_Q \leftarrow ((A - Q) \times (B - Q)) \cdot (C - Q) \)
13:     if |v_P| \leq \xi \text{ or } |v_Q| \leq \xi \text{ or } \text{sgn}(v_P) = \text{sgn}(v_Q) \) then
14:         return (FALSE, 0, 0, 0, 0)
15:     end if
16: return (TRUE, \( \frac{v_A}{v_A + v_B + v_C}, \frac{v_B}{v_A + v_B + v_C}, \frac{v_C}{v_A + v_B + v_C}, \frac{v_P}{v_P - v_Q} \))
Tolerance constraints:

- \( \mu_+ = 4\gamma_2^2 \min(\gamma_-, \delta_-) \)
- \( \xi_+ = 6\sqrt{3}\gamma_2^2 \min(\gamma_-, \delta_-) \)
- \( v_A > 0 \), enforced by guaranteeing \( |v_A| > \mu_- \) and requiring \( \mu_- > \mathcal{E}[v_A] \geq 47\epsilon L^3 \).
- \( v_P > 0 \), enforced by guaranteeing \( |v_P| > \xi_- \) and requiring \( \xi_- > \mathcal{E}[v_P] \geq 47\epsilon L^3 \).
- \( \min(\mu_+ - \mu, \mu - \mu_-) > 48\epsilon L^3 \).
- \( \min(\xi_+ - \xi, \xi - \xi_-) > 48\epsilon L^3 \).
- Similar bounds for \( v_B, v_C, \) and \( v_Q \)

11.5.7 Tetrahedron-Vertex

Let \( ABCD \) and \( P \) be a tetrahedron and a vertex. Let

\[
\begin{align*}
  v_A &= 6 \text{vol}(PBCD) & v_B &= 6 \text{vol}(APCD) \\
  v_C &= 6 \text{vol}(ABPD) & v_D &= 6 \text{vol}(ABCP)
\end{align*}
\]  

(11.20)

(11.21)

An intersection occurs if all have the same sign. Since these volumes all have the same sign, the barycentric weights are robustly computed as

\[
\begin{align*}
  \gamma_A &= \frac{v_A}{v_A + v_B + v_C + v_D} \\
  \gamma_B &= \frac{v_B}{v_A + v_B + v_C + v_D} \\
  \gamma_C &= \frac{v_C}{v_A + v_B + v_C + v_D} \\
  \gamma_D &= \frac{v_D}{v_A + v_B + v_C + v_D}
\end{align*}
\]  

(11.22)

(11.23)

If an intersection exists and no degeneracy is registered, then \( |v_A|, |v_B|, |v_C|, \) and \( |v_D| \) can be bounded. We begin by bounding the distances from \( P \) to the face planes of the tetrahedron. In Figure 11.11(a), point \( P \) projects to point \( J \) in triangle \( ABD \) but a point \( K \) on plane \( ABC \) outside triangle \( ABC \). The degeneracy assumption immediately gives \( \ell(PJ) > \delta_- \). For the more difficult case, \( PK \) must intersect one of the other faces of the
tetrahedron at \( L \), so that \( \ell(PK) \geq \ell(PL) > \delta_\ast \). Thus, the distance from \( P \) to any of the tetrahedron’s bounding planes is larger that \( \delta_\ast \).

Next consider the setup in Figure 11.11(b), where \( DP \) intersects \( ABC \) at \( R \). It can be seen that the distance \( \ell(RU) \) from \( R \) to an edge of triangle \( ABC \) can be bounded as \( \ell(RU) \geq \ell(\text{PU'}) > \delta_\ast \), since \( P \) is bounded away from plane \( ABD \). Similarly, \( \ell(RS) > \delta_\ast \) and \( \ell(RT) > \delta_\ast \). Using the same logic as Section 11.5.5.1, we conclude \( 2 \text{area}(ABC) > 6\sqrt{3}\delta_\ast^2 \).

If \( P \) projects to \( K \) in the plane of \( ABC \), then \( |v_D| = 6 |\text{vol}(ABCP)| = 2 \ell(PK)\text{area}(ABC) > 6\sqrt{3}\delta_\ast^3 \). The same bound is obtained for the other volumes. Thus, \( \min(|v_A|, |v_B|, |v_C|, |v_D|) \geq 6\sqrt{3}\delta_\ast^3 = \rho_+ \).

The logic for Algorithm 19 is very similar to that of Algorithm 18.

<table>
<thead>
<tr>
<th>Algorithm 19 Tetrahedron Vertex 3D</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1:</strong> function <code>Tetrahedron_Vertex(A, B, C, D, P)</code></td>
</tr>
<tr>
<td><strong>2:</strong> if Degenerate intersection then</td>
</tr>
<tr>
<td><strong>3:</strong> return (false, 0, 0, 0)</td>
</tr>
<tr>
<td><strong>4:</strong> ( v_A \leftarrow ((B - P) \times (C - P)) \cdot (D - P) )</td>
</tr>
<tr>
<td><strong>5:</strong> ( v_B \leftarrow ((P - A) \times (C - A)) \cdot (D - A) )</td>
</tr>
<tr>
<td><strong>6:</strong> ( v_C \leftarrow ((B - A) \times (P - A)) \cdot (D - A) )</td>
</tr>
<tr>
<td><strong>7:</strong> ( v_D \leftarrow ((B - A) \times (C - A)) \cdot (P - A) )</td>
</tr>
<tr>
<td><strong>8:</strong> if (</td>
</tr>
<tr>
<td><strong>9:</strong> return (false, 0, 0, 0)</td>
</tr>
<tr>
<td><strong>10:</strong> if ( \text{sgn}(v_A) \neq \text{sgn}(v_B) \text{ or } \text{sgn}(v_B) \neq \text{sgn}(v_C) \text{ or } \text{sgn}(v_C) \neq \text{sgn}(v_D) ) then</td>
</tr>
<tr>
<td><strong>11:</strong> return (false, 0, 0, 0)</td>
</tr>
<tr>
<td><strong>12:</strong> return (true, ( \frac{v_A}{v_A + v_B + v_C + v_D}, \frac{v_B}{v_A + v_B + v_C + v_D}, \frac{v_C}{v_A + v_B + v_C + v_D}, \frac{v_D}{v_A + v_B + v_C + v_D} ))</td>
</tr>
<tr>
<td><strong>13:</strong> end function</td>
</tr>
</tbody>
</table>

Floating point: \( \mathcal{E}[v_A] \leq 47\epsilon L^3, \mathcal{E}[v_B] \leq 47\epsilon L^3, \mathcal{E}[v_C] \leq 47\epsilon L^3, \mathcal{E}[v_D] \leq 47\epsilon L^3. \)

Tolerance constraints:
\[ \rho_+ = 6\sqrt{3}\delta_3 \]

- \( v_A > 0 \), enforced by guaranteeing \( |v_A| > \rho_- \) and requiring \( \rho_- > \mathcal{E}[v_A] \geq 47\epsilon L^3 \).

- \( \min(\rho_+ - \rho, \rho - \rho_-) > 48\epsilon L^3 \).

- Similar bounds for \( v_B, v_C, \) and \( v_D \)
CHAPTER 12

Examples

In Figure 12.1, a stretched armadillo is first cut incrementally along an S-shaped path before being diced into pieces. In Figure 12.2, we peel the skin off a sphere, demonstrating the ability to generate thin slices, even while cutting existing cuts. In Figure 12.3, use shaped blades to cut letters into a stretched thin sheet. The sheet fractures and deforms as the blades pass through it.

Since our algorithm cuts tetrahedralized volumes using an arbitrary cutting surface, we can use to perform boolean operations. In Figure 12.4, we cut a 20k-element cow surface from a 208 × 128 × 68 box. Our cutting algorithm accurately resolves the delicate geometry of the cow surface, including its horn, ear and tail. The cow volume is cut again by a bunny surface mesh, resulting in two volumes.
Figure 12.1: An armadillo mesh with 380K tetrahedra is progressively cut with an “S” and then cut into three pieces.

Figure 12.2: An apple is peeled demonstrating the ability to do incremental cuts and produce thin geometry, shown from the front (left) and the side (right). The skin thickness is 1/60 of the diameter of the ball.
Figure 12.3: We cut letters out of stretched elastic sheet, demonstrating the ability to mix cutting, re-cutting, and simulation.
Figure 12.4: We demonstrate the ability to construct tetrahedralized volumes from triangulated surfaces using our cutting algorithm. We cut a box by a cow surface and then cut it again with a bunny surface, resulting in two separate volumes. Details are accurately resolved in both volumes, and the corners are sharp.
CHAPTER 13

Discussions

While our approach addresses a number of limitations of the existing VNA, it is not without limitations. First, while our adaptive definition of embedded material does improve the ability to resolve cut surfaces at resolutions independent of the embedding mesh, it does so with a considerable algorithmic complexity compared to the original VNA. However, this is still significantly lower than the approach in [92]. Also, without this adaptive routine the algorithm would have the same complexity of the original VNA while adding the ability to pass through nodes, faces and edges. Second, safe tolerances for the mesh intersection routines are considerably larger in 3D than in 2D, which effectively prevents the use of floats for 3D intersections calculations.
References


