UNIVERSITY OF CALIFORNIA SAN DIEGO

Higher-order polygonal finite element analysis of nearly-incompressible isotropic elastic materials

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

in

Structural Engineering

by

Poorya Mirkhosravi

Committee in charge:

Professor Petr Krysl, Chair
Professor Michael Holst
Professor Hyonny Kim
Professor Melvin Leok
Professor Qiang Zhu

2018

DEDICATION

To my parents, Abbas and Maryam and to my brother, Reza

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

First and foremost, I would like to express my deepest gratitude to my advisor, Professor Petr Krysl for his support, patience, serenity of character, and for letting me explore my different research interests.

I would like to thank Professors Melvin Leok, Michael Holst, Qiang Zhu, and Hyonny Kim for serving on my committee. Specially Professors Leok and Holst for our numerous discussions which were tremendously helpful.

My special thanks goes to my fellow graduate students and post-docs who shared an office with me during my stay, Luis Monterrubio Salazar, Stephen Oberrecht, Jerlie Small, Ivana Escobar, Alireza Pakravan, and Phi Nguyen. To Stephen Oberrecht in particular, for his true friendship and for our countless talks on science and life.

And finally, I would like to sincerely thank my parents, Abbas and Maryam, and my brother, Reza, for their unwavering love and support during all these years.

Chapter 1, in part is currently being prepared for submission for publication of the material. Mirkhosravi, Poorya; Krysl, Petr. The dissertation author was the primary investigator and author of this material.

Chapter 2, in part is currently being prepared for submission for publication of the material. Mirkhosravi, Poorya; Krysl, Petr. The dissertation author was the primary investigator and author of this material.

Chapter 3, in part is currently being prepared for submission for publication of the material. Mirkhosravi, Poorya; Krysl, Petr. The dissertation author was the primary investigator and author of this material.

VITA

2006	B. S. in Civil Engineering, University of Tehran, Iran
2009	M. S. in Structural Engineering, Sharif University of Technology, Iran
2018	Ph. D. in Structural Engineering, University of California San Diego

ABSTRACT OF THE DISSERTATION

Higher-order polygonal finite element analysis of nearly-incompressible isotropic elastic materials

by

Poorya Mirkhosravi

Doctor of Philosophy in Structural Engineering

University of California San Diego, 2018

Professor Petr Krysl, Chair

In the first part of this thesis, we present a stable higher-order polygonal finite element method for modeling nearly-incompressible isotropic materials. Our method is based on applying the discontinuous Petrov-Galerkin methodology on a hybridized version of the ultraweak formulation of linear elasticity. As a result, the unknown degrees of freedom are defined only on the skeleton of the mesh (interface variables) and have a symmetric positive-definite coefficient matrix. The performance and convergence of the method is demonstrated with numerical examples.

In the second part of the thesis, we present a heuristic algorithm that generates coarsened non-uniform hexahedral meshes with higher resolution close to selected regions in the domain of 3D micro-CT and micro-MR images. Applying a coarsening step on areas of the problem domain in order to reduce the computational cost is inevitable; however, on the other hand, it is desirable to have a fine mesh in regions containing small vital features. This algorithm takes as input a very fine micro-CT data set, the location of the regions containing delicate geometrical details, and the coarsening factor and produces a ready-to-use graded mesh.

Chapter 1

Introduction

1.1 Motivation

This work is composed of two main parts. Both topics are motivated by the efforts in Professor Krysl's research group at UC San Diego in recent years to simulate the auditory system of marine mammals [CKH08, CKA10, CMS$^+$08]. Most of the tissues and cartilages modeled in these simulations are biological materials that can be modeled as nearly-incompressible. The finite element method has been used for mechanical and multi-physics simulation for several decades now. However, it is a well-known fact that classical displacement-based finite element methods do not perform well in nearly and totally incompressible regimes and experience the so-called locking problem. Therefore, developing a robust and stable method for analysis of nearly-incompressible elastic materials in the context of general polygonal meshes is the focus of the first part (chapter 2) of this dissertation. In addition to *stability*, the *accuracy* of the numerical method is also a concern. The mesh that is used to represent the geometry of the model is one of the factors affecting accuracy. Thus, the second part (chapter 3) is concerned with developing an algorithm to automatically create non-uniform hexahedral meshes from the CT-scan images of the tympanoperiotic complex of marine mammals. The meshes are used by other researchers to do vibrational analysis and compute the natural frequencies of the tympanoperiotic complex in marine mammals [KTC12].

1.2 Polygonal finite elements

The mesh generation step takes a big portion of the analysis time in real-world applications and using polygonal and polyhedral elements can make the mesh generation process faster. It may even reduce the number of required degrees of freedom in some examples. The other area that benefits from polytopal elements is the mesh refinement and coarsening because we can naturally get rid of hanging nodes. In recent years, there has been much effort made to use polytopal elements in different areas of solid mechanics like modeling fracture and polycrystalline materials. These elements have also been used in computational fluid mechanics problems and even in topology optimization.

1.2.1 Equations of linear elasticity

We will use polygonal elements to solve nearly-incompressible linear elasticity problems. The equations of linear elasticity read as follows:

$$\begin{aligned} -\mathrm{div}(C\colon (\nabla u + (\nabla u)^T)/2) = f & \qquad \text{in } \Omega \\ u = \bar{u} & \qquad \text{on } \Gamma_u \\ (C\colon (\nabla u + (\nabla u)^T)/2) \cdot n = \bar{t} & \qquad \text{on } \Gamma_t \end{aligned} \tag{1.1}$$

where u is the displacement vector and f is the vector of body force per unit volume defined over the two-dimensional region Ω with the boundary $\partial\Omega$. Γ_u and Γ_t are open subsets of the boundary and they satisfy $\Gamma_u \cap \Gamma_t = \varnothing$ and $\overline{\Gamma_u \cup \Gamma_t} = \partial\Omega$. Γ_u is the part of the boundary where the value of the displacement is prescribed while Γ_t is the region where we have a condition on the gradient of the displacement. n is the unit outward normal vector along Γ_t on the boundary. C is the stiffness tensor. Equation 1.1 is usually referred to as *the strong form* of the linear elasticity equation. *The (primal) weak form* is obtained by multiplying the first equation in 1.1 by a test function and integrating over Ω and applying integration by parts to shift the derivative from u to the test function. The primal weak form has been the starting point for classical finite element solutions of elasticity equations. The primal weak form of equation 1.1 can be written as [Hug87]

$$\begin{aligned}&\text{Find } u \in \mathcal{S} \text{ such that for all } v \in \mathcal{V},\\ &\int_\Omega ((\nabla u + (\nabla u)^T)/2) : \boldsymbol{C} : ((\nabla v + (\nabla v)^T)/2)\, \mathrm{d}\Omega = \int_\Omega f \cdot v \, \mathrm{d}\Omega + \int_{\Gamma_t} \bar{t} \cdot v \, \mathrm{d}\Gamma,\end{aligned} \tag{1.2}$$

where $\mathcal{S}$ and $\mathcal{V}$ denote the trial and test spaces, respectively. The trial space, $\mathcal{S}$, consists of functions in $\boldsymbol{H}^1(\Omega)$ which have a value of $\bar{u}$ on Γ_u. Similarly, the test space, $\mathcal{V}$, consists of functions in $\boldsymbol{H}^1(\Omega)$ which have a value of zero on Γ_u. So, the only space that we need to discretize is $\boldsymbol{H}^1(\Omega)$. $\boldsymbol{H}^1(\Omega)$ is the vector version of $H^1(\Omega)$ which is the set of square-integrable functions with the additional property that the square integral of the gradient is also bounded. The popular choice in the FEM community to discretize $H^1(\Omega)$ is to use the Lagrange basis functions for triangles or the tensor product of Lagrange basis functions in the case of quadrilaterals. One way to represent the Lagrange basis functions on triangles is in terms of barycentric coordinates. So, finding barycentric coordinates for polygons, the so-called *generalized barycentric coordinates* was the natural path to take in order to solve 1.2 on polygonal meshes. In the following, we briefly review some of the previous efforts in developing different barycentric coordinates on polygons.

1.3 Generalized barycentric coordinates (GBCs)

For a convex polygon P shown in figure 1.1, the n vertices are denoted by v_i, for $i = 1 \ldots n$. The generalized barycentric coordinates on this polygon are the n functions ϕ_i which satisfy the following two properties:

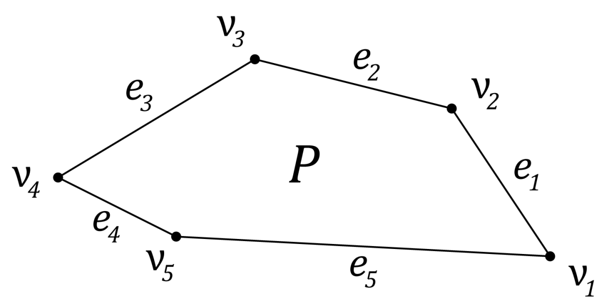


Figure 1.1: The order of vertices

- partition of unity property:

$$\sum_{i=1}^{n} \phi_i(x) = 1$$

- linear precision:

  For any linear function $f(x)$ on P, we have

$$f(x) = \sum_{i=1}^{n} f(v_i)\phi_i(x) \quad \text{for } x \in P$$

  In addition, there are other properties which are desirable to have:

- being non-negative:

$$\phi_i(x) \geq 0 \quad \text{for } i = 1 \ldots n$$

- Kronecker-delta property:

$$\phi_i(v_j) = \delta_{ij} \quad \text{for } i, j = 1 \ldots n$$

  where δ_{ij} is the Kronecker delta function. $\delta_{ij} = 1$ if i is equal to j. Otherwise, it is equal to zero.

- linearity on the edges:

$\phi_i(x)$ is a piecewise linear function on the boundary of the polygon, ∂P for $i = 1, \dots n$.

- smoothness within the element:

$$\phi_i(x) \in C^\infty(P) \quad \text{for } i = 1 \dots n$$

Some of the generalized barycentric coordinates in the literature can be written in a general format in terms of weight functions, $\omega_i(x)$. So, for an n-sided polygon, the GBCs, $\phi_1, \phi_2, \dots, \phi_n$, are defined as

$$\phi_i(x) = \frac{\omega_i(x)}{\sum_{j=1}^{n} \omega_j(x)} \qquad \text{for } i = 1, 2, \dots, n. \tag{1.3}$$

Other GBCs can be totally computational without a closed-form formula. For example, the maximum entropy coordinates in section 1.3.5 are obtained by numerically solving an optimization problem.

1.3.1 Wachspress coordinates

The first generalized barycentric coordinate (GBC) was proposed for convex polygons by Eugene Wachspress in 1971 using the ideas of projective geometry [Wac71, Wac75]. Warren [War96] extended the definition to higher dimensions (convex polytopes) in 1996. The explicit form of Wachspress coordinates are in terms of rational polynomials. As an example, these basis functions are computed for a pentagon with the vertex coordinates given in table A.1 in Appendix A. The final result is shown in figure 1.2.

Later, a more practical *local* formulation of Wachspress functions was proposed by Meyer et al. in 2002, [MBLD02]. In this formulation you only need to know the coordinates of vertices v_{j-1}, v_j, and v_{j+1} to find the weighting function ω_j and the Wachspress functions are calculated

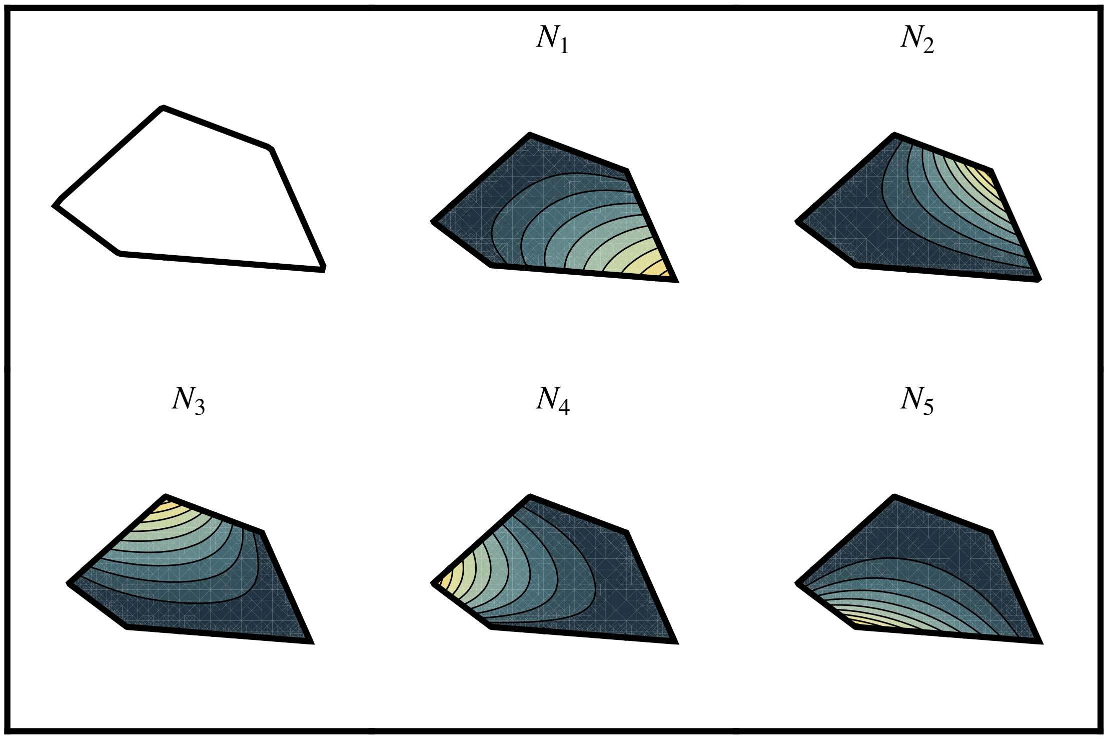


Figure 1.2: Wachspress basis functions

using equation 1.3. ω_j is defined as

$$\omega_j(x) = \frac{\cot\gamma_j + \cot\delta_j}{\|x - v_j\|^2} \qquad \text{for } j = 1, 2, \ldots, n.$$

where γ_j is the angle between the edges $\overline{x\, v_j}$ and $\overline{v_j\, v_{j-1}}$ and δ_j is the angle between the edges $\overline{x\, v_j}$ and $\overline{v_j\, v_{j+1}}$ (see figure 1.3).

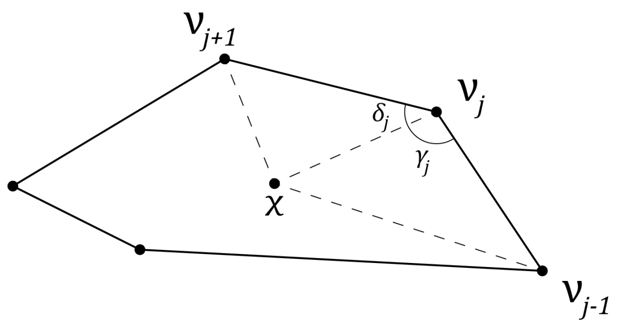


Figure 1.3: Definition of angles γ_j and δ_j

Alternatively, Floater et al. [FGS14] wrote the Wachspress weighting functions in terms of scaled vectors normal to the edges to avoid evaluating trigonometric functions,

$$\omega_j(x) = |p_{j-1} \times p_j| \qquad \text{for } j = 1, 2, \ldots, n\,.$$

where $p_j = \frac{n_j}{h_j}$ and $h_j = (v_j - x) \cdot n_j$ (see figure 1.4). This formulation is adopted in our work to develop $H(div)$ discretizations for polygons later in this chapter.

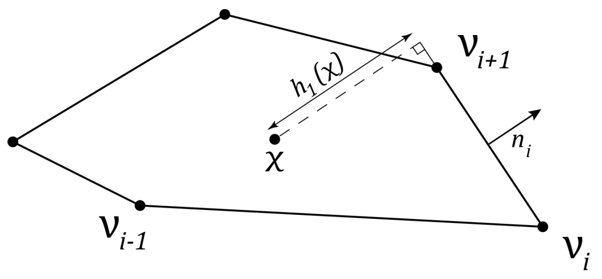


Figure 1.4: Definition of h_j

1.3.2 Metric coordinates

Metric coordinates were introduced by Malsch and Dasgupta [MD04a, MD04b] as a way to allow interior nodes inside convex polygons. But, one side effect is that the coordinates are no longer necessarily positive. The formulation is summarized in [HF06]. For an n-sided polygon, the metric coordinates, $\phi_1, \phi_2, \ldots, \phi_n$, can be written in the general form of equation 1.3 where

$$\omega_i(x) = b_{i-1}(x)A_{i-2}(x) - b_i(x)B_i(x) + b_{i+1}(x)A_{i+1}(x)\,.$$

$A_i(x)$ and $B_i(x)$ denote the areas of triangles $[x, v_i, v_{i+1}]$ and $[x, v_{i-1}, v_{i+1}]$ respectively as shown in figure 1.5.

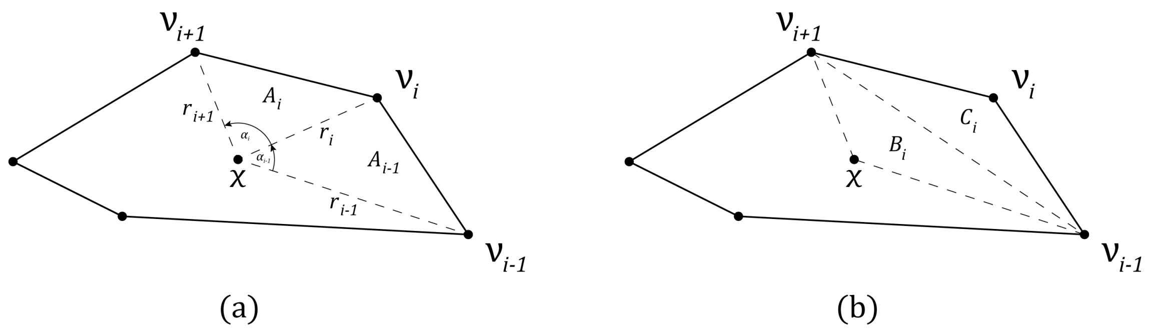


Figure 1.5: Definitions of A_i and B_i

$b_i(x)$ is defined as following

$$b_i(x) = \frac{1}{C_i\, q_{i-1}(x)\, q_i(x)}\,,$$

where C_i is the area of triangle $[v_{i-1}, v_i, v_{i+1}]$ (figure 1.5) and $q_i(x) = r_i(x) + r_{i+1}(x) - e_i$, with

$$r_i(x) = \|v_i - x\|$$
$$e_i = \|v_{i+1} - v_i\|\,.$$

1.3.3 Mean value coordinates

Mean value coordinates (MVC) were proposed by Floater in 2003 as a well-defined GBC on star-shaped polygons [Flo03] motivated by the Mean Value Theorem for harmonic functions. It was later generalized to three dimensions in 2005 [FKR05]. This method also follows the general form of equation 1.3. The weighting functions are defined as

$$\omega_i(x) = \frac{\tan\left(\frac{\alpha_{i-1}}{2}\right) + \tan\left(\frac{\alpha_i}{2}\right)}{\|v_i - x\|}\,,$$
$$\alpha_i = \alpha_i(x)\,, \quad 0 < \alpha_i < \pi$$

where α_i is defined in figure 1.5.

1.3.4 Natural neighbor-based coordinates

Another category of GBCs are the natural neighbor-based coordinates which are defined using the Voronoi diagram. The Sibson coordinates [Sib80] (also called the natural element coordinates) and the Laplace coordinates are two examples of natural neighbor-based coordinates.

The Sibson coordinate can be written in the general form of equation 1.3. The weighting function, ω_i, at any point, x, is defined by overlaying two Voronoi diagrams; the Voronoi diagram of the original vertices (see part (a) in figure 1.6) and the Voronoi diagram of the union of x with the original vertices (see part (b) in figure 1.6). $\omega_i(x)$ will be the area of the intersection of the Voronoi cell of vertex v_i in the first diagram and the Voronoi cell of vertex x in the second Voronoi diagram. For example, in figure 1.6, $\omega_2(x)$ is equal to the area of the region $ghdc$, A_{ghdc}. Finally, the Sibson basis function corresponding to v_2 is

$$\phi_2(x) = \frac{A_{ghdc}}{A_{edcba}}.$$

The details of implementation are described by Sukumar et al. in [SMB98].

1.3.5 Maximum entropy coordinates

The maximum entropy coordinates (MAXENT) were introduced by Sukumar in 2004 [Suk04] and was also developed independently by Arroyo and Ortiz in 2006 [AO06]. The idea is based on Jaynes' principle of maximum entropy [Jay57]. The unknown GBCs, $\phi_1, \phi_2, \ldots, \phi_n$, are determined by maximizing Shannon's entropy (the information-theoretic entropy), H, given as

$$H(\phi_1, \phi_2, \ldots, \phi_n) = -\sum_{i=1}^{n} \phi_i \log(\phi_i)$$

subject to the linear constraints $\sum_{i=1}^{n} \phi_i(x) = 1$ and $\sum_{i=1}^{n} \phi_i(x) v_i = x$. The method of Lagrange multipliers was used to solve this constrained optimization problem. Later in 2008, Hormann and

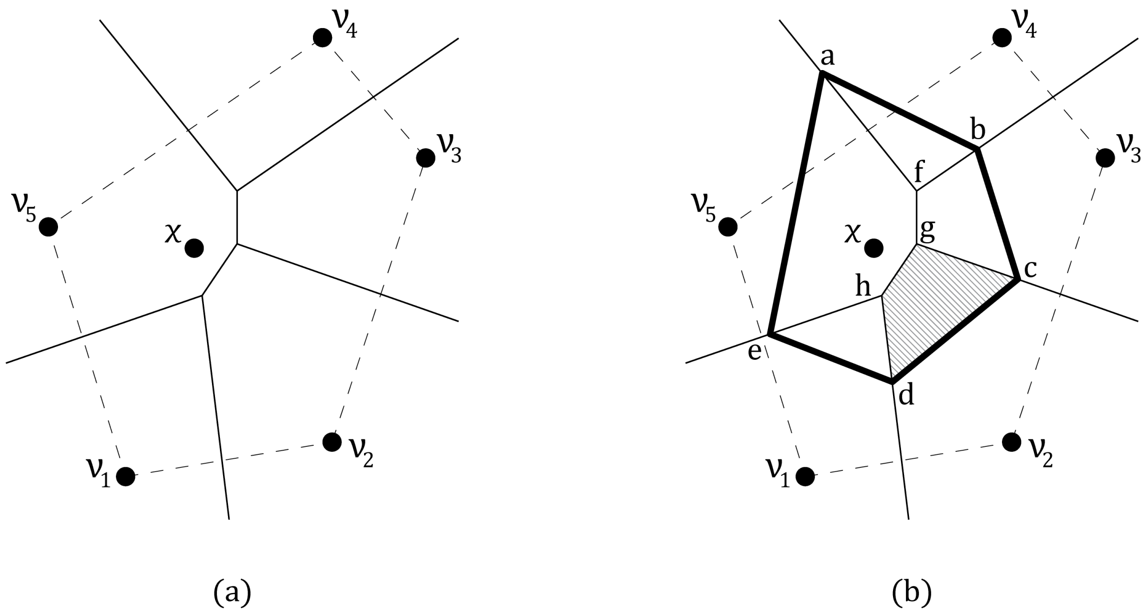


Figure 1.6: (a) Voronoi diagram of vertices $\{v_1, \ldots, v_5\}$ (b) Voronoi diagram of vertices $\{v_1, \ldots, v_5\} \cup \{x\}$

Sukumar enhanced the method by restoring the Kronecker-delta property on the boundary with the help of *Prior* functions [HS08].

1.4 Mixed finite element method

While the generalized barycentric coordinates make it possible for us to solve equation 1.2 on polygonal meshes, it does not lead to a stable method when the material is nearly incompressible. The root of the problem stems from the fact that norm of the elasticity modulus, C, goes to infinity as the Poisson's ratio, ν, approaches 0.5 in the near-incompressible regime (discussed in chapter 2). Using the so-called mixed formulation is one way to get rid of the stability problem in the incompressibility limit. This is achieved by defining more independent variables in addition to u in the strong form, equation 1.1 and deriving new weak forms accordingly. This family of methods is referred to as the mixed finite element method [BBF13]. Having multiple independent variables raises the question of how to discretize them. Unfortunately, using H^1 discretizations like the generalized barycentric coordinates on polygonal meshes for all the independent variables does not necessarily lead to a correct solution. The infinite-dimensional Hilbert spaces to which the independent variables in the mixed formulation belong matter and each space needs its own type of discretization. The common spaces are H^1, $H(\text{curl})$, $H(\text{div})$, and L^2 and their vector and matrix versions. Raviart and Thomas (RT) [RT77] and later Brezzi, Douglas, and Marini (BDM) [BDM85] proposed $H(\text{curl})$-conforming and $H(\text{div})$-conforming discretizations on meshes with triangular and square elements. Nedelec [Ned80, Ned86] generalized these discretizations to three dimensions for tetrahedral and cubic elements. To demonstrate the importance of these new discretizations the solution of the Poisson's equation, $-\nabla^2 u = f$, on $(0,1)\times(0,1)$ obtained with two different discretizations is shown in figure 1.7. The independent variables are u (the temperature) and σ (the flux). Both solutions approximate u using piecewise constants. The picture on the left shows u obtained when σ is approximated using continuous piecewise linears while the picture on the right is u when σ is approximated using Raviart-Thomas discretization. The solution on the right correctly captures the exact solution, $u_{ex} = x(1-x)y(1-y)$, while the one on the left is highly oscillatory and incorrect.

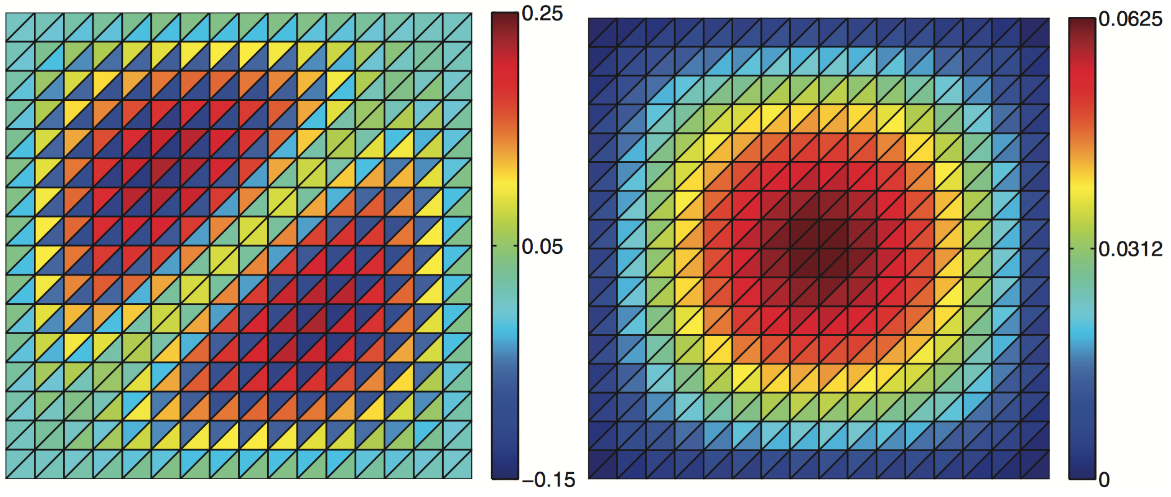


Figure 1.7: Numerical solution of the Poisson's problem. u is plotted in two different cases. Taken from [AFW10]

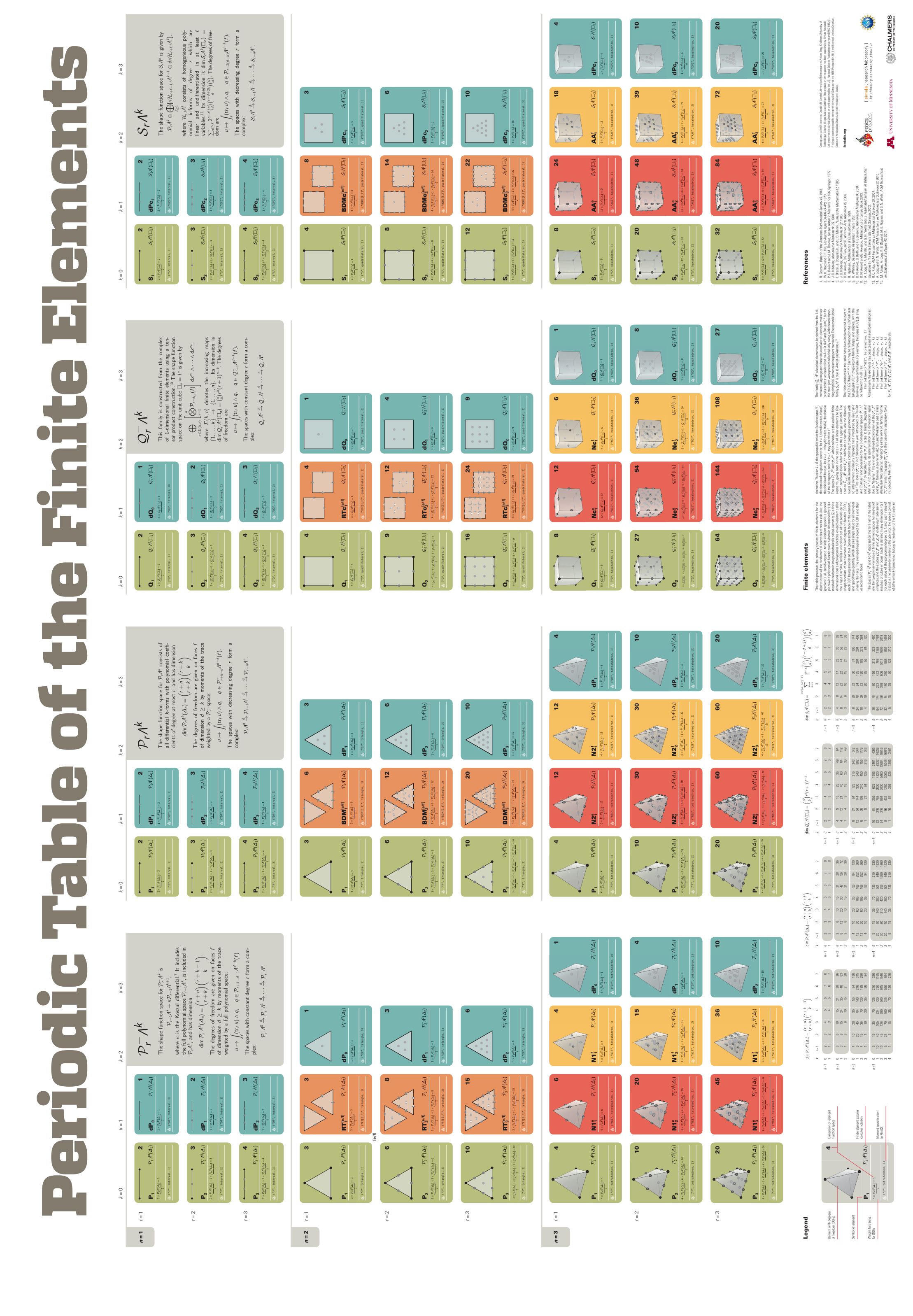
1.4.1 The Finite Element Exterior Calculus (FEEC) framework

In 2006, Arnold, Falk, and Winther [AFW06] used the language of differential forms to unify the mixed formulation discretizations into one framework called Finite Element Exterior Calculus (FEEC). In this framework, discretizations of the spaces in the de Rham complex are created using discrete differential forms [AFW09], especially the Whitney forms [Whi57], satisfying some properties like being a subcomplex and existence of bounded cochain projections between the original infinite-dimensional complex and the discrete subcomplex. Arnold et al. showed the equivalence between discrete differential forms of FEEC and the classical vector-valued spaces like RT and BDM through vector proxies and categorized these spaces into a *periodic table of finite elements* [AL14] (see figure 1.8).

1.4.2 Vector-valued GBC elements

The table in figure 1.8 does not list any polygonal elements and only contains triangular and square elements. One of the first efforts to address this issue was by Gillette et al. [GRB16]. They proposed ideas on how to utilize generalized barycentric coordinates to create basis functions resembling Whitney forms for polygons. Chen and Wang [CW17] extended this idea further and developed linear order $H(\text{curl})$- and $H(\text{div})$-conforming polygonal elements with smallest possible dimension. As an example, the $H(\text{div})$-conforming basis functions for the sample pentagon given in Appendix A is shown in figure 1.9. The Mathematica implementation of Chen and Wang approach is also listed in Appendix B.

We used these vector-valued basis functions to solve the mixed form of the Poisson's equation on the domain $\Omega = (0,1) \times (0,1)$ (see the mesh in figure 1.10) with homogeneous boundary condition, $u = 0$ on $\partial\Omega$. The mixed weak form for the problem $-\nabla^2 u = f$ reads as



Periodic Table of the Finite Elements
$\mathcal{P}_r^- \Lambda^k$
$\mathcal{P}_r \Lambda^k$
$\mathcal{Q}_r^- \Lambda^k$
$\mathcal{S}_r \Lambda^k$
$n = 1$
$n = 2$
$n = 3$
$k = 0$
$k = 1$
$k = 2$
$k = 3$
$r = 1$
$r = 2$
$r = 3$
Legend
Finite elements
References

Figure 1.8: The periodic table of finite elements. Taken from http://www.femtable.org

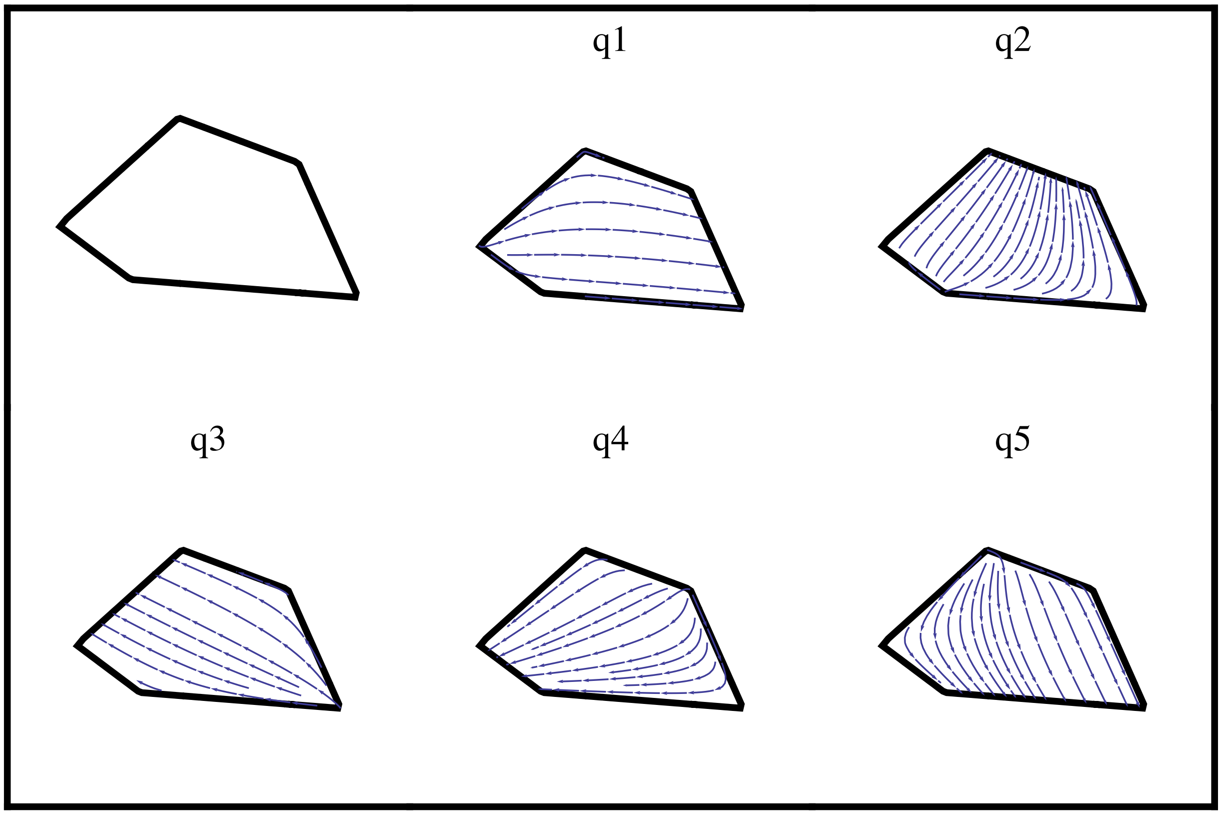


Figure 1.9: minimal degree $H(\text{div})$ basis functions

follows:

$$\begin{aligned}
&\text{Find } \sigma \in H(\text{div},\Omega),\ u \in L^2(\Omega) \\
&\int_\Omega \sigma\cdot\tau\,\mathrm{d}\Omega + \int_\Omega u\,\text{div}\tau\,\mathrm{d}\Omega = 0, \qquad \text{for all } \tau \in H(\text{div},\Omega), \\
&-\int_\Omega (\text{div}\sigma) v\,\mathrm{d}\Omega = \int_\Omega f v\,\mathrm{d}\Omega, \qquad \text{for all } v \in L^2(\Omega).
\end{aligned} \tag{1.4}$$

where the given right hand side is $f = 2\pi^2\sin(\pi x)\sin(\pi y)$. The exact solution for the given f is $u_{ex} = \sin(\pi x)\sin(\pi y)$ and $\sigma_{ex} = (\pi\cos(\pi x)\sin(\pi y), \pi\sin(\pi x)\cos(\pi y))$. The variables σ and τ are approximated using the $H(\text{div})$-conforming basis functions of Chen and Wang while piecewise constants are used to approximate u and v. The exact and computed solutions are shown in figures 1.11 and 1.12. It shows that using the combination of $H(\text{div})$-conforming and L^2-conforming subspaces leads to a correct numerical solution.

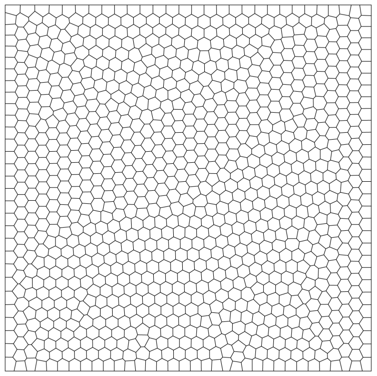
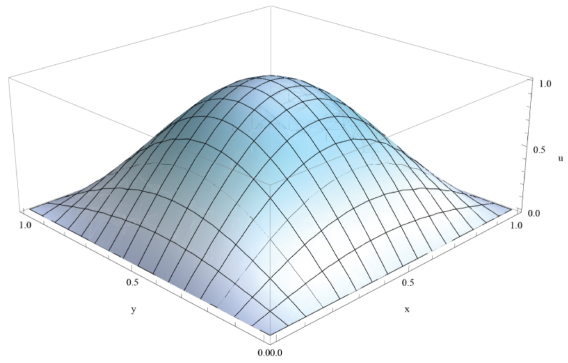
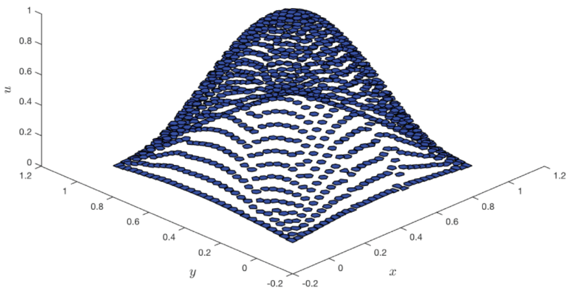


Figure 1.10: The polygonal mesh which was used to solve the mixed Poisson problem



(a)

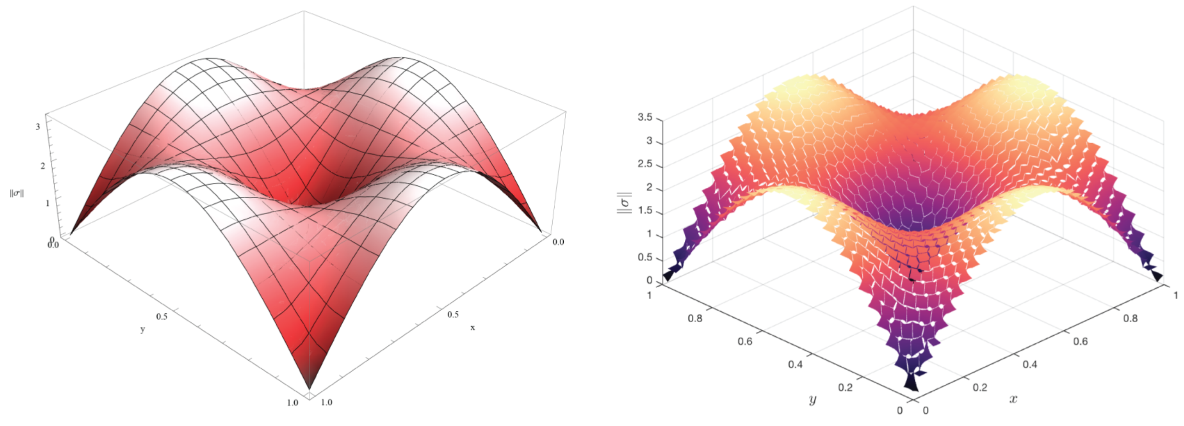


(b)

Figure 1.11: (a) $u_{ex} = \sin(\pi x)\sin(\pi y)$ (b) the mixed FEM solution for u

Next, we tried to solve the mixed formulation of the elasticity equation using Chen and Wang ($\mathcal{CW}$) basis functions. Two different scenarios were carried out. On the first attempt, a mixed weak form with weakly enforced symmetry for the stresses [AFW07] was used so that a Cartesian product of $\mathcal{CW}$ with itself can be used to approximate the stress matrix. The components of the displacement vector and the rotation matrix were approximated with piecewise constant functions. On the second attempt, the approximation of the components of the rotation matrix was changed to piecewise linears. Also, bubble functions defined in terms of GBCs were added to the previous approximation space for stresses as defined in the PEERS method of Arnold, Brezzi, and Douglas [ABD84].

While we were able to solve the Poisson's equation using $\mathcal{CW}$ basis functions, neither



(a) (b)

Figure 1.12: The norm of σ: (a) the exact solution (b) the mixed FEM solution

of the two attempts described above were successful in correctly solving the elasticity problem. This made us look at a method called discontinuous Petrov-Galerkin which was recently used [AFMD18] to solve the Poisson's equation on a polygonal mesh. In the next chapter, we extended this idea to solve linear elasticity problems on polygonal meshes.

Chapter 1, in part is currently being prepared for submission for publication of the material. Mirkhosravi, Poorya; Krysl, Petr. The dissertation author was the primary investigator and author of this material.

Chapter 2

The dPG framework

2.1 Introduction

The discontinuous Petrov-Galerkin framework was introduced in 2010 by Demkowicz and Gopalakrishnan [DG10, DG11a]. It can be interpreted as a minimum residual finite element method and it is usually applied on the ultraweak formulation [CD98] of the boundary value problem. dPG is a general method and can be applied to other variational formulations too as long as they are well-defined. Equivalently, the dPG can be interpreted as a Petrov-Galerkin method with an optimal test space [DG14]. We choose a finite-dimensional subspace of the trial space and the dPG will find the optimal test space associated to this trial space which involves inverting the Riesz operator. This is called the ideal dPG [DG11b] because the Riesz operator is infinite-dimensional and its inverse can only be computed in special cases. The practical dPG [GQ14] finds an approximate optimal test space by using an enriched discrete test space. This makes the Riesz operator finite-dimensional and its inverse can be computed by solving a global problem which is still computationally expensive. By using the broken ultraweak formulations [CDG16] the inversion of the Riesz operator becomes a local problem (at the element level). The method leads to symmetric and positive definite stiffness matrices and is even equipped with a natural residual-based error estimator. It is getting more attention over time and has been applied to different areas such as elasticity [KFD16, FKDLT17, BDGQ12], viscoelasticity [KKR+17], fluid problems [EDC14], electromagnetism [CDG16], wave propagation [PD17], and Schrödinger equation [DGNS17] to name a few. In this chapter, we first derive the ultraweak formulation and its broken version for the linear elasticity problem and then describe the dPG method in detail and derive the equations.

2.2 The variational formulation

2.2.1 The strong form

The equations of linear elasticity in the mixed form read

$$\begin{aligned} -\mathrm{div}\sigma &= f && \text{in } \Omega \\ \sigma &= C : \varepsilon && \text{in } \Omega \\ u &= \bar{u} && \text{on } \Gamma_u \\ \sigma \cdot n &= \bar{t} && \text{on } \Gamma_t \end{aligned} \tag{2.1}$$

where u and f are the displacement and body force vectors. σ and ε are the symmetric stress and strain tensors. For a two-dimensional problem, which is our main focus, these tensors take values in the space $\mathbb{S} := \mathbb{R}^{2\times 2}_{sym}$ over the two-dimensional region Ω with the boundary $\partial\Omega$. Γ_u and Γ_t are open subsets of the boundary and they satisfy $\Gamma_u \cap \Gamma_t = \varnothing$ and $\overline{\Gamma_u \cup \Gamma_t} = \partial\Omega$. On Γ_u the displacement field has a prescribed value of $\bar{u}$ and on Γ_t, the traction vector is specified as $\bar{t}$. n is the unit outward normal vector along Γ_t on the boundary and C is the elasticity tensor. Equation 2.1 is derived by writing the equation 1.1 as a first-order system.

2.2.2 The ultraweak formulation

Before deriving the weak form, we start by rewriting the second equation in 2.1 as $\varepsilon = S\colon \sigma$ where $S := C^{-1}$ is the 4-th order compliance tensor. In the limit of incompressibility ($\nu \to \frac{1}{2}$) the norm of C goes to infinity while the norm of S remains bounded. So, writing the constitutive equation in terms of S will result in robustness for nearly-incompressible materials. Also, since the small strain tensor is defined as $\varepsilon = \frac{1}{2}(\nabla u + (\nabla u)^T)$, the gradient of u can be written as $\nabla u = \varepsilon + \omega$ where the infinitesimal rotation tensor, ω is defined as

$$\omega := \frac{1}{2}(\nabla u - (\nabla u)^T). \tag{2.2}$$

Substituting $\varepsilon = \nabla u - \omega$ in $\varepsilon = S\colon \sigma$, we can rewrite 2.1 as

$$\begin{aligned} -\mathrm{div}\sigma &= f && \text{in } \Omega \\ \nabla u - \omega &= S\colon \sigma && \text{in } \Omega \\ u &= \bar{u} && \text{on } \Gamma_u \\ \sigma \cdot n &= \bar{t} && \text{on } \Gamma_t \end{aligned} \tag{2.3}$$

Now, as is common for all weak formulations, we start by multiplying the first two equations in 2.3 by test functions v (a vector function) and τ (a 2nd-order tensor function and not necessarily symmetric) respectively.

$$\begin{gathered} -\int_\Omega (\mathrm{div}\sigma)\cdot v\,\mathrm{d}\Omega = \int_\Omega f\cdot v\,\mathrm{d}\Omega \\ \int_\Omega \nabla u : \tau\,\mathrm{d}\Omega - \int_\Omega \omega : \tau\,\mathrm{d}\Omega = \int_\Omega \sigma : S : \tau\,\mathrm{d}\Omega \end{gathered} \tag{2.4}$$

In contrast to the mixed formulation used in the PEERS framework, in the ultraweak formulation the integration by parts is applied on both equations to shift the derivatives completely to the test functions. So, for the first integral in the first equation we have,

$$\begin{aligned}
\int_\Omega (\operatorname{div}\sigma)\cdot v\,\mathrm{d}\Omega &= \int_\Omega \sigma_{ij,j} v_i\,\mathrm{d}\Omega \\
&= \int_\Omega [(\sigma_{ij}v_i)_{,j} - \sigma_{ij}v_{i,j}]\,\mathrm{d}\Omega \\
&\downarrow \text{ using the Green's theorem} \\
&= \int_{\partial\Omega} (\sigma_{ij}v_i)n_j\,\mathrm{d}s - \int_\Omega \sigma_{ij}v_{i,j}\,\mathrm{d}\Omega \\
&= \int_{\Gamma_u} (\sigma_{ij}n_j)v_i\,\mathrm{d}s + \int_{\Gamma_t} (\sigma_{ij}n_j)v_i\,\mathrm{d}s - \int_\Omega \sigma_{ij}v_{i,j}\,\mathrm{d}\Omega \\
&\downarrow \text{ substituting the traction boundary condition} \\
&= \int_{\Gamma_u} (\sigma_{ij}n_j)v_i\,\mathrm{d}s + \int_{\Gamma_t} \bar{t}_i v_i\,\mathrm{d}s - \int_\Omega \sigma_{ij}v_{i,j}\,\mathrm{d}\Omega \\
&= \int_{\Gamma_u} (\sigma n)\cdot v\,\mathrm{d}s + \int_{\Gamma_t} \bar{t}\cdot v\,\mathrm{d}s - \int_\Omega \sigma : \nabla v\,\mathrm{d}\Omega.
\end{aligned} \tag{2.5}$$

If we only use test functions, v, that are zero vectors on Γ_u we can further simplify the equation to

$$\int_\Omega (\operatorname{div}\sigma)\cdot v\,\mathrm{d}\Omega = \int_{\Gamma_t} \bar{t}\cdot v\,\mathrm{d}s - \int_\Omega \sigma : \nabla v\,\mathrm{d}\Omega. \tag{2.6}$$

Also for the first integral in the second equation,

$$\begin{aligned}\int_\Omega (\nabla u) : \tau \,\mathrm{d}\Omega &= \int_\Omega u_{i,j}\tau_{ij}\,\mathrm{d}\Omega \\ &= \int_\Omega [(u_i\tau_{ij})_{,j} - u_i\tau_{ij,j}]\,\mathrm{d}\Omega \\ &\downarrow \text{ using the Green's theorem} \\ &= \int_{\partial\Omega} (u_i\tau_{ij})n_j\,\mathrm{d}s - \int_\Omega u_i\tau_{ij,j}\,\mathrm{d}\Omega \\ &= \int_{\Gamma_u} u_i(\tau_{ij}n_j)\,\mathrm{d}s + \int_{\Gamma_t} u_i(\tau_{ij}n_j)\,\mathrm{d}s - \int_\Omega u_i\tau_{ij,j}\,\mathrm{d}\Omega \\ &\downarrow \text{ substituting the displacement boundary condition} \\ &= \int_{\Gamma_u} \bar{u}_i(\tau_{ij}n_j)\,\mathrm{d}s + \int_{\Gamma_t} u_i(\tau_{ij}n_j)\,\mathrm{d}s - \int_\Omega u_i\tau_{ij,j}\,\mathrm{d}\Omega \\ &= \int_{\Gamma_u} \bar{u}\cdot(\tau n)\,\mathrm{d}s + \int_{\Gamma_t} u\cdot(\tau n)\,\mathrm{d}s - \int_\Omega u\cdot(\operatorname{div}\tau)\,\mathrm{d}\Omega.\end{aligned} \tag{2.7}$$

If we assume that the test functions, τ, have the property, $\tau n = 0$ on Γ_t we can further simplify the equation to

$$\int_\Omega (\nabla u) : \tau\,\mathrm{d}\Omega = \int_{\Gamma_u} \bar{u}\cdot(\tau n)\,\mathrm{d}s - \int_\Omega u\cdot(\operatorname{div}\tau)\,\mathrm{d}\Omega. \tag{2.8}$$

Substituting 2.6 and 2.8 in 2.4, we get

$$\begin{aligned}\int_\Omega \sigma : \nabla v\,\mathrm{d}\Omega &= \int_\Omega f\cdot v\,\mathrm{d}\Omega + \int_{\Gamma_t} \bar{t}\cdot v\,\mathrm{d}s \\ \int_\Omega \sigma : S : \tau\,\mathrm{d}\Omega + \int_\Omega u\cdot(\operatorname{div}\tau)\,\mathrm{d}\Omega + \int_\Omega \omega : \tau\,\mathrm{d}\Omega &= \int_{\Gamma_u} \bar{u}\cdot(\tau n)\,\mathrm{d}s\end{aligned} \tag{2.9}$$

To write the formal definition of the ultraweak form we need to specify the function spaces to which the trial and test variables belong.

- $L^2(\Omega)$

  $L^2(\Omega)$ is the space of square integrable scalar functions defined over Ω and the corresponding L^2 norm is written as $\|\cdot\|_{L^2(\Omega)}$ and is defined as $\|u\|^2_{L^2(\Omega)} = \int_\Omega |u|^2 \,\mathrm{d}\Omega$ where $|\cdot|$ is the absolute value.

- $\boldsymbol{L}^2(\Omega)$

  Similarly, we can define vector versions of $L^2(\Omega)$. So, $\boldsymbol{L}^2(\Omega;\mathbb{R}^2)$ or just simply $\boldsymbol{L}^2(\Omega)$, is the space of vector fields with each component belonging to $L^2(\Omega)$ and the norm is defined as $\|u\|^2_{\boldsymbol{L}^2(\Omega)} = \|u_1\|^2_{L^2(\Omega)} + \|u_2\|^2_{L^2(\Omega)} = \int_\Omega |u_1|^2 \,\mathrm{d}\Omega + \int_\Omega |u_2|^2 \,\mathrm{d}\Omega = \int_\Omega (u_1^2 + u_2^2)\,\mathrm{d}\Omega = \int_\Omega |u|^2 \,\mathrm{d}\Omega$ where $|\cdot|$ is the Euclidean norm.

- $\boldsymbol{L}^2(\Omega;\mathbb{S})$, $\boldsymbol{L}^2(\Omega;\mathbb{A})$ and $\boldsymbol{L}^2(\Omega;\mathbb{M})$

  Also, we can define matrix versions of $L^2(\Omega)$. For example, $\boldsymbol{L}^2(\Omega;\mathbb{S})$ is the space of symmetric matrix functions where each component is a function in $L^2(\Omega)$ and the norm is defined as $\|u\|^2_{\boldsymbol{L}^2(\Omega;\mathbb{S})} = \|u_{11}\|^2_{L^2(\Omega)} + \|u_{12}\|^2_{L^2(\Omega)} + \|u_{21}\|^2_{L^2(\Omega)} + \|u_{22}\|^2_{L^2(\Omega)}$ where $u_{12} = u_{21}$. Another example is $\boldsymbol{L}^2(\Omega;\mathbb{A})$ which is the space of skew-symmetric matrix functions with each component being a function in $L^2(\Omega)$. Its norm is defined similar to the norm of $\boldsymbol{L}^2(\Omega;\mathbb{S})$. Similarly, for a general 2×2 matrix $\mathbb{M}$, we can define $\boldsymbol{L}^2(\Omega;\mathbb{M})$.

- $H^1(\Omega)$

  $H^1(\Omega)$ is the space of scalar functions over Ω which belong to $L^2(\Omega)$ with their gradients also in $L^2(\Omega)$. The H^1 norm is written as $\|\cdot\|_{H^1(\Omega)}$ and is defined as $\|u\|^2_{H^1(\Omega)} = \|u\|^2_{L^2(\Omega)} + \|\nabla u\|^2_{\boldsymbol{L}^2(\Omega)} = \int_\Omega |u|^2 \,\mathrm{d}\Omega + \int_\Omega |\nabla u|^2 \,\mathrm{d}\Omega$.

- $\boldsymbol{H}^1(\Omega)$

  The vector version of $H^1(\Omega)$ is $\boldsymbol{H}^1(\Omega)$, the space of vector functions defined over Ω where each component belongs to $H^1(\Omega)$. Its norm is defined as $\|u\|^2_{\boldsymbol{H}^1(\Omega)} = \|u_1\|^2_{H^1(\Omega)} + \|u_2\|^2_{H^1(\Omega)}$ or alternatively, as $\|u\|^2_{\boldsymbol{H}^1(\Omega)} = \|u\|^2_{\boldsymbol{L}^2(\Omega)} + \|\nabla u\|^2_{\boldsymbol{L}^2(\Omega;\mathbb{M})}$.

- $\boldsymbol{H}^1_{\Gamma_u}(\Omega)$

$\boldsymbol{H}^1_{\Gamma_u}(\Omega)$ is a subspace of $\boldsymbol{H}^1(\Omega)$. Its members have the additional property that their components are zero on Γ_u.

- $H(\mathrm{div},\Omega)$

  The next space is $H(\mathrm{div},\Omega)$. It's the space of vector functions which belong to $\boldsymbol{L}^2(\Omega)$ with their divergences in $L^2(\Omega)$. The $H(\mathrm{div},\Omega)$ norm is defined as $\|u\|^2_{H(\mathrm{div},\Omega)} = \|u\|^2_{\boldsymbol{L}^2(\Omega)} + \|\mathrm{div}(u)\|^2_{L^2(\Omega)} = \int_\Omega |u|^2 \,\mathrm{d}\Omega + \int_\Omega |\mathrm{div}(u)|^2 \,\mathrm{d}\Omega$. Comparing the definition of $H(\mathrm{div},\Omega)$ with $H^1(\Omega)$ we can think of $H^1(\Omega)$ as $H^1(\mathrm{grad},\Omega)$.

- $\boldsymbol{H}(\mathrm{div},\Omega)$

  $\boldsymbol{H}(\mathrm{div},\Omega)$ is the matrix version of $H(\mathrm{div},\Omega)$. Its members are 2×2 matrix functions where each row is a vector function in $H(\mathrm{div},\Omega)$.

- $\boldsymbol{H}_{\Gamma_t}(\mathrm{div},\Omega)$

  $\boldsymbol{H}_{\Gamma_t}(\mathrm{div},\Omega)$ is a subspace of $\boldsymbol{H}(\mathrm{div},\Omega)$. Its members are the matrix function which belong to $\boldsymbol{H}(\mathrm{div},\Omega)$ with the additional property that the dot product of each row with the unit normal vector on Γ_t is zero.

Looking back at equation 2.9, the trial functions are u, σ, and ω. u is a vector function and it only appears in $\int_\Omega u \cdot \mathrm{div}\tau \,\mathrm{d}\Omega$ and no differential operators are applied on it. So, we can choose our largest space, $\boldsymbol{L}^2(\Omega)$. σ and ω are symmetric and skew-symmetric matrix functions, respectively and we do not see any derivatives applied on them in 2.9. Therefore, the former belongs to $\boldsymbol{L}^2(\Omega;\mathbb{S})$ and the latter in $\boldsymbol{L}^2(\Omega;\mathbb{A})$.

However, for test functions v, and τ, using $\boldsymbol{L}^2$ spaces is not enough anymore and more regularity is needed. For the vector function v, its gradient appears in the term $\int_\Omega \sigma : \nabla v \,\mathrm{d}\Omega$. Therefore, v should be differentiable and its gradient be square integrable. This forces v to be in $\boldsymbol{H}^1(\Omega)$. Moreover, in the derivation of equation 2.6 we assumed that all test functions v are zero on Γ_u. Thus, v has to be in $\boldsymbol{H}^1_{\Gamma_u}(\Omega)$.

The other test function, τ, is a general (not necessarily symmetric) matrix function. For the term $\int_\Omega u \cdot \operatorname{div}\tau \, \mathrm{d}\Omega$ to make sense, $\operatorname{div}\tau$ needs to be square integrable. Hence, τ should be in $\boldsymbol{H}(\operatorname{div}, \Omega)$. Moreover, in the derivation of equation 2.8 we assumed that all test functions τ have the property, $\tau n = 0$ on Γ_t. So, τ has to be in $\boldsymbol{H}_{\Gamma_t}(\operatorname{div}, \Omega)$. Therefore, the ultraweak formulation can be written as following:

The ultraweak formulation

Find $u \in \boldsymbol{L}^2(\Omega)$, $\sigma \in \boldsymbol{L}^2(\Omega; S)$, $\omega \in \boldsymbol{L}^2(\Omega; A)$

$$\begin{aligned}
\int_\Omega \sigma : \nabla v \, \mathrm{d}\Omega &= \int_\Omega f \cdot v \, \mathrm{d}\Omega + \int_{\Gamma_t} \bar{t} \cdot v \, \mathrm{d}\Omega, \\
&\quad \text{for all } v \in \boldsymbol{H}^1_{\Gamma_u}(\Omega). \\
\int_\Omega \sigma : \mathsf{S} : \tau \, \mathrm{d}\Omega + \int_\Omega \omega : \tau \, \mathrm{d}\Omega + \int_\Omega u \cdot \operatorname{div}\tau \, \mathrm{d}\Omega &= \int_{\Gamma_u} \bar{u} \cdot (\tau n) \, \mathrm{d}\Omega, \\
&\quad \text{for all } \tau \in \boldsymbol{H}_{\Gamma_t}(\operatorname{div}, \Omega),
\end{aligned} \tag{2.10}$$

To get a practical numerical method we need to choose finite-dimensional subspaces of infinite-dimensional $\boldsymbol{L}^2$, $\boldsymbol{H}^1$, and $\boldsymbol{H}(\operatorname{div})$ spaces which is usually called the *discretization*. In the discretized form of 2.10, all trial variables will belong to subspaces of $\boldsymbol{L}^2$ and can be discontinuous across element boundaries. However, the subspaces of $\boldsymbol{H}^1$ need to be C^0 continuous at the element boundaries and for subspaces of $\boldsymbol{H}(\operatorname{div})$, the normal components of the vector fields should be continuous across the element boundaries. So, we still need to satisfy these continuity requirements for the test functions and as it was mentioned in previous chapter, there are no higher-order $\boldsymbol{H}(\operatorname{div})$ discretizations available for general polygons at the moment in the literature [GRB16] and the options for higher-order $\boldsymbol{H}^1$ discretizations on polygons are very limited [Suk13, RGB14]. This takes us to the next section where we are going to alleviate the continuity requirements on the test functions by using the *hybridization* idea [CDG16], i.e. defining independent variables on the interface (skeleton) of the mesh. For functions $u \in \boldsymbol{H}^1(\Omega)$, the trace operator, $\mathrm{tr}_{\mathrm{grad}} : \boldsymbol{H}^1(\Omega) \to \boldsymbol{H}^{\frac{1}{2}}(\partial\Omega)$, is defined by restricting the function to the boundary,

$\mathrm{tr}_{\mathrm{grad}}\, u = u|_{\partial\Omega}$. For functions $\sigma \in \boldsymbol{H}(\mathrm{div},\Omega)$, the trace operator, $\mathrm{tr}_{\mathrm{div}} : \boldsymbol{H}(\mathrm{div},\Omega) \to \boldsymbol{H}^{-\frac{1}{2}}(\partial\Omega)$, is defined as $\mathrm{tr}_{\mathrm{div}}\, \sigma = \sigma|_{\partial\Omega} \cdot n$ [Mon03] where n is the unit outward normal vector on the boundary.

2.2.3 The hybridized (broken) version of the ultraweak formulation

To achieve a more relaxed continuity requirement for our test functions we start by writing the elasticity equations 2.3 on an arbitrary element $T \subseteq \Omega$ and multiplying by test functions v and τ. For simplicity, we assume homogeneous boundary conditions in this section. Problems with inhomogeneous boundary conditions can be solved by splitting the solution into two parts and moving the inhomogeneous data to the right-hand side [CH16].

$$\begin{aligned} -\int_T (\mathrm{div}\sigma)\cdot v\,\mathrm{d}\Omega &= \int_T f\cdot v\,\mathrm{d}\Omega, \\ \int_T \nabla u : \tau\,\mathrm{d}\Omega - \int_T \omega : \tau\,\mathrm{d}\Omega &= \int_T \sigma : S : \tau\,\mathrm{d}\Omega. \end{aligned} \tag{2.11}$$

Applying the integration by parts to both equations gives us

$$\begin{aligned} \int_T \sigma : \nabla v\,\mathrm{d}\Omega &= \int_T f\cdot v\,\mathrm{d}\Omega + \int_{\partial T} (\sigma n_T)\cdot v\,\mathrm{d}s, \\ \int_T \sigma : \mathsf{S} : \tau\,\mathrm{d}\Omega + \int_T \omega : \tau\,\mathrm{d}\Omega + \int_T u\cdot \mathrm{div}\tau\,\mathrm{d}\Omega &= \int_{\partial T} u\cdot(\tau n_T)\,\mathrm{d}s. \end{aligned} \tag{2.12}$$

Our goal is to keep the trial variables in $\boldsymbol{L}^2$ space just like the case of ultraweak formulation. Following the definition of the trace operator, the terms σn_T and u in the integrals on ∂T are in fact $\sigma|_{\partial T}\, n_T = \mathrm{tr}_{\mathrm{div}}\sigma$ and $u|_{\partial T} = \mathrm{tr}_{\mathrm{grad}}\, u$, respectively. These expressions are well-defined when $u \in \boldsymbol{H}^1$ and $\sigma \in \boldsymbol{H}(\mathrm{div})$. The problem is that the trace is not well-defined for functions in $\boldsymbol{L}^2$. Therefore, we introduce new variables $\hat{u}$ and $\hat{t}$ that live on ∂T to represent $\mathrm{tr}_{\mathrm{grad}}\, u$ and $\mathrm{tr}_{\mathrm{div}}\, \sigma$.

Then we can rewrite 2.12 as

$$\begin{aligned} \int_T \sigma : \nabla v \, \mathrm{d}\Omega &= \int_T f \cdot v \, \mathrm{d}\Omega + \int_{\partial T} \hat{t} \cdot v \, \mathrm{d}s \,, \\ \int_T \sigma : \mathsf{S} : \tau \, \mathrm{d}\Omega + \int_T \omega : \tau \, \mathrm{d}\Omega + \int_T u \cdot \mathrm{div} \tau \, \mathrm{d}\Omega &= \int_{\partial T} \hat{u} \cdot (\tau n_T) \, \mathrm{d}s \,. \end{aligned} \tag{2.13}$$

Using the following notation for the inner product over T,

$$\begin{aligned} (u,v)_T &= \int_T uv \, \mathrm{d}\Omega \,, && \text{if } u \text{ and } v \text{ are scalars,} \\ (u,v)_T &= \int_T u_i v_i \, \mathrm{d}\Omega = \int_T u \cdot v \, \mathrm{d}\Omega \,, && \text{if } u \text{ and } v \text{ are vectors,} \\ (u,v)_T &= \int_T u_{ij} v_{ij} \, \mathrm{d}\Omega = \int_T u : v \, \mathrm{d}\Omega \,, && \text{if } u \text{ and } v \text{ are tensors,} \end{aligned} \tag{2.14}$$

and also the $\langle \cdot, \cdot \rangle_{\partial T}$ notation for integrals on ∂T, equation 2.13 can be rewritten as

$$\begin{aligned} (\sigma, \nabla v)_T &= (f, v)_T + \langle \hat{t}, v \rangle_{\partial T} \,, \\ (\mathsf{S} : \sigma, \tau)_T + (\omega, \tau)_T + (u, \mathrm{div} \tau)_T &= \langle \hat{u}, \tau n_T \rangle_{\partial T} \,. \end{aligned} \tag{2.15}$$

Summing over all the elements T in a triangulation $\mathcal{T}$ we get

$$\begin{aligned} \sum_{T \in \mathcal{T}} (\sigma, \nabla_h v)_T &= \sum_{T \in \mathcal{T}} (f, v)_T + \sum_{T \in \mathcal{T}} \langle \hat{t}, v \rangle_{\partial T} \,, \\ \sum_{T \in \mathcal{T}} (\mathsf{S} : \sigma, \tau)_T + \sum_{T \in \mathcal{T}} (\omega, \tau)_T + \sum_{T \in \mathcal{T}} (u, \mathrm{div}_h \tau)_T &= \sum_{T \in \mathcal{T}} \langle \hat{u}, \tau n_T \rangle_{\partial T} \,. \end{aligned} \tag{2.16}$$

where ∇_h and div_h emphasize the piecewise actions of ∇ and div operators, respectively. Also, because of the element-wise nature of the equations, test functions v and τ inside any element $T_i \subseteq \Omega$, $v|_{T_i}$ and $\tau|_{T_i}$, only need to be in the spaces $\boldsymbol{H}^1(T_i)$ and $\boldsymbol{H}(\mathrm{div}, T_i)$, respectively. This property allows the test functions to have different values (hence, be discontinuous) at the element boundaries. In a formal definition, these test functions will belong to the so-called class of *broken Sobolev spaces* defined as following:

- $\boldsymbol{L}^2(\mathcal{T})$

  Basically each function in this space belongs to $\boldsymbol{L}^2(T)$ when restricted to element $T \in \mathcal{T}$. This is the same space as $\boldsymbol{L}^2(\Omega)$. Therefore, its norm is the same, $\|\cdot\|_{\boldsymbol{L}^2(\mathcal{T})} = \|\cdot\|_{\boldsymbol{L}^2(\Omega)}$.

- $\boldsymbol{H}^1(\mathcal{T})$

  Similar to $\boldsymbol{L}^2(\mathcal{T})$, each function in $\boldsymbol{H}^1(\mathcal{T})$ belongs to $\boldsymbol{H}^1(T)$ when restricted to element $T \in \mathcal{T}$. But in this case the broken and the unbroken spaces are different, $\boldsymbol{H}^1(\Omega) \subset \boldsymbol{H}^1(\mathcal{T})$. Its norm is defined as $\|\cdot\|^2_{\boldsymbol{H}^1(\mathcal{T})} = \sum_{T\in\mathcal{T}} \|\cdot|_T\|^2_{\boldsymbol{H}^1(T)}$.

- $\boldsymbol{H}(\mathrm{div},\mathcal{T})$

  Each function in this space belongs to $\boldsymbol{H}(\mathrm{div},T)$ when restricted to element $T \in \mathcal{T}$and $\boldsymbol{H}(\mathrm{div},\Omega) \subset \boldsymbol{H}(\mathrm{div},\mathcal{T})$. Its norm is defined as $\|\cdot\|^2_{\boldsymbol{H}(\mathrm{div},\mathcal{T})} = \sum_{T\in\mathcal{T}} \|\cdot|_T\|^2_{\boldsymbol{H}(\mathrm{div},T)}$.

For a consistent formulation, when we choose the test functions from the test spaces $\boldsymbol{H}^1(\Omega)$ and $\boldsymbol{H}(\mathrm{div},\Omega)$, i.e. respecting the previous inter-element continuity requirements of the test functions, we should be able to recover the (*unbroken*) ultraweak formulation 2.10. To show this, let $\mathscr{E}_{\mathrm{int}}$ be the set of all internal edges in the triangulation $\mathcal{T}$. Furthermore, assume the external edges are composed of two sets $\mathscr{E}_u := \partial\mathcal{T} \cap \Gamma_u$ and $\mathscr{E}_t := \partial\mathcal{T} \cap \Gamma_t$ where the former contains the edges with the displacement boundary condition and the latter contains the edges with the traction boundary condition. Changing the order of the summation for boundary integrals in 2.16 from element-wise to over the edges leads to

$$
\begin{aligned}
\sum_{T\in\mathcal{T}} \langle \hat{t}, v\rangle_{\partial T} &= \sum_{e\in\mathscr{E}_{\mathrm{int}}} \left(\int_{e^+} \hat{t}\cdot v^+ \,\mathrm{d}s + \int_{e^-} \hat{t}\cdot v^- \,\mathrm{d}s \right) \\
&\quad + \sum_{e\in\mathscr{E}_u} \int_{e^+} \hat{t}\cdot v^+ \,\mathrm{d}s + \sum_{e\in\mathscr{E}_t} \int_{e^+} \hat{t}\cdot v^+ \,\mathrm{d}s, \\
\sum_{T\in\mathcal{T}} \langle \hat{u}, \tau n_T\rangle_{\partial T} &= \sum_{e\in\mathscr{E}_{\mathrm{int}}} \left(\int_{e^+} \hat{u}\cdot(\tau^+ n^+) \,\mathrm{d}s + \int_{e^-} \hat{u}\cdot(\tau^- n^-) \,\mathrm{d}s \right) \\
&\quad + \sum_{e\in\mathscr{E}_u} \int_{e^+} \hat{u}\cdot(\tau^+ n^+) \,\mathrm{d}s + \sum_{e\in\mathscr{E}_t} \int_{e^+} \hat{u}\cdot(\tau^+ n^+) \,\mathrm{d}s.
\end{aligned}
\tag{2.17}
$$

For the internal edges, the two terms in parentheses come from the two elements sharing one internal edge and the plus and minus signs represent the values of the quantities on the different sides of the edge. The n^+ and n^- are the outward unit normal vectors of the neighboring elements on the shared edge e shown in figure 2.1.

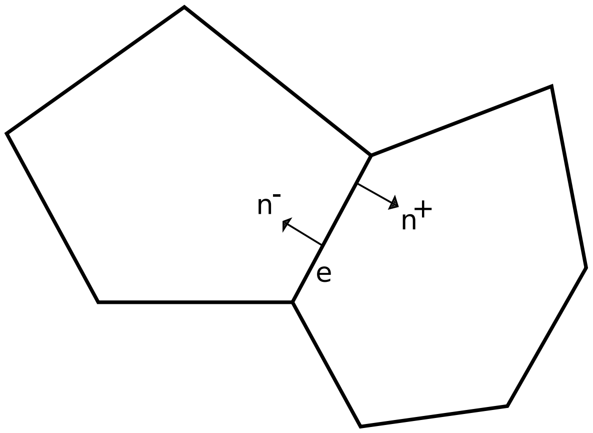


Figure 2.1: Neighboring elements

Using this notation, the continuity requirements at the element boundaries for functions in $\boldsymbol{H}^1(\Omega)$ and $\boldsymbol{H}(\mathrm{div},\Omega)$ can be expressed as

$$\begin{aligned} v^+ &= v^- && \text{for all } v \in \boldsymbol{H}^1(\Omega)\,, \\ (\tau^+ n^+) + (\tau^- n^-) &= 0 && \text{for all } \tau \in \boldsymbol{H}(\mathrm{div},\Omega)\,. \end{aligned} \tag{2.18}$$

Substituting 2.18 in equation 2.17 will simplify it to

$$\begin{aligned} \sum_{T\in\mathcal{T}} \langle \hat{t}, v\rangle_{\partial T} &= \sum_{e\in\mathscr{E}_u} \int_{e^+} \hat{t}\cdot v^+ \,\mathrm{d}s + \sum_{e\in\mathscr{E}_t} \int_{e^+} \hat{t}\cdot v^+ \,\mathrm{d}s\,, \\ \sum_{T\in\mathcal{T}} \langle \hat{u}, \tau n_T\rangle_{\partial T} &= \sum_{e\in\mathscr{E}_u} \int_{e^+} \hat{u}\cdot (\tau^+ n^+)\,\mathrm{d}s + \sum_{e\in\mathscr{E}_t} \int_{e^+} \hat{u}\cdot (\tau^+ n^+)\,\mathrm{d}s\,. \end{aligned} \tag{2.19}$$

Applying the homogeneous boundary conditions to $\hat{t}$ and $\hat{u}$ on edges in $\mathscr{E}_t$ and $\mathscr{E}_u$ respectively simplifies 2.19 to

$$
\begin{aligned}
\sum_{T\in\mathcal{T}} \langle \hat{t}, v\rangle_{\partial T} &= \sum_{e\in\mathscr{E}_u} \int_{e^+} \hat{t}\cdot v^+ \,\mathrm{d}s\,,\\
\sum_{T\in\mathcal{T}} \langle \hat{u}, \tau n_T\rangle_{\partial T} &= \sum_{e\in\mathscr{E}_t} \int_{e^+} \hat{u}\cdot (\tau^+ n^+) \,\mathrm{d}s\,.
\end{aligned}
\tag{2.20}
$$

So, by choosing the test functions with the properties of $v = 0$ on $\mathscr{E}_u$ and $\tau n = 0$ on $\mathscr{E}_t$ we can get rid of the remaining terms and we recover the original (unbroken) ultraweak formulation. Now, we can write 2.16 in its final form:

$\boxed{\textit{The broken version of the ultraweak formulation}}$

$$
\begin{aligned}
&\text{Find } u\in \boldsymbol{L}^2(\Omega),\, \sigma\in \boldsymbol{L}^2(\Omega;S),\, \omega\in \boldsymbol{L}^2(\Omega;A),\, \hat{u}\in \boldsymbol{H}^{\frac{1}{2}}_{\Gamma_u}(\partial\mathcal{T}),\, \hat{t}\in \boldsymbol{H}^{-\frac{1}{2}}_{\Gamma_t}(\partial\mathcal{T}),\\
&\qquad \sum_{T\in\mathcal{T}} (\sigma, \nabla_h v)_T - \sum_{T\in\mathcal{T}} \langle \hat{t}, v\rangle_{\partial T} = \sum_{T\in\mathcal{T}} (f, v)_T\,,\\
&\qquad\qquad \text{for all } v\in \boldsymbol{H}^1_{\Gamma_u}(\mathcal{T})\,,\\
&\sum_{T\in\mathcal{T}} (\mathsf{S}:\sigma, \tau)_T + \sum_{T\in\mathcal{T}} (\omega, \tau)_T + \sum_{T\in\mathcal{T}} (u, \mathrm{div}_h \tau)_T - \sum_{T\in\mathcal{T}} \langle \hat{u}, \tau n_T\rangle_{\partial T} = 0\,,\\
&\qquad\qquad \text{for all } \tau\in \boldsymbol{H}_{\Gamma_t}(\mathrm{div}, \mathcal{T})\,.
\end{aligned}
\tag{2.21}
$$

Comparing equation 2.21 with equation 2.10, two new unknowns ($\hat{u}$, and $\hat{t}$) have been added to the trial functions. But, now the test functions can be discontinuous across the boundaries of the elements.

2.3 The discontinuous Petrov-Galerkin (dPG) framework

There are three interpretations for the dPG method. Two of them were mentioned before. Here, we are interested in the third interpretation which is dPG as a mixed formulation.

2.3.1 The abstract form

We can think of dPG as a minimum residual method. The weak form can be written in an abstract form as

$$\begin{aligned}&\text{Find } u \in U \text{ such that}\\ &b(u,v) = l(v) \quad \text{for all } v \in V\end{aligned} \tag{2.22}$$

where b is a bilinear form $b : U \times V \to \mathbb{R}$ and l is a linear form (functional) $l : V \to \mathbb{R}$. U and V are in general infinite-dimensional Hilbert spaces. The operator form of this equation reads

$$\begin{aligned}&\text{Find } u \in U \text{ such that}\\ &Bu = L\end{aligned} \tag{2.23}$$

where $B : U \to V'$ is defined as $\langle Bu, v\rangle_{V' \times V} = b(u,v)$ and $L : V \to \mathbb{R}$ is defined as $Lv = l(v)$. $\langle \cdot, \cdot \rangle_{V' \times V}$ denotes the duality pairing on $V' \times V$.

To find an approximate solution to u, we search in a finite-dimensional subspace of U and pick the solution that minimizes the norm of the residual (i.e. has the shortest distance to the exact solution).

$$\underset{u_h \in U_h \subset U}{\arg\min} \|Bu_h - L\|^2_{V'} \tag{2.24}$$

Following the discussion in [KFD16, FKDLT17, DG14], the optimality condition is obtained by vanishing the directional derivative of $\|Bu_h - L\|^2_{V'}$ in the arbitrary direction $\delta u \in U_h$. So,

$$\begin{aligned}&(Bu_h - L, B\delta u)_{V'} = 0\\ \Rightarrow \quad &(R_V^{-1}(Bu_h - L), R_V^{-1} B\delta u)_V = 0\end{aligned} \tag{2.25}$$

where $R_V : V \to V'$ is the Riesz operator.

By defining the so-called error representation function $\psi = R_V^{-1}(Bu_h - L)$, we can rewrite

$(\psi, R_V^{-1} B\delta u)_V = 0$ in a mixed form as

$$\begin{cases} \text{Find } \psi \in V,\, u_h \in U_h \\ -(\psi, v)_V + b(u_h, v) = l(v) \qquad \forall v \in V \\ \qquad\qquad b(\delta u, \psi) = 0 \qquad \forall \delta u \in U_h \end{cases} \tag{2.26}$$

The first equation comes from the definition of ψ as follows

$$\begin{aligned} &\psi = R_V^{-1}(Bu_h - L) \\ &\implies R_V\psi = Bu_h - L \\ &\implies \langle R_V\psi, v\rangle_{V'\times V} = \langle Bu_h, v\rangle_{V'\times V} - Lv \\ &\implies (\psi, v)_V = b(u_h, v) - l(v) \\ &\implies \boxed{-(\psi, v)_V + b(u_h, v) = l(v)} \end{aligned} \tag{2.27}$$

To get the second equation we have

$$\begin{aligned} &(\psi, R_V^{-1}B\delta u)_V = 0 \\ &\implies (R_V^{-1}B\delta u, \psi)_V = 0 \\ &\implies \langle B\delta u, \psi\rangle_{V'\times V} = 0 \\ &\implies \boxed{b(\delta u, \psi) = 0} \end{aligned} \tag{2.28}$$

To be able to solve equation 2.26 we still need to discretize V. The so-called ideal dPG uses the optimal test space $V_h^{opt} = R_V^{-1}BU_h$ which requires finding the inverse of the Riesz map, R_V. This map is an infinite dimensional operator and computing its inverse is not possible in general. To get a practical dPG method, the optimal test space is approximated by a finite dimensional space V^{enr} with a dimension higher than the dimension of U_h. So, we can rewrite equation 2.26 as

$$\begin{cases} \text{Find } \psi_h \in V^{enr},\, u_h \in U_h \\ \quad -(\psi_h, v)_V + b(u_h, v) = l(v) \qquad \forall v \in V^{enr} \\ \qquad\qquad b(\delta u, \psi_h) = 0 \qquad \forall \delta u \in U_h \end{cases} \tag{2.29}$$

Or in operator form

$$\begin{bmatrix} -R_{V^{enr}} & B \\ B^T & 0 \end{bmatrix} \begin{bmatrix} \psi_h \\ u_h \end{bmatrix} = \begin{bmatrix} l \\ 0 \end{bmatrix} \tag{2.30}$$

where

$$\begin{aligned} (\psi_h, v)_V &= \langle R_{V^{enr}} \psi_h, v \rangle_{V' \times V} \\ b(u_h, v) &= \langle B u_h, v \rangle_{V' \times V} \\ b(\delta u, \psi_h) &= \langle B^T \psi_h, \delta u \rangle_{U' \times U} \end{aligned} \tag{2.31}$$

By static condensation and eliminating ψ_h we get

$$B^T R_{V^{enr}}^{-1} B u_h = B^T R_{V^{enr}}^{-1} l \tag{2.32}$$

Now, we choose the basis $\{\rho_i\}_{i=1}^{M}$ for V^{enr} and the basis $\{\phi_i\}_{i=1}^{N}$ for U_h where $M = \dim(V^{enr})$ and $N = \dim(U_h)$ (Note: $M > N$). So, $\psi_h = \sum_{i=1}^{M} (\psi_h)_i \rho_i$ and $u_h = \sum_{i=1}^{N} (u_h)_i \phi_i$ where $(\psi_h)_i$ and $(u_h)_i$ are the degrees of freedom. Using these bases we get

$$\begin{aligned} [R_{V^{enr}}]_{ij} &= (\rho_i, \rho_j)_V = [G]_{ij} \\ [B]_{ij} &= b(\phi_j, \rho_i) \\ [l]_i &= l(\rho_i) \end{aligned} \tag{2.33}$$

If the variational problem is written in the broken form then $\{\phi_i\}_{i=1}^{N} = \left\{\mathring{\phi}_i\right\}_{i=1}^{\mathring{N}} \cup \{\hat{\phi}_i\}_{i=1}^{\hat{N}}$ where $N = \mathring{N} + \hat{N}$ and $\left\{\mathring{\phi}_i\right\}_{i=1}^{\mathring{N}}$ and $\{\hat{\phi}_i\}_{i=1}^{\hat{N}}$ are the basis functions for the domain and skeleton (interface) degrees of freedom, $(\mathring{u}_h)_i$ and $(\hat{u}_h)_i$, respectively. In this case, The $[B]$ matrix is composed of two submatrices,

$$\begin{bmatrix} B \end{bmatrix} = \begin{bmatrix} \mathring{B} & \hat{B} \end{bmatrix} \tag{2.34}$$

where $[\mathring{B}]_{ij} = b(\mathring{\phi}_j, \rho_i)$ and $[\hat{B}]_{ij} = b(\hat{\phi}_j, \rho_i)$. Substituting this B in equation 2.32 will give us

$$\begin{bmatrix} \mathring{B}^T G^{-1} \mathring{B} & \mathring{B}^T G^{-1} \hat{B} \\ \hat{B}^T G^{-1} \mathring{B} & \hat{B}^T G^{-1} \hat{B} \end{bmatrix} \begin{bmatrix} \mathring{u}_h \\ \hat{u}_h \end{bmatrix} = \begin{bmatrix} \mathring{B}^T G^{-1} l \\ \hat{B}^T G^{-1} l \end{bmatrix}. \tag{2.35}$$

If we further condense out the domain degrees of freedom, $[\mathring{u}_h]$, we get a linear system with only the skeleton degrees of freedom as the unknowns

$$\begin{bmatrix} \mathbf{D} - \mathbf{C}\mathbf{A}^{-1}\mathbf{B} \end{bmatrix} \begin{bmatrix} \hat{u}_h \end{bmatrix} = \begin{bmatrix} \mathbf{g} - \mathbf{C}\mathbf{A}^{-1}\mathbf{f} \end{bmatrix}. \tag{2.36}$$

where

$$\begin{aligned} \mathbf{A} &= \mathring{B}^T G^{-1} \mathring{B} \\ \mathbf{B} &= \mathring{B}^T G^{-1} \hat{B} \\ \mathbf{C} &= \hat{B}^T G^{-1} \mathring{B} \\ \mathbf{D} &= \hat{B}^T G^{-1} \hat{B} \\ \mathbf{f} &= \mathring{B}^T G^{-1} l \\ \mathbf{g} &= \hat{B}^T G^{-1} l \end{aligned} \tag{2.37}$$

This is a global linear system for all skeleton dofs. In general, having to compute the inverse of the G matrix will be costly. But, in the broken version of the ultraweak formulation the basis functions of domain variables, ρ_i and $\mathring{\phi}_i$, are discontinuous across the elements and since $[G]_{ij} = (\rho_i, \rho_j)_V$, $[G]_{ij}$ is non-zero only if (domain) degrees of freedom i and j belong to the same element. Otherwise, it is zero. This leads to a block-diagonal G matrix when i and j are ordered elementwise. Then, G^{-1} is obtained by inverting these elementwise blocks. So, similar to $[G]_{ij}$, $[G^{-1}]_{ij}$ is non-zero only if (domain) degrees of freedom i and j belong to the same element and zero otherwise. With the same reasoning, since $[\mathring{B}]_{ij} = b(\mathring{\phi}_i, \rho_i)$, $[\mathring{B}]_{ij}$ is non-zero only if (domain) degrees of freedom i and j belong to the same element.

2.3.2 dPG for linear elasticity

To use the general framework 2.3 in the case of linear elasticity in the broken ultraweak form, the group variables for trial and test functions and the corresponding functional spaces are as follows:

$$\begin{aligned}
\mathsf{u} &= (\sigma, u, \omega, \hat{u}, \hat{t})\,,\\
\mathring{\mathsf{u}} &= (\sigma, u, \omega)\,,\\
\hat{\mathsf{u}} &= (\hat{u}, \hat{t})\,,\\
\mathsf{v} &= (\tau, v)\,,\\
U &= \boldsymbol{L}^2(\Omega;\mathbb{S}) \times \boldsymbol{L}^2(\Omega) \times \boldsymbol{L}^2(\Omega;\mathbb{A}) \times \boldsymbol{H}^{\frac{1}{2}}_{\Gamma_u}(\partial\mathcal{T}) \times \boldsymbol{H}^{-\frac{1}{2}}_{\Gamma_t}(\partial\mathcal{T})\,,\\
U_0 &= \boldsymbol{L}^2(\Omega;\mathbb{S}) \times \boldsymbol{L}^2(\Omega) \times \boldsymbol{L}^2(\Omega;\mathbb{A})\,,\\
\hat{U} &= \boldsymbol{H}^{\frac{1}{2}}_{\Gamma_u}(\partial\mathcal{T}) \times \boldsymbol{H}^{-\frac{1}{2}}_{\Gamma_t}(\partial\mathcal{T})\,,\\
V &= \boldsymbol{H}_{\Gamma_t}(\mathrm{div},\mathcal{T}) \times \boldsymbol{H}^1_{\Gamma_u}(\mathcal{T})\,.
\end{aligned} \tag{2.38}$$

The bilinear form and the linear functional on the right-hand side (see equation 2.21) are given as:

$$\begin{aligned}
b_0((\sigma, u, \omega), (\tau, v)) \quad &= (S:\sigma, \tau)_{\mathcal{T}} + (\omega, \tau)_{\mathcal{T}} + (u, \mathrm{div}_h \tau)_{\mathcal{T}} + (\sigma, \nabla_h v)_{\mathcal{T}}\,,\\
\hat{b}((\hat{u}, \hat{t}), (\tau, v)) \quad &= -\langle \hat{u}, \tau n_T \rangle_{\partial\mathcal{T}} - \langle \hat{t}, v \rangle_{\partial\mathcal{T}}\,,\\
l((\tau, v)) \quad &= (f, v)_{\mathcal{T}}\,.
\end{aligned} \tag{2.39}$$

where the following notation is used to simplify the equations,

$$\begin{aligned}
(\cdot,\cdot)_{\mathcal{T}} &:= \sum_{T\in\mathcal{T}} (\cdot,\cdot)_T\,,\\
\langle\cdot,\cdot\rangle_{\partial\mathcal{T}} &:= \sum_{T\in\mathcal{T}} \langle\cdot,\cdot\rangle_{\partial T}\,.
\end{aligned} \tag{2.40}$$

2.3.3 Discretization

Discretization of domain variables

To choose the finite-dimensional subspaces of the trial and test spaces we follow the approach of [AFMD18] which is based on bounding the polygon by a box or a triangle and using the more familiar higher-order discretizations of $\boldsymbol{H}^1$, $\boldsymbol{H}(\mathrm{div})$, and $\boldsymbol{L}^2$ spaces for squares or triangles. This is possible because of the broken ultraweak formulation. There are no continuity requirements across element boundaries for the domain variables σ, u, and ω and also for the test variables τ and v. So, we can use the basis functions of a bigger triangle and restrict the functions to the polygonal region of the element inside the triangle. We will use equilateral bounding triangles (Q_T) for all polygonal elements in the mesh $(T \in \mathcal{T})$ as shown in figure 2.2.

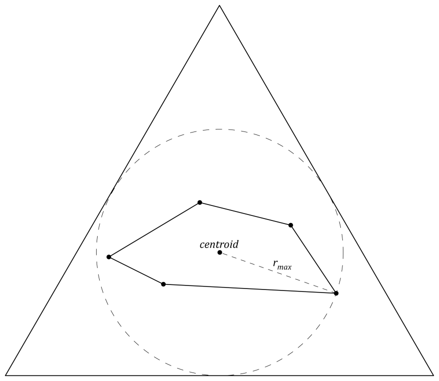


Figure 2.2: The equilateral triangle bounding a polygonal element

To find the coordinates of the bounding triangle, Q_T, we start by finding the smallest circle that contains the polygonal element. The center of the circle will be the centroid of the element, $c = (c_x, c_y) = (\frac{1}{n}\sum_{i=1}^{n} x_i, \frac{1}{n}\sum_{i=1}^{n} y_i)$ for a polygon with n vertices, $v_i = (x_i, y_i)$ for $i = 1 \ldots n$. The radius of the circle, r, will be the distance between the centroid and the farthest vertex from the centroid, $r = \max_{\{i=1\ldots n\}} d_i = \max_{\{i=1\ldots n\}} \sqrt{(x_i - c_x)^2 + (y_i - c_y)^2}$. Next, this circle will be inscribed in an equilateral triangle with the vertex coordinates, $q_1 = (c_x - \overline{AP}, c_y - r)$, $q_2 = (c_x + \overline{PB}, c_y - r)$, $q_3 = (c_x, c_y + \overline{OC})$. Looking at figure 2.3, $\overline{AP}$, $\overline{PB}$, and $\overline{OC}$ can be written in terms of the radius of the incircle, r.

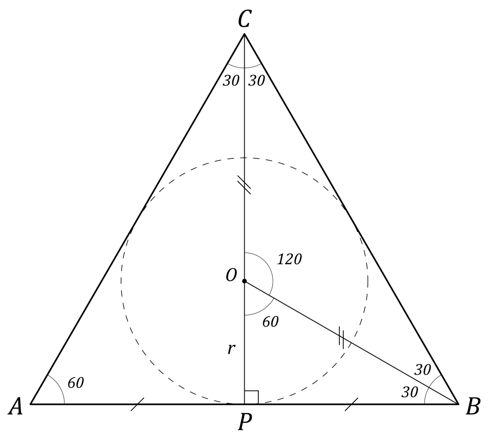


Figure 2.3: The inscribed circle of an equilateral triangle

$$\begin{aligned}
\overline{AP} = \overline{PB} = \overline{OP} \cdot \cot 30^\circ = r \cot 30^\circ = \sqrt{3}r\,, \\
\overline{OC} = \overline{OB} = \frac{\overline{OP}}{\sin 30^\circ} = \frac{\overline{OP}}{0.5} = 2r\,.
\end{aligned} \tag{2.41}$$

The coordinates of the three vertices of the bounding triangle (written in terms of c_x, c_y, and r) are listed in table 2.1:

Table 2.1: The coordinates of the vertices of the equilateral triangle bounding the polygonal element

vertices	x coordinate	y coordinate
q_1	$c_x - \sqrt{3}r$	$c_y - r$
q_2	$c_x + \sqrt{3}r$	$c_y - r$
q_3	c_x	$c_y + 2r$

To choose the discretizations of the spaces $\boldsymbol{H}^1(Q_T)$, $\boldsymbol{H}(\text{div}, Q_T)$, and $\boldsymbol{L}^2(Q_T)$ over the bounding triangle we use the fact that these spaces form the vector version of the de Rham complex,

$$\boldsymbol{H}^1(Q_T) \xrightarrow{\text{curl}} \boldsymbol{H}(\text{div}, Q_T) \xrightarrow{\text{div}} \boldsymbol{L}^2(Q_T) \tag{2.42}$$

where the curl and the div operators are applied row-wise. So, for a vector function v,

$$\operatorname{curl}(v) = \operatorname{curl}(\begin{bmatrix} v_1 \\ v_2 \end{bmatrix}) = \begin{bmatrix} -v_{1,2} & v_{1,1} \\ -v_{2,2} & v_{2,1} \end{bmatrix}, \tag{2.43}$$

and for a tensor function τ,

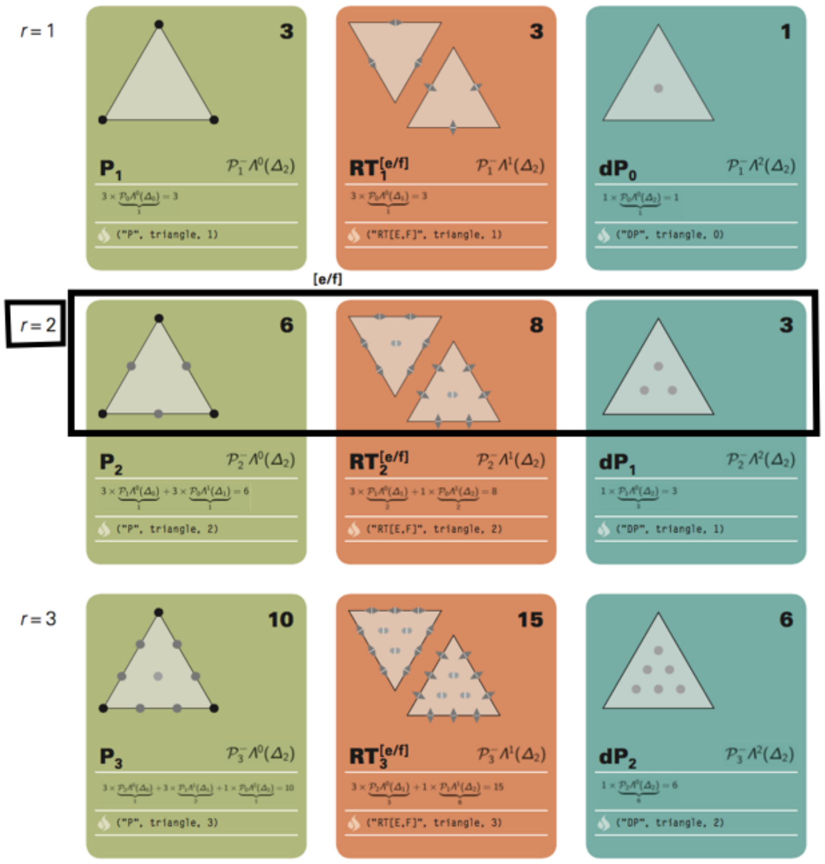
$$\operatorname{div}(\tau) = \operatorname{div}(\begin{bmatrix} \tau_{11} & \tau_{12} \\ \tau_{21} & \tau_{22} \end{bmatrix}) = \begin{bmatrix} \tau_{11,1} + \tau_{12,2} \\ \tau_{21,1} + \tau_{22,2} \end{bmatrix}. \tag{2.44}$$

The complex 2.42 is basically created by stacking two copies of the classical (scalar) de Rham complexes on top of each other. So, we can use two copies of the polynomial subcomplexes of the scalar de Rham complex to discretize 2.42, [AFW06, AFW10]. In the case of $\Omega \subseteq \mathbb{R}^n$, there are 2^{n-1} different polynomial subcomplexes to choose from. In our case, $Q_T \subset \mathbb{R}^2$, $n = 2$ and there are only two different options:

$$\mathcal{P}_r\Lambda^0(Q_T) \xrightarrow{d} \mathcal{P}_{r-1}\Lambda^1(Q_T) \xrightarrow{d} \mathcal{P}_{r-2}\Lambda^2(Q_T), \qquad r \geq 2 \tag{2.45}$$

$$\mathcal{P}_r\Lambda^0(Q_T) \xrightarrow{d} \mathcal{P}_r^-\Lambda^1(Q_T) \xrightarrow{d} \mathcal{P}_{r-1}\Lambda^2(Q_T), \qquad r \geq 1 \tag{2.46}$$

where $\mathcal{P}_r\Lambda^0(Q_T) = \mathcal{P}_r^-\Lambda^0(Q_T)$ and $\mathcal{P}_{r-1}\Lambda^n(Q_T) = \mathcal{P}_r^-\Lambda^n(Q_T)$. The complex 2.46 is the complex of Whitney forms [Whi57] written in terms of differential forms [AFW10].This is the polynomial complex that we are going to use. This sequence is shown on the periodic table of finite elements [AL14] for $r = 1, 2, 3$ in figure 2.4. r is the order of the sequence and it is not necessarily equal to the order of approximation. For example, for $r = 2$, $\mathcal{P}_2^-\Lambda^0$ is of second order while $\mathcal{P}_2^-\Lambda^2$ is only of first order.



r = 1
3
P1
$\mathcal{P}_1^-\Lambda^0(\Delta_2)$
("P", triangle, 1)
3
RT1[e/f]
$\mathcal{P}_1^-\Lambda^1(\Delta_2)$
("RT[E,F]", triangle, 1)
1
dP0
$\mathcal{P}_1^-\Lambda^2(\Delta_2)$
("DP", triangle, 0)
[e/f]
r = 2
6
P2
$\mathcal{P}_2^-\Lambda^0(\Delta_2)$
("P", triangle, 2)
8
RT2[e/f]
$\mathcal{P}_2^-\Lambda^1(\Delta_2)$
("RT[E,F]", triangle, 2)
3
dP1
$\mathcal{P}_2^-\Lambda^2(\Delta_2)$
("DP", triangle, 1)
r = 3
10
P3
$\mathcal{P}_3^-\Lambda^0(\Delta_2)$
("P", triangle, 3)
15
RT3[e/f]
$\mathcal{P}_3^-\Lambda^1(\Delta_2)$
("RT[E,F]", triangle, 3)
6
dP2
$\mathcal{P}_3^-\Lambda^2(\Delta_2)$
("DP", triangle, 2)

Figure 2.4: The Whitney complex in $\mathbb{R}^2$. The complex of order $r = 2$ is selected as an example. Taken from http://www.femtable.org

The dimension of each space in the sequence is given by the following formula:

$$\dim \mathcal{P}_r^- \Lambda^k = \binom{r+2}{r+k}\binom{r+k-1}{k} \tag{2.47}$$

As it is shown in the FEM table in figure 2.4, $\mathcal{P}_r^-\Lambda^1$ and $\mathcal{P}_r^-\Lambda^2$ spaces are the Raviart-Thomas [RT77] and discontinuous Lagrange spaces, respectively. $\mathcal{P}_r(Q_T)$ will be the space of all (scalar) polynomials in x and y of an order not greater than r defined over the triangular domain Q_T. The vector version of this space will be denoted by $\boldsymbol{\mathcal{P}}_r(Q_T)$ where each component is a member of $\mathcal{P}_r(Q_T)$. Similarly, we can define spaces of matrix polynomials like $\boldsymbol{\mathcal{P}}_r(Q_T;\mathbb{S})$ and $\boldsymbol{\mathcal{P}}_r(Q_T;\mathbb{A})$ for symmetric and skew-symmetric matrices, respectively. Specifically, for 2×2

matrices, $\mathbb{M}^{2\times 2}$, we have

$$\begin{aligned} \boldsymbol{\mathcal{P}}_r(Q_T;\mathbb{S}) &= \{ \begin{pmatrix} a & b \\ b & c \end{pmatrix} | a,b,c \in \mathcal{P}_r(Q_T)\} \\ \boldsymbol{\mathcal{P}}_r(Q_T;\mathbb{A}) &= \{ \begin{pmatrix} 0 & -a \\ a & 0 \end{pmatrix} | a \in \mathcal{P}_r(Q_T)\} \end{aligned} \tag{2.48}$$

The Raviart-Thomas spaces are defined [BBF13] as

$$\mathcal{RT}_r(Q_T) = \{\underline{p} \in \left(\boldsymbol{\mathcal{P}}_r(Q_T) + \begin{pmatrix} x \\ y \end{pmatrix} \mathcal{P}_r(Q_T) \right) | \underline{p}\cdot n \in \mathcal{R}_r(\partial Q_T)\} \tag{2.49}$$

where $\mathcal{R}_r(\partial Q_T) = \{\phi \in L^2(\partial Q_T) : \phi|_{e_i} \in \mathcal{P}_r(e_i), \text{for } i = 1,2,3\}$. The matrix version of the Raviart-Thomas space is defined as

$$\begin{aligned} \boldsymbol{\mathcal{RT}}_r(Q_T) = \{\tau = \begin{pmatrix} \tau_{11} & \tau_{12} \\ \tau_{21} & \tau_{22} \end{pmatrix} | \vec{\tau_1}^T &= \begin{pmatrix} \tau_{11} \\ \tau_{12} \end{pmatrix} \in \mathcal{RT}_r(Q_T) \\ \text{and} \quad \vec{\tau_2}^T &= \begin{pmatrix} \tau_{21} \\ \tau_{22} \end{pmatrix} \in \mathcal{RT}_r(Q_T)\}\,. \end{aligned} \tag{2.50}$$

The dimensions of these spaces are

$$\begin{aligned} \dim \mathcal{RT}_r(Q_T) &= (r+1)(r+3)\,, \quad \text{for } r \geq 0\,, \\ \dim \boldsymbol{\mathcal{RT}}_r(Q_T) &= 2(r+1)(r+3)\,, \quad \text{for } r \geq 0\,. \end{aligned} \tag{2.51}$$

So far, the definitions of these finite-dimensional spaces were limited to just one bounding triangle, Q_T. In the following, we extend the definitions to the whole mesh, $\mathcal{T}$.

$$\begin{aligned}
\boldsymbol{\mathcal{P}}_r(\mathcal{T}) &:= \{u|\ u|_T \in \boldsymbol{\mathcal{P}}_r(Q_T),\ \text{for all } T \in \mathcal{T}\}\,,\\
\boldsymbol{\mathcal{P}}_r(\mathcal{T};\mathbb{S}) &:= \{\sigma|\ \sigma|_T \in \boldsymbol{\mathcal{P}}_r(Q_T;\mathbb{S}),\ \text{for all } T \in \mathcal{T}\}\,,\\
\boldsymbol{\mathcal{P}}_r(\mathcal{T};\mathbb{A}) &:= \{\sigma|\ \sigma|_T \in \boldsymbol{\mathcal{P}}_r(Q_T;\mathbb{A}),\ \text{for all } T \in \mathcal{T}\}\,,\\
\boldsymbol{\mathcal{RT}}_r(\mathcal{T}) &:= \{\sigma|\ \sigma|_T \in \boldsymbol{\mathcal{RT}}_r(Q_T),\ \text{for all } T \in \mathcal{T}\}\,.
\end{aligned} \tag{2.52}$$

Using the above polynomial spaces we can define the discrete subspaces $U_{0_h} \subset U_0$ and $V_h \subset V$ (U_0 and V were previously defined in equation 2.38). For U_{0_h} and V_h, the spaces are chosen from complexes of order r and $r+\Delta r$, respectively. The criteria for choosing Δr is mentioned in the next section (after we discretized the skeleton variables).

$$\begin{aligned}
U_{0_h} &:= \boldsymbol{\mathcal{P}}_{r-1}(\mathcal{T};\mathbb{S}) \times \boldsymbol{\mathcal{P}}_{r-1}(\mathcal{T}) \times \boldsymbol{\mathcal{P}}_{r-1}(\mathcal{T};\mathbb{A})\,,\\
V_h &:= \boldsymbol{\mathcal{RT}}_{r-1+\Delta r}(\mathcal{T}) \times \boldsymbol{\mathcal{P}}_{r+\Delta r}(\mathcal{T})\,.
\end{aligned} \tag{2.53}$$

So, we have

$$\begin{aligned}
&\sigma \in \boldsymbol{\mathcal{P}}_{r-1}(\mathcal{T};\mathbb{S})\,,\\
&u \in \boldsymbol{\mathcal{P}}_{r-1}(\mathcal{T})\,,\\
&\omega \in \boldsymbol{\mathcal{P}}_{r-1}(\mathcal{T};\mathbb{A})\,,\\
&\tau \in \boldsymbol{\mathcal{RT}}_{r-1+\Delta r}(\mathcal{T})\,,\\
&v \in \boldsymbol{\mathcal{P}}_{r+\Delta r}(\mathcal{T})\,.
\end{aligned} \tag{2.54}$$

Δr is chosen such that $\dim V_h \geq \dim U_h$ (U_h is defined later in equation 2.56). In our computations, $\Delta r = 2$ was enough to satisfy the inequality.

Next, we talk about how to discretize the variables on the element interfaces, $\hat{u}$ and $\hat{t}$. We refer to these variables as skeleton or interface variables.

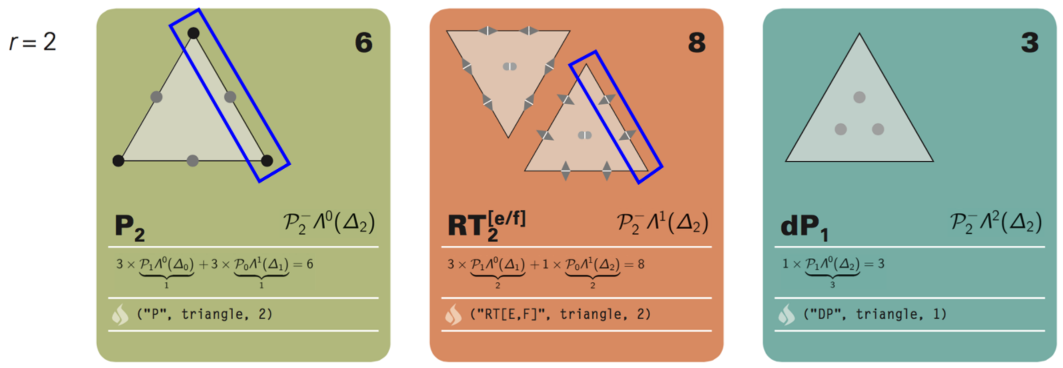
Discretization of skeleton variables

On the edges of each element T, the skeleton variables $\hat{u}$ and $\hat{t}$ represent $\mathrm{tr}_{\mathrm{grad}}\, u$ and $\mathrm{tr}_{\mathrm{div}}\, \sigma$, respectively where $u \in \boldsymbol{H}^1(T)$ and $\sigma \in \boldsymbol{H}(\mathrm{div}, T)$. In the previous section, for the discretization of trial variables defined on the domain we used the discretization of a complex of order r. Therefore, naturally we can use the traces of the same discrete complex to approximate $\hat{u}$ and $\hat{t}$.

The trace of $\boldsymbol{H}^1(T)$ should be compatible at the edges. So, $\mathcal{P}_r(e)$ will be used to approximate $\hat{u}$ on any edge e. The trace of $\boldsymbol{H}^1(T)$ should also be compatible at the vertices. This requires $\mathcal{P}_r(e)$ to be continuous at the vertices. Therefore, $\mathcal{P}_r^C(e)$.

The trace of $\boldsymbol{H}(\mathrm{div}, T)$ is compatible only at the edges. Thus, we can use $\mathcal{P}_{r-1}(e)$ to discretize $\hat{t}$ (discontinuous at the vertices).

For example, $\mathcal{P}_2^C(e)$ and $\mathcal{P}_1(e)$ (for the case of $r = 2$) are shown in figure 2.5 for a sample edge, e.



$r = 2$
6
$\mathbf{P_2}$
$\mathcal{P}_2^- \Lambda^0(\Delta_2)$
$3 \times \underbrace{\mathcal{P}_1\Lambda^0(\Delta_0)}_{1} + 3 \times \underbrace{\mathcal{P}_0\Lambda^1(\Delta_1)}_{1} = 6$
("P", triangle, 2)
8
$\mathbf{RT_2^{[e/f]}}$
$\mathcal{P}_2^- \Lambda^1(\Delta_2)$
$3 \times \underbrace{\mathcal{P}_1\Lambda^0(\Delta_1)}_{2} + 1 \times \underbrace{\mathcal{P}_0\Lambda^1(\Delta_2)}_{2} = 8$
("RT[E,F]", triangle, 2)
3
$\mathbf{dP_1}$
$\mathcal{P}_2^- \Lambda^2(\Delta_2)$
$1 \times \underbrace{\mathcal{P}_1\Lambda^0(\Delta_2)}_{3} = 3$
("DP", triangle, 1)

Figure 2.5: Example: The complex of order $r = 2$. Taken from http://www.femtable.org

The extension to the whole mesh will be

$$
\begin{aligned}
\mathcal{P}_r^C(\mathscr{E}) &:= \{\hat{u} | \hat{u}|_e \in \mathcal{P}_r^C(e) \quad \text{for all } e \in \mathscr{E}\}, \\
\mathcal{P}_{r-1}(\mathscr{E}) &:= \{\hat{t} | \hat{t}|_e \in \mathcal{P}_{r-1}(e) \quad \text{for all } e \in \mathscr{E}\}.
\end{aligned}
\tag{2.55}
$$

With these spaces in hand, we can now define the discrete trial space U_h:

$$U_h := \boldsymbol{\mathcal{P}}_{r-1}(\mathcal{T};\mathbb{S}) \times \boldsymbol{\mathcal{P}}_{r-1}(\mathcal{T}) \times \boldsymbol{\mathcal{P}}_{r-1}(\mathcal{T};\mathbb{A}) \times \boldsymbol{\mathcal{P}}_{r}^{C}(\mathscr{E}) \times \boldsymbol{\mathcal{P}}_{r-1}(\mathscr{E}) . \tag{2.56}$$

where $(u, \sigma, \omega, \hat{u}, \hat{t}) \in U_h$.

As an example, the degrees of freedom for trial variables are shown in figure 2.6 for $r = 2$.

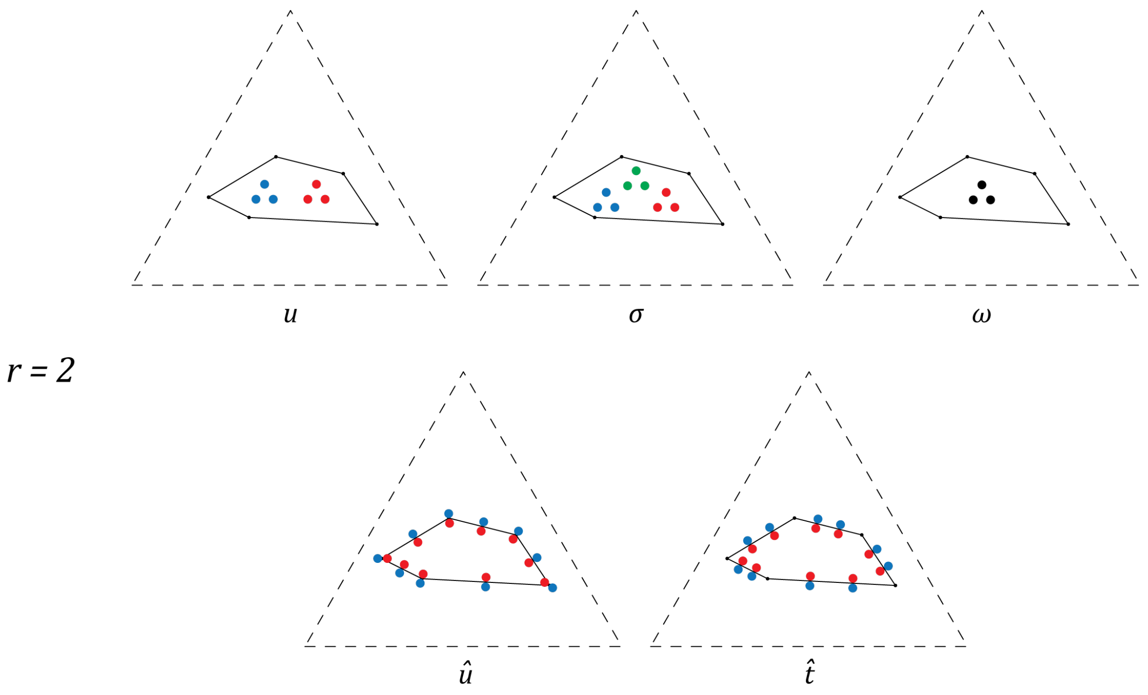


Figure 2.6: The degrees of freedom of trial variables for $r = 2$

2.3.4 Element matrices

The skeleton variables are the solution to the global linear system 2.36 and the domain variables are recovered in a post-processing step. The stiffness matrix on the left-hand side and the load vector on the right-hand side of 2.36 are computed by assembling the elemental contributions. So, to find the stiffness matrix we need to find the element matrices $[G]^{(el)}$ and

$[B]^{(el)}$. Then, $[K]^{(el)} = [B]^{(el)^T}[G]^{(el)^{-1}}[B]^{(el)}$. The final element stiffness matrix is obtained by static condensation to eliminate the degrees of freedom associated with domain variables. We refer to this final matrix as $[\hat{K}]^{(el)}$. The $[B]$ and $[G]$ matrices were defined in 2.33 for an abstract boundary value problem.

- The $[B]^{(el)}$ matrix

In the case of linear elasticity, the bilinear form is given in 2.39 and the corresponding $[B]^{(el)}$ matrix will be

$$[B]^{(el)} = \begin{bmatrix} \boldsymbol{B}_{\tau u} & \boldsymbol{B}_{\tau\sigma} & \boldsymbol{B}_{\tau\omega} & \boldsymbol{B}_{\tau\hat{u}} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{B}_{v\sigma} & \mathbf{0} & \mathbf{0} & \boldsymbol{B}_{v\hat{t}} \end{bmatrix}, \tag{2.57}$$

where

$$\boldsymbol{B}_{\tau u} = \begin{bmatrix} \boldsymbol{B}_{\vec{\tau_x} u_x} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{B}_{\vec{\tau_y} u_y} \end{bmatrix} \tag{2.58}$$

$$\boldsymbol{B}_{\tau\sigma} = \begin{bmatrix} \boldsymbol{B}_{\vec{\tau_x}\sigma_x} & \boldsymbol{B}_{\vec{\tau_x}\sigma_y} & \boldsymbol{B}_{\vec{\tau_x}\sigma_{xy}} \\ \boldsymbol{B}_{\vec{\tau_y}\sigma_x} & \boldsymbol{B}_{\vec{\tau_y}\sigma_y} & \boldsymbol{B}_{\vec{\tau_y}\sigma_{xy}} \end{bmatrix} \tag{2.59}$$

$$\boldsymbol{B}_{\tau\omega} = \begin{bmatrix} \boldsymbol{B}_{\vec{\tau_x}\omega} \\ \boldsymbol{B}_{\vec{\tau_y}\omega} \end{bmatrix} \tag{2.60}$$

$$\boldsymbol{B}_{v\sigma} = \begin{bmatrix} \boldsymbol{B}_{v_x\sigma_x} & \mathbf{0} & \boldsymbol{B}_{v_x\sigma_{xy}} \\ \mathbf{0} & \boldsymbol{B}_{v_y\sigma_y} & \boldsymbol{B}_{v_y\sigma_{xy}} \end{bmatrix}. \tag{2.61}$$

For the case of plane stress elasticity we have

$$\begin{aligned} [B_{\vec{\tau_x} u_x}]_{ij} &= (\phi_j^{u_x}, \mathrm{div}\rho_i^{\vec{\tau_x}})_T\,, \\ [B_{\vec{\tau_y} u_y}]_{ij} &= (\phi_j^{u_y}, \mathrm{div}\rho_i^{\vec{\tau_y}})_T\,. \end{aligned} \tag{2.62}$$

$$
\begin{aligned}
[B_{\overrightarrow{\tau_x}\,\sigma_x}]_{ij} &= (c3\phi_j^{\sigma_x}, \rho_i^{\overrightarrow{\tau_x}1})_T\,,\\
[B_{\overrightarrow{\tau_x}\,\sigma_y}]_{ij} &= -(c2\phi_j^{\sigma_y}, \rho_i^{\overrightarrow{\tau_x}1})_T\,,\\
[B_{\overrightarrow{\tau_x}\,\sigma_{xy}}]_{ij} &= (c1\phi_j^{\sigma_{xy}}, \rho_i^{\overrightarrow{\tau_x}2})_T\,,\\
[B_{\overrightarrow{\tau_y}\,\sigma_x}]_{ij} &= -(c2\phi_j^{\sigma_x}, \rho_i^{\overrightarrow{\tau_y}2})_T\,,\\
[B_{\overrightarrow{\tau_y}\,\sigma_y}]_{ij} &= (c3\phi_j^{\sigma_y}, \rho_i^{\overrightarrow{\tau_y}2})_T\,,\\
[B_{\overrightarrow{\tau_y}\,\sigma_{xy}}]_{ij} &= (c1\phi_j^{\sigma_{xy}}, \rho_i^{\overrightarrow{\tau_y}1})_T\,,
\end{aligned}
\tag{2.63}
$$

where $c1 = \frac{1}{2\mu}$, $c2 = \frac{\lambda}{2\mu(3\lambda+2\mu)}$, $c3 = \frac{\lambda+\mu}{\mu(3\lambda+2\mu)}$. Also,

$$
\begin{aligned}
[B_{\overrightarrow{\tau_x}\,\omega}]_{ij} &= (\phi_j^{\omega}, \rho_i^{\overrightarrow{\tau_x}2})_T\,,\\
[B_{\overrightarrow{\tau_y}\,\omega}]_{ij} &= -(\phi_j^{\omega}, \rho_i^{\overrightarrow{\tau_y}1})_T\,.
\end{aligned}
\tag{2.64}
$$

$$
\begin{aligned}
[B_{v_x\sigma_x}]_{ij} &= (\phi_j^{\sigma_x}, \nabla_1\rho_i^{v_x})_T\,,\\
[B_{v_x\sigma_{xy}}]_{ij} &= (\phi_j^{\sigma_{xy}}, \nabla_2\rho_i^{v_x})_T\,,\\
[B_{v_y\sigma_y}]_{ij} &= (\phi_j^{\sigma_y}, \nabla_2\rho_i^{v_y})_T\,,\\
[B_{v_y\sigma_{xy}}]_{ij} &= (\phi_j^{\sigma_{xy}}, \nabla_1\rho_i^{v_y})_T\,,.
\end{aligned}
\tag{2.65}
$$

The rest of block matrices are defined as follows

$$
\boldsymbol{B}_{\tau\hat{u}} = \begin{bmatrix} \boldsymbol{B}_{\overrightarrow{\tau_x}\,\hat{u}_x} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{B}_{\overrightarrow{\tau_y}\,\hat{u}_y} \end{bmatrix}
\tag{2.66}
$$

$$
\boldsymbol{B}_{v\hat{t}} = \begin{bmatrix} \boldsymbol{B}_{v_x\hat{t}_x} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{B}_{v_y\hat{t}_y} \end{bmatrix}
\tag{2.67}
$$

where the components of the block matrices are computed by evaluating the integrals on the edges of each element and assembling the values into the element block matrix:

$$
\begin{aligned}
[B_{\vec{\tau_x}\,\hat{u}_x}]_{ij} &= \sum_{assemble} -\langle \hat{\phi}_j^{\hat{u}_x}, \rho_i^{\vec{\tau_x}}\, n_e \rangle_{e\in\partial T}\,,\\
[B_{\vec{\tau_y}\,\hat{u}_y}]_{ij} &= \sum_{assemble} -\langle \hat{\phi}_j^{\hat{u}_y}, \rho_i^{\vec{\tau_y}}\, n_e \rangle_{e\in\partial T}\,,\\
[B_{v_x\hat{t}_x}]_{ij} &= \sum_{assemble} -\langle \hat{\phi}_j^{\hat{t}_x}, \rho_i^{v_x} \rangle_{e\in\partial T}\\
[B_{v_y\hat{t}_y}]_{ij} &= \sum_{assemble} -\langle \hat{\phi}_j^{\hat{t}_y}, \rho_i^{v_y} \rangle_{e\in\partial T}\,.
\end{aligned}
\tag{2.68}
$$

The functions $\phi^{u_x}, \phi^{u_y}, \phi^{\sigma_x}, \phi^{\sigma_y}, \phi^{\sigma_{xy}}, \phi^{\omega}$ are the basis functions discretizing the components of the u vector and the σ tensor and the only component necessary to represent the two dimensional ω tensor, respectively. Also, the functions $\hat{\phi}^{\hat{u}_x}, \hat{\phi}^{\hat{u}_y}, \hat{\phi}^{\hat{t}_x}, \hat{\phi}^{\hat{t}_y}, \rho^{v_x}, \rho^{v_y}$ discretize the components of $\hat{u}$, $\hat{t}$, and v, respectively. Lastly, $\rho^{\vec{\tau_x}}$, and $\rho^{\vec{\tau_y}}$ discretize the first and second row of the τ tensor. The specific basis functions used in our computations are the hierarchical basis functions defined in [FKDN15] which were used in the PolyDPG framework [AFMD18].

- The Gram (norm) matrix, $[G]^{(el)}$

The symmetric, positive-definite Gram matrix or the norm matrix for a polygonal element T depends on the inner product on the space V given in 2.39. So, we have

$$
[G]^{(el)} = \begin{bmatrix} \boldsymbol{G}_{\tau\tau} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{G}_{vv} \end{bmatrix}, \tag{2.69}
$$

where

$$
\boldsymbol{G}_{\tau\tau} = \begin{bmatrix} \boldsymbol{G}_{\vec{\tau_x}\,\vec{\tau_x}} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{G}_{\vec{\tau_y}\,\vec{\tau_y}} \end{bmatrix}, \quad \boldsymbol{G}_{vv} = \begin{bmatrix} \boldsymbol{G}_{v_x v_x} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{G}_{v_y v_y} \end{bmatrix}. \tag{2.70}
$$

Therefore, we can write $[G]^{(el)}$ as

$$[G]^{(el)} = \begin{bmatrix} \boldsymbol{G}_{\vec{\tau_x}\vec{\tau_x}} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{G}_{\vec{\tau_y}\vec{\tau_y}} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \boldsymbol{G}_{v_x v_x} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{G}_{v_y v_y} \end{bmatrix}, \tag{2.71}$$

where

$$\begin{aligned} [G_{\vec{\tau_x}\vec{\tau_x}}]_{ij} &= (\rho_j^{\vec{\tau_x}}, \rho_i^{\vec{\tau_x}})_T + (\mathrm{div}\rho_j^{\vec{\tau_x}}, \mathrm{div}\rho_i^{\vec{\tau_x}})_T\,, \\ [G_{\vec{\tau_y}\vec{\tau_y}}]_{ij} &= (\rho_j^{\vec{\tau_y}}, \rho_i^{\vec{\tau_y}})_T + (\mathrm{div}\rho_j^{\vec{\tau_y}}, \mathrm{div}\rho_i^{\vec{\tau_y}})_T\,, \\ [G_{v_x v_x}]_{ij} &= (\rho_j^{v_x}, \rho_i^{v_x})_T + (\nabla\rho_j^{v_x}, \nabla\rho_i^{v_x})_T\,, \\ [G_{v_y v_y}]_{ij} &= (\rho_j^{v_y}, \rho_i^{v_y})_T + (\nabla\rho_j^{v_y}, \nabla\rho_i^{v_y})_T\,. \end{aligned} \tag{2.72}$$

All the terms in equations 2.62, 2.63, 2.64,2.65,2.68, and 2.72 are either integrals over the polygonal domain, T , or integrals on the edges of the polygon, ∂T . The polygon is divided into subtriangles and the integral over each subtriangle is transformed to an integral over the standard triangle. Gaussian quadrature is then used to compute the result. Similarly, for integrals on the edges, a transformation to a standard interval is used.

2.4 Numerical Examples

2.4.1 Example 1: The unit square

As the first example, we consider a problem with a manufactured solution on the unit square to show the convergence of the method. The exact solution for the displacement field is chosen to be $u_{x_{ex}} = u_{y_{ex}} := \sin(\pi x)\sin(\pi y)$ and the corresponding exact applied body force for the plane stress case is

$$f_{x_{ex}} = f_{y_{ex}} = \frac{\pi^2 \mu}{\lambda + 2\mu} \left((-3\lambda - 2\mu)\cos(\pi x)\cos(\pi y) + (5\lambda + 6\mu)\sin(\pi x)\sin(\pi y) \right) , \tag{2.73}$$

where λ and μ are Lame constants. The chosen displacement fields results in homogeneous boundary conditions. The assumed parameters are the Poisson's ratio of $\nu = 0.499$ and elasticity modulus of $E = 210$ MPa (so, $\lambda = 34953.3$ and $\mu = 70.0467$). The problem is solved on a polygonal mesh of 1024 (32×32) elements with $r = 2$ which is shown in figure 2.7. All the polygonal meshes in this manuscript are created using the PolyMesher package [TPPM12]. $\Delta r = 2$ is used to create the test space. The computed displacement and stress components are shown in figures 2.8 and 2.9, respectively

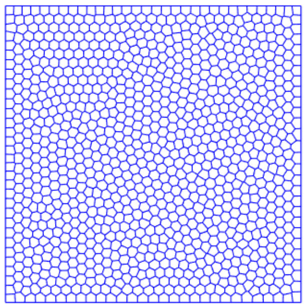
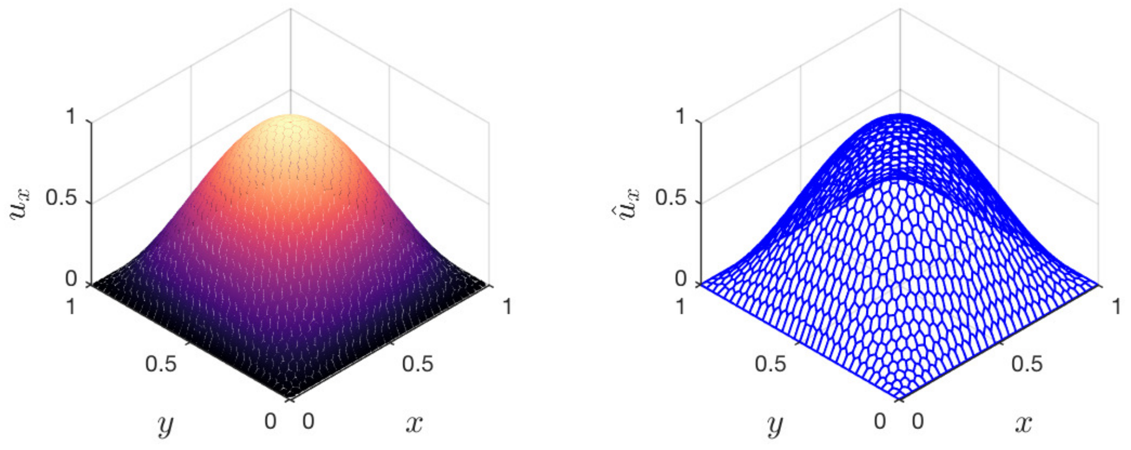


Figure 2.7: The mesh of 1024 polygonal elements

The relative errors in u, re_u and the relative error in σ, re_σ are calculated respectively as

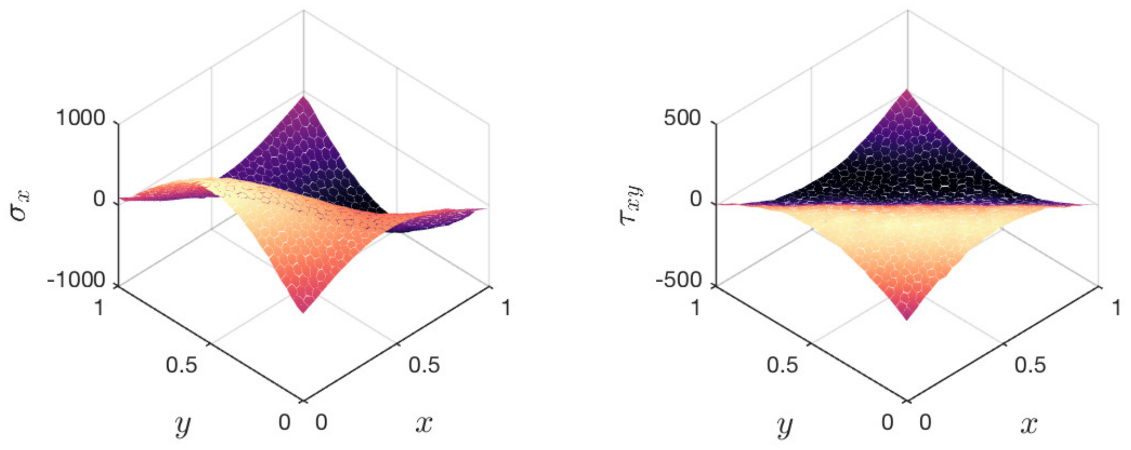
$$\begin{aligned} re_u &= \frac{\|u_{ex} - u_h\|_{\boldsymbol{L}^2(\Omega)}}{\|u_{ex}\|_{\boldsymbol{L}^2(\Omega)}} , \\ re_\sigma &= \frac{\|\sigma_{ex} - \sigma_h\|_{\boldsymbol{L}^2(\Omega)}}{\|\sigma_{ex}\|_{\boldsymbol{L}^2(\Omega)}} . \end{aligned} \tag{2.74}$$

The relative errors are plotted versus the number of degrees of freedom in figure 2.10. For both displacement and the stress the same rate of convergence of p is observed.



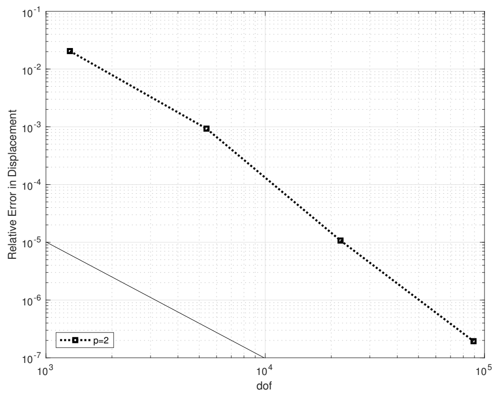
(a) The recovered domain variable, $u_x = u_y$ (b) The skeleton variable, $\hat{u}_x = \hat{u}_y$

Figure 2.8: The displacement field

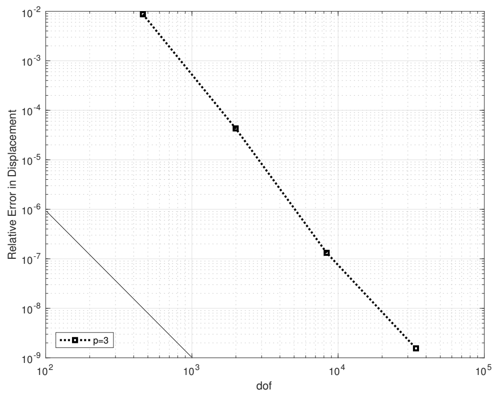


(a) The recovered domain variable, $\sigma_x = \sigma_y$ (b) The recovered domain variable, τ_{xy}

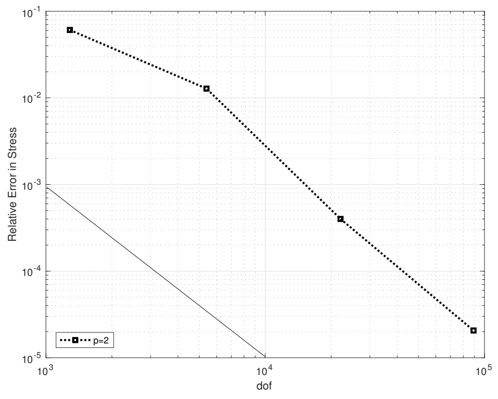
Figure 2.9: The stress field



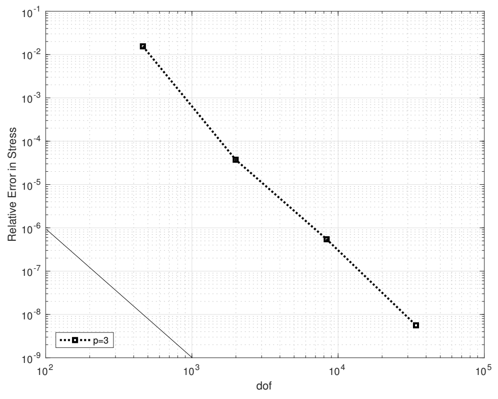
(a)



(b)



(c)



(d)

Figure 2.10: Relative error in the displacement and stress: (a) re_u for $r = 2$; (b) re_u for $r = 3$; (c) re_σ for $r = 2$; and, (d) re_σ for $r = 3$.

2.4.2 Example 2: The Cook's membrane

In the second example we look at the Cook's membrane problem which is a popular benchmark problem for modeling with nearly-incompressible materials. The geometry of the beam is shown in figure 2.11.

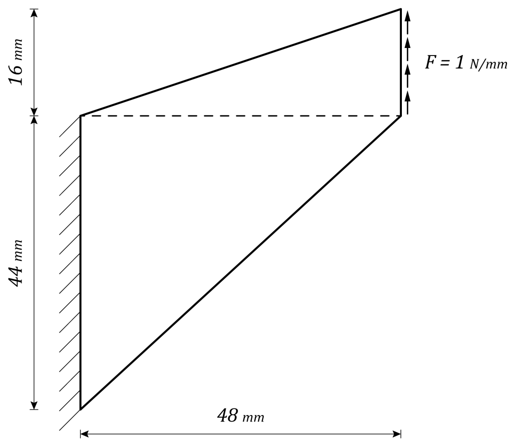


Figure 2.11: Cook's membrane

All the dimensions are in millimeters. The left edge is fully clamped and on the right edge an upward uniform force of 1 Newton per unit length (a total of 16 Newtons) is applied. The top and bottom edges are stress-free. The beam is made of a homogeneous material with material properties $nu = 0.499$ and $E = 210\ \frac{N}{mm^2}$. The deformed shape of the beam under the given load using a mesh of 150 polygons ($r = 2$ and $\Delta r = 2$) is shown in figure 2.12 (displacements are magnified by a factor of 5).

The stress components σ_x, τ_{xy}, and σ_y are plotted in figures 2.13, 2.14, and 2.15, respectively to show the combined shear and bending happening in the beam. The results are obtained using a mesh of 500 polygons. The method is able to correctly capture the correct solution without any checkerboarding or spurious oscillations in the stresses.

Also, the computed skeleton variables $\hat{u}_y$ and $\hat{u}_x$ and their recovered domain variable counterparts u_y and u_x are shown in figures 2.16, 2.18, 2.17, and 2.19, respectively.

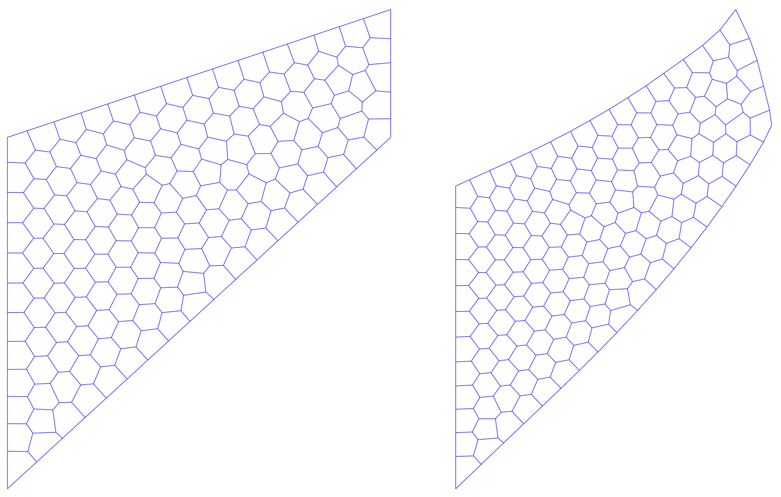


Figure 2.12: (a) The undeformed shape, (b) the deformed shape

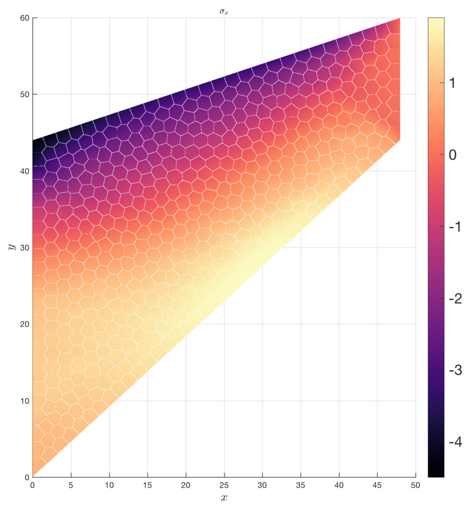


Figure 2.13: The normal stress σ_x

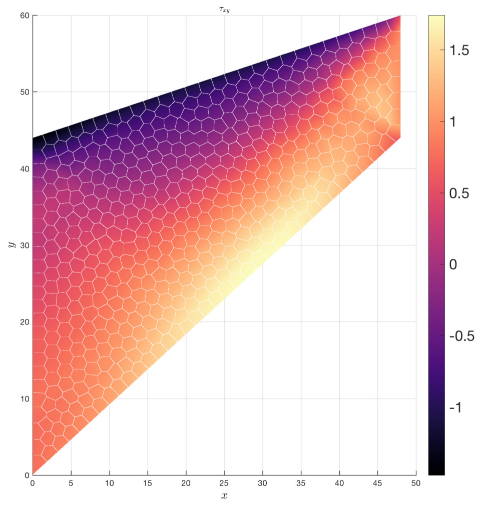


Figure 2.14: The shear stress τ_{xy}

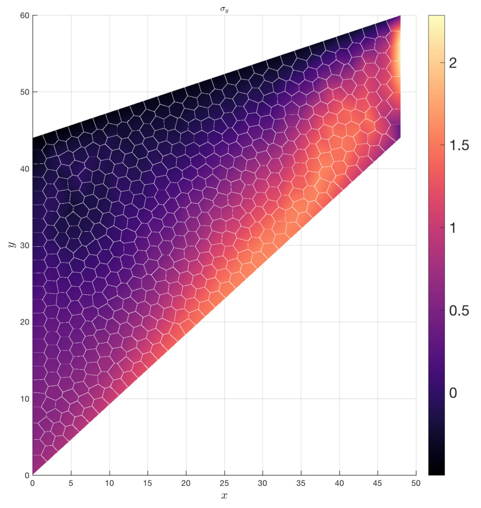


Figure 2.15: The normal stress σ_y

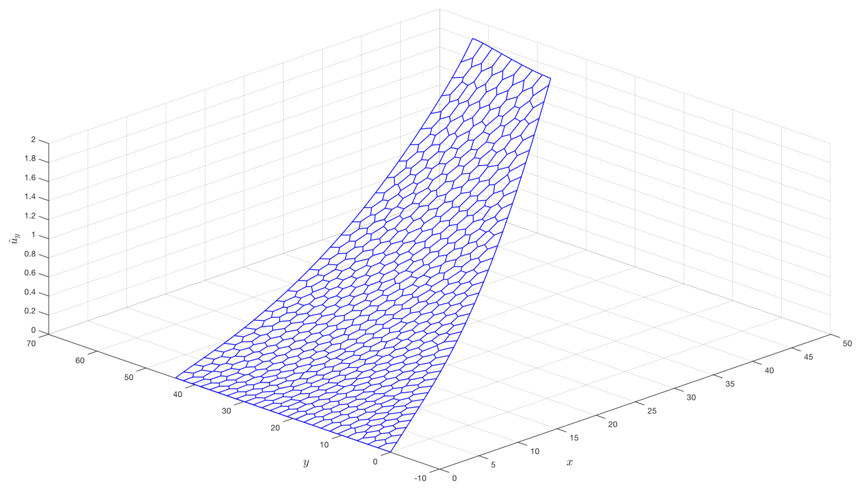


Figure 2.16: The skeleton variable $\hat{u}_y$

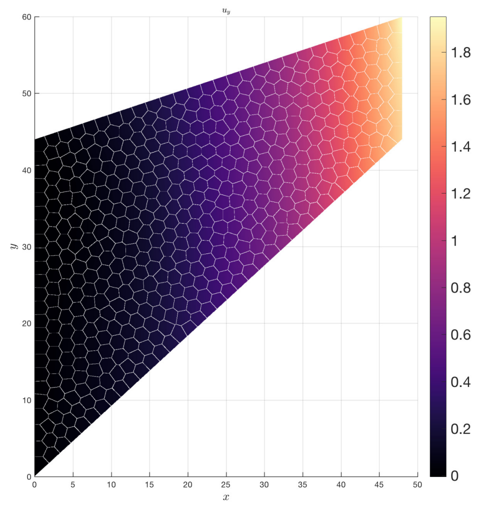


Figure 2.17: The domain variable u_y

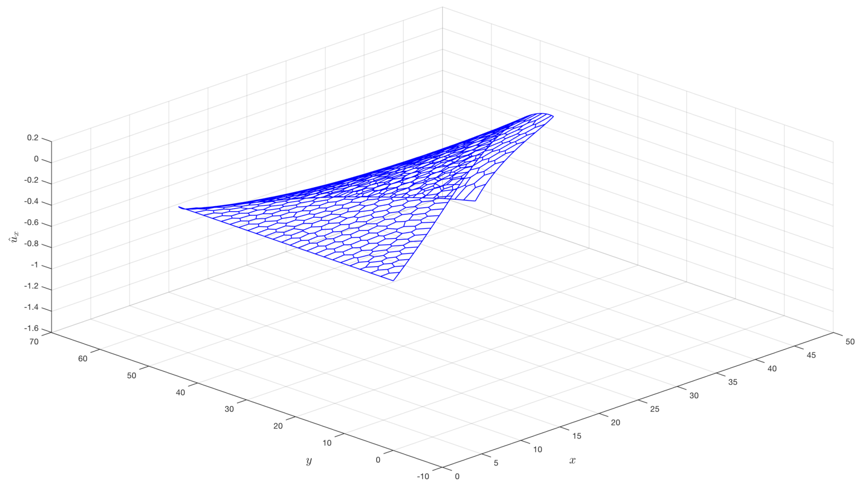


Figure 2.18: The skeleton variable $\hat{u}_x$

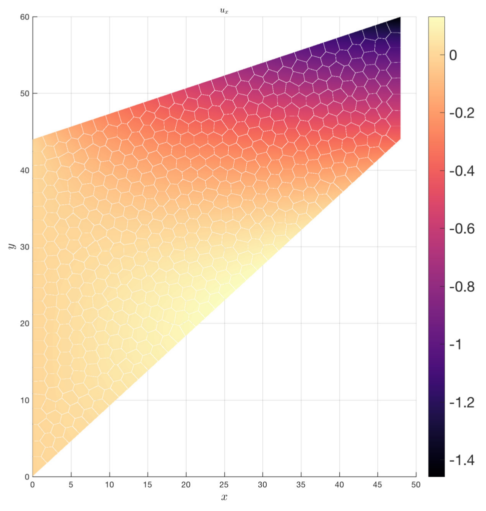


Figure 2.19: The domain variable u_x

Chapter 2, in part is currently being prepared for submission for publication of the material. Mirkhosravi, Poorya; Krysl, Petr. The dissertation author was the primary investigator and author of this material.

Chapter 3

Mesh generation

3.1 Introduction

With ever increasing interest in patient-specific simulations and in modeling all sorts of biomechanical processes we are more than ever in need of having an automatic mesh generation toolbox. The geometrical configuration is often captured via Computed Tomography (CT scan) or Magnetic Resonance Imaging (MRI) techniques. The produced 3D images are volumetric data sets in the form of stacks of 2D image slices. In order to do any numerical simulations like the finite element analysis we need to make a computational mesh out of the 3D image.

The common approach in image meshing starts with setting a threshold to implicitly define the bounding inner and outer iso-surfaces. The marching cube algorithm [LC87] is the widely used technique to extract these iso-surfaces. The algorithm goes through all the voxels and, based on the value of the volumetric data at the eight corners of the voxel, uses predefined templates to locally triangulate the voxel. The problem with the Marching Cube algorithm is that it often produces low quality elements and is only limited to uniform meshes. The other drawback is that the existing sharp features in the model are not preserved. The extended marching cube algorithm [KBSS01] and later the dual contouring method [JLSW02] were proposed to resolve these issues. The dual contouring method uses the intersection points and the normals at those points (the so-called Hermite data) and it can preserve the sharp features. In the next step, these extracted iso-surfaces serve as the boundary data to methods like advancing front techniques, octree-based methods and Delaunay mesh generation algorithms to fill the interior space between the iso-surfaces. The last step is quality improvement and surface smoothing [ZBX09]. Zhang et al. [ZBS05] uses dual contouring and the octree technique to produce adaptive mesh with high quality elements.

Most of the research in the past was focused on generating tetrahedral meshes out of the biomedical data sets. Generally, hexahedral meshes are harder to produce and still a robust and automatic hexahedral mesh generator is lacking. All-hex meshes are more accurate and in similar situations, mesh the domain with a smaller number of elements in comparison with tetrahedra. Because of the complexity of the geometries in biomedical images the meshes will be unstructured. The unstructured hexahedral meshing is reviewed in [Owe98, She07] and is classified as indirect and direct methods. In indirect methods, first a tetrahedral mesh of the domain is created and then each tetrahedron will be divided to several hexahedra [Epp96]. Four important categories of direct methods are

- The grid-based method,
- The medial surface method,
- The plastering method,
- The whisker weaving.

The grid-based method was proposed by Schneiders [Sch96] in 1996. It first fills the interior volume with a regular grid and in the next step, meshes the boundary of the volume. The result has high quality elements in the interior and elements with only acceptable quality on the boundary. The algorithm produces uniform meshes and there is no control on mesh density. To alleviate this problem and allow for different sizes of elements, the method was enhanced using an octree-based technique [SSW96, Sch97]. In medial surface method [PAS95], the medial axis (for 2D meshes) or the medial surface (in 3D cases) which is a geometrical property of the domain is used to decompose the domain into easily meshable subdomains. These subdomains are chosen from thirteen predefined polyhedra templates. At first it was proposed for convex domains but later it was improved to consider concave edges [PA97]. Plastering [Can91, BM93] is of advancing front type and it starts from a quadrilateral mesh of the boundary and goes into the interior space layer by layer. Seams and wedges are used to resolve the incompatibilities in the mesh during the layer addition. The best quality elements are generated on the boundary and it drops as we go more inside the domain. The whisker weaving algorithm [TBM96] is based on the concept of Spatial Twist Continuum (STC) [MB95, MBBM97]. STC is the dual of the hexahedral mesh. The algorithm starts with a quadrilateral mesh of the boundary and builds the element connectivities as it goes inside the domain. After that it finds the vertex locations. The whisker weaving algorithm is more focused on topology than on geometry and needs improvement on the robustness.

In our image meshing algorithm, we do not extract the iso-surface. Instead, we work on the segmented image itself. The mesh generation and the coarsening are done at the same time and in an automatic way, the important regions are kept less coarsened. An all-hex mesh with no hanging nodes is produced with a totally heuristic approach. Minimum interaction with the user is required. The user just needs to specify the coarsening factor and the important regions of the domain.

3.2 Proposed magnification map approach to obtain graded meshes

The micro-CT scans are stored as AVW images which are composed of 3D cubic voxels. Basically, the images are stored as 3D matrices of density values. Therefore, every voxel has a location and a density value. The proposed magnification map approach works in several steps which are shown in Figure 3.1 and are described in the following. At first, the 3D voxel set (AVW image) will be directly converted to a hexahedral mesh. The result of this straightforward conversion is a uniform structured cubic mesh. Next, we identify our region of interest in the domain. This region will have finer elements than the other parts of the domain when we go through all the steps. The assumed shape for this region is a sphere which is specified by the coordinates of the center (x_c, y_c, and z_c) and its radius (r_0). Then the region is stretched uniformly in all directions with a stretch ratio of λ. The user selects these parameters based on the location and size of the small geometrical features in the domain. The material points inside the sphere experience a radial displacement which is zero at the center and is linearly increasing toward the surface of the region. All the points outside the spherical region have a constant radial displacement equal to the displacement on the surface of the sphere. This step is called “Magnification”. Obviously the resulting mesh is not uniform anymore. The next step is the “Projection” step. In this step we create a new uniform grid with the same resolution as the original grid before “Magnification” but only bigger to be able to contain the magnified mesh. For convenience, the grid lines of this new grid and the original grid are aligned. Then we project the magnified mesh onto this new structured grid. To do so we look at the centroid of each element in the structured mesh and if the corresponding point in the magnified mesh was “in”, that element will be labeled as “in”. The “Projection” algorithm is shown in more detail in Algorithm 1. Next comes the coarsening step with the help of the “`shrink-wrap`” algorithm. The way the `shrink-wrap` algorithm works is discussed in section 3.3 . Next, we apply the inverse mapping to the material points to bring them back to their original position. We can think of this step as removing the magnifying glass that was held over part of the region. In the last step, we apply smoothing on the final non-uniform mesh to get a final smooth mesh that now has finer elements in the original spherical region and coarser elements in the rest of the domain. The final result of the whole process is producing a non-uniform mesh with finer elements in the areas with small but important geometrical details.

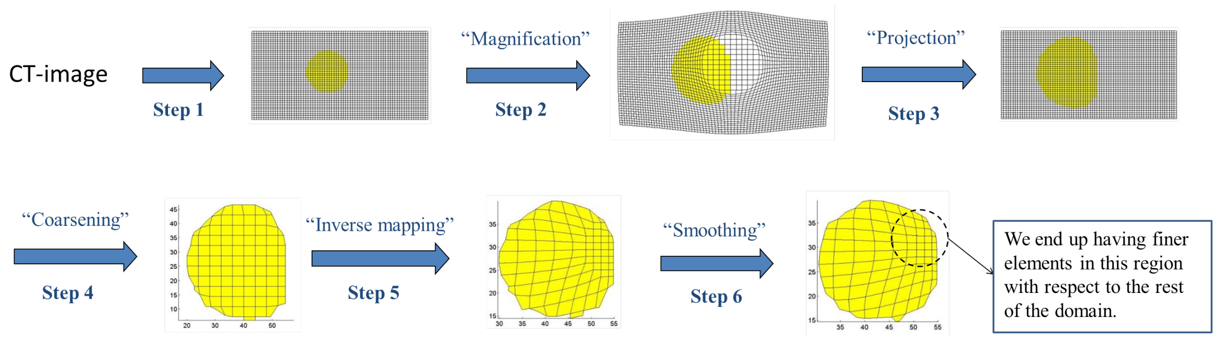


Figure 3.1: The flow of the algorithm

Algorithm 1 The Projection Algorithm

```
1: Create a blank 3D image and initialize all the IDs of its voxels to zero.
Ensure: this new blank image is big enough to contain the magnified version of the original
    image.
2: for each voxel $(vox_i)$ in this image do
3:     Obtain the location $(p_x, p_y)$ of the centroid of the voxel $vox_i$.
4:     Apply the inverse map on the centroid to find its coordinates in the original image.
5:     Using this coordinate, find the indices $(i, j, k)$ of the voxel in the original image that
       contains this mapped point.
6:     The ID of $vox_i \leftarrow$ the ID of voxel indexed by $(i, j, k)$ in the original image.
7: end for
```

3.3 The shrink-wrap algorithm

The goal of this stage is to produce a mesh coarser than the original resolution of the given AVW image. The amount of coarsening is set by a given factor of coarsening which is called `Voxels_per_edge` (`vpe` for short) in the program. So a typical cubic element in the coarse mesh would be of the size of `vpe` $\times$ `vpe` $\times$ `vpe` voxels of the original AVW image. Using a given threshold we classify the voxels of the original AVW image as "in" and "out". The voxels labeled "in" define the shape that we want to represent using the coarse mesh. Now, based on the "in" and "out" labels we classify the collections of `vpe` $\times$ `vpe` $\times$ `vpe` voxels (coarse elements) into 3 types:

0. "completely out",

Algorithm 2 The Inverse Map

```
1:  function INVERSE MAP($p_x, p_y, c_x, c_y, r, u_r$)
    $c_x, c_y$ are coordinates of the center of the spherical region
    $r$ is the radius of the region and $u_r$ is the radial displacement on the surface of the region
2:      $d = \sqrt{(p_x - c_x)^2 + (p_y - c_y)^2}$
3:      if $d < r + u_r$ then
4:          $U_r = -(\frac{u_r}{r+u_r})d$
5:      else
6:          $U_r = -u_r$
7:      end if
8:      $p_x = p_x + \frac{p_x - c_x}{d} U_r$
9:      $p_y = p_y + \frac{p_y - c_y}{d} U_r$
10:     return $p_x, p_y$        ▷ The coordinates of the point after application of the inverse map
11: end function
```

1. “partially in/partially out”, and

2. “completely in”.

Figure 3.2 shows these three different cases in a 2D example.

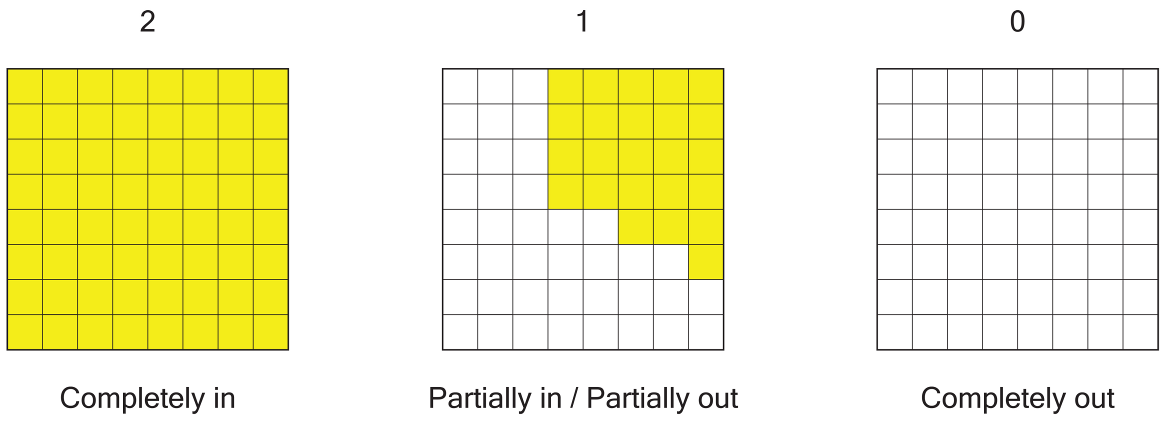


Figure 3.2: Different classes of coarse elements

In the next step we create a hexahedral coarse mesh such that each element corresponds to a pack of `vpe` $\times$ `vpe` $\times$ `vpe` voxels (refered to as `sub_volume` in the future). Elements corresponding to voxel collections with “0” label (completely out) will be discarded. Now, the `sub_volume` of each element has a label of “1” or “2”. The elements whose `sub_volume` has a label of “2” need no more modification and represent the correct shape which is a filled cube. But the elements with a `sub_volume` of label “1” need modification to conform to the actual shape represented by “in” voxels inside the `sub_volume` . The core of the algorithm is to shrink-wrap the sides of the coarse element around the “in” voxels. So for each element with a “partially in/partially out” `sub_volume` we try to move the vertices of the element closer to the “in” voxels. The `sub_volume` is a 3D matrix filled with zeros and ones, zero for voxels with “out” label and one for the “in” voxels. To decide which vertices to be moved we classify the vertices based on the value of the voxels located at the corners of the `sub_volume` . The vertices corresponding to corner voxels with a value of “1” (“in” voxel) are considered as “in”, hence will be fixed. The vertices corresponding to corner voxels with a value of “0” (“out” voxel) are considered as “out” and will be marked to move in later steps of the algorithm. Next, we look at the voxels on the edges of the `sub_volume` . If all the voxels on an edge are “out” (zero entries in the binary matrix) that edge will be considered as “out”, otherwise it will be labeled as “in”. We do not distinguish between “in” and “partially in/partially out” in this case.

Now that all the corners (vertices) and edges are classified we define three different moving mechanisms for the vertices:

- `edge_pull`,
- `face_pull`, and
- `volume_pull`.

These three mechanisms can only be applied on “out” vertices and they have no effect on “in” (fixed) vertices. We start by describing the `edge_pull` mechanism. Each vertex belongs to 3 edges, one in each Cartesian direction. We look at the x-direction and if the edge in this direction was not completely “out” the vertex will be moved in this direction until it meets an “in” voxel in the `sub_volume` . If this operation was successful the new position would be saved and returned. Otherwise, we repeat this process in y- and then z-directions. If none of these 3 attempts were successful it will be reported that the `edge_pull` was not successful. In `face_pull` mechanism, we look at the three intersecting faces at the vertex. Starting with yz-face, if the vertex on the opposite corner of this face is “in” (fixed) then we start moving the vertex towards this fixed vertex on the diagonal direction of the yz-face until it meets an “in” voxel on its way. The location where the vertex meets the “in” voxel will be saved and returned. If the vertex on the opposite corner of the yz-face was not fixed, we try xz- and xy-faces. If none of them were fixed the `face_pull` is unsuccessful. The last mechanism is the `volume_pull`. To `volume_pull` a vertex we look at the vertex at the opposite corner of the `sub_volume` (not any faces) and if this vertex was fixed we start moving the vertex on a straight line towards this fixed vertex until it reaches an “in” voxel somewhere inside the `sub_volume` . This location will be the result of the `volume_pull`. If the opposite vertex was not fixed the `volume_pull` is unsuccessful. These three mechanisms for moving the vertices are shown in Figure 3.3.

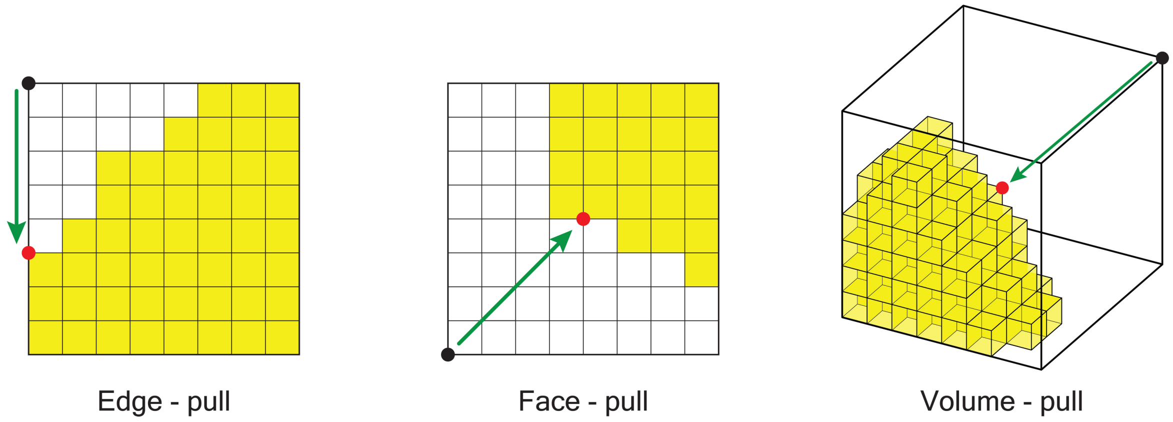


Figure 3.3: Three moving mechanisms acting on vertices

To modify a “partially in/partially out” element using these three mechanisms we look at each of the eight vertices of the element and if it should be moved (“out” vertex) we try to move it by applying the `edge_pull`. Again, we go through the eight vertices and if any of them have to move and have not moved yet we try the `face_pull` on them. Similarly, we repeat the process by trying to apply `volume_pull`. After these three attempts, if all the vertices that have to be moved were moved then the shrink-wrap of this element is successful. In case of failure of the shrink-warp the element will be deleted. But, if it was successful the final position of the moved vertices will be recorded in correspondence to the global node numbers of these vertices. The final position of a vertex depends on the modification results of all the elements sharing that vertex. For simple geometries the final location can be considered as the mean of the contributions of the neighboring elements. But, as it will be shown later the situation is more subtle. The general structure of the `shrink-wrap` function is shown in Algorithm 3 (shown in figure 3.4).

3.4 Treatment of narrow gaps using space-divider algorithm

In Figure 3.5, the application of shrink-wrap on some 2D examples is shown. The averaging to find the final position of the vertex works in situations like Figure 3.5-a but in other shown configurations it leads to spurious new connections between closely located separate regions. In these cases we have to properly add new vertices (shown with the red color in the figure) into the mesh. To do so we need to identify the disconnected regions in the vicinity of the vertices (the vertices that are going to move). Applying the `space-divider` function on a vertex gives us information about separate regions close to the vertex in terms of two variables, `bs` and `disjoint`. `bs` is the element numbers associated to the 8 boxes of voxels (`sub_volume` s) around the vertex (coarse elements sharing this node) with the following condition: $\texttt{bs}(i) = 0$ if the i-th element out of the eight neighboring elements has been deleted due to getting a “0” label in the coarsening process or due to `shrink-wrap` algorithm being unsuccessful. $\texttt{bs}(i) = 0$ also if the vertex is located on the boundary of the domain and there are not eight surrounding non-empty boxes of voxels. Otherwise, $\texttt{bs}(i)$ contains the element number (`id`) listed in the connectivity matrix. `disjoint` is a cell array and its length shows the number of separated regions in the 8 `sub_volume`s around the vertex. $\texttt{disjoint}(i)$ contains the element numbers of the coarse elements in the i-th separate region. So `disjoint` contains all the necessary information to decide how many additional vertices are needed and their final positions in the mesh. This is how the

Algorithm 3 The Shrink-wrap function

```
1: Set the desired factor of coarsening, $vpe$
2: Set the threshold range.
   Hence, defining ‘in’ and ‘out’ voxels (yellow vs. white voxels in the figures).
3: Classify groups of $vpe^3$ voxels in three different categories:
   0 - ‘all out’
   1 - ‘partially in/partially out’
   2 - ‘all in’
4: Create a hexahedral coarse mesh aligned with groups with labels 1 and 2.
5: for each coarse element in this mesh do
6:     if the group of voxels inside the element has a label of 1 (partially in/partially out) then
7:         (locally) classify the 8 vertices of this coarse element as ‘in’ (fixed) or ‘out’ (to be
           moved) based on the nearby voxels inside the element.
8:         (locally) classify the 12 edges of this coarse element as ‘out’ or ‘partially in/partially
           out’ based on the nearby voxels inside the element.
9:         Try to modify the coarse element as following:
10:        for each vertex of this element do
11:            if it has to be moved and it has not been moved yet then
12:                Apply Edge-pull to this vertex
13:                if it was successful then
14:                    label the vertex as ‘moved’
15:                end if
16:            end if
17:        end for
18:        for each vertex of this element do
19:            if it has to be moved and it has not been moved yet then
20:                Apply Face-pull to this vertex
21:                if it was successful then
22:                    label the vertex as ‘moved’
23:                end if
24:            end if
25:        end for
26:        for each vertex of this element do
27:            if it has to be moved and it has not been moved yet then
28:                Apply Volume-pull to this vertex
29:                if it was successful then
30:                    label the vertex as ‘moved’
31:                end if
32:            end if
33:        end for
34:        if all the free vertices are successfully moved (successful modification) then
35:            store the new position of each vertex
36:        else
37:            remove this coarse element from the mesh
38:        end if
39:    end if
40: end for
41:                ▷ Compute the final global location of vertices. Discussed in next section
```

Figure 3.4: The Shrink-wrap function

`space-divider` is used to obtain the final location of vertices: After we tried the `shrink-wrap` algorithm on all of the “partially in/partially out” coarse elements in the mesh we start looking at vertices. Each vertex is shared by at most eight neighboring coarse elements and the final position of the vertex depends on the result of the application of the `shrink-wrap` algorithm on these 8 coarse elements. If the vertex has not moved in any of the 8 coarse elements its position remains unchanged in the modified mesh. But, if the vertex has moved in at least one of the 8 coarse elements the `space-divider` should be applied on this vertex to see if any additional vertices are needed or not. If the `space-divider` identified m separated regions, $m-1$ additional nodes will be added to match these regions. The final position of each of these additional nodes is the average of the positions predicted by the `shrink-wrap` for that vertex in elements that compose that separated region. At the end, we need to modify the connectivity matrix appropriately.

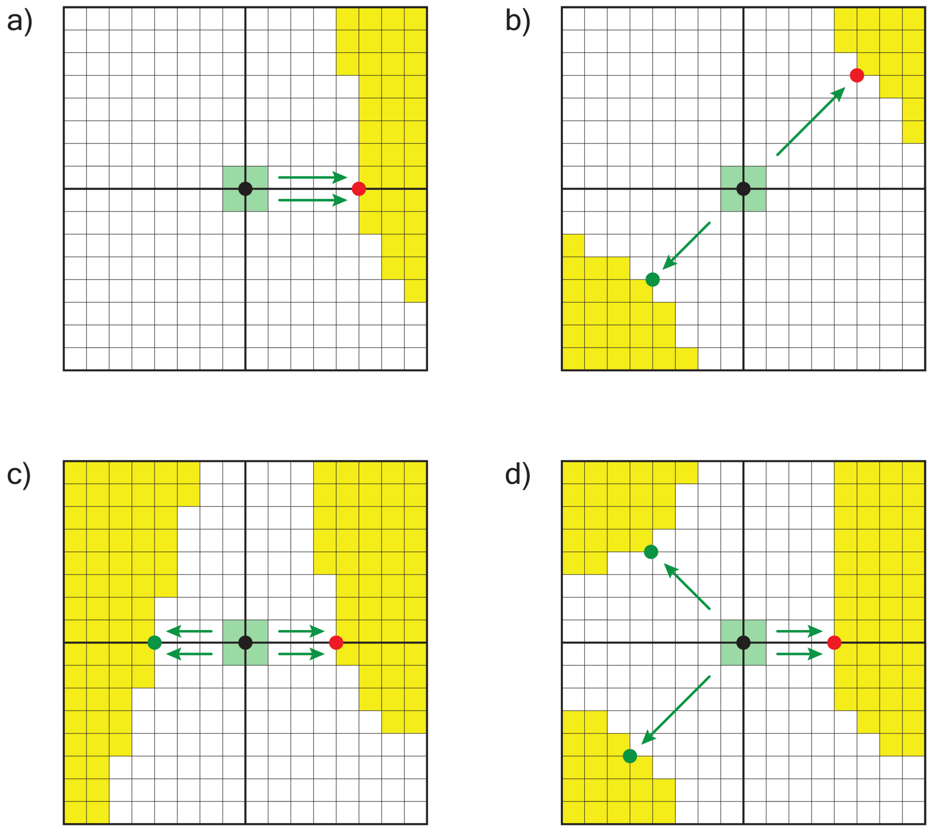


Figure 3.5: The application of the `space-divider` algorithm

Since the number of coarse elements are the same; so in the connectivity matrix, the old vertex numbers will be replaced with the new numbers of the additional nodes associated to coarse elements of a region.

3.4.1 The space-divider algorithm

This algorithm specifies how many disconnected regions are around a specific vertex and which coarse elements belong to each of these separated regions. In general, each vertex is shared by eight neighboring coarse elements. The coarse elements around a generic vertex are labeled as $b_1, b_2, \ldots, b_8$ in Figure 3.7. These coarse elements are building blocks of the disconnected regions. An example of having 2 disconnected regions is shown in Figure 3.8. The 2 regions are `region1`$= \{b_3, b_7\}$ and `region2`$= \{b_5\}$. This information is encoded in a variable called `nd_set`. `nd_set`(i) tells us to which region the i-th coarse element (b_i) belongs. So in this example `nd_set`$= [0,0,1,0,2,0,1,0]$. For a set of coarse elements to make a region, each coarse element should be sharing a face with at least one other coarse element in the set. The criteria for two regions to be disconnected is that the "in" voxels of the `sub_volumes` corresponding to the coarse elements of these regions should be totally separated (not connected through a face, or an edge, or a vertex) from each other. Using this criteria and the constraint on definition of a region, finding disconnected regions becomes a combinatorial problem. A number of different possible configurations which satisfy the constraint should be examined for satisfaction of the disconnectedness criteria to find the final answer. The pseudocode of the `space-divider` algorithm is shown in Algorithm 4 (shown in figure 3.6).

First, we talk about how to decide if a region is disconnected from the rest and then we see how to look at all the combinations. To be able to examine the eight neighboring elements, the variable `vox` is created by putting together the `sub_volumes` of these eight coarse elements to get a binary 3D matrix of dimension $(2\times$`vpe`$) \times (2\times$`vpe`$) \times (2\times$`vpe`$)$. Each of the eight coarse elements has 3 boundary faces (inner faces), so 24 faces in total which are shown in Figure 3.9.

Algorithm 4 The space-divider

```
1: function [DISJOINT, BS]=SPACE-DIVIDER($vtest, h, v, The\_grid, vpe, Deleteh$)
   $vtest$ is the considered vertex
   $h$ is the connectivity matrix and $v$ is the coordinates of the vertices
2:     [vox, empty_vox, bs] = adjacent_subvolume(vtest);
3:     [nd_set] = partition(vox, vpe);
4:     [nd_set, n_space] = remove_empty_boxes(nd_set);
5:     disjoint = cell(1,n_space);
6:     for i=1:n_space do
7:         disjoint{i} = bs(nd_set==i);
8:     end for
9:     return disjoint, bs
10: end function
```

Figure 3.6: The space-divider

Each face is a 2D binary matrix of dimension `vpe`$\times$`vpe`.

A face is labeled as "out" if all its entries are zeros ("out" voxels). For regions we start with the ones that contain only one coarse element like `reg1`$= \{b_1\}$. In this case we look at the 3 inner and 3 outer boundary faces of b_1 and if all of them are "out" then region `reg1` is considered as being separated from other coarse elements, hence the "out" label. For regions that are composed of more than one coarse element we consider the 3 inner and 3 outer boundary faces of all the coarse elements of the region. Then, we pick one of the coarse elements of the region at a time and compare its six faces to the faces of the rest of the coarse elements of the region in turn. If there was any intersection between faces of two elements those faces are not considered for decision on disconnectedness of the region. After comparing all the coarse elements of the region with each other, the remaining faces determine the label of the region. Figure 3.10 explains this procedure for the region `reg2`$= \{b_5, b_6, b_7\}$. This process is called `is_out` subfunction in the `space-divider` algorithm.

Now that we can figure out if a region is "out" or not, we start looking at different combinations of disconnected regions. $pat1, pat2, pat3$, and $pat4$ contain all the allowable regions composed of one, two, three, and four coarse elements, respectively:

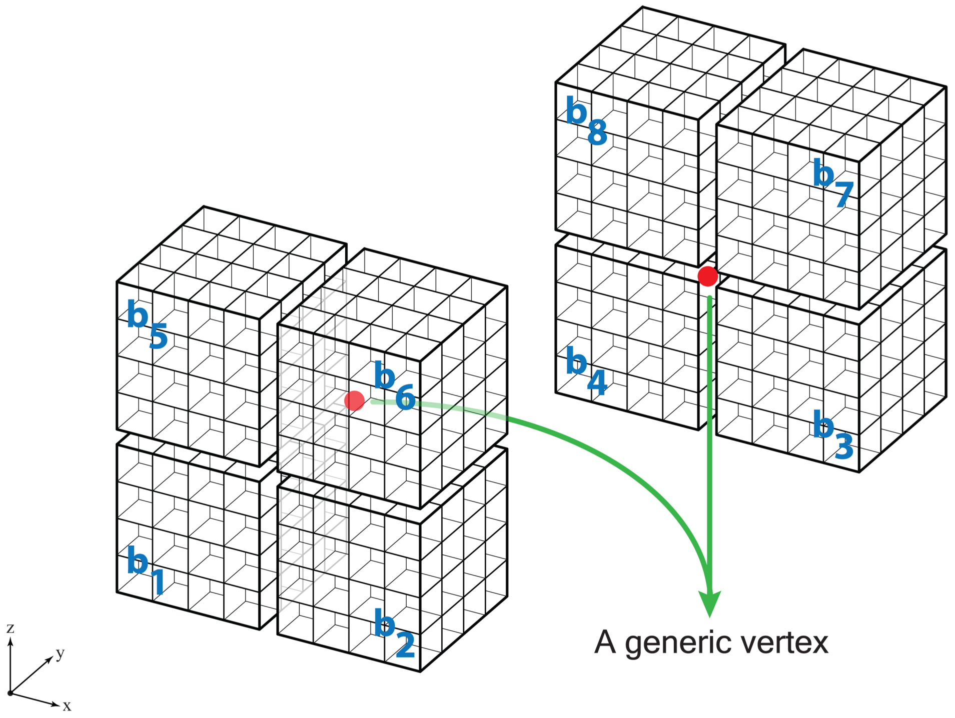


Figure 3.7: The eight coarse elements (boxes of voxels) around a generic vertex

$$\texttt{pat1} = \{\{1\},\{2\},\{3\},\{4\},\{5\},\{6\},\{7\},\{8\}\}$$

$$\texttt{pat2} = \{\{1,2\},\{1,4\},\{1,5\},\{2,3\}\{2,6\},\{3,4\},\{3,7\},\{4,8\},\{5,6\},\{5,8\},\{6,7\},\{7,8\}\}$$

$$\begin{aligned}\texttt{pat3} = \{&\{1,2,3\},\{1,2,4\},\{1,2,5\},\{1,2,6\},\{1,3,4\},\{1,4,5\},\{1,4,8\},\{1,5,6\},\{1,5,8\},\\ &\{2,3,4\},\{2,3,6\},\{2,3,7\},\{2,5,6\},\{2,6,7\},\{3,4,7\},\{3,4,8\},\{3,6,7\},\{3,7,8\},\\ &\{4,5,8\},\{4,7,8\},\{5,6,7\},\{5,6,8\},\{5,7,8\},\{6,7,8\}\}\end{aligned}$$

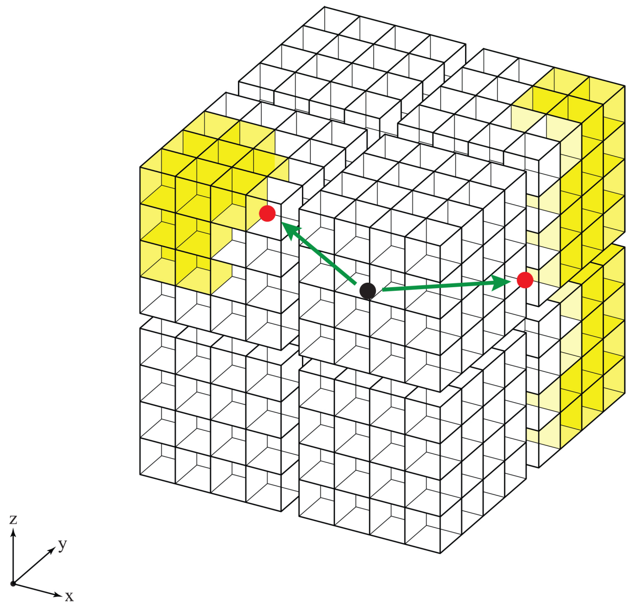


Figure 3.8: Two separate regions around a vertex (black) that needs an additional vertex (so, two red vertices)

$$\begin{aligned}\texttt{pat4} = \{&\{1,2,3,4\},\{5,6,7,8\},\{1,4,5,8\},\{2,3,6,7\},\{1,2,5,6\},\{3,4,7,8\},\\ &\{1,2,3,7\},\{1,2,4,6\},\{1,2,3,5\},\{1,2,6,7\},\{1,3,4,7\},\{1,3,4,5\},\\ &\{1,2,4,8\},\{1,4,5,6\},\{1,2,5,8\},\{2,3,4,8\},\{2,3,4,6\},\{2,3,7,8\},\\ &\{2,3,5,6\},\{2,6,7,8\},\{3,4,6,7\},\{3,4,5,8\},\{3,5,6,7\},\{3,5,7,8\},\\ &\{4,5,6,8\},\{1,4,7,8\},\{1,5,6,7\},\{2,5,6,8\},\{1,5,7,8\},\{4,6,7,8\},\\ &\{1,2,3,6\},\{1,2,4,5\},\{1,3,4,8\},\{1,5,6,8\},\\ &\{2,3,4,7\},\{2,5,6,7\},\{3,6,7,8\},\{4,5,7,8\}\}\end{aligned}$$

$$\texttt{patterns} = \texttt{pat1} \cup \texttt{pat2} \cup \texttt{pat3} \cup \texttt{pat4}$$

$$|\texttt{patterns}| = 8 + 12 + 24 + 38 = 82 \quad \text{(different patterns)}$$

We assume the 8 neighboring coarse elements to be connected at the beginning, hence `rembox`$= \{1,2,3,4,5,6,7,8\}$ showing the remaining elements. Then, we start going over these 82 different patterns starting with members of `pat1`. After applying the `is_out` algorithm on

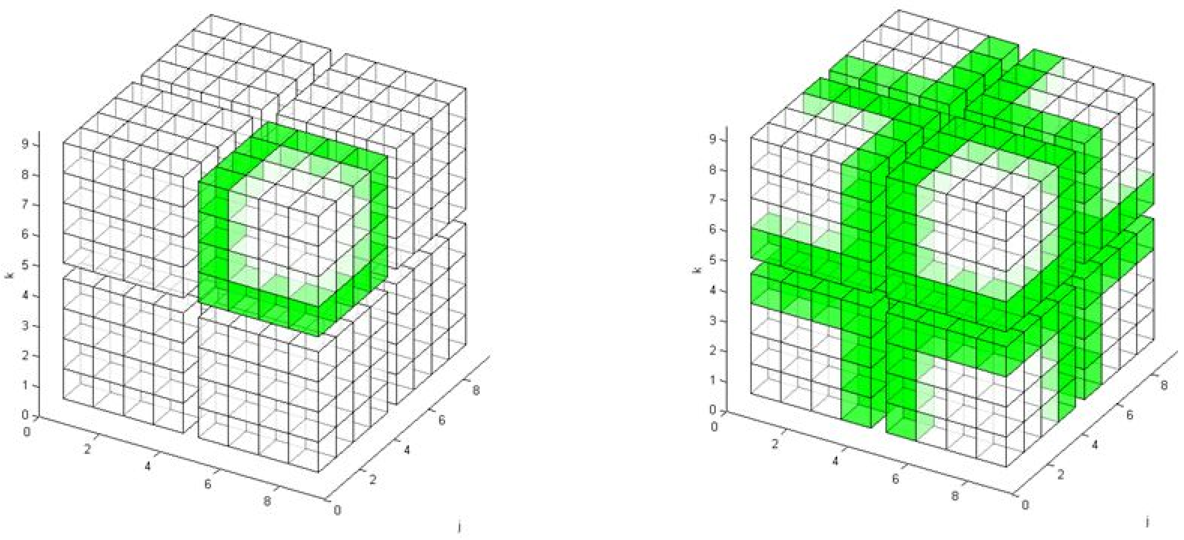


Figure 3.9: The 3 boundary faces of b_6 (Left), all the 24 faces (Right)

$b_1, b_2, \ldots, b_8$, if any of them ends up being disconnected, those elements will be removed from the `rembox`. Then, we look at patterns in `pat2`. If the elements of the pattern were present in `rembox` the `is_out` will be applied on them and in case of succession those elements also will be removed from the `rembox`. This process will continue with `pat3` and `pat4` until we go through all the patterns or the length of the `rembox` becomes smaller that the length of the patterns in `pat2`, `pat3` or `pat4`, whichever happens sooner. Note that there is no need to look at patterns with 5 or more elements in them because of the symmetry of `is_out` function. It means that there is no difference in result whether you feed the `is_out` function the $\{b_1, b_2, b_3\}$ region or its complement $\{b_4, b_5, b_6, b_7, b_8\}$. Another remark is that in the actual implementation some special configurations are looked at first to accelerate the process.

3.5 Results

In this section we look at two different examples of application of our algorithm on biomedical data sets. In the first example the focus is on the shrink-wrap and space-divider. So, the CT image is uniformly coarsened with a factor of 7. The initial resolution of the image is

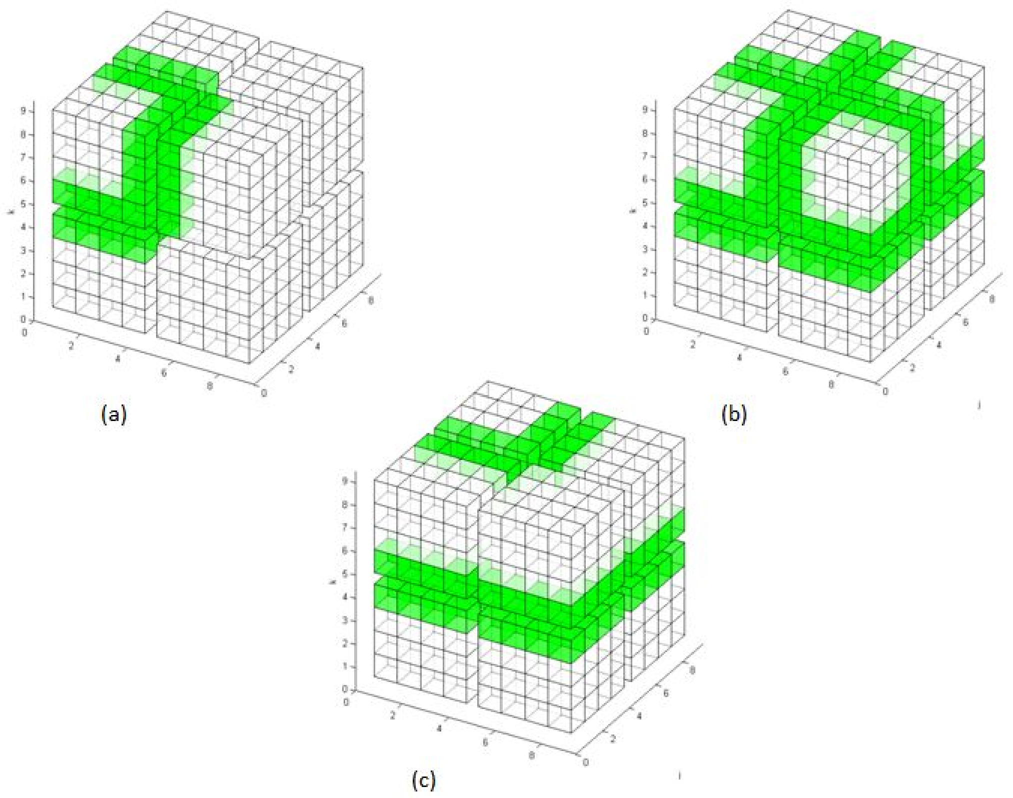


Figure 3.10: (a) The 3 inner and 3 outer boundary faces of b_6, (b) union of all the boundary faces of b_5, b_6, b_7, (c) boundary faces of $\{b_5, b_6, b_7\}$ after omitting the intersecting faces

$184 \times 157 \times 236$ voxels. The final mesh and a cut through the mesh are shown in Figure 3.11.

In the second example, the algorithm is applied on part of the tympanoperiotic complex of a bottlenose dolphin to make a finite element mesh of the ear bones. The input CT image has a resolution of $409 \times 282 \times 202$ voxels and a coarsening factor of 7 has been applied. The spherical region of interest is chosen to include part of the ossicular chain (Figure 3.12). We can see the less coarsened elements in that region and the smooth transition to coarser elements in Figure 3.13.

In application, our algorithm is used to obtain finite element meshes of the tympanoperiotic complex of bottlenose dolphins to do free vibration analysis (eigenvalue problem) and compute different vibration modes and frequencies.

To summarize, the important properties of the algorithm are the following:

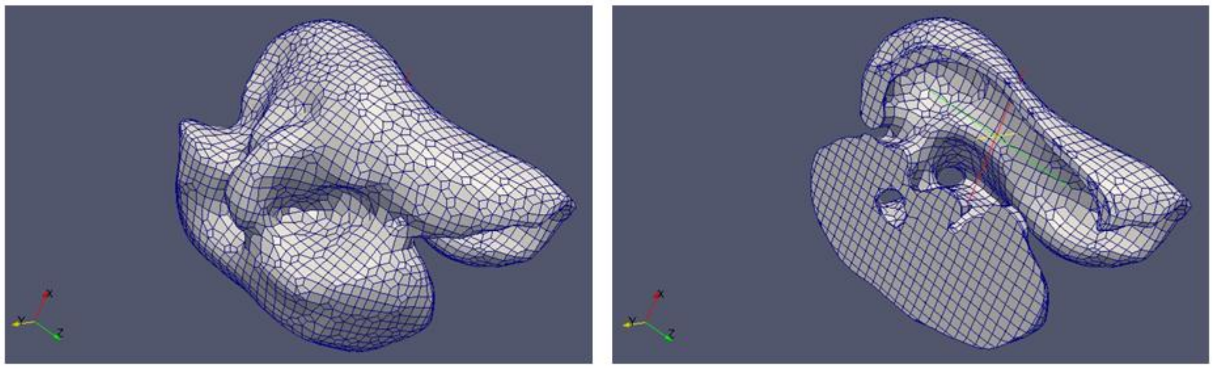


Figure 3.11: First example - The uniformly coarsened mesh (Left), A cut through the mesh (Right)

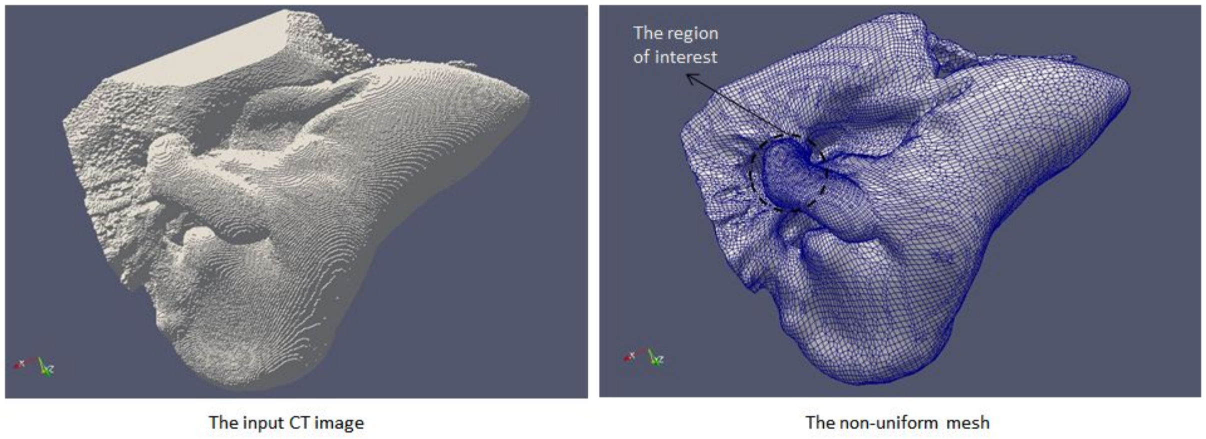


Figure 3.12: Second example - CT image of the ear bones (Left), The produced finite element mesh (Right)

- It creates an all-hex mesh.
- It doesn't introduce any hanging nodes.
- The produced mesh in non-uniform.
- It can preserve the correct topology (only up to a level).

Chapter 3, in part is currently being prepared for submission for publication of the material. Mirkhosravi, Poorya; Krysl, Petr. The dissertation author was the primary investigator and author of this material.

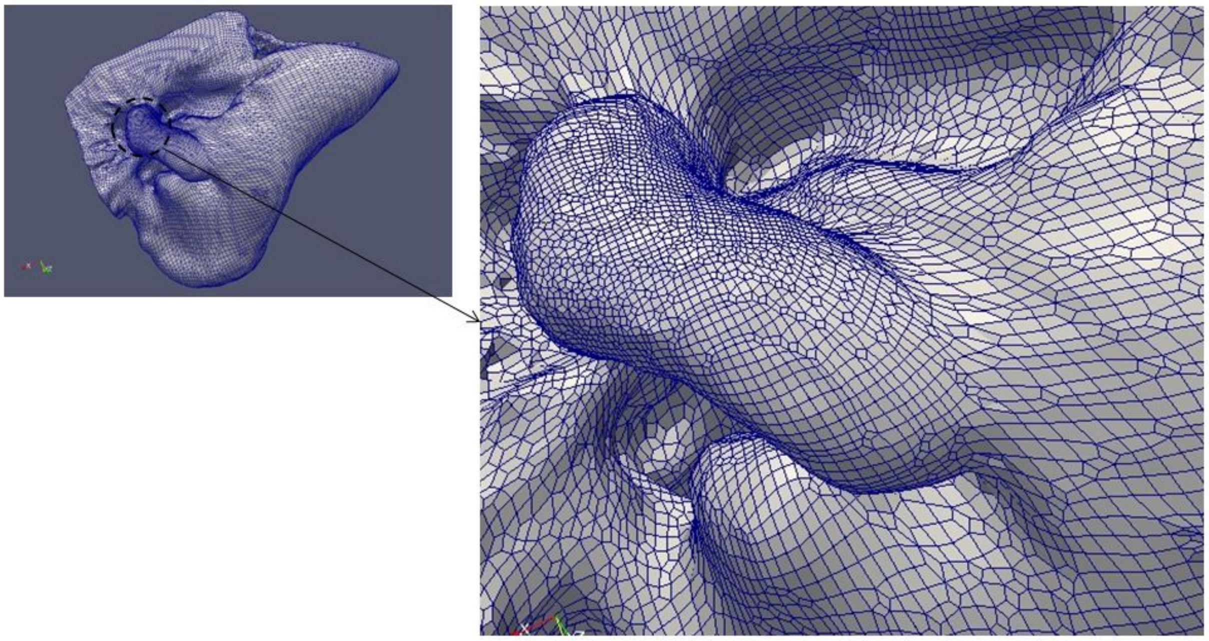


Figure 3.13: Second example - A better view of the spherical region of interest

Chapter 4

Summary and future direction

In the first part of the thesis (Chapter 3), the PolyDPG framework [AFMD18] is extended to linear elasticity problems. The discontinuous Petrov-Galerkin method is applied on the broken ultraweak formulation of linear elasticity. The ultraweak formulation lifted the C^0 continuity requirement on discretization of the trial variables. Its broken version went one step further and made it possible for test variables to be discontinuous across element boundaries with the expense of introducing some new variables on the skeleton of the mesh. However, at the end all the degrees of freedom associated with the domain variables were taken out of the system leading to a method with a reasonable computational cost. The method can be arbitrarily higher-order, works on general (convex or non-convex) polygons and adaptivity comes naturally because of the built-in error estimator of the dPG. The motivation for developing this method was to extend it to 3D so that it could be used alongside the meshing algorithm of the second part of the thesis (Chapter 4). This extension requires availability of arbitrarily higher-order basis functions (generalized barycentric coordinates) for general polygons. There are a few papers on quadratic GBCs on polygons. In [RGB14], Rand et al. showed how to use linear combinations of products of GBCs to construct 2nd-order basis functions on polygons. Also, Sukumar [Suk13] has extended his MAXENT framework to quadratic basis functions. However, there has been only one attempt at proposing higher-order GBCs by Mukherjee and Webb [MW15, MW16]. The robustness and scalability of their approach needs further investigation. In the second part of the thesis, we presented a heuristic algorithm that generates non-uniform hexahedral meshes with higher resolution close to selected regions in the domain of 3D micro-CT and micro-MR images. This algorithm takes as input a very fine micro-CT data set, the location of the regions containing delicate geometrical details, and the coarsening factor to produce a ready-to-use graded mesh. Goals for algorithm design were to reduce the user's interaction with the program itself, and to remain faithful during the coarsening step to the topology and geometry of the original problem. The algorithm can be hugely benefited by including Homology group based methods to further enhance its topology-preserving capabilities.

Appendix A

Example - How to calculate rational Wachspress functions

Following the derivation in [Das03], for a pentagon with the vertices given in table A, the Wachspress basis functions are computed symbolically using Mathematica.

Table A.1: The coordinates of the vertices of a pentagonal element

vertices	x coordinate	y coordinate
v_1	2	-1
v_2	1	0.5
v_3	-1	1
v_4	-3	-0.2
v_5	-1.8	-0.8

The main function is implemented as following:

```
computeWachspress[vertices_] := Module[
{xs, ys, $x, $y, xx, yy, a, b, aa, bb, k, Num, denum},
xs = vertices[[All, 1]];
ys = vertices[[All, 2]];
nside = Length[xs];
$x = Append[xs, xs[[1]]];
$y = Append[ys, ys[[1]]];
xx = Prepend[xs, xs[[nside]]];
yy = Prepend[ys, ys[[nside]]];
a = Table[(yy[[i + 1]] - yy[[i]])/(
xx[[i]] yy[[i + 1]] - xx[[i + 1]] yy[[i]]), {i, nside}];
b = Table[(xx[[i]] - xx[[i + 1]])/(
xx[[i]] yy[[i + 1]] - xx[[i + 1]] yy[[i]]), {i, nside}];
aa = Append[a, a[[1]]];
bb = Append[b, b[[1]]];
$L = Table[1 - a[[i]] x - b[[i]] y, {i, nside}];
k = Table[1, {i, nside}];
For[i = 2, i <= nside, i++,
k[[i]] =
k[[i - 1]] ((
aa[[i + 1]] ($x[[i - 1]] - $x[[i]]) +
bb[[i + 1]] ($y[[i - 1]] - $y[[i]]))/(
a[[i - 1]] ($x[[i]] - $x[[i - 1]]) +
b[[i - 1]] ($y[[i]] - $y[[i - 1]]))))];
Num = Table[
k[[i]] Product[$L[[1 + Mod[i + j, nside]]], {j, 1,
nside - 2}], {i, nside}]; denum = Expand[Apply[Plus, Num]];
Expand[Num]/denum]
```

Applying this function to the given vertex coordinates will give us the shape functions.

```
vertices = {{2,-1}, {1,1/2}, {-1,1}, {-3,-1/5}, {-9/5,-4/5}};
$N = computeWachspress[vertices]
```

The result is

$$N_1(x,y) = \frac{-\frac{5x^3}{136} - \frac{65x^2y}{408} - \frac{23x^2}{204} + \frac{5xy^2}{68} - \frac{43xy}{51} + \frac{137x}{408} + \frac{25y^3}{51} - \frac{65y^2}{204} - \frac{559y}{408} + 1}{-\frac{1039x^2}{7480} - \frac{4857xy}{7480} - \frac{713x}{22440} - \frac{3527y^2}{3740} - \frac{10147y}{22440} + \frac{622}{165}},$$
$$N_2(x,y) = \frac{\frac{5x^3}{748} + \frac{145x^2y}{1122} + \frac{173x^2}{1122} + \frac{15xy^2}{748} + \frac{603xy}{748} + \frac{3x}{4} - \frac{475y^3}{1122} - \frac{315y^2}{748} + \frac{49y}{44} + \frac{34}{33}}{-\frac{1039x^2}{7480} - \frac{4857xy}{7480} - \frac{713x}{22440} - \frac{3527y^2}{3740} - \frac{10147y}{22440} + \frac{622}{165}},$$
$$N_3(x,y) = \frac{-\frac{x^3}{88} - \frac{65x^2y}{264} - \frac{13x^2}{60} - \frac{13xy^2}{22} - \frac{421xy}{440} - \frac{153x}{440} - \frac{19y^3}{66} - \frac{113y^2}{660} + \frac{697y}{660} + \frac{289}{330}}{-\frac{1039x^2}{7480} - \frac{4857xy}{7480} - \frac{713x}{22440} - \frac{3527y^2}{3740} - \frac{10147y}{22440} + \frac{622}{165}},$$
$$N_4(x,y) = \frac{\frac{x^3}{136} + \frac{71x^2y}{408} + \frac{19x^2}{204} + \frac{137xy^2}{204} - \frac{31xy}{408} - \frac{209x}{408} + \frac{19y^3}{51} - \frac{47y^2}{68} - \frac{73y}{204} + \frac{1}{2}}{-\frac{1039x^2}{7480} - \frac{4857xy}{7480} - \frac{713x}{22440} - \frac{3527y^2}{3740} - \frac{10147y}{22440} + \frac{622}{165}},$$
$$N_5(x,y) = \frac{\frac{3x^3}{88} + \frac{9x^2y}{88} - \frac{5x^2}{88} - \frac{23xy^2}{132} + \frac{37xy}{88} - \frac{17x}{66} - \frac{5y^3}{33} + \frac{29y^2}{44} - \frac{59y}{66} + \frac{4}{11}}{-\frac{1039x^2}{7480} - \frac{4857xy}{7480} - \frac{713x}{22440} - \frac{3527y^2}{3740} - \frac{10147y}{22440} + \frac{622}{165}}.$$

The partition of unity property can be checked by executing the following code:

```
Simplify[Apply[Plus, $N]]
```

The result is $N_1(x,y)+N_2(x,y)+N_3(x,y)+N_4(x,y)+N_5(x,y) = 1$.

Also, the linear precision property is shown using the following code:

```
f[x_, y_] := alpha x + beta y + gamma

$f = f[vertices[[All, 1]], vertices[[All, 2]]]

Simplify[Apply[Plus, $f $N]]
```

f is an arbitrary linear function and is evaluated at all the vertices, $f_i = f(x_i,y_i)$. The result is $N_1(x,y)f_1 + N_2(x,y)f_2 + N_3(x,y)f_3 + N_4(x,y)f_4 + N_5(x,y)f_5 = f(x,y)$.

A three-dimensional version of figure 1.2 is produced using the following code snippet:

```
prel = Plot3D[0, {x, -3.5, 2.5}, {y, -2, 2},

RegionFunction ->

Function[{x,
y}, $L[[1]] >= 0 && $L[[2]] >= 0 && $L[[3]] >= 0 && $L[[4]] >=
0 && $L[[5]] >= 0], PlotRange -> {0, 1},
PlotLabel -> "The␣element", Mesh -> None,
BoundaryStyle -> Directive[Black, Thick],
ColorFunction -> Function[{x, y, z}, White]];

$p = Table[
Plot3D[$N[[i]], {x, -3.5, 2.5}, {y, -2, 2},
RegionFunction ->
Function[{x,
y}, $L[[1]] >= 0 && $L[[2]] >= 0 && $L[[3]] >= 0 && $L[[4]] >=
0 && $L[[5]] >= 0], PlotLabel -> "N" <> ToString[i],
PlotStyle -> Opacity[0.5], PlotPoints -> 75,
ColorFunction -> "StarryNightColors"], {i, nside}];

GraphicsGrid[{{prel, Show[$p[[1]], prel]}, {Show[$p[[2]], prel],
Show[$p[[3]], prel]}, {Show[$p[[4]], prel], Show[$p[[5]], prel]}},
Frame -> True, FrameStyle -> Directive[Black, Thick]]
```

The result is shown in figure A.1.

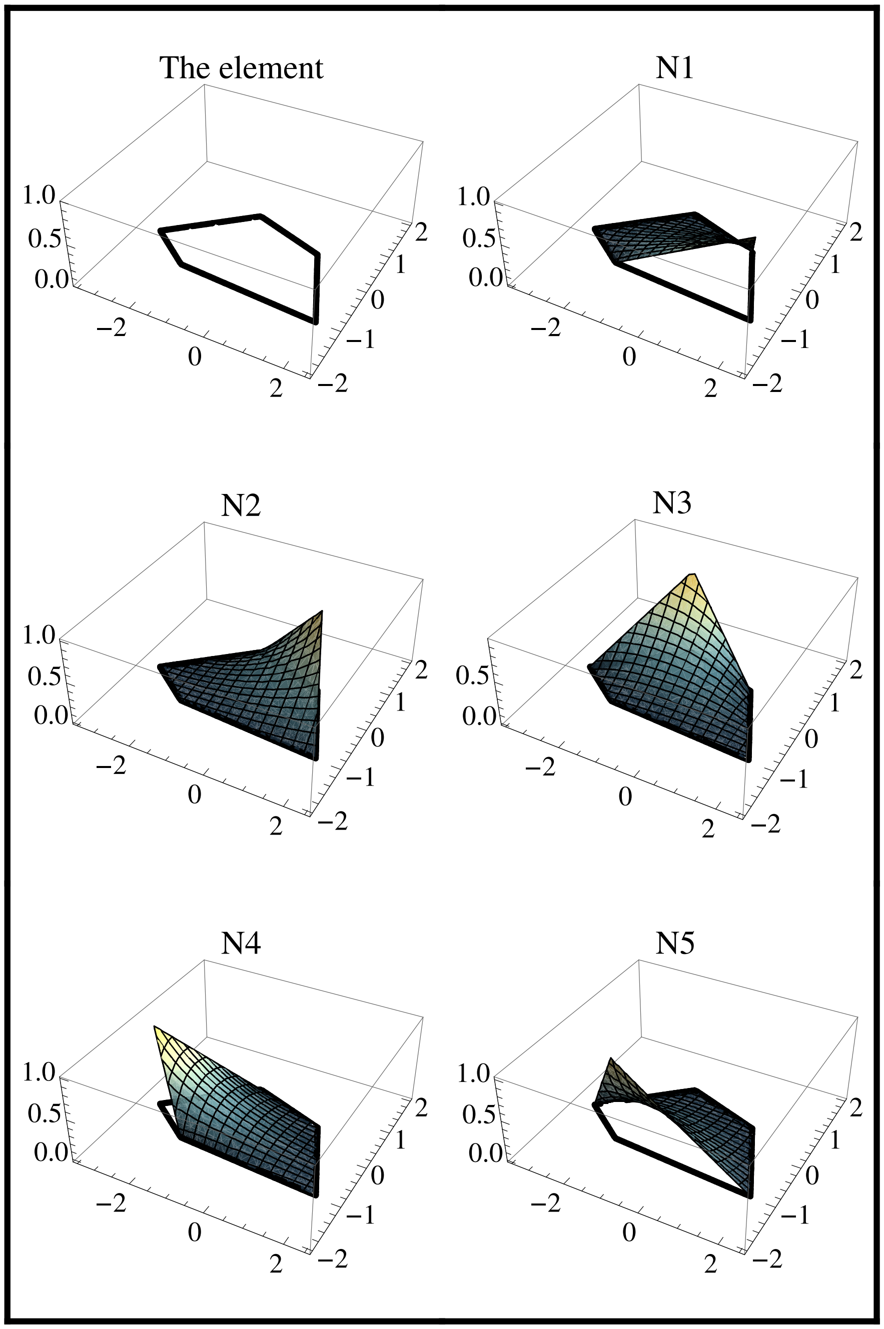


Figure A.1: A 3D plot of Wachspress basis functions

Appendix B

Example - Minimal $H(\mathrm{div})$ basis functions for a pentagon

The vector-valued $H(\mathrm{div})$-conforming basis functions for the pentagon in appendix A is computed using Mathematica,

```
vertices = {{2,-1},{1,1/2},{-1,1},{-3,-1/5},{-9/5,-4/5}};
$q = HdivBasis[vertices];
```

where the `HdivBasis` module is defined as follows

```
curl2d[N_] := {-D[N, y], D[N, x]}


HdivBasis[$v_] := Module[
{xs, ys, $x, $y, $e, $en, $t, $xstar, $d, $Tni, $Tn, $bm, $cm, \
$c0},
xs = $v[[All, 1]];
ys = $v[[All, 2]];
nside = Length[xs];
$x = Append[xs, xs[[1]]];
$y = Append[ys, ys[[1]]];
$N = computeWachspress[$v]; $e =
Table[{$x[[i + 1]]-$x[[i]], $y[[i + 1]]-$y[[i]]}, {i, nside}];
$en = Norm /@ $e;
$t = MapThread[Divide, {$e, $en}];
$xstar = {0, 0};
$n = Table[RotationMatrix[-Pi/2].$t[[i, All]], {i, nside}]; $d =
Table[Norm[
Cross[{$e[[i, 1]], $e[[i, 2]],
0}, {$xstar[[1]] - $v[[i, 1]], $xstar[[2]] - $v[[i, 2]],
0}]]/$en[[i]], {i, nside}]; $Tni =
MapThread[Times, {$en, $d}]/2;
$Tn = Apply[Plus, $Tni]; $bm =
Table[KroneckerDelta[i,j]*$en[[j]]-$en[[i]]*($Tni[[j]]/$Tn), {i,
nside}, {j, nside}]; $cm =
Table[Sum[(-1/nside) (k*$bm[[i, 1 + Mod[j+k-1, nside]]]), {k,
1, nside - 1}], {i, nside}, {j, nside}];
$c0 = $en/(2 $Tn);
Table[$c0[[i]] ({x, y} - $xstar) +
Sum[$cm[[i, j]] curl2d[$N[[j]]], {j, 1, nside}], {i, nside}]]
```

B.1 Kronecker-delta property

The objective is to show the following property for the computed $H(\text{div})$-conforming basis, $q_i, i = 1, \ldots, 5$,

$$q_i \cdot n_j = \delta_{ij} \qquad \text{for } i, j = 1, \ldots, 5,$$

where δ_{ij} is the Kronecker delta function.

- On the first edge connecting v_1 to v_2

```
Simplify[Dot[$q, $n[[1]]] /. {y -> 2 - (3 x)/2}]
```

  $\{1,0,0,0,0\}$

  So, it shows that $\{q_1 \cdot n_1, q_2 \cdot n_1, q_3 \cdot n_1, q_4 \cdot n_1, q_5 \cdot n_1\} = \{1,0,0,0,0\}$. Similarly, for the other edges we get:

- On the second edge connecting v_2 to v_3

```
Simplify[Dot[$q, $n[[2]]] /. {y -> -(1/4)*x + 3/4}]
```

  $\{0,1,0,0,0\}$

- On the third edge connecting v_3 to v_4

```
Simplify[Dot[$q, $n[[3]]] /. {y -> 8/5 + (3 x)/5}]
```

  $\{0,0,1,0,0\}$

- On the fourth edge connecting v_4 to v_5

```
Simplify[Dot[$q, $n[[4]]] /. {y -> -(17/10) - x/2}]
```

  $\{0,0,0,1,0\}$

- On the fifth edge connecting v_5 to v_1

```
Simplify[Dot[$q, $n[[5]]] /. {y -> -(1/19)*x - 17/19}]
```

  $\{0,0,0,0,1\}$

Bibliography

[ABD84] Douglas N Arnold, Franco Brezzi, and Jim Douglas. PEERS: a new mixed finite element for plane elasticity. *Japan Journal of Applied Mathematics*, 1(2):347, 1984.

[AFMD18] Ali Vaziri Astaneh, Federico Fuentes, Jaime Mora, and Leszek Demkowicz. High-order polygonal discontinuous Petrov–Galerkin (PolyDPG) methods using ultraweak formulations. *Computer Methods in Applied Mechanics and Engineering*, 332:686 – 711, 2018.

[AFW06] Douglas N Arnold, Richard S Falk, and Ragnar Winther. Finite element exterior calculus, homological techniques, and applications. *Acta numerica*, 15:1–155, 2006.

[AFW07] Douglas Arnold, Richard Falk, and Ragnar Winther. Mixed finite element methods for linear elasticity with weakly imposed symmetry. *Mathematics of Computation*, 76(260):1699–1723, 2007.

[AFW09] Douglas N Arnold, Richard S Falk, and Ragnar Winther. Geometric decompositions and local bases for spaces of finite element differential forms. *Computer Methods in Applied Mechanics and Engineering*, 198(21-26):1660–1672, 2009.

[AFW10] Douglas Arnold, Richard Falk, and Ragnar Winther. Finite element exterior calculus: from Hodge theory to numerical stability. *Bulletin of the American mathematical society*, 47(2):281–354, 2010.

[AL14] Douglas N Arnold and Anders Logg. Periodic table of the finite elements. *SIAM News*, 47(9):212, 2014.

[AO06] Marino Arroyo and Michael Ortiz. Local maximum-entropy approximation schemes: a seamless bridge between finite elements and meshfree methods. *International journal for numerical methods in engineering*, 65(13):2167–2202, 2006.

[BBF13] Daniele Boffi, Franco Brezzi, and Michel Fortin. *Mixed finite element methods and applications*, volume 44. Springer, 2013.

[BDGQ12] Jamie Bramwell, Leszek Demkowicz, Jay Gopalakrishnan, and Weifeng Qiu. A locking-free hp DPG method for linear elasticity with symmetric stresses. *Numerische Mathematik*, 122(4):671–707, 2012.

[BDM85] Franco Brezzi, Jim Douglas, and L Donatella Marini. Two families of mixed finite elements for second order elliptic problems. *Numerische Mathematik*, 47(2):217–235, 1985.

[BM93] Ted D Blacker and Ray J Meyers. Seams and wedges in plastering: a 3–D hexahedral mesh generation algorithm. *Engineering with computers*, 9(2):83–93, 1993.

[Can91] Scott Alan Canann. *Plastering and optismoothing: new approaches to automated, 3–D hexahedral mesh generation and mesh smoothing*. PhD thesis, Brigham Young University. Department of Civil Engineering., 1991.

[CD98] Olivier Cessenat and Bruno Despres. Application of an ultra weak variational formulation of elliptic PDEs to the two–dimensional Helmholtz problem. *SIAM journal on numerical analysis*, 35(1):255–299, 1998.

[CDG16] Carsten Carstensen, Leszek Demkowicz, and Jay Gopalakrishnan. Breaking spaces and forms for the DPG method and applications including Maxwell equations. *Computers & Mathematics with Applications*, 72(3):494–522, 2016.

[CH16] Carsten Carstensen and Friederike Hellwig. Low-order discontinuous Petrov–Galerkin finite element methods for linear elasticity. *SIAM Journal on Numerical Analysis*, 54(6):3388–3410, 2016.

[CKA10] Ted W Cranford, Petr Krysl, and Mats Amundin. A new acoustic portal into the odontocete ear and vibrational analysis of the tympanoperiotic complex. *PloS one*, 5(8):e11927, 2010.

[CKH08] Ted W Cranford, Petr Krysl, and John A Hildebrand. Acoustic pathways revealed: simulated sound transmission and reception in Cuvier's beaked whale (Ziphius cavirostris). *Bioinspiration & Biomimetics*, 3(1):016001, 2008.

[CMS$^+$08] Ted W Cranford, Megan F Mckenna, Melissa S Soldevilla, Sean M Wiggins, Jeremy A Goldbogen, Robert E Shadwick, Petr Krysl, Judy A St Leger, and John A Hildebrand. Anatomic geometry of sound transmission and reception in Cuvier's beaked whale (Ziphius cavirostris). *The Anatomical Record*, 291(4):353–378, 2008.

[CW17] Wenbin Chen and Yanqiu Wang. Minimal degree H(curl) and H(div) conforming finite elements on polytopal meshes. *Mathematics of Computation*, 86(307):2053–2087, 2017.

[Das03] Gautam Dasgupta. Interpolants within convex polygons: Wachspress shape functions. *Journal of Aerospace Engineering*, 16(1):1–8, 2003.

[DG10] Leszek Demkowicz and Jayadeep Gopalakrishnan. A class of discontinuous Petrov–Galerkin methods. part i: The transport equation. *Computer Methods in Applied Mechanics and Engineering*, 199(23-24):1558–1572, 2010.

[DG11a] Leszek Demkowicz and Jay Gopalakrishnan. A class of discontinuous Petrov–Galerkin methods. ii. optimal test functions. *Numerical Methods for Partial Differential Equations*, 27(1):70–105, 2011.

[DG11b] Leszek Demkowicz and Jayadeep Gopalakrishnan. Analysis of the DPG method for the poisson equation. *SIAM Journal on Numerical Analysis*, 49(5):1788–1809, 2011.

[DG14] Leszek F Demkowicz and Jay Gopalakrishnan. An overview of the discontinuous Petrov Galerkin method. In *Recent developments in discontinuous Galerkin finite element methods for partial differential equations*, pages 149–180. Springer, 2014.

[DGNS17] Leszek Demkowicz, Jay Gopalakrishnan, Sriram Nagaraj, and Paulina Sepulveda. A spacetime DPG method for the schrodinger equation. *SIAM Journal on Numerical Analysis*, 55(4):1740–1759, 2017.

[EDC14] Truman Ellis, Leszek Demkowicz, and Jesse Chan. Locally conservative discontinuous Petrov–Galerkin finite elements for fluid problems. *Computers & Mathematics with Applications*, 68(11):1530–1549, 2014.

[Epp96] David Eppstein. Linear complexity hexahedral mesh generation. In *Proceedings of the twelfth annual symposium on Computational geometry*, SCG '96, pages 58–67, New York, NY, USA, 1996. ACM.

[FGS14] Michael Floater, Andrew Gillette, and N Sukumar. Gradient bounds for Wachspress coordinates on polytopes. *SIAM Journal on Numerical Analysis*, 52(1):515–532, 2014.

[FKDLT17] Federico Fuentes, Brendan Keith, Leszek Demkowicz, and Patrick Le Tallec. Coupled variational formulations of linear elasticity and the DPG methodology. *Journal of Computational Physics*, 348:715–731, 2017.

[FKDN15] Federico Fuentes, Brendan Keith, Leszek Demkowicz, and Sriram Nagaraj. Orientation embedded high order shape functions for the exact sequence elements of all shapes. *Computers & Mathematics with applications*, 70(4):353–458, 2015.

[FKR05] Michael S Floater, Géza Kós, and Martin Reimers. Mean value coordinates in 3D. *Computer Aided Geometric Design*, 22(7):623–631, 2005.

[Flo03] Michael S Floater. Mean value coordinates. *Computer aided geometric design*, 20(1):19–27, 2003.

[GQ14] Jay Gopalakrishnan and Weifeng Qiu. An analysis of the practical DPG method. *Mathematics of Computation*, 83(286):537–552, 2014.

[GRB16] Andrew Gillette, Alexander Rand, and Chandrajit Bajaj. Construction of scalar and vector finite element families on polygonal and polyhedral meshes. *Computational Methods in Applied Mathematics*, 16(4):667–683, 2016.

[HF06] Kai Hormann and Michael S Floater. Mean value coordinates for arbitrary planar polygons. *ACM Transactions on Graphics (TOG)*, 25(4):1424–1441, 2006.

[HS08] Kai Hormann and Natarajan Sukumar. Maximum entropy coordinates for arbitrary polytopes. In *Computer Graphics Forum*, volume 27, pages 1513–1520. Wiley Online Library, 2008.

[Hug87] Thomas JR Hughes. *The finite element method: linear static and dynamic finite element analysis*. Prentice-Hall International, Inc, 1987.

[Jay57] Edwin T Jaynes. Information theory and statistical mechanics. *Physical review*, 106(4):620, 1957.

[JLSW02] Tao Ju, Frank Losasso, Scott Schaefer, and Joe Warren. Dual contouring of Hermite data. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '02, pages 339–346, New York, NY, USA, 2002. ACM.

[KBSS01] Leif P Kobbelt, Mario Botsch, Ulrich Schwanecke, and Hans-Peter Seidel. Feature sensitive surface extraction from volume data. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 57–66. ACM, 2001.

[KFD16] Brendan Keith, Federico Fuentes, and Leszek Demkowicz. The DPG methodology applied to different variational formulations of linear elasticity. *Computer Methods in Applied Mechanics and Engineering*, 309:579–609, 2016.

[KKR$^+$17] Brendan Keith, Philipp Knechtges, Nathan V Roberts, Stefanie Elgeti, Marek Behr, and Leszek Demkowicz. An ultraweak DPG method for viscoelastic fluids. *Journal of Non-Newtonian Fluid Mechanics*, 247:107–122, 2017.

[KTC12] Petr Krysl, Vanessa Trijoulet, and Ted W Cranford. Validation of a vibroacoustic finite element model using bottlenose dolphin experiments. In *The Effects of Noise on Aquatic Life*, pages 65–68. Springer, 2012.

[LC87] William E Lorensen and Harvey E Cline. Marching cubes: A high resolution 3D surface construction algorithm. In *ACM Siggraph Computer Graphics*, volume 21, pages 163–169. ACM, 1987.

[MB95] Peter Murdoch and Steven E Benzley. The spatial twist continuum. In *Proceedings, 4th International Meshing Roundtable*, volume 95, pages 243–251, 1995.

[MBBM97] Peter Murdoch, Steven Benzley, Ted Blacker, and Scott A Mitchell. The spatial twist continuum: A connectivity based method for representing all-hexahedral finite element meshes. *Finite elements in analysis and design*, 28(2):137–149, 1997.

[MBLD02] Mark Meyer, Alan Barr, Haeyoung Lee, and Mathieu Desbrun. Generalized barycentric coordinates on irregular polygons. *Journal of graphics tools*, 7(1):13–22, 2002.

[MD04a] Elisabeth Anna Malsch and Gautam Dasgupta. Interpolations for temperature distributions: A method for all non-concave polygons. *International Journal of Solids and Structures*, 41(8):2165–2188, 2004.

[MD04b] Elisabeth Anna Malsch and Gautam Dasgupta. Shape functions for polygonal domains with interior nodes. *International Journal for Numerical Methods in Engineering*, 61(8):1153–1172, 2004.

[Mon03] Peter Monk. *Finite element methods for Maxwell's equations*. Oxford University Press, 2003.

[MW15] Tapabrata Mukherjee and Jon P Webb. Hierarchical bases for polygonal finite elements. *IEEE Transactions on Magnetics*, 51(3):1–4, 2015.

[MW16] T Mukherjee and JP Webb. Polygonal finite elements of arbitrary order. *IEEE Transactions on Magnetics*, 52(3):1–4, 2016.

[Ned80] Jean-Claude Nedelec. Mixed finite elements in R 3. *Numerische Mathematik*, 35(3):315–341, 1980.

[Ned86] Jean-Claude Nedelec. A new family of mixed finite elements in R 3. *Numerische Mathematik*, 50(1):57–81, 1986.

[Owe98] Steven J. Owen. A Survey of Unstructured Mesh Generation Technology. In *7TH INTERNATIONAL MESHING ROUNDTABLE*, pages 239–267, 1998.

[PA97] Mark A Price and Cecil G Armstrong. Hexahedral mesh generation by medial surface subdivision: Part II. Solids with flat and concave edges. *International Journal for Numerical Methods in Engineering*, 40(1):111–136, 1997.

[PAS95] Mark A Price, CG Armstrong, and MA Sabin. Hexahedral mesh generation by medial surface subdivision: Part I. Solids with convex edges. *International Journal for Numerical Methods in Engineering*, 38(19):3335–3359, 1995.

[PD17] Socratis Petrides and Leszek F Demkowicz. An adaptive DPG method for high frequency time-harmonic wave propagation problems. *Computers & Mathematics with Applications*, 74(8):1999–2017, 2017.

[RGB14] Alexander Rand, Andrew Gillette, and Chandrajit Bajaj. Quadratic serendipity finite elements on polygons using generalized barycentric coordinates. *Mathematics of computation*, 83(290):2691–2716, 2014.

[RT77] Pierre-Arnaud Raviart and Jean-Marie Thomas. A mixed finite element method for 2–nd order elliptic problems. In *Mathematical aspects of finite element methods*, pages 292–315. Springer, 1977.

[Sch96] Robert Schneiders. A grid-based algorithm for the generation of hexahedral element meshes. *Engineering with Computers*, 12(3-4):168–177, 1996.

[Sch97] Robert Schneiders. An algorithm for the generation of hexahedral element meshes based on an octree technique. In *6th International Meshing Roundtable*, pages 195–196, 1997.

[She07] Jason F Shepherd. *Topologic and geometric constraint-based hexahedral mesh generation*. PhD thesis, The University of Utah, 2007.

[Sib80] Robin Sibson. A vector identity for the Dirichlet tessellation. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 87, pages 151–155. Cambridge University Press, 1980.

[SMB98] Natarajan Sukumar, Brian Moran, and T Belytschko. The natural element method in solid mechanics. *International journal for numerical methods in engineering*, 43(5):839–887, 1998.

[SSW96] R. Schneiders, R. Schindler, and F. Weiler. Octree-based generation of hexahedral element meshes. In *IN PROCEEDINGS OF THE 5TH INTERNATIONAL MESHING ROUNDTABLE*, pages 205–215, 1996.

[Suk04] N Sukumar. Construction of polygonal interpolants: a maximum entropy approach. *International journal for numerical methods in engineering*, 61(12):2159–2181, 2004.

[Suk13] N Sukumar. Quadratic maximum-entropy serendipity shape functions for arbitrary planar polygons. *Computer Methods in Applied Mechanics and Engineering*, 263:27–41, 2013.

[TBM96] T. J. Tautges, T. Blacker, and S. A. Mitchell. The whisker weaving algorithm: A connectivity-based method for constructing all–hexahedral finite element meshes. *International Journal for Numerical Methods in Engineering*, 39(19):3327–3349, 1996.

[TPPM12] Cameron Talischi, Glaucio H Paulino, Anderson Pereira, and Ivan FM Menezes. Polymesher: a general-purpose mesh generator for polygonal elements written in Matlab. *Structural and Multidisciplinary Optimization*, 45(3):309–328, 2012.

[Wac71] Eugene L Wachspress. A rational basis for function approximation. In *Conference on Applications of Numerical Analysis*, pages 223–252. Springer, 1971.

[Wac75] Eugene L Wachspress. *A rational finite element basis*. Elsevier, 1975.

[War96] Joe Warren. Barycentric coordinates for convex polytopes. *Advances in Computational Mathematics*, 6(1):97–108, 1996.

[Whi57] Hassler Whitney. *Geometric integration theory*. Princeton University Press, 1957.

[ZBS05] Yongjie Zhang, Chandrajit Bajaj, and Bong-Soo Sohn. 3D finite element meshing from imaging data. *Computer Methods in Applied Mechanics and Engineering*, 194(4849):5083 – 5106, 2005.

[ZBX09] Yongjie Zhang, Chandrajit Bajaj, and Guoliang Xu. Surface smoothing and quality improvement of quadrilateral/hexahedral meshes with geometric flow. *Communications in Numerical Methods in Engineering*, 25(1):1–18, 2009.