RWTH AACHEN UNIVERSITY

The present work was submitted to the

Visual Computing Institute
Faculty of Mathematics, Computer Science and Natural Sciences
RWTH Aachen University

Fast Global Hessian Assembly in Finite Element Simulations

Master Thesis

presented by

Hantao Hui
Student ID Number 408448

29th September, 2022

First Examiner:	Prof. Dr. Jan Bender
Second Examiner:	Prof. Dr. Torsten Kuhlen

Hiermit versichere ich, diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt, sowie Zitate kenntlich gemacht zu haben.

I hereby affirm that I composed this work independently and used no other than the specified sources and tools and that I marked all quotes as such.

Aachen, 29th September, 2022

Hantao Hui

Contents

Abstract

0.1 Abstract

When doing soft body simulations using finite element methods, the process to assemble the Hessian matrix can be computationally expensive. In this thesis, we propose a new method to cut down the computational cost in the assembly process. We notice that not all the local Hessian matrices need to be assembled. During the simulation, we identify the active vertices and elements using the gradient and neighborhood information. Then we compute the local Hessian matrices of active elements only and reuse the local Hessian matrices of inactive elements from previous iterations.

Then we test how different factors can affect our method. First, we test how different simulation parameters, namely stiffness properties and size of time steps, affect our method. The main conclusion is that, when using soft materials or small time steps, our method can perform better. Then we try to find out the relations between our method and the properties of the mesh object itself, namely uniformity and resolution. We conclude that uniform mesh in general performs better than non-uniform mesh. But we are not able to tell the relation between our method and the resolution of the mesh object. In the last, we adapt our method to fit some limitations. One of them is to skip the assembly process in the first Newton iteration, another one is to use our method with projected Newton's method. And we found both adaptions helpful to our method.

0.2 Übersicht

Bei der Simulation von deformierbaren Körpern mit Hilfe der Finite-Elemente-Methode kann der Zusammenbau der Hessematrix sehr rechenintensiv sein. In dieser Arbeit schlagen wir eine neue Methode vor, um die Rechenkosten für den Zusammensetzungsprozess zu senken. Wir stellen fest, dass nicht alle lokalen Hessematrizen aufgestellt werden müssen. Während der Simulation identifizieren wir die aktiven Eckpunkte und Elemente anhand der Gradienten- und Nachbarschaftsin-

formationen. Anschließend berechnen wir nur die lokalen Hessematrizen der aktiven Elemente neu und verwenden die lokalen Hessematrizen der inaktiven Elemente aus früheren Iterationen wieder. Dann testen wir, wie verschiedene Faktoren unsere Methode beeinflussen können.

Zunächst testen wir, wie sich verschiedene Simulationsparameter, nämlich Steifigkeitseigenschaften und Größe der Zeitschritte, auf unsere Methode auswirken. Die wichtigste Schlussfolgerung ist, dass unsere Methode besser funktioniert, wenn weiche Materialien oder kleine Zeitschritte verwendet werden. Danach versuchen wir herauszufinden, welche Beziehungen zwischen unserer Methode und den Eigenschaften des Berechnungsnetzes selbst bestehen, nämlich Einheitlichkeit und Auflösung. Wir kommen zu dem Schluss, dass ein gleichmäßiges Netz im Allgemeinen besser abschneidet als ein ungleichmäßiges Netz. Wir sind jedoch nicht in der Lage, die Beziehung zwischen unserer Methode und der Auflösung des Berechnungsnetzes zu erkennen. Zum Schluss passen wir unsere Methode an einige Einschränkungen an. Eine davon ist das Überspringen des Zusammensetzungsprozesses in der ersten Newton-Iteration, eine andere ist die Verwendung unserer Methode mit der projizierten Newton-Methode. Wir haben festgestellt, dass beide Anpassungen für unsere Methode hilfreich sind.

Chapter 1

Introduction

Many physics problems can be modeled as nonlinear systems. Solving these nonlinear systems with Newton-type methods often requires the expensive assembly of the coefficient matrix for each linear solve. However, these coefficient matrices may sometimes be similar from iteration to iteration.

One of these expensive matrices is the Hessian matrix. In Finite Element Method (FEM) simulations, the whole system is subdivided into many small elements. To compute the global Hessian matrix of the whole system, we need to compute the local Hessian matrix of each small element first, then assemble them. In this thesis, we discuss particularly a new method to assemble the global Hessian matrix when using the FEM method to simulate soft body dynamics.

We notice that a soft object can have different degrees of deformation inside its body. For example, Figure 1.1 shows a simulation of a deformed circle. We can see that the deformation is slowly spread from left to the right, which may suggest that when calculating the global Hessian matrix, the local Hessian matrices of the right part elements would remain unchanged. We also notice that when using stiff objects but with small time steps, we could obtain similar simulations as Figure 1.1, in which the deformation is slowly spread from left to the right.

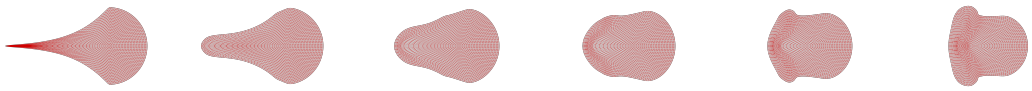


Figure 1.1: Example of a soft material simulation.
Here is a half-compressed circle returning to the rest mode. The images from left to right are the 0th, 20th, 40th, 60th, 80th, and 100th frames of the simulation.

In addition, when doing simulations with small time steps, the process of global Hessian matrix assembly can be the bottleneck of simulations. Compared with simulations with large time steps, the displacement will be small between

each frame. As a result, for each frame, the initial guess is very close to the solution of this nonlinear system and this global Hessian matrix is more likely to become a positive definite matrix. When using Newton-type methods, we also need to solve one or more linear system equations. With the property of positive definiteness, it gives us the chance to use some faster linear solvers, such as conjugate gradient (CG), Cholesky decomposition. However, the assembly process would cost the same time no matter what time step we choose. To sum up, in simulations with small time steps, the assembly process can easily become the bottleneck.

Based on such observations, we propose a novel idea for the fast global Hessian matrix assembly process in FEM. When doing simulations, we separate the mesh vertices into two parts, active vertices, and inactive vertices. For elements have at least one active vertices, we refer to them as active elements. We compute the local Hessians of active elements at current iteration. For elements that only contain inactive vertices, we refer to them as inactive elements. We reuse the local Hessians of inactive elements from the previous iteration. Then we assemble them into the global Hessian matrix.

Several questions naturally arise from this idea:

1. How can we separate the active and inactive vertices, and correspondingly active and inactive elements?

2. How does this method compare with vanilla Newton's method?

3. What are the limitations and possible improvements of this method?

These questions already provide good guidance through this thesis.

To start, in chapter 2 we show the current development of the research in physically-based simulations and the FEM method. In chapter 3 we give a brief introduction to the mathematical background of Newton's method and FEM simulations. In chapter 4 we introduce our method to assemble only part of local Hessians into the global Hessian. In chapter 5 we investigate how our method performs in different situations, such as soft materials and hard materials. In chapter 6 we give a conclusion of this thesis and show several unanswered questions which may be addressed in the future.

There are two points that we need to mention here. First, in chapter 4 we introduce our method, but our method is not a ready-to-use method. Because our method relies on two parameters, we are unable to pick the parameters automatically. In chapter 5, we pick the best parameters from the experiment results. Second, we don't have explicit time measurement in this thesis, but our method is to cut down the computational cost by assembling only part of local Hessian matrices. In principle, it should lead to faster physical simulations.

Chapter 2

Related Work

After the pioneering work by Terzopoulos et al, physically-based simulations has been wildly used in feature animation and visual effects [TPBF87]. After that many models have been introduced to simulate the physical world, such as position based dynamics [MHHR07] [MMC16], smoothed-particle hydrodynamics method [MCG03] [BK15], material point method [JST+16].

2.1 Numerical Solver

In physical simulations, it's very common to numerically compute the time integration. Backward Euler method, as one of the time integration schemes, has been commonly used in physical simulations, see [BW98], [HFS+01], [VM01]. And it's a typical method that would involve one or more nonlinear systems. Newton-Raphson method, as an iterative root-finding algorithm, can be used to solve the nonlinear equations while using the backward Euler method.

It is known that there exist several limitations of the Newton-Raphson method, the most one of them is that the Newton-Raphson method may fail to converge while using large time steps. To avoid this problem, the idea of converting the root-finding problem into an optimization problem, where robust solutions exist, has been used in many papers [KROM99] [GSS+15] [KP12]. Among all the optimization algorithms, Newton's Method and its variants, have been used frequently [GSS+15] [LBK17] [LGL+19].

2.2 Finite Element Methods

To numerically solve soft body simulations, the continuum is usually discretized by many small pieces. Many different techniques for spatial discretization have been developed, e.g. finite difference schemes [TPBF87], the boundary element

method [JP99], or the finite volume method [TBHF03]. Among many methods, the finite element method (FEM) as a general tool to numerically solve partial differential equations has been widely used in soft body simulations, see [SGK18] [LLJ+20] [LLK+20].

The optimization scheme can also be applied to FEM simulations [KKB18]. Under certain conditions, these nonlinear equations can be converted into energy minimization problems.

However, using Newton's method to solve FEM simulations is computationally expensive, and many methods have been introduced to reduce the computational cost. Kugelstadt et al. used an operator splitting approach for corotated linear elastic model, and it can achieve real-time performance [KKB18]. Liu et al. used Quasi-Newton Methods and the performance has been improved averagely more than 10 times faster than one iteration of Newton's method without losing quality [LBK17].

In this thesis, we will introduce our method to cut down the computational cost in FEM simulations by assembling fewer elements, while keeping the same number of Newton iterations. In particular we will use Newton's method in FEM simulations. Compared with Quasi-Newton Methods, it can provide faster converge rate. And our method should be able to applied to any non-linear elastic model, rather than limited to corotated linear elastic model.

Chapter 3

Background

3.1 Backward Euler Method

First, we start with equations of motion

$$\dot{\mathbf{x}} = \mathbf{v} \quad \mathbf{M}\dot{\mathbf{v}} = \mathbf{f}(\mathbf{x}, \mathbf{v}),$$

where $\mathbf{f}$ are forces, which usually is a function of position $\mathbf{x}$ and velocity $\mathbf{v}$. Also, $\mathbf{M}$ is usually a diagonal lumped-mass matrix.

To solve the equation numerically, we discretize the equation by time

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

Here Δt is the size of each time step, superscript symbols mean the value in different time steps. For example, $\mathbf{x}^t$ means the position vector $\mathbf{x}$ at the t-th time step, $\mathbf{x}^{t+1}$ means the position vector $\mathbf{x}$ at the next time step after the t-th time step.

We combine these two equations and it yields

$$\mathbf{M}\frac{\frac{\mathbf{x}^{t+1}-\mathbf{x}^t}{\Delta t} - \mathbf{v}^t}{\Delta t} = \mathbf{M}\frac{\mathbf{x}^{t+1} - \mathbf{x}^t - \Delta t\mathbf{v}^t}{\Delta t^2} = \mathbf{f}(\mathbf{x}^{t+1}, \frac{\mathbf{x}^{t+1} - \mathbf{x}^t}{\Delta t}).$$

Here $\mathbf{x}^{t+1}$ is the variable that we want to solve, $\mathbf{x}^t$, $\mathbf{v}^t$, Δt and $\mathbf{M}$ are known variables. So we can rewrite it as

$$\mathbf{h}(\mathbf{x}^{t+1}) = \mathbf{M}\frac{\mathbf{x}^{t+1} - \mathbf{x}^t - \Delta t\mathbf{v}^t}{\Delta t^2} - \mathbf{f}(\mathbf{x}^{t+1}, \frac{\mathbf{x}^{t+1} - \mathbf{x}^t}{\Delta t}) = 0. \tag{3.1}$$

Then we can solve the equation using the Newton-Raphson method. Given an initial guess $\mathbf{x}_0$, then iteratively update the solution as

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \mathbf{J}_\mathbf{h}(\mathbf{x}_n)^{-1}\mathbf{h}(\mathbf{x}_n), \tag{3.2}$$

where $\mathbf{J_h}(\mathbf{x}_n)$ is the Jacobian matrix of $\mathbf{h}$ with respect to $\mathbf{x}_n$. The subscript symbols mean the value in n-th iteration of Equation 3.2. We keep iterating using Equation 3.2 until

$$|\mathbf{h}(\mathbf{x}_n)| < \tau.$$

τ here controls how accurate the solution we want is. The smaller value τ is, the more accurate solution is. When the solution is accurate enough, $\mathbf{x}_n$ is the solution to Equation 3.1.

After solving $\mathbf{x}^{t+1}$, we can obtain $\mathbf{v}^{t+1}$ by

$$\mathbf{v}^{t+1} = \frac{\mathbf{x}^{t+1} - \mathbf{x}^t}{\Delta t}.$$

3.2 Minimization Problem

However, the Newton-Raphson method may fail sometimes due to its instability. To avoid this, a new idea of converting this root-finding problem into a minimization problem, where robust solutions exist, has been used [KROM99] [GSS+15]. To convert it, Gast et al. introduced several additional assumptions, particularly the one that $\mathbf{f} = -\nabla\Phi$ for some function $\Phi(\mathbf{x})$. More details and discussions about these assumptions can be found at [GSS+15].

Here we use the variable $\mathbf{x}$ to replace the unknown variable $\mathbf{x}^{t+1}$. Given these assumptions, Equation 3.1 can be rewritten as

$$\mathbf{h}(\mathbf{x}) = \mathbf{M}\frac{\mathbf{x} - \mathbf{x}^t - \Delta t\mathbf{v}^t}{\Delta t^2} + \nabla\Phi = 0. \tag{3.3}$$

If we set

$$E(\mathbf{x}) = \frac{1}{2\Delta t^2}(\mathbf{x} - \mathbf{x}^t - \Delta t\mathbf{v}^t)^T\mathbf{M}(\mathbf{x} - \mathbf{x}^t - \Delta t\mathbf{v}^t) + \Phi, \tag{3.4}$$

then we have $\nabla E(\mathbf{x}) = \mathbf{h}(\mathbf{x})$. In the following part of this thesis, we are going to use $\hat{\mathbf{x}}$ to represent $\mathbf{x}^t + \Delta t\mathbf{v}^t$. Gast et al. have shown that when all required assumptions are met, Equation 3.4 always has a global minimum, and any local minimum of Equation 3.4 is a solution to Equation 3.1.

In addition, gravity plays a significantly important role in physical simulations. Adding the gravity into Equation 3.3 yields

$$\mathbf{h}(\mathbf{x}) = \mathbf{M}\frac{\mathbf{x} - \hat{\mathbf{x}}}{\Delta t^2} - \mathbf{M}\mathbf{g} + \nabla\Phi = 0, \tag{3.5}$$

where $\mathbf{g}$ is the gravity vector. Furthermore, we can obtain the energy function

$$E(\mathbf{x}) = \frac{1}{2\Delta t^2}(\mathbf{x} - \hat{\mathbf{x}} - \Delta t^2\mathbf{g})^T\mathbf{M}(\mathbf{x} - \hat{\mathbf{x}} - \Delta t^2\mathbf{g}) + \Phi. \tag{3.6}$$

We will refer to $\frac{1}{2\Delta t^2}(\mathbf{x} - \hat{\mathbf{x}} - \Delta t^2\mathbf{g})^T\mathbf{M}(\mathbf{x} - \hat{\mathbf{x}} - \Delta t^2\mathbf{g})$ as the inertia part of Equation 3.6.

3.3 Newton's Method

To minimize the Equation 3.6, Newton's method can be used here. Though Gast et al. didn't mention it clearly, the energy Φ should be second-order differentiable, though this holds in most cases. As a result, the energy function E is second-order differentiable as well. To minimize $E(\mathbf{x})$, we can expand it using second-order Taylor approximations,

$$E(\mathbf{x}+\Delta\mathbf{x}) \approx E(\mathbf{x}) + \Delta\mathbf{x}^T\nabla E + \frac{1}{2}\Delta\mathbf{x}^T\mathbf{H}\Delta\mathbf{x}, \tag{3.7}$$

where $\mathbf{H}$ is the Hessian matrix, and ∇E is the gradient vector. If we assume the Hessian matrix $\mathbf{H}$ is a positive definite matrix, we can find the $\Delta\mathbf{x}$ which minimizes the Equation 3.7, by setting its derivative to zero. We obtain the following result

$$\begin{aligned} \Delta\mathbf{x} &= -\mathbf{H}^{-1}\nabla E \\ E(\mathbf{x}+\Delta\mathbf{x}) - E(\mathbf{x}) &\approx -\frac{1}{2}\Delta\mathbf{x}^T\mathbf{H}\Delta\mathbf{x} \end{aligned} \tag{3.8}$$

In conclusion, to minimize the energy function E, we can start with an initial guess $\mathbf{x}_0$, and iteratively update $\mathbf{x}_{n+1} = \mathbf{x}_n - \mathbf{H}^{-1}\nabla E$, until it reaches the minimum. $\mathbf{x}^*$ is a local minimum of Equation 3.6, if $\nabla E(\mathbf{x}^*) = \mathbf{0}$ and $\mathbf{H}(\mathbf{x}^*)$ is a positive semi-definite matrix. When computing the minimum numerically, it's usually hard to have strict $\nabla E(\mathbf{x}^*) = \mathbf{0}$. So we usually say $\mathbf{x}^*$ is minimum, if $|\nabla E(\mathbf{x}^*)| \leq \tau$, here τ controls the accuracy of the minimum. When τ becomes smaller, the minimum will become more accurate. Here we call $\Delta\mathbf{x} = -\mathbf{H}^{-1}\nabla E$ as one Newton step.

Algorithm 1 Line Search

```
1: procedure LINE SEARCH ( x, Δx, α )
2:     while E(x + αΔx) > E(x) + κα∇E^T Δx do
3:         α ← cα
4:     end while
5:     return α
6: end procedure
```

To ensure convergence to the minimum, the line search technique can be used here.After abtaining the Newton step, we introduce another minimization problem.

$$\min_{\alpha} \phi(\alpha) = \min_{\alpha} E(\mathbf{x}+\alpha\Delta\mathbf{x}).$$

However minimizing it can be as hard as minimizing Equation 3.6. In this thesis, we use backtracking line search technique [NW06]. It can avoid overshooting

problem and ensure the sufficient decrease of energy function. It means that the energy decreased should be larger than $\kappa\alpha\nabla E^T\Delta\mathbf{x}$, for some constant $\kappa \in (0, 1)$. This can be always achieved if α is small enough. So if we don't have sufficient decrease, then we replace α by $c\alpha$, where $c \in (0, 1)$. An overview of line search technique can be found at Algorithm 1. Here we start the α from 1.

Now we show an overview of Newton's method at Algorithm 2. Here we call one iteration of the while loop as one Newton iteration. The discussions about initial guess $\mathbf{x}_0$ can be found at Section 4.4.

Algorithm 2 Newton's Method

procedure NEWTON'S METHOD ($\mathbf{x}_0$)
  compute gradient ∇E, at current $\mathbf{x}_0$
  $n \leftarrow 0$
  while $||\nabla E||_2 > \tau$ **do**
    compute the Hessian $\mathbf{H}$, at current $\mathbf{x}_n$
    $\Delta\mathbf{x} \leftarrow -\mathbf{H}^{-1}\nabla E$
    $\alpha = \text{LINE SEARCH}(\mathbf{x}_n, \Delta\mathbf{x}, 1)$
    $\mathbf{x}_{n+1} \leftarrow \mathbf{x}_n + \alpha\Delta\mathbf{x}$
    compute gradient ∇E, at current $\mathbf{x}_{n+1}$
    $n \leftarrow n + 1$
  end while
  return $\mathbf{x}_{n+1}$
end procedure

3.4 Soft Body Simulations

Soft body simulations have been researched for many years. To describe the motion of deformation in the continuum, we can imagine a particle inside the continuum. The movement of a such particle is described by

$$\mathbf{x} = \mathbf{x}(\mathbf{X}, t), \text{ with } \mathbf{X} = \mathbf{x}(\mathbf{X}, t_0),$$

where $\mathbf{x}$ is the function to describe the movement of such particle position vector depending on time, and $\mathbf{X}$ is the position vector from reference configuration. Then we use the deformation gradient to describe the motion, which is defined as

$$\mathbf{F} = \frac{\partial\mathbf{x}(\mathbf{X}, t)}{\partial\mathbf{X}}.$$

In addition, $\mathbf{F}$ is a $d \times d$ matrix, where d is the dimension. Normally, d can be either 2 or 3.

Once we have the deformation gradient, we define a strain measure for this deformation. Strain describes the deformation in terms of the relative displacement of the particles in a body where rigid-body motions, such as rotations and translations, are excluded. There are many ways to describe the strain. First, the right Cauchy-Green deformation tensor is defined as

$$\mathbf{C} = \mathbf{F}^T\mathbf{F},$$

where $\mathbf{F}$ is the deformation gradient. If the particle is only doing rigid-body motion, $\mathbf{F}$ would be a unitary matrix, and $\mathbf{C}$ would be an identity matrix. Then Green Strain is defined as

$$\mathbf{E}_{\text{green strain}} = \frac{1}{2}(\mathbf{F}^T\mathbf{F} - \mathbf{I}).$$

If the particle is only doing rigid-body motion, then we have $\mathbf{E} = 0$.

When elastic deformation happens, it will create potential energy, which is referred to as strain energy. The strain energy for a single particle can also be considered as an energy density function. There are many ways to define the energy density function. In this thesis, we will focus on the Neohookean model [SB12], which is defined as

$$\Psi(\mathbf{F}) = \frac{\mu}{2} \cdot (\mathrm{tr}(\mathbf{F}^T\mathbf{F}) - d - 2\ln(\det(\mathbf{F}))) + \frac{\lambda}{2} \cdot (\det(\mathbf{F}) - 1), \tag{3.9}$$

where d is the dimension, it could be either 2 or 3, μ and λ are Lamé coefficients that define the material properties. The strain energy is defined by integrating the energy density function Ψ over the entire body Ω.

$$E_{\text{elastic}} = \int_{\Omega} \Psi(\mathbf{F})\, \mathrm{d}\mathbf{X} \tag{3.10}$$

Other than Lamé coefficients, there are other ways to describe the material properties, such as using Young's modulus E_{young} and Poisson's ratio ν. Young's modulus is a mechanical property that measures the tensile or compressive stiffness of solid material when the force is applied lengthwise. Poisson's ratio is a measure of the deformation of a material in directions perpendicular to the specific direction of loading. The conversions between Lamé coefficients and Young's modulus with Poisson's ratio are

$$\lambda = \frac{E_{\text{young}}\nu}{(1+\nu)(1-2\nu)} \quad \mu = \frac{E_{\text{young}}}{2(1+\nu)}$$

3.5 Finite Element Method

To solve the movement of a soft body with computers, we have to convert the continuous equation into discrete equations. In this thesis, we will particularly use finite element method (FEM) here.

We subdivide the deformable solid into a finite number of parts with simple geometry, so-called finite elements. And these elements are connected by their vertices. In this thesis, we only consider linear elements, such as triangles, and tetrahedra. And the physical behavior of such elements can be computed using well-known shape functions. For example, Figure 3.1 shows a way to subdivide a circle using triangles [Com22].

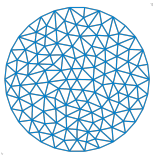


Figure 3.1: Example of subdividing a circle using triangles.

To numerically describe the system with elements, we give each vertex of all the elements a position vector, a velocity vector, and a mass value. The position vector and the velocity vector are the things we want to solve. And we concentrate the mass of an element on each vertex of this element, the mass per each vertex is

$$m_j = \sum_{i \in \mathcal{N}_j} \frac{1}{N_i} \rho V_i, \tag{3.11}$$

where $\mathcal{N}_j$ are the elements that are adjacent to j-th vertex, V_i is the volume of the i-th element and ρ is the density of the object, N_i is the number of vertices of i-th element. With the position vector, the velocity vector, and mass value we can compute the inertia part in Equation 3.6.

It's worth mentioning that this matrix is called mass lumping matrix, and it's different from the mass matrix inside the FEM concept. But when using linear elements, this lumping matrix can provide a good approximation to the mass matrix inside the FEM concept [ZTZ13].

Then, we need to use linear approximation to approximate Equation 3.10. When using linear elements, the deformation gradient is constant for every particle inside one element. So we have

$$E_{\text{elastic}}(\mathbf{x}) = \int_{\Omega} \Psi(\mathbf{F}) \, \mathrm{d}\mathbf{X} \approx \sum_{i}^{n_{\text{ele}}} \Psi(\mathbf{F}_i) V_i, \tag{3.12}$$

where n_{ele} is the number of elements, $\mathbf{F}_i$ is the deformation gradient of i-th element, and V_i is the volume of i-th element.

To counteract the deformation, a force would happen as well, and the force is defined as the negative gradient of the strain energy. So for one element, the

elastic force will be applied to all of its vertices. The total elastic force is the sum of all elastic forces and we have

$$\mathbf{f}(\mathbf{x}) = -\frac{\partial E_{\text{elastic}}(\mathbf{x})}{\partial \mathbf{x}} = -\sum_{i}^{n_{\text{ele}}} \frac{\partial E_{\text{elastic},i}(\mathbf{x})}{\partial \mathbf{x}}$$

Gast et al. showed the force $\mathbf{f}$ and energy E can satisfy the assumption used in Equation 3.5 and Equation 3.6 [GSS+15]. So we can use the minimization technique mentioned in the previous section.

In this thesis, we only consider the inertia part and elastic energy, which means Equation 3.6 would become

$$E(\mathbf{x}) = \frac{1}{2\Delta t^2}(\mathbf{x} - \hat{\mathbf{x}} - \Delta t^2 \mathbf{g})^T \mathbf{M}(\mathbf{x} - \hat{\mathbf{x}} - \Delta t^2 \mathbf{g}) + E_{\text{elastic}}(\mathbf{x}) \tag{3.13}$$

Here we show how to calculate the local elastic energy of an element. The derivation is from the course notes [SB12]. We will use a two-dimensional triangle as an example here. The derivation for three-dimensional tetrahedron can be found in Appendix A. Now, given an undeformed two-dimensional triangle $\triangle ABC$, with vertices coordinates $(X_1, Y_1), (X_2, Y_2), (X_3, Y_3)$. When the triangle has been deformed, similarly, we can have triangle $\triangle A'B'C'$ with vertices coordinates $(x_1, y_1), (x_2, y_2), (x_3, y_3)$, as shown in figure Figure 3.2.

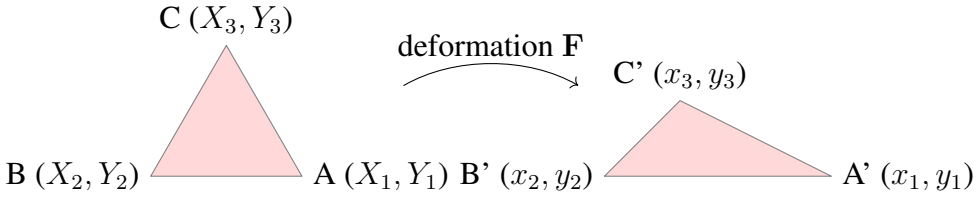


Figure 3.2: An Example of Triangle Deformation.

Now we can calculate

$$\mathbf{D}_m = \begin{pmatrix} X_1 - X_3 & X_2 - X_3 \\ Y_1 - Y_3 & Y_2 - Y_3 \end{pmatrix} \text{ and } \mathbf{D}_s = \begin{pmatrix} x_1 - x_3 & x_2 - x_3 \\ y_1 - y_3 & y_2 - y_3 \end{pmatrix}. \tag{3.14}$$

The deformation gradient $\mathbf{F}$ and the Green Strain are defined as

$$\mathbf{F} = \mathbf{D}_s \mathbf{D}_m^{-1} \text{ and } \mathbf{E}_{\text{green strain}} = \frac{1}{2}(\mathbf{F}^T\mathbf{F} - \mathbf{I}), \tag{3.15}$$

where $\mathbf{I}$ is the identity matrix. And $\mathbf{D}_m^{-1}$ is not going to change for the whole simulation and can be pre-computed.

Algorithm 3 Solve Soft Body simulations using FEM

initialize rest configuration position vector $\mathbf{x}_{\text{rest}}$
initialize physical properties μ, λ, ρ
for each element i **do**
    compute volume V_i
    compute $\mathbf{D}_{m,i}$ using Equation 3.14
    for each vertex j of i-th element **do**
        compute mass contribution $m \leftarrow \frac{1}{N_i} V_i \cdot \rho$
        update mass for j-th vertex $m_j \leftarrow m_j + m$
    end for
end for
assemble mass matrix $\mathbf{M}$
initialize current position and velocity vector $\mathbf{x}^0$ and $\mathbf{v}^0$
initialize time step Δt
initialize Lamé coefficients μ and λ
initialize gravity vector $\mathbf{g}$
$t \leftarrow 0$

procedure ENERGY ($\mathbf{x}$, $\mathbf{x}^t$, $\mathbf{v}^t$)
    compute $\hat{\mathbf{x}} \leftarrow \mathbf{x}^t + \Delta t \mathbf{v}^t$
    compute $E_{\text{inertia}} \leftarrow \frac{1}{2\Delta t^2}(\mathbf{x} - \hat{\mathbf{x}} - \Delta t^2 \mathbf{g})^T \mathbf{M} (\mathbf{x} - \hat{\mathbf{x}} - \Delta t^2 \mathbf{g})$
    $E_{\text{elastic}} \leftarrow 0$
    for each element i **do**
        compute $\mathbf{D}_{s,i}$ using Equation 3.14
        $\mathbf{F}_i \leftarrow \mathbf{D}_{s,i} \mathbf{D}_{m,i}$
        $\Psi(\mathbf{F}_i) \leftarrow \frac{\mu}{2} \cdot (\text{tr}(\mathbf{F}_i^T \mathbf{F}_i) - d - 2\ln(\det(\mathbf{F}_i))) + \frac{\lambda}{2} \cdot (\det(\mathbf{F}_i) - 1)$,
        $E_{\text{elastic},i} \leftarrow \Psi(\mathbf{F}_i) V_i$
        $E_{\text{elastic}} \leftarrow E_{\text{elastic}} + E_{\text{elastic},i}$
    end for
    return $E_{\text{elastic}} + E_{\text{inertia}}$
end procedure

while runnning simulation **do**
    minimize ENERGY$(\mathbf{x}, \mathbf{x}^t, \mathbf{v}^t)$, w.r.t. $\mathbf{x}$, using Algorithm 2
    $\mathbf{x}^{t+1} \leftarrow \mathbf{x}$
    $\mathbf{v}^{t+1} \leftarrow \frac{\mathbf{x}^{t+1} - \mathbf{x}^t}{\Delta t}$
    $t \leftarrow t + 1$
end while

When we have the deformation gradient $\mathbf{F}$ for each element, we can use the Neohookean model from Equation 3.9 to calculate the elastic energy for each element.

$$E_{\text{elastic},i} = \Psi(\mathbf{F}_i)V_i \tag{3.16}$$

Now we have shown how to calculate every part of Equation 3.13, the overall algorithm to solve soft body simulations can be found at Algorithm 3.

3.6 Assembly Process

In Equation 3.13, we map local position vector of each element $\mathbf{x}_i$ into the global position vector $\mathbf{x}$, so we can update all together. In this section, we show that extra attention must be paid when assembling gradient vectors and Hessian matrices.

When using Newton's method, we need to compute the gradient and the Hessian of the Equation 3.13. From Equation 3.5 and Equation 3.6 we already know that the gradient of Equation 3.13 is

$$\begin{aligned}\nabla E(\mathbf{x}) = \mathbf{h}(\mathbf{x}) &= \mathbf{M}\frac{\mathbf{x}-\hat{\mathbf{x}}}{\Delta t^2} - \mathbf{M}\mathbf{g} + \nabla E_{\text{elastic}} \\ &= \mathbf{M}\frac{\mathbf{x}-\hat{\mathbf{x}}}{\Delta t^2} - \mathbf{M}\mathbf{g} + \sum_i^{n_{\text{ele}}} \nabla E_{\text{elastic},i},\end{aligned}$$

and it's a vector with size dn_{ver}, where d is the number of dimension, and n_{ver} is the number of vertices. Also, we can derive the Hessian matrix of Equation 3.13 is

$$\mathbf{H} = \frac{\mathbf{M}}{\Delta t^2} + \mathbf{H}_{\text{elastic}} = \frac{\mathbf{M}}{\Delta t^2} + \sum_i^{n_{\text{ele}}} \mathbf{H}_{\text{elastic},i}, \tag{3.17}$$

and it's a $dn_{\text{ver}} \times dn_{\text{ver}}$ matrix.

Because we consider $\mathbf{M}$ as a constant matrix, Δt as constant, and $\mathbf{g}$ as a constant vector in this thesis. So $\mathbf{M}\frac{\mathbf{x}-\hat{\mathbf{x}}}{\Delta t^2} - \mathbf{M}\mathbf{g}$ and $\frac{\mathbf{M}}{\Delta t^2}$ would be very simple to compute. So we mostly focus on the $\nabla E_{\text{elastic}}$ and $\mathbf{H}_{\text{elastic}}$ here.

There are many ways to compute the local gradient and the Hessian matrix of a single element. For example, manually deriving the symbolic function of gradient and the Hessian matrix is one accurate but tedious option. In Appendix B, we introduce a technique, called Automatic Differentiation (AD), can easily compute the gradient and the Hessian matrix of a function specified by a computer program. And this is what we used in our implementation.

After computing the local elastic gradient and the elastic Hessian matrix, we can see that the global elastic gradient $\nabla E_{\text{elastic}}$ and the elastic Hessian $\mathbf{H}_{\text{elastic}}$ are assembled by summing up the local elastic gradient vectors and the local elastic Hessian matrices.

The position vector of a single element $\mathbf{x}_i$ is a vector with size $d(d+1)$, because a vertex has d coordinates, and an element has $d+1$ vertices. So, the local elastic gradient vector also has the size $d(d+1)$, and the local elastic Hessian has the size $d(d+1) \times d(d+1)$. So, we need to map the local elastic gradient vector and the elastic Hessian to the global size.

If the vector $\mathbf{x}_i$ are mapped into the global position vector $\mathbf{x}$ by the function $\mathcal{M}_i$. An example of this function $\mathcal{M}_i$ can be mapping the first component in $\mathbf{x}_i$ to the 10th component of the global position vector $\mathbf{x}$. Then we can map the local elastic gradient using $\mathcal{M}_i$ as well. So we obtain a dn_{ver} for the local elastic gradient vector. Then we can sum up all the local elastic gradient vectors. Similarly, we can use $\mathcal{M}_i$ to map both each columns and each rows of the local elastic Hessian, then we can obtain a local elastic Hessian matrix with size $dn_{\text{ver}} \times dn_{\text{ver}}$.

From Equation 3.8, the energy decreased approximates to $\frac{1}{2}\Delta\mathbf{x}^T\mathbf{H}^{\Delta}\mathbf{x}$, so ideally we want $\mathbf{H}$ to be a positive definite matrix, so the energy value can always be decreased no matter what $\Delta\mathbf{x}$ is. But $\mathbf{H}$ is not always a positive definite matrix. To deal with it, we can modify the matrix $\mathbf{H}$ to a positive definite matrix. Many modification algorithms rely on the eigendecomposition of the matrix. In practice, it's almost impossible to run an eigendecomposition on the global Hessian matrix $\mathbf{H}$ due to the complexity. So we could consider modifying the local elastic Hessian matrix $\mathbf{H}_{\text{elastic},i}$ to a positive semi-definite matrix because $\mathbf{H}_{\text{elastic},i}$ is a small matrix. After that, the global Hessian matrix $\mathbf{H} = \frac{\mathbf{M}}{\Delta t^2} + \Sigma_{i=0}^{n}\mathbf{H}_{\text{elastic},i}$ will be a positive definite matrix. And this technique has been widely used in simulations [LGL+19]. Furthermore, we will refer to this method as Projected Newton's (PN) method in the rest of this thesis. More discussions about the Hessian modifications can be found in Section 5.6.

Chapter 4

Fast Hessian Assembly

From Section 3.6 we showed the process of assembling global Hessian matrix. So we need to compute the local Hessian matrices of each element, map it into a large size matrix, then sum it up. So the assembly process can be extremely expensive if the number of elements is big. In the meanwhile, when using small time steps, the assembly process can become the bottleneck in the Newton iteration. Because the global Hessian matrix is more likely to become a positive definite matrix with small time steps. With the property of positive definiteness, we can use some faster linear solvers, such as conjugate gradient (CG), or Cholesky decomposition. So here we describe our new idea of only assembling part of the local Hessian matrices.

First we classify the vertices of the object as active vertices and inactive vertices. We have already known that, if $\mathbf{x}^*$ is the minimum of the Equation 3.6, then it means that the gradient vector $\nabla E(\mathbf{x}^*)$ would be close to zero. So, we here suggest classifying the active vertices based on the gradient value. In particular, we define the active vertices as the vertices whose absolute gradient value are larger than $\epsilon \cdot ||\nabla E||_\infty$. Here we introduce an additional parameter $\epsilon \in [0, 1)$.

Furthermore, we have noticed that, the deformation won't spread by skipping vertices. So we think the neighbors are also a key feature to identity the active vertices. So, once we find the active vertices from the previous step, we classify their k neighbors as active vertices as well. Here the k is another parameter introduced by our algorithm.

Now we have the active vertices, we can classify all the elements as active elements and inactive elements. Active elements are the elements which have at least one active vertex. Inactive elements are the elements which have only inactive vertices. Then we compute the local Hessian matrix for the active elements, and reuse the local Hessian from the previous Newton iteration for inactive elements.

Compared with the vanilla Newton's method, Algorithm 2, we add an additional step to find the active vertices and elements, then modify the step to assemble the global Hessian matrix. And if we choose $\epsilon = 0$ and $k = 0$ as our parameters, then our method would fall back into the vanilla Newton's method.

4.1 Initialization

To reuse the local Hessian matrices from the previous time steps, we have to keep a record of these local Hessian matrices and the global Hessian matrix at each Newton iteration. So before we start the simulation, we need to store the global Hessian matrix and a list of all the local Hessian matrices. We will use $\mathbf{H}_{\text{global}}$ and L_{locals} to denote the global Hessian matrix and the list of local elastic Hessians.

It's worth noting that, if the simulation starts from a rest configuration, then the elastic energy and gradient vector are mostly likely to be zero, but the local elastic Hessian matrices are mostly non-zero.

4.2 Active Vertices and Active Elements

Here we show the steps to identity the active vertices and active elements, and how to use this knowledge to speed up the assembly process.

We add the step of finding active vertices and elements before the step to compute the global Hessian matrix. A modified Newton's method can be found at Algorithm 4 . At this step, we have already computed the global gradient. Then we will have two steps to identity the active vertices. First, we iterate over all the vertices, find vertices whose gradient component is larger than $\epsilon \cdot ||\nabla E||_\infty$. Second, we identity active vertices using the neighborhood information. We find all k neighbors of active vertices from the first step. And classify them as active vertices as well.

Here we introduced two additional parameters, ϵ and k. ϵ here controls how many vertices will be considered as active vertices. The higher ϵ is, the less active vertices, which very likely lead to have less elements assembled when computing the Hessian. The higher k is, the more elements we assemble. We think the value of k may relate to how fast the deformation spreads, more discussion can be found at the physical point of view in subsection 5.1.1.

Now with the active vertices, we iterate over all the elements, if the element has at least one active vertex. Then we classify it as an active element.

Algorithm 4 Modified Newton's Method

procedure MODIFIED NEWTON'S METHOD ($\mathbf{x}_0$)
  compute gradient ∇E, at current $\mathbf{x}_0$
  $n \leftarrow 0$
  while $||\nabla E||_2 > \tau$ **do**
    find active vertices and elements using ∇E
    compute the Hessian $\mathbf{H}$ with active elements, at current $\mathbf{x}_n$
    $\Delta \mathbf{x} \leftarrow -\mathbf{H}^{-1} \nabla E$
    $\alpha = \text{LINE SEARCH}(\mathbf{x}_n, \Delta \mathbf{x}, 1)$
    $\mathbf{x}_{n+1} \leftarrow \mathbf{x}_n + \alpha \Delta \mathbf{x}$
    compute gradient ∇E, at current $\mathbf{x}_{n+1}$
    $n \leftarrow n + 1$
  end while
  return $\mathbf{x}_{n+1}$
end procedure

4.3 Assembly Process

We computed the active elements from the previous section, then we show how to assemble only the local elastic Hessian matrices of the active elements into the global Hessian matrix. We iterate over all the active elements, find the local elastic Hessian $\mathbf{H}_{\text{elastic},i,\text{old}}$ from the list of all local elastic Hessian matrices of previous Newton iteration L_{locals}. Then we compute the local elastic Hessian $\mathbf{H}_{\text{elastic},i,\text{new}}$, based on current Newton iteration. Then we compute the difference $\mathbf{H}_{i,\text{diff}} = \mathbf{H}_{\text{elastic},i,\text{new}} - \mathbf{H}_{\text{elastic},i,\text{old}}$. Now we extend $\mathbf{H}_{i,\text{diff}}$ to the suitable size, using the mapping function $\mathcal{M}_i$ to map both rows and columns, more details about $\mathcal{M}_i$ can be found at Section 3.6. Then we add $\mathbf{H}_{i,\text{diff}}$ into $\mathbf{H}_{\text{global}}$, and replace $\mathbf{H}_{\text{elastic},i,\text{old}}$ with $\mathbf{H}_{\text{elastic},i,\text{new}}$. In the end, we use the matrix $\mathbf{H}_{\text{global}}$ as the global Hessian matrix in Algorithm 4.

Because we iteratively update the local Hessian list, then add the difference of local Hessian matrices into the global Hessian matrix, the result is always the same as to sum up all the local Hessian matrices.

4.4 Initial Guess

Gast et al. suggested two ways of computing the initial guess, $\mathbf{x}_0 = \mathbf{x}^t + \Delta t \mathbf{v}^t$ and $\mathbf{x}_0 = \mathbf{x}^t + \Delta t \mathbf{v}^t + \Delta t^2 \mathbf{M}^{-1} \mathbf{f}(\mathbf{x}^t)$, then pick the one with lower energy value as the initial guess [GSS+15]. However, none of this work in our algorithm.

Because we would like to keep using $\mathbf{H}_{\text{global}}$ and L_{locals} when it comes to next time step, rather than recomputing them in the first Newton iteration of every time step. So in our algorithm, we use the position vector from the previous time step $\mathbf{x}^t$ as the initial guess to start the Algorithm 4.

To illustrate the reason behind it, here we use $\mathbf{x}_N$, $\mathbf{x}_{N-1}$ to denote the position vector $\mathbf{x}$ of the last Newton iteration and the second last Newton iteration in the previous time step. So $\mathbf{x}_N$ is the final solution of the previous time step minimization problem, which is $\mathbf{x}^t$. In the last Newton iteration of previous time step, we updated the local elastic Hessian list L_{locals} based on $\mathbf{x}_{N-1}$. Because elastic energy only depends on the position vector $\mathbf{x}$, so does the elastic Hessian matrix. And in the last Newton iteration, the difference between $\mathbf{x}_N$ and $\mathbf{x}_{N-1}$ is usually small. So if we reuse the local Hessian matrices from L_{locals} to assemble the global Hessian matrix at $\mathbf{x}_N$, this is a reasonable choice. We can do this by using $\mathbf{x}_N$ which is also $\mathbf{x}^t$ as the initial guess for the next time step. If we use a different initial guess, then we don't know how valid is to reuse matrices from the list L_{locals}.

Chapter 5

Simulations

In the previous chapter, we showed our method to assemble only some of the local Hessian matrices. But it's still unclear how to choose the parameters ϵ and k. So in this chapter, we will pick the best choice of the parameters from the experiment results, and see how effective our method can be.

In particular, we ran the experiments mostly using a soft beam simulation, as shown in Figure 5.1. In the right bottom corner of the beam, large deformation can happen.

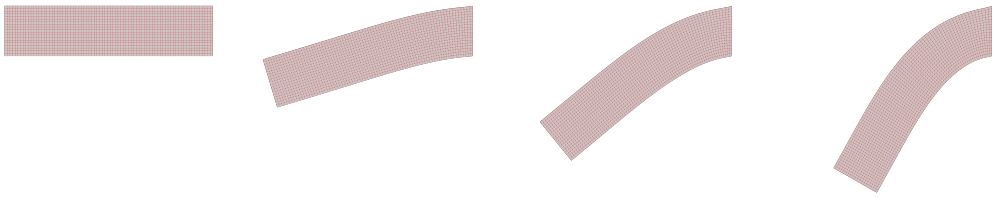


Figure 5.1: An example of beam simulation.
This is a beam falling due to gravity, but with the rightmost vertices fixed. The images from left to right are the 0th, 30th, 60th, 90th, and 120th frames of the simulation. Using different parameters, simulations may look different.

In the following sections, we will show how different factors can affect our method. We will start with using our method with different simulation parameters, which are stiffness properties and time steps. Then we will investigate how the mesh object itself can affect our method, particularly the uniformity and resolution of the mesh object. At last, we will adapt our method to handle special cases. We will introduce two modifications on our method, one is to skip the assembly process in the first Newton iteration, the other one is to use our method on Projected Newton's method.

5.1 Soft and Stiff Materials

As we have seen in Figure 1.1, soft material simulations can be the target of our method. So in this section, we test our method using different stiffness properties.

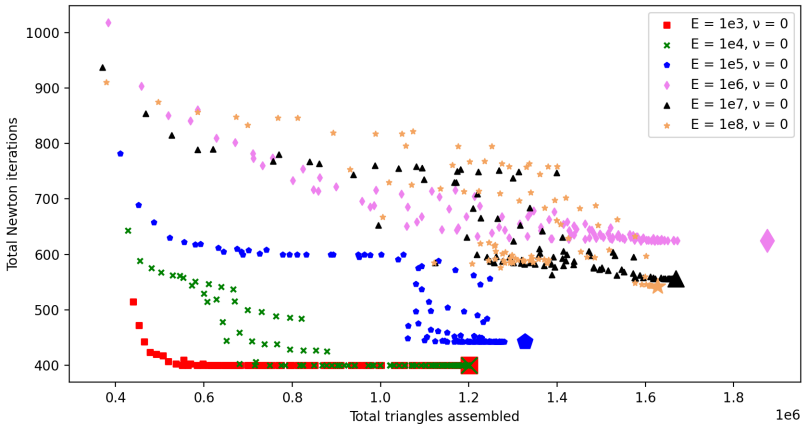
Table 5.1: Other parameters used in the simulation of Figure 5.1.

Parameter	Value	Unit	Description
ρ	10^3	kg/m^2	Density of mesh object
width	0.125	m	Width of each square
height	0.125	m	Height of each square
row	19		Number of squares in one column
col	79		Number of squares in one row
n	2		Number of elements (triangles) in one square
n_{ele}	3002		Number of elements in total
n_{ver}	1600		Number of vertices in total
Δt	0.01	s	Time steps for one frame
frame	200		Number of frames simulated
τ	10^{-7}		Stop criteria of Newton's method
c	0.9		Line search parameter
κ	10^{-4}		Line search parameter
g	9.8	m/s^2	Gravity constant

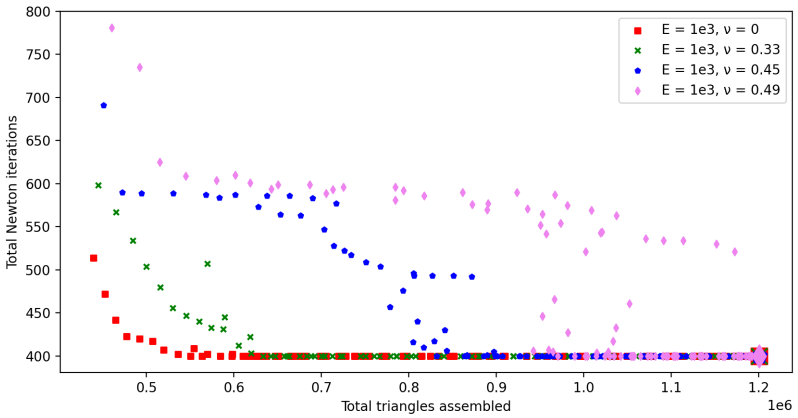
To compare our method in soft and stiff materials, we also tried many different combinations of stiffness properties. First, we ran the simulations with only varying Young's modulus E_{young}. We set the Poisson's ratio $\nu = 0$, then we tried $E_{\text{young}} = 10^3, 10^4, 10^5, 10^6, 10^7, 10^8$ GPa. Second, we ran the simulations with only varying Poisson's ratio ν. We set the Young's modulus $E_{\text{young}} = 10^3$ GPa, then we tried $\nu = 0, 0.33, 0.45, 0.49$. Other parameters can be found at Table 5.1

Here we show the simulation results in Figure 5.2. We ran the simulations with 200 frames. Then for one pair of parameters ϵ and k, we collect the information on how many Newton iterations were computed during these 200 frames, and how many elements were assembled during these 200 frames. In all the figures, the x-axis represents how many elements were assembled in this simulation and the y-axis represents the number of Newton iterations in this simulation. And one scatter point in the figure means a simulation result of one combination of ϵ and k. There is a large point, usually locating at the right bottom of the figure, which means the result of the vanilla Newton's method. And each color and its shape means one combination of the stiffness properties.

Now we can find the optimal parameters ϵ and k. We call a combination of ϵ and k as optimal, if it takes the same number of Newton iterations or less as vanilla



(a) Beam simulations with varying E_{young}.



(b) Beam simulations with varying ν.

Figure 5.2: Beam simulations with different stiffness properties.

Newton's method but assembles the least number of elements. We can simply find the scatter point which lies under the horizontal line with vanilla Newton's method but in the left-most position.

In Figure 5.2a, we compare the simulations results with varying E_{young}. We can see that, with smaller E_{young}, we can assemble fewer elements when using the optimal choice of ϵ and k. Similarly, in Figure 5.2b we compare the simulation results with varying ν, and we have the similar conclusion that we can assemble fewer elements when using the optimal choice of ϵ and k with smaller Poisson's ratio ν.

The full comparison can be found at Table 5.2. The first column is the stiffness properties in the simulation. The second column is the optimal parameters we find from the experiment data. The third column is the percentage of elements assembled compared with vanilla Newton's method, the less value means the better our method performs.

Table 5.2: The results of Figure 5.2 using different stiffness properties.

Stiffness Properties	Optimal ϵ and k	Elements Assembled
E_{young}= 10^3 GPa, ν = 0	$\epsilon = 2^{-1}, k = 8$	45.84%
E_{young}= 10^4 GPa, ν = 0	$\epsilon = 2^{-4}, k = 2$	59.52%
E_{young}= 10^5 GPa, ν = 0	$\epsilon = 2^{-7}, k = 2$	84.64%
E_{young}= 10^6 GPa, ν = 0	$\epsilon = 2^{-11}, k = 2$	84.74%
E_{young}= 10^7 GPa, ν = 0	$\epsilon = 2^{-10}, k = 3$	96.90%
E_{young}= 10^8 GPa, ν = 0	$\epsilon = 2^{-9}, k = 7$	99.63%
E_{young}= 10^3 GPa, ν = 0.33	$\epsilon = 2^{-2}, k = 4$	52.73%
E_{young}= 10^3 GPa, ν = 0.45	$\epsilon = 2^{-4}, k = 3$	69.29%
E_{young}= 10^3 GPa, ν = 0.49	$\epsilon = 2^{-5}, k = 4$	80.83%

5.1.1 Interpretations

To explain the phenomena we have observed, we come up with two different points of view to interpret the results.

1. Mathematical point of view

   From Equation 3.17, we know that the global Hessian equals to $\frac{\mathbf{M}}{\Delta t^2}+\mathbf{H}_{\text{elastic}}$. And we only change the stiffness properties, so $\frac{\mathbf{M}}{\Delta t^2}$ would always remain constant. From Section 3.5, we know that the elastic energy for one element is

   $$\begin{aligned} E_{\text{elastic},i}(\mathbf{x}_i) &= V_i\Psi(\mathbf{F}_i) \\ &= V_i(\frac{\mu}{2}(\text{tr}(\mathbf{F}_i^T\mathbf{F}_i) - d - 2\ln(\det(\mathbf{F}_i))) + \frac{\lambda}{2}(\det(\mathbf{F}_i) - 1)) \\ &= \mu f_i(\mathbf{x}_i) + \lambda g_i(\mathbf{x}_i), \end{aligned}$$

   where f_i and g_i is the function to map from $\mathbf{x}_i$ to the energy density function, but we don't need to know in detail what it is. And the total elastic energy is

   $$E_{\text{elastic}} = \sum_i^{n_{\text{ele}}} E_{\text{elastic},i}(\mathbf{x}_i) = \mu f(\mathbf{x}) + \lambda g(\mathbf{x}). \tag{5.1}$$

Because μ and λ is constant, so $\mathbf{H}_{\text{elastic}}$ can also be written as

$$\mathbf{H}_{\text{elastic}} = \mu\mathbf{H}_1 + \lambda\mathbf{H}_2,$$

where $\mathbf{H}_1$ and $\mathbf{H}_2$ is the Hessian matrix of f and g, and it's independent from stiffness properties μ and λ.

So when we have soft materials, which means a small value of μ and λ, we make the $\frac{\mathbf{M}}{\Delta t^2}$ become the dominant part of the global Hessian matrix. So only updating part of the matrix $\mathbf{H}_{\text{elastic}}$ can have less impact on the number of Newton iterations when the material becomes softer.

2. Physical point of view

   Elastic waves are also commonly seen in physical phenomena. For example, when an earthquake happens, there exists many waves. And we can make a broad distinction between body waves, which travel through the Earth, and surface waves, which travel at the Earth's surface. There are two types of body waves, P waves, and S waves. And we think the idea of body wave propagation can be used to help us to find the correct active vertices. More details about earthquake waves can be found at [She19].

   From either the velocity of P waves or the velocity of S waves

$$v_p = \sqrt{\frac{\lambda + 2\mu}{\rho}} \quad v_s = \sqrt{\frac{\mu}{\rho}},$$

   given by [She19], we can see that the softer material is, the slower elastic waves spread. When the wave propagation becomes slower, there will be more elements unaffected by the elastic waves, which means our method would perform better.

5.2 Small and Large Time Steps

When we do simulations, super soft materials may not exist in the real world. So we have to consider our methods with stiff materials. From the physical point of view of the interpretation in subsection 5.1.1, the number of elements affected by the elastic waves depends on how far the elastic waves travel. And that distance depends on both the speed of elastic waves and the size of time steps. So if we use stiff materials, using small time steps can also be helpful with our method.

In particular, we can here show that in some cases, simulations with small time steps and stiff materials can be converted into simulations with large time steps and soft materials. Now we introduce two independent simulation scenarios.

In the first simulation scenario, it uses the time step Δt, gravity vector $\mathbf{g}$,and Lamé coefficients λ and ν. At current time step, the simulation has position vector $\mathbf{x}^t$ and velocity vector $\mathbf{v}^t$. Using Equation 3.13 and Equation 5.1 for this simulation scenario is

$$\begin{aligned} E_{\text{sim1}}(\mathbf{x}) &= \frac{1}{2\Delta t^2}(\mathbf{x}-\mathbf{x}^t-\Delta t\mathbf{v}^t-\Delta t^2\mathbf{g})^T\mathbf{M}(\mathbf{x}-\mathbf{x}^t-\Delta t\mathbf{v}^t-\Delta t^2\mathbf{g}) \\ &\quad + E_{\text{elastic}} \\ &= \frac{1}{2\Delta t^2}(\mathbf{x}-\mathbf{x}^t-\Delta t\mathbf{v}^t-\Delta t^2\mathbf{g})^T\mathbf{M}(\mathbf{x}-\mathbf{x}^t-\Delta t\mathbf{v}^t-\Delta t^2\mathbf{g}) \\ &\quad + \mu f(\mathbf{x}) + \lambda g(\mathbf{x}). \end{aligned}$$

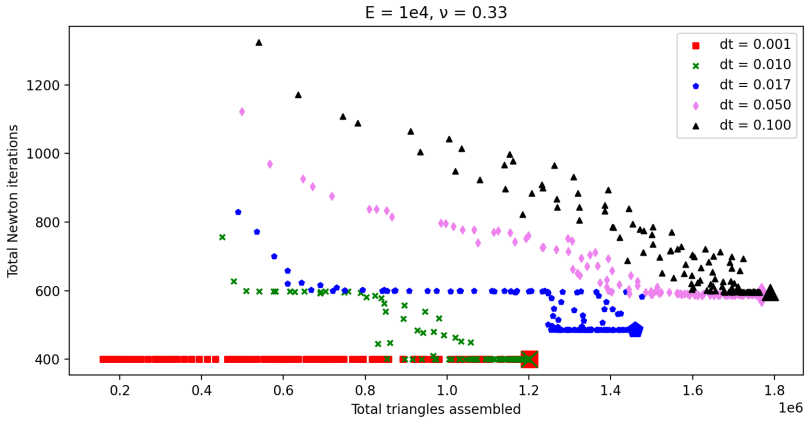
In the second simulation scenario, it uses the time step $c \cdot \Delta t$, gravity vector $\frac{\mathbf{g}}{c^2}$ and Lamé coefficients $\frac{\lambda}{c^2}$ and $\frac{\nu}{c^2}$, where $c \in (0, 1)$. At current time step, the simulation has the position vector $\mathbf{x}^t$ and velocity vector $\frac{\mathbf{v}^t}{c}$. So the second simulation scenario has stiffer material but smaller time steps compared with the first one. The Equation 3.13 for the second simulation would be

$$\begin{aligned} E_{\text{sim2}}(\mathbf{x}) &= \frac{1}{2(c\Delta t)^2}(\mathbf{x}-\mathbf{x}^t-c\Delta t\frac{\mathbf{v}}{c}-(c\Delta t)^2\frac{\mathbf{g}}{c^2})^T\mathbf{M} \\ &\qquad (\mathbf{x}-\mathbf{x}^t-c\Delta t\frac{\mathbf{v}}{c}-(c\Delta t)^2\frac{\mathbf{g}}{c^2}) + E_{\text{elastic}} \\ &= \frac{1}{2(c\Delta t)^2}(\mathbf{x}-\mathbf{x}^t-\Delta t\mathbf{v}-\Delta t^2\mathbf{g})^T\mathbf{M}(\mathbf{x}-\mathbf{x}^t-\Delta t\mathbf{v}-\Delta t^2\mathbf{g}) \\ &\quad + \frac{\mu}{c^2}f(\mathbf{x}) + \frac{\lambda}{c^2}g(\mathbf{x}) \\ &= \frac{1}{c^2}(\frac{1}{2\Delta t^2}(\mathbf{x}-\mathbf{x}^t-\Delta t\mathbf{v}-\Delta t^2\mathbf{g})^T\mathbf{M}(\mathbf{x}-\mathbf{x}^t-\Delta t\mathbf{v}-\Delta t^2\mathbf{g}) \\ &\quad + \mu f(\mathbf{x}) + \lambda g(\mathbf{x})) \\ &= \frac{1}{c^2}E_{\text{sim1}}(\mathbf{x}) \end{aligned}$$

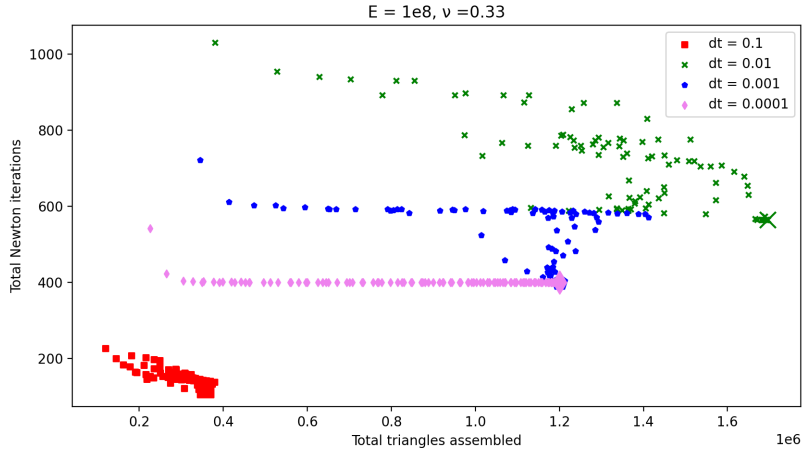
One thing worth mentioning is that, from Section 3.4, we have shown the conversion between Young's modulus, Poisson's ratio, and Lamé coefficients. If we assume that in the first simulation scenario, we have Young's modulus E_{young} and Poisson's ratio ν, then we can easily derive that in the second simulation scenario, we have Young's modulus $\frac{E_{\text{young}}}{c^2}$ and Poisson's ratio ν.

Clearly, E_{sim2} and E_{sim1} have the exact same minimization solution. At the next time step, both simulation scenarios would end up with the position vector $\mathbf{x}^{t+1}$, because of the different time steps, the first simulation scenario has the velocity vector $\mathbf{v}_{\text{sim1}}^{t+1} = \frac{\mathbf{x}^{t+1}-\mathbf{x}^t}{\Delta t}$, and the second simulation scenario has the velocity vector $\mathbf{v}_{\text{sim2}}^{t+1} = \frac{\mathbf{x}^{t+1}-\mathbf{x}^t}{c\cdot\Delta t} = \frac{\mathbf{v}_{\text{sim1}}^{t+1}}{c}$. And it means that, in the next frame, we can reuse

the above derivations, and these two scenarios will lead to the same minimization solution again. So for the entire simulation, both simulation scenarios will have the same minimization solution and the same visual result for each frame.



(a) Beam simulations with soft material and varying Δt.



(b) Beam simulations with stiff material and varying Δt.

Figure 5.3: Beam simulations with varying Δt.

Because the gravity vector g usually remains as a constant vector. Our derivation can only be used in gravity-free simulations, in which case, $g = 0$ m/s^2, so both g and $c \cdot$ g are zero vectors.

However, when gravity is constant but non-zero, then we can't use the derivations above. Now we show how our method performs when using different time steps with a constant non-zero gravity, with experiment data. Here we ran two simulations, similar to Figure 5.1, one with soft materials and one with stiff mate-

rials. And both of these two simulations, we tested with varying time steps, $\Delta t = 0.001, 0.010, 0.016, 0.050, 0.100$ s. Other parameters can be found at Table 5.1.

We show our experiment results in Figure 5.3. In Figure 5.3a, we use soft stiffness properties $E_{\text{young}} = 10^4$ GPa and $\nu = 0.33$. In Figure 5.3b, we use stiff stiffness properties $E_{\text{young}} = 10^8$ GPa and $\nu = 0.33$. We can see that in both soft and stiff materials, our method can be effective in small time steps. A full comparison can be found at Table 5.3.

Table 5.3: The results of Figure 5.3 using different time steps.

Stiffness Properties	Time steps	Optimal ϵ and k	Elements Assembled
$E_{\text{young}} = 10^4$GPa $\nu = 0.33$	$\Delta t = 0.001$s	$\epsilon = 2^{-1}, k = 0$	13.20%
	$\Delta t = 0.010$s	$\epsilon = 2^{-4}, k = 2$	71.07%
	$\Delta t = 0.017$s	$\epsilon = 2^{-9}, k = 1$	85.98%
	$\Delta t = 0.050$s	$\epsilon = 2^{-8}, k = 3$	90.81%
	$\Delta t = 0.100$s	$\epsilon = 2^{-10}, k = 3$	93.86%
$E_{\text{young}} = 10^8$GPa $\nu = 0.33$	$\Delta t = 10^{-1}$s	$\epsilon = 2^{-4}, k = 9$	95.66%
	$\Delta t = 10^{-2}$s	$\epsilon = 2^{-8}, k = 7$	98.49%
	$\Delta t = 10^{-3}$s	$\epsilon = 2^{-7}, k = 9$	98.46%
	$\Delta t = 10^{-4}$s	$\epsilon = 2^{-2}, k = 0$	29.11%

5.2.1 Interpretations

When using small time steps simulations, the interpretations from subsection 5.1.1 still hold.

1. When using smaller time steps, we make the matrix $\frac{\mathbf{M}}{\Delta t^2}$ become larger, which still can become the dominant factor in the global Hessian matrix.

2. When the stiffness properties become constant so is elastic wave speed, but using smaller time steps can also make the distance, in which waves travel, become smaller. As a result, it can leave more elements unaffected by the elastic wave.

But when we consider the time steps from a different perspective, it can also give us another interpretation. It is easy to see that the smaller the time step is, then the difference of $\mathbf{x}$ between each of Newton iterations would become smaller as well. The closer $\mathbf{x}$ are, the change of the Hessian matrix would become smaller as well, which makes reusing the local Hessian from the previous Newton iterations more reasonable.

5.3 Uniform and Non-uniform Grids

Now we do a simple free fall simulation without deformation. And the initial velocity is zero. Then Equation 3.5 would become

$$\mathbf{h}(\mathbf{x}) = \mathbf{M}(\frac{\mathbf{x} - \hat{\mathbf{x}} - \mathbf{g}\Delta t^2}{\Delta t^2}).$$

Because no deformation happens during the simulation, so all the components in the vector $\mathbf{x} - \hat{\mathbf{x}} - \mathbf{g}\Delta t^2$ would become the same. As a result, the distribution of the gradient $\mathbf{h}(\mathbf{x})$ would entirely depend on the mass matrix $\mathbf{M}$. If the mesh object has the same density everywhere, then the mass matrix would rely on the distribution of the volume of each element.

In the meanwhile, when we are doing simulations with deformation, the elastic energy for a single element is $E_{\text{element},i} = V_i\Psi(\mathbf{F}_i)$, where V_i is the volume of the element. As a result, $\nabla E_{\text{element},i} = V_i\nabla\Psi(\mathbf{F}_i)$. The larger the volume of the element, the more likely it has a larger gradient contribution. So the distribution of the volume of each element can have effects on the elastic gradient as well.

We have shown that the volume distribution can affect the distribution of gradient vector components. To continue discussing, here we introduce the concept about uniform and non-uniform mesh. A uniform mesh means that all the elements are congruent. And all vertices, except for the boundary vertices, are connected with the same number of elements. All the other mesh objects are non-uniform mesh. In Figure 5.4 we show an example of an uniform and a non-uniform mesh.

(a) Uniform mesh object **(b)** Non-uniform mesh object

Figure 5.4: An example of uniform and non-uniform mesh object.

For the uniform mesh, the mass matrix almost has the same value on its diagonal. The word almost here means that the boundary vertices have smaller mass, because they are connected with fewer elements. But when the mesh becomes larger, the ratio of boundary vertices will become smaller. In the free fall case, the gradient vector almost has the same value for all of its component, then almost all components are larger than $\epsilon \cdot ||\nabla E||_\infty$, then almost all vertices will become active

vertices, and our algorithm will almost fallback to the vanilla Newton's method. But if we use different non-uniform mesh objects, then our method may become more effective depending on the mass distribution and parameters.

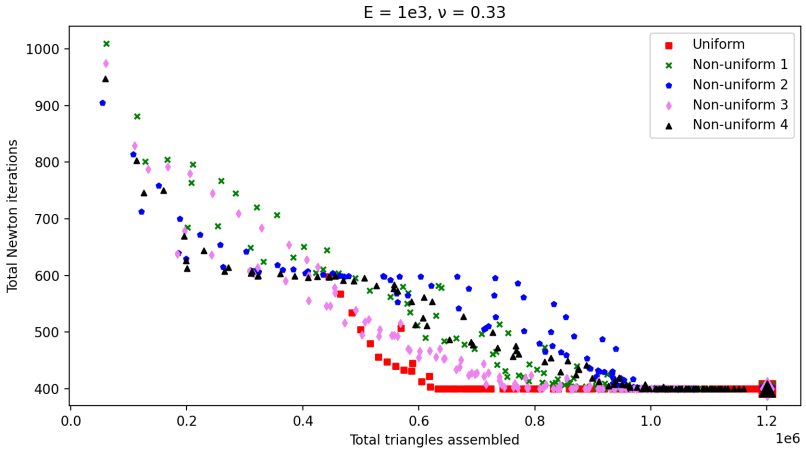
Though non-uniform mesh can perform better with our method in the free fall case. But we think this is accidental. So we ran the Figure 5.1 simulation using both uniform and non-uniform mesh. In the experiments, the non-uniform meshes are generated similar to Figure 5.4. We first generate random $n-1$ random numbers in the range $[0, 1]$. Here n is the number of columns. Then add 0 and 1 into the list and multiply the list with the width of the big rectangular. We use these random numbers as the coordinates for each column. Then we generate row coordinates in the same way. For each small rectangular, we create two triangles inside it by connecting the diagonal vertices. Then we try different random seeds to generate different non-uniform mesh.

In this experiment, we run a simulation that compared uniform mesh objects and non-uniform mesh objects with different random seeds. We use the parameters same as Table 5.1. Then test two pairs physical properties, one with $E_{\text{young}} = 10^3$ GPa, $\nu = 0.33$ and another one with $E_{\text{young}} = 10^4$ GPa, $\nu = 0.33$. The simulation results can be found at Figure 5.5. It's easy to see that in our experiments uniform mesh outperforms non-uniform mesh in the most cases. Except for the non-uniform case 4 when using $E_{\text{young}} = 10^4$ GPa, $\nu = 0.33$, it outperforms the uniform case. A detailed comparison can be found at Table 5.4.

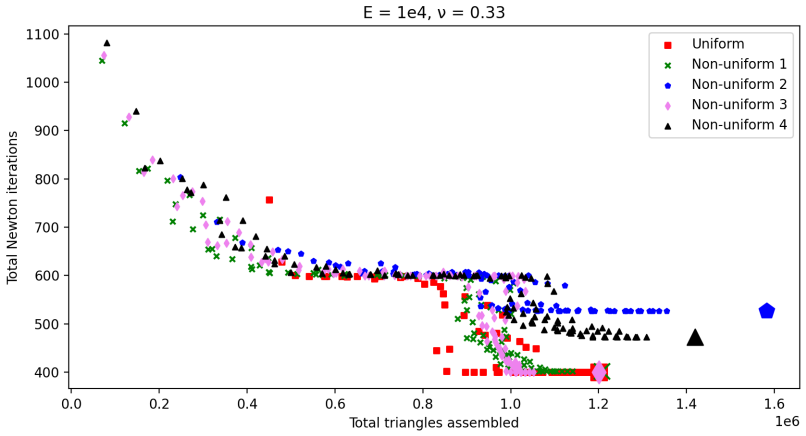
Table 5.4: The results of Figure 5.5 using uniform and non-uniform mesh.

Stiffness Properties	Label	Optimal ϵ and k	Elements Assembled
$E_{\text{young}} = 10^3$ GPa $\nu = 0.33$	uniform	$\epsilon = 2^{-2}, k = 4$	52.73%
	non-uniform 1	$\epsilon = 2^{-7}, k = 4$	72.59%
	non-uniform 2	$\epsilon = 2^{-8}, k = 4$	80.00%
	non-uniform 3	$\epsilon = 2^{-5}, k = 8$	62.22%
	non-uniform 4	$\epsilon = 2^{-7}, k = 3$	80.33%
$E_{\text{young}} = 10^4$ GPa $\nu = 0.33$	uniform	$\epsilon = 2^{-4}, k = 3$	74.68%
	non-uniform 1	$\epsilon = 2^{-10}, k = 5$	85.37%
	non-uniform 2	$\epsilon = 2^{-11}, k = 1$	67.01%
	non-uniform 3	$\epsilon = 2^{-9}, k = 7$	83.18%
	non-uniform 4	$\epsilon = 2^{-11}, k = 4$	83.52%

Right now we are unable to give a clear interpretation of how our algorithm performs on the different uniformity of the mesh objects. But we have clearly shown that uniformity does affect the performance of our algorithm. From the



(a) Beam simulations with uniform and non-uniform mesh using $E_{\text{young}} = 10^3\text{GPa}$ $\nu = 0.33$.



(b) Beam simulations with uniform and non-uniform mesh using $E_{\text{young}} = 10^4\text{GPa}$ $\nu = 0.33$.

Figure 5.5: Beam simulations using uniform and non-uniform mesh.

experiment results, our preliminary result is that uniform mesh should perform better in general.

5.4 Different Resolutions

Another perspective we have noticed is using our method with different resolutions of mesh objects. We use elements to approximate the real-world continuum materials. So if the resolution is high enough, then the elements can approximate the material accurately enough. Then our method may give a more accurate

classification between active and inactive elements, which means in this case, the resolution shouldn't have a significant effect on our method

However, it comes up with two important problems. First, we don't know what is high enough resolution. Second, when the resolution is too high, it will dramatically increase the running time and memory usage, which is very unrealistic to use in practice. So we have to use the resolution under certain constraints. So in this section, we are going to discuss the effect of different resolutions in general cases.

But due to complicated nonlinearity in Equation 3.9 and many factors in Newton's method, we are unable to analyze this problem any further from an analytical perspective. So we tried many different resolutions in experiments. The results of our experiments show that we also can't find a clear connection between the resolution and performance of our method.

Table 5.5: The results of Figure 5.6 using different resolutions.

Resolutions	square size	Optimal ϵ and k	Elements Assembled
10×40	0.25 m	$\epsilon = 2^{-7}, k = 0$	76.07%
15×60	0.1666 m	$\epsilon = 2^{-8}, k = 0$	94.23%
20×80	0.125 m	$\epsilon = 2^{-4}, k = 2$	70.31%
25×100	0.1 m	$\epsilon = 2^{-5}, k = 2$	65.05%
30×120	0.0833 m	$\epsilon = 2^{-5}, k = 3$	60.08%
40×160	0.0625 m	$\epsilon = 2^{-8}, k = 3$	55.70%

We ran the simulations using the same parameters as Table 5.1, except for the resolutions. Also we use the stiffness properties $E_{\text{young}} = 10^4$ GPa and $\nu = 0.33$. We ran six different simulations using resolutions 10×40, 15×60, 20×80, 25×100, 30×120, and 40×160. The simulation results can be found at Figure 5.6. Note that, the x-axis does not represent the total triangles assembled as before. Because when resolution changes, the number of triangles assembled will change naturally. So we use the number of triangles assembled divided by the number of triangles assembled in vanilla Newton's method, so the value will lie in the range of $(0, 1]$. And the comparison can be found at Table 5.5.

From Table 5.5, it seems that the higher the resolution, our method performs better. But we think that the outlier 15×60 is because of the unpredictable nonlinearity. So we can't give an accurate explanation yet. Nor can we find the connection between resolution and optimal ϵ and k.

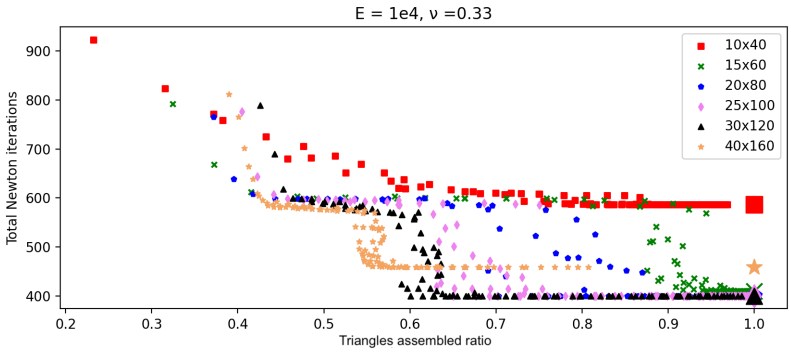


Figure 5.6: Beam simulations with different resolution.

5.5 No Hessian Assemble for the First Newton Iteration

We have talked about the free fall case in Section 5.3, in which case our method will fall back to vanilla Newton's method if using uniform mesh. In this section, we will talk about another idea to handle this case.

It's clear to us that in this free fall case, the Hessian matrix will always remain constant, as long as there are no additional forces. So there is no need to assemble any of the local elastic Hessian matrices. Also, we have noticed that when using our method to assemble only part of elements, we always assemble the most number of elements in the first Newton iteration, as shown in Figure 5.7. The simulation is using the parameters from Table 5.1, and the stiffness properties are $E_{\text{young}} = 10^4$ GPa and $\nu = 0.33$. And we use the optimal parameters $\epsilon = 2^{-4}$ and $k = 2$, as shown in Table 5.3. In Figure 5.7, the x-axis is the index of each Newton iteration, and the y-axis is the number of elements assembled in this Newton iteration. We show the first 300 Newton iterations due to the constraint of the page size. The red color means the first Newton iteration in its frame, and the blue color means the second and further Newton iterations in its frame. So we can see that the first Newton iteration always assembles the most number of elements.

So here we come up with another idea, in the first Newton iteration of each frame, we don't assemble any elements and reuse the global Hessian matrix from the last Newton iteration of the previous frame. It can significantly reduce the number of elements assembled and works perfectly well in the free fall case, but it also may increase the number of Newton iterations to converge. For brevity, we will refer to this new idea as the skip version of our method.

We ran many experiments to test how our method performs using the skip version, and compare it with the non-skip version. The results can be found at Figure 5.8. We ran the beam simulations using parameters as Table 5.1. And

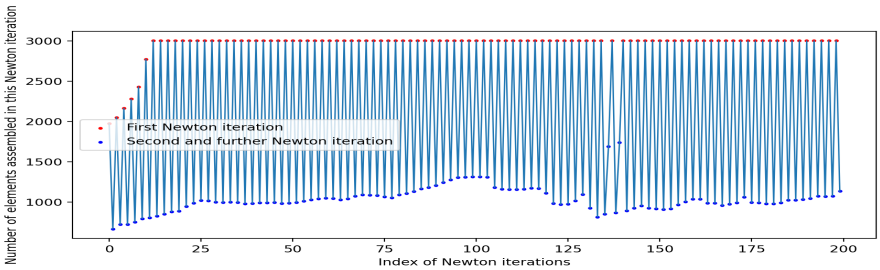


Figure 5.7: Number of elements assembled in the first 300 Newton iterations.

tested it with varying stiffness properties, we tested four cases with $E_{\text{young}} = 10^4, 10^5, 10^6, 10^7, 10^8$ GPa and $\nu = 0.33$.

We found the skip version of our method can work pretty well. Using the skip version can significantly assemble even much fewer elements, but also take a similar number of Newton iterations. Especially when using soft materials, it can take the same number of Newton iterations. It will take slightly more Newton iterations when using stiff materials. The full comparison can be found at Table 5.6. Because when using the skip version, sometimes it takes more Newton iterations than vanilla Newton's method, especially for stiff materials. The EA column for the skip version rows means to compare with vanilla Newton's method, even if it sometimes takes more Newton iterations.

5.5.1 Interpretations

To analyze the reason why it works, we can do a short recap of the initial guess. In Section 4.4 we have compared many options with the initial guess, and then decided to use the position vector from the previous frame $\mathbf{x}^t$ as the initial guess for $t+1$-th frame.

And we think the choice of our initial guess may also be helpful to reduce the negative impact of the skip version. Here we use the same notation $\mathbf{x}_N$, $\mathbf{x}_{N-1}$ to denote the position vector $\mathbf{x}$ of the last Newton iteration and the second last Newton iteration in the previous frame. When using Newton's method, $\mathbf{x}_N$ and $\mathbf{x}_{N-1}$ are generally very close. So is the Hessian matrix at $\mathbf{x}_{N-1}$ and $\mathbf{x}_N$.

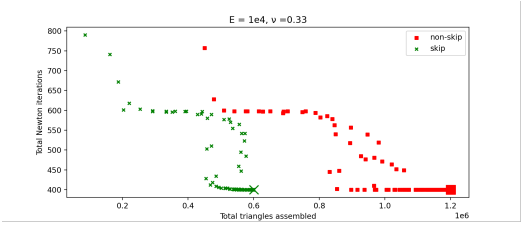
So using the skip version with our initial guess is equivalent to using the Hessian matrix at $\mathbf{x}_{N-1}$ as the Hessian matrix at $\mathbf{x}^{t+1}$, which is the same as $\mathbf{x}_N$. So in general, this replacement can be a reasonable choice.

In addition, we noticed that the skip version also works better when using soft materials. In such case, as we discussed in subsection 5.1.1, the Hessian matrix is dominated by the mass matrix, and the displacement of $\mathbf{x}$ is dominated by the inertia movement and external forces. For example, the skip version works perfectly in the free fall case, which has only inertia movement and gravity. So

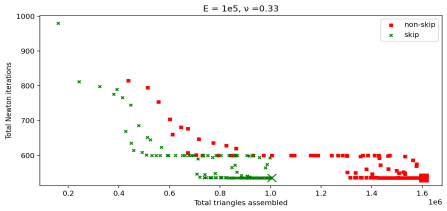
Table 5.6: The results of Figure 5.8 comparing the skip and non-skip version.

Stiffness Properties	Label	ϵ and k	EA	NI
$E_{\text{young}} = 10^4$ GPa	non-skip optimal	$\epsilon = 2^{-4}, k = 3$	74.68%	400
$\nu = 0.33$	skip	$\epsilon = 0, k = 0$	50%	400
	skip optimal	$\epsilon = 2^{-4}, k = 8$	44.96%	400
$E_{\text{young}} = 10^5$ GPa	non-skip optimal	$\epsilon = 2^{-7}, k = 4$	89.08%	535
$\nu = 0.33$	skip	$\epsilon = 0, k = 0$	62.62%	535
	skip optimal	$\epsilon = 2^{-7}, k = 4$	52.11%	535
$E_{\text{young}} = 10^6$ GPa	non-skip optimal	$\epsilon = 2^{-7}, k = 9$	85.02%	649
$\nu = 0.33$	skip	$\epsilon = 0, k = 0$	61.94%	602
	skip optimal	$\epsilon = 2^{-10}, k = 2$	57.11%	602
$E_{\text{young}} = 10^7$ GPa	non-skip optimal	$\epsilon = 2^{-4}, k = 9$	96.02%	567
$\nu = 0.33$	skip	$\epsilon = 0, k = 0$	70.55%	600
	skip optimal	$\epsilon = 2^{-7}, k = 4$	53.77%	600
$E_{\text{young}} = 10^8$ GPa	non-skip optimal	$\epsilon = 2^{-8}, k = 7$	98.49%	565
$\nu = 0.33$	skip	$\epsilon = 0, k = 0$	68.31%	586
	skip optimal	$\epsilon = 2^{-4}, k = 9$	68.00%	586

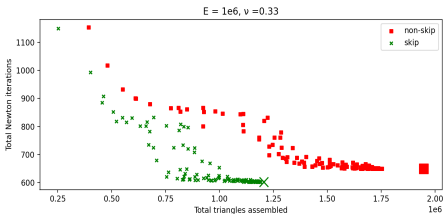
Due to the constraint of page size, EA here means the percentage of elements assembled, which is the same as the Elements Assembled column in the previous tables. NI here means the number of Newton iterations.



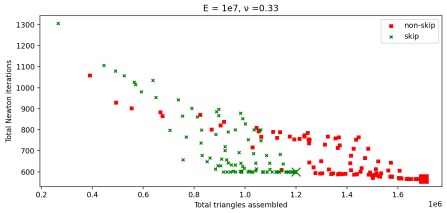
(a) Beam simulations with skip and non-skip method using $E_{\text{young}} = 10^4$GPa $\nu = 0.33$.



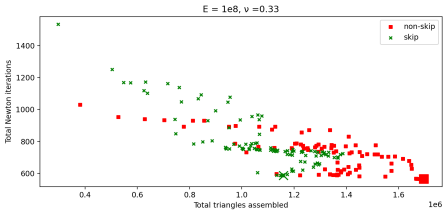
(b) Beam simulations with skip and non-skip method using $E_{\text{young}} = 10^5$GPa $\nu = 0.33$.



(c) Beam simulations with skip and non-skip method using $E_{\text{young}} = 10^6$GPa $\nu = 0.33$.



(d) Beam simulations with skip and non-skip method using $E_{\text{young}} = 10^7$GPa $\nu = 0.33$.



(e) Beam simulations with skip and non-skip method using $E_{\text{young}} = 10^8$GPa $\nu = 0.33$.

Figure 5.8: Beam simulations using skip and non-skip method.

this can be another perspective to interpret the skip version method. But when using stiff materials, the mass matrix is no longer the dominate part of the global

Hessian matrix. So using skip version may be a bad approximation of the correct Hessian matrix and takes more Newton iterations than vanilla Newton's method, which is consistent with our experiment results.

5.6 Projected Newton

In the previous sections, we have always used Newton's method. As we mentioned in Section 3.6, sometimes Newton's method may fail, and Projected Newton's (PN) method can be considered as an alternative. In this section, we will discuss using our method with the PN method.

There are many options to modify the matrix so it can be a positive semi-definite (PSD) matrix. Here we give two commonly used options. The first one is to truncate the negative eigenvalues by replacing them with zero. The second one is to flip the sign of negative eigenvalues.

We can modify the local elastic Hessian matrix directly. But there is another perspective to modifying the matrix, we will refer to it as the material model matrix in this thesis. From Equation 3.9, we can write the local elastic energy as

$$E_{\text{elastic},i} = V_i \Psi(\tilde{\mathbf{f}}_i(\mathbf{x}_i)),$$

where $\tilde{\mathbf{f}}_i$ is to flatten the deformation gradient $\mathbf{F}_i$ into a vector. For example,

$$\mathbf{F}_i = \begin{pmatrix} F_1 & F_4 & F_7 \\ F_2 & F_5 & F_8 \\ F_3 & F_6 & F_9 \end{pmatrix} \quad \tilde{\mathbf{f}}_i = \begin{pmatrix} F_1 & F_2 & F_3 & F_4 & F_5 & F_6 & F_7 & F_8 & F_9 \end{pmatrix}^T.$$

The second derivative can be written as

$$\mathbf{H}_{\text{elastic},i} = V_i \left(\frac{\partial \tilde{\mathbf{f}}_i}{\partial \mathbf{x}_i}\right)^T \frac{\partial^2 \Psi}{\partial \tilde{\mathbf{f}}_i^2} \frac{\partial \tilde{\mathbf{f}}_i}{\partial \mathbf{x}_i} + \frac{\partial \Psi}{\partial \tilde{\mathbf{f}}_i} \frac{\partial^2 \tilde{\mathbf{f}}_i}{\partial \mathbf{x}_i^2}).$$

It's worth mentioning that $\frac{\partial^2 \Psi}{\partial \tilde{\mathbf{f}}_i^2}$ is a $d^2 \times d^2$ matrix and $\frac{\partial^2 \tilde{\mathbf{f}}_i}{\partial \mathbf{x}_i^2}$ is a third order tensor. But because $\tilde{\mathbf{f}}_i$ is a linear function of $\mathbf{x}_i$, so $\frac{\partial^2 \tilde{\mathbf{f}}_i}{\partial \mathbf{x}_i^2}$ is a zero tensor. So the Hessian matrix is

$$\mathbf{H}_{\text{elastic},i} = V_i \left(\frac{\partial \tilde{\mathbf{f}}_i}{\partial \mathbf{x}_i}\right)^T \frac{\partial^2 \Psi}{\partial \tilde{\mathbf{f}}_i^2} \frac{\partial \tilde{\mathbf{f}}_i}{\partial \mathbf{x}_i}).$$

If we do the Hessian modification on $\frac{\partial^2 \Psi}{\partial \tilde{\mathbf{f}}_i^2}$, then $\mathbf{H}_{\text{elastic},i}$ can also be guaranteed to be a PSD matrix.

In this thesis, we are going to use the option that truncates the negative eigenvalues of the local elastic Hessian matrix $\mathbf{H}_{\text{elastic},i}$ directly. Because the PN method

itself is not the purpose of this thesis. So we here only give a short discussion of the reason behind our choice.

First, when doing the matrix modifications, usually the modification on the outmost level can give the matrix which is the closest to the unmodified matrix. For example, if we directly modify the global Hessian matrix, then it's very likely to give a PSD matrix which is closest to the global Hessian matrix. But this is impossible in practice because the global Hessian matrix is usually large. In the same way, direct modifying the local Hessian matrix can be closer to the local Hessian, than modifying the material model matrix. But the modification to the material model matrix has other advantages. For example, the eigendecompositions can be computed analytically [SGK19].

Theorem 5.6.1. *Let the symmetric matrix $A \in \mathbb{R}^{n\times n}$ have the spectral decomposition $A = Q\mathrm{diag}(\lambda_i)Q^T$ and let $\delta \geq 0$. The unique matrix with smallest eigenvalue at least δ nearest to A in the Frobenius norm is given by*

$$X = Q\mathrm{diag}(\tau_i)Q^T, \quad \tau_i = \begin{cases} \lambda_i, & \lambda_i \geq \delta \\ \delta, & \lambda_i < \delta. \end{cases}$$

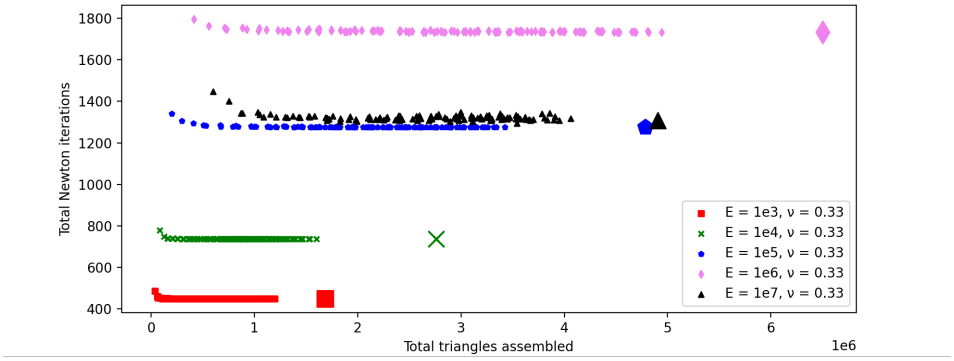
Second, from the Theorem 5.6.1, we can see that after truncating the negative eigenvalues, the matrix is a PSD matrix and the closest matrix to the original matrix in terms of the Frobenius norm [CH98].

Based on the arguments above, our choice can modify the elastic Hessian matrix to a PSD matrix, while trying to be as close to the original matrix as possible. And our experiment results also show that our choice takes the least number of Newton iterations to converge.

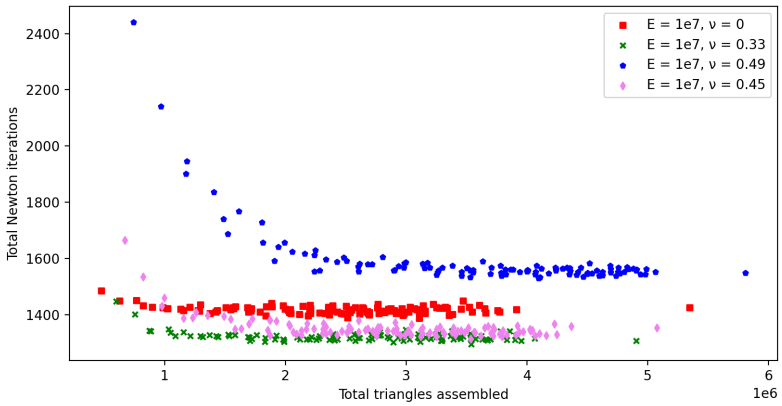
Table 5.7: Other parameters used in the simulation of Figure 1.1.

Parameter	Value	Unit	Description
ρ	10^3	kg/m^2	Density of mesh object
r	10	m	Radius of the circle
$\mathrm{n_{ele}}$	3750		Number of elements
$\mathrm{n_{ver}}$	1952		Number of vertices
Δt	0.01	s	Time steps for one frame
frame	200		Number of frames simulated
g	0	m/s^2	Gravity constant
τ	0.01		Stop criteria for Newton's method
c	0.9		Line search parameter
κ	10^{-4}		Line search parameter

Newton's method works well doing the simulations in Figure 5.1, so we test the PN method with simulations in Figure 1.1, which we found that Newton's method failed. The simulation is that we compress the left half of the circle, then let it return to the rest configuration.



(a) Beam simulations with PN method using $E_{\text{young}} = 10^3$GPa $\nu = 0.33$.



(b) Beam simulations with PN method using $E_{\text{young}} = 10^4$GPa $\nu = 0.33$.

Figure 5.9: Beam simulations using PN method.

The experiment parameters can be found at Table 5.7. And we tested our method with varying stiffness properties. Similar to the Section 5.1, we set up two experiments. The first one we tested varying E_{young}, here we used $E_{\text{young}} = 10^3, 10^4, 10^5, 10^6, 10^7$ GPa and $\nu = 0.33$. The second one we tested varying ν. We used $E_{\text{young}} = 10^7$ GPa, and $\nu = 0, 0.33, 0.45, 0.49$. The experiment results can be found at Figure 5.9. In Figure 5.9a, we show the results of varying E_{young}, and in Figure 5.9b, we show the results of varying ν. From the figures, we can see that our method can work with the PN method as well. Similar to the results in

Section 5.1, we also find that, in general, our method can assemble fewer elements when using softer materials. The full comparsion can be found at Table 5.8.

Table 5.8: The results of Figure 5.9 using different stiffness properties.

Stiffness Properties	Optimal ϵ and k	Elements Assembled
E_{young}= 10^3 GPa, ν = 0.33	$\epsilon = 2^{-3}, k = 2$	12.43%
E_{young}= 10^4 GPa, ν = 0.33	$\epsilon = 2^{-3}, k = 0$	11.45%
E_{young}= 10^5 GPa, ν = 0.33	$\epsilon = 2^{-2}, k = 4$	24.14%
E_{young}= 10^6 GPa, ν = 0.33	$\epsilon = 2^{-7}, k = 1$	38.36%
E_{young}= 10^7 GPa, ν = 0.33	$\epsilon = 2^{-4}, k = 1$	34.75%
E_{young}= 10^7 GPa, ν = 0.0	$\epsilon = 2^{-2}, k = 1$	18.41%
E_{young}= 10^7 GPa, ν = 0.45	$\epsilon = 2^{-3}, k = 1$	31.20%
E_{young}= 10^7 GPa, ν = 0.49	$\epsilon = 2^{-7}, k = 1$	56.00%

5.6.1 Interpretations

Though our method can work with the PN method, and we have a similar conclusion to vanilla Newton's method. And we think the arguments from subsection 5.1.1 still hold. But there is something else worth mentioning here.

So, suppose we have a simulation with very small time steps but in a hugely deformed shape. Because the time steps are small, the displacement between each frame is small too, which means at each frame, the initial guess can be very close to the minimum. As a result, the global Hessian matrix is very likely to be a positive definite matrix.

Then we look at the local Hessian matrix of each element. The elastic energy can only be minimized when the object is in its rest shape. It means that only when the position vector is very close to a rigid body motions of the rest configuration, the local elastic Hessian matrix can be guaranteed to be a PSD matrix. When the object is in a hugely deformed shape, the local Hessian matrix has a huge chance to be a non-PSD matrix.

So when using the PN method in this case. The global Hessian matrix is very likely to be a positive definite matrix, but the local Hessian matrix is very likely to be a non-PSD matrix. After using the PN method to modify the local Hessian matrix, the assembled global Hessian matrix can be very far from the correct global Hessian matrix. So it may take much more iterations to converge to the minimum than vanilla Newton's method.

In this case, we found our method can still be effective, but we are not sure about the real reason behind it. It can also be the reason that the PN method itself performs too badly, so the negative impact of our method is negligible.

Chapter 6

Conclusion and Outlook

6.1 Conclusion

This thesis starts with a problem that assembling the global Hessian matrix is computationally expensive in FEM simulations. In chapter 4, we introduced our method using the concept of active vertices and active elements, and assembling the local Hessian matrices of the active elements only. In chapter 5, we showed how our method performs with experiment results. When we look back at chapter 1 we listed several questions to guide our research. We will provide answers to them here.

1. How can we separate the active and inactive vertices, and correspondingly active and inactive elements?

   In chapter 4, we first identify the active vertices based on the gradient information, mainly the gradient component larger than a threshold. In addition, we also consider their neighbors as active vertices. Then we identify active elements if they have at least one active vertex. Furthermore, we noticed that our method may fall back to Newton's method in free fall cases. So in Section 5.5, we introduced an adaption to not assemble any elements in the first Newton iteration of each frame.

2. How does this method compare with vanilla Newton's method?

   In chapter 5, we found that our method is effective, particularly using soft materials or small time steps. It can assemble fewer elements but take the same number of Newton iterations as vanilla Newton's method. When using stiff materials and large time steps, our method will fall back to vanilla Newton's method. In addition, we have shown that the uniformity of the mesh object affects the performance of our method, our preliminary result

is that uniform mesh can perform better in general, which means it can assemble fewer elements than non-uniform mesh while keeping the same number of Newton iterations. At last, we tried our method with different resolutions of the mesh object, but we are unable to find the relations here.

3. What are the limitations and possible improvements of this method?

   First, we found that sometimes our method will fall back to vanilla Newton's method in the free fall case. We introduced a small modification of our algorithm in Section 5.5, by skipping the assembly process of the first Newton iteration of each frame. And we found this modification can significantly reduce the number of elements assembled, but it can take slightly more Newton iterations. Second, sometimes Newton's method may fail, people will use PN method as an alternative. In Section 5.6, we tried our method with the PN method, and our result is that our method can also assemble fewer elements and keep the same number of Newton iterations as the PN method. Similarly, our method performs better when using soft materials.

To sum up this thesis, we have clearly shown the potential to assemble part of the local Hessian matrices, while keeping the same number of Newton iterations to converge to the minimum.

6.2 Outlook

However, Several questions are remaining in this thesis. We think it may be worthwhile investigating them in the future.

First, in chapter 4, we introduced our method to identify the active vertices and active elements. This method is based on our previous observations and our guesses. At the moment, we are unable to find the most accurate way to identify them. Though we have shown the potential to assemble only part of the elements, we think it may exist a better way to draw the boundary between active and inactive part of the mesh object.

Here we show another possible way. As we have shown in Section 3.3, when using the second order Taylor approximation, the value decreased approximates to $-\frac{1}{2}\nabla E^T \mathbf{H}^{-1} \nabla E$. If we write it as $\nabla E^T \mathbf{H}^{-1} \nabla E = \mathbf{p}^T \mathbf{p}$, where $\mathbf{p} = \mathbf{L}\nabla E$ and $\mathbf{L}^T \mathbf{L} = \mathbf{H}^{-1}$. Then the components of the vector $\mathbf{p}$, instead of the vector ∇E, can be another perspective to separate the active and inactive vertices. Also the choice of the matrix $\mathbf{L}$ can be another factor to consider.

Second, there are several other perspectives to consider to apply our method, which we didn't address in this thesis. For example, in reality, damping force

happens everywhere, and it usually slows down the movement of the object. So when deformation is spread inside the object, even if it's a stiff material, it can be slowed down by the damping force. It's possible that the rest part of the body is not affected by the deformation because of the damping force. And this can be a good example to apply our method.

Third, we have only considered the inertia part, elastic energy, and gravity. But in reality, many external forces can happen, which we didn't consider. For example, collisions are common physical phenomena. They can usually be addressed as additional constraints to the minimization problem. There are many ways to solve the constraint minimization problem. For example, the penalty method is commonly used. However, it will change the sparsity pattern of the Hessian matrix and introduce additional terms to the Equation 3.13. Normally, the collisions can introduce new deformation. We think extending our method to a general physical simulation can be another promising direction.

Appendix A

Deformation Gradient

In this chapter, we show how to compute the deformation gradient of a tetrahedron. Given an undeformed three-dimensional tetrahedron T_{ABCD}, with vertices coordinates $(X_1, Y_1, Z_1), (X_2, Y_2, Z_2), (X_3, Y_3, Z_3), (X_4, Y_4, Z_4)$, after deformation, we can have tetrahedron $T_{A'B'C'D'}$ with vertices coordinates $(x_1,\ y_1,\ z_1)$, $(x_2,\ y_2,z_2)$, $(x_3,\ y_3,\ z_3)$, $(x_4,\ y_4,\ z_4)$, as shown in figure Figure 3.2.

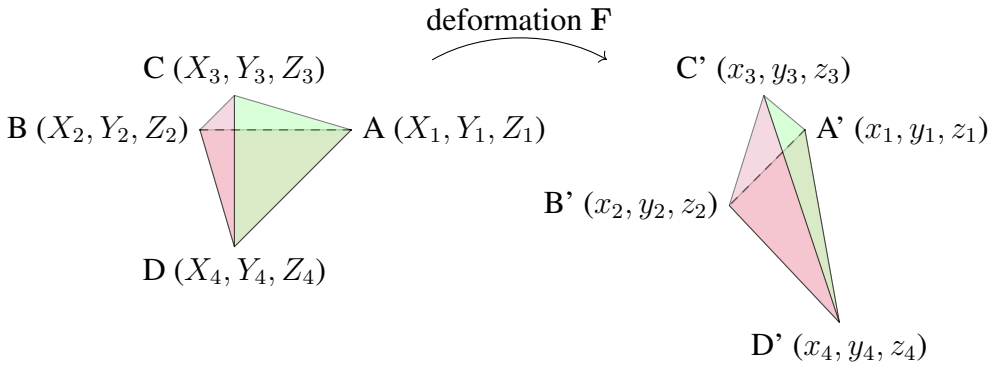


Figure A.1: An Example of Tetrahedron Deformation.

Now we can calculate

$$\begin{aligned} \mathbf{D}_m &= \begin{pmatrix} X_1 - X_4 & X_2 - X_4 & X_3 - X_4 \\ Y_1 - Y_4 & Y_2 - Y_4 & Y_3 - Y_4 \\ Z_1 - Z_4 & Z_2 - Z_4 & Z_3 - Z_4 \end{pmatrix} \\ \mathbf{D}_s &= \begin{pmatrix} x_1 - x_4 & x_2 - x_4 & x_3 - x_4 \\ y_1 - y_4 & y_2 - y_4 & y_3 - y_4 \\ z_1 - z_4 & z_2 - z_4 & z_3 - z_4 \end{pmatrix}. \end{aligned} \tag{A.1}$$

The deformation gradient F and the Green Strain are defined as

$$\mathbf{F} = \mathbf{D}_s \mathbf{D}_m^{-1} \text{ and } \mathbf{E}_{\text{green strain}} = \frac{1}{2}(\mathbf{F}^T\mathbf{F} - \mathbf{I}), \tag{A.2}$$

where $\mathbf{I}$ is the identity matrix. And $\mathbf{D}_m^{-1}$ is a constant matrix, and can be pre-computed.

Similarly, we can use the deformation gradient to compute the elastic energy for the tetrahedron element. For example, using Neohookean's model in Equation 3.9. The elastic energy for the element is

$$\begin{aligned} E_{\text{elastic},i} &= V_i \Psi(\mathbf{F}_i) \\ &= V_i(\frac{\mu}{2} \cdot (\text{tr}(\mathbf{F}_i^T \mathbf{F}_i) - 3 - 2\ln(\det(\mathbf{F}_i))) + \frac{\lambda}{2} \cdot (\det(\mathbf{F}_i) - 1)), \end{aligned}$$

where V_i is the volume of the tetrahedron, μ and λ are the Lamé coefficients as mentioned in Section 3.4.

Appendix B

Automatic Differentiation

In Section 3.6, we need to compute the local gradient and the Hessian matrix. Among many methods, Automatic Differentiation is a very useful technique to evaluate the derivatives of a function specified by a computer program. It can help us to compute the gradient and the Hessian of the element energy function. In particular, we use forward mode automatic differentiation in this thesis. To illustrate its usage, the code we are using here follows the Rust syntax, also we use the nalgebra, a linear algebra library, as the external dependency [Cro19].

First, we need to define a struct called *Dual*, shown in Figure B.1. The struct contains two fields, *value* and *gradient*. If we use it represent the result of a function $y = f(x)$, then *value* would mean the value y, and *gradient* would mean the gradient vector ∇y.

```
pub struct Dual<const N: usize> {
    value: f64,
    gradient: SVector<f64, N>,
}
```

Figure B.1: Code Example of *Dual*.

Then we could overload its basic operators, based on the differentiation rules and chain rule. Here we give an example how to implement the multiplication of Dual<N> in Figure B.2. Recall that $\nabla(\mathbf{f} \cdot \mathbf{g}) = \nabla\mathbf{f} \cdot \mathbf{g} + \nabla\mathbf{g} \cdot \mathbf{f}$, where $\mathbf{f}$ and $\mathbf{g}$ are both $\mathbb{R}^n \to \mathbb{R}$ real-valued vector function.

With the help of automatic differentiation, now we can easily compute the gradient and the Hessian of a local elastic energy function. Here we use a triangle as an example. We first initialize six Dual<6> objects. Assign the *value* of each object by the value of $x_1, y_1, x_2, y_2, x_3, y_3$, and the *gradient* of each object by the standard unit vectors $\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3, \mathbf{e}_4, \mathbf{e}_5, \mathbf{e}_6$. Then we simply use these objects,

```
impl<const N: usize> Mul<Self> for Dual<N> {
  type Output = Self;
  fn mul(self, rhs: Self) -> Self {
    Dual {
      value: self.value * rhs.value,
      gradient: self.gradient * rhs.value
              + rhs.gradient * self.value
    }
  }
}
```

Figure B.2: Code Example of Operator Overloading for Struct Dual.

rather than naive float variables, to compute the local elastic energy, the *gradient* field of the final result would be the local gradient vector. Using tetrahedron elements, would require twelve Dual<12> objects, and these objects are initialized correspondingly.

Similarly, we can show how to compute the Hessian matrix of a function. We define a new struct called *Triple*, shown in Figure B.3. The struct contains three fields, *value*, *gradient* and *hessian*. If we use it represent the result of a function $y = f(x)$, then *value* would mean the value y, *gradient* would mean the gradient vector ∇y, and *hessian* would mean the Hessian matrix $\mathbf{H}$

```
pub struct Triple<const N: usize> {
  value: f64,
  gradient: SVector<f64, N>,
  hessian: SMatrix<f64, N, N>,
}
```

Figure B.3: Code Example of *Triple*.

We can overload its basic operators based on the differentiation rules and chain. The product rule for the Hessian matrix is

$$\begin{aligned}\mathbf{H}_{f(\mathbf{x})\cdot g(\mathbf{x})}(\mathbf{x}) = f(\mathbf{x})\mathbf{H}_{g(\mathbf{x})}(\mathbf{x}) + g(\mathbf{x})\mathbf{H}_{f(\mathbf{x})}(\mathbf{x}) \\ + \nabla f(\mathbf{x})\nabla g(\mathbf{x})^T + \nabla g(\mathbf{x})\nabla f(\mathbf{x})^T,\end{aligned}$$

where f and g are two $\mathbb{R}^n \to \mathbb{R}$ real-valued vector function, and we assume that the gradient is a column vector. We give an example how to implement Triple <N>∗Triple<N> in Figure B.4.

```
impl<const N: usize> Mul<Self> for Triple<N> {
  type Output = Self;
  fn mul(self, rhs: Self) -> Self {
    Dual {
      value: self.value * rhs.value,
      gradient: self.gradient * rhs.value
              + rhs.gradient * self.value,
      hessian: self.hessian * rhs.value
      + rhs.hessian * self.value
      + self.gradient * rhs.gradient.transpose()
      + rhs.gradient * self.gradient.transpose(),
    }
  }
}
```

Figure B.4: Code Example of Operator Overloading for Struct Triple.

To compute the local elastic Hessian matrix, we first initialize six Triple<6> objects. Assign the *value* of each object by the value of $x_1, y_1, x_2, y_2, x_3, y_3$, the *gradient* of each object by the standard unit vectors $\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3, \mathbf{e}_4, \mathbf{e}_5, \mathbf{e}_6$, and the *hessian* of each object by a zero matrix. Then we use these objects to compute the local elastic energy, the *hessian* field of the final result would be the local Hessian matrix. In 3d cases, it would require twelve Triple<12> objects, and these objects are initialized correspondingly.

Bibliography

[BW98] David Baraff and Andrew Witkin. “Large Steps in Cloth Simulation”. In: *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’98. New York, NY, USA: Association for Computing Machinery, 1998, pp. 43–54. ISBN: 0897919998 (cited on page 3).

[BK15] Jan Bender and Dan Koschier. “Divergence-Free Smoothed Particle Hydrodynamics”. In: *Proceedings of the 14th ACM SIGGRAPH / Eurographics Symposium on Computer Animation*. SCA ’15. Los Angeles, California: Association for Computing Machinery, 2015, pp. 147–155. ISBN: 9781450334969 (cited on page 3).

[CH98] Sheung Hun Cheng and Nicholas J. Higham. “A Modified Cholesky Algorithm Based on a Symmetric Indefinite Factorization”. In: *SIAM Journal on Matrix Analysis and Applications* 19.4 (1998), pp. 1097–1110 (cited on page 36).

[Com22] Wikimedia Commons. *File:Finite element triangulation.svg — Wikimedia Commons, the free media repository*. [Online; accessed 5-September-2022]. 2022. URL: https://commons.wikimedia.org/w/index.php?title=File:Finite_element_triangulation.svg&oldid=664669905 (cited on page 10).

[Cro19] Sébastien Crozet. *nalgebra: a linear algebra library for Rust*. https://nalgebra.org/. 2019 (cited on page 45).

[GSS+15] Theodore F. Gast, Craig Schroeder, Alexey Stomakhin, Chenfanfu Jiang, and Joseph M. Teran. “Optimization Integrator for Large Time Steps”. In: *IEEE Transactions on Visualization and Computer Graphics* 21.10 (Oct. 2015), pp. 1103–1115. ISSN: 1941-0506 (cited on pages 3, 6, 11, 17).

[HFS+01] G. Hirota, S. Fisher, A. State, C. Lee, and H. Fuchs. “An implicit finite element method for elastic solids in contact”. In: *Proceedings Computer Animation 2001. Fourteenth Conference on Computer Animation (Cat. No.01TH8596)*. 2001, pp. 136–254 (cited on page 3).

[JP99] Doug L James and Dinesh K Pai. “Artdefo: accurate real time deformable objects”. In: *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*. 1999, pp. 65–72 (cited on page 4).

[JST+16] Chenfanfu Jiang, Craig Schroeder, Joseph Teran, Alexey Stomakhin, and Andrew Selle. “The Material Point Method for Simulating Continuum Materials”. In: *ACM SIGGRAPH 2016 Courses*. SIGGRAPH ’16. Anaheim, California: Association for Computing Machinery, 2016. ISBN: 9781450342896 (cited on page 3).

[KROM99] C. Kane, E.A. Repetto, M. Ortiz, and J.E. Marsden. “Finite element analysis of nonsmooth contact”. In: *Computer Methods in Applied Mechanics and Engineering* 180.1 (1999), pp. 1–26. ISSN: 0045-7825 (cited on pages 3, 6).

[KP12] Danny M. Kaufman and Dinesh K. Pai. “Geometric Numerical Integration of Inequality Constrained, Nonsmooth Hamiltonian Systems”. In: *SIAM Journal on Scientific Computing* 34.5 (2012), A2670–A2703 (cited on page 3).

[KKB18] T. Kugelstadt, D. Koschier, and J. Bender. “Fast Corotated FEM using Operator Splitting”. In: *Computer Graphics Forum* 37.8 (2018), pp. 149–160 (cited on page 4).

[LGL+19] Minchen Li, Ming Gao, Timothy Langlois, Chenfanfu Jiang, and Danny M. Kaufman. “Decomposed Optimization Time Integrator for Large-Step Elastodynamics”. In: *ACM Trans. Graph.* 38.4 (July 2019). ISSN: 0730-0301 (cited on pages 3, 14).

[LBK17] Tiantian Liu, Sofien Bouaziz, and Ladislav Kavan. “Quasi-Newton Methods for Real-Time Simulation of Hyperelastic Materials”. In: *ACM Trans. Graph.* 36.3 (May 2017). ISSN: 0730-0301 (cited on pages 3, 4).

[LLK+20] Andreas Longva, Fabian Löschner, Tassilo Kugelstadt, José Antonio Fernández-Fernández, and Jan Bender. “Higher-Order Finite Elements for Embedded Simulation”. In: *ACM Trans. Graph.* 39.6 (Nov. 2020). ISSN: 0730-0301 (cited on page 4).

[LLJ+20] Fabian Löschner, Andreas Longva, Stefan Jeske, Tassilo Kugelstadt, and Jan Bender. "Higher-Order Time Integration for Deformable Solids". In: *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA '20. Virtual Event, Canada: Eurographics Association, 2020 (cited on page 4).

[MMC16] Miles Macklin, Matthias Müller, and Nuttapong Chentanez. "XPBD: Position-Based Simulation of Compliant Constrained Dynamics". In: *Proceedings of the 9th International Conference on Motion in Games*. MIG '16. Burlingame, California: Association for Computing Machinery, 2016, pp. 49–54. ISBN: 9781450345927 (cited on page 3).

[MCG03] Matthias Müller, David Charypar, and Markus H Gross. "Particle-based fluid simulation for interactive applications." In: *Symposium on Computer animation*. Vol. 2. 2003 (cited on page 3).

[MHHR07] Matthias Müller, Bruno Heidelberger, Marcus Hennix, and John Ratcliff. "Position based dynamics". In: *Journal of Visual Communication and Image Representation* 18.2 (2007), pp. 109–118 (cited on page 3).

[NW06] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. 2e. New York, NY, USA: Springer, 2006 (cited on page 7).

[She19] Peter M. Shearer. *Introduction to Seismology*. 3rd ed. Cambridge University Press, 2019 (cited on page 23).

[SB12] 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*. SIGGRAPH '12. Los Angeles, California: Association for Computing Machinery, 2012. ISBN: 9781450316781 (cited on pages 9, 11).

[SGK18] Breannan Smith, Fernando De Goes, and Theodore Kim. "Stable Neo-Hookean Flesh Simulation". In: *ACM Trans. Graph.* 37.2 (Mar. 2018). ISSN: 0730-0301 (cited on page 4).

[SGK19] Breannan Smith, Fernando De Goes, and Theodore Kim. "Analytic Eigensystems for Isotropic Distortion Energies". In: *ACM Trans. Graph.* 38.1 (Feb. 2019). ISSN: 0730-0301 (cited on page 36).

[TBHF03] J. Teran, S. Blemker, V. Ng Thow Hing, and R. Fedkiw. “Finite Volume Methods for the Simulation of Skeletal Muscle”. In: *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA ’03. San Diego, California: Eurographics Association, 2003, pp. 68–74. ISBN: 1581136595 (cited on page 4).

[TPBF87] Demetri Terzopoulos, John Platt, Alan Barr, and Kurt Fleischer. “Elastically Deformable Models”. In: *SIGGRAPH Comput. Graph.* 21.4 (Aug. 1987), pp. 205–214. ISSN: 0097-8930 (cited on page 3).

[VM01] P. Volino and N. Magnenat-Thalmann. “Comparing efficiency of integration methods for cloth simulation”. In: *Proceedings. Computer Graphics International 2001*. 2001, pp. 265–272 (cited on page 3).

[ZTZ13] O.C. Zienkiewicz, R.L. Taylor, and J.Z. Zhu. “Chapter 16 - Adaptive Finite Element Refinement”. In: *The Finite Element Method: its Basis and Fundamentals (Seventh Edition)*. Ed. by O.C. Zienkiewicz, R.L. Taylor, and J.Z. Zhu. Seventh Edition. Oxford: Butterworth-Heinemann, 2013, pp. 545–572. ISBN: 978-1-85617-633-0 (cited on page 10).