Universidade do Porto

Faculdade de Ciências

Mestrado em Ciência de Computadores

Cloth Simulation Framework

Rui de Figueiredo Assunção

Supervisor:

Verónica Orvalho

A thesis submitted in fulfillment of the requirements

for the degree of Master of Science

at

Departamento de Ciência de Computadores

Porto, September of 2020

Acknowledgments

This thesis was the result of merging my passion for physics, programming and computer graphics, however it would not have been possible without the support and friendship of those around me.

First and foremost, I would like to express my gratitude to my supervisor, Professor Verónica Orvalho, who believed in me and encouraged me to seek the topics I enjoyed the most regardless of the challenge or area. Without her contribution, guidance and constant support, this work would not exist. Thank you Verónica.

I am also thankful to the countless professors I've encountered throughout my education that took time to share their knowledge with me. A special mention goes to Professor João Viana Lopes who developed my interest towards computation physics and simulation.

I, also, would like to thank my friends that have accompanied me previously and during the development of this thesis, in no particular order Filipe, Rodrigo, Ana, Olavo, Miguel, Nuno, Fábio and Yousif. Be it online or in presence, I am glad to have met and shared time with all of them. They provided a lot of support and happiness in both the best and the most difficult times.

Finally, a huge thank you to my family and Eduarda, who played a key role in keeping me sane and in track for this endeavor; who put up with my stress and terrible jokes; who reviewed and gave me constant feedback on the work; who supported me all the time and help shape the person I am. Without them, none of this would have been possible. Thank you.

Abstract

Displaying a computer generated piece of cloth or garment with a realistic behavior, such as adapting to surfaces and reacting to external forces or interactions, is a non-trivial task. Having skilled artists animate these objects with fluid behavior by hand would take an excruciatingly long time to achieve the desired results and, moreover, would not make them interactable. This problem is tackled by performing cloth simulation on the target object that we want, however this is also a complicated subject that is still under heavy research with a lot of technical challenges unsolved in terms of improving how realistic the results are, developing methods capable of constructing and simulating complex garments as well as performance bottlenecks that can limit use cases. There is a wide range of simulation methods that focus on different targets, such as algorithm speed versus result quality, or specific types of fabric; therefore no algorithm has managed to become dominant over the others as a complete solution to this problem.

In this line of thought, this thesis presents a cloth simulation framework that can serve as the base for the implementation of any simulation algorithm based on the mass-spring model. The framework is compatible with most simulation methods and provides multiple features that enable a user to quickly test different simulation algorithms and tune the parameters to achieve the desired cloth behavior. As a result, we simplify the task of iterating through different cloth simulation algorithms and their parameters for the entertainment industry.

We studied different techniques from the area of cloth simulation to achieve a solution to the problem of iterating and testing different techniques and parameters. The features included in the framework are: importing a 3D model as the target for the simulation; visualizing the model's particles; enabling user interaction with the cloth and the simulation parameters at runtime; and, finally, controlling external forces to affect the target. The framework creates an abstraction layer from the engine used, in order to facilitate the addition of new cloth simulation methods.

The validation was done throughout a series of experiments. We started by implementing two cloth simulation algorithms based on the symplectic Euler integration and the Verlet integration. We then used these algorithms to provide support for the visual validation of each of the features of the framework. Our system is generic and flexible (any simulation based on the mass-spring model can be integrated), easily extendable (the difficulty of the addition of an algorithm to the framework is primarily the difficulty of the implementation of the algorithm itself) and interactive (allows the user to interact with the cloth itself and the simulation parameters at runtime). With this system, experimenting different cloth simulation algorithms and tuning parameters to compare the results

produced can now be done more easily.

Keywords: framework, cloth simulation, mass-spring model

Resumo

Exibir uma peça de tecido ou de vestuário gerada digitalmente com um comportamento realista, tal como adaptar-se a superfícies e reagir a forças externas ou interações é uma tarefa não trivial. Ter artistas talentosos a animar estes objetos com comportamento fluído por processos manual levaria demasiado tempo para obter os resultados desejados e, além disso, não seriam interativos. Este problema é abordado pela execução de simulações de tecido no objeto-alvo que queremos, porém isto também é em si um assunto complicado que está sob intensa investigação com vários desafios por resolver em termos de melhorar o quão realistas os resultados são, desenvolver métodos capazes de construir e simular peças de vestuário complexas, assim como, limitações no desempenho que podem restringir a aplicabilidade. Existe uma míriade de métodos de simulação que se focam em diferentes objetivos, tal como a velocidade do algoritmo contra a qualidade dos resultados, ou tipos de tecido específicos; consequentemente, nenhum algoritmo foi capaz de se estabelecer como dominante em comparação com os outros para criar uma solução completa para este desafio.

Nesta linha de pensamento, esta tese apresenta uma framework de simulação de roupa que serve como base para a implementação de qualquer algoritmo baseado no modelo massa-mola. Esta *framework* é compatível com a maior parte dos métodos de simulação e fornece múltiplas funcionalidades que permitem ao utilizador testar diferentes algoritmos de simulação e afinar os parâmetros de simulação para obter o comportamento desejado do tecido. Em consequência, simplificamos a tarefa de iterar entre vários algoritmos de simulação de tecido e os seus parâmetros para a indústria de entretenimento.

Estudámos várias técnicas na área da simulação de tecido de forma a alcançar uma solução para o problema de iterar e testar diferentes técnicas e parâmetros. As funcionalidades incluídas da *framework* são a importação de modelos 3D como alvo da simulação, visualização das partículas do modelo, interação do utilizador com o tecido e com os parâmetros da simulação durante o tempo de execução e, por fim, controlo das forças externas que afetam o alvo. Esta *framework* cria uma camada de abstração do motor utilizado, de forma a facilitar a adição de novos métodos de simulação de tecido.

A validação foi feita através de uma série de experiências. Começámos por implementar dois algoritmos de simulação de tecido baseados na integração de Euler semi-implícita e na integração de Verlet; de seguida, utilizámos estes algoritmos para suportar a validação visual de cada uma das funcionalidades da *framework*. O nosso sistema é genérico e flexível (qualquer simulação baseada no modelo massa-mola pode ser integrada), facilmente expansível (a dificuldade da adição de um novo algoritmo resume-se

à implementação do mesmo) e interativa (permite ao utilizador interagir com o tecido e com os parâmetros da simulação em tempo real). Com este sistema, experimentar diferentes algoritmos de simulação de tecido e afinar parâmetros para comparar os resultados produzidos pode ser feito, agora, mais facilmente.

Palavras-chave: *framework*, simulação de tecido, modelo massa-mola

Contents

List of Figures

List of Tables

List of Acronyms

CG	Computer Graphics
CPU	Central Processing Unit
GPU	Graphics Processing Unit
FPS	Frames Per Second

Notation

- $\boldsymbol{d}_{ij}$ is a vector
- $\dot{\boldsymbol{u}} = \frac{\delta \boldsymbol{u}}{\delta t}$ is the time derivative
- $\boldsymbol{r}_i^t$ is the position of particle i at time t

1 Introduction

Cloth Animation *is a key piece in imparting the sense of realism on computer generated scenes, be it on a character's cloths presenting a fluid motion or a flag waving in the background. This is a common task on the entertainment industry and it drives the search for methods that automate the creation of these animations with high quality. We studied different techniques from the area of* ***cloth simulation*** *and present a cloth simulation framework that can serve as the base for the implementation of any simulation algorithm based on the mass-spring model. It provides access to key features that enable a user to quickly test different simulation algorithms and tune to parameters to achieve the desired cloth behavior. Currently, cloth simulation, is used in many different areas: film, video games, virtual reality, fashion design, online shopping, among others.*

This chapter discusses the main challenges of cloth simulation, describes the different applications of cloth simulation and provides a brief overview of our contribution.

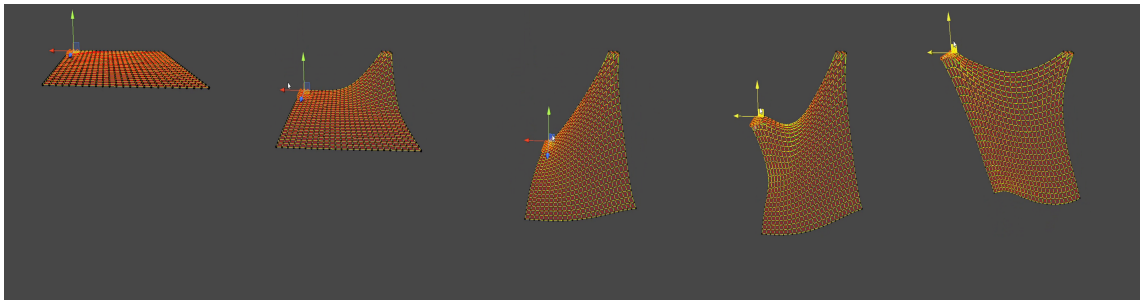

Figure 1.1: Example of a simulation running on the framework. The first three images show a square piece of cloth falling due to gravity and the following images depict one of the edges of the cloth being pulled up by the user.

1.1 Motivation

Soft body dynamics is a field of computer graphics that focuses on simulating realistic deformable objects (soft bodies), such as fluids or cloth. When used in films or video games, it creates a new layer of detail and richness to the world that is created for the viewer. Having an artist animate each individual piece of clothing manually would take many hours of work, and any change to the character's body or clothes would require recreating that animation. The logical step is to have an system that performs these animations automatically using simulation techniques. The high demand of quantity of animations and the high requirements for their quality (Bridson et al. [10]) of the entertainment industries resulted in a large interest on this topic, which transformed it into an area of intensive research for the last four decades.

In consequence of all the research done, a lot of different approaches to this problem have surged by multiple people, where the most popular are: the usage of geometry to approximate the look of the cloth, such as Weil [44], Chen and Tang [13]; a physical approach to approximate cloth as a continuum mechanics model, like Feynman [21], Terzopoulos et al. [40], Carignan et al. [12], Thomaszewski et al. [42]; modeling cloth as a set of discrete particles that interact, for instance Haumann [24], Breen et al. [9], Provot [34], Selle et al. [36]; and, more recently due to a large growth of Artificial Intelligence, the adoption of neural networks on this problem, including Oh et al. [33], Lahner et al. [31], Jangir et al. [26]. Besides these different approaches, we have also witnessed the formulation of algorithms that are target specific types of materials and cloth properties, *e.g.* Chen and Tang [13], Cirio et al. [16]. Finally, it is also important to note that some of the algorithms target quality and accuracy over speed[11], while others are aimed towards real time solutions[17].

As the examples above highlight, there are a lot of different possibilities when choosing the algorithm to perform cloth simulation, and choosing one that suits our needs can be very complicated and time consuming. Each algorithm performs better under the right conditions due to their differences, and subsequently none of them has emerged as the solution to all the needs in this area. As an example, particle based models require a much smaller amount of numerical computations than continuum based models, which leads to a preference of particle based algorithms for real-time simulations; on the other hand, continuum based algorithms are able to model complex garments, so they can be often used in state of the art for offline complex simulations [15].

When an artist is tasked with creating a simulation for a garment, they need to test multiple different algorithms and tune their parameters until they reach a solution that provides them with the result that fits their unique goals. This process of iterating over different methods can be very time consuming and changes to the models or the goals will cause the search for an algorithm and its parameters to start over. When a developer is implementing an existing method for cloth simulation or creating a new one, they need to be able to quickly test the current state of their implementation as well as test changes made to the algorithm. These are technological issues that we would like to tackle with our system in order to improve their workflow and reduce time costs of simulating cloth, thus ensuring that cloth simulation is not a bottleneck for computer graphics (CG) content on the entertainment industry.

1.2 Our solution

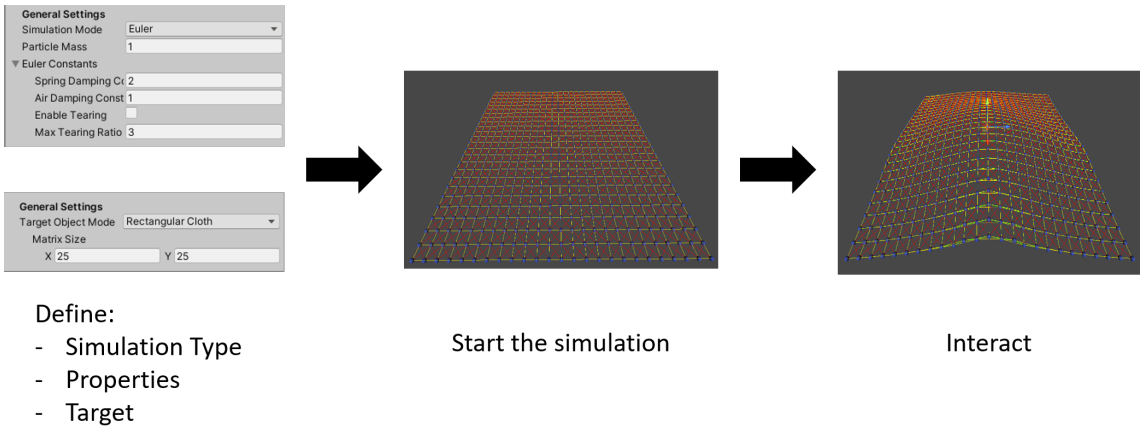

Figure 1.2: Overview of working with the framework. Start by defining the the algorithm, its properties and the target for the framework, then as soon as the simulation is running you can freely interact with it.

We focus our research on the development and/or testing of real time algorithms based on particle mass-spring models [24, 9, 45] for the video game and the virtual fashion industries. One of the common key points that these industries share is that they allow user interaction with the model as the simulation is running. Nonetheless, the resulting solution from this thesis can benefit other areas as well.

With this target in mind, *we develop a framework that is coupled with a cloth simulation algorithm and provides access to key features that facilitate testing and visualization of the algorithm in real time*. This system starts with an algorithm, a set of properties related to the simulation (such as the strength of particle interactions or the external forces) and a target object for the simulation. Then, the system exchanges information between the engine and the simulation to keep them both synchronized in terms of the position of the particles and regarding user interaction with the target object or the simulation properties. Switching between different cloth simulation methods implemented in the framework is as simple as selecting the different algorithm, and adding a new algorithm the framework is as difficult as the implementation of the algorithm itself (the integration is also simple). The target object can be a simple piece of rectangular cloth generated directly on the framework or an object created in an external 3D modeling software commonly used the CG industry, such as Maya or Blender.

This solution, on one hand, enables a developer to quickly make changes to an existing implementation and test the new version of the algorithm; on the other hand, it enables artists to interact with an algorithm and its parameters to find the appropriate combinations that suit their needs. These shorten the time required to obtain a simulation.

1.3 Objectives

During the development of the thesis, we defined a set of objectives that help guide the chapters of this thesis:

- Analysis of previous work done in particle-based simulation algorithms in real time;
- Design and implement the cloth simulation framework ;
- Implement different simulation algorithms on the framework;
- Validation of the results using the implemented algorithms.

1.4 Main contribution

The main contribution of this thesis is the development of a cloth simulation framework that:

- is **generic** and easily **extendable** as it allows for any cloth simulation algorithm based on the mass-spring method to be integrated quickly;
- provides real time **visualization** of the running algorithm;
- is **interactive** to allow the user to interact with the cloth itself and the simulation parameters in real time.

As a result, this work speeds up the process of testing real time cloth simulation algorithms, to diminish the time needed for the process of animating pieces of cloth, such as clothing garments and other scene fabrics to create more immersive environments.

1.5 Applications

The main application for cloth simulation is in the entertainment industry, namely the video game and animation films. It's an industry that has grown tremendously and, therefore, we see a lot of computer-generated content being created: from animated movies, such as *Monsters, Inc* (2001) and *Frozen* (2013, 2019), to live action movies, like *Hulk* (2003) and *Ted* (2012, 2015), and also video games, such as *Hitman* (2016, 2018), *Detroit: Become Human* (2018) and *Assassin's Creed Unity* (2018), all these examples among plenty more required cloth simulation during production.

Within the industry, the applications are mostly divided into *offline simulation* and *real time simulation*. Some industries are constraint to one of these options if they have certain requirements, *e.g.* allowing the user to interact with the target limits the usage to real time

approaches, while really high realism and accuracy requisites lean more towards offline simulations.

Offline simulations are computed ahead of time and saved so they only need to be imported later to the scene when they are needed in order to be rendered. These simulations can be tweaked and post-processed by the artists to achieve the best results possible at the cost of using very time-intensive algorithms. The artist usually goes through a number of iterations of adjusting different settings in order to reach a result that has as much quality as possible or required for that animation. By contrast, **real time simulations** are computed and rendered at runtime. These simulations have the capability of reacting to the user input and interaction, as well as respond to changes in the environment. Due to being computed at runtime they require fast and stable algorithms that are usually implemented on GPU hardware.

There are popular cloth simulators available that are used frequently on the CG industry, such as *nCloth* (plugin for Maya), *Houdini*, *Marvelous Designer and Clo3D,* however they mostly target high quality offline solutions; when searching for real time algorithms, one popular solution is *Havok Cloth*. Despite these available solutions, some companies also opt into creating their own simulators to provide them with more flexibility and control to meet their requirements as part of their own engines [3, 43, 35].

It's also common for companies to reduce the amount of cloth simulation necessary by providing the characters with tight clothing or armor that can be much more easily animated without the risk of simulation errors occurring and being visible, *e.g.* the characters *Kratos* and *Atreus* from the latest *God of War (2018),* as well as the main characters from the the *Final Fantasy 7 Remake* (2020). This simplifies the need for accurate simulations which can be specially error prone in attempting to solve intersections between two different pieces of simulated cloth.

While these concepts of offline and real time simulations have been fairly distinct, computing power keeps increasing every year, and solutions that weren't possible a few years ago are now possible and used, with solutions such as NVIDIA FleX by Macklin et al. [32]. In the future, with the increased usage of GPUs and enhanced hardware power we may see the techniques currently used in offline simulation possible in real time, merging the research done on both sides into real time, high quality and interactive simulations.

Lastly, while the major focus of discussion surrounded the topic of video games and film industries, there are other industries and areas that can take advantage of cloth simulation:

- **Fashion Design** and **Textile Engineering**: there are constant creations of new pieces of garment or textiles and the ability to create a virtual prototype helps identify any possible issues while developing. Using cloth simulation to analyze the proper-

ties of the cloth as well as visualize its behavior can provide important feedback on the iterative process of creating a new garment [39];

- **Virtual Humans**: an area gaining traction where the usage of virtual humans helps cross the boundaries created by distance and improve the expressiveness of virtual communication through the addition of new channels such as facial expressions and gestures . However, portraying humans in a virtual sense requires them to be wearing clothes, thus requiring real time simulations of the garments they are using [19, 8];

- **Online Shopping** and **Virtual Fitting Rooms**: allowing any possible buyer to visualize, experiment and interact with any piece of clothing at the comfort of his own home [29, 46]. This can also reduce the amount of refunds that occur during online shopping due to the items not matching what the buyer expects, being beneficial to both the buyer and the seller;

- **Immersive Virtual Worlds:** the idea where we transport a user or multiple users to a different reality where they can interact with their surroundings, specially with the recent growth of Virtual Reality [1]. The presence of cloth throughout the world can be very extensive, therefore cloth simulation that enables interaction is the key to improve the immersiveness of the user experience. An example of this is one of the largest applications called *VRChat* (2017), where the players are portrayed through custom made 3D character models, and they interact with each other.

1.6 Outline

This thesis is composed by the following six chapters:

- Chapter 2 (Related Work) provides an introduction to the work previously done in this area and the core concepts required to understand cloth simulation and the work presented further on this thesis.

- Chapter 3 (Cloth Simulation Framework) contains an explanation of the cloth simulation framework architecture and the roles of each component that form the framework, along with a discussion of the implementation of the features included.

- Chapter 4 (Physical Algorithms for Cloth Simulation) focuses on the implementation of two integration algorithms into the framework, Euler and Verlet integrations, as built-in examples of possible extensions of the framework.

- Chapter 5 (Results and Validation) verifies the validity of the system presented on the thesis coupled with the results of the algorithms implemented on the framework.

- Chapter 6 (Conclusions and Future Work) discusses the work done as part of the thesis, ending with some propositions of future work and some final thoughts regarding the topic of cloth simulation.

2 Related Work

Cloth simulation is a challenge that has been under research since the 1980s. Several approaches have been suggested to perform accurate and/or interactive real time simulations based on: geometric models, continuum models, particle models, neural networks and fusions of these. One of the more common techniques to perform real time cloth simulations are physical approaches that model cloth as particles and their interactions. This chapter provides an overview of the relevant work related to particle based cloth simulation. We start by reviewing some starting principles of what cloth is and how it can be represented in order to perform simulation. Then, we review the approach to time and space discretization and why it's important to discretize them. We follow up with a mass-spring model that is one of the most popular approaches to cloth simulation using physical methods, as well as how to perform each step in the simulation. Finally, we conclude this chapter with a brief discussion to sum up the core concepts that are more significant for this thesis.

2.1 Brief history of cloth simulation

Cloth simulation is an area that started being researched for computer graphics in the 1980s and has continued to grow ever since. The first advances were made by Weil [44] using a geometric approach, where he simulated the look of cloth by treating it as series of cables and using catenary curves. This technique was only suitable for still frame images, however it opened the doors for more research on the topic.

The next step of cloth simulation was the embrace of physical models for the cloth, and two different options emerged: using continuum models or using particle models. Continuum models were initially proposed by Feynman [21], who modeled cloth as elastic sheets; later, this technique was generalized for other elastic models by Terzopoulos et al. [40]. Meanwhile, there have been other attempts to find good continuum representations of cloth, such as in Carignan et al. [12] and more recently in King et al. [30], Bender and Deul [7], Gunamardi and Palit [23]. At the same time that continuum models were proposed, a different idea emerged for modeling based on particles; this model was pioneered by Haumann [24], Breen et al. [9], Provot [34] and due to its the popularity, studies regarding this particle approach are still common, *e.g.* in Hong et al. [25], Basori et al. [5].

Another extremely popular approach was the usage of triangle based cloth simulations suggested by Baraff and Witkin [4] and this became the base of many state of the art implementations that we use nowadays, as well as the base for more research on this approach, like Bridson et al. [10]. Lastly, since hardware improvements have led to the

developments of neural networks, they have also been used to perform cloth simulation, such as Oh et al. [33], Lahner et al. [31], Jangir et al. [26].

The literature for cloth simulation is very extensive, and it would be unwieldy to study all of these different methods and possibilities. Therefore, we will focus our research on the work that is relevant for our thesis, *i.e.* particle based models.

2.2 Cloth Representation

Before learning about simulations, we must understand how cloth is represented in computer graphics. Like most physical objects, a piece of cloth is a geometric shape that can be represented through a mesh, which is a collection of a small simple shapes such as triangles or squares (the latter also commonly referred to as “quads”). Figure 2.1 shows an example of a shirt that is made of multiple quads, as can be seen on the right image with the blue wireframe.

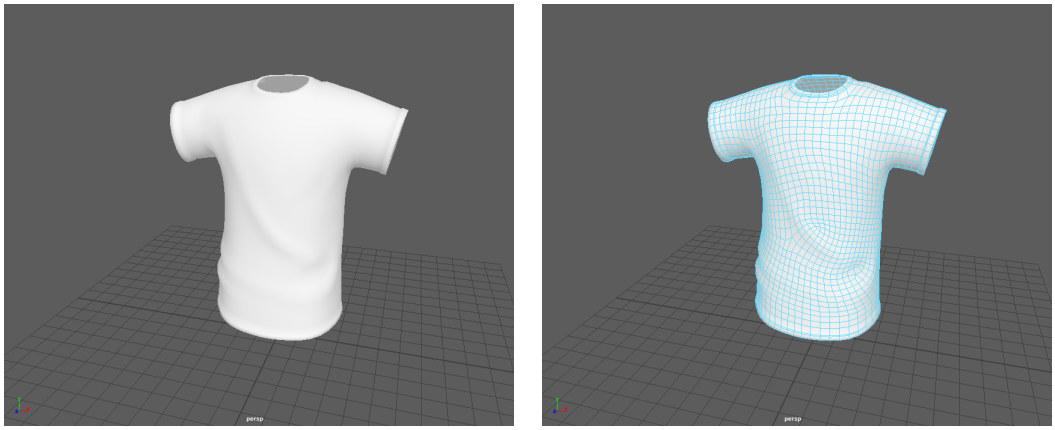

Figure 2.1: A 3D shirt model (left) as well as the the underlying mesh (right). In this case, the mesh is composed of quads as the base shape.
(source for 3D shirt model from artist *met_out*: https://www.turbosquid.com/3d-models/3ds-max-tshirt-soccer/570329)

Each quad in the mesh is composed of 4 vertices connected by edges and these vertices can be seen as the positions of particles that compose the object. This approach allows us to transform our continuous material into a discrete representation to use for our physically-based simulation by applying forces to each particle and moving it accordingly. Using this idea enables us to use the geometry of the mesh directly in the computation of the simulation.

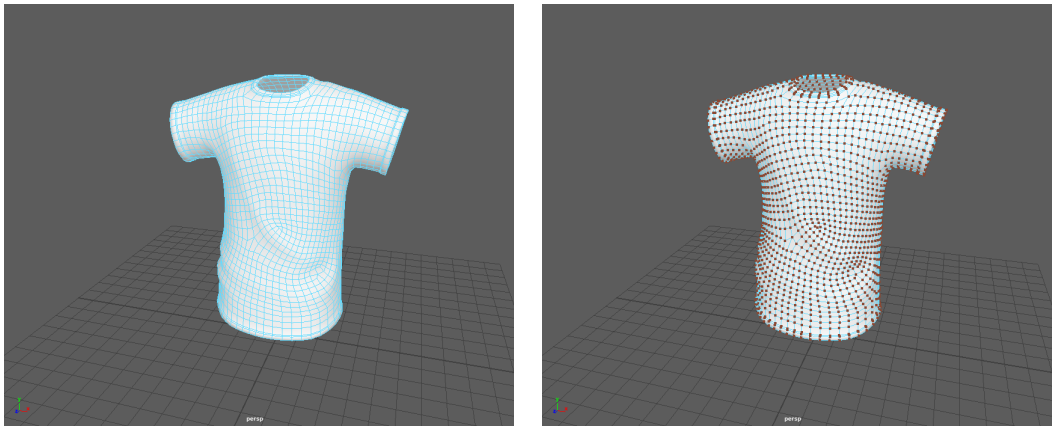

Figure 2.2: A 3D shirt mesh (left) and the particles positions at the edges of the quads as brown marks (right) (source for 3D shirt model from artist *met_out*: https://www.turbosquid.com/3d-models/3ds-max-tshirt-soccer/570329)

2.3 Particles

A particle i has a state that is a combination of its current position and velocity $\boldsymbol{q_i} = \langle \boldsymbol{r}_i, \boldsymbol{v}_i \rangle$, where $\boldsymbol{r}_i \in \mathbb{R}^3$ is the position 3D vector and $\boldsymbol{v}_i \in \mathbb{R}^3$ represents the velocity 3D vector. The simulation will be responsible for looking at the state of all the particles in a specific time t and using the interactions between the particles and external forces to determine the state of each particle for a new time $t + \delta t$. It's worth noting that, when performing each simulation step, we advance the time by a certain amount δt, so time will also be seen as a collection of small time steps.

This state will be updated over time in our simulation, according to constant physical laws that are given by the chosen physical model and handled by our simulator on each step.

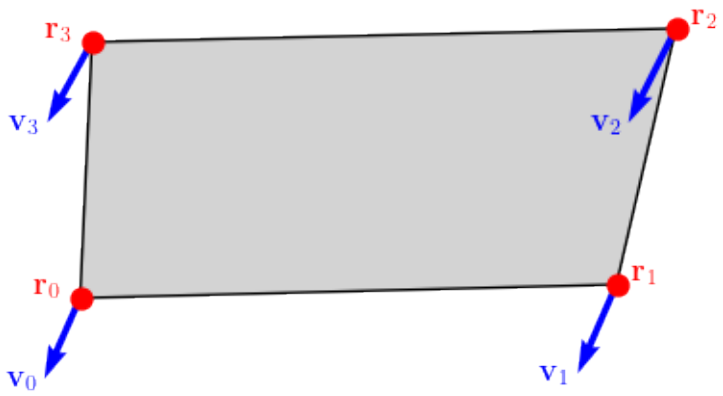

Figure 2.3: Example of positions (red) and velocities (blue) of the 4 particles that form a quad of a mesh

2.4 Forces

The simulation will be responsible for computing the forces that act on each particle, which will be a combination of internal and external forces:

- Internal forces originate from the interactions between the particles in the mesh and are the ones responsible for making our object behave like cloth. Changing how strong these forces are, empowers us to simulate different cloth materials such as linen, cotton and polyester.
- External forces are forces that originate from the outside environment, such as gravity or wind; these are not a requirement, but are very common to use alongside internal forces.

Forces applied on the particles will result in changes to their velocity, which in turn propagate to changes on the positions. This is the basic idea of how a physics based simulation works and how it can update the state of each particles over time.

2.5 Space and Time Discretization

It's important to emphasize that we have transformed time and space from continuous dimensions to discrete dimensions:

- **Discretization of space:** Cloth is represented by a finite number of particles, each with their own state. These particles interact with each other by applying internal forces.
- **Discretization of time:** We will update the particle states in time steps of duration δt.

There are simulation techniques that attempt to use continuous dimensions (space and/or time). Terzopoulos et al. [40], Carignan et al. [12] have done some of the early work on this approach, using continuum mechanics in order to simulate cloth as a continuous elastic model. However these techniques are substantially different and, thus, we will not study them for this thesis.

Using discretization is a strategy commonly used to approximate and simplify a problem. Time discretization in computer graphics is present in most digital media, such as videos or games. This is due to the fact that video is composed of a certain number of still images per second (frames per second) that are played sequentially, therefore there is an inherent time discretization while presenting the data to the user. This comes in handy, since we are interested in applying the same discretization on our simulation: given the state of an object in a specific frame, perform a simulation step to update the particles

to their position on the next frame. To make sure the simulation is stable, between each frame we can also perform multiple smaller steps, instead of one larger step. This is slightly more time intensive, however if we face issues like lack of stability or problems with collision detection, it may be required [4].

Space discretization is also a common solution when dealing with physics problems. In this case, we're studying a case where we transform a continuous material, such as cloth, into a discrete material that is made up of multiple small particles interacting with each other. This approach to cloth simulation was studied by Haumann [24], Breen et al. [9], Provot [34] and it will be the approach we will focus on for this thesis to perform the simulations. They propose a particle based model where the macroscopic properties of the cloth (such as tearing, bending, stretching) are approximated by the microscopic[1] spring-like interactions that act between pairs of particles. They are usually referred to as "Mass-Spring models".

2.6 Mass-Spring Model

As we have seen in the previous chapters, we are now dealing with cloth as a finite amount of particles that interact with each other. Now, we need to understand what type of interactions these are as well as what particles are interacting and this is where our mass-spring model comes into play. Like the name suggests, our particles behave like point masses, which are physical objects with a relevant mass but infinitely small size that we don't need to account for. These particles are connected in pairs by spring-like interactions, *i.e.* forces that always push the two objects object towards an equilibrium distance.

2.6.1 Spring Forces

The force exerted by a spring on a body is given by *Hooke's Law*, and it states that that force is proportional to the displacement in relation to the spring's rest length [37, 38]. If there are two particles, i and j, interacting with a spring with rest length L, then the formula for the force applied to particle i by j is given by

$$\boldsymbol{f}_{ij}(\boldsymbol{r}) = k_s \left(\|\boldsymbol{r}_j - \boldsymbol{r}_i\| - L \right) \widehat{\boldsymbol{r}_{ji}}, \tag{2.1}$$

where k_s is a stiffness constant characteristic of the string, $\|\cdot\|$ represents the Euclidean distance, and $\widehat{\boldsymbol{r}_{ji}}$ is the unit vector (vector of length 1) that points from particle j to particle

[1] The term *microscopic* is being used in relation to the forces between the imaginary particles we created, and not to actual microscopic, atomic or subatomic interactions.

i. The unit vector's purpose is to give us the direction of the force and it can be written as

$$\widehat{\boldsymbol{r}_{ji}} = \frac{\boldsymbol{r}_j - \boldsymbol{r}_i}{\|\boldsymbol{r}_j - \boldsymbol{r}_i\|}, \tag{2.2}$$

i.e. the vector from $\boldsymbol{r}_j$ to $\boldsymbol{r}_i$ normalized by it's own length. The force applied to particle j by particle i has the same magnitude as $\boldsymbol{f}_{ij}$, but points towards the opposite direction, so we can easily write it as

$$\boldsymbol{f}_{ji} = -\boldsymbol{f}_{ij}. \tag{2.3}$$

It's worth knowing that this formula is valid when we are dealing with small displacements from the rest length, but not for the extremes of maximum compression or stretching. However, these extreme cases are a lot more complicated to model and much less likely to happen in a stable simulation, so they are not included in our model.

These formulas can be understood more clearly by looking at Figure 2.4 with the example of the three possible situations. On the top, we have the case where the particles are positioned at exactly their equilibrium distance, which is the spring's rest length, and therefore there is no force being applied to any of the particles. On the middle case, the particles were compressed, and so the spring exerts a force to return to it's original state by pushing both particles away from each other. The bottom situation portraits the inverse case, where the particles are distanced, which results in a force that pushes the particles towards each other.

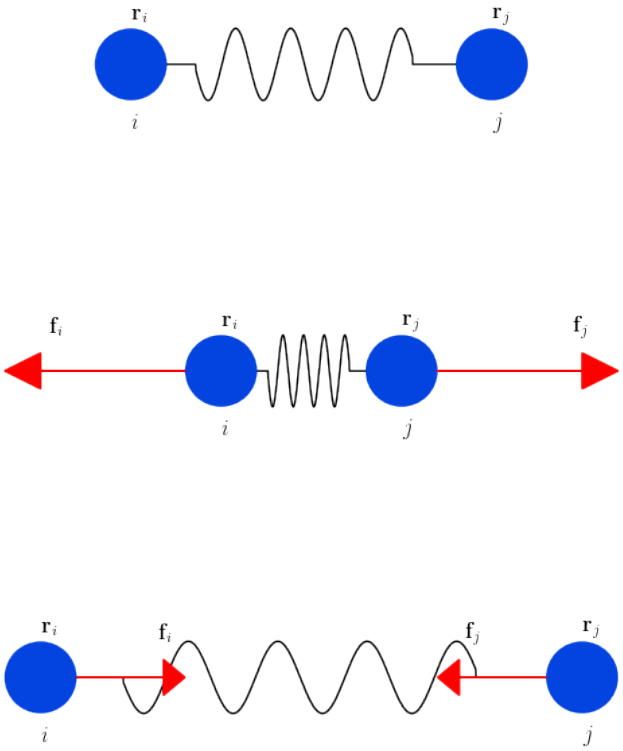

Figure 2.4: Example of two particles interacting with a spring in equilibrium (top), while being compressed (middle) and being stretched (bottom).

Now that we understand the interactions we will need to compute between the particles, we must also understand which particles will be interacting.

2.6.2 Springs

In order to keep our model simple, we cannot make each particle interact with every other particle that exists in the same object, as that would yield a model that is complex to compute. Instead we'll take a a more localized approach, where particles will only interact with their nearest neighbors. Local models stem from the idea that the main contributions to an object come from its closest neighbors, while the interactions with other objects far away are so small that they become negligible.

There are three main types of deformations that can occur, namely stretching, shearing and bending, which can be visualized on Figure 2.5. Most cloth does not suffer much deformation from stretching or shearing, however it can easily bend to create folds and wrinkles.

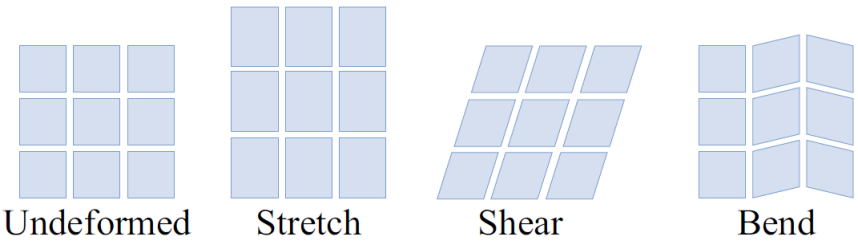

Figure 2.5: Visualization of a cloth patch under stretch, shear and bend deformations. Image from Tuur Stuyck. Cloth Simulation for Computer Graphics, volume 10, 2018 (page 22)

We need to model the resistance to these deformations, we need to link the particles with it's neighbors using [9, 28]:

1. **Stretch/Structural springs** that connect each particle with its four adjacent non-diagonal masses to resist stretching or compressing deformations;
2. **Shear springs** that connect each particle with its four adjacent diagonal masses to resist shearing deformations;
3. **Bend springs** that connect each particle with its four neighbor particles on every other row/column (non-adjacent particles) to resist stretching or compressing deformations.

An example of these springs in a mass-spring system is visible in Figure 2.6 below.

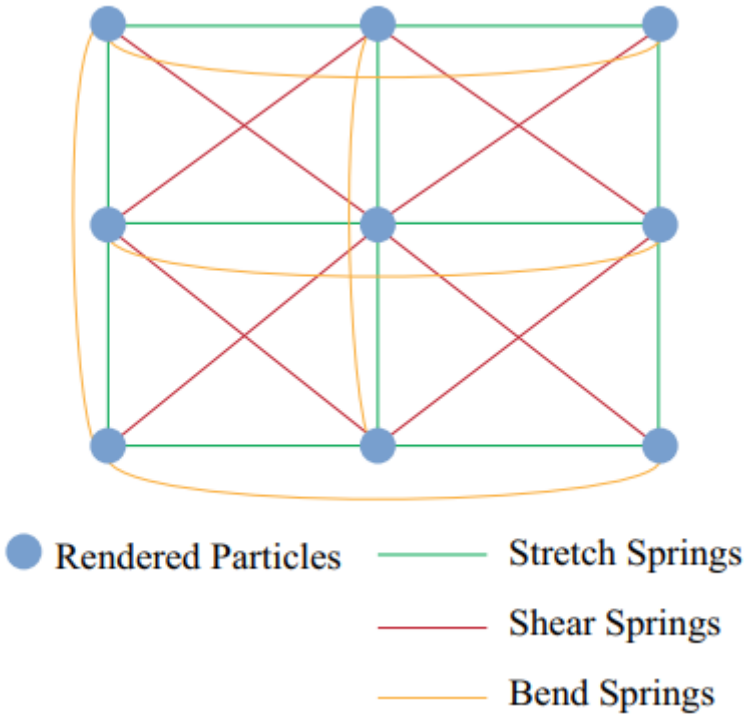

Figure 2.6: Mass-spring mode with stretch, shear and bend springs applied to a 3x3 matrix of particles. Image from Tuur Stuyck. Cloth Simulation for Computer Graphics, volume 10, 2018 (page 23)

It's important to differentiate between the different types of strings as each one can have different rest lengths as well as different stiffness constants k. Varying the values of stiffness of these spring types will allow us to change the simulation dramatically and approximate different types of real life materials. However, it's common for the stretch stiffness constant to have a higher value than the remaining two spring constants [38].

It's important to note that, while we refer to them by the names of the deformations they affect the most, their effects can also be slightly noticeable on the other types of resistance to deformations, *e.g.* shear springs can affect cloth stretching. Lastly, we could also use a different set of springs in our model to simulate different cloth behaviors, such as bending springs connecting every three particles instead of every other particle, nonetheless this is the most common structure used by many researchers and developers.

2.6.3 Spring Damping Forces

Damping is the process of dissipation of the internal energy of a system that occurs naturally over time, from sources such as air resistance or surface friction. It's also one of the crucial forces that should be included in our simulation to obtain stable results [9]. If we discard it, not only can we get unstable simulations, but also any interference on our mode can put it into a perpetual oscillatory state.

A simple approach to damping forces can be based on *Stoke's Law*, which states that the frictional force on a small sphere in a viscous fluid is proportional to its speed [6]. Adapting this to our mass spring model, we can write the damping force $\boldsymbol{d}$ exerted on a particle i connected to particle j as

$$\boldsymbol{d}_{ij}(\boldsymbol{v}) = k_d \cdot (\boldsymbol{v}_j - \boldsymbol{v}_i) = -\boldsymbol{d}_{ji}(\boldsymbol{v}), \tag{2.4}$$

where k_d is the damping coefficient that controls how strong the damp effect is. Similarly to the spring forces, the damping force on particle j has the same magnitude as $\boldsymbol{d}_{ij}$ but points on the opposite direction.

2.7 Simulation Step: Euler Integration

Now that we have learnt what forces exist on our system and how to calculate them, we need to be able to transform forces at a specific time-frame into changes on the particles' positions. From Newton's second law of motion we know that

$$\boldsymbol{F} = m\boldsymbol{a} \tag{2.5}$$

, where $\boldsymbol{F}$ is the vector sum of all the forces and $\boldsymbol{a} \in \mathbb{R}^3$ is the acceleration of a particle. Moreover, we can write the velocity and acceleration as

$$\boldsymbol{v}(t) = \frac{d\boldsymbol{r}(t)}{dt} \tag{2.6}$$

$$\boldsymbol{a}(t) = \frac{d\boldsymbol{v}(t)}{dt}. \tag{2.7}$$

However, as we have discussed in section 2.5, for our purposes, time will be dealt with as a discrete dimension. To discretize these equations, we can use the forward finite differences approximation to write them as:

$$\boldsymbol{v}(t) \approx \frac{\boldsymbol{r}(t+h) - \boldsymbol{r}(t)}{h} \tag{2.8}$$

$$\boldsymbol{a}(t) \approx \frac{\boldsymbol{v}(t+h) - \boldsymbol{v}(t)}{h}, \tag{2.9}$$

where h is a small time step. This approximation only uses the first degree derivative and subsequently assumes that the velocity and forces/acceleration are constant through the entire step.

These equations can be rewritten in terms of $\boldsymbol{r}(t+h)$ and $\boldsymbol{v}(t+h)$ as

$$\boldsymbol{r}(t+h) \approx \boldsymbol{r}(t) + h \cdot \boldsymbol{v}(t) \tag{2.10}$$

$$\boldsymbol{v}(t+h) \approx \boldsymbol{v}(t) + h \cdot \boldsymbol{a}(t), \tag{2.11}$$

or using discrete notation for the time

$$\boldsymbol{r}^{t+1} \approx \boldsymbol{r}^t + h \cdot \boldsymbol{v}^t \tag{2.12}$$

$$\boldsymbol{v}^{t+1} \approx \boldsymbol{v}^t + h \cdot \boldsymbol{a}^t, \tag{2.13}$$

where t represents the current time and $t+1$ represents the next time step.

We can combine Eq. (2.5) and Eq. (2.13) to find the equations

$$\boldsymbol{r}^{t+1} \approx \boldsymbol{r}^t + h \cdot \boldsymbol{v}^t \tag{2.14}$$

$$\boldsymbol{v}^{t+1} \approx \boldsymbol{v}^t + h \cdot m^{-1} \cdot \boldsymbol{F}^t \tag{2.15}$$

that will be used to update the state of each particle using the current state $\boldsymbol{r}, \boldsymbol{v}$, mass m and the total force $\boldsymbol{F}$ for that time.

This method of updating the position and velocity is referred to as **Euler/Explicit Integration** or **Forward Euler Integration**, and it assumes that, between each step, the velocities and forces are constant for the finite differences approximation. This is one of the sources of instability that comes from this method since, in reality, this does not happen and causes it to repeatedly overshoot the correct value [4]. Therefore, using this method will require small steps so that the velocity and force variations are negligible. There are more complex integration methods or alternatives to this method, such as correction terms, that have been explored by multiple people, such as Baraff and Witkin [4] and Desbrun et al. [18].

2.8 Simulation Step: Verlet Integration

There are many alternatives to using Euler Integration, in order to solve the laws of motion. One that provides more stable results and is more commonly used is **Verlet Integration**. This alternative takes a slightly different approach to the problem and works without using the velocity of the particles. Instead, however, it uses the current position and the previous position to approximate the velocity of the particle.

In order to achieve better stability, we use a more accurate approximation of $\boldsymbol{r}(t+h)$, by utilizing the second order derivative of the central finite differences approximation:

$$\boldsymbol{a}(t) \approx \frac{\frac{\boldsymbol{r}(t+h)-\boldsymbol{r}(t)}{h} - \frac{\boldsymbol{r}(t)-\boldsymbol{r}(t-h)}{h}}{h} = \frac{\boldsymbol{r}(t+h) - 2\boldsymbol{r}(t) + r(t-h)}{h^2}, \tag{2.16}$$

which can be rewritten in terms of $\boldsymbol{r}(t+h)$ as

$$\boldsymbol{r}(t+h) \approx 2\boldsymbol{r}(t) - \boldsymbol{r}(t-h) + h^2\boldsymbol{a}(t). \tag{2.17}$$

Using Eq. (2.5) for the acceleration, and writing the formula using discrete notation we obtain

$$\boldsymbol{r}^t \approx 2\boldsymbol{r}^t - \boldsymbol{r}^{t-1} + (\Delta t)^2/m \cdot \boldsymbol{F}, \tag{2.18}$$

which is the base for this alternative integration method. In this formula, $\Delta t = h$ and represents the time between steps.

The second strategy for improved stability is to use a position-based constraint system, instead of spring forces, to keep the particles close to their equilibrium states. These constraints will try to keep the distance between two interacting particles as close as possible to their equilibrium distance.

After performing an update step of the positions, distances between pairs of particles won't match their equilibrium distance, and fixing this is next challenge. It would be too computationally expensive to create an equation system and attempt to solve every constraint at once. On the other hand, solving each constraint independently by moving each particle along their common axis, would cause the latter constraints to break the earlier constraints, as shown on the figure below.

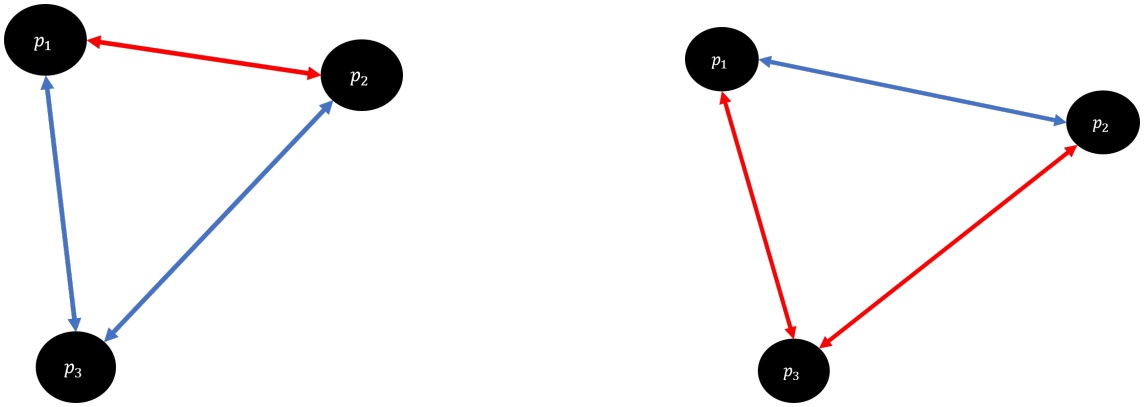

Figure 2.7: Example of how solving a constraint could break previously set constraints. Blue arrows show constraints that are already solved, while red arrows show constraints that are not properly solved. On the left image only the constraint between p_1 and p_2 is unsolved, however by moving these particles along their axis to solve their shared constraint, breaks the other two previously correct constraints.

One possible trick to solve this, proposed by Thomas Jakobsen [41], is a method called "relaxation" that consists of solving the constraints multiple times in a loop. While this seems like a naive approach, the idea is that, on each iteration, the constraints will be closer and closer to their optimal solution.

The last point of this integration algorithm relates to the damping force. Previously, on section 2.6.3, we mentioned a damping force that is associated with the forces of the springs. While the same formula could be applied here, this approach does not use the spring forces directly. If we rewrite Eq. (2.18) as

$$\boldsymbol{r}^t \approx \boldsymbol{r}^t + (\boldsymbol{r}^t - \boldsymbol{r}^{t-1}) + (\Delta t)^2/m \cdot \boldsymbol{F}, \tag{2.19}$$

we notice that the term $(\boldsymbol{r}^t - \boldsymbol{r}^{t-1})$ represents the amount moved between frames, which is related to the velocity. Despite velocity not being explicitely present on the Verlet algorithm, we can find it hidden as the different of positions.

Therefore, we can apply a new damping effect by reducing this "velocity" term by a constant [27]:

$$\boldsymbol{r}^t \approx \boldsymbol{r}^t + (1 - \alpha_d) \cdot (\boldsymbol{r}^t - \boldsymbol{r}^{t-1}) + (\Delta t)^2/m \cdot \boldsymbol{F}, \tag{2.20}$$

where $\alpha_d \in [0, 1]$ is the ratio that describes how much damping effect there is. If

$\alpha_d = 0$, then all the velocity is preserved between steps, and if $\alpha_d = 1$ then all velocity is lost between each step.

In comparison to Euler Integration, this approach uses a higher order approximation to calculate the new positions of the particles for each step, which in turn improves the stability. On the other hand, trying to solve these constraints for many iterations can create a stiff look, as the target object will always tend to the initial equilibrium state.

2.9 Conclusion

The goal of this dissertation is to build a framework capable of performing cloth simulation on a piece of cloth, and one of the most popular ones are particle based models, so we decided to focus our efforts on these.

This section presented the base concepts and ideas required to understand how to tackle this problem. The mass-spring's technique is to generate a set of particles from the object's mesh that interact through spring forces in a specific pattern, such as is presented in Figure 2.6. These interactions consist of different types of string that emulate the textile response to stretch, shear and bending deformations. Changing the strength of each type of interaction enables us to alter the behavior of the model and, thus, simulate different materials. This model is also capable of reacting to external forces such as wind and gravity.

Each simulation step is taken according to an integration algorithm and the current state of the model. Euler Integration will take all the internal and external forces exerted on each particle to compute the state of all the particles on the next step. On the other hand, Verlet Algorithm computes the state of the next particles using a combination of external forces and constraints. Using the Euler method can provide simulations with a more flexible look than the Verlet method, however it is also more unstable so it is something we have to be careful to deal with.

To conclude, the work seen presents an interesting simplification of cloth and a solution for performing the simulation, however with some possible pitfalls related to instability. In order to build the framework, it's important to have knowledge of the methods that we will want to support.

3 Cloth Simulation Framework

This chapter presents the architecture of the framework developed for this thesis, breaking it down into a set of components that communicate with each other. We start with a brief rundown of the most important features that Unity provides. Then, we follow up with a discussion of each of the components that form the framework, assessing their responsibilities towards the framework as well as the information they exchange with the other components. Afterwards, we discuss the implementations of the key remaining features of the framework. This chapters wraps up with a recapitulation of the framework components along with their purpose towards the framework.

3.1 Architecture Overview

This framework was written on top of an existing engine, called Unity, due to it's widespread usage throughout the CG industry as well as the bundle of built-in features. The main and most important idea behind this cloth simulation framework is that it should be generic and flexible, in order to allow us to extend components easily to add new algorithms. Using this approach, we can implement different simulation methods and compare them easily by simply replacing or activating a new component.

The framework is made up of two major parts that play different roles: a set of abstraction components and extendable components. The purpose of the first is to communicate with all the engine and all the components, creating an abstraction layer on top of the engine that the user can interact with; the latter is responsible for providing a base that any developer can expand to implement their own algorithms and include them in the framework.

To achieve to goals set to our framework, we divided the *abstraction components* into three managers:

- the **Particle Displayer**: related to the visualization and interaction with the model directly;
- the **Target Manager**: responsible for setting the target of the simulation and obtain the required data from it;
- the **Simulation Manager**: responsible for controlling and communicating with the simulation at all times; it serves as the main communication channel between the engine and the simulation, as well as the user and the simulation.

The *extendable component* is a component with two parts: a **base particle** and a **base simulator** classes, that any developer can extend when implementing new algorithms.

A simplified sketch can be seen on Figure 3.1 where the lines represent some sort of communication. Using this approach allows us defer some of the responsibilities to the managers and create an abstraction layer from the engine, thus keeping the code of the simulation and particle classes more concise, flexible and independent from the engine that we are using.

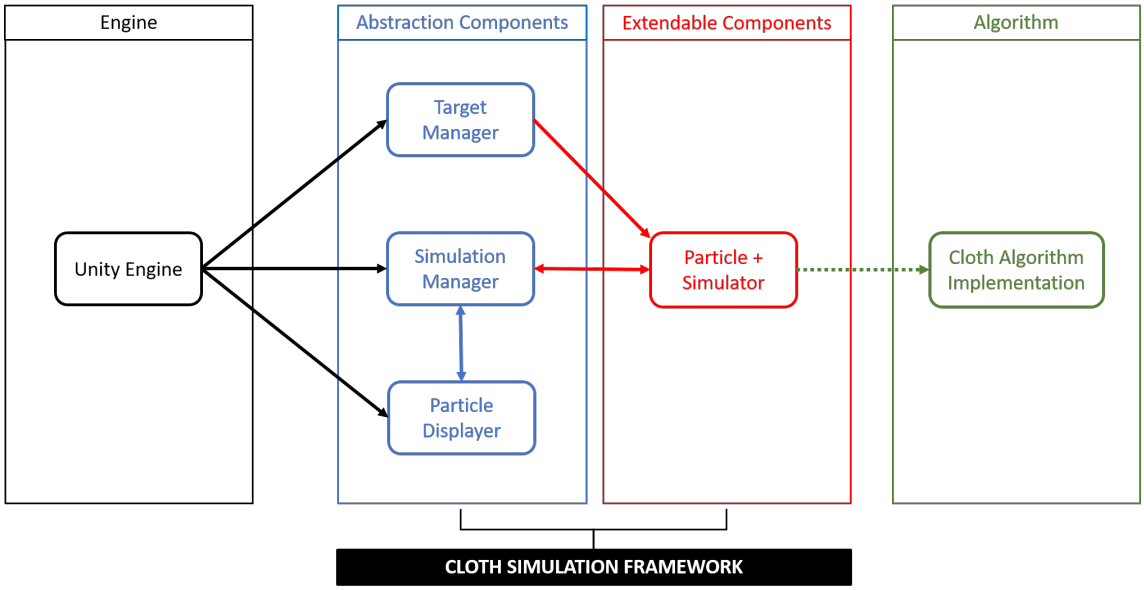

Figure 3.1: Simplified overview of the cloth simulation framework along with its components and the communication between them.

We will go more in depth about each of these components in the following chapters.

3.1.1 Implementation Details

All the components that we've mentioned were implemented as C# classes, due to the engine used. The managers are scripts attached to a *GameObject* in the Unity scene and so they are required to inherit from *Monobehaviour*, which is a class provided by Unity [2]. The base classes for the particle and simulator don't inherit from any other class, so it would be possible to use them elsewhere.

3.2 Overview of Unity and its features

The framework was developed on top of the Unity engine, which is a engine developed by *Unity Technologies*. While it's popular within the video game industry, it is a very flexible platform that is also widely used in other industries, such as film, animations, prototyping, architecture, among other situations. Since it serves as the base engine, it's important to

[2] *Monobehaviour* provides access to a few functions that controlled and called by the Unity engine. More information available at https://docs.unity3d.com/ScriptReference/MonoBehaviour.html

note some of the key features provided that will be useful for the work presented on this thesis [3].

First, Unity objects on a scene are called *GameObjects* and they can contain scripts attached to them. Each object has a *Transform* component that keeps track of its position, rotation and scale. We will attach the manager component scripts to an empty game object that will be responsible for performing the simulation.

Second, Unity provides an editor window which allows us to interact with objects in the scene at any time, be it while creating the scene or while the application is running. We can take advantage of this feature detect user interaction with the particles on the scene.

Third, Unity comes with a built-in editor for the public variables of scripts, however the editor can be further extended or completely overhauled by the developer; it allows us to change variables values at runtime. This is a quality of life feature that is incredibly versatile.

Fourth, Unity provides an Update loop for rendering and a FixedUpdate loop for physical updates. The Update loop will run once each frame, however since the number of FPS is not necessarily constant, this loop depends on the frame rate that we are currently working with. It's good practice to decouple the physics logic from the rendering logic to improve the stability of the code Fiedler [22]. The FixedUpdate loop is reliable, and will always run on the same period of time inside the application, regardless of rendering or performance. If the code on the FixedUpdate loop is not performant enough, Unity will slow down the application's time to make sure that this loop's FPS is kept constant[4][2]. All the code related to the numerical integration and simulation will be run on the FixedUpdate loop, while user interactions will be dealt with on the Update loop.

3.3 Framework specifications and how to use

The framework, which was built on top of Unity, currently uses version 2020.2.0a18, however any version between 2019 or 2020 of Unity should be fully compatible. This is the only requirement for using the framework.

In order to use the framework, the setup is simple:

- Create a Unity project or open an existing one;
- Drag the folder with all the scripts related to the framework onto the Assets folder or a sub-folder;
- Create an object in the scene and add the manager components to it;

[3] Unity is a complex engine with a lot of features included. I'll only cover the most important ones, however more information is available on their website https://unity.com/ and https://docs.unity3d.com/Manual/index.html

[4] https://docs.unity3d.com/Manual/ExecutionOrder.html

- Set all the properties in the managers;
- Start the application.

In order to facilitate the addition of new algorithms, the framework already includes two built-in algorithms (Euler integration and Verlet integration) to serve as a guide. Use these as examples for the implementation and add them as possibilities to the Simulation Manager also using the examples as guides.

3.4 Manager components

Now that we've explored some of the more important features provided by the engine, we can start looking at the manager components added to the objects. These managers play a key role on creating a flexible framework , and their function is to create a communication channel between the engine and the simulation . As we have seen on chapter 3.1, the three core managers are the the Target Manager, Particle Displayer and the Simulation Manager and they are aware of each other, working in cooperation.

3.4.1 Target Manager

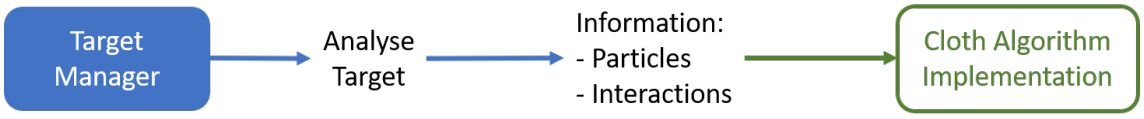

Figure 3.2: Overview of the responsibilities and communication of the Target Manager component with the other components.

The first manager component that we will look at is the Target Manager, which is responsible for setting the target of the simulation and determine some initial information about it. This manager defines if the simulation will target an imported object into the scene or if we want to create a simple rectangular cloth to perform the simulation on. This component only has a single task that runs at the start of the application, which is to determine the target of the simulation and create the initial state of the mass-spring model. Creating this initial state requires determining the positions of all the particles that form the model, as well as what particles are connected by each type of spring. This information of the initial state will be passed directly to the simulator, when the latter starts.

Creating a custom editor for this manager provides us with the ability to show or hide fields of properties. Taking advantage of this, we create an editor that adapts to the user's current choice:

- If the target mode is set to "Imported Object", then a field to point to the reference of the object is shown;

- Otherwise, if the target mode is set to “Rectangular Cloth”, a field for the dimensions to be specified appears.

Showing only the useful properties to the user, given his current choice is usually a good approach as we don’t want to overwhelm or show the user properties that will not be used by the manager. Figure 3.3 shows the custom editor of this script on both available modes.

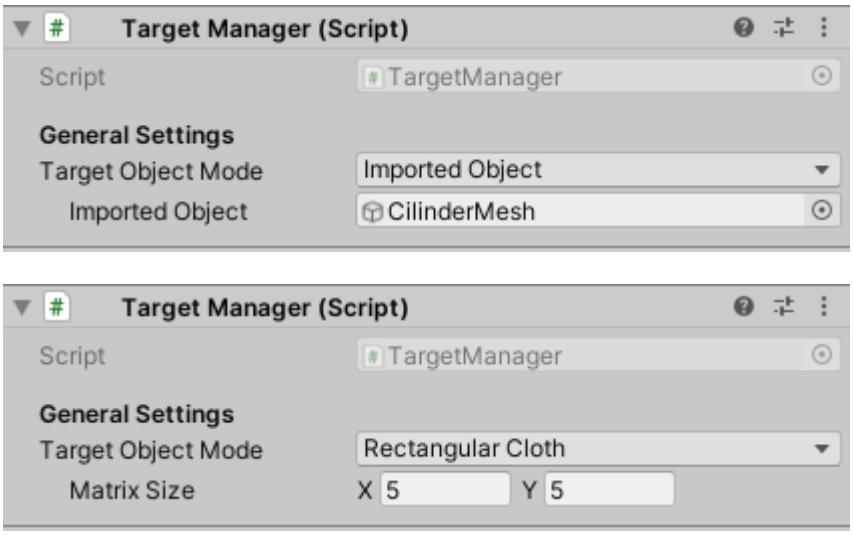

Figure 3.3: Custom editor for the target manager with the mode different modes: “Imported Object” on top image and “Rectangular Cloth” on the bottom image.

All the properties that can be set by the user are explained on Table 3.1 below.

Property	Type	Definition
Target Object Mode	Enum	Determines the mode of the target for the simulation. Can be “Imported Object” or “Rectangular Cloth”
Imported Object	GameObject	Reference to the Game Object that will serve as the target for the simulation. (Only available in “Imported Object” mode)
Matrix Size	(Int, Int)	Number of particles in each dimension for the generated cloth. (Only available in “Rectangular Cloth” mode)

Table 3.1: Description of the properties present in the Target Manager editor.

As a last note, the “Matrix Size” property is a vector of two integers that will generate a piece of cloth with the number of particle in each dimension given by this vector. Thus, a Matrix Size of (5, 5) would create a square mesh with 25 particles in total as can be seen on the figure below with all the expected interactions. In the case of “Imported Object”, the implementation is discussed later on section 3.6.2.

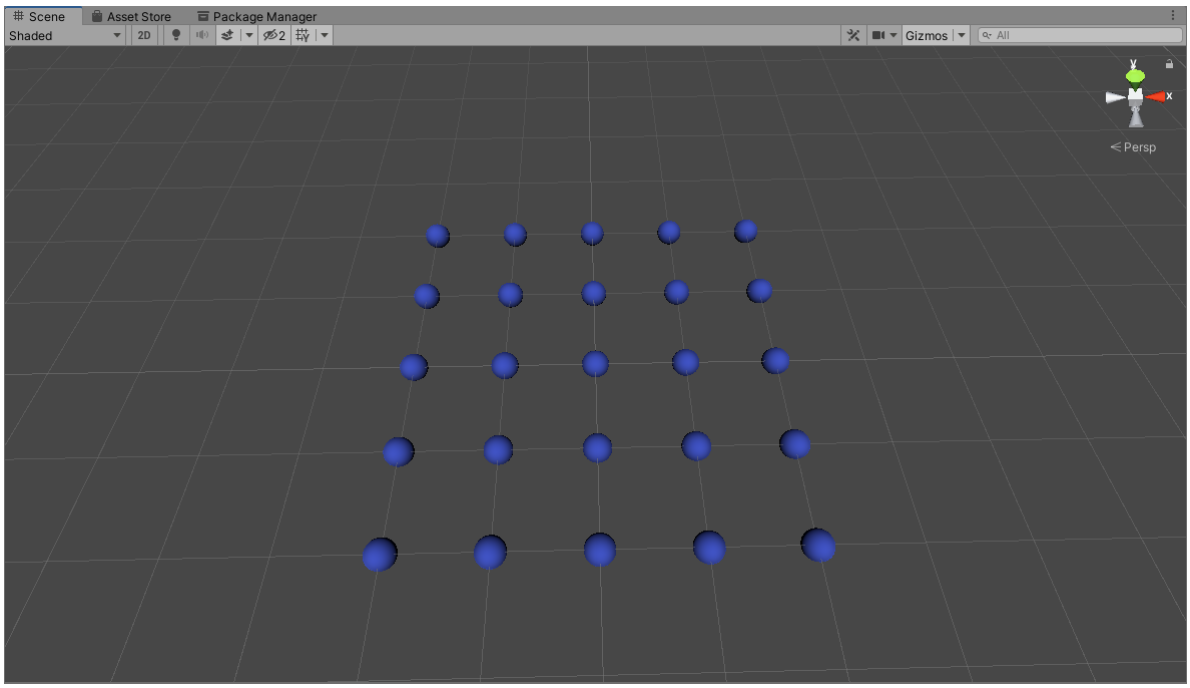

Figure 3.4: Rectangular cloth target created by the framework with the dimensions 5x5. The particles are represented by the blue spheres on the image.

3.4.2 Particle Displayer

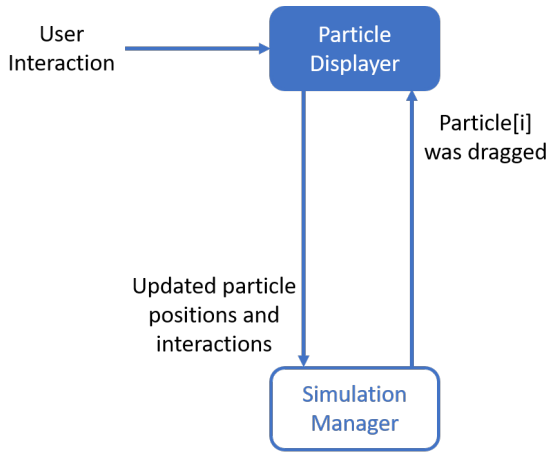

Figure 3.5: Overview of the responsibilities and communication of the Particle Displayer component with the other components.

The particle displayer is the component responsible for the visualization of the simulation as well as handling any interaction that the user makes through the editor view.

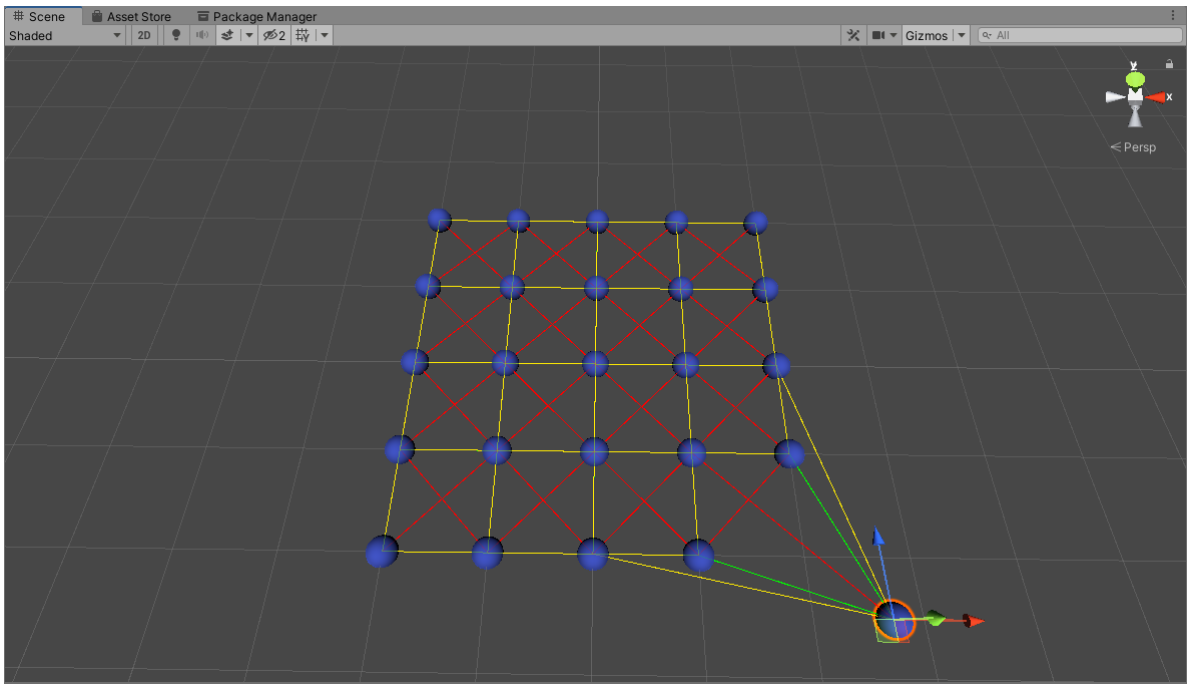

Figure 3.6: Visualization of the simulation provided by the displayer component. The colors in this image match the colors used previously on Figure 2.6: blue for particles, green for stretch springs, red for shear springs and yellow for bend springs.

At the start of the application, this script creates sphere objects on the coordinates of the particles from our mass-spring model. Then, on the Update loop, for each frame, this component will perform the following tasks:

1. Check if the user has interacted with any particle; if there has been any user interaction, this will be communicated to the Simulation Manager alongside the index of the moved particle as well as it's new location. Since the displayer does not need to know what type of simulation we are performing, we let the simulation manager carry the responsibility of relaying this information obtained to the simulator component.

2. Retrieve from the Simulation Manager the latest positions that have been computed by the simulator and, afterwards, update all the spheres in the scene to their newest positions for rendering;

3. Draw all the spring interactions that exist between the pairs of particles, by connecting them with a line.

In relation to the first step of detecting user interaction, this component can look for changes on the *Transform* of each particle, however this component also updates the position of particles on the second step. Overcoming this issue can be solved by using a "dirty" flag for the *Transform* that Unity already provides. At the start of each loop, on the first step, we will search for the particles that are marked with this flag; then, on the

second step, after updating each particle's position, we can manually set this flag back to it's original clean state.

In terms of the third step, we should only draw the interactions that we are using currently in our simulation. *E.g.* if we wish to simulate a model were there are no bend forces, the displayer component must detect if these types of interactions are enabled or disabled and draw them accordingly. By default, the colors used for the springs are the ones presented on Figure 3.6, in order to match the colors on Figure 2.6, however these can be changed on the component editor seen below.

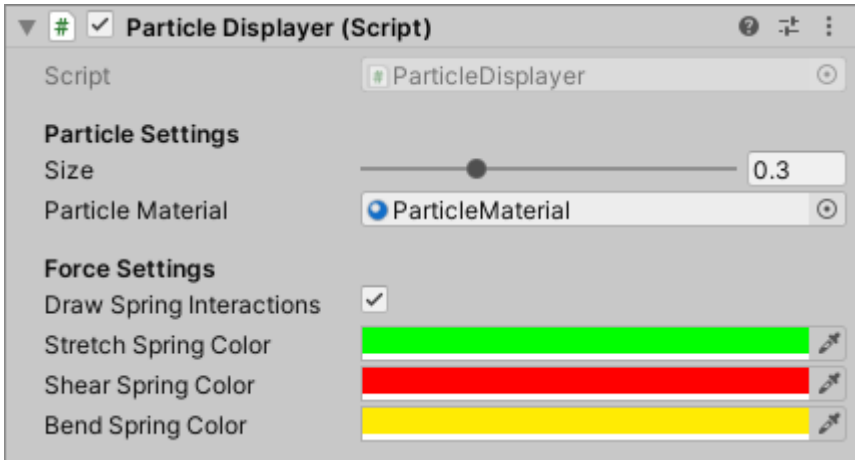

Figure 3.7: Preview of the editor for this component with the associated variables

This is the component editor, which uses Unity's default editor, that the user can change to adapt the visualization. While the names of the properties are fairly clear, Table 3.2 below provides a definition for each of them.

Property	Type	Definition
Size	Slider with range [0., 1.]*	Scale of the spheres spawned to display where the model particles are.
Particle Material	Material	Material to apply to the spheres which describes many visual properties, including color.
Draw Spring Interactions	Boolean	Enable or disable the lines that represent the spring interactions
Spring Color	Color	Color for the line that represents this type of spring interaction

Table 3.2: Description of the properties present in the Particle Displayer editor.
* The range of the slider can be adjusted on the Particle Displayer script.

3.4.3 Simulation Manager

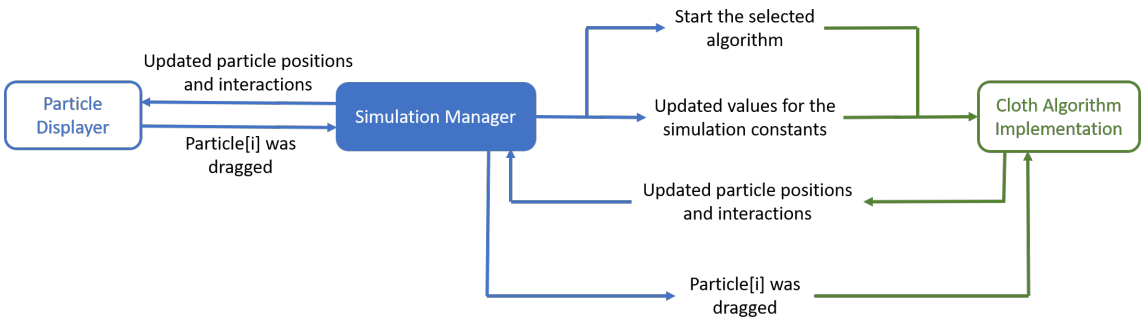

Figure 3.8: Overview of the responsibilities and communication of the Simulation Manager component with the other components.

Time to shift our attention now to the Simulation Manager component, which is the more complex manager due to it's core role. It provides main source of communications to and from the simulator component:

- Defines what type of integration algorithm will be used as the simulator;
- Defines the mass of the particles, the damping strength as well as any other additional constants required by the specific integration algorithm used;
- Controls properties related to the spring forces, *i.e.* if they are enabled, their strength and equilibrium distance;
- Enables or disables external forces, as well as controls properties that are specific to each external force (*e.g.* as how strong the effect of the gravity is);
- Initializes the appropriate simulator at the start of the application, and call the simulation update step on the FixedUpdate loop;
- Communications between the Displayer component and the simulator component, *e.g.* communicate to the simulation when the displayer detects the user has interacted with the target, as well as returning the updated positions provided by the simulator to the displayer.

Creating an interactive medium between the user and the simulation is important, as it provides direct feedback to changes on any of the properties used by the simulation. However, since there are a lot of properties to fiddle with, using a custom editor again is the best approach as to not overwhelm the user with variables.

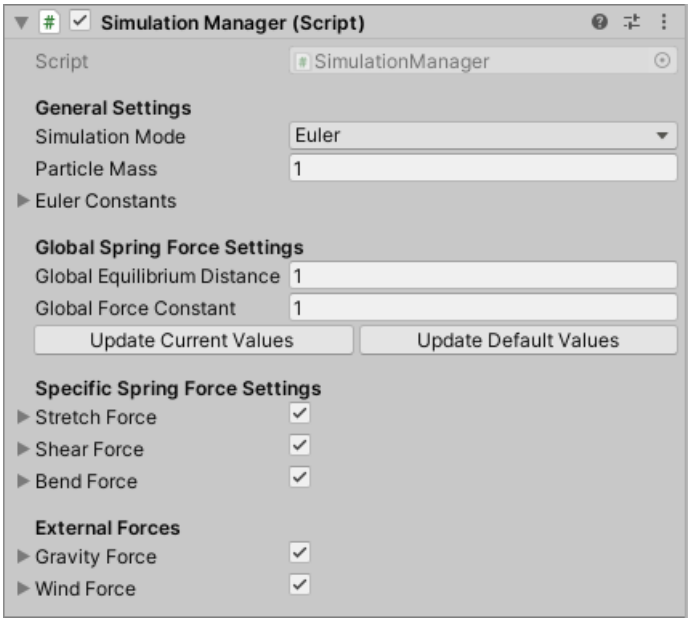

Figure 3.9: Preview of the Simulation Manager editor with some properties folded

As 3.9 shows there are a lot of fields inside this editor, so let's divide it into parts and briefly analyze each part that constitutes it. The first section are the "General Settings", and it contains the starting point for our simulation, which is the algorithm for performing the simulation as well as the variables required by that algorithm. In the current version of the framework there are two simulation modes: Euler Integration and Verlet Integration each of them with their own constants.

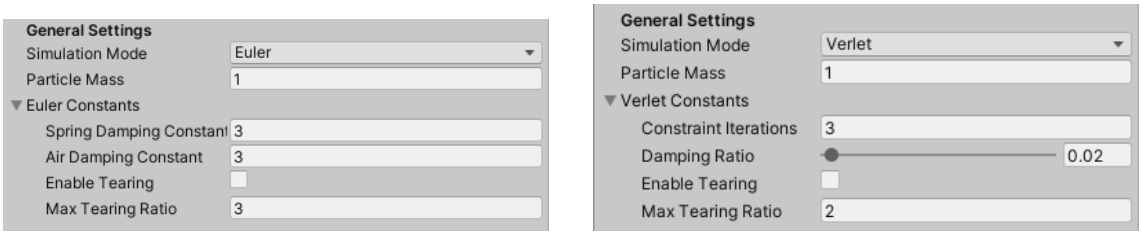

Figure 3.10: Preview of the general settings in the Simulation Manager editor, when selecting Euler mode (left) and Verlet mode (right).

The definition of each property is presented on the table below

Property	Type	Definition
Simulation Mode	Enum	Integration mode to use while performing the simulation (Euler Integration, Verlet Integration)
Particle Mass	Float	Mass to apply to each particle on the simulation
Spring Damping Constant	Float	Constant of the spring damping effect explained on Eq. (2.4). (Euler Mode)
Air Damping Constant	Float	Damping effect to simulate air friction and prevent continuous drifting after impulse. (Euler Mode)
Enable Tearing	Boolean	Allow spring interactions to rupture when under too much stress. (Euler Mode)
Max Tearing Ratio	Float	Constant to define maximum allowed stress by the spring before tearing. (Euler Mode)
Constraint Iterations	Int	Number of iterations to satisfy the interaction constraints. (Verlet Mode)
Damping Ratio	Slider with range [0., 1.]	Constant to define the damping effect on the velocity of the particles. (Verlet Mode)
Enable Tearing	Boolean	Allow constraints to break when under too much stress. (Verlet Mode)
Max Tearing Ratio	Float	Constant to define maximum allowed stress by the constraint before tearing. (Verlet Mode)

Table 3.3: Description of the properties present in general settings section of the Simulation Manager editor.

The next section that is shown in the editor is related to the spring forces. This part of the chapter has a “Global” part and a “Specific” part. The global part serves merely as utility and does not affect the simulation at all: it allows us to set the values for all the other forces at the same time. When setting all the forces, using this utility tool, the equilibrium distances of each force are set for a matrix of particles such as the ones generated by the Target Manager component, as visible in Figure 3.4, by taking into account the extra distance between particles on shear and bend strings compared to stretch springs.

The “Specific Spring” section controls the constants for each type of spring force that we are using on our model. Clicking the toggle enables or disables that specific interaction from being present on the simulation.

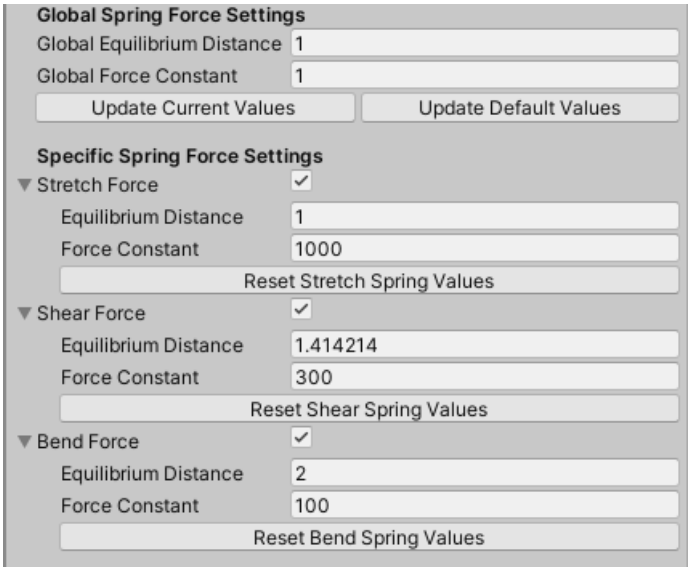

Figure 3.11: Preview of the spring force settings in the Simulation Manager editor.

Property	Type	Definition
-	Toggle/Boolean	Enable or disable that spring type for the simulation.
Equilibrium Distance	Float	Default equilibrium distance for that type of string interaction.
Force Constant	Float	Spring force constant for that type of string explained on Eq. (2.1)

Table 3.4: Description of the properties present in spring force settings section of the Simulation Manager editor. The global force settings are not included as their purpose is simply utility and does not affect the simulation, as explained above.

The last part of this editor contains the properties of the external forces. Currently, there is only gravity and wind force implemented, as they are simple to emulate, and so they are the ones present on this section.

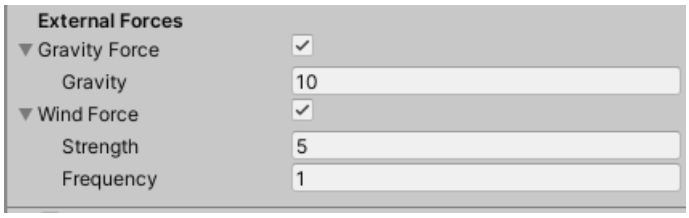

Figure 3.12: Preview of the external force settings in the Simulation Manager editor.

Property	Type	Definition
-	Toggle/Boolean	Enable or disable that external force for the simulation.
Gravity	Float	Magnitude of the gravity force.
Strength	Float	Maximum magnitude of the wind force.
Frequency	Float	Oscillation rate of the wind force.

Table 3.5: Description of the properties present in external force settings section of the Simulation Manager editor.

As a way to improve clarity over which forces are enabled and which forces are not,

when a force is disabled, all the properties related to it are also hidden, regardless of being an external force or a spring force. This keeps the editor as clean as possible, while making it easy to identify which forces are enabled with a quick glance.

Figure 3.13: Effect of disabling a force on the Simulation Manager editor.

3.5 Particle and Simulators

Now that we have looked through the tasks and functionalities of each manager, the last remaining component of the framework is composed by the Particles and Simulators. These two parts work in tandem and communicate with the managers to obtain the required information to perform the simulation. The framework provides two classes, BaseSimulator and BaseParticle, that serve as the base to extend from, when creating a new component of this type (thus the name **extendable components**). The included examples in the framework, Euler Integration and Verlet Integration, both use the structure provided by these classes. The development of these two examples is covered later on, on section §4 of this thesis.

3.5.1 BaseParticle and BaseSimulator

The importance of these classes lies with allowing proper communication with the manager components. By providing a set of public methods and variables, the managers know how to communicate with these and where to find the information they require.

Starting with the BaseParticle, this class's responsibility is to save the information about the particle state: its current position, particle mass, the total force accumulated for that step and if we want it to be a fixed point that won't be affected by the simulation.

The BaseSimulator contains an array of all the Particles that form the model and the indices of the particles that have a spring interaction between them. Besides these properties, it also provides some methods that serve as a communication channel to the managers (such as when to start the simulation or when to simulate the next step), as well as empty methods that are meant to be overridden by any other class that inherits from this (such as a method to perform the next step of the simulation).

Given this structure, to create a new Simulator component, the developer can extend

the BaseParticle class to add any additional required info. Then, the developer must extend the BaseSimulator class and override the required methods for the simulation to be performed, which are the method that defines the initial conditions when starting the simulation, the method to handle what to do when a user drags a particle to a new position and, finally, a method to perform a stop on the simulation. An overview of this structure is available on the figure below.

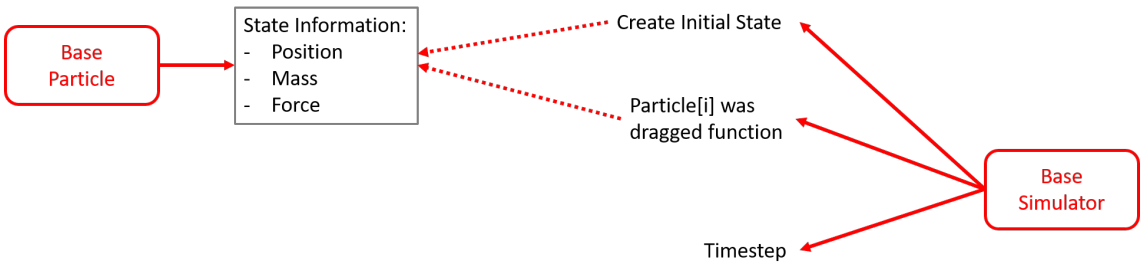

Figure 3.14: Overview of the BaseParticle and BaseSimulator, and the methods required to implement when extending them. As we have seen, the base simulator receives the initial state data from the Target Manager, and the information that a particle was dragged from the Simulation Manager.

3.6 Remaining Features

Now that we have completed the discussion of the components that generate the framework, there are still two features that remain to be addressed: the presence of environmental external forces, such as wind and gravity to affect the simulation, along with the capability to import models from other 3D external modeling software, such as Maya or Blender.

3.6.1 External Forces

This framework contains a naive implementation of two external forces that are commonly used when performing simulations: gravity and wind.

The implementation of **gravity** is determined by the equation

$$F_g = mg, \tag{3.1}$$

where m is the mass of the particle and g is a constant chosen by the user. The external force emulating gravity added to the particles at each frame, by definition, always points down.

The other force is **wind**, and it's implemented as an oscillating force, increasing and decreasing over time periodically, given by the following equation

$$F_w = A \cdot (1 - \cos^2(\pi v t)), \tag{3.2}$$

where A is the maximum strength of the force, υ is the frequency and t is the current time. In the current implementation, the force always points in the same direction. The plot of the function can be seen below

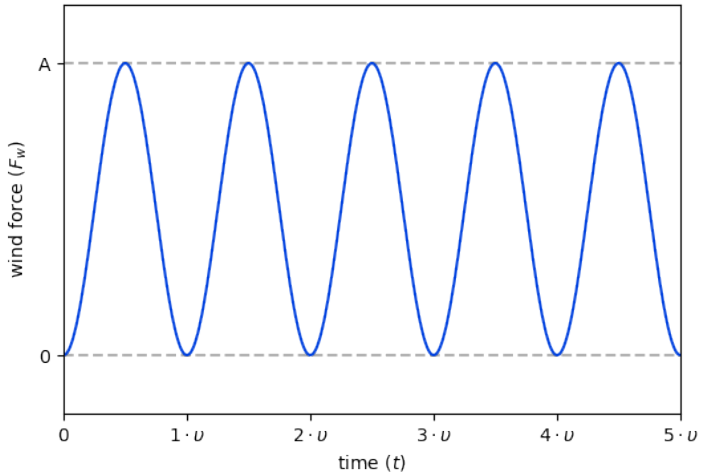

Figure 3.15: Plot of the wind force as an oscillating force of amplitude A and frequency υ

External forces are important because not only do they provide a more realistic behavior to the simulation model, but they also provide some insight on how the simulation reacts to changes in its surrounding environment. The base simulator class provides access to a method that checks if these forces are enabled on the simulation settings and, if so, iterates over the particles adding the force to all the particles on the model.

3.6.2 Importing Models

The remaining feature to be discussed is the ability to import any model produced outside the framework. This provides the ability to test cloth simulation algorithms in more complex targets than the built-in rectangular cloth piece. Unity already has the capability to import all the standard 3D model file formats used by the CG industry, such as FBX and OBJ.

When the model is loaded into Unity, we will be able to access its mesh and, using the mesh information, the goal is to create a model that follows the standard that we've been following in regards to particles and multiple different springs as seen on Figure 2.6. In order to make this task less error prone, we require the imported object's mesh to be built using quads as the base shape.

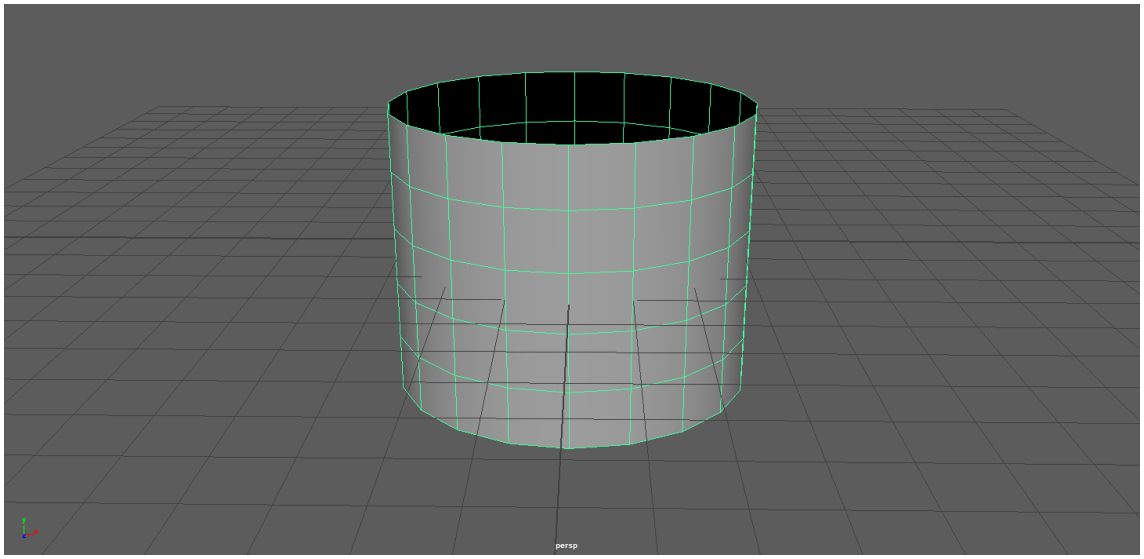

Figure 3.16: A 3D cylinder model created in Maya, a 3D modeling software, using quads for the mesh. The cylinder contains 5 subdivisions on its height and 20 radial subdivisions.

Although Unity always uses triangular meshes for rendering, if the original imported mesh uses quads then it also keeps the original information of the quad mesh.

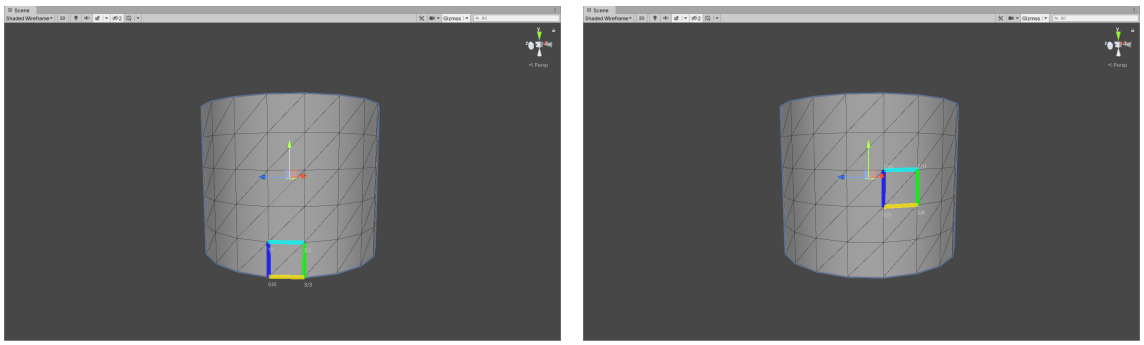

Figure 3.17: 3D cylinder model from 3.16 imported into Unity. The rendering shows the a triangular mesh, but the mesh also contains the quad mesh information. The left and right images emphasize two different quads of the mesh obtained by iterating through the indices of the mesh in groups of 4 indices at a time.

The first step towards creating our mass-spring model from the mesh is to determine the positions of all our particles. As discussed on section 2.2, the particles are placed at the edges of the vertices, so we can obtain them by iterating over each vertex of the mesh and save its position to an array. The array will contain all the initial positions of our mass-spring model particles.

The next step is to determine where we will create springs interactions. The three interactions we defined were:

- Stretch springs: placed between a particle and all the directly adjacent particles. We can obtain these by iterating over the quads that contain the particle and adding the particles directly connected on the same quad;

- Shear springs: placed between a particle and the diagonally adjacent particles. Similarly to the stretch springs, we can iterate over the quads that contain that particle and adding the particle that's on the opposite edge of the quad;
- Bend springs: placed between a particle and the particles on every other row/column on the adjacent directions. To find these, we iterate over the adjacent quads and search it for the appropriate particle.

Note that adjacent quads, share an edge and, consequently, two vertices, so some caution needs to be had, in order to prevent adding the same interaction multiple times. Another important factor is that these models can be arbitrarily complex and so we can't rely on all the spring interactions of a specific type sharing the same equilibrium distance, since the size and shape of the quads can vary along a mesh. Therefore, each spring interaction will have its own resting distance value; if we considered that the provided imported model is in an equilibrium state at the starting provided position, then the resting distance of each spring is defined as the distance between the particles that interact through that spring.

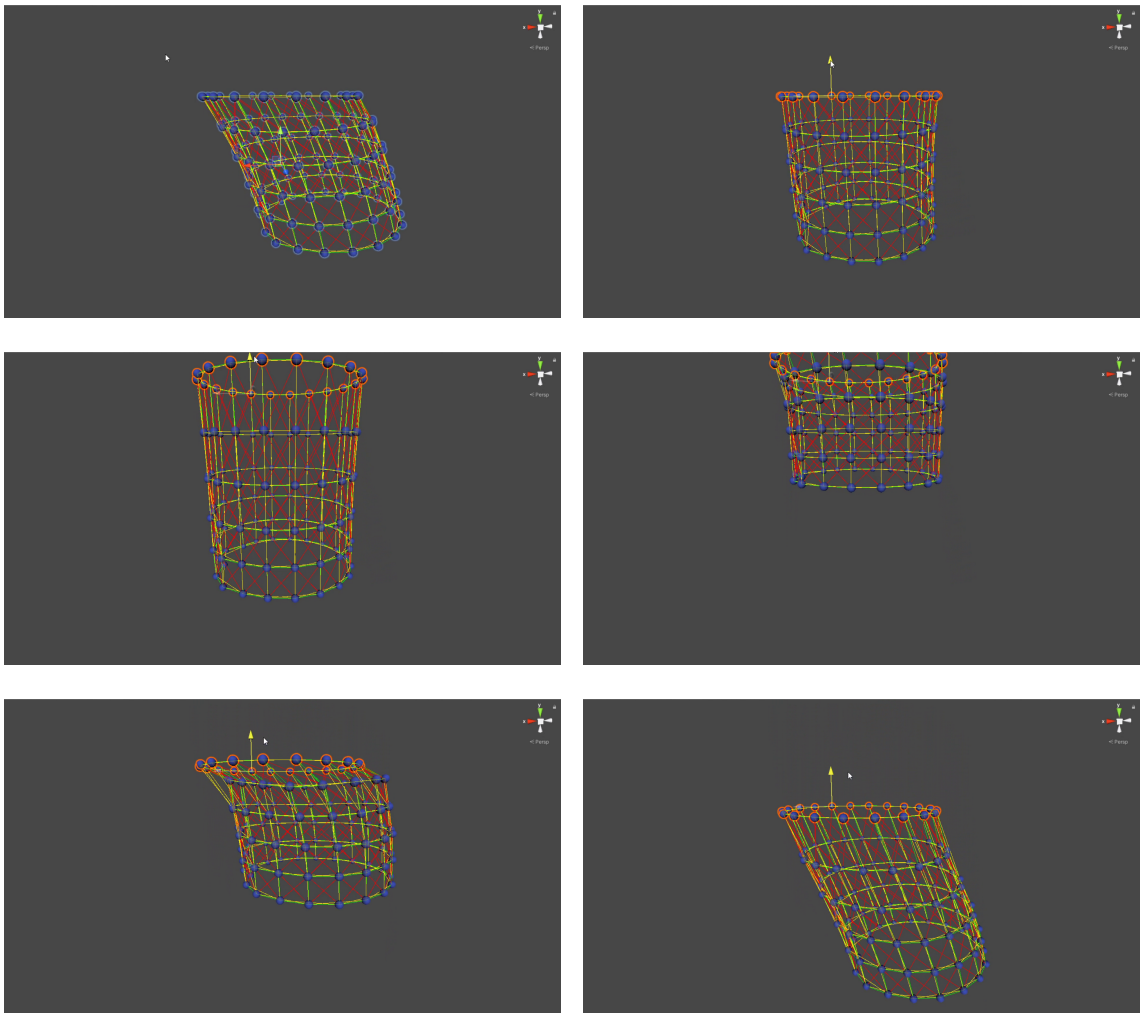

Figure 3.18: 3D Model of cylinder imported by the framework as a mass-spring model, which was subsequently fed to the symplectic Euler simulation algorithm. The top line of particles' positions were fixed. We used strong external forces with weaker spring forces to emphasize the model reacting to the environment and user interaction.

On this figure above, we can see the same cylinder that was created in Maya, as a mass-spring object, as the target for the simulation and the top particles of this model were set as having fixed positions. On the first two images we see the imported model reacting to the external forces: the gravity is making the model longer, and the wind causes a sideways deformation. The four remaining images display the user interacting with the top row of the particles from our model by pulling them up, and the model reacting to this interaction by springing the model up as well, and then moving back close to its original shape.

3.7 Conclusion

On this chapter, we hope the reader has learnt about what components form the framework along with their purpose, as well as the implementation of the remaining key features that are important in improving the flexibility of the framework. This chapter started with

an overview of the framework, it's abstraction and extendable components and how they interact. The abstraction components is composed of a set of managers (**Target Manager, Particle Displayer and Simulation Manager**) that create a communication layer between all the other parts: the engine, the user and the simulator. The extendable component (**BaseSimulator with BaseParticle**) serves as a base for developers to extend when implementing new algorithms.

Following up, we take a more in-depth look at each of these manager components, such as their roles and their communications, along with what control the user has over them through the User Interface (UI). We close the discussion with a section on the implementation of external forces, whose value is calculated for each frame and added to all the particles in the simulation, and importing external 3D models as part of the framework.

4 Physical Algorithms for Cloth Simulation

Now that we have discussed the tools that the framework provides us with, we can place ourselves on the shoes of a user that wishes to add new cloth simulation methods to the framework. The framework includes two built-in integration algorithms for performing the simulations: Euler Integration and Verlet Integration. This chapter delves into the implementation of each of them and presents some results achieved with the current implementations.

4.1 Euler Integration

We have already provided theoretical insight on the topic of Euler Integration, on section 2.7, and we established the following approximations updating velocity v and position r discretely

$$\boldsymbol{r}^{t+1} \approx \boldsymbol{r}^t + h \cdot \boldsymbol{v}^t \tag{4.1}$$

$$\boldsymbol{v}^{t+1} \approx \boldsymbol{v}^t + h \cdot m^{-1} \cdot \boldsymbol{F}^t. \tag{4.2}$$

In addition to these equations, on section 2.6.1 and section 2.6.3, we also defined the spring forces and spring damping forces as

$$\boldsymbol{f}_{ij}(\boldsymbol{r}) = k_s \left(\|\boldsymbol{r}_j - \boldsymbol{r}_i\| - L\right) \widehat{\boldsymbol{r}_{ji}} \tag{4.3}$$

$$\boldsymbol{d}_{ij}(\boldsymbol{v}) = k_d \cdot (\boldsymbol{v}_j - \boldsymbol{v}_i). \tag{4.4}$$

Armed with this knowledge, we can start discussing the implementation of this technique. Eq. (4.1) and Eq. (4.2) show that we need to keep track of each particle's positions, velocity and mass. Moreover, we'll need to calculate the sum of forces that arise from different sources, ergo we also need an accumulator that saves the current sum and is reset after every time step. All these properties can be stored in a class, called EulerParticle, that extends the BaseParticle. The base class provided by the framework already accounts for most of these properties, so we only need to add the velocity variable to EulerParticle. To finalize this class, we also need a time step function that updates the position r and velocity v for that particle, using equations 4.1 and 4.2.

Next up, we need to extend the BaseSimulator into the EulerSimulator class, which will take care of computing all the forces. In terms of external forces, the BaseSimulator

is already capable of computing them by iterating through the particles and adding the external forces to the particle's force accumulator. Then, calculating the forces associated with the springs is also relatively straightforward, as all the spring interactions are saved through the indices of the particles they connect. Iterating again through the array of particles we access the connected particles, compute their spring forces as well as damping forces, adding them to the particle's force accumulator. Since the forces applied on particle i and j are opposite, we can add them to the accumulators of both particles, in order to avoid recalculating the same interaction when the iteration reaches the other connected particle. The simulation step is then defined by calculating all the forces (internal and external) applied to each particle, and then performing the time step function on each EulerParticle.

To conclude the EulerSimulator class, we need to handle the initial state of the particles and determine how to handle user interaction. The initial positions for the particles are obtained from the Target Manager, and we define their initial velocity to be zero. As for the user interaction, we can can set the new position of the dragged EulerParticles as the new positions given by the Particle Displayer; the velocity could be set to zero, but I decided instead to approximate the velocity by the difference of the positions before and after the user interaction. Using this approximation for the user interaction keeps the momentum of the particle in case the user wants to flick particles around.

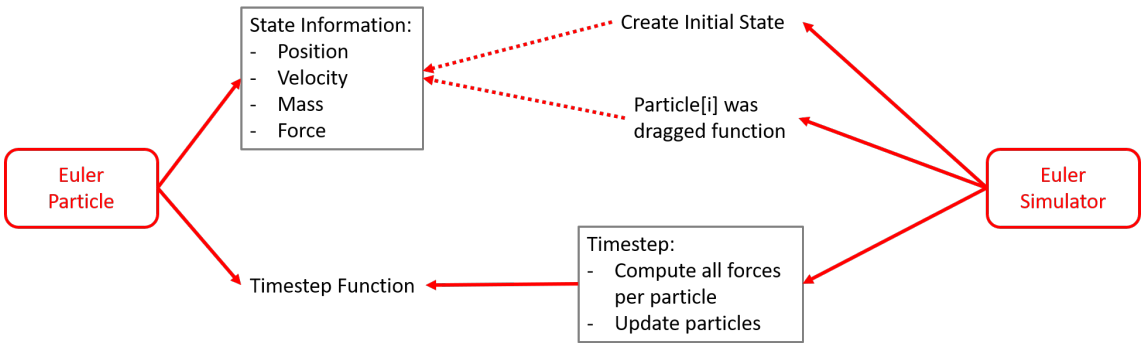

Figure 4.1: Overview of the EulerParticle and EulerSimulator and the methods implemented.

When testing this implementation, we used high spring constants to keep the particles closer together and simulate a non-stretchy material, but the cloth was very unstable and the structure would break down within the first few frames. With low spring constants, the cloth became very droopy which led to the strings becoming too stretched and instabilities also started to surge. These instabilities, as we've discussed previously, are common and expected due to the approximations used in this method.

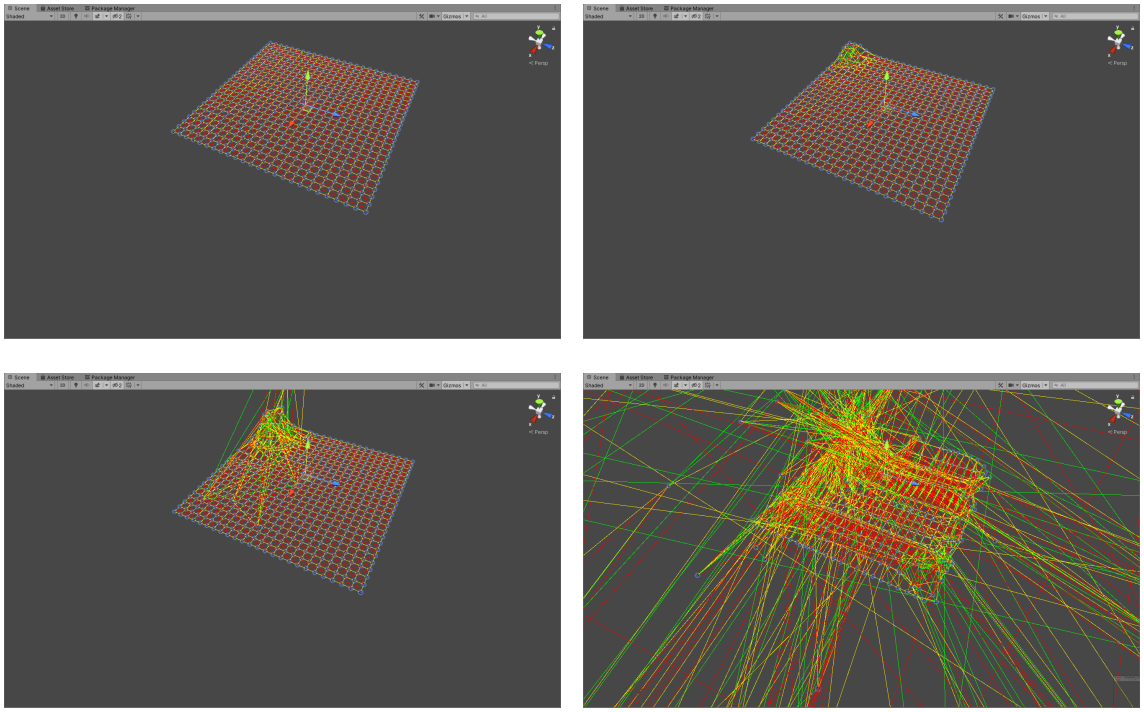

Figure 4.2: Display of the instabilities of the Euler Integration. These images were obtained from the simulation using high values for spring constants on a matrix of dimensions 25x25, where the upper left particle's position was fixated. The external forces present were gravity and wind.

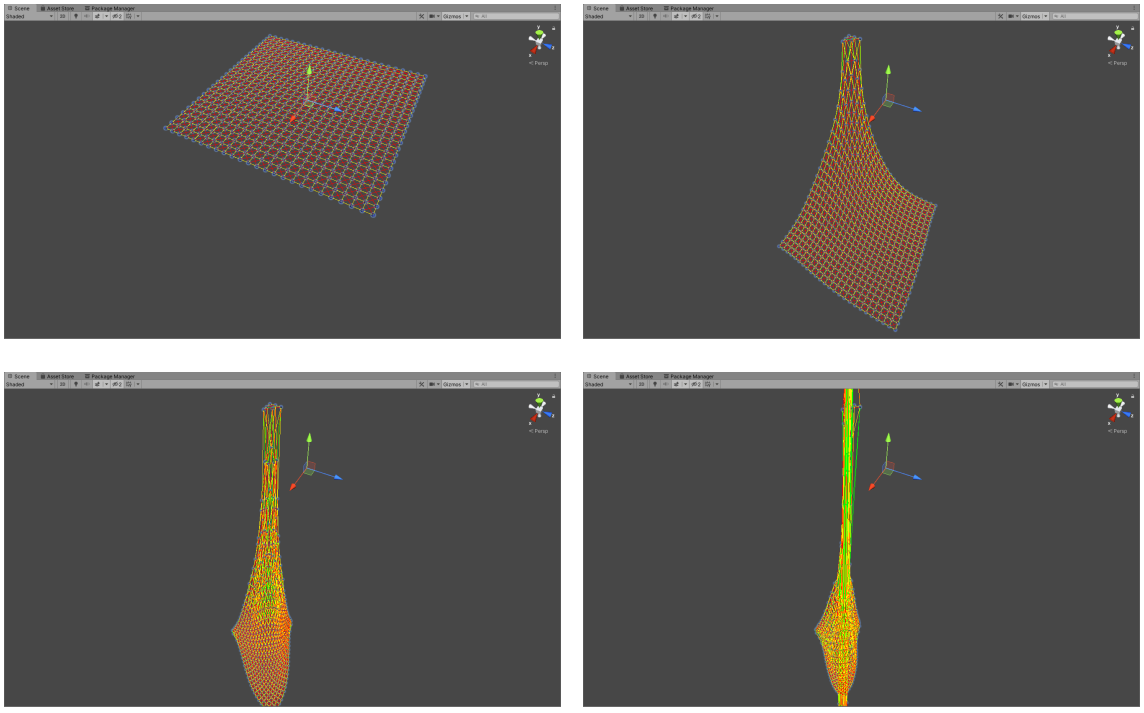

Figure 4.3: Display of the instabilities of the Euler Integration. These images were obtained from the simulation using low values for spring constants on a matrix of dimensions 25x25, where the upper left particle's position was fixated. The external forces present were gravity and wind.

As these figures show, Euler's Method would be an unreliable solution to cloth simulation. On the first Figure 4.2, the instabilities within a few frames of the simulation. On the second Figure 4.3, these instabilities don't start to show until the particles are under

a lot of stress: on the two top images, the cloth is still behaving realistically, without any issues; however, on the bottom left image, we start visualizing a lot of jitter on the top simulated particles; finally, on the bottom right image, these jittering particles have shot through space and are now very far away.

4.2 Symplectic Euler Integrator

The previous chapter displays the inherent issues with using Euler Integration. Despite that we can try to improve upon it, to create a more stable integration algorithm.

Euler Integration and Verlet Integration both belong to a type of integrators referred to as explicit integrators. On these methods, we try to solve the position of the i-th particle on the next time step $\boldsymbol{r}_i^{t+1}$, using the data from the state $\boldsymbol{q}_i^t = \langle \boldsymbol{r}_i^t, \boldsymbol{v}_i^t, \boldsymbol{a}_i^t \rangle$ evaluated at the current time step t. On the other hand, there are the methods know as implicit integrators, that calculate $\boldsymbol{r}_i^{t+1}$ using the state in the same time step $\boldsymbol{q}_i^{t+1} = \langle \boldsymbol{r}_i^{t+1}, \boldsymbol{v}_i^{t+1}, \boldsymbol{a}_i^{t+1} \rangle$. Using this idea, we can obtain the following equations[4, 38]:

$$\boldsymbol{r}^{t+1} = \boldsymbol{r}^t + h \cdot \boldsymbol{v}^{t+1} \tag{4.5}$$

$$\boldsymbol{v}^{t+1} \approx \boldsymbol{v}^t + h \cdot m^{-1} \cdot \boldsymbol{F}^{t+1}. \tag{4.6}$$

This is denominated as **Backwards Euler/Implicit Integration**. This technique is more common, because it's a lot more stable than using explicit Euler integration, thus allowing for bigger time steps on the simulation [4]. Obviously this comes at the cost of being a more difficult and more computationally expensive equation to solve.

There is a middle-ground between these two different approaches referred to as **symplectic Euler** or **semi-implicit Euler**, where we solve the velocity explicitely and the position implicitly. Using this technique, the equations of motion are as follows

$$\boldsymbol{r}^{t+1} = \boldsymbol{r}^t + h \cdot \boldsymbol{v}^{t+1} \tag{4.7}$$

$$\boldsymbol{v}^{t+1} \approx \boldsymbol{v}^t + h \cdot m^{-1} \cdot \boldsymbol{F}^t. \tag{4.8}$$

The usage of symplectic integrators has been successfully used by Eberhardt et al. [20], Choi and Ko [14], and it provides a more stable solution than explicit Euler Integration. This model will still not be perfect, due to the usage of a first order approximation on the finite differences approximation, nonetheless it should still yield good results. It's also possible to use symplectic integration with higher order finite differences approximations, to further improve the accuracy of the results [20, 14].

In terms of the implementation, there is only a small and very simple change to make on the EulerParticle class. The method for performing the time step in each particle must first update the velocity given the current particle state $\boldsymbol{q}_i^t = \langle \boldsymbol{r}_i^t, \boldsymbol{v}_i^t, \boldsymbol{a}_i^t \rangle$ and, afterwards, update the position of the particle with the updated velocity value.

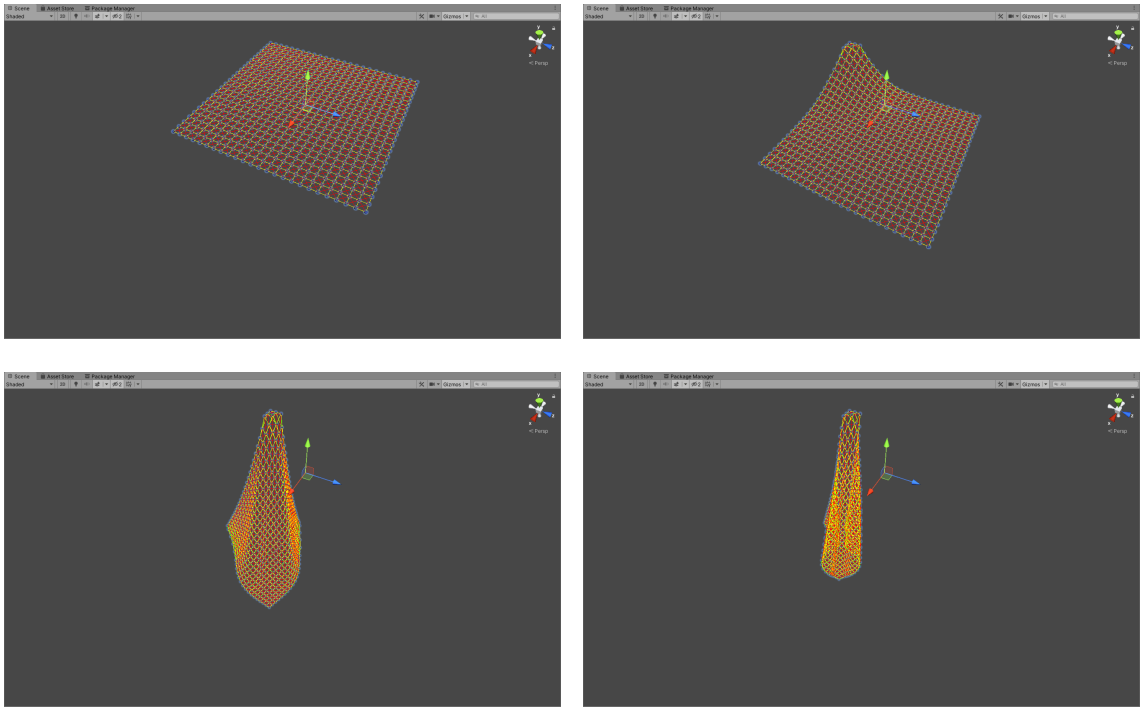

Figure 4.4: Display of Symplectic Euler Integration. These images were obtained from the simulation using high values for spring constants on a matrix of dimensions 25x25, where the upper left particle's position was fixated. The external forces present were gravity and wind.

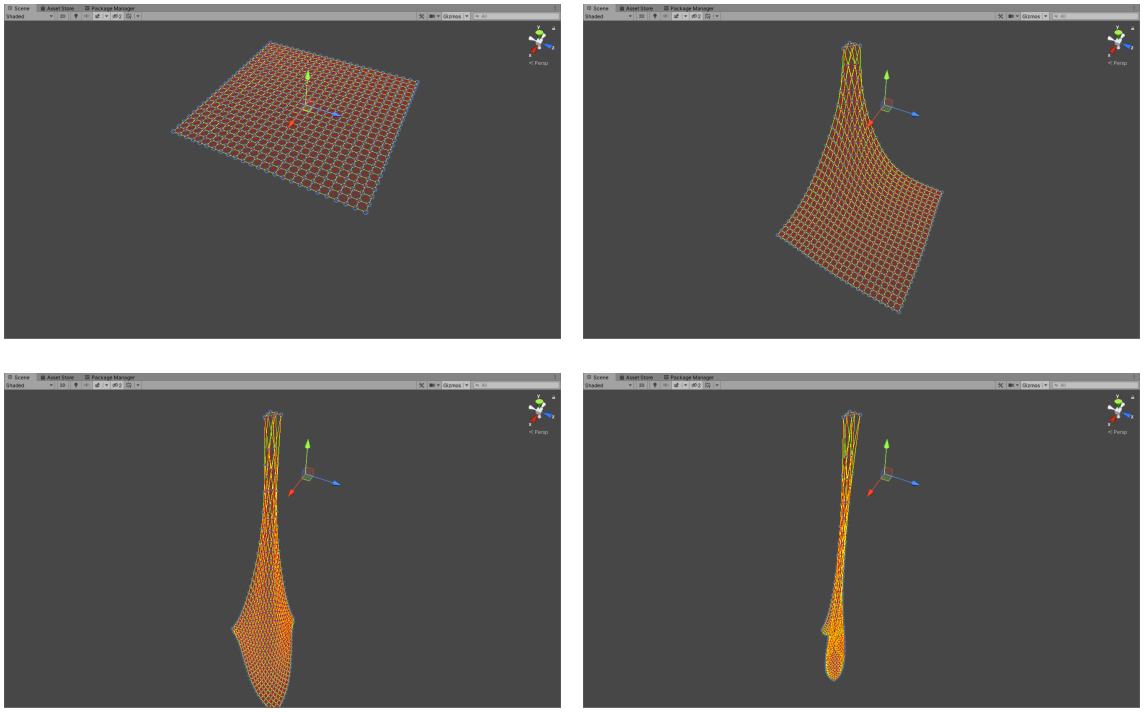

Figure 4.5: Display of Symplectic Euler Integration. These images were obtained from the simulation using low values for spring constants on a matrix of dimensions 25x25, where the upper left particle's position was fixated. The external forces present were gravity and wind.

These simulations displayed on Figure 4.4 and Figure 4.5 use the exact same values for the spring constants as the equivalent figures in the previous section (Figure 4.2 and Figure 4.3 respectively). There are no signs of instabilities nor are there any visible signs of jittering. When performing the simulation with low spring force constant values, we still get the same droopy appearance, as expected.

Comparing the results between explicit Euler and symplectic Euler, the latter integration scheme shows a great improvement in the matter of stability, and can provide a reliable solution to the cloth simulation problem.

4.3 Verlet Integration

Verlet Integration has already been discussed theoretically previously, on section 2.8, where the formula for updating the positions over time was defined as

$$\boldsymbol{r}^t \approx \boldsymbol{r}^t + (1 - \alpha_d) \cdot (\boldsymbol{r}^t - \boldsymbol{r}^{t-1}) + (\Delta t)^2 / m \cdot \boldsymbol{F}, \tag{4.9}$$

alongside the idea of using constraints to represent the string interactions instead.

The implementation of this new method is similar to the previously implementation of the Euler Integration studied on section 4.1. Eq. (4.9) demonstrates that each particle should save its current position, previous position, mass and accumulated force. We extend the class BaseParticle, to create the new class that will hold this information, called

VerletParticle. Since the framework's base class already accounts for most of these variables, we only need to add the previous position variable to the class. This class is completed by adding a time step function that, if the current particle is not fixed, it updates its position by applying the formula displayed on Eq. (4.9).

The next step is to extend the BaseSimulator, in order to create the simulation class for this algorithm, VerletSimulator, whose responsibility will be to compute the external forces and attempt to satisfy all the constraints set. Remembering that this method does use not spring forces, it only needs to calculate the external forces and add them to each particle's force accumulator. Satisfying the constraints is done by moving both particles alongside the line that connects them, each by half the amount required to satisfy the constraint; since both particles move half the distance, at the end of both movements, the constraint is satisfied. As a small caveat, since particles can be fixed, we need to account for that in the constraint satisfaction step: if only one of the particles that form the constraint can be moved, then it shall move the full distance to satisfy the constraint as seen on Figure 4.6. The simulation step is implemented as computing all the external forces exerted on the particles, performing the time step function on each VerletParticle, and finalizes by attempting to satisfy the constraints, iterating over them multiple times.

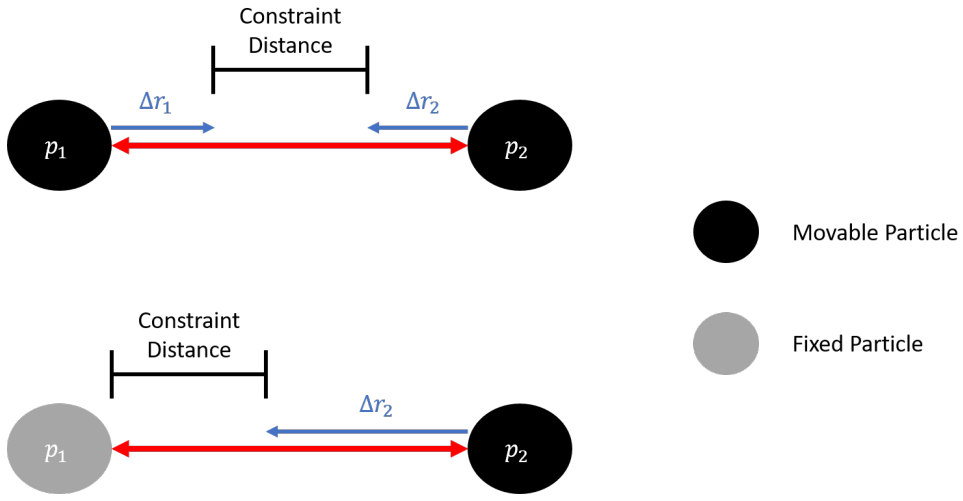

Figure 4.6: Satisfying the constraints on the Verlet integration algorithm with none or one fixed particles.

To conclude the VerletSimulator class, we need to set the initial state of the particles, create the constraints from the springs and handle the user interaction. Similarly to the EulerSimulator, the initial positions are obtained by the Target Manager, and since we want the particles to not have any velocity the start of the simulation, we set the previous positions as the same as the new positions. Creating the constraints is also done at the start, and for each spring interaction, we create a constraint for those particles, with the same resting distance as the one defined for that type of spring interaction. As for the user interaction we also attempt to keep the momentum given by the user to the particle;

we set the new position of the particle as the position given by the manager and we set the previous position to the particle's position before the interaction.

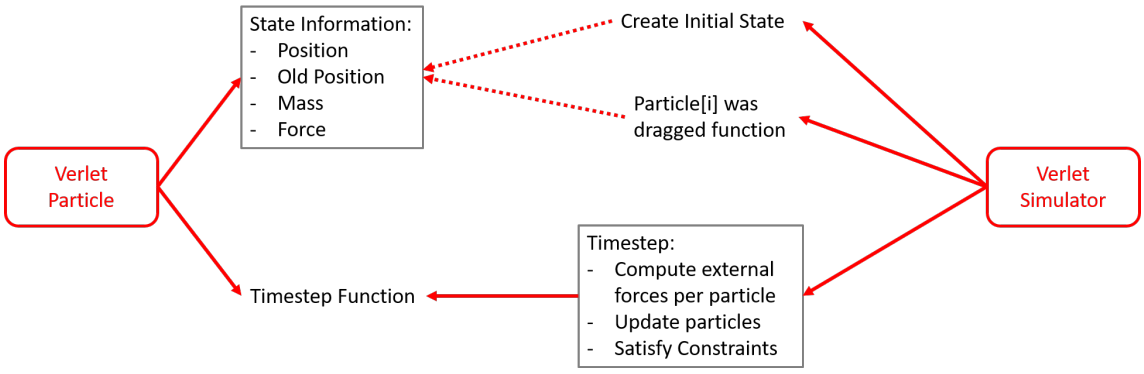

Figure 4.7: Overview of the VerletParticle and VerletSimulator and the methods implemented.

Since this implementation works with constraints, we can't set the spring forces to higher or lower values. Despite this, we can still control the number of iterations used to attempt to satisfy the constraints and, regardless of high or low number of iterations, there were never issues with stability.

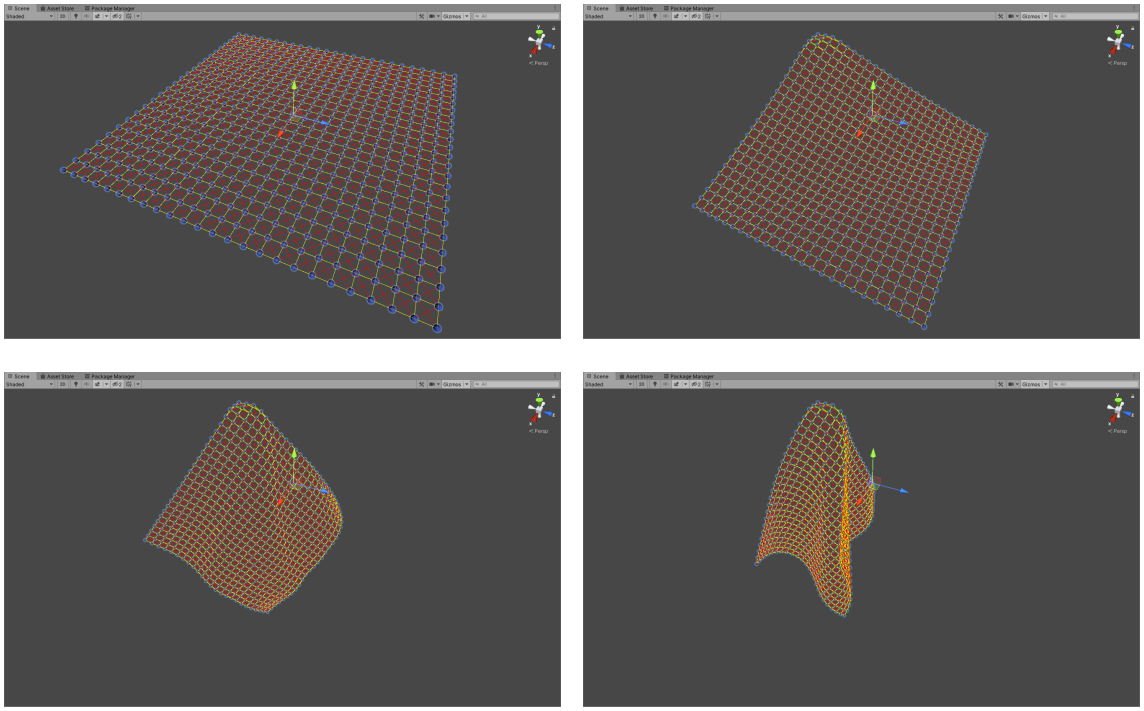

Figure 4.8: Display of Verlet Integration. These images were obtained from the simulation using high values for constraint iterations on a matrix of dimensions 25x25, where the upper left particle's position was fixated. The external forces present were gravity and wind.

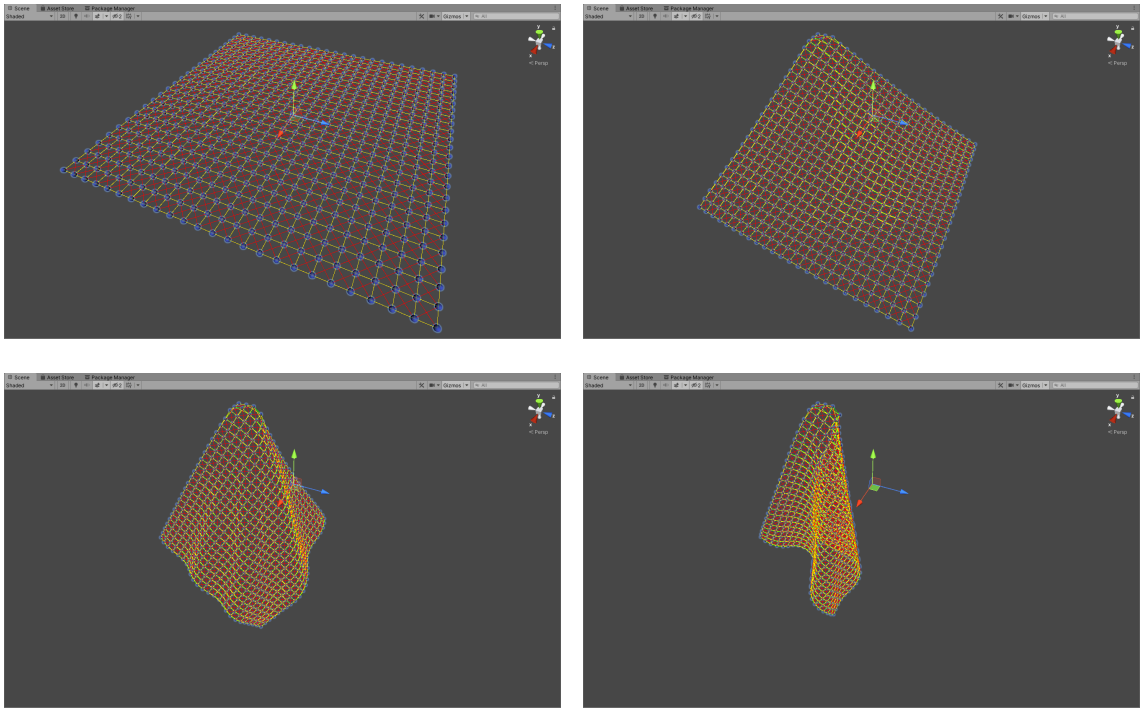

Figure 4.9: Display of Verlet Integration. These images were obtained from the simulation using low values for constraint iterations on a matrix of dimensions 25x25, where the upper left particle's position was fixated. The external forces present were gravity and wind.

Using a high number of iterations, as seen on Figure 4.8, resulted in the cloth getting closer to returning to its original equilibrium state, thus giving it a stiffer look; lower number of iterations resulted in the cloth taking a more flexible appearance, as seen on Figure 4.9. It is worth noting that even when using low number of iterations, the piece of cloth never took a the droopy appearance.

Independently of the number of iterations to solve the constraints, the Verlet integrator can also provide a reliable solution to the cloth simulation problem.

4.4 Tearable Cloth

Anther that can be easily implemented into our models is the ability to tear it when the cloth is placed under too much stress. While this may seem complicated at first, we can use a really simple approach to emulate this behavior: if two connected particles (by a spring or constraint) become too far apart, then we break their connection [38]. The amount of distance required to tear a connection should not be a constant, but instead proportional to the resting distance of that connection.

Both of our integrators have a variable named "Max Tearing Ratio" that defines the ratio for this proportion. As an example, if our maximum tearing ratio is set at "2", then a spring or constraint will be destroyed if the distance between the interacting particles is larger that 2 times the resting distance for that same interaction. This feature comes at

little computational cost, since we already need to iterate over every spring or constraint and calculate how far away from equilibrium it is.

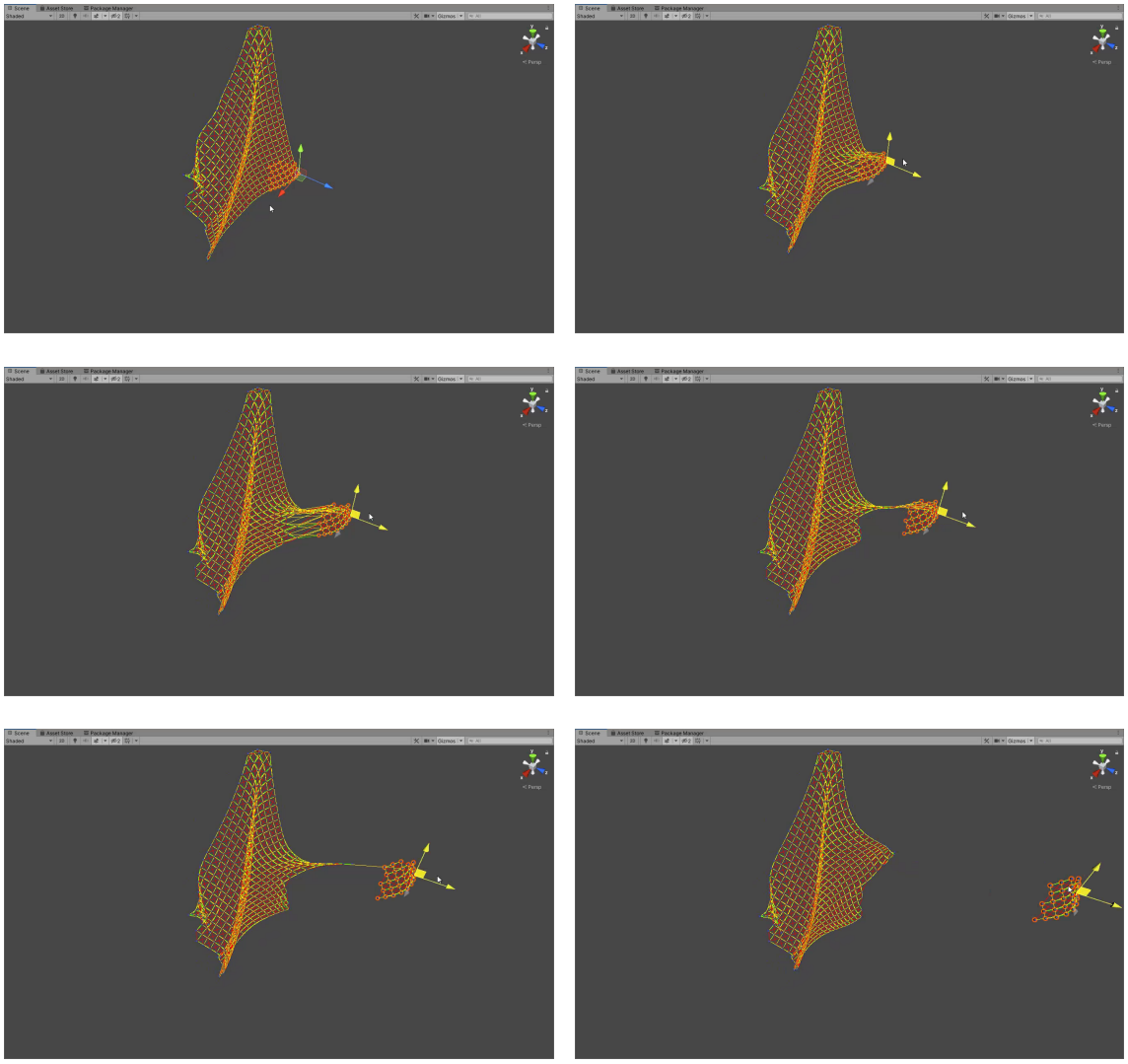

Figure 4.10: Tearing part the cloth by dragging part of it. These images were obtained from a simulation using the symplectic Euler method, with high values for spring constants and maximum tearing ratio of 3, on a matrix of dimensions 25x25, where the upper left particle's position was fixated.

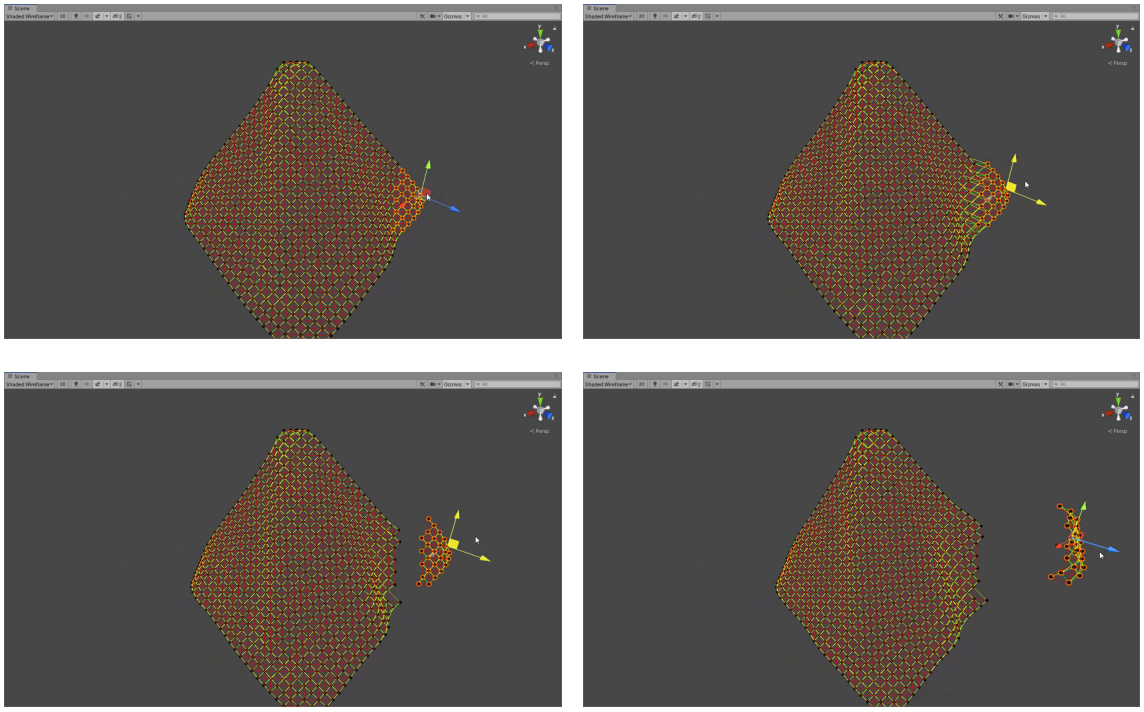

Figure 4.11: Tearing the cloth by dragging part of it. These images were obtained from a simulation using the Verlet method, with high values for constraint iterations and maximum tearing ratio of 2, on a matrix of dimensions 25x25, where the upper left particle's position was fixated.

The figures above show examples of cloth tearing in action, due to the interaction of a user dragging part of the cloth away, ripping it into two pieces. Since the implementation of cloth tearing is based on the same concept, we can visualize similar results on both algorithms:

- Some particles become distant from their neighbor particles;
- They reach the maximum tension allowed (defined by the maximum tearing ratio property of each model);
- The connections snap and we create a tear that separates these particles from their neighbors

The tears captured in the figures were catalyzed by quickly dragging away part of the particles, however tears can also occur naturally without user intervention, *e.g.* by using weaker spring forces and applying gravity similar to what was displayed on Figure 4.5. Allowing the user to tear cloth increases the realism of the simulations and, subsequently, the immersiveness of the experience of interacting with cloth.

4.5 Conclusion

This chapter focused on the implementations of two possible algorithms for performing cloth simulation.

We started by implementing an explicit Euler algorithm, although we found it to be too unstable to use, thus we had to improve it by the usage of a symplectic integrator instead. Using the symplectic Euler algorithm produced good and stable results. We also implemented a Verlet algorithm that also yielded good and stable results. We also added the ability on both methods to simulating cloth tearing when placed under too much stress.

At the end of this chapter, the reader should have a better understanding of the implementations of the algorithms based on Euler and Verlet integrations on the framework.

5 Results and Validation

The cloth simulation framework developed throughout this thesis was validated by the implementation of the simulation algorithms shown on section §4 as well as the usage of the key features of our manager with these algorithms. These algorithms were subsequently validated visually through a series of experiments. By the end of the chapter the reader can verify that this solution is capable of being extended by any mass-spring system, making it an effective system to compare and develop more algorithms based on this approach as well as testing the effects on the algorithm parameters in real time.

5.1 Validation of the Framework

Validation is a key point in our research, as we need to know that our framework facilitates the ability to test an algorithm. For testing the framework, we implemented two popular cloth simulation techniques that are based on mass-spring models, but that use different approaches for tackling this problem.

The first algorithm implemented used the symplectic Euler method and it occurred alongside the development of the framework to ensure that all the components were properly built and communicating as well as to improve any areas that could be enhanced. The second algorithm was implement after the framework was completed in order to ensure that the integration of a new algorithm could be done quickly and easily.

Posteriorly, these implementations of the algorithms were used to validate the usability of the framework along with the features provided, such as the external forces for wind and gravity, updating parameters in real time and importing 3D models from different software objects.

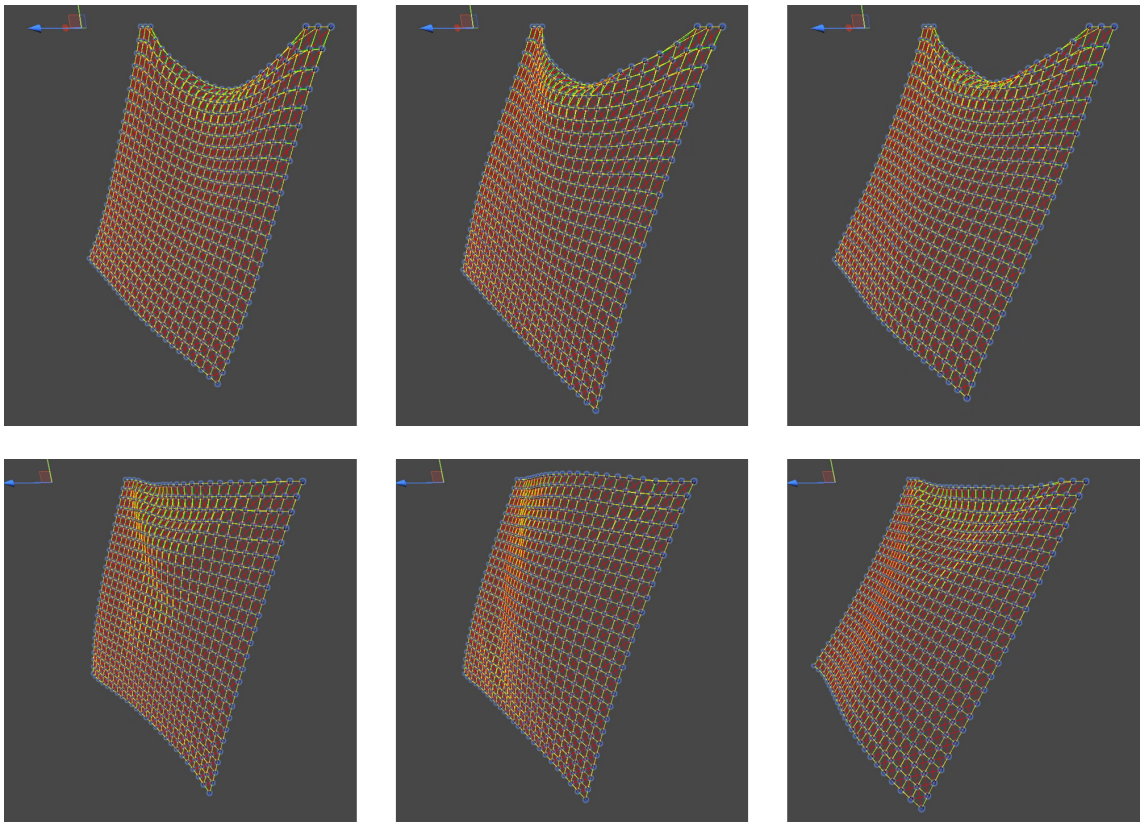

Figure 5.1: Examples of the Euler (top) and Verlet (bottom) algorithms working as extensions of the framework. Three particles on each corner were fixated to give the visual appearance of an hanging cloth.

The implementation of these two distinct algorithms for cloth simulation validates that this framework can be used as a foundation for the development of new particle-based algorithms or the implementation of existing ones .

5.1.1 Validation of the effects of external forces

Now that we have at least one cloth algorithm implemented into our framework, we can also validate the other features. Despite being possible to validate these features without a cloth simulation algorithm implemented, having access to one makes visual validation easier and more robust.

One possible way to validate the effect of gravity, is to simulate a piece of cloth and observe it falling, as seen on the figure below.

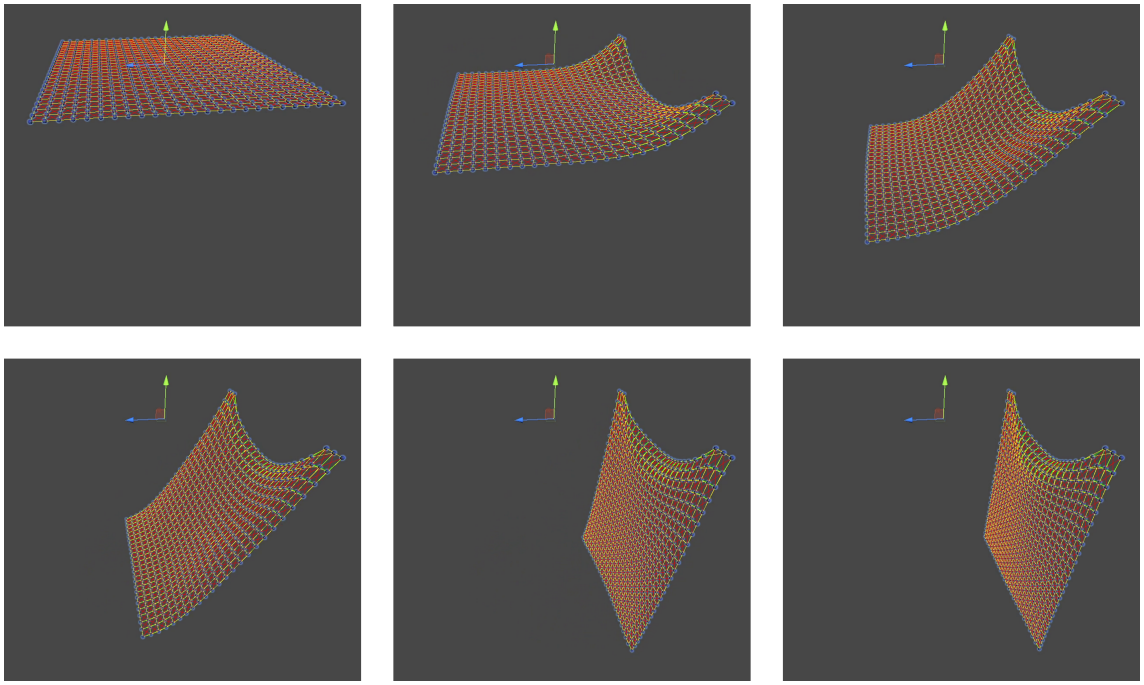

Figure 5.2: Examples of the gravity force being applied to each particle of the model. Simulation uses Euler method with 3 particles fixed on each side of the cloth.

Validating the wind force can be achieved by using the same scene as the one presented on the the figure above with the wind force added on top, in order to observe a waving motion, similar to a flag.

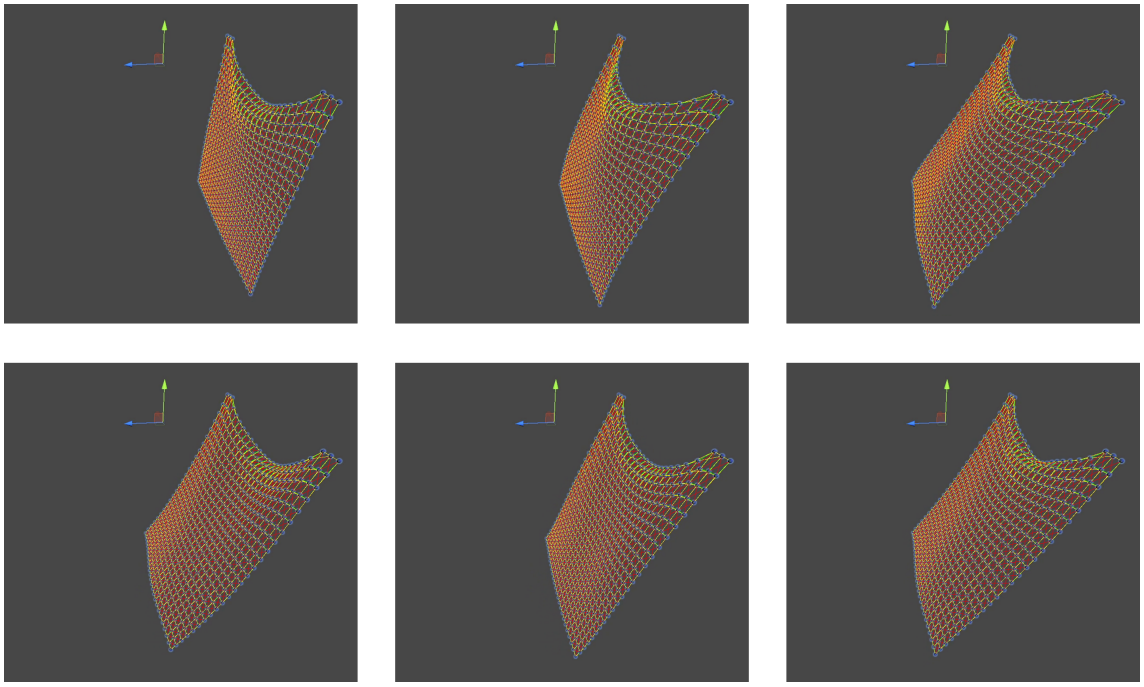

Figure 5.3: Examples of the wind force being applied to each particle of the model. Simulation uses Euler method with 3 particles fixed on each side of the cloth and gravity is also applied.

The simulation on Figure 5.3 was done as a continuation of the simulation presented on Figure 5.2 but, remembering that wind was implemented as an oscillating force, the

effect takes some time to ramp up, as seen on the first three images.

Both of these forces cause the expected effects on the simulation, therefore we assess this feature as validated. This result was also verified using the framework with the Verlet algorithm, whose results are similar.

5.1.2 Validation of user interaction

The last key feature of the framework that can be validated using the implemented algorithms is user interaction. This is divided in two categories: the ability to interact with the particles and move them around or flick them, as well as interacting with the simulation at runtime through the modification of the parameters available on the editor.

First, moving particles allows the user to interact directly with the model, and can be validated by the simulation reacting to the movement of the particles, as can seen on Figure 5.4 below. The user starts by slowly dragging the particles on the first three images, quickly flicks them on the fourth image and then stops the interaction for the following images.

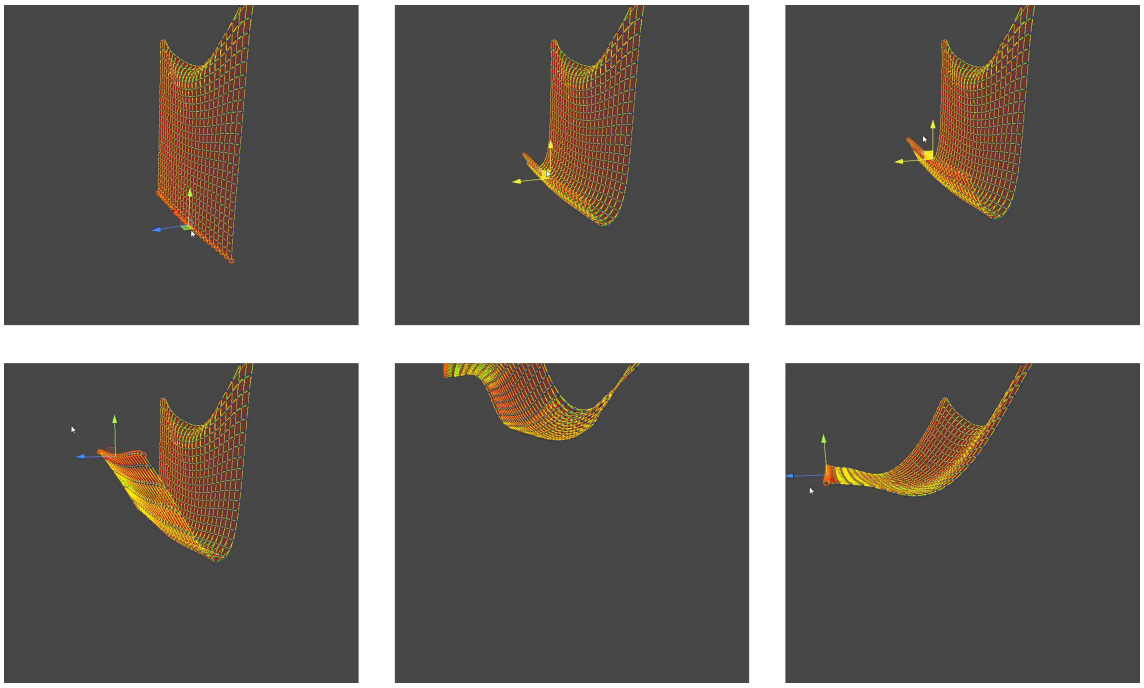

Figure 5.4: Examples of the user dragging and then flicking a set of selected particles from the model. Simulation uses Euler method with 3 particles fixed on each side of the cloth and gravity is also applied.

Second, the ability to interact with the variables set on the framework editor at runtime can be validated by modifying the value of one property while the simulation is running and observing the simulation being affected and adapted to the new properties, as can be seen on Figure 5.5 and Figure 5.6 below.

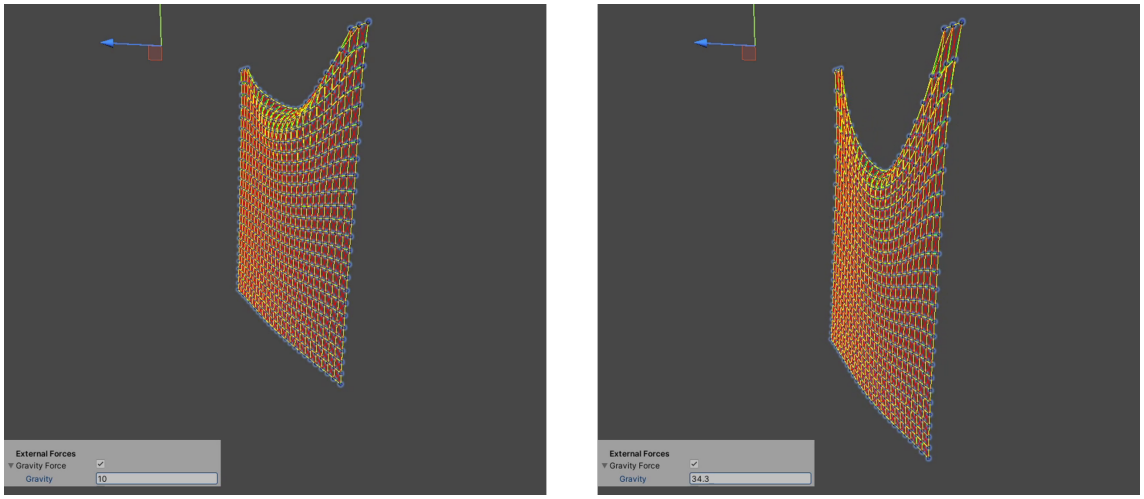

Figure 5.5: Examples of the user increasing the value of the gravity force at runtime: on the left the value for gravity is 10 units, and on the right the value is 34.3 units. Simulation uses Euler method with 3 particles fixed on each side of the cloth.

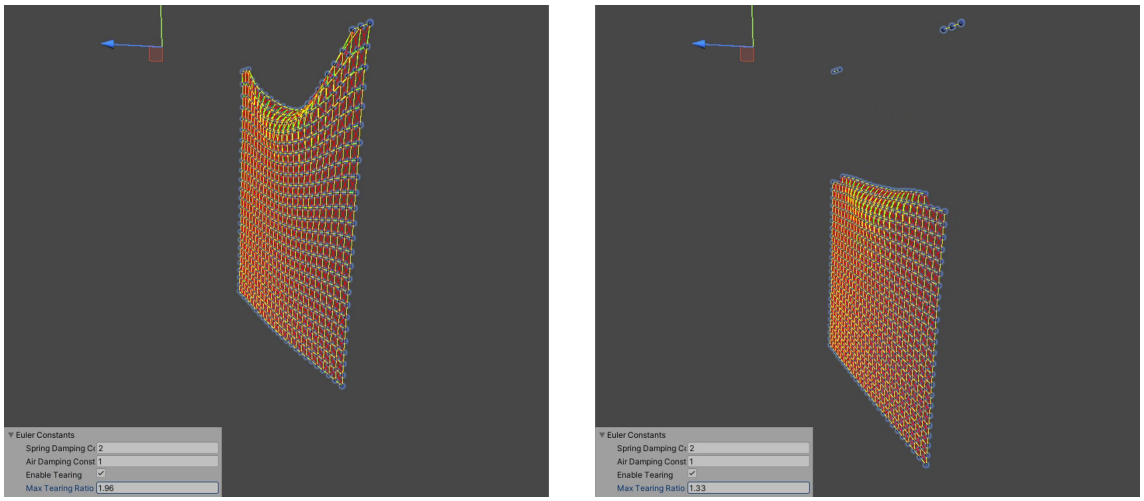

Figure 5.6: Examples of the user decreasing the value of the maximum tearing ratio force at runtime: on the left the value is 1.96 units, and on the right the value is 1.33 units. Simulation uses Euler method with 3 particles fixed on each side of the cloth and gravity is also applied.

The figures from this chapter display the simulation reacting to the user moving the visible particles in the scene as well as to changes on the properties of different simulation parameters, thus validating the user interaction feature. Similarly to the previous subsection (5.1.1), this result was also verified using the framework coupled with the Verlet algorithm and the obtained results were analogous.

5.1.3 Validation of importing models

The last feature of our framework that needs to be validated is the ability to create a mass-spring model ready to be fed to a simulation algorithm, which can be done by providing our framework with a 3D imported object and visualizing the particles and strings placed in

the proper positions. Using a simple model, such as the cylinder, makes the visualization of the placement of the particles and strings easier to analyze and validate.

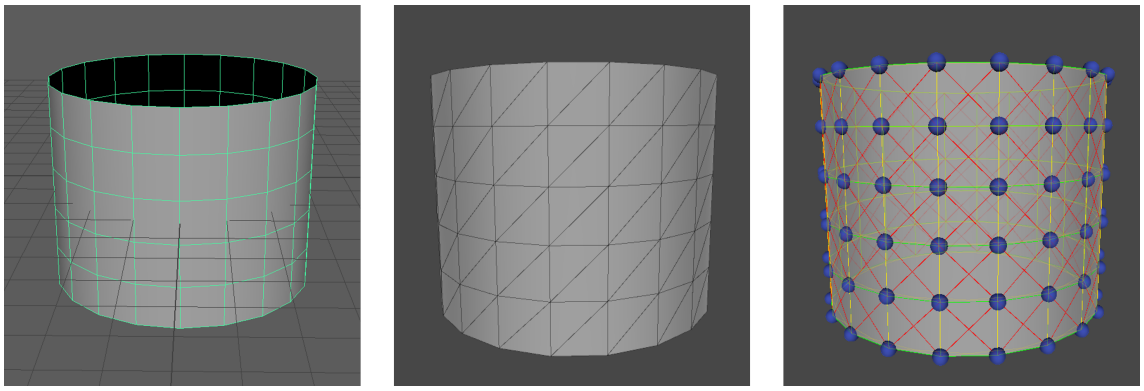

Figure 5.7: Example of a 3D model cylinder created in Maya (left image) being imported into Unity (middle) and then being imported by the framework (right) to produce a mass-spring model.

Since all the particles are correctly placed on the vertices and the springs connect the appropriate particles, we consider this feature to be validated. Images of this model being properly simulated by the algorithms implemented is also visible on 3.18.

5.2 Conclusion

This chapter provided validation to the flexible design of the framework, as well as validation of its key features. This framework provides a good base to implement multiple cloth simulation algorithms, visualize the results on different models and interact with it. Not requiring the developer or user to implement all these capabilities for each simulator would make the task of trying different simulation algorithms and iterating over parameters a much more tedious, long and arduous task.

6 Conclusion and Future Work

This thesis describes a framework for cloth simulation that is flexible and easily extendable. The system provides access to a lot of important features that can help the development or testing of real-time simulation algorithms. The creation of an abstract layer from the engine used allows for the implementation of simulation algorithms to be more independent and more easily compatible with other engines or frameworks. On this chapter we provide some insight on the work and results achieved from the development of this thesis. We conclude with possible future work to improve the framework followed by some final thoughts about cloth simulation.

6.1 Conclusion

During the development of this thesis, we were able to achieve all of the objectives we defined at the start. We started with an revision of important concepts for cloth simulation followed by an *analysis of previous work done in particle-based simulation algorithms in real time.*

Next, we *designed and described the implementation of the cloth simulation framework*, which resulted in a generic, flexible and easily extendable framework for cloth simulation algorithms. We followed up with an *implementation of the real time cloth simulation algorithms* studied (Sympletic Euler and Verlet) during the initial phase of investigation, which were useful to *validate the framework and its features*: user interaction, external forces and importing external 3D models.

Despite all the research and wide spread usage of cloth simulation, it still requires a lengthy process of trial and error to achieve good results. As such, we present a framework that provides flexibility, extensibility and interactivity to the implementation of cloth simulators, in order to speed up the process of testing different implementations and tuning parameters. In multiple industries related to film, video games and virtual fashion design, there is the need to simulate the animation of a cloth object, and due to the many available techniques and different parameters, this framework makes sure that the developer's and user's time is focused on choosing the right algorithms and parameters for their target and task.

In order to create the framework it was important not only to learn about the challenges faced of cloth simulations, but also research and implement our own simulation methods. Only after implementing these methods, was it possible to really understand what features are important to the framework as well as what information needs to be exchanged between the different framework components and the simulator code.

As a final note, the code for the implementation of this system is open-source and available online as a Github Project at https://github.com/ruifigueiredo19/ClothSimulationFramework.

6.2 Future Directions

The cloth simulation framework presented throughout this thesis enables users to integrate different particle-based algorithms in order to visualize and interact with them. Despite this, there are some limitations to our framework and its implementation that could be improved.

There are improvements that can be made to the components that form the framework:

- **Particle Displayer**
  - Using custom Unity gizmos to allow the user to fixate or free particles;
  - Information about under how much tension each spring is, by changing color shade or the width of the line that represents that connection.
- **Target Manager**
  - Allow the usage of a simulation mesh proxy in association with a rendering mesh [5];
  - Support maps for the simulation for the properties such as density maps and influence maps.
- **Simulation Manager**
  - Search for other alternatives for improving the integration of new algorithms even further, such as a custom UI with a graph editor.

Besides these improvements to the modules, the framework can also be improved by making it even more generic and supporting algorithms that are based on different popular approaches such as continuum models.

6.3 Final Thoughts

Cloth simulation is an incredibly rich and interesting area that is expanding as more industries emerge that can take advantage of these methods. As developments on this topic keep surging along with developments of the underlying hardware that we use, it will be interesting to see if and when the algorithms for offline and real time simulations

[5] This is a common approach used by many simulators to detach the rendered mesh from the simulation mesh. Meshes for rendering may require higher complexity than the mesh for simulation. *E.g.* http://obi.virtualmethodstudio.com/tutorials/clothproxies.html

will merge. Many companies are pushing towards implementing more and more systems that are computed in real time, in order to create more reactive and immersive systems for the viewers and the players. With technology setting higher and higher quality standards, it will be interesting to see how far we'll be able to push the current state of the art along with much more accessible it will be to new developers and artists.

References

[1] J. Allard and B. Raffin. Distributed Physical Based Simulations for Large VR Applications. In *IEEE Virtual Reality Conference (VR 2006)*, pages 89–96, 2006.

[2] John Austin. Fix Your (Unity) Timestep! https://johnaustin.io/articles/2019/fix-your-unity-timestep, 2019.

[3] David Baraff. Physically based modeling: Course Notes. *SIGGRAPH Course Notes, ACM SIGGRAPH*, 2(1):2–1, 2001.

[4] David Baraff and Andrew Witkin. Large steps in cloth simulation. pages 43–54, 1998.

[5] Ahmad Hoirul Basori, Mohammed Hazim Alkawaz, Tanzila Saba, and Amjad Rehman. An overview of interactive wet cloth simulation in virtual reality and serious games. *Computer Methods in Biomechanics and Biomedical Engineering: Imaging & Visualization*, 6(1):93–100, 2018.

[6] G. K. Batchelor. *An Introduction to Fluid Dynamics*. Cambridge University Press, 1967.

[7] Jan Bender and Crispin Deul. Efficient cloth simulation using an adaptive finite element method. In *VRIPHYS*, pages 21–30, 2012.

[8] Kevin Bowden, Tommy Nilsson, Christine Spencer, Kubra Cengiz, Alexandru Ghitulescu, and Jelte van Waterschoot. I Probe, Therefore I Am: Designing a Virtual Journalist with Human Emotions. 05 2017.

[9] David E. Breen, Donald H. House, and Phillip H. Getto. A physically-based particle model of woven cloth. *The Visual Computer*, 8(5-6):264–277, 1992.

[10] R. Bridson, S. Marino, and R. Fedkiw. Simulation of Clothing with Folds and Wrinkles. 2005. doi: 10.1145/1198555.1198573. URL https://doi.org/10.1145/1198555.1198573.

[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*, pages 594–603, 2002.

[12] Michel Carignan, Ying Yang, Nadia Magnenat Thalmann, and Daniel Thalmann. Dressing animated synthetic actors with complex deformable clothes. *ACM Siggraph Computer Graphics*, 26(2):99–104, 1992.

[13] Ming Chen and Kai Tang. A fully geometric approach for developable cloth deformation simulation. *The visual computer*, 26(6-8):853–863, 2010.

[14] Kwang-Jin Choi and Hyeong-Seok Ko. Stable but Responsive Cloth. 2005. doi: 10.1145/1198555.1198571. URL https://doi.org/10.1145/1198555.1198571.

[15] Kwang-Jin Choi and Hyeong-Seok Ko. Research problems in clothing simulation. *Computer-aided design*, 37(6):585–592, 2005.

[16] G. Cirio, J. Lopez-Moreno, and M. A. Otaduy. Yarn-Level Cloth Simulation with Sliding Persistent Contacts. *IEEE Transactions on Visualization and Computer Graphics*, 23(2):1152–1162, 2017.

[17] Frederic Cordier and Nadia Magnenat-Thalmann. A data-driven approach for real-time clothes simulation. In *12th Pacific Conference on Computer Graphics and Applications, 2004. PG 2004. Proceedings.*, pages 257–266. IEEE, 2004.

[18] Mathieu Desbrun, Peter Schröder, and Alan Barr. Interactive animation of structured deformable objects. 99(5):10, 1999.

[19] David DeVault, Ron Artstein, Grace Benn, Teresa Dey, Ed Fast, Alesia Gainer, Kallirroi Georgila, Jon Gratch, Arno Hartholt, Margaux Lhommet, et al. SimSensei Kiosk: A virtual human interviewer for healthcare decision support. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*, pages 1061–1068, 2014.

[20] B. Eberhardt, O. Etzmuß, and M. Hauth. Implicit-Explicit Schemes for Fast Animation with Particle Systems. pages 137–151, 2000.

[21] Carl Richard Feynman. *Modeling the appearance of cloth*. PhD thesis, Massachusetts Institute of Technology, 1986.

[22] Glenn Fiedler. Fix Your Timestep! https://gafferongames.com/post/fix_your_timestep/, 2004.

[23] Darian Gunamardi and Henry Novianus Palit. Optimisasi Cloth Simulation Menggunakan Parallel Computing Berbasis GPU pada Platform Nvidia CUDA. *Jurnal Infra*, 6(1):13–18, 2018.

[24] David Haumann. Modeling the physical behavior of flexible objects. *Topics in Physically-based Modeling, Eds. Barr, Barrel, Haumann, Kass, Platt, Terzopoulos, and Witkin, SIGGRAPH Course Notes*, 1987.

[25] Min Hong, Jae-Hong Jeon, Hyo-Sub Yum, and Seung-Hyun Lee. Plausible mass-spring system using parallel computing on mobile devices. *Human-centric Computing and Information Sciences*, 6(1):23, 2016.

[26] R. Jangir, G. Alenyà, and C. Torras. Dynamic Cloth Manipulation with Deep Reinforcement Learning. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4630–4636, 2020.

[27] Jesper Mosegaard. Mosegaards Cloth Simulation Coding Tutorial. https://viscomp.alexandra.dk/?p=147, 2009.

[28] Feng Ji, Ruqin Li, and Yiping Qiu. Three-dimensional garment simulation based on a mass-spring system. *Textile Research Journal*, 76(1):12–17, 2006.

[29] M. Keckeisen, S. L. Stoev, M. Feurer, and W. Strasser. Interactive cloth simulation in virtual environments. In *IEEE Virtual Reality, 2003. Proceedings.*, pages 71–78, 2003.

[30] Michael James King, P. Jearanaisilawong, and S. Socrate. A continuum constitutive model for the mechanical behavior of woven fabrics. *International journal of solids and structures*, 42(13):3867–3896, 2005.

[31] Zorah Lahner, Daniel Cremers, and Tony Tung. Deepwrinkles: Accurate and realistic clothing modeling. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 667–684, 2018.

[32] Miles Macklin, Matthias Müller, Nuttapong Chentanez, and Tae-Yong Kim. Unified particle physics for real-time applications. *ACM Transactions on Graphics (TOG)*, 33 (4):1–12, 2014.

[33] Young Jin Oh, Tae Min Lee, and In-Kwon Lee. Hierarchical Cloth Simulation Using Deep Neural Networks. In *Proceedings of Computer Graphics International 2018*, CGI 2018, pages 139–146, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450364010. doi: 10.1145/3208159.3208162. URL https://doi.org/10.1145/3208159.3208162.

[34] Xavier Provot. Deformation constraints in a mass-spring model to describe rigid cloth behaviour. pages 147–147, 1995.

[35] Witawat Rungjiratananon. Physics simulation R&D at Square Enix. In *SIGGRAPH Asia 2015 R&D in the Video Game Industry*, pages 1–2. 2015.

[36] A. Selle, J. Su, G. Irving, and R. Fedkiw. Robust High-Resolution Cloth Using Parallelism, History-Based Collisions, and Accurate Friction. *IEEE Transactions on Visualization and Computer Graphics*, 15(2):339–350, 2009.

[37] Raymond A. Serway and John W. Jewett. *Physics for scientists and engineers with modern physics*. Cengage learning, 2018.

[38] Tuur Stuyck. *Cloth Simulation for Computer Graphics*, volume 10. Morgan & Claypool Publishers, 2018.

[39] Andrew Taylor, Ertu Unver, and Graham Worth. Innovative potential of 3D software applications in fashion and textile design. *Digital Creativity*, 14(4):211–218, 2003.

[40] Demetri Terzopoulos, John Platt, Alan Barr, and Kurt Fleischer. Elastically Deformable Models. *SIGGRAPH Computer Graphics*, 21(4):205–214, 1987. ISSN 0097-8930. doi: 10.1145/37402.37427.

[41] Thomas Jakobsen. Advanced Character Physics. http://graphics.cs.cmu.edu/nsp/course/15-869/2006/papers/jakobsen.htm, 2003.

[42] Bernhard Thomaszewski, Markus Wacker, and Wolfgang Straßer. A consistent bending model for cloth simulation with corotational subdivision finite elements. 2005.

[43] Alexis Vaisse. Ubisoft Cloth Simulation: Performance Post-mortem & Journey from C++ to Compute Shaders. In *Game Developers Conference*. Ubisoft, March 2015.

[44] Jerry Weil. The synthesis of cloth objects. *ACM Siggraph Computer Graphics*, 20 (4):49–54, 1986.

[45] T. Yu, Z. Zheng, Y. Zhong, J. Zhao, Q. Dai, G. Pons-Moll, and Y. Liu. SimulCap : Single-View Human Performance Capture With Cloth Simulation. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5499–5509, 2019.

[46] Y. Zhao and C. Jiang. Online Virtual Fitting Room Based on a Local Cluster. In *2008 International Workshop on Education Technology and Training 2008 International Workshop on Geoscience and Remote Sensing*, volume 1, pages 632–635, 2008.