b
UNIVERSITÄT
BERN

Flesh Simulation with Application to Character Animation

Bachelor Thesis

submitted in fulfillment of the requirements for the degree
Bachelor of Science (B.Sc.)

at the

University of Bern
Institute of Computer Science

Examiner: Prof. Dr. David Bommes

Handed in by: Corina Danja Masanti
Matriculation number: 15-128-655
Date of submission: 06.11.2020

Abstract

This thesis explores the process of simulating the fleshy appearances of virtual characters. The focus lies on the formulation of the deformation energy. In order to accurately simulate a fleshy look, nonlinear hyperelastic energies have proven to be the best choice. But simulating volume-preserving biological tissues such as flesh yields more than a few mathematical and physical challenges. Achieving rest stability while working with a high Poisson's ratio can be named as an example. Inspired by the complexity of this field, this thesis serves as an entry-level document. Relevant mathematical and physical aspects are presented and explained in an introductory manner. In addition, this thesis includes some more detailed calculations to bridge the gap between an expert in the field and a beginner. At the end of this thesis, I am presenting a few results visually to conclude the studies.

Acknowledgements

I would like to thank Prof. Dr. Bommes for supervising my thesis and for helping me find this topic. Furthermore, I would like to express a special thanks to Nicolas Gallego-Ortiz for the numerous discussions and advice he gave me during the process. In addition, I would like to thank Breannan Smith for taking the time to answer my questions.

Contents

Chapter 1

Introduction

"Animation offers a medium of storytelling and visual entertainment which can bring pleasure and information to people of all ages everywhere in the world."
- Walt Disney

1.1 Motivation

With steadily increasing computational power, the demand for better-looking results is constantly growing. Especially in the field of animation and simulation, various research has been done to avoid mediocre results. In the entertainment sector, these findings were used to present movies of the highest quality for people all over the world. Animation studios like Pixar© have made groundbreaking progress over the years. This can easily be observed by comparing today's work with that from ten years ago.

As always, there are different requirements for each use case. In some cases, an exaggerated movement or reaction has a better effect on the viewer. A massive explosion, for example, looks more spectacular in a movie when it is not physically correct. In other scenarios, the animation should come as close as possible to reality. For instance, adding small details like visible breathing or wrinkles offers a way to get more convincing results. A good animation should create the illusion of a character with personality, thoughts, and emotions. But in order to create this

illusion, an animated character has to move and physically interact with its environment as it would in reality. Otherwise, the human brain would immediately recognize that some things do not add up. One aspect for achieving this realistic effect is the simulation of the fleshy appearances of virtual characters. In the paper *Stable Neo-Hookean Flesh Simulation* ([SGK18]), the authors addressed exactly this problem and formulated a new deformation energy for simulating human flesh. This thesis is based heavily on this paper. In the following, I will abbreviate the name of the paper with *SNH-FS*.

But before diving further into the content of the paper, a fundamental background is needed. In order to animate a physical movement, a basic understanding of the physics behind it is necessary. This knowledge requires some experience in the field of continuum mechanics. Unfortunately, for most of the students in computer science, it has yet to be learned. This thesis should help students of computer science who are interested in the field of animation and simulation but are still beginners. Therefore, the goal of this thesis is to explain the necessary physical and mathematical background. In addition, I will go deeper into the thematics of the paper *SNH-FS*. I will go through some of their calculations more detailed to help the reader to follow the thought process of the authors. I aim to get an understanding of their contribution to the field and extend the code they provided to test the energy with practical experiments.

1.2 Structure

In the following, I will deliver an introduction into the field of continuum mechanics and give a brief overview of the necessary mathematical background. Next, I will go through the ideas mentioned in the paper SNH-FS and include some calculations and visualizations that serve for a better understanding. Lastly, I will give insights into the implementation of the energy and show the results of the practical experiments I conducted.

Chapter 2

Background

The goal of the paper *SNH-FS* was to simulate the fleshy appearance of virtual characters. In order to follow their thought process, we first need to understand the physical and mathematical interpretation of a deformation. This chapter serves as an introduction into the field of continuum mechanics and describes the mathematical background needed to understand the calculations and conclusions. At the beginning of this chapter, I will define the notation that I will use throughout this whole thesis. Next, I will present some concepts used in continuum mechanics, which are crucial for this field. Thirdly, I will give some insights into the mathematics used in continuum mechanics. However, I will not include each proof explicitly, as there are already many resources for an interested reader. Finally, I will introduce a few calculations and conclusions that will be useful to understand the methods used in the paper *SNH-FS*.

2.1 Notation

This section defines the notation that is used throughout this thesis to avoid misunderstandings. I am using the common notation used in continuum mechanics taken from the book *Continuum Mechanics* ([Spe80]). Additionally, I am including some more specific declarations formulated and used by the authors of the paper *SNH-FS*.

2.1.1 General Notation

Scalars are represented by regular, normal-weight variables such as a, whereas tensors and matrices are represented by upper-case bold letters, such as $\mathbf{A}$. Vectors will be denoted by bold lower-case variables like $\mathbf{a}$.

2.1.2 Tensor Notation

Furthermore, I am using the tensor notation used in *SNH-FS*. Similar to Golub and Van Loan ([GV12]), the authors decided to define the vectorization vec$(\cdot)$ as column-wise flattening of a matrix into a vector:

$$\mathbf{A} = \begin{bmatrix} a & c \\ b & d \end{bmatrix} \qquad \text{vec}(\mathbf{A}) = \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$$

Additionally, 4^{th} order tensors in a form of matrix-of-matrices will be used in the calculations in chapter 3. These matrices are denoted by using blackboard bold:

$$\mathbb{A} = \begin{bmatrix} \begin{bmatrix} a & c \\ b & d \end{bmatrix} & \begin{bmatrix} i & k \\ j & l \end{bmatrix} \\ \begin{bmatrix} e & g \\ f & h \end{bmatrix} & \begin{bmatrix} m & o \\ n & p \end{bmatrix} \end{bmatrix} = \begin{bmatrix} [\mathbf{A}_{00}] & [\mathbf{A}_{01}] \\ [\mathbf{A}_{10}] & [\mathbf{A}_{11}] \end{bmatrix}$$

By vectorizing $\mathbb{A}$ we get the following result:

$$\text{vec}(\mathbb{A}) = \left[\text{vec}\left(\mathbf{A}_{00}\right) \,\middle|\, \text{vec}\left(\mathbf{A}_{10}\right) \,\middle|\, \text{vec}\left(\mathbf{A}_{01}\right) \,\middle|\, \text{vec}\left(\mathbf{A}_{11}\right) \right] = \breve{\mathbf{A}}$$

In order to point out that a matrix is vectorized, I am using the symbol $\breve{}$ above the letter. The term above is equivalent to

$$\breve{\mathbf{A}} = \begin{bmatrix} a & e & i & m \\ b & f & j & n \\ c & g & k & o \\ d & h & l & p \end{bmatrix}.$$

The advantage of this form is that we can write several expressions as a cross product. This property will be used in chapter 3 to simplify complicated expressions and calculations.

2.1.3 Summary

Here is a quick overview of the introduced notation:

a : Scalar

$\mathbf{A}$: Matrix or tensor

$\mathbf{a}$: Vector

$\mathbb{A}$: Matrix-of-matrices

$\check{\mathbf{A}}$: Vectorized matrix-of-matrices (also written as vec($\mathbb{A}$))

2.2 Concepts of Continuum Mechanics

This section gives a broad introduction to some of the concepts used in continuum mechanics. In continuum mechanics, we are less interested in small particles like atoms or molecules of an object but rather in pieces of matter that are large in comparison. The reason for this is that the calculation for the behaviour of individual atoms is challenging for larger systems. Therefore, we are concerned with the mechanical behaviour of solids and fluids on the macroscopic scale and treat the material as uniformly distributed throughout regions of space. With this approach, it is possible to define quantities such as displacement and density as continuous functions of the position ([Spe80], p. 1).

2.2.1 Deformation

When studying an object, we are among other properties interested in how it may change its shape over a certain period of time. If the object undergoes a change of shape, we call this process a deformation. In order to produce a deformation, there must be at least one force present that interacts with the object. Typically, we apply one or multiple forces over

an object and are then interested in its deformed state. The term *strain* is used as a measure of deformation, and we denote *stress* as the force per unit area.

Graphically, we can imagine a deformation with the help of a two-dimensional deformation map, as shown in Fig. 2.1. The ellipse on the left side represents an object in its rest state. A function ϕ maps this rest state of the ellipse to a deformed state as shown on the right side of the image.

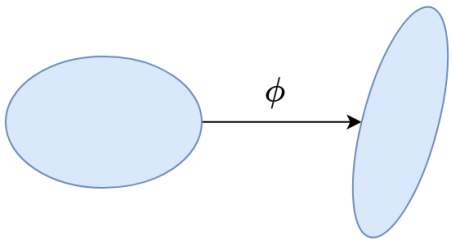


Figure 2.1: Deformation Map

We can imagine that we map each particle of a chosen object from its rest state to a deformed one. We can characterize each particle X of a body by a vector $\mathbf{x}$ containing its positional coordinates. This vector is the reference configuration. If we displace the particle, we can describe its new coordinate vector $\mathbf{x}'$ with

$$\mathbf{x}' = \phi(\mathbf{x}).$$

Example: A simple example of a deformation is the stretching of a cuboid along the x- and y-axis, as illustrated in Fig. 2.2.

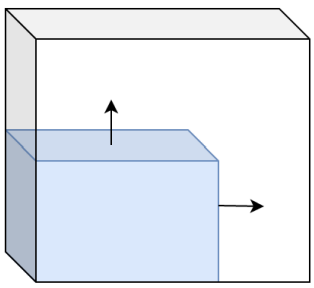


Figure 2.2: Stretching of a cuboid

The coloured volume in Fig. 2.2 represents the cuboid in its rest state. In this case, the vector $\mathbf{x}$ contains the three coordinate values for the x-, y-, and z-axis:

$$\mathbf{x} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

After the stretching, the new position for each particle can be calculated by

$$\mathbf{x}' = \phi(\mathbf{x}) = \begin{bmatrix} 1.5x + 0y + 0z \\ 0x + 2y + 0z \\ 0x + 0y + 1z \end{bmatrix} = \begin{bmatrix} 1.5x \\ 2y \\ z \end{bmatrix}. \tag{2.1}$$

2.2.2 Deformation Gradient

An essential quantity in continuum mechanics is the deformation gradient $\mathbf{F}$. It serves as a characterization of the deformation. With its help, we can calculate properties like the change of volume or length of an object during a deformation. We can obtain $\mathbf{F}$ by using the function ϕ discussed in the previous section and taking the derivative of each component of ϕ with respect to each component of the reference vector $\mathbf{x}$. In the following, we only work with deformations in the three-dimensional space. In that case, $\mathbf{F}$ is a (3×3)-matrix and can be calculated by

$$\mathbf{F} = \left[\frac{\phi(\mathbf{x})}{\partial x} \middle| \frac{\phi(\mathbf{x})}{\partial y} \middle| \frac{\phi(\mathbf{x})}{\partial z} \right]. \tag{2.2}$$

For simplification, I will use this representation of $\mathbf{F}$ in the upcoming chapters:

$$\mathbf{F} = \left[\mathbf{f_0} \middle| \mathbf{f_1} \middle| \mathbf{f_2} \right] = \begin{bmatrix} f_0 & f_3 & f_6 \\ f_1 & f_4 & f_7 \\ f_2 & f_5 & f_8 \end{bmatrix} \tag{2.3}$$

In this equation, $\mathbf{f_i}$ are the column vectors, and f_i symbolize the scalar entries of the matrix. $\mathbf{F}$ can be useful for us to find out more about the deformation. For example, we know that if $\mathbf{F}$ is equal to the identity matrix $\mathbf{I}$, there is no deformation present. That would be the case for rigid body displacements. In addition, $\mathbf{F}$ can be factorized and used to

calculate other quantities. That will be explained in section 2.4 after introducing the needed theorems in section 2.3.

Example: I am again taking the example of stretching the cuboid from Fig. 2.2. We can obtain the resulting deformation gradient of this example with the help of Eq. (2.2) applied to Eq. (2.1):

$$\mathbf{F} = \left[\frac{\phi(\mathbf{x})}{\partial x} \middle| \frac{\phi(\mathbf{x})}{\partial y} \middle| \frac{\phi(\mathbf{x})}{\partial z} \right] = \begin{bmatrix} \frac{\partial[1.5x]}{\partial x} & \frac{\partial[1.5x]}{\partial y} & \frac{\partial[1.5x]}{\partial z} \\ \frac{\partial[2y]}{\partial x} & \frac{\partial[2y]}{\partial y} & \frac{\partial[2y]}{\partial z} \\ \frac{\partial[z]}{\partial x} & \frac{\partial[z]}{\partial y} & \frac{\partial[z]}{\partial z} \end{bmatrix} = \begin{bmatrix} 1.5 & 0.0 & 0.0 \\ 0.0 & 2.0 & 0.0 \\ 0.0 & 0.0 & 1.0 \end{bmatrix}$$

As we can see, $\mathbf{F}$ is not equal to the identity matrix, so we are sure that the object was indeed deformed. For this simple example, this information might seem unnecessary because the visualisation already shows that the cuboid gets deformed, but for more complex deformations, this information is very useful.

2.2.3 Deformation Energy

We can deform an object by putting a certain amount of force into the system. We could, for example, stretch a spring. The spring then stores some amount of potential energy. If we let the spring go, we transfer the potential energy to kinetic energy, and the spring usually recovers into its rest state. This energy that the spring stores during the deformation is called *deformation energy* or *strain energy* Ψ. The strain energy density is the strain energy per unit volume. Any increment of the strain energy density is equal to the work done by the stresses in order to alter the strains ([Kor17], p.10). That means that energy is an indicator of how much force must be applied to deform an object in a certain way. Thus, we can use the deformation energy to express the relationship between the stresses and strains. We can illustrate this with a stress-strain curve shown in Fig. 2.3. The area under the stress-strain curve corresponds to the strain energy density.

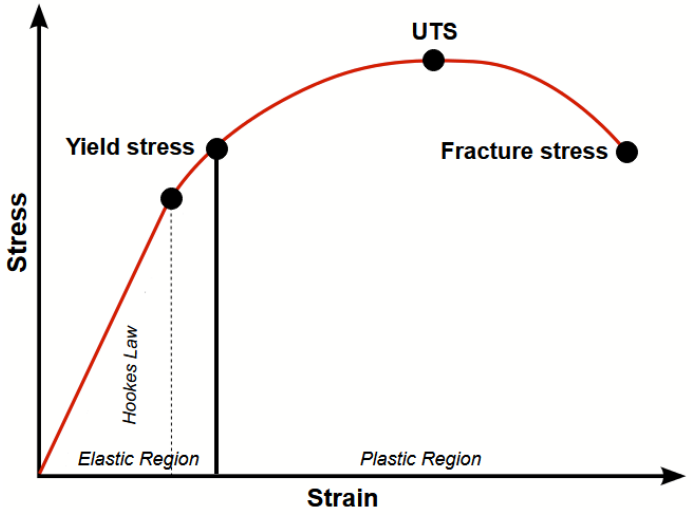


Figure 2.3: Example for a stress-strain curve[1]

As we can see in Fig. 2.3, with increasing stress, the object goes through different states. Before reaching the yielding point illustrated with *Yield stress*, the material has the ability to recover into its rest shape. This ability is called *Elasticity* ([Ber15], p. 211). After this point, the material cannot recover anymore due to permanent fractures during the deformation.

In addition, we can see that at the beginning of the curve, we have a straight line until the first black dot. That means that the relationship between the stresses and strains is linear. This is called *Linear Elasticity* ([Kor17], p. 5). Linear elasticity is strictly related to *Hooke's law*, which states that the resulting deformation is proportional to the applied force and that the object is able to recover into its rest shape under these conditions. Hence, we can also say that the material is consistent with *Hooke's law* in this interval.

After the straight line, we can spot a non-linear relationship before the yielding point. The material is still able to recover, but there is a non-linear relationship between the stresses and strains. The resulting deformation is larger than how Hooke's law would predict it. Materials that fall into this category are called *Hyperelastic Materials*. Hyperelasticity is a generalization of linear elasticity with a non-linear relationship and is suited for larger strain predictions ([Ber15], p. 218).

[1] the orginial image was taken from:
https://commons.wikimedia.org/wiki/File:Stress-strain_curve.svg

UTS in Fig. 2.3 is abbreviated for ultimate tensile strength and defines the maximal stress an object can bear before breaking. At the point of *Fracture stress* the material finally tears apart.

The behaviour of the object depends heavily on the material it consists of, and the stress-strain curve looks different for each material. We need to choose the energy function according to the material properties. The behaviour of human flesh can be put into the category of hyperelastic materials. Thus, the energy function also has to be hyperelastic for our purposes.

2.2.4 Material Constants

When we look at a deformation of an object, we need to consider the material the object consists of. Materials can be very stiff like steel or easily deformable like rubber. In order to measure the deformation of a specific material, we need the Poisson's ratio of said material. The Poisson's ratio is a material constant that is defined as

$$\nu = -\frac{\epsilon_{11}}{\epsilon_{22}} \in [-1, 0.5], \tag{2.4}$$

where ϵ_{11} is the lateral and ϵ_{22} the axial strain. The range in which ν lies in starts at -1 and goes up to 0.5 ([MR09]). In order to understand this quantity better, we can use an example: Imagine pulling a rubber band on each of its sides. After pulling a bit, we can observe that the band gets longer and the middle part gets narrower. The Poisson's ratio indicates the extent of this deformation. Some materials, such as rubber, are more easily deformable and therefore lead to a higher Poisson's ratio.

Usually, the Poisson's ratio of a material is positive. A negative value would mean that the material becomes wider in the cross-section when we stretch it. This behaviour is very uncommon in nature. Examples of materials with a negative Poisson's ratio are for instance discussed in *Foam structures with a negative Poisson's ratio* ([Lak87]) or *Advances in negative Poisson's ratio materials* ([Lak93]). Table 2.1 shows examples of positive Poisson's ratios of various materials.

Material	**Poisson's ratio**
C (graphite)	0.31
Sn (metal)	0.357
Cu	0.355
Zn	0.25
Ag	0.36
Au	0.45
Concrete	0.20–0.37
Titanium (dental alloy)	0.30–0.31
Bronze	0.34
18–8 Stainless steel	0.305
Natural rubber	0.4999
B_2O_3 glass	0.30
GeO_2 glass	0.20

Table 2.1: Different materials with their Poisson's ratio ([MR09], p. 3)

In the context of flesh simulation, the Poisson's ratio tells us how resistant flesh is to volume change. The Poisson's ratio of biological tissues such as flesh, fat, and muscles takes on higher values in the range of 0.45 and 0.5 ([SGK18]).

The calculation of the Poisson's ratio, as defined in Eq. (2.4), is a challenge. Fortunately, we can make use of the *Lamé Parameters*, the two material-specific constants μ and λ. With the help of these two constants, we can transform Eq. (2.4) into the form

$$\nu = \frac{\lambda}{2(\lambda + \mu)}. \tag{2.5}$$

This equation allows us to calculate the Poisson's ratio much easier ([Ber15], p. 231).

2.3 Mathematical Background

Now that we have established an understanding of the general concepts of continuum mechanics, we can look at the more technical part. Since mathematics play an important role in the field of interests, we need to build a solid background before diving further into more technical

calculations. This section covers all the essential concepts used later in the calculations. A basic understanding of linear algebra is assumed.

2.3.1 Singular Value Decomposition

The singular value decomposition (SVD) will play an important role while working with the deformation gradient. It represents the best possible approximation of a given matrix by a matrix of low rank. This approximation can be looked at as a compression of the given data ([LM15], p. 295). Firstly, we need to define what singular values are.

Definition 1 (**Singular Values**). *The singular values of a matrix* $\mathbf{A} \in \mathbb{R}^{m\times n}$ *are the square roots of the eigenvalues of* $\mathbf{A}\mathbf{A}^\mathsf{T}$.

The theorem of the singular value decomposition states that we can factor every $(m\times n)$-matrix into one orthogonal $(m\times m)$-, one orthogonal $(n\times n)$-, and one diagonal $(m \times n)$-matrix. More formally:

Theorem 1 (**The SVD Theorem**). *Let* $\mathbf{A} \in \mathbb{R}^{m\times n}$ *be a matrix having* r *positive singular values,* $m \geq n$. *Then there exist orthogonal matrices* $\mathbf{U} \in \mathbb{R}^{m\times m}$, $\mathbf{V} \in \mathbb{R}^{n\times n}$ *and a diagonal matrix* $\tilde{\mathbf{\Sigma}} \in \mathbb{R}^{m\times n}$ *such that*

$$\mathbf{A} = \mathbf{U}\tilde{\mathbf{\Sigma}}\mathbf{V}^\mathsf{T}$$

$$\tilde{\mathbf{\Sigma}} = \begin{bmatrix} \mathbf{\Sigma} & 0 \\ 0 & 0 \end{bmatrix}$$

where $\mathbf{\Sigma} = diag\ (\sigma_1, \sigma_2, \ldots, \sigma_r)$, *and* $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$ *are the positive singular values of* $\mathbf{A}$.

This definition and theorem were taken from *Numerical linear algebra with applications: Using MATLAB* ([For14], p. 113, p. 300).

2.3.2 Polar Decomposition

Another theorem I will be using in section 2.4 is the polar decomposition theorem:

Theorem 2 (**The Polar Decomposition Theorem**). *Let* $\mathbf{F}$ *be a non-singular square matrix. Then* $\mathbf{F}$ *can be decomposed uniquely into either of the following two products*

$$\mathbf{F} = \mathbf{RU}, \quad \mathbf{F} = \mathbf{VR}$$

where $\mathbf{R}$ *is an orthogonal matrix, and* $\mathbf{U}$ *and* $\mathbf{V}$ *are positive definite symmetric matrices.*

This theorem was taken from *Continuum Mechanics*, in which they include the proof for (3×3)-matrices ([Spe80], p. 12).

2.3.3 Frobenius Norm

The Frobenius norm is a matrix norm. It allows us to measure and compare entities in a multidimensional space.

Definition 2 (**Frobenius Norm**). *Let* $\boldsymbol{A}$ *be an* $(m \times n)$*-matrix in the real or complex domain. Then the Frobenius norm is defined as*

$$\|\mathbf{A}\|_F := \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} |a_{ij}|^2}.$$

We can further represent the norm with the trace of the matrix, in which $\mathbf{A}^*$ is the conjugate transpose of $\mathbf{A}$. We can then use the SVD of $\mathbf{A}$ and write the norm with respect to the singular values of $\mathbf{A}$, denoted by σ_i:

$$\|\mathbf{A}\|_F = \sqrt{\text{trace}(\mathbf{AA}^*)} = \sqrt{\sum_{i=1}^{\min\{m,n\}} \sigma_i^2} \tag{2.6}$$

2.4 Deformation Gradient

Now that we understand the necessary mathematical background, we can factorize the deformation gradient $\mathbf{F}$ and use it to calculate other useful quantities. This section describes these factorizations and calculations by using the theorems introduced in the previous section.

2.4.1 Singular Value Decomposition of F

Using the SVD theorem shown in Thm. 1, $\mathbf{F}$ can be written in the form of

$$\mathbf{F} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\mathsf{T}} \tag{2.7}$$

in which $\mathbf{\Sigma}$ is defined as

$$\mathbf{\Sigma} = \begin{bmatrix} \sigma_0 & 0 & 0 \\ 0 & \sigma_1 & 0 \\ 0 & 0 & \sigma_2 \end{bmatrix}. \tag{2.8}$$

Each σ_i denotes a singular value of $\mathbf{F}$. $\mathbf{U}$ and $\mathbf{V}$ are both orthogonal matrices that represent the rotation of $\mathbf{F}$. $\mathbf{\Sigma}$, on the other hand, indicates the scaling of each coordinate x_i by the factor σ_i. By using the standard convention, we would choose only nonnegative entries for $\mathbf{\Sigma}$. Unfortunately, this approach only works well for $\det(\mathbf{F}) \geq 0$ ([ITF04], p. 134). Hence, when working with inverted material, we might run into problems. The authors of the paper SNH-FS decided to move the reflections to $\mathbf{\Sigma}$. Therefore, $\mathbf{\Sigma}$ is allowed to have a negative entry. This has the effect that $\det(\mathbf{U})$ and $\det(\mathbf{V})$ are both equal to 1.

2.4.2 Polar Decomposition of F

With the help of Thm. 2 we can decompose the deformation gradient into the form

$$\mathbf{F} = \mathbf{R}\mathbf{S}, \tag{2.9}$$

where $\mathbf{R}$ is orthogonal and $\mathbf{S}$ is a positive definite symmetric matrix. $\mathbf{R}$ symbolises the rotation that $\mathbf{F}$ undergoes, whereas $\mathbf{S}$ contains the scaling along the orthogonal directions of $\mathbf{F}$.

2.4.3 Relative Volume Change

A piece of useful information about a deformation is the relative volume change of the deformed object. It can be calculated by the determinant of $\mathbf{F}$:

$$J = \det(\mathbf{F}) \tag{2.10}$$

For a normal deformation, J is a positive value. A determinant of zero would mean that we deform the object into a zero volume state, e.g. a plane or point. A negative determinant indicates an inversion.

2.4.4 Cauchy-Green

In chapter 3, I will also use the right Cauchy-Green tensor $\mathbf{C}$. It can be calculated by

$$\mathbf{C} = \mathbf{F}^{\mathsf{T}}\mathbf{F}. \tag{2.11}$$

$\mathbf{C}$ is a (3×3)-matrix for deformations in the 3D domain. Using $\mathbf{C}$, we can calculate the first right Cauchy-Green invariant I_C with

$$I_C = \mathrm{tr}(\mathbf{C}). \tag{2.12}$$

Invariants of the Cauchy-Green tensor are often used in formulations of the strain energy density.

Chapter 3

Stable Neo-Hookean Flesh Simulation

In this chapter, I will examine further the topic of the paper *SNH-FS*, in particular, the calculations and conclusions made by the authors. I will go through each step they did, and my goal is to fill in some gaps to help in understanding the thought process. For this, I will include some of their calculations in more detail and add additional explanations. I would like to note that the information provided in this chapter is taken from the paper SNH-FS if not stated otherwise. For simplification, I will not include a reference in each paragraph.

The structure of this chapter is similar to the one of the paper: At first, I will go through the process of the formulation of the Stable Neo-Hookean energy. This includes pointing out the requirements and looking at existing energies. Finally, I will show that a complete eigenanalysis can be performed on the constructed energy.

3.1 Deformation Gradient

In the following, I will use the definitions involving the deformation gradient $\mathbf{F}$ introduced in chapter 2. These definitions are summarized in Table 3.1 for a better overview.

Symbol	**Definition**
$\mathbf{F} = \mathbf{RS}$	Polar decomposition
$J = \det(\mathbf{F})$	Relative volume change
$\mathbf{C} = \mathbf{F}^\mathsf{T}\mathbf{F}$	Right Cauchy-Green tensor
$I_C = \mathrm{tr}(\mathbf{C})$	First right Cauchy-Green invariant

Table 3.1: Quantities derived from the deformation gradient

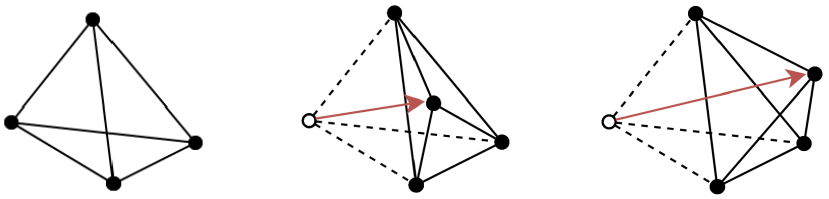
3.2 Energy Formulation

In this section, I will go through the process of formulating a new deformation energy. At first, I will explain what properties a deformation energy should have. Secondly, I will analyse existing energies. Finally, I will go through the formulation of the novel energy from the paper SNH-FS.

3.2.1 Stability

The core goal of the paper was to model deformations for virtual characters that have human-like features. In order to achieve better results than what has been done in current research, they formulated a new deformation energy. In chapter 2, I concluded that the appropriate energy for animating soft tissues such as flesh has to be hyperelastic. Another important property is the stability of the energy. We need a hyperelastic energy that is stable in the following four ways, which were introduced in the paper itself:

1. Inversion Stability: Given some arbitrary object, it is possible that while deforming the object, we can arrive at a zero volume state or even an entire inversion. For example, we can look at the tetrahedron shown in Fig. 3.1a. In Fig. 3.1b, we see a deformed state of this tetrahedron where the volume is scaled down to zero, and we are left with a simple triangle. In Fig. 3.1c, the tetrahedron arrives at an inverted state. The deformation energy has to be able to deal with both cases without creating severe artefacts. That means that the energy has to be singularity-free, meaning that it is defined for every possible point. This property should ideally hold without needing any filters or threshold.



(a) Rest state (b) Zero-volume state (c) Inverted state

Figure 3.1: Inversion of a tetrahedron

2. Reflection stability: While deforming an object, it can occur that we are dealing with reflections. An example of a reflection in 2D is shown in Fig. 3.2. The coloured triangle is reflected over the y-axis. A matrix that represents a reflection is orthogonal with a determinant equal to -1. The deformation energy needs to be well behaved under reflections. This has to hold, regardless of the reflection convention used in the SVD of $\mathbf{F}$. The reflection convention is explained in chapter 2 in subsection 2.4.1.

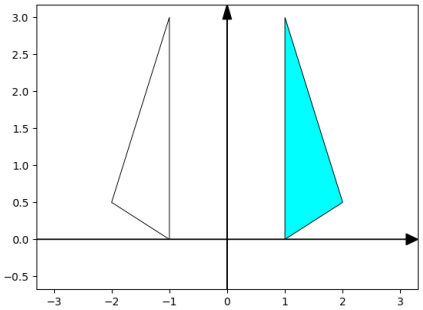


Figure 3.2: Reflection of a triangle over the y-axis

3. Rest stability: While deforming an object in a certain way, we apply one or multiple forces to that object, which influences the deformation. But if we remove all the forces, the object should be able to go back into its rest state. This ability was introduced in chapter 2 as elasticity.

4. Meta-stability under degeneracy: Here we are dealing with a special case of deformation. If we crush a volumetric object into a plane, line, or point, we change the object into a degenerate case. This process is illustrated for a cube in Fig. 3.3. Even with these extreme conditions,

the object should still be able to recover into its actual shape and not some other.

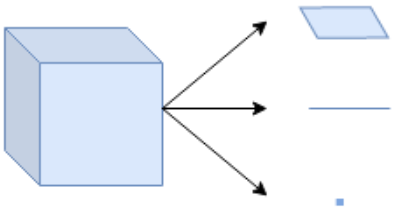


Figure 3.3: Illustration of degeneracy

Based on these four requirements, I will judge the existing models for the deformation energy. These criteria determine whether a proposed energy is suited for simulating the fleshy look or not.

3.2.2 Existing Neo-Hookean Energies

In previous literature, a few energies were proposed, which I will analyse in this section. They are listed in Table 3.2. All of these energies were formulated for hyperelastic materials.

Energy	**Author(s)**
$\Psi_{Neo} = \frac{\mu}{2}(I_C - 3) - \mu \log J + \frac{\lambda}{2}(\log J)^2$	e.g. Bonet and Wood 1997 ([BW97])
$\Psi_{\mathrm{A}} = \frac{\mu}{2}(I_C - 3) - \mu \log J + \frac{\lambda}{2}(J-1)^2$	Odgen 1997 ([Ogd97])
$\Psi_{\mathrm{B}} = \frac{\mu}{2}\left(J^{-2/3} I_C - 3\right) + \frac{\lambda}{2}(J-1)^2$	Bower 2009 ([Bow09])
$\Psi_{\mathrm{C}} = \frac{\mu}{2}\left(J^{-2/3} I_C - 3\right) + \frac{\lambda}{2}(J-1)$	Wang and Yang 2016 ([WY16])

Table 3.2: Summary of proposed energies ([SGK18])

According to the Valanis-Landel hypothesis, many hyperelastic energies can be split up into a 1D length, 2D area, and 3D volume term. The energies in Table 3.2 only contain the length and volume term. Hence, each of these energies can be separated into a 1D length term and a 3D volume term. The 1D length term penalizes the length changes an object

undergoes during a deformation, whereas the 3D volume term penalizes the change in volume of the object.

Energy	**1D length term**	**3D volume term**
Ψ_{Neo}	$\frac{\mu}{2}(I_C - 3)$	$-\mu \log J + \frac{\lambda}{2}(\log J)^2$
Ψ_{A}	$\frac{\mu}{2}(I_C - 3)$	$-\mu \log J + \frac{\lambda}{2}(J-1)^2$
Ψ_{B}	$\frac{\mu}{2}\left(J^{-2/3}I_C - 3\right)$	$\frac{\lambda}{2}(J-1)^2$
Ψ_{C}	$\frac{\mu}{2}\left(J^{-2/3}I_C - 3\right)$	$\frac{\lambda}{2}(J-1)$

Table 3.3: Energies split up into their 1D length and 3D volume term

1D Length Term

The term that is used for Ψ_{Neo} and Ψ_A is defined as

$$\Psi_M = \frac{\mu}{2}(I_C - 3).$$

This energy was originally proposed by Mooney in 1940 ([Moo40]). It is today known as the Neo-Hookean energy because of Rivlin ([Riv48]). If we expand the term with the singular values of the deformation gradient $\mathbf{F}$, we get the following equation:

$$\Psi_M = \frac{\mu}{2}\left(\sigma_0^2 + \sigma_1^2 + \sigma_2^2 - 3\right)$$

The term reaches its minimum at a zero volume state, meaning at $I_C = 0$, which leads to $\Psi_M = -3$. Because this is not desirable, Mooney added the hard constraint that J should be equal to 1. Hence, the energy is minimized at a volume-preserving configuration. Note that the energy is singularity free and well defined under inversion.

The second term is defined by

$$\Psi_R = \frac{\mu}{2}\left(J^{-2/3}I_C - 3\right).$$

It is used in Ψ_B and Ψ_C. This term was introduced by Rivlin in 1948 ([Riv48]). Using the singular values of $\mathbf{F}$, we get the following expression:

$$\Psi_R = \frac{\mu}{2}\left(\frac{\sigma_0^2 + \sigma_1^2 + \sigma_2^2}{(\sigma_0\sigma_1\sigma_2)^{\frac{2}{3}}} - 3\right)$$

Unfortunately, this term is not singularity free. If J is equal to zero, the result is not defined anymore.

3D Volume Term

The volume term of Ψ_{Neo} is described as

$$\Psi_{Neo,volume} = -\mu \log J + \frac{\lambda}{2}(\log J)^2$$

and results in numerical problems since the logarithmic function is not defined for $J < 0$ and grows unbounded for $J \to 0$. In conclusion, $\Psi_{Neo,volume}$ is not singularity free. The same applies to the 3D volume term of Ψ_A, namely

$$\Psi_{A,volume} = -\mu \log J + \frac{\lambda}{2}(J-1)^2.$$

The volume term of Ψ_B and Ψ_C is of the form

$$\Psi_{B,volume} = \frac{\lambda}{2}(J-1)^2$$

and does not have these problems. It is bounded, well defined, and invertible. After these observations, the authors of SNH-FS combined the robust length term with the robust volume term and received Ψ_D, which is defined as

$$\Psi_D = \frac{\mu}{2}(I_C - 3) + \frac{\lambda}{2}(J-1)^2.$$

Ψ_D is singularity free and well defined under inversion. Unfortunately, it does not satisfy the requirement of being rest stable, which I will discuss in the next section.

In addition, we can see that each of the proposed energies contains a term that is not well defined under certain conditions. That means that each of the energies in Table 3.2 is not singularity free or well defined under inversion and therefore confirms the need for the formulation of a new energy.

3.2.3 Rest Stabilization

Although Ψ_D meets almost all stated requirements, it is not rest stable. We can show that with the first Piola-Kirchhoff stress tensor (PK1). For our case, it is defined as

$$P(\mathbf{F}) = \frac{\partial \Psi}{\partial \mathbf{F}}(\mathbf{F}). \tag{3.1}$$

With the help of Eq. (3.1) we can calculate the PK1 of Ψ_D. For the following calculations, keep in mind that $I_C = \text{tr}(\mathbf{F}^\mathsf{T}\mathbf{F})$ and $J = \det(\mathbf{F})$ holds with $\mathbf{F}$ being the deformation gradient. I will use the explicit terms during the calculations for a better understanding.

$$\begin{aligned} P_D(\mathbf{F}) = \frac{\partial \Psi_D}{\partial \mathbf{F}}(\mathbf{F}) &= \frac{\partial}{\partial \mathbf{F}} \left[\frac{\mu}{2} (I_C - 3) + \frac{\lambda}{2} (J-1)^2 \right] \\ &= \frac{\partial}{\partial \mathbf{F}} \left[\frac{\mu}{2} \Big(\text{tr}(\mathbf{F}^\mathsf{T}\mathbf{F}) - 3 \Big) + \frac{\lambda}{2} (\det(\mathbf{F}) - 1)^2 \right] \\ &= \frac{\partial}{\partial \mathbf{F}} \frac{\mu}{2} \Big(\text{tr}(\mathbf{F}^\mathsf{T}\mathbf{F}) - 3 \Big) + \frac{\partial}{\partial \mathbf{F}} \frac{\lambda}{2} (\det(\mathbf{F}) - 1)^2 \end{aligned}$$

Looking at the two terms separately, we get:

$$\begin{aligned} &\frac{\partial}{\partial \mathbf{F}} \frac{\mu}{2} (\text{tr}(\mathbf{F}^\mathsf{T}\mathbf{F}) - 3) = \frac{\mu}{2} 2\mathbf{F} = \mu \mathbf{F} \\ &\frac{\partial}{\partial \mathbf{F}} \frac{\lambda}{2} (\det(\mathbf{F}) - 1)^2 = \frac{\lambda}{2} \frac{\partial \det(\mathbf{F})}{\partial \mathbf{F}} 2(\det(\mathbf{F}) - 1) = \lambda \frac{\partial \det(\mathbf{F})}{\partial \mathbf{F}} (\det(\mathbf{F}) - 1) \end{aligned}$$

Hence, P_D resolves to

$$P_D(\mathbf{F}) = \mu \mathbf{F} + \lambda \frac{\partial \det(\mathbf{F})}{\partial \mathbf{F}} (\det(\mathbf{F}) - 1).$$

In order to find out whether an energy is rest stable or not, we can set the input variable to the identity matrix $\mathbf{I}$. An energy is rest stable if $P(\mathbf{I}) = 0$ ([SGK18]). Unfortunately, this is not the case for Ψ_D:

$$P_D(\mathbf{I}) = \mu \mathbf{I} + \lambda \frac{\partial \det(\mathbf{I})}{\partial \mathbf{F}} (\det(\mathbf{I}) - 1) = \mu \mathbf{I} \neq 0$$

That is problematic because if we simulate a deformation, the object will shrink in volume when it should be back in its rest state. In order to solve this problem, the authors modified $(J-1)^2$ to $(J-\alpha)^2$. Using this

modification, the energy shifts to

$$\Psi_E = \frac{\mu}{2}(I_C - 3) + \frac{\lambda}{2}(J - \alpha)^2.$$

Now PK1 for Ψ_E can be calculated similarly as before, and we get:

$$\begin{aligned} P_E(\mathbf{F}) &= \frac{\partial \Psi_E}{\partial \mathbf{F}} = \frac{\partial}{\partial \mathbf{F}}\left[\frac{\mu}{2}(I_C - 3) + \frac{\lambda}{2}(J - \alpha)^2\right] \\ &= \frac{\partial}{\partial \mathbf{F}}\left[\frac{\mu}{2}\Big(\mathrm{tr}(\mathbf{F}^\mathsf{T}\mathbf{F}) - 3\Big) + \frac{\lambda}{2}(\det(\mathbf{F}) - \alpha)^2\right] \\ &= \mu\mathbf{F} + \lambda\frac{\partial \det(\mathbf{F})}{\partial \mathbf{F}}(\det(\mathbf{F}) - \alpha) \end{aligned}$$

Solving for an alpha that satisfies $P_E(\mathbf{I}) = 0$ gives us $\alpha = 1 + \frac{\mu}{\lambda}$. Now Ψ_E has to be changed accordingly:

$$\Psi_E = \frac{\mu}{2}(I_C - 3) + \frac{\lambda}{2}(J - 1 - \frac{\mu}{\lambda})^2$$

We can now expand the quadratic and obtain

$$\Psi_E = \frac{\mu}{2}(I_C - 3) - \mu(J - 1) + \frac{\lambda}{2}(J - 1)^2 + \left(\frac{\mu}{\lambda}\right)^2.$$

Since constants disappear under differentiation, this expression is functionally equivalent to

$$\Psi_E = \frac{\mu}{2}(I_C - 3) - \mu(J - 1) + \frac{\lambda}{2}(J - 1)^2. \tag{3.2}$$

Note that Ψ_E looks very similar to Ψ_{Neo}. The difference is that $log(J)$ is replaced with $(J - 1)$ in Ψ_E. Keep in mind that $(J - 1)$ is the first term of the Taylor approximation of $\log(J)$ at $J = 1$:

$$\log(J) = \boldsymbol{(J - 1)} - \frac{1}{2}(J - 1)^2 + \frac{1}{3}(J - 1)^3 + ...$$

Thus, Ψ_E can be looked at as an approximation of Ψ_{Neo} that is singularity-free and has rest stability. In addition, it has reflection stability because I_C contains the squared singular values of $\mathbf{F}$. Hence, any negation convention is irrelevant, and the J term contains the product of the singular values, so the sign convention is again irrelevant. Therefore, Ψ_E has inversion, reflection, and rest stability.

3.2.4 Meta-Stability under Degeneracy

We know from the last section that the energy Ψ_E has inversion, reflection, and rest stability. Now we are interested in how it behaves under degeneracy. The authors of the paper *SNH-FS* viewed this examination as a Drucker stability analysis (see e.g. [Bow09]). Drucker stability describes a set of criteria that a material can satisfy or not. It is a measurement of the stability of a material. The calculations can be found in the supplemental material of *SNH-FS*. I will only present the results here.

The energy remains stable when crushing the object into a *plane*. The object is able to return to its rest state. Thus, no further adjustments need to be done. When crushing the object into a *line*, the energy is meta-stable. The object will self-restore into the correct shape after any perturbations. A possible perturbation could be a momentum. The alternative would be a state that yields singularities, which is not desirable. Hence, we can leave the energy as it is. We can also crush the object to a *point*, which results in $\mathbf{F} = 0$. In this case, the material cannot recover into its rest state. This effect is small for a higher Poisson's ratio. For completeness, the authors nevertheless added a regularized origin barrier:

$$\Psi_{origin} = -\frac{\mu}{2}\log(I_C + \delta)$$

This term eliminates the unwanted behaviour for point compression for all positive Poisson's ratio, without interfering with the other requirements for the energy. It can be shown that the value of δ should be set to 1. This proof is also included in the supplemental material of the paper.

We can now write the final energy down as

$$\boxed{\Psi_{new} = \frac{\mu}{2}(I_C - 3) + \frac{\lambda}{2}(J - \alpha)^2 - \frac{\mu}{2}\log(I_C + 1)\,.} \tag{3.3}$$

With this adjustment the rest stability term shifts to $\alpha = 1 + \frac{\mu}{\lambda} - \left(\frac{\mu}{4}\right)\lambda$.

To note here is that the authors mainly introduced this origin barrier because of pedagogical completeness. In practice, the origin barrier was

not needed, and the version of the energy without it, formulated in Eq. (3.2), was used.

3.2.5 Lamé Reparameterization

Because of the origin barrier introduced in the last section, we need to make a reparametrization of the Lamé parameters λ and μ. As I explained in subsection 2.2.3, there is a non-linear relationship between the stresses and strains for hyperelastic materials for larger strain predictions. But for infinitesimal deformation, we should still be consistent with Hooke's law. In order to reach this goal, the model needs to reproduce the PK1 of linear elasticity, which is defined as

$$\mathbf{P}(\mathbf{F}) = 2\mu_{Lamé}\epsilon + \lambda_{Lamé}\,\mathrm{tr}(\epsilon)\mathbf{I}, \tag{3.4}$$

where $\epsilon = \frac{1}{2}(\mathbf{F} + \mathbf{F}^{\mathsf{T}}) - \mathbf{I}$ is the linearized strain tensor and $\mu_{Lamé}$ and $\lambda_{Lamé}$ are the Lamé parameters in linear elasticity. If we linearize the stress from Eq. (3.3) and keep it consistent with the form described in Eq. (3.4), the values shift to $\mu = \frac{4}{3}\mu_{Lamé}$ and $\lambda = \lambda_{Lamé} + \frac{5}{6}\mu_{Lamé}$. The equation for the Poisson's ratio then shifts to

$$\nu = \frac{\lambda - \left(\frac{5}{8}\right)\mu}{2\left(\lambda + \left(\frac{1}{8}\right)\mu\right)}.$$

These reparameterized expressions for the Lamé parameters were used in all of the tests made in the paper SNH-FS. In addition, with these formulations, it can be shown that the energy does not introduce any spurious minima in the range of $\nu \in [0, 0.5)$. The details are again given in the supplemental material of the paper SNH-FS.

3.3 Energy Analysis

In order to simulate physical deformations, we put some constraints on an object and minimize the deformation energy. Thus, we need a minimization tool. In this case, we will use Newton's method. I will further explain Newton's method in section 4.4. But first, we need to

know more about the properties of the energy. The goal of this chapter is to show that a complete eigenanalysis can be performed on the new energy, formulated in Eq. (3.3), which will help us form a qualitative understanding of the energy.

3.3.1 First Piola-Kirchhoff Stress (PK1)

In order to analyse the energy and build the Hessian terms, the first step is to calculate PK1 for Eq. (3.3) with $\alpha = 1 + \frac{\mu}{\lambda} - \left(\frac{\mu}{4}\right)\lambda$. Again, I_C is equal to $\mathrm{tr}(\mathbf{F}^\mathsf{T}\mathbf{F})$, J signifies $\det(\mathbf{F})$, and I am using the explicit terms during the calculations. With this, we can calculate P_{new} by

$$\begin{aligned} P_{new}(\mathbf{F}) &= \frac{\partial \Psi_{new}}{\partial \mathbf{F}} = \frac{\partial}{\partial \mathbf{F}}\left[\frac{\mu}{2}\left(I_C - 3\right) + \frac{\lambda}{2}(J-\alpha)^2 - \frac{\mu}{2}\log\left(I_C + 1\right)\right] \\ &= \frac{\partial}{\partial \mathbf{F}}\left[\frac{\mu}{2}\Big(\mathrm{tr}(\mathbf{F}^\mathsf{T}\mathbf{F}) - 3\Big) + \frac{\lambda}{2}(\det(\mathbf{F})-\alpha)^2 - \frac{\mu}{2}\log\Big(\mathrm{tr}(\mathbf{F}^\mathsf{T}\mathbf{F}) + 1\Big)\right]. \end{aligned}$$

Similar to before, I am looking at the terms separately:

$$\begin{aligned} &\frac{\partial}{\partial \mathbf{F}}\frac{\mu}{2}\Big(\mathrm{tr}(\mathbf{F}^\mathsf{T}\mathbf{F}) - 3\Big) = \mu\mathbf{F} \\ &\frac{\partial}{\partial \mathbf{F}}\frac{\lambda}{2}(\det(\mathbf{F})-\alpha)^2 = \lambda(\det(\mathbf{F})-\alpha)\frac{\partial \det(\mathbf{F})}{\partial \mathbf{F}} \\ &\frac{\partial}{\partial \mathbf{F}}\frac{\mu}{2}\log\Big(\mathrm{tr}(\mathbf{F}^\mathsf{T}\mathbf{F}) + 1\Big) = \frac{\mu}{2}\,2\mathbf{F}\,\frac{1}{\mathrm{tr}(\mathbf{F}^\mathsf{T}\mathbf{F}) + 1} = \mu\mathbf{F}\frac{1}{\mathrm{tr}(\mathbf{F}^\mathsf{T}\mathbf{F}) + 1} \end{aligned}$$

When we combine these terms again, we get to the final formula of P_{new}:

$$\begin{aligned} P_{new}(\mathbf{F}) &= \mu\mathbf{F} + \lambda(\det(\mathbf{F})-\alpha)\frac{\partial \det(\mathbf{F})}{\partial \mathbf{F}} - \mu\mathbf{F}\frac{1}{\mathrm{tr}(\mathbf{F}^\mathsf{T}\mathbf{F}) + 1} \\ &= \mu\left(1 - \frac{1}{\mathrm{tr}(\mathbf{F}^\mathsf{T}\mathbf{F}) + 1}\right)\mathbf{F} + \lambda(\det(\mathbf{F})-\alpha)\frac{\partial \det(\mathbf{F})}{\partial \mathbf{F}} \end{aligned}$$

By using I_C and J, we get the following expression for $P_{new}(\mathbf{F})$:

$$P_{new}(\mathbf{F}) = \mu\left(1 - \frac{1}{I_C + 1}\right)\mathbf{F} + \lambda(J-\alpha)\frac{\partial J}{\partial \mathbf{F}}$$

A convenient shorthand for computing $\frac{\partial J}{\partial \mathbf{F}}$ is to write the expression as a result of cross products:

$$\frac{\partial J}{\partial \mathbf{F}} = \left[\mathbf{f_1} \times \mathbf{f_2} \,\middle|\, \mathbf{f_2} \times \mathbf{f_0} \,\middle|\, \mathbf{f_0} \times \mathbf{f_1} \right], \tag{3.5}$$

where $\mathbf{f_i}$ signify the column vectors of $\mathbf{F}$ defined in Eq. (2.3). This formulation is also handy when analysing $\frac{\partial^2 J}{\partial \mathbf{F}^2}$.

3.3.2 The Energy Hessian Terms

In the following, I will derive the Hessian terms of the energy. In addition, I will examine the eigenvalues and eigenvectors. This information about the Hessian is important for the optimization process because, with it, we can determine the definiteness of the matrix.

The Hessian matrix will be a (9×9)-matrix. It is demanding to keep an overview while working with high dimensional matrices. In order to not lose sight, we can write the Hessian of the energy as a fourth-order matrix-of-matrices by using the scalar notation for $\mathbf{F}$ defined in Eq. (2.3):

$$\frac{\partial^2 \Psi_{new}}{\partial \mathbf{F}^2} = \frac{\partial P_{new}(\mathbf{F})}{\partial \mathbf{F}} = \begin{bmatrix} \left[\frac{\partial P_{new}(\mathbf{F})}{\partial f_0}\right] & \left[\frac{\partial P_{new}(\mathbf{F})}{\partial f_3}\right] & \left[\frac{\partial P_{new}(\mathbf{F})}{\partial f_6}\right] \\ \left[\frac{\partial P_{new}(\mathbf{F})}{\partial f_1}\right] & \left[\frac{\partial P_{new}(\mathbf{F})}{\partial f_4}\right] & \left[\frac{\partial P_{new}(\mathbf{F})}{\partial f_7}\right] \\ \left[\frac{\partial P_{new}(\mathbf{F})}{\partial f_2}\right] & \left[\frac{\partial P_{new}(\mathbf{F})}{\partial f_5}\right] & \left[\frac{\partial P_{new}(\mathbf{F})}{\partial f_8}\right] \end{bmatrix}$$

The advantage of this form is that we can look at each entry separately, which improves readability. Each entry of the Hessian is defined as

$$\begin{aligned} \frac{\partial P_{new}(\mathbf{F})}{\partial f_i} &= \frac{\partial}{\partial f_i}\left[\mu\left(1 - \frac{1}{I_C + 1}\right)\mathbf{F} + \lambda(J - \alpha)\frac{\partial J}{\partial \mathbf{F}}\right] \\ &\overset{\text{prod.rule}}{=} \underbrace{\frac{\partial \mathbf{F}}{\partial f_i}\mu\left(1 - \frac{1}{I_C + 1}\right)}_{\mathbf{T}_i} + \underbrace{\mu\frac{2}{(I_C + 1)^2}\mathbf{F}f_i}_{\mathbf{M}_i} \\ &\quad + \underbrace{\lambda\frac{\partial J}{\partial \mathbf{F}}\frac{\partial J}{\partial f_i}}_{\mathbf{G}_i} + \underbrace{\lambda(J - \alpha)\frac{\partial^2 J}{\partial \mathbf{F}\partial f_i}}_{\mathbf{H}_i}. \end{aligned} \tag{3.6}$$

This final equation looks quite complicated. But we can split it up into these four terms: A $\mathbf{T}_i$ (Tikhonov), $\mathbf{M}_i$ (Mu), $\mathbf{G}_i$ (volume Gradient), and a $\mathbf{H}_i$ (volume Hessian) term. In the following, I will examine each of these terms separately. This way, we do not have to deal with one large and complicated expression immediately.

3.3.3 The Tikhonov, Mu, and Gradient Terms

Tikhonov

The Tikhonov term from Eq. (3.6) is a (9×9)-matrix and can be written in form of a fourth-order matrix-of-matrices. It is defined as

$$\frac{\partial \mathbf{F}}{\partial f_i} \mu \left(1 - \frac{1}{I_C + 1}\right).$$

With the explicit term for I_C, this expression resolves to

$$\frac{\partial \mathbf{F}}{\partial f_i} \mu \left(1 - \frac{1}{\mathrm{tr}(\mathbf{F}^\mathsf{T}\mathbf{F}) + 1}\right).$$

Explicitly computing the Tikhonov term is not very difficult. In order to compute this term, we need to calculate $\mathbb{T} = \frac{\partial \mathbf{F}}{\partial f_i}$. So, we have mainly entries of zeros except for the i-th entry. Therefore, $\mathbb{T}$ can be calculated by

$$\mathbb{T} = \begin{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \\ \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \\ \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \end{bmatrix}.$$

If we vectorize $\mathbb{T}$, we get the identity matrix $\mathbf{I} \in \mathbb{R}^{9x9}$, which is of full rank, positive definite, and independent of the values in $\mathbf{F}$:

$$\text{vec}(\mathbb{T}) = \check{\mathbf{T}} = \mathbf{I} = \in \mathbb{R}^{9x9}$$

In addition, the Tikhonov term serves as a diagonal regularizer for the rest of the energy.

Mu

The Mu term from Eq. (3.6) is again a (9×9)-matrix and can be written in the form of a fourth-order matrix-of-matrices. It is described by the following expression:

$$\mu \frac{2}{(I_C + 1)^2} \mathbf{F} f_i$$

With the explicit term for I_C, the Mu term resolves to

$$\mu \frac{2}{(\text{tr}(\mathbf{F}^\intercal \mathbf{F} + 1)^2} \mathbf{F} f_i.$$

In order to compute the Mu term, we need to calculate $\mathbb{M} = \mathbf{F} f_i$. That means that the i-entry is squared, and the remaining entries do not change. Therefore, $\mathbb{M}$ takes on the following form:

$$\mathbb{M} = \begin{bmatrix} \begin{bmatrix} f_0^2 & f_0 f_3 & f_0 f_6 \\ f_0 f_1 & f_0 f_4 & f_0 f_7 \\ f_0 f_2 & f_0 f_5 & f_0 f_8 \end{bmatrix} & \begin{bmatrix} f_3 f_0 & f_3^2 & f_3 f_6 \\ f_3 f_1 & f_3 f_4 & f_3 f_7 \\ f_3 f_2 & f_3 f_5 & f_3 f_8 \end{bmatrix} & \begin{bmatrix} f_6 f_0 & f_6 f_3 & f_6^2 \\ f_6 f_1 & f_6 f_4 & f_6 f_7 \\ f_6 f_2 & f_6 f_5 & f_6 f_8 \end{bmatrix} \\ \begin{bmatrix} f_1 f_0 & f_1 f_3 & f_1 f_6 \\ f_1^2 & f_1 f_4 & f_1 f_7 \\ f_1 f_2 & f_1 f_5 & f_1 f_8 \end{bmatrix} & \begin{bmatrix} f_4 f_0 & f_4 f_3 & f_4 f_6 \\ f_4 f_1 & f_4^2 & f_4 f_7 \\ f_4 f_2 & f_4 f_5 & f_4 f_8 \end{bmatrix} & \begin{bmatrix} f_7 f_0 & f_7 f_3 & f_7 f_6 \\ f_7 f_1 & f_7 f_4 & f_7^2 \\ f_7 f_2 & f_7 f_5 & f_7 f_8 \end{bmatrix} \\ \begin{bmatrix} f_2 f_0 & f_2 f_3 & f_2 f_6 \\ f_2 f_1 & f_2 f_4 & f_2 f_7 \\ f_2^2 & f_2 f_5 & f_2 f_8 \end{bmatrix} & \begin{bmatrix} f_5 f_0 & f_5 f_3 & f_5 f_6 \\ f_5 f_1 & f_5 f_4 & f_5 f_7 \\ f_5 f_2 & f_5^2 & f_5 f_8 \end{bmatrix} & \begin{bmatrix} f_8 f_0 & f_8 f_3 & f_8 f_6 \\ f_8 f_1 & f_8 f_4 & f_8 f_7 \\ f_8 f_2 & f_8 f_5 & f_8^2 \end{bmatrix} \end{bmatrix}$$

Again, f_i stand for the scalar entries of $\mathbf{F}$. When vectorizing $\mathbb{M}$, the diagonal of the resulting matrix $\check{\mathbf{M}}$ consists of the squared values of f_i:

$$\mathrm{vec}(\mathbb{M}) = \check{\mathbf{M}} = \begin{bmatrix} f_0^2 & f_1f_0 & f_2f_0 & f_3f_0 & f_4f_0 & f_5f_0 & f_6f_0 & f_7f_0 & f_8f_0 \\ f_0f_1 & f_1^2 & f_2f_1 & f_3f_1 & f_4f_1 & f_5f_1 & f_6f_1 & f_7f_1 & f_8f_1 \\ f_0f_2 & f_1f_2 & f_2^2 & f_3f_2 & f_4f_2 & f_5f_2 & f_6f_2 & f_7f_2 & f_8f_2 \\ f_0f_3 & f_1f_3 & f_2f_3 & f_3^2 & f_4f_3 & f_5f_3 & f_6f_3 & f_7f_3 & f_8f_3 \\ f_0f_4 & f_1f_4 & f_2f_4 & f_3f_4 & f_4^2 & f_5f_4 & f_6f_4 & f_7f_4 & f_8f_4 \\ f_0f_5 & f_1f_5 & f_2f_5 & f_3f_5 & f_4f_5 & f_5^2 & f_6f_5 & f_7f_5 & f_8f_5 \\ f_0f_6 & f_1f_6 & f_2f_6 & f_3f_6 & f_4f_6 & f_5f_6 & f_6^2 & f_7f_6 & f_8f_6 \\ f_0f_7 & f_1f_7 & f_2f_7 & f_3f_7 & f_4f_7 & f_5f_7 & f_6f_7 & f_7^2 & f_8f_7 \\ f_0f_8 & f_1f_8 & f_2f_8 & f_3f_8 & f_4f_8 & f_5f_8 & f_6f_8 & f_7f_8 & f_8^2 \end{bmatrix}$$

This structure makes it possible to express $\check{\mathbf{M}}$ more conveniently. We can write $\check{\mathbf{M}}$ as an outer product of $\mathrm{vec}(\mathbf{F}) = \check{\mathbf{f}}$:

$$\check{\mathbf{M}} = \mathrm{vec}(\mathbf{F})\,\mathrm{vec}(\mathbf{F})^\mathsf{T} = \check{\mathbf{f}}\check{\mathbf{f}}^\mathsf{T}$$

This matrix is of rank one and has a single non-zero eigenvalue. In order to examine the eigenvalues, we can calculate

$$\|\check{\mathbf{f}}\|_2^2 = \sum_{n=0}^{8} |f_n|^2 = \|\mathbf{F}\|_F^2 = \sum_{n=0}^{3} \sigma_i^2 = \left(\sigma_0^2 + \sigma_1^2 + \sigma_2^2\right),$$

in which $\|\cdot\|_F$ stands for the Frobenius norm introduced in Def. 2, and σ_i are the singular values from $\mathbf{\Sigma}$ in the SVD of $\mathbf{F}$ stated in Eq. (2.8). This expression can be obtained by using Eq. (2.6). The eigenvector of $\check{\mathbf{M}}$ is $\check{\mathbf{f}}/\|\check{\mathbf{f}}\|$. The eigenvalue is always non-negative and large if $\mathbf{F}$ contains a large stretch.

Volume Gradient

The volume Gradient term from Eq. (3.6) is also a (9×9)-matrix and can be written in the form of a fourth-order matrix-of-matrices. The

Gradient term is defined as

$$\lambda \frac{\partial J}{\partial \mathbf{F}} \frac{\partial J}{\partial f_i},$$

with $J = \det(\mathbf{F})$. Since λ is just a scalar constant, we are more interested in the term

$$\mathbb{G} = \frac{\partial J}{\partial \mathbf{F}} \frac{\partial J}{\partial fi}.$$

As already mentioned, we can write $\partial J/\partial \mathbf{F}$ in the form of cross products (see Eq. (3.5)) to make its computation easier. We can then set $\text{vec}(\partial J/\partial \mathbf{F}) = \check{\mathbf{g}}$ and write the vectorized matrix $\check{\mathbf{G}}$ as an outer product of $\check{\mathbf{g}}$:

$$\text{vec}(\mathbb{G})) = \check{\mathbf{G}} = \text{vec}\left(\frac{\partial J}{\partial \mathbf{F}}\right) \text{vec}\left(\frac{\partial J}{\partial \mathbf{F}}\right)^{\mathsf{T}} = \check{\mathbf{g}}\check{\mathbf{g}}^{\mathsf{T}}$$

There is again a single non-zero, non-negative eigenvalue. We can calculate it by

$$\|\check{\mathbf{g}}\|_2^2 = \left\|\frac{\partial J}{\partial \mathbf{F}}\right\|_F^2 = \left[(\sigma_0\sigma_1)^2 + (\sigma_0\sigma_2)^2 + (\sigma_1\sigma_2)^2\right].$$

The corresponding eigenvector is $\check{\mathbf{g}}/\|\check{\mathbf{g}}\|$.

3.3.4 The Volume Hessian

The volume Hessian term from Eq. (3.6) is again a (9×9)-matrix. But this time, the computation is a bit trickier. The term is described by

$$\lambda(J - \alpha)\frac{\partial^2 J}{\partial \mathbf{F} \partial f_i}$$

with $J = \det(\mathbf{F})$. The term that needs special attention is the following:

$$\mathbb{H} = \frac{\partial^2 J}{\partial \mathbf{F} \partial f_i}$$

We can write this term again in the form of a fourth-order matrix-of-matrices:

$$\mathbb{H} = \frac{\partial^2 J}{\partial \mathbf{F} \partial f_i} = \begin{bmatrix} \frac{\partial}{\partial \mathbf{F}}\left[\frac{\partial J}{\partial f_0}\right] & \frac{\partial}{\partial \mathbf{F}}\left[\frac{\partial J}{\partial f_3}\right] & \frac{\partial}{\partial \mathbf{F}}\left[\frac{\partial J}{\partial f_6}\right] \\ \frac{\partial}{\partial \mathbf{F}}\left[\frac{\partial J}{\partial f_1}\right] & \frac{\partial}{\partial \mathbf{F}}\left[\frac{\partial J}{\partial f_4}\right] & \frac{\partial}{\partial \mathbf{F}}\left[\frac{\partial J}{\partial f_7}\right] \\ \frac{\partial}{\partial \mathbf{F}}\left[\frac{\partial J}{\partial f_2}\right] & \frac{\partial}{\partial \mathbf{F}}\left[\frac{\partial J}{\partial f_5}\right] & \frac{\partial}{\partial \mathbf{F}}\left[\frac{\partial J}{\partial f_8}\right] \end{bmatrix}$$

The entries of $\mathbb{H}$ have a specific form. For example, the first entry resolves to

$$\begin{aligned} \frac{\partial}{\partial \mathbf{F}}\left[\frac{\partial J}{\partial f_0}\right] &= \frac{\partial}{\partial \mathbf{F}} \frac{\partial}{\partial f_0} \left[f_0 f_4 f_8 + f_2 f_3 f_7 + f_1 f_5 f_6 - f_2 f_4 f_6 - f_0 f_5 f_7 - f_1 f_3 f_8\right] \\ &= \frac{\partial}{\partial \mathbf{F}} \left[f_4 f_8 - f_5 f_7\right] = \begin{bmatrix} 0 & 0 & 0 \\ 0 & f_8 & -f_5 \\ 0 & -f_7 & f_4 \end{bmatrix}. \end{aligned}$$

Vectorizing this matrix reveals the vector $[0,\, 0,\, 0,\, 0,\, f_8, -f_7,\, 0, -f_5,\, f_4]^\mathsf{T}$.

By repeating this procedure, $\check{\mathbf{H}}$ reveals the structure

$$\mathrm{vec}(\mathbb{H}) = \check{\mathbf{H}} = \begin{bmatrix} 0 & 0 & 0 & 0 & f_8 & -f_7 & 0 & -f_5 & f_4 \\ 0 & 0 & 0 & -f_8 & 0 & f_6 & f_5 & 0 & -f_3 \\ 0 & 0 & 0 & f_7 & -f_6 & 0 & -f_4 & f_3 & 0 \\ 0 & -f_8 & f_7 & 0 & 0 & 0 & 0 & f_2 & -f_1 \\ f_8 & 0 & -f_6 & 0 & 0 & 0 & -f_2 & 0 & f_0 \\ -f_7 & f_6 & 0 & 0 & 0 & 0 & f_1 & -f_0 & 0 \\ 0 & f_5 & -f_4 & 0 & -f_2 & f_1 & 0 & 0 & 0 \\ -f_5 & 0 & f_3 & f_2 & 0 & -f_0 & 0 & 0 & 0 \\ f_4 & -f_3 & 0 & -f_1 & f_0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

When we look at the matrix, we can see a pattern for the entries. $\check{\mathbf{H}}$ is

of the form:

$$\check{\mathbf{H}} = \begin{bmatrix} 0 & -\widehat{\mathbf{F_2}} & \widehat{\mathbf{F_1}} \\ \widehat{\mathbf{F_2}} & 0 & -\widehat{\mathbf{F_0}} \\ -\widehat{\mathbf{F_1}} & \widehat{\mathbf{F_0}} & 0 \end{bmatrix}.$$

Matrices that have this structure are called cross-product matrices. By looking at the entries separately, we can observe that they reveal the same structure. So, each $\widehat{\mathbf{F_i}}$ is also a cross-product matrix. This property is called *self-similarity*. Therefore, $\check{\mathbf{H}}$ is a *self-similar* cross-product matrix.

Volume Hessian Eigenvalues

In the following, I will determine the eigenvalues of $\check{\mathbf{H}}$. In order to calculate the eigenvalues of a matrix, we need the corresponding characteristic polynomial. The characteristic polynomial of a square matrix $\mathbf{A}$ is defined by

$$p(\epsilon) = \det(\mathbf{A} - \epsilon\,\mathbf{I}),$$

where ϵ symbolizes the eigenvalues of $\mathbf{A}$. The eigenvalues can be calculated by setting $p(\epsilon)$ to zero ([Spe80], p. 27-28). Because $\check{\mathbf{H}}$ is a (9×9)-matrix, we expect it to have nine eigenvalues: ϵ_0, ϵ_2, ..., ϵ_8. In order to determine the eigenvalues, we can factor $\check{\mathbf{H}}$ into the following two characteristic polynomials:

$$p_1(\epsilon) = \epsilon^3 - \mathrm{tr}(\mathbf{C})\epsilon - 2J \tag{3.7}$$

$$p_2(\epsilon) = \epsilon^3 - \mathrm{tr}(\mathbf{C})\epsilon^2 + \frac{1}{2}\left(\mathrm{tr}^2(\mathbf{C}) - \mathrm{tr}(\mathbf{C}^2)\right)\epsilon - \det(\mathbf{C}) \tag{3.8}$$

First, I examine $p_2(\epsilon)$ from Eq. (3.8). It is easier to solve because it corresponds to the characteristic polynomial of $\mathbf{C}$. Given its roots ϵ_α, ϵ_β, ϵ_γ, we can calculate six of the eigenvalues of $\check{\mathbf{H}}$, namely $\pm\sqrt{\epsilon_\alpha}$, $\pm\sqrt{\epsilon_\beta}$, $\pm\sqrt{\epsilon_\gamma}$. Using the singular values of $\mathbf{F}$, these eigenvalues can be written

in the following form:

$$\begin{aligned} \epsilon_3 &= \sqrt{\epsilon_\alpha} = \sigma_0 & \epsilon_6 &= -\sqrt{\epsilon_\alpha} = -\sigma_0 \\ \epsilon_4 &= \sqrt{\epsilon_\beta} = \sigma_1 & \epsilon_7 &= -\sqrt{\epsilon_\beta} = -\sigma_1 \\ \epsilon_5 &= \sqrt{\epsilon_\gamma} = \sigma_2 & \epsilon_8 &= -\sqrt{\epsilon_\gamma} = -\sigma_2 \end{aligned} \tag{3.9}$$

The remaining eigenvalues can be obtained by using $p_1(\epsilon)$ from Eq. (3.7). This equation represents a depressed cube. A depressed cube is a cubic that can be expressed by

$$t^3 + q_1 t + q_2.$$

$p_1(\epsilon)$ can be written in this form with $q_1 = -\operatorname{tr}(\mathbf{C})$ and $q_2 = -2J$. Using this knowledge, the roots of $p_1(\epsilon)$ and therefore the remaining eigenvalues ϵ_0, ϵ_1, ϵ_2 of $\check{\mathbf{H}}$ can be obtained by

$$\epsilon_k = 2\sqrt{\frac{I_C}{3}} \cos\left[\frac{1}{3}\left(\arccos\left(\frac{3J}{I_C}\sqrt{\frac{3}{I_C}}\right) + 2\pi k\right)\right] \quad k = 0, 1, 2. \tag{3.10}$$

These are all eigenvalues of $\check{\mathbf{H}}$. Three of the six eigenvalues (ϵ_3, ..., ϵ_8) have to be negative or equal to zero since the root function yields a positive and a negative value. In addition, the cosine function for calculating ϵ_0, ϵ_1, and ϵ_2 ensures that one or two of these eigenvalues are also negative.

We have now found the eigenvalues of the terms of $\frac{\partial^2 \Psi_{new}}{\partial \mathbf{F}^2}$, described in Eq. (3.6). We found out that except for the volume Hessian term, all terms only have non-negative eigenvalues. Thus, the volume Hessian is the only source of negative eigenvalues.

In order to investigate the behaviour of the volume Hessian term further, we need to look at $J = \det(\mathbf{F})$ a bit more in detail. J is not convex, which is problematic for the optimization process. Fortunately, the other terms of $\frac{\partial^2 \Psi_{new}}{\partial \mathbf{F}^2}$ serve as an additional regularization, as already stated for the Tikhonov term.

Volume Hessian Eigenvectors

In this section, I will show the eigenvectors of $\check{\mathbf{H}}$. According to the eigendecomposition, $\check{\mathbf{H}}$ can be factorized as $\check{\mathbf{H}} = \check{\mathbf{Q}}\mathbf{\Lambda}\check{\mathbf{Q}}^\mathsf{T}$. We can obtain

the eigenvectors by computing $\check{\mathbf{Q}}$. To be consistent with the procedure so far, I will use the tensor form symbolized by $\mathbb{Q}$:

$$\mathbb{Q} = \begin{bmatrix} [\mathbf{Q}_0] & [\mathbf{Q}_3] & [\mathbf{Q}_6] \\ [\mathbf{Q}_1] & [\mathbf{Q}_4] & [\mathbf{Q}_7] \\ [\mathbf{Q}_2] & [\mathbf{Q}_5] & [\mathbf{Q}_8] \end{bmatrix}$$

Each entry of $\check{\mathbf{Q}}$ is an eigenvector in the form of a (3×3)-matrix instead of a vector with nine entries. I am starting with the eigenvalues obtained by Eq. (3.8), namely ϵ_3, ϵ_4, ..., ϵ_8. The eigenvectors corresponding to these eigenvalues can all be written in the following form:

$$\mathbf{Q}_k = \frac{1}{\sqrt{2}}\mathbf{U}\mathbf{D}_k\mathbf{V}^\mathsf{T} \qquad \text{for } k = 3, 4, ..., 8 \tag{3.11}$$

$\mathbf{U}$ and $\mathbf{V}$ are taken from the SVD of $\mathbf{F}$, and $\frac{1}{\sqrt{2}}$ is a normalization factor. The difference of the eigenvectors lies in the matrix $\mathbf{D}_k$. For each eigenvalue ϵ_k for $k \in 3, 4, ..., 8$, $\mathbf{D}_k$ is defined as:

$$\mathbf{D_3} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix} \qquad \mathbf{D}_6 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

$$\mathbf{D_4} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ -1 & 0 & 0 \end{bmatrix} \qquad \mathbf{D}_7 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

$$\mathbf{D_5} = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad \mathbf{D}_8 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

These eigenvectors have a unique *pseudo-cross-product* structure. $\mathbf{D_3}$ is a cross-product matrix for the basis vector $[-1, 0, 0]^\mathsf{T}$. In addition, $\mathbf{Q}_6$ differs from $\mathbf{Q}_3$ only because of one sign. Therefore, $\mathbf{D}_3$ has been multiplied by a reflection, which makes $\mathbf{Q}_6$ a reflected pseudo-cross-product matrix. $\mathbf{Q}_4$ and $\mathbf{Q}_5$ are in the same way cross-product matrices for the

basis vectors $[0,-1,0]^\mathsf{T}$ and $[0,0,-1]^\mathsf{T}$. $\mathbf{Q}_7$ and $\mathbf{Q}_8$ are the corresponding reflected pseudo-cross-product matrices.

The remaining eigenvectors correspond to the eigenvalues ϵ_0, ϵ_1, and ϵ_2 derived from the depressed cubic. They are of the form

$$\mathbf{Q}_k = \frac{1}{\|\mathbf{D}_k\|_F}\mathbf{U}\mathbf{D}_k\mathbf{V}^\mathsf{T} \qquad \text{for } k = 0,1,2.$$

$\mathbf{U}$ and $\mathbf{V}$ are again taken from the SVD of $\mathbf{F}$, and $\frac{1}{\|\mathbf{D}_k\|_F}$ is the normalization factor. $\mathbf{D}_k$ is a diagonal matrix and is defined as

$$\mathbf{D}_k = \begin{bmatrix} \sigma_0\sigma_2 + \sigma_1\epsilon_k & 0 & 0 \\ 0 & \sigma_1\sigma_2 + \sigma_0\epsilon_k & 0 \\ 0 & 0 & \epsilon_k^2 - \sigma_2^2 \end{bmatrix} \qquad \text{for } k = 0,1,2.$$

Now we have found explicit expressions for all eigenvalues and eigenvectors of the volume Hessian term.

3.3.5 The Complete Eigensystem

Now it is time to analyse the complete system, which will be called $\check{\mathbf{A}}$ in the following. From Eq. (3.6), we know that $\check{\mathbf{A}}$ is defined as

$$\check{\mathbf{A}} = \mu\left(1 - \frac{1}{I_C+1}\right)\frac{\partial \mathbf{F}}{\partial f_i} + \mu\frac{2}{(I_C+1)^2}\mathbf{F}f_i + \lambda\frac{\partial J}{\partial \mathbf{F}}\frac{\partial J}{\partial f_i} + \lambda(J-\alpha)\frac{\partial^2 J}{\partial \mathbf{F}\partial f_i}.$$

With the expressions for each individual term that were derived in the last section, we can write $\check{\mathbf{A}}$ as

$$\check{\mathbf{A}} = \mu\left(1 - \frac{1}{I_C+1}\right)\mathbf{I} + \mu\frac{2}{(I_C+1)^2}\check{\mathbf{f}}\check{\mathbf{f}}^\mathsf{T} + \lambda\check{\mathbf{g}}\check{\mathbf{g}}^\mathsf{T} + \lambda(J-\alpha)\check{\mathbf{H}}.$$

Computing the eigenvalues from a sum of matrices is nontrivial. Fortunately, $\check{\mathbf{A}}$ has a special structure that can be used to obtain the expressions for the eigenvalues and eigenvectors. We can use the knowledge of the eigenvalues and eigenvectors of the individual terms that we have gathered before. For simplification, I do not include the calculations here and present only the results. The core goal of this section is to show that

we can compute the eigenvalues and eigenvectors relatively simple. For further information about the exact steps, an interested reader can have a look at the paper *SNH-FS* on p. 12:7 and 12:8.

For the final eigenvalues, we can use the regularization term $\mu_T = \mu(1 - \frac{1}{I_C+1})$. The first three eigenvalues can be calculated by

$$\epsilon_k = \lambda(J-\alpha)\bar{\epsilon}_k + \mu_T \qquad \text{for } k = 0, 1, 2.$$

In this expression, $\bar{\epsilon}_k$ are the roots of the equation

$$\bar{\epsilon}^3 + c_2\bar{\epsilon}^2 + c_1\bar{\epsilon} + c_0 = 0.$$

The variables c_0, c_1, and c_2 are defined as

$$\begin{aligned}
c_2 &= -\|\check{\mathbf{g}}\|_2^2\rho - I_C\eta \\
c_1 &= -(1+2J\rho)I_C - 6J\eta + \left(\|\check{\mathbf{g}}\|_2^2 I_C - 9J^2\right)\rho\eta \\
c_0 &= -(2+3J\rho)J + \left(I_C^2 - 4\|\check{\mathbf{g}}\|_2^2\right)\eta + 2J\left(I_C^2 - 3\|\check{\mathbf{g}}\|_2^2\right)\rho\eta
\end{aligned}$$

with

$$\eta = \frac{2\mu}{(I_C+1)^2\left(\lambda(J-1) - \frac{3}{4}\mu\right)}, \qquad \rho = \frac{\lambda}{\lambda(J-1) - \frac{3}{4}\mu}.$$

The remaining eigenvalues can be obtained by:

$$\begin{aligned}
\epsilon_3 &= \lambda(J-\alpha)\sigma_0 + \mu_T & \epsilon_6 &= -\lambda(J-\alpha)\sigma_0 + \mu_T \\
\epsilon_4 &= \lambda(J-\alpha)\sigma_1 + \mu_T & \epsilon_7 &= -\lambda(J-\alpha)\sigma_1 + \mu_T \\
\epsilon_5 &= \lambda(J-\alpha)\sigma_2 + \mu_T & \epsilon_8 &= -\lambda(J-\alpha)\sigma_2 + \mu_T
\end{aligned}$$

These are all eigenvalues of this system. We can now focus on the eigenvectors. The eigenvectors corresponding to ϵ_0, ϵ_1, and ϵ_2 can be calculated by

$$\mathbf{Q}_k = \frac{1}{\|\mathbf{D}_k\|_F}\mathbf{U}\mathbf{D}_k\mathbf{V}^\mathsf{T} \qquad \text{for } k = 0, 1, 2.$$

We have already seen this structure for the calculation of the eigenvectors

of the volume Hessian term. But this time, $\mathbf{D}_k$ is defined differently:

$$\mathbf{D}_k = \begin{bmatrix} \alpha_0 & 0 & 0 \\ 0 & \alpha_1 & 0 \\ 0 & 0 & \alpha_2 \end{bmatrix} \qquad \text{for } k = 0, 1, 2.$$

The diagonal entries of $\mathbf{D}_k$ are the following:

$$\begin{aligned}
\alpha_0 = \; & \bar{\epsilon}_k \left(\sigma_1 + \sigma_0\sigma_2\eta + J\sigma_1\rho\right) \\
& + \sigma_0\sigma_2 + \sigma_1\left(\sigma_0^2 - \sigma_1^2 + \sigma_2^2\right)\eta + J\sigma_0\sigma_2\rho \\
& + \sigma_0\left(\sigma_0^2 - \sigma_1^2\right)\sigma_2\left(\sigma_1^2 - \sigma_2^2\right)\rho\,\eta \\
\alpha_1 = \; & \bar{\epsilon}_k \left(\sigma_0 + \sigma_1\sigma_2\eta + J\sigma_0\rho\right) \\
& + \sigma_1\sigma_2 - \sigma_0\left(\sigma_0^2 - \sigma_1^2 - \sigma_2^2\right)\eta + J\sigma_1\sigma_2\rho \\
& - \sigma_1\left(\sigma_0^2 - \sigma_1^2\right)\sigma_2\left(\sigma_0^2 - \sigma_2^2\right)\rho\,\eta \\
\alpha_2 = \; & \bar{\epsilon}_k^2 - \bar{\epsilon}_k\left(\sigma_0^2 + \sigma_1^2\right)\left(\eta + \sigma_2^2\rho\right) \\
& - \sigma_2^2 - 2J\eta - 2J\sigma_2^2\rho + \left(\left(\sigma_0^2 - \sigma_1^2\right)\sigma_2\right)^2\rho\,\eta
\end{aligned}$$

The remaining six eigenvectors from ϵ_3, ϵ_4, ..., ϵ_8 are the same as we saw for the volume Hessian and can be calculated by Eq. (3.11). With this, we have found explicit expressions for all of the eigenvalues and eigenvectors of the complete system.

3.3.6 Conclusion

In subsection 3.3.4, we can see by the structure of Eq. (3.9) and Eq. (3.10) that it is possible to factor the Hessian into one (3×3)-eigensystem. This system is determined by the three pair-wise roots and the three entangled roots for the eigenvalues. That extends the findings from Teran et al. in [Ter05]. Furthermore, we saw that explicit expressions for the eigenvalues and eigenvectors of each Hessian term, as well as for the whole system, can be formed. That is important for the optimization process.

With this analysis, we can conclude that the proposed energy is complex enough to capture a desired level of detail but simple enough to be

expressed by closed-form expressions. In the next chapter, I will test the robustness of the model with various scenarios and check whether the claims of the authors hold or not.

Chapter 4

Practical Experiments

After the derivation of the novel deformation energy in chapter 3, this chapter contains some practical experiments to show the robustness of the model and check that each requirement stated in subsection 3.2.1 is satisfied. The experiments are quasi-static simulations of physical deformations. So, we deform an object in multiple small steps. Quasi-static simulation are well suited for flesh simulations ([Ter05]). These simulations are done by posing constraints over the object and then minimizing the deformation energy to find the final position of each vertex.

The authors of the paper *SNH-FS* already provided an implementation for an application of their formulated energy[1]. Firstly in this chapter, I will examine their implementation and show the results of the experiments with their code. Later, I will present the results of other scenarios, for which I had to change the code accordingly. In the end, I will include a discussion of this new energy formulation considering the results from this chapter.

4.1 Technology

The code the authors provided was written in C++ using the CMake build system[2]. It uses the library Eigen ([GJ10]), version 3.1.2 or newer.

[1] available at http://graphics.pixar.com/library/StableElasticity/snh_code.tar.bz2
[2] https://cmake.org/

The code consists of a core library `cubesim`, which provides the 3D mesh data structure, an implementation of the Stable Neo-Hookean material model, and Newton solvers that minimize the deformation energy over the meshes.

The images in this chapter were taken with the help of OpenFlipper, which is an open-source geometry processing and rendering framework ([MK10]). Furthermore, I used OpenFlipper to create a plugin for some of the experiments. Inside the plugin, I used the integrated library OpenVolumeMesh, which provides a data structure to handle arbitrary polyhedral meshes ([KBK13]).

4.2 First Experiments

In this section, I will show the results of the experiments I did with the provided code. The implementation performs a quasi-static simulation of the stretching of a cube. The cube can either be represented by a tetrahedral or a hexahedral mesh. In order to simulate the deformation, the implementation needs a value for each of the two Lamé parameters and a value for defining the desired resolution as input data. In addition, the user has to specify the directory into which the output files should be saved and the material model that should be selected. Since the simulation is quasi-static, the deformation is subdivided into 25 small steps instead of one large step. The output files are 26 static objects in the format .obj. The first file shows the object in its rest state, and the remaining files illustrate the 25 steps of increased deformation. The procedure of the implementation is roughly illustrated in Fig. 4.1.

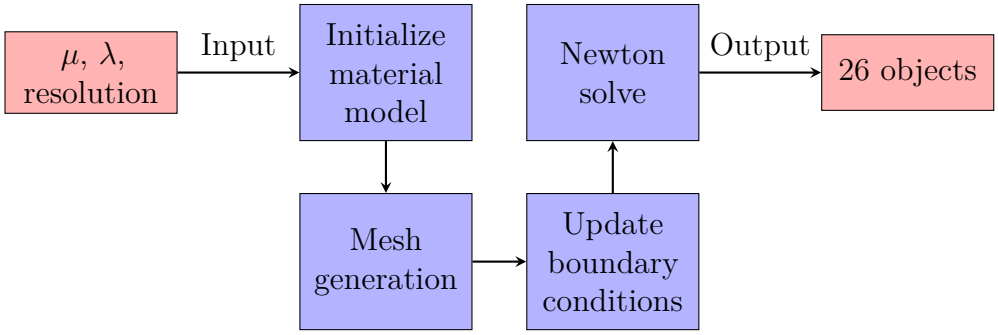


Figure 4.1: Illustration of the procedure

I will explain how the implementation works based on an example. For this, I am taking the input variables $\mu = 1.0$, $\lambda = 10.0$ and a resolution of 10.0. These values for the Lamé parameters result in a Poisson's ratio of 0.46. The command to start the execution for a tetrahedral mesh with these input variables is shown in Code snippet 4.1.

```
$ ./tetcli 10 stable_neo_hookean 1.0 10.0 output
```

Code snippet 4.1: Bash command for executing the code

Firstly, the implementation initializes the material model Stable Neo-Hookean. Then the generation of the mesh starts. The code creates a cube with 11 vertices (calculated by resolution + 1) in each axis. Hence, the cube consists of $11 * 11 * 11 = 1'331$ vertices in total. The number of hexahedra, respectively tetrahedra, are calculated by the following formulas:

$$\text{Tet count} = 6 * 10 * 10 * 10 = 6'000$$
$$\text{Cube count} = 10 * 10 * 10 = 1'000$$

Afterwards, the simulation with the 25 steps of the deformation starts. The code sets two faces of the cube as a boundary condition. A boundary condition in this context denotes a set of fixed vertices. A boundary face is a face for which each vertex is fixed. Fixed vertices already have their final position for the result of the deformation. By updating the boundary conditions, we stretch the cube. That means that these two boundary faces are moved further away from each other along the y-axis. We do this by updating the y-value of the fixed vertices with a new value `newNegativeBoundary` or `newPositiveBoundary` depending on whether the vertex should move in the positive or negative direction along the y-axis. The definition of these two values is shown in Code snippet 4.2. The variable `stepNum` is the value of the current step from the 25 deformation steps. For each step the stretching should increase. To achieve this, we use the variable `stepDelta`. Its value is equal to 0.1 and stays constant for these first experiments.

```
const double newNegativeBoundary = -1.0 - stepNum * ↩
    stepDelta;
```

```
const double newPositiveBoundary = 1.0 + stepNum * stepDelta;
```

Code snippet 4.2: Updating y-coordinates of vertices

After the new positions of the fixed vertices are declared, the `TetNewtonSolver`, respectively `CubeNewtonSolver`, is used to minimize the strain energy over the mesh and determine the position of the remaining vertices. The optimization process will be further explained in section 4.4. Finally, the mesh is saved in a .obj file in the directory `output`.

Tables 4.1 and 4.2 show, the total number of Newton iterations that had to be performed by the solver to reach an acceptable solution. The number of iterations are listed for a specific step number or an interval of step numbers. For example, step number 1 to 10 each required 3 Newton iterations. The step number denotes in which of the 25 steps of deformation we currently are. In Code snippet 4.2 I called this variable `stepNum`. The amount of iterations increases with increasing step number. This increase is not surprising, since the deformation is also increased with each step resulting in more calculations.

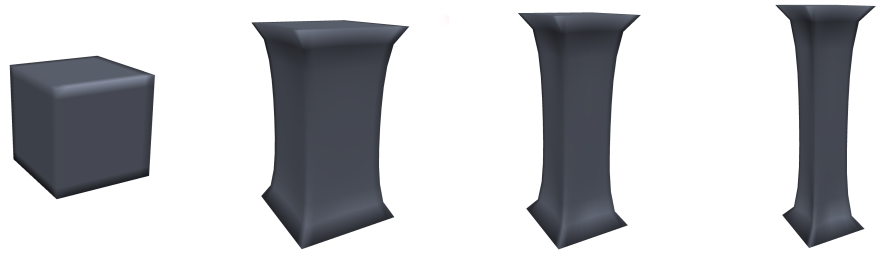
Step number	**Iterations**
1-10	3
11-21	4
22-25	5

Table 4.1: Newton iterations for a tetrahedral mesh

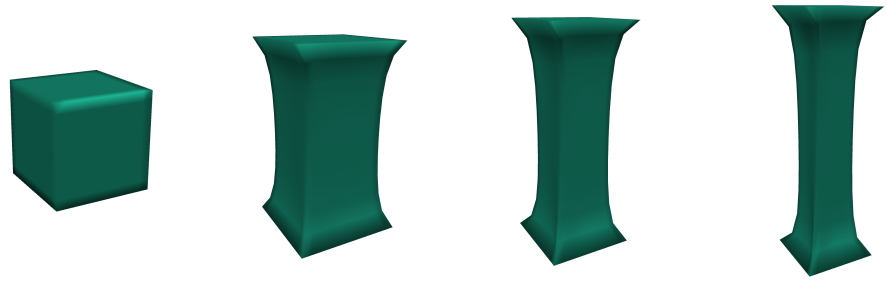
Step number	**Iterations**
1-7	4
8-14	5
15-18	6
19-22	7
23-25	8

Table 4.2: Newton iterations for a hexahedral mesh

The images in Fig. 4.2 show four of the resulting objects of this example. The choice of the input parameters is the same for each object. The image shows step 0, which contains the cube in its rest state, step 8, 16, and 24 for a tetrahedral and a hexahedral mesh. The resulting objects capture the deformation well and are artefact-free even for a larger stretch. Both the hexahedral and tetrahedral mesh show equally good results. Hence, the model performs well without restricting us to use only a certain type of mesh.



(a) Stretch test on a hexahedral mesh



(b) Stretch test on a tetrahedral mesh

Figure 4.2: Stretch test performed on a cube with (a) a hexahedral mesh and (b) a tetrahedral mesh with the same parameters

4.2.1 Comparison of different Input Variables

In order to examine the influence of the input variables, I experimented with different values. The importance of the resolution is straight forward: With a higher resolution, the cube consists of more vertices and hexahedra or tetrahedra. In conclusion, the results are smoother, but we also increase the computational costs. For this example, I choose a resolution of 30 and let the other variables stay the same as in the previous example, meaning $\lambda = 10.0$ and $\mu = 1.0$, in order to make a comparison possible. If we choose a resolution of 30, the cube consists of 29′791 vertices and 162′000 tetrahedra, respectively 27′000 hexahedra. Thus, the cube consists of 27 times more cells than in the previous example.

Tables 4.3 and 4.4 show the amount of Newton iterations until a solution is reached. If we compare the numbers from this example with the ones from Table 4.1 and 4.2, we can see that there are slightly more iterations needed than in the previous example. This increase can be expected due to the larger number of vertices and cells that the solver needs to

consider.

Step number	**Iterations**
1-7	3
8-13	4
14-20	5
21-25	6

Table 4.3: Newton iterations for a tetrahedral mesh (res=30)

Step number	**Iterations**
1-4	4
5-10	5
11-15	6
16-19	7
20-23	8
24-25	9

Table 4.4: Newton iterations for a hexahedral mesh (res=30)

Fig. 4.3 shows the results with a higher resolution of 30 compared to the lower resolution of 10 of the previous example on a tetrahedral mesh. It shows the results of the deformation step 25. We can see that a higher resolution clearly gives us more realistic results. Analogous, the same conclusion applies to the hexahedral mesh.

(a) Step 25 with a resolution of 10 (b) Step 25 with a resolution of 30

Figure 4.3: Results with a different resolution on a tetrahedral mesh

In addition, we can increase or decrease the Lamé parameters μ and λ, which influences the Poisson's ratio directly. If we decrease λ, the Poisson's ratio also decreases. We can set λ equal to 1.0 and let the other variables stay the same as in the previous example. Thus, μ is equal to 1.0, and the resolution is set to 30.0. The Poisson's ratio indicates the extent of the deformation. Materials with a higher value are more easily deformable, which I explained in subsection 2.2.4. Therefore, by

decreasing the Poisson's ratio, the extent of the deformation is also decreased. That affects the resulting object as well as the computational costs. Tables 4.5 and 4.6 show the amount of Newton iteration needed for the chosen input values. The amount of iterations has decreased for both meshes. We need fewer iterations because the extent of the deformation is smaller than before.

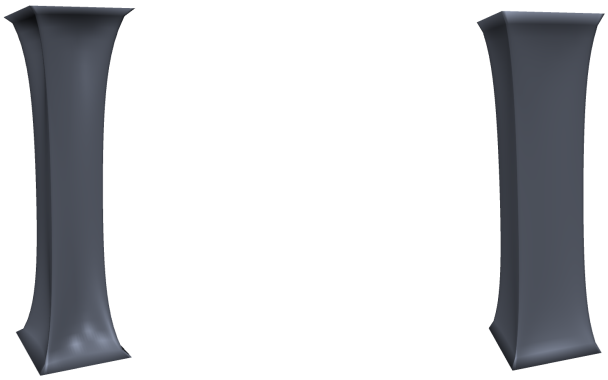
Step number	**Iterations**
1-25	3

Table 4.5: Newton iterations for a tetrahedral mesh ($\lambda = 1.0$)

Step number	**Iterations**
1-25	4

Table 4.6: Newton iterations for a hexahedral mesh ($\lambda = 1.0$)

Fig. 4.4 shows how the deformation is influenced by changing λ. With these inputs, the cube gets wider in the middle part, as a lower Poisson's ratio indicates that the material is more resistant to stretching.



(a) Step 25 with $\lambda = 10.0$ (b) Step 25 with $\lambda = 1.0$

Figure 4.4: Results with different values for λ on a hexahedral mesh

Now instead of decreasing the Poisson's ratio, let's increase it. We can achieve that by choosing $\mu = 0.1$ and set λ equal to 10.0 and let the resolution be 30. This is an extreme case because the Poisson's ratio is very close to 0.5. It is also an appropriate value for simulating flesh behaviour. Tables 4.7 and 4.8 illustrate the amount of Newton iterations needed for this configuration. As we can see, we need more Newton iterations to solve the system. That is caused by the increased deformation,

as the cube gets narrower in the middle part, which makes reaching a good solution more difficult.

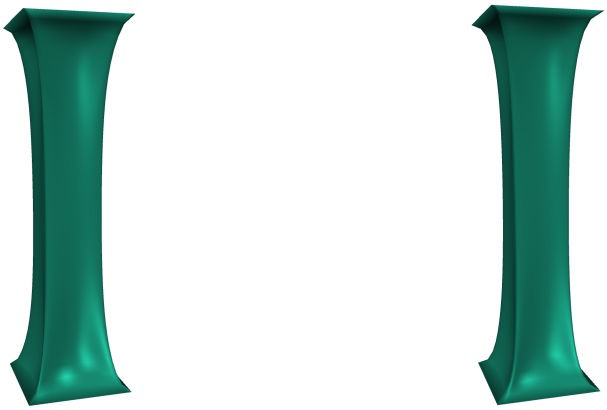
Step number	**Iterations**
1	3
2-3	4
4-6	5
7	6
8-11	7
12	8
13-20	9
21-25	11

Table 4.7: Newton iterations for a tetrahedral mesh ($\mu = 0.1$)

Step number	**Iterations**
1	6
2	7
3	8
4	9
5-6	10
7	11
8-9	12
10-11	13
12	14
13-14	15
15	16
16	17
17-19	18
20-25	20

Table 4.8: Newton iterations for a hexahedral mesh ($\mu = 0.1$)

A higher Poisson's value makes the cube more susceptible to stretching. We can see this effect a bit by comparing the two images in Fig. 4.5, although the difference is not extreme because the Poisson's ratios for the two examples are close to each other. The first image shows a material with a Poisson's ratio of approximately 0.46, and the second image shows a material with a Poisson's ratio approaching 0.5. The model behaves well with these input values, and no artefacts can be seen.



(a) Step 25 with $\mu = 1.0$ (b) Step 25 with $\mu = 0.1$

Figure 4.5: Results with different values for μ on a tetrahedral mesh

4.3 Additional Tests

For this section, I changed some parts of the given implementation to perform more tests for this new energy. The first thing I did was to change the code, such that I was able to perform a quasi-static simulation of the stretching of an arbitrary mesh. Similar to the provided code, there are again 25 deformation steps. The implementation expects as input a tetrahedral or hexahedral mesh in the format .ovm (OpenVolumeMesh). In the user interface, several values can be selected:

- Mesh type: Describes the cell shape of the mesh. The cell shape can be either *tetrahedral*, *regular hexahedral*, or *irregular hexahedral*.
- Lambda: One of the Lamé parameters that determine the Poisson's ratio.
- Mu: One of the Lamé parameters that determine the Poisson's ratio.
- Stretch axis: Axis along which the mesh is supposed to be stretched. This can be the x-, y-, or z-axis.
- Coordinate value for the first boundary: Determines the position at which the first boundary face lies.
- Coordinate value for the second boundary: Determines the position at which the second boundary face lies.

- Rate of step size: Rate at which the object should be stretched along the axis.

The rate of step size corresponds to the variable `stepDelta` in Code snippet 4.2. I made this variable adjustable because we are dealing with arbitrary meshes that could potentially be very large or very small. For better results, we might want to change the rate at which we stretch the object.

In order to get the desired results, I had to adjust the code accordingly. Fig. 4.6 illustrates the modified procedure of the code. The green fields are the steps that are different from the procedure before. Although the writing of the file was necessary before, it is explicitly listed here because the output format of the file has changed. In addition, the general procedure of updating the boundary conditions also stayed the same, but I had to adjust some parts so that the user input is processed correctly.

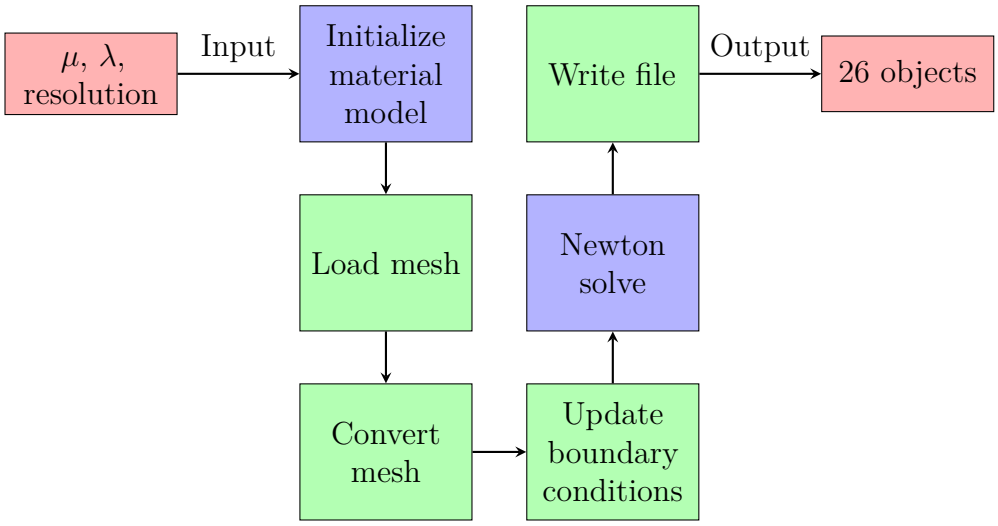


Figure 4.6: Illustration of the modified procedure

The initialization of the material model stays the same as before. But now the implementation needs to load the mesh the user provides into the system. It then converts the input mesh into the data structure that was used by the provided code. Additionally, if the user input states that a hexahedral mesh is irregular, each hexahedron gets subdivided into four tetrahedra. This step was necessary because the subsequent steps expect the hexahedral mesh to be regular. Then, the boundary conditions are

updated according to the input of the user. The stretching is again done by putting the boundary faces further away from each other along the stretch axis. Afterwards, the Newton solver calculates the final position of each vertex that is not fixed. Finally, the mesh is written into a file of the format .ovm (OpenVolumeMesh).

4.3.1 Cylinder

With the adjusted code, I was able to recreate the example of stretching a cylinder. This example was shown in the paper SNH-FS, and the authors claimed that their model can handle this deformation well, even with a high Poisson's ratio and no parameter tuning. For my experiments, I used a hexahedral mesh consisting of $1'290$ vertices and $1'036$ hexahedra. Similar to the previous examples of the stretching of a cube, Fig. 4.7 shows step 0 (initial cylinder), 8, 16, and 24 of the deformation. For the parameters, I chose $\lambda = 10.0$, $\mu = 1.0$, and a step size of 0.1, in order to reach a high Poisson's ratio and to get a large stretch of the object.

Figure 4.7: Stretch test on a cylinder

The images verify the claims of the authors. The deformation is captured well and no artifacts can be seen, even for a larger stretch. Therefore, we can assume that we are not restricted to a specific shape of the deformable object. In addition, Table 4.9 illustrates the amount of Newton iterations that were necessary for solving the system. The numbers are a bit higher for a larger stretch compared to the cube, which consists of a similar amount of hexahedra. But they are still in a reasonable interval.

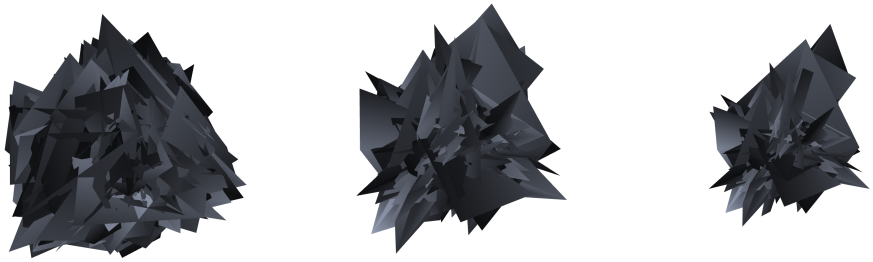
Step number	**Iterations**
1	2
2-7	3
8-12	4
13-14	5
15-16	6
17-18	7

Step number	**Iterations**
19-20	8
21	9
22	10
23-24	11
25	13

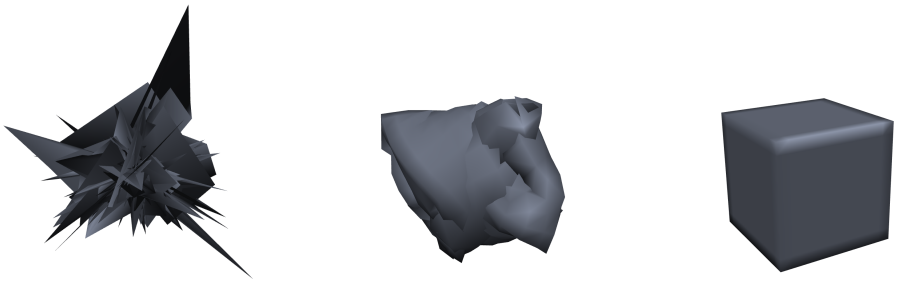
Table 4.9: Newton iterations for the deformation of a cylinder

4.3.2 Scramble Test

In order to highlight the robustness of the model, the authors performed a scramble test similar to those in Teran et al. ([Ter05]) and Stomakin et al. ([Sto12]). For this test, they randomly placed the vertices of a cube within a space of twice its rest volume. This test illustrates how the model performs under extreme, inverted configurations. Fig. 4.8 shows the results of the scramble test I performed.



(a) Results after 0, 1, and 2 Newton iterations (from left to right)



(b) Results after 10, 40, and 63 Newton iterations (from left to right)

Figure 4.8: Scramble test on a cube

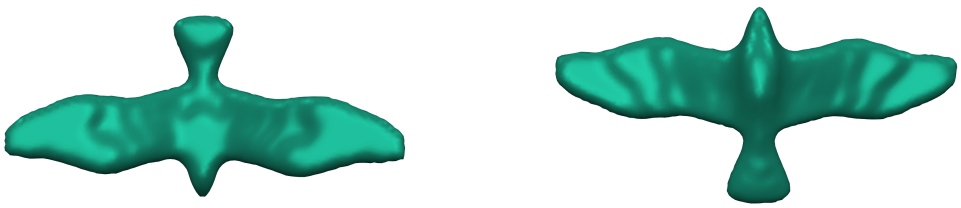
In Fig. 4.8, we can see the solution process for 0, 1, 2, 10, 40, and finally 63 Newton iterations with $\mu = 1.0$ and $\lambda = 10.0$. For this example, I had to alter the code a bit to get the desired functionalities. The initial hexahedral mesh is a cube consisting of $1'331$ vertices and $1'000$ hexahedra. Just like the authors, I fixed only four corner vertices and scrambled the remaining vertices of the cube within a space of twice the rest volume.

The last image of Fig. 4.8 shows the cube back in its rest state. Hence, the model is able to recover from this extreme deformation after 63 Newton iterations. In conclusion, the model has rest stability even under these inverted configurations.

4.3.3 Experiments on a more complex Mesh

After the experiments with simple meshes as the cube and cylinder, it would be interesting to see how the energy performs on a more complex mesh. For the following two experiments, I chose a tetrahedral mesh consisting of $5'650$ vertices and $20'203$ tetrahedra. The mesh is formed in the shape of a bird. Fig. 4.9 shows the object in its rest state from different angles.

(a) Front and back of the mesh (from left to right)

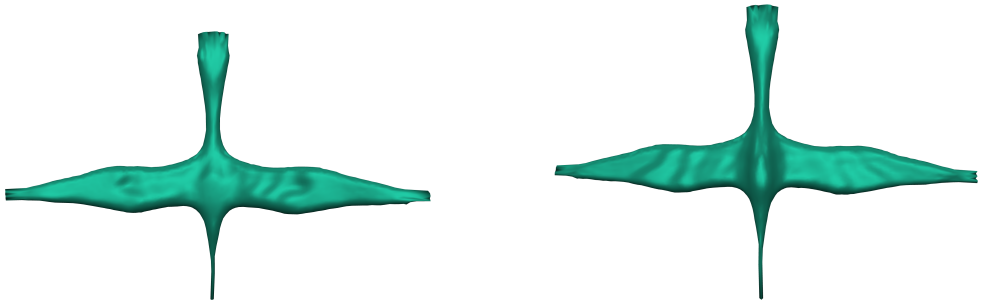


(b) Top and bottom of the mesh (from left to right)

Figure 4.9: Object in its rest state shown from different angles

For the first deformation, I stretched the bird in four directions: along the left and right wing, the head, and tail. That will give us an extreme stretch to further test the energy. For the parameters, I set λ equal to 10.0, μ equal to 1.0, and the step size equal to 0.01. The result of this deformation after ten steps is shown in Fig. 4.10.

(a) Front and back of the mesh (from left to right)



(b) Top and bottom of the mesh (from left to right)

Figure 4.10: Stretched object shown from different angles

As we can see, the wings lost their bend with this extreme stretch, and the head has become almost a straight line. Overall, the deformed object looks as good as it can under these extreme conditions, and the energy behaves well. There are some places where we can imagine which vertices I pulled to perform the stretch. For example, Fig. 4.11 shows how the outer part of the wings form almost a zig-zag pattern. Unfortunately, that was unavoidable because of the configuration of the mesh. If the vertices were aligned in a straight line, this effect would not be seen.

Figure 4.11: Irregularity on the right wing (from the bird's point of view)

Additionally, instead of just stretching the object, we should also test how the energy behaves for other deformations. For the next experiment with this mesh, I pushed the two wings together and pulled the body down. With this configuration, we can bend the wings and see how the energy behaves under these conditions. Again, I set λ equal to 10.0, μ equal to 1.0, and the step size equal to 0.01 Fig. 4.12 shows the deformed object after ten steps.

(a) Front and back of the mesh (from left to right)

(b) Top and bottom of the mesh (from left to right)

Figure 4.12: Bent object shown from different angles

In the images, we can see that the two wings bend differently, which is caused by the selection of vertices to pull. Fig. 4.13 shows the deformed object from both sides to get a better look at the deformation.

Figure 4.13: Side view of bent object

For both cases, the results are good, and no artefacts can be seen. With these experiments, we can conclude that the energy behaves well even for larger, more complex meshes. In addition, we have now seen different deformations than just the stretching along one axis of the object, and the energy was able to handle these configurations. That suggests that the energy is suited for arbitrary deformations on arbitrary meshes.

4.4 Optimization

In order to achieve realistic results, we not only need to choose a suitable model but also an appropriate algorithm to get the exact solution or a satisfying approximation within a reasonable amount of time. For the performed simulations, a standard Newton solver augmented with a line search was used. Linear systems were solved by using conjugate gradient implementation. This section gives an overview of these methods.

4.4.1 Newton's Method

The task of the optimization, in this case, is to find an acceptable minimum of the strain energy after changing the boundary conditions. We can formulate this problem more generally as

$$\text{minimize} \quad f(x) : \mathbb{R}^n \to \mathbb{R}.$$

In order to find a minimum, the condition $\nabla f(x) = 0$ has to be satisfied. For solving this problem, we can use Newton's method, which iteratively approaches an acceptable minimum. To achieve this, we need to know the correct search direction to approach a minima step by step. The search direction is given by the Newton step, which uses the second derivative of the function f:

$$\nabla x_{nt} := -\nabla^2 f(x)^{-1} \nabla f(x)$$

Now we need to know how much we can move in this direction. The line search algorithm can answer this question. It outputs the maximum amount to move in a given search direction. Newton's method can be very fast compared to other methods and produces a solution of high

accuracy. The main disadvantage is that the calculation and storage of the second derivative and computing the Newton step can be quite costly ([BBV04], p. 484-487, 496). Thus, the analysis of the Hessian of the deformation energy was a necessary step to show that the energy is suited for practical applications.

4.4.2 Conjugate Gradient

The conjugate gradient method is an algorithm that solves a system of linear equations. We can write the problem generally as

$$\mathbf{A}\mathbf{x} = \mathbf{k},$$

where $\mathbf{A}$ is a symmetric, positive definite $(n \times n)$-matrix. Both $\mathbf{A}$ and $\mathbf{k}$ are known. The conjugate gradient method delivers an approximation of the exact solution within a certain tolerance. It iteratively searches for an appropriate $\mathbf{x}$, starting with an initial estimate. It usually reaches convergence fast and is computationally efficient ([HS52], p. 409-412). In the code, the conjugate gradient method was used iteratively with the help of the library Eigen, which has already implemented the corresponding functionalities.

4.5 Discussion

The authors of the paper SNH-FS introduced a novel hyperelastic model for the deformation energy. My first approach for this thesis was to dive into the topic of flesh simulation. The most important topics concerning this learning process are introduced and explained in chapter 2. The next step was to understand and being able to explain the thought process of the authors. The paper itself can be a bit complicated for somebody who does not yet have a broad knowledge of the field. I documented this step in chapter 3. In the same chapter, it also appears to be clear why a new energy formulation was necessary. None of the existing energies satisfied all of the requirements stated in subsection 3.2.1. Some of them even produced severe artefacts or could not preserve the volume. Finally,

I conducted some experiments on my own to verify some of the author's claims. The implementation the authors provided helped me during this process, not only for the simulations but also to get a better understanding of the energy. In chapter 4, I first experimented with different values for the Poisson's ratio and was able to verify the claim that the model behaves well for a wide range of Poisson's ratio, even for one close to 0.5. In addition, I could recreate some other examples they covered in their paper. For each simulation included in this thesis, the model satisfies the requirements the authors listed without the need of using filter parameters. Furthermore, the scramble test validates the robustness of the model, and the deformations on the bird mesh suggest that the energy is suited for arbitrary deformations on arbitrary meshes. But in order to make a final statement of the quality of the energy, it would be necessary to make more specific tests, for example, the behaviour of a hand or an arm. In addition, the results should be compared with the ones produced from other energy formulations, similar to some of the tests the authors included in their paper. Unfortunately, this would have been beyond the scope of this thesis. From the findings I could require during my thesis, it nonetheless appears that there is a great improvement in the quality of the simulation with this new energy model for materials with a high Poisson's ratio.

List of Figures

List of Tables

Code

List of abbreviations

PK1 *first Piola-Kirchhoff stress tensor* 20, 23

SNH-FS *Stable Neo-Hookean Flesh Simulation* . . . 2–4, 15, 21, 30, 31

SVD *singular value decomposition* 11–13, 16, 26, 29

UTS *ultimate tensile strength* 9

Bibliography

[BBV04] Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[Ber15] Jörgen Bergström. "5 - Elasticity/Hyperelasticity". In: *Mechanics of Solid Polymers*. Ed. by Jörgen Bergström. William Andrew Publishing, 2015, pp. 209–307. ISBN: 978-0-323-31150-2. DOI: https://doi.org/10.1016/B978-0-323-31150-2.00005-4. URL: http://www.sciencedirect.com/science/article/pii/B9780323311502000054.

[Bow09] Allan F Bower. *Applied mechanics of solids*. CRC press, 2009.

[BW97] Javier Bonet and Richard D Wood. *Nonlinear continuum mechanics for finite element analysis*. Cambridge university press, 1997.

[For14] William Ford. *Numerical linear algebra with applications: Using MATLAB*. Academic Press, 2014.

[GJ10] Gaël Guennebaud, Benoît Jacob, et al. *Eigen v3*. http://eigen.tuxfamily.org. 2010.

[GV12] H Golub Gene and F Van Loan Charles. *Matrix computations. Vol. 3*. 2012.

[HS52] Magnus R Hestenes, Eduard Stiefel, et al. "Methods of conjugate gradients for solving linear systems". In: *Journal of research of the National Bureau of Standards* 49.6 (1952), pp. 409–436.

[ITF04] Geoffrey Irving, Joseph Teran, and Ronald Fedkiw. "Invertible finite elements for robust simulation of large deformation". In: *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*. 2004, pp. 131–140.

[KBK13] Michael Kremer, David Bommes, and Leif Kobbelt. "OpenVolumeMesh–A versatile index-based data structure for 3D polytopal complexes". In: *Proceedings of the 21st International Meshing Roundtable*. Springer, 2013, pp. 531–548.

[Kor17] Alexander M. Korsunsky. "Chapter 2 - Elastic and Inelastic Deformation and Residual Stress". In: *A Teaching Essay on Residual Stresses and Eigenstrains*. Ed. by Alexander M. Korsunsky. Butterworth-Heinemann, 2017, pp. 5–20. ISBN: 978-0-12-810990-8. DOI: https://doi.org/10.1016/B978-0-12-810990-8.00002-1. URL: http://www.sciencedirect.com/science/article/pii/B9780128109908000021.

[Lak87] Roderic Lakes. "Foam structures with a negative Poisson's ratio". In: *Science* 235 (1987), pp. 1038–1041.

[Lak93] Roderic Lakes. "Advances in negative Poisson's ratio materials". In: *Advanced Materials* 5.4 (1993), pp. 293–296.

[LM15] Joerg Liesen and Volker Mehrmann. *Linear algebra*. 1st ed. 2015. Springer International Publishing, Switzerland 2015: Springer, Cham, 2015. ISBN: 978-3-319-24344-3.

[MK10] Jan Möbius and Leif Kobbelt. "OpenFlipper: an open source geometry processing and rendering framework". In: *International Conference on Curves and Surfaces*. Springer. 2010, pp. 488–500.

[Moo40] Melvin Mooney. "A theory of large elastic deformation". In: *Journal of applied physics* 11.9 (1940), pp. 582–592.

[MR09] P. H. Mott and C. M. Roland. "Limits to Poisson's ratio in isotropic materials". In: *Phys. Rev. B* 80 (13 Oct. 2009), p. 132104. DOI: 10.1103/PhysRevB.80.132104. URL: https://link.aps.org/doi/10.1103/PhysRevB.80.132104.

[Ogd97] Raymond W Ogden. *Non-linear elastic deformations*. Courier Corporation, 1997.

[Riv48] RS Rivlin. "Large elastic deformations of isotropic materials IV. Further developments of the general theory". In: *Philosophical Transactions of the Royal Society of London. Series A,*

Mathematical and Physical Sciences 241.835 (1948), pp. 379–397.

[SGK18] Breannan Smith, Fernando De Goes, and Theodore Kim. "Stable Neo-Hookean Flesh Simulation". In: *ACM Trans. Graph.* 37.2 (Mar. 2018), 12:1–12:15. ISSN: 0730-0301. DOI: 10.1145/3180491. URL: http://doi.acm.org/10.1145/3180491.

[Spe80] A. J. M. Spencer. *Continuum Mechanics*. 2004th ed. 31 East 2nd Street, Mineola, N.Y. 11501: Dover Publications, Inc., 1980. ISBN: 0-486-43594-6 (pbk.)

[Sto12] Alexey Stomakhin et al. "Energetically consistent invertible elasticity". In: *Proceedings of the 11th ACM SIGGRAPH/Eurographics conference on Computer Animation*. 2012, pp. 25–32.

[Ter05] Joseph Teran et al. "Robust quasistatic finite elements and flesh simulation". In: *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*. 2005, pp. 181–190.

[WY16] Huamin Wang and Yin Yang. "Descent methods for elastic body simulation on the GPU". In: *ACM Transactions on Graphics (TOG)* 35.6 (2016), pp. 1–10.

Erklärung

gemäss Art. 28 Abs. 2 RSL 05

Name/Vorname: Masanti Corina Danja

Matrikelnummer: 15-128-655

Studiengang: Informatik

Bachelor ☒ Master ☐ Dissertation ☐

Titel der Arbeit: Flesh Simulation with Application to Character Animation

LeiterIn der Arbeit: Prof. Dr. David Bommes

Ich erkläre hiermit, dass ich diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen benutzt habe. Alle Stellen, die wörtlich oder sinngemäss aus Quellen entnommen wurden, habe ich als solche gekennzeichnet. Mir ist bekannt, dass andernfalls der Senat gemäss Artikel 36 Absatz 1 Buchstabe o des Gesetztes vom 5. September 1996 über die Universität zum Entzug des auf Grund dieser Arbeit verliehenen Titels berechtigt ist.

Bern, 02.11.2020

Ort/Datum

C. Masanti

Unterschrift