Sofia University "St. Kliment Ohridski"
Faculty of Mathematics and Informatics
Department of Numerical Methods and Algorithms

Cloth Simulation

Master's thesis
in
Mathematical Modeling and
Computational Mathematics

Author:
Dimitar V.
Trendafilov
FN: 25612

Supervisor:
Assist. Prof. Dr.
Tihomir B. Ivanov

March 21, 2021

This page intentionally left blank.

Contents

Chapter 1

Introduction

Cloths are everywhere, they comprise everything you wear and many items you touch on a daily basis. Therefore, one would suspect there are well established models we can use to simulate them, but actually this is not the case. In fact cloth simulation is notoriously hard both computationally and mathematically. All available design and modeling tools have narrow usages and are specific to a problem area. However, there is huge amount of research (going back as far as 1930s [33]) and development in the area.

Research is conducted by multiple industries. The textile industry is interested in how a pattern produced on cloths would look once printed (Figure 1.1c). In fact this is what initially started it all. Inherently the fashion industry is also involved. A usage one would probably not think of at first is computer graphics (Figure 1.1a). Mesh modeling tools such as Blender have build-in solutions and various free and licensed plug-ins. Also all popular game engines come with a build-in physics engine which integrates with its character clothing systems (Figure 1.1b). Each and every usage has its own constraints and requirements for model's characteristics.

As in all research fields, trends strongly correlate with computer science and computational hardware advancements. Older articles try to present models which involve the bare minimum of computation [1, 9, 10, 35]. With the popularisation of multi-threading, models started involving more precise numerical schemes [13], continuum mechanics [4, 44] and parallelization optimizations [3]. General purpose GPU programing just solidified the previous trend [41]. In more recent years artificial intelligence [23] and ray tracing [24] have started appearing.

(a) Disney constantly improves its tool set. Their bold character designs include challenging clothing, such as Elsa's made of gossamer fabric.

(b) Games strive for immersion. Legend has it that Rocksteady Studios had a full-time employee work on Batman's cape for one of the Arkham series' games.

(c) A lot of the literature deals with garment design.

Figure 1.1: Cloth simulation usages in different industries.

1.1 Terminology

Before we discuss the models we shall use, let us first introduce some of the terminology that will be used throughout the thesis. Here, we just give an intuitive understanding of those notions and we shall define some of them more rigorously later in the text.

Particle: A particle is a body consisting of a single point with no dimensions. Although a particle has no volume it has a mass m.

Stress: Stress (Figure 1.2) is a measure of the internal forces in a body between its particles. These internal forces are a reaction to the external forces applied on the body that cause it to separate, compress or slide. External forces are either surface forces or body forces. Stress is the average force per unit area that a particle of a body exerts on an adjacent particle, across an imaginary surface that separates them. In SI units, it is measured in Newtons per square meter N/m^2, which is also called Pascal written with the symbol Pa. Stress is a tensor quantity, usually denoted by the letter σ, but more on that in Section 3.2.

Normal and Shear Stress: Quantitatively, the stress on a given imaginary surface can be expressed by the Cauchy traction vector $\vec{T}$, defined as the traction force $\vec{F}$ per unit area between adjacent parts of the material across an imaginary separating surface S. The stress $\vec{T}$ that a particle P applies

on another particle Q across a surface S can have any direction relative to S. The vector $\vec{T}$ may be decomposed as the sum of two components: the normal stress (compression or tension) perpendicular to the surface, and the shear stress that is parallel to the surface.

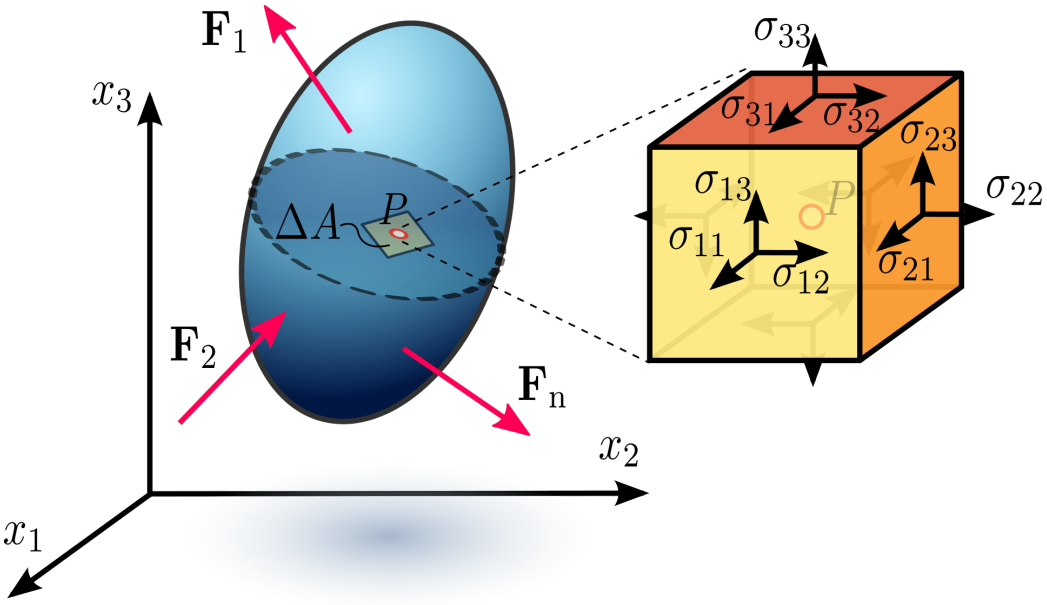


Figure 1.2: Stress In Continuum Mechanics. In blue we have some material on the boundary of which various external forces $\vec{F_i}$ where $i = \overline{1,n}$ are acting. P is point where material exists. In yellow a infinitesimally small cube around P with volume ΔA is shown. σ_{ij} where $i,j = \overline{1,3}$ are the stresses the volume experiences in different directions.

Strain: Strain is a description of deformation in terms of relative displacement of particles in the body that excludes rigid-body motions. For linear elastic materials the relation between stresses and induced strains is expressed by Hooke's law which we formulate later in the text.

Normal and Shear Strain: Strains are classified as either normal or shear. A normal strain is perpendicular to the face of an element, and a shear strain is parallel to it.

Warp and Weft: The two main components used in woven textile (Figure 1.3) are orthogonal threads called warp (those are the longitudinal ones,

stationary on the loom frame) and weft (the transverse ones, attached to the loom shuttle).

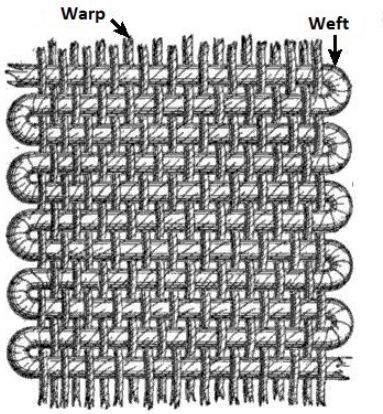


Figure 1.3: Textile threads.

1.2 Thesis Structure and Goals

The goal of this master's thesis is to pick several appropriate models from the literature, explain the mathematics behind them and give concrete recommendations on their usability and implementation. In all cases we will explore popular models in the field. They will be explained in terms of the original articles which introduced them. We will however try to address their known limitations using derivative work.

Knowing the copious amounts of articles and papers written on the subject and the countless available methods, some sieving had to be done. First of all, methods geared towards the drape problem [9] (what form would the cloth take if it covered a solid object) will be omitted. Due to the huge amounts of raw computational power available in computers nowadays we can solve for the dynamics of the cloth not just its draping behavior. Nevertheless these are the early works and have influenced modern ones.

We will split the thesis in two parts. In the first, corresponding to Chapter 2, we will explore models formulated in terms of systems of ODEs and based on a description of the fabric as a spring-mass system. Then in Chapter 3, computational models based on continuum mechanics and solved with the finite element numerical method will be analyzed. We will finish by giving a review and some words of advice based on the experience we had

while implementing and improving the models. The technical aspects are summarized in Chapter 4.

According to the literature, models in the first part generally have better computational performance, although this is less of an argument when implicit numerical methods are used. They can be used in soft-real time applications such as game physics. Their main goal is visual plausibility rather than physical correctness. In most cases a narrower range of fabrics can be simulated.

On the other hand, models in the second part are generally more computationally intensive and mathematically complex. Their main goal is physical correctness. A wider range of fabrics can be simulated and usually controls are based on their intrinsic physical properties.

In both cases a specific model was picked based on various considerations:

- Scalability and GPU portability which is extremely important nowadays.
- Universality. Algorithms dealing with extreme cases are not interesting, because in most cases they are built on top of preexisting general purpose ones.
- Popularity. The chosen models need to have large influence on other work in the field.
- Simplicity. In some cases models get overcomplicated in order to add support for friction, collision detection or other physical phenomena.

Chapter 2

Particle Mass-Spring Model

In this chapter, we consider a model, proposed in [35] which aims to describe rigid cloth behavior using deformation constraints on a mass-spring system. A lot of published research in the area [3, 41, 50, 52, 54] builds up on this work. Systems using the methods laid out in the aforementioned paper, usually aim for performance through parallelism and problem simplification, but it is a physically-based model nevertheless. The problem solved in [35] is derived from elasticity deformable models and improved in order to take into account the non-elastic properties of woven fabrics. Explicit time integration is used instead of using stiff springs[1] which can cause instabilities. Let us first explain some concepts.

Spring Natural Length: The natural length of a spring is its length when there is no load on it.

Spring Equilibrium Length: The equilibrium length of a spring is its length, when there might be load present on it, but has reached a static state.

Constitutive Equation/Relation: In physics and engineering, a constitutive equation/relation is a relation between two physical quantities (especially kinetic quantities as related to kinematic quantities) that is specific to a material or substance, and approximates the response of that medium to external stimuli, usually as applied fields or forces.

Hooke's Law [49]: Let us consider a simple helical spring that has one end attached to some fixed object, while the free end is being pulled by a force

[1] Springs which directly change particle positions instead of generating internal forces.

whose magnitude is F_s. Suppose that the spring has reached an equilibrium length. Let x be the amount by which the free end of the spring was displaced from its natural length. Hooke's law states that:

$$F_s = kx$$

where k is a positive real number, characteristic of the spring. Moreover, the same formula holds when the spring is compressed, with F_s and x both negative in that case. This is a constitutive relation between stress and strain for the spring's material.

2.1 Mathematical Formulation

The elastic model proposed in [35] considers the fabric as a mesh of $m \times n$ masses (particles), each connected to 12 others through massless springs of non-zero natural length. The mass-spring connections shown in Figure 2.1 are as follows:

- Springs linking mass $[i, j]$ with:
  - mass $[i, j + 1]$
  - mass $[i, j - 1]$
  - mass $[i + 1, j]$
  - mass $[i - 1, j]$

  Those are referred as "structural springs" and simulate the reaction to compression forces such as stretching.

- Springs linking mass $[i, j]$ with:
  - mass $[i + 1, j + 1]$
  - mass $[i - 1, j - 1]$
  - mass $[i - 1, j + 1]$
  - mass $[i + 1, j - 1]$

  Those are referred as "shear springs" and simulate the reaction to shear forces.

- Springs linking mass $[i, j]$ with:
  - mass $[i, j + 2]$

- mass $[i, j-2]$
- mass $[i+2, j]$
- mass $[i-2, j]$

Those are referred as "flexion springs" and simulate the reaction to flexion forces such as bending.

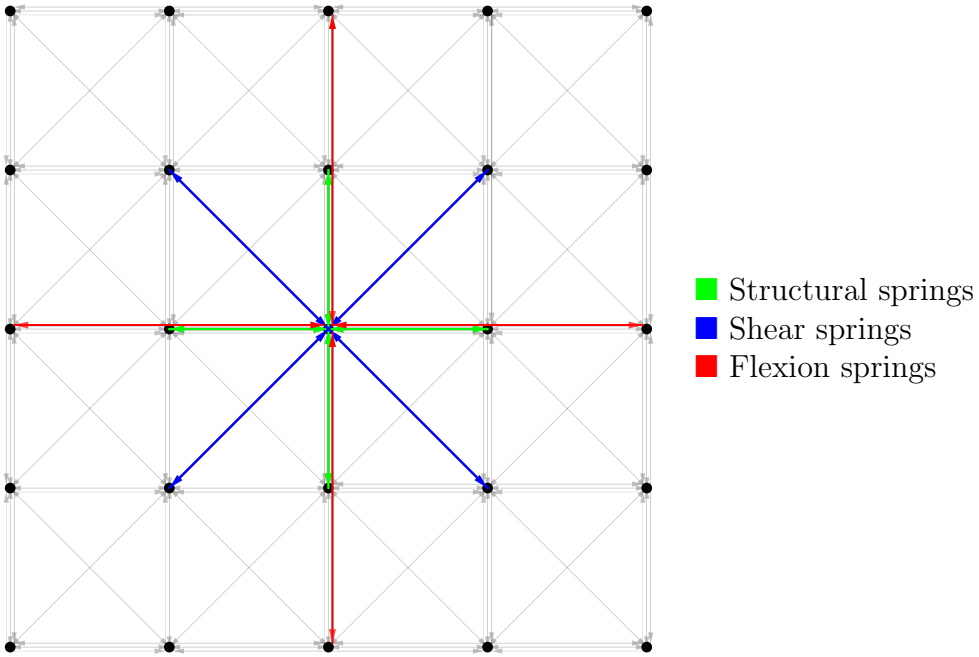


Figure 2.1: Particle spring connections.

Let the particle $[i, j]$ be positioned at $\vec{P}_{i,j}(t)$ at time t for $i = \overline{1, m}$ and $j = \overline{1, n}$. The evolution of the system is governed by Newton's second law of motion $\vec{F}_{i,j} = \mu \vec{a}_{i,j}$, where μ is the mass of each particle (we assume that all particles have the same mass) and $\vec{a}_{i,j}$ is the acceleration caused by the force $\vec{F}_{i,j}$. $\vec{F}_{i,j}$ is the resultant force from interior and exterior forces. The internal forces are resultant of springs tension linking the particle at position $P_{i,j}$ to its neighbors. Based on Hooke's law we can write:

$$\vec{F}_{internal}(P_{i,j}) = -\sum_{(k,l)\in R_{i,j}} K_{i,j,k,l} \left[\vec{l}_{i,j,k,l} - l^0_{i,j,k,l} \frac{\vec{l}_{i,j,k,l}}{\left\| \vec{l}_{i,j,k,l} \right\|} \right] \tag{2.1}$$

where:

- $R_{i,j}$ is the set of all pairs (k,l) such that $[k,l]$ is linked by a string to $[i,j]$
- $\vec{l}_{i,j,k,l} = \overrightarrow{P_{i,j}P_{k,l}}$
- $l^0_{i,j,k,l}$ is the natural length of the spring connecting $[k,l]$ to $[i,j]$
- $K_{i,j,k,l}$ is the stiffness of the spring connecting $[k,l]$ to $[i,j]$

The external forces are of various nature and can be modified according to the kind of load we wish to express. The basic ones that are further used in our numerical experiments are:

$$\vec{F}_{external}(P_{i,j}) = \vec{F}_{gravity}(P_{i,j}) + \vec{F}_{damp}(P_{i,j}) + \vec{F}_{wind}(P_{i,j}) \tag{2.2}$$

where:

- Let $\vec{\mathbf{g}}$ be the acceleration due to gravity, then $[i,j]$'s weight is expressed as:
$$\vec{F}_{gravity}(P_{i,j}) = \mu\vec{\mathbf{g}}$$
- The viscous damping will be given by:
$$\vec{F}_{damp}(P_{i,j}) = -C_{damp}\vec{v}_{i,j}$$
where C_{damp} is the damping coefficient and $\vec{v}_{i,j}$ is $[i,j]$'s velocity. The role of this damping is in fact to model in first approximation the dissipation of mechanical energy.
- Drag and lift depend on the properties of the fluid in which an object is submerged and on the size, shape and velocity of the object. We will introduce its effects using:
$$\vec{F}_{wind}(P_{i,j}) = \frac{1}{2}\rho A_{i,j}\left[(C_D - C_L)(\vec{v}^r_{i,j}\cdot\vec{n}_{i,j})\vec{v}^r_{i,j} + C_L\left|\vec{v}_{i,j}\right|^2\vec{n}_{i,j}\right]$$
where ρ is air density, $A_{i,j}$ the cloth's surface area corresponding to a particle $[i,j]$, $\vec{n}_{i,j}$ is the geometry normal, $\vec{v}_{i,j}$ and $\vec{v}^r_{i,j}$ are the velocity and relative velocity again for a particle $[i,j]$, C_D and C_L are drag and lift coefficients. For more details refer to Appendix A.

Using Equations (2.1) and (2.2) one can compute $\vec{F}_{i,j}(t)$, i.e., the sum of the external and internal forces acting at the point $P_{i,j}$ at time t, we also know $\vec{P}_{i,j}(0)$ which is inputed by the user. If we want to get $\vec{P}_{i,j}(t)$ we will need to

solve a system of ordinary differential equations based on the fundamental laws of dynamics:

$$\begin{cases} \vec{F}_{i,j} &= \mu\vec{a}_{i,j} \\ \frac{d\vec{v}_{i,j}}{dt} &= \vec{a}_{i,j} \\ \frac{d\vec{P}_{i,j}}{dt} &= \vec{v}_{i,j} \end{cases}, \text{ where } i = \overline{1,m} \text{ and } j = \overline{1,n} \tag{2.3}$$

2.2 Numerical Integration

The initial value problem (Equation (2.3)) can be solved using various numerical methods. We shall derive here several methods that are further used in the numerical experiments below.

2.2.1 Simple Explicit Method

The first scheme that we implemented is based on the Euler method and is the numerical scheme proposed in the original paper [35]. Let us introduce a uniform mesh $\bar{\omega}_{\Delta t} = \left\{t_n : t_n = t_0 + n\Delta t, n = \overline{0,N}, N = \frac{T-t_0}{\Delta t}\right\}$, where Δt is the simulation's time step given as an input argument, t_0 and T are the time at which the cloth patch was placed in the scene and the time at which it will be removed. Let $\vec{a}^n_{i,j}$, $\vec{P}^n_{i,j}$, $\vec{v}^n_{i,j}$ denote the approximate values of the solution at the point t_n, $n = \overline{0,N}$. When applied to Equation (2.3) the result is:

$$\begin{cases} \vec{a}^n_{i,j} &= \frac{1}{\mu}\vec{F}^n_{i,j} \\ \vec{v}^{n+1}_{i,j} &= \vec{v}^n_{i,j} + \frac{\Delta t}{\mu}\vec{F}^n_{i,j} \\ \vec{P}^{n+1}_{i,j} &= \vec{P}^n_{i,j} + \Delta t\vec{v}^n_{i,j} + \frac{\Delta t^2}{\mu}\vec{F}^n_{i,j} \end{cases}$$

However our numerical experiments have shown that this method behaves poorly which has led us to explore other integration options. We summarize below what we have observed for the applicability of this method to our problem.

Pros:	**Cons:**
• Extremely easy to implement. • Low storage complexity.	• Has numerical stability issues. • Due to the previous point it needs small time steps which in turn leads to performance decline.

2.2.2 Verlet Methods

The algorithm is quite old and first appeared in 1791 and has been rediscovered many times since then, most recently by Loup Verlet in the 1960s [43]

for use in molecular dynamics. The idea for experimenting with Verlet integration comes from derivative works [50, 54] of the one we currently discuss [35]. In fact, the algorithm is quite popular for particle simulations in computer graphics. The property of the method in which we are most interested is its good numerical stability, at no significant additional computational or storage cost over the simple explicit method.

Although, the algorithms' ideas can be framed in different ways the common thing is that iterations depend on knowing or obtaining acceleration separately. As a result precision is highly dependent on the provided acceleration values. The cloth model we are currently discussing is defined through particle forces which are trivially convertible to acceleration thanks to Newton's law. Therefore, it should not be surprising that most Verlet methods are a good fit to our problem. Although they are quite similar in precision they give wildly different results, which gives a great opportunity for comparisons.

Basic Störmer–Verlet Method

Using central difference approximation of the second derivative of the position $\vec{x}(t)$ of a given object (we denote the second derivative, i.e. acceleration, with $\vec{a}(t)$), a formula is derived for approximating the next position from the previous two without explicitly using the velocity $\vec{v}(t)$:

$$\frac{\Delta^2 \vec{x}_n}{\Delta t^2} = \frac{\vec{x}_{n+1} - 2\vec{x}_n + \vec{x}_{n-1}}{\Delta t^2} = \vec{a}_n$$

$$\vec{x}_{n+1} = 2\vec{x}_n - \vec{x}_{n-1} + \vec{a}_n \Delta t^2$$

The local error is quantified by inserting the exact values $\vec{x}\,(t_{n-1})$, $\vec{x}\,(t_n)$, $\vec{x}\,(t_{n+1})$ into the iteration and computing the Taylor expansions about t of $\vec{x}\,(t \pm \Delta t)$ in different time directions:

$$\vec{x}\,(t + \Delta t) = \vec{x}\,(t) + \vec{v}\,(t)\,\Delta t + \frac{\vec{a}\,(t)\,\Delta t^2}{2} + \frac{\vec{b}\,(t)\,\Delta t^3}{6} + \mathcal{O}\left(\Delta t^4\right)$$

$$\vec{x}\,(t - \Delta t) = \vec{x}\,(t) - \vec{v}\,(t)\,\Delta t + \frac{\vec{a}\,(t)\,\Delta t^2}{2} - \frac{\vec{b}\,(t)\,\Delta t^3}{6} + \mathcal{O}\left(\Delta t^4\right)$$

where $\vec{x}$ is the position, $\vec{v} = \dot{\vec{x}}$ the velocity, $\vec{a} = \ddot{\vec{x}}$ the acceleration and $\vec{b} = \dddot{\vec{x}}$ the jerk. Adding the two expansions gives:

$$\vec{x}\,(t + \Delta t) = 2\vec{x}\,(t) - \vec{x}\,(t - \Delta t) + \vec{a}\,(t)\,\Delta t^2 + \mathcal{O}\left(\Delta t^4\right)$$

When applied to Equation (2.3) the result is:

$$\begin{cases} \vec{a}_{i,j}^{n} &= \frac{1}{\mu}\vec{F}_{i,j}^{n} \\ \vec{P}_{i,j}^{n+1} &= 2\vec{P}_{i,j}^{n} - \vec{P}_{i,j}^{n-1} + \Delta t^2 \vec{a}_{i,j}^{n} \end{cases}$$

Not having velocities explicitly given in the formulas is an issue, because they are necessary for the calculation of the forces in the model. This deficiency can be dealt with by estimating the velocity using a backward difference formula:

$$\vec{v}_n = \frac{\vec{x}_n - \vec{x}_{n-1}}{\Delta t}$$

Sadly using a first-order accurate formula for the velocity approximation means that the global error of the method is $\mathcal{O}(\Delta t)$

Pros:	**Cons:**
• As simple as the simple explicit method. • Low computation and storage complexity. • Extremely stable.	• Global error of $\mathcal{O}(\Delta t)$.

Variable Time Discretization Step Verlet Methods

Another integration option is using a method which supports variable time discretization steps. Obviously, doing extremely large steps is not practical as the error will grow rapidly, however we can limit the maximum discretization step and work on getting a more "recent" result. This experiment is particularly interesting for real-time simulations as they are notorious for their fluctuating times. We will discuss two derivations, both of which do not use velocities. Both versions use the following non-uniform mesh:

$$\hat{\omega} = \{t_0 < t_1 < \cdots < t_n = t_{end}\}$$

where t_{end} is the simulation end time. Also we enforce $t_{i+1} - t_i < t_{max}$, where t_{max} is the largest allowed time discretization step. Every t_i, $i = \overline{0, n}$ corresponds to time at which simulation results were requested by the user.

[48] presents an extension of the classical Störmer–Verlet version for the purpose of implementing game physics simulations. The derivation begins the same way one would derive the Störmer–Verlet Method, the difference is in the assumptions made midway. First we write out the Euler Method:

$$\begin{cases} \vec{x}_{i+1} = \vec{x}_i + \vec{v}_i \Delta t_i + \dfrac{\vec{a}_i}{2}\Delta t_i^2 & \text{(2.4a)} \\ \vec{v}_{i+1} = \vec{v}_i + \vec{a}_i \Delta t_i & \text{(2.4b)} \end{cases}$$

We shift the whole equation back a step:

$$\begin{cases} \vec{x}_i = \vec{x}_{i-1} + \vec{v}_{i-1}\Delta t_{i-1} + \dfrac{\vec{a}_{i-1}}{2}\Delta t_{i-1}^2 & \text{(2.5a)} \\ \vec{v}_i = \vec{v}_{i-1} + \vec{a}_{i-1}\Delta t_{i-1} & \text{(2.5b)} \end{cases}$$

Now we substitute $\vec{v}_{i-1} = \vec{v}_i - \vec{a}_{i-1}\Delta t_{i-1}$ in Equation (2.5a) and do some equation rewriting:

$$\vec{x}_i - \vec{x}_{i-1} = \vec{v}_i\Delta t_{i-1} - \vec{a}_{i-1}\Delta t_{i-1}^2 + \frac{\vec{a}_{i-1}}{2}\Delta t_{i-1}^2$$

$$\vec{x}_i - \vec{x}_{i-1} = (\vec{v}_i - \frac{\vec{a}_{i-1}}{2}\Delta t_{i-1})\Delta t_{i-1}$$

Now multiply by $\frac{\Delta t_i}{\Delta t_{i-1}}$ and add $\vec{a}_{i-1}\Delta t_i^2$:

$$(\vec{x}_i - \vec{x}_{i-1})\frac{\Delta t_i}{\Delta t_{i-1}} + \vec{a}_{i-1}\Delta t_i^2 = (\vec{v}_i - \frac{\vec{a}_{i-1}}{2}\Delta t_{i-1})\Delta t_i + \vec{a}_{i-1}\Delta t_i^2 \tag{2.6}$$

Now we assume constant acceleration throughout the time step $\vec{a}_{i-1} = \vec{a}_i$. We also assume constant time step $\Delta t_{i-1} = \Delta t_i$, but only on the right hand side of Equation (2.6) (for the reasoning behind both assumptions and their repercussions check [48]):

$$(\vec{x}_i - \vec{x}_{i-1})\frac{\Delta t_i}{\Delta t_{i-1}} + \vec{a}_i\Delta t_i^2 = \vec{v}_i\Delta t_i + \frac{\vec{a}_i}{2}\Delta t_i^2$$

Substituting Equation (2.5a) in the right hand side gives:

$$(\vec{x}_i - \vec{x}_{i-1})\frac{\Delta t_i}{\Delta t_{i-1}} + \vec{a}_i\Delta t_i^2 = \vec{x}_{i+1} - \vec{x}_i$$

With this we arrive at our final form:

$$\vec{x}_{i+1} = \vec{x}_i + (\vec{x}_i - \vec{x}_{i-1})\frac{\Delta t_i}{\Delta t_{i-1}} + \vec{a}_i\Delta t_i^2 \tag{2.7}$$

When applied to Equation (2.3) the result is:

$$\begin{cases} \vec{a}_{i,j}^{\,n} & = \frac{1}{\mu}\vec{F}_{i,j}^{\,n} \\ \vec{P}_{i,j}^{\,n+1} & = \vec{P}_{i,j}^{\,n} + \left(\vec{P}_{i,j}^{\,n} - \vec{P}_{i,j}^{\,n-1}\right)\frac{\Delta t_n}{\Delta t_{n-1}} + \vec{a}_{i,j}^{\,n}\Delta t_n^2 \end{cases}$$

As we can see from the derivation above, Equation (2.7) is justified in a hand-wavy manner which is usually not ideal. Using only Taylor expansions

we can derive a better variable time discretization step method. First we expand with a time step of Δt_i forward and a time step backwards of Δt_{i-1}:

$$\vec{x}(t+\Delta t_i) = \vec{x}(t) + \frac{d\vec{x}}{dt}(t)\Delta t_i + \frac{1}{2}\frac{d^2\vec{x}}{dt^2}(t)\Delta t_i^2 + \mathcal{O}\left(\Delta t_i^3\right)$$

$$\vec{x}(t-\Delta t_{i-1}) = \vec{x}(t) - \frac{d\vec{x}}{dt}(t)\Delta t_{i-1} + \frac{1}{2}\frac{d^2\vec{x}}{dt^2}(t)\Delta t_{i-1}^2 + \mathcal{O}\left(\Delta t_{i-1}^3\right)$$

Now for some equation transformation:

$$\frac{\vec{x}(t+\Delta t_i) - \vec{x}(t)}{\Delta t_i} = \frac{d\vec{x}}{dt}(t) + \frac{1}{2}\frac{d^2\vec{x}}{dt^2}(t)\Delta t_i + \mathcal{O}\left(\Delta t_i^2\right)$$

$$\frac{\vec{x}(t-\Delta t_{i-1}) - \vec{x}(t)}{\Delta t_{i-1}} = -\frac{d\vec{x}}{dt}(t) + \frac{1}{2}\frac{d^2\vec{x}}{dt^2}(t)\Delta t_{i-1} + \mathcal{O}\left(\Delta t_{i-1}^2\right)$$

Adding the two equations eliminates the velocity term:

$$\frac{\vec{x}(t+\Delta t_i) - \vec{x}(t)}{\Delta t_i} + \frac{\vec{x}(t-\Delta t_{i-1}) - \vec{x}(t)}{\Delta t_{i-1}} = \\ = \frac{d^2\vec{x}}{dt^2}(t)\frac{\Delta t_i + \Delta t_{i-1}}{2} + \mathcal{O}\left(\max \Delta t^2\right)$$

With this we arrive at our final form:

$$\vec{x}(t+\Delta t_i) = \vec{x}(t) + (\vec{x}(t) - \vec{x}(t-\Delta t_{i-1}))\frac{\Delta t_i}{\Delta t_{i-1}} + \\ + \vec{a}(t)\frac{\Delta t_i + \Delta t_{i-1}}{2}\Delta t_i + \mathcal{O}\left(\max \Delta t^3\right)$$

When applied to Equation (2.3) the result is:

$$\begin{cases} \vec{a}_{i,j}^{\,n} & = \frac{1}{\mu}\vec{F}_{i,j}^{\,n} \\ \vec{P}_{i,j}^{\,n+1} & = \vec{P}_{i,j}^{\,n} + (\vec{P}_{i,j}^{\,n} - \vec{P}_{i,j}^{\,n-1})\frac{\Delta t_n}{\Delta t_{n-1}} + \vec{a}_{i,j}^{\,n}\frac{\Delta t_n + \Delta t_{n-1}}{2}\Delta t_n \end{cases}$$

Pros:	**Cons:**
• In terms of simplicity and precision it is similar to the Basic Störmer–Verlet Method. • Adds small computation overhead, but no storage overhead. • Still extremely stable.	• Have precision issues with small time steps.

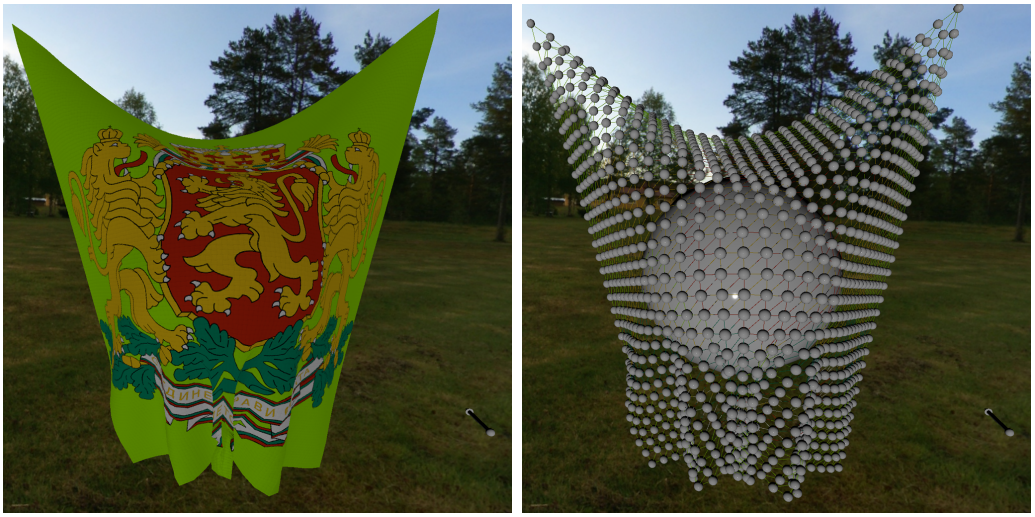


Figure 2.2: Results using the simple explicit method. The experiment uses relatively small $\Delta t = 0.0030s$. Springs have stiffness of $K = 43N/m$.

2.3 Results

Due to the nature of the problem, static images were not sufficient to display the model's behavior in various setups. Therefore we have resorted to video recordings. They are available in an attached media storage device under "Mass-Spring Model" directory with names corresponding to the experiments' order. Parameters not listed in the experiment's description can be seen in the videos. In addition to the provided clips Figures 2.2 and 2.3 display cloths with similar behavior using different numerical methods, from the information under the images we can see the differences in parameters.

Experiments were conducted using the following shared setup:

- 32×32 square particle grid with a $5.0m$ side.
- Air temperature of $20°C$

Experiment 1: The goal of this experiment is to show how the patch reacts when given a wide range of material properties. Numerical integration method explained in Section 2.2.1 is used, paired with a relatively small time step of $\Delta t = 0.0032s$. The timeline of the video is as follows:

0:00 - 0:23 : Spring stiffness K is modified from $10N/m$ to $100N/m$. We can see the changes in sagging of the patch due to those changes.

0:23 - 0.26 : Spring stiffness K is fixed at $100N/m$. The textile looks almost inextensible.

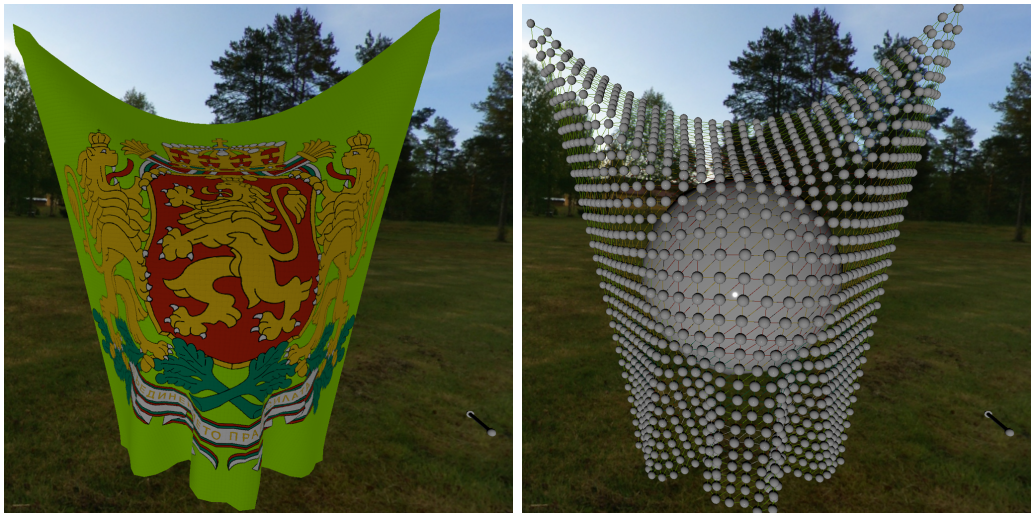


Figure 2.3: Results using Basic Störmer–Verlet method. The experiment uses relatively large $\Delta t = 0.0110s$. Stiff springs are used.

0:29 - 0.50 : Spring stiffness K is fixed at $76N/m$. Material density ρ is modified from $0.15kg/m^2$ to $1.0kg/m^2$. We can see sagging of the patch due to the increased weight.

Experiment 2: The goal of this experiment is to show how the wind model affects the patch's kinematics. Integration method from Section 2.2.1 is used, paired with a relatively small time step of $\Delta t = 0.0032s$. Wind speed is fixed at $10.0m/s$, lift coefficient C_L is also fixed at 0.200. The experiment is split into the following clips:

Experiment 2.1: Material density ρ is slowly modified from $0.3kg/m^2$ to $0.8kg/m^2$ and back. We can see that lower mass patches are more affected by wind.

Experiment 2.2: Material density ρ is fixed at $0.3kg/m^2$. Drag coefficient C_D is slowly modified from 0.106 to 0.433. This shows how wind lift and drag work as forces opposite to one another which is expected.

Experiment 2.3: All parameters are fixed with values of $\rho = 0.3kg/m^2$, $C_D = 0.106$ and $C_L = 0.200$. Wind direction is transverse to the patch and randomization is disabled. We can observe various turbulent effects.

Experiment 3: The goal of this experiment is to demonstrate basic Störmer–Verlet method (Section 2.2.2) stability and absence of oscillations. No-

tice that stiff springs are used. The simulation was ran at different time steps and the numerical integration scheme handled all of them perfectly. The timeline of the video is as follows:

0:00 - 0:09 : Time step Δt is fixed at $0.01s$.

0:13 - 0:21 : Time step Δt is fixed at $0.02s$.

0:25 - 0:33 : Time step Δt is fixed at $0.03s$.

0:34 - 0:41 : Time step Δt is fixed at $0.04s$.

0:49 - 0:53 : Time step Δt is fixed at $0.005s$.

0:55 - 0:55 : The particle grid is visualized.

By inspecting all used time steps one can notice that, although large steps do not cause stability issues, they result in cloth sagging due to slow force propagation across the particle system.

Experiment 4: The goal of this experiment is to demonstrate variable time discretization step Verlet methods' issues. The clip was shot using improved version from Section 2.2.2. The timeline of the video is as follows:

0:03 - 0:15 : At maximum $\Delta t = 0.0053s$ numerical errors cause vibrations.

0:16 - 0:21 : At maximum $\Delta t = 0.0113s$ vibrations stop.

0:22 - 0:24 : At maximum $\Delta t = 0.0165s$ numerical errors lead to catastrophic instabilities.

Experiment 5: The goal of this experiment is to demonstrate the simple explicit method's numerical instabilities. Spring stiffness K is fixed at $75N/m$. The timeline of the video is as follows:

0:00 - 0:07 : All parameters are fixed with values of $\rho = 1.0kg/m^2$, $\Delta t = 0.0032s$ and $C_{damp} = 0.027$. Using cursor control (see Section 4.3) it is shown that large fabric density can cause instabilities.

0:08 - 0:21 : All parameters are fixed with values of $\rho = 0.16kg/m^2$, $\Delta t = 0.0032s$ and $C_{damp} = 0.027$. We see that instabilities are not present for small fabric densities.

0:32 - 0:40 : All parameters are fixed with values of $\rho = 1.0kg/m^2$, $\Delta t = 0.0032s$ and $C_{damp} = 0.111$. We see larger damping coefficient values mitigate the issue.

0:44 - 1.02 : All parameters are fixed with values of $\rho = 1.0kg/m^2$, $\Delta t = 0.0032s$ and $C_{damp} = 0.385$. It is shown large damping coefficient values give the feeling that the atmosphere is too viscose

Smaller time steps could also help, but the current experiment uses the smallest time step at which the simulation can be ran in real-time, showing that this is not always an option.

Chapter 3

Continuum Mechanics Based Models

The model discussed in the previous chapter has its benefits such as evaluation speed and simplicity, but suffers from some innate deficiencies which are a deal breaker in some use cases. In this chapter we will turn to continuum mechanics. An arbitrary triangular mesh will be used to define elements for solving the equation of motion with the finite element method.

3.1 Mathematical Formulation

Using the Lagrange form we get the following ordinary differential equations which describe the dynamics of the model [4]:

$$M\ddot{x} + D\dot{x} + K(x - x_0) = f_{ext} \tag{3.1}$$

where M is the mass matrix with the masses of the particles[1] on the diagonal, D is the damping matrix and K is the stiffness matrix of the cloth. The vectors x, $\dot{x}$ and $\ddot{x}$ contain the positions, velocities and accelerations at the vertices of the finite element mesh while x_0 defines the rest state of the model and f_{ext} expresses the external forces applied to the system. Hence, we have $3n$ ODEs for a mesh with n vertices. Equation (3.1) comes from purely engineering considerations and it is up to us to build a numerical representation for it. Before we discuss the model in more detail, let us first roughly explain some material science terminology:

Isotropic Material: Isotropic material is a material which has properties at a particular point which if measured in different directions are uniform.

[1] We assume particles are at the vertices of the finite element mesh.

It is said that the property has isotropy. Glass and metals are examples of isotropic materials.

Anisotropic Material: Anisotropic material is a material which has properties at a particular point which change if measured in different directions as opposed to isotropic material. It is said that the material property has anisotropy and we can measure it. Nickel and Copper exhibit anisotropic mechanical properties.

Orthotropic Material: Orthotropic material is a material which has properties at a particular point, which differ along three mutually-orthogonal axes, where each axis has twofold rotational symmetry. Wood is a classical example of orthotropic material.

Homogeneous Material: Homogeneous material is a material which has the same properties at every point. It is uniform without irregularities.

From Section 1.1 we know fabrics' threads are knitted in two major orthogonal directions. Knowing this structure we can justify treating a cloth patch as a planar homogeneous orthotropic material.

The mass distribution is defined by a diagonal **mass matrix** M. This way the masses of the model are concentrated at the vertices of the mesh. We assume that the mass of a dynamic particle is proportional to the area "corresponding" to the respective vertex in the triangle mesh, times the material density. The area corresponding to a vertex can be defined by the Voronoi area which is bounded by the midpoints of the incident edges and the circumcenters of the adjacent triangles (for an in-depth explanation see Appendix B).

Rayleigh damping [12, 40] is used to simulate viscosity. It consists of two parts: mass and stiffness damping. The **damping matrix** D for a model with a mass matrix M and stiffness matrix K is determined by:

$$D = \alpha M + \beta K \tag{3.2}$$

where α and β are the damping coefficients for mass and stiffness damping, respectively.

The only element left undefined from Equation (3.1) is the **stiffness matrix** K. It is applied on $x - x_0$ which contains the 3D displacements. However, earlier we said that the material can be treated as a planar object. With that in mind in Section 3.2 we will give formulation of linear elasticity for such

objects. Then in Sections 3.3 to 3.5 a finite element method for linear elastostatics in 2D [22] will be derived. Finally in Section 3.6 using corotational dynamics [16] it will be shown how the stiffness matrix from Sections 3.3 to 3.5 is applied to the 3D problem.

3.2 Linear Elasticity

A key observation is that a cloth patch can be treated as a surface, it is said the problem is under plane or membrane stress/strain conditions. Therefore, deformation in the model is defined by a two-dimensional vector field $\vec{u}(\vec{m})$ which is used to compute the deformed point location $\vec{x}(\vec{m}) = \vec{m} + \vec{u}(\vec{m})$ of an undeformed point $\vec{m} \in \mathbb{R}^2$. This vector field is defined only in areas where material exists.

Hooke's law for continuous media connects the stresses and strains (see Section 1.1) inside a continuous elastic material by a linear relationship that is mathematically similar to Hooke's spring law from Chapter 2.

However, the strain state in a solid medium around some point cannot be described by a single vector. The same parcel of material, no matter how small, can be compressed, stretched, and sheared at the same time, along different directions. Likewise, the stresses in that parcel can be at once pushing, pulling, and shearing.

In order to capture this complexity, the relevant state of the medium around a point must be represented by two-second-order tensors [39], the strain tensor ϵ and the stress tensor σ. The analogue of Hooke's spring law for continuous media is then $\sigma = -C\epsilon$, where C is a fourth-order tensor (that is, a linear map between second-order tensors) usually called the stiffness tensor or elasticity tensor. Although most literature defines the following concepts in three dimensions we will redefine them in two dimensions. For our case, a fourth-order tensor for a 2×2 second-order tensors gives another 2×2 second-order tensor where each component is linearly dependent on all components of the first:

$$\sigma_{ij} = \sum_{k=1}^{2}\sum_{l=1}^{2} c_{ijkl}\epsilon_{kl}, \text{ or } \begin{bmatrix} \sum_{k=1}^{2}\sum_{l=1}^{2} c_{11kl}\epsilon_{kl} & \sum_{k=1}^{2}\sum_{l=1}^{2} c_{12kl}\epsilon_{kl} \\ \sum_{k=1}^{2}\sum_{l=1}^{2} c_{21kl}\epsilon_{kl} & \sum_{k=1}^{2}\sum_{l=1}^{2} c_{22kl}\epsilon_{kl} \end{bmatrix}$$

This means the tensor C is represented by $2 \times 2 \times 2 \times 2 = 16$ components. For a specific material and conditions those values can be experimentally derived.

From here on we assume σ and ϵ are Cauchy's linear stress and strain tensors respectively. There are other tensors which express the same phe-

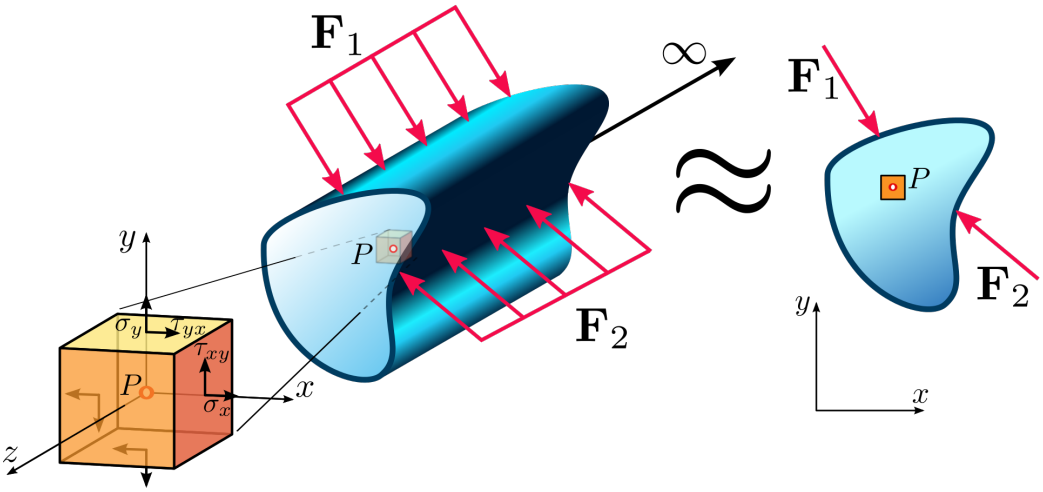


Figure 3.1: Plane state in a continuum.

nomenon with different properties, but they are not within the scope of this work.

Cauchy Stress Tensor: The cauchy stress tensor σ is defined with:

$$\sigma = \begin{bmatrix} \sigma_{xx} & \sigma_{xy} \\ \sigma_{yx} & \sigma_{yy} \end{bmatrix} \equiv \begin{bmatrix} \sigma_x & \tau_{xy} \\ \tau_{xy} & \sigma_y \end{bmatrix} \tag{3.3}$$

$$\sigma_{xx} \equiv \sigma_x,\ \sigma_{yy} \equiv \sigma_y,\ \sigma_{xy} = \sigma_{yx} \equiv \tau_{xy}$$

Applied on a unit-length vector $\vec{n}$, this gives the stress vector $\vec{T}$ across a surface with that normal at that point, i.e., $\vec{T} = \sigma \cdot \vec{n}$. The linear relation between $\vec{T}$ and $\vec{n}$ follows from the fundamental laws of conservation of linear momentum and static equilibrium of forces. Notice that the principle of conservation of angular momentum implies that the stress tensor is symmetric, i.e., $\sigma_{xy} = \sigma_{yx} \equiv \tau_{xy}$. σ_x and σ_y are called the orthogonal normal stresses and τ_{xy} the orthogonal shear stresses.

Strain Tensor: The strain tensor ϵ is defined with:

$$\epsilon = \frac{1}{2}\left(\frac{\partial \vec{u}}{\partial \vec{m}} + \frac{\partial \vec{u}}{\partial \vec{m}}^T\right) = \frac{1}{2}\left(\begin{bmatrix} \frac{\partial u_x}{\partial x} & \frac{\partial u_x}{\partial y} \\ \frac{\partial u_y}{\partial x} & \frac{\partial u_y}{\partial y} \end{bmatrix} + \begin{bmatrix} \frac{\partial u_x}{\partial x} & \frac{\partial u_y}{\partial x} \\ \frac{\partial u_x}{\partial y} & \frac{\partial u_y}{\partial y} \end{bmatrix}\right) =$$

$$= \begin{bmatrix} \frac{\partial u_x}{\partial x} & \frac{1}{2}\left(\frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x}\right) \\ \frac{1}{2}\left(\frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x}\right) & \frac{\partial u_y}{\partial y} \end{bmatrix} = \begin{bmatrix} \epsilon_{xx} & \epsilon_{xy} \\ \epsilon_{yx} & \epsilon_{yy} \end{bmatrix} \equiv \begin{bmatrix} \epsilon_x & \frac{\gamma_{xy}}{2} \\ \frac{\gamma_{yx}}{2} & \epsilon_y \end{bmatrix} \tag{3.4}$$

$$\epsilon_{xx} \equiv \epsilon_x,\ \epsilon_{yy} \equiv \epsilon_y,\ \gamma_{xy} \equiv \gamma_{yx}$$

γ_{xy} is the so called engineering shear strain.

Before we explain how one can derive the elasticity tensor we will formulate some abstract mechanical properties. They will be used as coefficients for the stiffness tensor.

Young's Modulus: The Young modulus is a mechanical property that measures the tensile stiffness of a solid material. It quantifies the relationship between tensile stress σ and axial strain ϵ in the linear elastic region of a material and is determined using the formula:

$$E \overset{\text{def}}{=\!=} \frac{\sigma}{\epsilon}$$

- Material tensile stiffness is rarely linear. However, for small deformations in most cases linearity holds.
- For isotropic materials E will be directionally independent, and therefore constant at a specific point. For anisotropic materials this does not hold and E needs to be computed for various directions and/or interpolated.
- The derived SI unit is the Pascal Pa, although it is usually expressed in Gigapascals GPa.

Cloths are not elastic under large deformations and tear, in fact most fabrics have exponential tensile stiffness. Young's modulus is not a perfect approximation, but we will still use it, hopefully under small deformations fabrics behave linearly enough, it is also really simple. If deformations get too large the simulation will lose realism anyway, because we do not simulate tearing. Under earlier orthotropy assumptions we need Young's modulus values measured in only directions parallel to fabric's threads.

Shear Modulus: Shear modulus, denoted by E_s is a measure of the elastic shear stiffness of a material and is defined as the ratio of shear stress to the shear strain. If we denote the shear stress with τ_{xy} and the shear strain with γ_{xy} the formula is:

$$E_s \overset{\text{def}}{=\!=} \frac{\tau_{xy}}{\gamma_{xy}}$$

- Similarly to Young's Modulus the derived SI unit is the Pascal Pa, although it is usually expressed in Gigapascals GPa.

Poisson's Ratio: Poisson's ratio ν is a measure of the Poisson effect, the deformation of a material in directions perpendicular to the direction of loading. The value of Poisson's ratio is the negative of the ratio of transverse strain to axial strain:

$$\nu_{xy} \overset{\text{def}}{=\!=} -\frac{\epsilon_y}{\epsilon_x}$$

- For isotropic materials $\nu_{xy} = \nu_{yx} = \nu$. However for anisotropic and orthotropic ones $\nu_{xy} \neq \nu_{yx}$.
- It is a dimensionless quantity.

Elasticity Tensor: The elasticity tensor is denoted by C. Although it has 16 components, due to symmetry in both stress and strain tensors only 9 of those are unique:

$$\sigma_{ij} = \sigma_{ji} \implies c_{ijkl} = c_{jikl}$$
$$\epsilon_{ij} = \epsilon_{ji} \implies c_{ijkl} = c_{ijlk}$$

This lets us express Hooke's law for anisotropic materials in matrix notation, also called Voigt notation or engineering notation.

$$\sigma = \begin{bmatrix}\sigma_{11}\\ \sigma_{22}\\ \sigma_{12}\end{bmatrix} \equiv \begin{bmatrix}\sigma_x\\ \sigma_y\\ \tau_{xy}\end{bmatrix}, \epsilon = \begin{bmatrix}\epsilon_{11}\\ \epsilon_{22}\\ 2\epsilon_{12}\end{bmatrix} \equiv \begin{bmatrix}\epsilon_x\\ \epsilon_y\\ \gamma_{xy}\end{bmatrix},$$
$$C = \begin{bmatrix}c_{1111} & c_{1122} & c_{1112}\\ c_{2211} & c_{2222} & c_{2212}\\ c_{1211} & c_{1222} & c_{1212}\end{bmatrix} \equiv \begin{bmatrix}C_{11} & C_{12} & C_{13}\\ C_{21} & C_{22} & C_{23}\\ C_{31} & C_{32} & C_{33}\end{bmatrix} \quad (3.5)$$

Now we derive the coefficients of C from Young's modulus, Shear modulus and Poisson ratios. Consider the strain and stress relation as a superposition of two effects: stretching in direction of the load and shrinking (caused by the load) in perpendicular direction:

$$\epsilon'_x = \frac{\sigma_x}{E_x} \qquad \epsilon''_x = -\frac{\nu_{yx}\sigma_y}{E_y} \qquad \epsilon_x = \epsilon'_x + \epsilon''_x = \begin{bmatrix}\frac{1}{E_x} & -\frac{\nu_{yx}}{E_y}\end{bmatrix}\begin{bmatrix}\sigma_x\\ \sigma_y\end{bmatrix}$$
$$\epsilon'_y = -\frac{\nu_{xy}\sigma_x}{E_x} \qquad \epsilon''_y = \frac{\sigma_y}{E_y} \qquad \epsilon_y = \epsilon'_y + \epsilon''_y = \begin{bmatrix}-\frac{\nu_{xy}}{E_x} & \frac{1}{E_y}\end{bmatrix}\begin{bmatrix}\sigma_x\\ \sigma_y\end{bmatrix}$$

From the Shear modulus we also have: $\gamma_{xy} = \frac{\tau_{xy}}{E_s}$

$$\implies \begin{bmatrix}\epsilon_x\\ \epsilon_y\\ \gamma_{xy}\end{bmatrix} = \begin{bmatrix}\frac{1}{E_x} & -\frac{\nu_{yx}}{E_y} & 0\\ -\frac{\nu_{xy}}{E_x} & \frac{1}{E_y} & 0\\ 0 & 0 & \frac{1}{E_s}\end{bmatrix}\begin{bmatrix}\sigma_x\\ \sigma_y\\ \tau_{xy}\end{bmatrix} \quad (3.6)$$

In fact we are interested in the inverse of this relation:

$$\begin{aligned} \epsilon_x &= \frac{\sigma_x}{E_x} - \frac{\nu_{yx}\sigma_y}{E_y} \\ \epsilon_y &= \frac{\sigma_y}{E_y} - \frac{\nu_{xy}\sigma_x}{E_x} \end{aligned} \quad \leftrightarrow \quad \begin{aligned} \sigma_x &= E_x\epsilon_x + \frac{\nu_{yx}E_x\sigma_y}{E_y} \\ \sigma_y &= E_y\epsilon_y + \frac{\nu_{xy}E_y\sigma_x}{E_x} \end{aligned}$$

After straightforward but rather lengthy computations we get:

$$\sigma_x = \begin{bmatrix} \frac{E_x}{1-\nu_{xy}\nu_{yx}} & \frac{E_x\nu_{yx}}{1-\nu_{xy}\nu_{yx}} \end{bmatrix} \begin{bmatrix} \epsilon_x \\ \epsilon_y \end{bmatrix}, \; \sigma_y = \begin{bmatrix} \frac{E_y\nu_{xy}}{1-\nu_{xy}\nu_{yx}} & \frac{E_y}{1-\nu_{xy}\nu_{yx}} \end{bmatrix} \begin{bmatrix} \epsilon_x \\ \epsilon_y \end{bmatrix}$$

$$\implies \begin{bmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{bmatrix} = \frac{1}{1-\nu_{xy}\nu_{yx}} \begin{bmatrix} E_x & E_x\nu_{yx} & 0 \\ E_y\nu_{xy} & E_y & 0 \\ 0 & 0 & E_s(1-\nu_{xy}\nu_{yx}) \end{bmatrix} \begin{bmatrix} \epsilon_x \\ \epsilon_y \\ \gamma_{xy} \end{bmatrix} \tag{3.7}$$

3.3 Partial Differential Equations

Theorem 3.3.1 (Gauss's Theorem)**:**
Suppose $\mathbb{R}^n \subset \Omega$ is compact and has a piecewise smooth boundary $\partial\Omega$. If $\vec{F}$ is a continuously differentiable vector field, defined in an open region that contains Ω, then:

$$\iiint_\Omega \nabla \cdot \vec{F} d\Omega = \oint_{\partial\Omega} \vec{F} \cdot \vec{n} ds$$

where $\vec{n}$ is the outer normal.

Theorem 3.3.2 (Integration by Parts)**:**
Under the assumptions of Theorem 3.3.1, the following relation holds true:

$$\iiint_\Omega \nabla \cdot \vec{j}\vec{v} d\Omega = -\iiint_\Omega \vec{j} \cdot \nabla\vec{v} d\Omega + \oint_{\partial\Omega} (\vec{j} \cdot \vec{n})\vec{v} ds$$

Having laid the physical foundation and the material science corresponding to our problem, we move forward to deriving the equations which govern the kinematic behavior of the cloth. This is done in the abstract setting of elliptic partial differential equations. For a more in-depth treatment and further reading refer to [14, 22] which are the main sources for what follows.

Consider an area Ω occupied by an elastic material and let ω denote an arbitrary subdomain of Ω with boundary $\partial\omega$ and exterior normal $\vec{n}$. Two types of forces can act on ω. First, there are forces acting on the whole volume, so called body forces. These are described by a force density $\vec{f}$, which expresses force per unit volume. The most common body force is gravity. Second, there are forces acting on the boundary $\partial\omega$. These are

assumed to have the form $\sigma \cdot \vec{n}$, where σ is the stress tensor from Section 3.2. $\sigma \cdot \vec{n}$ is the vector $(\sigma \cdot \vec{n})_i = \sum_{j=1}^{2} \sigma_{ij} n_j$. Summing body and contact forces we obtain the net force $\vec{F}$ on Ω:

$$\vec{F} = \int_{\omega} \vec{f} d\omega + \int_{\partial\omega} \sigma \cdot \vec{n} ds \tag{3.8}$$

Using Theorem 3.3.1 on Equation (3.8) we obtain:

$$\vec{F} = \iint_{\omega} \left(\vec{f} + \nabla \cdot \sigma \right) d\omega$$

In equilibrium $\vec{F} = \vec{0}$, and since ω is arbitrary, we have derived Cauchy's equilibrium equation:

$$\vec{f} + \nabla \cdot \sigma = \vec{0}$$

or, written in coordinate form,

$$\begin{cases} f_x + \frac{\partial \sigma_x}{\partial x} + \frac{\partial \tau_{xy}}{\partial y} = 0 \\ f_y + \frac{\partial \tau_{xy}}{\partial x} + \frac{\partial \sigma_y}{\partial y} = 0 \end{cases} \tag{3.9}$$

Bringing everything so far together plus introducing Dirichlet (Equation (3.10c)) and Neumann (Equation (3.10d)) boundary conditions we get:

$$\begin{cases} -\nabla \cdot \sigma = \vec{f} \ , in\ \Omega & \text{(3.10a)} \\ \sigma = C\epsilon, in\ \Omega & \text{(3.10b)} \\ \vec{u} = \vec{0} \ , on\ \Gamma_D & \text{(3.10c)} \\ \sigma \cdot \vec{n} = \vec{g}_N, on\ \Gamma_N & \text{(3.10d)} \end{cases}$$

3.4 Weak Form

Now that we have Equation (3.10) the next step is to derive its weak form. Let V be the Hilbert space:

$$V = \left\{ \vec{v} \in \left[H^1(\Omega) \right]^2 : \vec{v}|_{\Gamma_D} = \vec{0} \right\}$$

with a scalar product defined as:

$$\begin{array}{lll} (\vec{u}, \vec{v}) & = \iint_{\Omega} \vec{u} \cdot \vec{v} d\Omega & = \sum_{i=1}^{2} \iint_{\Omega} u_i v_i d\Omega \\ (\vec{u}, \vec{v})_{\partial\Omega} & = \oint_{\partial\Omega} \vec{u} \cdot \vec{v} ds & = \sum_{i=1}^{2} \oint_{\partial\Omega} u_i v_j ds \end{array}$$

Let us take an arbitrary test function $\vec{v} \in V$ and multiply Equation (3.10a) by it on both sides:

$$(\vec{f}, \vec{v}) = (-\nabla \cdot \sigma, \vec{v}) \tag{3.11}$$

The following notation is introduced:

$$A : B = \sum_{i,j=1}^{n,m} A_{ij} B_{ij}, \text{ where } A, B \in \mathbb{R}^{n \times m}$$

Using this notation and applying Theorem 3.3.2 we get:

$$(\sigma : \nabla \vec{v}) - (\sigma \cdot \vec{n}, \vec{v})_{\partial\Omega} = (\vec{f}, \vec{v})$$

where $\nabla \vec{v} \in \mathbb{R}^{2\times 2}$ and $[\nabla \vec{v}]_{ij} = \frac{\partial v_i}{\partial x_j}$. Finally, using the Neumann boundary condition $\sigma \cdot \vec{n} = \vec{g}_N$ on Γ_N, and the Dirichlet boundary condition that $\vec{v} = \vec{0}$ on Γ_D, we end up with:

$$(\sigma : \nabla \vec{v}) = (\vec{f}, \vec{v}) + (\vec{g}_N, \vec{v})_{\Gamma_N}, \quad \forall \vec{v} \in V$$

Actually, this can be simplified a bit more. If a matrix A is symmetric and another matrix B is anti-symmetric with zero diagonal, then $A : B = 0$. Now, recalling that any matrix can be decomposed into its symmetric and anti-symmetric part, $A = \frac{A+A^T}{2} + \frac{A-A^T}{2}$, it follows that:

$$\sigma : \nabla \vec{v} = \sigma : \frac{\nabla \vec{v} + \nabla \vec{v}^T}{2} + \sigma : \frac{\nabla \vec{v} - \nabla \vec{v}^T}{2} = \sigma(\vec{u}) : \epsilon(\vec{v})$$

where $\epsilon(\vec{v})$ and $\sigma(\vec{u})$ are operator form of Equations (3.3) and (3.4) meaning "tensor of the vector field". Hence the weak form is:

$$a(\vec{u}, \vec{v}) = l(\vec{v}), \quad \forall \vec{v} \in V \tag{3.12}$$

where the bilinear form $a(\cdot, \cdot)$ and the linear form $l(\cdot)$ are defined as:

$$a(\vec{u}, \vec{v}) = (\sigma(\vec{u}) : \epsilon(\vec{v})), \quad l(\vec{v}) = (\vec{f}, \vec{v}) + (\vec{g}_N, \vec{v})_{\Gamma_N}$$

3.5 Finite Elements Simulation

Using the Lax-Milgram lemma it can be shown [22] that Equation (3.12) has a unique solution $\vec{u} \in V$, which can be approximated with finite elements. To this end, we choose to approximate each component of u using continuous

piecewise linears. Let $\mathcal{K} = \{K\}$ be a triangulation of Ω, and let V_h be the finite element space:

$$V_h = \left\{ v \in V : v|_K \in [P_1(K)]^2, \forall K \in \mathcal{K} \right\}$$

The finite element method takes the form:

$$a(\vec{u_h}, \vec{v}) = l(\vec{v}), \quad \forall \vec{v} \in V_h \tag{3.13}$$

Using the engineering notation from Equation (3.7) we can write:

$$a(\vec{u_h}, \vec{v}) = \iint_\Omega \epsilon(\vec{v}) : \sigma(\vec{u}_h) dx = \\ = \iint_\Omega \vec{\epsilon}(\vec{v})^T \vec{\sigma}(\vec{u}_h) dx = \iint_\Omega \vec{\epsilon}(\vec{v})^T C \vec{\epsilon}(\vec{u_h}) dx \tag{3.14}$$

Now we will directly discuss the element matrices corresponding to Equation (3.14) and after that we will show the global stiffness matrix and the load vector. We do this, in order to focus on the parts which will make their way into the implementation.

Displacement in the interior of each triangular element is interpolated linearly by three shape functions $N_i(\vec{m})$:

$$\vec{u}_h(\vec{m}) = \sum_{i=1}^{3} N_i(\vec{m}) \vec{u_i}$$

Barycentric coordinates [27] are used to define the linear shape functions $N_i(\vec{m})$. The barycentric coordinates of a point $m = (x, y)^T$ in a two-dimensional triangle are defined by three linear polynomials shown in Figure 3.2:

$$N_1(\vec{m}) = \frac{1}{2A_e} \det \begin{bmatrix} 1 & 1 & 1 \\ x & x_2 & x_3 \\ y & y_2 & y_3 \end{bmatrix} = \frac{(x_2 y_3 - y_2 x_3) + x(y_2 - y_3) + y(x_3 - x_2)}{2A_e}$$

$$N_2(\vec{m}) = \frac{1}{2A_e} \det \begin{bmatrix} 1 & 1 & 1 \\ x & x_3 & x_1 \\ y & y_3 & y_1 \end{bmatrix} = \frac{(x_3 y_1 - y_3 x_1) + x(y_3 - y_1) + y(x_1 - x_3)}{2A_e}$$

$$N_3(\vec{m}) = \frac{1}{2A_e} \det \begin{bmatrix} 1 & 1 & 1 \\ x & x_1 & x_2 \\ y & y_1 & y_2 \end{bmatrix} = \frac{(x_1 y_2 - y_1 x_2) + x(y_1 - y_2) + y(x_2 - x_1)}{2A_e}$$

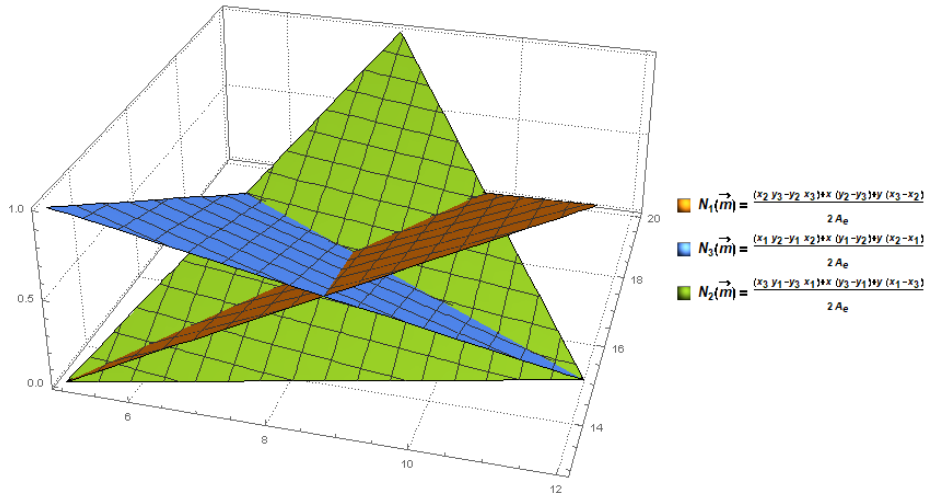


Figure 3.2: Graphs of the linear polynomials for triangle with vertices $v_1 = (12, 15)^T$, $v_2 = (8, 20)^T$ and $v_3 = (5, 13)^T$

$$\frac{\partial N_e}{\partial \vec{m}} = \frac{\partial}{\partial \vec{m}} \begin{bmatrix} N_1 \\ N_2 \\ N_3 \end{bmatrix} = \frac{1}{2A_e} \begin{bmatrix} \frac{\partial N_1}{\partial x} & \frac{\partial N_1}{\partial y} \\ \frac{\partial N_2}{\partial x} & \frac{\partial N_2}{\partial y} \\ \frac{\partial N_3}{\partial x} & \frac{\partial N_3}{\partial y} \end{bmatrix} = \frac{1}{2A_e} \begin{bmatrix} y_2 - y_3 & x_3 - x_2 \\ y_3 - y_1 & x_1 - x_3 \\ y_1 - y_2 & x_2 - x_1 \end{bmatrix}$$

We want to express $\vec{\epsilon_e}(\vec{u}_h)$ in terms of the shape functions and function values $\vec{u}(\vec{v}_i) = \vec{q}_i = (q_{i_x}, q_{i_y})^T$ in the mesh vertices.

$$\frac{\partial \vec{u}_h}{\partial x} = \frac{\partial}{\partial x} \left(\sum_{i=1}^{3} N_i(\vec{m}) \vec{q}_i \right) = \sum_{i=1}^{3} \begin{bmatrix} \frac{\partial N_i(\vec{m})}{\partial x} q_{i_x} & \frac{\partial N_i(\vec{m})}{\partial x} q_{i_y} \end{bmatrix}^T$$

$$\frac{\partial \vec{u}_h}{\partial y} = \frac{\partial}{\partial y} \left(\sum_{i=1}^{3} N_i(\vec{m}) \vec{q}_i \right) = \sum_{i=1}^{3} \begin{bmatrix} \frac{\partial N_i(\vec{m})}{\partial y} q_{i_x} & \frac{\partial N_i(\vec{m})}{\partial y} q_{i_y} \end{bmatrix}^T$$

$$\implies \vec{\epsilon_e}(\vec{u}_h) = \sum_{i=1}^{3} B_i \cdot \vec{q}_i, \text{ where } B_i = \begin{bmatrix} \frac{\partial N_i}{\partial x} & 0 \\ 0 & \frac{\partial N_i}{\partial y} \\ \frac{1}{2}\frac{\partial N_i}{\partial y} & \frac{1}{2}\frac{\partial N_i}{\partial x} \end{bmatrix}$$

$$B_e = \frac{1}{2A_e} \begin{bmatrix} y_2 - y_3 & 0 & y_3 - y_1 & 0 & y_1 - y_2 & 0 \\ 0 & x_3 - x_2 & 0 & x_1 - x_3 & 0 & x_2 - x_1 \\ x_3 - x_2 & y_2 - y_3 & x_1 - x_3 & y_3 - y_1 & x_2 - x_1 & y_1 - y_2 \end{bmatrix}$$

When we sum up B_e for elements of the mesh we get $B \in \mathbb{R}^{3n \times 3n}$ (n is the number of vertices/nodes in the triangulation) which is called the strain

matrix. Also $q \in \mathbb{R}^{3n}$ is generated from $\vec{q}_i$. Although we have written the element-wise explanation above using $\vec{u}_h$ and $\vec{\epsilon_e}(\vec{u}_h)$, the same holds for $\vec{\epsilon_e}(\vec{v})$, with that in mind we can write:

$$\epsilon = Bq, \quad \sigma = CBq, \quad v = Nq_v$$

where q_v are the values of the function $\vec{v}$ at the nodes. Substituting in Equation (3.13) we get the following linear system:

$$\left(\iint_\Omega q_v^T B^T C B d\Omega \right) q = \iint_\Omega q_v^T N^T f d\Omega + \int_{\Gamma_N} q_v^T N^T g_N ds$$

but this is correct for every q_v so we get:

$$\left(\iint_\Omega B^T C B d\Omega \right) q = \iint_\Omega N^T f d\Omega + \int_{\Gamma_N} N^T g_N ds$$

In our implementation $g_N = 0$ so we can simplify to:

$$Kq = F \tag{3.15}$$

where K is the stiffness matrix. The element stiffness matrix simplifies to $K_e = A_e B_e^T C B_e$, where A_e is the element's surface area. Let us note that if we consider the corresponding non-stationary problem and add damping, we trivially obtain the system given with Equation (3.1).

3.6 Corotational Formulation

In the previous sections, we obtained the stiffness matrix for a 2D problem. Now, we shall explain how one can transform this matrix, in order to obtain the stiffness matrix in 3D, i.e the stiffness matrix in Equation (3.1). Since it is obtained using the so-called corotational formulation, we shall further denote it with K^{CR}, i.e. Equation (3.1) will be further written as:

$$M\ddot{x} + D\dot{x} + K^{CR}(x - x_0) = f_{ext}$$

A stable simulation with a linear elasticity model can be performed efficiently with an implicit integration scheme. However, such a model is not suitable for large rotational deformations since nonlinear effects can cause undesired deformations [30]. To solve this problem a corotational formulation [17] is used similar to the one of [15]. In fact the undesired deformations in question are so prominent that before [15] and the introduction of its solution, the outlined linear elasticity model had been unusable.

In a corotational formulation the elastic forces are computed in a local 2D unrotated coordinate frame. Therefore, rotational part of the deformation needs to be extracted. In order to get the rotation matrix of a triangular element, one could determine the three-dimensional transformation $\vec{x} = A\vec{x_0}$ of an undeformed point $\vec{x_0}$ to its corresponding deformed position $\vec{x}$ and then extract the rotational part of A. However, we are only interested in the rotational transformation in the plane of the triangle, because the linear elasticity model has been defined under membrane stress/strain conditions. Therefore, the undeformed and deformed vertices are first projected in the two-dimensional space of the polygon's corresponding plane. For the triangular element in Figure 3.3b the plane vectors are determined as follows:

$$\vec{n} = \left(\vec{x}^b - \vec{x}^a\right) \times \left(\vec{x}^c - \vec{x}^a\right), \vec{p}_x = \frac{\vec{x}^b - \vec{x}^a}{|\vec{x}^b - \vec{x}^a|}, \vec{p}_y = \frac{\vec{n} \times \vec{p}_x}{|\vec{n} \times \vec{p}_x|}$$

Having those, the projection matrix[2] is:

$$P = \begin{bmatrix} \vec{p}_x^T \\ \vec{p}_y^T \end{bmatrix} \in \mathbb{R}^{2\times 3}$$

The projection matrix P_0 for the undeformed triangle is defined analogously. Two-dimensional coordinates can now be computed by $\bar{x} = P\vec{x}$ (notice $\mathbb{R}^2$ vectors are marked with a bar instead of arrow). Now we define the matrices:

$$S = \begin{bmatrix} \bar{x}_0^b - \bar{x}_0^a & \bar{x}_0^c - \bar{x}_0^a \end{bmatrix} \in \mathbb{R}^{2\times 2}$$

$$T = \begin{bmatrix} \bar{x}^b - \bar{x}^a & \bar{x}^c - \bar{x}^a \end{bmatrix} \in \mathbb{R}^{2\times 2}$$

With them we define the matrix TS^{-1} which contains the transformation of an undeformed point $\bar{x}_0$ to its corresponding deformed position $\bar{x}$, but without the translational part. Hence, the required rotation matrix $R \in \mathbb{R}^{2\times 2}$ can be extracted by a simple 2D polar decomposition of TS^{-1} explained in detail in Appendix C.

Instead of computing the forces for an element from Section 3.5 directly, in the corotational formulation we first rotate a vertex back to a local coordinate frame (see Figure 3.3c). Then the linear forces are computed and the results are transformed to the world coordinate frame (see Figure 3.3d). This transformation can be combined with the projection matrix P in order to project the 3D coordinates of a vertex in the 2D space of the corresponding triangle. The transposed projection matrix P^T is used to transform the

[2]Note we have talked about the transformation matrix A and now we talk about the projection matrices P and P_0. With all of that a rotation matrix R will be derived. All of those matrices are per mesh triangle element.

$\mathbb{R}^2$ forces of Equation(10) to $\mathbb{R}^3$. $R_{P,e}$ combines the projection and rotation for all three vertices of a triangular element.

$$R_{P,e} = \begin{bmatrix} P^T R & 0 & 0 \\ 0 & P^T R & 0 \\ 0 & 0 & P^T R \end{bmatrix} \in \mathbb{R}^{9\times 6}$$

Using $R_{P,e}$ the following element matrices are defined as:

$$K_e^{CR} = R_{P,e} K_e R_{P,e}^T \in \mathbb{R}^{9\times 9} \text{ and } \hat{K}_e^{CR} = R_{P,e} K_e \in \mathbb{R}^{9\times 6}$$

K_e^{CR} is called the element corotated stiffness matrix. $\hat{K}_e^{CR}$ is applied to the initial $\mathbb{R}^2$ positions $\bar{x}_0^i$, where $i = \overline{1,n}$. We can use $\mathbb{R}^3$ positions $\vec{x}_0^i$, where $i = \overline{1,n}$, but this simplifies things a bit.

Using K_e^{CR} and $\hat{K}_e^{CR}$ the finite element system (3.15) is transformed to $F^{CR} = K^{CR}x - \hat{K}^{CR}\bar{x}_0$, where $\bar{x}_0 = \begin{bmatrix}\bar{x}_0^0 & \dots & \bar{x}_0^n\end{bmatrix}^T \in \mathbb{R}^{2n}$. Note that in contrast to the system (3.15) the resulting forces here $\in \mathbb{R}^3$ since the projection is integrated in the corotational formulation.

3.7 Numerical Integration

Having the results from the previous sections, the system (3.1) can now be written as:

$$Ma + Dv + K^{CR}x - F_0^{CR} = f_{ext}$$

where the vectors f_{ext}, x and v contain the external forces, positions and velocities for each vertex and all $\in \mathbb{R}^{3n}$, K^{CR} is the corotated stiffness matrix from Section 3.6 and $F_0^{CR} = \hat{K}^{CR}\bar{x}_0$. From the backward difference formula we have:

$$a_{n+1} = \frac{v_{n+1} - v_n}{\Delta t}, \quad v_{n+1} = \frac{x_{n+1} - x_n}{\Delta t}$$

We substitute a_{n+1} and $x_{n+1} = x_n + \Delta t v_{n+1}$:

$$M\frac{v_{n+1} - v_n}{\Delta t} + Dv_{n+1} + K^{CR}(x_n + \Delta t v_{n+1}) = f_{ext,n} + F_0^{CR}$$

Now we multiply both sides by Δt:

$$M(v_{n+1} - v_n) + \Delta t D v_{n+1} + \Delta t K^{CR}(x_n + \Delta t v_{n+1}) = \Delta t f_{ext,n} + \Delta t F_0^{CR}$$

$$M(v_{n+1} - v_n) + \Delta t D v_{n+1} + \Delta t^2 K^{CR} v_{n+1} = \Delta t f_{ext,n} + \Delta t F_0^{CR} - \Delta t K^{CR} x_n$$

Now we subtract $\Delta t D v_n$ and $\Delta t^2 K^{CR} v_n$ from both sides:

$$M(v_{n+1} - v_n) + \Delta t D(v_{n+1} - v_n) + \Delta t^2 K^{CR}(v_{n+1} - v_n) = \\ = \Delta t f_{ext,n} + \Delta t F_0^{CR} - \Delta t K^{CR} x_n - \Delta t D v_n - \Delta t^2 K^{CR} v_n$$

We denote $\Delta v = v_{n+1} - v_n$ and substitute it in the equation:

$$M\Delta v + \Delta t D \Delta v + \Delta t^2 K^{CR} \Delta v = \\ = \Delta t f_{ext,n} + \Delta t F_0^{CR} - \Delta t K^{CR} x_n - \Delta t D v_n - \Delta t^2 K^{CR} v_n$$

After a final rewriting we arrive at the following form:

$$\left(M + \Delta t D + \Delta t^2 K^{CR}\right) \Delta v = \\ = -\Delta t \left(K^{CR} x_n - F_0^{CR} - f_{ext,n} + \Delta t K^{CR} v_n + D v_n\right) \quad (3.16)$$

To perform the time integration the velocity change Δv is determined by solving the linear system shown in Equation (3.16). This should provide stable time integration of the stiff equations even for large time steps. After solving the linear system for Δv we get the position change by $\Delta x = \Delta t(v_n + \Delta v)$.

3.8 Results

As we know by now the model requires a certain set of physically based coefficients. Measuring them can be tricky and expensive in terms of hardware needed(check Appendix D). In the implementation we have addressed this by importing existing and tested value listed in Tables 3.1 and 3.2 plus giving full user control over them to the user.

	$\rho[kg/m^2]$	$E_x[Pa]$	$E_y[Pa]$	ν_{xy}	ν_{yx}	$E_s[Pa]$
Viscose	0.23	245	366	0.249	0.167	0.38
Wool	0.26	866	1391	0.261	0.162	0.51
Polyacrylics	0.17	3057	1534	0.150	0.299	1.22
Polyester	0.26	2400	3600	0.250	0.167	5.23

Table 3.1: Fabric parameters taken from [15]

In order to make results from Chapter 2 and Chapter 3 comparable, experiments in this section use a square patch of cloth similar to Section 2.3

	ρ $[kg/m^2]$	E_x $[MPa]$	E_y $[MPa]$	ν_{xy}	ν_{yx}	E_{45} $[MPa]$	ν_{45}
100% Cotton	0.1503	32.559	12.436	0.566	0.216	0.821	1.136
100% Wool	0.2348	21.860	8.149	0.705	0.263	0.170	1.377
95% Wool + + 5% lycra	0.1782	0.250	0.851	0.071	0.244	0.076	0.599
100% Polyester	0.1646	5.152	11.674	0.381	0.864	0.478	1.366

Table 3.2: Fabric parameters taken from [34], where $E_s = \frac{E_{45}}{2(1+\nu_{45})}$

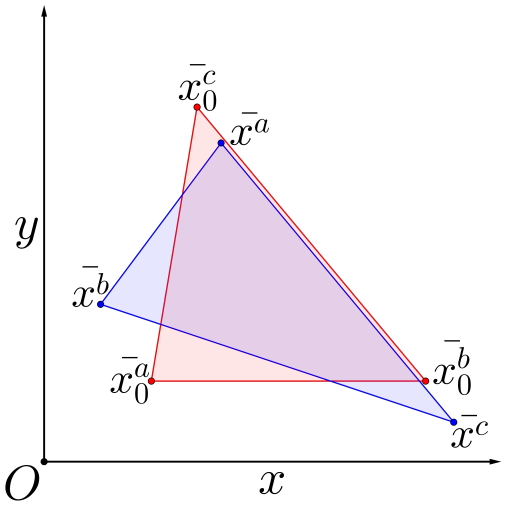
with a side of $5.0m$. The mesh[3] has 144 vertices and 242 right triangles. This results in a system with 432 linear equations. No wind is used and material parameters are taken from Table 3.1.

Figures 3.4a and 3.4b: These experiments were conducted using a relatively small time step. Their main goal is to highlight differences in produced results, based on material properties. Wool being the stiffer textile of the two sags far less, although it is more dense.

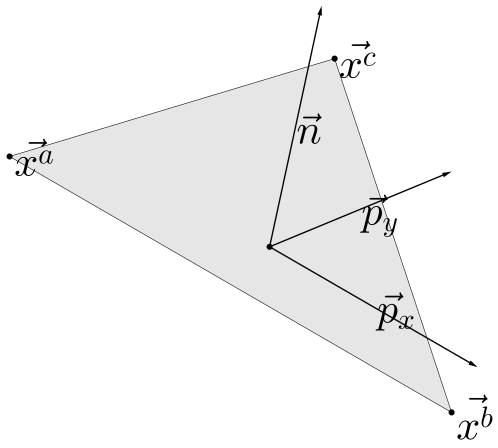
Figure 3.4c: This experiment uses a larger time step. Although wool's material properties were used and all other coefficients are kept the same, it looks different from Figure 3.4a. Oscillations and other instabilities were expected, but the more pronounced sagging is not and requires more investigation.

Figure 3.4d: This experiment uses the same parameters as the one in Figure 3.4a, except for larger stiffness damping coefficient β. The observation here is that model's results are extremely sensitives to damping parameters and elastic behavior changes a lot due to them. Notice that C_{damp} in Chapter 2 has a more pronounced effect on cloths movement rather than elasticity.

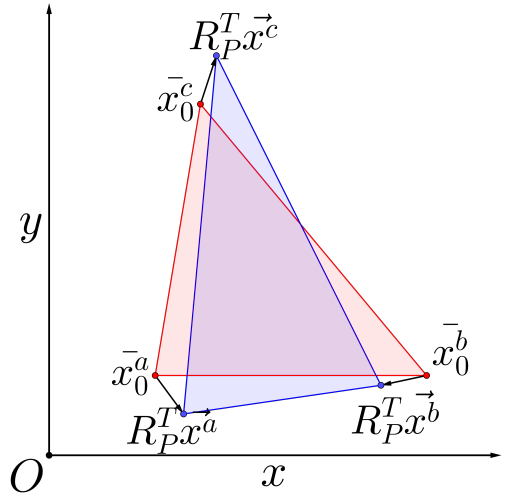
[3]We do not discuss triangulation because meshes in rendering software usually come triangulated.



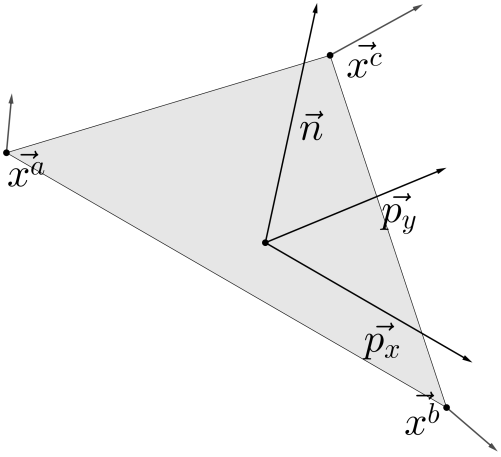
(a) Rest state $\bar{x}_0$ and deformed state $\bar{x}$ in 2D.



(b) Finite element polygon plane vectors.



(c) Displacement in 2D: $R_P^T x - \bar{x}_0$.



(d) $F_e^{CR} = R_P K_e (R_P^T x - \bar{x}_{0,e})$.

Figure 3.3: Finite elements in corotational formulation.

(a) Simulating wool with:

- $\Delta t = 0.003s$
- $\alpha = 0.2$, $\beta = 0.01$

.

(b) Simulating viscose with:

- $\Delta t = 0.003s$
- $\alpha = 0.2$, $\beta = 0.01$

.

(c) Simulating wool with:

- $\Delta t = 0.099s$
- $\alpha = 0.2$, $\beta = 0.01$

.

(d) Simulating wool with:

- $\Delta t = 0.003s$
- $\alpha = 0.2$, $\beta = 0.05$

.

Figure 3.4: Results using the presented finite element method.

Chapter 4

Technical Aspects

We have implemented the two different approaches, described in the previous two chapters, and developed a library of tools that allow for a user to create and interact with objects inside the physical simulation (sometimes referred to as "scene"). These tools include moving solid objects and controlling the cloth patch using the mouse. Here, we shall give some technical details with specific recommendation that could be useful if one tries to develop a similar application.

4.1 Vertex Storage

Vertex storage is one of those things every paper on the topic skips. In the initial implementation of the computational model from Chapter 2, we did not pay attention to how vertices were stored, because they are a regular grid and can be stored in a two dimensional array. Adjacency and topological information in that setup is trivial to extract. When implementing the model from Chapter 3 which works on arbitrary meshes it became obvious a better solution was needed. The data structure from Section 4.1.3 was implemented and used for both models. In the following chapters we will give reasoning about why we chose that particular storage strategy.

There is an array of options to choose from and not every method fits a particular model. Some containers can make a model's implementations simpler, others can make it run orders of magnitude faster. An algorithm might even need different or multiple data structures, for example when using GPGPU computations. In terms of classification there are constructive solid geometry(CSG) [18] and boundary representation(B-rep) [47]. CSG uses only primitive objects and boolean operators to combine them. B-rep of a model comprises of topological components (faces, edges and vertices) and

the connections between them. They are a lot more flexible and have a much richer operation set.

We will list some of the more popular general purpose B-rep structures and comment on their pros and cons. CSG options are practically unusable for our purposes. This is not meant to be neither an exhaustive list nor a guide, it is more of a reasoning for what is used in the presented implementations.

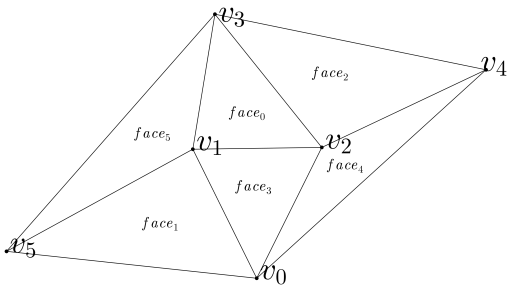


Figure 4.1: Example mesh faces and vertices.

4.1.1 Index/Vertex Buffer Data Structure

Represents a mesh as a set of vertices (vertex buffer) and a set of faces (index buffer). This is the most widely used mesh representation, being the input typically accepted by modern graphics hardware. It is useful as a storage format, but for more complex usages one will usually build another representation once loaded. Although in our implementation we store the cloth mesh in a data structure based on Section 4.1.3, each frame we convert it to this representation in order to render it.

The data structure stores mesh's vertex data, such as positions (x_i, y_i, z_i), in an array "vertex buffer" (see Table 4.1). An implementation can keep any additional per vertex data in that array. Multiple arrays can also be used as a cache locality optimization when different parts of an algorithm need only a subset of vertex's data (in this case vertex order needs to be kept consistent across the arrays). In addition to vertex data arrays an additional array is stored called "index buffer" (see Table 4.2). This array stores a stream of triplets. Each element in such a triplet is an index in the vertex buffer which in turn corresponds to a specific vertex. Each triplet is a mesh triangle, and can be stored in an arbitrary order in the index buffer.[1]

[1]There is a whole class of optimization software which reorders the triplets in the index buffer based on heuristics. Some of those heuristics aim to improve compression ratios

v_0			v_1			v_2			v_3		
x_0	y_0	z_0	x_1	y_1	z_1	x_2	y_2	z_2	x_3	y_3	z_3
			v_4			v_5					
			x_4	y_4	z_4	x_5	y_5	z_5			

Table 4.1: Vertex buffer, use Figure 4.1 for reference.

$face_0$			$face_1$			$face_2$			$face_3$		
v_1	v_2	v_3	v_5	v_0	v_1	v_2	v_4	v_3	v_0	v_2	v_1
			$face_4$			$face_5$					
			v_0	v_4	v_2	v_5	v_1	v_3			

Table 4.2: Index buffer Mesh, use Figure 4.1 for reference.

Pros:	**Cons:**
• Has lowest memory overhead. • Going through all faces and all vertices is trivial and has practically zero computational overhead. • Natively supported by all GPUs' input assembly stage.	• Extracting any kind of adjacency information is hard. • Most operations have $O(n)$ complexity where n is the vertex count, face count or their sum, because no meta-data is stored to accelerate the process.

4.1.2 Winged-Edge Data Structure

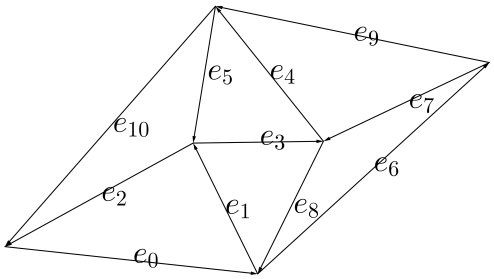
Perhaps the oldest data structure for a B-rep is Baumgart's winged-edge data structure [2]. It uses edges to keep track of almost everything. Its main appeal is the $\mathcal{O}(1)$ complexity when doing adjacency queries, such as getting face's vertices or edges, vertex's neighboring faces, edges or vertices, etc. Here, we shall only sketch the idea of this representation, because we will only use it, in order to better understand the structure, presented in Section 4.1.3 which is a very similar, but better option.

Face's vertices can be ordered[2] either clockwise or counter-clockwise[3] as long as the same convention is used throughout. A key observation is that edges form a circular linked list around faces, this is important for the algorithms which extract adjacency information. In the list below we explain

which helps when mesh is stored on media devices. Others optimize for GPU rasterisation performance.

[2]The vertex order inside a mesh's face is called winding.

[3]Counter-clockwise order is used in the examples.



e_0								e_1							
v_5	v_0	f_1	*nil*	e_2	e_1	e_6	e_{10}	v_0	v_1	f_1	f_3	e_0	e_2	e_3	e_8
e_2								e_3							
v_1	v_5	f_1	f_5	e_1	e_0	e_{10}	e_5	v_1	v_2	f_0	f_3	e_5	e_4	e_8	e_1
e_4								e_5							
v_2	v_3	f_0	f_2	e_3	e_5	e_9	e_7	v_3	v_1	f_0	f_5	e_4	e_3	e_2	e_{10}
e_6								e_7							
v_0	v_4	f_4	*nil*	e_8	e_7	e_9	e_0	v_4	v_2	f_4	f_2	e_6	e_8	e_4	e_9
e_8								e_9							
v_2	v_0	f_4	f_3	e_7	e_6	e_1	e_3	v_4	v_3	f_2	*nil*	e_7	e_4	e_{10}	e_6

e_{10}							
v_3	v_5	f_5	*nil*	e_5	e_2	e_0	e_9

Table 4.3: Winged-edge mesh edges, use Figures 4.1 and 4.2 for reference.

each edge data entry (numbering corresponds to the columns of an edge in Table 4.3):

1. Called "previous vertex" and denoted $Vert_P$ in Figure 4.2. Edge arrows in the figure start from this vertex. For every edge it is chosen arbitrarily and defines uniquely how the rest of the data entries below will be ordered[4].

2. Called "next vertex" and denoted $Vert_N$ in Figure 4.2. Edge arrows in the figure end at this vertex.

3. Called left face and denoted $Face_L$ in Figure 4.2. Each edge has at most two adjacent faces. $Face_L$ is the one in which if vertices are listed in winding order $Vert_P$ will come before $Vert_N$.

4. Called right face and denoted $Face_R$ in Figure 4.2. If an edge has a second adjacent face it is stored here, otherwise *nil* or other identifiable

[4]Order in the sense what will be stored as left or right face or vertex

value is stored.

5. Called "left face previous edge" and denoted $Edge_{LP}$ in Figure 4.2. This is the edge which stores $Vert_P$ as either previous or next vertex and stores $Face_L$ as either left or right face.[5]

6. Called "left face next edge" and denoted $Edge_{LN}$ in Figure 4.2. This is the edge which stores $Vert_N$ as either previous or next vertex and stores $Face_L$ as either left or right face.[5]

7. Called "right face previous edge" and denoted $Edge_{RP}$ in Figure 4.2. This is the edge which stores $Vert_N$ as either previous or next vertex and stores $Face_R$ as either left or right face.[5]

8. Called "right face next edge" and denoted $Edge_{RN}$ in Figure 4.2. This is the edge which stores $Vert_P$ as either previous or next vertex and stores $Face_R$ as either left or right face.[5]

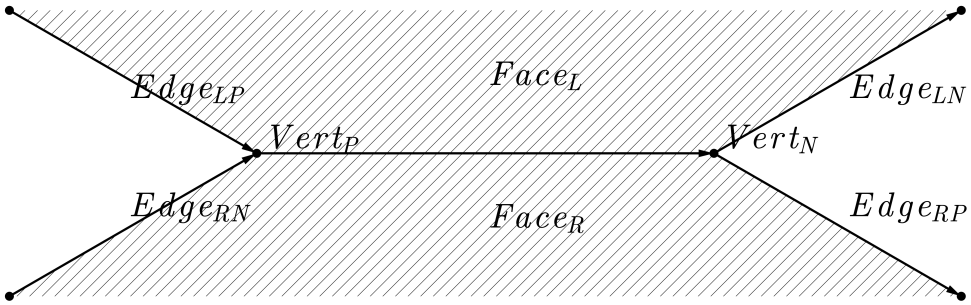


Figure 4.2: Winged-edge data structure edge links.

Table 4.4 shows that apart from position (x_i, y_i, z_i), vertices store one edge for which the vertex is parent. Table 4.5 shows that for each face only one of its border edges is stored. In practice those structures will store additional domain specific data. Note the similarities with the half-edge data structure.

[5]This formulation is valid even when $Edge_{LN}$ is nil. Also notice how the vertex and face in question are not stored in a predefined column. This makes adjacency queries use more conditional statements. The storage solution presented in Section 4.1.3 addresses this issue.

v_0				v_1				v_2				v_3			
x_0	y_0	z_0	e_1	x_1	y_1	z_1	e_2	x_2	y_2	z_2	e_4	x_3	y_3	z_3	e_5

v_4				v_5			
x_4	y_4	z_4	e_7	x_5	y_5	z_5	e_0

Table 4.4: Winged-edge mesh vertices, use Figure 4.1 and Table 4.3 for reference.

f_0	f_1	f_2	f_3	f_4	f_5
e_3	e_0	e_4	e_1	e_6	e_2

Table 4.5: Winged-edge mesh faces, use Figure 4.1 and Table 4.3 for reference.

Pros:	**Cons:**
• Cloth model specific data is easy to store. • Extracting both face, edge and vertex adjacency information is fast. • Very popular and has tons of great resources to use.	• Adjacency queries can get tricky to implement. • Has relatively high memory overhead. • Hard to use on GPU.

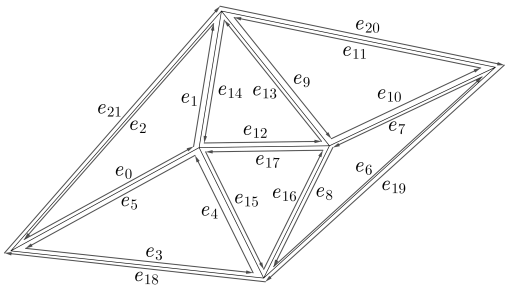
4.1.3 Half-Edge Data Structure

Also known as doubly connected edge list(DCEL) [29]. Originally the data structure is only capable of representing manifold surfaces, this means that every edge is bordered by exactly two faces. T-junctions, internal polygons, and breaks in the mesh are not allowed. Not having breaks in the mesh would make the representation unusable for us, however it is trivially extensible.

The half-edge data structure is called that because instead of storing the edges of the mesh, we store half-edges. As the name implies, a half-edge is a half of an edge and is constructed by splitting an edge down its length. We will call the two half-edges that make up an edge a pair(some sources call them twins). Half-edges are directed and the two edges of a pair have opposite directions.

Face's vertices can be ordered either clockwise or counter-clockwise[6] as long as the same convention is used throughout. A key observation is that edges form a circular linked list around faces, this is important for the algorithms which extract adjacency information (see Algorithms 1 to 3). In the

[6]Counter-clockwise order is used in the examples.



e_0				e_1				e_2				e_3			
v_5	e_5	f_5	e_1	v_1	e_{14}	f_5	e_2	v_3	e_{21}	f_5	e_0	v_5	e_{18}	f_1	e_4
e_4				e_5				e_6				e_7			
v_6	e_{15}	f_1	e_5	v_1	e_0	f_1	e_3	v_0	e_{19}	f_4	e_7	v_4	e_{10}	f_4	e_8
e_8				e_9				e_{10}				e_{11}			
v_2	e_{16}	f_4	e_6	v_3	e_{13}	f_2	e_{10}	v_2	e_7	f_2	e_{11}	v_4	e_{20}	f_2	e_9
e_{12}				e_{13}				e_{14}				e_{15}			
v_1	e_{17}	f_0	e_{13}	v_2	e_9	f_0	e_{14}	v_3	e_2	f_0	e_{12}	v_1	e_4	f_3	e_{16}
e_{16}				e_{17}				e_{18}							
v_0	e_8	f_3	e_{17}	v_2	e_{12}	f_3	e_{14}	v_0	e_3	*nil*	e_{21}				
e_{20}				e_{21}				e_{19}							
v_3	e_{11}	*nil*	e_{19}	v_5	e_2	*nil*	e_{20}	v_4	e_6	*nil*	e_{18}				

Table 4.6: Half-edge mesh edges, use Figures 4.1 and 4.3 for reference.

list below we explain each half-edge data entry (numbering corresponds to the columns of an edge in Table 4.6):

1. As previously stated, half-edges are directed. This entry stores the "parent vertex". Notice the child vertex is stored implicitly and it can be extracted from the parent vertex of the pair half-edge stored in the next data entry. Half-edge arrows in Figure 4.3 start from the vertex stored here.

2. Called "pair half-edge". This entry holds the half-edge in the opposite direction of the current one.

3. Each edge in the mesh has at most two adjacent faces. This data entry holds the one in which half-edge direction is the same as face winding direction. If there is no such face *nil* or other identifiable value is stored.

4. Imagine listing the vertices of the face stored in the previous data entry based on winding starting with vertex from the first entry. This data

entry holds the half-edge which has as parent the second vertex in the sequence and ends at the third. If the stored face is *nil* we take the half-edge which stores *nil* as face and has "parent vertex" the implicit child vertex of this half-edge.

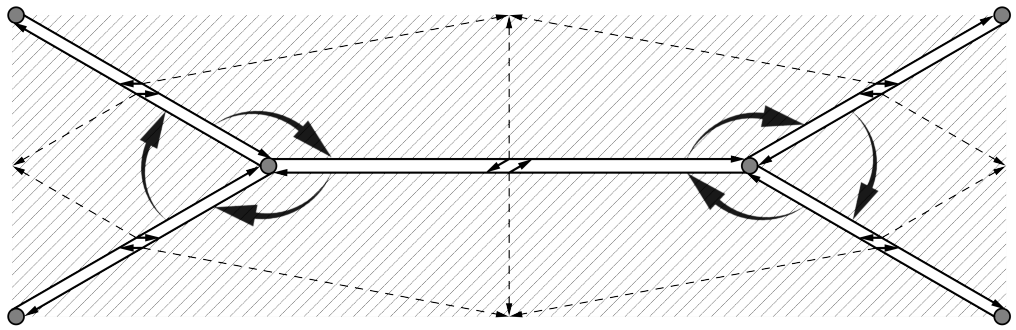


Figure 4.3: Half-edge data structure edge links.

Table 4.7 shows that apart from position (x_i, y_i, z_i), vertices store one arbitrary half-edge for which the vertex is parent. Table 4.8 shows that for each face only one of its border half-edges is stored. In practice those structures will store additional domain specific data. Note the similarities with the winged-edge data structure.

v_0				v_1				v_2				v_3			
x_0	y_0	z_0	e_6	x_1	y_1	z_1	e_{12}	x_2	y_2	z_2	e_{10}	x_3	y_3	z_3	e_2

v_4				v_5			
x_4	y_4	z_4	e_{11}	x_5	y_5	z_5	e_3

Table 4.7: Half-edge mesh vertices, use Figure 4.1 and Table 4.6 for reference.

f_0	f_1	f_2	f_3	f_4	f_5
e_{12}	e_3	e_9	e_{15}	e_6	e_0

Table 4.8: Half-edge mesh faces, use Figure 4.1 and Table 4.6 for reference.

The answers to most adjacency queries are stored directly in the data structures for the edges, vertices and faces and do not need further explanation, however we will give the procedures for other often needed queries. Iterating over the half-edges adjacent to a face can be seen in Algorithm 1. Since the half-edges around a face form a circular linked list, and the face

Algorithm 1 Face Adjacent Edges

```
1: procedure FaceToEdges(f)
2:     E ← ∅
3:     e ← Edge (f)
4:     repeat
5:         E ← E ∪ e
6:         e ← NextEdge (e)
7:     until e ≠ Edge (f)
8:     return E
9: end procedure
```

structure stores a pointer to one of these half-edges we simply need to iterate the list and terminate appropriately. Note that in these iterating procedures checks for empty elements are not needed. This is because we have only added breaks in the mesh through *nil* faces and we kept most manifold properties.

Algorithm 2 Vertex Adjacent Edges

```
1: procedure VertexToEdges(v)
2:     E ← ∅
3:     e ← Edge (v)
4:     repeat
5:         E ← E ∪ e   ▷ Returns only adjacent half-edges with this vertex,
   the other half are twin edges.
6:         e ← NextEdge (Pair (e))
7:     until e ≠ Edge (v)
8:     return E
9: end procedure
```

Similarly, we might be interested in iterating over the edges (Algorithm 2) or vertices (Algorithm 3) which are adjacent to a particular vertex. Referring to Figure 4.3, one will see that in addition to the circular linked lists around the borders of the faces, the pointers also form loops around the vertices.

For some reason every source which compares half-edge vs winged-edge data structures explains that the latter one is more storage efficient, but that is not correct. In fact implementations should spend the same amount of memory, but arranged differently. Those same sources also mention half-edge's better processing efficiency in which they are correct, various iterating procedures are simpler to implement and contain less conditional branching

Algorithm 3 Vertex Adjacent Vertices

```
1: procedure VertexToVertices(v)
2:     V ← ∅
3:     e ← Edge (v)
4:     repeat
5:         V ← V ∪ Vertex (Pair (e))
6:         e ← NextEdge (Pair (e))
7:     until e ≠ Edge (v)
8:     return V
9: end procedure
```

due to edges having direction and belonging to a single face and vertex. Therefore, our B-rep of choice is the half-edge data structure.

Pros:	Cons:
• Cloth model specific data is easy to store. It is even possible to store edge data for an adjacent face. • Extracting both face, edge and vertex adjacency information is really fast and trivial.	• Has relatively high memory overhead. • Hard to use on GPU.

4.2 Rendering

Displaying the results and various models' information is interesting in and of itself. For the purpose of rendering Falcor [6] is used. It is a real-time rendering framework supporting DirectX 12. According to developers its aim is to improve productivity of research and prototype projects. The framework exposes a simple physically based material shading. Our lighting is done by a single controllable directional light. In order to shade a material in a scene we need to provide some properties, such as surface normal, diffuse and specular color, roughness etc. Most of those are constant and come from a texture, however surface normals need to be computed on the fly, otherwise delicate wrinkles will not be noticeable. Normals are computed at each vertex using Algorithm 4 and interpolated inside a polygon.

If that was all we did the result would look odd. This is due to material translucency. It is a well researched effect. Translucency is caused by light passing through an object. Most shading models simulate only reflected light which makes translucent materials look dark. We address this by using a popular approximation for real-time renderers. When shading a point we

Algorithm 4 Vertex Normal

1: **procedure** VERTEXNORMAL(v_i)
2: $N_i \leftarrow 0$
3: **for all** triangles t adjacent to v_i **do**
4: $N_t \leftarrow$ normal of triangle t
5: $N_i \leftarrow N_i + N_t$
6: **end for**
7: **end procedure**
8: **return** $\frac{N_i}{|N_i|}$

artificially make the normal more parallel with the incident light direction. Depending on how close the two directions get the more translucent the material looks. We will not go into more detail here, but just illustrate the idea and the result in Figure 4.4.

4.3 User Interaction

A direct way of interaction with the simulated cloth is available. The user can "pinch" a point from the cloth defined by the cursor position and drag the point around using cursor movement. This tool can be seen in the video of Experiment 3 in Section 2.3.

In Section 4.3.1 we explain the procedure used to chose a point, in the cloth's mesh, to be manipulated. The procedure is executed when the user pushes the left mouse button and it is based on a ray-triangle intersection query.

Once a point from the cloth is picked the cloth model needs to be affected appropriately. This cannot be done using a boundary condition, because the chosen point might not be on the boundary. It can be done using external forces. However, the way we do it is by changing the closest particle's position to a new position defined my cursor's movement. Section 4.3.2 presents the algorithm used to interpret the cursor's movement.

4.3.1 Finding Pinched Point

This is a classical closest ray-mesh intersection problem. The ray origin and direction needed for intersection queries are derived from the cursor's position on screen, camera position and orientation in the simulation. The ray origin is actually the camera's position, the direction however is harder to extract.

Using Figure 4.5 notation the equation we derived is:

$$D = c_w + c_u\left(2m_x - 1\right) + c_v\left(1 - 2m_y\right)$$

- c_w - camera's world space forward direction with magnitude the image plane's distance from origin.
- c_u - camera's world space right direction with magnitude half image plane's width.
- c_v - camera's world space up direction with magnitude half image plane's height.
- (m_x, m_y) - cursor texture coordinates in the image plane.

Luckily due to GPU rendering constraints we will always have a triangulated version of our mesh, additionally all presented models work only on triangulated surfaces. With that in mind we only need to handle ray-triangle intersections. The intersection algorithm used [28] has been chosen based on the following premises:

- User interaction means soft real-time which in turn means intersection needs to be fast.
- Some simulated meshes can be made of thousands of polygons which puts even more pressure on performance.
- We want the feature to be implemented in a non-intrusive manner, so in-place method without caching or preprocessing are preferable.

The Möller–Trumbore ray-triangle intersection algorithm is surprisingly simple to derive given how fast it is. It takes advantage of the parameterization of the intersection point in terms of barycentric coordinates and in terms of the ray:

$$T\left(u, v\right) = \left(1 - u - v\right)V_0 + uV_1 + vV_2$$

$$O + tD = \left(1 - u - v\right)V_0 + uV_1 + vV_2$$

where V_0, V_1 and V_2 are the tested triangle's vertices and $T\left(u, v\right)$ are all its points. From earlier the ray's origin is O and D is its direction. t is how far along the ray a point is. We can rearrange the second equation's terms:

$$\begin{bmatrix} -D & V_1 - V_0 & V_2 - V_0 \end{bmatrix} \begin{bmatrix} t \\ u \\ v \end{bmatrix} = O - V_0$$

$$M = \begin{bmatrix} -D & V_1 - V_0 & V_2 - V_0 \end{bmatrix}$$
$$x^T = \begin{bmatrix} t & u & v \end{bmatrix}$$
$$b = O - V_0$$

The intersection if there is any is the solution to this linear system. Note we will also need to check if $t > 0$ and $0 \leq u, v \leq 1$ and t is intersection's distance. It is a system of the form $Mx = b$ and we know we can solve those. The secret sauce is how we solve it. For this we start by applying Cramer's rule with which we get:

$$t = \frac{\begin{vmatrix} b & E_1 & E_2 \end{vmatrix}}{\begin{vmatrix} -D & E_1 & E_2 \end{vmatrix}}$$
$$u = \frac{\begin{vmatrix} -D & b & E_2 \end{vmatrix}}{\begin{vmatrix} -D & E_1 & E_2 \end{vmatrix}}$$
$$v = \frac{\begin{vmatrix} -D & E_1 & b \end{vmatrix}}{\begin{vmatrix} -D & E_1 & E_2 \end{vmatrix}}$$

Where $E_1 = V_1 - V_0$ and $E_2 = V_2 - V_0$. From linear algebra we know that $\begin{vmatrix} A & B & C \end{vmatrix} = -(A \times C) \cdot B = -(C \times B) \cdot A$, thus we arrive at the following form:

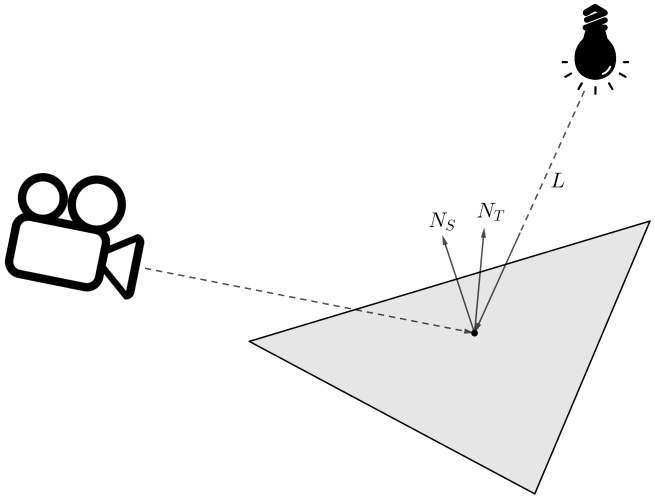
$$\begin{bmatrix} t \\ u \\ v \end{bmatrix} = \frac{1}{\begin{vmatrix} -D & E_1 & E_2 \end{vmatrix}} \begin{bmatrix} \begin{vmatrix} b & E_1 & E_2 \end{vmatrix} \\ \begin{vmatrix} -D & b & E_2 \end{vmatrix} \\ \begin{vmatrix} -D & E_1 & b \end{vmatrix} \end{bmatrix} =$$
$$= \frac{1}{(D \times E_2) \cdot E_1} \begin{bmatrix} (b \times E_1) \cdot E_2 \\ (D \times E_2) \cdot b \\ (b \times E_1) \cdot D \end{bmatrix} = \frac{1}{P \cdot E_1} \begin{bmatrix} Q \cdot E_2 \\ P \cdot b \\ Q \cdot D \end{bmatrix}$$

Where $P = D \times E_2$ and $Q = b \times E_1$. An implementation of the algorithm would compute P and Q only once, therefore the whole thing costs two cross products, a bunch of dot products and a ton of conditional statements.

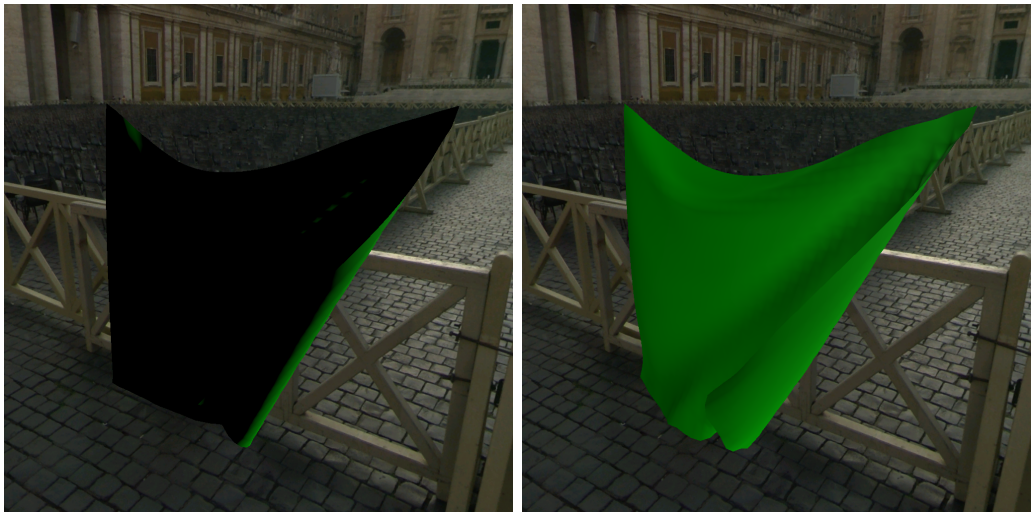
4.3.2 Cursor Delta to Cloth Displacement

Once a point P_0 is pinched every time the cursor moves, a displaced point P is computed. We define P as the intersection of the cursor's ray from the previous section and the plane through P_0 parallel to the image plane, check Figures 4.5 and 4.6. This procedure gives pretty intuitive results.

$$P = O - tD, \text{ where } t = \frac{n \cdot (P_0 - O)}{n \cdot D} \text{ and } n = \frac{(c_u \times c_v)}{|c_u \times c_v|}$$



(a) Visualization of the technique. N_S is the surface normal, L is the unit length vector from the light source to the currently shaded surface pixel. N_T is the modified surface normal.



(b) Cloth shading before.

(c) Cloth shading after.

Figure 4.4: Translucency approximation.

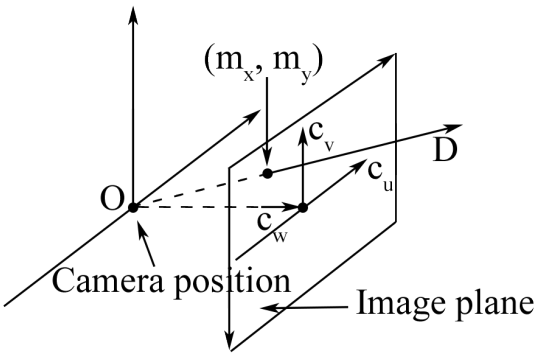


Figure 4.5: Visualization of cursor to ray conversion.

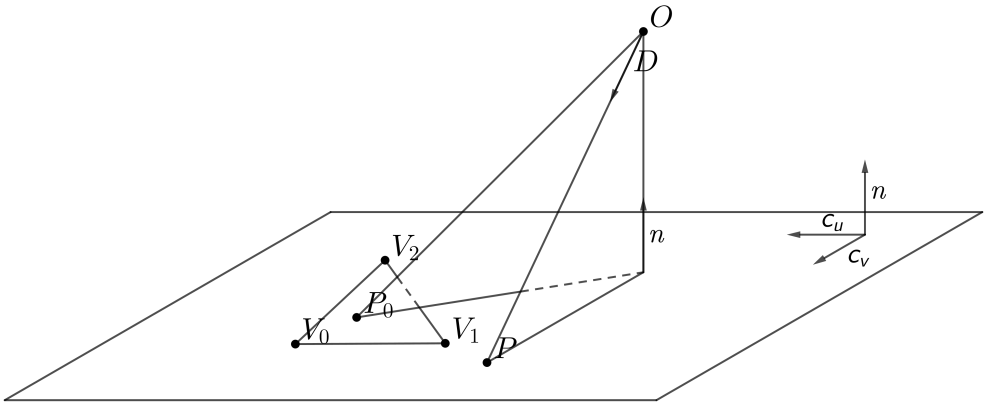


Figure 4.6: Visualization of cursor to displacement conversion.

Chapter 5

Conclusion

Now that we have the theory in place plus working and tested implementations it is time to give some general recommendations plus a comparison chart:

1. For real time applications use a computational model from Chapter 2. They are fast and simple to implement. Also they can be experimented with easily, thus making them a useful tool for developing tools and other models (more complicated solutions have high implementation overhead, also they might not run at real time so performing tests will be harder).
2. For offline applications use a model from Chapter 3. They are far more complex, but have less limitations.
3. In both cases go for a Half-Edge based vertex storage solution.
4. The devil is in the details. Simply moving particles around based on internal forces is not sufficient. The cloth needs to interact with the environment through external forces in a meaningful way. This means a good wind model and collision detection system are a must.

Chapter 2:	**Chapter 3:**
• Less computationally intensive. • Visual plausibility. • Narrow range of textiles.	• More computationally intensive. • Physical correctness. • Large range of textiles.

As a project with limited time and resources the presented work only scratches the surface of the vast body of work. A complete and robust solution requires multiple other components which handle physical interactions with the environment.

Collision Detection: Current implementation has rudimentary brute force collision detection, which handles a handful of primitives. A full fledged version would support arbitrary geometry and cloth self-collisions [11, 24]. Such a system also requires an acceleration structure such as bounding volume hierarchy [45]. Regardless of the solution and optimizations one picks, friction forces are a must, currently they are not available and collision response feels awkward.

Wind Simulation: In terms of wind control the current implementation lets the user define speed and a single direction in which the wind blows. Additionally for visual fidelity those parameters can be randomized based on a third variable. This works in the sense that it feeds external forces in the cloth simulation, however there is room for improvement. Wind can be simulated as a vector field generated by user-defined "wind sources". Furthermore, wind can be modeled as a particle system and produce various turbulent effects due to cloth affecting air currents around it.

Other Models: There is a ton of articles which deserve attention. There are nonlinear hinge models [1] (this one is actually based on [35] and uses an implicit integration scheme) and their extensions [7]. There are geometry based models such as [5] which promise improved stability. There are also modern and exotic artificial intelligence based works [23]. Position based dynamics [25, 31] is also really promising.

Computational Performance Optimizations: The presented models can be trivially parallelized, which will unlock the full potential of modern hardware. This was skipped as such optimizations usually lead to software complexity which would have hindered experiments.

When using an implicit numerical method, we always end up solving a large system of linear equations. A very popular and fast way to go about solving it are conjugate gradient methods [37]. Here are a few articles which use such a method [1, 4, 13, 15, 41, 44]. This optimization usually goes hand in hand with a sparse matrix storage solution, because as we know very well numerical methods produce matrices full of zeros.

Another promising technique is mesh subdivision [4, 32]. It is based on the idea that parts of the cloth need different level of detail, so one tries to subdivide only the parts which need it. That way we can save computation time and still have fine wrinkles.

Appendix A

Wind Model

Good wind effects can greatly improve a simulation's result, therefore, extra time was spent on improving cloth's interaction with it. Implementations share a common wind model based on work done by Disney for the movie Frozen [46]:

$$\vec{F}_{wind} = \frac{1}{2}\rho A \left[(C_D - C_L)(\vec{v}^r \cdot \vec{n})\vec{v}^r + C_L \left|\vec{v}\right|^2 \vec{n}\right]$$

where ρ is air density, A is the cloth surface area (Appendix B can be used), $\vec{n}$ is the geometry normal (Algorithm 4 can be used), $\vec{v}$ and $\vec{v}^r$ are the velocity and relative[1] velocity. C_D and C_L are separate user-controlled drag and lift coefficients and they are meant to handle object properties such as shape, material surface friction characteristics, etc. Notice this model uses quadratic terms which is a better model for fluids with high Reynolds number (e.g. air) and produces interesting turbulent effects. The lift and drag model allows for the simulation of gossamer fabrics.

The implementation has control over air temperature from which based on Table A.1, ρ is derived. However, tests showed that it has negligible effect on the overall appearance.

[1] Relative velocity is the difference between velocity coming from the wind field $\vec{v}_w$ and object velocity.

$T(°C)$	$\rho(kg/m^3)$
-25.0	1.4224
-20.0	1.3943
-15.0	1.3673
-10.0	1.3413
-5.0	1.3163
0.0	1.2922
5.0	1.2690
10.0	1.2466
15.0	1.2250
20.0	1.2041
25.0	1.1839
30.0	1.1644
35.0	1.1455

Table A.1: Data used to derive air density from temperature.

Appendix B

Voronoi Area

Definition B.0.1 (Voronoi Area)**:**
The Voronoi Area of a vertex in a triangle mesh (Figure B.1a) is the area bounded by the midpoints of the incident edges and the circumscribed circle centers of the adjacent triangles.

Theorem B.0.1:
For a non-obtuse triangle t with the vertices x_i, x_j and x_k the Voronoi Area of vertex x_i is determined by:

$$A_{Voronoi}(t, x_i) = \frac{1}{8}\left(\|x_k - x_i\|^2 \cot\phi_j + \|x_j - x_i\|^2 \cot\phi_k\right)$$

where ϕ_j , ϕ_k denotes the angle in the triangle at the vertex x_j, x_k respectively.

Proof. Figure B.1c visualizes the statement. Let O be the triangles' circumcenter. We must compute the area of the shaded region. Using the properties of perpendicular bisectors, we know that $\alpha + \beta + \gamma = \frac{\pi}{2}$, and therefore:

$$\phi_k = \beta + \gamma \implies \alpha = \frac{\pi}{2} - \phi_k \implies$$
$$\implies OH_1 = \frac{\|x_j - x_i\|}{2}\cot\phi_k \implies S_{x_i H_1 O} = \frac{1}{8}\|x_j - x_i\|^2 \cot\phi_k$$

$$\phi_j = \alpha + \gamma \implies \beta = \frac{\pi}{2} - \phi_j \implies$$
$$\implies OH_2 = \frac{\|x_k - x_i\|}{2}\cot\phi_j \implies S_{x_i H_2 O} = \frac{1}{8}\|x_k - x_i\|^2 \cot\phi_j$$

□

Summing these areas for the whole 1-ring[1] neighborhood, we can write the non-obtuse Voronoi Area for a vertex x_i visualized in Figure B.1b as a function of the neighbors x_j:

$$A_{Voronoi}(x_i) = \sum_{j \in R_1(i)} (\cot \alpha_{ij} + \cot \beta_{ij}) \left\| x_j - x_i \right\|^2$$

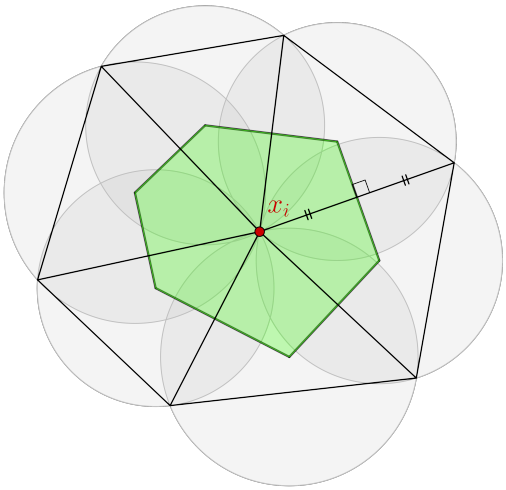
However, if there is an obtuse triangle among the 1-ring neighbors or among the triangles edge-adjacent to the 1-ring triangles, the Voronoi region either extends beyond the 1-ring, or is truncated compared to our area computation. In either case our derived formula no longer stands.

Inspired by [26] we will handle obtuse triangles. We define the new surface area for each vertex x_i, denoted A_{hybrid}: for each non-obtuse triangle, we use the circumcenter point, and for each obtuse triangle, we use the midpoint of the edge opposite to the obtuse angle. Algorithmically, this area around a point x_i can be easily computed as detailed in Algorithm 5. Note, these mixed areas tile the surface without overlapping. The error bounds are not as tight when local angles are more than $\frac{\pi}{2}$, and therefore numerical experiments are expected to be worse in areas with obtuse triangles.

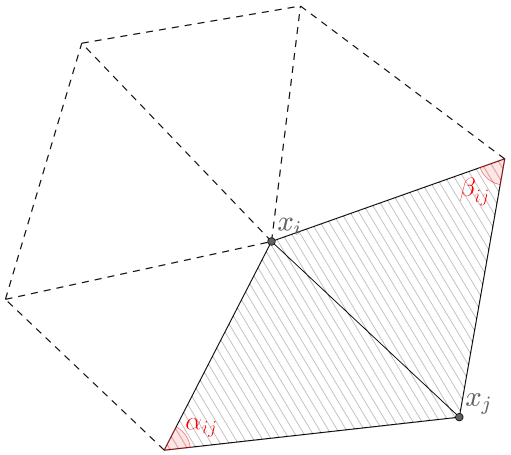
Algorithm 5 Voronoi Hybrid

```
1: procedure A_hybrid(x_i)
2:     A_hybrid ← 0
3:     for all triangles t in the 1-ring of x_i do
4:         if t is not obtuse then
5:             A_hybrid ← A_hybrid + A_Voronoi(t, x_i)
6:         else
7:             A_t ← area of triangle t
8:             if angle at x_i in t is obtuse then
9:                 A_hybrid ← A_hybrid + 1/2 A_t
10:            else
11:                A_hybrid ← A_hybrid + 1/4 A_t
12:            end if
13:        end if
14:    end for
15: end procedure
```

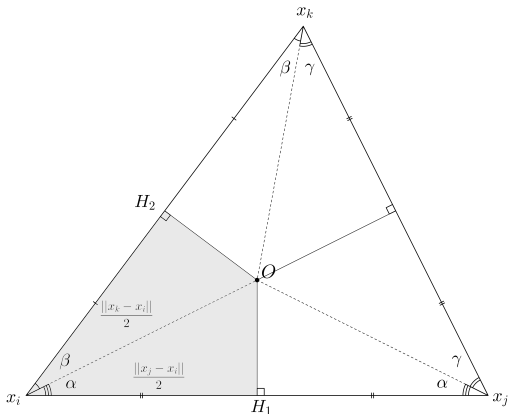
[1]The 1-ring of a vertex is the set of all adjacent mesh faces.



(a) Voronoi area (green shaded part) of vertex x_i with a 1-ring of obtuse triangles.



(b) Angle pairs used in $A_{Voronoi}(x_i)$ formula.



(c) $A_{Voronoi}(t, x_i)$ formula computes gray shaded area. Elements names match names used in formula's proof.

Figure B.1: Voronoi Area.

Appendix C

2D Polar Decomposition

Definition C.0.1 (Polar Decomposition)**:**
The polar decomposition of a square real matrix A is a factorization of the form $A = RS$, where R is an orthogonal matrix and S is a positive-semidefinite, both square and of the same size.

$$A = RS \text{ with } R^T R = I \text{ and } S = S^T$$

If we interpreted A as a linear transformation on $\mathbb{R}^m$ that transforms a column vector x to Ax, then in the polar decomposition $A = RS$, the factor R is an $m \times m$ real orthonormal matrix. The polar decomposition then can be seen as expressing the linear transformation defined by A into a scaling of the space $\mathbb{R}^m$ along each eigenvector e_i of A by a scale factor σ_i (the action of S), followed by a single rotation or reflection of $\mathbb{R}^m$ (the action of R).

It is possible to compute a polar decomposition using the results of singular value decomposition [53] (SVD):

$$R = UV, \quad S = V\Sigma V, \text{ where } A = U\Sigma V$$

However there is no simple SVD algorithm. The most common approach [19] is first to use Householder reflections to make A bidiagonal, then to perform an iteration involving QR Decomposition until the off-diagonal entries converge to zero. While this is numerically reliable, it is complicated to code, and by no means cheap.

A simpler method for computing polar decomposition is available [20, 38]. Compute the orthogonal factor by averaging the matrix with its inverse transpose until convergence: Set $R_0 = A$, then $R_{i+1} = \frac{1}{2}\left(R_i + R_i^{-T}\right)$ until $R_{i+1} - R_i \approx 0$. This is essentially a Newton algorithm for the square root of I, and converges quadratically when R_i is nearly orthogonal. However finding the R factor of a 2×2 matrix can be done explicitly.

Theorem C.0.1 (Explicit 2×2 polar decomposition)**:**
Every nonsingular $A \in \mathbb{R}^{2\times 2}$ has the polar decomposition:

$$A = RS \text{ with } R^T R = I \text{ and } S = S^T$$

for

$$S = \left|\det\left(A + |\det A|\, A^{-T}\right)\right|^{-\frac{1}{2}} \left(A^T A + |\det A|\, I\right) \tag{C.1}$$

and

$$R = \left|\det\left(A + |\det A|\, A^{-T}\right)\right|^{-\frac{1}{2}} \left(A + |\det A|\, A^{-T}\right) \tag{C.2}$$

A proof for Theorem C.0.1 can be found in [42]. Using ideas and results from the proof in question, we derive Algorithm 6 used to extract rotation matrix $R \in \mathbb{R}^{2\times 2}$ from transformation matrix A.

Algorithm 6 Extract Rotation

procedure $A = RS$
  if $\det A > 0$ **then**
    $\beta \leftarrow \sqrt{(a+d)^2 + (b-c)^2}$
    return $\frac{1}{\beta}\begin{bmatrix} a+d & b-c \\ c-b & a+d \end{bmatrix}$
  else
    $\beta \leftarrow \sqrt{(a-d)^2 + (b+c)^2}$
    return $\frac{1}{\beta}\begin{bmatrix} a-d & b+c \\ b+c & d-a \end{bmatrix}$
  end if
end procedure

Appendix D

Evaluation of Fabrics' Properties

Most if not all published cloth simulation models benefit from a physically based method of evaluating how well a cloth model behaves or for parameter identification. Despite the large body of work on cloth simulation models, little work has appeared in the computer graphics literature on estimating the parameters of these models so that they match the behavior of real fabrics. There are few measurement methods [8, 21, 36] and each comes with its own traits.

D.1 Kawabata Evaluation System

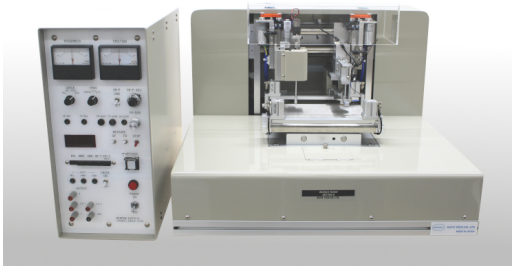
The system [36, 51] has been developed by a team from the department of polymer chemistry at the University of Kyoto (Japan) and it is a standard set of fabric measuring equipment which evaluates a list of the following characteristics:

1. Shear characteristics Figure D.1a
2. Tensile characteristics Figure D.1a
3. Bending characteristics Figure D.1b
4. Compressibility Figure D.1c
5. Surface roughness Figure D.1d
6. Surface friction Figure D.1d

(a) Shear and tensile characteristics tester.

(b) Pure bending characteristics tester.

(c) Compression characteristics tester.



(d) Surface roughness and friction characteristics tester.

Figure D.1: KES-F (Kawabata Evaluation Systems for Fabrics) specialized instruments

A cloth model would mainly be interested in the first four. Figure D.2 shows the actual test procedures and Figure D.3 their outputs.

[10] is a great example of a model which uses the empirical mechanical data produced by the Kawabata Evaluation System. That way it ties the model directly to the draping behavior of actual cloth.

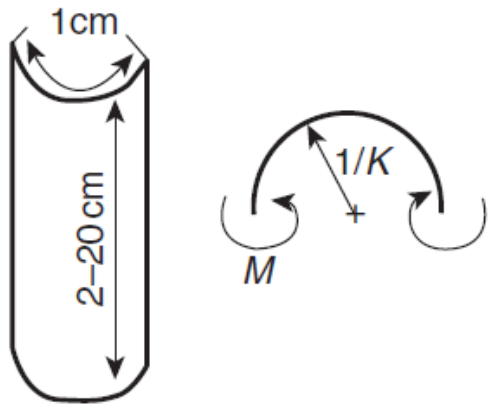
Note that the system does not measure dynamic cloth parameters, e.g. air drag or damping, which are of key importance for moving cloth. There are other measuring tools similar to the ones used in the Kawabata Evaluation System like the Textechno Statimat Tensile Tester from [34], but we will not discuss them any further.

D.2 Estimating Cloth Simulation Parameters from Video

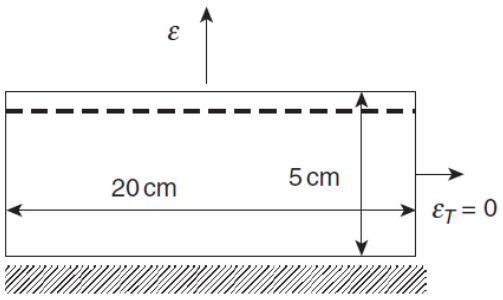
Not all parameters of all cloth simulators map directly to measurements that can be made by a system such as the Kawabata system. Hand tuning is always an option, a bad and time consuming option. The method discussed in [8] addresses the problem by using optimization to automatically determine these parameters from a sequence of video frames of the fabric under consideration. This method is easier to integrate and removes the need for expensive highly specialized equipment.

Parameters are optimized on a set of static shots and motion clips of a small swatch of a particular fabric. Swatch tests are designed to span the space of behaviors that are expected to be seen in the final sequences of motion so that all parameters can be tuned appropriately. Simulated annealing is used for the optimization step with an optimization function that assesses the extent to which the folds in the simulated and physical fabric match. This match is evaluated by means of a shape metric that uses projected light to detect surface orientation in real and simulated fabrics. The metric is tuned to be most sensitive along folds and to discount planar regions.

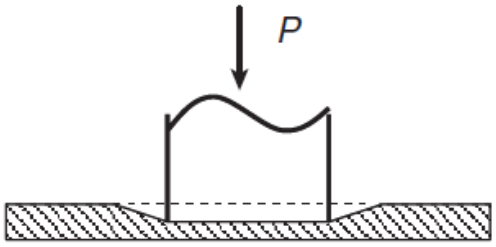
Perceptually motivated metric is a good fit to compare the motion of cloth in simulation with a video sequence of real fabric motion. The algorithm compares the two sequences frame by frame and computes an average error across the entire sequence. Real fabrics exhibit a wide variety of motion ranging from soft and flowing (satin) to stiff (linen). The metric captures the complex dynamics of cloth motion and also helps to distinguish between different fabrics.



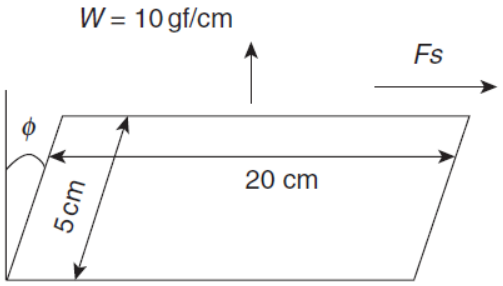
(a) Bending test. A specimen is bent between the curvatures $-2.5cm^{-1}$ and $2.5cm^{-1}$.



(b) Tensile test. Extension is applied along the 5 cm direction of the sample up to $500gf/cm$. The transverse contraction is not limited, so the test is a type of biaxial extension.

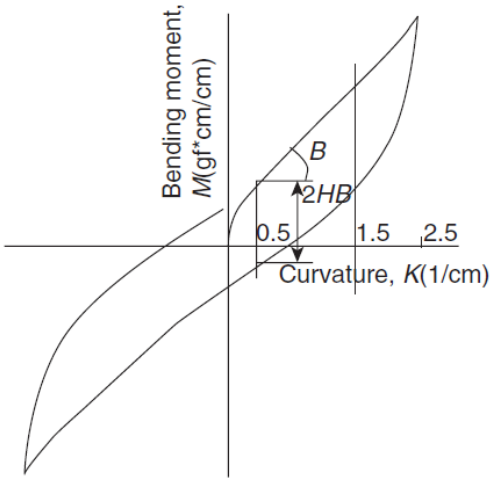


(c) Compression test. The specimen used is $2.5cm \times 2.0cm$, and therefore the effective pressure region is a circular area of two cm^2. The specimen is compressed within the direction of its thickness to a maximum pressure of $50gf/cm^2$.



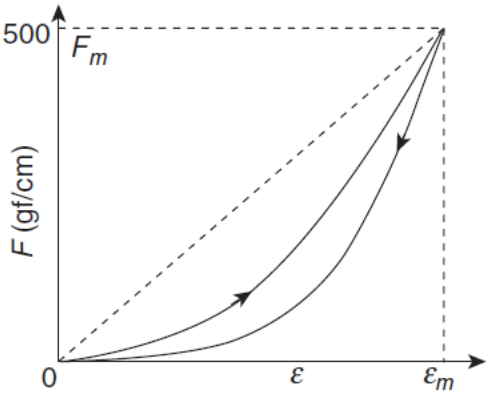
(d) Shear test. A rate of shear strain of 8.34^{-1} is applied to the specimen under a constant extension load ($10gf/cm$) up to a maximum shear angle of 8°.

Figure D.2: KES tests.



(a) Obtained bending curve. From which two parameters a remeasured:

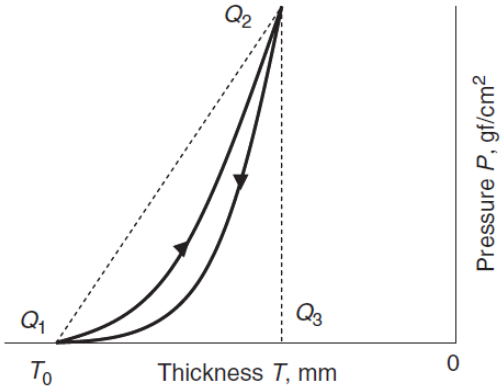
- Bending rigidity
- Hysteresis of bending moment



(b) Obtained load-extension hysteresis curve. From this curve, several parameters are derived:

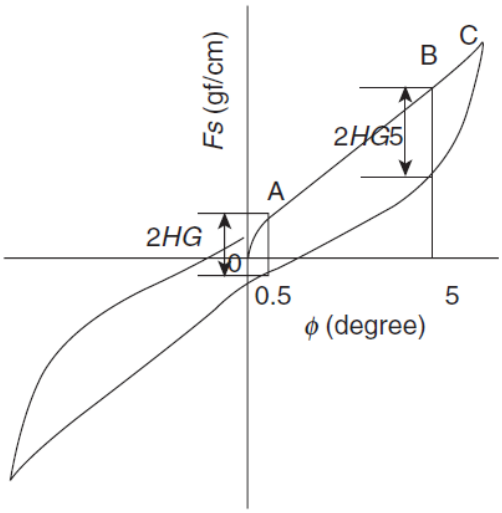
- Tensile energy
- Linearity of load-extension curve
- Tensile resistance
- Extensibility

Figure D.3: Example plots produced by KES tests.



(c) Obtained pressure–thickness curve. Similar to load-extension hysteresis curve, several parameters are derived:

- Compression energy
- Linearity of compression curve
- Compression energy



(d) Obtained shear-force/shear-angle hysteresis curve. From it, the following parameters are measured:

- Shear rigidity
- Hysteresis of shear force at shear angle of 0.5°
- Hysteresis of shear force at shearing angle of 5°

Figure D.3: Example plots produced by KES tests.

Bibliography

[1] David Baraff and Andrew Witkin. "Large steps in cloth simulation". In: *Proceedings of the 25th annual conference on Computer graphics and interactive techniques - SIGGRAPH '98*. 1998. DOI: 10.1145/280814.280821.

[2] Bruce G. Baumgart. "A polyhedron representation for computer vision". In: *Proceedings of the May 19-22, 1975, national computer conference and exposition on - AFIPS '75*. 1975. DOI: 10.1145/1499949.1500071.

[3] Jan Bender and Daniel Bayer. "Parallel simulation of inextensible cloth". In: *Virtual Reality Interactions and Physical Simulations (VRIPhys)*. Grenoble (France), Nov. 2008, pp. 47–56.

[4] Jan Bender and Crispin Deul. "Adaptive cloth simulation using corotational finite elements". In: *Computers & Graphics* 37.7 (Nov. 2013), pp. 820–829. DOI: 10.1016/j.cag.2013.04.008.

[5] Jan Bender, Daniel Weber, and Raphael Diziol. "Fast and stable cloth simulation based on multi-resolution shape matching". In: *Computers & Graphics* 37.8 (Dec. 2013), pp. 945–954. DOI: 10.1016/j.cag.2013.08.003.

[6] Nir Benty et al. *The Falcor Rendering Framework*. Aug. 2020. URL: https://github.com/NVIDIAGameWorks/Falcor.

[7] Miklos Bergou et al. *A Quadratic Bending Model for Inextensible Surfaces*. en. 2006. DOI: 10.2312/SGP/SGP06/227-230.

[8] Kiran S. Bhat et al. *Estimating Cloth Simulation Parameters from Video*. 2003. DOI: 10.2312/sca03/037-051.

[9] David E. Breen, Donald H. House, and Phillip H. Getto. "A physically-based particle model of woven cloth". In: *The Visual Computer* 8.5-6 (Sept. 1992), pp. 264–277. DOI: 10.1007/bf01897114.

[10] David E. Breen, Donald H. House, and Michael J. Wozny. "Predicting the drape of woven cloth using interacting particles". In: *Proceedings of the 21st annual conference on Computer graphics and interactive techniques - SIGGRAPH '94*. 1994. DOI: 10.1145/192161.192259.

[11] Robert Bridson, Ronald Fedkiw, and John Anderson. "Robust treatment of collisions, contact and friction for cloth animation". In: *Proceedings of the 29th annual conference on Computer graphics and interactive techniques - SIGGRAPH '02*. 2002. DOI: 10.1145/566570.566623.

[12] T. K. Caughey. "Classical Normal Modes in Damped Linear Dynamic Systems". In: *Journal of Applied Mechanics* 27.2 (June 1960), pp. 269–271. DOI: 10.1115/1.3643949.

[13] Kwang-Jin Choi and Hyeong-Seok Ko. "Stable but responsive cloth". In: *ACM SIGGRAPH 2005 Courses on - SIGGRAPH '05*. 2005. DOI: 10.1145/1198555.1198571.

[14] Robert Davis Cook. "Finite Element Modeling for Stress Analysis". In: 1st. USA: John Wiley & Sons, Inc., 1994. Chap. 3, pp. 41–47. ISBN: 0471107743.

[15] O. Etzmuss, M. Keckeisen, and W. Strasser. "A fast finite element solution for cloth modelling". In: *11th Pacific Conference onComputer Graphics and Applications, 2003. Proceedings.* IEEE Comput. Soc, 2003. DOI: 10.1109/pccga.2003.1238266.

[16] C.A. Felippa and B. Haugen. "A unified formulation of small-strain corotational finite elements: I. Theory". In: *Computer Methods in Applied Mechanics and Engineering* 194.21-24 (June 2005), pp. 2285–2335. DOI: 10.1016/j.cma.2004.07.035.

[17] Carlos A. Felippa. *A Systematic Approach to the Element-Independent Corotational Dynamics of Finite Elements*. Boulder, Colorado, Jan. 2000.

[18] James Foley. "Computer graphics: principles and practice". In: Reading, Mass: Addison-Wesley, 1995. Chap. 12, pp. 557–558. ISBN: 9780201848403.

[19] G. H. Golub and C. Reinsch. "Singular Value Decomposition and Least Squares Solutions". In: *Handbook for Automatic Computation: Volume II: Linear Algebra*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1971, pp. 134–151. ISBN: 9783642869402. DOI: 10.1007/978-3-642-86940-2_10.

[20] Nicholas J. Higham. "Computing the Polar Decomposition with Applications". In: *SIAM Journal on Scientific and Statistical Computing* 7.4 (Oct. 1986), pp. 1160–1174. DOI: 10.1137/0907079.

[21] Nebojsa Jojic and Thomas S. Huang. "Estimating Cloth Draping Parameters from Range Data". In: *International Workshop on Synthetic-Natural Hybrid Coding and 3-D Imaging*. Jan. 1997, pp. 73–76.

[22] Mats G. Larson and Fredrik Bengzon. "The Finite Element Method: Theory, Implementation, and Applications". In: Springer Berlin Heidelberg, 2013. Chap. 11, pp. 257–277. ISBN: 9783642332876. DOI: 10.1007/978-3-642-33287-6.

[23] Tae Min Lee, Young Jin Oh, and In-Kwon Lee. "Efficient Cloth Simulation using Miniature Cloth and Upscaling Deep Neural Networks". In: *CoRR* (2019). URL: http://arxiv.org/abs/1907.03953.

[24] François Lehericey, Valérie Gouranton, and Bruno Arnaldi. "GPU ray-traced collision detection for cloth simulation". In: *Proceedings of the 21st ACM Symposium on Virtual Reality Software and Technology*. Nov. 2015. DOI: 10.1145/2821592.2821615.

[25] Miles Macklin, Matthias Müller, and Nuttapong Chentanez. "XPBD". In: *Proceedings of the 9th International Conference on Motion in Games*. Oct. 2016. DOI: 10.1145/2994258.2994272.

[26] Mark Meyer et al. "Discrete Differential-Geometry Operators for Triangulated 2-Manifolds". In: *Mathematics and Visualization*. Springer Berlin Heidelberg, 2003, pp. 35–57. DOI: 10.1007/978-3-662-05105-4_2.

[27] August Ferdinand Möbius. *Der barycentrische Calcul*. Leipzig, Germany: Verlag von Johann Ambrosius Barth, 1827.

[28] Tomas Möller and Ben Trumbore. "Fast, Minimum Storage Ray-Triangle Intersection". In: *Journal of Graphics Tools* 2.1 (Jan. 1997), pp. 21–28. DOI: 10.1080/10867651.1997.10487468.

[29] D.E. Muller and F.P. Preparata. "Finding the intersection of two convex polyhedra". In: *Theoretical Computer Science* 7.2 (1978), pp. 217–236. DOI: 10.1016/0304-3975(78)90051-8.

[30] Matthias Müller and Markus Gross. "Interactive Virtual Materials". In: GI '04. London, Ontario, Canada: Canadian Human-Computer Communications Society, 2004, pp. 239–246. ISBN: 1568812272.

[31] Matthias Müller et al. “Position based dynamics”. In: *Journal of Visual Communication and Image Representation* 18.2 (Apr. 2007), pp. 109–118. DOI: 10.1016/j.jvcir.2007.01.005.

[32] Rahul Narain, Armin Samii, and James F. O’Brien. “Adaptive anisotropic remeshing for cloth simulation”. In: *ACM Transactions on Graphics* 31.6 (Nov. 2012), pp. 1–10. DOI: 10.1145/2366145.2366171.

[33] Frederick Thomas Peirce. “5—THE GEOMETRY OF CLOTH STRUCTURE”. In: *Journal of the Textile Institute Transactions* 28.3 (Mar. 1937), T45–T96. DOI: 10.1080/19447023708658809.

[34] Knezić Ž Penava Ž Šimić Penava D. “Determination of the Elastic Constants of Plain Woven Fabrics by a Tensile Test in Various Directions”. In: *FIBRES & TEXTILES in Eastern Europe 2014* 22.2 (2014), pp. 57–63.

[35] Xavier Provot. “Deformation Constraints in a Mass-Spring Model to Describe Rigid Cloth Behaviour”. In: *Proceedings of Graphics Interface ’95*. GI ’95. Quebec, Quebec, Canada: Canadian Human-Computer Communications Society, 1995, pp. 147–154. ISBN: 0-9695338-4-5.

[36] Kawabata S. *The Standardization and Analysis of Hand Evaluation*. Osaka: Textile Machinery Society of Japan, 1980.

[37] Jonathan R Shewchuk. *An Introduction to the Conjugate Gradient Method Without the Agonizing Pain*. Tech. rep. USA, 1994. DOI: 10.5555/865018.

[38] Ken Shoemake and Tom Duff. “Matrix Animation and Polar Decomposition”. In: *In Proceedings of the conference on Graphics interface ’92*. Morgan Kaufmann Publishers Inc, 1992, pp. 258–264.

[39] James G. Simmonds. *A Brief on Tensor Analysis*. Springer US, 1982. DOI: 10.1007/978-1-4684-0141-7.

[40] Baron Rayleighm John William Strutt. *The Theory of Sound*. en. Vol. 1. London, England: Macmillan and co., 1877.

[41] Min Tang et al. “A GPU-based Streaming Algorithm for High-Resolution Cloth Simulation”. In: *Computer Graphics Forum* 32.7 (2013), pp. 21–30.

[42] Frank Uhlig. “Explicit polar decomposition and a near-characteristic polynomial: The 2×2 case”. In: *Linear Algebra and its Applications* 38 (June 1981), pp. 239–249. DOI: 10.1016/0024-3795(81)90023-9.

[43] Loup Verlet. "Computer "Experiments" on Classical Fluids. I. Thermodynamical Properties of Lennard-Jones Molecules". In: *Phys. Rev.* 159 (1 July 1967), pp. 98–103. DOI: 10.1103/PhysRev.159.98.

[44] Pascal Volino, Nadia Magnenat-Thalmann, and Francois Faure. "A simple approach to nonlinear tensile stiffness for accurate cloth simulation". In: *ACM Transactions on Graphics* 28.4 (Aug. 2009), pp. 1–16. DOI: 10.1145/1559755.1559762.

[45] Robert Webb and Mike Gigante. "Using Dynamic Bounding Volume Hierarchies To Improve Efficiency of Rigid Body Simulations". In: *Visual Computing*. Springer Japan, 1992, pp. 825–842. DOI: 10.1007/978-4-431-68204-2_50.

[46] Keith Wilson et al. "Simulating wind effects on cloth and hair in Disney's Frozen". In: *ACM SIGGRAPH 2014 Talks on - SIGGRAPH '14*. 2014. DOI: 10.1145/2614106.2614120.

Online resources

[47] *Constructive solid geometry*. URL: https://en.wikipedia.org/wiki/Boundary_representation.

[48] Jonathan Dummer. *A Simple Time-Corrected Verlet Integration Method*. June 2, 2005. URL: http://archive.gamedev.net/archive/reference/articles/article2200.html.

[49] *Hooke's law*. URL: https://en.wikipedia.org/wiki/Hooke%27s_law.

[50] Thomas Jakobsen. *Advanced Character Physics*. Jan. 21, 2003. URL: https://www.gamasutra.com/view/feature/131313/advanced_character_physics.php.

[51] *Kawabata Evaluation System for Fabric*. June 22, 2020. URL: https://www.textileopedia.com/2020/06/kawabata-evaluation-system-for-fabric.html.

[52] Dean Macri. *Simulating Cloth for 3D Games*. July 3, 2012. URL: https://software.intel.com/content/www/us/en/develop/articles/simulating-cloth-for-3d-games.html.

[53] *Singular Value Decomposition*. URL: https://en.wikipedia.org/wiki/Singular_value_decomposition.

[54] Cyril Zeller. *Cloth Simulation*. Feb. 15, 2007. URL: https://developer.download.nvidia.com/whitepapers/2007/SDK10/Cloth.pdf.