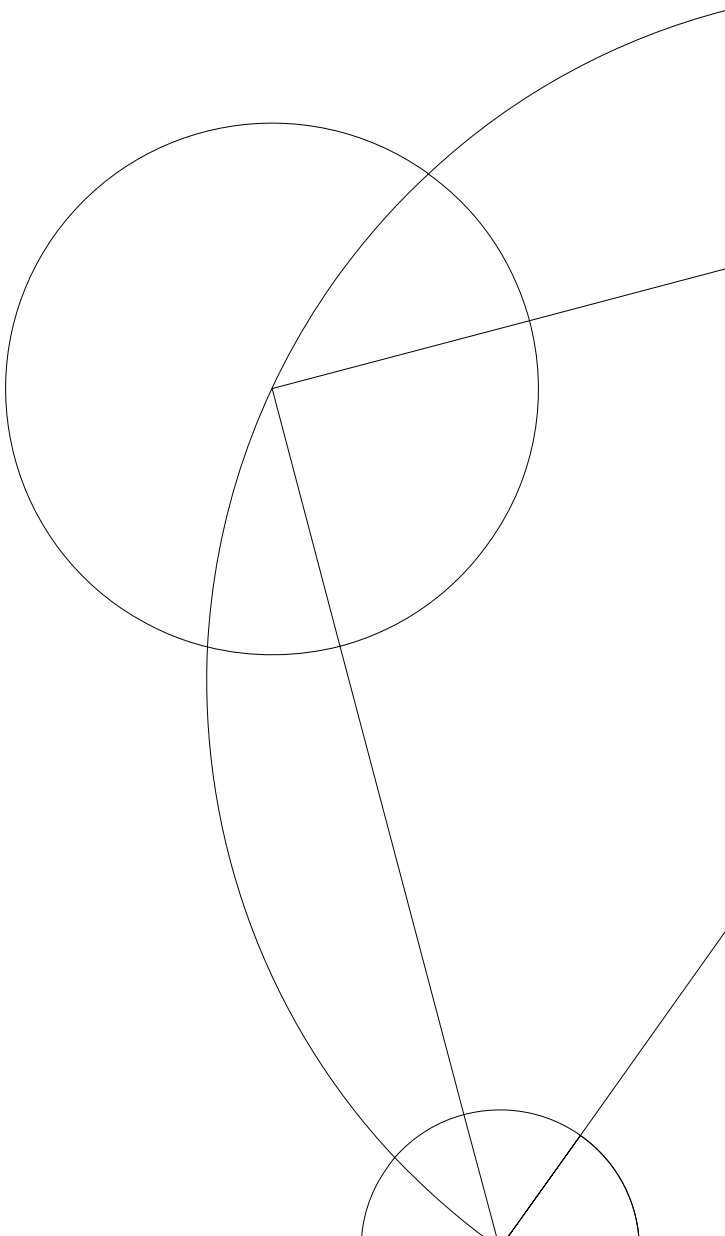
KØBENHAVNS UNIVERSITET

JINGZHENG LI

High Order FEM Deformable Body Simulator



Supervisor: Professor Kenny Erleben

Institute of Mathematics

Abstract

Deformable body simulation is a very important branch of physics-based simulation methods, and is widely used in virtual reality, animation, games and other related industries. Among them, finite element method is one of the main methods to realize deformation body simulation. However, since FEM requires numerous complex numerical calculations, its computational complexity will be so high that cannot meet people's requirements for visual effects and simulation speed. Therefore, we focus on the improvements on numerical method of the simulation and the FEM itself to build a more accurate and fast deformation simulator.

Among them, for the numerical method of simulation, we introduce several linear solvers including conjugate gradient method, conjugate residual method, etc., and propose several preconditioners to accelerate these linear solvers, such as Jacobi preconditioner, Multigrid preconditioner, etc. Through experimental verification, it is found that due to the complexity of energy in FEM, which will bring non-convexity to the linear system, the conjugate residual is generally better than the conjugate gradient when dealing with non-positive definite matrices. And after adding the preconditioner, the speed of reducing relative error in the iterative process will be obviously better than the situation without preconditioner.

For FEM, we propose high-order FEM to improve the accuracy of the simulation by increasing the order of the interpolation basis function. We give a general algorithm for solving the reference tetrahedral element shape function of isoparametric space by using the Vandermonde matrix, and then build a structure to solve the force and Hessian matrix under high-order FEM. Finally, since the deformation gradient is no longer a constant matrix in a high-order finite element, we propose two numerical quadrature methods including Gauss quadrature and Hammer quadrature to calculate the force matrix and Hessian matrix. Through experiments, it is found that the high-order FEM simulation is significantly better than the linear case, and can obtain more accurate simulation results with fewer iteration steps.

Keywords: FEM, implicit iteration, preconditioner, high-order shape function

Indhold

1 Introduction **6**

1.1 Research background and significance 6

1.2 Research status and development trend 7

1.2.1 An overview of the finite element method 7

1.2.2 Numerical Solution of finite element method 8

1.2.3 High order finite element method 10

1.3 Research content and chapter arrangement 11

1.3.1 Research content 11

1.3.2 Chapter arrangement 11

2 Basic Theory of Elastodynamics **13**

2.1 Finite strain theory 13

2.2 Constitutive models of elastic materials 15

2.2.1 St.Venant-Kirchhoff material model 16

2.2.2 Corotated linear material model 16

2.2.3 Neo-Hookean material model 17

2.3 Discrete-time integral solution method 17

2.3.1 Explicit time integration method 17

2.3.2 Semi-implicit & Implicit time integration method 18

2.3.3 Newton-Raphson method 20

3 Equation Solvers & Numerical Integration **23**

3.1 Matrix preconditioner 23

3.1.1 Jacobi (diagonal) preconditioner 24

3.1.2 Incomplete Cholesky Factorization preconditioner 24

3.1.3 Multigrid preconditioner 25
3.2 Linear solver of large sparse linear system 28
3.2.1 Preconditioned conjugate gradient 28
3.2.2 Preconditioned conjugate residual 29
3.2.3 Preconditioned generalized minimal residual method 30
3.3 Numerical Integration 31
3.3.1 Gauss Quadrature 32
3.3.2 Hammer Quadrature 34

4 Finite Element Method Simulation 38

4.1 Calculation of shape function 39
4.1.1 Linear FEM shape function 39
4.1.2 General FEM shape function 41
4.2 Calculate deformation gradient in FEM way 46
4.2.1 Linear FEM deformation gradient 46
4.2.2 General FEM deformation gradient 47
4.3 Calculate force & force gradient in FEM way 48
4.3.1 Compute force 48
4.3.2 Compute force gradient 50

5 Numerical Results & Summarize 53

5.1 Taichi programming language 53
5.2 Constitutive models 54
5.3 Numerical analysis 55
5.3.1 Explicit & implicit method 55
5.3.2 Preconditioner numerical method 57
5.4 High order FEM 58

5.5 Summary and outlook 60

5.5.1 Summary of thesis work 60

5.5.2 Outlook for future work 61

6 Appendix **66**

6.1 High order FEM basis function 66

6.2 Numerical quadrature coordinate and weight 68

6.3 Proof of Hammer theorem 1 70

6.4 Implementing details of high order FEM 71

1 Introduction

1.1 Research background and significance

With the development of computer hardware, the computing power has become increasingly powerful, which has promoted the rapid development of computer graphics and other related disciplines. Physically based simulation is an important branch of computer graphics, engineering computing and other interdisciplinary research fields, which can create a realistic and immersive experience in a virtual environment, and is widely used in computer animation, game development [29] , multi-body robots [20], etc.

Among them, the deformable body simulation is the focus of this research. In order to accurately simulate the deformation process of the elastic body, it is necessary to iteratively solve the FEM dynamic equation. The solution process has high computational complexity, especially for nonlinear deformable bodies, the computational complexity is even greater. Therefore, iterative compute the FEM dynamics equations as efficiently and stably as possible without loss of accuracy is the focus of this paper. We will mainly improve the numerical method of simulation and the FEM itself, focusing on the numerical method of quickly solving FEM, and the construction of high-order deformable body FEM simulator.

In terms of numerical method of simulation, including Euler grid method, Lagrangian particle method and hybrid method, each method can be suitable for different physical phenomena. In the process of deformable body simulation, various methods are involved with their own advantages and disadvantages. Among them, FEM is a numerical method based on a complex dynamic model, which can accurately describe the deformation characteristics of the deformable body. The simulation behavior in this method is jointly determined by mass parameters, stiffness parameters (i.e., the stress-strain relationship of the material) and damping parameters. The simulation results of FEM have the characteristics of high precision and high fidelity.

The iterative solution process of deformable body dynamic equation of FEM can be divided into two categories, explicit and implicit iterative method. The explicit iterative method directly differentiates the time of the simulation and has high computational efficiency. On the contrary, however, it is shown that the stability of the explicit iterative method is poor. In order to ensure the stability, the value of the time step strictly depends on the minimum size of the FEM element, and the too small a time step will in turn lead to a long solution time. Implicit iterative method solution can be obtained after meeting the convergence condition, and each iteration needs to solve a large-scale linear equation system. In practical applications, FEM simulation of deformable bodies must meet the requirements of efficient and stable application, so how to realize large-scale and fast numerical calculation including numerical integration, gradient descent, linear solver, etc. is a research difficulty.

From the perspective of simulation efficiency and stability, the equations are solved with the vertices of the grid in traditional simulation methods. In order to pursue realistic and vivid simulation effects, people often use high-resolution models, while too many vertices will lead to low efficiency, and the simulation process will be even more complicated due to the collision contact of multiple models, which will all have an impact on the simulation efficiency. In addition, the high-resolution grid has a high limit on the parameters of the simulation, where too large a time step or too high stiffness can bring instability to the simulation process. In order to improve the simulation accuracy of FEM for complex problems, the p-type FEM method can be adopted, which increases the FE basis function order while keeping the initial mesh resolution unchanged. This method has advantages of low requirements for grid quality, less preprocessing workload, and high calculation accuracy. Therefore, how to achieve a more efficient and accurate p-type FEM is also a research focus.

To sum up, this topic focuses on the field of deformable body simulation, aiming to apply more efficient numerical methods, obtain more realistic deformation effects, and pursue more rapid and stable simulation process, which has very important practical significance and application value.

1.2 Research status and development trend

1.2.1 An overview of the finite element method

In physically-based simulation, the choice of numerical method and the solution of the kinematic equations have a direct impact on deformable bodies, among which the FEM method is the most commonly used numerical method for discretizing PDEs in engineering applications. The related method was originally developed by PG. Ciarlet in [7], and its application in the field of graphics began in 1987 [27], which has been widely used in physically-based simulation, geometric modeling, geometric processing, computational fabrication and many other fields. In the field of geometric modeling, the deformation of models is highly dependent on FE, including how to build efficient real-time modeling methods [13]. In the field of geometric processing, FEM can be used for the reconstruction of surfaces and their textures [15], image retargeting [14], etc. In the field of computational fabrication, FEM has large-scale applications for 3D printing, detection of structural soundness, shape optimization [30, 31], etc.

For physical simulation, most simulation algorithms rely on solving PDEs as physical prerequisite and then solved by FEM, the most common examples are the diffusion equation, elastic deformation, and its variations such as elastoplasticity and viscoelasticity. These applications usually contain two desirable features: the ability to handle complex shapes robustly, and to achieve maximum performance with an acceptable error. Therefore, linear triangular or tetrahedron elements are often used

as mesh elements, as they can be generated robustly and lead to the most efficient FEM algorithm possible. K. Erleben [12] described the theory and implementation in detail of linear FEM in graphics, that is, dividing a three-dimensional deformable body into a series of elements, and using the displacement of these elements to represent the overall deformation. Although FEM is considered to be a reliable and accurate method for simulating deformable bodies, its main disadvantage is its computationally complex, especially for the implicit method, which is pretty time-consuming for solving the large-scale linear equations for each iteration time step, and the computational time will increase dramatically when the degrees of freedom of the model increases. Therefore, in the field of graphics, improvement of implicit iterative methods for deformable bodies has always been a research focus.

Among them, especially for implicit FEM, due to the complexity of energy E, it usually forms a non-positive definite Hessian matrix, which will bring large-scale non-convex optimization problems. It's hard to find a general best way to do this, but a currently common way is to use projections to get to the closes SPD matrix [11, 19]. Starting from the idea of Local-Global [18], S. Bouaziz and TT. Liu[6] adopted the Projective Dynamic (PD) method to introduce auxiliary variables into the energy system and perform Local-Global processing, which can be used to solve the implicit time integration using FEM when the elastic potential energy of the material model has a specific quadratic form. Then they reformulate projective dynamics and introduce the quasi-Newton perspective [19], induce a simplified matrix to replace the original Hessian matrix in Newton's method, and can use a variety of existing numerical methods to speed up the solution process (such as L-BFGS). The method makes the simulation of hyperelastic materials more efficient, usually more than 10 times faster than traditional Newton iterative methods. Then T. Du et al. established DiffPD [8] and DiffCloth [17] differentiable implicit FEM simulators accordingly to this projective method, and introduce projective dynamics and Cholesky factorization preconditioner into the forward FEM process to accelerate the backpropagation, and give the corresponding non-penetrating contact and friction models under the PD differentiable simulator.

1.2.2 Numerical Solution of finite element method

In the simulation process of implicit FEM, it is usually necessary to solve a system of large sparse linear equations. The basic idea of the iterative method is that given the initial value x_0, according to a certain iterative format, we can calculate $x_1, x_2, \cdots, x_k, \cdots$ in sequence to obtain x_k converges to x^*, when $k \to \infty$, where $x^* = A^{-1}b$ is the exact solution of the system of linear equations. Different iteration methods can be constructed according to different iteration formats, which are usually divided into stationary iterative method and Krylov subspace iteration method.

Krylov subspace method is the most important class of iterative methods. Accor-

ding to van der Vorst [28] and [25], the Krylov subspace method refers to the method of finding the optimal approximate solution in the Krylov subspace $\mathcal{K}_n(A, r_0)$. The main idea of this type of method is to find some approximate solution x_n of the real solution x^* in the n-dim Krylov subspace $\mathcal{K}_n$ by different methods, with the continuous expansion of the dimension of the Krylov subspace, to finally find a solution that can satisfy the accuracy. Since the optimality in this type of method involves different types of projections, therefore, the method is also known as the Krylov projection method. Assuming the initial approximation $x_0 = 0$, we need to find the approximate solution $x_n \in \mathcal{K}_n$ in the n-dim Krylov subspace $\mathcal{K}_n(A, r_0) = \text{span}\{r_0, Ar_0, \ldots, A^{n-1}r_0\}$, the Krylov subspace methods can be divided into four main categories:

- Ritz-Galerkin projection method: construct $x_n \in \mathcal{K}_n$ such that the residual is orthogonal to the current subspace, i.e., $b - Ax_n \perp \mathcal{K}_n$.
- Minimum residual norm method: construct $x_n \in \mathcal{K}_n$ such that the residual norm $\|b - Ax_n\|_2$ is the smallest.
- Petrov-Galerkin projection method: construct $x_n \in \mathcal{K}_n$ such that the residual $b - Ax_n \perp \mathcal{L}_n$, where $\mathcal{L}_n$ is another suitable n-dim subspace.
- Minimum error norm method: construct $x_n \in A^T\mathcal{K}_n$ such that the error norm $\|x_n - x^*\|_2$ is the smallest.

Among many Krylov subspace methods, given a large-scale sparse system of equations, we can choose a suitable linear solver according to the type of A: when A is symmetrically positive definite, CG is preferred. When A is not positive definite, CR, MINRES, SYMMLQ methods can be selected. When A is asymmetric, the choice will be very complicated. If the computational and storage costs are acceptable, the GMRES or FOM algorithm can be selected. However, the scale of practical problems is usually relatively large, so the restart GMRES(m), restart FOM(m) or BiCGSTAB algorithms are better choices. In addition, the convergence of the Krylov subspace method is closely related to many factors, such as the spectral condition number of the matrix A, the eigenvalue distribution of A, etc. The sensitivity of the solution of the equation system also depends on the condition number of the matrix A. When the condition number of the problem is relatively large, the Krylov subspace will converge very slowly, or even not converge. At this time, in order to allow the equation system to quickly calculate a solution with relatively high accuracy, preconditioning technology are often necessary.

Preconditioner aims to transform the original system of equations into a new mathematically equivalent system that can be quickly solved with the Krylov subspace method. The preconditioner refers to the matrix that accomplishes this transformation effect. Usually given a non-singular matrix M that approximates A, the system of equations after preconditioning has the form such that $M^{-1}Ax = M^{-1}b$, which has the same solution as the original system of equations. Saad gives some discussion

in [22], usually a good preconditioner M needs to satisfy two basic conditions: 1. The cost of constructing and applying preconditioner M should be as cheap as possible. 2. The preconditioned system of equations requires fewer iterations using the Krylov subspace method. In general, there are two main ways to construct a preconditioner. One is based on the PDE itself, but requires a detailed and in-depth understanding of the problem itself, which is often difficult in practical problems[5]. Another way is to construct preconditioner M only by relying on the information of matrix A, which are widely used currently, including incomplete LU decomposition, incomplete Cholesky decomposition, Multigrid method, etc. In the existing numerical methods, how to better combine linear solver and preconditioner to solve high-order FEM is still a problem that needs to be considered, so in this project we will focus on the application of these numerical methods.

1.2.3 High order finite element method

For numerically solving PDE problems, basis order, mesh resolution, and mesh element quality are the main factors that affect the accuracy of FEM. The traditional linear FEM usually takes a linear shape function on the mesh vertices for interpolation, which is highly dependent on the mesh quality to ensure the error limit. But this is often challenging in practice. Therefore, using high-order FEM to reduce the mesh quality requirements of the simulation itself is often a better solution.

Hierarchical element concept p-FEM was originally proposed by OC. Zienkiewicz et al.[32], and then in 1981 I. Babuška et al.[2] established the basic theory of p-FEM, and gave p-FEM priori error estimation of approximate convergence rate. They theoretically and strictly proved that the convergence rate of p-FEM under quasi-uniform triangular grid is better than that of traditional h-FEM, and it is verified that when simulating singularity problems in most practical cases under quasi-uniform grid, p-FEM converges twice as fast as h-FEM. Then I. Babuška showed how to effectively combine grid optimization and p-type extension in [1].

p-type FEM in graphics shares the same theorectical basis with engineering and has some applications. AW. Bargteil [4] and E Edwards [10] added second-order finite elements for elastic simulation and fluid simulation respectively, which uses the quadratic Bezier element, recalculates the force generated by 10 nodes on the second-order FE tetrahedron, and tries a variety of different numerical quadrature methods, which greatly improved the accuracy of the simulation. T. Schneider et al. [23] used p-refinement adaptive FEM to decouple the numerical approximation error of the mesh quality of the elliptical PDE. They pre-analyze mesh quality with priori error estimates and employ high-order shape functions at complex meshes to reduce expensive mesh optimization operations. It is proved that more robust FEM simulation can be obtained with only a small-time cost. In addition, M. Paszyński et al. [21] replaced traditional computationally-intensive grid and order refinement

algorithm kernel with a deep neural network, and learned how to optimize the grid elements and modify the order of the polynomial through NN. The traditional FEM simulation has fixed deformation grandient, which limits linear FEM to play a better role in graphics. Therefore, further research and discussion are needed on how to construct better high-order FEM simulator.

1.3 Research content and chapter arrangement

1.3.1 Research content

In order to accurately and efficiently simulate the deformation process of the elastic body, we will mainly improve the numerical method of simulation and the FEM itself, focusing on the numerical method of quickly solving FEM, and the construction of high-order deformable body FEM simulator.

1.3.2 Chapter arrangement

Chapter 1 Introduction: we first introduce the application requirements of deformable bodies simulation, and the research background and significance of adopting FEM for physically-based simulation. Then we describe the research status of implicit FEM, fast numerical method, p-type FEM respectively. After analyzing and discussing the research trend, we proposed the main research contents and chapter introductions of the thesis.

Chapter 2 Basic theory of elastodynamics: This chapter first introduces the finite strain theory and the constitutive model of elastic materials, focusing on the Cauchy-Green tensor, Piola-Kirchhoff stress tensor, energy density function, deformation gradient and other continuum mechanics contents. Then three iterative methods of dynamic equation solving methods are introduced, including explicit, semi-implicit and fully implicit methods. Then we describe the Newton-Raphson method of optimization method and the damped Newton method with step size information. We also show the iterative solving process and characteristics of different kinds of methods, and analyze the advantages and disadvantages of the three methods.

Chapter 3 Equation solvers & numerical integration: This chapter first introduces the linear equation solver and its corresponding matrix preconditioner algorithm, in which we focus on the preconditioned conjugate gradient (PCG), preconditioned conjugate residual (PCR) and preconditioned minimal residual (GMRES) method, and their corresponding Jacobi preconditioner, incomplete Cholesky factorization preconditioner (IC) and multigrid preconditioner (MG). Finally, we show two numerical integration methods, Gauss quadrature and Hammer quadrature, which are required for high-order FEM, and give the general algorithm of the numerical inte-

gration method.

Chapter 4 Finite element method simulation: This chapter first gives a simple construction method of FEM linear shape function, and constructs a general 2D and 3D shape function algorithm based on Vandermonde matrix. Then the solution methods of linear and general deformation gradient are constructed respectively based on different order shape functions. Finally, according to the energy density function, we give the general method of how to calculate the force and force gradient of implicit high-order FEM, and describe the method of constructing large sparse Hessian matrix for implicit FEM.

Chapter 5 Numerical Results & Summarize: Summarize the research work and experimental content of the thesis, including constitute model, numercial method, high order FEM, etc. Point out the areas that need to be improved according to the experimental results, and finally put forward some future improvement suggestions for this deformation body simulator.

Appendix: In the appendix we mainly give the 2/3-dimension FEM shape function, and position/weight parameters of Gauss quadrature Hammer quadrature. Then give the theorem proof of Hammer's quadrature. Finally, we show the computational details of high-order FEM, including high-order deformation gradient computation, force computation, Hessian matrix computation, and how to use Gauss quadrature on tetrahedral elements.

2 Basic Theory of Elastodynamics

2.1 Finite strain theory

Based on the theory of continuum mechanics, objects are usually regarded as continuum, which can be divided infinitely and keep their physical properties unchanged, thus simplifying objects into several small continuum that occupy a certain space and can be described by some physical quantities. Objects usually deform and move under the action of external conditions. In order to describe the deformation or motion of these objects, two configurations need to be introduced in a triangular element: 1. At time $t_0 = 0$, the continuum is in the initial state, and it is assumed that the area it occupies in space is Ω_0, and its boundary is $\partial\Gamma_0$; 2. At the current time t, due to the action of external force, the continuum deforms and moves, assuming that the area it occupies in space is Ω, and the boundary is $\partial\Gamma$. The following two coordinate systems are established in different conditions:

- In the initial state, the coordinate system used to describe the position vector X of the material point, the coordinates (X_1, X_2, X_3) are called material coordinates or Lagrange coordinates, and each material point corresponds to a unique material coordinate.
- In the current state, the coordinate system used to describe the position vector x of the material point after deformation, the coordinates (x_1, x_2, x_3) are called space coordinates or Euler coordinates.

The deformation or motion of the continuum can be represented by the mapping from X to x as $x = \phi(X) \approx FX + t$. And vice versa, there is also an inverse mapping $X = \phi^{-1}(x)$, which gives the initial position X of a particular material point with coordinates x at any moment. The expression of deformation gradient F is defined as the Jacobian matrix that

$$F = \frac{\partial \phi(X)}{\partial X} = \frac{\partial x}{\partial X} = \begin{pmatrix} \frac{\partial x_1}{\partial X_1} & \frac{\partial x_1}{\partial X_2} & \frac{\partial x_1}{\partial X_3} \\ \frac{\partial x_2}{\partial X_1} & \frac{\partial x_2}{\partial X_2} & \frac{\partial x_2}{\partial X_3} \\ \frac{\partial x_1}{\partial X_1} & \frac{\partial x_1}{\partial X_1} & \frac{\partial x_1}{\partial X_1} \end{pmatrix}. \tag{2.1}$$

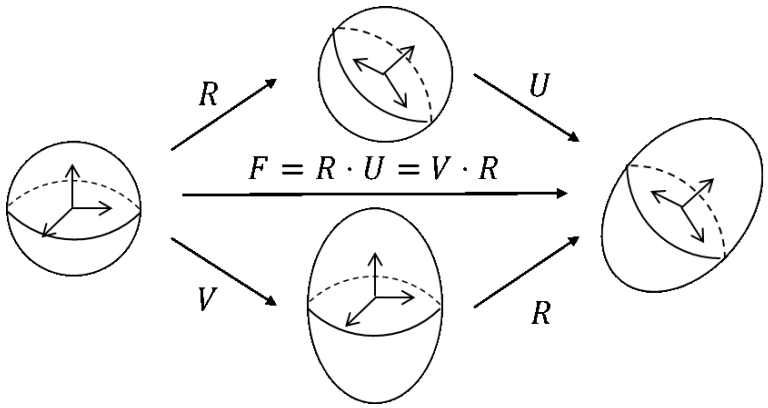
F is a second-order tensor, a fundamental variable in deformation geometry. F has left/right polar decomposition $F = R \cdot U = V \cdot R$, where R is the rigid-body rotation tensor, which is an orthogonal tensor, V and U are the left/right stretch tensor respectively, and they are both symmetric tensors, that is, $V = V^T$, $U = U^T$. Note that the left-right stretch tensor is the pure measurement of strain of the continuum, excludes the rigid body rotation.

The stretch tensors U and V represent the strain of the same material point, which have the same eigenvalues λ_1, λ_2, λ_3 (principle stretches), but different eigenvectors (different principal axis directions). They are also the three main values of the deformation gradient F. V and U have the same main invariants i_1, i_2, i_3, and the expressions are respectively

$$\begin{cases} i_1 = \mathrm{tr}U = \lambda_1 + \lambda_2 + \lambda_3, \\ i_2 = \dfrac{1}{2}\left(i_1^2 - \mathrm{tr}U^2\right) = \lambda_1\lambda_2 + \lambda_2\lambda_3 + \lambda_1\lambda_3, \\ i_3 = \det U = \lambda_1\lambda_2\lambda_3, \end{cases} \tag{2.2}$$

According to the left and right polar decomposition of the deformation gradient, the left and right Cauchy-Green tensors are

$$\begin{cases} B = F \cdot F^T = (V \cdot R) \cdot (V \cdot R)^T = V \cdot V^T = V^2 \\ C = F^T \cdot F = (R \cdot U) \cdot (R \cdot U)^T = U^T \cdot U = U^2 \end{cases} \tag{2.3}$$



Figur 1: Polar Decomposition

the left and right Cauchy-Green tensors B and C are both symmetric tensors with the same principal invariant, respectively

$$\begin{cases} I_1 = \mathrm{tr}B = \mathrm{tr}C = \lambda_1^2 + \lambda_2^2 + \lambda_3^2 = \|F\|_F^2, \\ I_2 = \dfrac{1}{2}\left((\mathrm{tr}C)^2 - \mathrm{tr}C^2\right) = \lambda_1^2\lambda_2^2 + \lambda_2^2\lambda_3^2 + \lambda_1^2\lambda_3^2 = \left\|F^T F\right\|_F^2 \\ I_3 = \det B = \det C = \lambda_1^2\lambda_2^2\lambda_3^2 = \det\left(F^T F\right). \end{cases} \tag{2.4}$$

The transformation relationship from the volume element $d\nu_0$ of the initial configuration to the volume element $d\nu$ of the current configuration is $d\nu = Jd\nu_0$, where $J = \det F = \lambda_1\lambda_2\lambda_3$ is the determinant of the deformation gradient F. Many rubber-like elastic materials can be approximately considered incompressible during deformation, and the continuum satisfies $J = \det F = 1$. According to the Nanson

formula, the change relationship between the initial configuration surface element dA and the current configuration surface element da such that $da = JF^{-T} \cdot dA$. Let the external normal vectors of dA and da be N, n respectively, and the surface force acting on da is t, then the Cauchy stress tensor σ satisfies

$$t = \sigma \cdot n \tag{2.5}$$

It is the real stress tensor generated at the current moment, which is a symmetric tensor. Assuming that in the initial configuration, the surface force acting on dA is T, then

$$\begin{cases} T = P \cdot N, \\ P = J\sigma \cdot F^{-T} \end{cases} \tag{2.6}$$

where P is the Piola-Kirchhoff (PK1) stress tensor of the first kind, which is not a true stress and is an asymmetric tensor, and such PK1 stress tensor has great importance for representing elastic energy.

We can see that the implicit integration method is much more complicated than the explicit method. Although it can guarantee the unconditional stability and accuracy of the numerical solution, the implicit integration method needs to solve a large linear equation system for each iteration, which leads to huge computational cost the simulation process. In addition, the implicit method needs to additionally calculate the Hessian matrix of energy, and it needs to introduce the contraction calculation of the 4th order tensor, which is usually very difficult to achieve.

2.2 Constitutive models of elastic materials

The mathematical description of the physical properties of a given material is often referred to as constitutive model, which usually refers to the equations associated with the deformation that trigger the corresponding stress and strain of the material. Its constitutive relation can be completely determined by its strain energy function $\Psi(F)$ (i.e., energy density function). Energy density function can be expressed as a function of the deformation gradient F or the three principal eigenvalues

$$\Psi = \Psi(F) = \Psi(\lambda_1, \lambda_2, \lambda_3) \tag{2.7}$$

Its first derivative can be expressed using the Piola-Kirchhoff stress tensor P with respect to F, such that

$$P(F) = \frac{\partial \Psi(F)}{\partial F} \tag{2.8}$$

Below we will mainly focus on isotropic materials, whose response to deformation is independent of the direction of strain, and briefly introduce three commonly used elastic object constitutive models [24].

2.2.1 St.Venant-Kirchhoff material model

The StVK constitutive model uses the Green nonlinear strain tensor, which can ensure the rotation invariance of the deformable body, and deformations that differ by a rigid body transformation are guaranteed to have the same strain energy. Therefore, StVK materials can exhibit reasonable material responses in deformation situations where linear elasticity does not apply. Its energy density function is

$$\Psi(F) = \mu \|E\|_F^2 + \frac{\lambda}{2} tr^2(E) \tag{2.9}$$

where $E = \frac{1}{2}\left(F^T F - I\right)$, the corresponding PK1 is

$$P(F) = 2\mu FE + \lambda tr(E) F \tag{2.10}$$

Although the StVK model offers significant advantages over the linear elastic model, its range is somewhat limited due to its poor resistance to strong compression: when the StVK elastic object is compressed, the restoring force produced by the model increases with the degree of compression. The strength of the restoring force reaches a maximum when the critical compression threshold is reached. At this point, further compression encounters a decrease in resistance and disappears when the volume is compressed to zero. Continued compression beyond the zero-volume point will create an opposing restoring force that pushes the body to fully flip along one or more axes. In practical computer simulation examples, this behavior typically manifests as local entanglement and flipping of materials when subjected to strong compressive forces.

2.2.2 Corotated linear material model

The Corotated linear constitutive model adopts the Cauchy strain tensor, which can effectively simulate the expansion and contraction of the object, but cannot correctly describe the rotation of the object. In fact, the rotation operation does not cause the deformation of the object and the change of the energy density. In the Cauchy strain tensor, however, the rotation will cause the volume of the object to change and thus produce errors. Therefore, corotated linear model combines the stress-strain relationship in linear materials with nonlinear properties to ensure rotational invariance, and uses polar decomposition $F = RS$ and constructs a new strain measurement $S - I$ whose energy density function is

$$\Psi(F) = \mu \|F - R\|_F^2 + \frac{\lambda}{2} tr^2(S - I), \tag{2.11}$$

where S and R are the polar decomposition of $F = RS$, where $S = R^T F$, and the corresponding PK1 is

$$P(F) = 2\mu(F - R) + \lambda tr\left(R^T F - I\right) R. \tag{2.12}$$

Corotated linear model mainly considers the rotation factor R in the polar decomposition on the basis of the linear elastic model. From the perspective of computational cost, additional overheads include polar decomposition, nonlinear solvers, etc.

2.2.3 Neo-Hookean material model

During the deformation process of traditional hyperelastic materials, the stress increases with the increase degree of the deformation, but after reaching a certain threshold, the stress decreases instead, so that the volume of the deformed body will be compressed to 0 under extreme compression, which may cause in model flipping and resulting in unstable simulation. Neo-Hookean is a constitutive model of nonlinear stress-strain relationship, which adds the logarithmic term $\log J$ of the volume to the energy to ensure that when $J \to 0$, the energy can be $\Psi \to \infty$ to resist extreme compression. Corresponding energy density function

$$\Psi(F) = \frac{\mu}{2}(I_1 - 3) - \mu \log(J) + \frac{\lambda}{2}\log^2(J), \tag{2.13}$$

where $I_1 = \|F\|_F^2 = \lambda_1^2 + \lambda_2^2 + \lambda_3^2$, μ is the shear modulus of infinitesimal deformation of the material, and its corresponding PK1 is

$$P(F) = \mu\left(F - F^{-T}\right) + \lambda \log(J) F^{-T} = \mu\left(F - \frac{1}{J}\frac{\partial J}{\partial F}\right) + \lambda \log(J)\frac{1}{J}\frac{\partial J}{\partial F}. \tag{2.14}$$

The advantage of the Neo-Hookean model is that it contains only one material constant and the unconditional stability of the model. For example, under the condition of small to moderate strain, the material constant obtained by fitting the stress-strain curve under one deformation form can be used to predict stress-strain curves for other deformation forms.

2.3 Discrete-time integral solution method

In the simulation method, in order to facilitate the calculation, in addition to the discrete representation of the model, the time dimension also needs to be discretely represented. The discretization strategy in the time dimension is called the discrete time integration method.

2.3.1 Explicit time integration method

Explicit integration is the most commonly used type of integration method, which is fast and easy to implement since it does not need to solve the linear equation systems. The explicit Euler integration method is the simplest integration method

with first-order precise integration accuracy. Its core idea is to use only the current state to predict the physical quantity of the next time step. The explicit function can be expressed as

$$\begin{cases} x_{n+1} = x_n + hv_n \\ v_{n+1} = v_n + hM^{-1}f(x_n) \end{cases} \tag{2.15}$$

where h is the time difference, x_n, v_n are the position and velocity at time n respectively. Another common explicit integration method is the Leap-Frog integration method, which has second-order accuracy. Its core idea is to use the central difference to calculate the derivative of the velocity, which is equivalent to calculating the position and velocity at staggered time points

$$\begin{cases} x_n = x_{n-1} + hv_{\frac{n-1}{2}} \\ v_{\frac{n+1}{2}} = v_{\frac{n-1}{2}} + hM^{-1}f(x_n) \end{cases} \tag{2.16}$$

In addition, the fourth-order precision Runge-Kutta integration (RK4) is also a common explicit integration method. The core idea is to sample multiple times in one timestep, and use these quantities to calculate next time step.

$$\begin{cases} a_1 = v_n, \quad a_2 = M^{-1}f(x_n) \\ b_1 = v_n + \dfrac{h}{2}a_2, \quad b_2 = M^{-1}f\left(x_n + \dfrac{h}{2}a_1\right) \\ c_1 = v_n + \dfrac{h}{2}b_2, \quad c_2 = M^{-1}f\left(x_n + \dfrac{h}{2}b_1\right) \\ b_1 = v_n + \dfrac{h}{2}c_2, \quad b_2 = M^{-1}f\left(x_n + \dfrac{h}{2}c_1\right) \end{cases} \tag{2.17}$$

$$\begin{cases} x_{n+1} = x_n + \dfrac{h}{6}(a_1 + 2b_1 + 2c_1 + d_1) \\ v_{n+1} = v_n + \dfrac{h}{6}(a_2 + 2b_2 + 2c_2 + d_2) \end{cases} \tag{2.18}$$

However, explicit integration cannot guarantee the stability of numerical solution. If there are no constraints on the state to be reached during the calculation process, the calculation results are likely to diverge and deviate more and more from the accurate value. In the actual simulation process, it is shown that the integration generally adopts very small timestep, and additional damping force to avoid the system instability caused by the increase of energy during the integration process. In the simulation experiments of this paper, especially for the deformable body with a large Young's modulus, an excessively large time step cannot be used, otherwise the stability of the system cannot be guaranteed. Several commonly used explicit integrals are described below.

2.3.2 Semi-implicit & Implicit time integration method

Commonly used implicit integration methods include implicit Euler integration, implicit Newmark integration, etc. This section only briefly introduces the semi-

implicit and implicit Euler integration method. The implicit Euler integration method has first-order accuracy and can be viewed as an improvement of explicit Euler integration method. Its approximate idea originates from the first-order Taylor expansion at time $t+h$

$$y(t) \approx y(t+h) - hy'(t+h) \tag{2.19}$$

where y' is the first derivative of the function $y(t)$ with respect to time. According to the above formula, we can rewrite the kinematic equations into the implicit Euler form

$$\begin{cases} x_{n+1} = x_n + hv_{n+1}, \\ v_{n+1} = v_n + hM^{-1}f(x_{n+1}). \end{cases} \tag{2.20}$$

Assuming that f is holonomic, i.e., depending on x only, we can substitute the second equation into the first one and change the problem into

$$x_{n+1} = x_n + hv_n + h^2M^{-1}f(x_{n+1}), \tag{2.21}$$

which is an equation concerned about x_{n+1}. Different representations of the internal force differ from the selection of material models. For nonlinear materials, the internal force is often nonlinear, so the equation is usually a nonlinear system of equations. Generally, we have two ways to solve this nonlinear root finding problem.

Commonly seen is the Baraff-Witkin [3] style solution which is under the assumption that x_{n+1} is not too far away from x_n. Then we can set $dx = x_{n+1} - x_n$, since we have

$$f(x_{n+1}) \approx f(x_n) + \nabla_x f(x_n)\, dx, \tag{2.22}$$

substituting this approximation into kinematic equation 2.20, we can calculate

$$\begin{aligned} x_n + dx &= x_n + hv_n + h^2M^{-1}\left(f(x_n) + \nabla_x f(x_n)\, dx\right), \\ \left(M - h^2\nabla_x f(x_n)\right) dx &= hMv_n + h^2 f(x_n), \end{aligned} \tag{2.23}$$

when we solve dx, we can get the position and velocity at time $n+1$ such that

$$x_{n+1} = x_n + dx, \quad v_{n+1} = \frac{dx}{h}. \tag{2.24}$$

The Baraff-Witkin style solution can be viewed as one iteration of Newton's method, also referred to as semi-implicit method, which can get us a rough approximation quickly. Now we integrate the nonlinear root finding problem 2.21 over x gives us:

$$x_{n+1} = \arg\min_x \left(\frac{1}{2}\|x - (x_n + hv_n)\|_M^2 + h^2E(x)\right) := \arg\min_x g(x) \tag{2.25}$$

where $E(x)$ is force energy, $f(x) = -\nabla_x E(x)$, $g(x) = \frac{1}{2}\|x - (x_n + hv_n)\|_M^2 + h^2E(x)$. And $\|\cdot\|_M^2$ is matrix norm defined as $\|Y\|_A^2 = Y^TAY$, and vector derivative as $\nabla_Y\left(Y^TAY\right) = \left(A + A^T\right)Y$. Take gradient ∇ and Hessian ∇^2 of x on $g(x)$, we can obtain

$$\begin{cases} \nabla_x g(x) = M(x - (x_n + h v_n)) + h^2 \nabla_x E(x) \\ \nabla_x^2 g(x) = M + h^2 \nabla_x^2 E(x) \end{cases} \tag{2.26}$$

among which $\nabla_x E(x)$ and $\nabla_x^2 E(x)$ are all model dependent. In addition, for non-singular M, we can find that

$$\nabla_x g(x_{n+1}) = 0 \Longleftrightarrow x_{n+1} = (x_n + h v_n) + h^2 M^{-1} f(x_{n+1}) \tag{2.27}$$

At this point we transform the nonlinear root finding problem into minimization problem. The general solution to this problem is the Newton-Raphson method, which we will refer to in later chapters. The core idea of this method is to find a descending direction of the function value, and then through multiple linear approximation iterations, the solution is to obtain the position of the new moment. In the Newton-Raphson method, we can solve the direction of descent dx as

$$\begin{cases} \nabla_x^2 g(x) \cdot dx = -\nabla_x g(x), \\ x_{n+1} = x_n + \alpha \cdot dx. \end{cases} \tag{2.28}$$

where α is stepsize that describe how far the solution will change in next step. Note that if we directly require the algorithm to iterate only once and set linear search step size $\alpha = 1$, we can obtain $x_{n+1} = x_n + dx$ which is exactly Baraff-Witkin style semi-implicit method.

2.3.3 Newton-Raphson method

In order to solve the fully implicit time integral, we need to introduce Newton's iteration method, the basic idea of the Newton-Raphson method is that when finding the minimum value of the objective function $f(x)$, first perform a second-order Taylor expansion near the iteration point x_k, and then find the minimum point of this quadratic function, and use this point as an approximation of the minimum point x^* of the desired objective function.

Let $X = (x_1,\ x_2,\ \ldots,\ x_N)^T \in \mathbb{R}^N$, objective function $f : \mathbb{R}^N \to \mathbb{R}$ is convex function, and the second order is continuously differentiable, our goal is to solve the unconstrained minimization problem $\min_X f(X)$. For $f(X)$ we can have the second order Taylor expansion

$$\varphi(X) = f(X_k) + \nabla f(X_k) \cdot (X - X_k) + \frac{1}{2} \cdot (X - X_k)^T \cdot \nabla^2 f(X_k) \cdot (X - X_k), \tag{2.29}$$

Because the necessary condition for $\min_X f(X)$to be a minimum is that X is the

stagnation point of $\varphi(X)$, that is

$$\nabla\varphi(X) = \nabla f(X_k) + \nabla^2 f(X_k) \cdot (X - X_k) = 0 \tag{2.30}$$

If $\nabla^2 f(X_k)$ is not a singular matrix, the iterative formula can be constructed as

$$X = X_k - \left(\nabla^2 f(X_k)\right)^{-1} \cdot \nabla f(X_k) \tag{2.31}$$

The advantages of Newton's method are: Newton's method is an algorithm with second-order convergence. When the iteration point enters the minimum point convergence region, the convergence speed will be very fast. However, since there is no step factor in the iteration formula of the original Newton method, for non-quadratic objective functions, the function value will sometimes not decrease but increase, that is, $f(X_{k+1}) > f(X_k)$, which indicates that the original Newton method cannot guarantee the stable decline of the function value. And in severe cases may even cause the divergence of the iteration point sequence $\{X_k\}$, resulting in the failure of the calculation.

To eliminate the shortcomings of the original Newton method, we can use the damped Newton method. After determining the direction d_k in each iteration, implement a line search along this direction, and seek the optimal step size α_k to ensure that the optimization direction is the descending direction and can drop a sufficient amount, that is

$$\alpha_k = \arg\min_{\alpha \in \mathbb{R}} f(X_k + \alpha d_k) \tag{2.32}$$

We can use the Armijo line search method to select the step size. Given the shrinkage amount and shrinkage factor $c, \tau \in (0, 1)$, if f decreases along the d_k direction at the kth iteration, it should satisfy

$$f(X_k + \alpha_k d_k) \leq f(X_k) + \alpha_k \cdot c \cdot (\nabla f(X_k))^T d_k \tag{2.33}$$

The basic idea of Armijo line search is to give the initial step size $\alpha_k^{(0)}$, if the descending requirement of the above formula is not satisfied under $\alpha_k^{(0)}$, then take a new step size $\alpha_k^{(1)} = \tau \cdot \alpha_k^{(0)}$, and repeat the calculation until the descending condition is satisfied. The selection method of the initial step size of the kth iteration $\alpha_k^{(0)}$ as

$$\alpha_{k+1}^{(0)} = \begin{cases} \alpha_k^{(i)}, \ if \ \alpha_k^{(i)} < \alpha_k^{(i-1)} \\ \min\left\{1, 2\alpha_k^{(i)}\right\}, \ else \end{cases} \tag{2.34}$$

At this point in Newton's method, the drop amount should satisfy

$$\frac{1}{2}\|f(X_k + \alpha_k d_k)\|^2 \leq \frac{1}{2}\|f(X_k)\|^2 - \alpha_k \cdot c \cdot \|f(X_k)\|^2 = \left(\frac{1}{2} - \alpha_k \cdot c\right)\|f(X_k)\|^2 \tag{2.35}$$

$$\Rightarrow \|f(X_k + \alpha_k d_k)\| \leq (1 - \mu \cdot \alpha_k)\|f(X_k)\| \tag{2.36}$$

where $\mu \in (0, 1)$. Finally, we can combine Armijo line search and Newton's method to form a damped Newton-Raphson method

Algorithm 1 Damped Newton Method

Input: $x = x_n$, μ, μ', ε, α

Output: $x = x_{n+1}$

function DAMPEDNEWTON(x_n, μ, μ', ε, α)
    $x \leftarrow x_n$, $\alpha \leftarrow 1$,$k \leftarrow 0$, $b \leftarrow F(x)$
    while $b > eps$ **do**
        Calculate $A = F'(x)$
        LinearSolver $A \cdot dx = -b$
        $\alpha_{ini} =$ INITIALIZESTEP(α_1, α_2)
        $\alpha =$ ARMIJOLINESEARCH($x, dx, \|F(x)\|, \alpha_{ini}, \mu$)
        Calculate $x = x + \alpha \cdot dx$
    end while
    return x
end function

function INITIALIZESTEP(α_1, α_2)
    if $\alpha_2 < \alpha_1$ **then**
        $\alpha_{ini} = \alpha_2$
    else
        $\alpha_{ini} = \min\{1, 2\alpha_2\}$
    end if
    return α_{ini}
end function

function ARMIJOLINESEARCH(x, dx, $\|F(x)\|$, α_{ini}, μ)
    $\alpha \leftarrow \alpha_{ini}$
    $F(x)_{temp} \leftarrow F(x + \alpha \cdot dx)$
    while $\|F(x)_{temp}\| > (1 - \mu \cdot \alpha) \cdot \|F(x)\|$ **do**
        $\alpha = 0.5 \cdot \alpha$
        $F(x)_{temp} = F(x + \alpha \cdot dx)$
    end while
    return α
end function

3 Equation Solvers & Numerical Integration

This chapter mainly describes the fast numerical methods used in the process of solving FEM, including the Krylov subspace iteration method and matrix preconditioner, as well as the numerical quadrature method required for high-order FEM. For a system of linear equations $Ax = b$, $A \in \mathbb{R}^{n \times n}$ is a non-singular sparse matrix, and the matrix dimension n is large enough. In this case, we will need a suitable iterative method to solve this large linear system of coefficients quickly. In particular, for high-order FEM, the deformation gradient is no longer a constant matrix in a tetrahedral element. At this time, we will also need a high-precision numerical integration method to accurately describe the deformation of the object under the high-order FEM.

3.1 Matrix preconditioner

The preconditioner technique is to transform the equation system $Ax = b$ into an equivalent equation system, which has good properties and is easier to solve. The preconditioning process is actually to make the coefficient matrix spectral distribution more concentrated. Since for the linear equation system, when A is ill-conditioned, or the elements on the main diagonal are small, or the eigenvalues are small, the convergence rate will be very slow. For example, solving SPD (symmetric positive definite) system with CG method, its convergence speed depends on the singular value distribution of the coefficient matrix A. Our expectation is that the spectral condition number of the coefficient matrix after preconditioning is small enough, or its singular values will be concentrated around 1. Similarly, for non-SPD systems, although the singular values of the coefficient matrix cannot explain its convergence, the concentration of the spectral distribution (except 0) can generally speed up the convergence speed of the iteration.

Generally, preconditioner has three forms. If M is an approximate non-singular matrix of A, the equivalent system of preconditioner is

$$\begin{cases} M^{-1}Ax = M^{-1}b \\ AM^{-1}y = M^{-1}y \\ M_1^{-1}AM_2^{-1}y = M_1^{-1}b \end{cases} \tag{3.1}$$

In practical problems, different preconditioners are selected according to different situations and the iterative method used, but the selected preconditioners generally should meet two conditions: 1. The spectral distribution of the preconditioned system should be concentrated, in other words, the iterative convergence speed should be fast; 2. The construction cost of the preconditioner should not be too high, and the iteration should not take too much time. But these two conditions are usually contradictory, so more often we have to do some trade off.

3.1.1 Jacobi (diagonal) preconditioner

One of the preconditioners is Jacobi preconditioner or diagonal preconditioner. We can divide the matrix A into strictly lower triangular, diagonal, and strictly upper triangular matrices

$$A = L + D + L^T, \tag{3.2}$$

then the preconditioner matrix is the diagonal matrix $M = diag\,(A) = D$.

3.1.2 Incomplete Cholesky Factorization preconditioner

Incomplete LU factorization (ILU) is a sparse approximation of LU factorization. The incomplete decomposition technique is generated on the basis of complete decomposition, which makes up for the defect that the matrix sparsity will be destroyed in the process of decomposition. ILU usually refers to $A = LU + R$, where L and U are upper and lower triangular sparse matrices respectively, and $R = A - LU$ is an error matrix.

Cholesky factorization is a special case of LU factorization. When A is a Hermitian matrix, A can be factorized into the product of lower triangular matrices LL^T. A simple recursive derivation idea is to set $A = \begin{bmatrix} a_{11} & A_{21}^T \\ A_{21} & A_{22} \end{bmatrix}$, $L = \begin{bmatrix} l_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix}$, $L^T = \begin{bmatrix} l_{11} & L_{21}^T \\ 0 & L_{22}^T \end{bmatrix}$. If $A = LL^T$, then there are

$$\begin{bmatrix} a_{11} & A_{21}^T \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix} \begin{bmatrix} l_{11} & L_{21}^T \\ 0 & L_{22}^T \end{bmatrix} = \begin{bmatrix} l_{11}^2 & l_{11}L_{21}^T \\ l_{11}L_{21} & L_{21}L_{21}^T + L_{22}L_{22}^T \end{bmatrix} \tag{3.3}$$

where $l_{11} = \sqrt{a_{11}}$, $L_{21} = \frac{1}{l_{11}} A_{21}$, $L_{21}L_{21}^T = A_{22} - L_{21}L_{21}^T$, set $A_{22}^{'} = A_{22} - L_{21}L_{21}^T$, then Cholesky factorization is transformed into recursive solution $A_{22}^{'} = L_{22}L_{22}^T$. The conditions of Cholesky factorization are strict, it is unique only when A is a positive definite Hermitian matrix, or when A is positive semi-definite and meets certain conditions. Similarly, for sparse matrix A, incomplete Cholesky factorization is usually used to ensure that the decomposed matrix can still maintain sparse characteristics.

Algorithm 2 Incomplete Cholesky factorization

Input: A

Output: L, L^T

function ICHOLESKY(A)
  $n \leftarrow A.size()$, $k \leftarrow 0$
  for $k \leq n$ **do**
    $A_{kk} = \sqrt{A_{kk}}$
    for $k+1 \leq i \leq n$ **do**
      if $A_{ik} \neq 0$ **then**
        $A_{ik} = A_{ik}/A_{kk}$
      end if
    end for
    for $k+1 \leq j \leq n$ **do**
      for $j \leq i \leq n$ **do**
        if $A_{ij} \neq 0$ **then**
          $A_{ij} = A_{ij} - A_{ik} \cdot A_{jk}$
        end if
      end for
    end for
  end for
  return $L_{ij} = A_{ij}$ and $L_{ij} = 0$ for $i < j$
end function

3.1.3 Multigrid preconditioner

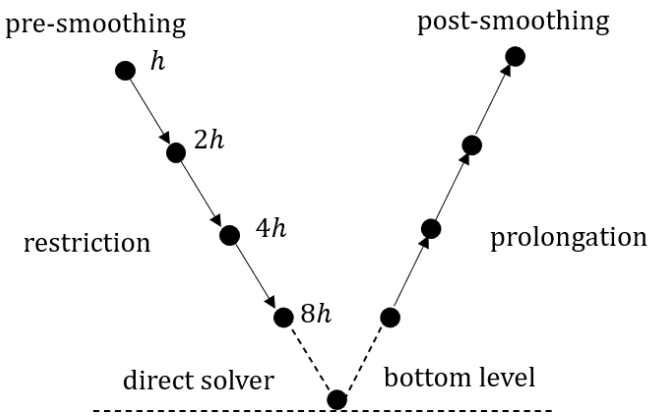
The multigrid method is to construct multiple layers of grids based on the discrete area finite element grid. These grids will become coarsen as the number of layers increases, which includes two main processes: fine grids smooth and coarse grid correction. By doing pre-smoothing iteration on the fine grid to eliminate the high-frequency error, and then transferring the residual after the smoothing iteration to the next layer of grid to continue the smoothing iteration to ensure that the high-frequency error is reduced while also reducing the low-frequency error. In this way, until the last layer of small-scale meshes is solved by the direct solver, the results are returned to the previous layer of meshes as the correction of the solution of this layer, and then perform post-smoothing iterations until to the first layer of meshes. At this point, we can obtained an approximate solution to the linear equations, and the error of this method is usually much smaller than the classical iterative method.

Multigrid method can usually be divided into geometric multigrid method (GMG) and algebraic multigrid method (AMG) according to the construction method. Because GMG usually needs to provide multi-layer finite element meshes, and the scale of linear equations with the coarsest meshes is still large, it is difficult to solve with direct solver. On the other hand, AMG can completely digitize the structure of the grid

level, and replace the actual grid of GMG by introducing auxiliary matrix to represent the virtual grid, which can greatly expand the scope of application of the multigrid method. Usually, Multigrid method will be divided into pre-smoothing, restriction, direct solver, promotion, post-smoothing several steps. The smoothing process can usually take the Gauss-Seidel iteration. Restriction and prolongation usually represent the coarsening and interpolation process of the grid. When the grid size reaches the bottom level, the direct solver can be used to directly obtain the exact solution.

Restriction: If $-A_{ij} > \theta \max_{k \neq i} (-A_{ik})$, then variable i can be considered to be strongly dependent on variable j, where $\theta \in [0, 1)$. This relationship can be represented by a graph, so the problem turns into selecting $C \subset V$ from the nodes in the graph $G = (V, E)$ such that C satisfies some condition H, where H is a heuristic condition. For example, a strong condition H_1 satisfies $\forall (i, j) \in V : (i \in C) \vee (j \in C) \vee (\exists k \in C, (i, k) \in V, (j, k) \in V)$, that is, for any two variables with strong dependencies, either one of the variables is selected, or a variable that two variables depend on together is selected, which can ensure that during interpolation, the value of i can take into account all strongly dependent variables. The weaker condition H_2 satisfies $\forall i \notin C : \exists j \in C, (i, j) \in V$, that is, any unselected variable strongly depends on at least one selected variable, and some strongly dependent variables during interpolation impact will not be considered. At the same time, there is more than one C that satisfies the condition, and a smaller C increases the error while also increasing the speed of the coarse grid computation.

Prolongation: usually represents the interpolation of the error by the backpropagation process $e_i = \sum_{j \in C_i} w_{ij} e_j$, where $C_i = \{j | (i, j) \in V, j \in C\}$. The weight can usually take a simple interpolation method $w_{ij} = -\frac{A_{ij}}{A_{ii}} \left(\frac{\sum_{k \neq i} A_{ik}}{\sum_{l \in C_i} A_{il}} \right)$, that is, only the direct influence of coarse grid variables is considered. A more precise interpolation can also be applied that $w_{ij} = -\frac{1}{A_{ii} + \sum_{k \in F_i^w} A_{ik}} \left(A_{ij} + \sum_{k \in F_i^s} \frac{A_{ik} A_{kj}}{\sum_{m \in C_i} A_{km}} \right)$, where $F_i^s = \{j | (i, j) \in V, j \notin C\}$ represents a strongly dependent variable that is not selected as a coarse grid, $F_i^w = \{j | j \neq i, (i, j) \notin V\}$ denotes a weakly dependent variable. That is, the effects of all strongly dependent variables are taken into consideration, including the direct effects of coarse grid variables and the two-step indirect effects.



Figur 2: V-Cycle Multigrid Method

The iterative mode introduced above is to start from the finest grid and continue to coarsen, pre-smoothing, coarsen ... to the coarsest grid, and then continue to interpolation, post-smoothing, interpolation ... back to the finest grid, which is known as V-Cycle

Algorithm 3 V-Cycle Multigrid

Input: A, x, b

Output: x

function V-CycleMultigrid(A, x, b)
  for $l = n, n-1, ..., 2$ **do**
    $x_l = x_l + GS_l^j(b_l - A_l x_l)$, $j = 1, 2, ..., m_1$
    $r_l = b_l - A_l x_l$
    $b_{l-1} = (I_{l-1})^T r_l$
  end for
  $x_1 = (A_1)^{-1} b_1$
  for $l = 2, 3, ..., n$ **do**
    $x_l = x_l + I_{l-1} x_{l-1}$
    $x_l = x_l + GS_l^j(b_l - A_l x_l)$, $j = 1, 2, ..., m_2$
  end for
  return x
end function

where GS_l^j represents the Gauss-Seidel smooth operator, j represents the number of iterations. I_l represents the mapping matrix between different layer grids. A popular solution now is to use the Multigrid algorithm as a preconditioner in combination with the Krylov subspace method, Apply Multigrid algorithm preconditioning the residual term in the linear solver as $z_k = \text{Multigrid}\,(r_k) = M^{-1} r_k$, where Multigrid$(\cdot)$ is the Multigrid process built for sparse matrices.

3.2 Linear solver of large sparse linear system

The conjugate gradient method constructed, which based on the Lanczos orthogonalization process, is a common algorithm for solving symmetric positive definite linear systems $Ax = b$. However, it fails for non-positive definite equations. Therefore, the method based on the Arnoldi orthogonalization process can be considered: minimal residual method (MINRES) or the conjugate residual (CR) method. The two algorithms are numerically equivalent, and both minimize the residual norm.

3.2.1 Preconditioned conjugate gradient

We can give a framework process on how to derive the Lanczos-type iterative solver: as the algorithm framework discussed in [26], select an initial iterative value x_0 for the linear system $Ax = b$, then we can give the following recurrence relation, and give some constraints:

$$\begin{cases} r_0 = b - Ax_0, \\ p_0 = r_0, \\ x_{j+1} = x_j + \alpha_j p_j, \\ r_{j+1} = r_j - \alpha_j A p_j, \\ p_{j+1} = r_{j+1} + \beta_j p_j, \end{cases} \tag{3.4}$$

where $j = 0, 1, \cdots,\ r_j = b - Ax_j$ is the jth residual vector, and p_j is the vector of the jth search direction. Take the Krylov subspace $\mathcal{K}_n = \text{span}\{r_0, Ar_0, \cdots, A^{n-1}r_0\}$ and the constraint subspace $\mathcal{L} = \mathcal{L}_n\left(A^H, \tilde{r}_0\right)$. Then for the latter two iterative relationships, $\mathcal{L}$ is selected as $\mathcal{K}_n(A, r_0)$ and $A\mathcal{K}_n(A, r_0)$ respectively, that is, satisfy the orthogonal relationship respectively

$$r_{j+1} \perp \mathcal{L}, \quad Ap_{j+1} \perp \mathcal{L} \tag{3.5}$$

At this time, different parameters α_j, β_j derive different Krylov subspace algorithms, which just correspond to the conjugate gradient algorithm, and the conjugate residual algorithm, where A is a Hermitian positive definite matrix. At this point, we can give the preconditioned conjugate gradient algorithm

Algorithm 4 Preconditioner Conjugate Gradient

Input: x_0, r_0, p_0

Output: x

function ConjugateResidual(x_0, r_0, p_0)
  $x \leftarrow x_0$, $p \leftarrow r_0$, $k \leftarrow 0$
  $r \leftarrow M^{-1}(b - Ax_0)$
  while $\|r\| > eps$ **do**
    $\alpha_k := \dfrac{r_k^T r_k}{(Ap_k)^T M^{-1} p_k}$
    $x_{k+1} := x_k + \alpha_k p_k$
    $r_{k+1} := r_k - \alpha_k M^{-1} A p_k$
    $\beta_k := \dfrac{r_{k+1}^T r_{k+1}}{r_k^T r_k}$
    $p_{k+1} := r_{k+1} + \beta_k p_k$
    $k = k + 1$
  end while
  return x
end function

3.2.2 Preconditioned conjugate residual

The conjugate gradient method is constructed based on the fact that the negative gradient directions $-\nabla f(x_k)$, $-\nabla f(x_{k+1})$ of two adjacent iterations of the gradient are linearly related and intersect each other, which has high convergence speed. But its basic requirement for its stable convergence is that the matrix A is symmetric and positive definite, otherwise the convergence will be very slow or even unable to converge. The conjugate residual method differs from conjugate gradient method primarily in that it involves more numerical operations and requires more storage, but A is only required to be Hermitian. When we calculate the force gradient, the Hessian matrix is not necessarily positive definite, so the CR algorithm is more suitable. Combined with the aforementioned PCR mathematical structure.

Algorithm 5 Preconditioner Conjugate Residual

Input: x_0, r_0, p_0

Output: x

function ConjugateResidual(x_0, r_0, p_0)
  $x \leftarrow x_0$, $p \leftarrow r_0$, $k \leftarrow 0$
  $r \leftarrow M^{-1}(b - Ax_0)$
  while $\|r\| > eps$ **do**
    $\alpha_k := \dfrac{r_k^T A r_k}{(Ap_k)^T M^{-1} A p_k}$
    $x_{k+1} := x_k + \alpha_k p_k$
    $r_{k+1} := r_k - \alpha_k M^{-1} A p_k$
    $\beta_k := \dfrac{r_{k+1}^T A r_{k+1}}{r_k^T A r_k}$
    $p_{k+1} := r_{k+1} + \beta_k p_k$
    $Ap_{k+1} := Ar_{k+1} + \beta_k A p_k$
    $k = k + 1$
  end while
  return x
end function

3.2.3 Preconditioned generalized minimal residual method

Generalized minimal residual method (GMRES) is also the most commonly used method for solving large asymmetric matrix problems. The difference is that GMRES is based on the Arnoldi orthogonalization method. Similarly, we can define the Krylov subspace $\mathcal{K}_n = \mathbf{span}\{r_0, Ar_0, \cdots, A^{n-1}r_0\}$, Arnoldi can obtain a set of orthonormal basis $\{q_1, q_2, \cdots, q_n\}$ for $\mathcal{K}_n$, where $q_1 = \frac{r_0}{\|r_0\|}$. Arnoldi relational expression is $AQ_n = Q_n \tilde{H}_n$, where $Q_n = \{q_1, q_2, \cdots, q_n\}$, and $\tilde{H}_n = \begin{bmatrix} H_n \\ h_{n+1} e_n^T \end{bmatrix}$, H_n is $n \times n$ upper Hessenberg matrix. GMRES is a least squares method limited on the Krylov subspace $\mathcal{K}_n$. At each iteration of the Arnoldi algorithm, GMRES finds $z \in \mathcal{K}_n$ such that $\|r_0 - Az\|$ will be minimized. Because $\mathcal{K}_n = \mathbf{span}\{q_1, q_2, \cdots, q_n\}$, we can set $z = Q_n y,\ y \in \mathbb{R}^n$, based on the Arnoldi relation we can obtain

$$r_0 - Az = r_0 - AQ_n y = \|r_0\| q_1 - Q_{n+1} \tilde{H}_n y = Q_{n+1}\left(\|r_0\| e_1 - \tilde{H}_n y\right), \tag{3.6}$$

where $e_1 = (1, 0, \cdots, 0)^T$ is an $n+1$-dimensional vector, since $Q_{n+1}^T Q_{n+1} = I_{n+1}$,

$$\|r_0 - Az\|^2 = \left\| Q_{n+1}\left(\|r_0\| e_1 - \tilde{H}_n y\right)\right\|^2 = \left\|\|r_0\| e_1 - \tilde{H}_n y\right\|^2. \tag{3.7}$$

Therefore, the linear system $Az = r_0$ is limited to $z \in \mathcal{K}_n$, and the approximate solution of the minimum residual quantity is $z_n = Q_n y_n$, where y_n satisfies $\arg\mathbf{min}_{\mathbf{y}} \|r_0 - AQ_n y\| = \arg\mathbf{min}_{\mathbf{y}} \left\|\|r_0\| e_1 - \tilde{H}_n y\right\|$. The approximate solution of

the minimum residual quantity of the linear equation system $Ax = b$ is

$$x_n = x_0 + z_n = x_0 + Q_n y_n, \tag{3.8}$$

and the residual is

$$b - Ax_n = b - A\left(x_0 + Q_n y_n\right) = r_0 - AQ_n y_n = \|r_0\| e_1 - \tilde{H}_n y \tag{3.9}$$

Algorithm 6 Generalized Minimal Residual Method

Input: A, b, x_0

Output: x

function GMRES(A, b, x_0)
  $r_0 \leftarrow b - Ax_0$, $\beta \leftarrow$, $\|r_0\|$, $q_1 \leftarrow r_0/\beta$
  while $error = \|b - Ax_j\| / \|b\| > eps$ **do**
    $y_k = \arg \mathbf{min}_{\mathbf{y}} \left\| \beta e_1 - \tilde{H}_k y \right\|$
    $x_k = x_0 + Q_k y_k$
    $error = \dfrac{\|b - Ax_k\|}{\|b\|}$
    $k = k + 1$
  end while
  return x
end function

3.3 Numerical Integration

When calculating the stiffness matrix and mass matrix in the FEM method, it is often necessary to calculate the definite integrals of a large number of complex functions, and it is usually difficult to directly solve the analytical solutions of these complex integrals. Therefore, numerical integration methods are widely used in finite element analysis to solve such problems.

The numerical integration formula is generally be used to calculate the weighted integral $\int_a^b w(x) f(x)\, dx$, where $w(x) \geq 0$, $\int_a^b w(x)\, dx \geq 0$, the general numerical integral formula is in the following form

$$\int_a^b f(x) w(x)\, dx \approx \sum_{i=0}^{n} H_i f(x_i) \tag{3.10}$$

That is, use the weighted sum of $n+1$ functions to approximate the value of the integral. Here we need to focus on the selection of x_i and H_i, which determines the type and precision of the quadrature formula.

3.3.1 Gauss Quadrature

Firstly, we select x_i conforms to the Lagrange interpolation polynomial. At this time, $f(x)$ can be decomposed into the sum of polynomials and residual terms that $f(x) = L_n(x) + R[f]$, where

$$L_n(x) = \sum_{i=1}^{n} f(x_i) l_i(x), \tag{3.11}$$

$$R[f] = \frac{1}{(n+1)!} f^{(n+1)}(\xi)(x - x_0)(x - x_1)\cdots(x - x_n), \tag{3.12}$$

then we can convert the continuous weighted integral into

$$\int_a^b f(x) w(x) dx = \sum_{i=0}^{n} \left(\int_a^b w(x) l_i dx f(x_i) \right) + \int_a^b w(x) R[f] dx \tag{3.13}$$

$$:= \sum_{i=0}^{n} H_i f(x_i) + error \tag{3.14}$$

Our goal is to make the error term in the above formula as small as possible. It can be found that when $f(x)$ is a polynomial of no more than nth degree, the error is 0. Therefore, we call it has k degree algebraic precision, when the numerical integration formula is accurate for polynomials of no more than kth degree. The numerical integration formula contains $n+1$ H_i and $n+1$ x_i, with a total of $2n+2$ degrees of freedom, so by properly selecting the node x_i, its algebraic precision can be up to $2n+1$. At this time, we call the quadrature formula with $2n+1$ degree algebraic precision as Gaussian quadrature, and its node x_i $(i = 0, 1, \cdots, n)$ is called Gauss quadrature point, which we usually use orthogonal polynomials to to search.

Define x_i as zero-points of orthogonal polynomial $p(x)/q(x) = (x - x_0)\cdots(x - x_n)$ such that $\int_a^b w(x) p(x) q(x) dx = 0$, and satisfy $f(x) = p(x) q(x) + r(x)$, then

$$\int_a^b w(x) f(x) dx = \int_a^b w(x) p(x) q(x) dx + \int_a^b w(x) r(x) dx = \int_a^b w(x) r(x) dx \tag{3.15}$$

$$= \sum_{i=0}^{n} H_i r(x_i) = \sum_{i=0}^{n} H_i f(x_i) \tag{3.16}$$

In this way, we can find that the zeros of the orthogonal polynomials are exactly the points of integration for the Gauss quadrature formula. Orthogonal polynomials usually have multiple selections, such as Legendre polynomials, Chebyshev polynomials, Radau polynomials, Labotto polynomials, etc. Different polynomial selections will

bring different types of Gauss quadrature formulas. Here we take the Gauss-Legendre quadrature formula as an example, denote Legendre polynomial $P_n(x)$ such that

$$\int_{-1}^{1} P_m(x) P_{m+1}(x)\, dx = 0,\ m = 0, 1, \cdots, n, \tag{3.17}$$

where the definition of the Legendre polynomial can be derived from the differential equation, and when considering only the case where n is a non-negative integer over the interval $x \in [-1, 1]$, the solution of the equation $P_n(x)$ is a nth-order polynomial with respect to x such that

$$\frac{d}{dx}\left(\left(1-x^2\right)\frac{d}{dx}P_n(x)\right) + n(n-1)P_n(x) = 0, \tag{3.18}$$

$$\Rightarrow P_n(x) = \frac{1}{2^n n!}\frac{d^n}{dx^n}\left(x^2-1\right)^n = \frac{1}{2^n}\sum_{s}^{[n/2]}\frac{(-1)^s(2n-2s)!}{s!(n-s)!(n-s)!}x^{n-2s}, \tag{3.19}$$

Take $H_{n+1}(x) = \frac{1}{a_{n+1}}P_{n+1}(x)$, where a_{n+1} is the first coefficient of $P_{n+1}(x)$, and we find that the zero point of $w_{n+1}(x)$ is the same as the zero point of $P_{n+1}(x)$. Therefore, we can finally calculate the weight of Gauss-Legendre quadrature formula with the Legendre polynomial that

$$H_k = \int_{-1}^{1}\frac{H_{n+1}(x)}{(x-x_k)H'_{n+1}(x_k)}dx = \int_{-1}^{1}\frac{P_{n+1}(x)}{(x-x_k)P'_{n+1}(x_k)}dx, \tag{3.20}$$

$$\Rightarrow H_k = \frac{2(1-x_k^2)}{(n+1)^2[P_{n+1}(x_k)]^2}, \tag{3.21}$$

the corresponding numerical quadrature error is

$$R[f] = \frac{2^{2n+3}[(n+1)!]^4}{(2n+3)[(2n+2)!]^3}\cdot f^{(2n+2)}(\eta). \tag{3.22}$$

For example: We can compute the third-order Gaussian quadrature nodes as

$$P_3(x) = \frac{1}{2^3 3!}\frac{d^3}{dx^3}\left(x^2-1\right)^3 = 0 \Rightarrow x_1 = 0,\ x_2 = \sqrt{\frac{3}{5}},\ x_3 = -\sqrt{\frac{3}{5}}$$

quadrature weights are

$$H_1 = \frac{2(1-x_0^2)}{(3+1)^2[P_{3+1}(x_0)]^2} = \frac{2}{16[P_{3+1}(0)]^2} = \frac{8}{9}$$

$$H_2 = H_3 = \frac{2(1-x_2^2)}{(3+1)^2[P_{3+1}(x_2)]^2} = \frac{2\left(1-\frac{3}{5}\right)}{16\left[P_{3+1}\left(\pm\sqrt{\frac{3}{5}}\right)\right]^2} = \frac{5}{9}$$

We can give the calculation scheme of the Gauss quadrature formula of any order as

Algorithm 7 Gauss quadrature

Input: n-order, m-dim, $f(x_k)$
Output: QuadValue
function GaussQuadrature(n-order, f)
    x_k, H_k = GaussLegendre(n-order, m-dim)
    $x_k^{'} = \frac{b-a}{2}t + \frac{a+b}{2}, t \in [-1,1]^m$, $x_k \in [a,b]^m$
    QuadValue $= \left(\frac{b-a}{2}\right)^m \sum_{k=1}^{n} H_k f\left(x_k^{'}\right)$
    return QuadValue
end function
function GaussLegendre(n-order, m-dim)
    $x_k \leftarrow P_n(x_k) = \frac{1}{2^n} \sum_s^{[n/2]} \frac{(-1)^s (2n-2s)!}{s!(n-s)!(n-s)!} {x_k}^{n-2s} = 0$
    $H_k \leftarrow \frac{2\left(1-x_k^2\right)}{(n+1)^2 [P_{n+1}(x_k)]^2}$
    return x_k, H_k
end function

In this numerical integration scheme, quadrature points are not equally spaced, and the corresponding 2D and 3D formulas are

$$\int_{-1}^{1}\int_{-1}^{1} f(\xi,\eta)d\xi d\eta \approx \sum_{i=1}^{n}\sum_{j=1}^{m} H_i H_j f(\xi_i,\eta_j) \tag{3.23}$$

$$\int_{-1}^{1}\int_{-1}^{1}\int_{-1}^{1} f(\xi,\eta,\mu)d\xi d\eta d\mu \approx \sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{k=1}^{l} H_i H_j H_k f(\xi_i,\eta_j,\mu_k) \tag{3.24}$$

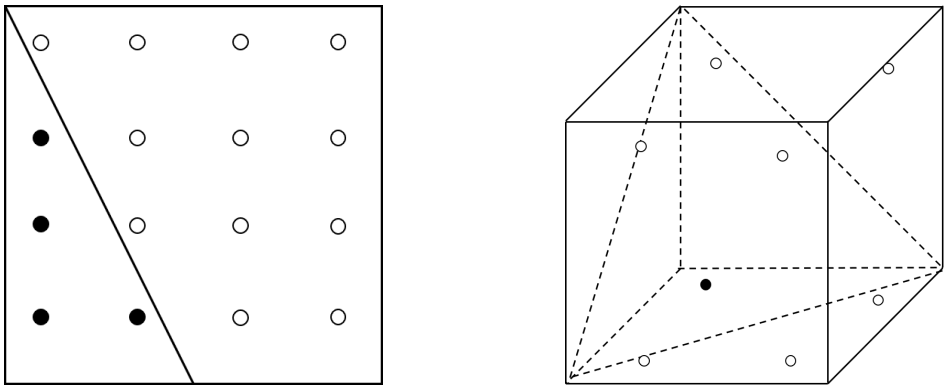
The coordinates of the quadrature points and their corresponding weight coefficients can be precomputed and are shown in the Appendix. Note that the integral interval of the Gauss quadrature is usually $[-1,1]$, so it is necessary to perform interval transformation on $[a,b]$ such that $x = \frac{b-a}{2}t + \frac{a+b}{2}$

$$\int_a^b f(x)\,dx = \frac{b-a}{2}\int_{-1}^{1} f\left(\frac{b-a}{2}t + \frac{a+b}{2}\right)dt \tag{3.25}$$

3.3.2 Hammer Quadrature

Gaussian quadrature formulas usually have accurate integration results for continuous functions in the regular interval $[-1,1]^n$, such as integration over rectangular or hexahedral elements in FEM. But triangular and tetrahedral elements are often more prevalent in FEM because of their simplicity and ease of use. In addition, in

the situation of elastic deformation, there will be discontinuous functions on the deformation elements, and the calculation of Gauss quadrature on tetrahedral elements containing such elements may have significant discretization errors, and the convergence of the solution will also be weakened. We can see from the figure below that in the triangle and tetrahedron regions, when the order of the Gauss quadrature formula is low, only a few points can effectively participate in the calculation. Therefore, we introduce Hammer quadrature [9] which is more specific to triangular and tetrahedral elements.



(a) (b)

Figur 3: Gaussian quadrature of discontinuous functions on triangular/tetrahedron

In triangular and tetrahedron elements, the natural coordinates are area coordinates and volume coordinates, and the integral has the following specific form

$$I_{2\,\mathrm{dim}} = \int_0^1 \int_0^{1-L_1} f\left(L_1, L_2, L_3\right) dL_2 dL_1 \tag{3.26}$$

$$I_{3\,\mathrm{dim}} = \int_0^1 \int_0^{1-L_1} \int_0^{1-L_2-L_1} f\left(L_1, L_2, L_3, L_4\right) dL_3 dL_2 dL_1 \tag{3.27}$$

Below we give the calculation method of the Hammer integral point and weight. First of all, we know that

$$I_{2\,\mathrm{dim}} = \int_0^1 \int_0^{1-L_1} L_1^{\alpha} L_2^{\beta} L_3^{\gamma} dL_2 dL_1 = \frac{\alpha!\beta!\gamma!}{(\alpha+\beta+\gamma+2)!}. \tag{3.28}$$

Among them $(\alpha, \beta, \gamma = 0, 1, 2, \cdots)$, if $\alpha+\beta+\gamma \leq m$ can be established exactly, but cannot be exactly when $\alpha+\beta+\gamma = m+1$, then we call the numerical integration has m degree algebraic precision. At this point we can give the theorem:

Theorem: When determining the unknowns in Hammer's numerical integration formula, the numerical integration result is equal to the exact solution, i.e.

$$\int_0^1 \int_0^{1-L_1} L_1^\alpha L_2^\beta L_3^\gamma dL_2 dL_1 = w_1 \left(\frac{1}{3}\right)^{\alpha+\beta+\gamma} + \sum_{p=1}^{n} w_{2p} \left[\left(\frac{1-a_p}{2}\right)^{\alpha+\beta} a_p^\gamma + \left(\frac{1-a_p}{2}\right)^{\alpha+\gamma} a_p^\beta + \left(\frac{1-a_p}{2}\right)^{\beta+\gamma} a_p^\alpha \right], \tag{3.29}$$

where w_1 is the weight at the center point of the triangle, and w_{2p} is the weight of the integration point distributed inside the triangle. As a condition, the maximum value of $\alpha + \beta + \gamma$ is equal to the number of independent equations that can be provided. The conclusion consists of two meanings:

1. For the condition of $\alpha + \beta + \gamma = 0$ and $\alpha + \beta + \gamma = 1$, only one independent equation can be provided.
2. For the condition of multiple combinations of $\alpha + \beta + \gamma = m(m \geq 1)$, only one independent equation can be provided.

We give a proof of this theorem in Appendix. Based on this theorem, we can re-describe the algebraic precision of the numerical integration on the area coordinates of the triangle element as: if the numerical integral formula can be established exactly for $\alpha \leq m,\ \beta = \gamma = 0$, while not exactly for $\alpha = m + 1,\ \beta = \gamma = 0$, this numerical integration formula has m-order algebraic precision, and can be established exactly for all polynomials with $\alpha + \beta + \gamma \leq m$.

For example: for a planar triangular element, when the number of quadrature points increases to 7, the numerical quadrature formula is:

$$\begin{aligned} I_7 =& w_1 f\left(\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right) + w_{21}\left[f\left(\frac{1-a}{2}, \frac{1-a}{2}, a\right) + f\left(\frac{1-a}{2}, a, \frac{1-a}{2}\right) + f\left(a, \frac{1-a}{2}, \frac{1-a}{2}\right)\right] \\ &+ w_{22}\left[f\left(\frac{1-b}{2}, \frac{1-b}{2}, b\right) + f\left(\frac{1-b}{2}, b, \frac{1-b}{2}\right) + f\left(b, \frac{1-b}{2}, \frac{1-b}{2}\right)\right] \end{aligned} \tag{3.30}$$

We can find that only one position of the 7 quadrature points can be determined, and the positions of the remaining 6 quadrature points are determined by the parameters a, b. In addition, there are three weight coefficients $w_1,\ w_{21},\ w_{22}$. According to the previous theorem, take $\alpha = 1, 2, 3, 4, 5$, respectively, $\beta = \gamma = 0$

$$\begin{cases} \frac{1}{3}w_1 + w_{21} + w_{22} = \frac{1}{6} \\ \frac{1}{9}w_1 + w_{21}\left[\frac{(1-a)^2}{2} + a^2\right] + w_{22}\left[\frac{(1-b)^2}{2} + b^2\right] = \frac{1}{12} \\ \frac{1}{27}w_1 + w_{21}\left[\frac{(1-a)^3}{4} + a^3\right] + w_{22}\left[\frac{(1-b)^2}{4} + b^2\right] = \frac{1}{20} \\ \frac{1}{81}w_1 + w_{21}\left[\frac{(1-a)^3}{8} + a^3\right] + w_{22}\left[\frac{(1-b)^2}{8} + b^2\right] = \frac{1}{30} \\ \frac{1}{243}w_1 + w_{21}\left[\frac{(1-a)^3}{16} + a^3\right] + w_{22}\left[\frac{(1-b)^2}{16} + b^2\right] = \frac{1}{42} \end{cases} \Longrightarrow \begin{cases} a = \frac{1}{21}\left(9 + 2\sqrt{15}\right) = 0.797427 \\ b = \frac{1}{21}\left(9 + 2\sqrt{15}\right) = 0.059716 \\ w_1 = \frac{9}{80} = 0.1125 \\ w_{21} = \frac{155-\sqrt{15}}{2400} = 0.0629696 \\ w_{21} = \frac{155-\sqrt{15}}{2400} = 0.0661971 \end{cases}$$

The numerical integration is accurate at this time. However, when take $\alpha = 6$, $\beta = \gamma = 0$, the analysis solution is $\int_0^1 \int_0^{1-L_1} L_1^\alpha L_2^\beta L_3^\gamma dL_2 dL_1 = \frac{1}{56}$, while the numerical integral $I_7 = \frac{1411}{79380}$. Therefore, the 7-nodes Hammer quadrature has 5th degree algebraic precision. The Hammer quadrature for 3-dim can also be calculated in a similar way. The weight coefficients and integration point coordinates are listed in Appendix. The advantage of Hammer quadrature over Gauss quadrature is that the distribution of integration nodes is more uniform in the entire triangular or tetrahedron region. Generally speaking, fewer Hammer quadrature points can achieve higher accuracy.

Algorithm 8 Hammer quadrature

Input: n-order, m-dim, $f(L_k)$, $L_k \in [0,1]^m$

Output: QuadValue

1: **function** HammerQuadrature(n-order, f)
2: w_k, L_k = HammerWeight(n-order, m-dim)
3: QuadValue = $\sum_k w_k f(L_k)$
4: **return** QuadValue
5: **end function**
6: **function** HammerWeight(n-order, m-dim)
7: $w_k, L_k \leftarrow \sum_{p=1}^{n} w_{2p}\left[\left(\frac{1-a_p}{2}\right)^{\alpha+\beta} a_p^\gamma + \left(\frac{1-a_p}{2}\right)^{\alpha+\gamma} a_p^\beta + \left(\frac{1-a_p}{2}\right)^{\beta+\gamma} a_p^\alpha\right]$, for $\alpha = 1, 2, ..., n-2, \beta = \gamma = 0$
8: **return** w_k, L_k
9: **end function**

The last thing we need to note is that when applying numerical integration, the order of numerical integration will directly affect the accuracy and workload of the calculation. If it is not selected properly, it will lead to inaccurate calculation data and even calculation failure. The principles for selecting integrals are usually as follows: 1. To ensure the accuracy of the integral; 2. To ensure that the overall stiffness matrix is non-singular.

4 Finite Element Method Simulation

After having the energy density function defined based on the continuum body, we also need to discretize the space, because what we really care about is the energy of the whole body, which is an integral on the whole space, i.e. the finite element method. FEM is a relatively general numerical method for approximating the solution of PDEs with irregular grids. In the field of graphics simulation, FEM are also common for simulating natural phenomena such as deformable bodies and fluids. In order to simulate deformation, the object is usually represented as a group of continuous non-overlapping elements. Inside the element, the physical quantity to be solved is expressed as the interpolation form of the physical quantity of the vertex of the element. The interpolation function is called shape function or basis function. For the shape function $N_i(x)$ of element node i, the displacement within the element can be approximated by the displacement at the node as

$$u(x) = \sum_i u_i N_i(x). \tag{4.1}$$

We can introduce the vector space of polynomials $\mathbb{P}_n$ as

$$\mathbb{P}_n = \left\{ P(x_1, \cdots, x_d) = \sum_{i_1+\cdots+i_d=0}^{i_1+\cdots+i_d=n} c_{i_1,\cdots,i_2} x_1^{i_1} \cdots x_d^{i_d}, c_{i_1,\cdots,i_2} \in \mathbb{R} \right\}, \tag{4.2}$$

in particular, in 2-dim and 3-dim the polynomial space can be denote as

$$\mathbb{P}_n = \left\{ P(x,y) = \sum_{i+j=0}^{i+j=n} c_{ij} x^i y^j,\ c_{ij} \in \mathbb{R} \right\}, \tag{4.3}$$

$$\mathbb{P}_n = \left\{ P(x,y,z) = \sum_{i+j+k=0}^{i+j+k=n} c_{ijk} x^i y^j z^k,\ c_{ijk} \in \mathbb{R} \right\}. \tag{4.4}$$

It is also easy to verify the dimension of $\mathbb{P}_n$

$$\dim(\mathbb{P}_n) = \sum_{i=0}^{n} \binom{d+i-1}{i} = \binom{d+n}{n} = \begin{cases} \dfrac{(n+1)(n+2)}{2},\ d=2 \\ \dfrac{(n+1)(n+2)(n+3)}{6},\ d=3 \end{cases}. \tag{4.5}$$

Meanwhile, we should take the same number of interpolation nodes. The shape function can be used to do the interpolation function of the element. The element shape function mainly depends on the type of the element, the type of nodes, and

the number of nodes in the element. Shape functions can be divided into two types: Lagrange type (which does not require the slope or curvature of the function at the nodes) and Hermite type (which requires the slope or curvature of the shape function at the nodes). No matter which type of interpolation is used, they are required to have a certain degree of smoothness in the entire interval to ensure that the bilinear form makes sense. Here we only discuss the Lagrange shape function.

According to the description by Sifakis et al [24], triangular type discretization can usually construct non-uniform meshes and better approximate regions with complex boundaries, and the triangulation discretization is one of the simplest form of volume discretization because its description of the interpolation relationship is relatively simple. Therefore, in 2D space we choose triangles as discrete elements, and in 3D space we choose tetrahedrons as discrete elements.

4.1 Calculation of shape function

We start from 2D space, on the triangular element $\Delta(1,2,3)$, we would like to construct a complete polynomial of degree n such as

$$P_n(x,y) = \sum_{i+j=0}^{n} c_{ij} x^i y^j \tag{4.6}$$

$$= c_{00} + c_{10}x + c_{01}y + c_{20}x^2 + c_{11}xy + c_{02}y^2 + \cdots + c_{n0}x^n + c_{n-1,1}x^{n-1}y + \cdots + c_{0n}y^n, \tag{4.7}$$

which have $1 + 2 + \cdots + (n+1) = \frac{1}{2}(n+1)(n+2)$ coefficients,

4.1.1 Linear FEM shape function

We will start from 2-dim triangular elements, The most common is linear FEM ($n = 1$), where the first-order element (linear element) is a piecewise first-order polynomial with three undetermined coefficients on each triangle element $I_i = [x_{i-1}, x_i] \times [y_{i-1}, y_i]$, degrees of freedom is 3, exactly determined by the three vertices of a triangle. At this time, the polynomial is a piecewise linear function

$$P_1(x,y) = c_1 + c_2 x + c_3 y \tag{4.8}$$

First, we introduce the area coordinates, let $\widetilde{\Delta}(i,j,k)$ be any 2D triangular element of area S. Take any point P within $\widetilde{\Delta}(i,j,k)$, which is (x,y) under the Cartesian coordinate system, draw lines with three vertices through point P. Divide $\widetilde{\Delta}(i,j,k)$ into three triangles $\widetilde{\Delta}i$, $\widetilde{\Delta}j$, $\widetilde{\Delta}k$, of area S_1, S_2, S_3. Let $L_1 = \frac{S_1}{S}, L_2 = \frac{S_2}{S}, L_3 = \frac{S_3}{S}$.

Apparently $L_1 + L_2 + L_3 = 1$, and

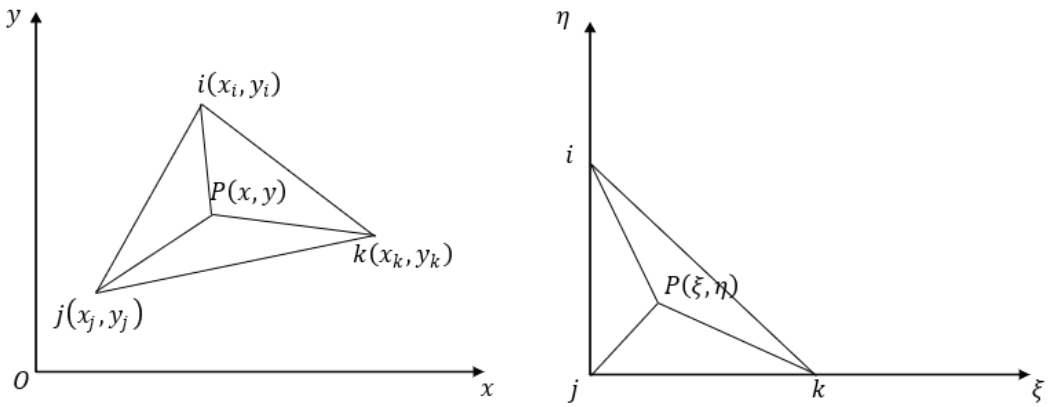
$$S = \frac{1}{2}\begin{vmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{vmatrix}, S_1 = \frac{1}{2}\begin{vmatrix} 1 & x & y \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{vmatrix}, S_2 = \frac{1}{2}\begin{vmatrix} 1 & x_1 & y_1 \\ 1 & x & y \\ 1 & x_3 & y_3 \end{vmatrix}, S_3 = \frac{1}{2}\begin{vmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x & y \end{vmatrix} \quad (4.9)$$

From this, the conversion relationship between the area coordinates and the rectangular coordinates can be established as

$$\begin{cases} L_1 = \dfrac{S_1}{S} = \dfrac{1}{S}\left[(x_2y_3 - x_3y_2) + (y_2 - y_3)\,x + (x_3 - x_2)\,y\right], \\ L_2 = \dfrac{S_2}{S} = \dfrac{1}{S}\left[(x_3y_1 - x_1y_3) + (y_3 - y_1)\,x + (x_1 - x_3)\,y\right], \\ L_3 = \dfrac{S_3}{S} = \dfrac{1}{S}\left[(x_1y_2 - x_2y_1) + (y_1 - y_2)\,x + (x_2 - x_1)\,y\right], \end{cases} \quad (4.10)$$

and for any (x, y) we have

$$\begin{pmatrix} 1 \\ x \\ y \end{pmatrix} = \begin{pmatrix} 1 \\ x_1 \\ y_1 \end{pmatrix} L_1 + \begin{pmatrix} 1 \\ x_2 \\ y_2 \end{pmatrix} L_2 + \begin{pmatrix} 1 \\ x_3 \\ y_3 \end{pmatrix} L_3, \quad (4.11)$$



(a) any triangular (b) isoparameter triangular

Figur 4: triangular element in Cartesian and isoparametric space

Define the local reference element as an isosceles triangle $\Delta(i, j, k)$ with right-angled side 1. Then for any point (ξ, η) in reference triangle element, define its displacement function as

$$\begin{cases} u(\xi, \eta) = \sum_{i=1}^{3} N_i(\xi, \eta)\, u_i, \\ v(\xi, \eta) = \sum_{i=1}^{3} N_i(\xi, \eta)\, v_i \end{cases} \quad (4.12)$$

where shape function N_i shape function can be calculated as

$$\begin{cases} N_1 = \dfrac{\Delta i}{\Delta\left(i,j,k\right)} = \dfrac{\xi}{2} \cdot 2 = \xi, \\ N_2 = \dfrac{\Delta j}{\Delta\left(i,j,k\right)} = \dfrac{\eta}{2} \cdot 2 = \eta, \\ N_3 = \dfrac{\Delta k}{\Delta\left(i,j,k\right)} = \dfrac{1-\xi-\eta}{2} \cdot 2 = 1-\xi-\eta. \end{cases} \tag{4.13}$$

From global element $\widetilde{\Delta}\left(i,j,k\right)$ to local reference element $\Delta\left(i,j,k\right)$ we can create a one-to-one affine map, together with is inverse map

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{pmatrix} \begin{pmatrix} \xi \\ \eta \end{pmatrix} + \begin{pmatrix} x_1 \\ y_1 \end{pmatrix}, \tag{4.14}$$

$$\begin{pmatrix} \xi \\ \eta \end{pmatrix} = \begin{pmatrix} \dfrac{\left(y_3 - y_1\right)\left(x - x_1\right) - \left(x_3 - x_1\right)\left(y - y_1\right)}{\left(y_2 - y_1\right)\left(x_3 - x_1\right) - \left(x_2 - x_1\right)\left(y_3 - y_1\right)} \\ \dfrac{\left(y_2 - y_1\right)\left(x - x_1\right) - \left(x_2 - x_1\right)\left(y - y_1\right)}{\left(y_2 - y_1\right)\left(x_3 - x_1\right) - \left(x_2 - x_1\right)\left(y_3 - y_1\right)} \end{pmatrix} \tag{4.15}$$

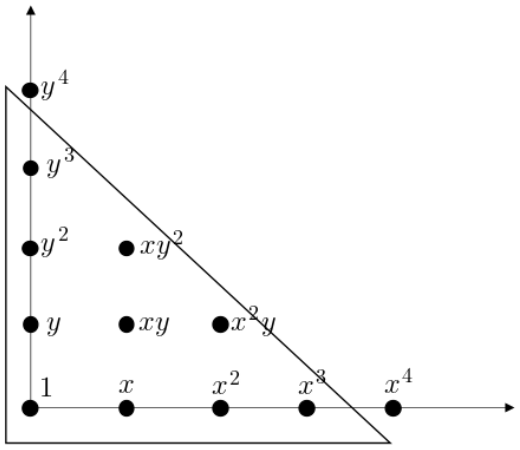
Actually, for the inverse map, we can find $\begin{pmatrix} \xi \\ \eta \end{pmatrix} = \begin{pmatrix} L_1 \\ L_2 \end{pmatrix}$, this means that interpolation on the global element is actually equivalent to interpolating on the local reference element such that

$$\begin{pmatrix} x \\ y \end{pmatrix} = \sum_{i=1}^{3} N_i\left(\xi,\eta\right) \begin{pmatrix} x_i \\ y_i \end{pmatrix} \Longleftrightarrow \begin{pmatrix} x \\ y \end{pmatrix} = \sum_{i=1}^{3} L_i\left(x,y\right) \begin{pmatrix} x_i \\ y_i \end{pmatrix} \tag{4.16}$$

For this isoparametric mapping, we can understand that any point in the global element can always uniquely be determined by a point in local reference element. Therefore, we only need to know the local reference shape function to be able to calculate the corresponding point in global coordinates. This one-to-one correspondence is true for both higher dimensions and higher order FEM.

4.1.2 General FEM shape function

In order to improve the accuracy of the FEM, it is necessary to increase the dimension of the trial function space U_h. There are usually two ways: one is to refine the meshing, so that the maximum diameter h of the element is reduced, and the node parameters are increased. The second is to increase the degree of piecewise polynomial, that is, to introduce higher order elements. For example, we can take the piecewise quadratic and cubic polynomials can increase degrees of freedom in each element.



Figur 5: Pascal triangle element, select higher order elements

According to the interpolation function 4.6, the two-dimensional Lagrange type shape function can use the Passcal triangle for analysis when constructing the polynomial displacement function. The principle of choice is to use polynomials with symmetry to ensure geometric isotropy of the polynomials, while preserving low-order terms for better approximation. When taking nth-order finite element, there are $\frac{(n+1)(n+2)}{2}$ interpolation nodes in total, and each node has two degrees of freedom. Let the displacement at the $\Delta(1,2,3)$ interpolation point be $u_1, u_2, \cdots, u_{\frac{(n+1)(n+2)}{2}}$, where $u_i = (u_i^x, u_i^y)^T$. Then the interpolation function on each node can be written as

$$\begin{cases} P_n(x_1, y_1) = u_1, \\ P_n(x_2, y_2) = u_2, \\ \vdots \\ P_n\left(x_{\frac{(n+1)(n+2)}{2}}, y_{\frac{(n+1)(n+2)}{2}}\right) = u_{\frac{(n+1)(n+2)}{2}}, \end{cases} \tag{4.17}$$

$$\Longrightarrow \begin{cases} c_1 + c_2x_1 + \cdots + c_{\frac{n(n+1)}{2}+1}x_1^n + \cdots + c_{\frac{(n+1)(n+2)}{2}}y_1^n = u_1, \\ c_1 + c_2x_2 + \cdots + c_{\frac{n(n+1)}{2}+1}x_2^n + \cdots + c_{\frac{(n+1)(n+2)}{2}}y_2^n = u_2, \\ \vdots \\ c_1 + c_2x_{\frac{(n+1)(n+2)}{2}} + \cdots + c_{\frac{(n+1)(n+2)}{2}}y_{\frac{(n+1)(n+2)}{2}}^n = u_{\frac{(n+1)(n+2)}{2}}, \end{cases} \tag{4.18}$$

After a little tidying up

$$\begin{pmatrix} 1 & x_1 & \cdots & y_1^n \\ 1 & x_2 & \cdots & y_2^n \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{\frac{(n+1)(n+2)}{2}} & \cdots & y_{\frac{(n+1)(n+2)}{2}}^n \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_{\frac{(n+1)(n+2)}{2}} \end{pmatrix} = \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_{\frac{(n+1)(n+2)}{2}} \end{pmatrix}$$

$$:= \mathcal{AC} = \mathcal{U} \Longrightarrow \mathcal{C} = \mathcal{A}^{-1}\mathcal{U} \tag{4.19}$$

In addition, for any point in the global element, there is also a nth-order interpolation function such that

$$P_n(x,y) = (1, x, y, \cdots, x^n, \cdots, y^n)\,\mathcal{C} = (1, x, y, \cdots, y^n)\,\mathcal{A}^{-1}\mathcal{U} \tag{4.20}$$

$$:= \widetilde{\mathcal{N}}\mathcal{U} = \left(\widetilde{N}_1, \widetilde{N}_2, \cdots, \widetilde{N}_{\frac{(n+1)(n+2)}{2}}\right)\mathcal{U} \tag{4.21}$$

$$= \widetilde{N}_1 u_1 + \widetilde{N}_2 u_2 + \cdots + \widetilde{N}_{\frac{(n+1)(n+2)}{2}} u_{\frac{(n+1)(n+2)}{2}}, \tag{4.22}$$

where $\tilde{\mathcal{N}} = (1, x, y, \cdots, y^n)\,\mathcal{A}^{-1}$ represents the shape function under the global element. Since we care about the shape function itself, take out $\tilde{\mathcal{N}}\mathcal{A} = (1, x, y, \cdots, y^n)$ and transpose the matrix

$$\mathcal{A}^T\widetilde{\mathcal{N}}^T = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ x_1 & x_2 & & x_{\frac{(n+1)(n+2)}{2}} \\ y_1 & y_2 & \cdots & y_{\frac{(n+1)(n+2)}{2}} \\ \vdots & \vdots & \ddots & \vdots \\ y_1^n & y_2^n & \cdots & y^n_{\frac{(n+1)(n+2)}{2}} \end{pmatrix} \begin{pmatrix} \widetilde{N}_1 \\ \widetilde{N}_2 \\ \vdots \\ \widetilde{N}_{\frac{(n+1)(n+2)}{2}} \end{pmatrix} = \begin{pmatrix} 1 \\ x \\ y \\ \vdots \\ x^n \\ \vdots \\ y^n \end{pmatrix} \tag{4.23}$$

At this point we get a Vandermonde matrix that can solve any order shape function. According to Cramer's rule, we can obtain global shape function

$$\widetilde{N}_i = \frac{\det(\mathcal{A}_i)}{\det(\mathcal{A})} \tag{4.24}$$

where $\mathcal{A}_i$ represents the matrix of the i-th column of $\mathcal{A}$ becomes $(1, x, y, \cdots, y^n)^T$. For example, when we take $n = 1$, the high-order FEM degenerates into linear FEM, and the interpolation function degenerates into

$$\begin{pmatrix} 1 & 1 & 1 \\ x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{pmatrix} \begin{pmatrix} \widetilde{N}_1 \\ \widetilde{N}_2 \\ \widetilde{N}_3 \end{pmatrix} = \begin{pmatrix} 1 \\ x \\ y \end{pmatrix} \tag{4.25}$$

$$\Longrightarrow \begin{cases} \tilde{N}_1 = \dfrac{[(x_2y_3 - x_3y_2) + (y_2 - y_3)x + (x_3 - x_2)y]}{\det\begin{pmatrix} 1 & 1 & 1 \\ x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{pmatrix}}, \\ \tilde{N}_2 = \dfrac{[(x_3y_1 - x_1y_3) + (y_3 - y_1)x + (x_1 - x_3)y]}{\det\begin{pmatrix} 1 & 1 & 1 \\ x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{pmatrix}}, \\ \tilde{N}_3 = \dfrac{[(x_1y_2 - x_2y_1) + (y_1 - y_2)x + (x_2 - x_1)y]}{\det\begin{pmatrix} 1 & 1 & 1 \\ x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{pmatrix}}. \end{cases} \tag{4.26}$$

Similarly, the same affine map exists from the global element to the local reference element is

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{pmatrix} \begin{pmatrix} \xi \\ \eta \end{pmatrix} + \begin{pmatrix} x_1 \\ y_1 \end{pmatrix}, \tag{4.27}$$

We can thus construct the Vandermonde matrix on the local reference element

$$\begin{pmatrix} 1 & 1 & \cdots & 1 \\ x_1 & x_2 & \cdots & x_{\frac{(n+1)(n+2)}{2}} \\ y_1 & y_2 & \cdots & y_{\frac{(n+1)(n+2)}{2}} \\ \vdots & \vdots & \ddots & \vdots \\ y_1^n & y_2^n & \cdots & y^n_{\frac{(n+1)(n+2)}{2}} \end{pmatrix} \begin{pmatrix} N_1 \\ N_2 \\ \vdots \\ N_{\frac{(n+1)(n+2)}{2}} \end{pmatrix} = \begin{pmatrix} 1 \\ \xi \\ \eta \\ \vdots \\ \xi^n \\ \vdots \\ \eta^n \end{pmatrix} \tag{4.28}$$

For example, if we choose linear FEM again, we can obtain shape function such that

$$\begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} N_1 \\ N_2 \\ N_3 \end{pmatrix} = \begin{pmatrix} 1 \\ \xi \\ \eta \end{pmatrix} \Longrightarrow \begin{cases} N_1 = 1 - \xi - \eta, \\ N_2 = \xi, \\ N_3 = \eta. \end{cases} \tag{4.29}$$

If we take $n = 2$, the interpolation points of the second-order FEM on the isoparametric space are $x_1 = (0, 0)$, $x_2 = \left(\frac{1}{2}, 0\right)$, $x_3 = (1, 0)$, $x_4 = \left(\frac{1}{2}, \frac{1}{2}\right)$, $x_5 = (0, 1)$, $x_6 = \left(0, \frac{1}{2}\right)$, then we have

$$\begin{pmatrix} c & 1 & 1 & 1 & 1 & 1 & 1 \\ x & 0 & \frac{1}{2} & 1 & \frac{1}{2} & 0 & 0 \\ y & 0 & 0 & 0 & \frac{1}{2} & 1 & \frac{1}{2} \\ xy & 0 & 0 & 0 & \frac{1}{4} & 0 & 0 \\ x^2 & 0 & \frac{1}{4} & 1 & \frac{1}{4} & 0 & 0 \\ y^2 & 0 & 0 & 0 & \frac{1}{4} & 1 & \frac{1}{4} \end{pmatrix} \begin{pmatrix} N_1 \\ N_2 \\ N_3 \\ N_4 \\ N_5 \\ N_6 \end{pmatrix} = \begin{pmatrix} 1 \\ \xi \\ \eta \\ \xi\eta \\ \xi^2 \\ \eta^2 \end{pmatrix} \Rightarrow \begin{cases} N_1 = 1 - 3\xi - 3\eta + 4\xi\eta + 2\xi^2 + 2\eta^2, \\ N_2 = 4\xi - 4\xi\eta - 4\xi^2, \\ N_3 = -\xi + 2\xi^2, \\ N_4 = 4\xi\eta, \\ N_5 = -\eta + 2\eta^2, \\ N_6 = 4\eta - 4\xi\eta - 4\eta^2. \end{cases} \tag{4.30}$$

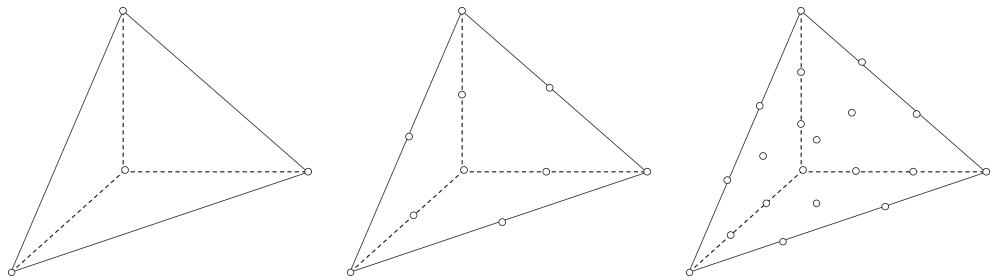
Since the stiffness matrix involves partial derivatives of N_i, we can use the chain rule to differentiate N_i with respect to ξ and η, yielding

$$\begin{cases} \dfrac{\partial N_i}{\partial \xi} = \dfrac{\partial N_i}{\partial x}\dfrac{\partial x}{\partial \xi} + \dfrac{\partial N_i}{\partial y}\dfrac{\partial y}{\partial \xi} \\ \dfrac{\partial N_i}{\partial \eta} = \dfrac{\partial N_i}{\partial x}\dfrac{\partial x}{\partial \eta} + \dfrac{\partial N_i}{\partial y}\dfrac{\partial y}{\partial \eta} \end{cases} \Longrightarrow \begin{pmatrix} \dfrac{\partial N_i}{\partial \xi} \\ \dfrac{\partial N_i}{\partial \eta} \end{pmatrix} = \begin{pmatrix} \dfrac{\partial x}{\partial \xi} & \dfrac{\partial y}{\partial \xi} \\ \dfrac{\partial x}{\partial \eta} & \dfrac{\partial y}{\partial \eta} \end{pmatrix} \begin{pmatrix} \dfrac{\partial N_i}{\partial x} \\ \dfrac{\partial N_i}{\partial y} \end{pmatrix} := J \begin{pmatrix} \dfrac{\partial N_i}{\partial x} \\ \dfrac{\partial N_i}{\partial y} \end{pmatrix} \tag{4.31}$$

inversely we have

$$\begin{pmatrix} \frac{\partial N_i}{\partial x} \\ \frac{\partial N_i}{\partial y} \end{pmatrix} = J^{-1} \begin{pmatrix} \frac{\partial N_i}{\partial \xi} \\ \frac{\partial N_i}{\partial \eta} \end{pmatrix} \tag{4.32}$$

The transition from 2D triangles to 3D tetrahedra is natural, and we can give an algorithm for computing the shape function for any tetrahedron.



(a) linear isoparametric (b) quadratic isoparametric (c) cubic isoparametric

Figur 6: Linear, quadratic, cubic reference elements in isoparametric space

At this time, the selection of the interpolation polynomial obeys the tetrahedron number, the number of n-order FEM nodes is $\frac{(n+1)(n+2)(n+3)}{6}$, and the affine map is

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x_2 - x_1 & x_3 - x_1 & x_4 - x_1 \\ y_2 - y_1 & y_3 - y_1 & y_4 - y_1 \\ z_2 - z_1 & z_3 - z_1 & z_4 - z_1 \end{pmatrix} \begin{pmatrix} \xi \\ \eta \\ \mu \end{pmatrix} + \begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix} \tag{4.33}$$

Algorithm 9 Isoparametric shape function

Input: n-order, $n = 1, 2, ...$

Output: $\{N_1, N_2, ..., N_{\frac{(n+1)(n+2)(n+3)}{6}}\}$, $\{\frac{\partial N_i}{\partial x}, \frac{\partial N_i}{\partial y}, \frac{\partial N_i}{\partial x}\}$

function IsoparametricShapeFunction(n-order)
    x_i, y_i, z_i = IsoparametricTetrahedron(n-order)
    Calculate $N(\xi^i, \eta^j, \mu^k)$
    Calcualte Jacobian $\frac{\partial(x,y,z)}{\partial(\xi,\eta,\mu)}$
    Calculate derivative $\frac{\partial(N_i)}{\partial(x,y,z)}$
    return $\{N_1, N_2, ..., N_{\frac{(n+1)(n+2)(n+3)}{6}}\}$
end function
function IsoparametricShapeFunction(n-order)
    $(x_i, y_i, z_i) = (0, 0, 0), (1, 0, 0), \cdots, (0, 0, 1)$
    return $x_i, y_i, z_i, \cdots, x_i^n, y_i^n, z_i^n$
end function

where calculate $N(\xi^i, \eta^j, \mu^k)$ will solve the linear equation

$$\begin{pmatrix} N_1 \\ N_2 \\ \vdots \\ N_{\frac{(n+1)(n+2)(n+3)}{6}} \end{pmatrix} = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ x_1 & x_2 & \cdots & x_{\frac{(n+1)(n+2)(n+3)}{6}} \\ y_1 & y_2 & \cdots & y_{\frac{(n+1)(n+2)(n+3)}{6}} \\ z_1 & z_2 & \cdots & z_{\frac{(n+1)(n+2)(n+3)}{6}} \\ \vdots & \vdots & \ddots & \vdots \\ z_1^n & z_2^n & \cdots & z_{\frac{(n+1)(n+2)(n+3)}{6}}^n \end{pmatrix}^{-1} \begin{pmatrix} 1 \\ \xi \\ \eta \\ \mu \\ \vdots \\ \xi^n \\ \vdots \\ \eta^n \\ \vdots \\ \mu^n \end{pmatrix} \tag{4.34}$$

and calculate derivative of shape function is in such form

$$\begin{pmatrix} \frac{\partial N_i}{\partial x} \\ \frac{\partial N_i}{\partial y} \\ \frac{\partial N_i}{\partial z} \end{pmatrix} = \begin{pmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} & \frac{\partial z}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} & \frac{\partial z}{\partial \eta} \\ \frac{\partial x}{\partial \mu} & \frac{\partial y}{\partial \mu} & \frac{\partial z}{\partial \mu} \end{pmatrix}^{-1} \begin{pmatrix} \frac{\partial N_i}{\partial \xi} \\ \frac{\partial N_i}{\partial \eta} \\ \frac{\partial N_i}{\partial \mu} \end{pmatrix} \tag{4.35}$$

4.2 Calculate deformation gradient in FEM way

4.2.1 Linear FEM deformation gradient

We mentioned in Chapter 2 that on discrete tetrahedron elements, the deformation mapping function ϕ. can be represented by a piecewise linear function. For any point X in tetrahedron element, mapping function can be represented as

$$\phi(X) = A_i X + b_i, A_i \in \mathbb{R}^{3\times 3}, b_i \in \mathbb{R}^3, \tag{4.36}$$

The above formula implies the barycentric coordinate interpolation method of the tetrahedron element. Using the deformation gradient $F = \frac{\partial \phi}{\partial X} = A_i$, the formula above can be expressed as

$$\phi(X) = FX + b, \tag{4.37}$$

For any tetrahedron, use X_1, X_2, X_3, X_4 to represent the four vertices, and x_1, x_2, x_3, x_4 to represent the deformed position, then for each deformation function $x_i = \phi(X_i)$, we can obtain

$$\begin{cases} x_1 = FX_1 + b \\ x_2 = FX_2 + b \\ x_3 = FX_3 + b \\ x_4 = FX_4 + b \end{cases} \Rightarrow \begin{cases} x_1 - x_4 = F(X_1 - X_4) \\ x_2 - x_4 = F(X_2 - X_4) \\ x_3 - x_4 = F(X_3 - X_4) \end{cases} \tag{4.38}$$

According to the conversion of the above formula, eliminate the translation vector b, and the matrix representation can be obtained as

$$\begin{bmatrix} x_1 - x_4 & x_2 - x_4 & x_3 - x_4 \end{bmatrix} = F \begin{bmatrix} X_1 - X_4 & X_2 - X_4 & X_3 - X_4 \end{bmatrix},$$
$$D_s = FD_m \Longrightarrow F = D_s D_m^{-1} \tag{4.39}$$

where D_m is the vector difference matrix in the initial coordinate system, which is a constant matrix and can be pre-calculated depending on the shape of the deformable body. D_s is the vector difference matrix in the deformed coordinate system.

4.2.2 General FEM deformation gradient

The aforementioned way of calculating deformation gradient is based on purely geometry, which is usually used to describe the situation where the deformation within an element is linear. But when the deformation is no longer linear, that is, the position of a point in the element after deformation cannot be calculated by linear interpolation of the element vertices, this method is no longer applicable.

Generally, we can directly calculate according to numerical methods deformation gradient [16]. From the discussion in the shape function chapter, we know that x can be written as a combination of global coordinates and local reference shape functions, that is $x = \sum_i x_i N_i$, where x_i is 3-dim vector. Define $b = (\xi, \eta, \mu)$. According to the chain rule

$$\begin{aligned} F = \frac{\partial x}{\partial X} &= \frac{\partial x}{\partial b} \cdot \frac{\partial b}{\partial X} = \frac{\partial x}{\partial b} \cdot \left(\frac{\partial X}{\partial b} \right)^{-1} \\ &= \frac{\partial}{\partial b} \left(\sum_i x_i N_i \right) \frac{\partial}{\partial b} \left(\sum_i X_i N_i \right)^{-1} \\ &= \left(\sum_i x_i \frac{\partial N_i}{\partial b} \right) \left(\sum_i X_i \frac{\partial N_i}{\partial b} \right)^{-1} := D_s D_m^{-1} \end{aligned}$$

For example, we take linear FEM such that $N_1 = \xi$, $N_2 = \eta$, $N_3 = \mu$, $N_4 = 1 - \xi - \eta - \mu$, then

$$\begin{aligned} F_{linear} &= \left(\sum_{i=1}^{4} x_i \frac{\partial N_i}{\partial (\xi, \eta, \mu)} \right) \left(\sum_{i=1}^{4} X_i \frac{\partial N_i}{\partial (\xi, \eta, \mu)} \right)^{-1} \\ &= \left((x_1, x_2, x_3, x_4) \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ -1 & -1 & -1 \end{pmatrix} \right) \left((X_1, X_2, X_3, X_4) \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ -1 & -1 & -1 \end{pmatrix} \right)^{-1} \\ &= \begin{bmatrix} x_1 - x_4 & x_2 - x_4 & x_3 - x_4 \end{bmatrix} \cdot \begin{bmatrix} X_1 - X_4 & X_2 - X_4 & X_3 - X_4 \end{bmatrix}^{-1} \end{aligned}$$

It is worth noting that in linear FEM, $\frac{\partial N_i}{\partial(\xi,\eta,\mu)}$ is a constant matrix, which means that the deformation gradient is the same everywhere within an element, which is also the main reason for the limit of linear FEM accuracy. Because FEM usually cannot meet the ideal condition in the process of meshing, that each finite element is small enough (in other words, the mesh quality is high enough). In the case of such poor grid quality, if we still assume that the deformation gradient is the same everywhere, then the linear FEM will cause a huge error.

High-order FEM can solve this problem to a certain extent, taking quadratic FEM as an example. There are 10 nodes in an element, at this time, the 10 shape functions and their corresponding partial derivatives are as below (more calculation details can be referred in the appendix)

$$\begin{pmatrix} N_0 \\ N_1 \\ N_2 \\ N_3 \\ \vdots \end{pmatrix} = \begin{pmatrix} \xi\left(2\xi-1\right) \\ \eta\left(2\eta-1\right) \\ \mu\left(2\mu-1\right) \\ \left(\xi+\eta+\mu-1\right)\left(2\xi+2\eta+2\mu-1\right) \\ \vdots \end{pmatrix} \tag{4.40}$$

$$\begin{pmatrix} dN_0 \\ dN_1 \\ dN_2 \\ dN_3 \\ \vdots \end{pmatrix} = \begin{pmatrix} 4\xi-1 & 0 & 0 \\ 0 & 4\eta-1 & 0 \\ 0 & 0 & 4\mu-1 \\ 4\xi+4\eta+4\mu-3 & 4\xi+4\eta+4\mu-3 & 4\xi+4\eta+4\mu-3 \\ \vdots & \vdots & \vdots \end{pmatrix} \tag{4.41}$$

We can find that the partial derivative of N_i is not a constant matrix in the reference element at this time. Therefore, F will be different inside the tetrahedral element according to the position, which can describe the deformation of the object more accurately in the case of poor mesh quality.

4.3 Calculate force & force gradient in FEM way

4.3.1 Compute force

In the former chapter, we introduce energy density function $\Psi\left(x\right)=\Psi\left(\phi\left(X\right)\right)$ for $x=\phi\left(X\right)$. Recall that $\phi\left(X\right)\approx FX+t$, we have $\Psi\left(x\right)=\Psi\left(FX+t\right)$. Since in high-order FEM, F is not the same everywhere within an element, the elasticity energy should be like

$$E\left(x\right)=\int_{\Omega}\Psi\left(F\right)dX=W_i\sum_{e_i}\sum_{\substack{quadrature\\points}}w_{points}\Psi\left(F_i\left(x\right)\right) \tag{4.42}$$

where quadrature points are the interpolation nodes of high-order FEM, w_{points} is the weight of the interpolation node, and W_i is the volume of the element. When the discrete way of energy degenerates to linear FEM, the numerical product term becomes $\Psi(F(x))$. According to the kinetic energy theorem, the internal force of an object can be represent by the negative derivative of the elastic energy to the displacement. At this time, combined with the Piola-Kirchhoff stress, we can deduce the expression of the force f_j on each node in an element under high-order FEM for

$$f_j = -\frac{\partial E_i}{\partial x_j} = -W_i \sum_{\substack{quadrature \\ points}} w_{points} \frac{\partial F_i(\xi,\eta,\mu)}{\partial x_j} : P_i \tag{4.43}$$

$$= -W_i \sum_{\substack{quadrature \\ points}} w_{points} \frac{\partial D_s(\xi,\eta,\mu)}{\partial x_j} D_m^{-1}(\xi,\eta,\mu) : P_i \tag{4.44}$$

where $\frac{\partial F_i(\xi,\eta,\mu)}{\partial x_j}$ is a 4th order tensor and P_i is a 2nd order tensor (3×3 matrix), so we need to perform tensor contraction.

For linear FEM, define $x_j = \left(x_j^0, x_j^1, x_j^2\right)$ stands for 3-dimension, $j = 0, 1, 2$ stands for any three points of a tetrahedron. For any 2nd order tensor $\frac{\partial F}{\partial x_j^k} = \frac{\partial D_s}{\partial x_j^k} D_m^{-1}$, we can calculate its variable part

$$\frac{\partial D_s}{\partial x_0^0} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}^T, \cdots, \frac{\partial D_s}{\partial x_2^2} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}^T, \tag{4.45}$$

Therefore, we can obtain $\frac{\partial F}{\partial x_j^k} = e_k e_j^T D_m^{-1}$, at this point we can perform tensor contraction that

$$\frac{\partial F}{\partial x_j^k} : P = e_k e_j^T D_m^{-1} : P = \mathrm{tr}\left(D_m^{-T} e_j e_k^T P\right) = \mathrm{tr}\left(e_k^T P D_m^{-T} e_j\right) = e_k^T P D_m^{-T} e_j = \left[P D_m^{-T}\right]_{kj} \tag{4.46}$$

From this we know $\frac{\partial \Psi}{\partial x_j}$ is the j-th column of PD_m^{-T}, because the internal forces of the tetrahedron sum should be 0. Therefore, element force should be a 3×4 matrix that

$$H = [f_0, f_1, f_2, f_3] = -W_i \left[\left[P D_m^{-T}\right]_0, \left[P D_m^{-T}\right]_1, \left[P D_m^{-T}\right]_2, -\sum_j \left[P D_m^{-T}\right]_j \right]. \tag{4.47}$$

The force calculation of high-order FEM is basically the same as the linear FEM calculation pipeline, but $\frac{\partial D_s}{\partial x_j^k}$ and D_m^{-T} will become a parameterized matrix about (ξ, η, μ), and the number of nodes j will also increase, so that the form of force matrix H cannot be calculated in advance. We need to update the deformation gradient

under different (ξ, η, μ) in real time when performing tensor contraction, and then use the numerical quadrature formula to calculate the final force.

Algorithm 10 Force calculation

Input: Deformation gradient F, Piola–Kirchhoff stress P

Output: Force matrix H^*

function ForceCalculation(F, P)
  for each node $x_i = (x_i^0, x_i^1, x_i^2)$ **do**
    for each (ξ, η, μ) **do**
      $D_s(\xi, \eta, \mu) = \left[\sum_i x_i \frac{\partial N_i}{\partial \xi}, \sum_i x_i \frac{\partial N_i}{\partial \eta}, \sum_i x_i \frac{\partial N_i}{\partial \mu}\right]$
      $D_m(\xi, \eta, \mu) = \left[\sum_i X_i \frac{\partial N_i}{\partial \xi}, \sum_i X_i \frac{\partial N_i}{\partial \eta}, \sum_i X_i \frac{\partial N_i}{\partial \mu}\right]$
      $F(\xi, \eta, \mu) = D_s D_m^{-1}, P = P(F)$
      $\left(\frac{\partial D_s}{\partial x_i}\right)^T = \left(\left(\frac{\partial D_s}{\partial x_i^0}\right)^T, \left(\frac{\partial D_s}{\partial x_i^1}\right)^T, \left(\frac{\partial D_s}{\partial x_i^2}\right)^T\right)^T$
      $f_i = -W_i \left[tr\left(\left(\frac{\partial D_s}{\partial x_i^0}\right)^T P D_m^{-T}\right), tr\left(\left(\frac{\partial D_s}{\partial x_i^1}\right)^T P D_m^{-T}\right), tr\left(\left(\frac{\partial D_s}{\partial x_i^2}\right)^T P D_m^{-T}\right)\right]^T$
      $H = -W_i [f_1, f_2, \cdots, f_n]$
    end for
    $H^* = numerical\_quadrature(H)$
  end for
  return H^*
end function

4.3.2 Compute force gradient

Recall that in implicit FEM, apart from $\nabla_x E(x)$, we still need $\nabla_x^2 E(x)$, i.e.

$$\frac{\partial f_i}{\partial x} = -\frac{\partial^2 E_i}{\partial x^2} = -W_i \sum_{\substack{quadrature \\ points}} w_{points} \frac{\partial F_i}{\partial x} : \frac{\partial P_i}{\partial F_i} : \left(\frac{\partial F_i}{\partial x}\right)^T, \tag{4.48}$$

among which, we have already known $\frac{\partial F_i}{\partial x}$ and $\frac{\partial F_i^T}{\partial x}$, only need to compute $\frac{\partial P_i}{\partial F_i}$, which is 4th order tensor. Take Neo-Hookean as an example

$$P_{NH} = \mu\left(F - F^{-T}\right) + \lambda \log(J) F^{-T} \tag{4.49}$$

$$\frac{dP_{NH}}{dF} = \mu\left(dF + F^{-T} dF^T F^{-T}\right) + \lambda \frac{d(\det F)}{J} F^{-T} - \lambda \log(J) F^T dF^T F^{-T} \tag{4.50}$$

$$= \mu dF + (\mu - \lambda \log(J)) F^{-T} dF^T F^{-T} + \lambda tr\left(F^{-1} dF\right) F^{-T} \tag{4.51}$$

In implicit FEM, assuming that there are n nodes on a tetrahedral element, the dimension of an element Hessian matrix is

$$\dim\left(vec\left(\frac{\partial F}{\partial x}\right)^T vec\left(\frac{\partial P}{\partial F}\right) vec\left(\frac{\partial F}{\partial x}\right)\right) = (3n\times 9)\times(9\times 9)\times(9\times 3n) = 3n\times 3n \tag{4.52}$$

For linear FEM, the element Hessian matrix is a 12×12 matrix, firstly we focus on the first tensor contraction $\frac{\partial F_i}{\partial x} : \frac{\partial P_i}{\partial F_i} := dP$, because in the previous chapter we already know $\frac{\partial F_i}{\partial x} : \frac{\partial P}{\partial F_{ij}} = \frac{\partial P}{\partial F_{ij}} D_m^{-T}$, then

$$\begin{aligned}\frac{\partial F_i}{\partial x} : \begin{pmatrix} \begin{pmatrix}\frac{\partial P}{\partial F_{00}}\\ \frac{\partial P}{\partial F_{10}}\\ \frac{\partial P}{\partial F_{20}}\end{pmatrix} & \begin{pmatrix}\frac{\partial P}{\partial F_{01}}\\ \frac{\partial P}{\partial F_{11}}\\ \frac{\partial P}{\partial F_{21}}\end{pmatrix} & \begin{pmatrix}\frac{\partial P}{\partial F_{02}}\\ \frac{\partial P}{\partial F_{12}}\\ \frac{\partial P}{\partial F_{22}}\end{pmatrix}\end{pmatrix} &= \begin{pmatrix} \begin{pmatrix}\frac{\partial P}{\partial F_{00}}\\ \frac{\partial P}{\partial F_{10}}\\ \frac{\partial P}{\partial F_{20}}\end{pmatrix} D_m^{-T} & \begin{pmatrix}\frac{\partial P}{\partial F_{01}}\\ \frac{\partial P}{\partial F_{11}}\\ \frac{\partial P}{\partial F_{21}}\end{pmatrix} D_m^{-T} & \begin{pmatrix}\frac{\partial P}{\partial F_{02}}\\ \frac{\partial P}{\partial F_{12}}\\ \frac{\partial P}{\partial F_{22}}\end{pmatrix} D_m^{-T}\end{pmatrix} \\ &:= \begin{pmatrix}(dP_{00}) & (dP_{01}) & (dP_{02})\\ (dP_{10}) & (dP_{11}) & (dP_{12})\\ (dP_{20}) & (dP_{21}) & (dP_{22})\end{pmatrix}\end{aligned} \tag{4.53}$$

Then we need to calculate $dP : \left(\frac{\partial F_i}{\partial x}\right)^T$, similar to the previous method of tensor contraction, we can get $dP : \left(\frac{\partial F_i}{\partial x}\right)^T = (dP_{ij})\, D_m^{-T}$. At this point, all the variables are formed into a Hessian matrix K consists of $\frac{\partial H}{\partial x_j^k}$, and its basic structure is as follows

$$K = \begin{bmatrix} \left(\frac{\partial H}{\partial x_0^0}\right)_{00} & \cdots & \left(\frac{\partial H}{\partial x_3^2}\right)_{00} \\ \left(\frac{\partial H}{\partial x_0^0}\right)_{10} & \cdots & \left(\frac{\partial H}{\partial x_3^2}\right)_{10} \\ \vdots & & \vdots \\ \left(\frac{\partial H}{\partial x_0^0}\right)_{22} & \cdots & \left(\frac{\partial H}{\partial x_3^2}\right)_{22} \\ -\left(\frac{\partial H}{\partial x_0^0}\right)_{00} - \left(\frac{\partial H}{\partial x_0^0}\right)_{10} - \left(\frac{\partial H}{\partial x_1^1}\right)_{20} & \cdots & -\left(\frac{\partial H}{\partial x_3^2}\right)_{00} - \left(\frac{\partial H}{\partial x_3^2}\right)_{10} - \left(\frac{\partial H}{\partial x_3^2}\right)_{20} \\ \vdots & & \vdots \\ -\left(\frac{\partial H}{\partial x_1^1}\right)_{02} - \left(\frac{\partial H}{\partial x_1^1}\right)_{12} - \left(\frac{\partial H}{\partial x_1^1}\right)_{22} & \cdots & -\left(\frac{\partial H}{\partial x_3^2}\right)_{02} - \left(\frac{\partial H}{\partial x_3^2}\right)_{12} - \left(\frac{\partial H}{\partial x_3^2}\right)_{22} \end{bmatrix} \tag{4.54}$$

High order FEM algorithm does not have the property of $\frac{\partial D_s}{\partial x_j^k} = e_k e_j^T$ so the calculation of the Hessian matrix needs to be done in a more complicated way. The first calculation method is to convert the force gradient into three matrix calculations that $vec\left(\frac{\partial F}{\partial x}\right)^T vec\left(\frac{\partial P}{\partial F}\right) vec\left(\frac{\partial F}{\partial x}\right)$, but large sparse matrix calculation is usually a pretty expensive thing, especially for high-order FEM, the matrix dimension will increase by three times of nodes quantity, so we still divide it into elements for calculation. Firstly, we calculate the first tensor contraction

$$dP := \begin{pmatrix} \left(\frac{\partial F}{\partial x_0^0}\right) & \left(\frac{\partial F}{\partial x_0^1}\right) & \left(\frac{\partial F}{\partial x_0^2}\right) \\ \vdots & \vdots & \vdots \\ \left(\frac{\partial F}{\partial x_n^0}\right) & \left(\frac{\partial F}{\partial x_n^1}\right) & \left(\frac{\partial F}{\partial x_n^2}\right) \end{pmatrix} : \frac{\partial P}{\partial F} = \begin{pmatrix} \left(\frac{\partial P}{\partial F} : \frac{\partial F}{\partial x_0^0}\right) & \left(\frac{\partial P}{\partial F} : \frac{\partial F}{\partial x_0^1}\right) & \left(\frac{\partial P}{\partial F} : \frac{\partial F}{\partial x_0^2}\right) \\ \vdots & \vdots & \vdots \\ \left(\frac{\partial P}{\partial F} : \frac{\partial F}{\partial x_n^0}\right) & \left(\frac{\partial P}{\partial F} : \frac{\partial F}{\partial x_n^1}\right) & \left(\frac{\partial P}{\partial F} : \frac{\partial F}{\partial x_n^2}\right) \end{pmatrix} \tag{4.55}$$

where we assume that there are n nodes in an element, then the first tensor contraction dP is a $(n \times 3) \times (3 \times 3)$ 4th order tensor, and the calculation method is similar to force calculation. Next, we calculate the second tensor contraction $dH = dP : \left(\frac{\partial F}{\partial x}\right)^T$, we choose dP_i to be a 3×3 2nd tensor in dP, and dF_i to be a 3×3 2nd tensor in $\frac{\partial F}{\partial x}$, $i = 0, \ldots, 3n-1$. Then for any 3×1 vector dH_i^j $(j = 0, \ldots, n-1)$, we can calculate

$$dH_i^j = -W_i \left[tr\left(dP_i (dF_{3j})^T\right), tr\left(dP_i (dF_{3j+1})^T\right), tr\left(dP_i (dF_{3j+2})^T\right) \right] \tag{4.56}$$

Then we can assemble the Hessian matrix by columns with dH_i^j that

$$dH_i = \left[dH_i^0, dH_i^1, \ldots, dH_i^n\right]^T \tag{4.57}$$

Algorithm 11 Force gradient calculation

Input: Deformation gradient F, Piola–Kirchhoff stress P, PK1 gradient $\frac{\partial P}{\partial F}$
Output: Hessian matrix dH^*

function ForceCalculation(F, P, $\frac{\partial P}{\partial F}$)
  for each node $x_i = (x_i^0, x_i^1, x_i^2)$ **do**
    for each (ξ, η, μ) **do**
      $D_s(\xi, \eta, \mu) = \left[\sum_i x_i \frac{\partial N_i}{\partial \xi}, \sum_i x_i \frac{\partial N_i}{\partial \eta}, \sum_i x_i \frac{\partial N_i}{\partial \mu}\right]$
      $D_m(\xi, \eta, \mu) = \left[\sum_i X_i \frac{\partial N_i}{\partial \xi}, \sum_i X_i \frac{\partial N_i}{\partial \eta}, \sum_i X_i \frac{\partial N_i}{\partial \mu}\right]$
      $F(\xi, \eta, \mu) = D_s D_m^{-1}, P = P(F), \frac{\partial P}{\partial F} = \frac{\partial P}{\partial F}(F)$
      $\left(\frac{\partial D_s}{\partial x_i}\right)^T = \left(\left(\frac{\partial D_s}{\partial x_i^0}\right)^T, \left(\frac{\partial D_s}{\partial x_i^1}\right)^T, \left(\frac{\partial D_s}{\partial x_i^2}\right)^T\right)^T$
      $dP = \left(\left(\frac{\partial P}{\partial F} : \frac{\partial F}{\partial x_i^0}\right)_{3\times3}, \left(\frac{\partial P}{\partial F} : \frac{\partial F}{\partial x_i^1}\right)_{3\times3}, \left(\frac{\partial P}{\partial F} : \frac{\partial F}{\partial x_i^2}\right)_{3\times3}\right)_{n\times3}$
      $dH_i^j = -W_i \left[tr\left(dP_i (dF_{3j})^T\right), tr\left(dP_i (dF_{3j+1})^T\right), tr\left(dP_i (dF_{3j+2})^T\right) \right]$
      $dH_i = [dH_i^0, dH_i^1, \ldots, dH_i^n]^T$
      $dH = [dH_0, dH_1, \ldots, dH_{3n-1}]$
    end for
    $dH^* = numerical\_quadrature(dH)$
  end for
  return dH^*
end function

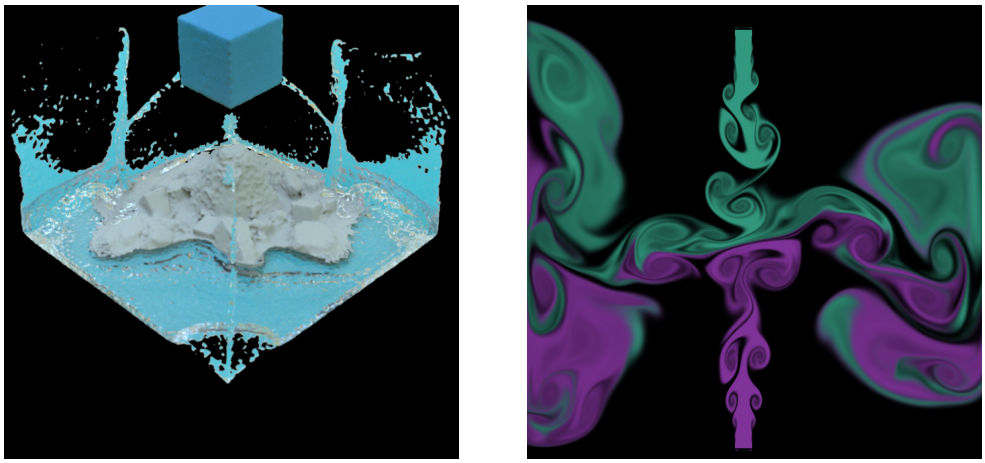
5 Numerical Results & Summarize

5.1 Taichi programming language

Taichi is a domain-specific language (DSL) embedded in Python, the code written in Taichi is valid Python code but is compiled and executed in Taichi's runtime, and the rest is treated as native Python code and executed in Python's virtual machine. The code inside a kernel or a Taichi function is in the Taichi scope. The code in the Taichi scope is compiled by Taichi's runtime and executed in parallel on CPU or GPU devices for high-performance computation.

Taichi uses $@ti.kernel$ to decorate Taichi kernels, which can be called from the Python scope to perform computations. Taichi functions are defined as $@ti.func$, they should be called in Taichi kernels or other Taichi functions. Taichi kernels and functions share the same language rules as Python, but the Taichi front-end compiler will convert them into language specifications such as compilable, static types, parallelism, differentiability, etc. It is worth noting that $ti.kernel$ corresponds to $\_\_global\_\_$ in CUDA, while $ti.func$ corresponds to $\_\_device\_\_$.

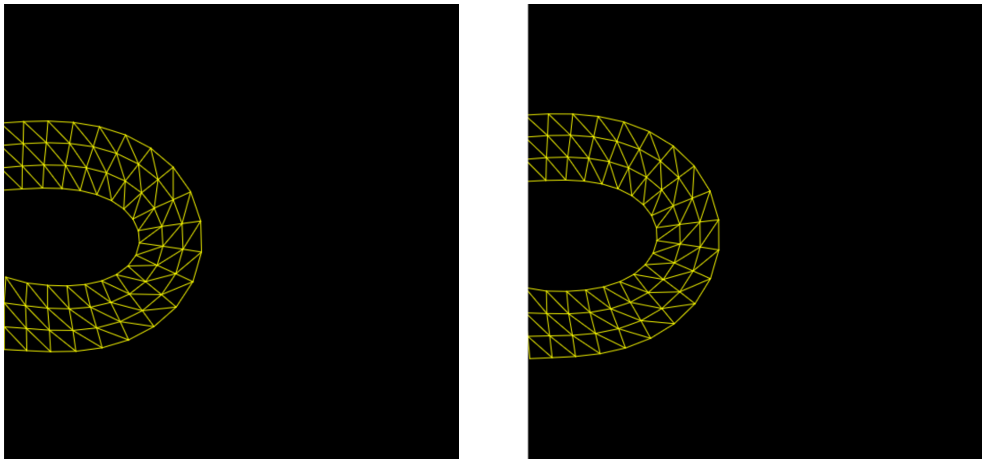
Taichi can well support multi-threaded computing that people can choose $arch = ti.cpu$ or $ti.gpu$ to initialize Taichi and run on CPU or GPU (CUDA, OpenGL, Vulkan...). The outermost range for loops in the Taichi kernel will be automatically parallelized, which usually consists of two forms, $range - for\ loops$ or $struct - for\ loops$. Range-for loops are no different from Python for loops, which can be nested. Struct-for loops are the key to sparse computing in Taichi, as it will only loop over active elements in a sparse field, while in the dense field, all elements are in active state.



(a) Material Point Method (b) MGPCG Euler Grid

Figur 7: MPM: 82 lines of particle calculation code, 650,000 particles, 65fps; Euler grid: 290 lines of grid calculation code, 1080×1080 grid, 32fps

Taichi also comes with an *autodiff* system, which can support gradient evaluation through *kernel.grad*(). A common problem in physics simulation is that it is not a big problem to calculate the energy of an object, but it is always difficult to calculate the force on each particle or node, such as FEM. Usually we can obtain the force according to $f = -\frac{\partial E}{\partial x}$. After completing the kernel function for calculating the potential energy, we can directly use Taichi's autodiff system to obtain the derivative without converting it into tensor contraction $\frac{\partial F}{\partial x} : P$ (although some performance will be sacrificed due to lack of mathematics). In addition, a deeper application of this autodiff system is differentiable simulation, as a controller optimization system equipped with differentiable simulators converges one to four orders of magnitude faster than those using model-free reinforcement learning algorithms.



(a) FEM with Autodiff: 15fps (b) FEM with tensor contraction: 30fps

Figur 8: FEM with and without autodiff system

5.2 Constitutive models

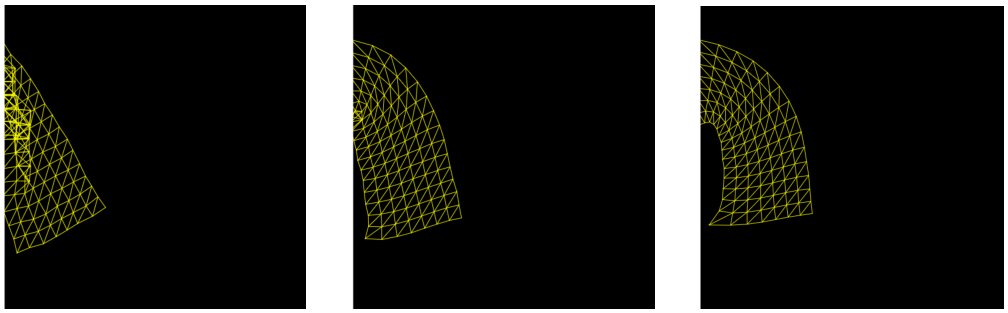
The three ways to calculate force in the StVK, Corotated linear, Neo-Hookean constitutive model are separately as below:

```
StVK:
stvk=0.5*((F.transpose() @ F)-Eye)
P= F@(2*LameMu[None]* stvk +LameLa[None]*stvk.trace()*Eye)
H = W[i] * P @ (B[i].transpose())
```

```
Corotated linear:
P = 2*LameMu[None]*(F-R) + LameLa[None]*((R.transpose())@F-Eye).trace()*R
H = W[i] * P @ (B[i].transpose())
```

Note: The reason why J takes max is to avoid the calculation error of the log term due to numerical precision problems

```
Neo-Hookean:
J = max(F.determinant(), 0.01)
P = LameMu[None] * (F - F_inv_T) + LameLa[None] * ti.log(J) * F_inv_T
H = W[i] * P @ (B[i].transpose())
```



(a) StVK model (b) Corotated linear model (c) Neo-Hookean model

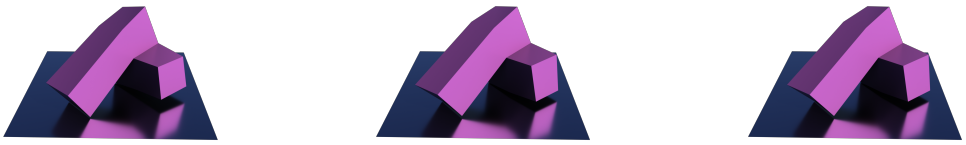
Figur 9: Cantilever beam compression with different constitutive models

We can see from figure 9 that these three models can maintain a certain degree of rotational invariance, and there is no obvious error in the bending process of the cantilever beam. But at the junction of the cantilever beam, different models have great differences in the resistance energy to compression. Among them, when the StVK model is strongly compressed, and the pressure exceeds the threshold, there will be obvious volume compression and generate reverse recovery force, which will cause the local entanglement and inversion of the material near the connection. Corotated linear also shows slight model folding. The Neo-Hookean model does not have such problems, because the existence of the $\log(J)$ term, along with the reduction in volume, the resulting restoring force will gradually increase, resulting in excellent compression resistance.

5.3 Numerical analysis

5.3.1 Explicit & implicit method

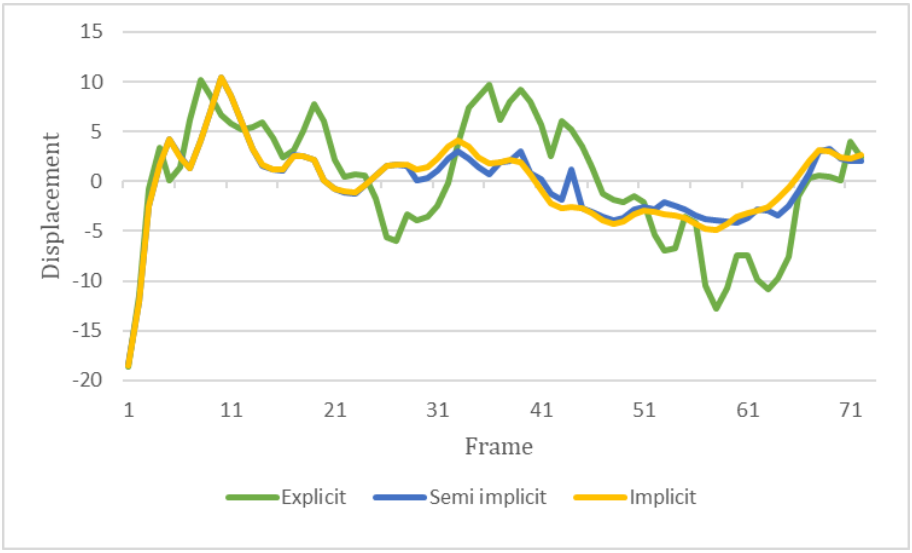
In the experiment, we tested the effect of the irregular cube deformable body model falling from the air and contacting the floor. All the external forces of the object are the same. The model contains a total of 20 vertices and 24 tetrahedron elements.



(a) Explicit (b) Semi-Implicit (c) Implicit

Figur 10: Simulation effect of explicit, semi-implicit, implicit numerical method, $dt = 1e^{-3}$, frame=45

The change curve of the sum of the displacements of the model at the force point is shown in the figure below. The figure shows the deformation process of the model falling naturally from the air between frame40-frame110 by the explicit, semi-explicit and implicit numerical methods.



(a) model displacement

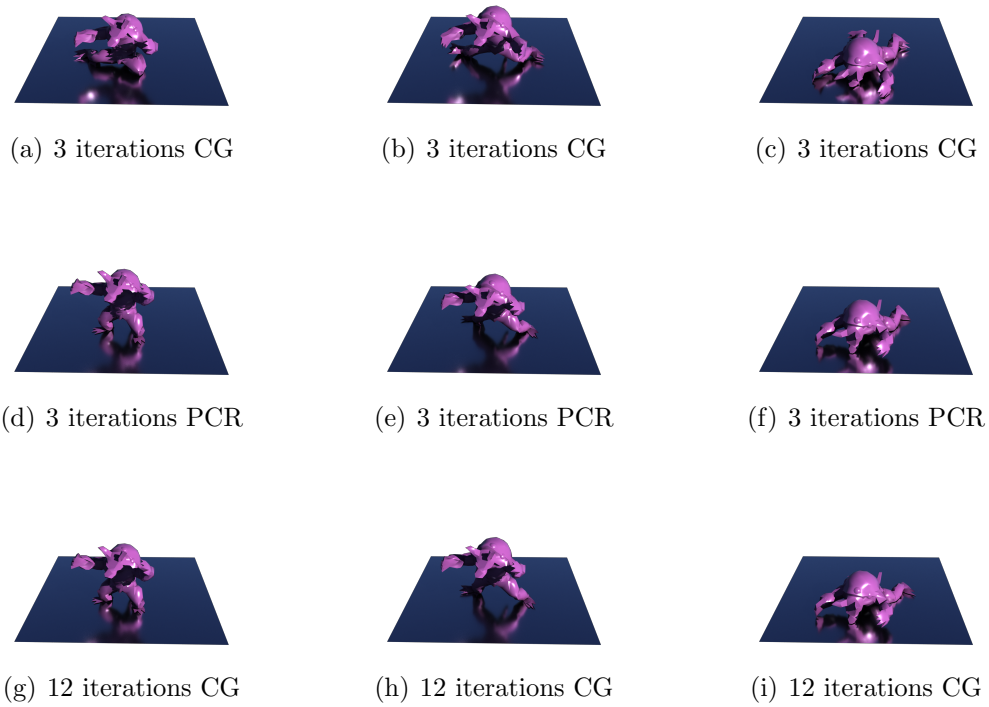
Figur 11: Displacement curve of the model under the same force

It can be seen from the figure that in the initial drop process, the explicit and semi-implicit simulation methods are basically the same as the implicit method. However, as the elastic force is generated due to the collision with the plane, the explicit begins to deviate. As the time step goes on, the deviation effect is getting larger and larger, and the magnitude of the change is getting more and more severe, which implies the instability of the explicit simulation method. The semi-implicit method also has some deviations after a certain time step, but the deformation trend of the two is still consistent, and the error is acceptable. Therefore, when the model is simple, the semi-implicit method can still output relatively stable simulation prediction results. Because the explicit method saves the process of solving the linear equation system, it can reach 120fps in this simulation process. Semi-implicit will slightly reduce performance because it needs to solve a linear equation system, and

can reach 45fps in this simulation experiment. The implicit method needs to solve the equation system multiple times because it requires Newton iteration to perform the nonlinear optimization process, so it will consume a lot of performance, which is about 5fps in this experiment.

5.3.2 Preconditioner numerical method

In the experiment, we tested the effect of the armadillo deformable body model falling from the air and contacting the floor. All the external forces of the object are the same. The model contains a total of 616 vertices, 1770 elements and 2954 edges.



(a) 3 iterations CG (b) 3 iterations CG (c) 3 iterations CG

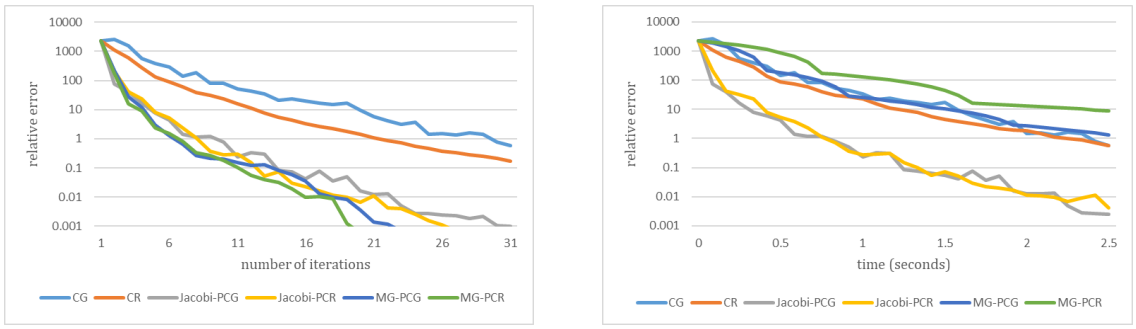
(d) 3 iterations PCR (e) 3 iterations PCR (f) 3 iterations PCR

(g) 12 iterations CG (h) 12 iterations CG (i) 12 iterations CG

Figur 12: 3 iterations CG, 3 iterations PCR, 12 iterations CG, simulation effects at frame 30, frame45, frame60

The above figure shows the deformation simulation effect of the armadillo model applying CG and PCR method with different iterations in frame30, frame40, and frame60 separately. The experiments all used the implicit iterative method, and the applied external force are all the same. It can be found that the CG method of 3 iterations and 12 iterations shows very different deformation effect. The conjugate residual method with preconditioner after 3 iterations, however, the deformation trend can be basically consistent with the result of 12 iterations. which shows that the error of PCR method after 3 iterations is basically the same as that of CG method after 12 iterations. It shows that the PCR method has a faster speed of reducing the error, so as to achieve a better simulation effect.

The figure below shows the relative error curve of the model after several iterations and several times. The figure illustrates that using the conjugate gradient, conjugate residual, Jacobi PCG, Jacobi PCR, Multigrid PCG, Multigrid PCR numerical methods to simulate the deformation process of the model falling from the air naturally.



(a) relative error after several iterations

(b) relative error after several time

Figur 13: Relative error of simulation after several iterations and time for different linear solvers

From the figure, we can see that in the iterative process, because of the non-convexity of the linear system, the conjugate residual is generally better than the conjugate gradient in dealing with the non-positive definite Hessian matrix, which does not consume much performance in comparison. After applying the preconditioner, the speed of reducing the error will be significantly better than not using the preconditioner, and the multigrid preconditioner will be better than Jacobi. However, since the multi-level will increase the computational burden, the performance consumed by each iteration will also be greatly increased. In contrast, the Jacobi preconditioner can quickly reduce the error without consuming too much performance, and can achieve better simulation results at the same time.

5.4 High order FEM

In linear FEM, only 4 nodes need to be considered for a tetrahedron, but for high-order FEM, more nodes need to be considered in a tetrahedron (10 nodes for quadratic FEM and 20 nodes for cubic FEM). So here we need to consider a new mesh layout. Taking the Armadillo model as an example, the model has a total of 616 vertices, 1138 faces, 1770 tetrahedra, and 2954 edges. The existing vertices data in Armadillo Mesh represents the indices of four vertices on 1770 tetrahedra (1770×4). In high-order FEM, we need to reprocess the vertices data (second-order FEM: 1770×10, third-order FEM: 1770×20)

- **Handling shared points:** In quadratic FEM, the added interpolation points on a tetrahedron are on 6 edges, so there are 616+2954 vertices in total. In order

to deal with the shared points problem, we borrow the indices of 2954 edges as the indices of added interpolation points, and add the indices of 6 edges as new vertices after the four vertices. In the cubic FEM, two new interpolation points are added to each edge. Similarly, we can supplement the indices of 6 new points on the 10 vertices on the basis of the quadratic FEM.

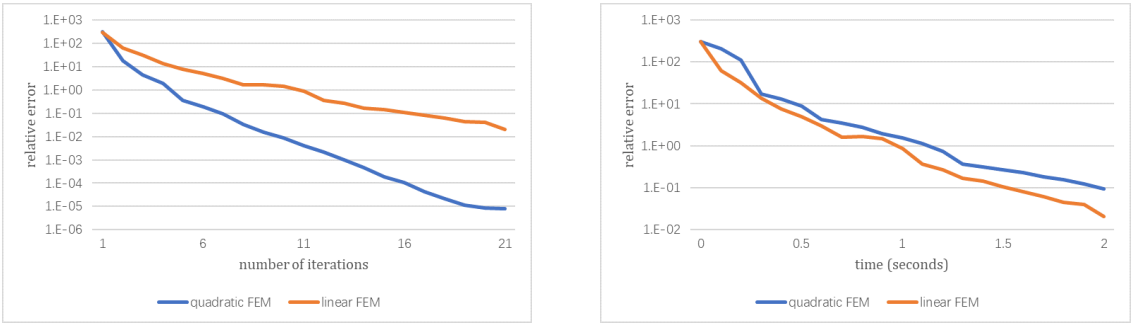
```
for i in range(1770):
newdata[i] = np.append(verticesdata[i], edgesdata[i]+616)
```

- **Handling shared faces:** In the cubic FEM, since there exist interpolation points on the four faces of the tetrahedron, there are a total of 616+2954×2+1138 vertices. In order to deal with the problem of shared faces, we can borrow the indices of 1138 faces as the indices of new interpolation points, and add the number of 4 faces as new vertices on the basis of the 16 vertices on the edges.

```
for i in range(1138):
newdata[i] = np.append(verticesdata[i], facesdata[i]+616+2954*2)
```

For the solution process of force and Hessian matrix of high-order FEM, please refer to Appendix.

The relative error curve of the model after several iterations and several times is shown in the figure below. The figure shows the deformation process of using linear FEM and quadratic FEM to simulate the natural fall of the armadillo model from the air.



(a) relative error after several iterations

(b) relative error after several time

Figur 14: Relative errors of linear and quadratic FEM simulations after several iterations and times

From the figure, we can see that in the iterative process, the quadratic FEM can obtain more accurate simulations in fewer iterations than linear FEM, and its error reduction efficiency is much higher than that of the linear FEM. However, correspondingly, the cost of performance will become very huge. At this time, the size of the Hessian matrix will be expanded from 5310×5310 (armadillo model) to

14172×14172. Therefore, in the process of solving the linear system of equations, each iteration of the Krylov subspace method takes a lot of time, so that if we upgrade all the FE order, it does not achieve much better results than linear FEM in the same time.

5.5 Summary and outlook

5.5.1 Summary of thesis work

This paper mainly focuses on the research and application of fast and stable iterative methods in FEM-based deformation body simulation. First of all, we introduce the current situation and problems of deformation body simulation research, introduce the continuum mechanics, and give several constitutive models of elastic materials. Then we describe several common numerical solution methods for deformable body simulation, with emphasis on semi-implicit and implicit iterative methods, and give a damped Newton algorithm with adaptive time step.

Since people have higher requirements for visual effects and simulation speed, we focus on improving the numerical method and FEM simulation to build a more accurate and faster physical simulator. For the numerical method of simulation, we propose several linear solvers including conjugate gradient method and conjugate residual method and then introduce several preconditioners to accelerate these linear solvers, such as Jacobi preconditioner, Multigrid preconditioner and so on.

We propose a high-order FEM technique to improve the accuracy of the simulation by increasing the order of the interpolation basis function. We give a general algorithm for solving the reference tetrahedron element shape function in isoparametric space by using the Vandermonde matrix, and then give the method to calculate the force and Hessian matrix under high-order FEM. Finally. Since deformation gradient is not a constant matrix in a high-order FEM any more, we give numerical quadrature methods including Gauss quadrature and Hammer quadrature to calculate the force and force gradient.

The simulation experiments use the new Taichi programming language. First of all, we simulated the performance of different material on compression, tension and bending, and found that Neo-Hookean has good expressiveness in various extreme cases because of the rationality of its mathematical expression. Then, we verify different linear solvers that can reduce the error in the iterative process. Due to the complexity of energy in FEM, it will bring non-convexity to the linear system, and the conjugate residual will be better than the conjugate gradient when dealing with non-positive definite matrices. And after adding the preconditioner, the speed of reducing the error in the iterative process will be significantly better than that without the preconditioner. Finally, based on the linear FEM, we increase the order of the

interpolation function, and found that the high-order FEM simulation is better than the linear case, and more accurate simulation results can be obtained with fewer iteration steps.

5.5.2 Outlook for future work

It is hoped that the simulation technology studied in this paper can be further improved from the following aspects:

- In practical calculations, increasing the order of all basis functions used in this paper can improve the simulation accuracy, but at the cost of a sharp increase in solution time. To overcome this problem, we also need to explore a more efficient adaptive FEM. The adaptive finite element method is a numerical method that can automatically refine the mesh or upgrade the element basis function according to the behavior of the solution, and obtain the maximum calculation accuracy with the minimum calculation amount. It can effectively overcome the difficulty of solving the rapidly increasing degrees of freedom caused by uniform element upgrade. Among them, the p-type adaptive algorithm can upgrade the low-quality grid cells with severe field changes or large grid sizes while keeping the initial grid division unchanged. It has the advantages of less workload and high calculation accuracy. In addition, the basis function of the solution area has a selective upgrade adjustment, which can not only obtain a good calculation accuracy, but also reduce the calculation memory. Therefore, how to construct a fast and effective p-adaptive FEM is a direction worthy of further study.

- FEM usually needs to consider non-convex optimization problems, and this paper only uses appropriate numerical methods to reduce the influence of the non-positive definiteness of the Hessian matrix as much as possible. However, for the optimization problem in the deformation body simulation, due to the appearance of the energy E, the FEM usually brings about a large non-convex problem, so additional auxiliary variables need to be added. The Projective Dynamic method adopted by S. Bouaziz and TT. Liu, which performs Local-Global processing on the energy system by introducing auxiliary variables, can be used to solve the implicit time integration using FEM when the elastic potential energy of the material model has a specific quadratic form, and introducesquasi-Newton method to replace the Hessian matrix with a simplified matrix, which can bring a variety of existing numerical methods to accelerate the solution process (such as L-BFGS), making the simulation more efficient. Therefore, how to effectively integrate the high-order FEM technology and the projective dynamic method in this paper also needs further research.

- The recent surge of differentiable physics witnessed the emergence of differentiable simulators as well as their success in various inverse problems that have

simulation inside an optimization loop. Thanks to the development of deep learning and its gradient-based optimization method, increasingly more research in graphics began to combine deep learning acceleration strategies. This method usually transfers the time-consuming complex geometry processing, equation solving, etc. to offline network training to speed up the performance of the simulator. With additional knowledge of gradients, a differentiable simulator provides more guidance on the evolution of a physics system, which has shown great beneficial in the use of computer-aided design of soft robots. In addition, combining with neural networks to construct composite material models with more complex nonlinear de- formation behaviors in a data-driven manner can also help us better reconstruct real deformable materials. Therefore, how to construct a suitable differentiable simulator for high-order FEM is also a very interesting further research direction.

Litteratur

[1] Ivo Babuška. The p and hp versions of the finite element method: the state of the art. *Finite elements*, pages 199–239, 1988.

[2] Ivo Babuska, Barna A Szabo, and I Norman Katz. The p-version of the finite element method. *SIAM journal on numerical analysis*, 18(3):515–545, 1981.

[3] David Baraff and Andrew Witkin. Large steps in cloth simulation. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 43–54, 1998.

[4] Adam W Bargteil and Elaine Cohen. Animation of deformable bodies with quadratic bézier finite elements. *ACM Transactions on Graphics (TOG)*, 33(3):1–10, 2014.

[5] Michele Benzi. Preconditioning techniques for large linear systems: a survey. *Journal of computational Physics*, 182(2):418–477, 2002.

[6] Sofien Bouaziz, Sebastian Martin, Tiantian Liu, Ladislav Kavan, and Mark Pauly. Projective dynamics: Fusing constraint projections for fast simulation. *ACM transactions on graphics (TOG)*, 33(4):1–11, 2014.

[7] Philippe G Ciarlet and Pierre-Arnaud Raviart. General lagrange and hermite interpolation in rn with applications to finite element methods. *Archive for Rational Mechanics and Analysis*, 46(3):177–199, 1972.

[8] Tao Du, Kui Wu, Pingchuan Ma, Sebastien Wah, Andrew Spielberg, Daniela Rus, and Wojciech Matusik. Diffpd: Differentiable projective dynamics with contact. *arXiv e-prints*, pages arXiv–2101, 2021.

[9] DA794241 Dunavant. High degree efficient symmetrical gaussian quadrature rules for the triangle. *International journal for numerical methods in engineering*, 21(6):1129–1148, 1985.

[10] Essex Edwards and Robert Bridson. Detailed water with coarse grids: Combining surface meshes and adaptive discontinuous galerkin. *ACM Transactions on Graphics (TOG)*, 33(4):1–9, 2014.

[11] Kenny Erleben and Sheldon Andrews. Inverse kinematics problems with exact hessian matrices. In *Proceedings of the Tenth International Conference on Motion in Games*, pages 1–6, 2017.

[12] Kenny Erleben, Jon Sporring, Knud Henriksen, and Henrik Dohlmann. *Physics-based animation*. Charles River Media Hingham, 2005.

[13] Alec Jacobson, Ilya Baran, Jovan Popovic, and Olga Sorkine. Bounded biharmonic weights for real-time deformation. *ACM Trans. Graph.*, 30(4):78, 2011.

[14] Peter Kaufmann, Oliver Wang, Alexander Sorkine-Hornung, Olga Sorkine-Hornung, Aljoscha Smolic, and Markus Gross. Finite element image warping. In *Computer Graphics Forum*, volume 32, pages 31–39. Wiley Online Library, 2013.

[15] Michael Kazhdan and Hugues Hoppe. Screened poisson surface reconstruction. *ACM Transactions on Graphics (ToG)*, 32(3):1–13, 2013.

[16] Theodore Kim and David Eberle. Dynamic deformables: implementation and production practicalities. In *ACM SIGGRAPH 2020 Courses*, pages 1–182. 2020.

[17] Yifei Li, Tao Du, Kui Wu, Jie Xu, and Wojciech Matusik. Diffcloth: Differentiable cloth simulation with dry frictional contact. *arXiv preprint arXiv:2106.05306*, 2021.

[18] Tiantian Liu, Adam W Bargteil, James F O'Brien, and Ladislav Kavan. Fast simulation of mass-spring systems. *ACM Transactions on Graphics (TOG)*, 32(6):1–7, 2013.

[19] Tiantian Liu, Sofien Bouaziz, and Ladislav Kavan. Quasi-newton methods for real-time simulation of hyperelastic materials. *Acm Transactions on Graphics (TOG)*, 36(3):1–16, 2017.

[20] Miles Macklin, Kenny Erleben, Matthias Müller, Nuttapong Chentanez, Stefan Jeschke, and Viktor Makoviychuk. Non-smooth newton methods for deformable multi-body dynamics. *ACM Transactions on Graphics (TOG)*, 38(5):1–20, 2019.

[21] Maciej Paszyński, Rafał Grzeszczuk, David Pardo, and Leszek Demkowicz. Deep learning driven self-adaptive hp finite element method. In *International Conference on Computational Science*, pages 114–121. Springer, 2021.

[22] Yousef Saad. *Iterative methods for sparse linear systems*. SIAM, 2003.

[23] Teseo Schneider, Yixin Hu, Jérémie Dumas, Xifeng Gao, Daniele Panozzo, and Denis Zorin. Decoupling simulation accuracy from mesh quality. *ACM transactions on graphics*, 2018.

[24] Eftychios Sifakis and Jernej Barbic. Fem simulation of 3d deformable solids: a practitioner's guide to theory, discretization and model reduction. In *Acm siggraph 2012 courses*, pages 1–50. 2012.

[25] Valeria Simoncini and Daniel B Szyld. Recent computational developments in krylov subspace methods for linear systems. *Numerical Linear Algebra with Applications*, 14(1):1–59, 2007.

[26] Tomohiro Sogabe. Extensions of the conjugate residual method. *Department of Applied Physics, University of Tokyo, Tokyo, Japan*, 2006.

[27] Demetri Terzopoulos, John Platt, Alan Barr, and Kurt Fleischer. Elastically deformable models. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 205–214, 1987.

[28] Henk A Van der Vorst. *Iterative Krylov methods for large linear systems*. Number 13. Cambridge University Press, 2003.

[29] Danyong Zhao, Yijing Li, and Jernej Barbič. Asynchronous implicit backward euler integration. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 1–9, 2016.

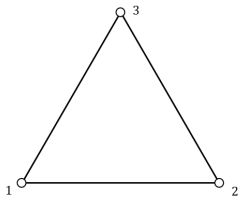
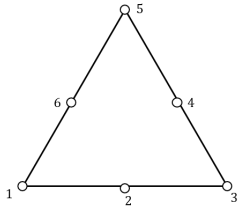
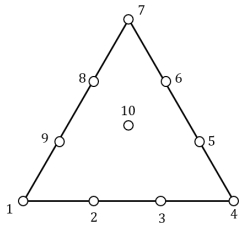
[30] Qingnan Zhou, Julian Panetta, and Denis Zorin. Worst-case structural analysis. *ACM Trans. Graph.*, 32(4):137–1, 2013.

[31] Bo Zhu, Mélina Skouras, Desai Chen, and Wojciech Matusik. Two-scale topology optimization with microstructures. *ACM Transactions on Graphics (TOG)*, 36(4):1, 2017.

[32] Olgierd C Zienkiewicz, JP De SR Gago, and Don W Kelly. The hierarchical concept in finite element analysis. *Computers & Structures*, 16(1-4):53–65, 1983.

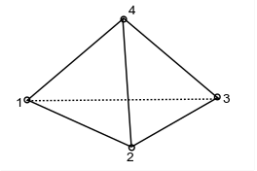
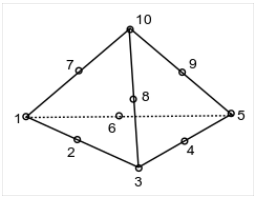
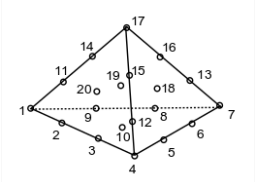
6 Appendix

6.1 High order FEM basis function

Element shape	Isoparametric Shape function
	$N_1 = L_1,\ N_2 = L_2,\ N_3 = L_3,$
	$N_1 = 2\left(L_1 - 1\right)L_1,\ N_2 = 4L_1L_2,$ $N_3 = 2\left(L_2 - 1\right)L_2,\ N_4 = 4L_2L_3,$ $N_5 = 2\left(L_3 - 1\right)L_3,\ N_6 = 4L_3L_1.$
	$N_1 = \frac{1}{2}\left(3L_1 - 1\right)\left(3L_1 - 2\right)L_1,\ N_2 = \frac{9}{2}L_1L_2\left(3L_1 - 1\right),$ $N_3 = \frac{9}{2}L_1L_2\left(3L_2 - 1\right),\ N_4 = \frac{1}{2}\left(3L_2 - 1\right)\left(3L_2 - 2\right)L_2,$ $N_5 = \frac{9}{2}L_2L_3\left(3L_2 - 1\right),\ N_6 = \frac{9}{2}L_3L_2\left(3L_3 - 1\right),$ $N_7 = \frac{1}{2}\left(3L_3 - 1\right)\left(3L_3 - 2\right)L_3,\ N_8 = \frac{9}{2}L_3L_1\left(3L_3 - 1\right),$ $N_9 = \frac{9}{2}L_3L_1\left(3L_1 - 1\right),\ N_{10} = 27L_1L_2L_3.$

Tabel 1: 2D plane FEM shape function

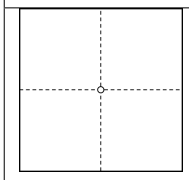
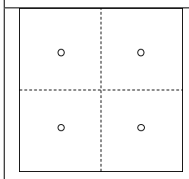
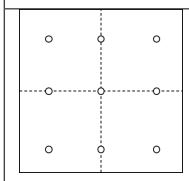
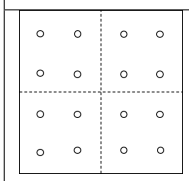
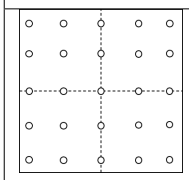
Note: Among them, $L_1,\ L_2,\ L_3$ are the area coordinates in the shape function of the triangular element, which are equivalent to $L_1 = \xi,\ L_2 = \eta,\ L_3 = 1 - \xi - \eta$ in the isoparametric element.

Element shape	Isoparametric Shape function
	$N_1 = L_1,\ N_2 = L_2,\ N_3 = L_3,\ N_4 = L_4.$
	$N_1 = L_1\left(2L_1 - 1\right),\ N_2 = 4L_1L_2,$ $N_3 = L_2\left(2L_2 - 1\right),\ N_4 = 4L_2L_3,$ $N_5 = L_3\left(2L_3 - 1\right),\ N_6 = 4L_3L_1,$ $N_7 = 4L_1L_4,\ N_8 = 4L_2L_4,$ $N_9 = 4L_3L_4,\ N_{10} = L_4\left(2L_4 - 1\right).$
	$N_1 = \frac{1}{2}\left(3L_1 - 1\right)\left(3L_1 - 2\right)L_1,\ N_2 = \frac{9}{2}L_1L_2\left(3L_1 - 1\right),$ $N_3 = \frac{9}{2}L_1L_2\left(3L_2 - 1\right),\ N_4 = \frac{1}{2}\left(3L_2 - 1\right)\left(3L_2 - 2\right)L_2,$ $N_5 = \frac{9}{2}L_2L_3\left(3L_2 - 1\right),\ N_6 = \frac{9}{2}L_3L_2\left(3L_3 - 1\right),$ $N_7 = \frac{1}{2}\left(3L_3 - 1\right)\left(3L_3 - 2\right)L_3,\ N_8 = \frac{9}{2}L_3L_1\left(3L_3 - 1\right),$ $N_9 = \frac{9}{2}L_3L_1\left(3L_1 - 1\right),\ N_{10} = 27L_1L_2L_3,$ $N_{11} = \frac{9}{2}L_1L_4\left(3L_1 - 1\right),\ N_{12} = \frac{9}{2}L_2L_4\left(3L_2 - 1\right),$ $N_{13} = \frac{9}{2}L_3L_4\left(3L_3 - 1\right),\ N_{14} = \frac{9}{2}L_1L_4\left(3L_4 - 1\right),$ $N_{15} = \frac{9}{2}L_2L_4\left(3L_4 - 1\right),\ N_{16} = \frac{9}{2}L_3L_4\left(3L_4 - 1\right),$ $N_{17} = \frac{1}{2}\left(3L_4 - 1\right)\left(3L_4 - 2\right)L_4,\ N_{18} = 27L_2L_3L_4,$ $N_{19} = 27L_1L_3L_4,\ N_{20} = 27L_1L_2L_4.$

Tabel 2: 3D space FEM shape function

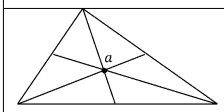
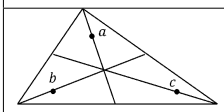
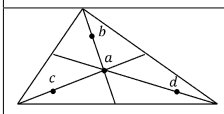
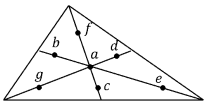
Note: Among them, $L_1,\ L_2,\ L_3,\ L_4$ are volume coordinates in the shape function of the tetrahedron element, which are equivalent to $L_1 = \xi,\ L_2 = \eta,\ L_3 = \mu,\ L_4 = 1 - \xi - \eta - \mu$ in the isoparametric element.

6.2 Numerical quadrature coordinate and weight

Position	Error	Quadrature Points Coordinate	Weight
	$O\left(h^1\right)$	0.0000000000	2.0000000000
	$O\left(h^3\right)$	$\pm$0.5773502692	1.0000000000
	$O\left(h^5\right)$	0.0000000000, $\pm$0.7745966692	0.8888888889, 0.5555555555
	$O\left(h^7\right)$	$\pm$0.3399810435, $\pm$0.8611363116	0.6521451548, 0.3478548451
	$O\left(h^9\right)$	0.0000000000, $\pm$0.5384693101, $\pm$0.9061798459	0.5688888889, 0.4786286705, 0.2369268850

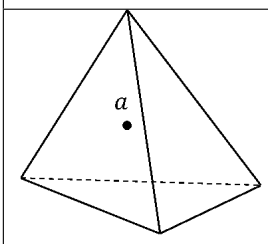
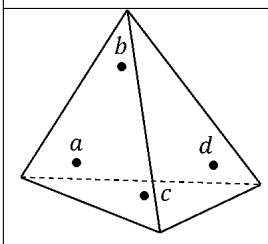
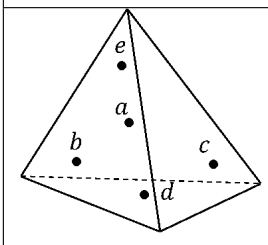
Tabel 3: Gauss quadrature points and weight

Note: The quadrature points and weights here only provide the case of one dimension of $[-1, 1]$. When the Gaussian quadrature applied for higher-dimensional situations, we need to reasonably combine the $order^{dim}$ quadrature points, and calculate the weight according to the combination of the quadrature points.

Position	Error	Quadrature Points Coordinate	Weight
	$O\left(h^2\right)$	$a=\left(\frac{1}{3},\frac{1}{3},\frac{1}{3}\right)$	1
	$O\left(h^3\right)$	$a=\left(\frac{2}{3},\frac{1}{6},\frac{1}{6}\right), b=\left(\frac{1}{6},\frac{2}{3},\frac{1}{6}\right), c=\left(\frac{1}{6},\frac{1}{6},\frac{2}{3}\right)$	$\frac{1}{3},\frac{1}{3},\frac{1}{3}$
	$O\left(h^4\right)$	$a=\left(\frac{1}{3},\frac{1}{3},\frac{1}{3}\right),$ $b=\left(\frac{3}{5},\frac{1}{5},\frac{1}{5}\right), c=\left(\frac{1}{5},\frac{3}{5},\frac{1}{5}\right), d=\left(\frac{1}{5},\frac{1}{5},\frac{3}{5}\right)$	$-\frac{27}{48},$ $\frac{25}{48}$
	$O\left(h^6\right)$	$a=\left(\frac{1}{3},\frac{1}{3},\frac{1}{3}\right)$ $b=(\alpha_1,\beta_1,\beta_1),\ c=(\beta_1,\alpha_1,\beta_1), d=(\beta_1,\beta_1,\alpha_1),$ $e=(\alpha_2,\beta_2,\beta_2), f=(\beta_2,\alpha_2,\beta_2),\ g=(\beta_2,\beta_2,\alpha_2)$ $\alpha_1=0.0597158717,\ \beta_1=0.4701420641$ $\alpha_1=0.7974269853,\ \beta_2=0.1012865073$	0.2250000000, 0.1323941527, 0.1259391805

Tabel 4: 2D Hammer quadrature points and weight

Note: Since the triangular integral domain involves the variable itself, the sum of the weight coefficients should be equal to 1/2, so the weight coefficients listed in the table should be multiplied by 1/2.

Position	Error	Quadrature Points Coordinate	Weight
	$O\left(h^2\right)$	$a=\left(\frac{1}{3},\frac{1}{3},\frac{1}{3}\right)$	1
	$O\left(h^3\right)$	$a=\left(\frac{2}{3},\frac{1}{6},\frac{1}{6}\right), b=\left(\frac{1}{6},\frac{2}{3},\frac{1}{6}\right), c=\left(\frac{1}{6},\frac{1}{6},\frac{2}{3}\right)$	$\frac{1}{3},\frac{1}{3},\frac{1}{3}$
	$O\left(h^4\right)$	$a=\left(\frac{1}{3},\frac{1}{3},\frac{1}{3}\right),$ $b=\left(\frac{3}{5},\frac{1}{5},\frac{1}{5}\right), c=\left(\frac{1}{5},\frac{3}{5},\frac{1}{5}\right), d=\left(\frac{1}{5},\frac{1}{5},\frac{3}{5}\right)$	$-\frac{27}{48},$ $\frac{25}{48}$

Tabel 5: 3D Hammer quadrature points and weight

Note: Since the tetrahedron integral domain involves the variable itself, the sum of the weight coefficients should be equal to 1/6, so the weight coefficients listed in the table should be multiplied by 1/6.

6.3 Proof of Hammer theorem 1

When $\alpha+\beta+\gamma=0$, the corresponding conditional equation is

$$\int_0^1\int_0^{1-L_1} L_1^{\alpha}L_2^{\beta}L_3^{\gamma}dL_2dL_1=\int_0^1\int_0^{1-L_1} dL_2dL_1=\frac{1}{2}=w_1+\sum_{p=1}^{n}w_{2p}. \tag{6.1}$$

When $\alpha+\beta+\gamma=1$, since α,β,γ are symmetric, exchange α,β,γ has no effect on the numerical integration result. Therefore, the numerical integration formula is accurate for $f(L_1,L_2,L_3)=L_1/L_2/L_3$. Thence we have

$$\begin{aligned}\frac{1}{6}\int_0^1\int_0^{1-L_1} L_1dL_2dL_1&=\frac{1}{6}\int_0^1\int_0^{1-L_1} L_2dL_2dL_1\\&=\frac{1}{6}\int_0^1\int_0^{1-L_1} L_3dL_2dL_1=w_1\left(\frac{1}{3}\right)+\sum_{p}^{n}w_{2p},\end{aligned} \tag{6.2}$$

in addition, the area coordinate has the property of $L_1+L_2+L_3=1$, so

$$\int_0^1\int_0^{1-L_1}(L_1+L_2+L_3)\,dL_2dL_1=\frac{1}{2}=3\left(w_1+\sum_{p=1}^{n}w_{2p}\right). \tag{6.3}$$

Obviously, the two cases of $\alpha+\beta+\gamma=0/1$ can only provide one independent equation.

When $\alpha+\beta+\gamma=2$, since $L_1L_2=L_1(1-L_1-L_3)=L_1-L_1^2-L_1L_3$, we can get

$$\int_0^1\int_0^{1-L_1} L_1L_2dL_2dL_1+\int_0^1\int_0^{1-L_1} L_1L_3dL_2dL_1=\int_0^1\int_0^{1-L_1} L_1-L_1^2dL_2dL_1, \tag{6.4}$$

and because L_1,L_2,L_3 have symmetry, it can be seen that

$$\int_0^1\int_0^{1-L_1} L_1L_3dL_2dL_1=\frac{1}{2}\left(\int_0^1\int_0^{1-L_1} L_1dL_2dL_1-\int_0^1\int_0^{1-L_1} L_1^2dL_2dL_1\right). \tag{6.5}$$

Therefore, if the numerical integration formula can be accurately established for the integrand functions of L_1 and L_1^2, then the integrand functions of $L_2^2,\ L_3^2,\ L_1L_2,\ L_1L_3,\ L_2L_3$

can all be accurately established, that is the multiple combinations of $\alpha+\beta+\gamma=2$ can only provide one independent equation. Similarly, for the case of $\alpha+\beta+\gamma=3$, considering the symmetry, the corresponding basic combination forms are only $L_1^3,\ L_1^2L_2,\ L_1L_2L_3$. Because $L_1^2L_2=L_1^2\left(1-L_1-L_3\right)=L_1^2-L_1^3-L_1^2L_3$, it is easy to get

$$\int_0^1\int_0^{1-L_1}L_1^2L_2dL_2dL_1=\frac{1}{2}\int_0^1\int_0^{1-L_1}L_1^2-L_1^3dL_2dL_1, \tag{6.6}$$

That is, the equations provided when the integrand are L_1^3 and $L_1^2L_2$ respectively are not independent. And because $L_1L_2L_3=L_1L_2\left(1-L_1-L_2\right)=L_1L_2-L_1^2L_2-L_1L_2^2$, we can get

$$\int_0^1\int_0^{1-L_1}L_1L_2L_3dL_2dL_1=\int_0^1\int_0^{1-L_1}L_1L_2dL_2dL_1-2\int_0^1\int_0^{1-L_1}L_1^2L_2dL_2dL_1 \tag{6.7}$$

That is, the equations provided when the integrand functions are $L_1^2L_2$ and $L_1L_2L_3$ respectively are not independent either. Therefore, $\alpha+\beta+\gamma=3$ can only provide an independent equation. By analogy, it can be proved that the various combinations of $\alpha+\beta+\gamma=m$ can only provide an independent equation.

6.4 Implementing details of high order FEM

For quadratic FEM, on the reference tetrahedron element, the coordinates of the 10 nodes are

$$x_0=(0,0,0)\,,\ x_1=(1,0,0)\,,\ x_2=(0,1,0)\,,\ x_3=(0,0,1)$$
$$x_4=\left(\frac{1}{2},0,0\right)=\frac{1}{2}\left(x_0+x_1\right),\ x_5=\left(\frac{1}{2},\frac{1}{2},0\right)=\frac{1}{2}\left(x_1+x_2\right),\ x_6=\left(0,\frac{1}{2},0\right)=\frac{1}{2}\left(x_0+x_2\right)$$
$$x_7=\left(\frac{1}{2},0,\frac{1}{2}\right)=\frac{1}{2}\left(x_1+x_3\right),\ x_8=\left(0,\frac{1}{2},\frac{1}{2}\right)=\frac{1}{2}\left(x_2+x_3\right),\ x_9=\left(0,0,\frac{1}{2}\right)=\frac{1}{2}\left(x_0+x_3\right)$$

At this point, we can differentiate the local reference units separately as

$$\partial\begin{pmatrix}u\left(2u-1\right)\\v\left(2v-1\right)\\w\left(2w-1\right)\\\left(u+v+w-1\right)\left(2u+2v+2w-1\right)\\4uv\\4vw\\4uw\\-4u\left(u+v+w-1\right)\\-4v\left(u+v+w-1\right)\\-4w\left(u+v+w-1\right)\end{pmatrix}=\begin{pmatrix}4u-1&0&0\\0&4v-1&0\\0&0&4w-1\\4u+4v+4w-3&4u+4v+4w-3&4u+4v+4w-3\\4v&4u&0\\0&4w&4v\\4w&0&4u\\-8u-4v-4w+4&-4u&-4u\\-4v&-8v-4u-4w+4&-4v\\-4w&-4w&-8w-4u-4v+4\end{pmatrix} \tag{6.8}$$

Therefore, we can obtain the deformation gradient $F = D_s D_m^{-1}$, where $D_s = [M_1 \,|M_2|\, M_3]$, and M_1, M_2, M_3 can be represented as

$$M_1 = (4u-1)x_0 + (4u+4v+4w-3)x_3 + 4vx_4 + 4wx_6 + (-8u-4v-4w+4)x_7 + (-4v)x_8 + (-4w)x_9$$
$$M_2 = (4v-1)x_1 + (4u+4v+4w-3)x_3 + 4ux_4 + 4wx_5 + (-4u)x_7 + (-8v-4u-4w+4)x_8 + (-4w)x_9$$
$$M_3 = (4w-1)x_2 + (4u+4v+4w-3)x_3 + 4vx_5 + 4ux_6 + (-4u)x_7 + (-4v)x_8 + (-8w-4u-4v+4)x_9$$

Take $x_4, \ldots, x_9$ above, replace F as the matrix constitute of x_1, x_2, x_3

$$M_1 = (4u+2v-1)x_0 + (-4u-2w+2)x_1 + (2w-2v)x_2 - x_3$$
$$M_2 = (2u-2w)x_0 + (4v+2w-1)x_1 + (-4u-2v+2)x_2 + (-2u+2v-1)x_3$$
$$M_3 = (-2u-2v-2w+2)x_0 + (2v-2u)x_1 + (2u+4w-1)x_2 + (-2u+2w-1)x_3$$

Similar to the derivative of deformation gradient $\frac{\partial F}{\partial x} = \frac{\partial D_s}{\partial x} D_m^{-1}$, we can calculate the partial derivatives of D_s as

$$\frac{\partial D_s}{\partial x_0^x} = \begin{pmatrix} 4u-1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \quad \frac{\partial D_s}{\partial x_0^y} = \begin{pmatrix} 0 & 0 & 0 \\ 4u-1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \quad \frac{\partial D_s}{\partial x_0^z} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 4u-1 & 0 & 0 \end{pmatrix}$$
$$\frac{\partial D_s}{\partial x_1^x} = \begin{pmatrix} 0 & 4v-1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \quad \frac{\partial D_s}{\partial x_1^y} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 4v-1 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \quad \frac{\partial D_s}{\partial x_1^z} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 4v-1 & 0 \end{pmatrix}$$
$$\frac{\partial D_s}{\partial x_2^x} = \begin{pmatrix} 0 & 0 & 4w-1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \quad \frac{\partial D_s}{\partial x_2^y} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 4w-1 \\ 0 & 0 & 0 \end{pmatrix}, \quad \frac{\partial D_s}{\partial x_2^z} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 4w-1 \end{pmatrix}$$

$$\frac{\partial D_s}{\partial x_3^x} = \begin{pmatrix} 4u+4v+4w-3 & 4u+4v+4w-3 & 4u+4v+4w-3 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix},$$
$$\frac{\partial D_s}{\partial x_3^y} = \begin{pmatrix} 0 & 0 & 0 \\ 4u+4v+4w-3 & 4u+4v+4w-3 & 4u+4v+4w-3 \\ 0 & 0 & 0 \end{pmatrix},$$
$$\frac{\partial D_s}{\partial x_3^z} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 4u+4v+4w-3 & 4u+4v+4w-3 & 4u+4v+4w-3 \end{pmatrix}$$

$$\frac{\partial D_s}{\partial x_4^x} = \begin{pmatrix} 4v & 4u & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \frac{\partial D_s}{\partial x_4^y} = \begin{pmatrix} 0 & 0 & 0 \\ 4v & 4u & 0 \\ 0 & 0 & 0 \end{pmatrix}, \frac{\partial D_s}{\partial x_4^z} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 4v & 4u & 0 \end{pmatrix}$$
$$\frac{\partial D_s}{\partial x_5^x} = \begin{pmatrix} 0 & 4w & 4v \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \frac{\partial D_s}{\partial x_5^y} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 4w & 4v \\ 0 & 0 & 0 \end{pmatrix}, \frac{\partial D_s}{\partial x_5^z} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 4w & 4v \end{pmatrix}$$
$$\frac{\partial D_s}{\partial x_6^x} = \begin{pmatrix} 4w & 0 & 4u \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \frac{\partial D_s}{\partial x_6^y} = \begin{pmatrix} 0 & 0 & 0 \\ 4w & 0 & 4u \\ 0 & 0 & 0 \end{pmatrix}, \frac{\partial D_s}{\partial x_6^z} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 4w & 0 & 4u \end{pmatrix}$$

$$\frac{\partial D_s}{\partial x_7^x} = \begin{pmatrix} -8u-4v-4w+4 & -4u & -4u \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \frac{\partial D_s}{\partial x_7^y} = \begin{pmatrix} 0 & 0 & 0 \\ -8u-4v-4w+4 & -4u & -4u \\ 0 & 0 & 0 \end{pmatrix},$$
$$\frac{\partial D_s}{\partial x_7^z} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ -8u-4v-4w+4 & -4u & -4u \end{pmatrix}$$
$$\frac{\partial D_s}{\partial x_8^x} = \begin{pmatrix} -4v & -8v-4u-4w+4 & -4v \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \frac{\partial D_s}{\partial x_8^y} = \begin{pmatrix} 0 & 0 & 0 \\ -4v & -8v-4u-4w+4 & -4v \\ 0 & 0 & 0 \end{pmatrix},$$
$$\frac{\partial F}{\partial x_8^z} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ -4v & -8v-4u-4w+4 & -4v \end{pmatrix}$$
$$\frac{\partial D_s}{\partial x_9^x} = \begin{pmatrix} -4w & -4w & -8w-4u-4v+4 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \frac{\partial D_s}{\partial x_9^y} = \begin{pmatrix} 0 & 0 & 0 \\ -4w & -4w & -8w-4u-4v+4 \\ 0 & 0 & 0 \end{pmatrix},$$
$$\frac{\partial D_s}{\partial x_9^z} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ -4w & -4w & -8w-4u-4v+4 \end{pmatrix}$$

At this point, we notice that $\frac{\partial F}{\partial x}$ is a $10 \times 3 \times (3 \times 3)$ tensor, and the Piola-Kirchhoff tensor is still a 3×3 matrix, so tensor contraction is stilled required here. Also, because both F and $\frac{\partial F}{\partial x}$ are no longer a constant matrix, this means that the deformation within an element is no longer linear, and each point has a different amount of deformation. Therefore, we need to do numerically integral for this tetrahedron. Integral on tetrahedron can be represented by Gauss quadrature as

$$\int_0^1 \int_0^{1-x} \int_0^{1-x-y} f_i(x,y,z)\, dzdydx \approx \frac{1}{8} \sum_{(u,v,w)} w_i f_i(u,v,\ w),\ for \begin{cases} u = \frac{1}{2}x + \frac{1}{2} \\ v = \frac{1}{2}y + \frac{1}{2} \\ w = \ \frac{1}{2}z + \frac{1}{2} \end{cases} and\ u+v+w \leq 1 \tag{6.9}$$

At this point we can calculate the elastic force on a tetrahedron as

$$f_{3\times 10} = \left[\frac{1}{8}\sum_{(u,v,w)} w_i f_0, \ldots, \frac{1}{8}\sum_{(u,v,w)} w_i f_9\right] \tag{6.10}$$

where W_i tetrahedron volume, w_i Gauss quadrature weight. Finally, we can assemble the force matrix as

$$\begin{aligned} H = [f_0, f_1, \ldots, f_9] &= -W_i\left[\frac{\partial \Psi}{\partial x_0}, \frac{\partial \Psi}{\partial x_1}, \ldots, \frac{\partial \Psi}{\partial x_9}\right] = -W_i\left[\frac{\partial D_s}{\partial x_0}D_m^{-1} : P, \ldots, \frac{\partial D_s}{\partial x_9}D_m^{-1} : P\right] \\ &= -W_i\left[tr\left(\left(\frac{\partial D_s}{\partial x_0}\right)^T PD_m^{-T}\right), \ldots, tr\left(\left(\frac{\partial D_s}{\partial x_9}\right)^T PD_m^{-T}\right)\right], \\ \left(\frac{\partial D_s}{\partial x_i}\right)^T &= \begin{pmatrix} \left(\frac{\partial D_s}{\partial x_i^x}\right)^T \\ \left(\frac{\partial D_s}{\partial x_i^y}\right)^T \\ \left(\frac{\partial D_s}{\partial x_i^z}\right)^T \end{pmatrix}, \; f_i = \begin{pmatrix} tr\left(\left(\frac{\partial D_s}{\partial x_i^x}\right)^T PD_m^{-T}\right) \\ tr\left(\left(\frac{\partial D_s}{\partial x_i^y}\right)^T PD_m^{-T}\right) \\ tr\left(\left(\frac{\partial D_s}{\partial x_i^z}\right)^T PD_m^{-T}\right) \end{pmatrix}. \end{aligned}$$

For implicit FEM, we also need to solve for the Hessian matirx dH. In linear FEM, the element Hessian is a 12×12 matrix. However, for 2-order FEM, it becomes a 30×30 matrix.

Usually, the second derivative of the force gradient will follow $\frac{\partial^2\Psi(F)}{\partial x^2} = \frac{\partial F_i}{\partial x} : \frac{\partial P}{\partial F_i} : \frac{\partial F_i^T}{\partial x} = vec\left(\frac{\partial F_i}{\partial x}\right)^T vec\left(\frac{\partial P}{\partial F_i}\right) vec\left(\frac{\partial F_i}{\partial x}\right)$ to calculate. We consider a simple Piola-Kirchhoff stress (eg: corotated linear model) that $P = 2\mu\left(F - UV^T\right)$, then

$$vec\left(\frac{\partial P}{\partial F_{ij}}\right) = vec\left(2\mu\frac{\partial F}{\partial F_{ij}}\right) = 2\mu I_{9\times 9}. \tag{6.11}$$

Next we consider the tensor contraction of $vec\left(\frac{\partial F_i}{\partial x}\right)^T vec\left(\frac{\partial F_i}{\partial x}\right)$. Because $\frac{\partial F_i}{\partial x}$ of the second-order FEM is a tensor of $(10\times 3)\times(3\times 3)$, define each 3×3 matrix as dF_i, where $i = 0, \ldots, 29$. For any 3×1 vector dH_i^j $(j = 0, \ldots, 9)$, we can find that

$$dH_i^j = -W_i\Big[tr\left(dF_i(dF_{3j})^T\right), tr\left(dF_i(dF_{3j+1})^T\right), tr\left(dF_i(dF_{3j+2})^T\right)\Big]^T, \tag{6.12}$$

dH_i^j can be assembled into dH_i as a 30×1 vector that

$$dH_i = \left[dH_i^0, dH_i^1, \ldots, dH_i^9\right]^T. \tag{6.13}$$

Similar to calculating node force, numerical integration is also required for Hessian matrix.